Object Detection and Segmentation with Deep Learning: From Fixed to Variable-Length Representations

by

Abhineet Kumar Singh

A thesis submitted in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science
University of Alberta

Abstract

There are two ways to model the output space of visual recognition tasks like object detection and segmentation. The conventional approach is to use a continuous-valued and fixed-sized feature map. A newer alternative is to use a variable-length sequence of discrete tokens that are output by the network one at a time through autoregression.

The token-based approach combines elements from both computer vision and natural language processing and is inspired by image and video captioning models that take one or more images as input to produce a variable-length sentence as output. It has two main advantages. Firstly, it allows us to succinctly model the output space in domains where the output-size is highly variable. For example, the number of objects in a video can vary widely across videos and the fixed-sized representation used in conventional object-detectors introduces a great deal of sparsity in the loss computation which makes training difficult. Secondly, it eliminates the need to perform heuristics-based postprocessing on the raw network output to convert it into a form suitable for downstream processing, thereby allowing true end-to-end training. For example, usable bounding boxes can be constructed directly from the tokens output by a token-based detector without having to perform confidence thresholding and non-maximum suppression.

This thesis uses a wide range of real-world applications to make a case for the benefits of the second approach. It also presents a novel way to extend a token-based object detector for video detection and semantic segmentation.

Preface

This thesis was submitted in partial fulfillment for the degree of Doctor of Philosophy (Ph.D.) in Computing Science at the University of Alberta. The thesis is an original work by Abhineet Kumar Singh and the presented work was accomplished between September 2017 and April 2025.

Part of the material for this thesis is based on the following papers:

- Chapter 3
  - **A. Singh**, H. Kalke, M. Loewen and N. Ray, "River Ice Segmentation With Deep Learning," in IEEE Transactions on Geoscience and Remote Sensing (TGRS), vol. 58, no. 11, pp. 7570-7579, Nov. 2020, doi: 10.1109/TGRS.2020.2981082.
- Chapter 4
  - **A. Singh**, M. Pietrasik, G. Natha, N. Ghouaiel, K. Brizel and N. Ray, "Animal Detection in Man-made Environments," 2020 IEEE Winter Conference on Applications of Computer Vision (WACV), Snowmass, CO, USA, 2020, pp. 1427-1438, doi: 10.1109/WACV45572.2020.9093504.
- Chapter 5
  - **A. Singh**, I. Jasra, O. Mouhammed, N. Dadheech, N. Ray, and J. Shapiro, "Towards Early Prediction of Human iPSC Reprogramming Success," in Machine Learning for Biomedical Imaging (MELBA), vol. 2, pp. 390–407, Oct. 2023, doi: 10.59275/j.melba.2023-3d9d.

Acknowledgements

I would like to extend my most sincere gratitude to my supervisors Prof. Nilanjan Ray and Prof. Hong Zhang. I greatly appreciate their kindness in allowing me to be a part of their research group as well as the invaluable guidance and encouragement they have provided over the years.

I would also like to acknowledge the help and support of the many people from other departments and organizations who have collaborated with me on the various projects that form an important part of this thesis. In particular, I would like to thank Dr. Douglas Hallett from ISL Engineering, Prof. Mark Loewen and Hayden Kalke from the Civil and Environmental Engineering Department, Ken Brizel, Dr. Nehla Ghouaiel and Marcin Pietrasik from ACAMP, Dr. Nidheesh Dadheech, Dr. Ila Jasra and Omar Mouhammed from Alberta Diabetes Institute, and Mojtaba Hedayatpour from Mojow Autonomous Solutions.

Finally, I would like to thank the University of Alberta Computing Science Department in addition to my supervisors for funding my PhD program through research and teaching assistantships.

Table of Contents

List of Tables

List of Figures

List of Algorithms

List of Abbreviations

MOT	*Multi-Object Tracking*
LLM	*Large Language Model*
GPU	*Graphics Processing Unit*
EOS	*End Of Sequence token*
SOS	*Start Of Sequence token*
DVC	*Dense Video Captioning*
VIT	*Vision Transformer*
SNR	*Signal-to-Noise Ratio*
SOT	*Single-Object Tracking*
NLP	*Natural Language Processing*
VL	*Vision-Language*
RIS	*Referring Image Segmentation*
GRU	*Gated Recurrent Unit*
SVM	*Support Vector Machine*
CNN	*Convolutional Neural Network*
UAV	*Unmanned Aerial Vehicle*
IOU	*Intersection-Over-Union*
fw	*frequency-weighted*
MAE	*Mean Absolute Error*
FPS	*Frames Per Second*
ARIS	*Alberta River Ice Segmentation dataset*
ATV	*All-Terrain Vehicle*
ACAD	*ACAMP Canadian Animal Detection dataset*
AP, mAP	*mean Average Precision*
RP, mRP	*value of the Recall–Precision curve where Recall=Precision*
cRP	*class-agnostic version of mRP*
ROC	*Receiver Operating Characteristics*
AUC, ROC-AUC	*Area Under ROC Curve*

RP-AUC	*Area Under Recall–Precision Curve*
cRP-AUC	*Area Under class-agnostic Recall–Precision Curve*
GT	*Ground Truth*
TP	*True Positives*
TN	*True Negatives*
FP	*False Positives*
FN	*False Negatives*
FN-DET	*FNs corresponding to missed detections*
FN-CLS	*FNs corresponding to misclassified detections*
FP-DUP	*FPs corresponding to duplicate detections*
FP-NEX	*FPs corresponding to detections of nonexistent objects*
IPSC	*Induced Pluripotent Stem Cell*
DfC	*Differentiating Stem Cell*
MLP	*Multi-Layer Perceptron*
MHA	*Multi-Headed Attention*
RLE	*Run Length Encoding*
3D-C	*3D mask flattening in C-order (row-major)*
3D-F	*3D mask flattening in Fortran-order (column-major)*
LAC	*Length-As-Class*
TAC	*Time-As-Class*
LTAC	*Length-and-Time-As-Class*
CW	*Class-Wise tokenization*
IW	*Instance-Wise tokenization*
TPU	*Tensor Processing Unit*

List of Models

SVM	*Support Vector Machine classifier [2, 3, 4]*
DeepLab	*DeepLabv3+ [5] with Xception-65 [6] backbone*
UNet	*UNet [7] with VGG-16 [8] backbone*
SegNet	*SegNet [9] with VGG-16 [8] backbone*
DenseNet	*DenseNet [10] with 9 layers*
RES101	*Faster RCNN [11] with ResNet-101 backbone [12]*
INRES	*Faster RCNN [11] with Inception-ResNetv2 [13] backbone*
NAS	*Faster RCNN [11] with NAS [14, 15] backbone*
RFCN	*RFCN [16] with ResNet-101 [12] backbone*
RETINA	*RetinaNet [17, 18] with ResNet-50 [12] backbone*
SSDIN	*SSD [19] with Inceptionv2 [20] backbone*
SSDMO	*SSD [19] with MobileNetv2 [21] backbone*
YOLO	*YOLOv3 [22] with DarkNet-53 backbone*
XGB	*XGBoost image classifier [23]*
CNC	*ConvNext image classifier [24]*
CND	*Cascade Mask RCNN [25] with ConvNext [24] backbone*
SWC	*Swin transformer image classifier [26]*
SWD	*Cascade Mask RCNN [25] with Swin transformer [26] backbone*
SWS	*UPerNet [27] with Swin transformer [26] backbone*
IDOL	*IDOL [28] video instance segmentation model*
SEQ	*SeqFormer [29] video instance segmentation model*
VITA	*VITA [30] video instance segmentation model*
P2S	*Pix2Seq static object detection model [31]*
P2S-VID	*Pix2Seq video object detection model (chapter 6)*
P2S-SEG	*Pix2Seq static semantic segmentation model (sec. 7.2)*
P2S-VIDSEG	*Pix2Seq video semantic segmentation model (sec. 7.3)*

List of Symbols

V	*number of tokens in the vocabulary*	*Eq. 6.1*
H	*number of coordinate bins used to discretize image-space*	*Eq. 6.1*
C	*number of classes*	*Eq. 6.1*
r	*number of reserved tokens*	*Eq. 6.1*
n	*number of object instances in an image or video*	*Sec. 6.1.1*
N	*number of frames in a video sequence*	*Sec. 6.1.2*
F_i	*i^{th} frame in a video sequence*	*Sec. 6.1.2*
(lx_i, ty_i)	*x-y pixel coordinates of top left corner of object bounding box in the i^{th} frame*	*Sec. 6.1.2*
(rx_i, by_i)	*x-y pixel coordinates of bottom right corner of object bounding box in the i^{th} frame*	*Sec. 6.1.2*
L	*maximum sequence length*	*Sec. 6.1.2.2*
T	*temporal stride between consecutive video temporal window used for training or inference*	*Sec. 6.1.2.4*
G	*frame gap between consecutive frames of a video temporal window*	*Sec. 6.1.2.4*
B	*Training batch size*	*Sec. 6.2.1.1*
S	*size of segmentation mask (after any subsampling) from which RLE is generated*	*Sec. 7.2.1*
P	*size of sliding window patch extracted from (possibly resized) image*	*Sec. 7.2.2*
I	*size of the image (after possible resizing) from which patches are extracted*	*Sec. 7.2.2*

Chapter 1

Introduction

1.1 Motivation

1.1.1 Conventional Modeling

Deep learning models for computer vision tasks like object detection and segmentation conventionally produce fixed-sized outputs, where the output size is determined by the architecture itself and is independent of the contents of the input image or video. This is true of models based on both convolutional neural networks (CNNs) [32] and transformers [33, 34, 26]. Further, this output is usually in the continuous domain, that is, it consists of real numbers rather than discrete integers.

This is not well suited to many visual recognition tasks where the outputs are inherently discrete in nature. The two tasks that exemplify this best are object detection and multi-object tracking (MOT). In both cases, the output size depends on the input and is highly variable across inputs since the number of objects in an image or the number of trajectories in a video can vary widely in real-world scenarios from none to many tens of instances. In addition, the space of all possible objects in an image is not only *much* larger than the number of objects that might realistically be present in an image but it is also far too large to be sampled densely by any fixed-sized representation. This problem is greatly compounded in MOT and video object detection since the space of all possible objects or trajectories in a video increases exponentially with the length of the video while the actual number of such instances remains the same as in a single image. Any fixed-sized modeling of this output space must therefore employ an output size that is much larger than the actual number of output instances that the model needs to produce.

This in turn leads to two main problems. Firstly, it causes the loss function to become very sparse since the training signal only comes from objects that are actually present in the training images and these objects constitute a tiny fraction of the space of all possible objects that the network output, and therefore the loss function, needs to represent. As a consequence, a vast majority of this output corresponds to the background and the resultant class imbalance needs to be handled through complex loss engineering. Examples of this engineering include careful domain-specific anchor box design along with one-to-many ground truth (GT) to anchor box associations [11], focal loss [1] and hard example mining [35]. Secondly, we need to perform a lot of heuristics-based postprocessing on the raw network output to produce the small number of discrete outputs that we actually need for downstream processing. Confidence thresholding and non-maximum suppression are a couple of examples of this postprocessing typically employed in object detection. This introduces a disconnect between what the network learns from the training data and how it actually performs during inference. In some cases, especially with MOT, this postprocessing can have a greater impact on the overall performance than the underlying model itself [36, 37]. Further, when the detector or tracker is used as one component of a larger system that is otherwise fully differentiable, the presence of these heuristics makes it difficult to train the entire system end-to-end since most of these are non-differentiable.

An important example of this problem is furnished by anchor boxes or similar methods of sampling the space of possible boxes, that are used in both RCNN [38, 11, 19, 1, 39, 40] and YOLO [41, 42, 22, 43, 44, 45, 46, 47, 48] families of detectors. Let us consider the standard Faster RCNN [11] network output which is a feature map of size $33 \times 33 \times 80$. Each one of the $33 \times 33 = 1089$ pixels in this feature map is an anchor point that has a fixed number of associated anchor boxes to represent various scales and aspect ratios. Since each of these boxes needs to be encoded by 4 numbers, this feature map can represent $80/4 = 20$ anchor boxes. This gives the total number of boxes output by the network as $1089 \times 20 = 21780$. Figure 1.1 shows a couple examples[1] of these boxes, although only 1% of all the boxes are shown here for clarity. This number remains fixed irrespective of the contents of any actual image. As a result, even if the image has only one object or no objects at all, the network will still output values for all of the nearly 22K boxes. This leads to enormous sparsity in the loss function since most training images have < 10 objects, so that $> 99.95\%$ of

[1]These examples have been borrowed from this online article

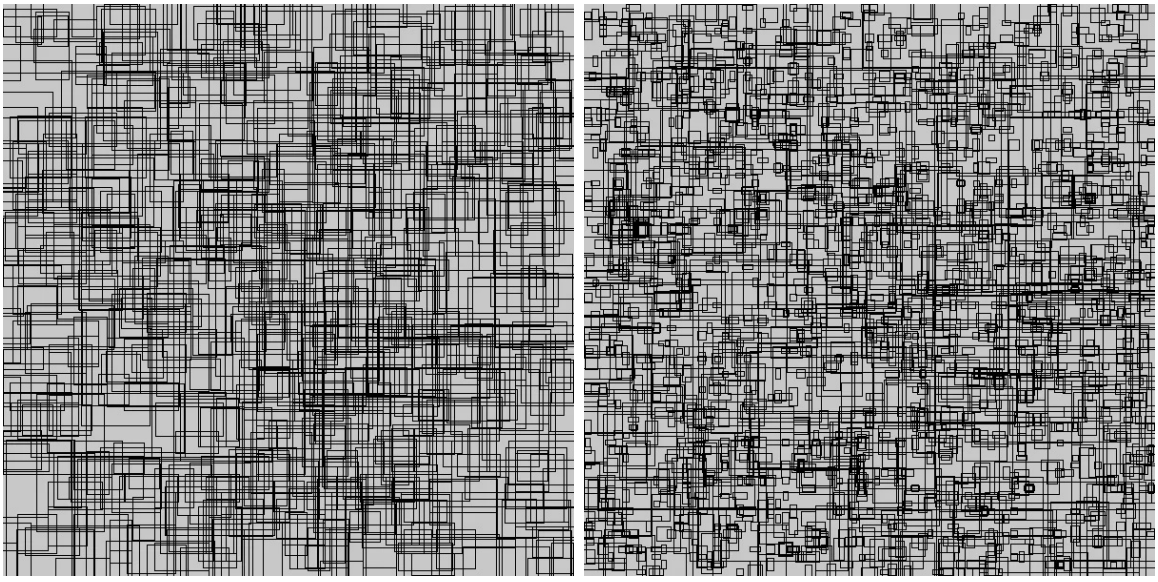

Figure 1.1: Two examples of anchor boxes as used by RetinaNet [1]. Only 1% of all the anchor boxes are shown here for clarity. The images on the left and right respectively show boxes that have been fine-tuned for detecting larger and smaller objects.

these anchor boxes corresponding to background image patches.

1.1.2 Language Modeling

The above issues with conventional modeling can be resolved by modeling the output space with a discrete variable-length representation. There has in fact been a trend towards such discretization in the vision literature over the last couple of years by representing the outputs as sequences of discrete tokens [31, 49, 50, 51, 52, 53, 54, 55, 56, 57]. This seems to have been inspired by the remarkable success of transformer-based natural language processing (NLP) models [58] in recent years, especially large language models (LLMs) [59, 60] like Chat-GPT [61, 62]. The tokens in these models are produced one at a time, similar to the way a language translation model outputs words from the target language one at a time. This is usually done by autoregression [31, 49], where the next output token depends on the past output tokens along with the input signal. The input is a sentence from the source language in case of language translation and can be one or more images in the case of vision tasks.

The problem of variable-length outputs is handled by a special end of sequence (EOS) token that is output by the network to indicate that it has completed

producing all the required tokens for the current input. For example, if each object is represented by 4 tokens and there are 20 objects in an image, the network will output 80 tokens followed by EOS. On the contrary, if the image has only one object, the network will output 4 tokens followed by EOS. While the EOS token allows the network to produce variable length outputs, there is still an architectural limit on the maximum number of tokens that it can produce, as imposed by the maximum sequence length L. There is also a limit on the total number of unique tokens that can constitute this sequence, as imposed by the vocabulary size V. Language modeling effectively simplifies vision tasks into classification problems since the L tokens can be thought of as solutions to L sequential V-class classification problems. Each token is represented by a probability distribution over the V tokens in the vocabulary. These probability distributions are output as a single $L \times V$ matrix, where each row represents a probability distribution over all the tokens in the vocabulary. Even though L and V are architectural hyperparameters just like the size of the output feature map in conventional models, and therefore also independent of the actual input image, they do not cause the same loss sparsity issue as the conventional models.

This can be best illustrated by a quantitative example. Let us assume that $V = 32K$ and $L = 500$. The corresponding 500×32000 probability distribution matrix actually has two orders of magnitude *more* values than the $33 \times 33 \times 80$ feature map in the anchor box example ($16M$ versus $87K$). However, training the token-based model involves learning to produce only 500 probability distributions, irrespective of the vocabulary size. This is in contrast to training the conventional model which must learn to regress nearly $22K$ boxes or $87K$ real numbers. Further, each GT object can provide a training signal for 4 of these 500 distributions in language modeling, while it will only provide a training signal for 1 of the $22K$ boxes in conventional modeling. This makes the loss sparsity in the latter two orders of magnitude greater than the former. Also, at least in my experience, probability distributions seem to be easier to learn than regressing over real numbers. This discretization also helps during inference since the *argmax* process of converting probabilities into tokens filters out a lot of the prediction noise that remains in the regression output of conventional models.

Against the above advantages, language models do have the practical disadvantage of being significantly more difficult to train. The large size of the probability matrix requires more computational resources to train since it must fit

in the limited memory available on the graphics processing unit (GPU), that is currently necessary for accelerating the training process to make it time-feasible. Transformer-based models in general and language models in particular also require large amounts of training data and larger batch sizes to train well [63, 64], all of which also increases the amount of GPU memory and training time needed.

This thesis studies the extent to which the theoretical advantages can be realized in practice in the face of these challenges. For this purpose, I propose several new token-based modeling schemes for video object detection and semantic segmentation and compare these with conventional modeling on a wide range of real world problems. It turns out that the limited training hardware available for this project prevents me from demonstrating significant performance advantage of token-based models over conventional models in this thesis. However, this work still represents an important step in the direction of discretizing vision tasks and does provide evidence of the immense potential of these models to improve with better training hardware and more efficient implementations.

It is also important to note that the new way of representing video detection and semantic segmentation outputs with tokens has value in its own right beyond its current or future performance benefits since it opens up the possibility of improving the tokenization in clever ways to encode additional attributes that would be either impossible or much more difficult with conventional modeling. An example of this is the IW tokenization in Section 7.3.3 that supplements semantic segmentation with instance information by simply reordering the tokens. This would require an entire channel to be added to the output mask with conventional segmentation models.

1.1.3 MOT to Language Modeling

My original thesis topic was actually to design an end-to-end differentiable MOT system. The goal of MOT is to detect each object of interest that enters the scene in a video and track it as it moves about and then exits the scene, in order to output its complete trajectory as a sequence of boxes. As mentioned earlier, the principal challenge here is that the number and lengths of trajectories can vary widely across videos, which makes it very difficult to represent these through a fixed-sized output. To the best of my knowledge, a true end-to-end differentiable MOT system that takes video frames as input and directly outputs the trajectories has not been proposed yet.

Most existing MOT methods use some variant of tracking-by-detection [65] where

a separate object detector first detects all the objects in each video frame and then an association algorithm links these objects across frames. This involves a lot of domain-specific heuristics-based postprocessing which does not generalize well and often has more impact on the overall tracking performance than the association algorithm itself [36, 37]. I spent most of the first three years of my program working on adapting a modular tracking-by-detection framework [66] to replace each of its non-differentiable components with differentiable deep learning-based alternatives so as to make the entire MOT pipeline differentiable and integrate it with the object detector to create an end-to-end trainable system. However, I discovered that the heuristics played such an important role in determining the overall tracking performance and were so finely tuned to the existing components that replacing the latter with improved deep learning-based models did not improve the tracking performance and in many cases actually made it slightly worse.

I therefore abandoned this project and instead started looking into ways to dispense with the separate object detector and all the accompanying heuristics. The most promising idea I had, on which I spent much of the next two years, was to represent MOT as a sequence prediction problem. This idea was inspired by image and video captioning tasks, especially dense video captioning (DVC) [67]. As opposed to standard video captioning which describes a short video clip showing a single event with a single sentence, the goal of DVC is to describe multiple events in a long video clip through one or more sentences, while also localizing each event in time. Since a trajectory can be thought of as an event, a DVC network [68, 69] could be modified to output a sentence for each trajectory. Each word would then represent a single box in that trajectory and a set of sentences would represent all the trajectories in the video clip.

I did manage to adapt an end-to-end differentiable DVC model [68] for MOT but could not get it to learn meaningful trajectories even after several months of experimentation with many feature extractors and tokenization strategies. This may have been because the DVC model was designed only to detect high-level events and did not have the granularity required to learn frame-level information like the bounding boxes that make up trajectories. It might also have been because I simply did not have enough computational resources to retrain it for a completely different domain from what it had been designed for. In retrospect, it was probably a bad idea to start with the most complex of the three captioning tasks instead of starting with the simplest one by adapting an image captioning network for static object detection

followed by standard video captioning for video detection and finally DVC for MOT.

When I finally started running out of time to complete my program, I decided that this was too complex a task to solve with my very limited computational resources in the little time I had left. I therefore decided to simplify it to adapting image captioning for object detection which is probably what I should have started with in the first place. While looking for a suitable image captioning network to adapt, I came across the Pix2Seq framework from Google [31] which had already done this. However, this framework had not yet been extended for video detection so I worked on that instead and succeeded in getting it working in a few months (chapter 6).

Over the course of my program, I have also completed several projects unrelated to my main thesis topic, both as industrial internships and collaborations with other departments. Three of these projects have led to publications [70, 71, 72] while my main thesis research has not resulted in any publication so far due to the failure of both my main ideas. Since extending Pix2Seq for video detection is insufficient to merit a doctoral-level thesis by itself, I needed to link the tokenization idea with the three published works to create a single coherent narrative. Two of these projects are about object detection and instance segmentation [71, 72] and therefore applicable for my video detector. However, the third project [70] involves semantic segmentation and contains neither video annotations nor instance level information. I have therefore adapted Pix2Seq for this new task (chapter 7) to be able to include this project as part of this thesis too. This also serves as an example of applying language modeling to a dense prediction task which does not inherently require discrete outputs but nonetheless turns out to work quite well when modeled thus.

1.2 Contributions

To summarize, following are my main contributions in this thesis:

- I have adapted conventional deep learning models to solve a wide range of novel problems in the domain of object detection and segmentation.
- I have presented a new way to perform video object detection by language modeling.
- I have presented new ways to perform semantic and panoptic segmentation in images and videos by language modeling.

- I have compared the language models with the conventional models through a wide range of experiments to highlight their current limitations along with the promise they hold for future improvements.

I would like to reiterate that, even though I have not been able to show significant performance benefits of language modeling due to hardware limitations, these new ways of representing vision outputs are significant contributions in their own right. This is both due to their demonstrated suitability for end-to-end video processing and the potential they offer for clever tokenization schemes to embed additional information in the output that would be very difficult or even impossible with conventional modeling.

1.3 Thesis Outline

A brief review of existing literature relevant to my proposed token-based models is presented in chapter 2. This is followed by details of three projects I have undertaken in the course of my doctoral program in chapters 3 through 5. These chapters are adapted from respective publications and serve to demonstrate the application of conventional deep learning models for object detection and segmentation. Chapters 6 and 7 respectively present details of my adaptation of Pix2Seq for object detection in videos and semantic segmentation in both images and videos. Chapter 8 compares these token-based models with the conventional models from chapters 3 to 5. This chapter also includes results for some of my experiments with their configuration parameters to make them better. Finally, chapter 9 concludes with a summary of the findings in chapter 8 along with suggestions for future improvements.

Chapter 2

Background

This chapter presents a brief review of recent language modeling methods for solving vision tasks like object detection, segmentation and tracking. Reviews of conventional deep learning models for tackling these tasks have been deferred to chapters 3 - 5.

2.1 Pix2Seq

This section briefly describes the Pix2Seq framework [73], especially the object detection component [31], that chapters 6 and 7 are based on.

2.1.1 Overview

Pix2Seq is a language modeling framework for computer vision that comprises a series of papers produced by the same research group at Google DeepMind and implemented within the same codebase [73]. The original paper [31] dealt with static object detection in images. It was soon followed by a multi-task version [49] for performing instance segmentation, keypoint detection and image captioning, in addition to object detection. Next, the authors replaced the autoregressive transformer decoder with a diffusion module [50] to be able to do dense prediction and applied this for image generation and captioning. Finally, they adapted this diffusion-based dense prediction model to perform panoptic segmentation in images and videos [51], although the video component was very rudimentary and seemed rather like an afterthought. Around this time, a different research group at Google proposed a related method for DVC named Vid2Seq [74], though this is an LLM and is not implemented within the main Pix2Seq repository. The Pix2Seq group has

also proposed several secondary methodological and architectural improvements that are implemented in the same repository. These include the Recurrent Interface Network [75] and improved noise scheduling [76] for diffusion and an improved transformer architecture with two-level hierarchical token generation [77].

2.1.2 Object Detection

2.1.2.1 Tokenization

Pix2Seq object detector [31] represents each object in the image by five tokens - four for the (x, y) coordinates of the bounding box corners and one for the class. The image-space is quantized into 2000 bins to represent the continuous coordinates with discrete tokens and shared tokens are used for both x and y coordinates. The tokens for all the objects are output sequentially followed by the EOS token so the network and output a total of $5 \times n + 1$ tokens for an image with n objects. Further details of Pix2Seq tokenization strategy are included in Section 6.1.1.

2.1.2.2 Architecture

The network has a transformer-based encoder-decoder architecture [33] with an autoregressive decoder. During training, it takes the image and the GT token sequence prefixed with the start of sequence (SOS) token as input and is trained to output the same token sequence without the SOS token but suffixed with the EOS token. For example, if an image has two objects, the input and output token sequences would respectively be [SOS, ty_1, lx_1, by_1, rx_1, cls_1, ty_2, lx_2, by_2, rx_2, cls_2] and [ty_1, lx_1, by_1, rx_1, cls_1, ty_2, lx_2, by_2, rx_2, cls_2, EOS]. Here, (lx_i, ty_i) and (rx_i, by_i) are the quantized coordinates of the top left and bottom right corners of the i^{th} object bounding box[1], while cls_i is its class token. The autoregressive constraint is imposed through a causal mask [33] that prevents the network from using future tokens in the input sequence to influence its next output token. This ensures that the k^{th} token in the output sequence is affected only by the first $k-1$ tokens in the input sequence. The network is therefore trained to output, for example, the token lx_1 with $[SOS, ty_1]$ as the input tokens and by_2 with $[SOS, ty_1, lx_1, by_1, rx_1, cls_1, ty_2, lx_2]$ as the input tokens. During inference, the network gets only the SOS token as input to bootstrap it to output tokens for all the objects in the image. Section 6.2.1 includes more details of Pix2Seq architecture.

[1] this notation is reused in Section 6.1.2 with slightly different meaning

2.1.2.3 Localization

One of the most remarkable aspects of this method is that it is able to produce spatially precise bounding boxes without needing any localization cues, thereby completely doing away with the need to sample the space of all possible boxes. This in turn makes it particularly suitable for video object detection and MOT since the space of possible trajectories in a video is exponentially larger than the space of possible boxes in an image, which makes sampling it impractical. The authors demonstrated an interesting fact about the learnt token embeddings which helps to explain this ability. Even though the coordinate and class tokens are part of the same shared vocabulary, the learnt coordinate embeddings show greater cosine similarity not only with other coordinate tokens but specifically with nearby coordinate tokens. This implies that the network is not only able to learn that the coordinate and class tokens belong to two separate categories of tokens but that coordinate tokens corresponding to spatialy proximal regions of the image are more similar to each other than to tokens from more distant parts of the image.

2.1.2.4 Object ordering

An important consideration when representing objects by a sequence is the order in which these objects appear in that sequence. There are many ways to order the objects but there are no domain-agnostic theoretical reasons for considering one method as superior to others. The authors experimented with several ways to sort the objects, including bounding box area and distance from the top left corner, but empirically determined that random ordering works best. As a result, they not only randomize the order of objects but randomly reorder the objects in each image for every forward pass of that image during training. The network is therefore trained to be able to output multiple sequences of tokens for the same image, though with all of them containing the same subsequences of 5 tokens, only in different orders. This is somewhat analogous to how captioning networks are trained with multiple captions for the same image, but all of them are semantically equivalent.

2.1.2.5 Sequence Augmentation

The authors discovered that training the model with only real boxes results in somewhat low recall rates because the network has a tendency to output the EOS token too soon. To fix this, they augmented the sequence of real boxes with a large

number of synthetic boxes sampled from the background. These are generated by both adding noise to the real objects (e.g. random shifting and scaling) as well as by sampling completely random background regions. These so-called noise boxes effectively provide a means to include background patches in the training set. While this is somewhat similar to conventional bounding box sampling techniques like anchor boxes, it is on a different plane numerically because the total number of such boxes is only a few tens rather than tens of thousands. Similar to the random ordering of real objects, the noise objects are re-generated for each forward pass, so that the network is trained with a new set of synthetic boxes each time it sees the same image. This essentially teaches the network that the noise boxes can be anywhere in the background except on the real objects and all such sets of background boxes are semantically equivalent, exactly like every possible ordering of real boxes. It is undesirable for the network to learn to mimic the random coordinates of noise boxes so no loss is assigned to the coordinate tokens of these boxes but the corresponding class tokens are weighted the same as those of real boxes. The network is therefore able to learn that any sequence of coordinate tokens that does not correspond to actual boxes belongs to the special noise class that is created for the synthetic boxes.

2.1.2.6 Postprocessing

The authors do perform some heuristics-based postprocessing during inference to filter out invalid boxes from the raw network output. They consider output boxes labeled as noise to be potential real objects that get assigned to the real (i.e., non-noise) class with the maximum probability among all the real classes.[2]. The usual dense-to-sparse object detection heuristics like confidence thresholding and NMS are then performed using this probability value as the confidence score for that object. The assumption here is that any invalid object will either not have a real-class probability high enough to exceed the confidence threshold or a nearby valid object will have higher probability for the same class and will therefore be able to suppress the invalid box. Even though such postprocessing is an unwelcome trait if the goal is to use the detector as one component of a larger system which needs to be trained end-to-end, it has minimal impact on the overall detection performance as measured by the average precision (AP) (44.7% vs. 43.7% on COCO) since it is mainly a trick to reduce missing boxes.

[2]since the object is labeled as noise, the noise class token is assigned the highest probability among all the class tokens but we pick the real class whose token got the second highest probability

The impact on recall is greater (61.7% vs. 55.4%) but still not very significant. In other words, the detector remains competitive with the state of the art even without sequence augmentation or postprocessing. We can therefore completely dispense with these tricks and still have a viable detector.

2.1.2.7 Performance

The overall performance of this detector was shown to be similar to other standard detectors like Faster-RCNN [11] and DETR [78]. This is to be expected since detection performance seems to have reached a plateau in terms of output modeling and most improvements since the original RCNN [79] have been achieved by increasing the dataset and network sizes.

2.1.3 Multi-Task

A multi-task version of the Pix2Seq detector was proposed in [49] to support instance segmentation, human keypoint detection and image captioning, in addition to object detection. This uses a single shared vocabulary along with a single prediction head to produce output for all the tasks. It takes a task-specific prompt as input to figure out which task is to be performed. For example, a task prompt consisting only of the special `detect` token causes it to perform object detection. Similarly, a prompt that starts with the `segment` token, followed by 5 tokens representing the bounding box coordinates and class of an object, causes it to segment the specified object. If this same prompt were to start with the `keypoint` token, the network would instead detect keypoints on the object. Finally, a prompt consisting only of the `describe` token results in the network outputting a natural language sentence describing the image. The vocabulary has 35K tokens in all, out of which 32K are English words, 1K are coordinate tokens and remaining are reserved and class tokens.

2.1.3.1 Instance Segmentation

As noted above, the variety of instance segmentation that this network performs is different from what is typically meant by this task. An instance segmentation method would normally take an image as input and output bounding boxes and masks for all the objects in that image. This network, on the other hand, requires a specific object as input and produces a mask only for that object. We would first need to run it on the detection task and then run it again for every single object detected in the first

step to be able to complete the standard instance segmentation task. This not only makes it inefficient at runtime, but probably also makes the training less effective since the network cannot benefit from the global image context, being trained to segment only one object at a time.

The instance segmentation output modeling is a straightforward extension of object detection by means of representing the segmentation masks with polygons [80, 81] instead of pixelwise binary labels. Since a polygon does not have a canonical starting point, the authors randomize the starting point each time the same image is shown to the network, so it is able to learn an order-agnostic representation of that polygon. They also support disjointed masks by allowing multiple polygons to represent the same object, in which case the polygons are separated by a special separator token. However, they do limit the total number of points across all the polygons that represent an object to 128.

The authors employ several heuristics at inference to improve the performance. They found that polygon masks tend to be noisy, a shortcoming that they attribute to the absence of geometric regularization. If the model is repeatedly asked to generate polygons for the same object, it produces slightly different polygons, some of which might be of low quality. The authors resolve this issue by averaging the corresponding masks through a voting scheme where pixels that are foreground in at least half of the generated masks are considered to be foreground in the final averaged mask. They found that averaging over 8 samples in this way gave a 6% improvement in AP over using a single sample. They also tried running inference on only the image patch corresponding to the bounding box instead of the entire image but this only gave a small 1.3% improvement.

2.1.3.2 Human Keypoint Detection

Similar to the instance segmentation masks, human keypoints are also modeled after object detection by using a couple of coordinate tokens for every keypoint. In the general case, each keypoint can be accompanied by a class token to represent the specific body part that the keypoint corresponds to. In this paper, however, the authors simplified the representation by eliminating the class token and simply outputting a fixed set of 14 keypoints since that is how many are available in the COCO dataset that the authors use for all their testing.

2.1.4 Dense Video Captioning

Vid2Seq [74] was introduced by a different group at Google Research and is not implemented as part of the Pix2Seq repository, but it does use many of the same concepts. This performs DVC by adding a couple of time tokens to each sentence to mark the start and end of the corresponding subclip in the video. Time is quantized by dividing the video into 100 equally-spaced timestamps. The model was trained unsupervised on the massive unlabeled YT-Temporal 1B [82] dataset. The authors use speech recognition to transcribe narration audio into subtitles, followed by Google cloud API to convert the subtitles into sentences. They consider each sentence as a pseudo-event caption, that is, sentence boundaries become pseudo-event boundaries and each sentence is assumed to describe a single event. This simple heuristic apparently generalizes quite well to real captions. The model used here is an LLM with 314M parameters and required 64 TPUs for training with a total of 2TB RAM. Such hardware requirements render this model of limited relevance to academic research beyond informing us that such unsupervised training can work.

2.1.5 Image Generation and Captioning

The next step in this progression was BIT Diffusion [50] where the authors generate the output sequence using diffusion [83] instead of autoregression. The rationale for this substitution was that diffusion is known to work better for very long sequences and also has less stringent computation and memory requirements for such sequences. This makes it more suitable for dense prediction tasks like image generation where, for example, a 512×1024 RGB image requires $> 1.5M$ tokens. However, diffusion brings back the problem of fixed-sized and continuous valued outputs that besets conventional modeling and this makes it unsuitable for sparse and discrete prediction tasks like object detection and MOT that I am interested in.

The encoding scheme that the authors used to convert the discrete output back into continuous values is quite simple – they represent each token by the corresponding bits (i.e. binary digits), convert these bits into real numbers (i.e. 0 and 1 become 0.0 and 1.0 respectively) followed by shifting and scaling these numbers to -1.0 and $+1.0$. During inference, the diffusion model outputs real numbers between -1.0 and $+1.0$ and they use simple thresholding to decode this output back into discrete tokens.

The authors tested this idea on two prediction tasks – one dense and the other

sparse. The dense prediction task is discrete image generation. Here, they represent each of the three sub-pixels in an RGB image by 8 bits so that each pixel is represented by 24 bits and the image as a whole is represented by a 24-channel output. They experimented with three different bit conversion schemes for pixel values - direct 8-bit representation, assigning a unique 8-bit binary code to each possible pixel value such that consecutive values differ by only 1 bit, and shuffling the standard 8-bit representation. They found that all three schemes can work but the direct 8-bit representation outperforms the other two.

The authors use image captioning as the sparse prediction task. They use a vocabulary of 32K words so they encode each word using 15 analog bits since 15 is the minimum number of bits needed to represent integers up to 32K. They limit the maximum number of tokens in the output sequence to 64 for a total sequence length of 960 bits. They showed that this model performs similarly to the standard autoregressive version in [49], which indicates that densifying the output does not necessarily cause performance degradation in the sparse output domain.

2.1.6 Panoptic Segmentation

The final paper in the Pix2Seq series is a generalist model [51] that extends BIT diffusion for panoptic segmentation. Panoptic segmentation combines both semantic and instance segmentation so that the panoptic mask can be represented by a 2-channel image – one channel for the class ID and the other one for the instance ID. The authors use 8 bits to represent each pixel so that we get an $H \times W \times 16$ analog bit mask for an $H \times W$ image. The instance IDs are shuffled each time the same image is shown to the network, similar to the randomization strategies in the earlier papers. Since they use 8 bits to represent instance IDs, this imposes a limit of 256 on the maximum number of object instances that can be detected by the network.

The authors train the model by corrupting the analog bit mask with noise and training the network to denoise this corrupted mask. An important difference in the bit encoding scheme here as compared to [50] is that the bits are scaled and shifted to $[-b, b]$ instead of $[-1, +1]$ that was used by the latter. The parameter b has significant impact on performance by affecting the signal-to-noise ratio (SNR) of the corrupted masks - smaller b leads to lower SNR and thus better models since masks become more difficult to denoise which in turn allows a better denoising model to be trained. The authors show empirically that $b = 0.1$ works better than $b = 1.0$.

This paper briefly considers video panoptic segmentation too, but only for the streaming or online setting. The network takes only one frame as input so it does not fully utilize video information. Video denoising is instead performed by simply concatenating the predicted masks from previous timesteps to the noisy mask for the current timestep. This network is supposed to be able to do implicit frame-to-frame tracking, although no results are presented for the same. In fact, the entire video segmentation portion of the paper seemed like an afterthought rather than a core component of the paper.

2.2 Language modeling in Visual Recognition

This section provides additional examples of language modeling in computer vision. These models are categorized by the principal vision tasks that they address, although many of these are multi-task models that can handle multiple tasks simultaneously.

2.2.1 Learnt Tokenization

The specific way in which the output of a vision task is discretized and tokenized can be either designed manually or it can be learnt from data. There are many more methods in the first category because learning the tokenization strategy is apparently quite difficult and requires large amounts of data and computation. Learnt tokenization is also less satisfying from the perspective of interpretability. All the Pix2Seq methods covered so far are examples of manual tokenization. This section provides two examples of learning the tokenization strategy from data while the next section provides more examples of manually designed tokens.

2.2.1.1 Depth Estimation

UViM [84] is a multi-task model from Google DeepMind that supports depth estimation in addition to panoptic segmentation and image colorization. It proposes a two-stage architecture with a base model and a language model that are trained separately. It also proposes the so-called restricted oracle which learns to tokenize the dense visual output. The latter may, for example, be the depth image, segmentation mask, or colored image. The restricted oracle takes this visual output as its input and produces a fixed-sized sequence of tokens, referred to as the guiding code. The base model takes this code as an additional input along with the source

image and is trained jointly with the oracle to reconstruct the visual output using a standard reconstruction loss like pixelwise cross-entropy or mean squared error. This effectively makes the oracle and the base model respectively equivalent to the encoder and decoder components of an autoencoder that learns to reconstruct the visual output within the constraint of a fixed-sized discrete bottleneck in the middle, as represented by the guiding code.

The oracle is then frozen and the language model is trained to reproduce its discrete output from the source image alone. Note that the oracle is used only during training. During inference, the language model, which learnt to mimic the oracle output, produces the guiding code from the input image alone and this goes into the base model along with the image to produce the dense visual output. The authors use the standard vision transformer (VIT) [34] architecture for both the base model and the oracle, along with a standard encoder-decoder transformer architecture [85] for the language model, where the latter also has a VIT backbone. Training the oracle in stage 1 is difficult because the discrete bottleneck it imposes does not have a gradient. The authors solve this by mapping the embeddings to be quantized to their nearest-neighbours in a fixed-sized dictionary of fixed-dimensional embeddings [86]. Training in this way also turned out to be unstable and they had to use additional tricks including the Linde-Buzo-Gray splitting algorithm [87] to cull suboptimal embeddings.

2.2.1.2 Instance Segmentation

AiT [88] is another multi-task model that supports depth estimation and instance segmentation. AiT employs a similar two-stage architecture as UViM and a VQ-VAE based tokenizer [86] to learn the tokenization strategy. The authors refer to the base model, oracle and language model of UViM as the tokenizer, detokenizer and tasks solver respectively. They propose two new techniques to improve the performance of this model. The first one is to replace the standard one-hot tokens that are typically used in language modeling with the so-called soft tokens. A soft token is represented by a probability distribution over the vocabulary and when such a token is to be fed into the detokenizer or the autoregressive token-predictor, its embedding is generated by taking the weighted average of the embeddings of all the tokens in the vocabulary. The second novelty is a mask augmentation technique to handle corrupted or missing values in the label image. This is particularly common in depth estimation where occluded areas do not have their depth defined. The authors randomly mask patches

of the label image with valid values so the VQ-VAE tokenizer learns to fill in the known values of these patches and this apparently helps it to predict the values of missing patches as well.

2.2.2 Manual Tokenization

2.2.2.1 Single Object Tracking

SeqTrack [52] is an example of performing single object tracking (SOT) with language modeling though it performs only frame-to-frame tracking and therefore does not benefit from long-term motion information. SeqTrack shares much of its design with Pix2Seq, including the encoder-decoder architecture comprising an image encoder with VIT [34] backbone and an autoregressive sequence decoder. It likewise represents each object with 4 tokens (although in a slightly different $[x, y, w, h]$ format that encodes box center and size rather than corner coordinates), discretizes image-space into bins (4000 in SeqTrack vs. 2000 in Pix2Seq) and uses shared tokens for both x and y coordinates. However, since it needs to output the position of only one object in each frame, it is able to restrict the maximum sequence length to 5. The SOS token followed by x, y, w, h from the previous frame becomes the input sequence and x, y, w, h for the current frame followed by the EOS token is the output sequence.

The authors enlarge the template image to the same size as the search image by adding nearby background to the bounding box patch, which they found to help improve performance. They then extract features from both patches using the VIT backbone, concatenate these features and feed them into the encoder to produce the final features that combine information from both images. However, only a subset of these combined features that correspond to the search image are fed into the decoder, along with the input sequence, to produce the output sequence that represents the position of the object in the current frame. They still employ the usual SOT heuristics like online template updating and window penalty to improve their localization accuracy. SeqTrack is relatively heavy on hardware requirements since the authors needed 8 $\times$ Tesla A100 80 GB GPUs with combined 640 GB GPU memory to be able to train it with a batch size of 64. It also needs a lot of training data since they had to combine training data from 4 large datasets – COCO [89], LaSOT [90], GOT-10k [91], and TrackingNet [92]. It is comparatively light at inference, however, since it can run at 5 - 40 FPS on a single Geforce RTX 2080 Ti GPU, depending on the size and input resolution of the VIT backbone.

2.2.2.2 Vision-Language Tasks

This section covers some examples of vision-language (VL) tasks that combine images with text input or output.

2.2.2.2.1 VL Tracking

VL Tracking is a more complex version of SOT where the object to be tracked is specified by a natural language description in addition to the bounding box. MMTrack [53] applies language modeling to VL Tracking, with an overall methodology very similar to SeqTrack, except the obvious difference that it uses both the image and natural language description as inputs. It uses a similar encoder-decoder architecture but with separate encoders for image and text inputs - RoBERTa-Base [93] for text and OSTrack [94] for image - followed by a single autoregressive decoder. The main novelty here is in the form of the composite conditional queries that are composed of two parts - a language query with text embedding and a vision query with bounding box coordinates. Training MMTrack is much less demanding on hardware when compared to SeqTrack. The authors were able to train it on only a couple of Geforce RTX 2080 Ti GPUs with a total of only 22 GB GPU memory, albeit with a reduced batch size of 32. It was shown to be capable of running inference on a single 2080 Ti at 36 FPS so its inference speed is comparable to SeqTrack.

2.2.2.2.2 Referring Image Segmentation

Referring image segmentation (RIS) is a VL task whose objective is to create the instance segmentation mask of an object given the image and a natural language description of the object to be segmented. Unlike standard instance segmentation which only requires objects belonging to a predefined set of classes to be segmented, RIS involves segmenting any objects that can be described by a natural language sentence. PolyFormer [57] uses language modeling to accomplish this by representing the mask with a sequence of polygons whose vertices are output autoregressively. This is very similar to the instance segmentation component of the multi-task version of Pix2Seq [49]. The primary difference between the two is that the latter accepts a bounding box and class as input to specify the object to be segmented while the former takes a natural language description of the object. Another difference is that PolyFormer uses a regression-based decoder which outputs the floating-point coordinates of the polygon vertices instead of discrete

tokens, supposedly in order to increase localization accuracy by eliminating the quantization loss. It accomplishes this by using bilinear interpolation [95] to generate feature embeddings for floating point coordinates from those of the nearby indexed locations. Similar to [49], PolyFormer allows the object to be represented by multiple disconnected polygons by using a special separator token between the vertices of these polygons.

SeqTR [96] is another multi-task model that does RIS, in addition to other visual grounding tasks like phrase localization and referring expression comprehension. Similar to PolyFormer, it casts visual grounding as a point-prediction problem where either the bounding box or the binary segmentation mask is represented as a sequence of discrete coordinate tokens. Similar to UniTab, SeqTR is heavily inspired by Pix2Seq and uses the same coordinate discretization strategy along with the same encoder-decoder architecture, adapted to take text as input in addition to the image. The text is encoded by a one layer bidirectional gated recurrent unit (GRU), followed by max pooling the features along the channel dimension and then taking the Hadamard product with the image features to fuse them together. It claims to propose a new mask contour sampling scheme to convert binary masks into polygons but it is not clear how this is different from the representation in Polyformer and [49], except that it does not appear to support multiple polygons to represent disjointed masks.

2.2.2.2.3 Grounded Image Captioning

Grounded Image Captioning a VL task that combines object detection and image captioning. It involves accompanying the text description of standard image captioning with bounding boxes in such a way that each noun entity in the description is aligned with a single box. UniTab [56] is a multi-task model that performs four different VL tasks including grounded captioning, visual grounding, image captioning and visual question answering, although its focus is on grounded captioning. UniTab is strongly inspired by Pix2Seq and can be thought of as an extension of the same that adds text tokens to the existing bounding box tokens. It uses the same transformer based encoder-decoder architecture as Pix2Seq, with the main difference being that it takes text as input alongside the image, and uses RoBERTa-Base [93] and ResNet-101 [12] as the respective feature extractors. It adds two special tokens - $<obj>$ and $</obj>$ - that are respectively inserted before any word to be grounded and after the box tokens corresponding to that word. This

token allows the model to seamlessly switch between the text and bounding box tokens that are both part of a single shared vocabulary.

2.2.2.3 Set Prediction

This section covers two examples of output tokenization being cast as a set prediction problem [97, 98] instead of language modeling. They are both architecturally inspired by Pix2Seq but differ from it in having fixed-sized outputs like conventional models and using bipartite graph matching loss [78, 99] instead of autoregressive loss.

2.2.2.3.1 3D Object Detection

Point2Seq [54] has applied tokenization for 3D object detection from point clouds. The authors emphasize three main differences from Pix2Seq. Firstly, Pix2Seq outputs a single sequence representing all the objects while Point2Seq represents each object by a sequence and outputs the sequences of all the objects in parallel. Secondly, Point2Seq uses a continuous word representation instead of discrete tokens to make it easier to integrate with existing loss functions designed for 3D object detection. Finally, it uses a scene-to-sequence decoder instead of a transformer since it works better for detecting small and sparse targets that are common in 3D object detection. Point2Seq represents each 3D object by 5 words - region, location, orientation, size, category - where each word is composed of two or three parameters except the category word which has only one. Its architecture is composed of three main parts – a 3D backbone [100, 101] that converts the point cloud into a Bird-Eye-View (BEV) feature map, a scene to sequence decoder that takes the BEV feature map along with region cues as input and outputs the 5 words for each 3D box, and a similarity based sequence matching scheme to match the predicted sequences with the GT sequences for computing the set loss [78]. The authors tested this method on the ONCE [102] and Waymo Open [103] datasets and showed it to be superior to both point [100] and anchor-based [101] conventional detectors.

2.2.2.3.2 Pose Estimation

Obj2Seq [55] is a multi-task framework that performs human pose estimation in addition to object detection and image classification. It is somewhat similar to the multi-task version of Pix2Seq [49] and differs from the latter primarily in using object queries and a bipartite graph matching loss [78, 99] as opposed to the autoregressive

loss [33] that is standard in language modeling. In fact, it can more appropriately be conceptualized as an extended version of DETR [78, 99], with a class-conditioned query generator, rather than a general-purpose sequence generation framework for object-level vision tasks that it claims to be.

Obj2Seq employs a two-step pipeline. It first takes an image along with a set of class prompts as input and detects all objects belonging to each class in the prompt. It then passes the corresponding object queries to the so-called object transformer decoder which outputs a sequence describing the required attributes for each object. The specific attributes that are output in the second step depend on which task is to be performed. For example, it outputs the location and size for object detection and a set of keypoints for pose estimation. The latter is composed of 38 attributes, out of which 4 are for the bounding box and the remaining represent the x and y offsets of 17 keypoints. Similar to DETR, the number of detected objects is fixed as part of the architecture and this brings back the problem of sparsity that we are trying to solve with language modeling. For example, in their experiments on the COCO dataset, the authors set this to 100 objects per class and then select the top 20 classes for detecting the objects out of the total 80 classes in this dataset. This results in a total of 2000 objects which greatly exceeds the actual number of objects that might be present in any image.

2.3 Language modeling in Generalist Models

There has also been a trend towards trying to train a single generalist model that is able to output tokens corresponding to a variety of heterogenous tasks from many different and completely unrelated domains and modalities. Unfortunately, most of these models require vast amounts of data and industrial-levels of computational resources to be trained so they are of limited academic interest.

For example, GATO [104] was trained to perform 604 diverse tasks from multiple modalities. These included stacking blocks with a robotic arm, playing Atari video games, captioning images, chatting and many more tasks, all using the same network with the same weights. However, it required a 16×16 TPUv3 slice with a total of 8 TB of memory to train, which renders it completely infeasible in most academic settings. There is nothing particularly novel or clever about the tokenization strategies used in GATO since it relies mainly on the brute-force of the large network size and truly vast amounts of training data to make it work. Text and images are encoded

by SentencePiece [105] and VIT [34] while continuous floating point values are mu-law encoded into the range [-1, 1] and then discretized into 1024 bins. The token embeddings are likewise generated using standard techniques, including a lookup table into a learnable embedding matrix for text, discrete and continuous values and a single ResNet block for image patches. GATO was trained on a combination of control-task and vision–language datasets, although $> 85\%$ weightage was given to the former which consisted of 596 tasks, 63M episodes and 1.5T tokens.

OFA [106] is another multi-modal generalist model, although not quite as comprehensive as GATO, and restricted to vision and language domains. It supports 5 cross-modal tasks - visual grounding, grounded captioning, image-text matching, image captioning, and visual question answering - in addition to 2 vision tasks - image infilling an object detection - and a single language task – text infilling. Its multi-model training data is also much smaller than GATO, being comprised only of 20M image-text pairs, though, unlike GATO, it needed to be fine-tuned on task-specific datasets before it could be tested on each task. Also, the model sizes in OFA going upto 940M parameters and are still too large for most academic settings.

Unified-IO [107] is another example that combines vision tasks like pose estimation, object detection, depth estimation and image generation with VL tasks like region captioning and referring expression and NLP tasks like question answering and paraphrasing.

Chapter 3

River Ice Segmentation

This chapter is adapted from [70] and details my adaptation of conventional deep learning based semantic segmentation models for river ice segmentation. This is the first of the three projects where I applied conventional modeling to solve a novel problem. This chapter is focused only on the conventional models and the comparison of these models with my proposed language models is deferred to Sections 8.4.3.1 and 8.5. This work was done in collaboration with the Department of Civil and Environmental Engineering at the University of Alberta.

3.1 Introduction

The study of surface ice concentration and variation over time and place is crucial for understanding the process of river ice formation. The computation of temporal and spatial ice distributions can help to validate models of this process. The additional ability to distinguish frazil ice from the sediment-carrying anchor ice can also improve our estimation accuracy of the sediment transportation capacity of the river. Towards this end, a large amount of video data has been captured using an unmanned aerial vehicle (UAV) and bridge-mounted game cameras from two Alberta rivers during the winters of 2016 and 2017. My objective in this chapter is to analyze this data and perform dense pixel-wise segmentation on these images to automatically compute the concentrations of the two types of ice.

The main challenge in this task is the lack of labeled data since it is extremely difficult and time consuming to manually segment images into the three categories, owing to the arbitrary shapes that the ice pans can assume. As a result, there are currently only 50 labeled images (Figure 3.1) to accompany 564 unlabeled test

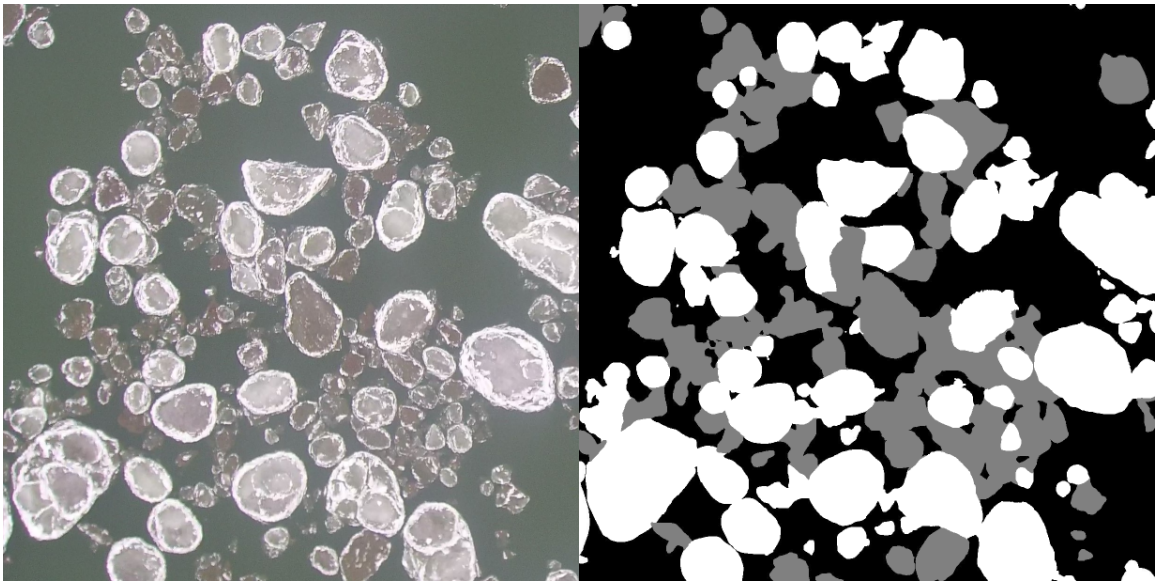

Figure 3.1: A sample training image with corresponding label where white, gray and black pixels respectively denote frazil ice, anchor ice, and water.

images and over 100 minutes of unlabeled high resolution (4K) videos. These labeled images along with 205 additional images with only ice-water labeling had previously been used to train a Support Vector Machine (SVM) classifiers [2, 3, 4] to perform segmentation. These models provided water-ice classification accuracies ranging from 80.1% - 93.5% and surface ice concentration errors of 0.7% - 3.3%. Though these methods were fairly successful at separating ice from water, they had difficulty in distinguishing between frazil and anchor ice pans, especially in cases where they are not physically separated and are hard to differentiate, even for human eyes. This chapter is mainly concerned with handling these more difficult cases.

To address the limitations of SVM-based ice classification, this work uses recent semantic segmentation methods based on deep CNNs. Since CNNs need large amounts of training data to work well, several data augmentation techniques (Section 3.3.3) have been used to generate enough training images. Detailed ablation studies have also been performed (Section 3.4.2.2, 3.4.2.3) to evaluate the extent to which few and partially labeled training images can be used to obtain results that are conducive to generating further training data with minimum human effort and thus set up a bootstrapping process.

3.2 Background

Surface ice is formed on rivers in cold regions like Canada when the air temperature remains below freezing for extended periods of time. It begins with the formation of frazil ice crystals and flocs as water columns become supercooled [108]. Being naturally adhesive [109], these ice crystals sinter together to form larger flocs whose increasing buoyancy brings them to the surface as slush. As the moving slush is in turn subjected to freezing temperatures, it forms solid frazil pans and rafts which can become circular with upturned edges on colliding with each other. Figure 3.1 shows examples of these formations.

Anchor ice can be formed on the river bed by direct deposition of frazil particles and flocs or by in-situ growth of ice crystals on the bottom sediments [110]. The growth of anchor ice accumulations occurs in multiple stages [111] and in some cases they can become thick enough to reach the water surface. However, as the thickness increases, the hydrodynamic drag and buoyancy forces also increase which may cause the anchor ice to release from the bed. Warming of the water by solar radiation can also cause the release of anchor ice. Once released, the ice floats to the surface forming anchor ice pans which can carry entrapped sediments and other materials from the river bed and transport them downstream.

An improved understanding of the importance of such transportation to the overall sediment budget of the river would be useful for developing and validating models of river processes. Thus, we require accurate estimates of the surface concentrations of sediment-carrying anchor ice in the river for which field measurements are unavailable. Note that, in order to accurately estimate the sediment transported by anchor ice, we also need to know the sediment concentration (kg/m^3) in the ice and the thickness of the pans. In this chapter, we are only considering the problem of obtaining accurate estimates of the surface concentration of the pans from digital image data. Since there is far too much data for manual analysis to be practical, one of the objectives of this work is to estimate this concentration from digital images and videos of the river surface in an automated or semi-automated manner using deep learning. Distinguishing ice from water is relatively straightforward and has been accomplished fairly successfully using simple techniques like thresholding [112] as well as classic machine learning methods like SVM with handcrafted features [2, 3, 4]. The main goal of this work is therefore to be able to distinguish between frazil and anchor ice pans with high

accuracy using state of the art deep learning techniques.

River ice images bear a significant resemblance to microscopic images of cells in the bloodstream which initially suggested the use of existing cell classification networks from medical imaging. There are several promising studies employing a range of architectures including ConvNet [113], LeNet [114], Resnet [115] and Inception [116] that might have provided the base networks for such an approach. However, a more detailed examination of these studies revealed that medical imaging tasks are mainly concerned with the detection and localization of specific kinds of cells rather than performing pixel-wise segmentation that is necessary to estimate ice concentration.

Further, unsupervised and semi-supervised video segmentation techniques were considered to better utilize the large amount of unlabeled but high-quality video data that has been collected. Most of these methods use optical flow for performing motion segmentation [117], though some appearance based [118] and hybrid [119, 120] methods have also been proposed. An unsupervised bootstrapping approach has been proposed in [121] where motion segmented images are used as training data to learn an implicit representation of this object under the assumption that all moving pixels belong to the same foreground object. This learnt model is then used to refine the motion segmentation and the improved results are in turn used to bootstrap further refinements.

Unfortunately, two underlying assumptions in [121], and motion segmentation, in general, render such methods unsuitable for the current task. First, they assume that there is a single moving foreground object whereas the objective here is to distinguish between two different types of moving ice, both of which are foreground objects. Second, they assume a static background while the river, which makes up the background here, is itself moving. Preliminary attempts to perform optical flow-based motion segmentation on the river ice videos confirmed its unsuitability for this work. There is a method [122] for performing simultaneous optical flow estimation and segmentation which might be able to address these limitations to some extent. However, it had only a Matlab implementation available that was far too slow for our purpose so its further exploration was deferred to future work.

Finally, it seems that very limited existing work has been done on the application of deep learning for surface ice analysis as the only one that was found [123] uses microwave sensor data instead of images.

Table 3.1: Trainable parameter counts for the four models

Model	**DeepLab**	**UNet**	**SegNet**	**DenseNet**
Parameters	41,050,787	12,284,019	11,546,739	90,948

3.3 Methodology

3.3.1 Data Collection and Labeling

Digital images and videos of surface ice conditions were collected from two Alberta rivers - North Saskatchewan River and Peace River - in the 2016-2017 winter seasons. Images from North Saskatchewan River were collected using both Reconyx PC800 Hyperfire Professional game cameras mounted on two bridges in Edmonton as well as a Blade Chroma UAV equipped with a CGO3 4K camera at the Genesee boat launch. Data for the Peace River was collected using only the UAV at the Dunvegan Bridge boat launch and Shaftesbury Ferry crossing. The game camera captured 3.1 megapixels resolution still images at one-minute frequency while the UAV camera captured 4K videos (8.3 megapixels) of up to 10 minutes duration.

Large 3840×2160 UAV images were cropped into several 1280×1080 images to make labeling more convenient while the smaller 2048×1536 game camera images were only cropped to remove text information added by the camera software. More than 200 of these images were labeled for binary ice-water classification but only 50 of these were labeled into 3 classes to distinguish between the two types of ice and the water. Only the latter images were used for training in this work. More details of the data collection and labeling process along with images of the camera setups are available in [3, Section 4.1]. We make both labeled and unlabeled data publicly available as the Alberta River Ice Segmentation (ARIS) dataset [124].

3.3.2 Image Segmentation

Since neither cell classification nor video segmentation methods seemed promising, it was decided to rely only on supervised image segmentation. After extensive research through several excellent resources for these methods [125, 126], four of the most widely cited and best performing methods with publicly available implementations were selected. Descriptions of these methods that follow have been kept brief and high-level because of the empirical nature of this work and the target audience.

The first of these models is UNet [7] from the medical imaging community. It was

introduced for neuronal structure segmentation in electron microscopic images and won the ISBI challenge 2015. As the name suggests, UNet combines a contracting part with a symmetric expanding part to yield a U-shaped architecture that can both utilize contextual information and achieve good localization owing to the two parts respectively. It was shown to be trainable with relatively few training samples while relying heavily on patch based augmentation which seemed to make it an ideal fit for this study.

The second network is called SegNet [9] and was introduced for segmenting natural images of outdoor and indoor scenes for scene understanding application. It uses a 13-layer VGG net [8] as its backbone and features a somewhat similar architecture as UNet. The contracting and expanding parts are called encoder and decoder respectively. The upsampling units in the decoder are not trainable, instead sharing weights with the corresponding max-pooling layers in the encoder. Keras [127] implementations were used for both UNet and SegNet, available as part of the same repository [128][1].

The third method is called DeepLab [131] and is one of the best performing methods in the Tensorflow research models repository [132]. It uses convolutions with upsampled filters - the so called atrous convolutions [133] - both to achieve better control over the feature response resolution and to incorporate larger context without increasing the computational cost. It also uses pyramidal max pooling to achieve scale-invariance and combines its last layer output with a fully connected conditional random field layer to improve localization accuracy while maintaining spatial invariance. This work uses a more recent version called DeepLabv3+ [5] which adds a decoder module to produce sharper object boundaries and uses the powerful Xception backbone architecture [6] for further performance improvements. DeepLab was also tested with two newer backbone architectures: Auto DeepLab [134], a segmentation specific architecture discovered by neural architecture search [14, 15], and a modified version of ResNet101 [12] where the first 7×7 convolution has been replaced by three 3×3 convolutions similar to PSPNet [135]. The three backbones were found to perform quite similarly and the corresponding results have therefore been relegated to Section A.1. The fourth method is based on the DenseNet architecture [10]. To the best of our knowledge, this architecture has not yet been applied for segmentation, but is included here owing to its ability to

[1]This repository also includes two variants of the FCN architecture [129, 130] that were also tested but did not perform as well as UNet and SegNet so are excluded here.

provide comparable performance with a much smaller network size. The basic idea of DenseNet is to connect each hidden layer of the network to all subsequent layers so that the feature maps output by each layer are used as input in all subsequent layers. This allows better multi-resolution feature propagation and reuse, while drastically reducing the total number of parameters and mitigating the vanishing gradient problem. Results are only included here for a 9-layer architecture, though experiments were done using architectures with up to 21 layers which did not provide any performance improvement. As shown in Table 3.1, DenseNet has by far the fewest parameters of the four models and is over two orders of magnitude smaller than the next smallest model.

3.3.3 Data Augmentation and Training

A simple sliding window approach was used to extract a large set of sub-images or patches from each training image. The window was moved by a random stride between 10% to 40% of the patch size P. This process was repeated after applying random rotations to the entire image between 15 to 345 degrees divided into four bands of equal width to allow for multiple rotations for each image. Finally, each patch was also subjected to horizontal and vertical flipping to generate two additional patches. All resultant patches were combined together to create the dataset for each P. For testing a model, patches of size P were extracted from the test image using a stride of P, segmentation was performed on each patch and the results were stitched back to get the final result.

All models were trained and tested using patch sizes $P \in \{256, 384, 512, 640, 800, 1000\}$. DenseNet turned out to perform best with $P = 800$ while all other models did so with $P = 640$. All results in Section 3.4 were therefore obtained using these patch sizes. The 50 labeled images were divided into two sets of 32 and 18 for generating the training and testing/validation images, respectively. Results on some of the unlabeled videos (Section 3.4.3.2) were also generated using models trained on all 50 images.

UNet and SegNet were both trained for 1000 epochs and the training and validation accuracies were evaluated after each. The trained model used for testing was the one with either the maximum validation accuracy or the maximum mean accuracy depending on how well the training and validation accuracies were matched in the two cases. DeepLab was trained for between $100K$ and $200K$ steps.

Batch size of 10 was used for $P = 256$ and 2 for $P \in \{640, 800, 1000\}$, with the latter chosen due to memory limitations. $P = 384$ was tested with batch sizes 6 and 8 while $P = 512$ was tested with 6 and 2. Most tests were conducted using the default stride of 16 with corresponding atrous rates of $[6, 12, 18]$, though one model with $P = 256$ was also trained using Stride 8 with atrous rates of $[12, 24, 36]$.

DenseNet training was a bit more complicated. Simply using all the pixels for training caused the network to rapidly converge to a model that labeled all pixels with the class that had the most training pixels - water in most cases. To get meaningful results, the number of pixels belonging to each of the classes had to be balanced. Therefore $10,000$ random pixels belonging to each class were selected in each epoch using Gaussian sampling, with a different set of pixels chosen each time, and only these were used for computing the loss. An epoch here refers to a single pass over all images in the training set while training the network which consists of multiple such passes till convergence is achieved. Training images with less than $10,000$ pixels in any class were discarded. The number of epochs were between $1000 - 1600$ for all P. In all cases, the performance metrics in Section 3.4.1 were computed on the validation set every 10 epochs and training was stopped when these became high enough or remained unchanged for over 100 epochs.

3.3.4 Ablation Experiments

One of the principal difficulties in training deep models for performing segmentation is the lack of sufficient labeled data due to the tedious and time-consuming manual segmentation of images. This problem is exacerbated in the current task because of the difficulty in distinguishing between the two types of ice that exhibit both high intraclass variation and significant appearance overlap, in addition to arbitrary and difficult to delineate shapes. As a result, a highly desirable attribute of a practically applicable model would be its ability to learn from as few images as possible, including partially labeled ones.

Two different types of ablation experiments were performed in order to explore the suitability of the tested models in this regard. The first one was to train the models using different subsets of the training set. The second one was to consider the labels from only a small subset of pixels in each image to simulate the scenario of partially labeled training data. Note that the input image itself was left unchanged so that the models did have access to all the pixels but the loss function minimized

during training was computed using only the labels from the selected pixels.

SegNet exhibited similar performance patterns as UNet in the ablation tests, while being slightly worse on average, probably because they share the same base network. SegNet results have thus been excluded in Section 3.4.2.2 and 3.4.2.3 for the sake of brevity. All the code and data used for training, augmentation and ablation tests is made publicly available [136][137], along with detailed instructions for using it, in order to to facilitate further work in this domain and easy replication of our results.

3.4 Results

3.4.1 Evaluation Metrics

3.4.1.1 Segmentation

Following evaluation metrics are typically used in image segmentation [130, 138]:

- Pixel accuracy:

$$\texttt{pix\_acc} = \frac{\sum_i n_{ii}}{\sum_i t_i} \tag{3.1}$$

- Mean accuracy:

$$\texttt{mean\_acc} = \frac{1}{C} \sum_i \frac{n_{ii}}{t_i} \tag{3.2}$$

- Mean Intersection-Over-Union (IOU):

$$\texttt{mean\_iou} = \frac{1}{C} \sum_i \frac{n_{ii}}{t_i + \sum_j n_{ji} - n_{ii}} \tag{3.3}$$

- Frequency Weighted IOU:

$$\texttt{fw\_iou} = \sum_k (t_k)^{-1} \sum_i \frac{t_i n_{ii}}{t_i + \sum_j n_{ji} - n_{ii}} \tag{3.4}$$

where C is the number of classes, n_{ij} is the number of pixels of class i predicted to belong to class j ($1 \leq i, j \leq C$), and t_i is the total number of pixels of class i in the ground truth. Note that accuracy metrics measure only the rate of true positives while IOU also accounts for false positives. Hence, **accuracy and IOU**

Table 3.2: Class frequencies (%) in the two test sets indicating the ratio of pixels that are classified as water and the two types of ice. The 18-image test set was used for most of the experiments while the 46-image set was used for the image and pixel-level ablation tests (Sections 3.4.2.2, 3.4.2.3).

test images	water	anchor ice	frazil ice
18	55.7	17.5	26.8
46	61.0	16.6	22.4

are respectively equivalent to recall and precision metrics that are more widely used in pattern recognition and are referenced thus in the remainder of this thesis. Note also that `pix_acc` is a frequency weighted version of `mean_acc` and is thus referred to as **frequency weighted (fw) recall** in Table 3.3.

Using these metrics to measure the combined segmentation performance over all three classes can lead to biased results when the number of image pixels is not evenly distributed between the classes. This is particularly so in the current work whose main objective is to distinguish between the two types of ice. However, as shown in Table 3.2, more than half the pixels in both sets of test images are of water while anchor ice, which is the most difficult to segment, covers only about 17% of the pixels. Therefore, results in the next section are mostly restricted to class-specific versions of these metrics.

3.4.1.2 Ice Concentration

In addition to these segmentation metrics, the estimated ice concentration accuracy has also been used as an additional metric. This is computed through a three-step process. First, column-wise ice concentration is obtained for each frame by computing the percentage of all pixels in each column that are classified as ice (combined, anchor or frazil). These values are stacked together for all columns from left to right to form a vector of the same size as the frame width. Next, the mean absolute error (**MAE**) is computed between the ice concentrations vectors produced by the GT and a model prediction to obtain the overall accuracy for that frame. Finally, the median of MAE values over the entire test set is taken as the final metric. Median has been preferred over mean as being significantly more robust to outliers.

3.4.1.3 Unsupervised

The above metrics are only applicable to labeled images so that unlabeled videos can only be evaluated qualitatively but the conclusions thus obtained are usually somewhat subjective. In order to ameliorate this, an unsupervised metric named **mean ice concentrations difference** has been proposed to measure the consistency of segmentation results between consecutive video frames as a proxy for its accuracy. This metric is computed as the mean absolute difference between the ice concentration vectors of each pair of consecutive frames in the video. Segmentation consistency over the entire video is summarized by taking the average of these differences over all pairs of consecutive frames. The rationale behind this metric is that, since the ice and/or river are moving slowly and the video speed in frames per second (FPS) is fairly high, the ice concentration changes very gradually and its difference between consecutive frames remains small. A model that fails to generalize well to the videos would give inconsistent results in corresponding patches from nearby frames that would therefore result in a high mean concentration difference.

Experiments were also done using direct pixel-wise differences between the segmentation masks themselves, both with and without incorporating motion estimation by optical flow [139, 140, 141]. Results were less consistent, however, and thus excluded here.

3.4.2 Quantitative Results

3.4.2.1 Overview

As shown in Table 3.3, all of the deep models provide significant improvement over SVM for all cases except a couple instances of frazil ice. This is most notable for anchor ice where an increase of $12-20\%$ in recall and $6-20\%$ in precision is achieved in absolute terms, with the respective relative increases being $19-34\%$ and $13-44\%$. It is noteworthy that the two best models – DeepLab and UNet – provide greater performance improvement, in both absolute and relative terms, with respect to precision than recall over all 4 categories. This is particularly impressive since high precision is usually harder to achieve than recall, as testified by its lower values across all models and categories.

Further, we see that DenseNet and SegNet fall slightly behind SVM on frazil ice, especially with respect to recall, even though they have the two highest recalls on anchor ice. This trend of an inverse relationship between anchor and frazil ice recall

Table 3.3: Segmentation recall and precision along with ice concentration median MAE for SVM and all deep models trained and tested on the 32 and 18 image sets respectively. The *fw* in *ice+water (fw)* stands for frequency weighted and the corresponding recall and precision metrics refer to `pix_acc` (Eq. 3.1) and `fw_iou` (Eq. 3.4) as detailed in Section 3.4.1. Relative increase over SVM in recall and precision is computed as (`model_value`−`svm_value`)/`svm_value`×100 while the relative decrease in median MAE is computed as (`svm_mae` − `model_mae`)/`svm_mae` × 100.

Model	anchor ice		frazil ice		ice+water		ice+water (fw)		Median MAE (%)	
	Metric value	Relative increase	Metric value	Relative increase	Metric value	Relative increase	Metric value	Relative increase	Metric value	Relative decrease
Recall (%)									**anchor ice**	
SVM	61.54	-	75.41	-	78.12	-	84.93	-	8.37	-
DeepLab	74.46	21.00	**87.51**	**16.05**	**86.38**	**10.57**	**90.87**	**7.00**	**4.71**	**43.80**
UNet	73.75	19.85	84.27	11.75	85.13	8.97	88.69	4.42	6.51	22.29
DenseNet	76.96	25.06	71.06	-5.77	81.42	4.22	85.02	0.11	7.24	13.59
SegNet	**82.31**	**33.75**	68.99	-8.51	83.06	6.32	85.90	1.14	6.48	22.61
Precision (%)									**frazil ice**	
SVM	43.32	-	63.07	-	65.84	-	76.84	-	7.18	-
DeepLab	**62.39**	**44.03**	**77.14**	**22.32**	**77.25**	**17.33**	**84.32**	**9.72**	**4.52**	**37.01**
UNet	54.89	26.72	71.17	12.84	73.19	11.17	81.73	6.36	6.80	5.22
DenseNet	48.98	13.07	60.97	-3.32	67.69	2.82	77.49	0.84	7.20	-0.30
SegNet	52.80	21.90	62.60	-0.73	69.60	5.72	78.46	2.10	6.64	7.51

was consistently observed in the ablation tests too (Section 3.4.2.2). It seems that learning to better distinguish anchor from frazil ice often comes at the cost of either a decrease in the capability to recognize frazil ice itself or an overcorrection which causes some of the more ambiguous cases of frazil to be misclassified as anchor ice. It is likely that the loss function can be minimized equally well by over-fitting either to frazil or to anchor ice, thus leading to two stable training states.

Comparing between the deep models themselves, DeepLab turns out to be the best overall, followed closely by UNet. It is interesting to note that, while DenseNet and SegNet provide better recall with anchor ice, their corresponding precision is lower. Since recall does not penalize false positives while precision does, this probably indicates that DenseNet and SegNet misclassify frazil as anchor ice more often than DeepLab and UNet, which is consistent with the inverse relationship hypothesis. Further, it can be seen that the performance difference between deep models and SVM decreases when all three classes are considered and even more so when the averaging is frequency weighted. As mentioned before, these are the cases where high segmentation accuracy of water starts to dominate. Finally, the greater difficulty of recognizing anchor ice over frazil ice is confirmed by its significantly lower recall and precision in almost all cases.

Table 3.3 shows the ice concentration estimation accuracy of all the models in terms of median MAE. It should be noted that recall and precision are better indicators of raw segmentation performance since MAE suffers from an averaging effect where false positives can cancel out false negatives to provide an overall concentration value that happens to be closer to the ground truth. As a result, MAE does not always provide a true indication of the actual recognition ability of the model. As an example, SegNet has slightly better MAE than UNet on frazil ice even though it has significantly lower $(9-16\%)$ precision and recall. Nevertheless, except for the single case of DenseNet with frazil ice, the deep models are consistently better than SVM here too, especially on anchor ice. As with the segmentation metrics, DeepLab is the best model and provides around 3% absolute and 40% relative improvement over SVM.

3.4.2.2 Ablation study with training images

For this study, models were trained using 4, 8, 16, 24 and 32 images and each one was tested using the same 18-image test set. Results for both anchor and frazil ice are given in Figure 3.2. Contrary to expectation, a distinct pattern of improvement with more images is not shown by most of the models. There is a slight improvement in anchor ice performance but it seems too weak to clearly demonstrate model improvement with increase in training images. A more likely conclusion is that the test set is just too similar to the training set and does not contain enough challenging variation to allow the extra information from additional training images to be reflected in the performance numbers. This is lent some credence by the fact that the 50 labeled images were specifically chosen for their ease of labeling owing to the highly tedious and time-consuming nature of performing pixel-wise segmentations, in addition to the high degree of ambiguity and subjectivity in classifying ice in the more difficult cases. Combined with the fact that they were labeled by the same person, it would not be unusual for them to be similar, both in terms of content and level of challenge.

The inverse relation between anchor and frazil ice recall that was observed in the previous section is apparent here too. For instance, the plot lines for anchor and frazil ice (Figure 3.2 (a),(c)) are virtually reflections of each other for all models including SVM, except perhaps for DeepLab. DeepLab itself exhibits nearly constant recall, though with a clearer upward trend in precision. It is also overall the best model, as in the previous section. SVM shows the strongest improvement in anchor ice recall, along with the corresponding decline in frazil ice, while DenseNet does so

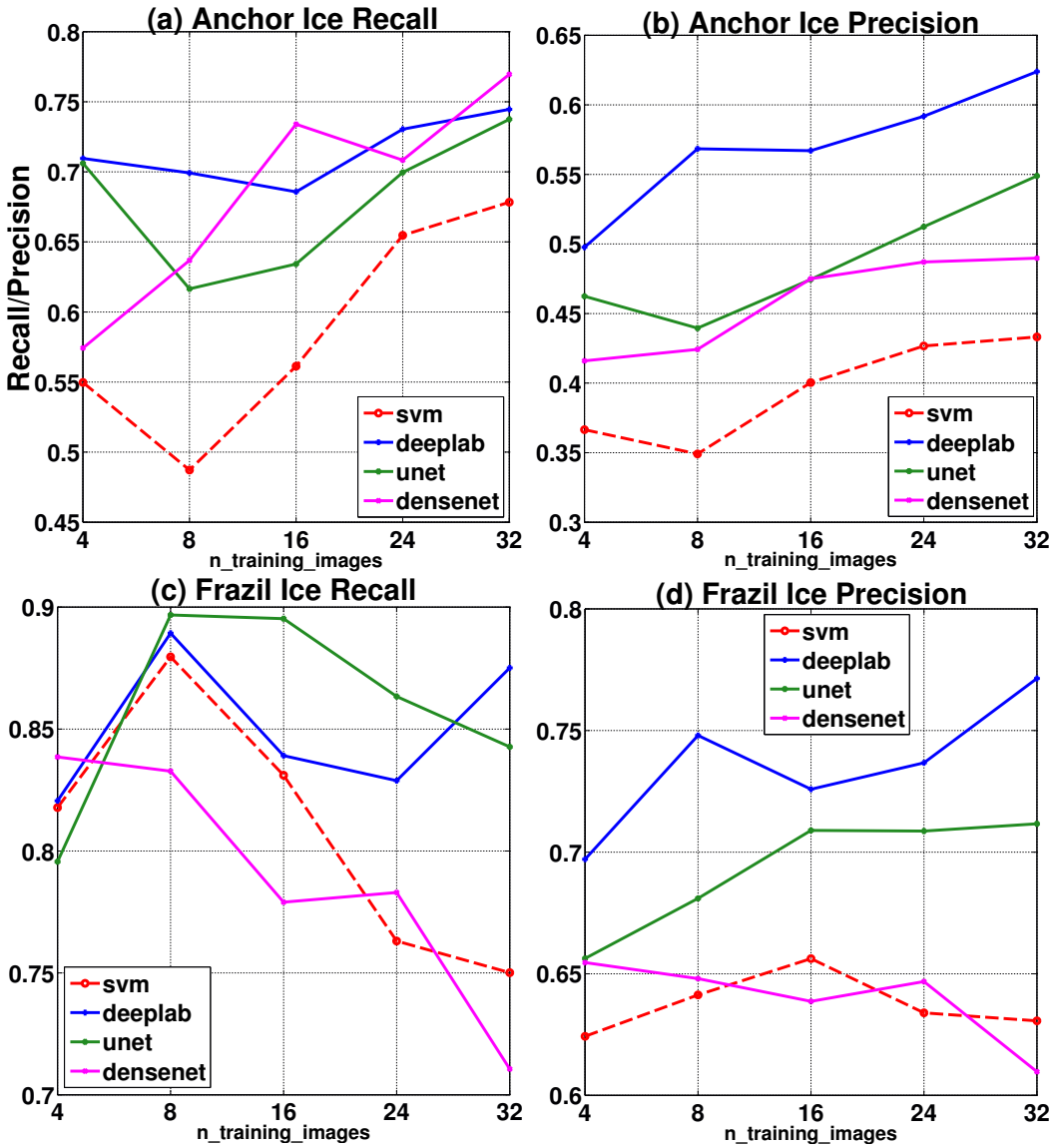

Figure 3.2: Results of ablation tests with training images for (a-b) anchor ice and (c-d) frazil ice. Note the variable Y-axis limits.

among the deep models. Also, Figure 3.2 (a) and (b) illustrate the superiority of deep models over SVM for anchor ice recognition more clearly than Table. 3.3. DeepLab and UNet largely maintain an appreciable superiority over SVM for frazil ice too (Figure 3.2 (c), (d)), though the overall improvement there is less distinct. Finally, DenseNet does seem to be the worst performing deep model, especially for frazil ice, but its competitiveness is still noteworthy considering that it has over two orders of magnitude fewer parameters than the other models (Table 3.1).

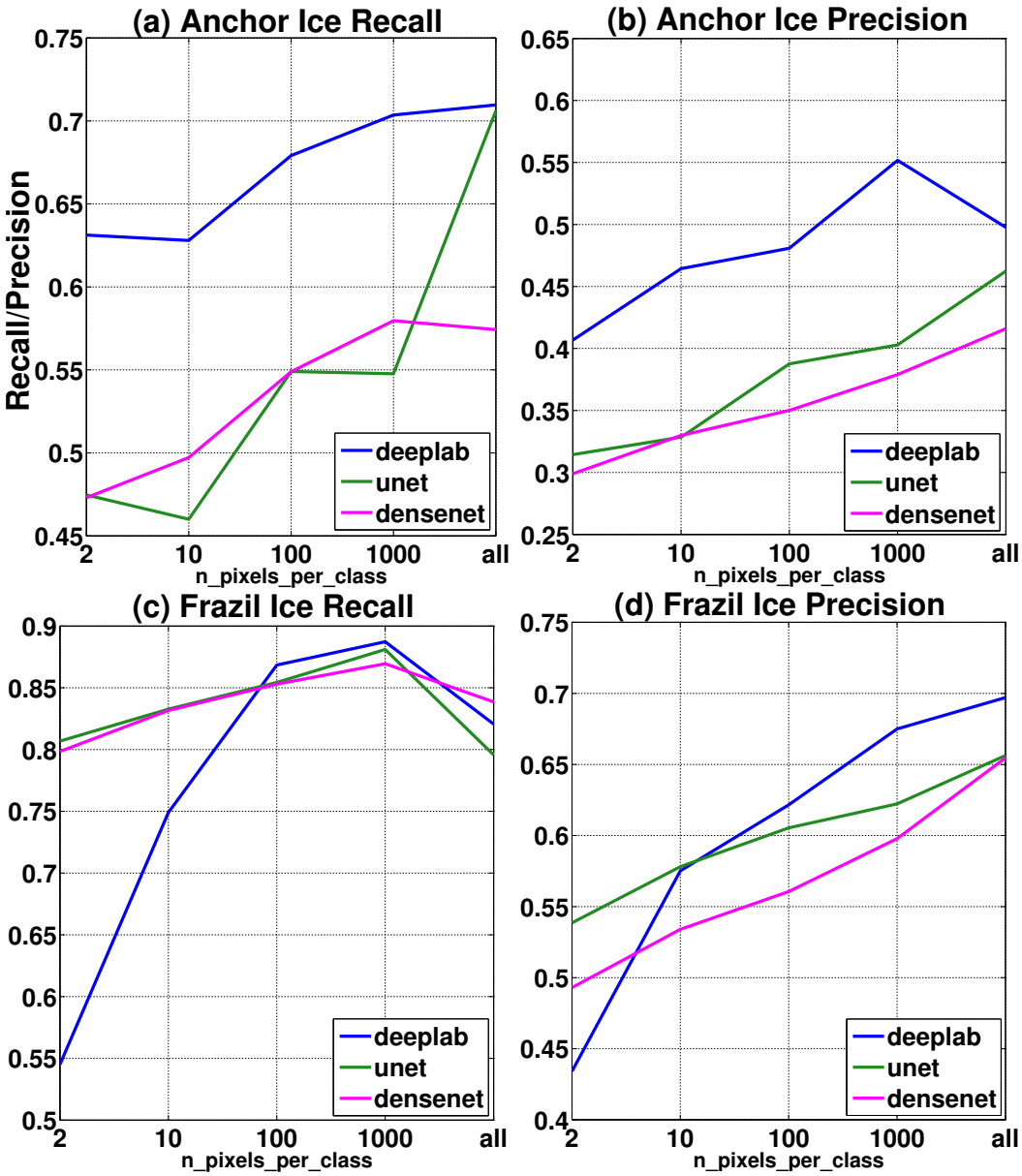

Figure 3.3: Results of ablation tests with selective pixels for (a-b) anchor ice and (c-d) frazil ice. Note the variable Y-axis limits.

3.4.2.3 Ablation study with selective pixels

This study was performed by training models using 2, 10, 100 and 1000 pixels per class selected randomly from each $P \times P$ training patch. The training set was generated from only 4 training images and not subjected to augmentation. Also, $P = 640$ was used for all models including DenseNet to ensure that the number of training pixels

remained identical for all of them. Further, in an attempt to counteract the limited challenges available in the 18-image test set, these models were tested on all of the remaining 46 labeled images. Finally, SVM was not included here because its super-pixel based method [2] does not lend itself well to training using randomly selected pixels.

Results are given in Figure 3.3. It turns out that selective pixel training has surprisingly little impact on quantitative performance, except perhaps in the case of UNet with anchor ice and DeepLab with frazil ice. Though there is a more strongly marked upward trend in performance compared to training images (Figure 3.2), it is not as significant as would be expected. The case of 2 pixels per class is particularly remarkable. When combined with the fact that the unaugmented training set contained only 46 patch images, this training was done using only 92 pixels per class or 276 pixels in all. This might be another indicator of the limited challenges available in the test set. This is further confirmed by the qualitative results on videos (Section 3.4.3.2) that show a much more strongly marked difference than would be inferred by these plots.

3.4.3 Qualitative Results

3.4.3.1 Images

Figure 3.5 shows the results of applying the optimal configurations of the four models (as used in Section 3.4.2.1) to segment several images from the unlabeled test set. Additional results are in Figures A.3 and A.4. Several interesting observations can be made. Firstly, both UNet and SegNet misclassify water as frazil ice in several cases where water covers most of the image, e.g. in image 3. DenseNet too seems to be susceptible to this issue, albeit to a much lesser extent, though a careful examination of its video results (Figure 3.4) shows this problem to be more prevalent than the images alone indicate. Secondly, DeepLab results show the largest degree of discontinuity between adjacent patches due to its tendency to occasionally produce completely meaningless segmentations on some individual patches. Image 5 is an example. Thirdly, and consistent with the quantitative results of the previous section, DenseNet is overall the best performing model, even though its results are slightly more fragmented than the others. This is particularly noticeable in the more difficult cases of distinguishing between frazil and anchor ice when they both form part of the same ice pan. Images 1 and 7 are examples.

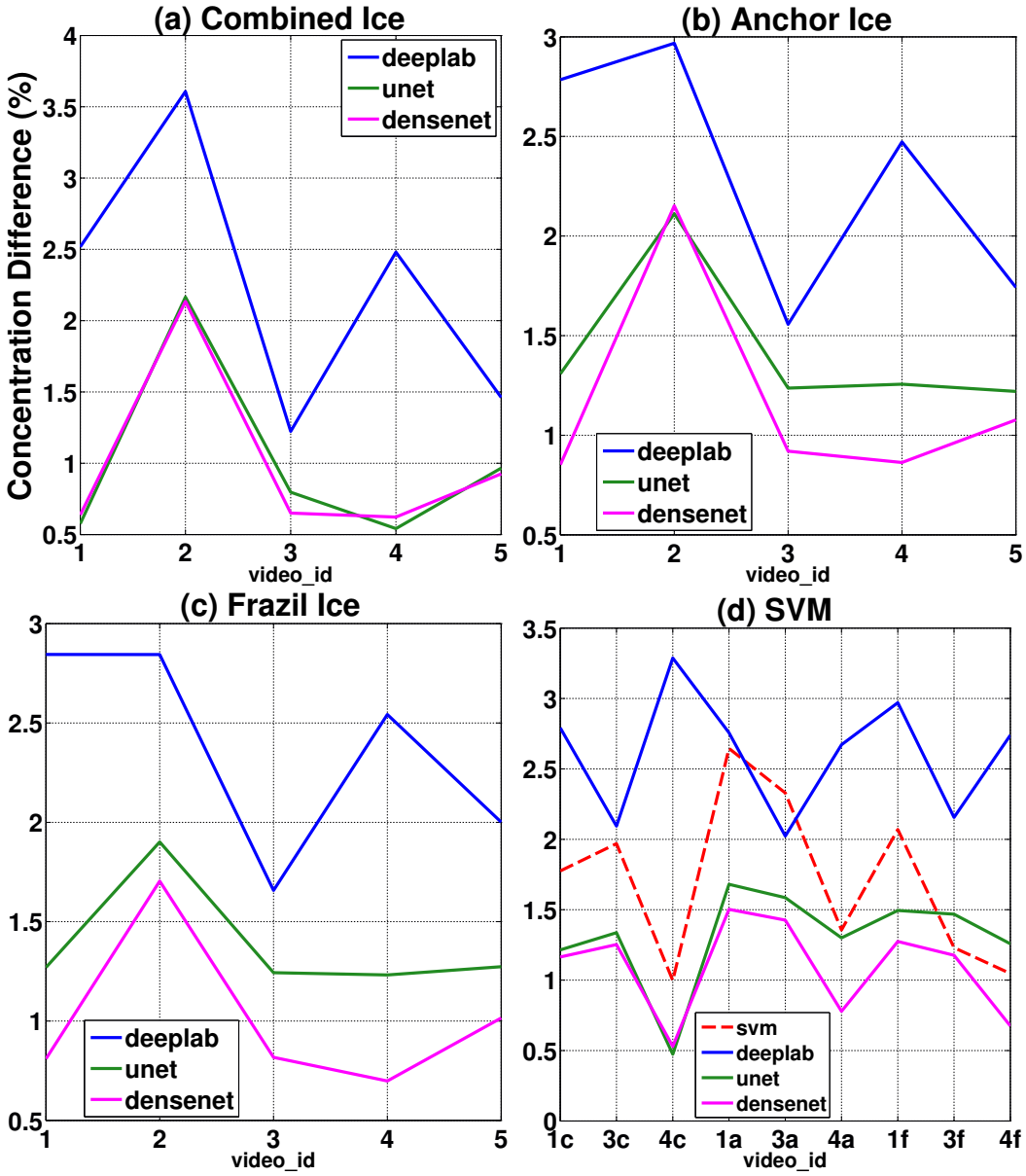

Figure 3.4: Mean ice concentration difference between consecutive frames for (a) both types of ice combined, (b) anchor ice, (c) frazil ice and (d) all three for SVM sequences. Details of videos corresponding to the IDs are in Table A.3 and suffixes `c`, `a` and `f` in (d) respectively refer to combined, anchor and frazil ice.

Qualitative results on labeled test image are available in the accompanying data [137] as well as in Google Photos albums whose categorized links are given in Table A.1 for convenience.

3.4.3.2 Videos

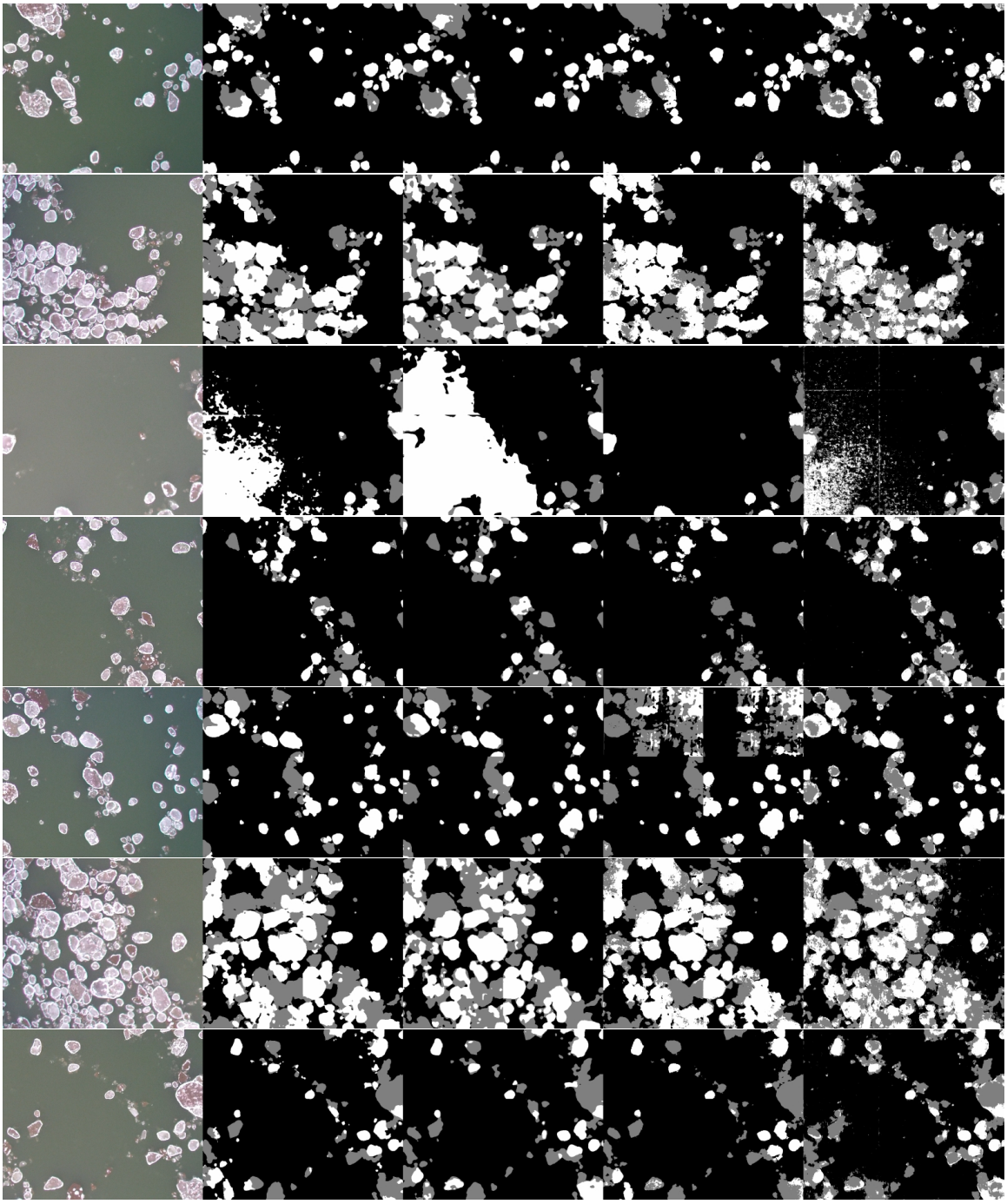

Figure 3.5: Results of testing the deep models on unlabeled images: left to right: raw image, UNet, SegNet, DeepLab, DenseNet Individual rows are referred to in the text by numbers from 1-7 representing rows from top to bottom.

All the deep models were evaluated on 1 to 2 minutes sequences from 5 videos captured on 3 different days and containing wide variations in the scale and form of ice pans. SVM took 5 minutes to process each frame so could only be evaluated on 30 seconds of video 1 and 10 seconds each of videos 3 and 4. Also, selective pixel models were only evaluated on videos 1 and 3. Results for all videos are available in the accompanying data [137]. Tables A.3 and A.4 respectively provide details of the tested sequences and categorized links for some of the results.

The most noticeable point in these results is that DeepLab is susceptible to completely misclassifying randomly distributed individual patches, which can lead to strong discontinuities when these patches are stitched together to create complete frames and the corresponding video. This is quantitatively confirmed by Figure 3.4 that shows the mean ice concentration difference between consecutive frames in all of the videos for each of the two types of ice as well as both combined. It can be seen there that DeepLab has significantly higher values than both UNet and DenseNet, being more than 3 times higher in several cases, while DenseNet almost always has the smallest difference, thus indicating the most consistent segmentation results. Apart from confirming the limited challenges available in the labeled test sets, this inversion of relative performance between the 3 models as compared to the quantitative results in Section 3.4.2, shows that DenseNet is able to generalize to new scenarios much better than DeepLab. The latter has a tendency to overfit to the training images while UNet provides a good balance between generalization and overfitting. Figure 3.4 (d) shows the concentration differences over the 3 sequences on which SVM was also tested. SVM classifies pixels in groups of super pixels so it doesn't suffer from the issue of misclassifying entire patches that the deep models are susceptible to. Even so, its generalization ability is poor enough to give significantly higher concentration differences as compared to both DenseNet and UNet, though being slightly better than DeepLab. Moreover, this group classification technique has the disadvantage of giving blocky appearance to its segmentation masks, whose boundaries are often too coarse to correspond well with the actual ice pans.

Examination of the ablation test videos gives another important result that was mentioned in Section 3.4.2.3 - selective pixel training has significantly greater impact in practice than indicated by Figure 3.3. The segmentation masks seem to become more grainy and sparse as the number of pixels is decreased and there is a very noticeable difference between using 2 and 1000 pixels. Similarly, a greater difference

is apparent between the results produced by models trained on 4 and 32 images than suggested by Figure 3.2.

3.5 Conclusions

This chapter presented the results of using four state of the art deep CNNs for segmenting images and videos of river surface into water and two types of ice. Three of these models - UNet, SegNet and DeepLab - are previously published and well studied methods while the fourth one - DenseNet - is a new method, though based on an existing architecture. All of the models provided considerable improvements over previous attempts based on SVM. These were particularly significant for the most challenging case of anchor ice where around 20% increase in both recall and precision were obtained by the four models combined. The single best model - DeepLab - provided respective improvements of $13-19\%$ in absolute and $21-44\%$ in relative terms. Frazil ice performance was slightly less impressive but still surpassed SVM by $12-14\%$ in absolute and $16-22\%$ in relative terms. Significant improvements were obtained in ice concentration estimation accuracy too, with DeepLab providing around 3% absolute and 40% relative decrease in MAE over SVM for both types of ice.

Among the four models, DeepLab gave the best quantitative performance on the labeled test set. It provided $5-10\%$ improvement in precision while maintaining similar recall rates for anchor ice and $3-15\%$ better recall and precision for frazil ice. However, it showed poor generalization ability by giving the worst qualitative results on the unlabeled images and videos - up to 7-fold mean concentration difference on the videos. DenseNet, on the other hand, gave relatively poor quantitative results but demonstrated excellent generalization ability on the unlabeled data. UNet provided a good balance between the two and might be taken to be the single best model tested here, if such a one needs to be chosen.

Chapter 4

Animal Detection

This chapter is adapted from [71] and details my application of conventional deep learning based object detection, instance segmentation and object tracking methods for animal detection in man-made environments. This is the second of the three projects where I applied conventional modeling to solve novel problems. This chapter covers only conventional models and the comparison of these models with language models is deferred to Sections 8.4.2.1 and 8.5. This work was done during two industrial internships at a local non-profit named ACAMP which closed in 2023.

4.1 Introduction

Object detection is an important field in computer vision that has seen very rapid improvements in recent years using deep learning [142, 143, 144]. Most detectors are trained and tested on benchmark datasets like COCO [89], Open Images [145], KITTI [146] and VOC [147]. In order to apply these in a particular domain like animal detection, a model pre-trained on one of these datasets is fine-tuned on domain-specific data, usually by training only the last few layers. This is known as transfer learning [148, 149] and is often enough to obtain good performance in the new domain as long as it does not differ drastically from the original. The goal of this work is to use transfer learning to adapt state of the art object detection methods for detecting several types of large Alberta animals in real-time video sequences captured from one or more monocular cameras mounted on moving ground vehicles. The animals that most commonly stray into human habitations include: deer, moose, coyotes, bears, elks, bison, cows and horses. There are two deployment scenarios:

- Detecting threats in an autonomous all-terrain vehicle (ATV) patrolling the

Edmonton International Airport perimeter for security and surveillance purposes.

- Finding approaching animals in side–mounted cameras on buses plying the Alberta highways, in order to issue a timely warning to the driver for collision avoidance.

The main challenge here is the scarcity of existing labeled data with sufficient specificity to the target domain to yield good models by fine-tuning pre-trained detection networks. Although several of the large public datasets like COCO [89] do include some of the more common animals like bears and horses, these rarely include the Canadian varieties that are the focus of this work and often feature incorrect backgrounds. Even larger classification datasets like Imagenet [150] do include images of many of the target animals but only provide image level labels so the bounding boxes would have to be added manually. There are also several animal specific datasets [151, 152] but these likewise fall short of meeting the target requirements. For example, they have aerial viewpoints [153, 154], incorrect species [155, 156, 157, 158, 159, 160, 161] or habitats [162, 163] and no bounding box annotations [164, 160, 165, 166].

The lack of training data was addressed by collecting and labelling a sufficiently large number of images of the target animals. This was initially confined to videos since labeling videos was easier to semi-automate (Section 4.3.1) and training detectors on videos showing the animals in a variety of poses seemed to concur better with deployment on camera videos captured from moving vehicles. However, tests showed that detection performance is far more sensitive to the range of *backgrounds* present in the training set rather than variations in the appearance of the animal itself (Section 4.4). Though static images helped to resolve this to a certain extent, they are much harder to obtain in large numbers and a lot more time-consuming to label. More importantly, neither static nor video images of animals are easy to acquire with the kinds of structured man-made surroundings that the airport perimeter and highways present. This chapter thus proposes a solution based on synthetic data generation using a combination of interactive mask labelling, instance segmentation and automatic mask generation (Section 4.3.4).

Another significant challenge is the need for the detector to be fast enough to process streams from up to 4 cameras in real time while running on relatively low-power machines since both the deployment scenarios involve mobile computation where limited power availability makes it impractical to run a multi-GPU system. This is addressed using RetinaNet [18] and YOLOv3 [22] which turned out to be

surprisingly competitive with respect to much slower models. To the best of our knowledge, this is also the first large-scale study of applying deep learning for animal detection in general and their Canadian varieties in particular. It presents interesting insights about transfer learning gained by training and testing the models on static, video and synthetic images in a large variety of configurations. Finally, it provides practical tips that might be useful for real world deployment of deep learning models. Code and data are made publicly available to facilitate further work in this field [167].

4.2 Related Work

Animal recognition in natural images is a well researched area with applications mostly in ecological conservation. As in the case of available data, most of the existing work is not closely allied to the domain investigated in this work. Three main categories of methods can be distinguished from the literature corresponding to the type of input images used. The first category corresponds to aerial images captured from unmanned aerial vehicles (UAVs). A recent work [168] introduced an active learning [169] method called transfer sampling that uses optimal transport [170] to handle domain shift between training and testing images that occurs when using training data from previous years for target re-acquisition in follow-up years. This scenario is somewhat similar to the current work so transfer sampling might have been useful here but most of this work had already been done by the time [168] became available. Further, it would need to be reimplemented since its code is not released and the considerable domain difference between aerial and ground imagery is likely to make adaptation difficult. Finally, most domain adaptation methods, including [168], require unlabeled samples from the target domain which are not available in the current case. Other examples of animal detection in UAV images include [171, 172, 154, 153] but, like [168], all of these are focused on African animals.

The second category corresponds to motion triggered camera trap images. These have been reviewed in [173] and [174]. The latter reported similar difficulties in generalizing to new environments as were found here. The earliest work using deep learning was [175] where graph cut based video segmentation is first used to extract the animal as a moving foreground object and then a classification network is run on the extracted patch. A more recent work [176], that most closely resembles ours, tested two detectors - Faster RCNN [177] and YOLOv2 [42] - and reported respective

Class	Videos (Real)		Static Images			Total
	Seq	Images	Real	Syn	Total	
Bear	92	25715	1115	286	1401	27116
Bison	88	25133	0	0	0	25133
Cow	14	5221	0	0	0	5221
Coyote	113	23334	1736	260	1996	25330
Deer	67	23985	1549	286	1835	25820
Elk	78	25059	0	0	0	25059
Horse	23	4871	0	0	0	4871
Moose	97	24800	0	260	260	25060
Total	**572**	**158118**	**4400**	**1092**	**5492**	**163610**

Table 4.1: Annotation counts for both real and synthetic data (**seq**, **syn**: sequences, synthetic)

accuracies of 93% and 76%. However, the evaluation criterion used there is more like classification than detection since it involves computing the overlaps of all detected boxes with the ground truth and then comparing the class of only the maximum overlap detection to decide if it is correct. Other recent works in this category, most of them likewise dealing mainly with classification, include [178, 166, 165, 179, 180, 181].

The third category, which includes this work, involves real-time videos captured using ground-level cameras. An important application of such methods is in ethology for which many general purpose end-to-end graphical interface systems have been developed [182, 183, 184, 185, 186]. Methods specialized for particular species like cows [187], beef cattle [188] and tigers [157] have also been proposed, where the latter includes re-identification that is typically done using camera trap images. Surveillance and road safety applications like ours are much rarer in the literature and it seems more common to employ non-vision sensors and fencing/barrier based solutions, probably because many animal vehicle collisions happen in the dark [189]. Examples include infrared images [190], thermal and motion sensors [191], ultra wide band wireless sensor network [192] and kinect [193].

Figure 4.1: Sample collected images: (clockwise from top left) bear (Calgary Zoo), deer (Google Images), coyote (Google Images), elk (Nature Footage), horse (YouTube), cow (YouTube), moose (YouTube) and bison (Calgary Zoo)

4.3 Methodology

4.3.1 Data Collection

To facilitate the large number of training images needed, a combination of video and static images was used. Video was collected both directly with handheld video cameras around Calgary area, such as the Calgary Zoo, as well as online via YouTube and Nature Footage [194]. Due to the large quantity of static images that we required, downloading them one by one was not feasible. Instead, ImageNet [150] was used as it provides a searchable database of images with links, so they can be downloaded in bulk using scripts. However, not all animal species are available there and not all available ones have enough images. Google Images was thus also used by searching for specific taxonomic classification and downloading the results in bulk using browser extensions. After downloading static images, it was necessary to verify that all were of the intended animal and remove any mislabeled or unhelpful images. Figure 4.1 shows sample images of all animals while Table 4.1 provides quantitative details. We make this dataset publicly available as the ACAMP Canadian Animal Detection (ACAD) dataset.

4.3.2 Labeling

4.3.2.1 Bounding Boxes

Annotation was done using a heavily modified version of an open source image annotation tool called LabelImg [195]. This tool takes a video file or sequence of images as input and allows the user to annotate bounding boxes with class labels over them. The SiamFC tracker [196] was integrated with the tool to make video annotation semi–automated so that the user only needs to manually annotate the animal in the first frame, track it till the tracker starts drifting, fix the box in the last tracked frame, start tracking again and repeat this process till all frames are labeled.

4.3.2.2 Segmentation Masks

Pixel wise masks were needed to generate high-quality synthetic data (Section 4.3.4). Annotation tools that support masks do exist [197, 198, 199, 200, 201, 202, 203], including AI assisted services [204], but all have issues such as too coarse masks [197, 198, 199, 200], Linux incompatibility [202], paid or propriety license [204, 203] or cloud-data restriction [201]. Also, it was desirable to semi-automate mask generation using the already existing bounding boxes, which is not allowed by any of the tools. Mask annotation functionality was thus added to the labelling tool with support for 3 different modalities to add or refine masks - drawing, clicking to add boundary points and painting with variable sized brushes.

Semi-automated mask generation was done using a combination of motion based interpolation, edge detection and tracking. An approximate mask is generated for a given frame by estimating the motion between its bounding box and that in a previous frame whose mask has already been annotated. In addition, holistically nested edge detection (HED) [205] followed by adaptive thresholding is used to obtain a rough boundary of the animal that can be refined by painting. Finally, the SiamMask tracker [206], that outputs both bounding boxes and segmentation masks, was integrated with the labelling tool to generate low-quality masks in a fully automated manner. Mask labelling was a slow and laborious task and took anywhere from 1 - 8 minutes per frame depending on animal shape and background clutter. An arguably more sophisticated pipeline for rapid generation of segmentation masks has recently been proposed [207]. However, it became available too late to be utilized in this project, does not provide a publicly available

implementation and its proposed pipeline includes human involvement on too large a scale to be practicable here. A recent video mask prediction method [208] likewise came out too late and also rendered unnecessary by SiamMask.

4.3.3 Object Detection

Object detection has improved considerably since the advent of deep learning [142] within which two main categories of detectors have been developed. The first category includes methods based on the RCNN architecture [79, 209] that utilize a two-step approach. A region proposal method is first used to generate a large number of class agnostic bounding boxes that show high probability of containing a foreground object. Some of these are then processed by a classification and regression network to give the final detections. Examples include Fast [38] and Faster [177, 11] RCNN and RFCN [16]. The second category includes methods that combine the two steps into a single end to end trainable network. Examples include YOLO [41, 42, 22], SSD [19] and RetinaNet [17, 18]. Apart from its high-level architecture, the performance of a detector also depends on the backbone network used for feature extraction. Three of the most widely used families of performance-oriented backbones include ResNet [12, 210, 211], Inception [212, 20, 6, 13] and Neural Architecture Search (NAS) [14, 15]. Several architectures have also been developed with focus on high speed and low computational requirements. The most widely used among these are the several variants of MobileNet [213, 21, 214].

Five high level detector architectures have been used here – Faster RCNN, RFCN, SSD, RetinaNet and YOLO. Three different backbone networks are used for Faster RCNN - ResNet101, InceptionResnetv2, NAS - and two for SSD - Inceptionv2 [20], Mobilenetv2 [21] - for a total of 8 detectors. ResNet101 and ResNet50 are used as backbones for RFCN and RetinaNet respectively. All 3 variants of YOLO [41, 42, 22] were experimented with, though only YOLOv3 [22] results are included here as being the best. These methods were chosen to cover a good range of accuracies and speeds among modern detectors.

All of the above are *static* detectors that process each frame individually without utilizing the temporal correlation inherent in video frames. Detectors have also been developed to incorporate this information for reducing missed detections due to issues like partial occlusions and motion blur. Examples include Seq-NMS [215], TCNN [216, 217], TPN [218], D&T [219] and FGFA [220, 221]. However, none of these have

Figure 4.2: Synthetic data samples with corresponding source and target images for (top) coyote on airport and (bottom) moose on highway. Each row shows (left to right) animal source image, target background image and crops of synthetic images generated using (clockwise from top left) manual labeling, Mask RCNN, Gaussian blending (no mask) and SiamMask.

compatible implementations and most need either optical flow, patch tracking or both to run in parallel with a static detector which makes them too slow to be used here. LSTM-SSD [222, 223] is the only recent video detector that is both fast and open source but attempts to incorporate this here showed its implementation [224] to be too buggy and poorly documented to be usable without significant reimplementation effort not warranted by the modest improvement it seemed likely to provide. Instead, a simple algorithm was devised to combine the DASiamRPN tracker [225] with YOLO (Section 4.4.2.6) to gauge the potential benefit of temporal information in videos.

4.3.4 Synthetic Data Generation

Experiments showed that detectors have limited ability to generalize to new backgrounds (Section 4.4.2.1). A solution considered first was to collect static images with as much background variation as possible to cover all target scenarios. This proved to be impracticable due the difficulty of finding and labeling sufficient quantities of static images, exacerbated by our target scenarios consisting of man-made environments where it is extremely rare to find animals at all. As a result, synthetic data was generated by extracting animals from existing labeled images and adding them to images of the target backgrounds. Attempts were initially made to do this without masks by selecting only the best matching source

images for each target background through techniques like histogram matching and then using Gaussian blending to smoothen the transition from source to target background. However, this failed to generate images that could either be perceived as realistic by humans or improve detection performance (Section 4.4.3). Pixel wise masks were therefore generated by manually labelling a sparse collection of frames with as much background variation as possible and then training instance segmentation models (Section 4.3.5) to automatically generate masks for remaining frames with similar backgrounds. SiamMask tracker [206] was also used towards the end of the project to make this process fully automated. Generating synthetic images was much faster than labelling masks and only took about 1-10 seconds/frame. Most of the effort was focused on generating static images since experiments (Section 4.4.2.2) showed that videos do not help to improve detectors much. It is also significantly harder to generate realistic videos as that requires camera motion in the source and target video clips to be identical. Images were generated from 14 airport and 12 highway backgrounds with 11 source images for bears and deer, and 10 for coyotes and moose. Figure 4.2 shows examples.

4.3.5 Instance Segmentation

Instance segmentation distinguishes between each instance of an object as opposed to semantic segmentation that only identifies categories of objects. The former intuitively seems more suitable for extracting animal boundaries from bounding boxes since it uses object-level reasoning, whereas the latter is more oriented towards pixel-level classification. This was confirmed by experiments with several state of the art semantic segmentation methods, including DeepLab [131, 5], UNet [7] and SegNet [9]. All of these generated masks that were too fine-grained to cleanly segment out the animal from its background, instead producing many small groups of background pixels inside the animal and, conversely, animal pixels within the background. Three instance segmentation methods were then considered – SharpMask/DeepMask [226, 227], Mask RCNN [228] and FCIS [229]. Mask RCNN was found to produce the highest quality masks so only its results are included.

4.3.6 Implementations and Training

Table 4.2 lists all implementations used here. Training was done by fine tuning models pre-trained on large benchmark datasets – COCO [89] for Mask RCNN and

Table 4.2: Implementation details for the various methods (TF: Tensorflow, PT: PyTorch)

Methods	**Implementations**
All static detectors except YOLO	TF (Object Detection API) [230]
YOLOv3, YOLOv2, YOLOv1	PT [231], TF [232], Darknet [233]
Mask RCNN, Sharpmask, FCIS	TF [230], TF [234], MXNet [235]
SiamFC, SiamMask, DASiamRPN	TF [236], PT [237], PT [238]
DeepLab, UNet/SegNet, HED	TF [132], Keras [128], OpenCV[239]

Table 4.3: Class configurations for training (**c** refers to the number of classes)

c	**Animals**	**Comments**
6	all except cow, horse	these have only ∼5K images
4	bear, deer, moose, coyote	synthetic images
3	bear, deer, coyote	real static images

all detectors; ImageNet [150] for Sharpmask and FCIS; ADE20K [240] for DeepLab, UNet and SegNet. HED and all trackers were used directly with pretrained weights without any fine tuning.

In order to avoid class bias while training, number of samples from all classes must be similar [241]. Number of labeled images, however, varies significantly between animals (Table 4.1), especially when the source type – video or static – is taken into account. Therefore, experiments were done with 3, 4 and 6 classes (Table 4.3) in addition to all 8 to cover a range of scenarios while maintaining class balance.

4.4 Results

4.4.1 Evaluation Metrics

Object detectors are usually evaluated using their **mean average precision (mAP)** [242], defined as the mean, over all classes, of the area under the recall–precision curve for each class. Although a good measure of the overall threshold-independent performance, mAP may not accurately represent deployment accuracy where a single threshold must be chosen. Since mAP considers the variation of recall and precision with threshold *separately for each class*, and this can differ greatly between classes (Figure 4.3 (c)), it is more indicative of accuracy when a different threshold can be chosen for each class to optimize the recall-precision characteristics for that class. It is also difficult to interpret mAP to gauge the practical usability of a detector in terms of how likely it is to miss objects or give false detections. This chapter therefore proposes

Table 4.4: Training configurations for both real and synthetic data (**c**, **img**, **seq** respectively refer to the number of classes, images, and sequences).

#	c	Details	Train	Test
			img (seq)	img (seq)
1	8	1K video images/class sampled from complete sequences	8001 (33)	150117 (539)
2	8	1K video images/class sampled evenly across all sequences	8156	149962
3	6	10K video images/class sampled from complete sequences	60003 (218)	88023 (317)
4	6	5% images from the start of each video sequence	7169	140857
5	3	500 static images/class	1500	2900
6a-6d	6	1, 2, 5, 10 images sampled evenly from each of 67 sequences	402, 804, 2010,4020	103235
7	3	20K video images/class tested on static images	60000	4400
8a, 8b	3	1K static images/class tested on video, synthetic images	3000	73034, 598
9	4	20K video images/class tested on synthetic images	80008	780
10a, 10b	3, 4	3, 4 class models trained on 28% of synthetic images, tested on rest	234, 312	598, 780

another metric obtained by first averaging recall and precision for each threshold over all classes and then taking the **recall–precision (RP)** value at the threshold where the two are equal. This metric is named **mean Recall-Precision (mRP)** and provides a more interpretable measure of performance when using a single threshold for all classes.

Further, this work deals mainly with human-in-the-loop type security applications where detections alert humans to take suitable measures after verification. In such cases, simply detecting an object can be far more crucial than classifying it correctly. For example, when used as an early warning system for bus drivers, misclassification would have little impact on the driver's response as long as the animal is detected early enough. A less stringent evaluation criterion named **class-agnostic Recall-Precision (cRP)** is thus also used that treats all animals as belonging to the same class so that misclassifications are not penalized.

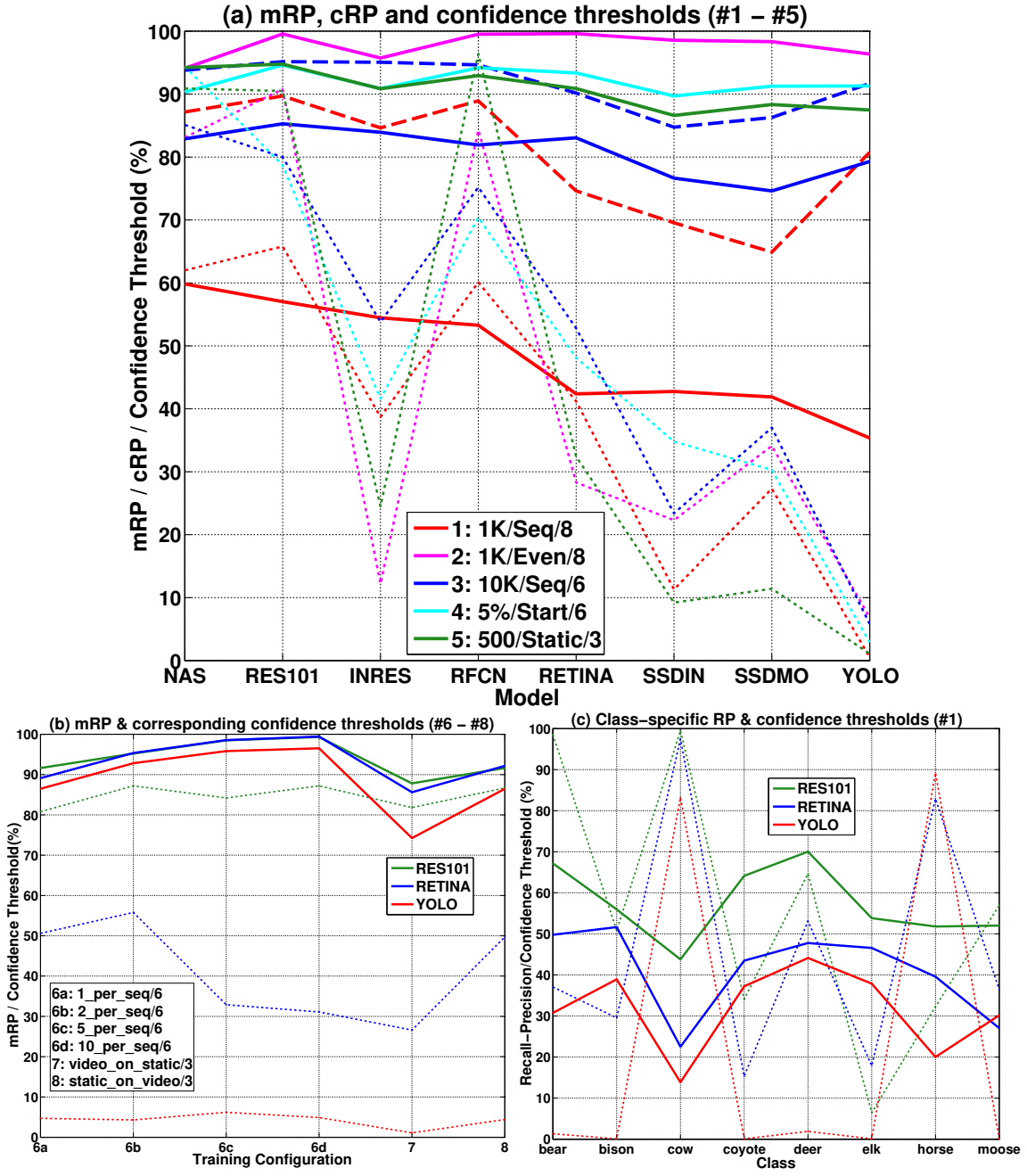

Figure 4.3: Detection mRP (solid), corresponding confidence thresholds (dotted) and cRP (dashed, only #1, #3): (a) #1 - #5 for all 8 models (b) #6 - #8 for 3 models (c) class-specific #1 results for 3 models. Model Acronyms: **NAS**, **RES101**, **INRES** - Faster RCNN w/ NAS, ResNet101, Inception-ResNetv2 backbones; **RFCN** - RFCN w/ ResNet101; **RETINA** - RetinaNet w/ ResNet50; **SSDIN**, **SSDMO** - SSD w/ Inceptionv2, MobileNetv2; **YOLO** - YOLOv3. Best viewed under high magnification.

4.4.2 Real Data

4.4.2.1 How well do detectors generalize ?

Figure 4.3 summarizes the results for several training and testing configurations (Table 4.4) used to study the generalization ability of detectors in a range of scenarios. These are henceforth referred to by their **numeric IDs** (first column of Table 4.4) and detectors by **acronyms** (Figure 4.3) for brevity.

Figure 4.3 (a) gives results for all detectors in #1 - #5. The large difference between #1 and #2 clearly demonstrates the inability of detectors to generalize to unseen backgrounds. Both have 1K video images/class but the latter has these sampled from all sequences to allow training over nearly all backgrounds in the test set while the former does not get any frames from the test sequences. This is sufficient for the detectors to achieve near perfect mRPs in #2 while giving far poorer performance with only 35-60% mRP in #2. A similar trend is seen, though to a lesser extent, in #3 and #4. The former, with 10K images/class from complete sequences, is significantly outperformed by the latter with only 5% images from the start of each sequence (or $\sim$1.2K images/class). The smaller difference here is attributable to the much greater frame count in #3 and the fact that #4 uses consecutive frames from each sequence which contain a smaller range of backgrounds than the evenly sampled frames in #2. Performance in #5 is comparable to #4, even though #5 involves testing over a far greater proportion of unseen backgrounds, probably because most static images depict animals in their natural habitats (Section 4.3.1) which, exhibiting limited variability, allow the detectors to generalize relatively well.

Figure 4.3 (a) also shows cRP, though only for #1 and #3 since remaining configurations all had cRP $> 90\%$ whose inclusion would have cluttered the plots so these have been deferred to Table B.2. As expected, cRPs are significantly higher than mRPs for all models, though the gain is most notable for YOLO, particularly in #1 where it more than doubles its performance, outperforming both the SSDs as well as RETINA. This suggests, and qualitative examination has confirmed, that the form of overfitting YOLO is susceptible to involves associating backgrounds to specific animals whose training images had similar backgrounds. For example, if a particular scene is present in bear training images but not in those of deer, a test image of a similar scene, but containing deer, would have the animal detected as bear. The other models are susceptible to this too but to a smaller degree and more

often miss the animal altogether.

4.4.2.2 How much are video annotations worth ?

Figure 4.3 (b) shows results for #6 - #8; only 3 detectors are included to reduce clutter since the others showed similar performance patterns. #6 involved training with 1, 2, 5 and 10 frames/sequence, with the sequence count limited to 67 by the class with the fewest sequences (deer) to maintain class balance. All 4 models were tested on the same 67 sequences using frames not included in any of their training sets. It can be seen that even 1 frame/sequence is enough for all detectors to give 90% mRP, which improves only marginally with 2 and 5 frames, plateauing thereafter. Further, though RETINA does catch up with RES101 using ≥ 2 frames, YOLO is unable to even narrow the gap, which might indicate that domain specialization cannot entirely overcome architectural limitations. #7 and #8 show the relative utility of video and static frames by training on one and testing on the other. As expected, static models demonstrate far superior generalizability by outperforming the video models by 4-12% mRP even though the latter are trained and tested on 20× more and 16× fewer frames respectively. Performance gap between #7 and #8 is also larger for worse performing models, especially YOLO that has twice the gap of RETINA, which reaffirms its poor generalizability. Finally, the fact that #8 has lower mRP than #6a even though the former has nearly 15× more images with varied backgrounds shows the importance of domain specialization.

4.4.2.3 How do the detector accuracies compare ?

RES101 turns out to be the best overall, though NAS, RFCN and INRES remain comparable in all configurations. NAS even has a slight edge in #1, showing its better generalizability under the most difficult scenarios. Conversely, the shortcomings of 1-stage detectors compared to their 2-stage counterparts are also most apparent in #1. This is particularly notable for RETINA that is comparable to RES101 and significantly better than the other 1-stage detectors in all remaining configs. YOLO likewise performs much poorer relative to the two SSDs while being similar and even better in other configs. This might indicate that 1-stage detectors in general, and YOLO in particular, are more prone to overfitting with limited training data. From a practical standpoint, though, YOLO redeems itself well by its relatively high cRPs, outperforming RETINA in both #1 and #3.

Table 4.5: Speed, GPU memory consumption and maximum batch size for each detector. Refer Figure 4.3 for model names. (Setup: Titan Xp 12GB, Threadripper 1900X, 32GB RAM)

Model	Batch Size 1		Batch Size 4		Max Batch Size	
	memory (MB)	**speed (FPS)**	**memory (MB)**	**speed (FPS)**	**batch size**	**speed (FPS)**
NAS	9687	1.36	-	-	3	1.39
INRES	7889	3.95	8145	4.68	8	4.49
RES101	5077	19.61	5589	25.35	36	27.12
RFCN	5041	19.8	5553	32.12	76	26.94
RETINA	4785	31.5	5553	43.51	120	53.28
YOLO	1487	71.41	2039	104.25	48	119.64
SSDIN	3631	68.35	3631	155.63	160	181.66
SSDMO	1999	78.67	2031	167	480	246.56

4.4.2.4 How important is the confidence threshold ?

Figure 4.3 shows confidence thresholds corresponding to mRP or class-specific RP using dotted lines. Figure 4.3 (c) shows that the threshold corresponding to the class-specific RP varies widely between classes - much more than the RP itself. As mentioned in Section 4.4.1, this motivates the use of mRP instead of mAP as a practical evaluation criterion. Further, Figure 4.3 (a, b) show that the optimal mRP threshold itself varies greatly between the detectors too. Therefore, choosing a single threshold for all of them might not provide a true picture of their relative performance in practice. It is also evident, especially in Figure 4.3 (b), that a weak correlation exists between the relative performance and threshold, with better performing detectors usually also having higher thresholds. Notable exceptions to this are INRES and SSDIN, both having smaller thresholds than their respective mRP levels. Since both use different variants of Inception, this might be due to an architectural peculiarity thereof. Also notable are the very low thresholds of YOLO - often $< 5\%$ and sometimes even $< 1\%$.

4.4.2.5 How resource intensive are the detectors ?

Since both of our target deployment scenarios of ATV and highway buses involve mobile systems with limited power availability, it is important for the detector to be as lightweight as possible. Table 4.5 shows the speed in FPS along with GPU memory consumption for batch sizes 1 and 4, where the latter is chosen to represent the 4 cameras needed for a simultaneous 360° field-of-view. The maximum batch size

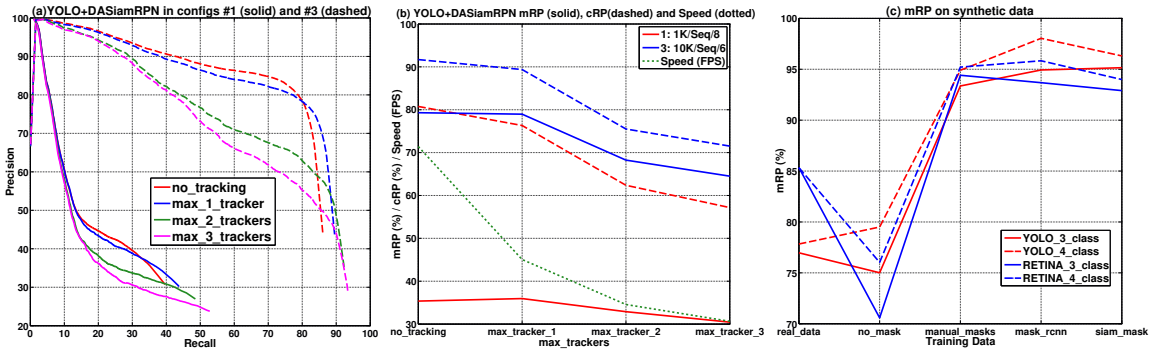

Figure 4.4: Results for (a - b) DASiamRPN + YOLO on config #1 and (c) RETINA and YOLO on synthetic data

that can be run on a 12GB Titan Xp GPU is also shown for scalability comparison. SSDMO turns out to be the fastest, though YOLO is comparable at batch size 1 and also has significantly smaller memory footprint. However, YOLO does not scale as well in either speed or memory and ends up with only a tenth of the maximum batch size of SSDMO and less than half the corresponding speed. NAS and INRES are the slowest and most memory intensive by far and unsuitable for realtime applications. RFCN and RES101 are similar with unit batch size, probably due to their identical backbone, though RFCN scales better, allowing more than twice the maximum batch size and 28% higher speed with batch size 4. Finally, RETINA provides the best compromise between performance and speed. Its mRP is comparable to RES101 in most configs and is fast enough to process 4 camera streams simultaneously at 10 FPS each and thus capture an animal visible for a fraction of a second.

4.4.2.6 Can tracking reduce false negatives ?

As mentioned in Section 4.3.3, tracking was used in an attempt to reduce false negatives by utilizing temporal information in videos. DASiamRPN [225] was used as the tracker since it is one of the fastest available Siamese type trackers. YOLO was used as the detector since its PyTorch implementation was easier to integrate with that of DASiamRPN, its speed with batch size 1 (necessary to use tracking) is among the fastest and its poor performance in #1 provides ample scope for improvement. Algorithm 1 gives the detailed steps for combining the tracker and detector, though the high level idea is simple - associate detections with existing trackers, create new trackers for unassociated detections and remove trackers that remain unassociated for too long or those with the lowest confidence when tracker

count exceeds a threshold. Figure 4.4 (a) shows the mean Recall vs. Precision plots while Figure 4.4 (b) gives mRP / cRP and speeds. Tracking mostly helps only when the detector finds an animal in at least one frame in a sequence and misses it in several subsequent ones. It turns out that this seldom happens in practice so that the resultant increase in recall is very slight and is offset by a significant decrease in precision through tracking false positives. The latter can be mitigated by removing unassociated trackers frequently but this leads to a large drop in recall and is therefore not used in these results. There is thus no net gain in mRP/cRP using tracking, rather significant drops with > 1 trackers. When combined with the reduction in FPS, it does not seem like an effective way to reduce false negatives.

4.4.2.7 Can multi-model pooling reduce false negatives ?

Another way to reduce missing detections is to run multiple detectors simultaneously and pool their outputs. A large variety of methods were explored to pool YOLO, SSDIN and SSDMO but none managed to increase recall enough to offset the fall in precision and the net mRPs were even worse than those from tracking. Descriptions of these methods and corresponding results have thus been relegated to Section B.3.

4.4.3 Synthetic data

A training set was constructed from synthetic data by selecting images corresponding to 3 animal poses per background, with a different combination of poses selected randomly for each background. All remaining images were used for testing. Table 4.4 denotes the corresponding configs as #10a and #10b for 3 and 4 class models respectively. Corresponding real data configurations are #8b and #9 with 1K static and 20K video images/class respectively. Seperate models were trained for each of the 4 methods of extracting animals from source images (Section 4.3.4) – Gaussian blending, manual masks, Mask RCNN and SiamMask. All were tested on images generated by manual masks.

As shown in Figure 4.4 (c), models trained on synthetic data significantly outperform those trained on real data as long as masks are used. This is remarkable considering that only 78 frames/class were used for the former compared to 1K or 20K for the latter. This reiterates the results in Section 4.4.2.2 where #6a with 67 images outperformed #8a with the same 1K images as #8b. However, unlike there, YOLO does manage to match RETINA here, which suggests that high enough

degree of specialization can indeed overcome its architectural shortcomings. More importantly, there is no perceptible difference in mRP between models corresponding to the three segmentation methods. This shows that even the fully unsupervised and visibly coarse masks from SiamMask provide useful information for training detectors that is comparable to precise manual masks. At the same time, mask quality does indeed matter since the no mask / Gaussian blending models perform even worse than real data.

4.5 Conclusions

This chapter presented a large scale study of animal detection with deep learning where 8 state of the art detectors were compared in a wide range of configurations. A particular focus of the study was to evaluate their generalization ability when training and test scenarios do not match. It was shown that none of the detectors can generalize well enough to provide usable models for deployment, with missed detections on previously unseen backgrounds being the main issue. Attempts to increase recall using tracking and multi-model pooling proved ineffective. Synthetic data generation using segmentation masks to extract animals from images of natural habitats and inserting them in target scenes was shown to be an effective solution. An almost fully automated way to achieve this was demonstrated by the competitiveness of coarse unsupervised masks with precise manual ones in terms of the performance of detectors trained on the corresponding synthetic images. RETINA and YOLO were shown to be competitive with larger models while being sufficiently lightweight for multi-camera mobile deployment.

Chapter 5

Human iPSC Segmentation

This chapter is adapted from [72] and details my application of conventional deep learning based static and video instance segmentation models along with image classifiers for detecting and classifying human induced pluripotent stem cells (iPSCs) in time-lapse microscopy images. This is the last of the three projects where I applied conventional modeling to novel domains. This chapter is focused only on the conventional models and the comparison of these models with language models is deferred to Sections 8.4.2.2, 8.4.3.2 and 8.5. This work was done in collaboration with the Alberta Diabetes Institute.

5.1 Introduction

5.1.1 Motivation

The goal of this work is to apply machine learning to automate the identification of human iPSCs that show promise for clinical cell therapies in regenerative medicine. IPSCs are generated by reprogramming a patient's own cells back in time to make more malleable cells with differentiation potential for generating any cells or tissues of interest. This technology has shown great potential for transforming regenerative cell therapies, drug and disease modeling, tissue repair and regeneration, and personalized gene-corrected products. However, the pipeline for iPSC generation, characterization and cell banking is a highly labor-intensive, time-consuming and costly one. The monetary cost of research-grade iPSC line generation is estimated at USD 10,000-25,000 while that of clinical-grade iPSC line is approximately USD 800,000 based on published reports [243]. The entire process of optimal iPSC line generation and

selection can take up to 35 days and requires a further 3 months to produce large scale iPSCs for therapeutic application in patients.

Additionally, quality control techniques for growing iPSCs to limit inter- or intra-patient iPSC line variability, which is currently assessed manually, remain imperfect in large-scale biomanufacturing. The current solution relies on the judgement of an expert cell biologist, who determines precise iPSC induction, confirms pluripotency based on morphological changes and assesses molecular characterization for multiple clones - all tasks that remain highly effort-intensive and subjectively biased. Manual cell quality control therefore cannot be used to scale up the production of iPSCs and derived products for therapeutic applications. An automated method enabling high-throughput surveillance and validation of cell identity, growth kinetics, and morphological features is desirable throughout the entire manufacturing process. The screening is multifold and needed to not only select optimal cells which have been fully converted to iPSCs during reprogramming stage, but also to exclude unstable and pseudo iPSC contaminants during the expansion stage. Automating this process using machine learning would therefore be ground-breaking in improving iPSC bioprocess efficiency and yield, thereby drastically reducing the time and cost involved in the generation of iPSC-based products for therapeutic applications. This chapter presents some early but promising steps in this direction.

5.1.2 Background

5.1.2.1 iPSC Reprogramming

[244] demonstrated that mouse embryonic or adult fibroblasts can be reprogrammed into pluripotent stem cells by introducing four genes encoding transcription factors, namely Oct3/4, Sox2, Klf4, and c-MYC [245, 246]. Generated stem cells showed similar morphological or functional behavior as embryonic pluripotent stem cells and were thus termed iPSCs. Soon thereafter, [245] reported directed conversion of human fibroblasts into pluripotent stem cells, termed as human iPSCs. With the discovery of Yamanaka's human iPSC technology, patient-derived stem cells have huge potential in regenerative medicine [247]. Human iPSCs show merit not only in delivering any desired cell types for treating degenerative diseases, tissue repairing, disease modeling, and drug screening [248, 249], but they also solve two major problems associated with other pluripotent stem cells such as embryonic stem cells

[246], namely immune tolrenace after transplantation and ethical concerns. However, there still exist technical and biomedical challenges including the risk of teratoma formation and the uncertainty of efficient nuclear reprogramming completeness due to variability and inconsistencies in the selection of optimal cells [246]. There are two major problems to be solved before human iPSCs can be applied as a standardized technique, Firstly, manually monitoring the quality of growing iPSC colonies that is currently practiced does not scale. Secondly, only colonies that satisfy clinical good manufacturing practice (GMP) standards need to be identified for use in downstream applications. Hence, there is an urgent need for automated quality control, thereby also lending it an element of objectivity and standardization.

5.1.2.2 Machine learning in iPSC Recognition

Though many applications of machine learning for iPSC recognition in images have been presented in the literature [250, 251, 23, 252, 253, 254, 255], there are none that include both detection and classification or use time-lapse imaging, which is the object of this study. To the best of our knowledge, [23] presented the method that comes closest to this work though that too differs in several key respects. It utilizes fluorescence imaging and the commercial closed-source IMARIS software to segment cells and it captures 3D shape information that is the basis for extracting morphological features to train the classifier it uses. Our aim is to make open-source cell segmentation possible without fluorescence and with only the 2D pixel data in standard phase-contrast microscpy images.

5.1.2.3 Deep Learning in Visual Recognition

In the past decade, deep learning [256] has been applied extensively in computer vision [257], especially for recognition tasks like image classification [258], object detection [259], instance segmentation [260], semantic segmentation [261], and object tracking [262, 263, 264]. It has likewise seen broad application in medical image analysis [265, 266] including segmentation in general [267] and cell segmentation in particular [268]. The latter is the task most relevant to this work, though cell tracking [269, 270, 271, 272, 273] is also important here. More recently, the advent of transformers [33] has led to significant performance improvements [274] over the CNN-based architectures that had been prominent earlier. [26] proposed the Swin transformer to further improve

the original vision transformer [34] using shifted windows. This currently appears to be the backbone of choice in most state of the art models, though that might change with the recent introduction of ConvNext [24] as a competitive CNN-based alternative. For this project, we needed instance segmentation models, preferably ones that could benefit from the temporal information in time-lapse images. Therefore, we selected top-performing static and video segmentation models (Section 5.2.2) with publicly available code from a popular leaderboard [275]. We also searched through the leaderboards on a couple of cell tracking and segmentation benchmark challenges [273, 276] but failed to find any models with publicly available code.

5.2 Methodology

5.2.1 Data Collection

5.2.1.1 Cell culturing

Cells were cultured in a Class-II biocontainment compliant lab with manipulation of cells in a sterile environment using a laminar flow hood with high efficiency particulate air filtration. Cells were maintained at 37°C with 5% CO_2 within humidified incubators. Human iPSCs reprogrammed from patient-derived peripheral blood mononuclear cells (PBMCs) were previously established and characterized for pluripotency in the laboratory and cryostored in ultra-low temperature freezers for long-term storage. Before starting the time-lapse imaging, cells from a frozen vial were thawed in a 37°C waterbath and cultured on a matrix-coated 6-well plate in iPSC growth media at a seeding density of 100,000 cells per well according to previously published protocol [277]. Once the cells got attached, fresh media was replenished in each well every day prior to the time-lapse imaging which was initiated on day 6, right after cell-seeding.

5.2.1.2 Time-lapse imaging

Time-lapse images were captured at 15-minute intervals on a Nikon BioStudio-T microscope using the NIS-Elements cell observation and image analysis software. The images were captured with a 4x lens using a full plate scanning module. A total of 275 images were captured spanning 68.75 hours. The raw images were 10992×10733 pixels or 714 megapixels in size, though the usable circular cell culture region in the center comprised only about 85 megapixels with a diameter of 5200 pixels. A sample

image is shown in Figure C.2. Manual examination showed that images before frame 146 (or 36.5 hours) were unsuitable for our experiments since they contained too many clones, with most being too small and indistinct for reliable labeling. Three of the remaining frames - 155, 186 and 189 - were blurry due to camera shake and had to be discarded too. As a result, a total of 127 frames were used for all experiments.

5.2.1.3 Annotation

The original 714 megapixels images are too large to be processed directly so they were first divided into several regions of interest (ROIs) (Figure C.2) varying in size from 1700×900 to 4333×3833. These were then annotated in 3 stages.

5.2.1.3.1 Selective Uncategorized Semi-Automated Labeling

The 127 frames were first divided into three sets representing different levels of cell development – 146-200, 201-250 and 251-275. Set-specific ROI sequences were then created, that is, each ROI spanned only one of the three sets instead of all 127 frames. This strategy was chosen to include a good representation of cellular appearance from all stages of development in the labeled data while requiring minimal amounts of overall labeling. There were 3, 8 and 6 ROIs from the three sets respectively and these 17 ROIs had a total of 656 frames and 4768 cells.

These were then labeled to mark the locations and pixel-wise masks of cells through a custom-designed graphical labeling tool. This tool integrates the SiamMask tracker [278] to semi-automate the labeling process by propagating manually-created masks into future frames by joint unsupervised segmentation and tracking. Tracking was stopped manually when it started to fail and then restarted from the last frame where it worked, after making any required fixes in the intermediate frames. The labeling tool also supports using a previously trained segmentation model, if available, to automatically generate initial cell candidates that can then be manually modified instead of having to be drawn from scratch, although this capability was not used at this stage. Note that this relatively labor-intensive stage did not require involvement from iPSC detection experts since the labeled cells were not categorized into good and bad. Table C.1 provides numerical details of the dataset generated in this stage.

5.2.1.3.2 Exhaustive Automated Labeling

A Swin transformer instance segmentation model [26] was first trained on the annotations from the previous stage. Next, ROI sequences spanning all 127 frames were created. These were designed to cover as much of the circular well area containing cells as possible, while minimizing overlap between different ROIs. A total of 31 sequences were created by extending 11 of the 17 ROIs from the previous stage to the remaining frames and creating 20 new ones. These sequences had 3937 frames with 22,598 cells in all. Table C.2 provides numerical details of these sequences. Finally, the trained Swin transformer instance segmentation model was used to automatically detect and segment cells in each of these frames.

5.2.1.3.3 Categorized Retrospective Labeling

An iPSC detection expert first manually categorized the cells in frame 275 from each of the 31 ROIs into good and bad. The two categories were respectively named iPSC and differentiating cell (**DfC**) to reflect their likely future growth outcomes. A semi-automated interactive MOT tool was then used to propagate these labels backwards in time by tracking each cell line from frame 275 to 146. This process accounted for cell division and fusion events[1] by giving each child cell the same label as the parent in case of division and requiring that all merging cells have the same label in case of fusion. A violation of the latter requirement would have meant that the labels provided by the human expert were incorrect but this never happened, which is a sign of their reliability. Note that cell lines that disappear before reaching frame 275 cannot be categorized in this way so these have been excluded from all experiments.

The tracking algorithm was kept simple due to time and computational constraints. Cells were associated between neighbouring frames on the basis of location and shape, similar to the IOU tracker [37]. Possible fusion and division events were detected using heuristics based on association failures along with the extent of change in the size, shape, and location of associated cells. These events then needed to be manually confirmed or rejected using a convenient mouse-based interface. The detailed algorithm is provided in Algorithm 2. Due to this simplicity, the retrospective labelling process is currently more time- and labor-intensive than ideal and it took between 10 and 30 minutes to label each ROI sequence, depending on density of cells and frequency of division and fusion events.

[1]Note that a division event when going backwards in time is actually a fusion event and vice versa

We have made the code for all three stages publicly available [279] along with the annotated data and trained models to facilitate reproducibility of our results and further work in this domain.

5.2.2 Models

We selected two state-of-the-art image classification models to allow head-to-head comparison with the XGBoost classifier-based approach by [23] (**XGB**) which is the only existing method in literature that is relevant to this work. The models we chose are based on Swin transformer [26] (**SWC**) and ConvNext [24] (**CNC**) architectures. We also selected five instance segmentation models since both cell detection and classification are needed in the absence of fluorescence that was used by [23] to identify cells. Two of these are static detectors that process each frame independently and discard any video information. They are both variants of Cascade Mask RCNN [25] but differ in their backbone architectures, these being the same as the classifiers chosen above - Swin transformer [26] (**SWD**) and ConvNext [24] (**CND**). The remaining three are video detectors that combine information from multiple video frames to make their decisions. One of these - **IDOL** [28] - is an online model that only uses information from past frames. The other two models - SeqFormer [29] (**SEQ**) and **VITA** [30] - are batch models that use information from the entire video sequence including both past and future frames. All three video detectors use versions of the Swin transformer backbone. We also experimented with two ResNet [12] variants of VITA but they did not perform well and so have been excluded here. In addition, we trained a Swin transformer semantic segmentation model and tried several ensemble techniques to combine semantic and instance segmentation results but these did not yield significant performance improvements and are therefore likewise excluded.

5.2.3 Training

Since retrospective labeling is the most time-consuming part of our pipeline, it is desirable to minimize the need to label backwards to as few frames as possible. Therefore, in addition to comparing modern deep learning-based methods with XGB, we also wanted to evaluate the extent to which model performance on early-stage images depends on the lateness of the frames on which it is trained. We constructed two different training datasets to achieve this - an early-stage set with 38 frames from 163 to 202 and a late-stage set with 73 frames from 203 to 275.

Models trained on both datasets were evaluated on the 16 frames from 146 to 162. We also had to adapt the method proposed by [23] to work with our images since that work computes several of the features using 3D image information which is not available in our case. It turned out that only 7 of the 11 features used there could be suitably approximated with 2D data so we trained XGB using only these 7 features (Section C.2). All models were trained to convergence using mostly default settings recommended for each model by the respective authors except for minor tinkering with batch sizes, data augmentation strategies and scaling factors to make the models fit in the limited GPU memory available. The classification datasets were constructed from image patches corresponding to the bounding box around each labeled cell, with an additional 5 pixel border added for context. Models were trained on several different GPU configurations - video detectors: 2 × Tesla A100 40 GB, static detectors: 2 × RTX 3090 24 GB, classifiers: 3 × RTX 3060 12 GB and 3 × GTX 1080 Ti 11 GB.

5.3 Evaluation

5.3.1 Classification Metrics

We used standard receiver-operating-characteristic (ROC) curves and the corresponding area-under-curve (AUC) metric to compare the models. Note that head-to-head comparison between detectors and classifiers is difficult since the former both detect and classify cells while the latter only do classification on all the cells in the GT. Two types of detector failures need to be accounted for in order to render such a comparison meaningful:

- False Positives (FP): Detections without matching GT cells
  - Misclassification (**FP-CLS**): a GT DfC is detected but misclassified as iPSC
  - Duplicates (**FP-DUP**): the same GT iPSC is detected multiple times and classified each time as iPSC
  - Non-Existent (**FP-NEX**): an iPSC is detected where no GT cell exists (neither iPSC nor DfC)
- False Negatives (FN): GT cells without matching detections
  - Misclassification (**FN-CLS**): a GT iPSC is detected but misclassified as DfC
  - Missing detection (**FN-DET**): a GT iPSC is not detected at all (neither as iPSC nor as DfC)

FP-NEXs were ignored when computing the classification metrics since manual examination[2] showed that virtually all of these corresponded to one of two cases, neither of which is important in our application:

- **FP-NEX-WHOLE**: unlabeled cells whose labels could not be inferred by retrospective labeling (Section 5.2.1.3.3)
- **FP-NEX-PART**: parts of labeled cells, mostly in scenarios involving ambiguously-shaped cells that could plausibly be interpreted as undergoing division or fusion events, but were labeled as whole cells.

FP-DUPs were also ignored since multiple detections of an iPSC have little impact in our application. Finally, FN-DETs had to be discarded since ROC curves can only be generated by varying the cut-off used in filtering detections based on their confidence values which are unavailable for undetected cells.

In addition to the AUC of the complete ROC curve, we also used partial AUCs [280] with FP thresholds of 0.1%, 1% and 10%. Even a small number of FPs can be extremely detrimental in our application due to the high cost of culturing non-viable cells so that a model that performs better at these FP rates is preferable to another that is better overall but underperforms here.

5.3.2 Detection Metrics

The exclusion of FN-DETs while computing the classification metrics can make these biased in favour of detectors with high rates of missing cells but high accuracy for the few cells that they do detect. We accounted for this by incorporating the following detection metrics to evaluate and compare only the detectors:

- Frequency of FN-DETs, FP-DUPs and FP-NEXs
- Standard detection metrics [281] of mean average precision (**AP**) and AUC of the recall-precision curve (**RP-AUC**)

5.3.3 Temporal Metrics

The ability to detect iPSCs as early as possible is crucial to our application. Therefore, we also evaluated the models in each of the 16 test frames one-by-one to judge how their performance varied over time. A model that performs better in earlier frames would be preferable to one that performs better overall but underperforms in earlier frames.

[2] visualizations for all detection failures are included in apendix C.5

5.3.3.1 Subsequential Inference

Video detectors detect cells spanning all the frames in their input video instead of frame-by-frame. This is incompatible with temporal evaluation since even detections in early frames are done using information from all 16 frames in the test set. To resolve this, we used incremental inference where the detections for each frame are generated by running the detector on a subsequence comprising only that frame and all of the preceding ones so that information from future frames is not used. For example, detections for frames 1, 2 and 3 are respectively generated by running the detector only on subsequences comprising frames (1), (1, 2) and (1, 2, 3).

We used another variant of subsequential inference to evaluate the impact of the number of frames on the performance of video detectors so we can judge whether patterns in the way that cell boundaries change over time provide useful information about their eventual outcome to these detectors. Here, we divided the 16-frame sequence into a set of non-overlapping subsequences, each with a fixed size, and ran inference on each subsequence independently. For example, with a subsequence size of 2, our 16-frame sequence is divided into 8 subsequences - (1, 2), (3, 4), ..., (15, 16). Detections for all frames in each subsequence are then generated by running on only the frames in that subsequence. We experimented with subsequence sizes of 1, 2, 4 and 8.

5.4 Results

5.4.1 Classification

ROC curves for both early and late-stage models are shown in the top row of Figure 5.1. It can be seen that the early models perform much better than the late ones, which is expected since their training images are much more similar to the test images. However, the extent of this difference does indicate that cell-appearance changes so much over the course of just 9.5 hours (temporal gap between early and late-stage training images) that long-term retrospective labelling right to the very early-stage images is likely to be essential to generate training data for models that can do early-stage iPSC detection reliably. The significant performance advantage of deep learning models over XGB is also apparent here. The main shortcoming of XGB seemed to be its inability to handle class imbalance in the training set. Since about 75% of all cells were DfCs, XGB apparently learnt to classifty nearly all cells as DfCs. We

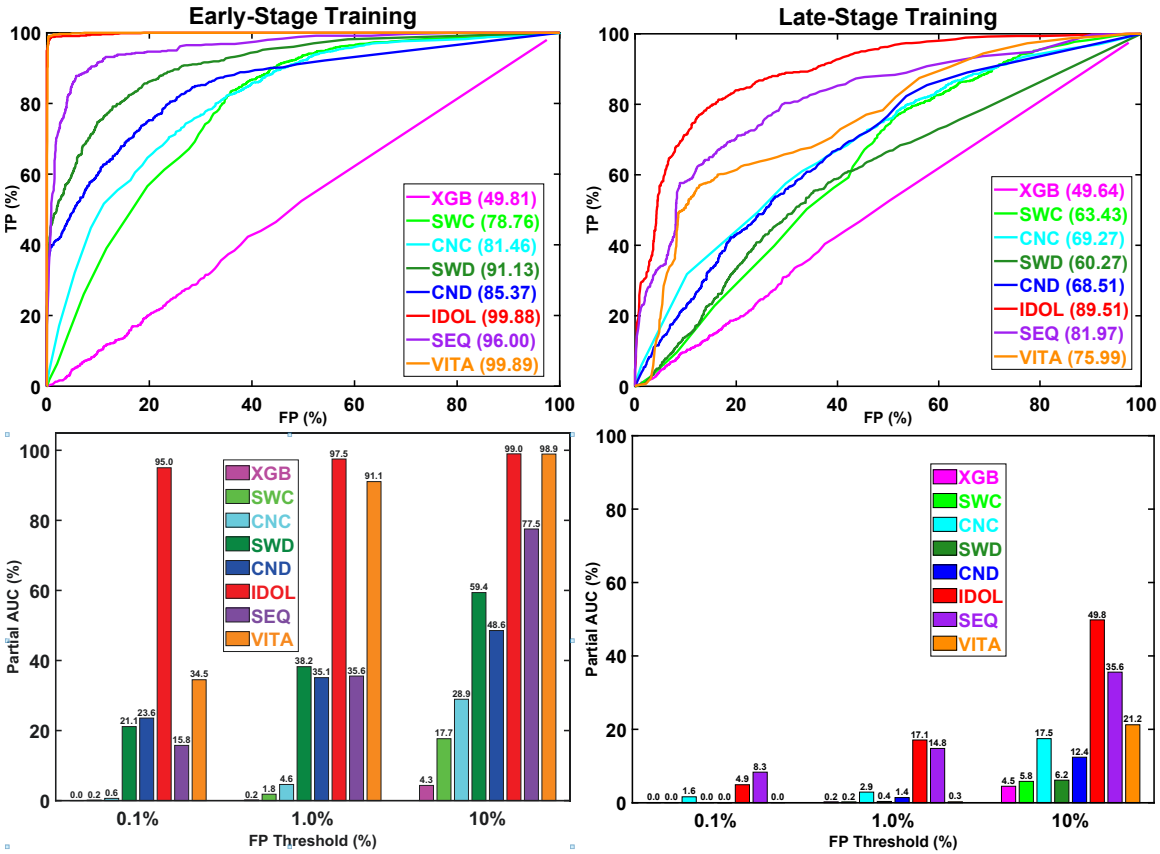

Figure 5.1: Classification metrics for (left) early and (right) late-stage models. Top: ROC curves (respective AUC values are in the legend); bottom: partial AUCs. Please refer Section 5.2.2 for model acronyms.

tried all the standard techniques to handle class imbalance [282] but this is the best performance we could get from it.

Further, all the video detectors are consistently better than the static ones, which seems to confirm the human experts' supposition that temporal information is crucial for making good predictions. Also, the static detectors do outperform the classifiers but only significantly so in the early-stage case. Since the precise shape of cell boundaries is not available to the classifiers, this lends some weight to the additional supposition that cell-shape is important for recognizing iPSCs. However, as already noted above, the cell-shapes change too rapidly for this information to be generalizable from the later stages to the earlier ones. Finally, IDOL turns out to be the best model overall even though it is the smallest and fastest of the three video detectors, while the much larger VITA shows a susceptibility to overfitting, as

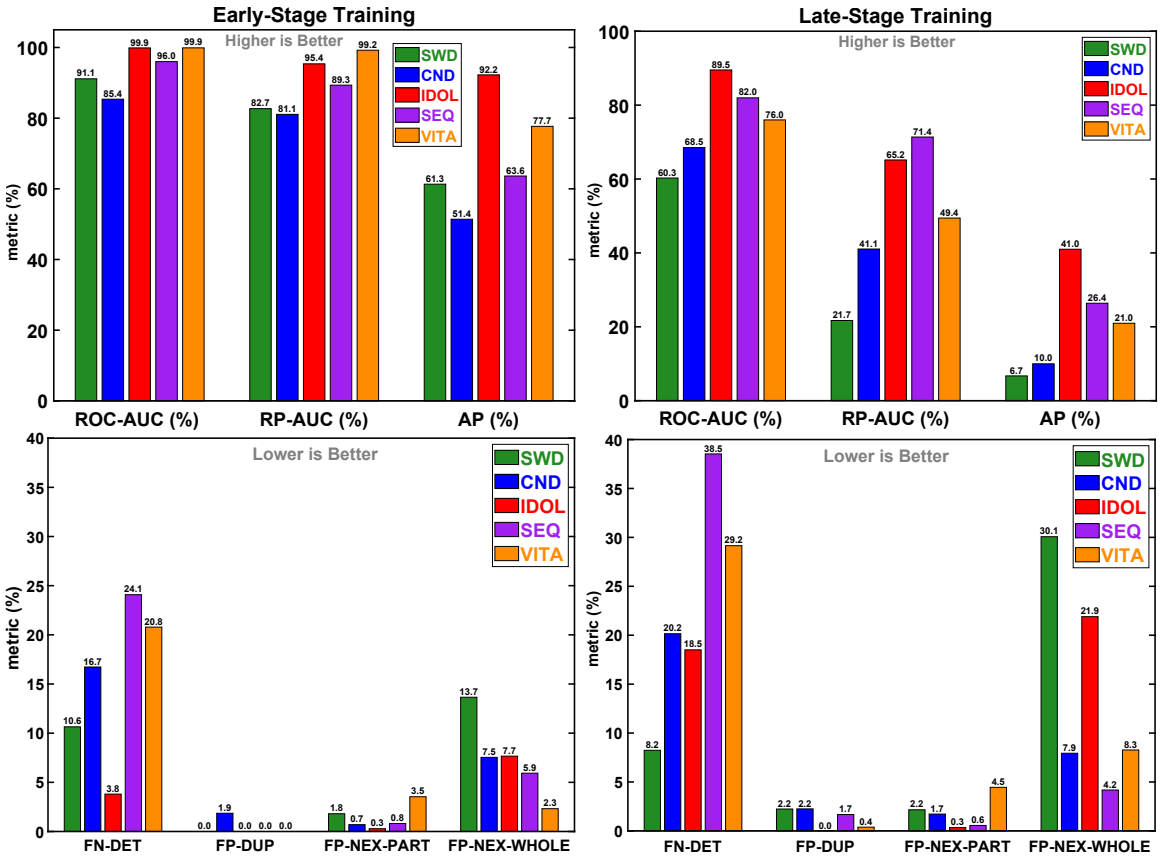

Figure 5.2: Detection metrics for (left) early and (right) late-stage models. ROC-AUC is included for comparison.

demonstrated by its sharp decline between the early and late-stage cases. This kind of overfitting is exhibited by both the static detectors too, though it is more strongly marked in case of SWD.

Partial AUCs are shown in the bottom row of Figure 5.1. Relative performance between the models is broadly similar to that for overall AUC, though IDOL shows greater performance advantage over other models in the early-stage case, especially for 0.1% FP. The detectors also show greater improvement over the classifiers in these high-precision scenarios. We also analysed the temporal evolution of partial AUCs by evaluating these frame-by-frame to generate 3D plots with time dimension on the Z-axis (Section C.6) but could not find any useful patterns beyond those apparent in these 2D plots.

5.4.2 Detection

As shown in Figure 5.2, relative detection performance is mostly consistent with classification accuracy, and IDOL is still the best model overall. IDOL also shows the smallest drop in performance between ROC-AUC and AP while the two static detectors show the largest drops amounting to nearly 10-fold for SWD and 7-fold for CND in the late-stage scenario. Among the video detectors, VITA suffers the largest performance drop, especially in the late-stage case which underlines its tendency to overfit. Further, both SEQ and VITA have very high FN-DET rates that are also at odds with their relatively good classification performance. Conversely, SWD has much lower rates of FN-DETs than its ROC-AUCs would suggest, especially in the late-stage case where it outperforms all other models by a significant margin, though this does come at the cost of a corresponding rise in FP rates. We can also note that this increase in FPs is dominated by only one subtype, namely FP-NEX-WHOLE, while FP-DUP and FP-NEX-PART show little change, not only for SWD but also for IDOL and VITA. It appears that either due to the greater range of cell-appearances in the late-stage dataset (owing to its greater size) or larger disparity in cell-appearance with respect to the test set, these models learnt to detect a lot more of the unlabeled cells than their early-stage counterparts. The latter were therefore better than the former at discriminating between the unlabeled and labeled cells.

5.4.3 Temporal

Frame-wise AUCs are shown in the top row of Figure 5.3. Somewhat contrary to expectation, AUC does not show consistent increase with time even though test images are becoming more similar to the training images. In fact, many of the early-stage models do show a weak upward trend while late-stage ones do not show any. This is particularly unexpected since the latter perform significantly worse overall and show signs of overfitting, which should lead to a more strongly marked increase in accuracy as resemblance between the test and training images increases. A possible explanation might be that the late-stage training images are just too far away from the test set for any intra-set variations in test images to make enough of a difference in their resemblance to training images to benefit the models.

Bottom row of Figure 5.3 shows the impact of subsequence length on the video detectors. These plots include only ROC-AUC but detection metrics showed similar patterns and have thus been relegated to appendix C.4. In order to incorporate the

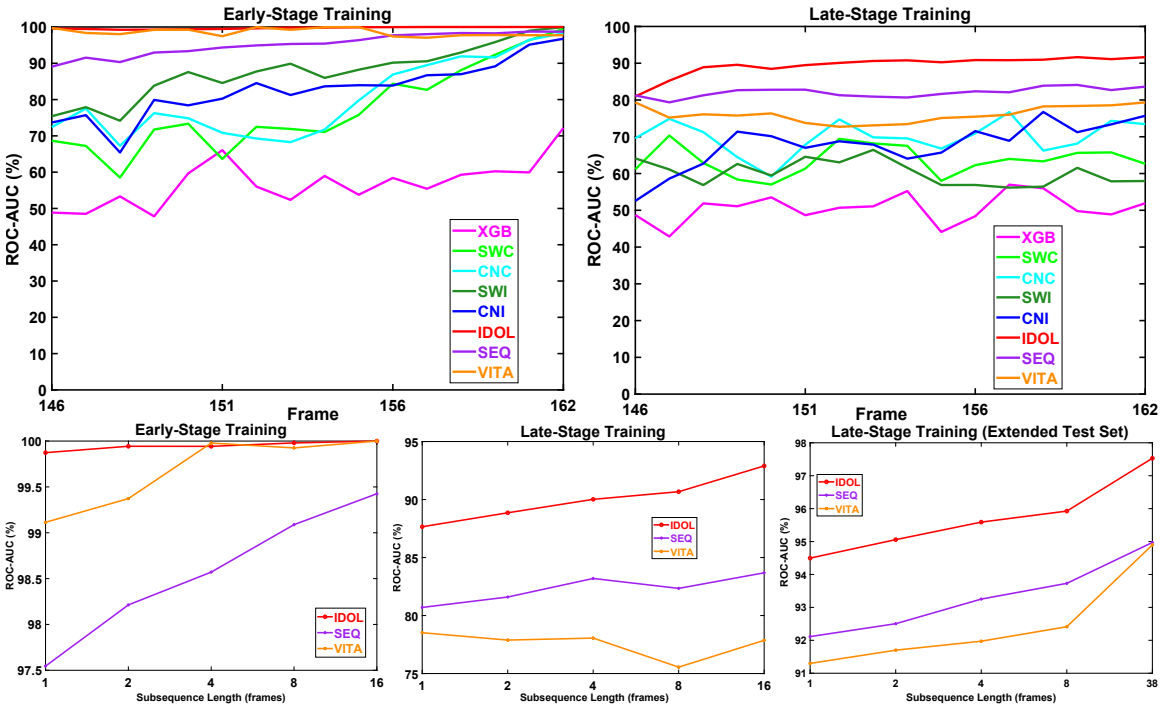

Figure 5.3: Temporal metrics - top: frame-wise ROC-AUCs for (left) early and (right) late-stage models; bottom: subsequential ROC-AUCs for video detectors - left and center plots show early and late-stage models tested on the standard test set of frames 146 - 162 while the right one shows the latter tested on an extended test set with frames 146 - 201. Note the curtailed and variable Y-axis ranges in the subsequential ROC-AUC plots.

longest-term temporal information possible with our dataset, we also tested the late-stage model on an extended test set comprising frames 146 to 201. Unfortunately, there is a visual discontinuity between frames 184 and 185 due to which this 54-frame sequence had to be divided into two subsequences - one with 38 frames from 146 to 184 and another with 16 frames from 185 to 201 - so that the longest subsequence length was 38 instead of 54. There is indeed a general upward trend with subsequence length but it is much weaker than might be expected. Early-stage models showed negligible impact of subsequence length while the greatest overall gains were 5.2% in the second case and 3.6% in the third case, achieved respectively by IDOL and VITA. Neither of these seem sufficient considering the 4 to 9 hours worth of extra temporal information available to the models. This might indicate that changes in cell-appearance over time are not as useful for recognizing iPSCs as supposed by experts. More likely, it might mean that existing video detectors are simply not good

enough to exploit this information sufficiently well and better long-term models are needed.

5.5 Conclusions and Future Work

This chapter presented a labeling, training and evaluation pipeline along with baseline performance results for early-stage prediction of iPSC reprogramming outcome to help select the best quality clones. These are still early days of research in this domain and it is difficult to say how practical such an automation can be. While it is clear that deep learning models hold significant advantage over previous methods, they also show signs of overfitting and it is unclear how much training data would be needed to overcome such issues.

We are currently working to improve the tracking algorithm in the retrospective labeling system to make the process faster and less tedious since that is the current bottleneck in the labelling pipeline. We are trying out several state-of-the-art real-time multi object trackers, especially those specialized for tracking cells [270, 271, 272], to replace the simple IOU based algorithm used in this work. We are also looking into ways to improve this algorithm by exploiting recursive parent-child relationships in our labels that can be used to construct hierarchical tree-like structures. Transformer architectures that support such structures [283, 284, 285, 286] might help to incorporate cell division and fusion events directly into the algorithm to not only improve the cell association reliability but also reduce the incidence of false positives in the detection of these events, which is the most time-consuming and tedious aspect of this process. A recent method uses graph neural networks to exploit these structures [269] for cell tracking. We are trying to incorporate it into our pipeline and also improve it further using transformers. If these efforts prove successful, we hope to make long term retrospective labelling spanning multiple weeks feasible so any such demands for data can be met.

We are also exploring ways to better exploit long-term patterns in the evolution of cell-appearance over time to improve prediction quality since the current ability of video detectors in this regard does not appear to commensurate with the importance that human experts attach to this information. This discrepancy might be due to both the limited temporal span of our data or the models being unable to benefit from this information because they do not learn from sufficiently wide temporal windows. We hope to resolve the former by extending our data from its current span of only 3 days

to as much as 5 weeks once the retrospective labeling system have been improved enough to make such long-term labeling feasible. The latter can be addressed by using more memory-efficient models since increasing the size of temporal windows for training is an extremely memory-expensive process for which our computational resources are insufficient.

Another related issue we are trying to address is that of the black-box nature of deep learning which makes it very difficult for the medical experts in our team to figure out the reasons behind particular failures of the models. It makes it equally difficult for us to incorporate their suggestions to fix such failures. We are hoping that recent advances in the field of interpretable deep learning [287, 288, 289, 290, 291], especially with regard to transformers [292], video processing [293] and medical imaging [294], might help with these limitations.

Chapter 6

Language Modeling for Video Detection

This chapter details my adaptation of Google's Pix2Seq framework [31, 49] for video object detection. I first briefly revisit the modeling of static object detection in Pix2Seq (Section 6.1.1) and then explain how I adapted it for video object detection (Section 6.1.2). This is followed by a brief description of Pix2Seq network architecture (Section 6.2.1) and how I adaped it for video processing (Section 6.2.2).

Note that many of the tokenization concepts in this chapter and the next one can be better illustrated with animations rather than still images. Since it is not possible to include animations here, I have created a website for this project [295] which contains the videos corresponding to many of the images included here. The links to the actual videos are included in the caption to each such image.

6.1 Tokenization

6.1.1 Static Object Detection

Pix2Seq performs object detection in static images by representing each object by five tokens - four for the bounding box corner coordinates and one for the class:

ty, lx, by, rx, cls

where (lx, ty) and (rx, by) are the x-y pixel coordinates of the top left and bottom right corners of the bounding box respectively, while cls is the class token. These coordinates are in image-space which is discretized into bins. The number of such bins can be either greater than the image resolution to achieve sub-pixel accuracy or

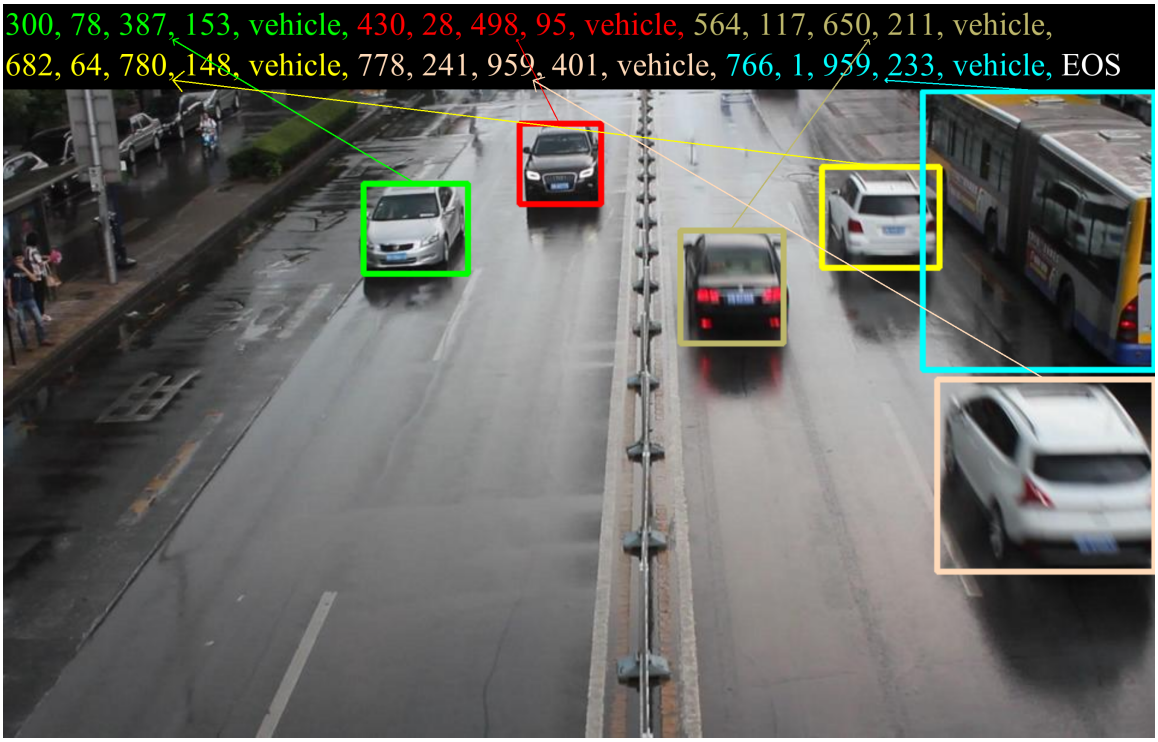

Figure 6.1: Visualization of object tokenization in Pix2Seq on an image from UA-DETRAC dataset. This dataset has a single class, represented here by the *vehicle* token. An animated version of this image is available here.

less than it to reduce vocabulary size. Both x and y coordinates are represented by the same shared set of tokens so that the total number of tokens in the vocabulary V is given by:

$$V = H + C + r \tag{6.1}$$

where H is the number of coordinate bins, C is the number of classes and r is number of reserved tokens (e.g. EOS and padding token). Pix2Seq uses $H = 2000$ and $V = 3000$ by default.

The tokens for all the objects in the image are produced sequentially so that the total number of output tokens $= 5 \times n + 1$ where n is the number of objects in the image and the extra token at the end is the EOS token that marks the end of all objects. The order of objects is not defined and is randomized each time the same image is shown to the network during training so the network is able to learn an order-agnostic representation of the objects. Figures 6.1 and 6.2 show examples of static object detection tokenization for single- and multi-class cases respectively.

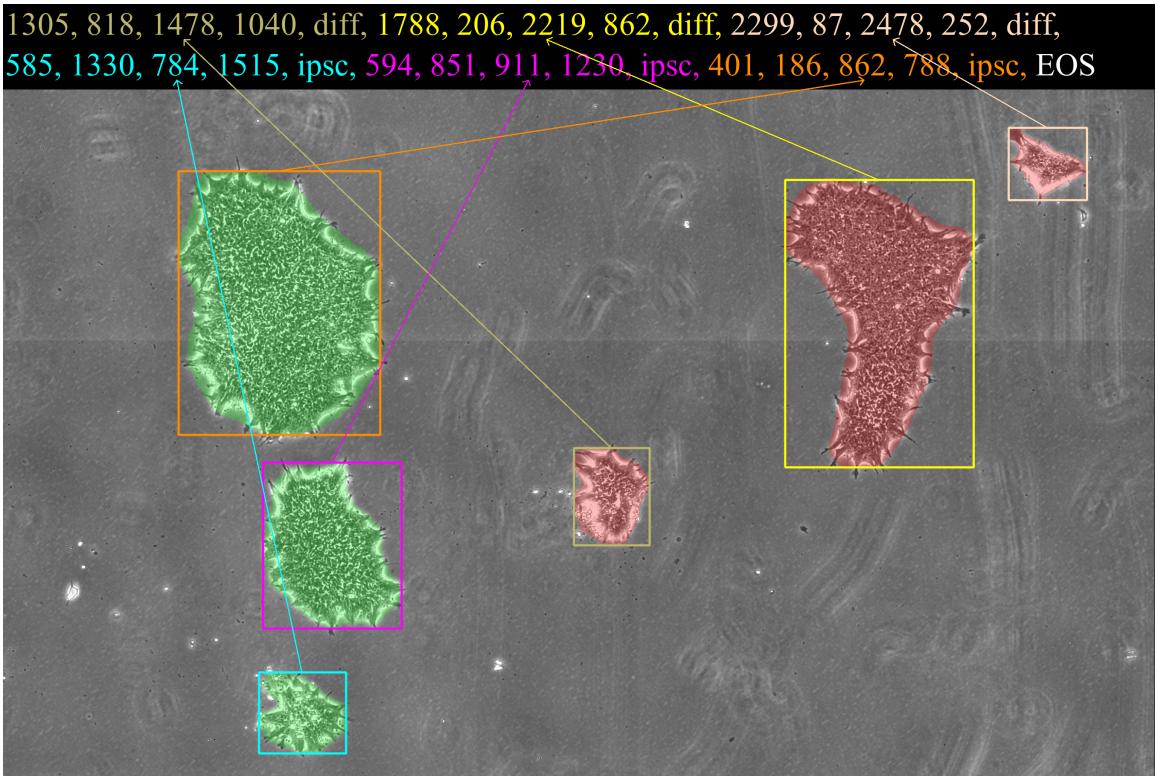

Figure 6.2: Visualization of object tokenization in Pix2Seq on an image from IPSC dataset. This dataset has two classes of cells - IPSC and differentiating - shown here with green and red masks and represented with *ipsc* and *diff* tokens respectively. An animated version of this image is available here.

6.1.2 Video Object Detection

I adapted this tokenization strategy for video object detection by considering all the bounding boxes of an object in an N-frame temporal window as a single 3D polygon or tubelet [216] that can be represented by a sequence of N 4-tuples, one for each bounding box, followed by the class token. Therefore, each object can be represented by $4 \times N + 1$ tokens and all n objects by $n \times (4 \times N + 1) + 1$ tokens. For example, we need:

- 9 tokens per object for $N = 2$:
  $ty_1, lx_1, by_1, rx_1, ty_2, lx_2, by_2, rx_2, cls$
- 13 tokens per object for $N = 3$:
  $ty_1, lx_1, by_1, rx_1, ty_2, lx_2, by_2, rx_2, ty_3, lx_3, by_3, rx_3, cls$

where (lx_i, ty_i) and (rx_i, by_i) are respectively the coordinates of the top left and bottom right corners of the bounding box in the i^{th} frame F_i.

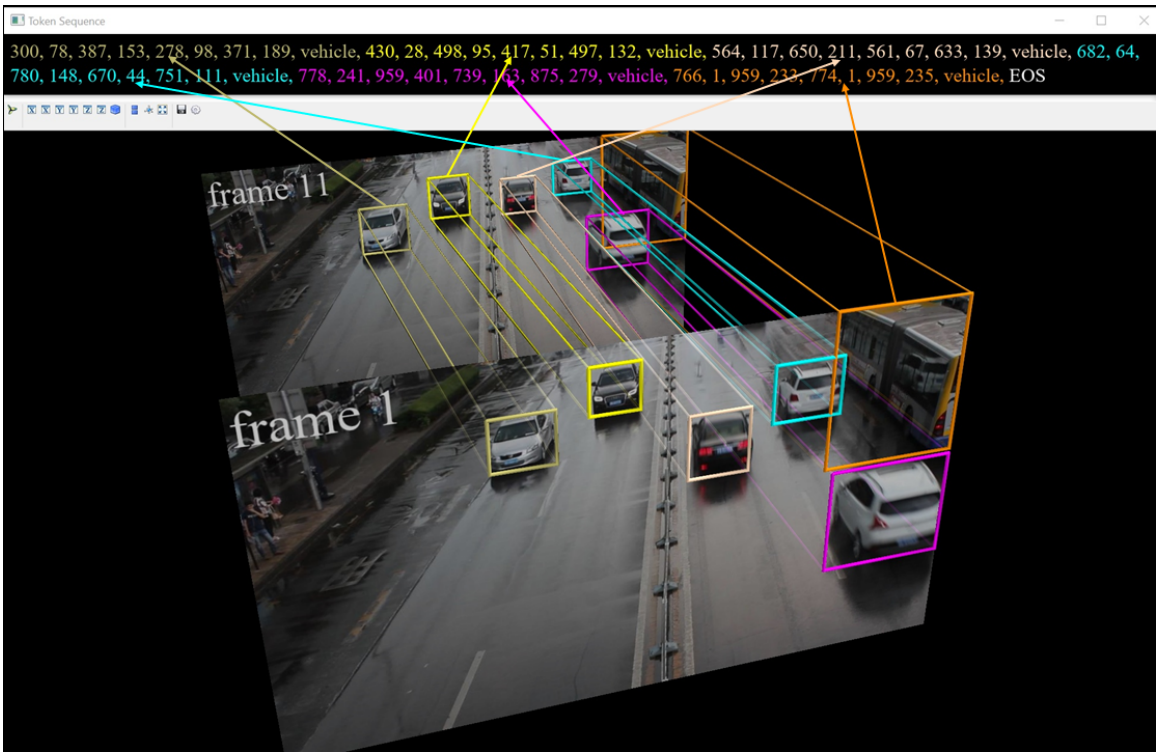

Figure 6.3: Visualization of video object tokenization on a video clip from UA-DETRAC dataset with $N = 2$ and $G = 10$. An animated version of this image is available here.

Figures 6.3 and 6.4 show a couple of examples of video detection tokenization.

6.1.2.1 *NA* Token

It is possible that an object is not present in each of the N frames in the temporal window. I account for this possibility by adding a special NA token to the vocabulary to denote non-existence Missing objects usually happen because an object either enters or leaves the scene in the middle of the temporal window, though sometimes it can also happen if the object is briefly occluded in the middle of the window. Following are examples of all three cases:

- $N = 3$, object leaves the scene in the third frame:
  $ty_1, lx_1, by_1, rx_1, ty_2, lx_2, by_2, rx_2, NA, NA, NA, NA, cls$
- $N = 4$, object enters the scene in the third frame:
  $NA, NA, NA, NA, NA, NA, NA, NA, ty_3, lx_3, by_3, rx_3, ty_4, lx_4, by_4, rx_4, cls$
- $N = 6$, object occluded in the fourth and fifth frames:

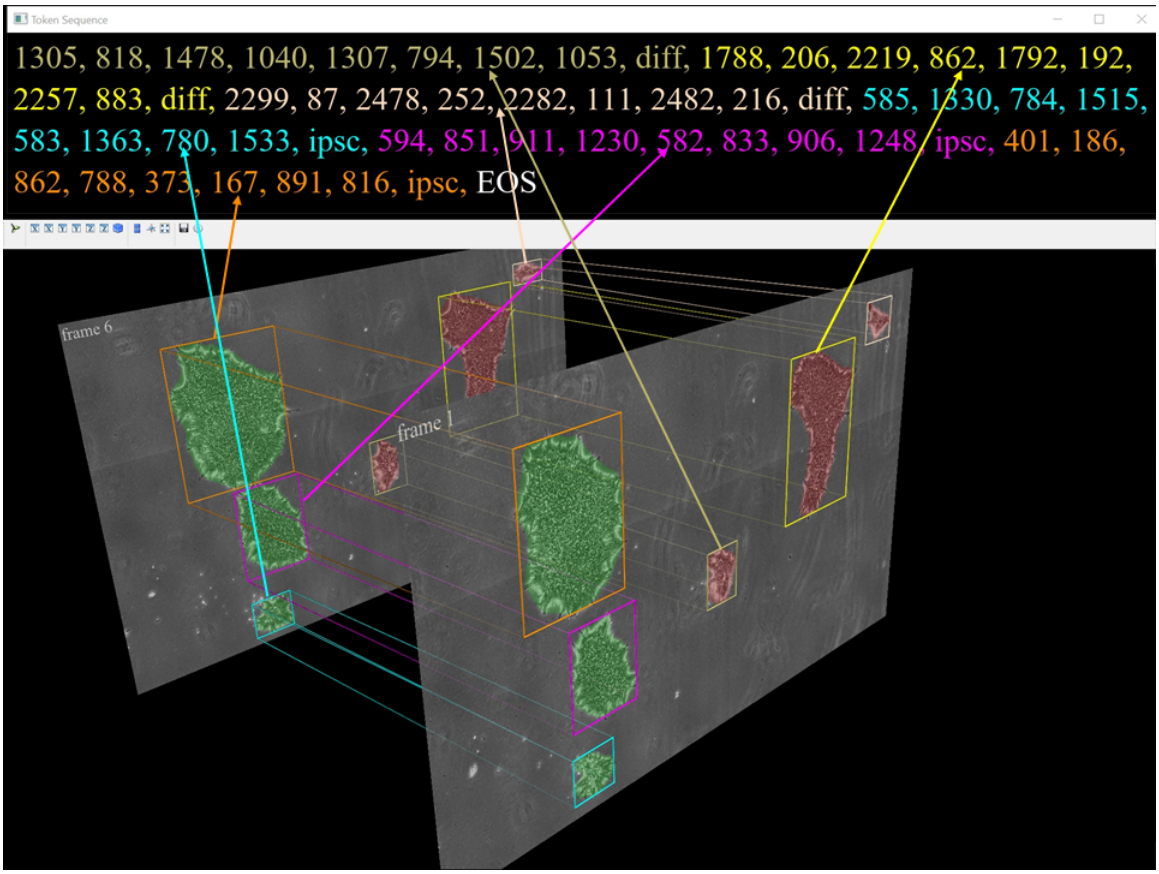

Figure 6.4: Visualization of video object tokenization on a video clip from IPSC dataset with $N = 2$ and $G = 5$. An animated version of this image is available here.

$$ty_1, lx_1, by_1, rx_1, ty_2, lx_2, by_2, rx_2, ty_3, lx_3, by_3, rx_3, NA, NA, NA, NA, NA, NA, NA, NA,$$
$$ty_6, tx_6, by_6, rx_6, cls$$

NA tokens become more common as N increases, especially in scenarios where the camera field of view is limited and objects are fast-moving. An animated example from UA-DETRAC dataset [296] with $N = 6$ is available here.

6.1.2.2 Limits on N

In theory, N could be large enough to cover the entire video, in which case video detection graduates into MOT. In practice, however, it is severely limited by the amount of available GPU RAM. Using the smallest ResNet-50 based Pix2Seq architecture (Section 6.2.1) and without freezing any layers, $N = 16$ is the maximum that can be trained on a RTX 3090 GPU with 24 GB RAM, which is the maximum available on a consumer-grade GPU. It should be possible to get close to

$N = 50$ on a Tesla A100 with 80 GB RAM, which is the maximum available on a workstation GPU. With the backbone frozen, it is possible to go as high as $N = 64$ on 24 GB RAM, though this limits the batch size to 1 per GPU or total 6 over the 3 dual-GPU servers available to me (Section 8.3). This is far too small to be able to learn something as complex as the the set of all objects in a 64-frame temporal window.

I have successfully trained models upto $N = 32$, although performance drops sharply beyond $N = 8$ (Section 8.5.4), indicating that such models cannot be successfully trained on my existing hardware. This is very likely due to the above problem of too small batch sizes, exacerbated by the frozen backbone and the relatively small number of videos in the datasets (Table 8.1). Another constraint on how large N can be made in practice is the the number of training iterations required for the model to reach convergence, which increases rapidly with N.Finally, the maximum sequence length L, that must be $\geq$ number of tokens required to represent all the objects in each temporal window, imposes another limit on N. With $N = 64$, we need $64 \times 4 + 1 = 257$ tokens to represent a single object and therefore several thousand tokens for tens of objects which are not unusual in many real-world scenarios. Pix2Seq has a default limit of $L = 512$ on the maximum number of tokens that can be produced by the network. I have managed to increase this to as high as $L = 4096$ but any increase in L causes both GPU memory consumption and training time to rise quickly, even if N itself is not changed.

6.1.2.3 1D Coordinate Tokens

The problem of L rising rapidly with increase in N can be partially ameliorated by using 1D coordinate tokens corresponding to a flattened image instead of the standard 2D (x, y) tokens. Each bounding box can then be represented by only 2 tokens instead of 4 and each object by $2 \times N + 1$ tokens, thus reducing L nearly by half. For example:

- $N = 3$, object leaves the scene in the third frame:
  $tl_1, br_1, tl_2, br_2, NA, NA, cls$

where tl_i and br_i are respectively the flattened coordinates of the top left and bottom right corners of the bounding box in the i^{th} frame F_i.

However, using 1D coordinates greatly increases the required vocabulary size V since now we need a total of H^2 different coordinates tokens instead of H and therefore $V = H^2 + C + r$. As explained in Section 7.2.1.1, it is important that $V \leq 32K$. This severely limits the number of coordinate bins H that we can use, which in turn

significantly reduces the localization accuracy of the detector. Also, while increasing V does not affect the GPU memory consumption and training time to the same extent as L, it does cause both to increase too, thereby limiting the batch size we can use and the number of epochs we can train for. I did perform some experiments with $H = 160$ and $H = 256$, using respective vocabulary sizes of $V = 18K$ and $V = 28K$ (Section 8.5.5), but found the models to be training far too slowly to be practicable. It appears that the 1D coordinate tokenization is too much of a change from the standard 2D version used in training the pretrained weights for successful fine-tuning to be viable with the limited dataset and computational resources available to me.

6.1.2.4 Temporal Windows

Since we cannot process the entire video at once, we divide it into N-frame temporal windows which are processed one at a time in a sliding window manner. There are two hyperparameters we can adjust to construct more varied temporal windows, both as a form of training data augmentation and to improve inference results.

The first parameter is the temporal stride T which is the number of frames that separate two consecutive windows. We can use a stride $T < N$ to introduce redundancy during inference which can help to deal with false negatives. For example, with $N = 3$ and $T = 1$, we get frames 1, 2, 3 in the first temporal window; 2, 3, 4 in the second one; 3, 4, 5 in the third one and so on:

(F_1, F_2, F_3), (F_2, F_3, F_4), (F_3, F_4, F_5), (F_4, F_5, F_6), ...

As a result, we have detection outputs for F_2 from two different temporal windows while the outputs for F_3 onwards come from three different windows. For any N in general, $T = 1$ provides N-way redundancy for all but the first $N - 1$ frames in the video. These redundant outputs can be combined using non-maximum suppression to fill-in objects that might have been missed in individual temporal windows but not all of them.

The second parameter is the frame gap G between successive frames in the same temporal window. We can make $G > 1$ as a form of training data augmentation to allow the network to learn to deal with faster inter-frame motions, which would be useful, for example, in handling dropped frames during live inference. Even for offline inference, $G > 1$ can provide us with even more overlapping temporal windows for each frame, thereby increasing the redundancy further. For example, with $N = 6$, $T = 3$ and $G = 2$, we get temporal windows:

$(F_1, F_3, F_5, F_7, F_9, F_{11})$, $(F_4, F_6, F_8, F_{10}, F_{12}, F_{14})$, $(F_7, F_9, F_{11}, F_{13}, F_{15}, F_{16})$, ...

All my experiments so far have been restricted to $G = 1$ but $G > 1$ provides an interesting avenue for future exploration.

6.2 Network Archtectures

6.2.1 Static Image Input

Pix2Seq supports two backbone architectures - ResNet-50 [12] and VIT [34] - each with three input sizes - 640×640, 1024×1024 and 1333×1333. I have used the smallest version - 640×640 ResNet-50 - for most of my experiments so as to maximize N and training batch size. I did perform a few experiments using 1024×1024 and 1333×1333 ResNet-50 as well as 640×640 VIT but did not observe any significant performance improvements. Note that I have only adapted the ResNet-50 version for video modeling since the video fusion architectures proposed in Section 6.2.2 are incompatible with VIT. I have left the adaptation of VIT for video processing as future work since it requires far too much GPU memory for any video version to be practicably trainable with my existing computational resources. The remainder of this section is therefore restricted to describing the ResNet-50 variant of Pix2Seq network architecture. The network itself has the standard transformer-based encoder-decoder architecture [33] where the encoder performs only self-attention with image features while the autoregressive decoder does self-attention with sequence features as well as cross-attention between sequence and image features.

6.2.1.1 Encoder

The encoder takes the 640×640 RGB image as input and applies multi-headed self-attention [33] to convert it into 400×256 features which are used as input for the decoder. This diagram in Figure 6.5 summarizes the encoder architecture. Here,

- B is the batch size
- *FLATTEN FEAT* refers to the spatial flatenning of the 20×20 ResNet-50 feature maps into 1D feature vectors of size 400
- *POS EMBED* is the operation of adding positional embedding to the flattened features
- *PROJ MLP* is a multi-layer perceptron (MLP) that projects the flattened features to 400×256

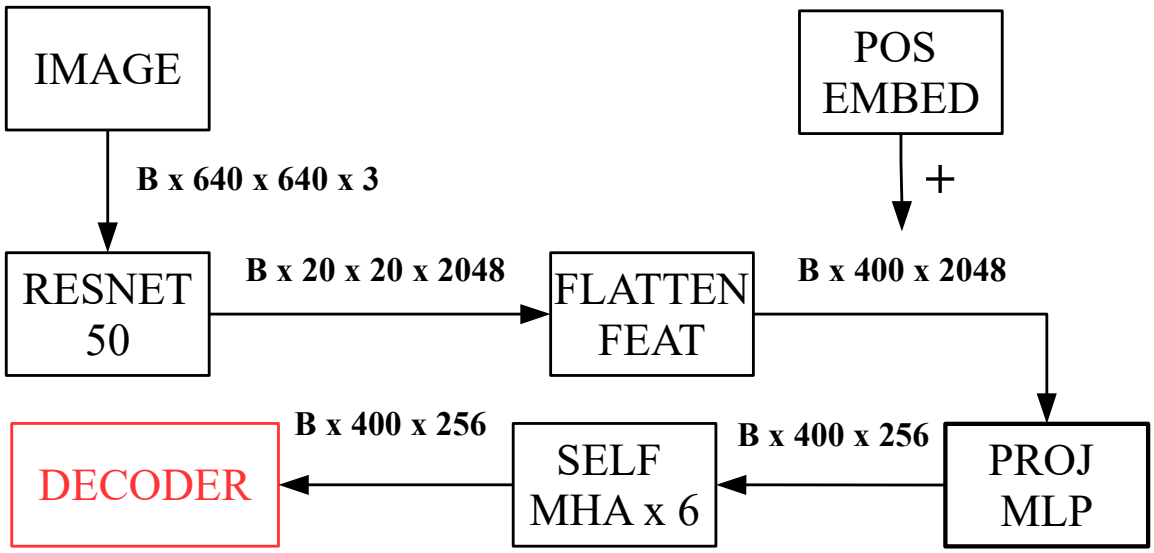

Figure 6.5: Flow diagram representing the high-level processing in the Pix2Seq encoder. Please refer Section 6.2.1.1 for details.

– *Self MHA X 6* module refers to the multi-headed attention (MHA) module that applies self-attention to the image features which is repeated 6 times.

6.2.1.2 Decoder

The decoder takes the 400×256 image features from the encoder along with the sequence tokens. It then applies multi-headed self-attention to the sequence embedding features, followed by cross-attention between the sequence and image features. This self + cross MHA operation is repeated 6 times just like the self-MHA operation in the encoder. Figure 6.6 summarizes the decoder architecture. Here,

– *SEQUENCE TOKENS* refers to the sequence of target tokens constructed from the GT objects and padded to the maximum sequence length (500 in this case)
– *INPUT EMBEDDING* is obtained by table-lookup into the 3000×256 weight matrix of a single linear layer where each row corresponds to one token in the vocabulary of size $V = 3000$
– *OUTPUT EMBEDDING* is obtained by projecting the 256 dimensional feature vectors to 3000 dimensional ones using the same linear layer whose weights are used in the input embedding module
– *SELF MHA* module applies self-attention to the sequence embedding features
– *CROSS MHA* module applies cross-attention between sequence features outputted

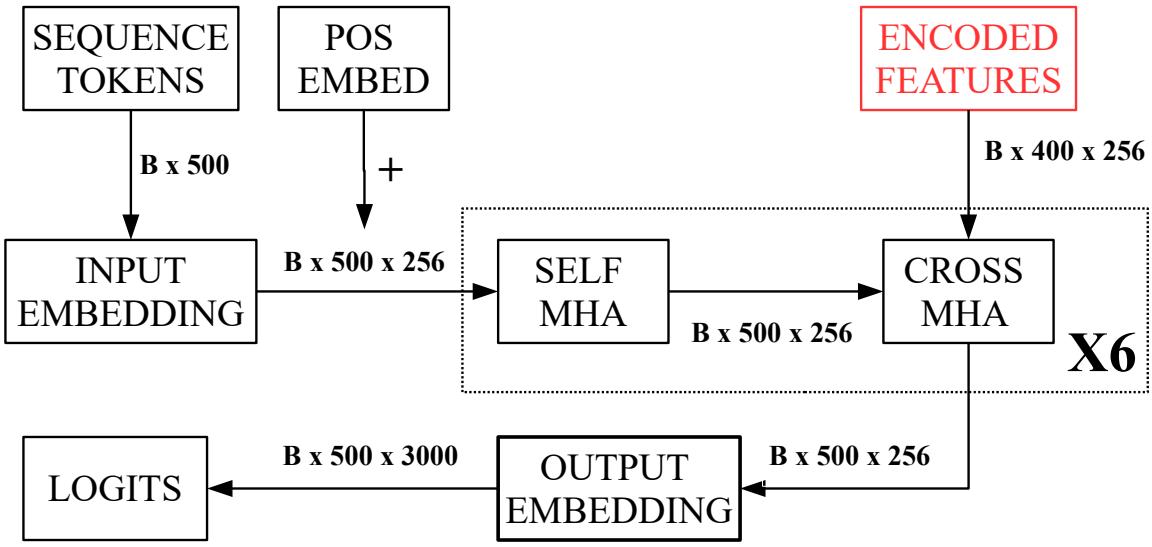

Figure 6.6: Flow diagram representing the high-level processing in the Pix2Seq decoder. Please refer Section 6.2.1.2 for details.

by the self MHA module and the image features from the encoder

6.2.2 Video Input

I have determined experimentally that, like most transformer-based language models, Pix2Seq is very difficult to train from scratch on the relatively small datasets that I am working with. Therefore, we want to be able to use as much of the pre-trained weights as possible. This in turn requires that the baseline architecture is modified as little as possible. With this in mind, I have come up with three ways to adapt the architecture for processing videos. These differ in the stage of the encoder-decoder pipeline at which the features from individual video frames are fused together.

6.2.2.1 Early Fusion

This method replaces the ResNet-50 backbone with a video-specific backbone such as the Video Swin Transformer [297] or 3D-ResNet [298] so that the feature fusion happens within the backbone itself. I have only experimented with Video Swin Transformer so far. Figure 6.7 summarizes the early-fusion architecture. This method only changes the number of backbone feature maps from 2048 to 768 while the rest of the pipeline remains unchanged. This means that we are unable to use pretrained weights only for the projection MLP that projects the flattened backbone

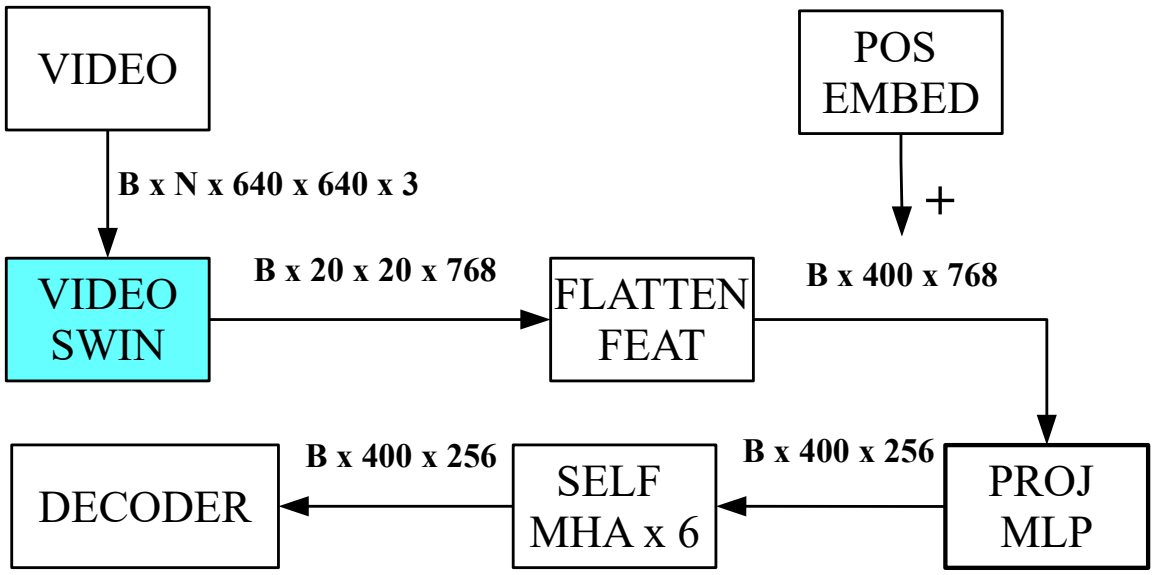

Figure 6.7: Flow diagram representing high-level processing in the early-fusion video encoder with the video Swin transformer backbone. Please refer Section 6.2.2.1 for details.

features from 400×768 to 400×256. Of course, we also lose the pretrained weights for the ResNet50 backbone itself but the video swin transformer implementation I am using [299] comes with its own weights which seem to work well enough as long as the backbone is not frozen while training. Unlike the Pix2Seq pretrained weights which were trained for token-based object detection, this backbone was trained on the Kinetics video action recognition dataset [300] with conventional (i.e. non token-based) modeling. As a result, the overall network fails to generalize to token-based tasks if the backbone is kept frozen (Section 8.5.3.1). I have experimented with both tiny and base variants of this backbone but found them to have similar performance inspite of the latter having more than three times the number of parameters (87.64M versus 27.85M), probably because of the much smaller batch size necessitated by this larger network size.

6.2.2.2 Middle Fusion

As shown in Figure 6.8, we first flatten the temporal dimension along the batch dimension to replace B with $B \times N$ images (and feature maps) while leaving the rest of the encoder pipeline unchanged. The result of this reshaping is that all the subsequent operations up to and including self-MHA are performed to each one of

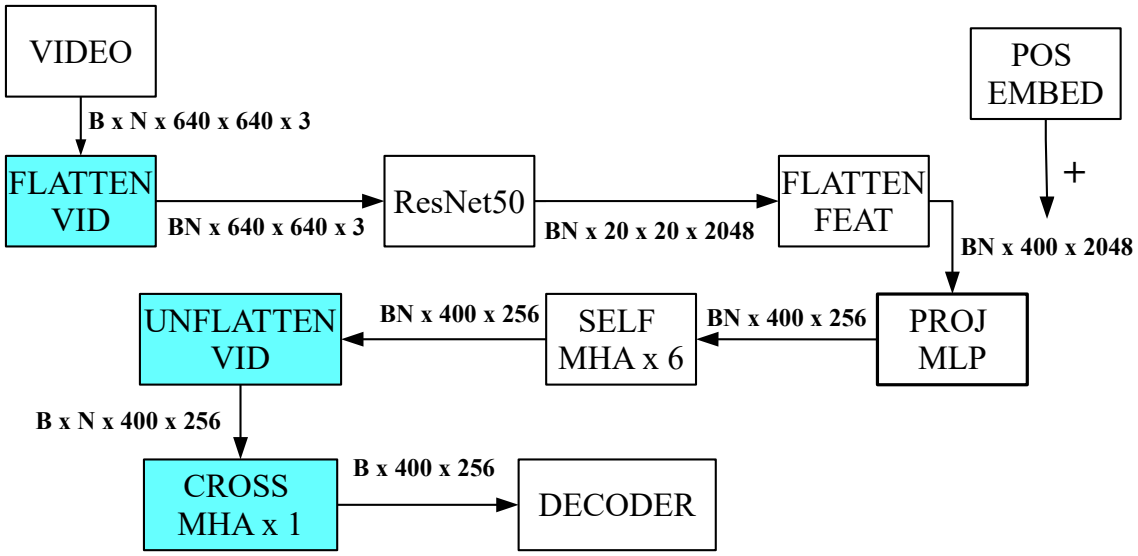

Figure 6.8: Flow diagram representing high-level processing in the middle-fusion video encoder. Please refer Section 6.2.2.2 for details.

the video frames independently as if we had a batch size of $B \times N$ instead of B. Once we have the $BN \times 400 \times 256$ output from the self-MHA module, we unflatten the temporal dimension to separate out the 400×256 features for each one of the N video frames.

We then apply some form of cross-attention between the features from different video frames to fuse them together. The specific technique I have used is pairwise compositional cross-MHA as shown in Figure 6.9. Here, we first apply cross-MHA between the features of F_1 and F_2, then between the output of this operation and features of F_3, then between the output of this operation and features of F_4 and so on. For the sake of simplicity, all the cross-MHA operations share weights in my implementation although it would also be possible to have separate weights for each one. Another way to perform video cross-MHA is hierarchical consecutive cross-MHA as explained in Section D.1. There are probably many other ways to perform this operation but these are the only two that I have considered and the pairwise compositional variant is the one I have used for all of my experiments since the hierarchical version is less stable to train due to the excessive number of cross-MHA operations it requires for larger values of N ($N \times (N-1)/2$ in hierarchical versus $N-1$ in compositional).

Note that, unline the self-MHA operation, the video cross-MHA operation cannot be repeated multiple times since the input and output shapes of each such operation

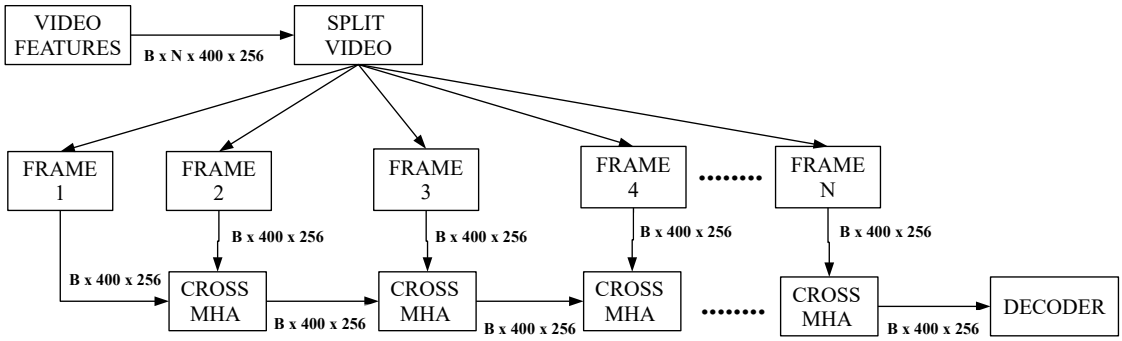

Figure 6.9: Flow diagram for the pairwise compositional variant of the cross-MHA module in the middle-fusion video encoder. Please refer Section 6.2.2.2 for details.

are different. Middle-fusion allows us to use all the pretrained weights but we have to learn the compositional cross-MHA weights from scratch. This can sometimes lead to a bit of instability during training, especially for larger values of N. Nevertheless, middle fusion outperforms the other two methods in most cases (Section 8.5.3), albeit by small margins, so this is the one I have used for most of my experiments.

6.2.2.3 Late Fusion

This is the only method where feature fusion happens in the decoder, unlike the other two methods where it happens in the encoder so that the decoder remains exactly the same as in the static architecture. Figure 6.10 shows the late-fusion encoder and decoder. The encoder here is identical to the middle-fusion encoder as far as generating the $BN \times 400 \times 2048$ backbone features, after which the temporal dimension is unflattened to separate frame-specific features. 3D position embedding is then added to these to encode information about the position of each frame within the video. The 3D position embedding parameters are the only ones for which we cannot use pre-trained weights and therefore have to learn from scratch. The 400 features for each of the N frames in each video are then concatenated together into $400 \times N$ features.This creates an overall feature map of size $B \times 400N \times 2048$ which is processed normally for the remainder of the encoder pipeline, except that now we have $400 \times N$ features instead of 400. Each of these $400 \times N$ image features is then cross-attended with each of the 500 sequence features in the decoder so that every single frame is directly able to attend to every single output token.

In theory, we would expect that the flexibility of every frame being able to directly affect every token would make it possible to train better models since visual

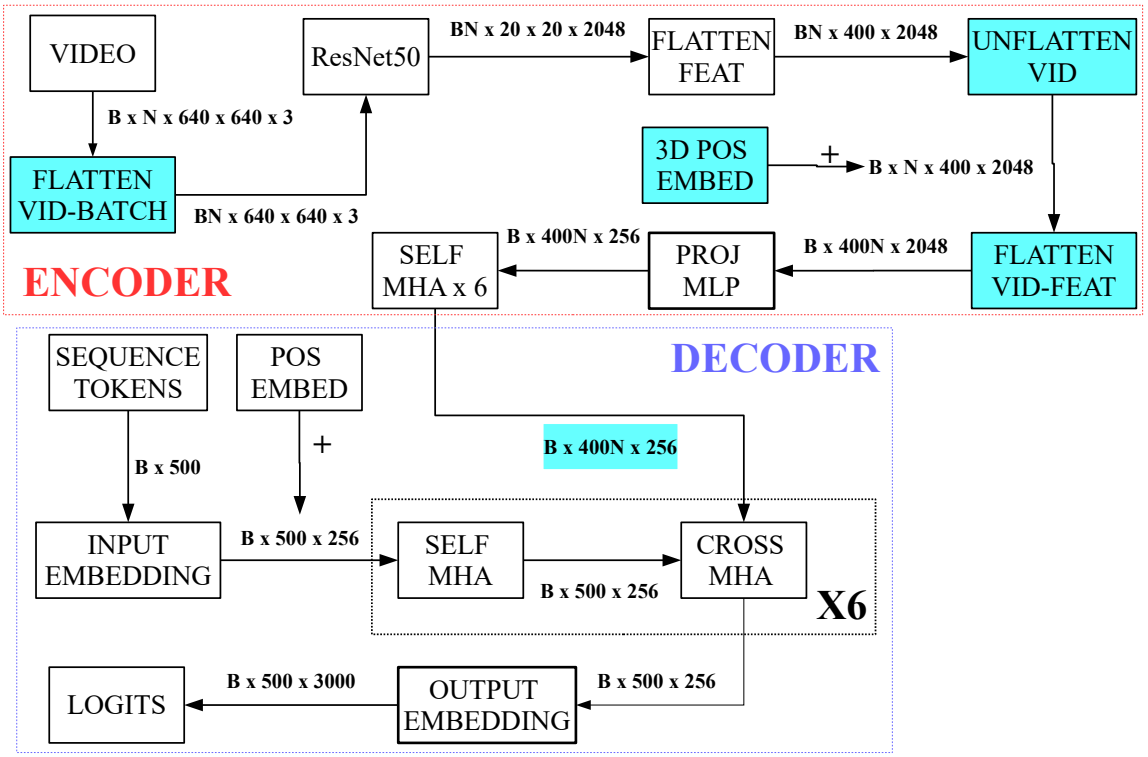

Figure 6.10: Flow diagram representing high-level processing in the late-fusion video encoder and decoder. Please refer Section 6.2.2.3 for details.

information from any frame can be used to improve the prediction of tokens corresponding to both past and future frames. This should be particularly useful for handling occlusions since the frame where an object is actually occluded contains little to no visual cues about this occluded state but this information can be obtained from past and future frames where the object is not occluded. Late-fusion is also the method with the fewest parameters for which we are unable to use the Pix2Seq pretrained weights (409K in late fusion versus 800K in middle-fusion and 24M in early-fusion) However, in practice, I have not observed any consistent performance improvement with late-fusion compared to the other two methods.

Chapter 7

Language Modeling for Semantic Segmentation

This chapter details my adaptation of the Pix2Seq framework [31] for semantic segmentation in both images and videos. Section 7.1 briefly explains the run length encoding (RLE) representation I used to represent segmentation masks with sequences of discrete tokens. Section 7.2 follows with details of how I tokenized RLE for the case of static images including the sliding window patches (Section 7.2.2) and Lengths-As-Class (LAC) encoding (Section 7.2.3) I employed to deal with high resolution and multi-class masks. Finally, section 7.3 concludes with an extension of this representation for videos, including the Time-As-Class (TAC) and Lengths-and-Time-As-Class (LTAC) schemes (Section 7.3.1) I used to compress the RLE further to make it feasible to incorporate multiple masks without exceeding the token sequence length beyond practicable limits. I have used the same network architectures for semantic segmentation as object detection (Section 6.2) so I do not cover these again.

7.1 Run Length Encoding (RLE)

I have chosen to tokenize semantic segmentation masks using the run-length encoding (RLE) representation [301]. This is a lossless data compression technique that flattens the segmentation mask into a 1D vector and represents this as a sequence of *runs*. A *run* is a continuous sequence of non-zero pixel values that can be represented by a pair of integers - *start*, *length* - where *start* is the index of the first pixel in the

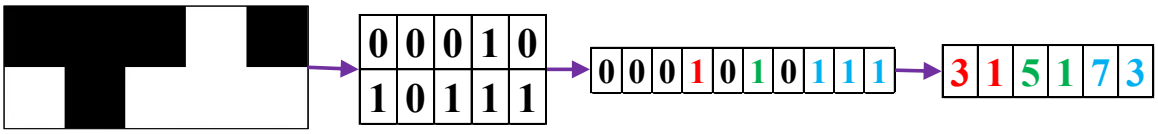

Figure 7.1: Generating RLE sequence for a 2×5 binary mask using row-major flattening. Note that the start indices use 0-based indexing instead of the 1-based indexing commonly used in RLE.

sequence and *length* is the number of pixels in the sequence. An example[1] is shown in Figure 7.1. A binary segmentation mask can thus be represented by a sequence of these pairs:

$start_1, length_1, start_2, length_2, start_3, length_3, \ldots$

while a multi-class mask would need a sequence of triplets since each run would also have the class ID:

$start_1, length_1, class_1, start_2, length_2, class_2, start_3, length_3, class_3, \ldots$

The mask can be flattened in either row-major and column-major order. Both of these lead to similar sequence lengths on average, although one might be more suitable than the other for specific cases. For example, row-major would provide shorter sequences if most of the objects are short and wide while column-major would work better for tall and narrow objects.

I also considered two alternative mask representations including polygons [80, 81, 302, 303, 49] and quadtrees [304]. However, statistical tests showed that all of these representations require roughly the same number of tokens as RLE so I chose the latter since it provides two advantages over the others. Firstly, it is much easier to implement, especially when generalizing to multi-class and video segmentation. Secondly, it is likely to be more robust to noisy tokens during mask reconstruction at inference since a single *run*, which forms the geometrical unit of RLE tokenization, has a much smaller impact on the overall mask quality than a polygon or quadtree. For example, if a few RLE tokens go missing at inference, it is likely to have minimal impact on the overall mask quality since each run usually affects only a few nearby pixels in the same row (or column). Also, the gaps or artifacts in the mask thus created might be remedied relatively easily by image processing techniques like morphological operations [305]. However, a single missing polygon token could severely degrade the mask in a way that cannot be easily fixed by image processing.

On the other hand, RLE does have an important disadvantage in that it makes

[1]This example has been borrowed from this online article

object-level information difficult to learn since it represents each object by a large number of independent tokens. It would be an interesting area of future work to figure out how to add object-level information to the mask tokens without losing the robustness benefit provided by the small quantum of RLE.

7.2 Static Image Segmentation

7.2.1 Tokenization

As mention in section 6.2.2, Pix2Seq is very difficult to train from scratch, so it is important that we are able to use as much of the pretrained weights as possible. This in turn requires that the baseline architecture be modified as little as possible. This imposes a couple of architectural constraints on RLE tokenization – the size of the vocabulary V and the maximum sequence length L.

7.2.1.1 Architectural Constraints

Out of the box, Pix2Seq has $L = 512$ and $V = 3K$ in the original object-detection-only variant [31], though the multi-task version [49] implemented in the same codebase supports V upto $32K$. The choice of how to tokenize RLE involves a trade-off between these two constraints, wherein a mask can be encoded by fewer tokens (thereby decreasing L) by adding more unique tokens to the vocabulary (thereby increasing V) and vice versa. Also, increasing either L or V causes both training time and GPU memory consumption to rise too, though this increase is significantly more pronounced for L than V. Hence, my overall objective was to keep the RLE sequence as short as possible while allowing the number of required tokens to increase up to $32K$. I have been able to get the model working with RLE sequence lengths up to $3K$ and vocabulary sizes up to $28K$. It is possible to get training started with sequence lengths up to $8K$ but anything much above $3K$ results in the training crashing soon afterwards, irrespective of V, so it appears that 24 GB GPU RAM is simply not enough for $L >> 3K$.

7.2.1.2 Mask Flattening

Let us assume that we have an $S \times S$ binary segmentation mask. If we employ the conventional practice of flattening the mask into a 1D vector, we can use a single token to represent the *starts* but we would need S^2 different tokens in the vocabulary

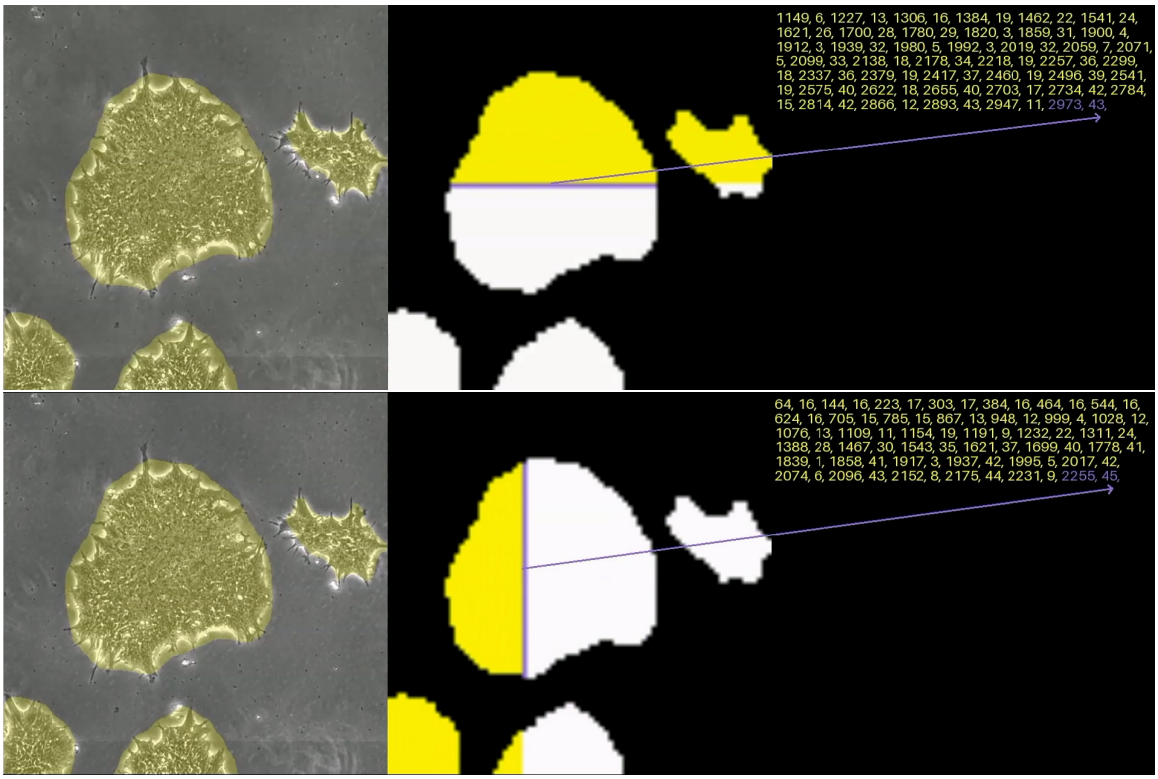

Figure 7.2: Visualization of RLE tokenization of binary segmentation masks with row-major (top) and column-major (bottom) flattening of the masks. Each figure shows (from left to right) the source image patch with the foreground mask drawn on it in yellow, binary version of this mask with the already tokenized segment in yellow, and the corresponding tokens. The run that is currently being tokenized is shown in purple. Animated versions of these figures are available here and here.

for the start indices. Alternately, we can skip the flattening and just use the 2D coordinates directly to represent the start of each run. In this case, we need only S start tokens but now we need 3 tokens to represent each run, which increases L by 50%. As mentioned above, reducing L is more important than reducing V so I have mostly been working with the flattened version, though I did train a couple models with 2D start tokens as well. Fig 7.2 shows examples of RLE tokenization for binary segmentation mask with both row-major (top) and column-major (bottom) mask flattening.

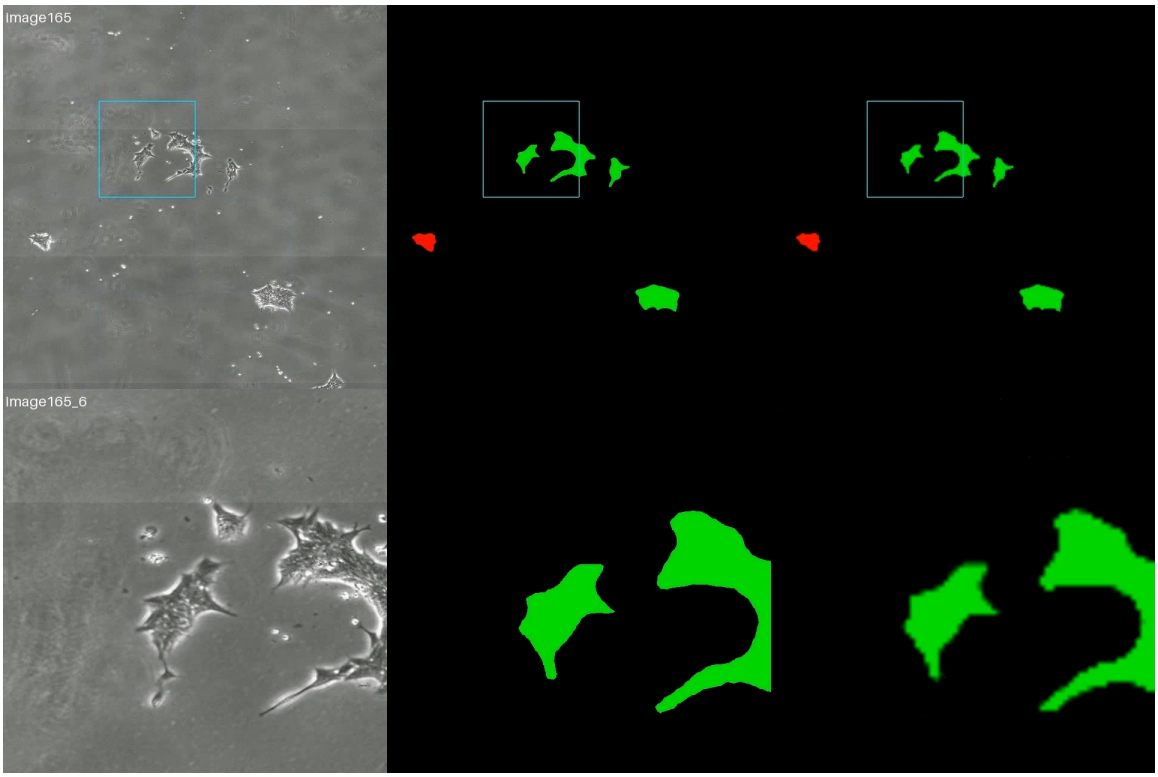

Figure 7.3: An example of the sliding window patch extraction and mask subsampling process on an image from the IPSC dataset. The top row shows (from left right) source image resized to 2560×2560 with patch location shown by the blue box, corresponding mask at its full resolution of 2560×2560, and this mask subsampled by a factor of 8 to 320×320. The bottom row shows (from left right) 640×640 patch corresponding to the blue box, corresponding patch mask at its full resolution 640×640, and this mask subsampled by a factor of 8 to 80×80. This subsampled 80×80 mask is the one that is used for generating the RLE sequence. An animated version of this figure is available here.

7.2.1.3 Shared Tokens

Further, we have the option to share the same set of tokens for both *starts* and *lengths* since the *lengths* can theoretically be as large as the *starts* when a single run covers the entire mask. However, in practice, *very* few runs extend across multiple rows (or columns) so I decided to use separate tokens for *lengths* and imposed a limit of S on the maximum length that a run can have. Runs that exceed S can be split into multiple runs. For example with $S = 80$, an overlong run $(300, 125)$ can be split into two runs $(300, 80)$ and $(380, 45)$. In theory, this can increase the total number of runs (and therefore L) dramatically but, as mentioned above, it is extremely rare for runs

to extend across multiple rows (or columns) so it does not matter much in practice.

Having separate tokens for *starts* and *lengths* also provides a significant advantage during inference when we need to resolve the $L \times V$ probability distribution into actual tokens. If *starts* and *lengths* are using the same set of S^2 tokens (assuming the flattened mask case), we would need to take the *argmax* over all the S^2 probability values to generate each length token. Using a separate set of S tokens for *lengths* allows us to take the *argmax* over only these S probabilities which helps to correct the run in cases where the network confounds the positioning of *starts* and *lengths* tokens.

7.2.2 Sliding Windows

I collected statistics on the lengths of the RLE sequences required to represent segmentation masks for complete images over the entire IPSC and ARIS datasets. This turned out to be well over 512 for many images even when they were resized down to $I = 320$, which itself reduced the resolution of many IPSC sequences by a factor of more than 10 (Table C.3). In addition, even $S = 320$ is not feasible with 1D start tokens because of the impracticably large vocabulary size $V = S^2 = 320^2 = 102.4K$ that this requires. Although such resolutions can be managed using 2D start tokens, this increases L by 50% which in turn significantly reduces the training batch sizes that can be used.

I resolved these issues by first extracting smaller patches of size $P < I$ from those images in a sliding window manner and then training on these patches instead of the complete images. This is similar to how I handled over-large images in the river ice segmentation project (Section 3.3.3). An example is shown in Figure 7.3. I also applied all the patch dataset augmentation techniques from Section 3.3.3. These include employing random strides smaller than P to generate overlapping patches, and applying random geometric transforms like rotation and horizontal and vertical flipping to the patches thus generated.

7.2.2.1 Redundancy

Similar to overlapping temporal windows (Section 6.1.2.4), overlapping patches can be used to increase redundancy during inference. In fact, when doing video segmentation (Section 7.3), we can get redundancy from both temporal and spatial windows. However, unlike object detection, there is no straightforward method to

combine information from multiple masks of the same region in the image to create a composite mask that is better than any of its constituents. The best method I could think of was to employ a pixel-level voting strategy where we collect the class label for every pixel from all the patches that contain that pixel and then use the most frequently occurring class as the final label for that pixel. However, even this resulted in a slight overall degradation of mask quality as compared to simply using the class labels from the last patch that contains each pixel (Section 8.5.4). I also experimented with other strategies like pixel-wise *or*, *and*, *min* and *max* but none performed as well as the voting scheme.

7.2.2.2 Subsampling

I found empirically that patch sizes ranging from $P = I/4$ to $P = I/8$ are small enough to produce RLE sequences of suitable sizes (i.e. with $512 \leq L \leq 3072$) on both IPSC and ARIS datasets, provided that the masks themselves are subsampled down to between $S = 80$ and $S = 160$ (Tables E.1 - E.2). I am using the smallest version of the Pix2Seq architecture which takes 640×640 images as input and this gives the target size for the patches.

IPSC sequences have a large range of image sizes from 900×1700 to 3833×4333 (Table C.3). Therefore, in addition to L exceeding the required limit for the larger images, extracting patches directly from the source images leads to widely varying magnification levels between the different sequences. Therefore, for most of my experiments on this dataset, I first resized the images to $I = 2560$ before extracting the patches. This allows patches that are a quarter of the full image to reach the target size $P = I/4 = 2560/4 = 640$ and is also large enough that we do not lose much resolution in most of the images (Table C.3). Unlike IPSC, the ARIS dataset has much more uniform sizes across its images (Table A.2) so extracting patches directly from the source images with $P = 640$ works fine on this dataset. Also, ARIS images are small enough that directly resizing them to $I = 640$ and skipping patch generation altogether also works without losing too much resolution.

After extracting the patches, I finally subsampled the $P = 640$ patch masks down to either $S = 80$ or $S = 160$ and generated the RLE training data using these subsampled masks. I collected statistics on the amount of degradation in the mask quality as a result of this subsampling, in terms of segmentation metrics (Section E.3) obtained by comparing the subsampled masks to the original ones. This turned out to be $< 10\%$ in most cases (Table E.9), which is sufficient for practical purposes.

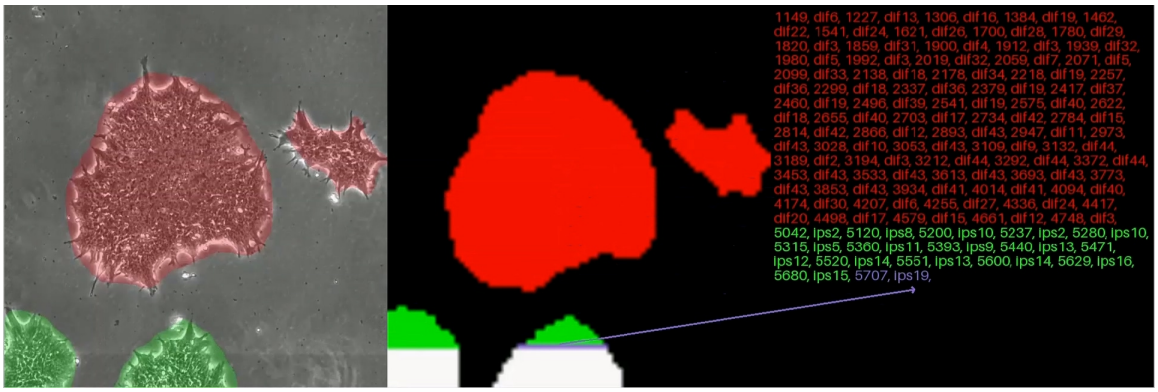

Figure 7.4: Visualization of LAC tokenization of multi-class segmentation mask. The figure shows (from left to right) the source image patch with the two classes shown in red and green, binary version of this mask with the already tokenized segment in red and green depending on the class, and the corresponding tokens. The run that is currently being tokenized is shown in purple. The LAC tokens are shown here as concatenations of class name and length but each such combination represents a single unique token. Animated versions of this figures is available here.

7.2.3 Lengths-As-Class (LAC)

As mentioned before, naïve encoding of multi-class segmentation masks requires 3 tokens per run since we need an extra token for the class. This can unnecessarily increase the sequence length by 50% so we can go back to using 2 tokens per run by combining the length and class tokens into a single composite token that represents both. This can be done by considering the lengths as classes too, such that each unique combination of length and class is represented by a separate LAC token. The total number of LAC tokens is then the product of the maximum length of a run (i.e. S) and the number of classes C. The first S LAC tokens correspond to runs of class 1, next S tokens correspond to class 2 and so on.

This tokenization is particularly efficient when C is small. For example, IPSC dataset has $C = 2$ so that, with $S = 80$, we get the number of *starts* tokens $= 80 \times 80 = 6400$, number of LAC tokens $= 80 \times 2 = 160$ and vocabulary size $V = 6400 + 160 = 6560$. Without LAC tokenization, number of *lengths* tokens $= 80$, number of class tokens $= 2$ and $V = 6400 + 80 + 2 = 6482$. Therefore, we are able to reduce the RLE sequence length substantially without any significant increase in V. However, for very large C, e.g. $C = 80$ as in the COCO dataset [89], V can nearly double ($12.8K$ versus $6.56K$), though still remaining very much within the feasible range (i.e. $< 32K$). Figure 7.4 shows examples of RLE tokenization for multi-class

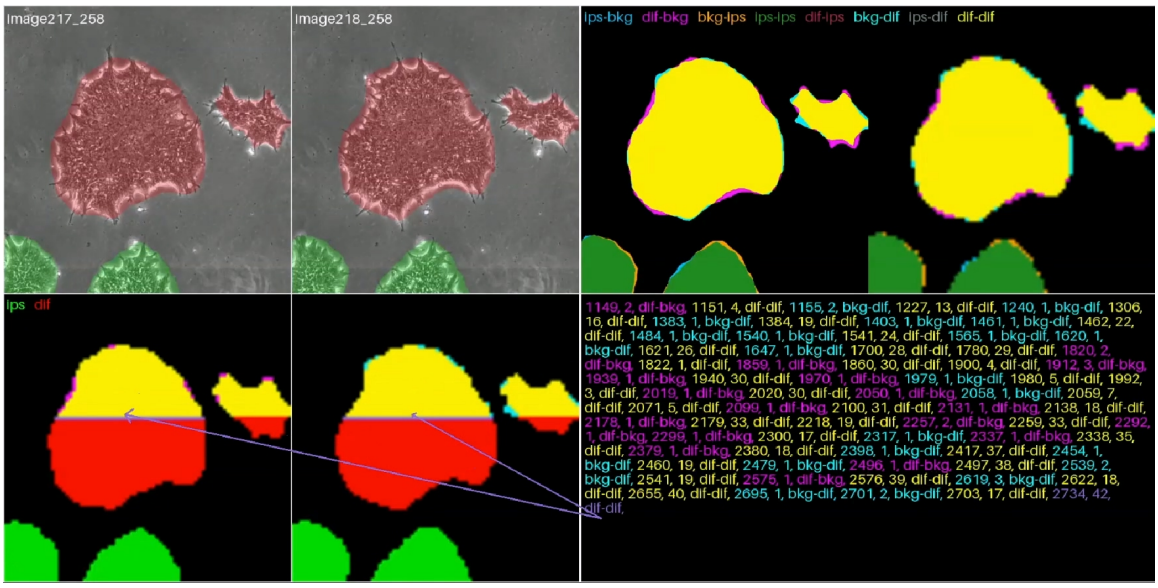

Figure 7.5: Visualization of TAC tokenization for multi-class video segmentation masks with $N = 2$. The top row shows (from left to right) F_1, F_2, full resolution TAC mask, and subsampled TAC mask. The TAC masks show 8 TAC classes whose colors are shown at the top. The bottom row shows individual subsampled F_1 and F_2 masks, partially colored with TAC colors for runs whose tokens are shown on the right. Tokens are colored according to the TAC class in each run except the current one that is shown in purple. Animated version of this figure is available here.

segmentation masks, both with and without LAC tokenization.

7.3 Video Segmentation

A straightforward extension of this idea to perform semantic segmentation on videos would involve flattening the $N \times S \times S$ 3D mask into a 1D vector of size $N \times S^2$ using either row-major or column-major ordering.

As shown in Figure E.1, row-major or C ordering (3D-C) results in the runs for individual images in the video simply getting concatenated together, i.e. all the runs for F_1 come together in a sequence, followed by the runs for F_2 and so on. This does not account for spatiotemporal consistencies in the masks since the runs corresponding to the same object from different video frames are completely unrelated. Column-major or Fortran ordering (3D-F) partially accounts for spatiotemporal mask consistency but has large numbers of very short runs due to small changes in the object position or shape between consecutive frames. This not only significantly increases

Table 7.1: TAC tokens for IPSC dataset with binary and multi-class masks for $N = 2$ and $N = 3$. Here, *bkg* refers to the background while *ips* and *dif* are the two classes in the IPSC dataset. Both classes are represented by *cell* in the binary case. Note that there is no TAC token corresponding to class combination with *bkg* in every frame (i.e. *bkg*-*bkg* for $N = 2$ and *bkg*-*bkg*-*bkg* for $N = 3$) since RLE only represents foreground pixels, which requires that atleast one class must be non-*bkg*. Also, $N = 3$ multi-class case would have $(C + 1)^N - 1 = 3^3 - 1 = 26$ TAC tokens, out of which only 15 are shown for brevity.

Binary Mask ($C = 1$)					
$N = 2$	*bkg*-*cell*	*cell*-*bkg*	*cell*-*cell*		
$N = 3$	*bkg*-*bkg*-*cell*	*bkg*-*cell*-*bkg*	*bkg*-*cell*-*cell*	*cell*-*bkg*-*bkg*	*cell*-*bkg*-*cell*
	cell-*cell*-*bkg*	*cell*-*cell*-*cell*			
Multi-Class Mask ($C = 2$)					
$N = 2$	*ips*-*bkg*	*dif*-*bkg*	*bkg*-*ips*	*ips*-*ips*	*dif*-*ips*
	bkg-*dif*	*bkg*-*ips*	*dif*-*dif*		
$N = 3$	*ips*-*bkg*-*bkg*	*dif*-*bkg*-*bkg*	*bkg*-*ips*-*bkg*	*ips*-*ips*-*bkg*	*dif*-*ips*-*bkg*
	bkg-*dif*-*bkg*	*ips*-*dif*-*bkg*	*dif*-*dif*-*bkg*	*bkg*-*bkg*-*ips*	*ips*-*bkg*-*ips*
	dif-*bkg*-*ips*	*bkg*-*bkg*-*dif*	*ips*-*bkg*-*dif*	*dif*-*bkg*-*dif*	*bkg*-*dif*-*dif*
			.....		

the sequence length but the very short, often unit sized, runs are also likely to be difficult to train on. An example is shown in Figure E.2.

Another problem with the straightforward 3D flattening of video masks (either 3D-C or 3D-F) is that the number of *starts* tokens increases linearly with N by a factor of S^2. This becomes infeasible for $N > 5$ with $S = 80$ and $N > 1$ with $S = 160$.

7.3.1 Time-As-Class (TAC)

These issues can be largely resolved by extending the LAC idea into Time-As-Class (TAC) tokenization to combine the temporal dimension with class IDs so that every possible combination of class IDs across the video frames is represented by a separate TAC token. For example, Table 7.1 shows TAC tokens for $N = 2$ and $N = 3$ for both binary and multi-class cases. Due to the combinatorial nature of TAC tokenization, the total number of TAC tokens becomes $(C + 1)^N - 1$. Even though this increases exponentially with N, V remains practicable for up to $N = 14$ and $N = 9$ respectively for binary and multi-class cases with $S = 80$ (Table E.6) and upto $N = 12$ and $N = 6$ with $S = 160$ (Table E.8). This is aided by the fact that the number of *starts* tokens

now becomes independent of N since the 3D video mask with C classes has effectively been collapsed into a 2D mask with $(C+1)^N - 1$ classes. Figure 7.5, E.3 and E.4 show examples of TAC tokenization for both binary and multi-class cases with $N = 2$ and $N = 3$.

7.3.1.1 Length-and-Time-As-Class (LTAC)

TAC and LAC techniques can be combined to represent each run with only 2 tokens for multi-class video masks too. However, the number of LTAC tokens $= S \times (C+1)^N - 1$ becomes impractical for $N > 8$ and $N > 5$ respectively for binary and multi-class cases. Figure E.5 shows examples of LTAC tokenization for both cases.

7.3.2 Class-wise (CW) Tokenization

As mentioned above, TAC and LTAC (and to a lesser extent LAC) tokenization schemes suffer from the problem of exponential increase in V when C becomes high (e.g. $C = 80$ in COCO dataset [89]). This can be ameliorated by decoupling the class ID from the RLE sequence so that the latter is composed only of *starts* and *lengths*. A simple way to achieve this is to generate RLE tokens for the binary mask corresponding to each class and then concatenating these RLE sequences, separated by the respective class tokens to mark the end of each sequence. The class tokens would therefore serve the dual purpose of separating the RLE sequences for the different classes and specifying the classes themselves. This allows the number of classes for the purpose of computing V for these tokenization schemes to remain constant at $C = 2$, irrespective of the actual number of classes in the dataset.

On the flip side, this representation at least partially nullifies the advantage of small quantum that RLE provides as far as classification accuracy is concerned. When each run is classified separately, even if a few of them are misclassified, it does not affect the overall classification accuracy greatly. However, when all the runs corresponding to a class are classified by a single token, an error in the latter causes all of those runs to become misclassified. Related to this is the fact that the class tokens are greatly outweighed by the coordinate tokens since there are anywhere from a few tens to a few hundreds of coordinate tokens for every class token. This last problem can be handled by increasing the weight of the class token (Section 8.5.1) so that all the coordinate tokens combined have about the same weightage as the single class token.

7.3.3 Instance-wise (IW) Tokenization

If object instance information is available, as in the IPSC dataset, RLE sequences can be generated for the binary masks corresponding to each object instead of each class. These sequences can again be concatenated, separated by the respective class tokens, similar to CW tokenization. This representation allows us to perform instance segmentation in addition to semantic segmentation and therefore achieve full video panoptic segmentation. This also partially solves the problem of weight imbalance between coordinate and class tokens because the number of runs required to represent each object is usually much smaller than those needed to represent all the pixels belonging to each class. Figure E.6 shows an example of IW tokenization for static segmentation masks.

This idea occured to me very late in the development of this thesis and I have only implemented and tested it for static segmentation so far. Early results have shown it to be comparable to the other tokenization schemes in terms of semantic segmentation performance. However, its instance segmentation performance does not compare favourably with object detection models, mainly because of the low resolution of the mask. Training with sufficiently high batch sizes is currently only possible with $S = 80$ and $S = 128$ using respective image sizes of $I = 640$ and $I = 1024$. Given the high resolution of the IPSC images (Table C.3), this requires the original masks to be downsampled by a factor of anywhere from 10 to 50 which is far too large to be able to extract spatially accurate bounding boxes. I have trained a couple of models with $S = 512$ and $S = 640$ too but I had to use batch sizes that are too small to train these successfully. As a result, these models underperform in terms of both semantic and instance segmentation. I have left further exploration of these tokenization schemes as part of future work when more GPU memory is available.

Chapter 8

Results

This chapter presents the results of applying language modeling for both video detection and semantic segmentation on datasets from chapters 3 to 5. Video detection results are compared with those from the baseline Pix2Seq static detection model [31] as well as the conventional deep learning models from chapters 4 and 5. Semantic segmentation results are compared with the conventional models from chapter 3 for the ARIS dataset and a Swin transformer model [26, 306] for the IPSC dataset.

8.1 Datasets

8.1.1 Object Detection

I have evaluated object detection on the folowing three datasets:

- ACAD: I have trained on only 3 of the 8 configurations (Table 4.4) - #1, #3 and #4 - since these are the only ones that contain video information.
- IPSC: I have trained on both early and late-stage training configurations (Section 5.2.3) and tested both sets of models on the same test set containing the first 16 images from each sequence.
- UA-DETRAC [296]: I have included some results from this dataset too since this is a class-agnostic large-scale dataset with long and diverse sequences containing relatively long-term motion information that is missing from the other two datasets. This allows us to at least partially rule out dataset limitations when comparing the static Pix2Seq detector with my proposed video version, especially for larger

Table 8.1: Quantitative details of the datasets used for testing video detection. Number of objects is the total number of frame-level objects (ignoring instance information) while the number of trajectories is a total number of video-level object instances. For example, if an object enters the scene in frame 1 and leaves the scene in the frame 151 without being occluded in any frame, this will count as a single trajectory but 150 objects.

Dataset		ACAD			IPSC		UA-DETRAC
		#1	#3	#4	Early-stage	Late-stage	
Number of Classes		8	6	6	2	2	1
Number of Sequences	Training	33	218	528	31	31	49
	Testing	539	317	535	31	31	37
	Total	572	535	535	31	31	86
Number of Images	Training	8,001	60,003	7,392	1,178	2,263	65,972
	Testing	150,117	88,023	140,628	496	496	51,809
	Total	158,118	148,026	148,020	1,674	2,759	117,781
Number of Trajectories	Training	38	240	578	288	265	5,171
	Testing	612	358	598	240	240	2,229
	Total	650	598	1,176	528	505	7,400
Number of Objects	Training	8,226	65,731	7,937	7,463	11,847	541,933
	Testing	163,157	94,364	152,152	3,248	3,248	659,060
	Total	171,383	160,095	160,089	10,711	15,095	1,200,993

values of N. UA-DETRAC has 100 sequences divided into 60 training and 40 test sequences. I had to exclude 11 training and 3 test sequences since these have long stretches of empty frames (i.e. where no objects are present) and these cause issues with my data processing pipeline. As a result, I used 49 sequences for training and 37 for testing.

Table 8.1 provides quantitative details for these datasets.

8.1.2 Semantic Segmentation

I have evaluated semantic segmentation on the folowing two datasets from chapters 3 and 5:

– ARIS: I have tested on the standard configuration with 32 training and 18 test images (Section 3.3.3). I have also performed image-level ablation tests (Section 3.3.4) with 4, 8, 16 and 24 training images, all of these being tested on the same 18 test images. This dataset does not contain video labels so I have only used it for testing the static segmentation models.

– IPSC: I have tested on both early and late-stage training configurations (Section 5.2.3) and, unlike ARIS, this dataset does contain video labels so I have used it for testing both static and video segmentation models.

8.2 Metrics

8.2.1 Object Detection

I have used a wide range of object detection metrics over the course of my projects since different metrics are more suited to different application domains. These metrics include mAP, mRP and cRP for ACAD (Section 4.4.1), and ROC-AUC, RP-AUC, FN-DET, FP-DUP, and FP-NEX for IPSC (Section 5.3.1, 5.3.2). Although many of these detection metrics are strongly correlated with each other in many scenarios, we have empirically found RP-AUC to correspond best with the overall detection performance for most practical purposes. My principal objective in this chapter is to compare the baseline Pix2Seq static detection model with my video detection model as well as compare between different variants of the latter so I use RP-AUC as the single main metric for this purpose. I do still use the domain-specific metrics proposed in chapters 4 and 5 when comparing the token-based models with the conventional models from those chapters.

RP-AUC measures both localization and classification performance so I also use a class-agnostic version of RP-AUC which I have termed **cRP-AUC**. Similar to the cRP metric in chapter 4, cRP-AUC is computed by considering all the predicted and GT objects to belong to the same class so that misclassifications are not penalized and we are able to measure the localization performance alone. In many practical applications, especially those involving human-in-the-loop systems, being able to correctly detect the presence or absence of an object is more important than classifying it correctly and this metric allows us to measure the suitability of a detector for such systems.

8.2.2 Semantic Segmentation

I have used three main metrics for measuring semantic segmentation performance. The first two are recall and precision as defined in Section 3.4.1.1 and are used for both ARIS and IPSC datasets. The third one is the Dice score, also known as the Dice-Sørensen coefficient [307, 308], which is used only for the IPSC dataset. Dice

score is widely used as a single metric to represent the overall segmentation quality by incorporating both recall and precision. It is defined as:

$$\texttt{dice}_i = \frac{2 \times n_{ii}}{t_i + \sum_j n_{ji}} \tag{8.1}$$

where $1 \leq i \leq C$ is the class index, n_{ij} is the number of pixels of class i predicted to belong to class j and t_i is the total number of pixels of class i in the ground truth. I have also used ice concentration median MAE (Section 3.4.1.2) for the ARIS dataset.

8.3 Training

8.3.1 Setup

I have performed most of the training on three GPU servers, each with 2 × Geforce RTX 3090 24GB GPUs. I have also trained some of the smaller models on a fourth GPU server with 3 × Geforce GTX 1080Ti 11GB GPUs. Finally, I used a fifth GPU server with a Geforce RTX 3090 24GB and a Geforce RTX 3060 12GB for running inference, including validation. Towards the end of my program, I was able to rent a couple of Tesla A100 80GB GPUs and also build another dual RTX 3090 server of my own so a few of the models have also been trained using these. More details of these servers are provided in Table F.1. I trained all the models with the default hyperparameter settings provided by the Pix2Seq authors, except for adjusting the batch size to the maximum that would fit on the GPUs, along with the vocabulary size V and the maximum sequence length L as needed for each model configuration.

8.3.2 Validation

Pix2Seq codebase does not support performing validation as part of the training run. While I did add support for this, I found that it not only slowed down training significantly, but also caused random crashes due to running out of GPU memory. As a workaround, I implemented a remote validation pipeline where the inference server periodically polls the training server for new checkpoints and runs inference on the latest checkpoint thus found. I set the time period between successive polling attempts to the maximum of two hours or the inference time. Since inference requires much less GPU memory than training, it is possible to simultaneously perform validation for multiple training runs on the same inference server.

I performed validation directly on the test set rather than a subset of the training set, as is considered the standard practice. I had to do this both in order to reduce the total training time as well as to utilize as many of the limited number of labeled images for training as possible. A separate validation set is principally needed to prevent overfitting but it turned out that each of our test sets is sufficiently similar to the corresponding training set that the test performance reached a plateau in every single case and did not decline even when training was continued for several hundreds of thousands of iterations after this plateauing. Some of the test sets are too large to complete inference within a reasonable timeframe (e.g. < 10 hours) In such cases, I validated on a small representative subset of the test set (e.g. 10 frames per sequence) after empirically confirming that the performance trend on this subset was consistent with that on the full test set.

8.3.3 Distributed Training

Pix2Seq codebase does support multi-machine distributed training and I used it to train a few models over two or three of the dual RTX 3090 servers in order to use the combined 96 or 144 GB of GPU memory. It would have been extremely beneficial to have been able to do this for all (or at least most) of the models, especially on larger datasets like ACAD and UA-DETRAC. However, Pix2Seq distributed training implementation is optimized for Tensor Processing Units (TPUs) rather than GPUs and this, combined with the relatively slow network connection between the GPU servers, made the overhead so high that the GPU usage during these runs was $< 50\%$ nearly the entire time, and the GPUs spent a good fraction of that time idling (Figure F.1). In addition, one of my servers suffers from hardware incompatibility between its motherboard and RTX 3090 GPUs which causes it to restart randomly after a while if it is used in a distributed training setup. These issues in turn extended the training time to such an extent (often to several weeks) that it made distributed training impracticable for more than a few models. Also, in my experience, a model whose training has been interrupted multiple times due to server restarts never seems to perform as well as a model that was trained continuously in a single training session, especially in case of multi-machine distributed training. This might possibly be due to some obscure bug in the Tensorflow distributed training implementation or perhaps all the optimizer parameters are not getting restored correctly in Pix2Seq so the training does not resume seamlessly after a server restart.

8.4 Performance Overview

This section presents results comparing my proposed token-based video detection and semantic segmentation models with the conventual deep learning models from chapters 3 to 6, along with the baseline Pix2Seq static detection model [31]. Unless otherwise specified, all Pix2Seq models have the 640×640 ResNet-50 as their backbone and the video models use the middle-fusion video architecture (Section 6.2.2.2) with $N = 2$. For the sake of brevity, Pix2Seq static and video detection models are referred to as **P2S** and **P2S-VID** respectively for the remainder of this thesis. The static and video semantic segmentation models are likewise abbreviated as **P2S-SEG** and **P2S-VIDSEG** respectively. I experimented with many different configurations or variants of P2S, P2S-VID, P2S-SEG and P2S-VIDSEG (Section 8.5) but this section only summarizes the best results I found. The specific model configurations that I have included here for each dataset are detailed in Table F.3. Note that I was only able to train a small fraction of all the models I would have liked to have trained due to limited time and computational resources, so these results very likely do not indicate the best performance that these models are capable of, especially in the case of the video models with larger values of N.

8.4.1 Summary

Following are the key takeaways from the results presented in the remainder of this chapter:

- Both static and video language models perform about the same as similarly sized conventional models. As with deep learning in general, the overall performance depends more on the size of the backbone than any specific output modeling.
- P2S-SEG and P2S-VIDSEG models compare more favourably against conventional segmentation models, especially on ARIS dataset, than P2S and P2S-VID do against conventional detectors.
- Pix2Seq models are better at localizing objects than classifying them correctly so that their class-agnostic performance tends to compare more favourably with conventional models than their overall performance.
- Related to the last point is that Pix2Seq models are relatively less robust to class imbalance and tend to overfit to the more numerous class. This causes P2S-SEG

and P2S-VIDSEG to compare more favourably against conventional models in terms of segmentation recall rather than precision.

- This problem of poor classification performance can be partially ameliorated by equalizing the weights assigned to class tokens during training so that each class token has the same weight as all of the corresponding bounding box coordinate tokens combined (Section 8.5.1).

- P2S-VID does perform slightly better overall than the baseline P2S but this improvement is mainly due to the output redundancy (Section 6.1.2.4) obtained by using $T < N$ and this performance advantage mostly disappears by setting $T = N$.

- P2S-VIDSEG does not show any consistent improvement over P2S-SEG, probably because the stride-based redundancy advantage does not apply to semantic segmentation.

- There is no consistent improvement in detection or segmentation performance with increase in N (Section 8.5.4).

- Static models trained to predict video outputs by processing only the first frame in each video temporal window (Section 8.5.3.2) are able to keep up with the video models surprisingly well, even for large values of N, indicating that the latter are not able to make sufficient use of the video information.

- Video models exhibit strong signs of being bottlenecked, especially for larger values of N, by the low batch sizes (Section 8.5.2) and possibly also the relatively small amount of training data I had to use.

8.4.2 Object Detection

8.4.2.1 ACAD

Figure 8.1 summarizes object detection performance on all three configurations of the ACAD dataset. The overall performance of the language models is comparable to conventional models with similarly-sized backbones, especially RETINA with its identical ResNet-50 backbone and YOLO with its comparable DarkNet-53. Once classification accuracy and overfitting are removed from the equation, Pix2Seq models are in fact able to match the significantly larger RES101, RFCN and NAS,

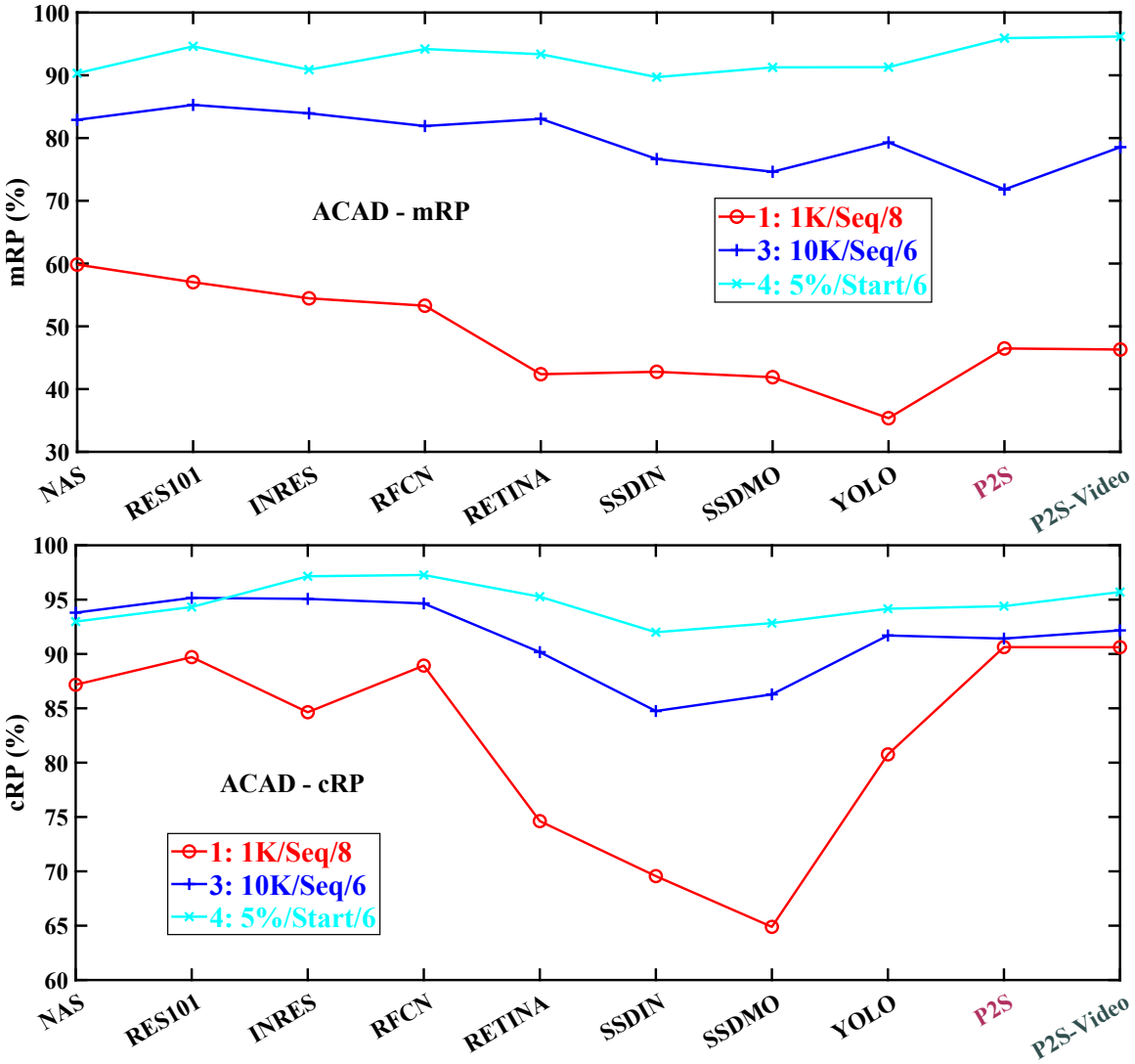

Figure 8.1: Object detection results on ACAD dataset configurations #1, #3 and #4 (Table 4.4) in terms of (top) mRP and (bottom) cRP (Section 4.4.1). Note that different Y axis limits have been used on the two plots to maximize the visible performance difference between the models. P2S and P2S-VID respectively refer to the static and video detection Pix2Seq models. Remaining model acronyms are specified in the caption of Figure 4.3. P2S-VID uses middle fusion architecture with $N = 2$ and was trained with frozen backbone and class token weight equalization. A bar plot version of this figure is available in Figure F.2

as can be seen from the cRP results for all three configs and mRP results for config #4. Even though config #4 has the smallest training set, it is the easiest to handle from a classification standpoint because the training set contains frames from nearly

every sequence in the test set and therefore the model is able to train on every combination of backgrounds and foregrounds available therein. While the other two configs contain more frames overall, they have no overlap between the training and test sequences and offer the models access to far fewer combinations of foregrounds and backgrounds in the test set. Models like P2S and P2S-VID that have a tendency to overfit are therefore penalized much more heavily by these configs. P2S-VID mostly performs about the same as P2S, but it does significantly outperform the latter in the challenging config #3 which is the only one that has enough frames to at least partially alleviate the bottleneck imposed by the relatively small datasets.

8.4.2.2 IPSC

Figure 8.2 shows classification metrics for both early and late stage configurations of the IPSC dataset. The language models rank between the static and video instance segmentation models, with P2S-VID being closer to the latter than the former. This is quite impressive considering that all of these conventional models have significantly larger backbones - either Swin Transformer or ConvNext - than the Pix2Seq models. In addition, since these models perform instance segmentation rather than simple object detection, they have access to the object masks in addition to the bounding boxes during training. This additional information provides these models with a significant advantage in classifying the two types of cells since the shape of the cell boundary is one of the most important cues used by human experts to do the same. P2S-VID shows a more strongly marked improvement over P2S here, especially on the more challenging late-stage configuration and in terms of partial AUC that measures classification accuracy under high precision scenarios.

Figure 8.3 shows the high-level detection metrics RP-AUC and AP on this dataset. Both language models compare significantly more favourably with the conventional models over the high-level detection metrics than the classification metrics. This make sense given the relatively poor classification performance of these models. P2S-VID in particular stands out here, outperforming P2S by nearly 20% in RP-AUC and 10% AP on the late-stage dataset. In fact, P2S-VID turned out to be among the two best models on this dataset, with only IDOL significantly outperforming it in AP. Figure 8.4 shows the corresponding low-level detection metrics. P2S-VID remains among the two best models in terms of FN DET, but both language models perform poorly on the FP metrics, especially FP NEX-WHOLE. This is again consistent with

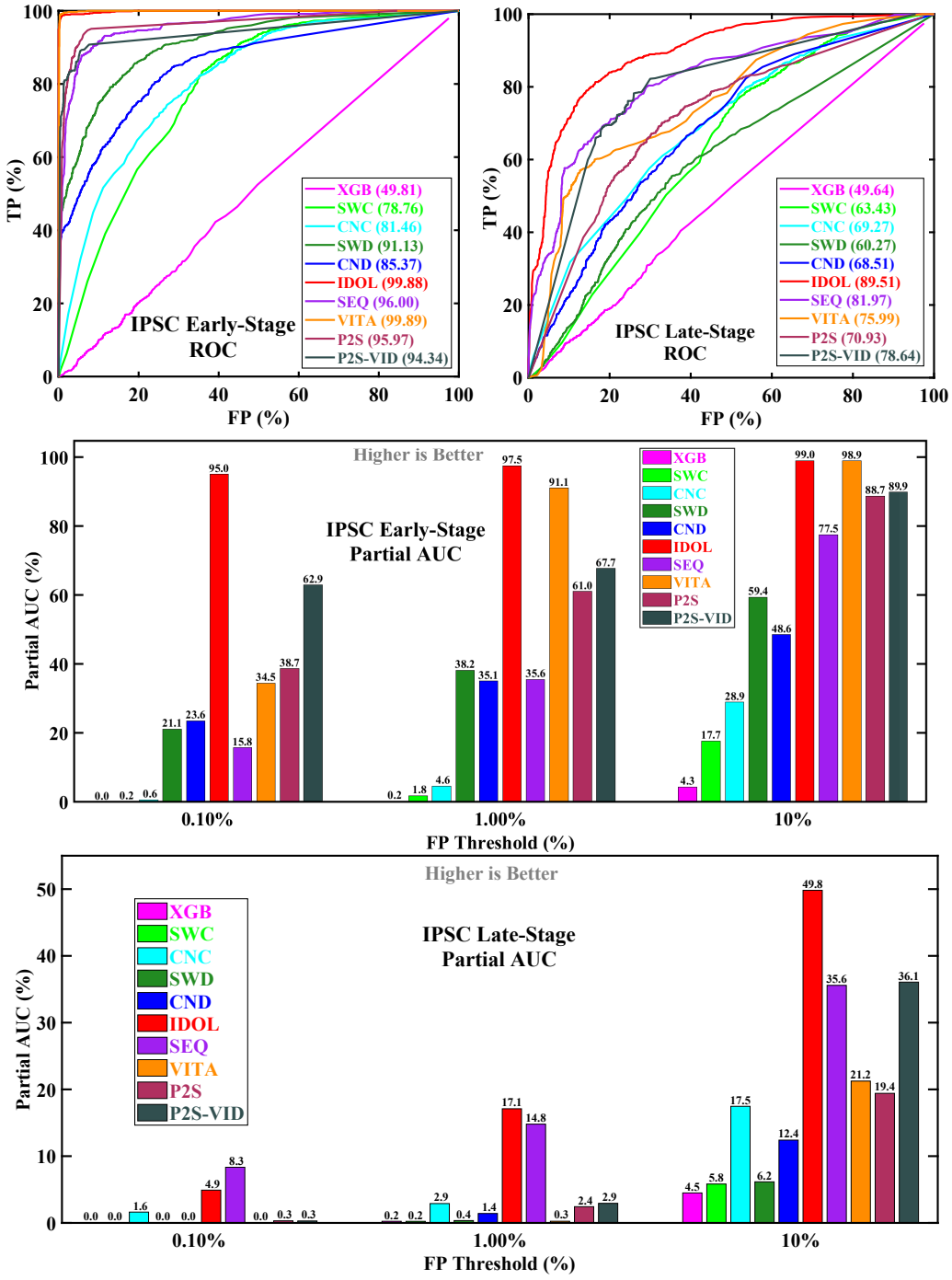

Figure 8.2: Classification metrics for both early and late-stage training configurations of the IPSC dataset. Top row shows ROC curves with respective AUC values (%) in the legend. Middle and bottom rows show the partial ROC-AUC for three different FP thresholds.

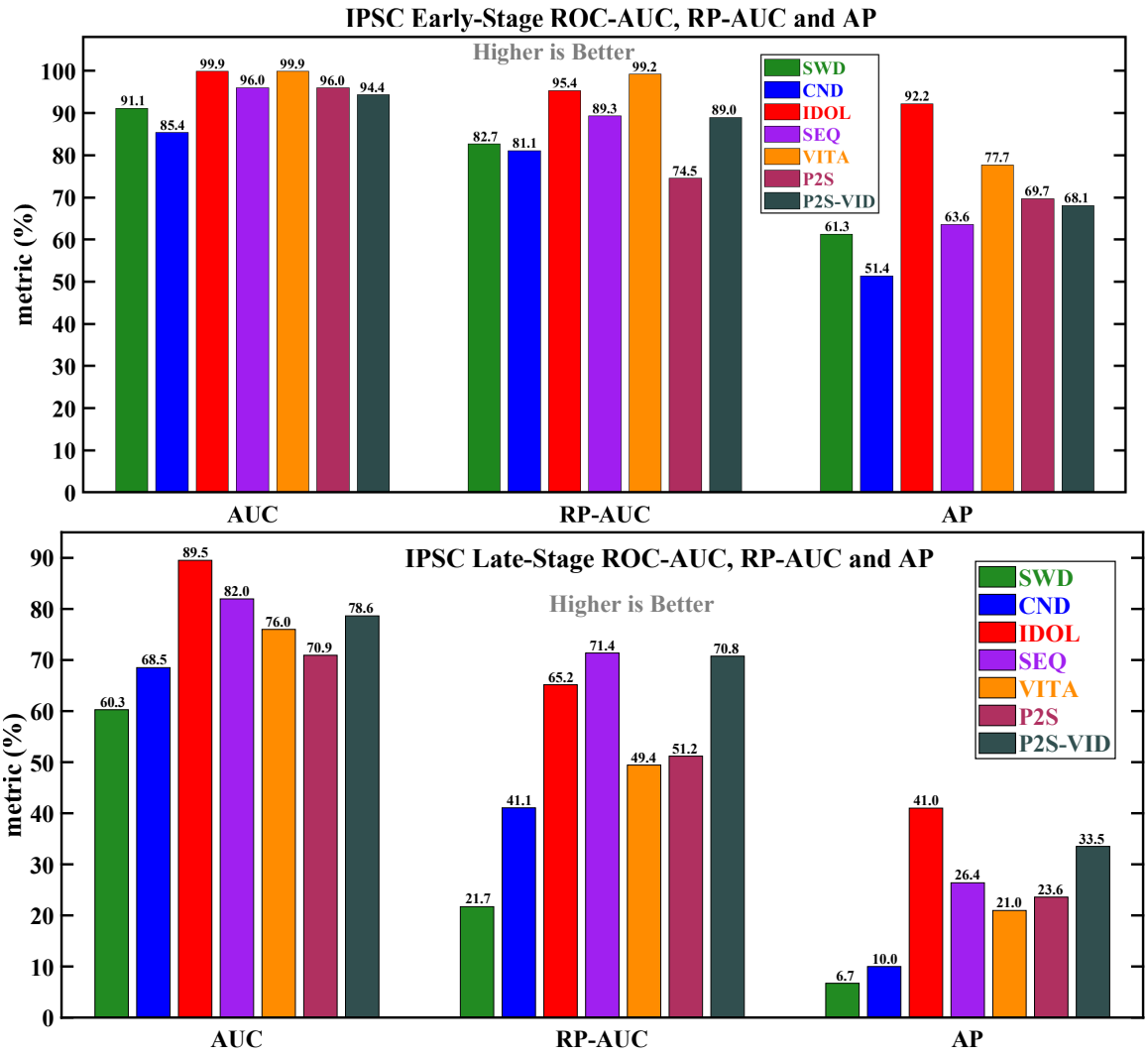

Figure 8.3: High-level detection metrics for both early and late-stage training configurations of the IPSC dataset. ROC-AUC is also shown for comparison.

their tendency to misclassify. As mentioned in Section 5.4.2, FP NEX essentially measures how many of the unlabeled cells are incorrectly detected and classified as IPSCs, something which the relatively poor classification ability of the language models makes them susceptible to.

8.4.2.3 UA-DETRAC

Figure 8.14 shows a summary of results on the UA-DETRAC dataset for P2S and P2S-VID with $N = 2$ to $N = 32$. I had hoped that the size of this dataset would allow the severe bottleneck on the video models to be at least partially overcome

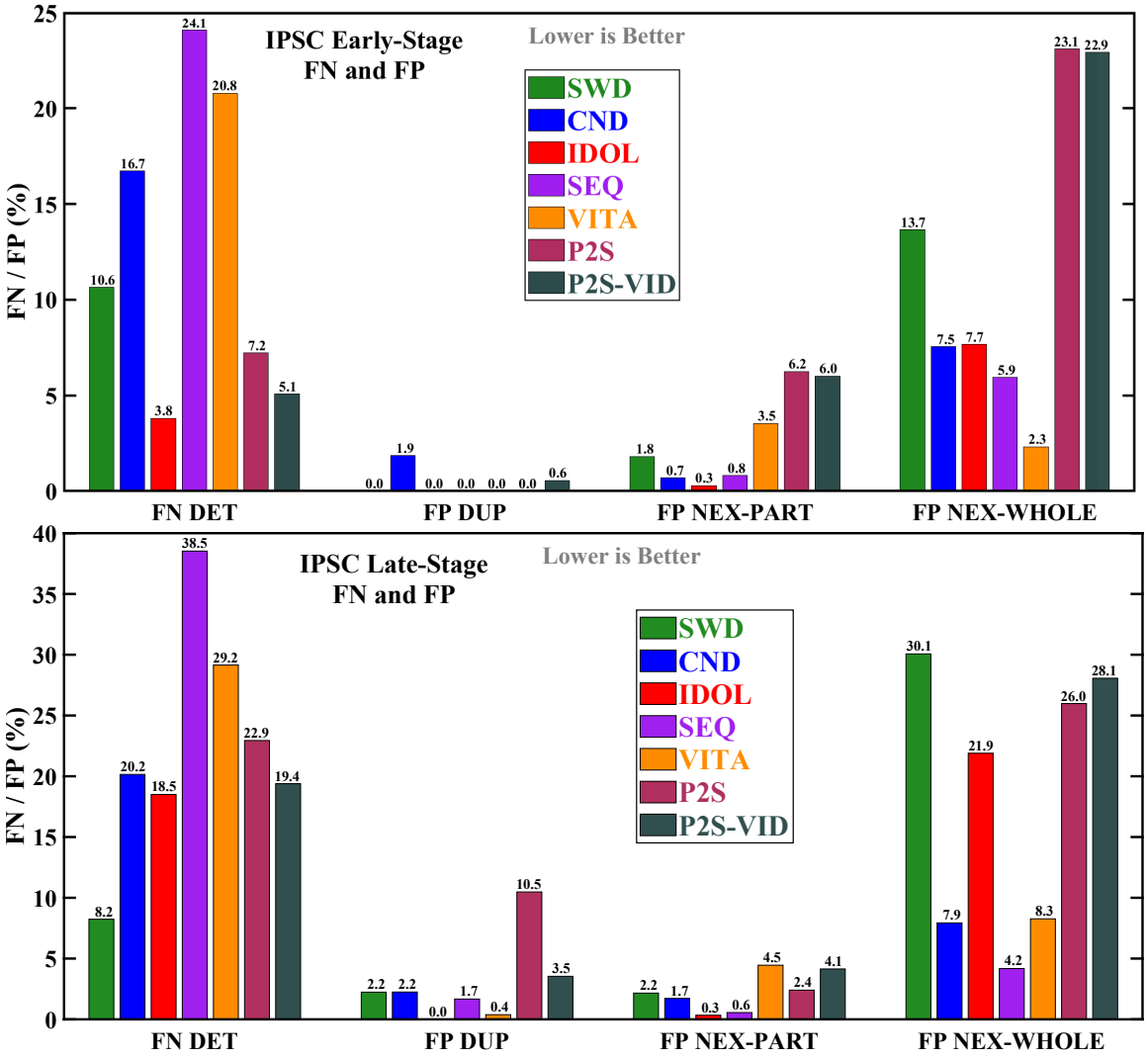

Figure 8.4: Low-level detection metrics for both early and late-stage training configurations of the IPSC dataset. Note the different Y-axis limits on the two plots.

but that turned out not to be the case. All the models, including P2S, appear to be limited by the batch size here, judging by the identical ceiling of around 85% that they all reach before N becomes too large. As seen in the example of ACAD #3 (Figure 8.8), the training batch size seems to be an even more important bottleneck than the size of the dataset. In fact, the optimal batch size probably increases with the size and complexity of the dataset, so the models are likely to be even more bottlenecked on UA-DETRAC than ACAD. It is true that the ACAD example demonstrated performance limitation only on the classification task and this is not relevant with UA-DETRAC since it has only one class. However, UA-DETRAC has

much longer and more complex trajectories than ACAD. This makes its localization task a lot more challenging and therefore just as likely to be bottlenecked as classification on ACAD. P2S-VID models upto $N = 8$ are able to mostly match P2S with $T = 1$ but the performance drops sharply after that, thus confirming the inadequacy of my existing hardware for training such large models. I would expect that all the models, including P2S, would be able to reach $> 90\%$ if trained with sufficiently large batch sizes.

I would like to conclude this section with a recent result which is a promising step in this direction. I generated this by training models on two Tesla A100 80 GB GPUs that I rented to produce publication-quality results that are competitive with the current state of the art. VSTAM [309] is the top-ranked model on the leaderboard [310] at the time of this writing and has been so since it was released nearly 3 years ago. However, as shown in Table 8.2, one of my models was able to outperform VSTAM by 0.75%, so would take this spot once these results are published. This model uses the late-fusion architecture which turns out to significantly outperform the other two fusion schemes once the batch size bottleneck is alleviated, as I had hypothesized in Section 6.2.2.3 based on its theoretical advantage of cross-attending every token with each of the N frames. Also, this model was trained with the smallest possible video length of $N = 2$ because even 160 GB GPU RAM is apparently insufficient to relieve the bottleneck on models with higher N (Table F.2). I was hoping to rent 4 or even 8 of these GPUs to better exploit the video length but these configurations were unfortunately not available. This result is perhaps the most convincing evidence I have that most of the language modeling results in this chapter are of bottlenecked models and do not represent their true potential.

8.4.3 Semantic Segmentation

8.4.3.1 ARIS

Table 8.3 presents a summary of results on the ARIS dataset. P2S-SEG performs remarkably well here and turns out to be either the first or the second best model in nearly every case. As in the last section, language modeling is particularly effective at class agnostic tasks, as represented by the ice+water metrics, where it outperforms all the other models by a large margin. Somewhat paradoxically, its performance drops significantly at the frequency weighted version of this metric, especially in terms of precision. All other models find this metric easier since it gives greater weightage to

Table 8.2: Performance of current state of the art video detection models on UA-DETRAC. These results were generated using the entire UA-DETRAC dataset, including the 14 sequences with empty frames (Section 8.1.1). This makes them not directly comparable to other UA-DETRAC results in this chapter, although I would expect the extra sequences to have minimal impact on the performance.

Model	mAP (%)
TFEN [311]	82.42
YOLOv3-SPP [312]	84.96
MSVD_SPP [313]	85.29
SpotNet [314]	86.8
FFAVOD-SpotNet [315]	88.1
VSTAM [309]	90.39
P2S-VID	**91.14**

Table 8.3: Segmentation recall, precision and ice concentration median MAE on ARIS dataset for P2S-SEG and conventional deep learning models along with SVM. The *fw* in *ice+water (fw)* stands for frequency weighted and the corresponding recall and precision metrics refer to `pix_acc` (Eq. 3.1) and `fw_iou` (Eq. 3.4), as detailed in Section 3.4.1. Relative increase over SVM in recall and precision is computed as $(\texttt{model\_value} - \texttt{svm\_value})/\texttt{svm\_value} \times 100$ while the relative decrease in median MAE is computed as $(\texttt{svm\_mae} - \texttt{model\_mae})/\texttt{svm\_mae} \times 100$. The best and the second best models in each case are shown in bold and highlighted in green and yellow respectively. This data is shown as a bar plot in Figure F.3.

Model	anchor ice		frazil ice		ice+water		ice+water (fw)		Median MAE (%)	
	Metric Value	Relative Increase	Metric Value	Relative Increase	Metric Value	Relative Increase	Metric Value	Relative Increase	Metric Value	Relative Decrease
	Recall (%)								anchor ice	
SVM	61.54	-	75.41	-	78.12	-	84.93	-	8.37	-
Deeplab	74.46	21.00	**87.51**	**16.05**	**86.38**	**10.57**	**90.87**	**7.00**	**4.71**	**43.80**
UNet	73.75	19.85	84.27	11.75	85.13	8.97	88.69	4.42	6.51	22.29
DenseNet	76.96	25.06	71.06	-5.77	81.42	4.22	85.02	0.11	7.24	13.59
SegNet	**82.31**	**33.75**	68.99	-8.51	83.06	6.32	85.90	1.14	6.48	22.61
P2S-SEG	**82.70**	**34.39**	**87.47**	**15.99**	**98.27**	**25.79**	**91.27**	**7.46**	**5.34**	**36.20**
	Precision (%)								frazil ice	
SVM	43.32	-	63.07	-	65.84	-	76.84	-	7.18	-
Deeplab	**62.39**	**44.03**	**77.14**	**22.32**	**77.25**	**17.33**	**84.32**	**9.72**	**4.52**	**37.01**
UNet	54.89	26.72	71.17	12.84	73.19	11.17	**81.73**	**6.36**	6.80	5.22
DenseNet	48.98	13.07	60.97	-3.32	67.69	2.82	77.49	0.84	7.20	-0.30
SegNet	52.80	21.90	62.60	-0.73	69.60	5.72	78.46	2.10	6.64	7.51
P2S-SEG	**57.73**	**33.26**	**71.88**	**13.98**	**87.57**	**33.01**	77.98	1.48	**6.09**	**15.18**

water which constitutes a majority of these images (Table 3.2) and is relatively easy to separate from ice. Figure 8.5 shows the results of image-level ablation testing on

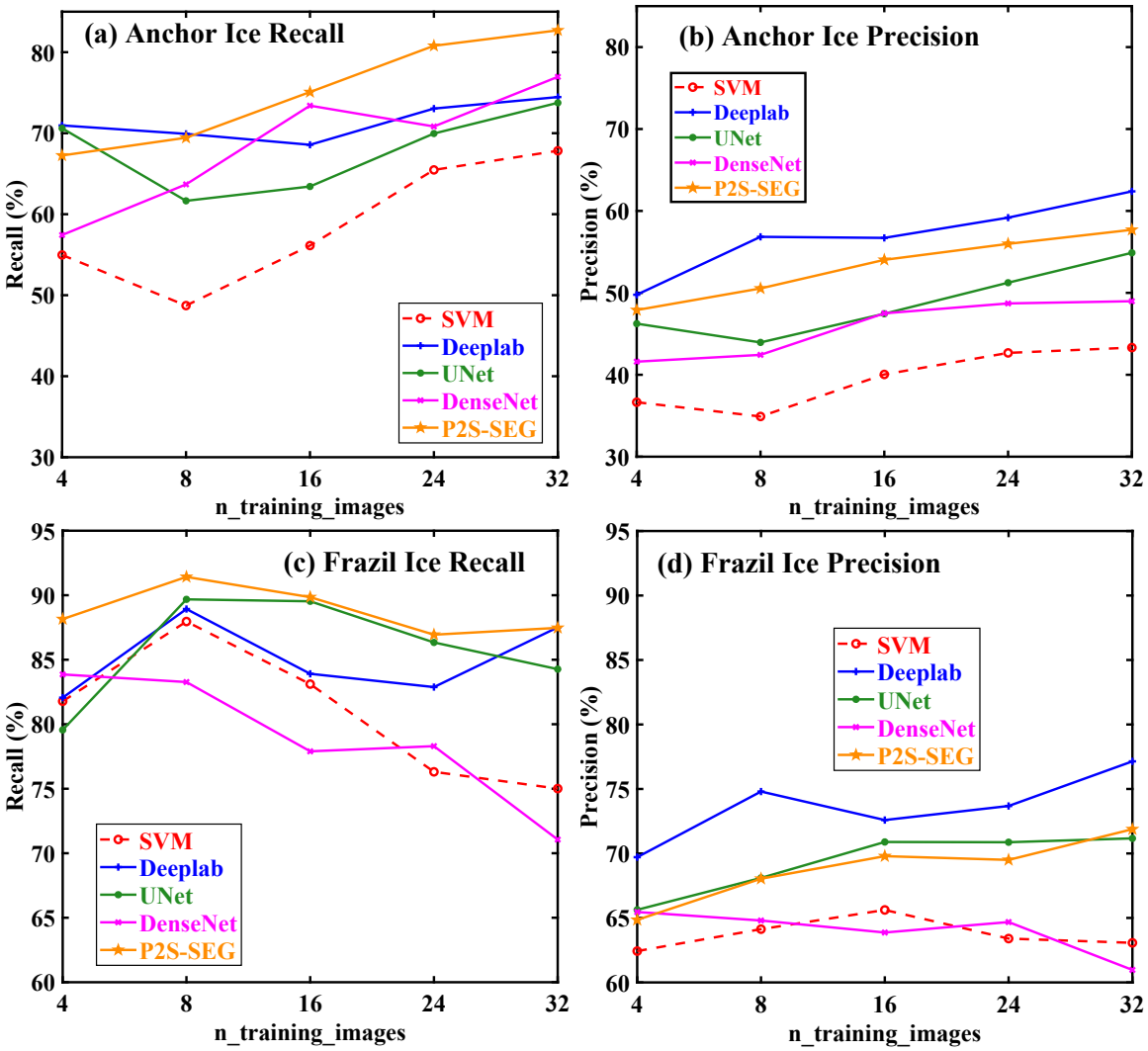

Figure 8.5: Results of ablation tests with training images on ARIS dataset for (a-b) anchor ice and (c-d) frazil ice. Note the variable Y-axis limits between the top and bottom plots.

this dataset. P2S-SEG remains the best model in terms of recall and among the best two models in terms of precision. It predictably finds frazil ice precision most difficult to handle since there is significantly more frazil ice than anchor ice in the training images (Table 3.2). This causes the model to overfit to this class and misclassify anchor ice as frazil ice which in turns lowers the corresponding precision.

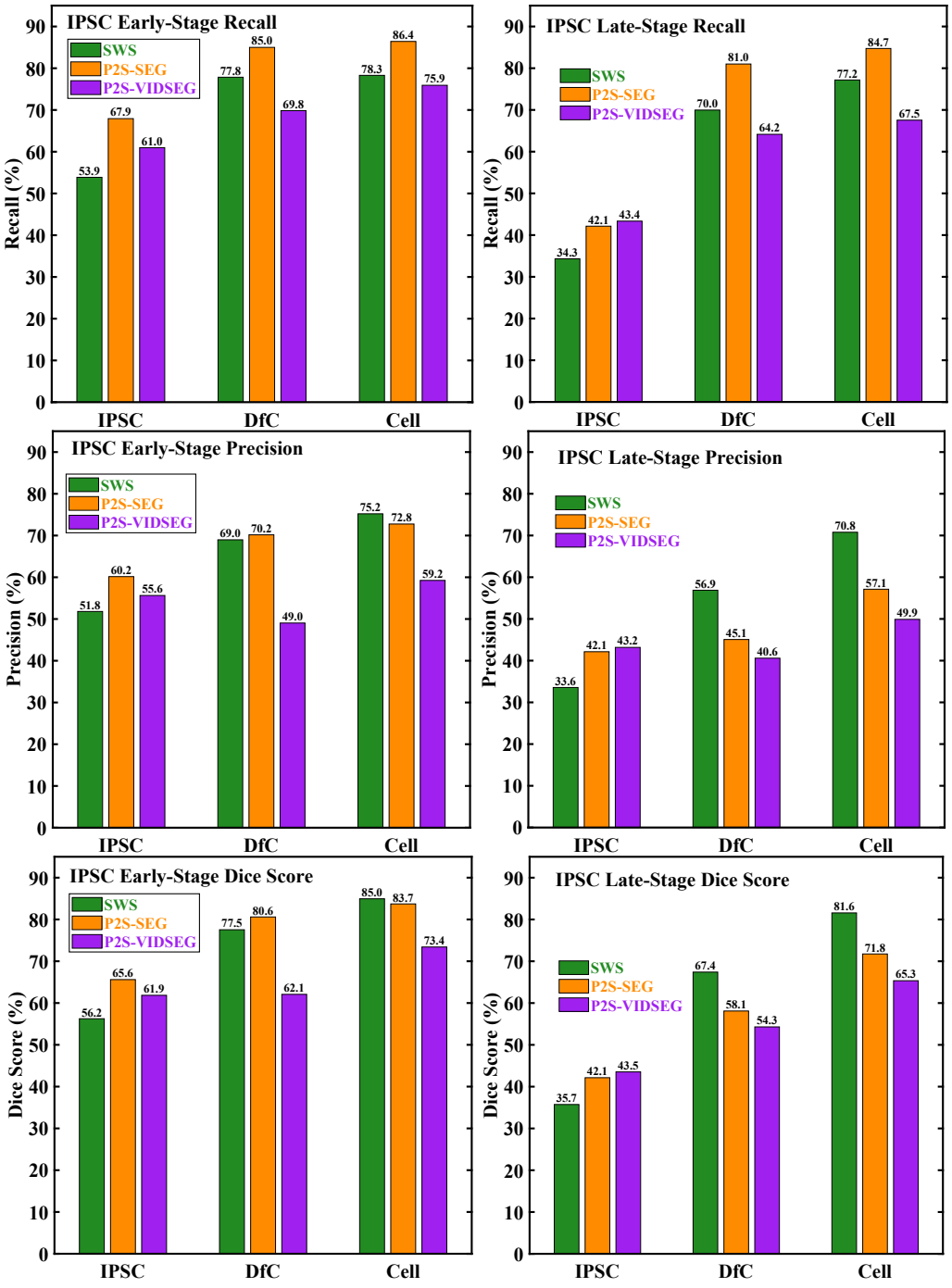

Figure 8.6: Semantic segmentation results on both (left) early and (right) late-stage IPSC datasets in terms of (top to bottom) recall, precision and dice score. *IPSC* and *DfC* on the x-axis refers to the two classes of cells while *Cell* represents the class-agnostic case where all the cells are considered as belonging to the same class.

8.4.3.2 IPSC

Figure 8.6 shows the segmentation results on both early and late-stage training configurations of the IPSC dataset. Conventional modeling is represented here by the single model SWS [26] which can be taken to represent the current state of the art in semantic segmentation. Language modeling compares less favourably against conventional modeling on this dataset than on ARIS, possibly because SWS is a newer and bigger transformer-based model as opposed to the older CNN-based models used on ARIS. Even so, P2S-SEG performs about the same as SWS overall, being slightly better on the early-stage dataset and slightly worse on the late-stage variant. Similarly, P2S-SEG is notably better on *IPSC* class while SWS is better on *DfC* and *Cell*. Finally, P2S-SEG fares better in terms of recall while SWS has an edge on precision. P2S-SEG also outperforms P2S-VIDSEG in nearly every case except the most challenging case of *IPSC* recognition on the late-stage dataset, where the latter outperforms both P2S-SEG and SWS on all three metrics.

8.4.3.2.1 Caveat

Semantic segmentation is very difficult to evaluate quantitatively and the performance numbers exhibit complex trade-offs between the various metrics as well as between the three classes. A couple of example are provided in Tables F.4 and F.5 that list the values of these metrics along with some others (Section 4.4.1) on the validation set during the training runs for P2S-SEG on early-stage and P2S-VIDSEG on late-stage dataset respectively. I chose P2S-SEG and P2S-VIDSEG models for inclusion here by looking for a good balance between all these conflicting considerations among candidate checkpoints once the validation performance had plateaued over all the metrics.

When looking for the best compromise, I did have a bias in favour of *IPSC* among classes (*Anchor Ice* for ARIS) and Dice score among metrics, since these seem to me to best represent the overall segmentation quality. Further, there are 2-3 times as many *DfC*s as *IPSC*s in the GT (Table C.2) so models tend to perform better on *DfC* (and *Cell*) later in their training run when they start overfitting to the more numerous class, something I wanted to avoid. A different choice of trade-offs can generate quite different comparative plots so these results should be interpreted guardedly. I have not implemented a live validation pipeline for SWS so the results shown here were generated using the latest checkpoint once the training curve had plateaued. This is

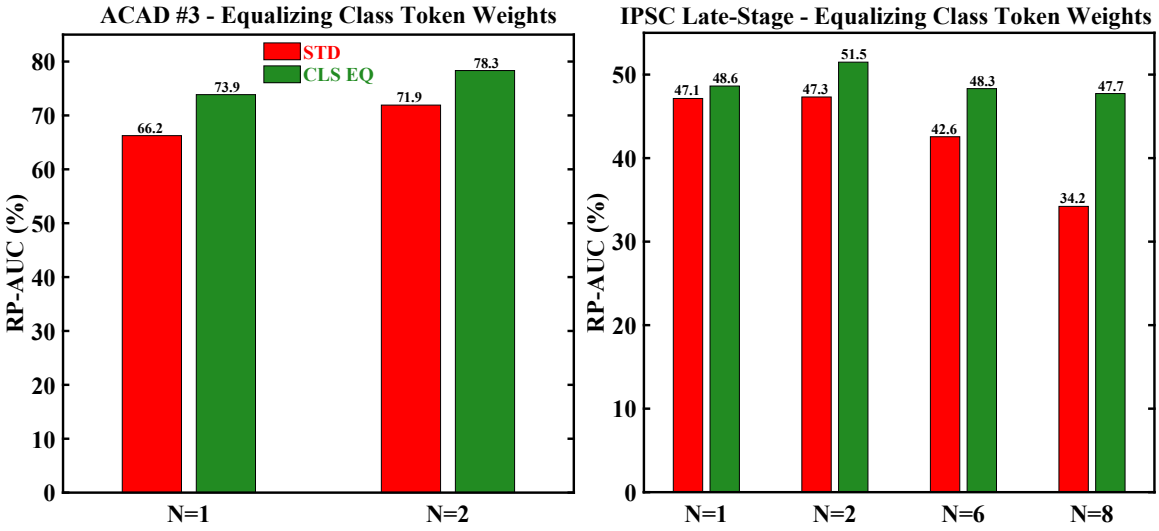

Figure 8.7: Performance impact of equalizing the class token weights on (left) ACAD #3 and (right) IPSC late-stage datasets. *N=1* denotes the baseline P2S model. Models trained with and without class token weight equalization are respectively shown in green and red and denoted with *CLS EQ* and *STD* in the legend. Note the different Y-axis limits in the two plots.

why SWS shows a consistent tendency to outperform P2S-SEG in terms of *DfC* and *Cell* but is outperformed by the latter in terms of *IPSC*. The best we can say with certainty is that the overall performance of language and conventional models is fairly similar, the former being slightly better at recall and the latter at precision.

8.5 Experiments with Parameters

This section presents the results of my experimentation with some of the important model parameters that were useful in finding the optimal models that are reported in Section 8.4. I used the IPSC late-stage dataset for most of these experiments since it is small enough to allow training a large number of models while at the same time also being challenging enough to be able to discriminate between these models.

8.5.1 Equalizing Class Token Weights

A possible reason for the poor classification performance of Pix2Seq models, especially in object detection, might be that the weight assigned to the class tokens during training is equal to that assigned to every single coordinate token. Since the number of coordinate tokens is $4 \times N$ times greater than the number of class tokens, this

effectively means that the localization task is assigned $4 \times N$ times greater weightage than the classification task. In order to address this disparity, I trained detection models with each class token assigned the same weight as all of the corresponding coordinate tokens combined.

As shown in Figure 8.7, this does improve the classification performance appreciably, particularly on large datasets and with higher vlues of N. The degree of improvement also increases with N, especially in the IPSC case. This makes sense since the factor by which coordinate tokens are over-weighted with respect to class tokens without equalization increases linearly with N as does the overemphasis on the localization task. This improvement does come at the cost of significantly slower training, at least in the case of larger datasets. For example, ACAD #3 took 700K iterations and close to 12 days to reach convergence with class equalization and only 300K iterations and just over 5 days without it. These disparities were not as strongly marked on the IPSC dataset, where both sets of models took approximately the same amount of time to converge.

8.5.2 Training Batch Size

Extensive experiments have shown to me that the training batch size matters a lot more for large datasets like ACAD and UA-DETRAC rather than smaller ones like IPSC and ARIS. It also has a greater impact on video models than static ones, especially with larger values of N. Finally, it has far greater impact on the classification task (i.e. RP-AUC) than localization (i.e. cRP-AUC). Fig. 8.8 shows an example for both P2S and P2S-VID training on ACAD #3. It can be seen that the peak RP-AUC on the validation set nearly doubles from 37% to 66% for P2S and nearly triples from 28% to 78% for P2S-VID. On the other hand, cRP-AUC peaks at just above 90% in all four cases and this is reached in just a few thousand iterations as opposed to RP-AUC which takes from 200K to 700K iterations to attain its plateau. The significantly greater relative increase in the case of P2S-VID (2.76 times vs. 1.75 times for P2S), along with the much lower absolute peak with the smaller batch size (28% vs. 37%) confirms the much greater bottleneck faced by the video models.

P2S-VID here uses only $N = 2$ and this bottleneck only gets worse as N increases. In my experience, and subject to the size and complexity of the dataset, the optimal batch size required for a video model to achieve its full potential increases at least

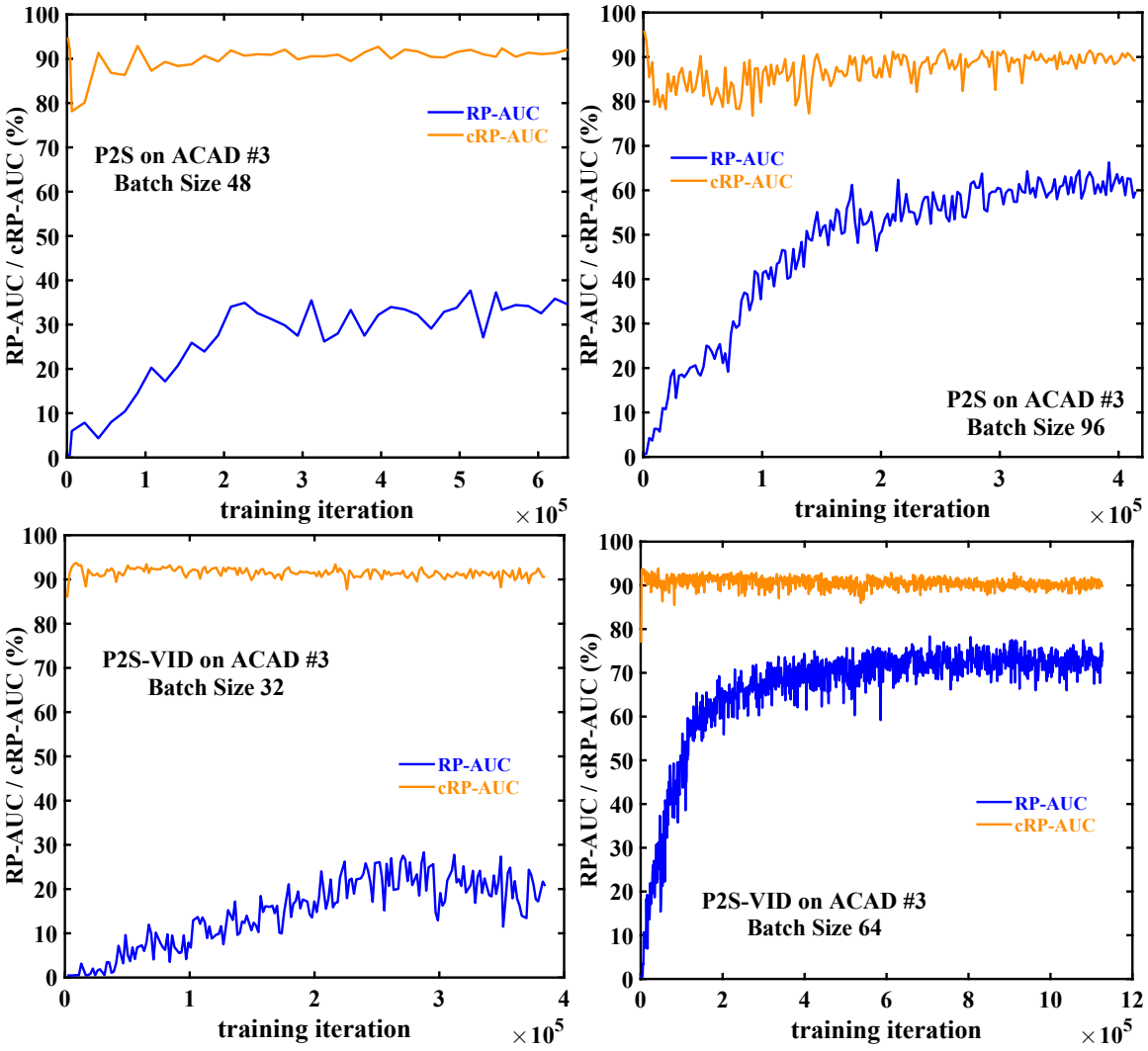

Figure 8.8: Impact of batch size on validation performance when training (top) P2S and (bottom) P2S-VID on ACAD #3. The left plots show the results for training on a single RTX 3090 GPU with batch sizes 48 and 32 while the ones on the right show dual-GPU training with batch sizes 96 and 64.

linearly with N. However, the GPU memory required to train a video model also increases linearly with N even if the batch size remains unchanged. Since the total amount of available GPU memory is fixed, we have to actually *decrease* the batch size linearly with N.

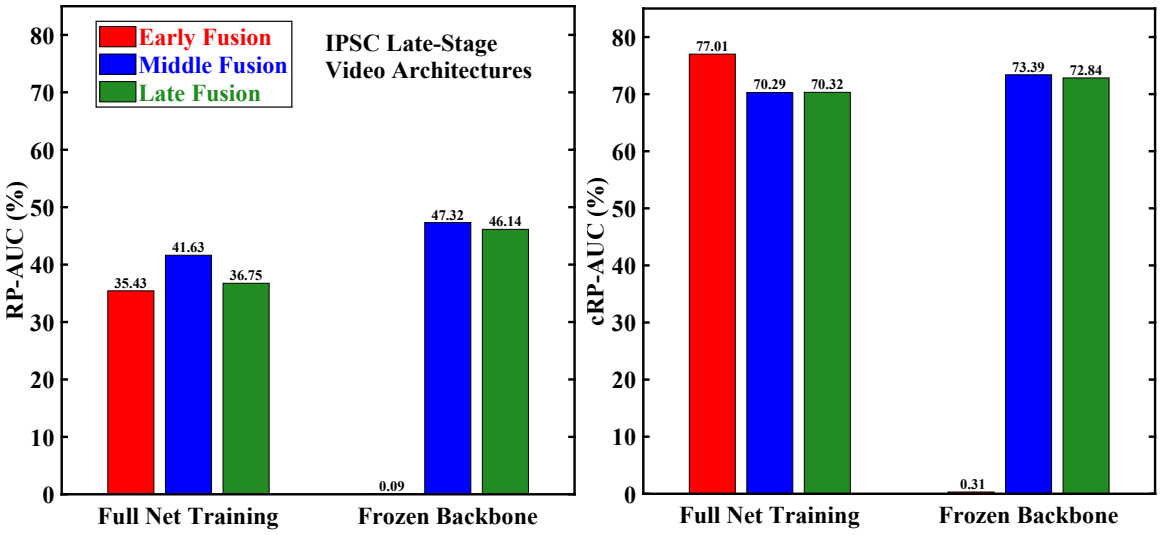

Figure 8.9: Comparing the three video architectures (Section 6.2.2) with $N = 2$ and trained with and without frozen backbone. The left and right plots show RP-AUC and cRP-AUC respectively. All models were trained on the IPSC late-stage dataset.

8.5.3 Video Architecture

Figure 8.9 shows comparative results for the three video architectures (Section 6.2.2) with $N = 2$. Middle-fusion outperforms the other two models in terms of mRP while early-fusion performs best in terms of cRP. I have seen this trend of early-fusion models performing much better on the localization task than classification in other models I have trained as well. The video Swin transformer backbone I am using for early-fusion was pre-trained for human action-recognition and therefore unsurprisingly finds it easier to localize objects than to classify them correctly. Although late-fusion seems slightly inferior to middle-fusion in these results, I have seen it outperform the latter with similar margins on other values of N so, on the whole, the two models can be said to perform at par.

8.5.3.1 Frozen Backbone

I would have expected that the entire Pix2Seq network would need to be retrained in order to work well on a new task – either video detection or semantic segmentation - but this turned out not to be the case. As shown in Figure 8.9, keeping the backbone frozen actually leads to significantly better video detection performance for both middle and late-fusion architectures. However, this is not true for early-fusion, which completely fails to learn anything useful with its video Swin

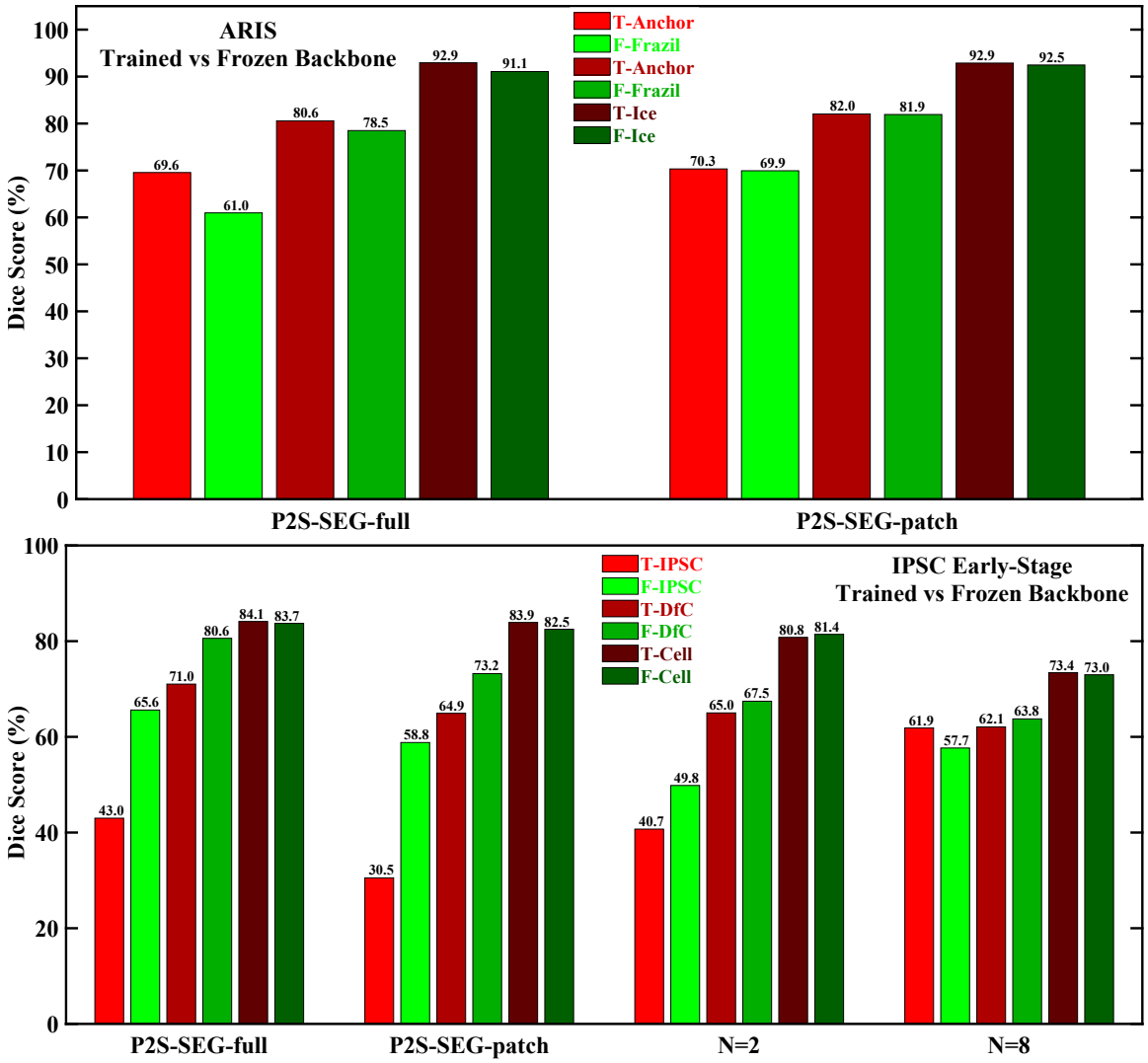

Figure 8.10: Comparing the segmentation performance of models trained with and without frozen backbones on the (top) ARIS and (bottom) IPSC early-stage datasets. Models trained with and without frozen backbones are respectively shown in shades of green and red and denoted in the legend with prefixes F and T. The P2S-SEG suffixes *patch* and *full* respectively refer to models trained with and without sliding window patches. In the former case, $P = 640$, $S = 80$ for both datasets, $I = 1280$ for ARIS and $I = 2560$ for IPSC. In the latter case, $I = P = 640$ for both datasets, $S = 160$ for ARIS and $S = 320$ for IPSC.

transformer backbone frozen. This backbone was pretrained for action recognition with conventional modeling and it seems that weights optimized for conventional modeling cannot generalize to language modeling without retraining. Nevertheless, these weights are useful in fine-tuning the network for language modeling since I

found that training the early-fusion models without loading the pretrained weights for its backbone leads to significantly worse performance. Note that freezing the backbone allows us to use much higher batch sizes (e.g. 80 vs. 20 with $N = 2$) which must partially account for this improvement as well.

Figure 8.10 shows that the benefit of keeping the backbone frozen is much less obvious in case of semantic segmentation. This might be explained by the fact that RLE tokenization differs from static detection tokenization a lot more than does video detection. In fact, given this difference, it is remarkable that frozen backbone is mostly able to keep up with full network training even in case of P2S-VIDSEG with $N = 8$. This might at least partly be due to the bottleneck imposed by the insufficient batch size, without which the full network training might well lead to much better performance.

8.5.3.2 Video Output with Static Input

I wanted to find out how much useful information the network is able to learn from video frames so I trained static models to predict video output (either detection or segmentation) using only the first frame F_1 in each video temporal window as input. This means that the model is trained to produce the same output as a P2S-VID or P2S-VIDSEG model but it only has access to the first frame in each temporal window, rather than all N frames. It therefore needs to use the first frame to predict the contents of the future $N-1$ frames in the sequence. Since there is no more video input, we can use the baseline P2S architecture for these models. I trained models with $N = 2$, $N = 4$, $N = 6$ and $N = 8$, all with the backbone frozen, and all on the IPSC late-stage dataset. Note that these static-video models can be trained with the same (and much larger) batch size as P2S, irrespective of N. This gives them an advantage over the true video models whose batch size decreases linearly with N. The video stride T is important in evaluating these models since $T = 1$ ensures that each frame would be the first frame in some temporal window so that the static input models can output valid boxes only for the first frame and still not be penalized during inference. However, this is unlikely to happen in practice since the model is trained to output boxes for all N frames and therefore will be penalized during training for learning a simple strategy like this.

P2S-VID results for IPSC late-stage and UA-DETRAC datasets are shown in Figures 8.11 and 8.12 respectively. All video models were trained with middle-fusion architecture and without class-weight equalization. Static input models show

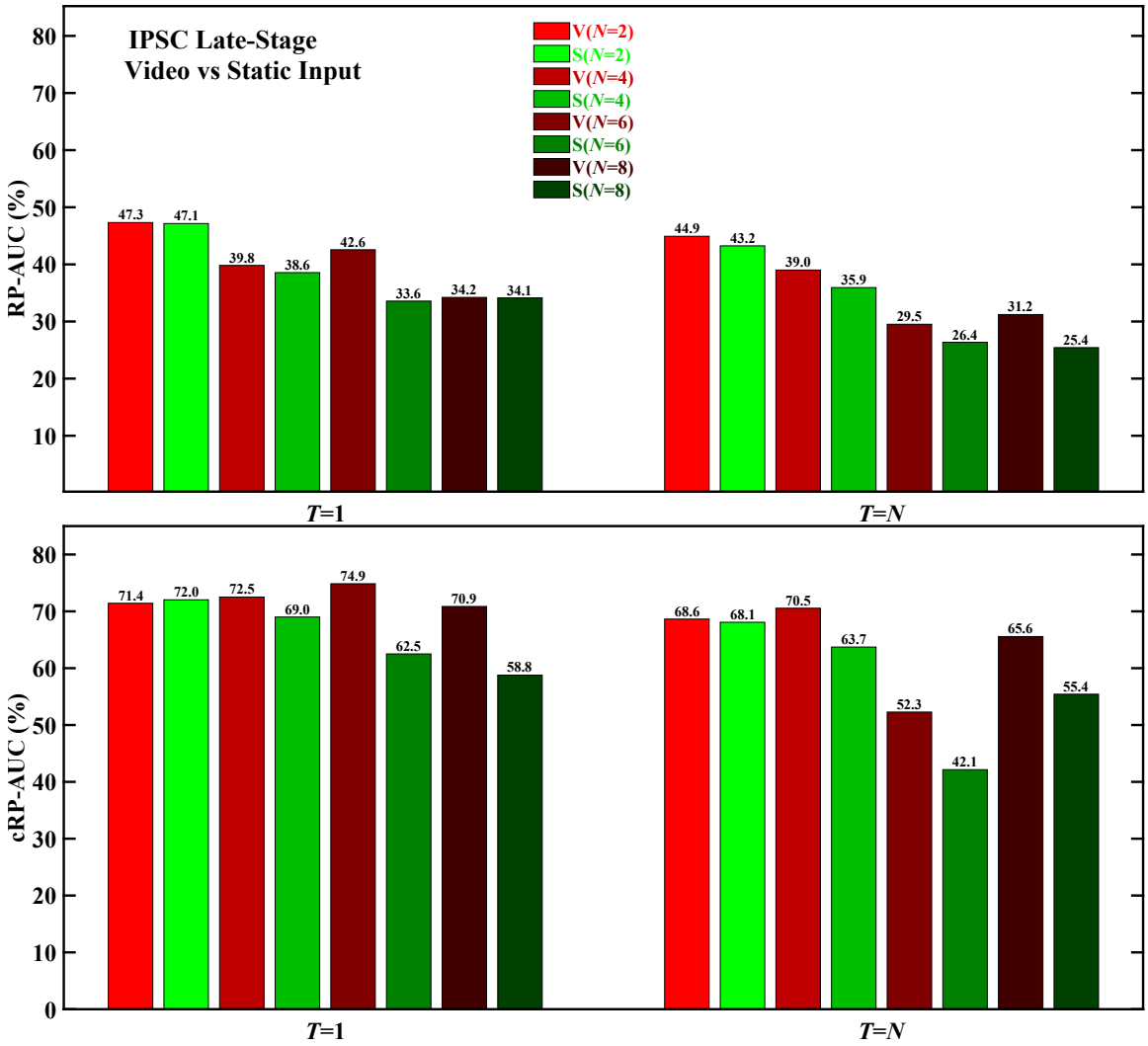

Figure 8.11: Performance impact of replacing N video frames with only the first frame in the sequence as input to P2S-VID models on IPSC late-stage dataset in terms of (top) RP-AUC and (bottom) cRP-AUC. The two cases are respectively shown in shades of red and green and denoted with V and S in the legend. Darker shades of both colors represent higher N.

surprisingly little performance loss over the video models even for N as high as 32, though this loss is greater for $T = N$ than $T = 1$, as expected. Similarly, the performance loss predictably increases with N, except for the odd case of $N = 8$ with $T = 1$ on IPSC (Section 8.5.4). The performance loss is also greater for cRP-AUC than RP-AUC, which makes sense since the first frame is usually sufficient to classify an object but we need the remaining frames to localize it

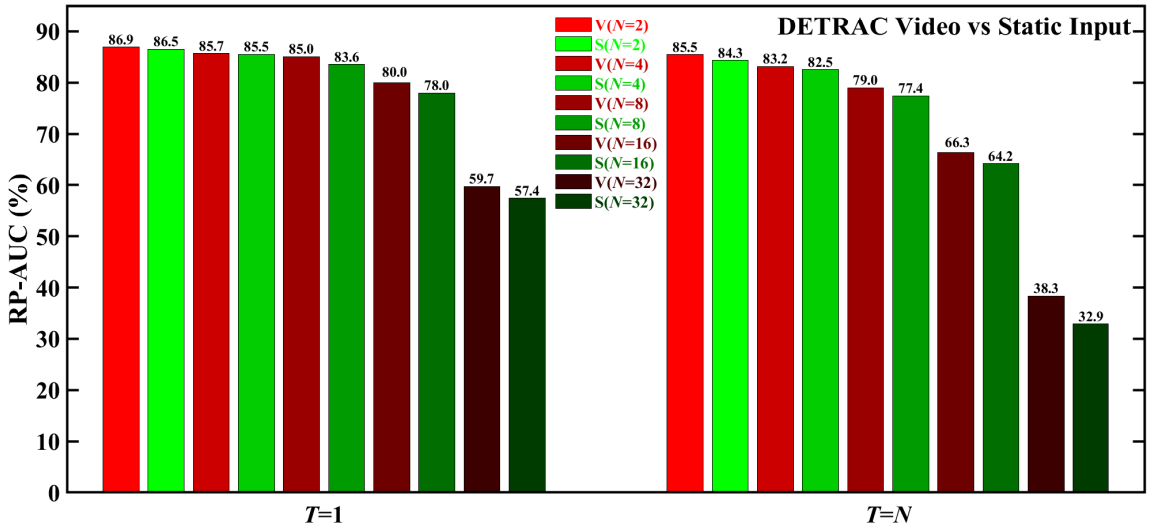

Figure 8.12: Performance impact of replacing N video frames with only the first frame in the sequence as input to P2S-VID models on UA-DETRAC. The two cases are respectively shown in shades of red and green and denoted with V and S in the legend. Darker shades of both colors represent higher N.

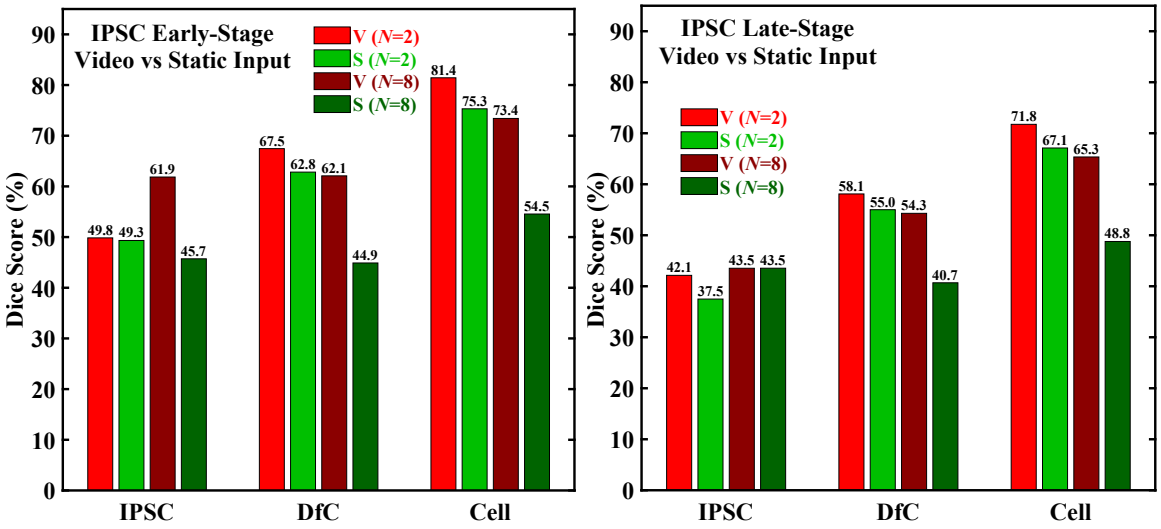

Figure 8.13: Performance impact of replacing N video frames with only the first frame in the sequence as input to P2S-VIDSEG models. The two cases are denoted with V and S in the legend and shown in shades of red and green respectively. Results are shown for both (left) early and (right) late-stage IPSC datasets in terms of Dice score Recall and precision exhibited the same trends as Dice score and have therefore be relegated to Figure F.4.

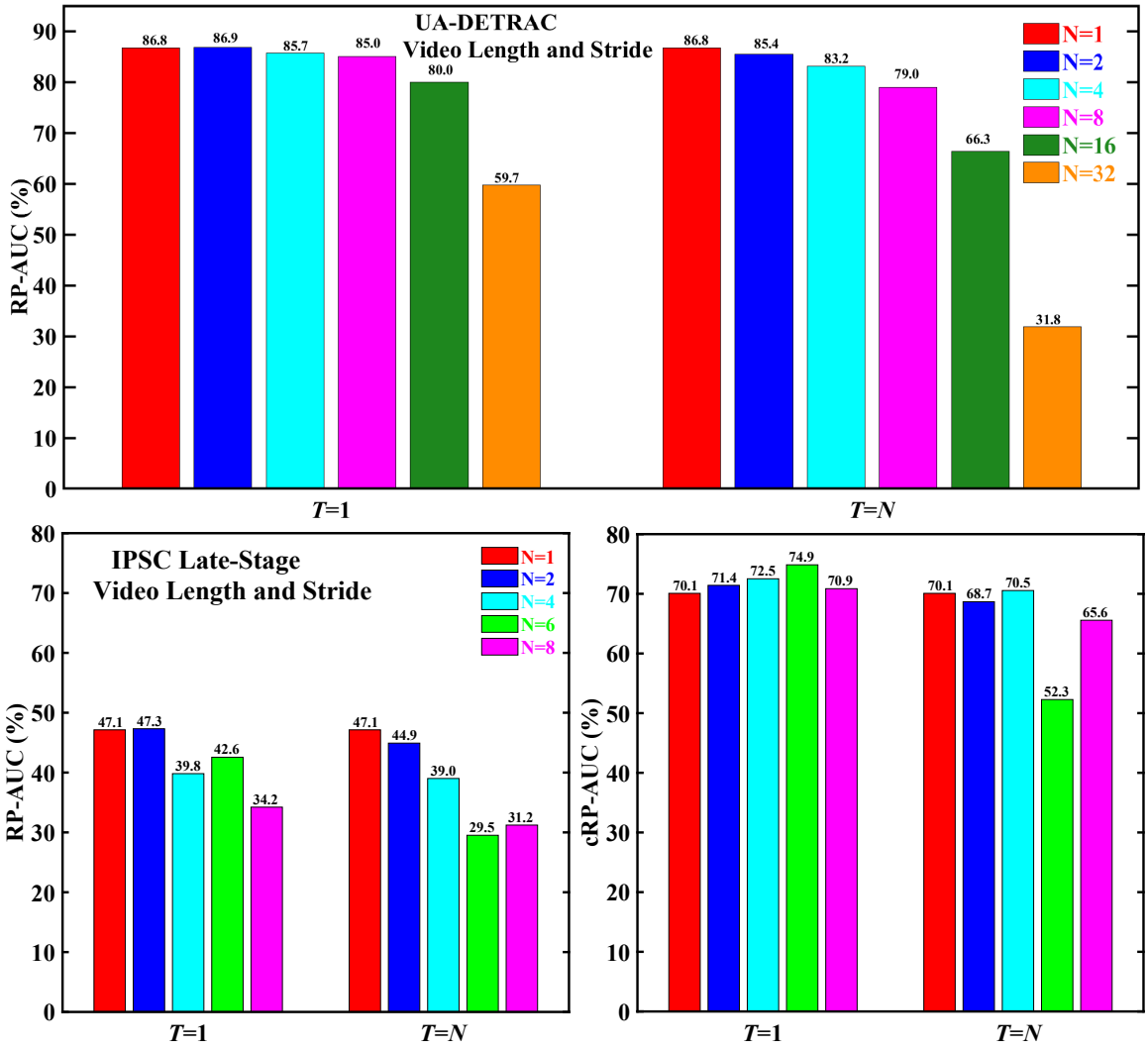

Figure 8.14: Impact of video length N on video detection performance over the (top) UA-DETRAC and (bottom) IPSC late-stage datasets. *N=1* denotes the baseline P2S model and is included for comparison. All the P2S-VID models use the middle-fusion architecture and were trained with frozen backbone and without class token weight equalization.

correctly in those frames. The fact that static input models are able to perform so well even with $T = N$ is explained at least partly by the relatively predictable nature of trajectories in both datasets. Cells do not move a lot, instead mostly remaining in place and changing their shape, and this makes the bounding boxes in the IPSC dataset relatively easy to predict. The vehicles in UA-DETRAC likewise show very similar and nearly linear motion in most sequences, except in a few where

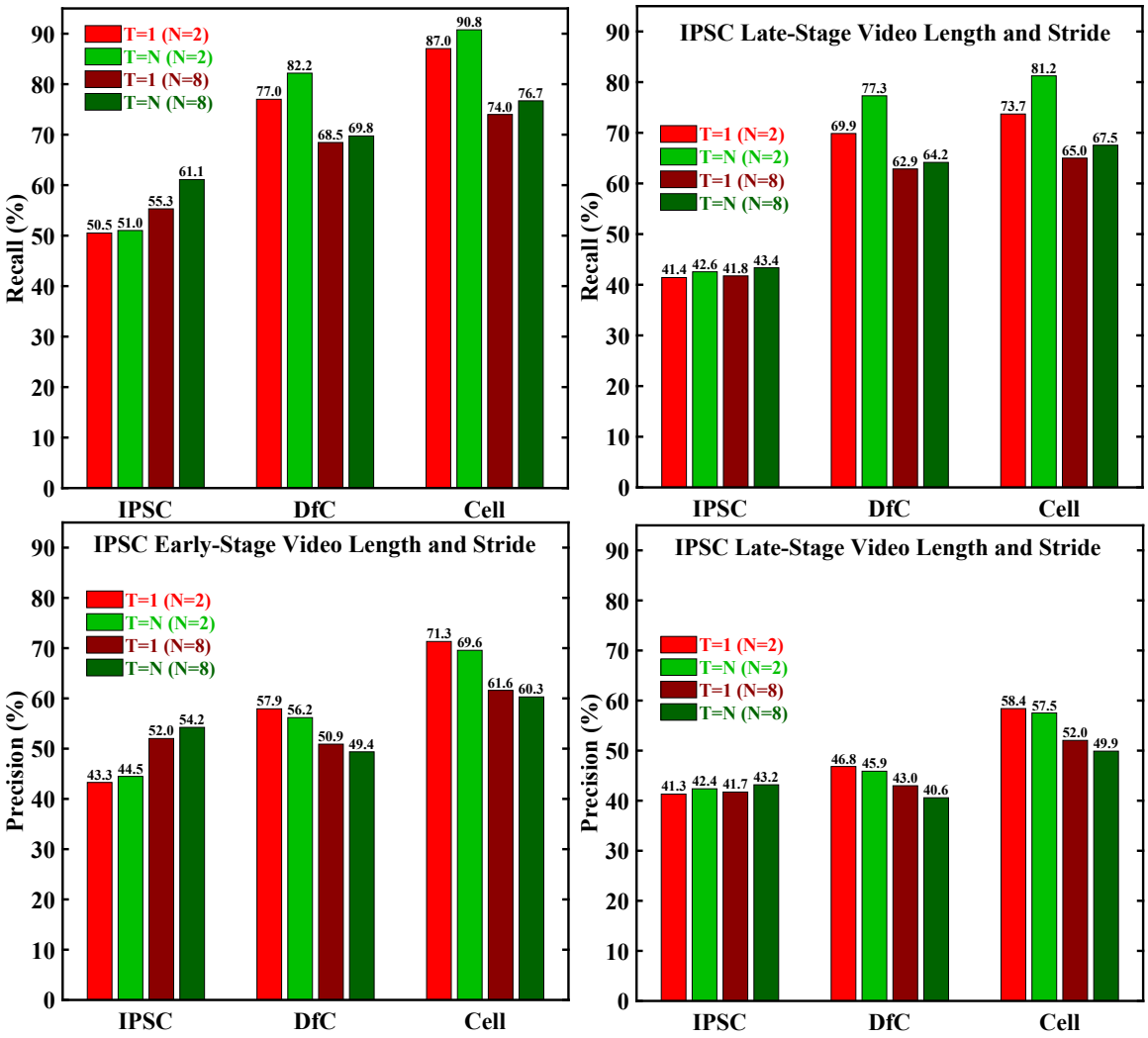

Figure 8.15: Impact of video length N and stride T on P2S-VIDSEG performance over (left) early and (right) late-stage configurations of the IPSC dataset. $T = 1$ and $T = N$ are respectively represented with shades of red and green. Lighter shades of each colour represent $N = 2$ while the darker shades represent $N = 8$. Dice score showed similar patterns as precision and has thus been relegated to Figure F.4.

they remain stopped at traffic lights for extended periods of times and it is in these sequences that the static input models perform the worst relative to the video models. The relatively poor performance of video models is probably also another indicator of the batch size bottleneck on the video models which prevents them from making full use of the video information.

P2S-VIDSEG results for both configurations of IPSC dataset are shown in Figure 8.13. Video input has more consistent and strongly marked performance advantage

over static input here, especially for $N = 8$. In part, this is due to the absence of temporal redundancy here, since $N = T$ is used in all cases. It might also be due to greater integration of the outputs corresponding to different frames within the same shared tokens in video segmentation through TAC (Section 7.3.1), as opposed to frame-specific tokens employed by video detection.

8.5.4 Video Length and Stride

Figure 8.14 summarizes the impact of video length N and stride T on P2S-VID performance over both UA-DETRAC and IPSC late-stage datasets. Neither dataset shows any consistent improvement in performance with N. Quite the contrary, in fact. The only case that shows any signs of steady improvement is IPSC cRP-AUC with $T = 1$, at least as far as $N = 6$. There is, however, fairly consistent improvement in going from $T = N$ to $T = 1$ and the degree of this improvement also increases with N, which is to be expected because of the greater redundancy. IPSC RP-AUC with $N = 8$ is a notable exception to this latter trend and shows hardly any improvement over $T = N$ which is unusual for N so large. For comparison, $N = 6$ shows an improvement of 13% between $T = 1$ and $T = N$ while $N = 8$ shows only 3%. When combined with the fact that the static-input version is able to match the performance of this model (Section 8.5.3.2), I suspect that this might be a buggy model. I still need to investigate this further and possibly retrain this model to be certain.

Figure 8.15 shows the impact of N and T on video segmentation. As mentioned in Section 7.2.2.1, combining redundant outputs from overlapping temporal windows is far less straightforward for segmentation than it is for detection. These plots show the results of the voting strategy that I found to work best, though its overall performance impact is still minimal. Non-redundant output ($T = N$) fares slightly better than redundant output ($T = 1$) in terms of recall over all three classes. $T = N$ is also better in terms of precision for *IPSC* but $T = 1$ has a slight edge over *DfC* and *Cell*. Like all Pix2Seq models, P2S-VIDSEG models have a tendency to overfit to the more numerous class, which in this case corresponds to the background, followed by *DfC*. Combining pixel labels from multiple temporal windows therefore results in some of the pixels, mostly near the cell boundaries, getting misclassified as either background or *DfC*. Misclassification as background explains the drop in recall across the board while misclassification as *DfC* explains the drop in precision for *IPSC* along with concurrent increase for *DfC* and *Cell*.

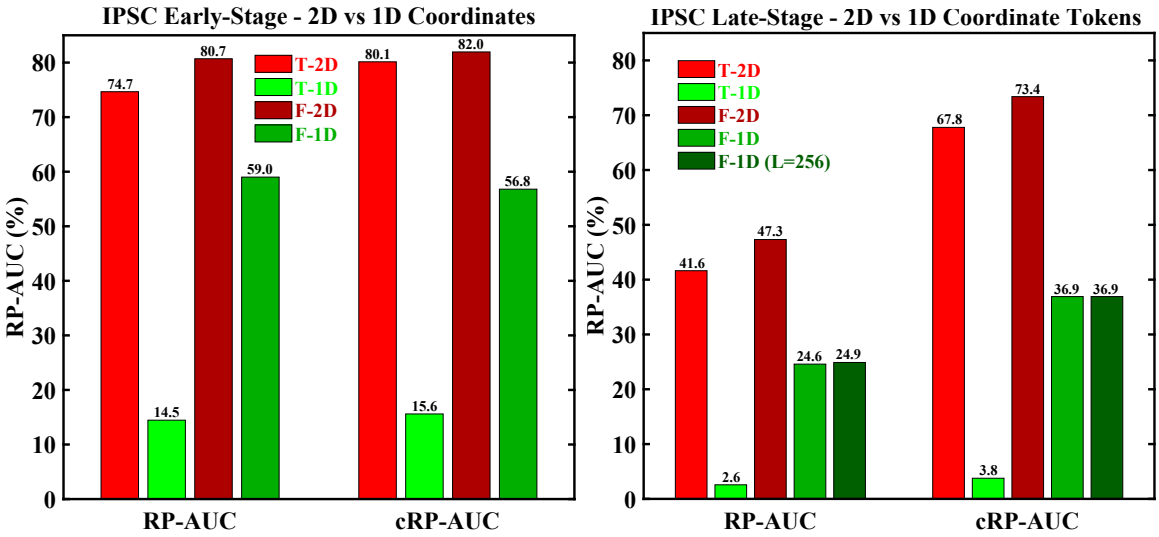

Figure 8.16: Comparing the standard 2D coordinate tokenization of P2S-VID with its 1D variant on the (left) early and (right) late-stage IPSC datasets. 2D and 1D tokenization models are respectively shown in shades of red and green. Models trained with and without frozen backbone are shown respectively in darker and lighter shades and denoted with the prefixes F and T in the legend. All models use $N = 2$.

In terms of video length, $N = 2$ significantly outperforms $N = 8$ for *DfC* and *Cell* but the latter is appreciably better for *IPSC*, significantly more so over the early-stage datset and with $T = N$. This can be partly attributed to the much greater batch size bottleneck on $N = 8$ and partly to my bias in favoring *IPSC* performance over *DfC* (Section 8.4.3.2.1) when selecting the models for inclusion here.

8.5.5 1D Coordinate Tokens

As mentioned in Section 6.1.2.3, 1D coordinate tokens can be used to partially solve the problem of L becoming too large as N increases. I trained a few models to judge the practicability of this approach. As shown in Figure 8.16, it turned out to be not doable, at least on my existing hardware. The 1D model was able to achieve barely half the performance of the standard 2D model on the late-stage dataset, although it fared slightly better on the easier early-stage variant. An interesting finding here was that training the 1D model without the backbone frozen causes a further sharp decline in performance. This is surprising since one would expect that learning a new coordinate tokenization different from the one that was used for pretraining the backbone would benefit from fine-tuning the backbone on the new tokenization, but

that is not the case. Note that the 2D model was trained with the default Pix2Seq vocabulary parameters $H = 2K$ and $V = 3K$ while the 1D version had $H = 160$ and $V = 28K$. Although both 2D and 1D models were trained with the same B, it is possible that this poor performance might be another case of a bottleneck introduced by B since the much higher V in the 1D case probably requires much larger B to work. This is also supported by the much greater performance advantage of frozen-backbone models with 1D tokenization than with 2D tokenization. Freezing the backbone allowed the 1D models to be trained with $B = 64$ while the full-network trained models were restricted to $B = 18$. The performance drop can also be partially attributed to the decrease in localization accuracy due to the drop in H by a factor of more than 10 from $H = 2000$ to $H = 160$. I also trained a 1D model with L reduced by half to 256 to take advantage of the shorter sequences but this turned out to have no impact on the overall performance.

Chapter 9

Conclusions and Future Work

9.1 Conclusions

This thesis has introduced a new way to perform object detection in videos and semantic segmentation in images and videos by modeling the outputs of these tasks as sequences of discrete tokens. I have proposed these new methods as another step in the direction of the more general tokenization of visual recognition tasks that has been happening over the last few years through the paradigm of language modeling. I have presented theoretical arguments for why such tokenization can help to solve the problems consequent upon trying to model the inherently discrete and variable-length outputs that are common in vision tasks with the continuous-valued and fixed-length representations in conventional modeling. I have tested these models on a wide range of real-world problems with in-depth experiments to demonstrate their competitiveness with the state of the art in conventional modeling. Although I have not been able to demonstrate that my methods offer significant and consistent performance advantage over conventional models, I do present strong evidence to suggest that this is not due to any intrinsic weakness in the models themselves. Rather, it is likely to be a consequence of the bottleneck that is imposed upon these models by the small training batch sizes that I have been constrained to use by my limited computational resources. Once these constraints are lifted and the models can be trained to their full potential, I am confident that they will be able to justify their theoretical advantages with practical performance benefits.

9.2 Future Work

I am currently planning to lease a few Tesla A100 80 GB GPUs to train my models on large benchmark datasets with sufficiently high batch sizes, both as a means to validate the above hypothesis and to generate results for peer-reviewed publications. In addition to improving the performance of my existing models with better training, the language modeling paradigm provides immense scope for designing better models through more efficient and clever tokenization schemes. As mentioned in Section 7.1, one such avenue would be to find a way to add object-level information to the RLE tokenization I have proposed without losing the benefits of the small quantum and corresponding robustness to inference noise that it provides. Section 7.3.2 proposed a possible way to achieve this by generating object-level binary RLE tokens (instead of image-level tokens) separated by class tokens to mark the end of each object. Such a representation can encode both instance and semantic masks and therefore perform panoptic segmentation. I have completed implementing this for static segmentation but still need to extend it for video segmentation. Early results are promising, although exhibiting strong signs of batch size bottleneck with higher resolution masks, which I hope to alleviate with the better GPUs.

A further improvement upon this would be to add some information to the tokens to be able to link the object instances from multiple consecutive temporal windows into long trajectories without having to resort to heuristics. This would achieve my original thesis objective of creating a true end-to-end differentiable MOT pipeline. Associating video-objects in this way would make it necessary for us to inject some information from previous forward passes of the network into future forward passes. One way to achieve this might be to construct a hierarchical autoregressive framework where the entire $L \times V$ output of the base-level autoregression (that currently exists) is used to construct a kind of composite token representing a single temporal window. A sequence of these composite tokens from multiple temporal windows then becomes the input for a higher level autoregressive layer that can combine information from multiple forward passes of the base-level to produce a single consolidated sequence of tokens that can represent entire object trajectories spanning tens or even hundreds of forward passes of the base-level. I would imagine that training such a hierarchical architecture that combines gradient information from multiple forward passes would require very large batch sizes and

probably vast amounts of data. We might, however, be able to get away with training the base and high-level autoregressive layers separately by first training the base-level by itself and then freezing it and training only the high-level autoregressive network. We could then alternate between the two stages.

With respect to the Pix2Seq framework itself, there are several implementation tasks that I was unable to complete due to time constraints. One of these is to adapt the VIT backbone for video processing and possibly extend it with a version suited for the newer Swin transformer. VIT requires around 12 times more memory than ResNet-50 so this had perhaps best be done once sufficient GPU memory is available to train such an architecture with large enough batch sizes. Another and simpler task is to implement the mask-to-RLE conversion process in Tensorflow so it can be integrated into the training pipeline to generate the RLE tokens online during training. This conversion is currently implemented in Numpy which requires that the RLE sequences be generated offline and this in turn means that all the augmented images also have to be generated before the training starts. This works for small datasets like IPSC and ARIS but would be impracticable for larger benchmark datasets like COCO. Online RLE generation would provide several additional benefits including deploying more sophisticated data augmentation techniques, extending class-token weight equalization to segmentation, and randomizing the order of runs in the RLE sequence similar to the randomized order of objects in the object detector. All of these should allow us to train more robust segmentation models. A related extension would be to modify instance segmentation component of the the multi-task version of Pix2Seq [49] to represent the segmentation masks using RLE tokens instead of polygons. Similarly, we could design a new multi-task model to perform both video detection and video segmentation in response to different prompts.

Bibliography

[1] T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, "Focal Loss for Dense Object Detection," in *ICCV*, Oct 2017, pp. 2999–3007.

[2] H. Kalke and M. Loewen, "Predicting Surface Ice Concentration using Machine Learning," in *19th Workshop on the Hydraulics of Ice Covered Rivers*. Whitehorse, Yukon, Canada,: CGU HS Committee on River Ice Processes and the Environment, July 9-12 2017.

[3] ——, "Support Vector Machine Learning Applied to Digital Images of River Ice Conditions," submitted for review to Cold Regions Science and Technology, September 2017.

[4] H. Kalke, "Sediment Transport by Released Anchor Ice: Field Measurements and Digital Image Processing," Master's thesis, University of Alberta, 2017.

[5] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation," *arXiv:1802.02611*, 2018.

[6] F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions," in *CVPR*. IEEE Computer Society, 2017, pp. 1800–1807.

[7] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *MICCAI (3)*, ser. Lecture Notes in Computer Science, vol. 9351. Springer, 2015, pp. 234–241.

[8] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *CoRR*, vol. abs/1409.1556, 2014.

[9] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 12, pp. 2481–2495, 2017.

[10] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," in *CVPR*, 2017, pp. 2261–2269.

[11] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, June 2017.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *CVPR*, pp. 770–778, 2016.

[13] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning," in *AAAI*, 2017.

[14] B. Zoph and Q. V. Le, "Neural Architecture Search with Reinforcement Learning," in *ICLR*, 2017.

[15] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning Transferable Architectures for Scalable Image Recognition," in *CVPR*, 2018, pp. 8697–8710.

[16] J. Dai, Y. Li, K. He, and J. Sun, "R-FCN: Object Detection via Region-based Fully Convolutional Networks," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 379–387.

[17] T.-Y. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie, "Feature Pyramid Networks for Object Detection," *CVPR*, pp. 936–944, 2017.

[18] T.-Y. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, "Focal Loss for Dense Object Detection," *ICCV*, pp. 2999–3007, 2017.

[19] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single Shot MultiBox Detector," in *ECCV*, 2016.

[20] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," in *CVPR*, 2016, pp. 2818–2826.

[21] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *CVPR*, 2018, pp. 4510–4520.

[22] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," *CoRR*, vol. abs/1804.02767, 2018. [Online]. Available: http://arxiv.org/abs/1804.02767

[23] H. Zhang, X. Shao, Y. Peng, Y. Teng, K. M. Saravanan, H. Zhang, Y. Wei, and H. Li, "A novel machine learning based approach for iPS progenitor cell identification," *PLoS Computational Biology*, vol. 15, 2019.

[24] Z. Liu, H. Mao, C. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A ConvNet for the 2020s," *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11 966–11 976, 2022.

[25] Z. Cai and N. Vasconcelos, "Cascade R-CNN: High Quality Object Detection and Instance Segmentation," *TPAMI*, vol. 43, no. 5, pp. 1483–1498, 2021.

[26] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows," *ICCV*, pp. 9992–10 002, 2021.

[27] T. Xiao, Y. Liu, B. Zhou, Y. Jiang, and J. Sun, "Unified perceptual parsing for scene understanding," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 418–434.

[28] J. Wu, Q. Liu, Y. Jiang, S. Bai, A. L. Yuille, and X. Bai, "In Defense of Online Models for Video Instance Segmentation," in *ECCV*, 2022.

[29] J. Wu, Y. Jiang, S. Bai, W. Zhang, and X. Bai, "SeqFormer: Sequential Transformer for Video Instance Segmentation," in *ECCV*, 2021.

[30] M. Heo, S. Hwang, S. W. Oh, J.-Y. Lee, and S. J. Kim, "VITA: Video Instance Segmentation via Object Token Association," in *NeurIPS*, 2022.

[31] T. Chen, S. Saxena, L. Li, D. J. Fleet, and G. E. Hinton, "Pix2seq: A Language Modeling Framework for Object Detection," in *Proceedings of the 10th International Conference on Learning Representations (ICLR)*, 2022.

[32] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Advances in neural information processing systems*, vol. 25, 2012.

[33] A. Vaswani, N. M. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is All you Need," *NIPS*, 2017.

[34] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," *ICLR*, 2021.

[35] A. Shrivastava, A. K. Gupta, and R. B. Girshick, "Training Region-Based Object Detectors with Online Hard Example Mining," *CVPR*, pp. 761–769, 2016.

[36] A. Bewley, Z. Ge, L. Ott, F. T. Ramos, and B. Upcroft, "Simple Online and Realtime Tracking," *ICIP*, pp. 3464–3468, 2016.

[37] E. Bochinski, V. Eiselein, and T. Sikora, "High-Speed Tracking-by-Detection without using Image Information," *AVSS*, pp. 1–6, 2017.

[38] R. Girshick, "Fast R-CNN," in *ICCV*, Dec 2015, pp. 1440–1448.

[39] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *TPAMI*, vol. 42, pp. 386–397, 2020.

[40] Z. Cai and N. Vasconcelos, "Cascade R-CNN: High Quality Object Detection and Instance Segmentation," *TPAMI*, vol. 43, pp. 1483–1498, 2021.

[41] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *CVPR*, pp. 779–788, 2015.

[42] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," *CVPR*, pp. 6517–6525, 2017.

[43] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal speed and accuracy of object detection," *ArXiv*, vol. abs/2004.10934, 2020.

[44] G. R. Jocher, A. Stoken, J. Borovec, NanoCode, A. Chaurasia, TaoXie, C. Liu, Abhiram, Laughing, tkianai, yxNONG, A. Hogan, lorenzomammana, AlexWang, J. Hájek, L. Diaconu, Marc, Y. Kwon, Oleg, wanghaoyang, Y. Defretin, A. Lohia, ml ah, B. Milanko, B. Fineran, D. P. Khromov, D. Yiwei, Doug, Durgesh, and F. Ingham, "ultralytics/yolov5: v5.0 -

YOLOv5-P6 1280 models, AWS, Supervise.ly and YouTube integrations," 2021. [Online]. Available: https://github.com/ultralytics/yolov5

[45] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors," *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7464–7475, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:250311206

[46] G. R. Jocher, A. Stoken, J. Borovec, NanoCode, A. Chaurasia, TaoXie, C. Liu, Abhiram, Laughing, tkianai, yxNONG, A. Hogan, lorenzomammana, AlexWang, J. Hájek, L. Diaconu, Marc, Y. Kwon, Oleg, wanghaoyang, Y. Defretin, A. Lohia, ml ah, B. Milanko, B. Fineran, D. P. Khromov, D. Yiwei, Doug, Durgesh, and F. Ingham, "Ultralytics YOLOv8: A computer vision model architecture for detection, classification, segmentation, and more," 2023. [Online]. Available: https://yolov8.com/

[47] C.-Y. Wang, I.-H. Yeh, and H. Liao, "YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information," *ArXiv*, vol. abs/2402.13616, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:267770251

[48] J. R. Terven, D.-M. Córdova-Esparza, and J.-A. Romero-González, "A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS," *Mach. Learn. Knowl. Extr.*, vol. 5, pp. 1680–1716, 2023.

[49] T. Chen, S. Saxena, L. Li, T.-Y. Lin, D. J. Fleet, and G. Hinton, "A Unified Sequence Interface for Vision Tasks," in *Proceedings of the 36th Annual Conference on Neural Information Processing Systems (NIPS)*, 2022.

[50] T. Chen, R. Zhang, and G. Hinton, "Analog bits: Generating discrete data using diffusion models with self-conditioning," in *Proceedings of the 11th International Conference on Learning Representations (ICLR)*, 2023.

[51] T. Chen, L. Li, S. Saxena, G. E. Hinton, and D. J. Fleet, "A Generalist Framework for Panoptic Segmentation of Images and Videos," *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 909–919, 2023.

[52] X. Chen, H. Peng, D. Wang, H. Lu, and H. Hu, "SeqTrack: Sequence to Sequence Learning for Visual Object Tracking," *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14 572–14 581, 2023.

[53] Y. Zheng, B. Zhong, Q. Liang, G. Li, R. Ji, and X. Li, "Toward Unified Token Learning for Vision-Language Tracking," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 34, pp. 2125–2135, 2023.

[54] Y. Xue, J. Mao, M. Niu, H. Xu, M. B. Mi, W. Zhang, X. Wang, and X. Wang, "Point2Seq: Detecting 3D Objects as Sequences," *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8511–8520, 2022.

[55] Z. Chen, Y. Zhu, Z. Li, F. Yang, W. Li, H. Wang, C. Zhao, L. Wu, R. Zhao, J. Wang *et al.*, "Obj2seq: Formatting objects as sequences with class prompt for visual tasks," *Advances in Neural Information Processing Systems*, vol. 35, pp. 2494–2506, 2022.

[56] Z. Yang, Z. Gan, J. Wang, X. Hu, F. Ahmed, Z. Liu, Y. Lu, and L. Wang, "Unitab: Unifying text and box outputs for grounded vision-language modeling," in *European Conference on Computer Vision*. Springer, 2022, pp. 521–539.

[57] J. Liu, H. Ding, Z. Cai, Y. Zhang, R. K. Satzoda, V. Mahadevan, and R. Manmatha, "Polyformer: Referring image segmentation as sequential polygon generation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 18 653–18 663.

[58] G. Țucudean, M. Bucos, B. Drăgulescu, and C. D. Căleanu, "Natural Language Processing with Transformers: A Review," *PeerJ Computer Science*, vol. 10, 2024.

[59] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, Y. Du, C. Yang, Y. Chen, Z. Chen, J. Jiang, R. Ren, Y. Li, X. Tang, Z. Liu, P. Liu, J. Nie, and J. rong Wen, "A Survey of Large Language Models," *ArXiv*, vol. abs/2303.18223, 2023.

[60] S. Minaee, T. Mikolov, N. Nikzad, M. A. Chenaghlu, R. Socher, X. Amatriain, and J. Gao, "Large Language Models: A Survey," *ArXiv*, vol. abs/2402.06196, 2024.

[61] A. Radford and K. Narasimhan, "Improving Language Understanding by Generative Pre-Training," 2018.

[62] P. P. Ray, "ChatGPT: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope," *Internet of Things and Cyber-Physical Systems*, vol. 3, pp. 121–154, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S266734522300024X

[63] S. d'Ascoli, H. Touvron, M. L. Leavitt, A. S. Morcos, G. Biroli, and L. Sagun, "ConViT: Improving vision transformers with soft convolutional inductive biases," in *International conference on machine learning*. PMLR, 2021, pp. 2286–2296.

[64] J. Maurício, I. Domingues, and J. Bernardino, "Comparing Vision Transformers and Convolutional Neural Networks for Image Classification: A Literature Review," *Applied Sciences*, vol. 13, no. 9, 2023. [Online]. Available: https://www.mdpi.com/2076-3417/13/9/5521

[65] W. Luo, X. Zhao, and T. Kim, "Multiple Object Tracking: A Review," *ArXiv*, vol. abs/1409.7618v4, 2017.

[66] Y. Xiang, A. Alahi, and S. Savarese, "Learning to Track: Online Multi-object Tracking by Decision Making," *ICCV*, pp. 4705–4713, 2015.

[67] R. Krishna, K. Hata, F. Ren, L. Fei-Fei, and J. Carlos Niebles, "Dense-captioning events in videos," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 706–715.

[68] L. Zhou, Y. Zhou, J. J. Corso, R. Socher, and C. Xiong, "End-to-End Dense Video Captioning with Masked Transformer," *CVPR*, pp. 8739–8748, 2018.

[69] T. Wang, R. Zhang, Z. Lu, F. Zheng, R. Cheng, and P. Luo, "End-to-End Dense Video Captioning with Parallel Decoding," *ICCV*, 2021.

[70] A. Singh, H. Kalke, M. R. Loewen, and N. Ray, "River Ice Segmentation With Deep Learning," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 58, pp. 7570–7579, 2020.

[71] A. Singh, M. Pietrasik, G. Natha, N. Ghouaiel, K. Brizel, and N. Ray, "Animal Detection in Man-made Environments," *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 1427–1438, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:204900826

[72] A. Singh, I. Jasra, O. Mouhammed, N. Dadheech, N. Ray, and J. Shapiro, "Towards Early Prediction of Human iPSC Reprogramming Success," *Machine Learning for Biomedical Imaging*, vol. 2, pp. 390–407, 2023. [Online]. Available: https://melba-journal.org/2023:014

[73] T. Chen, "Pix2Seq Codebase: Multi-tasks with generative modeling (autoregressive and diffusion)," online: https://github.com/google-research/pix2seq.

[74] A. Yang, A. Nagrani, P. H. Seo, A. Miech, J. Pont-Tuset, I. Laptev, J. Sivic, and C. Schmid, "Vid2Seq: Large-Scale Pretraining of a Visual Language Model for Dense Video Captioning," *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10 714–10 726, 2023.

[75] A. Jabri, D. J. Fleet, and T. Chen, "Scalable Adaptive Computation for Iterative Generation," *arXiv preprint arXiv:2212.11972*, 2022.

[76] T. Chen, "On the Importance of Noise Scheduling for Diffusion Models," *arXiv preprint arXiv:2301.10972*, 2023.

[77] T. Chen and L. Li, "FIT: Far-reaching Interleaved Transformers," *arXiv preprint arXiv:2305.12689*, 2023.

[78] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-End Object Detection with Transformers," *ECCV*, 2020.

[79] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," in *CVPR*, 2014, pp. 580–587. [Online]. Available: https://doi.org/10.1109/CVPR.2014.81

[80] L. Castrejón, K. Kundu, R. Urtasun, and S. Fidler, "Annotating Object Instances with a Polygon-RNN," *CVPR*, pp. 4485–4493, 2017.

[81] D. Acuna, H. Ling, A. Kar, and S. Fidler, "Efficient Interactive Annotation of Segmentation Datasets with Polygon-RNN++," *CVPR*, pp. 859–868, 2018.

[82] R. Zellers, J. Lu, X. Lu, Y. Yu, Y. Zhao, M. Salehi, A. Kusupati, J. Hessel, A. Farhadi, and Y. Choi, "Merlot reserve: Neural script knowledge through vision and language and sound," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 16 375–16 387.

[83] J. Ho, A. Jain, and P. Abbeel, "Denoising Diffusion Probabilistic Models," *Advances in neural information processing systems*, vol. 33, pp. 6840–6851, 2020.

[84] A. Kolesnikov, A. Susano Pinto, L. Beyer, X. Zhai, J. Harmsen, and N. Houlsby, "UViM: A unified modeling approach for vision with learned guiding codes," *Advances in Neural Information Processing Systems*, vol. 35, pp. 26 295–26 308, 2022.

[85] A. Vaswani, N. M. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is All you Need," *NIPS*, 2017.

[86] A. Van Den Oord, O. Vinyals *et al.*, "Neural Discrete Representation Learning," *Advances in neural information processing systems*, vol. 30, 2017.

[87] Y. Linde, A. Buzo, and R. Gray, "An algorithm for vector quantizer design," *IEEE Transactions on communications*, vol. 28, no. 1, pp. 84–95, 2003.

[88] J. Ning, C. Li, Z. Zhang, C. Wang, Z. Geng, Q. Dai, K. He, and H. Hu, "All in tokens: Unifying output space of visual tasks via soft token," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 19 900–19 910.

[89] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common Objects in Context," in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds. Cham: Springer International Publishing, 2014, pp. 740–755.

[90] H. Fan, H. Bai, L. Lin, F. Yang, P. Chu, G. Deng, S. Yu, Harshit, M. Huang, J. Liu *et al.*, "Lasot: A high-quality large-scale single object tracking benchmark," *International Journal of Computer Vision*, vol. 129, pp. 439–461, 2021.

[91] L. Huang, X. Zhao, and K. Huang, "Got-10k: A large high-diversity benchmark for generic object tracking in the wild," *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 5, pp. 1562–1577, 2019.

[92] M. Muller, A. Bibi, S. Giancola, S. Alsubaihi, and B. Ghanem, "Trackingnet: A large-scale dataset and benchmark for object tracking in the wild," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 300–317.

[93] L. Yinhan, O. Myle, G. Naman, D. Jingfei, J. Mandar, C. Danqi, L. Omer, and L. Mike, "RoBERTa: A robustly optimized BERT pretraining approach," *arXiv preprint arXiv:1907.11692*, pp. 1–13, 2019.

[94] B. Ye, H. Chang, B. Ma, S. Shan, and X. Chen, "Joint feature learning and relation modeling for tracking: A one-stream framework," in *European Conference on Computer Vision*. Springer, 2022, pp. 341–357.

[95] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, "Spatial Transformer Networks," in *NIPS*, 2015.

[96] C. Zhu, Y. Zhou, Y. Shen, G. Luo, X. Pan, M. Lin, C. Chen, L. Cao, X. Sun, and R. Ji, "SeqTR: A simple yet universal network for visual grounding," in *European Conference on Computer Vision*. Springer, 2022, pp. 598–615.

[97] S. H. Rezatofighi, V. K. Bg, A. Milan, E. Abbasnejad, A. Dick, and I. Reid, "Deepsetnet: Predicting sets with deep neural networks," in *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2017, pp. 5257–5266.

[98] L. Pineda, A. Salvador, M. Drozdzal, and A. Romero, "Elucidating Image-to-Set Prediction: An Analysis of Models, Losses and Datasets," *arXiv preprint arXiv:1904.05709*, 2019.

[99] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, and J. Dai, "Deformable DETR: Deformable Transformers for End-to-End Object Detection," *ICLR*, 2021.

[100] T. Yin, X. Zhou, and P. Krahenbuhl, "Center-based 3d object detection and tracking," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 11 784–11 793.

[101] Y. Yan, Y. Mao, and B. Li, "Second: Sparsely embedded convolutional detection," *Sensors*, vol. 18, no. 10, p. 3337, 2018.

[102] J. Mao, M. Niu, C. Jiang, H. Liang, J. Chen, X. Liang, Y. Li, C. Ye, W. Zhang, Z. Li *et al.*, "One million scenes for autonomous driving: Once dataset," *arXiv preprint arXiv:2106.11037*, 2021.

[103] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine *et al.*, "Scalability in perception for autonomous driving: Waymo open dataset," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 2446–2454.

[104] S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-maron, M. Giménez, Y. Sulsky, J. Kay, J. T. Springenberg, T. Eccles, J. Bruce, A. Razavi, A. Edwards, N. Heess, Y. Chen, R. Hadsell, O. Vinyals, M. Bordbar, and N. de Freitas, "A Generalist Agent," *Transactions on Machine Learning Research*, 2022.

[105] T. Kudo and J. Richardson, "SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, E. Blanco and W. Lu, Eds. Brussels, Belgium: Association for Computational Linguistics, Nov. 2018, pp. 66–71. [Online]. Available: https://aclanthology.org/D18-2012/

[106] P. Wang, A. Yang, R. Men, J. Lin, S. Bai, Z. Li, J. Ma, C. Zhou, J. Zhou, and H. Yang, "OFA: Unifying architectures, tasks, and modalities through a simple sequence-to-sequence learning framework," in *International conference on machine learning*. PMLR, 2022, pp. 23 318–23 340.

[107] J. Lu, C. Clark, R. Zellers, R. Mottaghi, and A. Kembhavi, "Unified-IO: A unified model for vision, language, and multi-modal tasks," in *The Eleventh International Conference on Learning Representations*, 2022.

[108] S. F. Daly, "Report on frazil ice," International Association for Hydraulic Research Working Group on Thermal Regimes, Hanover, New Hampshire, Tech. Rep., August 1994.

[109] E. Kempema and R. Ettema, "Anchor Ice Rafting: Observations from the Laramie River," *River Research and Applications*, vol. 27, pp. 1126 – 1135, 11 2011.

[110] G. Tsang, *Frazil and Anchor Ice: A Monograph*. Ottawa, ON: NRC Subcommittee on Hydraulics of Ice Covered Rivers, 1982.

[111] D. J. Kerr, H. T. Shen, and S. F. Daly, "Evolution and hydraulic resistance of anchor ice on gravel bed," *Cold Regions Science and Technology*, vol. 35, no. 2, pp. 101 – 114, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0165232X02000435

[112] S. Ansari, C. Rennie, O. Seidou, J. Malenchak, and S. Zare, "Automated monitoring of river ice processes using shore-based imagery," *Cold Regions Science and Technology*, vol. 142, pp. 1 – 16, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0165232X16303056

[113] L. Zhang, L. Lu, I. Nogues, R. M. Summers, S. Liu, and J. Yao, "DeepPap: Deep Convolutional Networks for Cervical Cell Classification," *IEEE Journal of Biomedical and Health Informatics*, vol. 21, no. 6, pp. 1633–1643, Nov 2017.

[114] A. Shpilman, D. Boikiy, M. Polyakova, D. Kudenko, A. Burakov, and E. Nadezhdina, "Deep Learning of Cell Classification Using Microscope Images of Intracellular Microtubule Networks," in *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Dec 2017, pp. 1–6.

[115] H. Lei, T. Han, W. Huang, J. Y. Kuo, Z. Yu, X. He, and B. Lei, "Cross-Modal Transfer Learning for HEp-2 Cell Classification Based on Deep Residual Network," in *IEEE International Symposium on Multimedia (ISM)*, Dec 2017, pp. 465–468.

[116] Y. Li and L. Shen, "A Deep Residual Inception Network for HEp-2 Cell Classification," in *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*. Springer International Publishing, 2017, pp. 12–20.

[117] A. Faktor and M. Irani, "Video Segmentation by Non-Local Consensus Voting," in *BMVC*, 2014.

[118] S. Caelles, K.-K. Maninis, J. Pont-Tuset, L. Leal-Taixé, D. Cremers, and L. Van Gool, "One-Shot Video Object Segmentation," in *CVPR*, 2017.

[119] M. Grundmann, V. Kwatra, M. Han, and I. Essa, "Efficient Hierarchical Graph-Based Video Segmentation," in *CVPR*, 2010.

[120] S. Jain, B. Xiong, and K. Grauman, "FusionSeg: Learning to combine motion and appearance for fully automatic segmention of generic objects in videos," *CVPR*, 2017.

[121] D. Pathak, R. Girshick, P. Dollár, T. Darrell, and B. Hariharan, "Learning Features by Watching Objects Move," in *CVPR*, 2017.

[122] Y.-H. Tsai, M.-H. Yang, and M. J. Black, "Video Segmentation via Object Flow," *CVPR*, pp. 3899–3908, 2016.

[123] J. Chi and H.-c. Kim, "Prediction of Arctic Sea Ice Concentration Using a Fully Data Driven Deep Neural Network," *Remote Sensing*, vol. 9, no. 12, 2017. [Online]. Available: http://www.mdpi.com/2072-4292/9/12/1305

[124] A. Singh, H. Kalke, M. Loewen, and N. Ray, "Alberta River Ice Segmentation Dataset," 2019. [Online]. Available: https://dx.doi.org/10.21227/ebax-1h44

[125] mrgloom, "Awesome Semantic Segmentation," Github, 2017, https://github.com/mrgloom/awesome-semantic-segmentation.

[126] Dr.Tang, "Semantic-Segmentation," Github, 2017, https://github.com/tangzhenyu/SemanticSegmentation_DL.

[127] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[128] D. Gupta, "Image Segmentation Keras : Implementation of Segnet, FCN, UNet and other models in Keras," Github, 2017, https://github.com/divamgupta/image-segmentation-keras.

[129] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *CVPR*. IEEE Computer Society, 2015, pp. 3431–3440.

[130] E. Shelhamer, J. Long, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 4, pp. 640–651, 2017.

[131] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 4, pp. 834–848, 2018.

[132] L.-C. Chen, Y. Zhu, and G. Papandreou, "DeepLab: Deep Labelling for Semantic Image Segmentation," Github, 2017, hhttps://github.com/tensorflow/models/tree/master/research/deeplab.

[133] F. Yu and V. Koltun, "Multi-Scale Context Aggregation by Dilated Convolutions," in *ICLR*, 11 2016.

[134] C. Liu, L.-C. Chen, F. Schroff, H. Adam, W. Hua, A. Yuille, and L. Fei-Fei, "Auto-DeepLab: Hierarchical Neural Architecture Search for Semantic Image Segmentation," in *CVPR*, 2019.

[135] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid Scene Parsing Network," in *CVPR*, 2017.

[136] A. Singh, "River Ice Segmentation with Deep Learning," online: https://github.com/abhineet123/river_ice_segmentation, April 2019.

[137] "Alberta River Ice Segmentation Dataset," http://dx.doi.org/10.21227/ebax-1h44, 2019. [Online]. Available: http://dx.doi.org/10.21227/ebax-1h44

[138] M. Ker?ner, "Image Segmentation Evaluation," Github, 2017, https://github.com/martinkersner/py_img_seg_eval.

[139] J. yves Bouguet, "Pyramidal implementation of the Lucas Kanade feature tracker," *Intel Corporation, Microprocessor Research Labs*, 2000.

[140] G. Farnebäck, "Two-Frame Motion Estimation Based on Polynomial Expansion," in *Image Analysis*, J. Bigun and T. Gustavsson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 363–370.

[141] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks," in *CVPR*, 2017. [Online]. Available: http://lmb.informatik.uni-freiburg.de//Publications/2017/IMKDB17

[142] C. Szegedy, A. Toshev, and D. Erhan, "Deep Neural Networks for Object Detection," in *NIPS*, 2013.

[143] J. Huang, V. Rathod, C. Sun, M. Zhu, A. K. Balan, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. P. Murphy, "Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors," *CVPR*, pp. 3296–3297, 2017.

[144] L. Liu, W. Ouyang, X. Wang, P. W. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, "Deep Learning for Generic Object Detection: A Survey," *CoRR*, vol. abs/1809.02165, 2018. [Online]. Available: http://arxiv.org/abs/1809.02165

[145] A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, T. Duerig, and V. Ferrari, "The Open Images Dataset V4: Unified image classification, object detection, and visual relationship detection at scale," *arXiv:1811.00982*, 2018.

[146] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets Robotics: The KITTI Dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[147] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal Visual Object Classes (VOC) Challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, Jun. 2010.

[148] S. J. Pan and Q. Yang, "A Survey on Transfer Learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, pp. 1345–1359, 2010.

[149] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *NIPS*, 2014.

[150] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR*, 2009.

[151] "Labeled Information Library of Alexandria: Biology and Conservation," online: http://lila.science/datasets1, August 2019.

[152] P. Gray, "Awesome Deep Ecology," online: https://github.com/patrickcgray/awesome-deep-ecology, August 2019.

[153] B. Kellenberger, D. Marcos, and D. Tuia, "Detecting mammals in UAV images: Best practices to address a substantially imbalanced dataset with deep learning," *Remote Sensing of Environment*, vol. 216, pp. 139 – 153, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0034425718303067

[154] B. Kellenberger, D. Marcos Gonzalez, and D. Tuia, "Best practices to train deep models on imbalanced datasets - a case study on animal detection in aerial imagery," in *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 01 2018.

[155] A. Khosla, N. Jayadevaprakash, B. Yao, and L. Fei-Fei, "Novel Dataset for Fine-Grained Image Categorization," in *First Workshop on Fine-Grained Visual Categorization, CVPR*, Colorado Springs, CO, June 2011.

[156] O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. V. Jawahar, "Cats and Dogs," in *CVPR*, 2012.

[157] S. Li, J. Li, W. Lin, and H. Tang, "Amur Tiger Re-identification in the Wild," *CoRR*, vol. abs/1906.05586, 2019. [Online]. Available: http://arxiv.org/abs/1906.05586

[158] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona, "Caltech-UCSD Birds 200," California Institute of Technology, Tech. Rep. CNS-TR-2010-001, 2010.

[159] G. V. Horn, "NABirds Dataset," online:http://dl.allaboutbirds.org/nabirds1, August 2019.

[160] A. Swanson, M. Kosmala, C. Lintott, R. Simpson, A. Smith, and C. Packer, "Snapshot Serengeti, high-frequency annotated camera trap images of 40 mammalian species in an African savanna," *Scientific Data*, vol. 2, pp. 150 026–, Jun. 2015. [Online]. Available: https://doi.org/10.1038/sdata.2015.26

[161] J. Parham, C. Stewart, J. Crall, D. Rubenstein, J. Holmberg, and T. Berger-Wolf, "An Animal Detection Pipeline for Identification," in *WACV*, 2018.

[162] S. Beery, G. Van Horn, and P. Perona, "Recognition in Terra Incognita," in *Computer Vision – ECCV 2018*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds. Cham: Springer International Publishing, 2018, pp. 472–489.

[163] S. Beery, D. Morris, and P. Perona, "The iWildCam 2019 Challenge Dataset," *arXiv preprint arXiv:1907.07617*, 2019.

[164] G. V. Horn, O. M. Aodha, Y. Song, Y. Cui, C. Sun, A. Shepard, H. Adam, P. Perona, and S. Belongie, "The iNaturalist Species Classification and Detection Dataset," in *CVPR*, June 2018, pp. 8769–8778.

[165] M. A. Tabak, Mohammad, S. Norouzzadeh, and D. W. W. et. al, "Machine learning to classify animal species in camera trap images: Applications in ecology," *Methods in Ecology and Evolution*, vol. 10, no. 4, pp. 585–590, 26 November 2018.

[166] M. Willi, R. T. Pitman, and A. W. C. et. al, "Identifying animal species in camera trap images using deep learning and citizen science," *Methods in Ecology and Evolution*, 2018.

[167] "Deep Learning for Animal Detection in Man-made Environments," online: https://github.com/abhineet123/animal_detection, January 2020.

[168] B. Kellenberger, D. Marcos, S. Lobry, and D. Tuia, "Half a Percent of Labels is Enough: Efficient Animal Detection in UAV Imagery using Deep CNNs and Active Learning," *CoRR*, vol. abs/1907.07319, 2019. [Online]. Available: http://arxiv.org/abs/1907.07319

[169] D. Tuia, M. Volpi, L. Copa, M. Kanevski, and J. Munoz-Mari, "A Survey of Active Learning Algorithms for Supervised Remote Sensing Image Classification," *IEEE Journal of Selected Topics in Signal Processing*, vol. 5, no. 3, pp. 606–617, June 2011.

[170] M. Cuturi, "Sinkhorn Distances: Lightspeed Computation of Optimal Transport," in *NIPS*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 2292–2300. [Online]. Available: http://papers.nips.cc/paper/4927-sinkhorn-distances-lightspeed-computation-of-optimal-transport.pdf

[171] B. Kellenberger, M. Volpi, and D. Tuia, "Fast animal detection in UAV images using convolutional neural networks," in *IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, July 2017, pp. 866–869.

[172] N. Rey, M. Volpi, S. Joost, and D. Tuia, "Detecting animals in African Savanna with UAVs and the crowds," *ArXiv*, vol. abs/1709.01722, 2017.

[173] S. Schneider, G. W. Taylor, S. Linquist, and S. C. Kremer, "Past, present and future approaches using computer vision for animal reidentification from camera trap data," *Methods in Ecology and Evolution*, vol. 10, no. 4, pp. 461–470, April 2018.

[174] S. Beery, G. Van Horn, and P. Perona, "Recognition in Terra Incognita," in *Computer Vision – ECCV 2018*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds. Cham: Springer International Publishing, 2018, pp. 472–489.

[175] G. Chen, T. X. Han, Z. He, R. Kays, and T. Forrester, "Deep convolutional neural network based species recognition for wild animal monitoring," in *2014 IEEE International Conference on Image Processing (ICIP)*, Oct 2014, pp. 858–862.

[176] S. Schneider, G. W. Taylor, and S. C. Kremer, "Deep Learning Object Detection Methods for Ecological Camera Trap Data," *CoRR*, vol. abs/1803.10842, 2018. [Online]. Available: http://arxiv.org/abs/1803.10842

[177] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-time Object Detection with Region Proposal Networks," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'15. Cambridge, MA, USA: MIT Press, 2015, pp. 91–99. [Online]. Available: http://dl.acm.org/citation.cfm?id=2969239.2969250

[178] M. S. Norouzzadeh, A. Nguyen, M. Kosmala, A. Swanson, M. S. Palmer, C. Packer, and J. Clune, "Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning," *Proceedings of the National Academy of Sciences*, vol. 115, no. 25, pp. E5716–E5725, 2018. [Online]. Available: https://www.pnas.org/content/115/25/E5716

[179] H. Yousif, J. Yuan, R. Kays, and Z. He, "Object detection from dynamic scene using joint background modeling and fast deep learning classification," *Journal of Visual Communication and Image Representation*, vol. 55, pp. 802–815, 2018.

[180] H. Yousif, J. Yuan, R. Kays, and Z. He, "Fast human-animal detection from highly cluttered camera-trap images using joint background modeling and deep learning classification," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2017, pp. 1–4.

[181] C. Zhu, T. H. Li, and G. Li, "Towards Automatic Wild Animal Detection in Low Quality Camera-Trap Images Using Two-Channeled Perceiving Residual Pyramid Networks," in *ICCVW*, Oct 2017, pp. 2860–2864.

[182] H. J. Mönck, A. Jörg, T. von Falkenhausen, J. Tanke, B. Wild, D. Dormagen, J. Piotrowski, C. Winklmayr, D. Bierbach, and T. Landgraf, "BioTracker: An Open-Source Computer Vision Framework for Visual Animal Tracking," *ArXiv*, vol. abs/1803.07985, 2018.

[183] V. H. Sridhar, D. G. Roche, and S. Gingins, "Tracktor: Image-based automated tracking of animal movement and behaviour," *Methods in Ecology and Evolution*, vol. 10, no. 6, pp. 815–820, 2019. [Online]. Available: https://besjournals.onlinelibrary.wiley.com/doi/abs/10.1111/2041-210X.13166

[184] Z. Werkhoven, C. Rohrsen, C. Qin, B. Brembs, and B. de Bivort, "MARGO (Massively Automated Real-time GUI for Object-tracking), a platform for high-throughput ethology," *bioRxiv*, 2019. [Online]. Available: https://www.biorxiv.org/content/early/2019/03/30/593046

[185] F. Romero-Ferrero, M. G. Bergomi, R. C. Hinz, F. J. H. Heras, and G. G. de Polavieja, "idtracker.ai: Tracking all individuals in small or large collectives of unmarked animals," *Nature Methods*, vol. 16, no. 2, pp. 179–182, 2019. [Online]. Available: https://doi.org/10.1038/s41592-018-0295-5

[186] J. Patman, S. C. J. Michael, M. M. F. Lutnesky, and K. Palaniappan, "BioSense: Real-Time Object Tracking for Animal Movement and Behavior Research," in *2018 IEEE Applied Imagery Pattern Recognition Workshop (AIPR)*, Oct 2018, pp. 1–8.

[187] T. T. Zin, I. Kobayashi, P. Tin, and H. Hama, "A General Video Surveillance Framework for Animal Behavior Analysis," in *2016 Third International Conference on Computing Measurement Control and Sensor Network (CMCSN)*, May 2016, pp. 130–133.

[188] A. Ter-Sarkisov, J. D. Kelleher, B. Earley, M. Keane, and R. J. Ross, "Beef Cattle Instance Segmentation Using Fully Convolutional Neural Network," in *BMVC*, 2018.

[189] D. C. Wilkins, K. M. Kockelman, and N. Jiang, "Animal-vehicle collisions in Texas: How to protect travelers and animals on roadways," *Accident Analysis & Prevention*, vol. 131, pp. 157–170, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0001457519301125

[190] D. Forslund and J. Bjarkefur, "Night vision animal detection," in *2014 IEEE Intelligent Vehicles Symposium Proceedings*, June 2014, pp. 737–742.

[191] M. K. Grace, D. J. Smith, and R. F. Noss, "Reducing the threat of wildlife-vehicle collisions during peak tourism periods using a Roadside Animal Detection System," *Accident Analysis & Prevention*, vol. 109, pp. 55–61, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0001457517303597

[192] W. Xue, T. Jiang, and J. Shi, "Animal intrusion detection based on convolutional neural network," in *2017 17th International Symposium on Communications and Information Technologies (ISCIT)*, Sep. 2017, pp. 1–5.

[193] Q. Zhu, J. Ren, D. Barclay, S. McCormack, and W. Thomson, "Automatic Animal Detection from Kinect Sensed Images for Livestock Monitoring and Assessment," in *IEEE International Conference on Computer and Information Technology*, Oct 2015, pp. 1154–1157.

[194] "Nature Footage," https://www.naturefootage.com, 2019.

[195] Tzutalin, "LabelImg: A graphical image annotation tool and label object bounding boxes in images," online: https://github.com/tzutalin/labelImg, 2015.

[196] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. S. Torr, "Fully-Convolutional Siamese Networks for Object Tracking," in *Computer Vision – ECCV 2016 Workshops*, G. Hua and H. Jégou, Eds. Cham: Springer International Publishing, 2016, pp. 850–865.

[197] A. Dutta and A. Zisserman, "The VIA Annotation Software for Images, Audio and Video," in *Proceedings of the 27th ACM International Conference on Multimedia*, ser. MM '19. New York, NY, USA: ACM, 2019. [Online]. Available: https://doi.org/10.1145/3343031.3350535

[198] A. Dutta, A. Gupta, and A. Zissermann, "VGG Image Annotator (VIA)," http://www.robots.ox.ac.uk/ vgg/software/via/, 2016.

[199] K. Wada, "labelme: Image Polygonal Annotation with Python," https://github.com/wkentaro/labelme, 2016.

[200] A. Bréhéret, "Pixel Annotation Tool," https://github.com/abreheret/PixelAnnotationTool, 2017.

[201] B. Russell, A. Torralba, J. Yuen, and et. al, "LabelMe: The open annotation tool," online: http://labelme2.csail.mit.edu/Release3.0/index.php, 2019.

[202] R. Kawamura, "RectLabel: An image annotation tool to label images for bounding box object detection and segmentation," online: https://rectlabel.com/, 2019.

[203] "Labelbox," online: https://labelbox.com/, 2019.

[204] "Playment," online: https://playment.io, 2019.

[205] S. Xie and Z. Tu, "Holistically-Nested Edge Detection," in *ICCV*, Dec 2015, pp. 1395–1403.

[206] Q. Wang, L. Zhang, L. Bertinetto, W. Hu, and P. H. Torr, "Fast Online Object Tracking and Segmentation: A Unifying Approach," in *CVPR*, 2019.

[207] R. Benenson, S. Popov, and V. Ferrari, "Large-scale interactive object segmentation with human annotators," *CVPR*, vol. abs/1903.10830, 2019.

[208] Y. Zhu*, K. Sapra*, F. A. Reda, K. J. Shih, S. Newsam, A. Tao, and B. Catanzaro, "Improving Semantic Segmentation via Video Propagation and Label Relaxation," in *CVPR*, June 2019. [Online]. Available: https://nv-adlr.github.io/publication/2018-Segmentation

[209] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Region-Based Convolutional Networks for Accurate Object Detection and Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, pp. 142–158, 2016.

[210] K. He, X. Zhang, S. Ren, and J. Sun, "Identity Mappings in Deep Residual Networks," *CoRR*, vol. abs/1603.05027, 2016. [Online]. Available: http://arxiv.org/abs/1603.05027

[211] ——, "Deep Residual Networks with 1K Layers," online: https://github.com/KaimingHe/resnet-1k-layers, April 2016.

[212] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," in *CVPR*, 2015.

[213] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[214] A. Howard, M. Sandler, G. Chu, L. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for MobileNetV3," *CoRR*, vol. abs/1905.02244, 2019. [Online]. Available: http://arxiv.org/abs/1905.02244

[215] W. Han, P. Khorrami, T. L. Paine, P. Ramachandran, M. Babaeizadeh, H. Shi, J. Li, S. Yan, and T. S. Huang, "Seq-NMS for Video Object Detection," *CoRR*, vol. abs/1602.08465, 2016.

[216] K. Kang, W. Ouyang, H. Li, and X. Wang, "Object Detection from Video Tubelets with Convolutional Neural Networks," *CVPR*, pp. 817–825, 2016.

[217] K. Kang, H. Li, J. Yan, X. Zeng, B. Yang, T. Xiao, C. Zhang, Z. Wang, R. Wang, X. Wang, and W. Ouyang, "T-CNN: Tubelets with Convolutional Neural Networks for Object Detection from Videos," *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 1–1, 2018.

[218] K. Kang, H. Li, T. Xiao, W. Ouyang, J. Yan, X. Liu, and X. Wang, "Object Detection in Videos with Tubelet Proposal Networks," *CVPR*, pp. 889–897, 2017.

[219] C. Feichtenhofer, A. Pinz, and A. Zisserman, "Detect to Track and Track to Detect," in *ICCV*, 2017.

[220] X. Zhu, Y. Wang, J. Dai, L. Yuan, and Y. Wei, "Flow-Guided Feature Aggregation for Video Object Detection," *ICCV*, pp. 408–417, 2017.

[221] X. Zhu, Y. Xiong, J. Dai, L. Yuan, and Y. Wei, "Deep Feature Flow for Video Recognition," in *CVPR*, July 2017, pp. 4141–4150.

[222] M. Liu and M. Zhu, "Mobile Video Object Detection with Temporally-Aware Feature Maps," *CVPR*, pp. 5686–5695, 2018.

[223] M. Liu, M. Zhu, M. White, Y. Li, and D. Kalenichenko, "Looking Fast and Slow: Memory-Guided Mobile Video Object Detection," *CoRR*, vol. abs/1903.10172, 2019. [Online]. Available: http://arxiv.org/abs/1903.10172

[224] "Tensorflow Mobile Video Object Detection," Github, 2019, https://github.com/tensorflow/models/research/lstm_object_detection.

[225] Z. Zhu, Q. Wang, L. Bo, W. Wu, J. Yan, and W. Hu, "Distractor-aware Siamese Networks for Visual Object Tracking," in *ECCV*, 2018.

[226] P. O. Pinheiro, T. Lin, R. Collobert, and P. Dollár, "Learning to Refine Object Segments," in *ECCV*, 2016.

[227] P. H. O. Pinheiro, R. Collobert, and P. Dollár, "Learning to Segment Object Candidates," in *NIPS*, 2015. [Online]. Available: http://papers.nips.cc/paper/5852-learning-to-segment-object-candidates

[228] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," in *ICCV*, 2017, pp. 2980–2988.

[229] Y. Li, H. Qi, J. Dai, X. Ji, and Y. Wei, "Fully Convolutional Instance-Aware Semantic Segmentation," in *CVPR*, 2017, pp. 4438–4446.

[230] "Tensorflow Object Detection API," online: https://github.com/tensorflow/models/tree/master/research/object_detection, Sept 2019.

[231] G. Jocher, "YOLOv3 in PyTorch," online: https://github.com/ultralytics/yolov3, September 2019.

[232] Trieu, "Darkflow," online: https://github.com/thtrieu/darkflow, March 2018.

[233] J. Redmon, "Darknet," online: https://github.com/pjreddie/darknet, August 2018.

[234] "TensorFlow implementation of DeepMask and SharpMask," online: https://github.com/aby2s/sharpmask, September 2019.

[235] "Fully Convolutional Instance-aware Semantic Segmentation," online: https://github.com/msracver/FCIS, September 2019.

[236] L. Bertinetto and J. Valmadre, "SiamFC - TensorFlow," online: https://github.com/torrvision/siamfc-tf, September 2019.

[237] Q. Wang, "SiamMask," online: https://github.com/foolwood/SiamMask, September 2019.

[238] ——, "DaSiamRPN," online: https://github.com/foolwood/DaSiamRPN, September 2019.

[239] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[240] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba, "Scene Parsing through ADE20K Dataset," in *CVPR*, 2017.

[241] S. Wang, W. Liu, J. Wu, L. Cao, Q. Meng, and P. J. Kennedy, "Training deep neural networks on imbalanced data sets," in *International Joint Conference on Neural Networks*, 2016.

[242] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, "Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors," in *CVPR*, 2017.

[243] C.-Y. Huang, C.-L. Liu, C.-Y. Ting, Y.-T. Chiu, Y.-C. Cheng, M. W. Nicholson, and P. C. H. Hsieh, "Human iPSC Banking: Barriers and Opportunities," *J. Biomed. Sci.*, vol. 26, no. 1, p. 87, Oct. 2019.

[244] K. Takahashi and S. Yamanaka, "Induction of pluripotent stem cells from mouse embryonic and adult fibroblast cultures by defined factors," *Cell*, vol. 126, no. 4, pp. 663–676, Aug. 2006.

[245] K. Takahashi, K. Tanabe, M. Ohnuki, M. Narita, T. Ichisaka, K. Tomoda, and S. Yamanaka, "Induction of pluripotent stem cells from adult human fibroblasts by defined factors," *Cell*, vol. 131, no. 5, pp. 861–872, Nov. 2007.

[246] L. Ye, C. Swingen, and J. Zhang, "Induced pluripotent stem cells and their potential for basic and clinical sciences," *Curr. Cardiol. Rev.*, vol. 9, no. 1, pp. 63–72, Feb. 2013.

[247] K. Takahashi and S. Yamanaka, "Induced pluripotent stem cells in medicine and biology," *Development*, vol. 140, no. 12, pp. 2457–2461, Jun. 2013.

[248] G. Amabile and A. Meissner, "Induced pluripotent stem cells: current progress and potential for regenerative medicine," *Trends Mol. Med.*, vol. 15, no. 2, pp. 59–68, Feb. 2009.

[249] S. Yamanaka, "A fresh look at iPS cells," *Cell*, vol. 137, no. 1, pp. 13–17, Apr. 2009.

[250] D. Kusumoto, M. Lachmann, T. Kunihiro, S. Yuasa, Y. Kishino, M. Kimura, T. Katsuki, S. Itoh, T. Seki, and K. Fukuda, "Automated Deep Learning-Based System to Identify Endothelial Cells Derived from Induced Pluripotent Stem Cells," *Stem Cell Reports*, vol. 10, pp. 1687 – 1695, 2018.

[251] A. Waisman, A. La Greca, A. M. Möbbs, M. A. Scarafía, N. L. Santín Velazque, G. Neiman, L. N. Moro, C. Luzzani, G. E. Sevlever, A. S. Guberman, and S. G. Miriuka, "Deep learning neural networks highly predict very early onset

of pluripotent stem cell differentiation," *Stem Cell Reports*, vol. 12, no. 4, pp. 845–859, Apr. 2019.

[252] T. Hirose, J. Kotoku, F. Toki, E. K. Nishimura, and D. Nanba, "Label-free quality control and identification of human keratinocyte stem cells by deep learning-based automated cell tracking," *Stem Cells (Dayton, Ohio)*, vol. 39, pp. 1091 – 1100, 2021.

[253] C. Coronnello and M. G. Francipane, "Moving Towards Induced Pluripotent Stem Cell-based Therapies with Artificial Intelligence and Machine Learning," *Stem Cell Reviews and Reports*, vol. 18, pp. 559 – 569, 2021.

[254] D. Kusumoto, S. Yuasa, and K. Fukuda, "Induced Pluripotent Stem Cell-Based Drug Screening by Use of Artificial Intelligence," *Pharmaceuticals*, vol. 15, 2022.

[255] Y. Lan, N. Huang, Y. Fu, K. Liu, H. Zhang, Y. Li, and S. Yang, "Morphology-Based Deep Learning Approach for Predicting Osteogenic Differentiation," *Frontiers in Bioengineering and Biotechnology*, vol. 9, 2022.

[256] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Q. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan, "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions," *Journal of Big Data*, vol. 8, 2021.

[257] J. Chai, H. Zeng, A. Li, and E. W. Ngai, "Deep Learning in Computer Vision: A Critical Review of Emerging Techniques and Application Scenarios," *Machine Learning with Applications*, vol. 6, p. 100134, 2021.

[258] A. Byerly, T. Kalganova, and R. Ott, "The Current State of the Art in Deep Learning for Image Classification: A Review," in *Sai*, 2022.

[259] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. PietikÃ¤inen, "Deep Learning for Generic Object Detection: A Survey," *IJCV*, vol. 128, no. 2, pp. 261–318, 2020.

[260] W. Gu, S. Bai, and L.-Y. Kong, "A review on 2D Instance Segmentation based on Deep Neural Networks," *Image Vis. Comput.*, vol. 120, p. 104401, 2022.

[261] Y. Mo, Y. Wu, X. Yang, F. Liu, and Y. Liao, "Review the state-of-the-art technologies of semantic segmentation based on deep learning," *Neurocomputing*, vol. 493, pp. 626–646, 2022.

[262] S. M. Marvasti-Zadeh, L. Cheng, H. Ghanei-Yakhdan, and S. Kasaei, "Deep Learning for Visual Tracking: A Comprehensive Survey," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–26, 2021.

[263] L. Jiao, D. Wang, Y. Bai, P. Chen, and F. Liu, "Deep Learning in Visual Tracking: A Review," *IEEE transactions on neural networks and learning systems*, 2021.

[264] S. K. Pal, A. Pramanik, J. Maiti, and P. Mitra, "Deep learning in multi-object detection and tracking: state of the art," *Applied Intelligence*, vol. 51, pp. 6400 – 6429, 2021.

[265] S. Suganyadevi, V. Seethalakshmi, and K. Balasamy, "A review on deep learning in medical image analysis," *International Journal of Multimedia Information Retrieval*, vol. 11, pp. 19 – 38, 2021.

[266] L. Cai, J. Gao, and D. Zhao, "A review of the application of deep learning in medical image classification and segmentation," *Annals of Translational Medicine*, vol. 8, 2020.

[267] X. Liu, L. Song, S. Liu, and Y. Zhang, "A Review of Deep-Learning-Based Medical Image Segmentation Methods," *Sustainability*, 2021.

[268] T. Wen, B. Tong, Y. Liu, T. Pan, Y. Du, Y. Chen, and S. Zhang, "Review of research on the instance segmentation of cell images," *Computer methods and programs in biomedicine*, vol. 227, 2022.

[269] T. Ben-Haim and T. Riklin-Raviv, "Graph Neural Network for Cell Tracking in Microscopy Videos," in *ECCV*, 2022.

[270] Y. Chen, Y. Song, C. Zhang, F. Zhang, L. J. Oâ€™Donnell, W. Chrzanowski, and W. T. Cai, "Celltrack R-CNN: A Novel End-To-End Deep Neural Network For Cell Segmentation And Tracking In Microscopy Images," *ISBI*, pp. 779–782, 2021.

[271] J. Wang, X. Su, L. Zhao, and J. Zhang, "Deep Reinforcement Learning for Data Association in Cell Tracking," *Frontiers in Bioengineering and Biotechnology*, vol. 8, 2020.

[272] J.-B. Lugagne, H. Lin, and M. J. Dunlop, "DeLTA: Automated cell segmentation, tracking, and lineage reconstruction using deep learning," *PLOS Computational Biology*, vol. 16, no. 4, pp. 1–18, 04 2020.

[273] V. Ulman, M. Mavska, K. E. G. Magnusson, O. Ronneberger, C. Haubold, N. Harder, P. Matula, P. Matula, D. Svoboda, M. Radojević, I. Smal, K. Rohr, J. Jaldén, H. M. Blau, O. Dzyubachyk, B. P. F. Lelieveldt, P. Xiao, Y. Li, S.-Y. Cho, A. C. Dufour, J.-C. Olivo-Marin, C. C. Reyes-Aldasoro, J. A. Solís-Lemus, R. Bensch, T. Brox, J. Stegmaier, R. Mikut, S. Wolf, F. A. Hamprecht, T. Esteves, P. Quelhas, Ö. Demirel, L. Malmström, F. Jug, P. TomanÃ§ak, E. H. W. Meijering, A. Muñoz-Barrutia, M. Kozubek, and C. O. de Solórzano, "An Objective Comparison of Cell Tracking Algorithms," *Nature methods*, vol. 14, pp. 1141 – 1152, 2017.

[274] Y. Liu, Y. Zhang, Y. Wang, F. Hou, J. Yuan, J. Tian, Y. Zhang, Z. Shi, J. Fan, and Z. He, "A Survey of Visual Transformers," *IEEE transactions on neural networks and learning systems*, vol. PP, 2021.

[275] Y. Xiang, "PapersWithCode - Instance Segmentation," online: https://paperswithcode.com/task/instance-segmentation, 2023.

[276] S. Anjum and D. Gurari, "CTMC: Cell Tracking with Mitosis Detection Dataset Challenge," *CVPR Workshops*, pp. 4228–4237, 2020.

[277] I.-H. Park, P. H. Lerou, R. Zhao, H. Huo, and G. Q. Daley, "Generation of human-induced pluripotent stem cells," *Nat. Protoc.*, vol. 3, no. 7, pp. 1180–1186, 2008.

[278] Q. Wang, L. Zhang, L. Bertinetto, W. Hu, and P. H. S. Torr, "Fast Online Object Tracking and Segmentation: A Unifying Approach," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1328–1338, 2018.

[279] A. Singh, "iPSC prediction with deep learning," online: https://github.com/abhineet123/ipsc_prediction, 2023.

[280] D. K. McClish, "Analyzing a Portion of the ROC Curve," *Medical Decision Making*, vol. 9, pp. 190 – 195, 1989.

[281] J. Huang, V. Rathod, C. Sun, M. Zhu, A. K. Balan, A. Fathi, I. S. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. P. Murphy, "Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors," *CVPR*, 2017.

[282] B. Krawczyk, "Learning from imbalanced data: Open challenges and future directions," *Progress in Artificial Intelligence*, vol. 5, pp. 221–232, 2016.

[283] Y.-S. Wang, H. yi Lee, and Y.-N. V. Chen, "Tree Transformer: Integrating Tree Structures into Self-Attention," in *Conference on Empirical Methods in Natural Language Processing*, 2019.

[284] M. Ahmed and R. E. Mercer, "Encoding Dependency Information inside Tree Transformer," *Proceedings of the Canadian Conference on Artificial Intelligence*, 2021.

[285] W. Wang, K. Zhang, G. Li, S. Liu, Z. Jin, and Y. Liu, "A Tree-structured Transformer for Program Representation Learning," *ArXiv*, vol. abs/2208.08643, 2022.

[286] S. Zhong, S. Song, G. Li, and S. H. G. Chan, "A Tree-Based Structure-Aware Transformer Decoder for Image-To-Markup Generation," *Proceedings of the 30th ACM International Conference on Multimedia*, 2022.

[287] X. Li, H. Xiong, X. Li, X. Wu, X. Zhang, J. Liu, J. Bian, and D. Dou, "Interpretable Deep Learning: Interpretation, Interpretability, Trustworthiness, and Beyond," *Knowledge and Information Systems*, vol. 64, pp. 3197 – 3234, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:232290756

[288] W. Samek, G. Montavon, S. Lapuschkin, C. J. Anders, and K.-R. Müller, "Explaining Deep Neural Networks and Beyond: A Review of Methods and Applications," *Proceedings of the IEEE*, vol. 109, pp. 247–278, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:232046495

[289] C. Molnar, *Interpretable Machine Learning*. Lulu. com, 2020.

[290] W. Samek, G. Montavon, S. Lapuschkin, C. J. Anders, K.-R. Müller *et al.*, "Toward interpretable machine learning: Transparent deep neural networks and beyond," *arXiv preprint arXiv:2003.07631*, vol. 2, 2020.

[291] W. Samek, G. Montavon, A. Vedaldi, L. K. Hansen, and K.-R. Müller, *Explainable AI: interpreting, explaining and visualizing deep learning*. Springer Nature, 2019, vol. 11700.

[292] A. Ali, T. Schnake, O. Eberle, G. Montavon, K.-R. Müller, and L. Wolf, "XAI for Transformers: Better Explanations through Conservative Propagation," in *Proceedings of the 39th International Conference on Machine Learning*, vol. 162, 17–23 Jul 2022, pp. 435–451.

[293] C. J. Anders, G. Montavon, W. Samek, and K.-R. Müller, "Understanding patch-based learning of video data by explaining predictions," *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, pp. 297–309, 2019.

[294] Y. Brima and M. Atemkeng, "What do Deep Neural Networks Learn in Medical Images?" *arXiv preprint arXiv:2208.00953*, 2022.

[295] A. Singh, "Video Detection and Segmentation with Language Modeling," online: https://webdocs.cs.ualberta.ca/~asingh1/p2s/.

[296] L. Wen, D. Du, Z. Cai, Z. Lei, M.-C. Chang, H. Qi, J. Lim, M.-H. Yang, and S. Lyu, "UA-DETRAC: A new benchmark and protocol for multi-object detection and tracking," *CVIU*, vol. 193, p. 102907, 2020.

[297] Z. Liu, J. Ning, Y. Cao, Y. Wei, Z. Zhang, S. Lin, and H. Hu, "Video Swin Transformer," *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3192–3201, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:235624247

[298] C. Feichtenhofer, H. Fan, J. Malik, and K. He, "SlowFast Networks for Video Recognition," *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 6201–6210, 2018.

[299] M. Innat, "Keras 3 Implementation of Video Swin Transformers for 3D Video Modeling," online: https://github.com/innat/VideoSwin, December 2024.

[300] W. Kay, J. Carreira, K. Simonyan, B. Zhang, C. Hillier, S. Vijayanarasimhan, F. Viola, T. Green, T. Back, A. Natsev, M. Suleyman, and A. Zisserman, "The Kinetics Human Action Video Dataset," *ArXiv*, vol. abs/1705.06950, 2017.

[301] A. Robinson and C. Cherry, "Results of a prototype television bandwidth compression scheme," *Proceedings of the IEEE*, vol. 55, no. 3, pp. 356–364, 1967.

[302] J. Liang, N. Homayounfar, W.-C. Ma, Y. Xiong, R. Hu, and R. Urtasun, "PolyTransform: Deep Polygon Transformer for Instance Segmentation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 9131–9140.

[303] J. Lazarow, W. Xu, and Z. Tu, "Instance segmentation with mask-supervised polygonal boundary transformers," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 4382–4391.

[304] K. Chitta, J. M. Álvarez, and M. Hebert, "Quadtree Generating Networks: Efficient Hierarchical Scene Parsing with Sparse Convolutions," *IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 2009–2018, 2020.

[305] P. Soille, *Morphological Image Analysis: Principles and Applications*. Springer-Verlag, 2003.

[306] "Swin Transformer for Semantic Segmentaion," online: https://github.com/SwinTransformer/Swin-Transformer-Semantic-Segmentation.

[307] L. R. Dice, "Measures of the Amount of Ecologic Association Between Species," *Ecology*, vol. 26, no. 3, pp. 297–302, 1945. [Online]. Available: http://www.jstor.org/stable/1932409

[308] T. Sørensen, T. Sørensen, T. Biering-Sørensen, T. Sørensen, and J. T. Sorensen, "A method of establishing group of equal amplitude in plant sociobiology based on similarity of species content and its application to analyses of the vegetation on Danish commons." Kongelige Danske Videnskabernes Selskab, 1948.

[309] M. Fujitake and A. Sugimoto, "Video sparse transformer with attention-guided memory for video object detection," *IEEE Access*, vol. 10, pp. 65 886–65 900, 2022.

[310] "Object Detection on UA-DETRAC," online: https://paperswithcode.com/sota/object-detection-on-ua-detrac.

[311] M. Fujitake and A. Sugimoto, "Temporal feature enhancement network with external memory for object detection in surveillance video," in *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, 2021, pp. 7684–7691.

[312] K.-J. Kim, P.-K. Kim, Y.-S. Chung, and D.-H. Choi, "Performance enhancement of YOLOv3 by adding prediction layers with spatial pyramid pooling for vehicle detection," in *2018 15th IEEE international conference on advanced video and signal based surveillance (AVSS)*. IEEE, 2018, pp. 1–6.

[313] ——, "Multi-scale detector for accurate vehicle detection in traffic surveillance data," *IEEE Access*, vol. 7, pp. 78 311–78 319, 2019.

[314] H. Perreault, G.-A. Bilodeau, N. Saunier, and M. Héritier, "SpotNet: Self-attention multi-task network for object detection," in *2020 17th Conference on Computer and Robot Vision (CRV)*. IEEE, 2020, pp. 230–237.

[315] ——, "FFAVOD: Feature fusion architecture for video object detection," *Pattern Recognition Letters*, vol. 151, pp. 294–301, 2021.

[316] A. Singh, "Labeling tool demo," online: https://youtu.be/ZkjcP8s0QVQ, January 2020.

[317] ——, "Animal mask generation with segmentation methods," online: https://www.youtube.com/playlist?list=PLYIOwkVj6L8hErKjWOGtEDpbWZVq1BuRb, January 2020.

Appendix A

River Ice Segmentation

This appendix provides results on additional unlabeled images (Figure A.3, A.4), details of tested video sequences (Table A.3) and categorized links for Google photos albums showing results on labeled images (Table A.1) and Google Drive folders containing video results (Table A.4). Each row in the albums shows (from left to right) the original image, its ground truth label and its segmentation result. All these results are also available in the accompanying data [137].

Results comparing different backbone networks with DeepLab are also included in Section A.1.

Table A.1: Google Photos albums showing labeled qualitative results for ablation testing

Model	**n_training_images**					**n_pixels_per_class**			
SVM	32	24	16	8	4	-			
Deeplab	32	24	16	8	4	1000	100	10	2
UNet	32	24	16	8	4	1000	100	10	2
SegNet	32	24	16	8	4	1000	100	10	2
DenseNet	32	24	16	8	4	1000	100	10	2

A.1 Comparing DeepLab Backbone Networks

This section provides results comparing three backbone networks for DeepLab - Xception65 (used in the main paper) [6], Auto-DeepLab [134] and a modified version of ResNet101 [12] where the first 7×7 convolution has been replaced by three 3×3 convolutions similar to PSPNet [135].

Table A.2: Dimensions of the 50 labeled images

Image ID	Height	Wdth
1	1463	1462
2	973	1724
3	1228	1228
4, 5	1081	1281
6	1191	1192
7	1234	1234
8	1009	1134
9	1307	1539
10-12	1081	1281
13	1343	1357
14-50	1081	1281

Table A.3: Details of the 5 video sequences used for testing

ID	Name	Frame Count	Start Frame	End Frame
1	20160121 / YUN00001	3600	1	3600
2	20160122 / YUN00002	1800	701	2500
3	20160122 / YUN00020	1800	2001	3800
4	20161203 / Deployment_1_YUN00001	1800	901	2700
5	20161203 / Deployment_1_YUN00002	1800	1	1800

Table A.4: Google Drive links for video results

Type	Video ID					Type	Video ID				
deeplab_densenet_unet						selective_pixel_ablation					
combined	1	2	3	4	5	densenet	1			3	
frazil	1	2	3	4	5	deeplab	1			3	
anchor	1	2	3	4	5	unet	1			3	
svm_densenet_unet						compare	1				
combined	1	3		4		training_images_ablation					
frazil	1	3		4		densenet	1	2	3	4	5
anchor	1	3		4		deeplab	1	2	3	4	5

A.1.1 Overview

Table A.5 shows the results for both segmentation recall and precision as well as ice concentration median mean absolute error (MAE). It can be seen that Xception65 has similar performance as the other newer backbones and is even slightly better in most metrics except segmentation recall where Auto-DeepLab and ResNet101 (PSP) dominate for anchor and frazil ice respectively.

Backbone	anchor ice	frazil ice	ice+water	ice+water(fw)	Median MAE (%)
Recall (%)					Anchor Ice
Xception65	74.46	87.51	**86.38**	**90.87**	**4.71**
Auto-Deeplab	**78.69**	81.28	85.70	90.21	5.06
ResNet101 (PSP)	67.07	**93.24**	85.16	90.81	4.78
Precision (%)					Frazil Ice
Xception65	**62.39**	**77.14**	**77.25**	**84.32**	**4.52**
Auto-Deeplab	59.44	73.43	75.57	83.66	5.24
ResNet101 (PSP)	59.79	76.47	76.34	84.01	5.49

Table A.5: Segmentation recall and precision along with ice concentration median MAE for DeepLab with three different backbone architectures trained and tested on the 32 and 18 image sets respectively.

A.1.2 Ablation Tests

Figure A.1 and A.2 respectively show the results of training images and selective pixel ablation tests. As in the previous section, Xception65 remains competitive with the other backbones though it seems to be slightly more outperformed here especially with fewer training images in Figure A.1 and frazil ice selective pixel tests in Figure A.2.

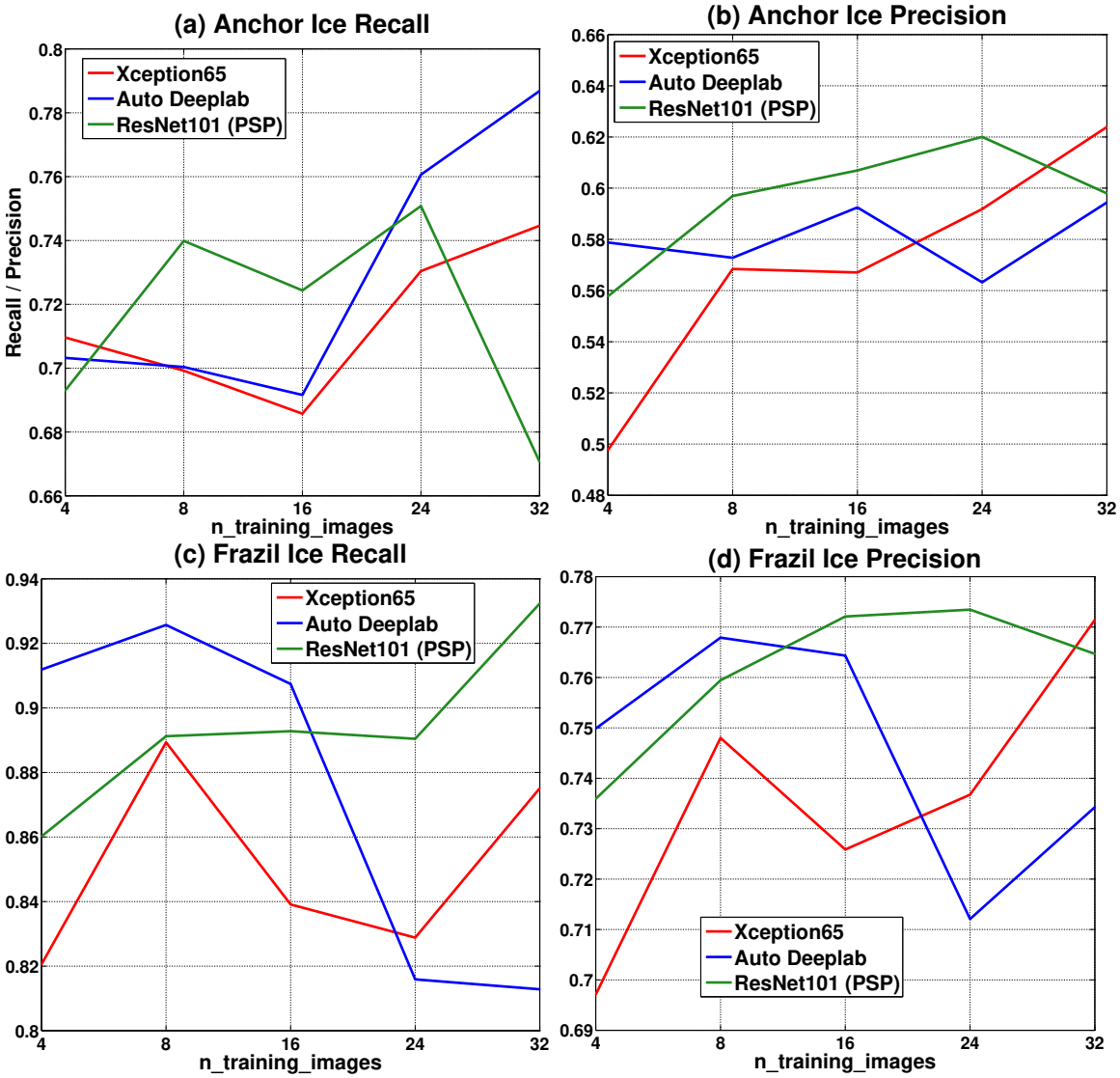

Figure A.1: Results of ablation tests with training images for (a-b) anchor ice and (c-d) frazil ice. Note the variable Y-axis limits.

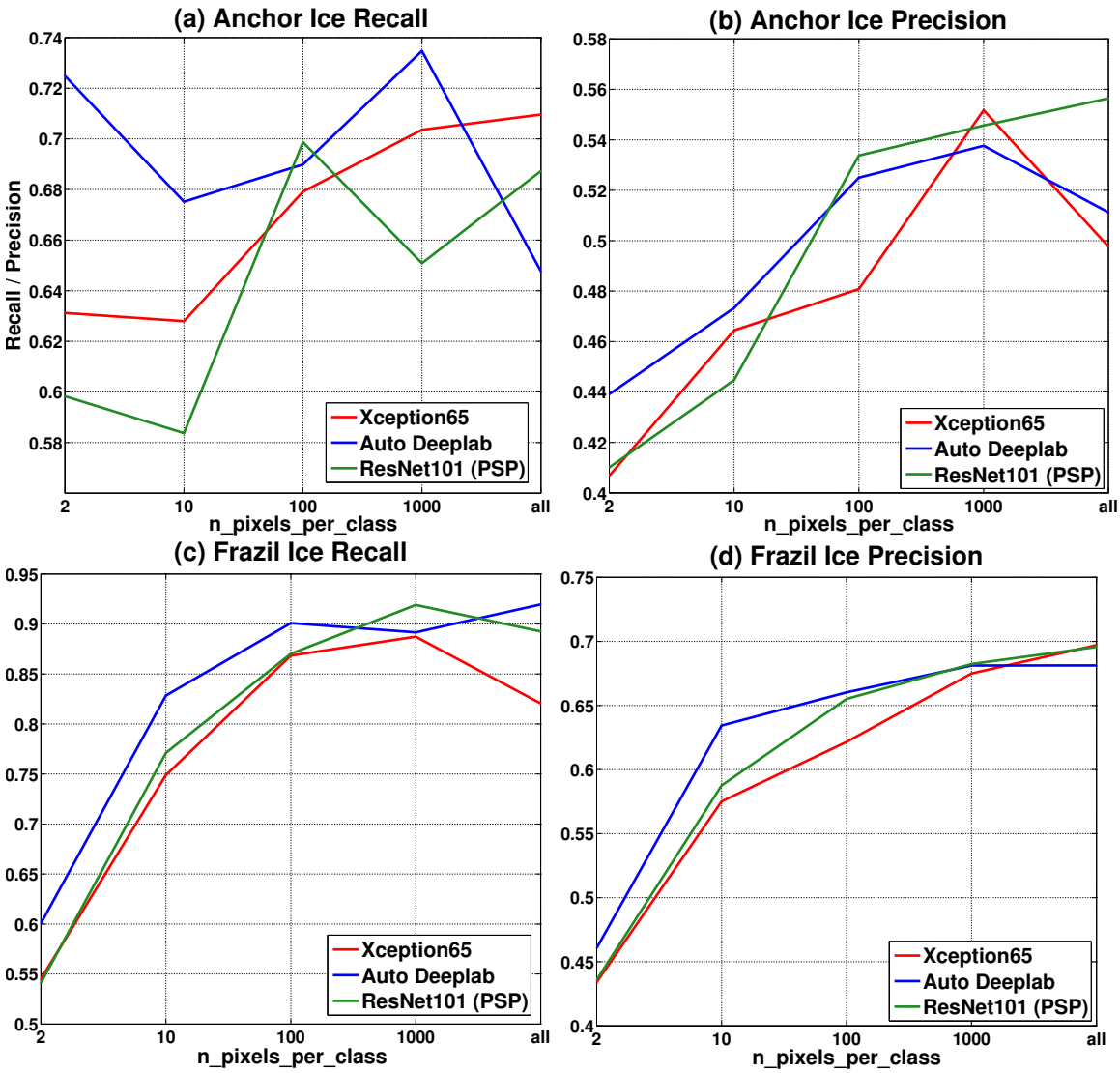

Figure A.2: Results of ablation tests with selective pixels for (a-b) anchor ice and (c-d) frazil ice. Note the variable Y-axis limits.

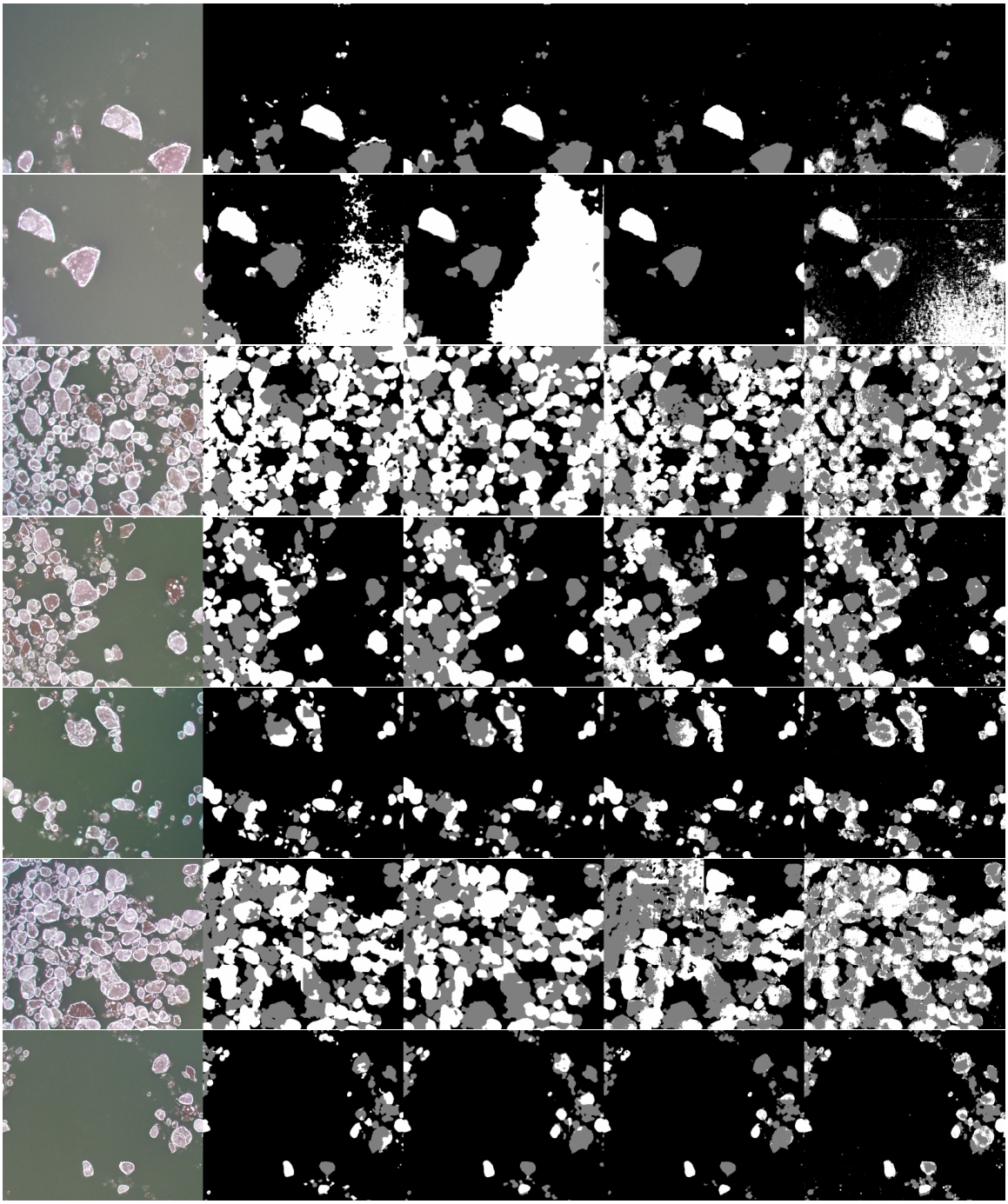

Figure A.3: Results of applying the best configurations of the four models on unlabeled images: left to right: Raw image, UNet, SegNet, Deeplab, DenseNet

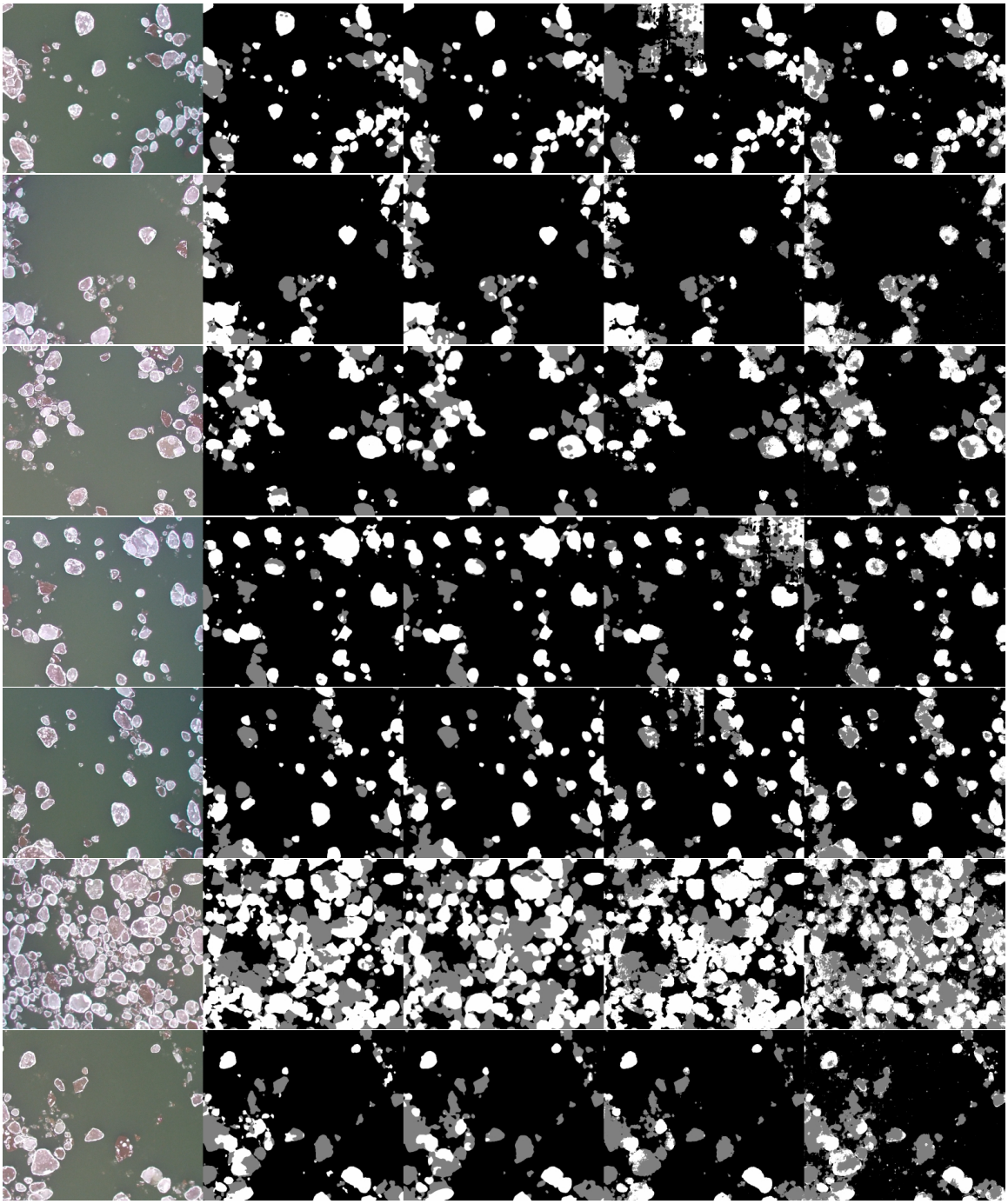

Figure A.4: Results of applying the best configurations of the four models on unlabeled images: left to right: raw image, UNet, SegNet, Deeplab, DenseNet

Appendix B

Animal Detection

B.1 Introduction

This appendix provides details of the tracking (Section B.2) and multi-model pooling experiments (Section B.3) along with the corresponding results. Class-agnostic Recall-Precision (cRP) for training configurations #1 - #5 (Table B.2) along with class-specific results for #1 and #2 (Tables B.3 - B.6) are also included. Finally, links are provided for more synthetic data samples generated using the four mask generation methods detailed in the paper (Table B.1).

A video demonstrating the end-to-end real and synthetic data generation pipeline created using the labeling tool accompanies this document. A higher resolution version of the same is also available on YouTube [316] along with mask generation results produced by several instance and semantic segmentation methods [317].

B.2 YOLO+DASiamRPN

Algorithm 1 details the process used for combining YOLO with DASiamRPN tracker to reduce false negatives using temporal information in videos. The *cumulative_ score* in line 48 is a composite measure of overall tracker fitness defined as:

$$cumulative\_score = cumulative\_confidence \times \frac{assoc\_count + 1}{unassoc\_count + 1} \tag{B.1}$$

where *assoc_ count* is the number of frames that the tracker has been successfully associated in since its initialization, *unassoc_ count* is the number of frames since it was last associated and *cumulative_ confidence* is the product of tracker confidence

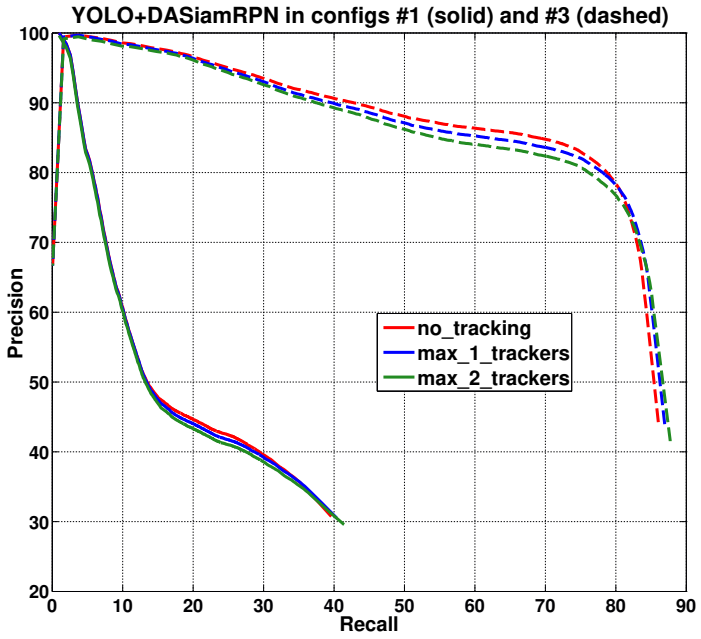

Figure B.1: Mean Recall vs Precision for DASiamRPN + YOLO in configurations #1 (solid) and #3 (dashed) using unassociated tracker removal with $max\_unassoc = 2$

background	**animal**			
airport	bear	coyote	deer	moose
highway	bear	coyote	deer	moose

Table B.1: Google Photos albums showing samples of synthetic data generated using all 4 types of mask generation methods - (left to right in each row) manual masks, Mask RCNN, SiamMask and no mask/Gaussian blending,

in all tracked frames so far as well as the confidence of the detection on which it was initialized.

Removing trackers that have remained unassociated for too long (lines 30-34) can help to increase precision by reducing tracked boxes corresponding to false positive detections. However, this also leads to a significant drop in recall since false negatives often occur in several consecutive frames which causes trackers corresponding to real objects to be removed. Thus, the overall performance, shown in Figure B.1, is nearly identical to not using tracking at all. Experiments were done with $max\_unassoc \in \{2, 3, 4, 5\}$ and best results were obtained with $max\_unassoc = 2$, which is the only one included here.

Algorithm 1 YOLO+DASiamRPN

min_conf, *min_iou*, *max_unassoc*, *max_trackers*, *min_init_gap* ← **params**
trackers ← ∅
for *frame* **in** *video_sequence* **do**
    detections ← YOLO (*frame*)
    associated_detections ← ∅
    associated_trackers ← ∅
    for *t* **in** *trackers* **do**
        t.update(*frame*)
        if *t.confidence* ¡ *min_conf* **then**
            trackers ← *trackers* \ *t*
            continue
        end if
        if *t* **in** *associated_trackers* **then**
            continue
        end if
        for *d* **in** *detections* **do**
            if *d* **in** *associated_detections* **then**
                continue
            end if
            if *d*.class = *t*.class **and** iou(*d*, *t*) ¿ *min_iou* **then**
                associated_detections ← *associated_detections* ∪ *d*
                associated_trackers ← *associated_trackers* ∪ *t*
                t.unassoc_count ← 0
                t.assoc_count ← *t.assoc_count* + 1
                continue
            end if
        end for
    end for
    unassociated_trackers ← *trackers* \ *associated_trackers*
    for *t* **in** *unassociated_trackers* **do**
        t.unassoc_count ← *t.unassoc_count* + 1
        if *t.unassoc_count* > *max_unassoc* **then**
            trackers ← *trackers* \ *t*
        end if
    end for
    unassociated_detections ← *detections* \ *associated_detections*
    tracker_init_gap ← *tracker_init_gap* + 1
    if *tracker_init_gap* > *min_init_gap* **then**
        tracker_init_gap ← 0
        for *d* **in** *unassociated_detections* **do**
            t ← initialize_new_tracker(*frame*, *d*)
            t.unassoc_count ← 0
            t.assoc_count ← 1
            trackers ← *trackers* ∪ *t*
        end for
    end if
    if |*trackers*| > *max_trackers* **then**
        trackers ← **sort** *trackers* **by** *cumulative_score*
        trackers ← $\{t_i | t_i \in trackers, 1 \leq i \leq max\_trackers\}$
    end if
end for

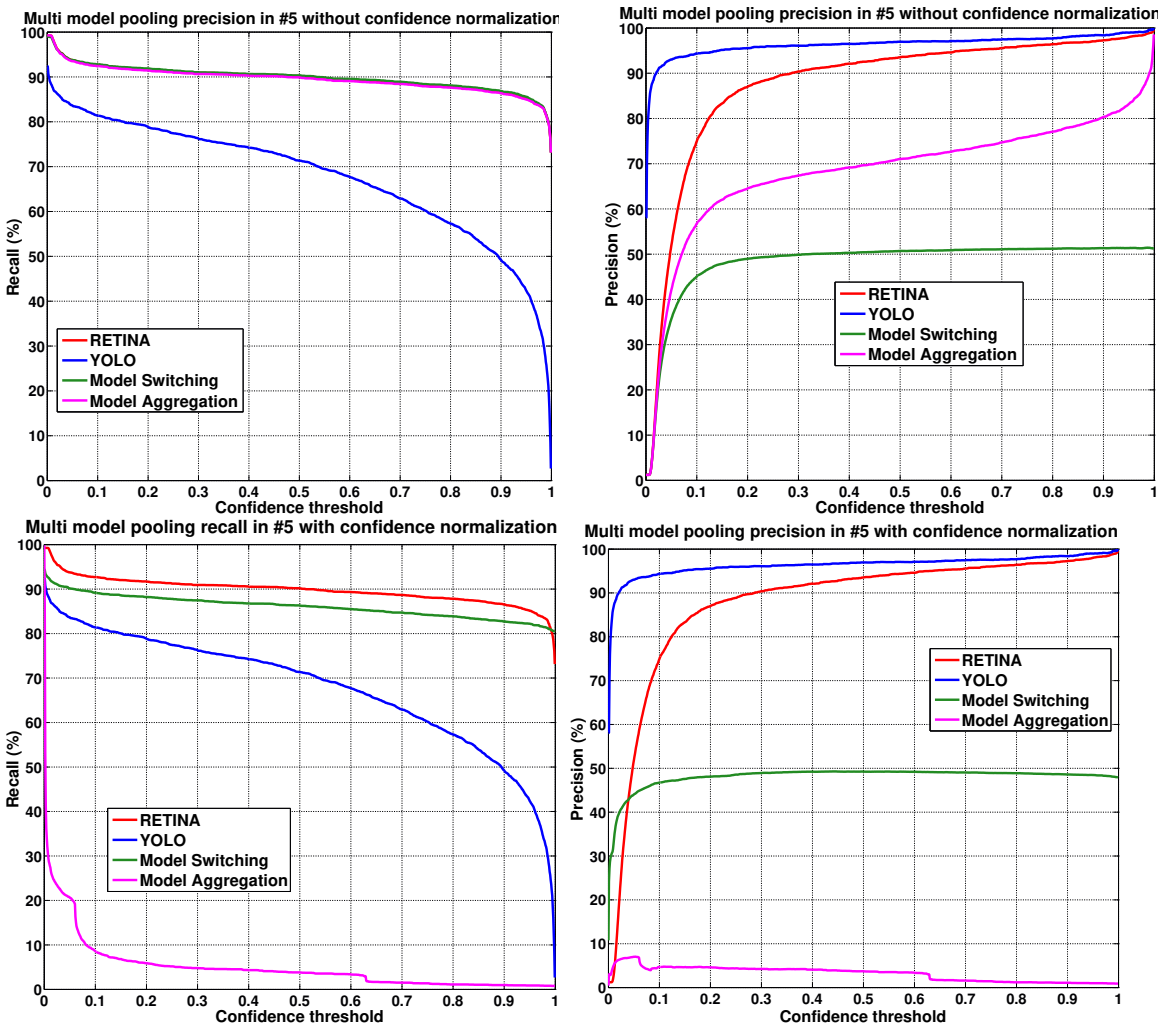

Figure B.2: Mean Recall and Precision for RETINA and YOLO pooling in #5 with and without confidence normalization

Detection cRP (%) for configuration #1 - #5					
Model	**1: 1K/Seq/8**	**2: 1K/Even/8**	**3: 10K/Seq/6**	**4: 5%/Start/6**	**5: 500/Static/3**
NAS	87.17	96.42	93.8	92.98	95.83
INRES	84.64	96.13	95.08	94.32	92.73
RES101	89.71	99.65	95.16	97.15	95.93
RFCN	88.93	99.54	94.64	97.27	93.88
RETINA	74.62	99.62	90.18	95.27	92.01
SSDIN	69.56	98.62	84.75	92	89.56
SSDMO	64.89	98.42	86.28	92.84	90.62
YOLO	80.76	96.67	91.7	94.16	90.36

Table B.2: Class agnostic Recall-Precision (cRP) for training configurations #1 - #5

B.3 Muti-Model Pooling

Muti-model pooling can be an effective way to reduce false negatives only if the pooled models exhibit distinct patterns of missing objects. As shown in Section 4.2 of the main paper, model recall depends strongly on the range of backgrounds in the training set. It would thus seem that model separation can best be achieved by training them on images with significantly different backgrounds, e.g. with snowy winter and green summer scenes or under sunny and cloudy / rainy weather conditions. However, the existing labeled data does not contain sufficient variety to divide it into class-balanced subsets that each have relatively homogeneous backgrounds differing significantly from those in other subsets. Model separation was instead achieved by training different architectures on the same data.

Two pooling strategies were employed:

1. Model Switching: All detections from the model with the highest confidence detection are retained in each frame while all others models' detections are discarded.

2. Model Aggregation: Detections from all models are pooled followed by class-agnostic non-maximum suppression applied to boxes from different models.

Both strategies require inter-model confidence comparison which is not straightforward to do since, as shown in Figure 3 of the main paper (and Tables B.3 - B.6 here), the meaning of confidence magnitude in terms of detection accuracy varies significantly between models. As a result, raw confidence values from different models need to be normalized before they can be compared. Several ways of doing this were explored but the one that worked best was to scale the confidence values of each model so that its mRP threshold becomes 0.5 while the range of values remains [0, 1].

Experiments were done by pooling YOLO and RETINA as well as YOLO, SSDIN and SSDMO but the former combination gave much better results so only these are included here. Similarly, configurations #1, #3 and #5 were all tested but only #5 is included as it produced the best pooling results relative to the individual detectors. Figure B.2 gives results for the two strategies in #5, both with and without confidence normalization. Somewhat surprisingly, none of the normalization techniques managed to outperform unnormalized confidence based pooling. Also, none of the pooling strategies managed to the increase recall over the better of the pooled detectors while

always leading to a significant drop in precision. This shows that pooling is not a practicle way to improve recall, at least not with the relatively simple methods employed here so as not to compromise speed.

B.4 Class-agnostic Recall-Precision (cRP)

Table B.2 shows the cRPs for all models in configurations #1 - #5. It is interesting to note that the large relative increase in cRP exhibited by YOLO in #1 and #3 is not present in the other easier configurations. This might indicate that YOLO manifests its tendency to overfit to specific background-foreground combinations most strongly with very limited training data.

B.5 Class-Specific Results

Results for configuration #1 are in tables B.3 and B.4 while those for #2 are in tables B.5 and B.6. The second column gives the recall – precision (RP) value corresponding to the score threshold where the two are equal while the threshold itself is given in the third column. The next three columns provide the recall, precision and their average for the score threshold corresponding to the overall mRP, i.e. where the mean recall and mean precision are equal. The last column gives the total number of ground truth boxes available for each class. Note that many images had multiple objects so that this count is more than the number of images in the corresponding test set.

It can be seen that the large inter-class variation of RP thresholds holds for all the models as well as for both low and high performance scenarios, represented by #1 and #2 respectively. AP and RP show considerable variation too, though only in #1. This is probably because animals with the larger ranges of backgrounds in the training set are often falsely detected in test images of remaining animals, thus leading to significant biases in their recall and precision.

B.6 Synthetic data samples

Table B.1 provides links to Google Photos albums showing synthetic data samples generated using all 4 methods - (left to right in each row) manual masks, Mask RCNN, SiamMask and no masks.

Training Configuration #1: 1K/Seq/8							
NAS							
	Class Specific			**mRP threshold 62.00 %**			
class	**AP(%)**	**RP(%)**	**Score(%)**	**Recall(%)**	**Precision(%)**	**Average(%)**	**GT**
bear	79.01	72.81	86.84	78.55	60.97	69.76	25320
bison	72.07	65.47	31.08	55.69	80.55	68.12	31612
cow	42.89	42.99	98.6	60.16	10.73	35.45	4222
coyote	75.33	67.87	16.66	52.53	94.45	73.49	22337
deer	63.89	59.88	30.88	49.29	67.3	58.3	24465
elk	65.77	57.19	3.45	36.53	89.99	63.26	25353
horse	71.33	65.58	94.89	80.64	41.41	61.03	4840
moose	50.31	47.36	97.2	65.56	33.14	49.35	25033
average	**65.07**	**59.89**	**57.45**	**59.87**	**59.82**	**59.84**	**163182**
INRES							
	Class Specific			**mRP threshold 38.70 %**			
class	**AP(%)**	**RP(%)**	**Score(%)**	**Recall(%)**	**Precision(%)**	**Average(%)**	**GT**
bear	68.02	67.09	16	59.49	75.72	67.61	25320
bison	61.51	57.97	15.8	52.58	66.44	59.51	31612
cow	26.19	29.58	98.77	48.48	6.24	27.36	4222
coyote	67.31	61.38	3.49	50.83	90.74	70.79	22337
deer	67.34	64.49	58.18	68.49	61.02	64.75	24465
elk	59.12	54.26	4.5	42.42	65.98	54.2	25353
horse	50.11	49.13	96.45	62.95	22.22	42.59	4840
moose	49.53	49.04	47.33	50.46	47.36	48.91	25033
average	**56.14**	**54.12**	**42.57**	**54.47**	**54.46**	**54.46**	**163182**
RES101							
	Class Specific			**mRP threshold 65.80 %**			
class	**AP(%)**	**RP(%)**	**Score(%)**	**Recall(%)**	**Precision(%)**	**Average(%)**	**GT**
bear	70.95	67.76	98.52	84.5	42.27	63.38	25320
bison	56.19	55.56	49.13	48.64	58.46	53.55	31612
cow	44.79	45.33	99.54	63.26	14.59	38.93	4222
coyote	69.19	63.72	35.17	55.63	75.72	65.68	22337
deer	74.95	69.86	64.2	69.45	70.32	69.88	24465
elk	63.21	53.34	6.36	42.46	76.87	59.67	25353
horse	52.22	47.54	33.88	42.02	62.34	52.18	4840
moose	57.1	52.86	57.77	50.45	55.46	52.96	25033
average	**61.07**	**57**	**55.57**	**57.05**	**57**	**57.03**	**163182**
RFCN							
	Class Specific			**mRP threshold 60.10 %**			
class	**AP(%)**	**RP(%)**	**Score(%)**	**Recall(%)**	**Precision(%)**	**Average(%)**	**GT**
bear	66.95	67.8	73.86	73.36	62.64	68	25320
bison	59.79	57.16	42.37	50.14	62.48	56.31	31612
cow	39.19	43.98	99.91	50.09	7.04	28.57	4222
coyote	57.38	52.65	16.13	35.78	86.97	61.38	22337
deer	68.22	65.28	72.45	69.97	59.87	64.92	24465
elk	56.99	52.48	21.35	42.43	72.3	57.36	25353
horse	48.14	43.39	94.48	59.15	24.95	42.05	4840
moose	48.24	47.53	54.37	45.48	49.81	47.64	25033
average	**55.61**	**53.78**	**59.36**	**53.3**	**53.26**	**53.28**	**163182**

Table B.3: Training configuration #1 class-specific results for NAS, INRES, RES101 and RFCN

Training Configuration #1: 1K/Seq/8							
RETINA							
class	**Class Specific**			**mRP threshold 41.20 %**			**GT**
	AP(%)	**RP(%)**	**Score(%)**	**Recall(%)**	**Precision(%)**	**Average(%)**	
bear	50.16	50.86	36.15	47.53	52.61	50.07	25320
bison	53.61	51.01	28.55	45.93	59.69	52.81	31612
cow	14.22	18.95	97.88	46.31	6.66	26.48	4222
coyote	43.94	42.57	15.13	32.99	71.96	52.47	22337
deer	47.7	47.82	53.13	53.83	42.61	48.22	24465
elk	47.46	46.71	17.82	36.56	63.72	50.14	25353
horse	36.16	38.24	84.68	51.01	14.08	32.55	4840
moose	25.73	27	36.93	24.89	27.53	26.21	25033
average	**39.87**	**40.4**	**46.28**	**42.38**	**42.36**	**42.37**	**163182**
SSDIN							
class	Class Specific			mRP threshold 11.30 %			**GT**
	AP(%)	RP(%)	Score(%)	Recall(%)	Precision(%)	Average(%)	
bear	48.2	49.42	3.06	38.59	58.53	48.56	25320
bison	39.95	46.49	10.54	45.95	46.85	46.4	31612
cow	28.56	32.64	99.6	69.23	7.09	38.16	4222
coyote	40.68	44.75	1.73	35.81	66.03	50.92	22337
deer	37.36	41.02	2.63	34.08	54.78	44.43	24465
elk	38.71	42.8	1.13	30.12	65.25	47.69	25353
horse	46.31	46.28	85.14	59.52	16.62	38.07	4840
moose	23.46	27.77	15.25	28.87	26.63	27.75	25033
average	**37.9**	**41.4**	**27.39**	**42.77**	**42.72**	**42.75**	**163182**
SSDMO							
class	**Class Specific**			**mRP threshold 27.30 %**			**GT**
	AP(%)	**RP(%)**	**Score(%)**	**Recall(%)**	**Precision(%)**	**Average(%)**	
bear	54.58	54.29	22.15	52.66	56.2	54.43	25320
bison	39.8	43.17	30.05	43.85	42.56	43.2	31612
cow	28.22	33.73	98.99	52.7	12.92	32.81	4222
coyote	42.28	43.4	3.9	35.56	57.13	46.35	22337
deer	38.14	41.86	65.72	49.43	36.83	43.13	24465
elk	40.22	41.73	1.18	27.58	73.55	50.56	25353
horse	38.47	38.04	96.24	49.09	20.16	34.62	4840
moose	26.45	29.56	7.31	24.41	35.58	29.99	25033
average	**38.52**	**40.72**	**40.69**	**41.91**	**41.87**	**41.89**	**163182**
YOLO							
class	**Class Specific**			**mRP threshold 0.50 %**			**GT**
	AP(%)	**RP(%)**	**Score(%)**	**Recall(%)**	**Precision(%)**	**Average(%)**	
bear	19.3	31.12	1.93	34.36	29.69	32.02	25320
bison	18.07	39.39	0.1	27.62	52.2	39.91	31612
cow	13.39	16.32	87.34	45.1	5.01	25.05	4222
coyote	17.57	37.74	0.1	22	58.65	40.33	22337
deer	28.71	44.69	2.69	50.23	40.31	45.27	24465
elk	27.98	38.62	0.1	33.72	45.72	39.72	25353
horse	17.16	22.71	91.92	50.87	7.89	29.38	4840
moose	13.65	30.63	0.1	20.56	41.89	31.22	25033
average	**19.48**	**32.65**	**23.03**	**35.56**	**35.17**	**35.36**	**163182**

Table B.4: Training configuration #1 class-specific results for RETINA, SSDIN, SSDMO and YOLO

Training Configuration #2: 1K/Even/8							
NAS							
class	**Class Specific**			**mRP threshold 83.10 %**			**GT**
	AP(%)	**RP(%)**	**Score(%)**	**Recall(%)**	**Precision(%)**	**Average(%)**	
bear	97.78	93.94	93.93	95.12	90.44	92.78	25268
bison	97.53	94.44	59.38	92.85	97.57	95.21	31294
cow	97.03	94.93	45.74	92.76	98.7	95.73	4340
coyote	96.96	91.61	90.7	93.11	87.62	90.36	22300
deer	97.89	93.54	66.8	91.72	95.9	93.81	24374
elk	98.5	95.17	86.24	95.51	94.66	95.08	25292
horse	97.82	95.71	91.63	96.45	92.95	94.7	4759
moose	98.36	95.15	84.48	95.25	94.91	95.08	24943
average	**97.73**	**94.31**	**77.36**	**94.1**	**94.09**	**94.09**	**162570**
INRES							
class	**Class Specific**			**mRP threshold 12.20 %**			**GT**
	AP(%)	**RP(%)**	**Score(%)**	**Recall(%)**	**Precision(%)**	**Average(%)**	
bear	97.37	96.37	3.03	95.7	99.05	97.37	25268
bison	92.13	90.73	3.37	90.31	96.07	93.19	31294
cow	98.62	98.23	20.62	98.32	96.3	97.31	4340
coyote	98.67	96.75	24.83	97.47	94.66	96.07	22300
deer	98.53	97.03	38.24	97.92	93.09	95.5	24374
elk	98.13	97.2	10.21	97.13	97.69	97.41	25292
horse	90.18	90.1	3.77	90	97.45	93.72	4759
moose	99.5	98.27	69.91	99.21	91.52	95.36	24943
average	**96.64**	**95.59**	**21.75**	**95.76**	**95.73**	**95.74**	**162570**
RES101							
class	**Class Specific**			**mRP threshold 90.80 %**			**GT**
	AP(%)	**RP(%)**	**Score(%)**	**Recall(%)**	**Precision(%)**	**Average(%)**	
bear	99.72	99.53	88.64	99.52	99.59	99.55	25268
bison	99.79	99.42	86.91	99.35	99.53	99.44	31294
cow	99.86	99.65	89.18	99.65	99.72	99.69	4340
coyote	99.92	99.34	92.52	99.39	99.31	99.35	22300
deer	99.87	99.5	96.2	99.57	99.39	99.48	24374
elk	99.94	99.84	75.47	99.81	99.92	99.86	25292
horse	99.77	99.62	98.57	99.66	99.23	99.44	4759
moose	99.91	99.6	75.28	99.49	99.75	99.62	24943
average	**99.85**	**99.56**	**87.85**	**99.56**	**99.55**	**99.56**	**162570**
RFCN							
class	**Class Specific**			**mRP threshold 84.30 %**			**GT**
	AP(%)	**RP(%)**	**Score(%)**	**Recall(%)**	**Precision(%)**	**Average(%)**	
bear	99.65	99.44	79.47	99.39	99.58	99.49	25268
bison	99.63	99.09	79.33	98.99	99.23	99.11	31294
cow	99.88	99.72	49.54	99.65	99.86	99.76	4340
coyote	99.95	99.48	89.86	99.57	99.31	99.44	22300
deer	99.86	99.61	73.09	99.48	99.71	99.6	24374
elk	99.86	99.75	91.57	99.79	99.53	99.66	25292
horse	99.74	99.58	95.96	99.66	99.23	99.44	4759
moose	99.86	99.56	81.89	99.52	99.59	99.56	24943
average	**99.8**	**99.53**	**80.09**	**99.51**	**99.5**	**99.51**	**162570**

Table B.5: Training configuration #2 class-specific results for NAS, INRES, RES101 and RFCN

Training Configuration #2: 1K/Even/8							
RETINA							
class	**Class Specific**			**mRP threshold 28.30 %**			**GT**
	AP(%)	**RP(%)**	**Score(%)**	**Recall(%)**	**Precision(%)**	**Average(%)**	
bear	99.88	99.66	29.67	99.68	99.64	99.66	25268
bison	99.91	99.34	25.21	99.13	99.44	99.28	31294
cow	100	99.82	52.52	99.95	99.54	99.75	4340
coyote	99.99	99.64	42.03	99.96	99.39	99.67	22300
deer	99.98	99.67	30.56	99.72	99.64	99.68	24374
elk	99.99	99.8	38.48	99.89	99.75	99.82	25292
horse	99.89	99.41	26.15	98.76	99.6	99.18	4759
moose	99.96	99.72	27.8	99.7	99.74	99.72	24943
average	**99.95**	**99.63**	**34.05**	**99.6**	**99.59**	**99.6**	**162570**
SSDIN							
class	**Class Specific**			**mRP threshold 22.30 %**			**GT**
	AP(%)	**RP(%)**	**Score(%)**	**Recall(%)**	**Precision(%)**	**Average(%)**	
bear	99.63	98.36	34.83	98.97	97.3	98.14	25268
bison	99.79	99.18	13.43	98.87	99.6	99.23	31294
cow	99.82	98.43	27.94	98.71	97.92	98.31	4340
coyote	99.65	97.54	32.92	98.37	96.24	97.3	22300
deer	99.88	98.89	13.25	98.23	99.46	98.84	24374
elk	99.96	99.36	13.82	99.1	99.79	99.45	25292
horse	99.78	98.72	19.4	98.59	99.09	98.84	4759
moose	99.76	98.55	13.25	97.67	99.03	98.35	24943
average	**99.78**	**98.63**	**21.1**	**98.56**	**98.55**	**98.56**	**162570**
SSDMO							
class	**Class Specific**			**mRP threshold 34.10 %**			**GT**
	AP(%)	**RP(%)**	**Score(%)**	**Recall(%)**	**Precision(%)**	**Average(%)**	
bear	99.64	98.56	19.39	97.7	99.26	98.48	25268
bison	99.72	98.37	49.9	98.83	97.1	97.96	31294
cow	99.88	98.55	46.9	99.24	96.55	97.89	4340
coyote	99.87	98.52	35.81	98.67	98.41	98.54	22300
deer	99.9	98.81	27.96	98.35	99.11	98.73	24374
elk	99.93	99.09	19.37	98.49	99.66	99.07	25292
horse	99.69	99.01	23.16	97.48	99.34	98.41	4759
moose	99.73	97.47	37.34	97.85	97.18	97.52	24943
average	**99.79**	**98.55**	**32.48**	**98.33**	**98.32**	**98.33**	**162570**
YOLO							
class	**Class Specific**			**mRP threshold 6.80 %**			**GT**
	AP(%)	**RP(%)**	**Score(%)**	**Recall(%)**	**Precision(%)**	**Average(%)**	
bear	98.92	96.74	11.85	97.33	94.12	95.72	25268
bison	98.18	96.17	5.81	95.98	96.56	96.27	31294
cow	98.56	95.18	7.31	95.35	94.84	95.09	4340
coyote	96.97	94.57	5.5	94.24	95.37	94.8	22300
deer	99.04	96.04	14.66	96.94	94.09	95.52	24374
elk	97.97	97.75	0.38	97.22	99.97	98.59	25292
horse	97.5	95.15	5.73	94.94	96.39	95.67	4759
moose	99.85	99.22	3.63	98.96	99.58	99.27	24943
average	**98.37**	**96.35**	**6.86**	**96.37**	**96.37**	**96.37**	**162570**

Table B.6: Training configuration #2 class-specific results for RETINA, SSDIN, SSDMO and YOLO

Appendix C

Human iPSC Segmentation

C.1 Datasets

Numerical details of the annotated data generated by the first and last stages of the annotation process are given in Tables C.1 and C.2 respectively.

Table C.1: Numerical details of the selective uncategorized labeled data generated by the first stage of the annotation process.

Frame Range	**Sequences**	**Frames**	**Cells**
146– 200	3	130	948
201 - 250	8	377	2952
251 - 275	6	149	868
146 - 275	17	656	4768

Table C.2: Numerical details of the categorized retrospective labeled data generated by the last stage of the annotation process.

	Train			**Test**			**All**		
Frames	**Start**	**End**	**Total**	**Start**	**End**	**Total**	**Start**	**End**	**Total**
Early Train	163	202	1178	146	162	496	146	275	3937
Late Train	203	275	2263						
Cells	**iPSC**	**Diff**	**Total**	**iPSC**	**Diff**	**Total**	**iPSC**	**Diff**	**Total**
Early Train	1879	5380	7259	790	2467	3257	6056	16542	22598
Late Train	3387	8695	12082						

Table C.3: Quantitative details of image sizes and number of objects per image in the 31 ROI sequences

sequence	height	width	objects per image		
			mean	min	max
overall			5.7	1	20
10127_9782_12527_11782	2000	2400	6.4	5	9
10161_9883_13561_12050	2167	3400	7.9	6	11
11927_12517_15394_15550	3033	3467	14.6	11	19
12094_17082_16427_20915	3833	4333	13.1	6	19
12527_11015_14493_12615	1600	1966	6.1	4	8
12794_8282_14661_10116	1834	1867	6.4	5	8
12994_10915_15494_12548	1633	2500	5.3	4	6
16627_11116_18727_12582	1466	2100	5.0	5	6
7777_10249_10111_13349	3100	2334	5.9	3	7
8094_13016_11228_15282	2266	3134	6.1	4	7
9861_9849_12861_11516	1667	3000	7.1	6	9
10228_10182_12394_11915	1733	2166	3.8	3	6
10494_8849_12494_9849	1000	2000	3.8	2	5
11661_13082_13594_14849	1767	1933	8.3	5	11
12394_17282_14327_20782	3500	1933	7.5	4	13
12761_10682_14894_11782	1100	2133	4.5	3	7
12861_8815_15027_10115	1300	2166	6.4	5	8
12961_11916_14661_12816	900	1700	1.2	1	2
13894_13749_16527_15316	1567	2633	6.1	5	8
14094_17682_15894_19749	2067	1800	9.7	4	20
15827_11316_17627_12749	1433	1800	5.5	5	7
15927_17249_17627_19582	2333	1700	4.5	1	9
17094_13782_19127_16348	2566	2033	8.5	6	11
17861_11316_19661_12616	1300	1800	2.7	2	4
4961_15682_7127_16949	1267	2166	4.3	3	7
6661_13749_9061_14816	1067	2400	4.6	3	6
7594_11916_9927_13149	1233	2333	2.0	2	2
7694_8682_10194_9682	1000	2500	1.7	1	2
7727_10749_9961_11749	1000	2234	4.0	2	5
8461_17782_10194_19016	1234	1733	1.7	1	2
9261_13449_11494_14382	933	2233	3.0	1	4

C.2 XGBoost Features

XGBoost classifier was trained with the following features:

- area: number of pixels enclosed by the cell
- pixel-min: maximum of the pixel values enclosed by the cell
- pixel-max: minimum of the pixel values enclosed by the cell
- pixel-std: standard deviation of the pixel values enclosed by the cell
- circlicity: ratio of the number of pixels in the circumcircle of the cell to that in the cell itself.
- speed: $\sqrt{dx^2 + dy^2}$ where (dx, dy) is the change in location of the cell centroid from the previous frame
- displacement: $\sqrt{init\_dx^2 + init\_dy^2}$ where $(init\_dx, init\_dy)$ is the change in the location of the cell centroid from the first frame

C.3 Tracking Algorithm

A simplified high-level algorithm describing the interactive semi-automated tracking process used for retrospective labelling is given in Algorithm 2.

C.4 Temporal Detection Metrics Results

Figure C.1 shows the impact of subsequence length on the detection metrics of RP-AUC and AP. Similar to ROC-AUC, there is a general trend of performance improvement with increase in subsequence length though not as strong as ROC-AUC and even an inverse trend being apparent in several cases.

C.5 Visualization Videos

All visualization videos are available in this Google Drive folder:
https:
//drive.google.com/drive/folders/1L1NXhSQvLpRSN4WBmbiv9lZYYP3zYgVS.

There are four types of videos, each in its own subfolder:

Algorithm 2 Human-Assisted Tracking for Retrospective Labeling

prev_tracks ← ∅
for *frame* **in** ROI sequence from last to first **do**
    curr_dets ← cells detected in current frame by the instance segmentation model
    prev_tracks ← tracked cells from previous frame if any else ∅
    curr_tracks ← ∅
    for each pair of cells in *curr_dets* with IOU > 0.5 **do**
        Ask user to choose the duplicate cell from the pair if any and discard all such cells
    end for
    Perform IOU-based pairwise association between *prev_tracks* and *curr_dets*

    for each unassociated cell in *curr_dets* **do**
        Ask user to decide if it is a false positive and discard it if so
    end for
    for each unassociated track in *prev_tracks* **do**
        Ask user to associate it with a cell in *curr_dets* or discard it as having ended
    end for
    for each pair of associated cells with a relative change in height, width or area > 0.5 **do**
        Ask user to choose if this ambiguous association represents cell division, fusion or misassociation and update tracks accordingly
    end for
    Extend each associated track in *prev_tracks* with the corresponding cell from *curr_dets* and add the extended track to *curr_tracks*
    Add a new track to *curr_tracks* for each remaining unassociated cell in *curr_dets*
    prev_tracks ← *curr_tracks*
end for

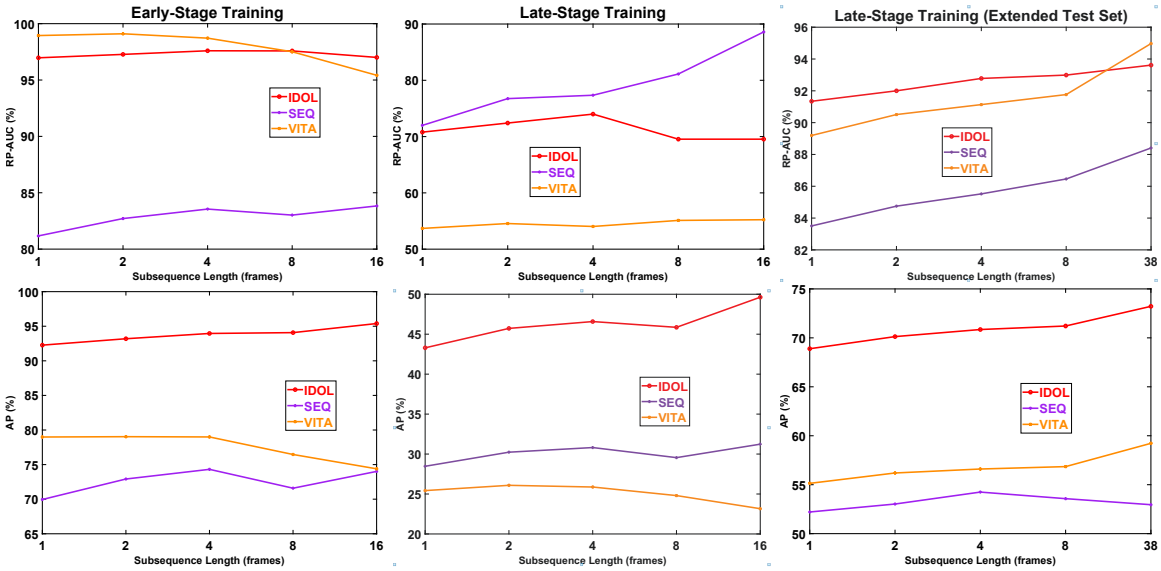

Figure C.1: Subsequential results for (top) RP-AUC and (bottom) AP. Left and center plots show early and late-stage models tested on the standard test set of frames 146 - 162 while the right one shows the latter tested on an extended test set with frames 146 - 201. Note the curtailed and variable Y-axis ranges.

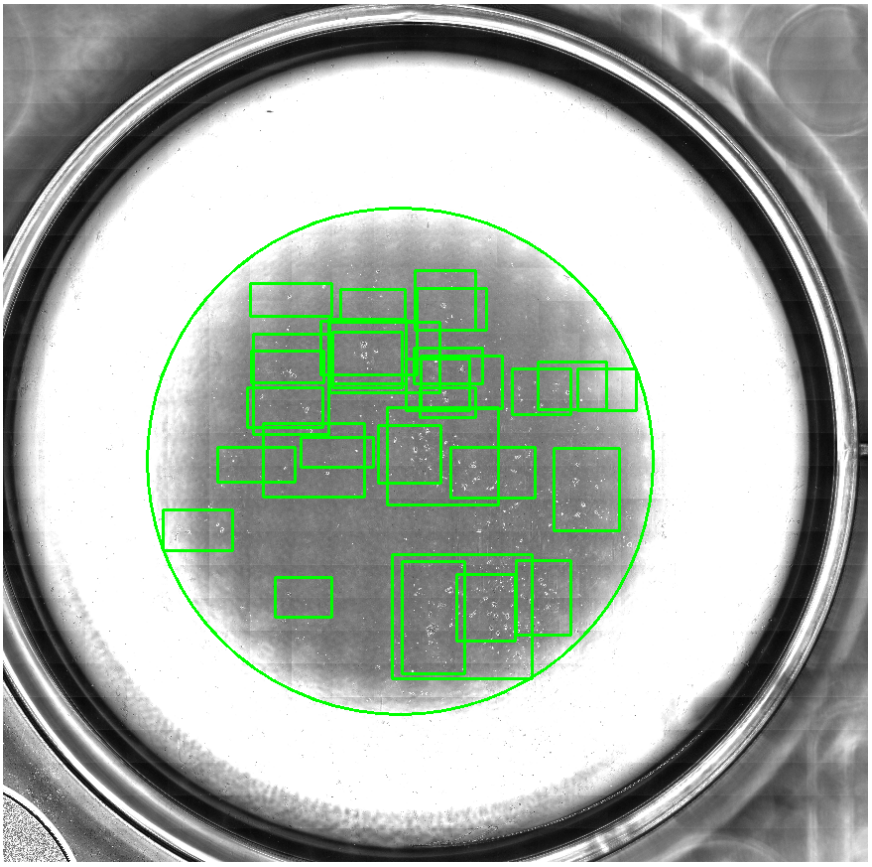

Figure C.2: A sample 714 MP raw image with the 31 ROIs shown in green bounding boxes.

- detection failures: 2 FPS videos showing all 6 types of detection failures for each detector in both early and late-stage training configurations
  - FP videos contain one frame for each detection failure so that each source frame is present multiple times if it contains multiple FPs
  - FN videos show all the missing detections for each source frame together so each source frame is present only once
  - each frame contains three images showing, from left to right, GT annotations, all detections in the frame and a single FP detection or GT cells corresponding to all the FNs
  - in case of FP-NEX-PART, the third frame shows both the FP detection and the GT cell with which it partially overlaps
  - first two images show iPSCs and DfCs in green and red respectively while the third image shows FPs in magenta and GTs in cyan

  - black bar at the bottom of each frame shows the ROI and frame info in the first line while the second line shows some stats that can be ignored
- detection results: 2 FPS videos showing the GT and detected cells for all source frames in each of the 31 ROIs
  - iPSCs and DfCs are shown in green and red respectively
- retrospective labeling: 2 FPS videos showing the result of retrospective labelling for each of the 31 ROIs
  - frames are ordered backwards in time since that is how the tracking was done
  - the mask colour of each cell indicates its unique identity while the bounding box colour indicates its class – green for iPSC and red for DfC
- cell lines: a single 5 FPS video showing the 8 cell lines generated by backwards tracking of a single target in one of the ROIs
  - cell lines are arranged in a 2×4 grid
  - frames are ordered backwards in time
  - a cell line is considered to have diverged into two by a cell division event while a fusion event terminates both parent cell lines and creates a new one
  - multiple cell lines have many frames in common but each has at least one frame with a unique cell-identity that makes this line distinct from others

All videos are formatted as MPEG-4 with mp4v codec and should be playable on any video player on Linux, Windows or Mac.

C.6 3D Interactive Plots

Frame-wise partial AUC plots are available as interactive HTML files in this Google Drive folder. These can be opened in any web browser.

Appendix D

Language Modeling for Video Detection

D.1 Hierarchical Video Cross-MHA

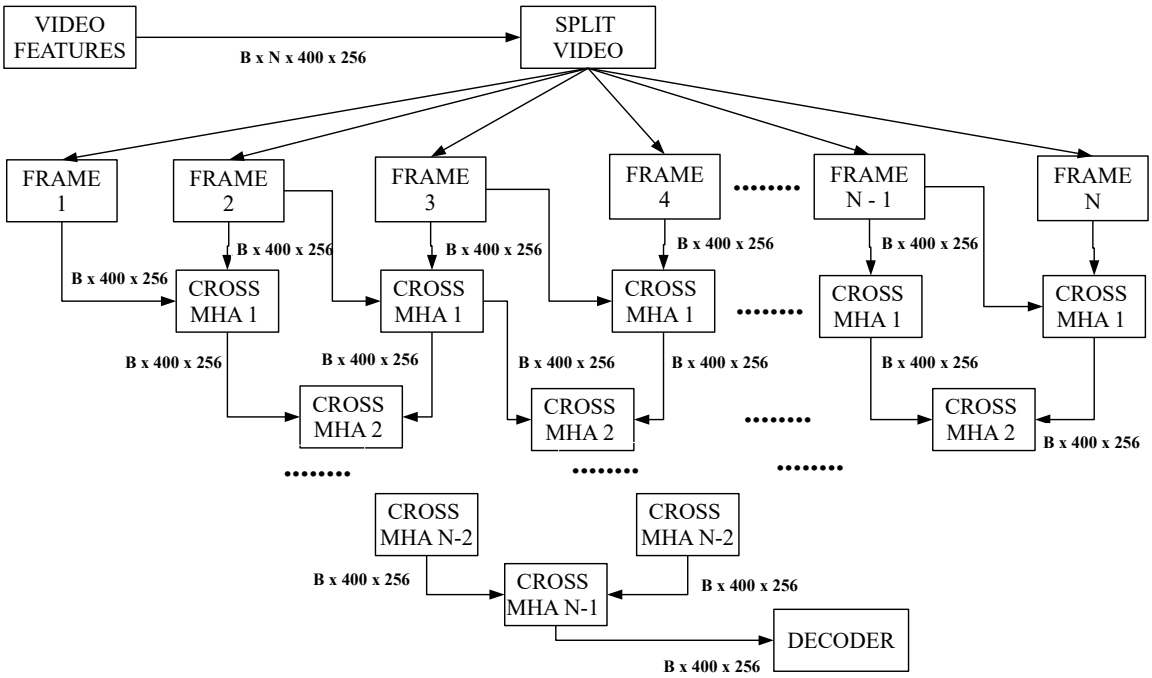

Figure D.1: Flow diagram for the hierarchical variant of the cross-MHA module in the middle-fusion video encoder.

This appendix provides an alternative way to perform the video cross-MHA operation in the middle-fusion video encoder. As shown in fig. D.1, we first apply

cross-MHA between pairs of consecutive frames, i.e. (F_1, F_2), (F_2, F_3), (F_3, F_4) and so on to obtain $N-1$ feature maps, each of size 400×256, corresponding to the $N-1$ frame-pairs. Consecutive pairs from these $N-1$ feature maps are then cross-attended in a second level cross-MHA operation to yield $N-2$ feature maps, again with the same 400×256 size. This process is repeated for a total of $N-1$ levels of cross-MHA operations to finally yield a single 400×256 feature map that is passed to the decoder.

Appendix E

Language Modeling for Semantic Segmentation

E.1 RLE length Statistics

This section provides statistics about the RLE length (L) required to represent masks from the complete IPSC dataset for various combinations of image size I, patch size P and mask size S for both static (Tables E.1 - E.2) and video (Tables E.3 - E.4) segmentation. The first two rows in each case show the mean and maximum RLE lengths while the remaining rows show the percentage of images for which the lengths exceed the threshold in the first column. The smallest threshold that can be used as L in each case is highlighted in yellow. I have found empirically that small fraction of lengths ($< 2\%$) exceeding L does not have a significant impact on the segmentation results so that L does not necessarily need to be $\geq$ maximum length. Binary and multi-class LAC both use 2 tokens per run so ended up having identical lengths for all images over this dataset. Note that this does not always have to be true in principle. For example, if two runs with different classes are next to each other in the same row, they will be represented by a single run in the binary case but two different runs in the multi-class case, but such scenarios do not occur in the IPSC dataset. Multi-class masks with separate class tokens use 3 tokens per run so have 1.5 times the lengths of the other two cases.

Table E.1: Statistics of static mask RLE lengths over the IPSC dataset with images resized from $I = 1280$ to $I = 80$, without patches and mask subsampling (so that $S = P = I$). Please refer Section E.1 for more details.

Image size (I)	1280	640	320	160	80
Binary and Multi-Class LAC					
Mean	2363.3	1185.9	597.2	302.6	154.8
Max	5426	2716	1362	684	348
> 512	96.85%	89.71%	58.75%	8.46%	0.00%
> 1024	89.61%	58.24%	8.15%	0.00%	0.00%
> 1536	77.32%	24.64%	0.00%	0.00%	0.00%
> 2048	57.94%	7.87%	0.00%	0.00%	0.00%
> 3072	24.28%	0.00%	0.00%	0.00%	0.00%
> 4096	7.85%	0.00%	0.00%	0.00%	0.00%
> 5120	0.43%	0.00%	0.00%	0.00%	0.00%
Multi-Class with separate class tokens					
Mean	3545.2	1778.9	896	454	232.4
Max	8139	4074	2043	1026	522
> 512	99.03%	94.74%	82.83%	36.50%	0.10%
> 1024	94.74%	82.45%	35.38%	0.03%	0.00%
> 1536	89.61%	58.24%	8.15%	0.00%	0.00%
> 2048	82.30%	34.90%	0.00%	0.00%	0.00%
> 3072	57.94%	7.87%	0.00%	0.00%	0.00%
> 4096	34.65%	0.00%	0.00%	0.00%	0.00%
> 5120	16.38%	0.00%	0.00%	0.00%	0.00%
> 6144	7.85%	0.00%	0.00%	0.00%	0.00%
> 7168	1.50%	0.00%	0.00%	0.00%	0.00%

E.2 Vocabulary Sizes

This section provides tables of vocabulary sizes for $S = 40$ (Table E.5), $S = 80$ (Table E.6), and $S = 160$ (Table E.8) for video segmentation with varying N and all the encoding strategies proposed in Section 7.3. These tables demonstrate the exponential increase in the number of TAC tokens with N. We can also see that, for smaller mask sizes (e.g. $S = 40$), straightforward 3D mask flattening remains more practicable for higher values of N due to its linear increase in V with N. Note that these tables only account for limits on V and not those on L, which would likely limit N to lower values depending on the complexity of the dataset.

Table E.2: Statistics of static mask RLE lengths over the entire IPSC dataset with (top left) $I = P = 640$, (top right) $I = P = 1024$, (bottom left) $I = 2560, P = 640$, and (bottom right) $I = 2560, P = 1024$. Please refer Section E.1 for more details.

Image size (I)	640	640	640	640
Subsampling	1	2	4	8
Mask Size (S)	640	320	160	80
Binary and Multi-Class LAC				
MEAN	1185.9	597.2	302.6	154.8
MAX	2716	1362	684	348
> 512	89.71%	58.75%	8.46%	0.00%
> 1024	58.24%	8.15%	0.00%	0.00%
> 1536	24.64%	0.00%	0.00%	0.00%
> 2048	7.87%	0.00%	0.00%	0.00%
> 3072	0.00%	0.00%	0.00%	0.00%
Multi-Class with separate class tokens				
MEAN	1778.5	895.8	453.9	232.3
MAX	4074	2043	1026	522
> 512	94.72%	82.81%	36.49%	0.10%
> 1024	82.43%	35.37%	0.03%	0.00%
> 1536	58.23%	8.15%	0.00%	0.00%
> 2048	34.89%	0.00%	0.00%	0.00%
> 3072	7.87%	0.00%	0.00%	0.00%
> 4096	0.00%	0.00%	0.00%	0.00%

Image size (I)	1024	1024	1024	1024
Subsampling	1	2	4	8
Mask Size (S)	1024	512	256	128
Binary and Multi-Class LAC				
MEAN	1892.4	950.4	479.4	243.5
MAX	4342	2176	1092	548
> 512	95.22%	84.68%	42.06%	0.71%
> 1024	84.56%	41.30%	0.53%	0.00%
> 1536	62.05%	10.08%	0.00%	0.00%
> 2048	40.94%	0.48%	0.00%	0.00%
> 3072	9.93%	0.00%	0.00%	0.00%
> 4096	0.46%	0.00%	0.00%	0.00%
Multi-Class with separate class tokens				
MEAN	2836.4	1424.7	718.7	365.2
MAX	6513	3264	1638	822
> 512	98.20%	92.39%	70.58%	18.48%
> 1024	92.28%	69.34%	17.28%	0.00%
> 1536	84.49%	41.27%	0.56%	0.00%
> 2048	68.93%	16.75%	0.00%	0.00%
> 3072	40.94%	0.48%	0.00%	0.00%
> 4096	16.38%	0.00%	0.00%	0.00%
> 5120	5.92%	0.00%	0.00%	0.00%
> 6144	0.46%	0.00%	0.00%	0.00%

Image size (I)	2560	2560	2560	2560
Patch size (P)	640	640	640	640
Subsampling	1	2	4	8
Mask Size (S)	640	320	160	80
Binary and Multi-Class LAC				
MEAN	714.4	357.6	179.1	89.9
MAX	2666	1332	642	318
> 512	60.62%	27.05%	0.57%	0.00%
> 1024	27.05%	0.66%	0.00%	0.00%
> 1536	3.20%	0.00%	0.00%	0.00%
> 2048	0.66%	0.00%	0.00%	0.00%
Multi-Class with separate class tokens				
MEAN	1075.4	538.5	269.9	135.8
MAX	3999	1998	963	477
> 512	74.93%	47.96%	5.43%	0.00%
> 1024	47.77%	5.49%	0.00%	0.00%
> 1536	27.17%	0.68%	0.00%	0.00%
> 2048	5.53%	0.00%	0.00%	0.00%
> 3072	0.67%	0.00%	0.00%	0.00%

Image size (I)	2560	2560	2560	2560
Patch size (P)	1024	1024	1024	1024
Subsampling	1	2	4	8
Mask Size (S)	1024	512	256	128
Binary and Multi-Class LAC				
MEAN	1177.9	593.2	295.4	148.5
MAX	5392	2694	1348	676
> 512	82.57%	53.31%	9.95%	0.16%
> 1024	52.96%	10.54%	0.16%	0.00%
> 1536	29.17%	1.10%	0.00%	0.00%
> 2048	10.02%	0.15%	0.00%	0.00%
> 3072	0.97%	0.00%	0.00%	0.00%
Multi-Class with separate class tokens				
MEAN	1776.9	889.7	446.1	224.4
MAX	8088	4041	2022	1014
> 512	90.72%	72.81%	36.82%	2.22%
> 1024	72.66%	36.46%	2.21%	0.00%
> 1536	53.17%	10.54%	0.16%	0.00%
> 2048	36.37%	2.30%	0.00%	0.00%
> 3072	10.51%	0.15%	0.00%	0.00%
> 4096	2.34%	0.00%	0.00%	0.00%
> 5120	0.62%	0.00%	0.00%	0.00%

Table E.3: Statistics of video mask RLE lengths over the IPSC dataset with $N = 2$, $I = 2560$, $P = 640$ and $S = 80, 160$. Please refer Section E.1 for more details.

I=2560 P=640	Binary				*N=2*	Multi-Class			
	3D-C	3D-F	TAC	LTAC		3D-C	LAC	TAC	LTAC
AVG	154.9	189.2	231.8	154.5	*S=80*	233.9	155.4	232.5	154.7
MAX	564	1310	954	636		846	564	954	636
> 512	0.12%	2.82%	5.55%	0.24%		5.08%	0.26%	5.63%	0.25%
> 1024	0.00%	0.15%	0.00%	0.00%		0.00%	0.00%	0.00%	0.00%
AVG	307.3	670	511	341.2	*S=160*	461.9	309	511.4	341.4
MAX	1116	5028	1971	1314		1674	1116	1971	1314
> 512	17.76%	52.65%	42.89%	22.12%		37.19%	18.40%	42.89%	22.17%
> 1024	0.18%	20.07%	8.53%	1.07%		4.97%	0.26%	8.55%	1.13%
> 1536	0.00%	6.56%	1.07%	0.00%		0.26%	0.00%	1.13%	0.00%
> 2048	0.00%	2.19%	0.00%	0.00%		0.00%	0.00%	0.00%	0.00%

Table E.4: Statistics of video mask RLE lengths over the IPSC dataset with $I = 2560$, $P = 640$, $S = 80$ and $N = 3, 4, 6, 8, 9$. Please refer Section E.1 for more details.

I=2560 P=640	Binary			*S = 80*	Multi-Class		
	3D-C	TAC	LTAC		LAC	TAC	LTAC
AVG	231.5	329.3	219.6	*N=3*	231.2	329.6	219.4
MAX	816	1419	946		836	1419	946
> 512	4.84%	20.54%	4.43%		4.93%	20.60%	4.51%
> 1024	0.00%	0.61%	0.00%		0	0.61%	0
AVG	306.8	417.1	278.0	*N=4*	307.8	417.3	278.2
MAX	1066	1779	1186		1114	1779	1186
> 512	18.35%	33.43%	12.55%		18.49%	33.47%	12.55%
> 1024	0.08%	3.71%	0.03%		0.21%	3.73%	0.03%
>1536	0.00%	0.03%	0.00%		0.00%	0.03%	0.00%
AVG	455.0	575.1	383.4	*N=6*	456.5	575.3	383.5
MAX	1566	2262	1508		1662	2262	1508
> 512	36.78%	48.77%	28.60%		36.78%	48.77%	28.64%
> 1024	4.59%	14.23%	2.23%		4.73%	14.27%	2.23%
> 1536	0.05%	2.23%	0.00%		0.20%	2.23%	0.00%
> 2048	0.00%	0.10%	0.00%		0.00%	0.10%	0.00%
AVG	599.5	718.2	478.8	*N=8*	601.4	718.3	478.9
MAX	2064	2826	1884		2204	2826	1884
> 512	49.62%	58.49%	40.25%		49.62%	58.49%	40.26%
> 1024	17.77%	25.68%	7.80%		17.92%	25.69%	7.82%
> 1536	2.83%	7.80%	0.61%		2.88%	7.82%	0.61%
> 2048	0.05%	1.55%	0.00%		0.17%	1.55%	0.00%
AVG	670.5	784.3	522.9	*N=9*	672.5	784.5	523.0
MAX	2310	3087	2058		2470	3087	2058
> 512	53.86%	62.09%	44.67%		53.86%	62.09%	44.67%
> 1024	23.52%	29.68%	11.04%		23.52%	29.72%	11.04%
> 1536	4.44%	11.04%	1.30%		4.57%	11.04%	1.30%
> 2048	1.03%	2.64%	0.02%		1.18%	2.64%	0.02%
> 3072	0.00%	0.02%	0.00%		0.00%	0.02%	0.00%

Table E.5: Vocabulary sizes (V) for video segmentation with $S = 40$, varying values of N from $N = 2$ to $N = 20$ and both binary and multiclass cases. The maximum possible N for each case such that $V < 32K$ is highligted in yellow. V is shown in units of thousands.

Binary								S=40	Multi-Class							
N	S	C+1	encoding	starts tokens	lengths tokens	class tokens	V (K)		N	S	C+1	encoding	starts tokens	lengths tokens	class tokens	V (K)
2	40	2	3D-C	3200	40		3.24	N=2	2	40	3	3D-C	3200	40	2	3.24
2	40	2	3D-F	3200	80		3.28		2	40	3	LAC	3200		80	3.28
2	40	2	TAC	1600	40	3	1.64		2	40	3	TAC	1600	40	8	1.65
2	40	2	LTAC	1600		120	1.72		2	40	3	LTAC	1600		320	1.92
3	40	2	3D-C	4800	40		4.84	N=3	3	40	3	3D-C	4800	40	2	4.84
3	40	2	3D-F	4800	120		4.92		3	40	3	LAC	4800		80	4.88
3	40	2	TAC	1600	40	7	1.65		3	40	3	TAC	1600	40	26	1.67
3	40	2	LTAC	1600		280	1.88		3	40	3	LTAC	1600		1040	2.64
4	40	2	3D-C	6400	40		6.44	N=4	4	40	3	3D-C	6400	40	2	6.44
4	40	2	3D-F	6400	160		6.56		4	40	3	LAC	6400		80	6.48
4	40	2	TAC	1600	40	15	1.66		4	40	3	TAC	1600	40	80	1.72
4	40	2	LTAC	1600		600	2.2		4	40	3	LTAC	1600		3200	4.8
6	40	2	3D-C	9600	40		9.64	N=6	6	40	3	3D-C	9600	40	2	9.64
6	40	2	3D-F	9600	240		9.84		6	40	3	LAC	9600		80	9.68
6	40	2	TAC	1600	40	63	1.7		6	40	3	TAC	1600	40	728	2.37
6	40	2	LTAC	1600		2520	4.12		6	40	3	LTAC	1600		29120	30.72
8	40	2	3D-C	12800	40		12.84	N=8	8	40	3	3D-C	12800	40	2	12.84
8	40	2	3D-F	12800	320		13.12		8	40	3	LAC	12800		80	12.88
8	40	2	TAC	1600	40	255	1.9		8	40	3	TAC	1600	40	6560	8.2
8	40	2	LTAC	1600		10200	11.8		8	40	3	LTAC	1600		262400	264
9	40	2	3D-C	14400	40		14.44	N=9	9	40	3	3D-C	14400	40	2	14.44
9	40	2	3D-F	14400	360		14.76		9	40	3	LAC	14400		80	14.48
9	40	2	TAC	1600	40	511	2.15		9	40	3	TAC	1600	40	19682	21.32
9	40	2	LTAC	1600		20440	22.04		9	40	3	LTAC	1600		787280	788.88
12	40	2	3D-C	19200	40		19.24	N=12	12	40	3	3D-C	19200	40	2	19.24
12	40	2	3D-F	19200	480		19.68		12	40	3	LAC	19200		80	19.28
12	40	2	TAC	1600	40	4095	5.74		12	40	3	TAC	1600	40	531440	533.08
12	40	2	LTAC	1600		163800	165.4		12	40	3	LTAC	1600		2.1E+07	21259
14	40	2	3D-C	22400	40		22.44	N=14	14	40	3	3D-C	22400	40	2	22.44
14	40	2	3D-F	22400	560		22.96		14	40	3	LAC	22400		80	22.48
14	40	2	TAC	1600	40	16383	18.02		14	40	3	TAC	1600	40	4782968	4784.6
14	40	2	LTAC	1600		655320	656.92		14	40	3	LTAC	1600		1.9E+08	191320
16	40	2	3D-C	25600	40		25.64	N=16	16	40	3	3D-C	25600	40	2	25.64
16	40	2	3D-F	25600	640		26.24		16	40	3	LAC	25600		80	25.68
16	40	2	TAC	1600	40	65535	67.18		16	40	3	TAC	1600	40	4.3E+07	43048
16	40	2	LTAC	1600		2621400	2623		16	40	3	LTAC	1600		1.7E+09	2E+06
18	**40**	**2**	3D-C	**28800**	**40**		28.84	N=18	**18**	**40**	**3**	3D-C	**28800**	**40**	**2**	28.84
18	40	2	3D-F	28800	720		29.52		18	40	3	LAC	28800		80	28.88
18	40	2	TAC	1600	40	262143	263.78		18	40	3	TAC	1600	40	3.9E+08	387422
18	40	2	LTAC	1600		1E+07	10487		18	40	3	LTAC	1600		1.5E+10	2E+07
20	40	2	3D-C	32000	40		32.04	N=20	20	40	3	3D-C	32000	40	2	32.04
20	40	2	3D-F	32000	800		32.8		20	40	3	LAC	32000		80	32.08
20	40	2	TAC	1600	40	1048575	1050.2		20	40	3	TAC	1600	40	3.5E+09	3E+06
20	40	2	LTAC	1600		4.2E+07	41945		20	40	3	LTAC	1600		1.4E+11	1E+08

Table E.6: Vocabulary sizes (V) for video segmentation with $S = 80$, varying values of N from $N = 2$ to $N = 16$ and both binary and multiclass cases. The maximum possible N for each case such that $V < 32K$ is highligted in yellow. V is shown in units of thousands.

Binary								S=80	Multi-Class							
N	*S*	*C+1*	encoding	*starts* tokens	*lengths* tokens	class tokens	*V* (K)		*N*	*S*	*C+1*	encoding	*starts* tokens	*lengths* tokens	class tokens	*V* (K)
2	80	2	3D-C	12800	80		**12.88**		2	80	3	3D-C	12800	80	2	**12.88**
2	80	2	3D-F	12800	160		**12.96**	*N=2*	2	80	3	LAC	12800		160	**12.96**
2	80	2	TAC	6400	80	3	**6.48**		2	80	3	TAC	6400	80	8	**6.49**
2	80	2	LTAC	6400		240	**6.64**		2	80	3	LTAC	6400		640	**7.04**
3	80	2	3D-C	19200	80		**19.28**		3	80	3	3D-C	19200	80	2	**19.28**
3	80	2	3D-F	19200	240		**19.44**	*N=3*	3	80	3	LAC	19200		160	**19.36**
3	80	2	TAC	6400	80	7	**6.49**		3	80	3	TAC	6400	80	26	**6.51**
3	80	2	LTAC	6400		560	**6.96**		3	80	3	LTAC	6400		2080	**8.48**
4	80	2	3D-C	25600	80		**25.68**		4	80	3	3D-C	25600	80	2	**25.68**
4	80	2	3D-F	25600	320		**25.92**	*N=4*	4	80	3	LAC	25600		160	**25.76**
4	80	2	TAC	6400	80	15	**6.5**		4	80	3	TAC	6400	80	80	**6.56**
4	80	2	LTAC	6400		1200	**7.6**		4	80	3	LTAC	6400		6400	**12.8**
6	80	2	3D-C	38400	80		**38.48**		6	80	3	3D-C	38400	80	2	**38.48**
6	80	2	3D-F	38400	480		**38.88**	*N=6*	6	80	3	LAC	38400		160	**38.56**
6	80	2	TAC	6400	80	63	**6.54**		6	80	3	TAC	6400	80	728	**7.21**
6	80	2	LTAC	6400		5040	**11.44**		6	80	3	LTAC	6400		58240	**64.64**
8	80	2	3D-C	51200	80		**51.28**		8	80	3	3D-C	51200	80	2	**51.28**
8	80	2	3D-F	51200	640		**51.84**	*N=8*	8	80	3	LAC	51200		160	**51.36**
8	80	2	TAC	6400	80	255	**6.74**		8	80	3	TAC	6400	80	6560	**13.04**
8	80	2	LTAC	6400		20400	**26.8**		8	80	3	LTAC	6400		524800	**531.2**
9	80	2	3D-C	57600	80		**57.68**		9	80	3	3D-C	57600	80	2	**57.68**
9	80	2	3D-F	57600	720		**58.32**	*N=9*	9	80	3	LAC	57600		160	**57.76**
9	80	2	TAC	6400	80	511	**6.99**		**9**	**80**	**3**	**TAC**	**6400**	**80**	**19682**	**26.16**
9	80	2	LTAC	6400		40880	**47.28**		9	80	3	LTAC	6400		1574560	**1581**
10	80	2	3D-C	64000	80		**64.08**		10	80	3	3D-C	64000	80	2	**64.08**
10	80	2	3D-F	64000	800		**64.8**	*N=10*	10	80	3	LAC	64000		160	**64.16**
10	80	2	TAC	6400	80	1023	**7.5**		10	80	3	TAC	6400	80	59048	**65.53**
10	80	2	LTAC	6400		81840	**88.24**		10	80	3	LTAC	6400		4723840	**4730.2**
12	80	2	3D-C	76800	80		**76.88**		12	80	3	3D-C	76800	80	2	**76.88**
12	80	2	3D-F	76800	960		**77.76**	*N=12*	12	80	3	LAC	76800		160	**76.96**
12	80	2	TAC	6400	80	4095	**10.58**		12	80	3	TAC	6400	80	531440	**537.92**
12	80	2	LTAC	6400		327600	**334**		12	80	3	LTAC	6400		4.3E+07	**42522**
14	80	2	3D-C	89600	80		**89.68**		14	80	3	3D-C	89600	80	2	**89.68**
14	80	2	3D-F	89600	1120		**90.72**	*N=14*	14	80	3	LAC	89600		160	**89.76**
14	**80**	**2**	**TAC**	**6400**	**80**	**16383**	**22.86**		14	80	3	TAC	6400	80	4782968	**4789.5**
14	80	2	LTAC	6400		1310640	**1317**		14	80	3	LTAC	6400		3.8E+08	**382644**
16	80	2	3D-C	102400	80		**102.48**		16	80	3	3D-C	102400	80	2	**102.48**
16	80	2	3D-F	102400	1280		**103.68**	*N=16*	16	80	3	LAC	102400		160	**102.56**
16	80	2	TAC	6400	80	65535	**72.02**		16	80	3	TAC	6400	80	4.3E+07	**43053**
16	80	2	LTAC	6400		5242800	**5249.2**		16	80	3	LTAC	6400		3.4E+09	**3E+06**

Table E.7: Vocabulary sizes (V) for video segmentation with $S = 128$, varying values of N from $N = 2$ to $N = 16$ and both binary and multiclass cases. The maximum possible N for each case such that $V < 32K$ is highligted in yellow. V is shown in units of thousands.

Binary								S=128	Multi-Class							
N	S	C+1	encoding	starts tokens	lengths tokens	class tokens	V (K)		N	S	C+1	encoding	starts tokens	lengths tokens	class tokens	V (K)
2	128	2	3D-C	32768	128		32.9	N=2	2	128	3	3D-C	32768	128	2	32.9
2	128	2	3D-F	32768	256		33.02		2	128	3	LAC	32768		256	33.02
2	128	2	TAC	16384	128	3	16.52		2	128	3	TAC	16384	128	8	16.52
2	128	2	LTAC	16384		384	16.77		2	128	3	LTAC	16384		1024	17.41
3	128	2	3D-C	49152	128		49.28	N=3	3	128	3	3D-C	49152	128	2	49.28
3	128	2	3D-F	49152	384		49.54		3	128	3	LAC	49152		256	49.41
3	128	2	TAC	16384	128	7	16.52		3	128	3	TAC	16384	128	26	16.54
3	128	2	LTAC	16384		896	17.28		3	128	3	LTAC	16384		3328	19.71
4	128	2	3D-C	65536	128		65.66	N=4	4	128	3	3D-C	65536	128	2	65.67
4	128	2	3D-F	65536	512		66.05		4	128	3	LAC	65536		256	65.79
4	128	2	TAC	16384	128	15	16.53		4	128	3	TAC	16384	128	80	16.59
4	128	2	LTAC	16384		1920	18.3		4	128	3	LTAC	16384		10240	26.62
6	128	2	3D-C	98304	128		98.43	N=6	6	128	3	3D-C	98304	128	2	98.43
6	128	2	3D-F	98304	768		99.07		6	128	3	LAC	98304		256	98.56
6	128	2	TAC	16384	128	63	16.58		6	128	3	TAC	16384	128	728	17.24
6	128	2	LTAC	16384		8064	24.45		6	128	3	LTAC	16384		93184	109.57
8	128	2	3D-C	131072	128		131.2	N=8	8	128	3	3D-C	131072	128	2	131.2
8	128	2	3D-F	131072	1024		132.1		8	128	3	LAC	131072		256	131.33
8	128	2	TAC	16384	128	255	16.77		**8**	**128**	**3**	**TAC**	**16384**	**128**	**6560**	**23.07**
8	128	2	LTAC	16384		32640	49.02		8	128	3	LTAC	16384		839680	856.06
9	128	2	3D-C	147456	128		147.58	N=9	9	128	3	3D-C	147456	128	2	147.59
9	128	2	3D-F	147456	1152		148.61		9	128	3	LAC	147456		256	147.71
9	128	2	TAC	16384	128	511	17.02		9	128	3	TAC	16384	128	19682	36.19
9	128	2	LTAC	16384		65408	81.79		9	128	3	LTAC	16384		3E+06	2535.7
12	128	2	3D-C	196608	128		196.74	N=12	12	128	3	3D-C	196608	128	2	196.74
12	128	2	3D-F	196608	1536		198.14		12	128	3	LAC	196608		256	196.86
12	**128**	**2**	**TAC**	**16384**	**128**	**4095**	**20.61**		12	128	3	TAC	16384	128	531440	547.95
12	128	2	LTAC	16384		524160	540.54		12	128	3	LTAC	16384		7E+07	68041
14	128	2	3D-C	229376	128		229.5	N=14	14	128	3	3D-C	229376	128	2	229.51
14	128	2	3D-F	229376	1792		231.17		14	128	3	LAC	229376		256	229.63
14	128	2	TAC	16384	128	16383	32.9		14	128	3	TAC	16384	128	5E+06	4799.5
14	128	2	LTAC	16384		2097024	2113.41		14	128	3	LTAC	16384		6E+08	612236
16	128	2	3D-C	262144	128		262.27	N=16	16	128	3	3D-C	262144	128	2	262.27
16	128	2	3D-F	262144	2048		264.19		16	128	3	LAC	262144		256	262.4
16	128	2	TAC	16384	128	65535	82.05		16	128	3	TAC	16384	128	4E+07	43063
16	128	2	LTAC	16384		8388480	8404.86		16	128	3	LTAC	16384		6E+09	6E+06

E.3 Segmentation Metrics for Subsampled Masks

This section provides statistics about the degradation in mask quality produced by subsampling (Table E.9). This is quantitatively represented by the three main segmentation metrics used in this thesis – dice score (Section 8.2.2), recall and precision (Section 3.4.1.1) – averaged over the entire ARIS and IPSC datasets. Qualitatively, most subsampled masks are visually almost indistinguishable from the original when these metrics are $> 90\%$ and sometimes even when they are $> 80\%$.

Table E.8: Vocabulary sizes (V) for video segmentation with $S = 160$, varying values of N from $N = 2$ to $N = 16$ and both binary and multiclass cases. The maximum possible N for each case such that $V < 32K$ is highligted in yellow. V is shown in units of thousands.

Binary								S=160	Multi-Class							
N	S	C+1	encoding	starts tokens	lengths tokens	class tokens	V (K)		N	S	C+1	encoding	starts tokens	lengths tokens	class tokens	V (K)
2	160	2	3D-C	51200	160		51.36	N=2	2	160	3	3D-C	51200	160	2	51.36
2	160	2	3D-F	51200	320		51.52		2	160	3	LAC	51200		320	51.52
2	160	2	TAC	25600	160	3	25.76		2	160	3	TAC	25600	160	8	25.77
2	160	2	LTAC	25600		480	26.08		2	160	3	LTAC	25600		1280	26.88
3	160	2	3D-C	76800	160		76.96	N=3	3	160	3	3D-C	76800	160	2	76.96
3	160	2	3D-F	76800	480		77.28		3	160	3	LAC	76800		320	77.12
3	160	2	TAC	25600	160	7	25.77		3	160	3	TAC	25600	160	26	25.79
3	160	2	LTAC	25600		1120	26.72		3	160	3	LTAC	25600		4160	29.76
4	160	2	3D-C	102400	160		102.56	N=4	4	160	3	3D-C	102400	160	2	102.56
4	160	2	3D-F	102400	640		103.04		4	160	3	LAC	102400		320	102.72
4	160	2	TAC	25600	160	15	25.78		4	160	3	TAC	25600	160	80	25.84
4	160	2	LTAC	25600		2400	28		4	160	3	LTAC	25600		12800	38.4
6	160	2	3D-C	153600	160		153.76	N=6	6	160	3	3D-C	153600	160	2	153.76
6	160	2	3D-F	153600	960		154.56		6	160	3	LAC	153600		320	153.92
6	160	2	TAC	25600	160	63	25.82		**6**	**160**	**3**	**TAC**	**25600**	**160**	**728**	**26.49**
6	160	2	LTAC	25600		10080	35.68		6	160	3	LTAC	25600		116480	142.08
8	160	2	3D-C	204800	160		204.96	N=8	8	160	3	3D-C	204800	160	2	204.96
8	160	2	3D-F	204800	1280		206.08		8	160	3	LAC	204800		320	205.12
8	160	2	TAC	25600	160	255	26.02		8	160	3	TAC	25600	160	6560	32.32
8	160	2	LTAC	25600		40800	66.4		8	160	3	LTAC	25600		1E+06	1075.2
9	160	2	3D-C	230400	160		230.56	N=9	9	160	3	3D-C	230400	160	2	230.56
9	160	2	3D-F	230400	1440		231.84		9	160	3	LAC	230400		320	230.72
9	160	2	TAC	25600	160	511	26.27		9	160	3	TAC	25600	160	19682	45.44
9	160	2	LTAC	25600		81760	107.36		9	160	3	LTAC	25600		3E+06	3174.7
12	160	2	3D-C	307200	160		307.36	N=12	12	160	3	3D-C	307200	160	2	307.36
12	160	2	3D-F	307200	1920		309.12		12	160	3	LAC	307200		320	307.52
12	**160**	**2**	**TAC**	**25600**	**160**	**4095**	**29.86**		12	160	3	TAC	25600	160	531440	557.2
12	160	2	LTAC	25600		655200	680.8		12	160	3	LTAC	25600		9E+07	85056
14	160	2	3D-C	358400	160		358.56	N=14	14	160	3	3D-C	358400	160	2	358.56
14	160	2	3D-F	358400	2240		360.64		14	160	3	LAC	358400		320	358.72
14	160	2	TAC	25600	160	16383	42.14		14	160	3	TAC	25600	160	5E+06	4808.7
14	160	2	LTAC	25600		2621280	2646.88		14	160	3	LTAC	25600		8E+08	765300
16	160	2	3D-C	409600	160		409.76	N=16	16	160	3	3D-C	409600	160	2	409.76
16	160	2	3D-F	409600	2560		412.16		16	160	3	LAC	409600		320	409.92
16	160	2	TAC	25600	160	65535	91.3		16	160	3	TAC	25600	160	4E+07	43072
16	160	2	LTAC	25600		10485600	10511.2		16	160	3	LTAC	25600		7E+09	7E+06

E.4 Visualization

This section presents visualization images for the various RLE tokenization schemes for video segmentation. Figures E.1 and E.2 respectively show 3D-C and 3D-F tokenization schemes for binary masks. Figures E.3 and E.4 show TAC tokenization for $N = 2$ and $N = 3$ respectively. Figure E.5 shows LTAC tokenization for $N = 2$. Please refer Figure 7.5 for details of the layout Figures E.3 - E.5. Each figure has an animated version on the project website [295] whose link is provided in its caption.

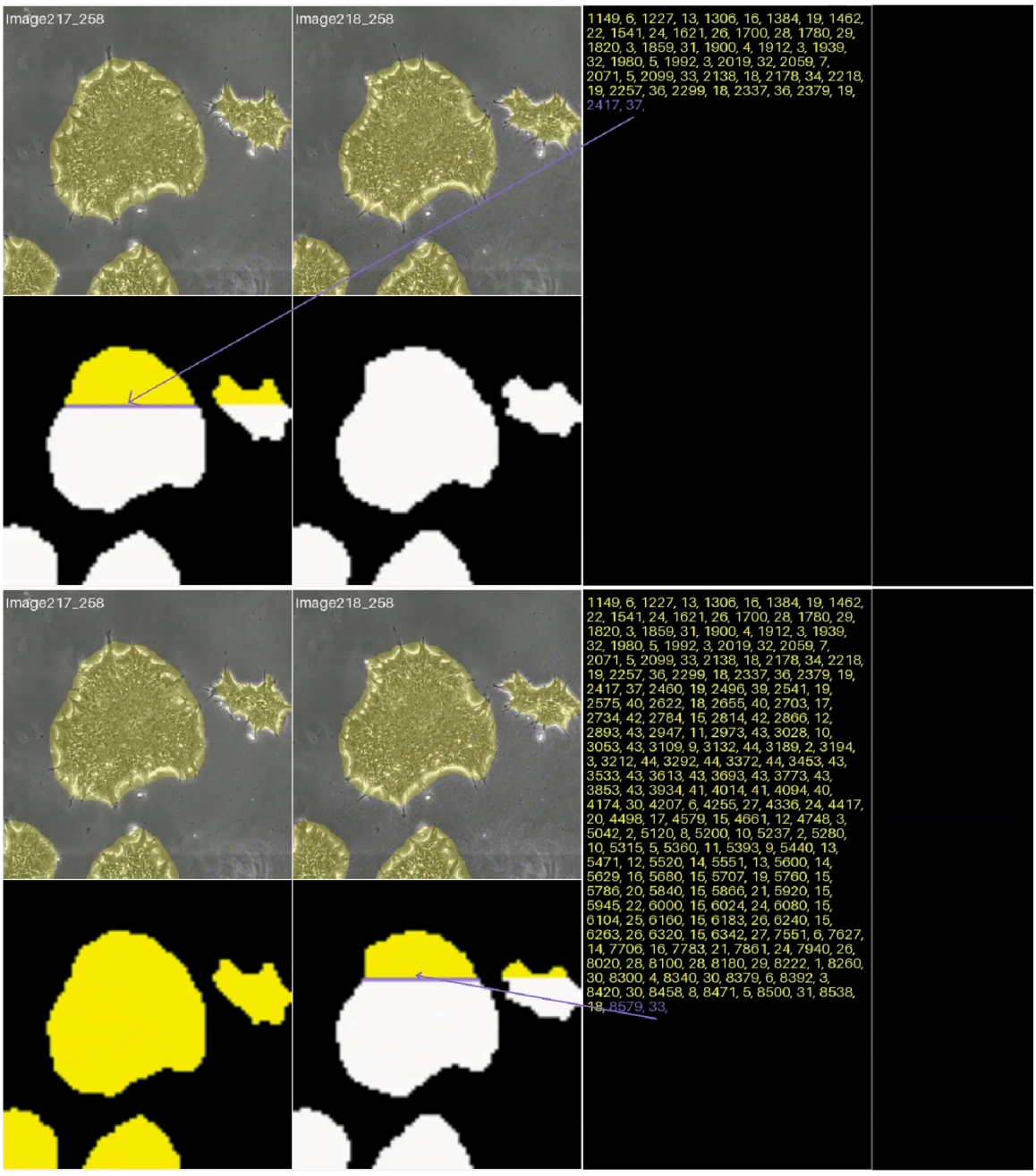

Figure E.1: Visualization of 3D-C RLE tokenization of binary video segmentation masks with $N = 2$, showing runs corresponding to the same part of the same object in F_1 (top) and F_2 (bottom). We can see that their tokens are completely unrelated in the sequence. Animated version of this figure is available here.

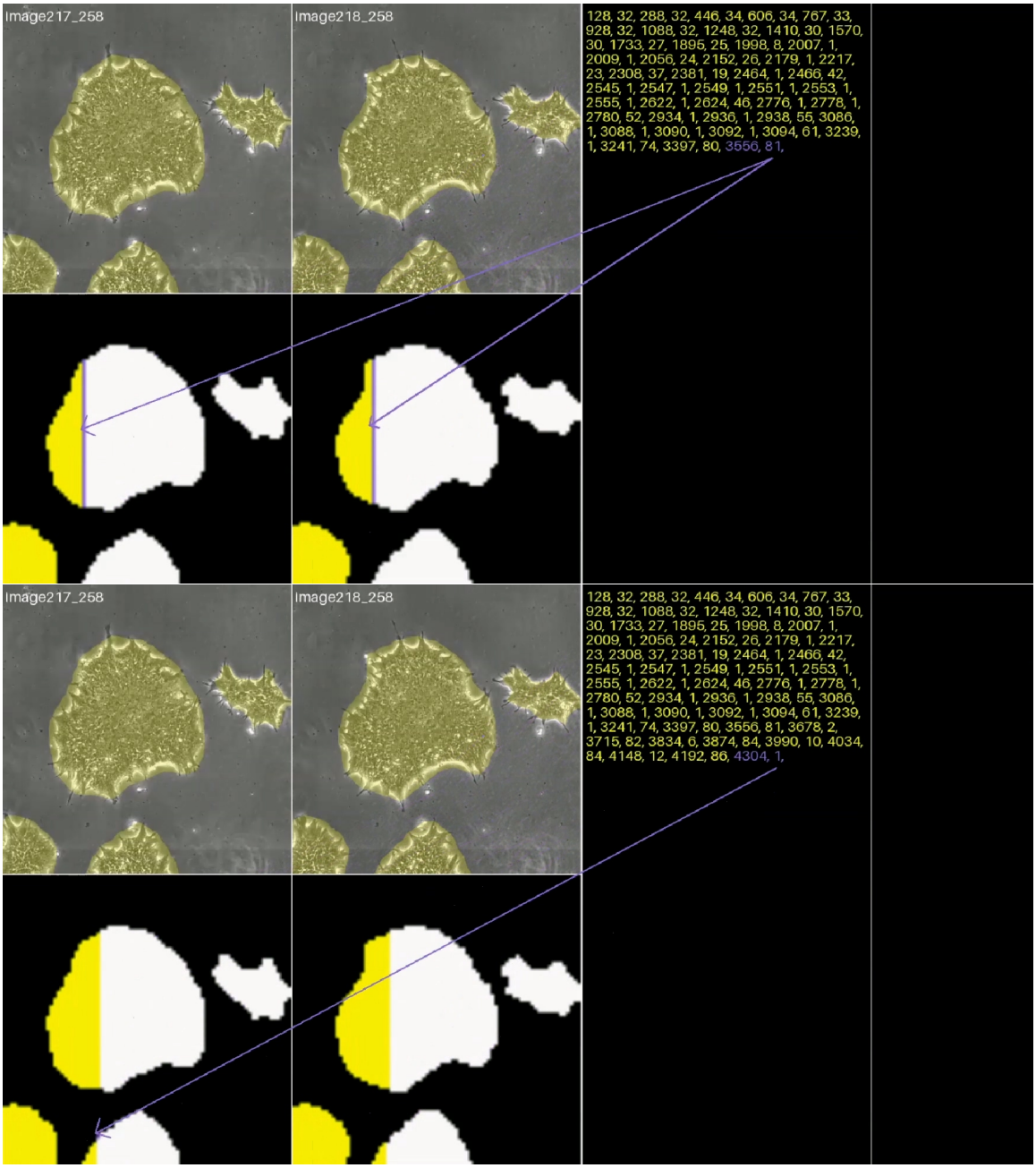

Figure E.2: Visualization of 3D-F RLE tokenization of binary video segmentation masks with $N = 2$. The top half shows the same part of the same object in F_1 and F_2 represented by the same run and the bottom half shows one of the unit sized runs created by a tiny change in the shape of the cell between the two frames. Animated version of this figure is available here.

Table E.9: Segmentation metrics representing mask quality degradation by subsampling, averaged over the entire ARIS and IPSC datasets.

ARIS								
Image size (I)	Patch size (P)	Mask Size (S)	Anchor Ice Metrics (%)			Frazil Ice Metrics (%)		
			Dice Score	Precision	Recall	Dice Score	Precision	Recall
2560	640	320	97.79	100.00	100.00	99.31	100.00	100.00
2560	640	160	95.49	90.00	100.00	98.07	100.00	100.00
2560	640	80	91.88	90.00	100.00	95.68	90.00	100.00
1280	640	320	98.31	96.71	99.50	98.87	97.78	100.00
1280	640	160	95.20	90.99	98.50	96.73	93.77	100.00
1280	640	80	89.64	81.69	96.50	92.88	87.01	100.00
640	640	320	97.42	94.98	99.09	98.24	96.54	100.00
640	640	160	92.79	86.62	97.29	94.96	90.47	100.00
640	640	80	84.94	74.06	93.45	89.17	80.70	100.00
IPSC								
Image size (I)	Patch size (P)	Mask Size (S)	IPSC Metrics (%)			DfC Metrics (%)		
			Dice Score	Precision	Recall	Dice Score	Precision	Recall
2560	640	320	99.71	99.46	100.00	99.30	98.66	100.00
2560	640	160	99.21	98.56	100.00	97.99	96.32	100.00
2560	640	80	98.32	97.08	100.00	95.71	92.49	100.00
2560	1024	256	99.19	98.49	100.00	98.36	96.92	100.00
2560	1024	128	98.25	96.86	100.00	96.39	93.49	100.00
2560	1024	64	96.63	94.28	100.00	92.96	87.92	100.00
640	640	320	98.33	96.78	100.00	98.31	96.70	100.00
640	640	160	95.30	91.47	100.00	95.12	90.85	100.00
640	640	80	90.11	83.55	100.00	89.35	81.32	100.00
1024	1024	256	96.86	94.14	100.00	96.83	93.94	100.00
1024	1024	128	93.25	88.20	100.00	92.95	87.14	100.00
1024	1024	64	87.34	79.77	100.00	86.11	76.45	100.00

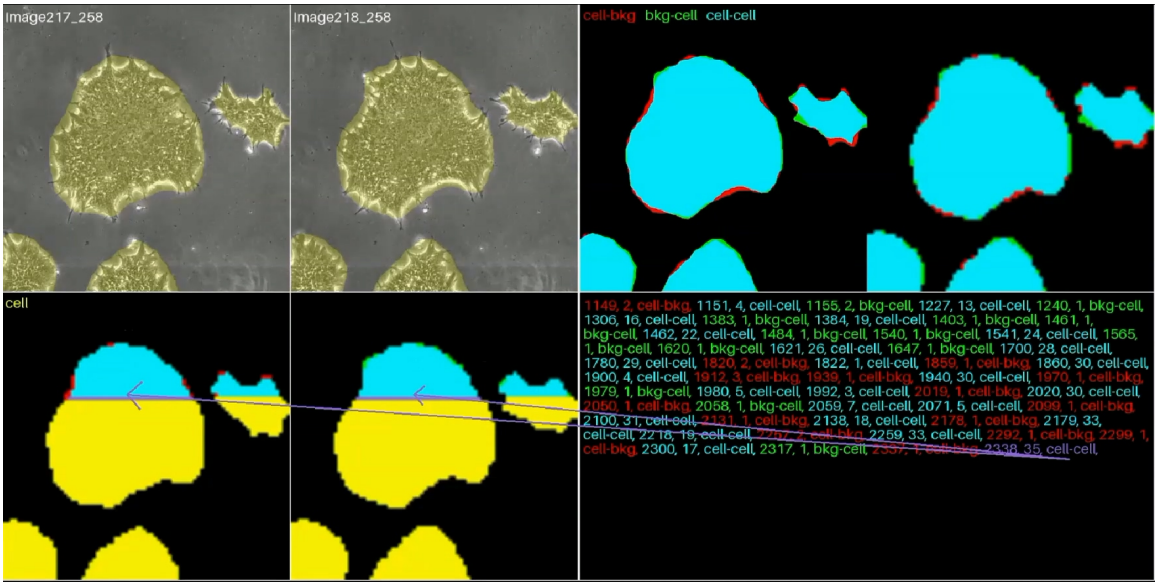

Figure E.3: Visualization of TAC tokenization for binary video segmentation masks with $N = 2$. Animated version of this figure is available here.

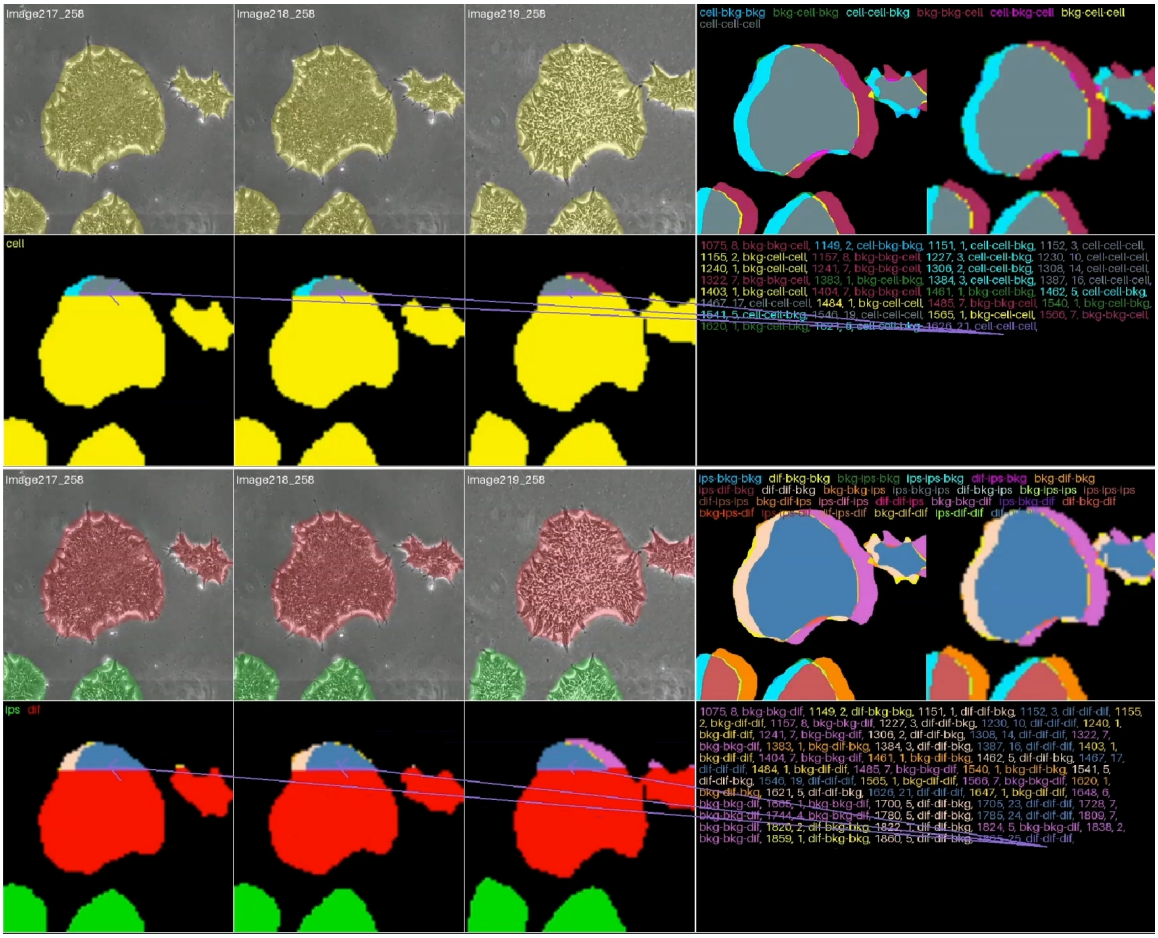

Figure E.4: Visualization of TAC tokenization with $N = 3$ for (top) binary and (bottom) multi-class masks. Animated versions of these figures are available here and here respectively.

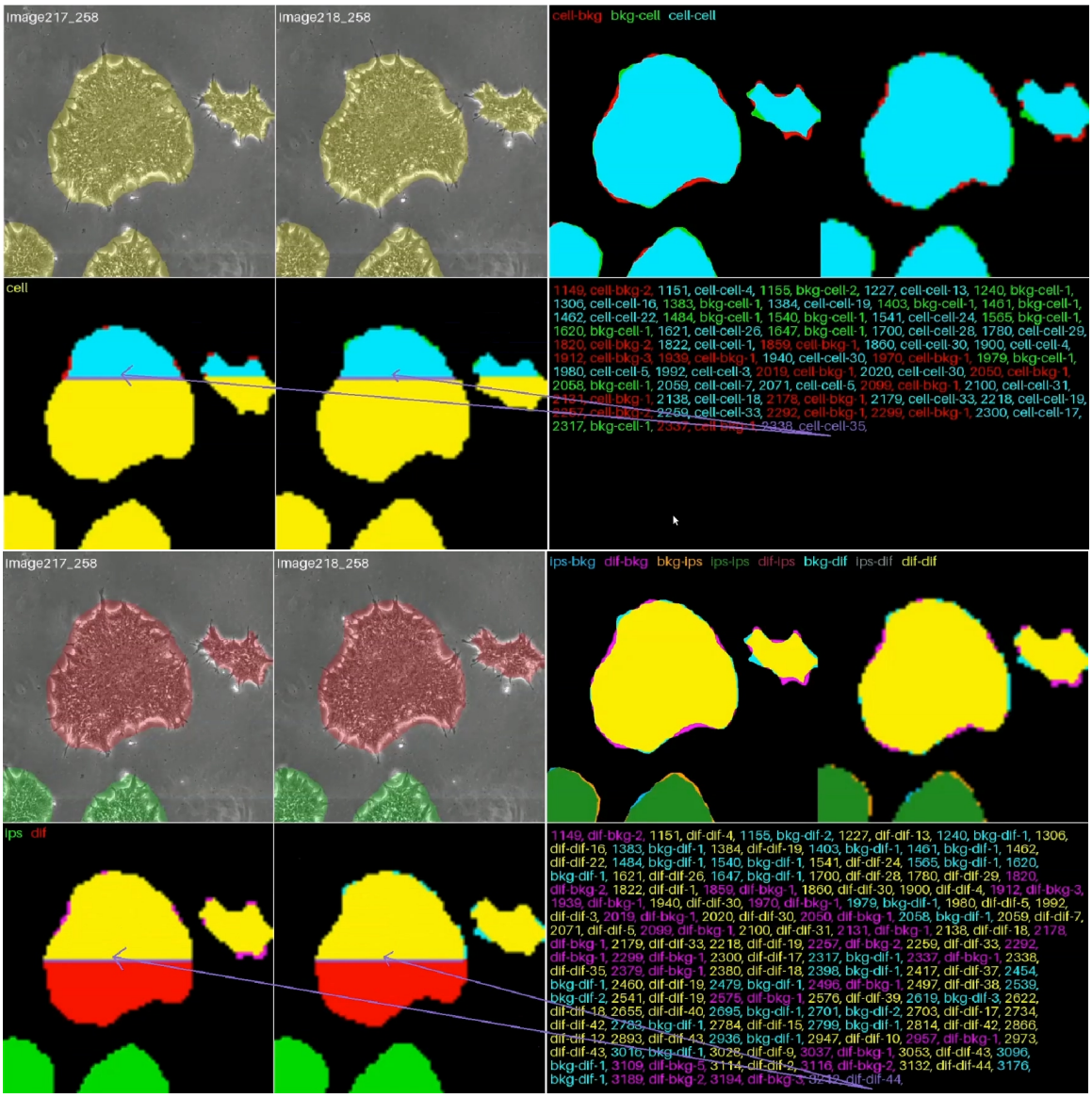

Figure E.5: Visualization of LTAC tokenization for video segmentation masks with $N = 2$ for (top) binary and (bottom) multi-class cases. Animated versions of these figures are available here and here respectively.

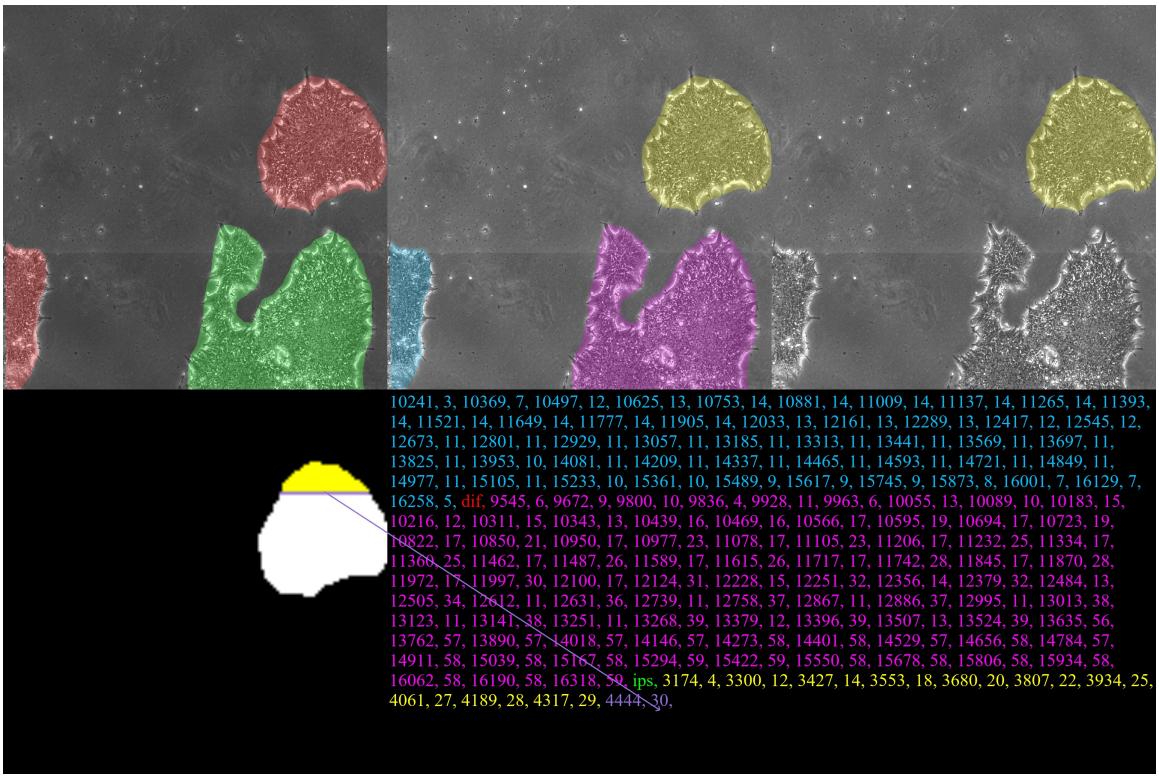

Figure E.6: Visualization of IW tokenization for multi-class static segmentation mask. The first image in the top row shows the class mask where *IPSC* and *DfC* cells are shown in green and red respectively. The second image shows the instance mask where each individual cell is shown in a different color. The last image shows the cell whose binary RLE tokens are currently being generated. The corresponding subsampled binary mask is shown in the first image in the second row. The tokens for the other two cells have already been generated (shown in corresponding colors) and terminated by the respective class tokens. An animated version of this figure is available here.

Appendix F

Results

This appendix provides some supplementary data and configuration details that have been used in chapter 8.

Table F.1: Details of the GPU servers used for model training and inference.

Type	CPU			RAM	GPUs	GPU RAM
	Model	Speed	Cores / Threads			
Primary Training Servers	Intel Core i7-6800K	3.4 GHz	6 / 12	64 GB	2 x Geforce RTX 3090	48 GB
	Intel Xeon E5-2620v4	2.1 GHz	8 / 16	32 GB	2 x Geforce RTX 3090	48 GB
	Intel Core i7-6700K	4.0 GHz	4 / 8	32 GB	2 x Geforce RTX 3090	48 GB
	Intel Core i7-6800K	3.4 GHz	6 / 12	64 GB	2 x Geforce RTX 3090	48 GB
Rented Training Server	AMD EPYC-Milan	2.45 GHz	16 / 16	64 GB	2 x Tesla A100 80 GB	160 GB
Secondary Training / Inference Server	Intel Core i7-3820	3.6 GHz	4 / 8	32 GB	3 x Geforce GTX 1080 Ti	33 GB
Inference Server	Intel Core i5-10400	2.9 GHz	6 / 12	48 GB	Geforce RTX 3090 Geforce RTX 3060	36 GB

Figure F.1: Visualization of system resource usage during a distributed training run across 3 GPU servers with 2 × RTX 3090 GPUs each. The top half shows the GPU usage on the *nvtop* tool while the bottom half shows the CPU usage on the *htop* tool. The blue and orange lines in the *nvtop* plots show the GPU processor and memory utilization respectively. In an ideal training scenario without networking and storage overheads, the two lines would be virtually overlapping. Animated version of this figure is available here.

Table F.2: Performance on UA-DETRAC dataset of P2S and P2S-VID models trained with more GPU RAM to partially alleviate batch size bottleneck. Please refer Section 8.4.2.3 and Table 8.2 for more details.

Model		*N*	mAP (%)	*B*	GPUs	GPU RAM
P2S-VID	Early Fusion	2	75.129	80	2 x Tesla A100	160 GB
	Middle Fusion	2	89.55	320	2 x Tesla A100	160 GB
		3	87.97	216	2 x Tesla A100	160 GB
		4	88.17	160	8 x RTX 3090	192 GB
	Late Fusion	**2**	**91.14**	**256**	**2 x Tesla A100**	**160 GB**
		3	89.76	168	2 x Tesla A100	160 GB
		4	89.47	96	8 x RTX 3090	192 GB
P2S		1	88.62	288	6 x RTX 3090	144 GB

Table F.3: Details of the models whose results are reported in Section 8.4.

P2S

Dataset		Backbone	Input Size	Frozen Backbone	Class Equalization	Coord Tokens	*H*	*V*	*L*	GPUs	*B*
ACAD	#1	ResNet-50	640	Yes	No	2D	2K	3K	512	1 x RTX 3090	48
	#3	ResNet-50	640	Yes	Yes	2D	2K	3K	512	2 x RTX 3090	96
	#4	ResNet-50	640	Yes	No	2D	2K	3K	512	1 x RTX 3090	48
IPSC	Early-Stage	VIT-B	640	Yes	No	2D	2K	3K	512	1 x RTX 3090	4
	Late-Stage	VIT-B	640	Yes	No	2D	2K	3K	512	1 x RTX 3090	4
UA-DETRAC		ResNet-50	640	Yes	No	2D	2K	3K	512	3 x GTX 1080 Ti	60

P2S-VID

Dataset		*N*	Video Architecture	Backbone	Input Size	Frozen Backbone	Class Equalization	Coord Tokens	*H*	*V*	*L*	GPUs	*B*
ACAD	#1	2	Middle Fusion	ResNet-50	640	Yes	Yes	2D	2K	3K	512	2 x RTX 3090	64
	#3	2	Middle Fusion	ResNet-50	640	Yes	Yes	2D	2K	3K	512	2 x RTX 3090	64
	#4	2	Middle Fusion	ResNet-50	640	Yes	No	2D	2K	3K	512	2 x RTX 3090	72
IPSC	Early-Stage	6	Late Fusion	ResNet-50	640	Yes	No	2D	2K	3K	512	2 x RTX 3090	16
	Late-Stage	8	Middle Fusion	ResNet-50	640	Yes	Yes	2D	2K	3K	512	3 x GTX 1080 Ti	12
UA-DETRAC		2	Middle Fusion	ResNet-50	640	Yes	No	2D	2K	3K	512	2 x RTX 3090	80
		4	Middle Fusion	ResNet-50	640	Yes	No	2D	2K	3K	512	2 x RTX 3090	40
		8	Middle Fusion	ResNet-50	640	Yes	No	2D	2K	3K	2048	4 x RTX 3090	28
		16	Middle Fusion	ResNet-50	640	Yes	No	2D	2K	3K	3072	2 x Tesla A100	24
		32	Middle Fusion	ResNet-50	640	Yes	No	2D	2K	3K	5120	6 x RTX 3090	6

P2S-SEG

Dataset		Backbone	Input Size	Frozen Backbone	Encoding	Start Tokens	*I*	*P*	*S*	*V*	*L*	GPUs	*B*
ARIS		ResNet-50	1024	No	LAC	1D	1280	1024	128	18K	3072	2 x RTX 3090	4
IPSC	Early-Stage	ResNet-50	640	Yes	LAC	2D	640	640	320	8K	3072	2 x RTX 3090	8
	Late-Stage	ResNet-50	640	Yes	LAC	2D	2560	640	80	8K	512	1 x RTX 3090	48

P2S-VIDSEG

Dataset		*N*	Backbone	Input Size	Frozen Backbone	Encoding	Start Tokens	*I*	*P*	*S*	*V*	*L*	GPUs	*B*
IPSC	Early-Stage	8	ResNet-50	640	No	TAC	1D	2560	640	80	15K	3072	2 x RTX 3090	8
	Late-Stage	8	ResNet-50	640	Yes	TAC	1D	2560	640	80	15K	3072	2 x RTX 3090	8

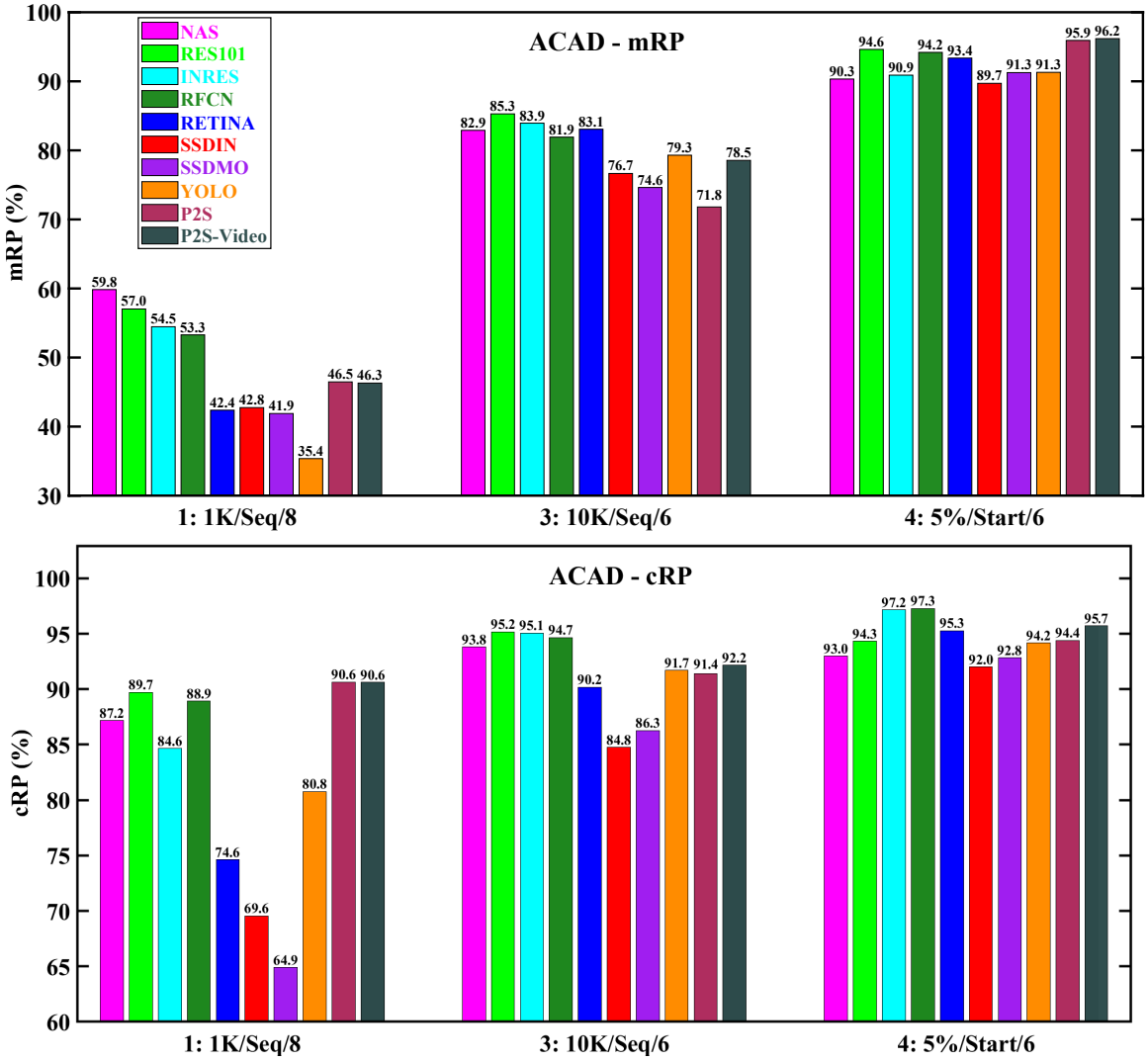

Figure F.2: Bar plot version of Figure 8.1

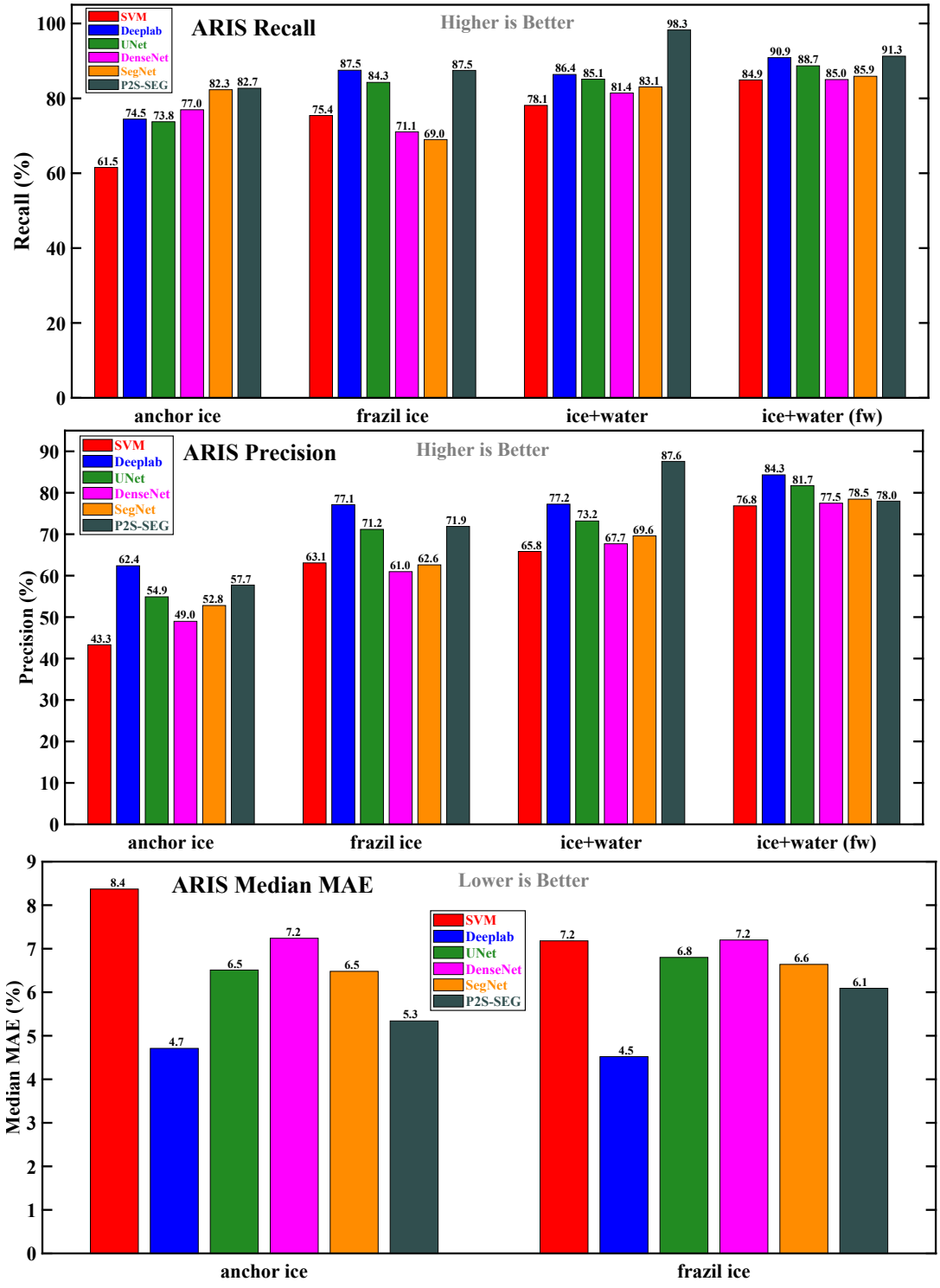

Figure F.3: Bar plot version of the data in Table 8.3

Table F.4: Segmentation metrics (%) on validation set while training P2S-SEG on IPSC early-stage dataset.

training iteration	pix_acc	fw_iu	Recall			Precision			Dice Score		
			ipsc	dfc	cell	ipsc	dfc	cell	ipsc	dfc	cell
4418	8.26	3.44	7.14	6.75	12.88	1.5	6.64	5.67	2.78	6.8	10.43
280543	74.23	61.51	55.09	65.18	84.28	40.44	55.49	69.83	46	66.53	81.63
296006	79.07	66.41	64.41	77.46	84.1	55.39	64.73	70.64	60.98	75.5	82.21
311469	76.02	63.2	58.03	70.06	84.43	45.22	58.86	70.1	50.93	69.81	81.83
326932	76.64	64.2	59.66	69.75	84.4	46.3	59.8	70.47	52.4	70.37	82.08
342395	77.72	65.62	62.85	74.16	83.74	51.75	63.21	70.64	57.53	74	82.21
357858	76.03	65.32	59.9	71.24	83.11	47.68	62.34	71.35	53.57	73.12	82.71
373321	77.47	65.69	60.1	74.49	83.61	50.34	63.37	70.91	55.7	74.22	82.38
388784	78.18	66.63	63.74	75.21	83.75	53.2	64.19	71.32	58.73	75.05	82.71
402038	77.43	65.3	63.41	71.93	84.98	49.98	61.61	71.6	55.47	72.22	82.88
417501	80.05	68.04	65.7	78.71	84.55	56.68	66.59	72.03	62.26	77.31	83.21
432964	78.33	67.38	63.23	76.12	83.66	52.78	65.75	72.02	58.78	76.35	83.22
448427	79.11	67.93	65.46	78.98	82.93	57.74	67.11	71.22	63.64	77.76	82.61
463890	79.08	66.37	61.45	74.63	86.09	49.23	63.17	72.13	54.83	73.83	83.3
479353	78.29	66.52	63.06	75.8	84.19	51.92	64.86	71.57	57.27	75.6	82.89
494816	80.88	67.97	65.07	79.55	85.81	54.88	67.36	72.04	60.58	77.62	83.21
510279	80.83	68.12	63.43	79.69	85.63	53.93	67	72.2	59.3	77.63	83.31
523533	80.47	67.36	62.82	77.25	86.54	50.75	65.99	72.31	56.28	76.45	83.39
538996	80.65	68.1	64.73	79.08	85.81	54.62	66.51	72.56	59.71	77.17	83.57
554459	80.88	68.43	65.9	79.63	85.86	55.24	67.42	72.78	61.13	77.74	83.75
569922	81.49	68.25	64.52	79.67	86.4	55.01	66.71	72.36	60.61	77.12	83.44
585385	81.85	68.92	66.1	82.28	86.19	56.65	68.08	72.88	62.06	78.57	83.81
600848	81.91	69.35	66.14	82.08	86.19	57.42	68.54	73.14	62.74	78.96	83.97
616311	80.26	67.43	61.52	77.65	86.13	49.4	65.71	72.23	55.06	76.24	83.33
629565	81.66	69.06	65.01	80.82	86.4	54.14	68.27	73.03	59.78	78.71	83.89
645028	80.87	68.23	66.06	79.4	86.32	54.24	66.76	72.86	59.75	77.26	83.79
660491	81.81	69.15	65.84	80.97	86.37	56.08	68.44	73.02	61.55	78.68	83.9
675954	83.19	69.88	67.93	85	86.42	60.16	70.17	72.76	65.61	80.6	83.72
691417	81.46	68.69	66.33	80.44	86.23	55.23	67.72	72.73	61	78.09	83.71
706880	80.71	68.09	64.29	77.96	86.23	51.98	66.58	72.76	57.82	77.11	83.72
722343	82.1	69.54	66.43	82.88	85.8	57.02	69.23	72.87	62.65	79.68	83.79
735597	82.41	69.25	67.77	82.57	86.69	57.36	68.7	72.95	62.77	79.14	83.86
751060	82.31	69.6	65.87	82.51	86.21	56.74	68.93	72.99	62.26	79.3	83.87
766523	83.56	69.25	67.64	82.68	87.8	56.29	68.61	72.76	62.19	79.03	83.68
781986	81.68	68.13	67.94	78.39	87.29	54.57	66.08	72.67	60.38	76.38	83.68
797449	81.31	69.31	66.91	80.88	85.65	57.37	68.1	73.13	62.68	78.61	83.97
812912	81.49	68.48	64.25	79.86	86.77	53.47	66.77	72.79	59.02	77.25	83.75
828375	82.68	69.38	66.26	83.84	86.57	57.5	68.77	72.97	62.85	79.22	83.88
841629	83.04	69.42	66.18	83	86.81	57.4	68.34	72.73	62.77	78.67	83.68
857092	82.65	68.51	67.32	82.19	87.22	55.87	68.14	72.33	61.5	78.76	83.39
872555	82.36	69.42	68.09	81.77	86.53	57.14	68.07	72.97	62.9	78.41	83.83
888018	82.8	69.41	66.47	82.93	86.75	57.4	68.75	72.84	62.73	79.14	83.74
903481	83.75	69.38	65.78	84.5	87.5	55.47	69.3	72.59	60.82	79.87	83.6
918944	81.75	69.16	65.87	80.94	86.07	55.54	68.62	72.78	61.07	78.98	83.69
934407	82.11	69.32	67.57	81.46	86.35	56.63	68.33	72.94	62.16	78.83	83.82
949870	83.82	69.96	67.09	85.08	86.93	58.87	69.69	72.69	64.13	80.11	83.64
963124	82.82	69.38	64.48	82.11	87.28	56.27	67.79	73.23	61.13	78.3	84.03
978587	81.73	68.28	65.66	79.49	87.09	53.46	66.95	72.64	59.15	77.15	83.6
994050	82.16	68.75	65.41	81.07	86.89	55.14	67.14	72.8	60.43	77.72	83.73
1009513	82.5	69.61	66.9	82.5	86.51	58.42	68.45	73.16	63.65	78.82	83.97

Table F.5: Segmentation metrics (%) on validation set while training P2S-VIDSEG on IPSC late-stage dataset.

training iteration	pix_acc	fw_iu	Recall			Precision			Dice Score		
			ipsc	dfc	cell	ipsc	dfc	cell	ipsc	dfc	cell
4092	11.04	1.21	11.85	11.7	12.66	11.59	1.06	1.38	11.68	2.08	2.71
8184	2.22	1	42.94	2.4	2.69	42.94	1.04	1.27	42.94	1.94	2.45
16368	2.92	2.18	40.21	2.64	3.62	40.19	1.77	2.77	40.24	3.35	5.06
24552	6.1	4.8	40.73	6.29	7.03	40.73	4.52	5.6	40.73	8.05	9.96
32736	10.01	7.1	31.95	10.29	11.65	31.92	6.52	8.4	32.12	11.57	14.67
40920	14.79	11.61	29.57	15.74	17.33	28.69	11.13	13.68	29.18	18.78	22.78
45012	18.99	13.78	41.53	20.22	21.54	41.53	13.13	15.91	41.53	21.88	26.41
53196	24.26	18.41	42.94	25.86	27.03	42.94	17.65	20.88	42.94	28.22	33.16
61380	29.42	22.15	42.97	32.49	32.99	42.97	21.58	25.16	42.99	33.71	38.93
69564	22.43	18.45	21.69	16.7	28.2	17.92	13	23.12	20.72	21.27	35.71
77748	29.27	22.84	16.58	16.52	39.17	8.1	15.02	30.3	12.46	21.51	45.05
81840	36.53	28.14	26.77	31.15	43.19	24.11	22.78	33.08	26.63	34.24	47.97
85932	39.1	29.21	22.94	30.56	48.56	14.81	22.28	35.52	17.91	33.7	50.83
94116	46.4	33.37	35.83	47.84	52.7	35.61	30.46	38.31	36.69	43.8	54.18
110484	49.53	36.58	33.51	50.92	55.23	32.4	33.89	41.11	33.94	47.42	56.77
114576	51.4	37.46	32.31	52.93	58.49	32.09	34.37	42.91	33.31	47.96	58.63
122760	44.51	32.67	23.81	25.08	59.09	11.94	19.64	42.73	16.73	29.27	58.29
130944	55.82	38.95	42.81	58.98	62.65	42.79	36.31	44.51	43.01	49.95	60.45
139128	52.57	38.58	32	54.7	59.33	31.61	35.42	44.2	32.52	49.14	59.83
143220	49.65	35.16	25.21	34.6	62.67	14.5	25.49	43.5	18.68	36.91	59.28
151404	53.92	40.48	37.38	55.2	61.26	37.28	36.89	46.58	38.25	50.38	62.44
159588	57.43	43.42	26.67	54.86	65.77	24.84	37.13	50.02	27.89	50.78	65.86
167772	56.93	40.62	29.35	52.84	66.84	25.23	34.14	47.53	28.32	47.61	63.1
175956	59.04	42.32	23.82	55.08	70.02	22	35.16	50.12	24.57	48.64	65.6
180048	61.38	44.62	26.56	58.52	70.01	24.67	38.78	50.92	27.42	52.09	66.49
188232	60.04	43.54	43.35	64.16	67.55	43.18	40.59	49.9	43.53	54.3	65.35
196416	55.09	41.46	25.33	44.28	67.19	19.26	31.48	50.2	22.75	44.04	65.64
204600	61.8	45.71	38.39	64.61	69.46	38.38	42.38	52.24	38.61	55.53	67.45
212784	61.57	44.61	32.7	63.15	70.67	32.1	39.79	51.99	33.53	53.4	67.52
216876	55.69	39.46	31.36	32.4	72.63	13.73	23.9	50.72	19.26	34.6	66.11
225060	61.87	46.51	31.4	63.39	70.3	30.67	41.81	53.57	32.03	55.36	68.71
233244	57.9	45.76	21.46	51.31	67.15	18.21	37.36	53.19	21.85	50.43	68.39
241428	60.41	44.99	36.19	61.77	68.99	36	40.42	52.25	36.93	53.92	67.57
249612	58.93	44.96	31.27	50.05	69.48	25.91	35.3	52.49	30.43	47.96	67.35
253704	59.05	45.18	24.98	44.75	70.29	16.44	32.81	53.28	21.4	44.99	68.31
261888	64.25	46.93	30.64	61.93	74.26	26.67	40.4	54.38	29.85	53.94	69.41
270072	60.77	44.65	29.8	48.12	73.38	22.84	33.22	53.14	27.61	45.86	68.12
278256	64.24	47.08	33.5	65.81	73.29	33.06	41.9	54.46	35.18	55.32	69.35
286440	63.47	46.91	26.73	57.14	74.67	22.41	38.25	55.03	26.36	51.75	69.92
290532	63.63	45.32	29.03	49.25	77.3	20.46	33.18	54.22	25.35	45.85	69.35
298716	65.57	48.46	39.04	69.99	73.97	39.04	44.66	56.03	39.32	58.14	70.95

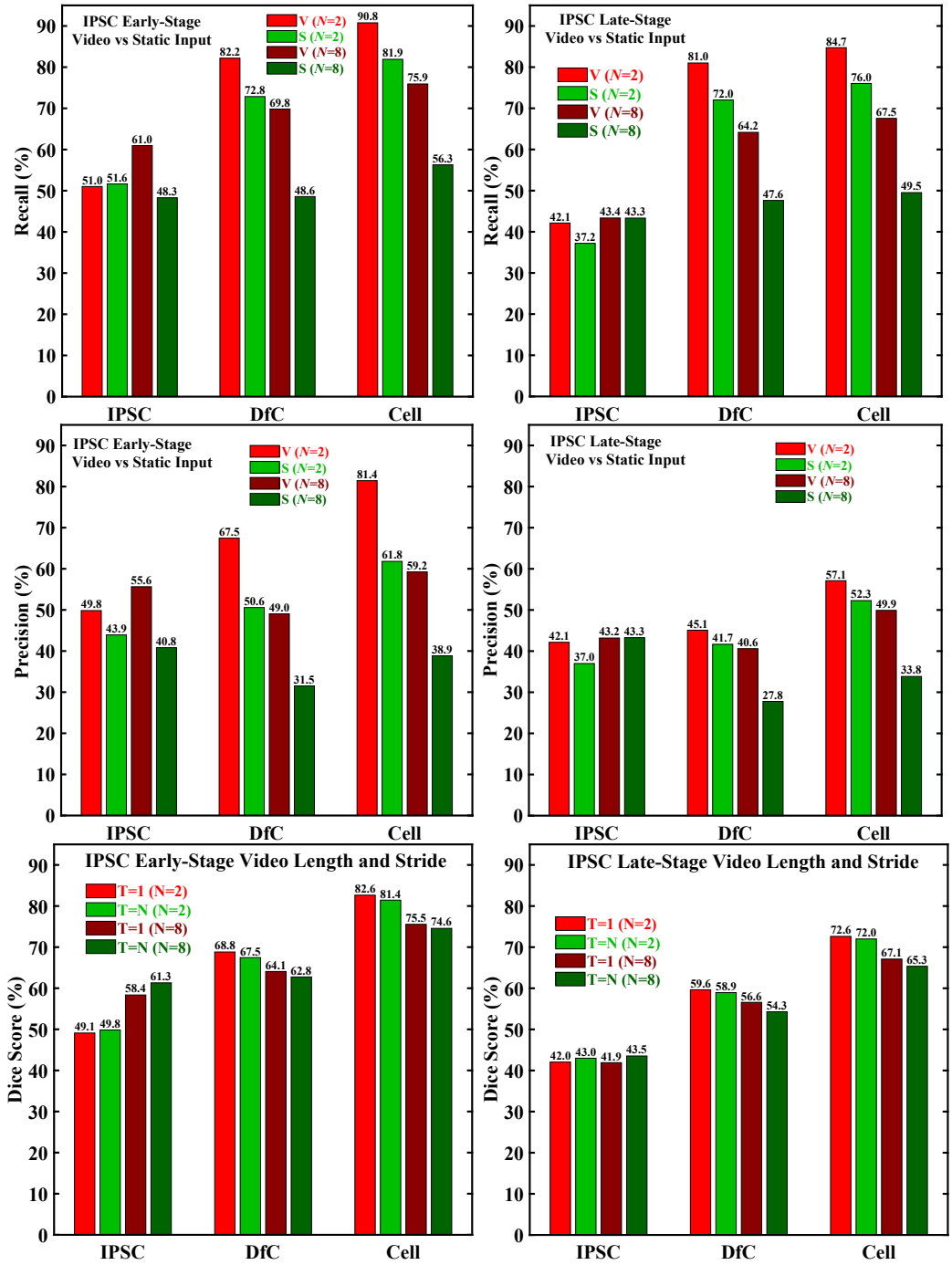

Figure F.4: Impact of (top and middle) replacing N video frames with the first frame and (bottom) video length N and stride T on P2S-VIDSEG performance over (left) early and (right) late-stage configurations of the IPSC dataset.