Enhancing Traffic Sign Detection with Temporal Information Fusion: Improving Efficiency of Automated Maintenance Detection

Hoogervorst, Willem

Universiteit
Leiden
The Netherlands

Enhancing Traffic Sign Detection with Temporal Information Fusion

Improving Efficiency of Automated Maintenance Detection

Willem Hoogervorst

Dr. S. Böhringer, Assistant Professor at LUMC

F. de Vries, Data Scientist at CGI

Defended on September 29, 2022

MASTER THESIS

STATISTICS AND DATA SCIENCE

UNIVERSITEIT LEIDEN

Foreword

This Master thesis is written by Willem Hoogervorst under supervision of Dr. Stefan Böhringer and Folkert de Vries, in collaboration with CGI. The analysis that is performed is based on results from cited references. The data holds no confidentiality or personal information.

The python code written for this research can be found at the following repository:
`https://github.com/WillemHoogervorst/Thesis-public-dashcam-temporal`

All data for this research is gathered and owned by CGI.

First and foremost, I would like to thank Folkert de Vries and the iAMLab team at CGI for welcoming me into their midst over the past six months. I had a very pleasant time working at the CGI office, participating in the daily standup and team activities. Folkert, you were an excellent counselor and sparring partner to me. I could not have done this thesis research without you. I would like to thank Stefan Böhringer for his advice, his pragmatism, and steering me in the right direction by tempering my overly ambitious plans, keeping the objective realistic. It kept amazing me how fast and seemingly effortless you could get to the essence of my research process during our regular meetings.

Contents

Abstract

This thesis research aims to improve traffic sign detection within dashcam footage by using temporal information. Essentially, a video is a set of images displayed at a fast rate. Temporal information lies in the similarity across subsequent frames. However, current state-of-the-art object detection frameworks only use single images. To test whether temporal information can increase the performance of a Convolutional Neural Network (CNN), we train three models: YoloV5, a 3D CNN and a 4D CNN. YoloV5 is used to benchmark the other models against a state-of-the-art framework for object detection. Second, the existing architecture of YoloV5 is adopted as a basis for the 3D CNN. After tuning the hyperparameters for the 3D CNN, performance is compared to YoloV5. Third, the 3D CNN is changed into a 4D CNN that processes sets of frames. By combining the frames within a set, the information in each frame is fused together, including the temporal information across the frames. We call this temporal information fusion (TIF). Comparing the performance of the 3D CNN to those of the 4D CNN shows the effect of TIF. In this research, a balanced dataset containing 444 sets of frames containing traffic signs from dashcam videos is used to train and test the models. The objective is to correctly classify the traffic signs on the frames. The results show that TIF can increase the accuracy of a CNN model by 2%, purely through the addition of TIF. The main drawback of using TIF is an increase in processing time. Instead of a single image, the network needs to process a set of images, which naturally will take longer. The results in this research can form a basis to explore TIF in object detection further.

1 Introduction

This thesis research aims to improve traffic sign detection within dashcam footage by using temporal information. Initially, the research was set to use dashcam footage to recognize situations where road signs need to be replaced or maintained. This could be achieved by comparing historical road sign detection data to new data. For instance, road sign X is recognized and correctly classified 380 out of 400 times a vehicle equipped with a dashcam drove past it on a given day. The next day, road sign X gets passed by a dashcam car 420 times, but now only 2 out of 420 classifications is correct. This could be an indication that something has happened to the sign and should trigger further inspection or maintenance. Figure 1 shows an example of a road sign for which a scenario of such a sudden drop in correct classifications is possible. Other causes of sudden changes can be caused by traffic colliding with road signs, destroying (parts of) the signs, heavy winds tearing off signs from poles, portals and buildings, or storms causing trees to crash into road signs.

Figure 1: Example of a road sign that is receptive to automated maintenance detection
source: `https://www.destentor.nl/brummen/omstreden-verkeersborden-brummen-beklad~a3a26b98/100585389/`

The scenario of the road sign in figure 1 is very sudden: a road sign becomes unreadable overnight. Oftentimes, this degradation of readability happens gradually. Where sudden destruction of road signs gets noticed and reported to authorities, weatherworn signs may remain situated for years without maintenance. If correct classification of these signs decreases as the readibility becomes worse, at some point a threshold is crossed. Figure 2 shows examples of road signs that may not undergo maintenance for years without this automated maintenance detection.

Automated maintenance detection requires a digital inventory of the road signs

source: https://www.prinsenbeeknieuws.nl/nl/nieuws/zou-het-helpen/#.Yw4SMHZByUk

source: https://www.wegenforum.nl/viewtopic.php?t=22037

Figure 2: Examples of road signs with gradual degradation, that may go unnoticed without automated maintenance detection

on the Dutch highways. In 2020, the Dutch government launched a project to become the first country in the world with a digital overview of road signs. A sign identification code and the exact location is stored for each road sign and published as open data by the National Data portal for Road traffic (NDW). The Dutch road network contains thousands of traffic lights and roughly 3 million road signs divided over approximately 200 classes. CGI has developed a Proof of Concept data pipeline which recognizes 9 different types of road signs, including traffic lights. This pipeline combines YoloV5: an object detection framework (Jocher et al., 2021) with GPS data from the vehicle that is equipped with a dashcam. This combination produces records that can be matched and compared with the data of the NDW for the purpose of incentificing road sign maintenance. The term 'object detection' is used in a variety of meanings, often connected to machine learning. In this paper, we define object detection as the localization and classification of an object of interest in an image or video. The quality of object detection should be further improved to help create greater confidence in the maintenance signals generated by the intended solution of automated maintenance detection. The direction of improvement that is pursued in this research is to perform object detection on pieces of video material instead of single images. Three models will be compared: YoloV5 as a benchmark, a simplified version of YoloV5 that we will call the 3D CNN, and a 4D CNN that processes video material instead of single images and tries to extract the extra information therein.

1.1 Research Goal

YoloV5 processes single images, whereas the input material consists of dashcam videos. This uncovers the opportunity to make use of temporal information across subsequent frames in a video. This might help decrease false classification and create a more stable registration. A single frame within a video might suffer from occlusion (where something partially blocks the view on the object of interest), motion blur, for instance caused by camera shaking, and other incidental variations. Especially in road sign detection, these mishaps are not uncommon. Branches of trees and overtaking vehicles might occlude a road sign. Vibrations from the road might cause motion blur through camera shaking. The angle of the sunlight might obscure the color and content of a road sign for a split second. If one could combine the information from multiple frames into a single detection, the model would be less susceptible to these problems. The goal of this research is to explore ways to exploit this temporal information and use it to boost the performance of the object detection model.

1.2 Convolutional Neural Networks in principle

YoloV5 performs object detection using Convolutional Neural Networks (CNN). In section 1.3, we'll discuss what separates YoloV5 from a plain CNN. For now, it is enough to get some understanding of how a CNN works and why it is suitable for object detection in images. A CNN is a special form of an Artificial Neural Network (ANN). These ANNs are biologically inspired on nerve systems like the human brain. CNNs show exceptional results in and are primarily used for solving difficult image-driven pattern recognition tasks (O'Shea and Nash, 2015). The layers within the CNN are comprised of neurons organised into three dimensions: the spatial dimensionality of the input (height and width) and the depth (channels) (O'Shea and Nash, 2015). Depth, in case of images, refers to the color channels in an image. Color images consist of pixels, where each pixel contains 3 values for Red, Green and Blue (RGB). This way, any image can be brought down to a 3D tensor of dimensions (h x w x 3) where h, w are image height and width in pixels, respectively.
A CNN typically consists of a number of convolutional layers combined with pooling layers and normalization layers. A convolutional layer takes its name from a linear operation between matrices called convolution (Albawi et al., 2017). This operation is performed on the input tensor and a kernel in an element-wise fashion, resulting in an output tensor, as shown in Figure 3. A pooling layer aggregates information within local regions, thereby strongly reducing the number of parameters in the

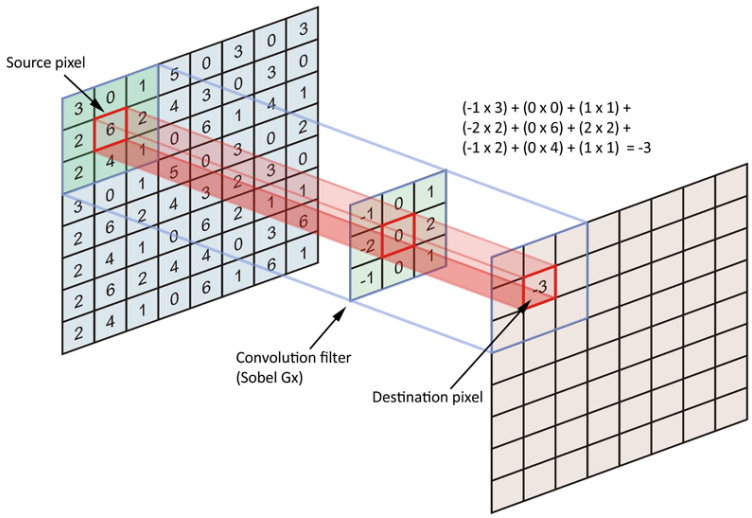


Figure 3: Visualization of Convolution Operation in a single Pixel
source: `https://datascience.stackexchange.com/questions/23183/why-convolutions-always-use-odd-numbers-as-filter-size`

model. "Information aggregation can make the representation more robust to the translation and elastic distortions in some degree" (Sun et al., 2017). YoloV5 barely makes use of pooling layers. Instead, it uses strided convolution to summarize the data, which is explained in Ayachi et al. (2020). In brief, strided convolution skips one or more source pixels after each operation in the element-wise convolution.

Normalization layers like Batch Normalization are essential in being able to train a CNN (Thakkar et al., 2018). Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training. This slows down the training by requiring lower learning rates and careful parameter initialization. We refer to this phenomenon as internal covariate shift, and address the problem by normalizing layer inputs (Ioffe and Szegedy, 2015). YoloV5 makes use of Batch Normalization after every convolutional layer. A summary and visualization of both the YoloV5 network and the networks created in this research can be found in Appendix A. Another important element of a CNN (or ANNs in general) is the activation function. As stated, the network is motivated by biological neurons, which may generate action potentials based on the input they receive. After all convolutional computations are completed, the activation function determines whether the neuron is activated. Popular activation functions for CNNs are ReLU (Rectified Linear Unit) and SiLU (Sigmoid Linear Unit). Ramachandran et al. (2017) found that SiLU consistently outperformed ReLU (among other activation functions) tested on multiple architectures. "The activation of the SiLU is computed by the sigmoid function multiplied by its input" (Elfwing et al., 2018). Figure 4 shows both popular activation functions in a graph.

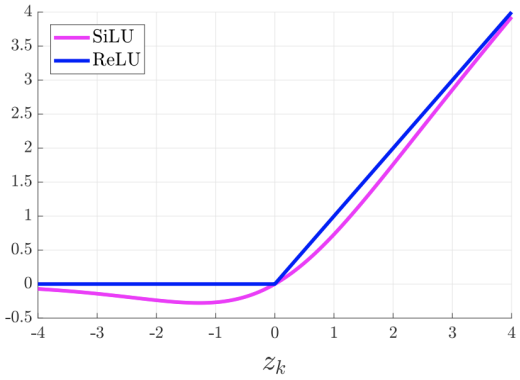


Figure 4: SiLU and ReLU activation functions
source: `https://paperswithcode.com/method/silu`

The combination of convolutional, pooling, normalization, and other optional layers is called the 'network backbone'. The backbone refers to the part of the network that performs feature extraction from the input. The backbone is then paired with the 'network head', which performs tasks like classification or object detection. For a relatively simple task like classification, the head would typically consist of a number of fully connected layers. The final layer returns a number of values equal to the number of classes one wishes to classify. "It is also suggested that ReLU may be used between these layers, as to improve performance" (O'Shea and Nash, 2015). In order to perform more complex tasks like object detection, a more intricate combination of layers needs to be designed. The connections between neurons across layers of a CNN (or any type of ANN) are weighted. When training a CNN, these weights are systematically being updated through backpropagation. After each forward pass of a batch of inputs through the network, the computed output (prediction) is compared to the known, desired output (target). The goal is to update the weights in the model in such a way that optimizes how close the prediction is to the target. For this, an optimizer uses a Loss function that can be minimized to approach the optimal solution. Stochastic Gradient Descent (SGD) is a well-known optimizer. It can be seen as a stochastic approximation of gradient descent optimization. Gradient descent iteratively computes the gradient of the loss function and steps in the direction of the steepest descent. An iteration that consists of a forward pass and a backpropagation step is often called an epoch. Usually, an epoch is split into batches. A batch consists of a part of the data and is sent through the layers of the network. Larger batches speed up the training, whereas smaller batches smoothen the optimization. At some point, the model stops improving. This is called convergence. To determine when convergence is reached and the training phase needs to be stopped, a number of stopping rules can be implemented. Usually

two types of rules are used: an improvement threshold and a maximum epoch. If the improvement that is reached during an epoch drops below the threshold for a number of epochs in a row, the model is considered to have converged. The maximum epoch is used to safeguard against extremely long training times. If convergence is not reached before the maximum epoch (in YoloV5, the default is epoch 300), this stopping rule brings a halt to the training.

1.3 YoloV5: a Family of Object Detection Architectures and Models

This section is dedicated to understanding the YoloV5 framework. It will be used as a benchmark: how well can an extensively developed framework produce a model and how does such a model perform in this context, with limited data resources at its disposal? It also functions as inspiration for the network architectures of the 3D and 4D CNNs. You Only Look Once (Yolo) is a concept in image recognition in which object localization and classification happen simultaneously (Redmon et al., 2016). YoloV5 is a state-of-the-art open source tool, developed by Ultralytics (Jocher et al., 2021). It takes single images of any size as input, applies a variety of data augmentation methods and returns bounding boxes with class labels along with a confidence measure. Each bounding box is defined by center coordinates (x,y) and width & height measures. A class label is a predefined number that corresponds to a particular type of road sign, in this case. The confidence measure is a value between 0 and 1 that shows the confidence of the model that an object of the class label is detected within the bounding box.
In order to estimate these bounding boxes, YoloV5 uses Anchor boxes. An anchor box is a rectangle with a certain aspect ratio. By default, YoloV5 uses nine different anchor boxes: three different shapes for each of three different sizes (granularities). Their default initial values, trained on the MS COCO dataset (Lin et al., 2015) are shown in table 1.

Table 1: YoloV5 default anchors, trained on the MS COCO dataset.
Values are width x height in pixels.

	Anchors		
Granularity	1	2	3
small	10x13	16x30	33x23
medium	30x61	62x45	59x119
large	116x90	156x198	373x326

The set of anchor boxes should contain the smallest and largest sizes of boxes the model should be able to detect, as well as different shapes (aspect ratios) that are

present in the data. Think of tall, thin objects versus short, wide objects or just square objects. In the road sign data, we can expect many rectangular sized bounding boxes, but also slightly taller boxes for traffic lights. The specs of these anchor boxes should therefore depend on the objects the network should be able to detect. During training, the network randomly creates many (thousands) instances of anchor boxes on the image (or actually the convolution thereof), known as 'prior boxes'. For each prior box, the Intersection over Union (IoU) with the ground truth is calculated, and only the ones with an IoU above a certain threshold are kept. The selected prior boxes can be used as feedback to update the dimensions of the anchor boxes. Automatically updating the anchor boxes during training is called AutoAnchor in YoloV5. In case of many classes for which the default nine Anchors would not be enough, the amount of Anchors can be increased. The selected prior boxes can then be reduced to a single bounding box per object, either by Non-Max Suppression (NMS), or computing a new, aggregated bounding box through Weighted Boxes Fusion (WBF). In the YoloV5 framework, the network splits into three classification heads, one for each size group of anchor boxes. So each head will be using its own three dedicated Anchor boxes out of the total of nine Anchor boxes. The heads are fed with output from different levels (layers) within the network (see Appendix A.1). Each classification head performs the process of creating and selecting prior boxes as explained before for all three of its dedicated Anchor boxes. Large anchor boxes, with high dimensions (bottom row in table 1), are applied on macro-scope. Deep convolutions contain highly aggregated, summarized information. This information no longer contains all the details of the original image, but is an abstract representation of the macroscopic features of the objects on the image. This means that it can be used to detect objects with big features that appear large on the images. On the other hand, small anchor boxes work best on shallow convolutions, where more details are still preserved, in order to detect small objects.

1.3.1 Data Augmentation

In order to improve the training potential within a training set of data, a number of data augmentation techniques are used. Visualizations of these techniques can be found in Appendix B.

1. Mosaic is a technique that pastes multiple input images side by side on a canvas with set dimensions. The original aspect ratios of the images are maintained and the part of the canvas that is not covered shows a uniform grey, plain background. Data augmentation techniques like Mosaic aim to increase the variability of the input images, so that the object detection model has

higher robustness to images obtained from different environments. Mosaic also provides the certainty of having equally sized augmented images, regardless of the image size of the raw data. YoloV5 combines four images onto a mosaic, by default. It was introduced in YoloV4 (Bochkovskiy et al., 2020).

2. Copy-paste (Ghiasi et al., 2021). Here, one or more objects within a labeled image are copied onto another image, sometimes partially occluding the objects therein.

3. Random Affine is the collective name for transformations like rotation, scaling, translation and shearing. An affine transformation is a geometric transformation that preserves lines and parallelism, but not necessarily distances and angles. Translation, in this respect, is moving the image along the X or Y axis within the mosaic. Shearing is transforming an image by multiplying the pixel coordinates by a shear matrix. This results in a parallelogram-shaped version of the original image. For road signs, rotation is not likely to be helpful, because the orientation of the objects is usually the same.

4. MixUp (Zhang et al., 2018). In this augmentation technique, images within the mosaic will overlap and receive some level of opacity. For dashcam analysis, this can mimic real-life effects when reflections are visible on the windshield or when the weather is foggy.

5. Albumentations are duplications of the original image with changes that are common in photoshop. For instance the parameter value for Contrast, Hue, Brightness or Blur receives a random value. For this augmentation method, a third party package is used (Buslaev et al., 2020). YoloV5 only uses a specific subset of albumentations: Blur, MedianBlur, ToGray, CLAHE, RandomBrightnessContrast, RandomGamma, and ImageCompression.

6. Augment HSV. HSV stands for Hue, Saturation, and Value, which influence the color in an image. Hue defines which 'color of the rainbow' a pixel has. Saturation can be seen as the warmth of a color, or its intensity. Value is simply how light or dark a color is. HSV can be computed from RGB values and vice versa.

7. Random Horizontal Flip is the last data augmentation technique. This simply flips, or mirrors the image or the entire mosaic horizontally.

These data augmentation techniques contain a number of hyperparameters that can be trained towards a better performance. Examples of such hyperparameters are

the probability that an image is flipped horizontally, or the amount of degrees an image is rotated before being pasted onto the mosaic canvas. Appendix C shows these hyperparameters for each of the augmentation techniques.

1.3.2 Training Strategies

In order to improve the efficiency and performance of the network, YoloV5 uses a number of training strategies.

1. AutoAnchor. During training, the dimensions (and thereby aspect ratios) of the anchors are trained, in order to best match the objects in the training data. YoloV5 does this by default, but AutoAnchor can be turned off if the user wishes to force manual anchors.

2. Warmup and Cosine LR scheduler are both strategies that dynamically update the learning rate during training of the network. During the warmup phase, the learning rate is kept small to overcome optimization challenges early in training (Goyal et al., 2018). After the warmup, the Cosine LR (learning rate) scheduler decays the LR with a sinusoidal ramp, as described in section 5.1 of He et al. (2018). This decay could be implemented in a cyclic manner. However, Smith and Topin (2018) show that training with a single LR cycle and a large maximum LR dramatically decrease training times. YoloV5 uses this single cycle approach.

3. Exponential Moving Average (EMA) is used when the weights in the network are updated. The optimizer usually calculates an average of a number of previous data points, to reduce volatility in the weights. With EMA, recent data points are given higher weights than older ones when calculating this average. Maintaining a moving average of the trained parameters is usually beneficial when training a model (Abadi et al., 2015).

4. Mixed precision training is a 'trick' that cuts the precision of the weights, activations and gradients to half-precision during training. "This technique works for large scale models with more than 100 million parameters trained on large datasets" (Micikevicius et al., 2018). It significantly reduces memory consumption and increases the computation speed.

5. Hyperparameter Evolution "is a method of Hyperparameter Optimization using a Genetic Algorithm (GA) for optimization" (Jocher et al., 2021). YoloV5 includes a staggering number of 29 hyperparameters in their model. These

impact the data augmentation, but also the optimizer, loss, and other aspects of the framework. Grid-searching the parameter spaces for these hyperparameters becomes very time-consuming, and unknown interactions between them might lead to suboptimal results. This makes GA a suitable method. YoloV5 provides automated hyperparameter evolution, but also proposes a default set of hyperparameters that have been evolved for the MS COCO dataset (Lin et al., 2015).

1.3.3 Loss

YoloV5 predicts bounding boxes and the object classes therein, and estimates a confidence level on that class prediction. In order to train this, three different types of losses are combined:

1. Class loss. This computes the loss for the classification of an object that is expected within the bounding box. For this, YoloV5 uses Binary Cross-entropy loss with logits (BCEWithLogitsLoss). This loss function is extended with Focal loss (Lin et al., 2018). Focal loss is used to accentuate classes that are uncommon in the data. This prevents the classifier from stopping to predict these uncommon classes, when the data is imbalanced.

2. Objectness loss teaches the network to predict good IoU values for the prior boxes. It helps the network in selecting only the best prior boxes that are approved for classification. Objectness loss is also computed with the BCE-WithLogitsLoss function and Focal loss.

3. Location loss. Bounding boxes with a high enough Objectness score will receive location loss in order to improve its coordinates. YoloV5 predicts center coordinates (x,y) and width and height values. Prediction of these outputs is trained through Location loss. Location loss is also known as Box loss and is computed by $1 - \frac{\text{intersection}}{\text{union}}$.

1.4 Similar Research

Application of temporal information fusion is applicable in numerous fields besides dashcam analysis. In literature, different implementations of temporal information fusion were found. Many of them use CNNs to train their models. There is some ambiguity in the naming of these models. Some use '3D convolutions' to reference the three-dimensional kernels in the CNN. Others use '4D CNN' like this research, reflecting the four dimensions in the input data. Ji et al. (2013) use temporal

information fusion for Human action recognition. From a single frame, it is difficult to detect the action of bringing a cell phone to your ear. By using a set of consecutive frames instead, the temporal information allows detection of human actions. Human action recognition is also used by Qiu et al. (2019) to introduce their Local and Global Diffusion (LGD) framework. They test the performance of the LGD framework on the Kinetics dataset (Carreira et al., 2019). This dataset contains up to 650k videos covering hundreds of different human actions like playing an instrument, shaking hands or hugging. Another framework that is trained on the same Kinetics dataset (and other datasets) is proposed by Wang et al. (2018). Their Non-local Neural Networks include Non-local blocks, that compute the temporal information within sets of frames. These Non-local blocks are easily combined with other operations like convolutions.

Another terrain where utilizing temporal information can be useful, is in Brain-Computer Interfaces. Sakhavi et al. (2018) propose a classification framework for classifying motor imagery Electro-Encephalogram (EEG) data. This data typically contains a great amount of measurements in short succession. They show that by using temporal information, they can improve the performance of their static predecessor: common spacial patterns (CSP) (Ramoser et al., 2000). The CSP algorithm reduces the EEG signal from a series to a single value, destroying the temporal information. A different application of video analytics is found in sport events classification. Rangasamy et al. (2020) review a variety of deep-learning approaches for match analysis. Here, different implementations of CNNs, Long short-term memory (LSTM) and Gated Recurrent Unit (GRU) models process video material, where they use temporal information to classify slices of frames as events. Examples of these events are: corner kicks, penalties, and scoring points in matches of sports like football, tennis, and ice hockey. Automatic classification of such events can help teams to analyze their matches, or create match summaries for social media.

2 Methods

This research assesses the effect of adding a temporal dimension on the performance of a Convolutional Neural Network (CNN). Hereto, we train three different neural networks. These networks are: 1. An off-the-shelf pre-trained network called YoloV5s; 2. A 3D CNN backbone based on the YoloV5 approach with a Linear Classifier head, made from scratch; 3. A 4D CNN that includes a temporal dimension and implements TIF.

2.1 Data

The data that is used in this research is gathered and labeled by CGI. It contains 444 reference images distilled from raw dashcam footage, evenly distributed across the classes. Balancing the data was needed, since preliminary results showed convergence towards one class that was predominantly present in the data. For 189 images, labeling was done using a YOLO Annotation Tool (Malik, 2018). This produced a text file per image containing the bounding box measurements and the road sign label. Labeling these images by hand is very tedious, even while using this annotation tool. Therefore, we trained the YoloV5s model with this smaller dataset of 189 images. We then performed inference with this trained network on a new dataset, using a high confidence threshold of 0.9. These inferred labels and an additional dataset were added to the dataset to reach the aforementioned 444 reference images. The four classes that were used for training the models are depicted in figure 5 alongside their class label and number of instances in the dataset.

(a) Give way
class label 0
(111 instances)

(b) Speed limit
class label 1
(111 instances)

(c) Keep right
class label 2
(111 instances)

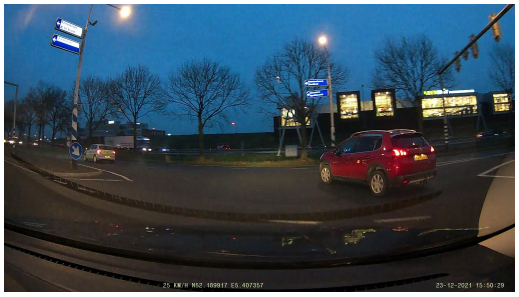
(d) Traffic light
class label 3
(111 instances)

Figure 5: The four road sign classes used in this research, alongside their class count (in brackets)
source: `https://verkeersregels.vvn.nl/verkeersborden-en-tekens`

We are aware that a traffic light actually is not a road sign, but we will regard it as one, since this does not affect the rest of this research. The class 'Keep right' was originally a combined class of 'Keep left' and 'Keep right' signs, due to their similarity. However, almost all instances in the Netherlands are 'Keep right', due to the fact that the Dutch drive on the right side of the road. Therefore, for simplicity, we will call this class 'Keep right'. Throughout this thesis, you might find the label 'Keep Left/Right', which refers to the same class. The choice for these road signs is two-fold. Firstly, these signs are generally consistent in shape, size and color, which makes them suitable for classification. Secondly, they are mutually separable in terms of either shape or color. We decided to classify all different speed limit signs into a single class. If more data would be available, further distinctions between different speed limit signs could be trained and inferred by the model.

2.1.1 Data for YoloV5

YoloV5 can be trained with the entire frame within a dashcam video due to its combined localization and classification capabilities. Figure 6 contains two examples of such frames, accompanied by their labels. One image may contain multiple objects, so the annotation file for a certain image may contain multiple labels, as shown in figure 6d. Each row represents one road sign. The first number of each row represents the class label. The class label is followed by four decimal numbers representing the location and size of the bounding box that tightly fits around the road sign. These four are normalized by the dimensions of the image, so they will have values between 0 and 1. The first two values are the center coordinates (x, y) and the last two decimal numbers contain information about the width and height of the bounding box. Figure 7 shows a graphical representation of this Yolo annotation format. The dataset for YoloV5 contains 274 train and 130 test images, which is less than the 444 mentioned before. This is due to the fact that there can be multiple road signs on one image.



(a) Dashcam image containing a single 'Keep right' sign

```
2 0.140625000 0.509375000 0.026562500 0.063194444
```

(c) Annotation for figure 6a

(b) Dashcam image portraying a situation with multiple objects of interest

```
3 0.549609375 0.391666667 0.028125000 0.077777778
3 0.422070312 0.298611111 0.016796875 0.056944444
3 0.360937500 0.303819444 0.017968750 0.052083333
3 0.260546875 0.398611111 0.022656250 0.075000000
0 0.157812500 0.371875000 0.019531250 0.040972222
0 0.588281250 0.384722222 0.032031250 0.052777778
2 0.155859375 0.466319444 0.011718750 0.028472222
```

(d) Annotation for figure 6b

Figure 6: Examples of raw dashcam frames (figures 6a and 6b) that were used for training the YoloV5 model and their labels in Yolo format (figures 6c and 6d)

As mentioned in section 1.3.1, YoloV5 is able to perform a number of data augmentation techniques. Figure 8 shows what this looks like in practice on this dashcam image dataset. The annotated road signs have been plotted along with their label numbers. The Mosaic augmentation technique mixes four original frames and places them on a gray background. This way, the resulting image has a rectangular

Figure 7: Visualization of the Yolo Annotation Format
source: `https://github.com/ultralytics/yolov5/wiki/Train-Custom-Data`

shape of known height and width (in pixels). It also looks like some adjustments to the HSV values have been made for this image.

Figure 8: YoloV5 augmented mosaic of dashcam images

2.1.2 Data for 3D CNN

For the 3D CNN classification model, we preprocessed the data by making cutouts of the bounding boxes that are defined in the Yolo annotations. We need to rescale the images so that we have uniform input dimensions that can be processed by the CNN. This resulted in 444 images of 160x160 pixels. Eight examples out of the total 444 images are shown in figure 9.

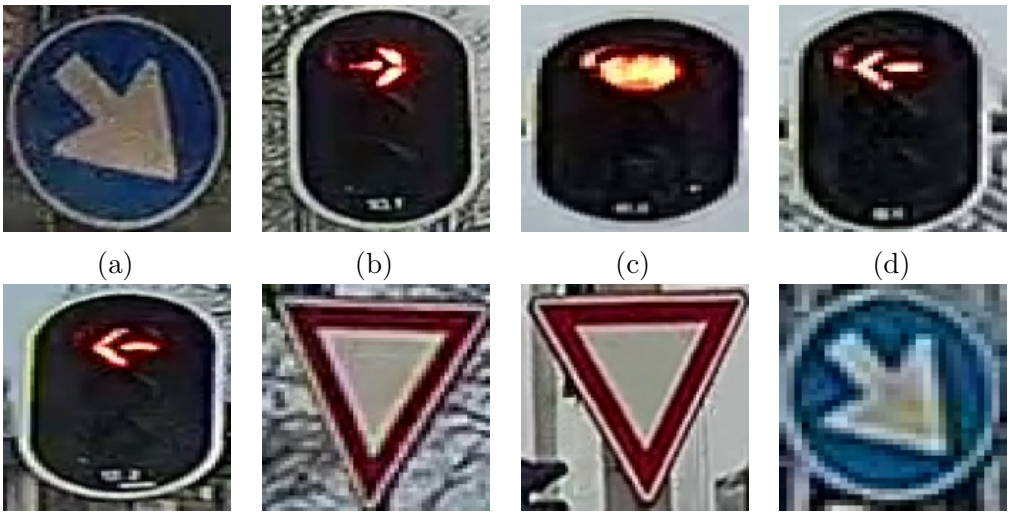


Figure 9: Examples of bounding box cutouts of the original frames in figure 6. 9a originates from fig. 6a, 9b - 9h originate from fig. 6b

2.1.3 Data for 4D CNN

In order to extract temporal information from the data, we distill sets of frames around the reference frames (which coincide with the data for the 3D CNN) from the raw video material. Like the data for the 3D CNN, the frames are cutouts from the original images, scaled to 160x160 px. We define three hyperparameters for the distillation procedure:

- Frame stride is defined by the number of frames that we leave out between two frames. The dashcam records videos at 60 frames per second. Using a frame stride helps when two consecutive frames at 60fps are too similar, and reduces redundant information. He et al. (2022) call this sparse feature aggregation (SFA) with a fixed stride. They even propose a temporal-adaptive SFA, where an adaptive number of frames is dropped, based on their similarity. We will use a fixed stride in this research, but tune this hyperparameter through grid-search.

- Window length is simply the amount of frames within each set. We take an equal amount of frames on both sides of the reference frame. Together with the reference frame itself, this therefore always results in an odd value for window length.

- Region of Interest (ROI) is the factor with which we increase the width and height of the bounding box in the reference frame. Since we are making cutouts of the bounding box in each frame, we need to increase the size of the cutout

to prevent road signs from falling out of the new frame.

Figure 10 visualizes these three hyperparameters.

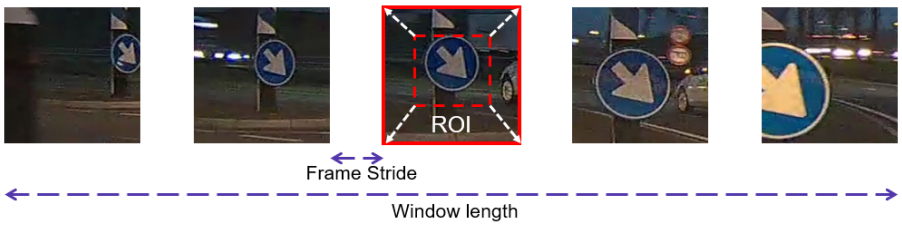


Figure 10: Hyperparameters frame stride, window length, and region of interest (ROI), used in the 4D CNN. The frame in the center is the reference frame.

2.2 Models

Three different architectures will produce three different models:

1. YoloV5s, as introduced in section 1.3
2. 3D CNN classification model for single images
3. 4D CNN classification model using temporal information fusion

This section describes the implementation of the architectures that produce these three models. A summary and visualization of both the YoloV5 network and the networks created in this research can be found in Appendix A.

2.2.1 YoloV5s

To get an idea of how well current state-of-the-art models perform on this dataset, we train YoloV5s on our data. The *s* in YoloV5s stands for small. YoloV5 comes in different sizes. Larger models contain more parameters. This improves the learning capacity of the model and will provide better results in nearly all cases, if the dataset is sufficiently large. It comes at the cost of longer run times and requires more GPU memory. For this research, we deemed the small network the best match for the available data. A large part of the layer structure in YoloV5s was used as inspiration for the 3D CNN. We will use the hyperparameter settings that have been pretrained on the COCO dataset for training the YoloV5s network. These settings are specified in appendix C. Using these settings means that a selection of the possible data augmentation techniques is used, among which Mosaic, Augment HSV and some Random Affine transformations. The weights will be trained during 300 epochs.

YoloV5s automatically keeps track of training and validation loss. It also computes metrics like precision, recall and mean average precision (mAP) after each epoch. When convergence is reached, or the set amount of epochs have been completed, the best model is saved. A confusion matrix is stored, as well as graphs for the F_1-score, and Precision/Recall curves. More information on these metrics can be found in section 2.3.

2.2.2 3D CNN

There are two main reasons for developing our own 3D CNN, instead of trying to incorporate the temporal dimension right away. Firstly, adapting the YoloV5 framework into a 4D version is too complicated to fit within the scope of a thesis research. YoloV5 is an elaborate framework that involves a multitude of concepts that improve and enhance the model. It would take a lot of time to fully comprehend the code and concepts therein. And if we would succeed in adapting the code, the performance comparison might be influenced or obscured by one or more elements of the framework. Alternatively, developing a 3D CNN first helps to control complexity and establish a baseline for incorporation of temporal information. Therefore, we decided to start with developing a 3D CNN. Once this model is able to classify, adjustments towards including the temporal dimension would be less complicated. Also, comparing our own 3D CNN with the 4D CNN would be a fairer comparison than comparing the 4D CNN with YoloV5. Throughout the research, we try to keep the building blocks of the model similar to YoloV5. For starters, the layers within the backbone and neck of the CNN are almost the same. The major difference between YoloV5 and our 3D CNN is the classification head. YoloV5 performs an elaborate object detection where a number of bounding boxes are fitted and then aggregated into a single best fit, which is done on three levels of granularity by using anchor boxes. This all happens simultaneously with the classification itself. To simplify this, we decided to make cutouts of the image within the bounding box. These cutouts, together with the class labels form a generic classification problem. This problem can be solved with a classification head consisting of fully connected layers and a SoftMax function on top of the backbone and neck from YoloV5. Also, the three levels of granularity are reduced to one. Other differences are mainly optimization and performance boosting mechanisms that have been omitted in the 3D CNN. All data augmentation methods enlisted in section 1.3.1 are omitted, as well as the training strategies from section 1.3.2. Instead of the advanced learning rate scheduler that YoloV5 uses, a simple form of warmup and LR scheduling was implemented. Two LR hyperparameters were used: a low LR (1e−4) for three warmup epochs, as

well as for the second half of the training time, and a high LR (1e−3) for the first half of the training time, after the three warmup epochs. So in case of 100 epochs, during epoch 1-3 the LR= 1e−4. During epoch 4-50 the LR= 1e−3. Finally, during epoch 51-100 the LR= 1e−4 again. The reason for this LR schedule is straightforward. During warmup, the objective is to avoid reaching an early local optimum. Therefore the LR is kept low. After warmup, the LR is increased to accelerate the training and move faster towards the (global) optimum. After some time, we decrease the LR again to avoid stepping back and forth over an optimum due to a large step size. Only a single loss function is used: Negative Log Likelihood Loss (NLLLoss), as opposed to the three different types of loss used by YoloV5. This is because this model does not perform any localization and objectness scoring. NLLLoss is often used in multiclass classification problems and works well with both balanced and unbalanced datasets. In the early stages, before balancing the dataset, NLLLoss was picked as the preferred loss function for these reasons. Because we use NLLLoss, the SoftMax function in the classification head becomes a LogSoftMax function. In order to prevent the model from overfitting on the data, early stopping criteria were defined. These criteria define when the model has converged to a point where no further training is required. If we keep training the model beyond this point, over-fitting occurs: the neural network becomes very good at predicting the classes for the images in the training data, but gradually becomes worse at classifying new images. The following criteria are used in this research for both the 3D and 4D CNN:

1. Train loss + Convergence threshold <= Best Train loss
2. Validation loss <= Best Validation loss
3. Validation accuracy > Best Validation accuracy

When criteria 1 and either one of criteria 2 and 3 are true for some epoch E, then epoch E is considered the new best epoch. If the best epoch does not change during the next 30 epochs, we stop training the network and store the model parameters that were in the model during the best epoch. This patience parameter P, for which we have chosen the value of 30 epochs, can also be optimized. If P is decreased, one risks to stop the training too early; the network might escape a local minimum and improve further in an epoch beyond P. If P is increased, one risks to train the model for an unnecessary long time. As the amount of runs grows, minimizing P gets more interesting in terms of reducing training times.

2.2.3 4D CNN

In order to include a temporal dimension in our model and process sets of frames instead of single images, adjustments to the 3D CNN need to be made. Firstly, the data need to be shaped into 4D tensors, representing height, width, color (RGB), and time. The time axis is embodied by a number of frames in chronological order around the reference frame. We can tune this by defining the window length, as discussed in section 2.1.3. Secondly, the neural network needs to be able to handle the temporal dimension. For this, we use the depth in 3D Convolutional layers. These layers process tensors with dimensions for height, width, depth and channels. We add a dimension to the kernel size (k), stride (s) and padding (p) parameters in order to make convolutions over the depth dimension as well. This creates cube-shaped kernels that summarize the information on the road signs across the frames in the temporal window. Because we have a variable window length, either a set of (k, s, p) parameters that is compatible with all values for window length needs to be found, or dynamically adjustable parameters need to be implemented. The second option was chosen to ensure that the network reduces the depth, so summarizes the temporal information, into a tensor with depth 1. In image classification using CNNs, small, odd kernel sizes (e.g. 3x3x3) are often preferred (Sood and Singh, 2022). Depending on the window length (= tensor depth), the kernel depths of the first three convolutional layers are either 1, 3 or 5. Kernel depth = 1 is used in case the earlier convolutions already reduced the depth to 1. In table 2, these values are specified.

Table 2: Kernel depths in first three Conv layers for each window length

	Window length			
Layer	5	7	9	11
Conv 1	3	3	5	5
Conv 2	3	3	3	5
Conv 3	1	3	3	3

We keep the kernel stride at 1 in the depth dimension for all convolutional layers, since the depth is relatively small. Also, we already pre-process the stride by dropping intermediate frames, as mentioned in section 2.1.3. This is much more memory-efficient, since the intermediate frames are not loaded into the 4D tensors, but omitted preemptively. No zero-padding is used. Usually, this is done to use the outer pixels on the edges and corners of images as many times as the inner pixels, in order to keep their influence on the outcome on the same level. Here, in the temporal

dimension, it is probably a good idea not to correct for this, because we expect the outer frames in a temporal window to be less important than the inner frames.

2.3 Performance Metrics

The performance can be measured in a variety of ways. The choice for which metrics are used is based upon literature and on the metrics that are used in YoloV5. The main performance measures used to compare different models are precision, recall, and F_1-score. The F_1-score is the harmonic mean of precision and recall. Precision and recall can be calculated from the number of true positives (n_{TP}), false positives (n_{FP}), and false negatives (n_{FN}). "A traffic sign instance is regarded as the true positive if the center of the detected bounding box falls in the ground-truth bounding box and the classes are the same. n_{FP} equals to the difference between the number of all detected traffic signs and n_{TP}. n_{FN} equals to the difference between the number of all traffic sign ground-truth and n_{TP}" (Li et al., 2018). Because the classification problem is multi-class, we need to average the individual F_1-scores of the classes to find the overall F_1-score of a model. Therefore, we will use $F_{1,macro}$, which is the unweighted mean of the F_1-scores for the individual classes.
N.B. in single-label classification problems, like we have for the 3D and 4D CNNs, $F_{1,micro}$ is essentially the same as accuracy. Communicating both $F_{1,micro}$ and accuracy would be redundant. The last option, $F_{1,weighted}$, takes the support for each class into account. Since we have balanced classes in this dataset, $F_{1,macro}$ and $F_{1,weighted}$ will provide the same results.
To gain a better insight in the additional value that temporal information can bring, another aim of this research is to get a better view of the performance of the current model. High rates of false positives and false negatives indicate much room for improvement. An F_1-score close to 1 would show that very little improvement is expected from temporal information, as the detection model already performs well. Besides these we use a confusion matrix to gain insights in which road signs seem similar or which classes might cause potential problems.

2.4 Inference on New Data

A useful experiment to test the robustness of the trained models would be to use a new dataset and predict the classes of the road signs therein. A different dashcam would almost always produce slight differences in resolution, orientation, or lighting of the videos. In this research, a dashcam compilation video was downloaded from YouTube (Dashcam NL, 2021). A number of random frames was collected from the video and useful frames containing one or more of the required classes were selected.

Five frames were selected, on which in total 28 road signs within the four classes could be distinguished. After the first run, 6 signs were omitted due to a very low resolution, or because they partly fell off of the frame. The final dataset therefore contained 22 reference frames. Table 3 shows the distribution across the classes.

Using the same Yolo annotation tool, bounding boxes were drawn and class labels

Table 3: Class Distribution of the Data for the YouTube Experiment

Class	**Label**	**Count**
0	Give Way	5
1	Speed Limit	3
2	Keep Right	2
3	Traffic Light	12

were assigned. With these ingredients, using the same data pipelines as before, the data is ready to be processed by the 3D and 4D CNNs. Because the frame stride and window length are depending on the frame rate of the video, we have to make sure these are similar to the frame rate of the dashcam used for the original dataset. The YouTube video has a duration of 493 seconds. The total amount of frames in the video is 11839. $\frac{11839}{493} \approx 24$ fps. This is likely to be due to the compression of the file by YouTube. We can correct for this by halving the frame stride. This would result in the same sets of frames in each temporal window as if the video would contain twice as much frames per second. As a result, the difference in frame rate is a lot smaller (48 vs. 60) compared to using a frame stride of 2.

3 Results

In this section, we show the results of the hyperparameter tuning for the 3D and 4D CNNs. The best hyperparameters for the 3D CNN are used for both CNNs. For the 4D CNN, only three hyperparameters are tuned: window length, frame stride, and Region of Interest. Then, we will compare these models to estimate the increase in performance that temporal information fusion brings. The performance is then put into perspective by comparing it with the results of the YoloV5 model.

3.1 Hyperparameter tuning 3D CNN

In the early stages of this research, we aimed to develop a 3D CNN that shows sufficient performance to analyze the improvement due to temporal information fusion. The coordinate descent strategy was adopted to tune the hyperparameters towards a better performing network. This means that we would change a single

parameter value, keeping the other parameters fixed, and run the training again. If the accuracy improved and the loss decreased, we kept the new value. Otherwise, the value was changed back to its previous state. Six hyperparameters were changed this way, increasing the accuracy from 42.0% to 88.4% on the initial dataset of 189 images. The biggest improvements occurred when changing the optimizer from Stochastic Gradient Descent (SGD) to AdaGrad, and after decreasing both learning rates (high and low) by a factor 10. A detailed table of the hyperparameter tuning runs can be found in Appendix D.

The accuracy and loss progression over the epochs during the best run is shown in figure 11. After 30 epochs, the validation loss starts to rise again. This usually indicates overfitting on the data. However, the validation accuracy does not drop, contrary to expectation.

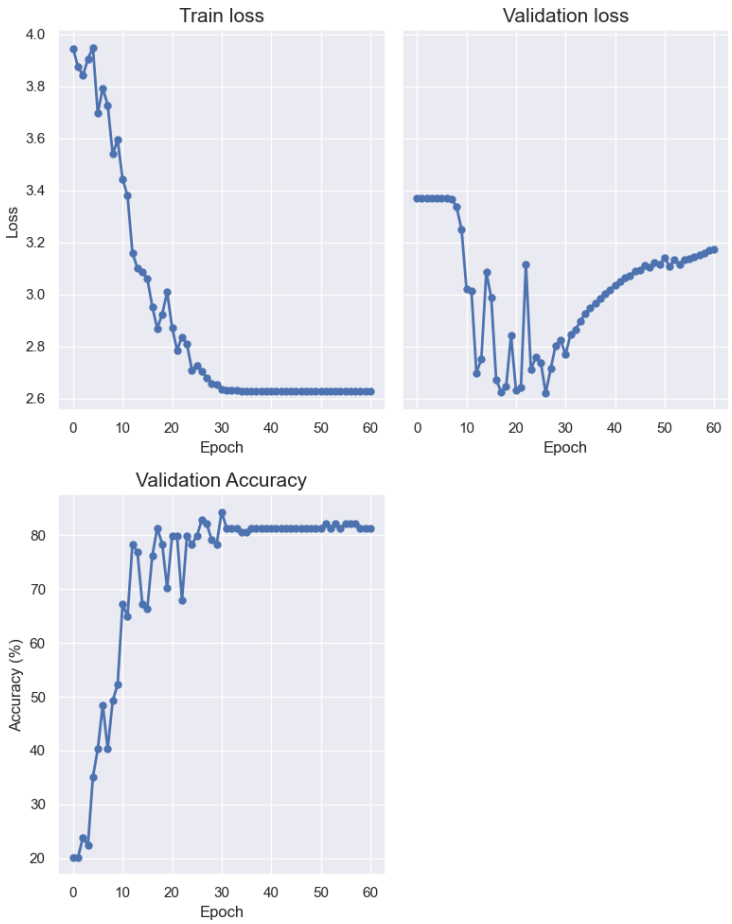


Figure 11: Training and validation loss, and validation accuracy progression of best hyperparameters run for the 3D CNN

During epochs 30 through 60, the graphs nicely show the purpose of early stopping, as described in section 2.2.2. Without early stopping, the training would keep running until the maximum amount of epochs is reached. The training loss is not likely to

decrease even further, the validation loss will probably keep rising, and the accuracy would stay the same, at best.

3.2 Hyperparameter tuning 4D CNN

Our main objective of tuning the hyperparameters for the 4D CNN is to explore in what way the temporal information in the data is used optimally. Therefore, only the hyperparameters regarding the temporal aspect are tuned. These are: Window length, Frame stride, and Region of Interest. The other hyperparameters of the model are based on the hyperparameter tuning of the 3D CNN. It is likely that some additional optimization is possible when tuning all hyperparameters simultaneously. However, since the amount of possible combinations would explode, we opted to leave that outside of the scope of this research. The values for the three temporal hyperparameters are shown in table 4. They were chosen to cover a large part of the parameter space, while trying to keep a manageable amount of combinations. Some explorative pre-processing was done to examine the outer limits of the parameter space we deemed likely to be an effective time/effort trade-off. The length of the temporal window largely influences the amount of memory that is needed for the data. A larger length means more frames around the reference frame, and therefore more memory per item. The minimal possible length is 1, but this would omit the idea of temporal information completely. Because we want to keep the amount of frames before and after the reference frame equal, the next possible window length is 3. We expected that the temporal information drawn from only three frames would be small, so this value was also omitted. From literature, we found a research that aggregating the temporal information across 7-9 frames yields the best performance (He et al., 2022). This led us to focus on values around 7 and 9 frames of window length.

Table 4: Temporal hyperparameter tuning values

Hyperparameter	**Values**
Window length	{ 5, 7, 9, 11 }
Frame stride	{ 2, 5, 10, 20, 30 }
Region of Interest	{ 1.2, 1.5, 1.8, 2.1 }

The frame stride determines how many raw video frames are being left out between two consecutive frames in the temporal window. This means that for low values, the frames are more similar and grow increasingly more dissimilar as the frame stride increases. We based the values of this hyperparameter on viewing a small sample of the data for different values. Because the dashcam shoots videos at 60 fps,

choosing low values often results in frames that are almost identical. This would be problematic, because feeding a neural network the same image multiple times only results in bias towards the data. It does not help in building a general idea of what a certain type of road sign looks like. On the other hand, choosing too high values for frame stride may result in road signs (partially) disappearing from the frame. This introduces noise in the data, because the model is told that there is a road sign present, while in fact, the road sign is only present in the middle set of frames. If the frame stride is extremely large, it may even occur that the road sign only remains visible on the reference frame.
The Region of Interest (ROI) defines the window around the road sign on the reference frame that is used across all frames in the temporal window. Increasing the ROI also increases the likelihood that the outer frames in the temporal window still contain the road sign. However, it also increases the amount of background around the road signs, which is effectively noise in the data. Naturally, it feels like there should be some optimal spot, where the road signs are still present in each frame and the amount of noise is kept to a minimum. In reality, this optimal spot is highly dependent on a number of factors. Firstly, increasing values for window length and frame stride are likely to work better in combination with a larger ROI. This is due to the movement of the dashcam towards the road sign. This causes the sign to grow larger and transition from the center of the image towards the outside (in the scenario of driving on a straight road). Secondly, it depends on the velocity of the car. With a high velocity, subsequent frames are less similar and can only contain the same road sign with a high ROI (and/or low frame stride). Alternatively, when the car has stopped, for instance due to a red traffic light, the frames remain extremely similar. Thirdly, the road layout largely determines how fast a road sign shifts across subsequent frames. For instance, when a car turns a sharp corner, the horizontal displacement of a road sign within the scope of the dashcam is much larger than on a straight road, when you can see it coming from a distance. There may be some bias in the data if the probability that a road sign is close to a sharp corner is larger for a particular type of road sign. For instance, a Keep right sign will rarely be present on a long straight road, but is often found on large intersections.

The full grid search for tuning these three hyperparameters contains $4 * 5 * 4 = 80$ runs. The ten best scoring runs are stored in table 5. For the full table of hyperparameter tuning results, see Appendix E.

From table 5, we see that the best performing model uses Window length = 7, Frame stride = 2, and ROI = 1.5. It reached an F_1-score of 0.83 and an accuracy of 83%. Within these ten best runs, we see all values for window length and ROI, but

only two out of the five values used for frame stride.

Table 5: Ten best grid search results with the 4D CNN, used for hyperparameter tuning

Run	$\mathbf{F_{1,macro}}$	**Accuracy**	**Length**	**Stride**	**ROI**
6	0.832	0.833	7	2	1.5
17	0.810	0.811	5	5	1.2
3	0.787	0.788	5	2	1.8
7	0.791	0.788	7	2	1.8
4	0.773	0.773	5	2	2.1
26	0.766	0.773	9	5	1.5
22	0.758	0.758	7	5	1.5
25	0.746	0.758	9	5	1.2
13	0.726	0.735	11	2	1.2
28	0.739	0.750	9	5	2.1

For the best performing run, run 6, the loss and accuracy progression is shown in figure 12. Here, similar behavior as in figure 11 is observed: as soon as the training loss stops decreasing, the validation loss rises. Here, the accuracy neither decreases as a result of the rising validation loss. During epoch 42-43, the model probably escapes a local minimum and improves further shortly thereafter. This would explain the sudden rise in training loss, drop in validation loss and accuracy, which recovers in the next couple of epochs.

3.3 Model Comparison

After tuning the hyperparameters, the three models (YoloV5, 3D CNN & 4D CNN) can be compared. Figure 13 shows the Precision-Recall (PR) curves for each model. Precision is the fraction of true positives among all positives. Recall is the fraction of true positives among all true instances. A high precision can be achieved by only classifying the instances for which the model is highly confident that they belong to specific class. At the same time, the model may fail to classify a large amount of all instances that in fact belong to that class. A high recall can be achieved by classifying all instances to a class: it has successfully classified all true instances. At the same time, it has falsely classified the instances of the other classes as belonging to the class. To find the best model, one needs to balance precision and recall. Therefore, we use PR curves to show how well a model has found this balance. A better model shows a PR curve that reaches the top right corner closer.
We expect all models to have curves that flow from (0,1) towards (1,0). Figures 13a and 13b do not show this trend, which means that they must have some predictions

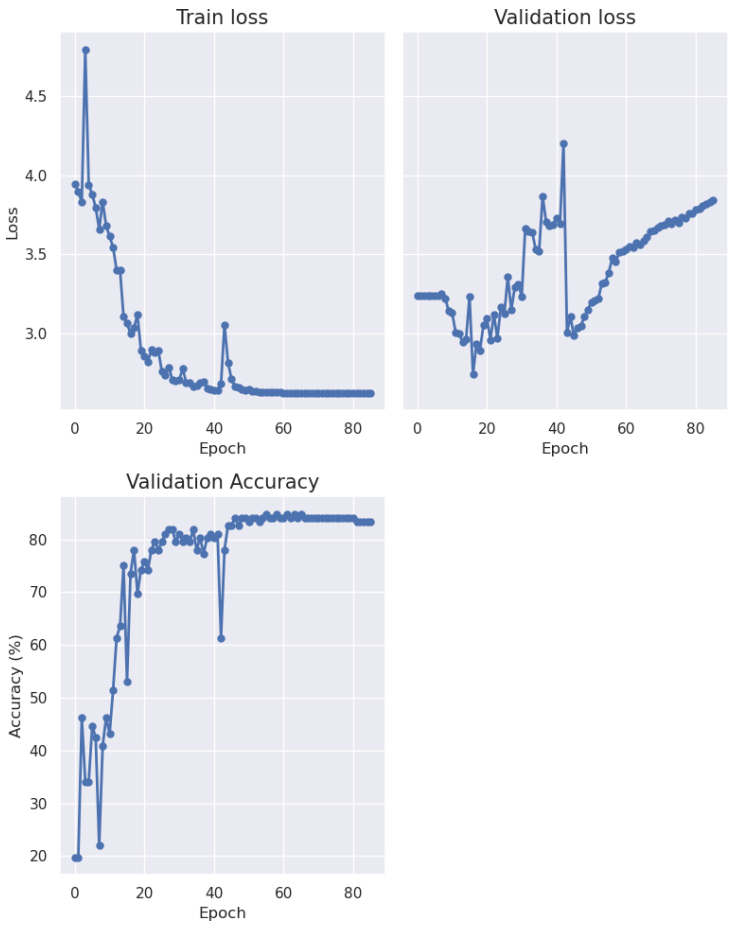
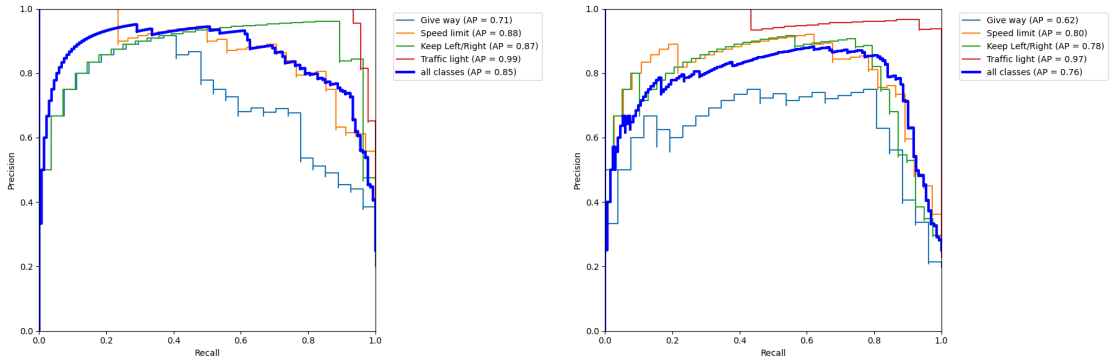


Figure 12: Training and validation loss, and validation accuracy progression of best hyperparameters run (run 6) for the 4D CNN

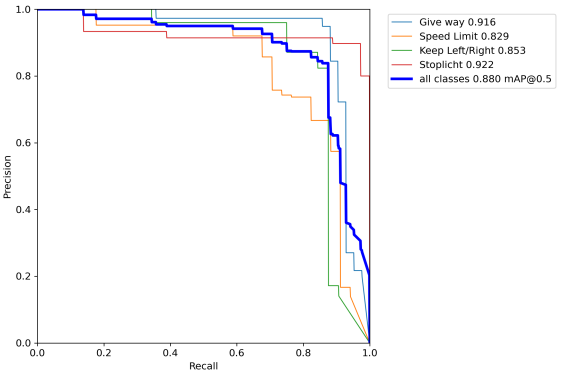
with a very high confidence that are false positives. Figure 13c shows the trend that we expect, which means that we can count on the confidence level for the predictions to be accurate.

From the PR curves, we get an idea of how well the models classify the four different road signs. To gain an even better understanding of the classification results, confusion matrices can be used. Besides the accuracy per class, these matrices show what pairs of road signs often are confused by the model. If two road signs are consistently mistaken, this could probably be solved by training the model further on data of those road signs. Alternatively, a closer look on the current data might show mistakes in the labeling. Figure 14 shows the confusion matrices for the three models. From figure 14a, we can tell that 'Give way' signs are often mistaken for 'Speed limit' signs by the 3D CNN. This could be explained by the similar color pattern of the signs. Additional data on these road signs might help the model in differentiating these signs, for instance based on their shape or the number that is present on speed limit signs. The bottom row shows that there are no false positives



(a) 3D CNN

(b) 4D CNN

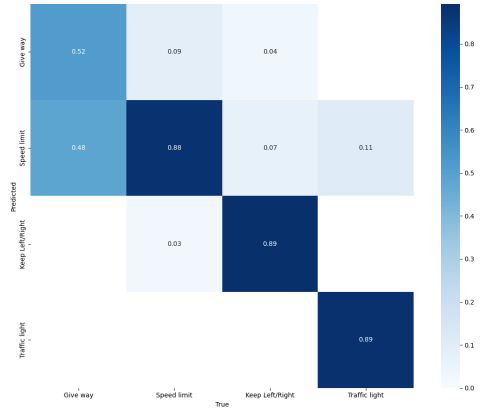


(c) YoloV5

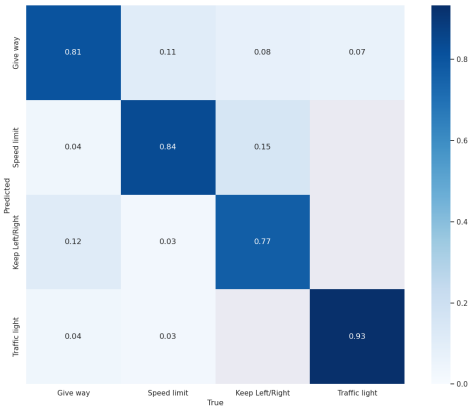
Figure 13: Precision-Recall curves for the three models. An ideal model shows curves that are plotted along the top and right side edges of the graph.

for 'Traffic light', so no images that contain either a 'Give way', 'Speed limit', or 'Keep right' was falsely classified as 'Traffic light'. This confirms the high Precision we saw in figure 13a for this class. Figure 14b does not show ambiguity in predicting one of the classes, like we saw in the 3D CNN. Again, we see a high score for traffic lights. Also, there is no sign of extreme overclassification of one of the classes: the false positives are quite equally distributed over the other classes. The confusion matrix for the YoloV5 model in figure 14c is slightly different from the other two matrices. The model can perform multi-object localization and classification. This means that it can also miss objects. Missing, or not noticing objects is represented as background false negative (background FN) in the confusion matrix. Similarly, the model may mistakenly localize and label objects on an image, while in fact the object does not belong to one of the specified classes. These cases count as background false positives (background FP). The confusion matrix shows that all FN and FP fall in the background category. This means that there are no road signs in the data that get classified as one of the other road signs in the data. It only fails to notice on average 12% of the road signs, and notices an extra 25% that does not belong to any of the four classes in the model. Just like in the 3D and 4D CNN, the model is

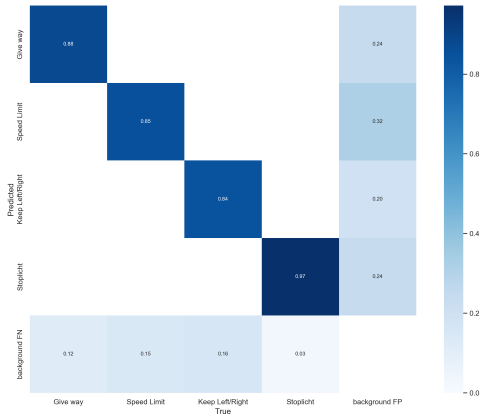
best at predicting traffic lights, compared to the other three classes.



(a) 3D CNN



(b) 4D CNN



(c) YoloV5

Figure 14: Confusion matrices for the three models. An ideal model shows 1.0s on the main diagonal.

All in all, looking at the accuracy and F_1-scores aggregated over all four classes, we come to the results in table 6.

Table 6: Metrics of Best Model Runs for Model Comparison

Model	**F_1-score**	**Accuracy**
YoloV5s	0.85	88.5%
3D CNN	0.80	81.3%
4D CNN	0.83	83.3%

3.4 YoloV5 Benchmark

In order to see how the 3D and 4D CNNs relate to current state-of-the-art object detection algorithms, we trained YoloV5 on the same dataset. The training proceeded for the full 300 epochs, so not all early stopping criteria were met. Looking at the loss

graphs in figure 15, we see training losses converging towards zero. The validation Box and Classification losses also show decreasing trends, whereas only the validation Objectness loss goes up in the later stages of training. The precision and recall metrics stop increasing around 0.85, even though there seems to be quite some variability throughout epochs 100 and onward.

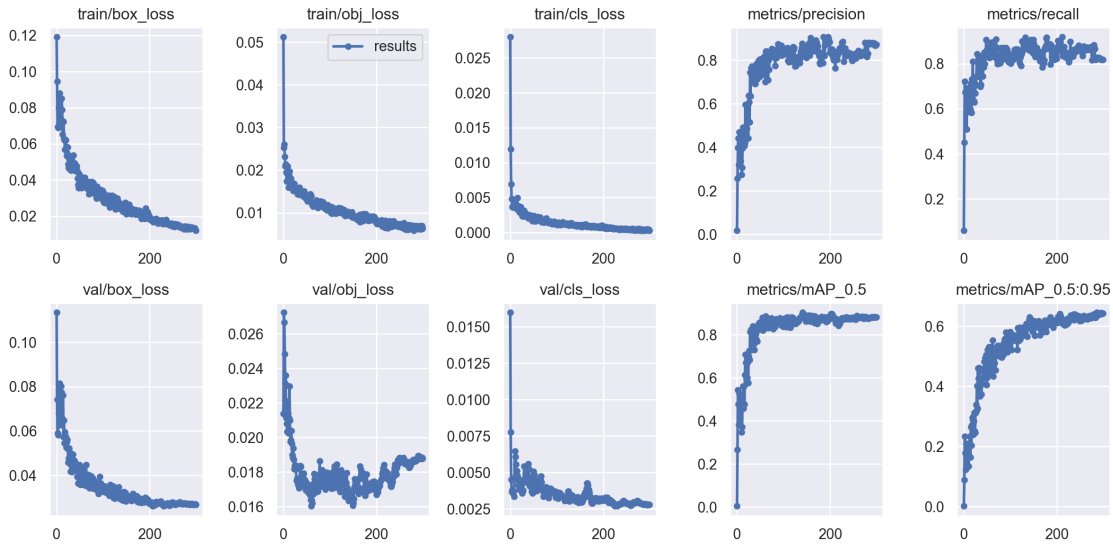


Figure 15: Loss and performance metrics for the YoloV5s benchmark run

YoloV5 also directly gives insight in predictions on a small sample of the validation set. The bounding boxes and labels are plotted on the original image for both the 'true' data and the predictions. The predictions also come with a confidence score between 0 and 1. Two instances of this validation sample are depicted in figure 16. On the left side, the predictions correspond closely with the annotations. Both visible road signs receive a confidence score close to 1, which means that the model is highly confident that the inferred bounding boxes contain the correct road signs. On the right side, some data is missing. The original frame clearly contains two 'Give way' signs and one 'Keep right' sign. This instance was probably added through inference using the pretrained YoloV5 model, as explained in section 2.1. The additional data only contains inferences with a confidence score of at least 0.9. As we can see in figure 16f, the 'Keep right' sign gets recognized with a confidence score of 0.7. The other 'Give way' sign, which is a bit further away, was not annotated and is also not detected by the model.

3.5 Inference on New Data: YouTube Experiment

Using the model parameters from the best runs of the 3D and 4D CNNs, inference was made on the small data sample of 22 reference frames from the YouTube dashcam

(a) Original (d) Original

(b) Annotation (e) Annotation

(c) Prediction (f) Prediction

Figure 16: Examples of accurate annotation and prediction (left) and incomplete annotation and prediction (right) using the YoloV5 model

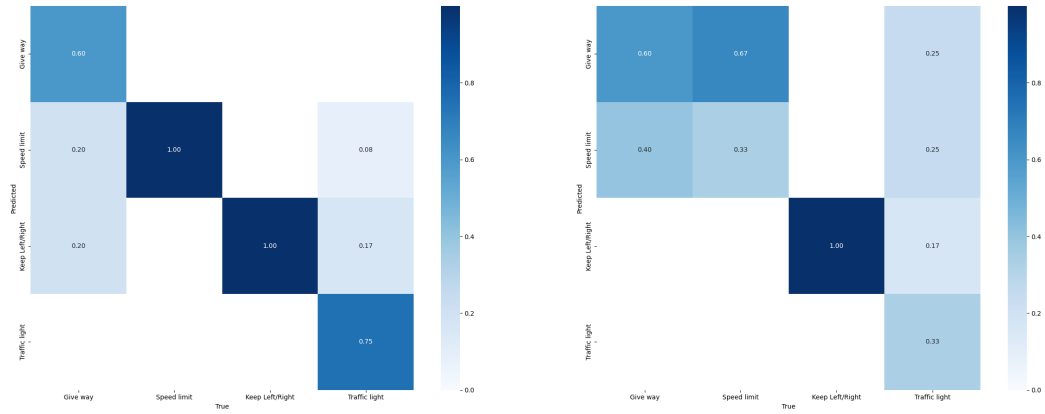
video. The results in table 7 show that the 3D CNN model is significantly more robust that the 4D CNN. It also confirms the hypothesis made in section 2.4, that using frame stride 1 is more in line with the trained model than using a frame stride of 2 frames. Looking at the confusion matrices in figure 17, we see that both models

Table 7: Results for the YouTube Experiment

Model	**Stride**	**Accuracy**
3D CNN	-	77.3%
4D CNN	2	36.4%
4D CNN	1	45.5%

correctly classify all 'Keep right' signs, but have difficulty predicting the 'Give way' signs and Traffic lights. A remarkable difference is observed in the 'Speed limit' class. The 3D CNN correctly classifies all instances, whereas the 4D model performs much worse. Upon closer inspection of the data, two out of the three 'Speed limit'

signs in this sample were quite far away from the dashcam and therefore have a low resolution. Because the 4D CNN uses sets of 7 frames each, it suffers 7 times from a low resolution, instead of only once in the 3D CNN. This might explain the difference in predictions for this class.



(a) 3D CNN (b) 4D CNN with stride 1

Figure 17: Confusion matrices for the YouTube Experiment.

4 Discussion

This research shows evidence that using temporal information fusion in CNNs helps in automated road sign detection. The methods aimed to make a minimalistic comparison between two models with and without temporal information fusion. We refrained from advanced strategies like data augmentation, Cosine LR schedulers and Exponential Moving average to avoid interference with the effect of interest. Using this approach, the results show a 2% increase in accuracy of the model, improving the single image model (81.3%) through temporal information fusion (83.3%). The state-of-the-art benchmark model, YoloV5, that was trained on the same data, shows an accuracy of 88.5%. As expected, it outperforms the 3D CNN. The F_1-scores show similar results. This similarity is likely due to the balance of the classes in the data. A common notion across all three models is the high performance on the 'Traffic light' class.

4.1 Possible Improvements and Further Research

During this research, some directions for further improvement were discovered. The temporal hyperparameters can be tuned further, at least for Frame stride and ROI. The parameter space for Frame stride between 2 and 5 frames and a Region of Interest varying with steps of 0.1 or even smaller can be explored to boost the performance

even further. Also the interactions with other hyperparameters can be studied. In this research, we only used images of 160x160 pixels. Changing this can also impact the performance of the model. The amount of channels after the first convolution can also be increased in order to increase the learning capacity of the neural network. These options were left outside the scope of this research. We expect the impact that these changes have will be similar for the 3D and 4D CNNs. However, there might be some interaction with the temporal nature of the difference between these two models.

Section 2.1.3 mentioned a research by He et al. (2022), which proposes a temporal-adaptive frame stride. Dashcam analysis is complicated by movement of the camera at different speeds, sometimes remaining stationary for a short while. When stationary, the frame stride should be higher compared to situations moving at high speed, in order to have the same amount of change between subsequent frames. Thus, further research could explore an adaptive frame stride, based on the speed of the car at the time of capturing the video. If this causes the change between subsequent frames to be similar for all data instances, it would likely enable further optimization of the ROI.

While evaluating the loss plots for the 3D and 4D CNNs, as mentioned in section 3.1, we noticed increasing validation losses, while maintaining high accuracy scores. Looking into this behavior to find out what causes this could prove useful in improving the model. The training loss graph in figure 12 shows a loss spike after epoch 3. Because a constant low learning rate is used during warmup, this spike can be expected as soon as the learning rate grows by a factor 10. Goyal et al. (2018) propose a gradually ramped learning rate during warmup. This ramp avoids a sudden increase of the learning rate, allowing healthy convergence at the start of training.

Another useful addition to the current setup could be to use metrics for top-X classifications. Currently, only the single highest predicted class (top-1) is used to estimate the performance of the model, e.g. through the accuracy. However, if the model classifies some road sign image with probability scores of 45% for 'Speed sign' and 48% for 'Give way', it feels naive to measure this the same way as an image with scores of 3% 'Speed sign' and 94% 'Give way'. With the top-1 approach, this would be the case. Using multiple metrics for both top-1 and top-X, or using a weighted metric that takes the predicted probability scores into account, could give better insight in the models performance.

To make a fair comparison between the 3D and 4D CNNs, we could consider to use all of the frames in the sets of frames used for the 4D CNN, statically for training

the 3D CNN. Currently, the 4D CNN makes use of a lot more data compared to the 3D setup, although the dataset also contains much more noise. The impact of this noise, when used in the static image 3D CNN environment, can be quantified when we compare 3D CNNs trained on the full dataset or on the dataset containing only the reference frames.
Due to using semi-automatic image annotation, as mentioned in section 2.1 and visualized in section 3.4, the data contains errors and missing items in the annotations. Since the same dataset is used for all three models, we do not expect this to have a large impact on the results. If the data quality would have been higher, this would likely boost the performance across all three models. Annotating all images by hand would have been better. However, due to the time-consuming nature of annotating data by hand, we opted for this semi-automatic strategy.

4.2 Pros and Cons for using Temporal Information Fusion

The results show an increase in accuracy of 2.0% by using TIF. Even though this seems like a marginal increase, on a database of 3 million road signs this would mean an additional 60 thousand correctly classified road signs. Moreover, when put into the perspective of automated maintenance detection, this might drastically reduce the amount of road signs that manually need to be checked.
Data augmentation methods are much less developed for video classification, as opposed to static image classification. Yun et al. (2020) recently proposed VideoMix, but this is one of the few proven video data augmentation strategies. Data augmentation techniques are useful for harnessing against overfitting the model on the available data. Without a comparable arsenal of augmentation techniques, video classification might require significantly more data to avoid overfitting, compared to static image classification.
Processing sets of frames is slower compared to using single images. Although quantifying processing speeds for the 3D and 4D CNNs was outside the scope of this research, we developed an intuition for this increase in processing time. The cause is obvious: the amount of data for each instance (reference frame) is a factor equal to the length of the temporal window larger for the temporal dataset. E.g. if the window length is 7, each instance contains 7 times more data compared to a single frame in the 3D CNN. This has influence on the data preprocessing: extracting the frames from the raw video, on the amount of operations that are done in the convolutional layers during each forward pass through the network, and also increases the amount of weights in the model that have to be updated on every backpropagation step. Training a 4D CNN therefore takes a lot longer for the same amount of annotated

images than training a 3D CNN would take. To compare processing speeds for inference, some experiments could be done. These days, inference on static images with a trained 3D CNN can happen (almost) real-time. We can expect that inference using a 4D CNN takes longer, but experiments will have to prove how drastic the differences are.
Looking at the results of the YouTube experiment in section 3.5, the 4D CNN model has got some improvement to make in terms of robustness. However, due to the small size of the experiment, the outcome might be circumstantial. Larger or more experiments will have to take place in order to make stronger statements about the robustness.

5 Reference list

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. `https://www.tensorflow.org/`. Software available from tensorflow.org.

Albawi, S., Mohammed, T. A., and Al-Zawi, S. (2017). Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6.

Ayachi, R., Afif, M., Said, Y., and Atri, M. (2020). Strided Convolution Instead of Max Pooling for Memory Efficiency of Convolutional Neural Networks. In Bouhlel, M. S. and Rovetta, S., editors, *Proceedings of the 8th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications (SETIT'18), Vol.1*, Smart Innovation, Systems and Technologies, pages 234–243, Cham. Springer International Publishing.

Bochkovskiy, A., Wang, C.-Y., and Liao, H.-Y. M. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. arXiv:2004.10934 [cs, eess].

Buslaev, A., Iglovikov, V. I., Khvedchenya, E., Parinov, A., Druzhinin, M., and Kalinin, A. A. (2020). Albumentations: Fast and flexible image augmentations. *Information*, 11(2).

Carreira, J., Noland, E., Hillier, C., and Zisserman, A. (2019). A Short Note on the Kinetics-700 Human Action Dataset. arXiv:1907.06987 [cs].

Dashcam NL (2021). De grootste idioten op de nederlandse wegen! _ compilatie #52. https://youtu.be/_zgJqtqUtCM.

Elfwing, S., Uchibe, E., and Doya, K. (2018). Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11.

Ghiasi, G., Cui, Y., Srinivas, A., Qian, R., Lin, T.-Y., Cubuk, E. D., Le, Q. V., and Zoph, B. (2021). Simple Copy-Paste is a Strong Data Augmentation Method for Instance Segmentation. arXiv:2012.07177 [cs].

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. (2018). Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. arXiv:1706.02677 [cs].

He, F., Li, Q., Zhao, X., and Huang, K. (2022). Temporal-adaptive sparse feature aggregation for video object detection. *Pattern Recognition*, 127:108587.

He, T., Zhang, Z., Zhang, H., Zhang, Z., Xie, J., and Li, M. (2018). Bag of Tricks for Image Classification with Convolutional Neural Networks. arXiv:1812.01187 [cs].

Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Number: arXiv:1502.03167 arXiv:1502.03167 [cs].

Ji, S., Xu, W., Yang, M., and Yu, K. (2013). 3D Convolutional Neural Networks for Human Action Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):221–231. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

Jocher, G., Stoken, A., Chaurasia, A., Borovec, J., NanoCode012, TaoXie, Kwon, Y., Michael, K., Changyu, L., Fang, J., V, A., Laughing, tkianai, yxNONG, Skalski, P., Hogan, A., Nadar, J., imyhxy, Mammana, L., AlexWang1900, Fati, C., Montes, D., Hajek, J., Diaconu, L., Minh, M. T., Marc, albinxavi, fatih, oleg, and wanghaoyang0106 (2021). ultralytics/yolov5: v6.0 - YOLOv5n 'Nano' models, Roboflow integration, TensorFlow export, OpenCV DNN support. https://doi.org/10.5281/zenodo.5563715.

Li, D., Zhao, D., Chen, Y., and Zhang, Q. (2018). DeepSign: Deep Learning based Traffic Sign Recognition. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6. ISSN: 2161-4407.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. (2018). Focal Loss for Dense Object Detection. arXiv:1708.02002 [cs].

Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., and Dollár, P. (2015). Microsoft COCO: Common Objects in Context. *arXiv:1405.0312 [cs]*. arXiv: 1405.0312 version: 3.

Malik, M. I. (2018). ManzarIMalik/YOLO-Annotation-Tool. `https://github.com/ManzarIMalik/YOLO-Annotation-Tool`.

Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., and Wu, H. (2018). Mixed Precision Training. arXiv:1710.03740 [cs, stat].

O'Shea, K. and Nash, R. (2015). An Introduction to Convolutional Neural Networks. Number: arXiv:1511.08458 arXiv:1511.08458 [cs].

Qiu, Z., Yao, T., Ngo, C.-W., Tian, X., and Mei, T. (2019). Learning Spatio-Temporal Representation With Local and Global Diffusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12056–12065.

Ramachandran, P., Zoph, B., and Le, Q. V. (2017). Searching for Activation Functions. Number: arXiv:1710.05941 arXiv:1710.05941 [cs].

Ramoser, H., Muller-Gerking, J., and Pfurtscheller, G. (2000). Optimal spatial filtering of single trial EEG during imagined hand movement. *IEEE Transactions on Rehabilitation Engineering*, 8(4):441–446. Conference Name: IEEE Transactions on Rehabilitation Engineering.

Rangasamy, K., As'ari, M. A., Rahmad, N. A., Ghazali, N. F., and Ismail, S. (2020). Deep learning in sport video analysis: a review. *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, 18(4):1926–1933. Number: 4.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788.

Sakhavi, S., Guan, C., and Yan, S. (2018). Learning Temporal Information for Brain-Computer Interface Using Convolutional Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 29(11):5619–5629. Conference Name: IEEE Transactions on Neural Networks and Learning Systems.

Smith, L. N. and Topin, N. (2018). Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates. arXiv:1708.07120 [cs, stat].

Sood, S. and Singh, H. (2022). Effect of Kernel Size in Deep Learning-Based Convolutional Neural Networks for Image Classification. *ECS Transactions*, 107(1):8877. Publisher: IOP Publishing.

Sun, M., Song, Z., Jiang, X., Pan, J., and Pang, Y. (2017). Learning Pooling for Convolutional Neural Network. *Neurocomputing*, 224:96–104.

Thakkar, V., Tewary, S., and Chakraborty, C. (2018). Batch Normalization in Convolutional Neural Networks — A comparative study with CIFAR-10 data. In *2018 Fifth International Conference on Emerging Applications of Information Technology (EAIT)*, pages 1–5.

Wang, X., Girshick, R., Gupta, A., and He, K. (2018). Non-local Neural Networks. *arXiv:1711.07971 [cs]*. arXiv: 1711.07971.

Yun, S., Oh, S. J., Heo, B., Han, D., and Kim, J. (2020). VideoMix: Rethinking Data Augmentation for Video Classification. arXiv:2012.03457 [cs].

Zhang, H., Cisse, M., Dauphin, Y. N., and Lopez-Paz, D. (2018). mixup: Beyond Empirical Risk Minimization. arXiv:1710.09412 [cs, stat].

Appendices

A Neural Network Architectures

This appendix contains summaries and visualizations of the YoloV5 network and the networks created in this research.

A.1 YoloV5s Architecture Summary and Visualization

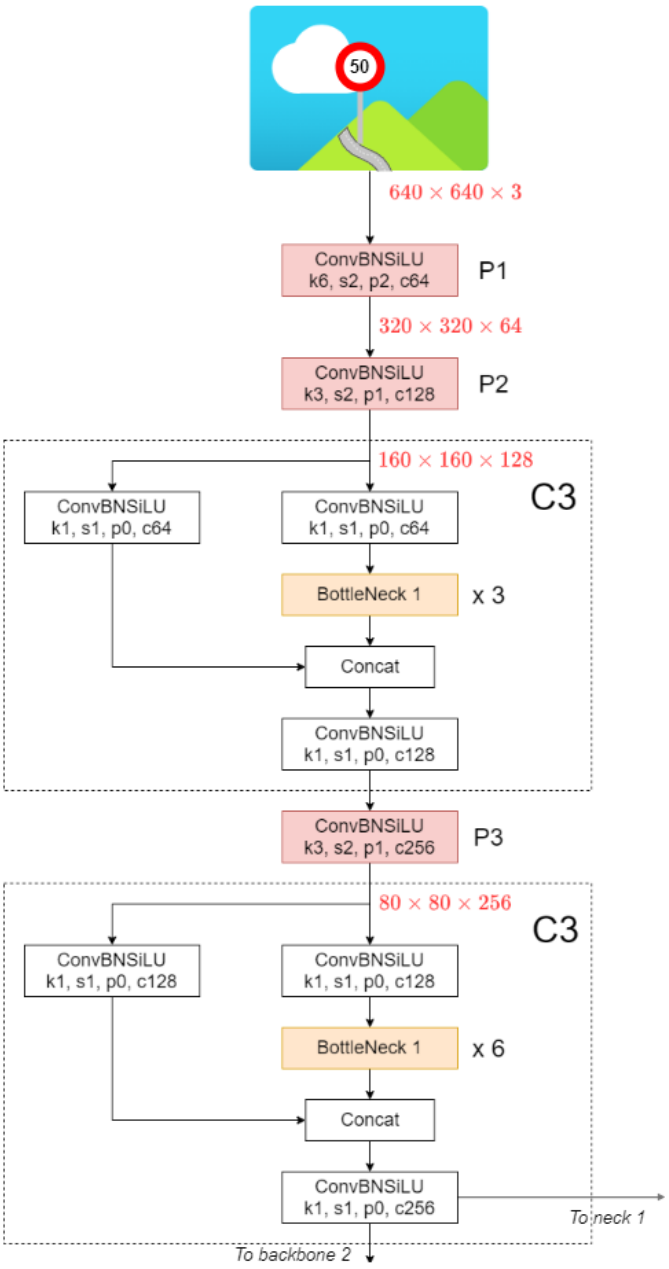


Figure 18: YoloV5s Backbone 1

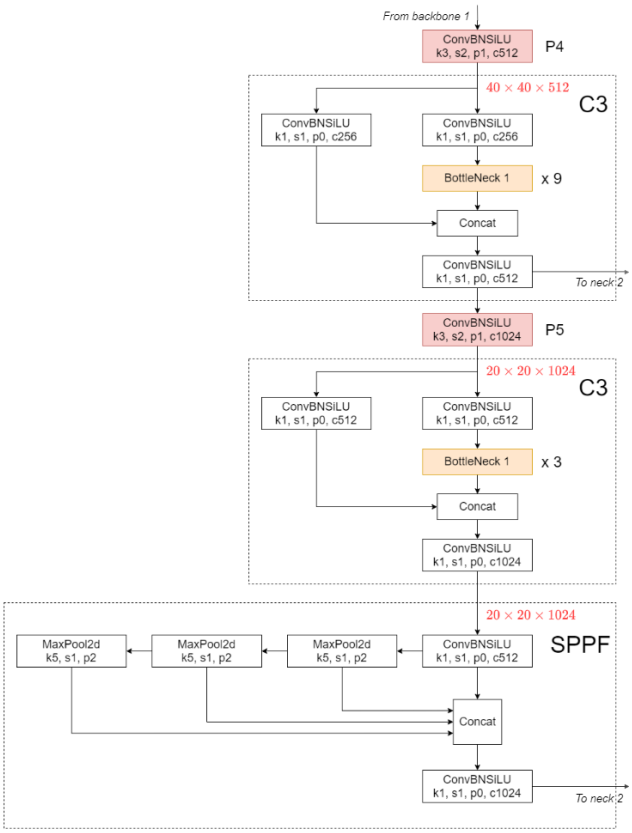


Figure 19: YoloV5s Backbone 2

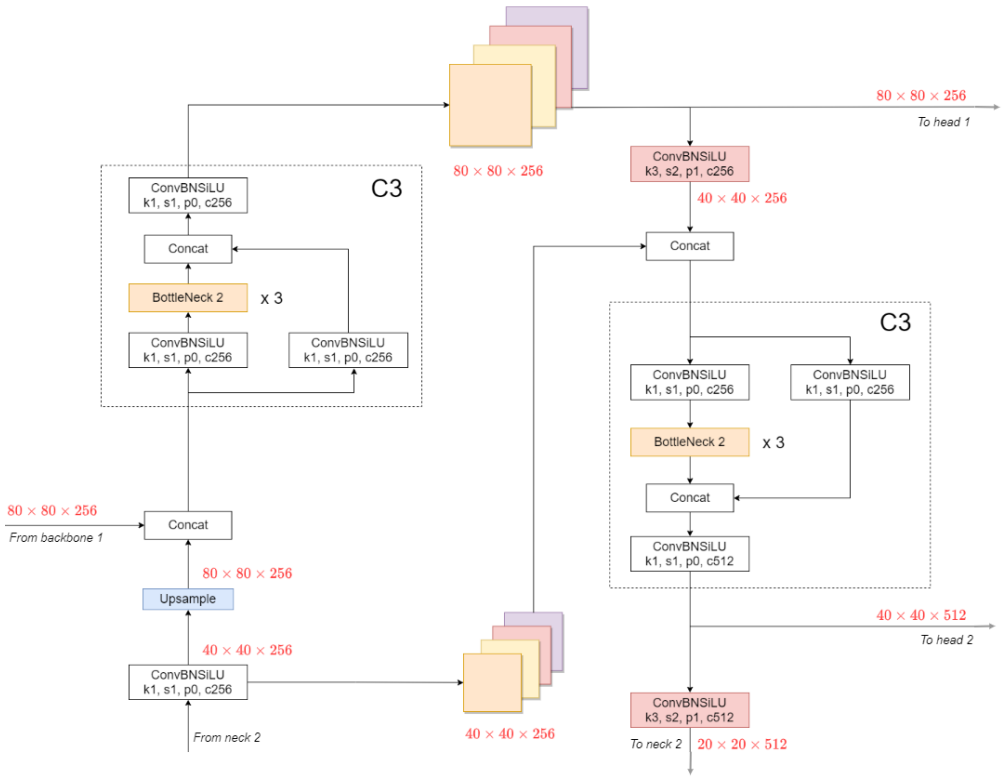


Figure 20: YoloV5s Neck 1

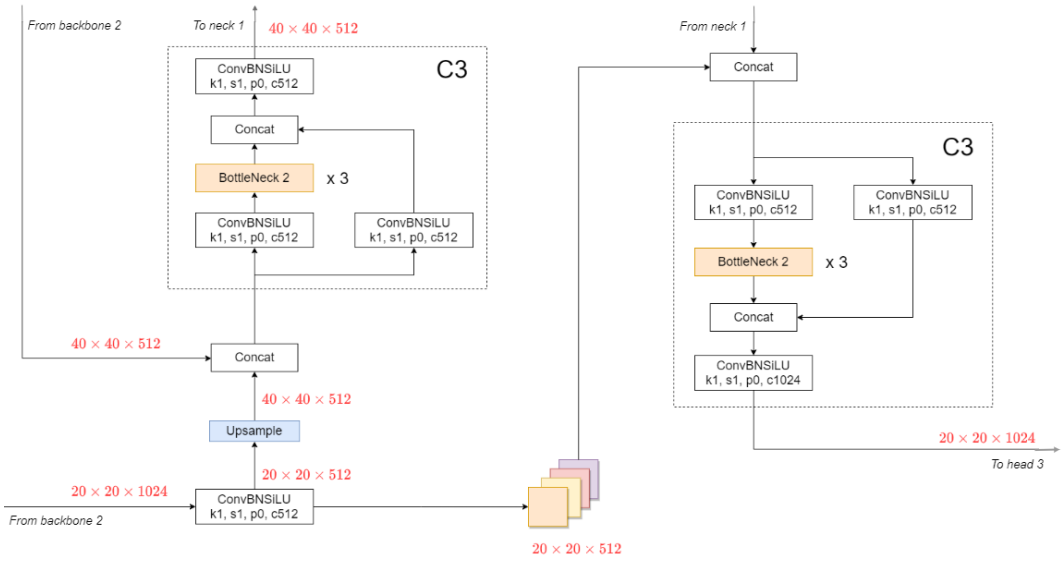


Figure 21: YoloV5s Neck 2

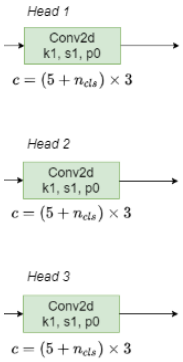


Figure 22: YoloV5s Head

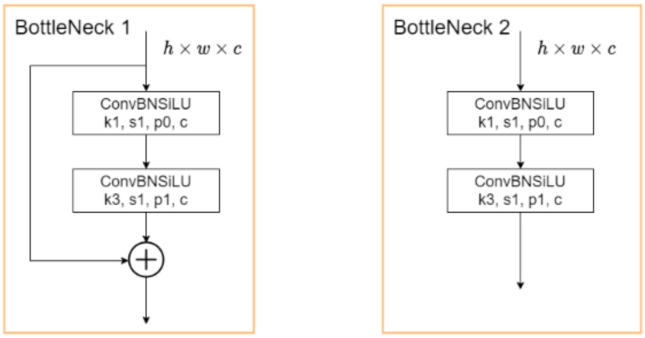


Figure 23: YoloV5s Bottlenecks

The entire YoloV5s architecture visualization is split up into six parts. The outgoing and incoming arrows are labeled to show how the parts are connected. A ConvBNSiLU layer is composed of three elements: a convolutional layer (Conv2d in PyTorch), a Batch Normalization layer (BatchNorm2d in PyTorch), and the SiLU activation function. The arguments k, s, p, and c represent kernel, stride, padding, and output channels, respectively. For instance, the first ConvBNSiLU layer has arguments k6, s2, p2, c64. This means the convolutional layer uses a kernel of shape 6x6 with stride 2 in both directions, the input is zero-padded with 2 rows and columns at each side, and the output has 64 channels.

The two types of bottlenecks in figure 23 are elements that recur throughout the architecture.

A.2 3D CNN Architecture Summary and Visualization

The 3D CNN has some adaptations with respect to YoloV5s. The input image is much smaller and the output channels in each layer are reduced by a factor 2. The structure of the backbone and neck is identical for YoloV5s and the 3D CNN. A major difference from YoloV5s is the network head in figure 28. The fully connected layers (called Linear in PyTorch) in the head are alternated by ReLU activation functions. The Bottlenecks remain the same as in figure 23

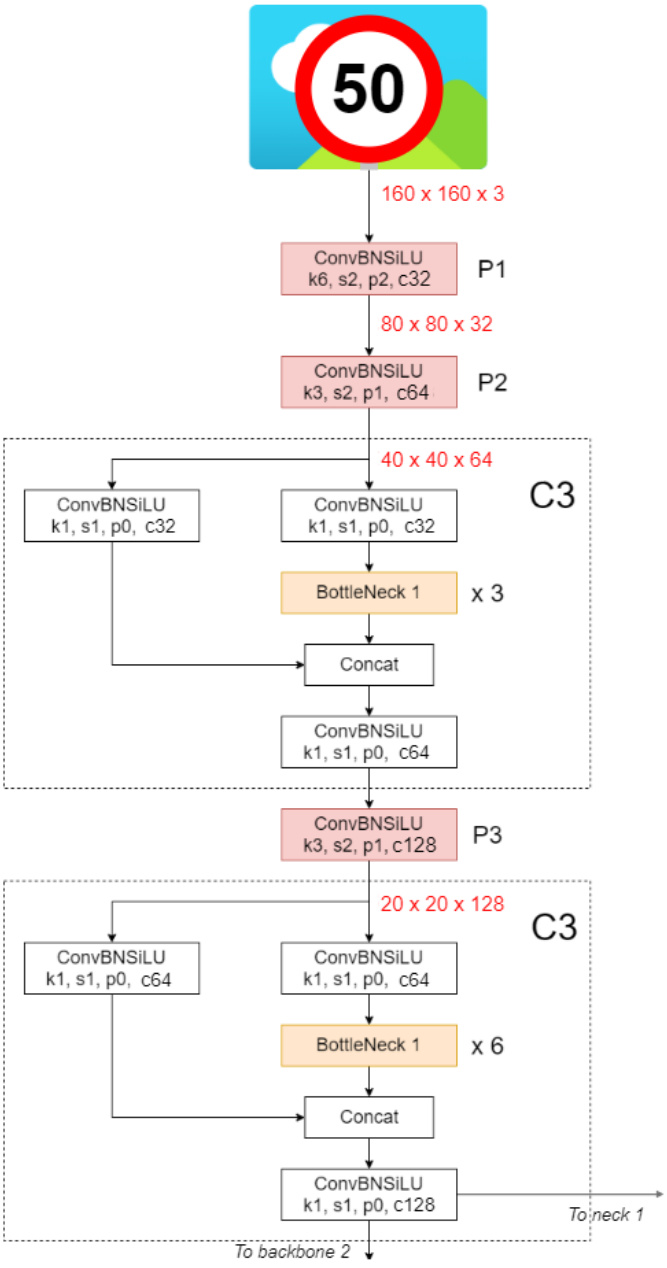


Figure 24: 3D CNN Backbone 1

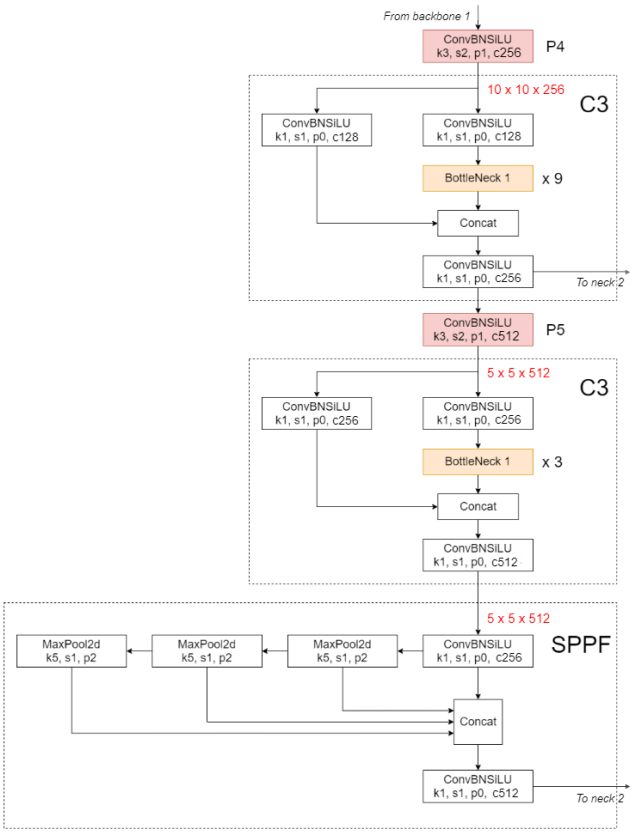


Figure 25: 3D CNN Backbone 2

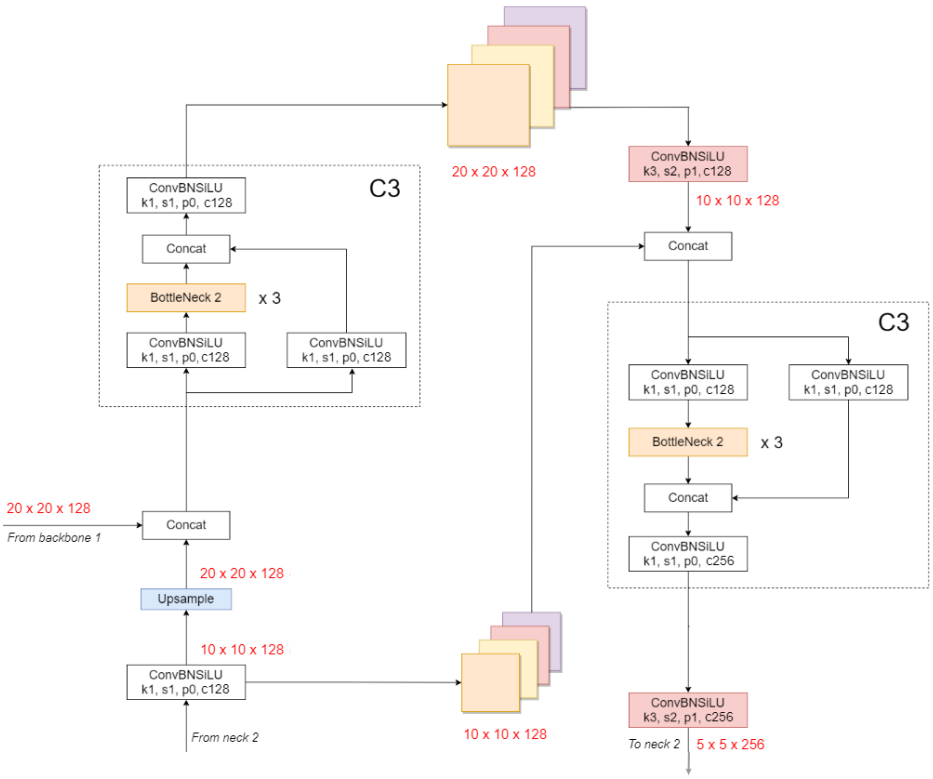


Figure 26: 3D CNN Neck 1

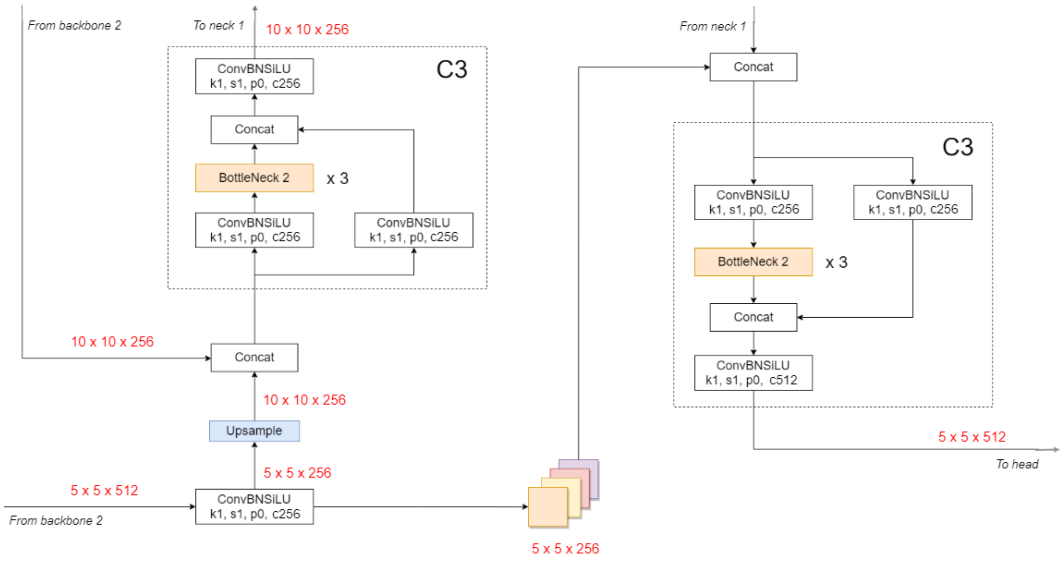


Figure 27: 3D CNN Neck 2

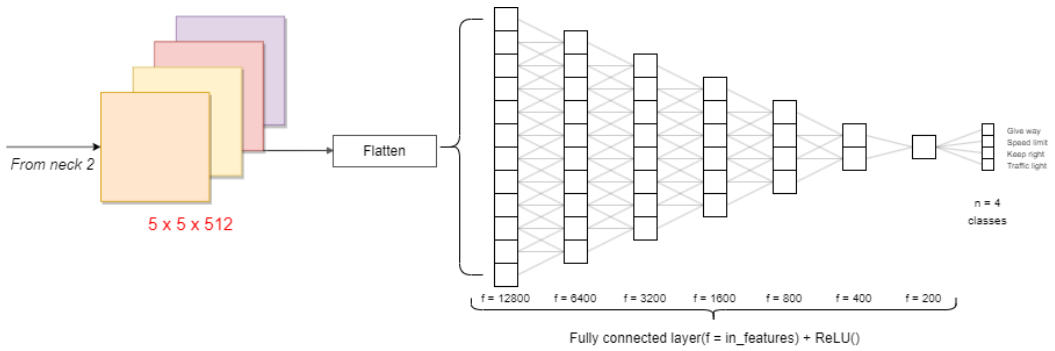


Figure 28: 3D CNN Head

A.3 4D CNN Architecture Summary and Visualization

For the 4D CNN, most of the changes are restricted to the backbone of the network. The input is multiple frames, stacked in the depth dimension. All convolutional layers are changed to Conv3d layers and all Batch Normalization layers are now BatchNorm3d layers. The window length is a variable in the form of input depth. This is denoted by w in figure 29. Based on w, the arguments of the convolutional layers are defined. This is specified in table 8. After the first three convolutions, the depth has been reduced to 1, so the remaining parts of the architecture (Backbone 2, Neck 1 & 2, Head) are the same as in figures 25, 26, 27, and 28. For those parts, the kernel depth is always 1, the stride in the depth dimension is always 1, and the padding in the depth dimension is always 0.

Table 8: Specification of Variables in the 4D CNN Architecture based on Window length w

w	d1	d2	k_1	k_2	k_3
5	3	1	3	3	1
7	5	3	3	3	3
9	5	3	5	3	3
11	7	3	5	5	3

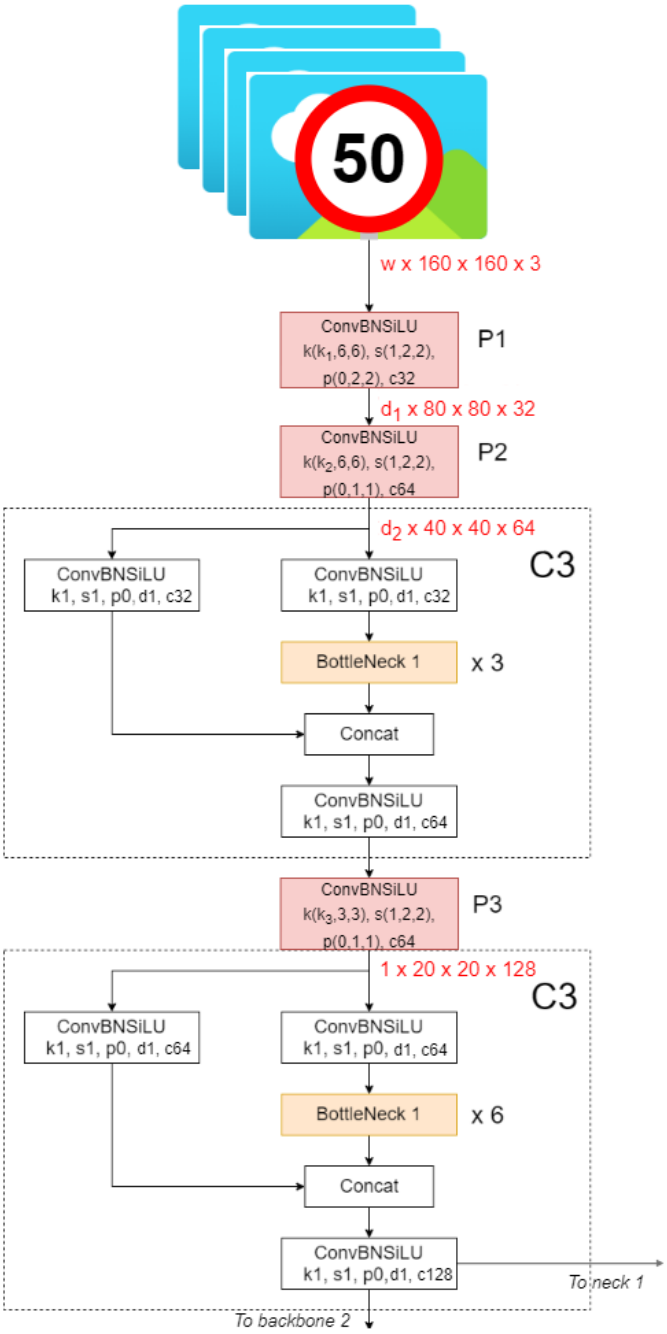


Figure 29: 4D CNN Backbone 1

B Data Augmentation Techniques

This appendix contains visualizations of the data augmentations used by YoloV5.

B.1 Mosaic

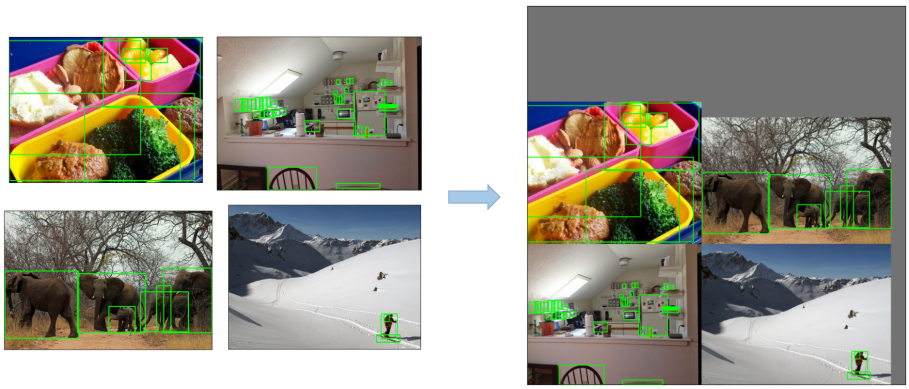


Figure 30: Mosaic in YoloV5
source: `https://github.com/ultralytics/yolov5/issues/6998`

B.2 Copy-paste

Figure 31: Copy-paste in YoloV5
source: `https://github.com/ultralytics/yolov5/issues/6998`

B.3 Random Affine

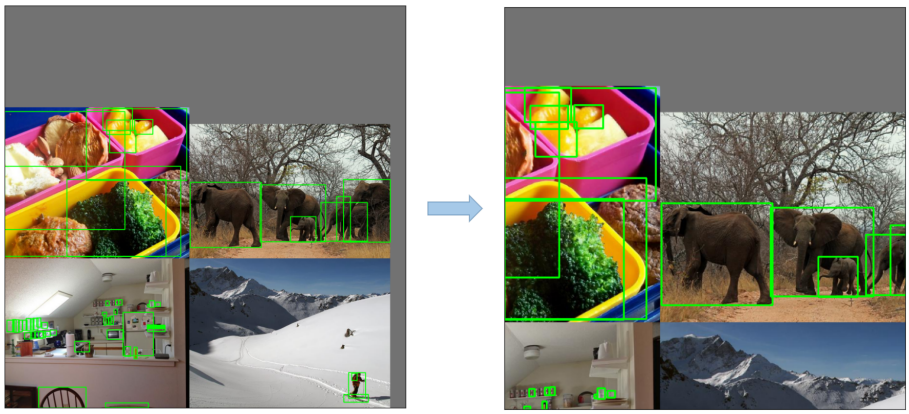


Figure 32: Random Affine in YoloV5
source: `https://github.com/ultralytics/yolov5/issues/6998`

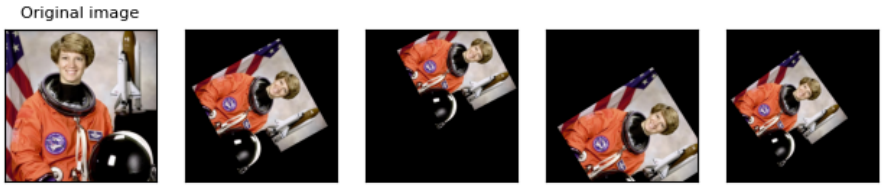


Figure 33: Random Affine with rotation, scale and translation (no shearing).
source: `https://pytorch.org/vision/stable/auto_examples/plot_transforms.html#randomaffine`

B.4 MixUp

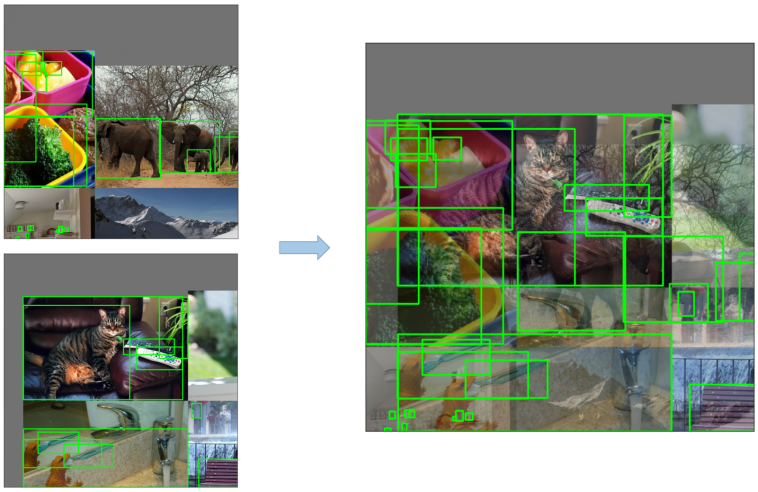


Figure 34: MixUp in YoloV5
source: `https://github.com/ultralytics/yolov5/issues/6998`

B.5 Albumentations

Figure 35: Albumentations
source: `https://github.com/albumentations-team/albumentations/`

B.6 Augment HSV

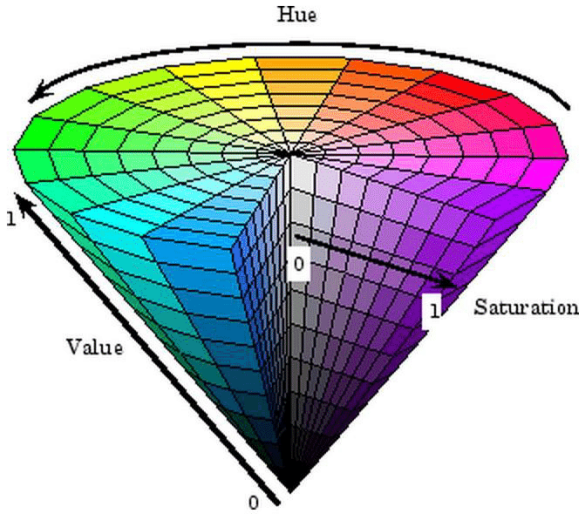


Figure 36: Hue, Saturation, and Value influence on color
source: `https://www.researchgate.net/figure/HSV-color-space-Hue-saturation-value_fig1_284698928`

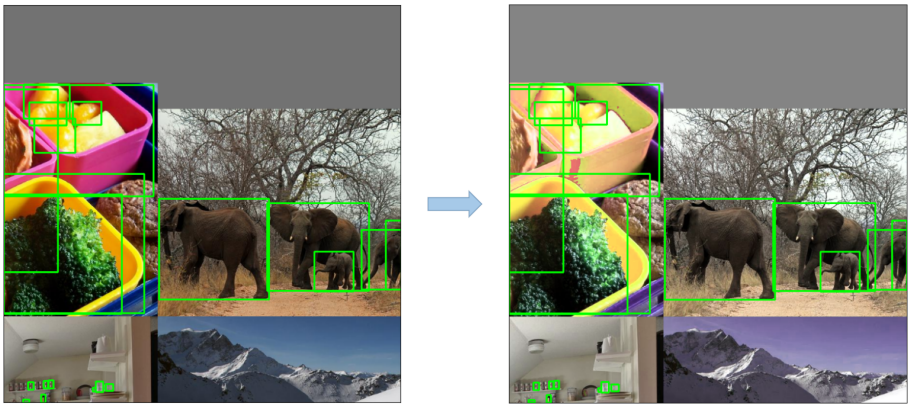


Figure 37: Augment HSV in YoloV5
source: https://github.com/ultralytics/yolov5/issues/6998

B.7 Random Horizontal Flip

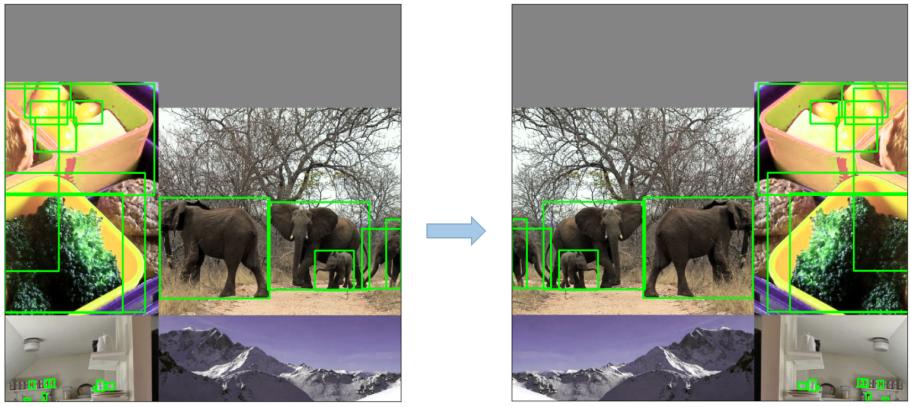


Figure 38: Random Horizontal Flip in YoloV5
source: https://github.com/ultralytics/yolov5/issues/6998

C Hyperparameter settings YoloV5s

- Initial learning rate (lr0): 0.01
- Final OneCycleLR learning rate (lrf): 0.01
- Momentum (momentum): 0.937
- Optimizer weight decay (weight_decay): 0.0005
- Numberof warmup epochs (warmup_epochs): 3.0
- Initial momentum during warmup (warmup_momentum): 0.8
- Initial bias learning rate during warmup (warmup_bias_lr): 0.1

- Box or Location loss gain (box): 0.05
- Class loss gain (cls): 0.5
- Class Binary Cross-Entropy loss positive weight (cls_pw): 1.0
- Objectness loss gain (obj): 1.0
- Objectness Binary Cross-Entropy loss positive weight (obj_pw): 1.0
- Intersection over Union of predicted & ground truth bounding boxes training threshold (iou_t): 0.2
- Anchor-multiple threshold (anchor_t): 4.0
- Focal loss gamma. If fl_gamma=0, it falls back to normal Cross-Entropy loss (fl_gamma): 0.0

Data augmentation hyperparameters:

- Image HSV-Hue augmentation fraction (hsv_h): 0.015
- Image HSV-Saturation augmentation fraction (hsv_s): 0.7
- Image HSV-Value augmentation fraction (hsv_v): 0.4
- Image rotation (degrees): 0.0
- Image translation fraction (translate): 0.1
- Image scale gain (scale): 0.5
- Image shear degradation (shear): 0.0
- Image perspective fraction (perspective): 0.0
- Image flip up-down probability (flipud): 0.0
- Image flip left-right probability (fliplr): 0.5
- Image mosaic probability (mosaic): 1.0
- Image mixup probability (mixup): 0.0
- Segment copy-paste probability (copy_paste): 0.0

D Hyperparameter Tuning 3D CNN

Table 9 contains the results of a series of training runs with the 3D CNN. The first six columns contain hyperparameters. Their column heads stand for:

- **ch**: amount of channels created from RGB input in the first convolution. This impacts the size of the network and the amount of parameters therein.
- **ep**: amount of epochs; training iterations.
- **lf**: loss function. CE is Categorical CrossEntropy, and NLL is Negative Log Likelihood.
- **opt**: optimizer. SGD is Stochastic Gradient Descent, and AdaG is AdaGrad.
- $\mathbf{lr_{low}}$: low learning rate. Used during warmup and from half of the total epochs onward.
- $\mathbf{lr_{high}}$: high learning rate. Used after warmup until half of the total epochs.

Columns seven and eight (acc & loss) contain the performance metrics Accuracy and Loss, respectively. Accuracy is in percentage. Note that the loss can only be compared for runs with the same loss function. The four columns on the right hand side contain prediction rates for the four classes in the data. This can be used to determine whether the model converges heavily towards one class.

Table 9: Training runs with the initial 3D CNN setup, used for hyperparameter tuning

Hyperparameters						Metrics		Predictions per class			
ch	**ep**	**lf**	**opt**	$\mathbf{lr_{low}}$	$\mathbf{lr_{high}}$	**acc**	**loss**	**p0**	**p1**	**p2**	**p3**
32	30	CE	SGD	1e-2	1e-2	45.5	1.386294	6	146	26	11
32	30	CE	SGD	1e-3	1e-3	30.2	1.386294	116	33	23	17
32	30	NLL	SGD	1e-2	1e-2	48.1	4.142461	18	120	20	31
32	30	NLL	SGD	1e-3	1e-2	35.4	4.142450	3	8	148	35
32	100	NLL	SGD	1e-3	1e-2	57.1	4.141755	7	139	0	43
64	90	NLL	SGD	1e-3	1e-2	24.3	4.137717				
16	100	NLL	SGD	1e-3	1e-2	24.3	4.141876	4	14	128	43
32	100	NLL	AdaG	1e-3	1e-2	68.3	4.255235	47	59	27	56
32	100	NLL	AdaG	1e-4	1e-3	88.4	3.472060	32	94	19	44
16	100	NLL	AdaG	1e-4	1e-3	74.6	3.703147	47	84	23	35
16	100	NLL	AdaG	1e-4	1e-3	72.5	4.041786	42	85	25	37

In row six it can be noticed that the prediction rates are missing. This is due to a memory overflow during epoch 90, which caused the run to stop before reaching the

set amount of epochs. Prediction rates were only saved at the end of a run, hence the missing values. The memory overflow was caused by running the model with a channel depth of 64 after the first convolution. This resulted in a much larger amount of weights in the network, eventually leading to a memory overflow.

E Hyperparameter Tuning 4D CNN

Run	$\mathbf{F_{1,macro}}$	**Accuracy**	**Length**	**Stride**	**ROI**
1	0.685	66.667	5	2	1.2
2	0.680	68.939	5	2	1.5
3	0.787	78.788	5	2	1.8
4	0.773	77.273	5	2	2.1
5	0.628	64.394	7	2	1.2
6	0.833	83.333	7	2	1.5
7	0.792	78.788	7	2	1.8
8	0.702	71.212	7	2	2.1
9	0.670	68.182	9	2	1.2
10	0.621	61.364	9	2	1.5
11	0.703	70.455	9	2	1.8
12	0.666	66.667	9	2	2.1
13	0.726	73.485	11	2	1.2
14	0.644	65.909	11	2	1.5
15	0.730	73.485	11	2	1.8
16	0.693	69.697	11	2	2.1
17	0.811	81.061	5	5	1.2
18	0.668	68.182	5	5	1.5
19	0.594	59.848	5	5	1.8
20	0.718	71.212	5	5	2.1
21	0.654	65.152	7	5	1.2
22	0.758	75.758	7	5	1.5
23	0.686	68.182	7	5	1.8
24	0.673	67.424	7	5	2.1
25	0.747	75.758	9	5	1.2
26	0.766	77.273	9	5	1.5
27	0.645	63.636	9	5	1.8
28	0.740	75.000	9	5	2.1

Run	$F_{1,macro}$	Accuracy	Length	Stride	ROI
29	0.652	66.667	11	5	1.2
30	0.691	68.182	11	5	1.5
31	0.648	65.152	11	5	1.8
32	0.667	68.182	11	5	2.1
33	0.721	71.212	5	10	1.2
34	0.658	65.909	5	10	1.5
35	0.652	64.394	5	10	1.8
36	0.608	60.606	5	10	2.1
37	0.675	68.182	7	10	1.2
38	0.720	71.212	7	10	1.5
39	0.667	67.424	7	10	1.8
40	0.655	65.909	7	10	2.1
41	0.616	62.879	9	10	1.2
42	0.622	65.909	9	10	1.5
43	0.570	59.091	9	10	1.8
44	0.637	65.152	9	10	2.1
45	0.610	60.606	11	10	1.2
46	0.736	72.727	11	10	1.5
47	0.564	58.333	11	10	1.8
48	0.595	59.091	11	10	2.1
49	0.719	71.970	5	20	1.2
50	0.652	65.152	5	20	1.5
51	0.625	65.152	5	20	1.8
52	0.614	62.121	5	20	2.1
53	0.660	65.909	7	20	1.2
54	0.683	69.697	7	20	1.5
55	0.636	64.394	7	20	1.8
56	0.656	66.667	7	20	2.1
57	0.574	59.848	9	20	1.2
58	0.613	62.879	9	20	1.5
59	0.601	61.364	9	20	1.8
60	0.679	69.697	9	20	2.1
61	0.614	61.364	11	20	1.2
62	0.623	64.394	11	20	1.5
63	0.667	71.212	11	20	1.8

Run	$F_{1,macro}$	Accuracy	Length	Stride	ROI
64	0.614	63.636	11	20	2.1
65	0.587	59.091	5	30	1.2
66	0.656	65.152	5	30	1.5
67	0.601	59.091	5	30	1.8
68	0.580	57.576	5	30	2.1
69	0.640	62.879	7	30	1.2
70	0.526	53.788	7	30	1.5
71	0.551	56.061	7	30	1.8
72	0.679	67.424	7	30	2.1
73	0.549	53.788	9	30	1.2
74	0.625	59.848	9	30	1.5
75	0.565	56.436	9	30	1.8
76	0.589	60.000	9	30	2.1
77	0.593	59.000	11	30	1.2
78	0.523	52.000	11	30	1.5
79	0.600	60.000	11	30	1.8
80	0.672	67.000	11	30	2.1