Rheinische
Friedrich-Wilhelms-
Universität Bonn

Institute for Computer Science
Department VI
Autonomous Intelligent Systems

Rheinische Friedrich-Wilhelms-Universität Bonn

Master Thesis

Recursive Transformer based Video Semantic Segmenation

Author:
Aysha Athar Siddiqui

First Examiner:
Prof. Dr. Sven Behnke

Second Examiner:
PD. Dr. Volker Steinhage

Supervisor:
M.Sc. Angel Villar-Corrales

Date: September 9, 2023

Declaration

I hereby declare that I am the sole author of this thesis and that none other than the specified sources and aids have been used. Passages and figures quoted from other works have been marked with appropriate mention of the source.

09.09.2023

Place, Date

Signature

Abstract

Video Semantic Segmentation is the challenging task of predicting a class value for each pixel in every frame of a video sequence. Existing image segmentation models could be easily extended to obtain the semantic segmentation of a video sequence by applying them frame-by-frame. However, this naive approach fails to model the temporal continuity between frames, thus leading to noisy video segmentations that suffer from multiple artifacts, such as flickering or ghosting. In this thesis, inspired by existing work on image segmentation, we aim to extend the recent transformer-based Segmenter model to perform Video Semantic Segmentation in a recursive manner. Segmenter consists of a Transformer-based ViT encoder, as well as a Fully-Connected or Transformer-based decoder, which maps the encoded features into semantic classes. To extend this architecture for Video Semantic Segmentation, we propose VideoSegmenter that introduces a Predictor and Corrector module that models spatio-temporal relations between consecutive frames. We conduct various experiments and evaluate our model on the Cityscapes and Synpick datasets to show the effectiveness of modeling temporal continuity by adding the two proposed modules to the model architecture.

Contents

1 Introduction

Understanding and interpreting scenes in a video sequence is a crucial task in fields like autonomous driving [30, 43], robotics [31, 25] and augmented reality [35, 36] that is solved by means of Video Semantic Segmentation. Video Semantic Segmentation refers to the task of assigning each pixel in a frame extracted from a video sequence to a semantic class value. This is a particularly difficult task to tackle given the large amounts of raw data available and the lack of annotated samples for the data as is the case for popular datasets like Cityscapes [10] and CamVid [4] wherein labeled semantic maps are provided only for one out of every 30 frames. The naive way to solve a Video Semantic Segmentation task is to simply use an existing image segmentation model and apply it frame-by-frame to each sequence in the video. However, the issue arises wherein there are lack of annotated frames available for each frame which leads to noisy and flickered predictions. Moreover, using these kind of models also ignores the temporal continuity that is present in one video sequences as each segmentation map is computed independently.

In order to tackle this issue, researchers have come up with various ways to account for the temporal features existing in a video sequence. These could take the form of adding another layer such as Optical Flow [42, 12] on top of an existing image segmentation models. Another method involves integrating memory based mechanisms like Recurrent Neural Networks [38], ConvLSTMS [24, 14] or Transformers [41, 20] that learn from information extracted from past frames. This acts like a memory and helps in learning temporal information that in turn results in stable predictions and handles irregularities such as occlusions, ghosting and flickering.

The recently proposed Segmenter [34] model makes use of the Vision Transformer as the encoder (ViT encoder) [13] that was introduced to handle image data with transformers along with a linear/mask decoder to get segmentations from images. The goal of this thesis is to extend this model to also work well for video sequences. To this end, we propose the VideoSegmenter model which makes use of the ViT encoder and additionally introduce two modules namely, the Predictor and Corrector to model temporal continuity. The Predictor takes into account past frames in a sequence in order to predict the embeddings of the current frame. The corrector then fuses these predicted embeddings with the em-

beddings from the ViT encoder in order to generate embeddings that are rich with spatio-temporal information from the sequence. Finally, the decoder takes these embeddings and produces segmentation maps for the corresponding video frames provided.

Our experiments show that VideoSegmenter outperforms the results obtained by applying the Segmenter [34] model frame-by-frame. We also show that the results obtained are consitent across the time frames and result in less noisy/flickered predictions as compared to the frame-by-frame model. The main contributions of this thesis are:

1. We propose the VideoSegmenter model, an extension to the Segmenter model [34] in order to segment video sequences.

2. We show that the model performs well in its abilities to capture temporal relations between video sequences.

3. We present the qualitative and quantitative results based on our experiments conducted.

The thesis is structured as follows:

- *Chapter 2: Theoretical Background*: This chapter talks about the fundamental building blocks needed and discusses the concepts used throughout the course of the thesis.

- *Chapter3: Related work*: This chapter discusses the various approaches that exist in the Video Semantic Segmentation domain.

- *Chapter 4: Proposed Method*: Model architecture for VideoSegmenter along with the training strategy is explained in detail.

- *Chapter 5: Experiments and Results* This section gives the qualitative and quantitative results of the experiments conducted during the course of thesis. It provides insights into how well the model performed.

- *Chapter 6: Conclusion and Future Work*

2 Theoretical background

This chapter summarizes the fundamental concepts required throughout the course of the thesis. It starts with a simple explanation to the Multi Layer Perceptron and goes on to explain more advanced models employed throughout this thesis.

2.1 Multi-Layer Perceptron

One of the early developments in the neural network field was the formation of the Multi-Layer Perceptron (MLP) architecture [15]. It consists of a simple feed forward neural network which comprises of an input layer, one/many hidden layers in between and an output layer. To be more specific, each neuron in a specific layer is connected to every neuron in the subsequent layer. Each of these neurons have a weight attached to it which decides the contribution of the specific neuron to all other neurons connected to it in the subsequent layer. The equation below elaborates this:

$$Y = WX + b \tag{2.1}$$

wherein W denotes the weights of each neuron in a layer, X denotes the input and b is the bias term.

Now, in order for this MLP to be able to learn complex data problems, an activation function is introduced to bring in non-linearity, else the MLP acts like a linear regression problem. The weighted sum Y is passed through an activation such as the Sigmoid [27] or ReLU [1] which then results in the output $\hat{Y}$:

$$\hat{Y} = \phi(Y) \tag{2.2}$$

wherein ϕ represents the activation function.

Since MLPs fall in the category of feed forward networks, wherein the output of each layer is propagated to each subsequent layers, they lack the capabilities of handling sequential data. To overcome this, Recurrent Neural Networks (RNNs) were introduced which are briefly described below.

2.2 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) [15] introduced the concept of temporal sharing of parameters. At each time step, instead of computing the weighted sum between the input layer and its weights directly as in the case of an MLP, the RNN combines the input x_t with the previous hidden state h_{t-1} to form the input, which is then propagated forward through the activation function to result in h_t which forms the hidden state of time step t. This hidden state is used in combination with the input vector at the subsequent time steps. This way the model retains memory of the past sequences through the hidden states h_t and proves to be of importance in problem statements related to the processing of sequential data.

However, RNNs fail to capture long-term dependencies along large sequences of data as it suffers from the problem of vanishing gradients due to the way in which the gradients are backpropogated through time.

In order to mitigate this issue, Long-Short-Term Memory networks were introduced in the field which have the capability of modeling long-range dependencies well.

2.3 Long-Short-Term Memory Networks

Long-Short-Term Memory Networks (LSTM) [17] are a variation of RNNs that introduced memory cells and gating mechanisms which decide on what information to remember and forget. This leads to the network having a better capacity to retain memory of longer sequences of data.

In order to summarize it briefly, there are three gating mechanisms in the LSTM as depicted in figure 2.1 that determine how the inputs flow through the model, the input gate which decides how much information from the current input $x^{(t)}$ should be added to the memory cell state, the forget gate that decides on the information that needs to be forgotten from the previous cell state, and output gate that gives the results $y^{(t)}$ based on the cell state. The Sigmoid [27] activation function is used for the input, forget and output gates while the hyperbolic tangent is used to generate the cell state and final outputs.

While LSTMs prove to be a strong architecture for modeling data sequentially, they still have their limitations like slower training times and limited parallelism properties.

In order to solve some of these issues, Transformers were introduced.

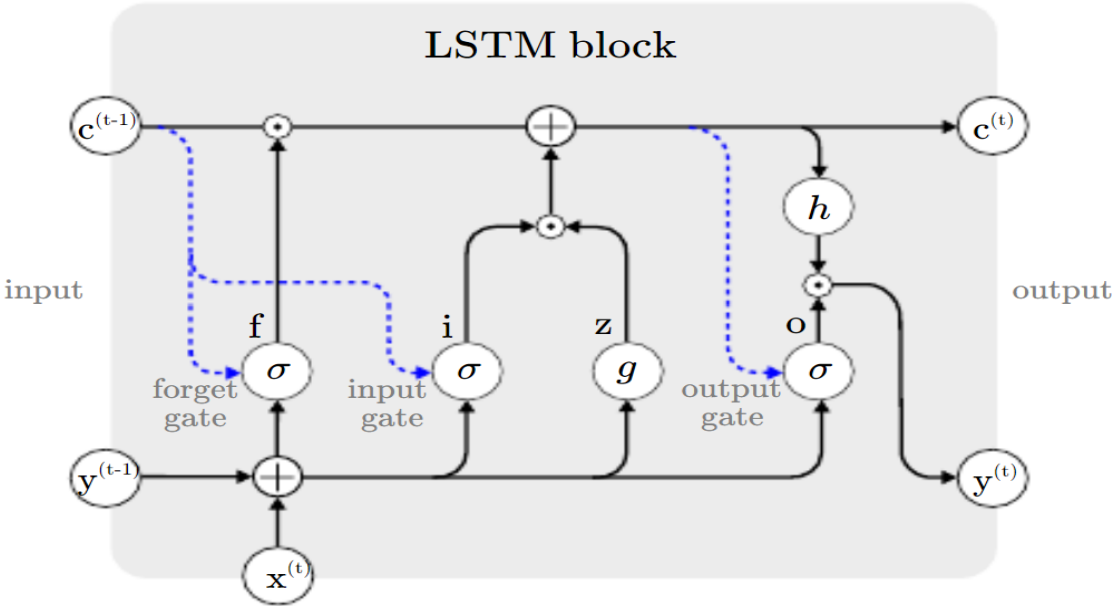


Figure 2.1: Long-Short-Term Memory network Architecture, figure from [39]. The three gates, input, forget and output gate as seen in the figure are the backbone of the LSTM structure and maintain the flow of information within a cell.

2.4 Transformers

Transformers [40] have gained widespread attention in the sequence-to-sequence domain particularly due to their capabilities to learn long range dependencies in the input data. The architecture uses self attention mechanisms which proves to be of importance as it has the abilities to parallelise input sequences which helps in dealing with longer sequences and capturing long range dependencies between the sequences. Due to their capabilities they have been widely used in language modeling tasks [11, 26] to perform various tasks like question answering systems and sentiment analysis.

The transformer architecture [40] consists of a transformer encoder-decoder structure consisting of multiple layers. Each layer consists of two sublayers, the multi-head self-attention layer and a feed forward layer. Residual connections [16] are added in between the multi-head self-attention and feed forward layers whereas layernorm [2] is applied after the two layers as shown in figure 2.3. The multi-head self-attention mechanism allows the modeling of long range dependencies and simultaneously attends to information from multiple representations subspaces and different positions.

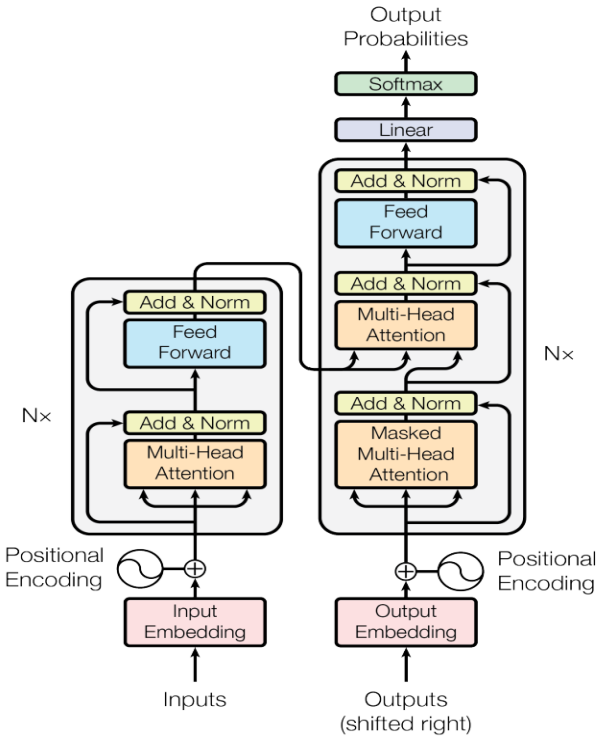


Figure 2.2: Transformer Model Architecture, figure from [40]

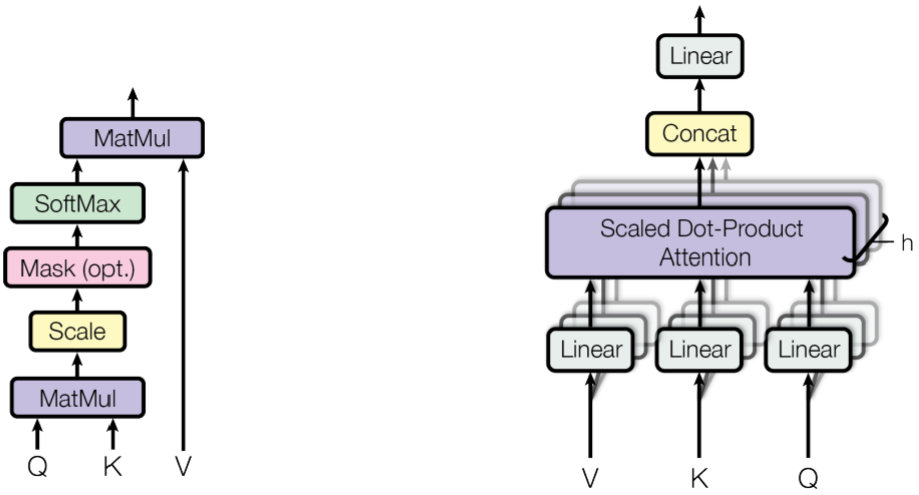
Self Attention

Self attention is the core operation used in transformers. An input sequence X is transformed to Query (Q), Key (K) and Value (V) vectors each of dimension d_k by using learned linear transformations. Self Attention captures how important each token is with respect to all other tokens in a sequence. These are calculated by means of attention scores that are determined by taking the dot product between the query vector of token i with key vector of token j. This dot product is then scaled by a dimension d_k. A softmax is applied to get the attention weights as follows:

$$S = \text{softmax}(\frac{QK^T}{\sqrt{d_k}}) \tag{2.3}$$

Finally, the attention weights S are multiplied by V to generate the output of the attention module:

$$\begin{aligned} \text{Attention}(Q, K, V) &= SV \\ \text{Attention}(Q, K, V) &= \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \end{aligned} \tag{2.4}$$



(a) Self attention

(b) Multi-head attention

Figure 2.3: Attention mechanisms in the transformer model, figures taken from [40]

Multi head self attention

This is an extension to the self-attention mechanism that allows the model to capture different types of patterns that exist in the data as it contains multiple heads as shown in figure 2.3b. Specifically, the input embedding is split into multiple heads wherein each of them is transformed to Query, Key and Value matrices. In the end, the output is obtained by concatenating the outputs of each head and linearly projecting them to result in the final output.

Positional encoding

Transformers process the input sequence at once in parallel and do not have information of the sequential order of elements like RNNs. Tasks like Natural Language Processing where the order of the word matters or Video processing fields where the position of the frame in the sequence matters, positional encoding is needed. These are added to input of the transformer encoder so as to make use of the order of the sequence, which can be done via sinusoidal positional embedding or learnable embeddings. The sinusoidal positional encoding method models embeddings using sine and cosine functions. Learnable encodings on the other hand learn throughout the training process and are optimised via backpropogation. [40] claims that both types of embeddings produce similar results. In this thesis, we make use of learnable positional encodings.

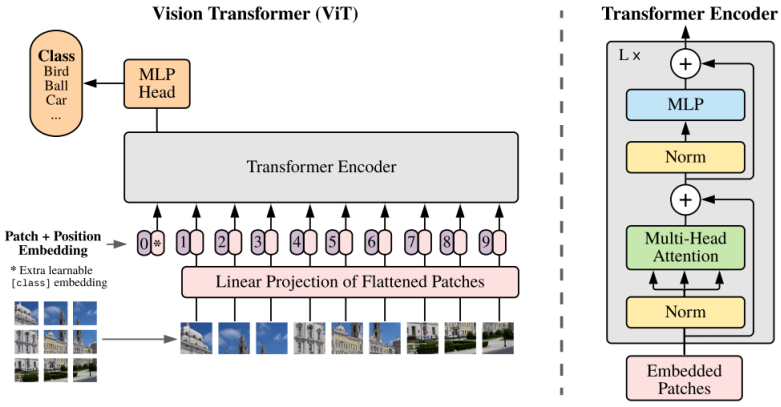


Figure 2.4: Vision transformer Model, figure extracted from [13]. The images are divided into patches of fixed size and are then processed by the Transformer Encoder whose architecture is depicted on the right.

2.5 Vision Transformer

Since the above transformer model has gained widespread attention particularly in the Natural Language Processing (NLP) field, computer vision researchers have come up with a modification to the original transformer [40] in order to apply it to images introduced by Dosovitskiy et al. [13] They propose the Vision Transformer (ViT) model which splits an image $x \in R^{H \times W \times C}$, where H, W and C represent the height, width and channels of the image respectively, into patches of a fixed size (P, P) resulting in $N = HW/P^2$ patches. These patches are flattened into a sequence of patches namely $x_p \in R^{N \times (P^2 C)}$. Since the transformer uses a fixed latent vector of size d, these patches are flattened using a trainable linear projection to map them to a d-dimensional embedding. Similar to the vanilla transformer [40], positional encodings are added to the patch embeddings to augment them with information of their position in an image which are then processed by the encoder which consists of L layers with Multi-head attention, LayerNorm, MLP and residual connections as shown in figure 2.4. The ViT architecture [13] makes use of learnable 1D positional embeddings.

3 Related work

This chapter will elaborate on the ongoing research and success found in the Video Semantic Segmentation fields. We divide them into three categories namely, Optical Flow methods, Conditional Random Field models and Memory-based models.

3.1 Optical Flow Methods

Optical Flow is widely used in the computer vision field in tasks relating to visual data. It aims at estimating how pixels in a frame from a video sequence will change/transform in the next consecutive frame. In simpler words, it gives us an estimate abut the flow of objects during the course of a video sequence. This helps in enhancing the segmentations as it also accounts for flow information. It can particularly be made useful in scenarios where the dataset is dynamic and the scenes are constantly changing. Ding et al. [12] make use of optical flow estimation together with semantic information gathered via a shared encoder across the input data. They claim that using a shared encoder is more beneficial when used with flow information exchange as it increases the capabilities of the representations gathered as opposed to the paper by Wang et al. [42]. Further, a smoothness loss function is applied to the flow prediction results in order to achieve better flow quality. Ding et al. only makes use of the flow guided module during training phase to reduce the time taken at inference.

Nilsson et al. [21] combines the use of optical flow together with a Gated Recurrent Unit. They warp the semantic segmentation map along the optical flow. This warped embedding is fed to the gated recurrent units (GRU) hidden state. The input to the GRU is an estimate of the segmentation map that they compute using a single frame Convolutional NeuraL Network (CNN). Combining these two predicted segmentation maps is proved to estimate an accurate semantic segmentation wherein frames in which significant motion occurs are segmented well.

A drawback of using Optical Flow methods for the task of Video Segmentation is that they work together with the Segmentation model and often prove to be computationally expensive.

3.2 Conditional Random Fields Models

Conditional Random Fields (CRF) approach introduced by [6, 7] are utilized by Chandra et al. [5] to be compatible with video sequences. They introduce temporal connections to account for video sequences in addition to spatial connections. Spatial connections in a CRF based model are captured via pairwise potentials which penalize inconsistent/dissimilar labellings between neighboring pixels. This helps the model to assign similar labellings between neighboring frames of a video sequence thereby including spatial relationships present in an image. In order to also account for temporal data, pairwise potentials between patches in different frames are taken into consideration. These pairwise potentials are generated via a CNN network.

Although these models prove to be effective in solving the task of Video Semantic Segmentation, they are computationally expensive since calculating pairwise potentials for video data requires large training and inference times that makes them unsuitable for real time prediction of semantic segmentation maps.

3.3 Memory-Based Models

In order to model temporal continuity between frames, memory based mechanisms are used in order to retain information from the previous frames in a sequence to account for temporal consistency between frames in a video sequence. These can be done by leveraging memory-based architectures like Recurrent Neural Networks (RNNs) [38], Convolutional Long-Short-Term memory networks (ConvLSTMs) [28], Gated Recurrent Unites (GRUs) [8], Attention based networks and transformers [40] that model sequential relations well.

LSTM-SegNet

Since Recurrent Neural Networks (RNNs) are used to model sequence data, it is worthwhile to mention LSTM-SegNet [24]. that solves the task of video semantic segmentation by incorporating ConvLSTM [28] layers. They extend the classical encoder-decoder style SegNet architecture [3] wherein the encoder is a VGG16 [32] network and the decoder is a reversed version of the encoder. Pfeuffer et al. go on to insert ConvLSTM layers in three positions as show in figure 3.1: in between the encoder and decoder, after the decoder before the softmax layer and finally a combination of both(in between and after the softmax layer), in order to determine which combination works well. They claim that adding a ConvLSTM layer after the decoder just before the softmax layer might prove to be useful as

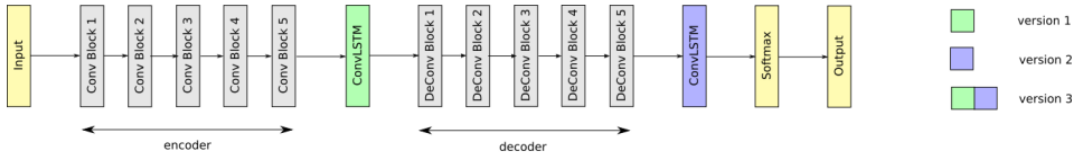


Figure 3.1: LSTM-SegNet architecture from [24]. The three positions to insert a convLSTM layer are highlighted here and color coded schemes are represented on the right.

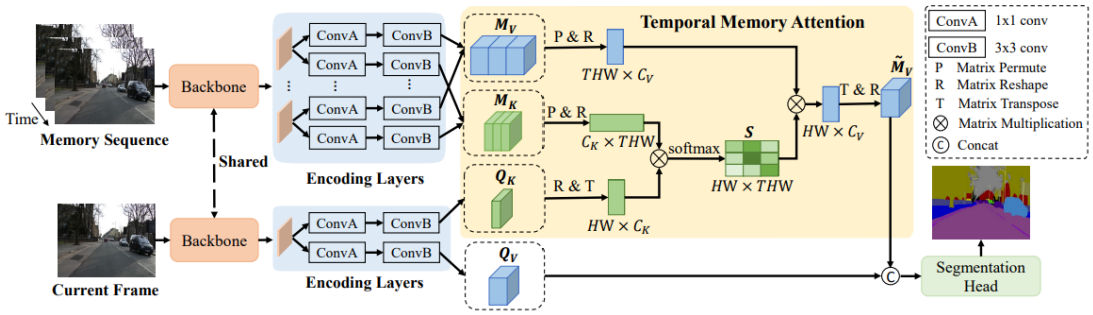


Figure 3.2: TMANet Model Architecture. The current frame along with the memory sequence are fed to the shared backbone which are then passed to the encoding layers to embed the features extracted. They are processed by the TMA block that models temporal relations.

each sequence is then processed independently and the backpropogation of error is prevented. Finally, a recurrent method combines the results from the previous frames and the current frame to obtain the results. Their results show that adding a convLSTM layer just before the softmax layer provides best results.

Recurrent Fully Convolutional Networks for Video Segmentation (RFCN)

Valipour et al. [37] leverage the properties of Recurrent Neural Networks (RNNs) for the task of video segmentation. They build upon the concept of Fully Convolutional Networks (FCNs) that model spatial relations well by making use of convolutional layers that enable the model to understand the context of each image in a video sequence. To account for temporal relations between consecutive frames, they incorporate recurrent connections to the FCN structure so that the model makes use of previously seem frames to account for the segmentation map of the current frame. The recurrent unit takes the form either an LSTM, GRU or Conv-GRU.

Temporal Memory Attention Network (TMANet)

In order to include temporal information, Wang et al. [41] use a memory network to predict a frame in a sequence where the memory network represents frames from the previous time steps. In particular, it feeds the current frame along with the memory frames to a shared ResNet [16] backbone to extract features. In order to encode the features extracted, these are processed onto the encoder to extract memory features and the current frame features separately. The vectors obtained are further fed to the temporal memory attention module (TMA) as illustrated in figure 3.2 to build temporal relations. To calculate the temporal memory attention, memory vectors and current frame features are combined. Finally, a segmentation head gives the desired results.

4 Proposed Method

This chapter first describes in detail the Segmenter [34] model in section 4.1 since our VideoSegmenter model heavily relies on it. Section 4.2 defines the flow of the proposed VideoSegmenter model and its components (ViT encoder, Predictor, Corrector and Decoder) in the sub sections. Further, Section 4.3 mentions the loss functions used during training of the model and finally, Section 4.4 mentions the hyper-parameters used and training strategy implemented.

4.1 Segmenter

Directly inspired by [34], the work on this thesis aims at extending the Segmenter model to also work well for video sequences. The Segmenter model, as depicted in figure 4.1, is based on the transformer architecture inspired by the ViT [13]. It takes the input image $x \in R^{H \times W \times C}$ and splits it into N patches of fixed size. These patches are flattened and linearly projected to d-dimensional patch embeddings. In order to account for positional information, learnable positional encodings are added which result in a sequence of tokens $Z_o = [Z_{o,1}, ..., Z_{o,N}]$ which is processed by the transformer encoder that contains multi-head self-attention blocks followed by MLP blocks with residual connections and layer normalization layers resulting in embedding from the encoder as $Z_L = [Z_{L,1}, Z_{L,2}, ..., Z_{L,N}]$. These encodings are then processed by means of a linear decoder or a mask transformer in order to predict the semantic segmentation maps for the corresponding input frame.

The linear decoder maps the patch embeddings from the encoder to patch-level class logits by applying a point-wise linear layer. These are then reshaped and bilinearly upsampled to the original image size to which a softmax layer is applied to get the segmentation maps.

The Mask transformer decoder on the other hand makes use of K learnable class embeddings cls, wherein K is the number of classes, which are utilised to generate class masks. The embeddings from the encoder Z_L are processed together with the class embeddings cls via the decoder which is a transformer encoder consisting of M layers. The output from the transformer decoder results in L2-normalised patch embeddings $Z'_M \in R^{N \times D}$ along with class embeddings c^T. The scalar product of

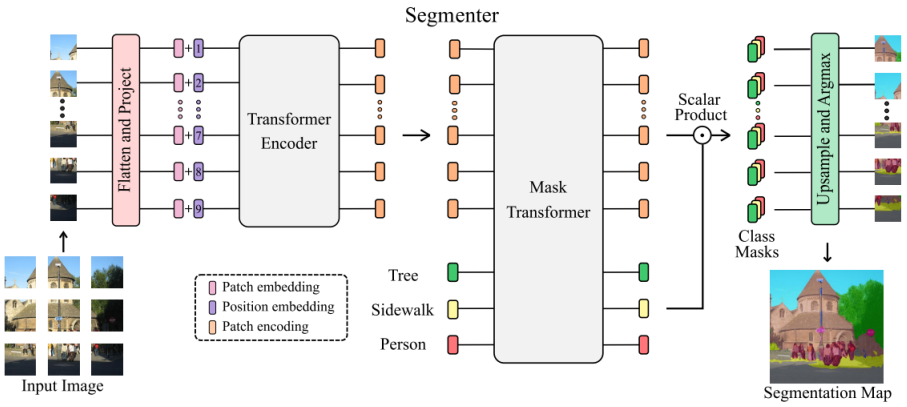


Figure 4.1: The architecture of the Segmenter Model taken from [34]

these two values is taken to produce a patch sequence which is then reshaped and bilinearly upsampled. Finally, as with the linear decoder, a softmax layer is applied to generate the segmentation maps.

4.2 VideoSegmenter

The Segmenter [34] model introduced above can be used as a Video Semantic Segmentation model when applied frame-by-frame. However, this leads to predicted segmentation maps wherein the model fails to learn temporal relations as it treats each frame separately and uses no information from the past frames in a sequence to predict the next segmentation maps. To mitigate this problem, we introduce the VideoSegmenter model which introduces a method to learn the temporal relations contained between frames in a sequence by introducing two modules, the Predictor and the Corrector.

Given a sequence of frames $x_1, x_2, x_3, ...x_N$ extracted from a video sequence, we aim to predict the segmentation maps $s_1, s_2, s_3, ...s_N$ for each frame in the video sequence. The VideoSegmenter model consists of four main blocks, the ViT encoder (section 4.2.1), predictor (section 4.2.2), corrector 4.2.3 and decoder 4.2.4 explained in detail in the mentioned sub-sections. Similar to the Segmenter model, each frame extracted from a video sequence is broken down into patches of a fixed size. These patches are then flattened and linearly projected. Learnable positional encoding is added to each patch to retain spatial information. Patch embeddings along with the positional encodings from each frame in the sequence is fed to the ViT encoder which is transformer consisting of L layers. In order to incorporate temporal relations between frames, we make use of the Predictor module. This transformer module predicts the patch embeddings of the current

time step by considering information from the previous time steps. Essentially, if we have a sequence of four frames, in order to predict the segmentation map of the fourth frame, ViT embeddings of the previous three frames in the sequence are fed to the predictor along with a learnable temporal encoding that is added to it to retain temporal relations. The prediction from the predictor is fused with the output of the ViT encoder (ViT embeddings) via the corrector transformer block which models both the embeddings resulting in embeddings that are rich in spatial as well as temporal features. The embeddings generated by the corrector module are passed through the decoder, which is either a linear decoder or a mask transformer decoder, in order to produce patch-level logits that are then reshaped and upsampled to the original image size. Lastly, a softmax layer is applied to get the segmentation maps of the sequence. The model architecture of Segmenter is depicted in figure 4.1

4.2.1 ViT Encoder

Each frame $x \in R^{H \times W \times C}$ extracted from a video sequence where H, W and C represent the height, width and channels respectively, is divided into N patches each of size (P, P) generating a sequence of patches $x_p \in R^{N \times P^2 \times C}$. Each patch of size (P, P) is flattened to a 1D vector and linearly projected with a learnable trasnform to get patch embeddings $x_o \in R^{N \times D}$. In order to retain positional information, learnable positional encodings $p \in R^{N \times D}$ are added to the sequence of patch embeddings to generate the input sequence to the ViT encoder as in equation 4.1

$$y_o = x_o + p \in R^{N \times D} \tag{4.1}$$

The encoder consists of L transformer encoder layers and generates a sequence of encodings $y_L \in R^{N \times D}$. Each layer consists of a multi-headed self attention block (MSA) followed by a simple MLP block. A residual connection is added after every block and a layernorm layer is applied before every block as shown in equations 4.2 and 4.3

$$y'_l = \text{MSA } (\text{LN } (y_{l-1})) + y_{l-1} \tag{4.2}$$

$$y_l = \text{MLP } (\text{LN } (y'_l)) + y'_l \tag{4.3}$$

Finally a layernorm is applied in the end to generate the final output from the transformer encoder as in equation 4.4

$$y_L = \text{LN}(y_l) \tag{4.4}$$

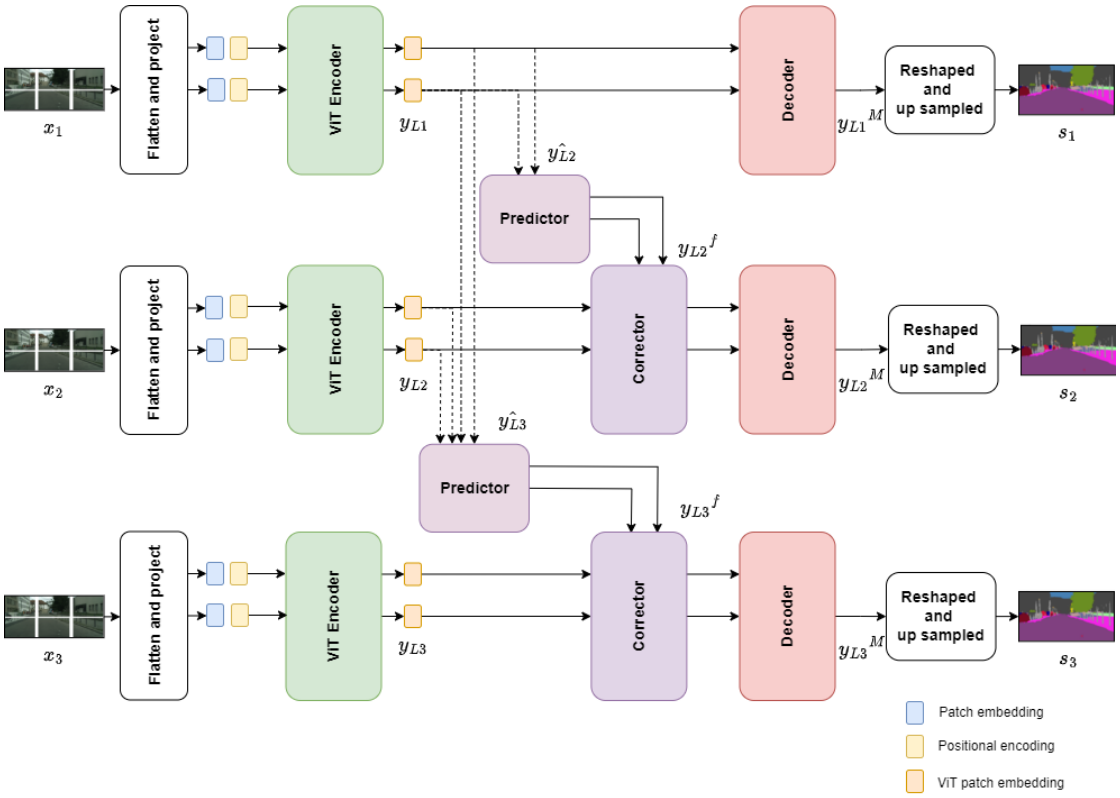


Figure 4.2: VideoSegmenter Model Architecture which has four learnable modules, the ViT encoder, Predictor, Corrector and Decoder blocks in order to predict semantic segmentation maps of the input sequence provided.

4.2.2 Predictor

The predictor is the most significant change to our model as opposed to any image segmentation model that is applied to a video segmentation task. This module predicts the patch embedding of a current frame given a sequence of past patch embeddings from the ViT encoder as inputs by modeling the information contained within nearby frames in a video sequence as illustrated in figure 4.3.

The embeddings y_L from the encoder are fed to the predictor in an auto-regressive manner to predict future patch embeddings, $\hat{y_L}$ as in equation 4.5. In order to predict segmentation maps of an image at time step $t + 2$, i.e $\hat{y_{L3}}$, the ViT encoder embeddings of the previous two time steps, i.e. y_{L2} and y_{L1} are used. To account for temporal continuity, a set of learnable positional embeddings are added to the ViT embeddinfs prior to the predictor module. The output from the predictor module gives a representation of the embedding of a future frame being

predicted by looking at the frames until the current time step.

$$\hat{y_{LT}} = Predictor(y_{L1}, y_{L1}, ..., y_{LT-1}) \tag{4.5}$$

4.2.3 Corrector

The corrector module acts like a fusion block. It takes in the predicted patch embeddings from the predictor, $\hat{y_{LT}}$ and fuses it with the patch embeddings generated by the ViT encoder, y_{LT} for the current frame being processed. It is the only module in the model that has access to its current ViT encoder patch embeddings. The motivation behind adding this module is that since most datasets lack annotated frames, the patch embeddings from the ViT encoder of the current frame being processed can act as the "ground truth" and the embeddings from the predictor module can act as the "predicted embeddings" as depicted in figure 4.3. This way the corrector module fuses the features extracted from both the ViT encoder and the preditor to result in spatio-temporal rich embeddings by taking into account the previous frames features. The input to the corrector module is as follows -

$${y_L}^f = Corrector([(\hat{y_L}, y_L)]) \tag{4.6}$$

4.2.4 Decoder

This module decodes the patch level embeddings from the corrector module to obtain segmentation maps. We implement two kinds of decoder architectures following the works of [34], the linear decoder and the mask transformer decoder.

Linear Decoder

A simple and effective way of obtaining segmentation maps is to pass each embedding from the sequence through a linear layer that produces patch-level logits which are then reshaped into 2-D feature maps and bilinearly upsampled to the original image size. A softmax layer is applied to obtain the segmentation maps of each image in the sequence.

Mask Transformer

A more advanced approach is to design the decoder as a transformer block following the works of [34] as depicted in figure 4.4. The mask transformer takes the patch embeddings from the corrector module and additionally a set of K learnable class

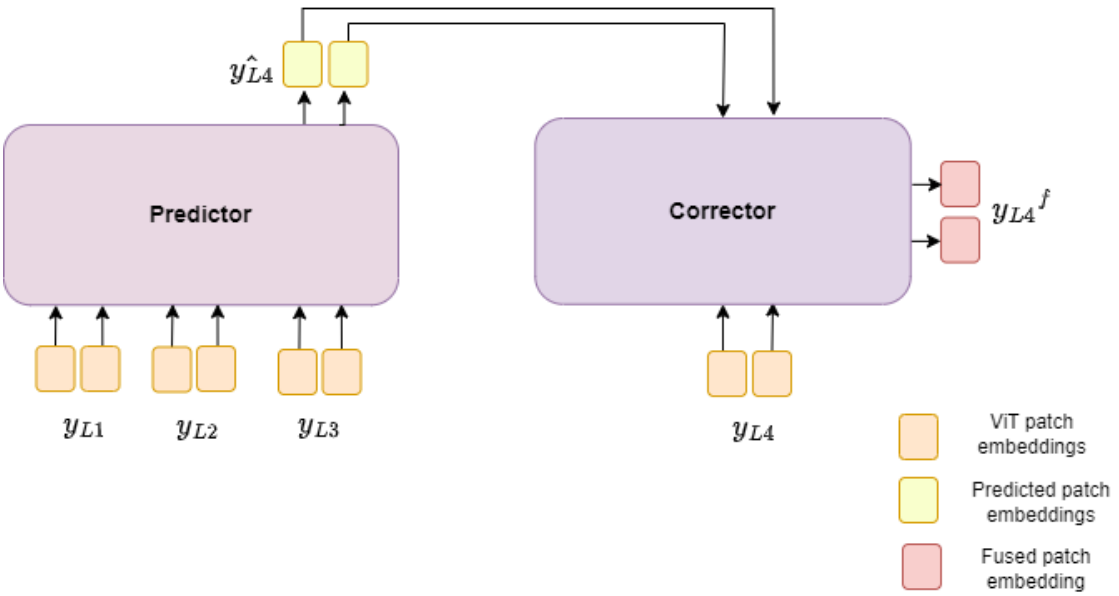


Figure 4.3: The Predictor-Corrector Architecture. Predictor takes in the patch embeddings from the ViT encoder to predict the embeddings at the subsequent time step. Out of the predicted patch embeddings, only the last one is used, which is fed to the corrector module that fuses predicted embeddings with the ViT patch embeddnfs and produces a spatio-temporal rich embeddings to be used by the decoder to produce segmentation maps.

embeddings, cls that are randomly initialized. The transformer block consists of M layers. The scalar product between the class embedding output cls' and the L2 normalized embeddings ${y_{L1}}^{M}$ are taken to generate K class masks as shown in equation 4.7. These class masks are then reshaped and bilinearly upsampled to generate the feature map. A softmax layer is then applied as in the case of the linear decoder to obtain the segmentation maps.

$$ClassMasks = {y_{L1}}^{M} \cdot {cls'}^{T} \tag{4.7}$$

4.3 Training Losses

To learn to assign pixels to the correct semantic class, we employ the Cross Entropy Loss function as shown in equation 4.8. For Cityscapes, since annotations are available only for the 19th frame, cross entropy loss is calculated only on one

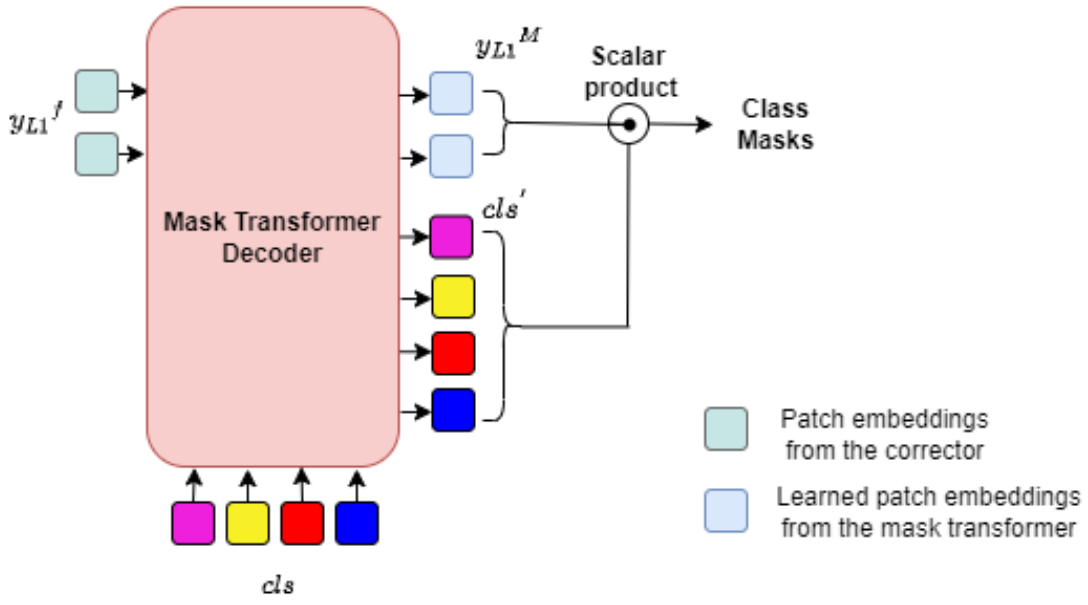


Figure 4.4: The Mask Transformer decoder. cls represent the class embeddings and $y_{L1}{}^{f}$ represent the output embeddings from the corrector module.

frame.

$$L_i = -\sum_{j=1}^{C} y_{ij} log(p_{ij})$$
$$L_{total} = \frac{1}{N}\sum_{i=1}^{N} L_i \tag{4.8}$$

where,

- L_i is the loss for pixel i.
- y_{ij} denotes the true label vector (one hot encoded) of pixel i for class j.
- p_{ij} denotes the predicted probability of pixel i for class j.
- C denotes the total number of classes.

Additionally, we add a temporal regulariser that calculates the Mean Squared Error (MSE) between the predicted patch embeddings from the predictor and the ViT embeddings from the ViT encoder. The ViT embeddings kind of act like the "ground truths" for the frames for which there are no segmenations provided and enforces temporal regularization. This is calculated as in equation 4.9.

$$L = \sum_{i=1}^{N} (y_{Li} - \hat{y_{Li}})^2 \tag{4.9}$$

where,

- $\hat{y_{Li}}$ denotes the predicted embeddings obtained from the predictor module
- y_{Li} denotes the ViT encoder embeddings obtained from the ViT encoder

4.4 Implementation details

This section describes the implementation details along with hyper parameter settings used in this thesis. All models are implemented using Pytorch [22].

Table 4.2 summarizes the hyper parameter settings used for both datasets during training of VideSegmenter model.

We trained Segmenter [34] using the SGD [15] optimizer to get results for frame-by-frame segmentations, wherein each frame in a video snippet is considered individually without considering any previous frames (history) of the video sequence. This serves as the baseline model for the course of this thesis.

Segmenter [34] uses the Vision Transformer (ViT) [13] to train the encoder. The ViT has various variants like "tiny", "base" and "large". They differ in the number of layers, heads and the dimension size used in the transformer blocks. Details are mentioned in Table 4.1.

Table 4.1: Variants in the Vision Transformer as described in [13].

Model	#layers	#heads	dim
tiny	12	3	192
base	12	12	768
large	24	16	1024

For the most part, we make use of the "tiny" variant due to computation limitations. We also use the "base" version to train a model on Synpick to experiment with patch size as a small ablation study.

The ViT encoder is initialised using pretrained weights obtained from training the frame-by-frame model. These were kept frozen throughout all experiments.

The predictor and corrector modules are regular transformers as explained in section 2.4 and consist of six and three layers for the Cityscapes[10] and eight and two layers for the Synpick[23] dataset, respectively.

The decoder takes the form of a linear decoder or a mask transformer that consists of two layers.

We use patch sizes of 16 × 16 to train all models. Additionally we also train a model on Synpick that uses a patch size of 8 × 8.

Training is done following the MM-Segmentation [9] library. All images are randomly cropped to a fixed size of 768 × 768 for Cityscapes and 128 × 128 for Synpick. They are randomly flipped with a probability of 0.5. Photometric distortion [29] is also applied as means of data augmentation.

Dropout [33] and Stochastic Depth [18] with values 0.1 are used as regularizers for the experiments conducted. Stochastic depth randomly drops blocks of the transformer with a probability of 0.1 while dropout randomly drops the neurons given as input to the transformer block. All models use the scheduling process similar to the one used by Strudel et al. [34] that employs the poly learning rate decay as $lr_0 = lr(1 - \frac{N_{iter}}{N_{total}})^{0.9}$ wherein N_{iter} represents the current iteration and N_{total} represent the total number of iterations.

Cityscapes [10] is trained using an Adam optimizer [19] with a learning rate of 0.0005 while Synpick [23] is trained using the SGD [15] optimizer with a learning rate of 0.005.

A sequence size of four frames is used to train all models. Since Cityscapes [10] consists of only one annotated frame per video sequence, we keep the annotated frame as the last frame in the sequence while training. We also experiment with keeping the annotated frame as the second frame in the sequence but see no significant improvement/decline in performance.

Table 4.2: Summarized hyper-parameters used during training on VideoSegmenter.

Hyper-param	**Cityscapes**	**Synpick**
ViT variant	tiny	tiny/base
Layers_pred	6	8
Layers_corr	3	2
Image size	768 × 768	128 × 128
Optim	Adam	SGD
LR	5e4	5e-3
Patch size	16 × 16	16 × 16 / 8 × 8

5 Experiments

In this chapter, we explain the datasets used in section 5.1 to evaluate our VideoSegmenter model. Further, Section 5.2 mentions the two metrics, Mean Intersection over Union and Mean Accuracy used to evaluate the models. Finally, in sections 5.3 and 5.4, we present our experimental findings both quantitatively and qualitatively on both datasets.

5.1 Datasets

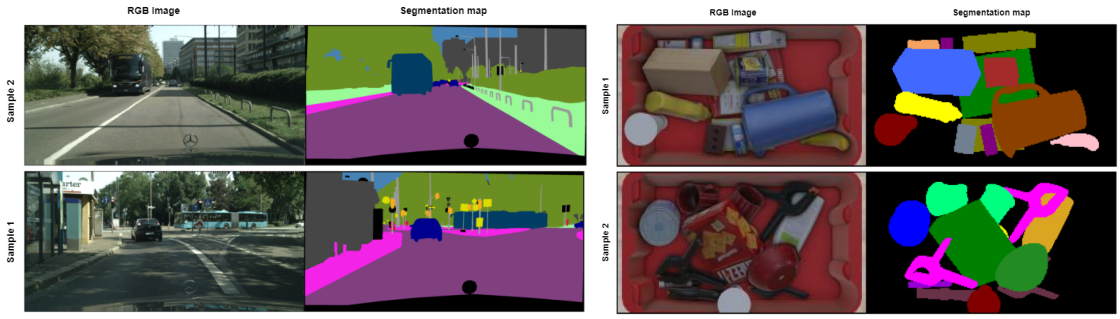
We evaluate our model for Video Semantic Segmenation on the Cityscapes [10] and Synpick [23] datasets.

Cityscapes [10]

This benchmark dataset widely used in urban scene understanding and autonomous driving tasks contains 5000 RGB images collected over 50 different German cities. They are split into 2975 images used for training, 1525 for testing and 500 for the validation set. Figure 5.1a shows two frames from the dataset along with their segmentation maps. There are 30 labeled classes available, however only 19 of them are used in order to train the network. Each sequence contains 30 frames, of them annotations are available only for 19th frame. We use a sequence size of four in order to train the network and make sure that the 19th frame is one of the frames included in the sequence.

Synpick [23]

We also test our model on Synpick [23], a synthetic dataset used for dynamic scene understanding in bin-picking. The scene consists of a gripper robot that moves in different directions pushing 21 objects contained in the box. We make sure that the robotic gripper is present in all cases. A snippet from the datasets sample is shown in figure 5.1b



(a) Cityscapes dataset sample (b) Synpick dataset sample

Figure 5.1: Two sample frames from both datasest with their segmentation maps

5.2 Metrics

This section elaborates on the metrics used to evaluate our model, namely Mean Intersection over Union and Mean Accuracy.

Mean Intersection over Union (MIoU)

Mean Intersection over Union (MIoU) is a common metric used to evaluate segmentation results which measures the degree of overlap between the predicted and ground truth segmentation maps of each frame. It is calculated as in equation 5.1 for multi class segmentation predictions.

$$\begin{aligned} \text{Intersection over union } (IoU) &= \frac{groundtruth \cap predicted}{groundtruth \cup predicted} \\ \text{Mean Intersection over Union } (MIoU) &= \frac{IoU}{N} \end{aligned} \tag{5.1}$$

wherein N denotes the number of classes.

Mean Accuracy

We report Mean Accuracy across the semantic classes. For each of the class labels, mean accuracy is calculated and then averaged over these class-wise accuracies to produce the final metric value. Calculation is done as in equation 5.2

$$\text{Mean Accuracy} = \frac{1}{N} \sum_{i=1}^{N} \frac{\text{TP}_i}{\text{TP}_i + \text{FN}_i} \tag{5.2}$$

wherein,

- TP_i denotes the number of correctly classified pixels for class i (True positives).
- FN_i denotes the number of pixels that are misclassified for class i (False negatives) .
- N denotes the number of classes.

5.3 Quantitative results

This section explains the results obtained on the two datasets quantitatively. Table 5.1 summarizes the results on the best performing models on both datasets, i.e. Cityscapes and Synpick as compared to Segmenter's baseline model. VideoSegmenter trained with a mask transformer decoder and with MSE temporal regularisation achieves the best results on Cityscapes, while the model trained with the base variant of ViT with a patch size of 8×8 along with a linear decoder achieves best results on Synpick. Further, tables 5.2 and 5.5 talk in detail about individuals experiments on both datasets separately. We report MIoU and Mean accuracy for all models.

Table 5.1: Summarized best results on Cityscapes and Synpick datasets

	Cityscapes		**Synpick**	
	MIoU	Mean Acc	MIoU	Mean Acc
Segmenter	68.59	77.72	73.8	82.3
VideoSegmenter/Linear	70.55	78.74	75.13	83.38
VideoSegmenter/Mask w/ MSE	**73.46**	**82.11**	—	—
VideoSegmenter/Linear/Base w/o MSE	—	—	**87.49**	**92.72**

Cityscapes

Table 5.2 summarizes the experiments conducted on Cityscapes along with their metric findings. VideoSegmenter trained with the linear decoder shows a rise of +1.5% in MIoU as compared to the one trained on Segmenter model (frame by frame results). When experimenting with using MSE as the temporal regularisation loss, we notice a 0.5% rise in MIoU. Another significant improvement is noticed when using a mask transformer decoder as opposed to a linear decoder with the highest MIoU recorded on Cityscapes to be 73.46 when trained using the mask transformer and the additional MSE loss.

We believe training Cityscapes using a more powerful encoder (e.g. the ViT base variant) will yield best results. This experiment was not performed due to large training times and memory consumption. However, the experiment with the base variant and a patch size of 16 × 16 was performed on the frame-by-frame model which resulted in a MIoU of 77.94%. This gives us the intuition that our VideoSegmenter model would have produced better results when trained using the base variant instead of the tiny variant.

Table 5.2: Quantitative results obtained on the Cityscapes dataset.

Results - Cityscapes		
Model	MIoU	Mean Acc
Segmenter (Baseline)	68.59	77.72
VideoSegmenter/Linear w/o MSE	70.11	78.4
VideoSegmenter/Linear w/ MSE	70.55	78.74
VideoSegmenter/Mask w/o MSE	72.92	81.46
VideoSegmenter/Mask w/ MSE	**73.46**	**82.11**

Table 5.3 gives insights on the Intersection over Union per class and gives the top four classes that show considerable improvements in values as compared to the baseline model. Moving classes like the bicycle and rider categories show significant improvements. This shows that our model is able to model temporal continuity and accounts for moving objects between frames. Table 5.4 mentions the two classes that perform slightly worse than the baseline model.

It is to be noted that our model relies heavily on the Segmenter models results since the ViT encoder embeddings are frozen throughout training. This results in similar findings with considerable improvements.

Table 5.3: The top four classes that give a significant rise in IoU_{class} values when compared with the Segmenter model on Cityscapes dataset.

Class	**Segmenter-IoU**$_{class}$	**VideoSegmenter-IoU**$_{class}$
wall	55.86	59.50
traffic light	51.6	56.28
rider	45.97	52.34
bicycle	67.92	70.02

Synpick

Table 5.5 summarises the experimental results obtained on the Synpick dataset. When using the tiny version of the ViT as the pretrained encoder and patch size

Table 5.4: The only two classes that give worse values for IoU_{class} when compared with the per class IoU values for Segmenter vs VideoSegmenter both trained using the linear decoder.

Class	**Segmenter-IoU**$_{class}$	**VideoSegmenter-IoU**$_{class}$
sky	93.37	93.31
truck	64.62	63.29

of 16 × 16, we obtain best results when trained using a linear decoder with Mean Squared Error (MSE) regularisation. It is noted that using mask transformer only slightly improves results on Synpick as compared to the improvements noticed on Cityscapes.

A significant rise +12% is gained when training using the pretrained weights from the segmenter model trained on the base variant of the ViT and a patch size of 8 × 8. Decreasing the patch size leads to better results as the model is able to capture boundaries in a better manner but comes with the overhead of increasing training times due to the large patch embeddings sequence.

Table 5.5: Quantitative results obtained on the Synpick dataset.

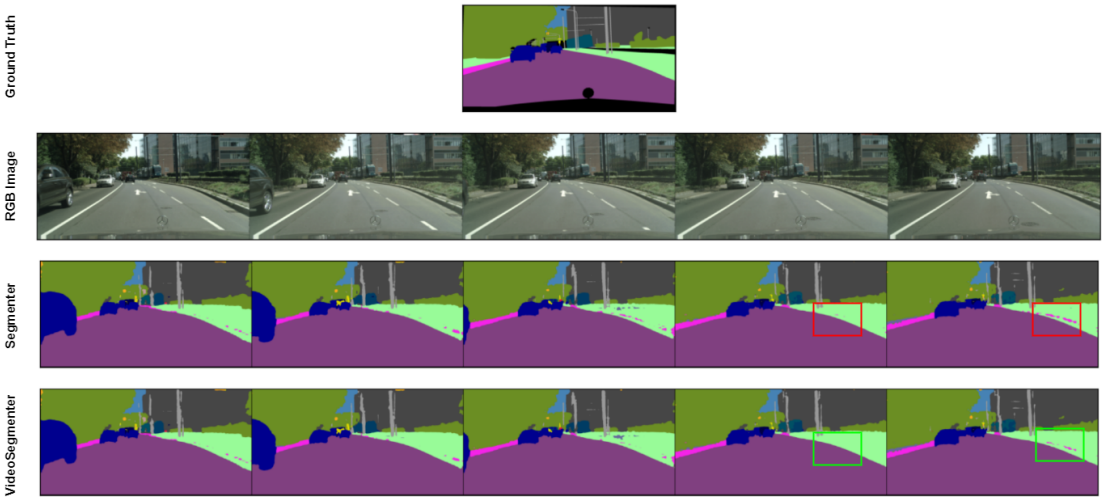
Results - Synpick		
Model	MIoU	Mean Acc
Segmenter (Baseline)	73.8	82.3
VideoSegmenter/Linear w/o MSE	74.56	82.66
VideoSegmenter/Linear w/ MSE)	75.17	83.43
VideoSegmenter/Mask w/o MSE	74.72	83.65
VideoSegmenter/Linear/Base w/o MSE	**87.50**	**92.75**

5.4 Qualitative results

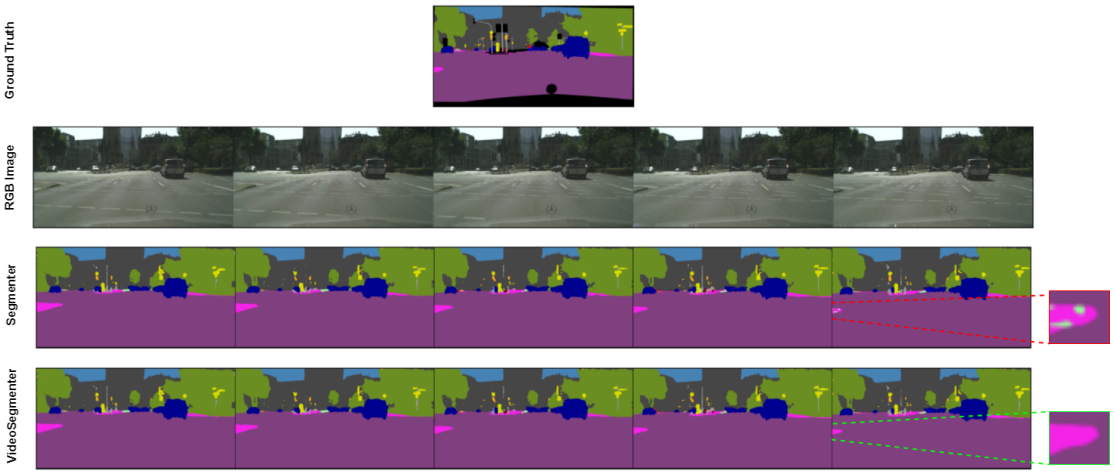
This section describes the qualitative results from the experiments that were done during the course of the thesis.

Cityscapes

Figure 5.2 shows two examples of segmentation maps predicted when using a mask transformer as our decoder which is our best performing model. Results show a comparison between visuals from the Segmenter model vs our VideoSegmenter results. Some improvements from our model are highlighted in the images and prove to be more consistent across time steps. Our model is trained using a



(a) Terrain is predicted a little more consistently (highlighted in green) as compared to the predictions from segmenter model



(b) The enlarged part of the image shows the difference between our model and the segmenter model. Our model correctly predicts the sidewalk even in the last frame as it makes use of embeddings from previous frames.

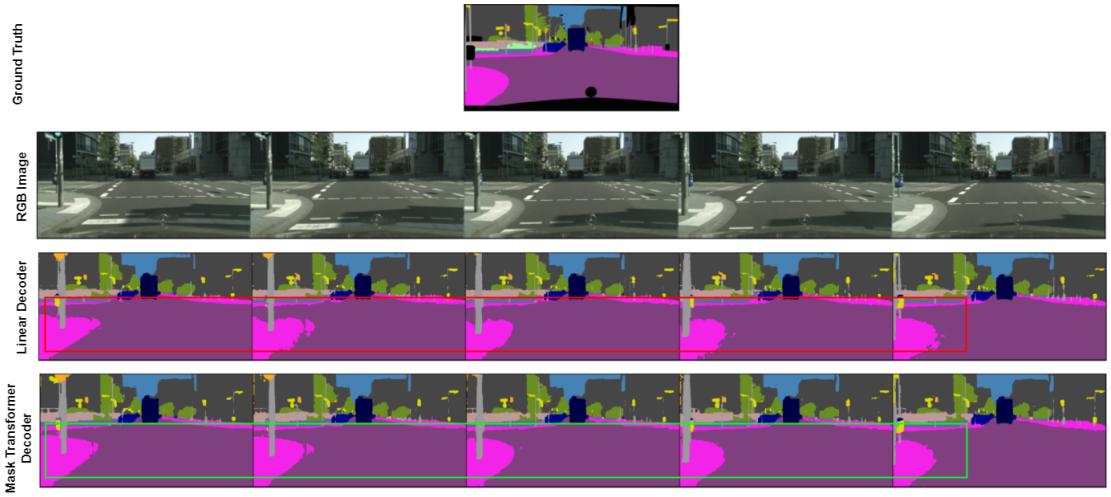
Figure 5.2: Results on Cityscapes dataset comparing the baseline model trained using Segmenter [34] with out best performing VideoSegmenter model trained using the mask transformer with regularization MSE loss

sequence size of 4. Example 5.2b shows that even when the frame gets out of the training horizon, our model classifies the sidewalk correctly as opposed to the noisy prediction generated by the Segmenter model. This shows that using the history of past frames to predict the future frames prove to give more consistent results.

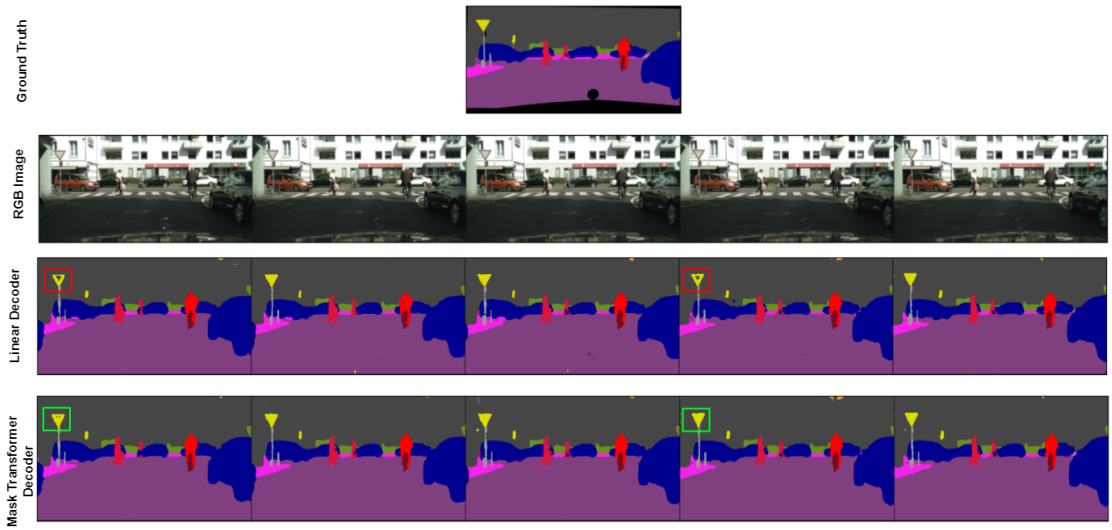
Further, results from training using a linear decoder vs a mask transformer are visualised in figure 5.3 . It can be seen that there is a significant improvement in visuals when training using a mask transformer as the decoder. The segmentation maps are more consistent across the time frames. Figure 5.3b shows significant improvements when predicting sidewalks while figure 5.3a shows more consistency in predicting the traffic signs across the frames as highlighted by the boxes.

Synpick

Figure 5.4 shows improvements obtained by our model as compared to the results from the frame by frame method used in Segmenter [34] and highlights them. Figure 5.4a shows that the object belonging to the class banana depicted in light pink changes its shape when predicted using the frame by frame model by Segmenter [34] from frame three onwards. However, when tested on our VideoSegmenter model, it provides consistent results and the shape of the object is maintained until the 8th frame as outlined in green colour. As a small ablation study on the variant of the ViT [13] transformer block used to train the Segmenter [34] model and patch size, we experiment with the base version of the ViT along with a patch size of 8×8. As seen in figure 5.5, we see that the object boundaries are much more clearer and well-defined. A particularly difficult class, the clamp can be seen predicted quite well as compared to Segmenter results also visualized in figure 5.5.

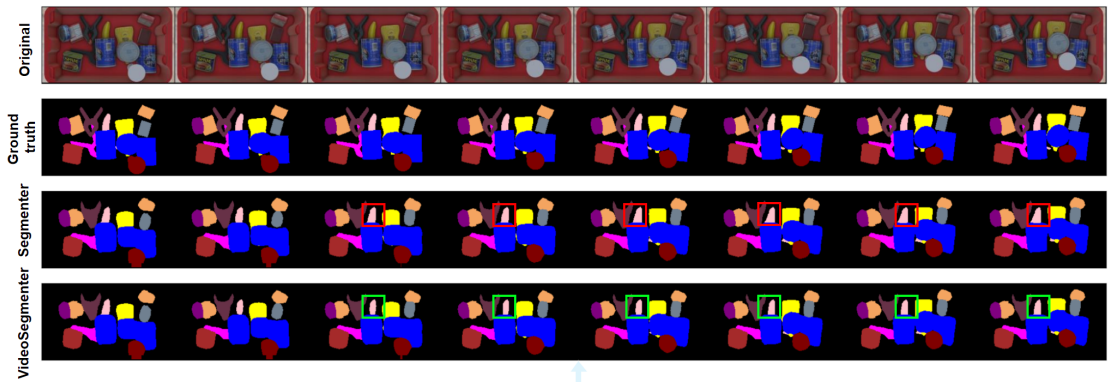


(a) The sidewalks highlighted in red and green show the difference in segmentation maps.

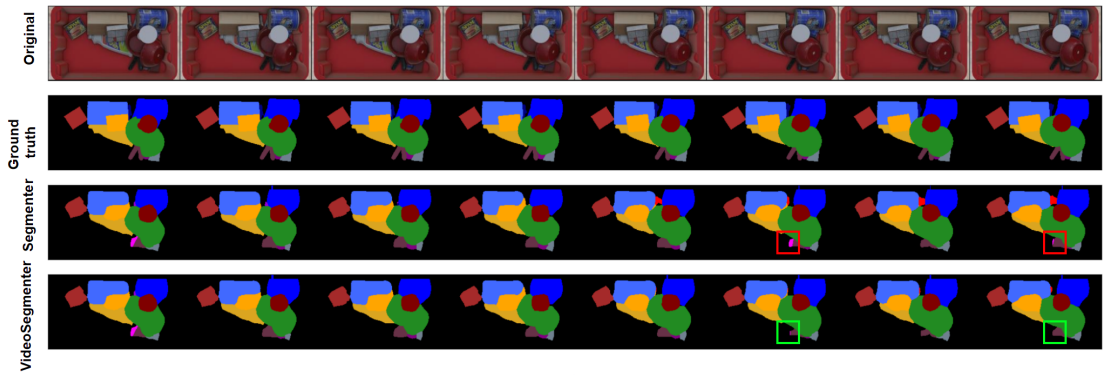


(b) The traffic sign can be seen predicted consistently in all frames. The difference is highlighted in red and green.

Figure 5.3: Results on Cityscapes demonstrating the effectiveness of using a mask transformer instead of a linear decoder.



(a) Objected highlighted is a banana depicted in a ligh pink shade. The segmentation obtained by using Segmenter [34] model changes the shape to being fat than the actual object as highlighted in red. Our model correctly predicts the object as the frame progresses.



(b) The object highlighted is a clamp depicted in brown. In frames 6 and 8, the Segmenter [34] model gives noisy predictions. These are improved when employing our model as highlighted in green.

Figure 5.4: Qualitative results on the Synpick dataset comparing results obtained by training Segmenter [34] and our VideoSegmenter model using the linear decoder with regularization loss.

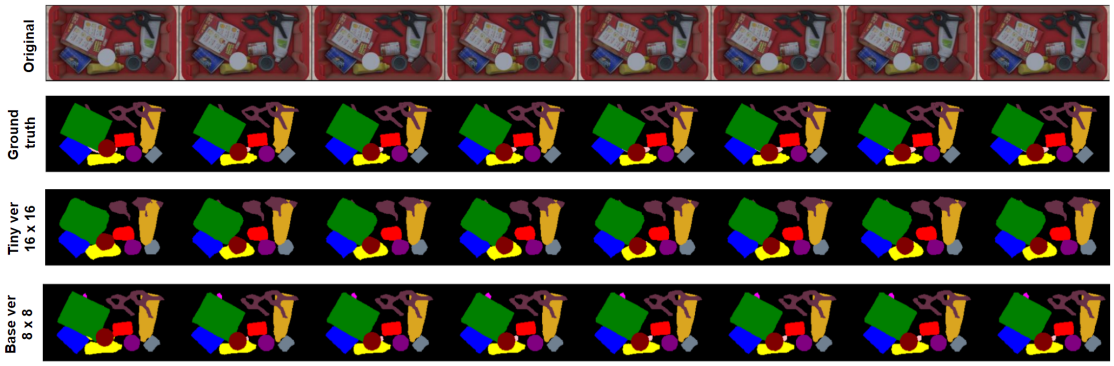


Figure 5.5: Synpick results when using the tiny and base variants of the ViT [13]. The base variant uses a patch size of 8×8. The latter gives more defined outlines for the objects and predicts objects like clamp depicted in brown very well.

6 Conclusion and Future Work

This chapter summarizes the work done during the course of this thesis and presents the future scope in the field that can be done to improve results.

Our proposed VideoSegmenter model that provides segmentation for each frame in a video sequence in an auto-regressive manner by incorporating temporal relations between consecutive frames successfully improves the results by around 2% for both datasets. Additionally, we show qualitatively that our model gives more consistent results across a time frames, which in turn leads to less noisy predictions as compared to results from a frame-by-frame segmentation model that does not account for temporal relations.

We also employ means to enforce temporal regularisation between consecutive frames by employing the Mean Squared Error loss between the predicted embeddings and the ViT embeddings, leading to even better results. However, results can be further improved by fine-tuning the weighting hyperparameter in a better way.

Although we achieve significant results through our VideoSegmenter model, due to time and computation constraints, some experiments were left out. These are outlined below that could be conducted in order to achieve better results.

We use the tiny version of the ViT transformer variant [13] to model our ViT encoder. Employing the pretrained base/large version of the ViT transformer would lead to better results. However, the model size is too large and would require large amounts of memory and training times.

Decreasing patch sizes to a small number like 8×8 as experimented with the Synpick [23] dataset leads to better, sharper and more defined results. This could not be employed with the Cityscapes dataset owning the high training times due to a larger patch sequence size. In the future, this could be tested to lead to much better results.

In conclusion, our VideoSegmenter architecture learns to model temporal continuity between frames for the task of Video Semantic Segmentation clearly outperforming the strong Segmenter baseline. Addressing the above points is crucial and would lead to more promising results in the future.

List of Figures

List of Tables

Bibliography

[1] Abien Fred Agarap. *Deep learning using rectified linear units (relu)*. 2019. arXiv: 1803.08375 [cs.NE].

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer normalization*. 2016. arXiv: 1607.06450 [stat.ML].

[3] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. "Segnet: a deep convolutional encoder-decoder architecture for image segmentation." In: *Ieee transactions on pattern analysis and machine intelligence* 39.12 (2017), pp. 2481–2495.

[4] Gabriel Brostow, Julien Fauqueur, and Roberto Cipolla. "Semantic object classes in video: a high-definition ground truth database." In: *Pattern recognition letters* 30 (Jan. 2009), pp. 88–97. DOI: 10.1016/j.patrec.2008.04.005.

[5] Siddhartha Chandra, Camille Couprie, and Iasonas Kokkinos. "Deep spatio-temporal random fields for efficient video segmentation." In: *Proceedings of the ieee conference on computer vision and pattern recognition*. 2018, pp. 8915–8924.

[6] Siddhartha Chandra and Iasonas Kokkinos. "Fast, exact and multi-scale inference for semantic image segmentation with deep gaussian crfs." In: *Computer vision–eccv 2016: 14th european conference, amsterdam, the netherlands, october 11–14, 2016, proceedings, part vii 14*. Springer. 2016, pp. 402–418.

[7] Siddhartha Chandra, Nicolas Usunier, and Iasonas Kokkinos. "Dense and low-rank gaussian crfs using deep embeddings." In: *Proceedings of the ieee international conference on computer vision*. 2017, pp. 5103–5112.

[8] Kyunghyun Cho et al. *On the properties of neural machine translation: encoder-decoder approaches*. 2014. arXiv: 1409.1259 [cs.CL].

[9] MMSegmentation Contributors. *MMSegmentation: openmmlab semantic segmentation toolbox and benchmark*. https://github.com/open-mmlab/mmsegmentation. 2020.

[10] Marius Cordts et al. "The cityscapes dataset for semantic urban scene understanding." In: *Proceedings of the ieee conference on computer vision and pattern recognition*. 2016, pp. 3213–3223.

[11] Jacob Devlin et al. *Bert: pre-training of deep bidirectional transformers for language understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[12] Mingyu Ding et al. "Every frame counts: joint learning of video segmentation and optical flow." In: *Proceedings of the aaai conference on artificial intelligence*. Vol. 34. 07. 2020, pp. 10713–10720.

[13] Alexey Dosovitskiy et al. "An image is worth 16x16 words: transformers for image recognition at scale." In: *Arxiv preprint arxiv:2010.11929* (2020).

[14] Mohsen Fayyaz et al. "Stfcn: spatio-temporal fcn for semantic video segmentation." In: *Arxiv preprint arxiv:1608.05971* (2016).

[15] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. http://www.deeplearningbook.org. Cambridge, MA, USA: MIT Press, 2016.

[16] Kaiming He et al. *Deep residual learning for image recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[17] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory." In: *Neural computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[18] Gao Huang et al. *Deep networks with stochastic depth*. 2016. arXiv: 1603.09382 [cs.LG].

[19] Diederik P. Kingma and Jimmy Ba. *Adam: a method for stochastic optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[20] Jiangtong Li et al. "Video semantic segmentation via sparse temporal transformer." In: *Proceedings of the 29th acm international conference on multimedia*. MM '21. Virtual Event, China: Association for Computing Machinery, 2021, pp. 59–68. ISBN: 9781450386517. DOI: 10.1145/3474085.3475409. URL: https://doi.org/10.1145/3474085.3475409.

[21] David Nilsson and Cristian Sminchisescu. *Semantic video segmentation by gated recurrent flow propagation*. 2017. arXiv: 1612.08871 [cs.CV].

[22] Adam Paszke et al. *Pytorch: an imperative style, high-performance deep learning library*. 2019. arXiv: 1912.01703 [cs.LG].

[23] Arul Selvam Periyasamy, Max Schwarz, and Sven Behnke. "Synpick: a dataset for dynamic bin picking scene understanding." In: *2021 ieee 17th international conference on automation science and engineering (case)*. IEEE. 2021, pp. 488–493.

[24] Andreas Pfeuffer, Karina Schulz, and Klaus Dietmayer. "Semantic segmentation of video sequences with convolutional lstms." In: *2019 ieee intelligent vehicles symposium (iv)*. IEEE. 2019, pp. 1441–1447.

[25] Vishnu Pradeep et al. "Self-supervised sidewalk perception using fast video semantic segmentation for robotic wheelchairs in smart mobility." In: *Sensors* 22.14 (2022), p. 5241.

[26] Alec Radford et al. "Improving language understanding by generative pre-training." In: (2018).

[27] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors." In: *Nature* 323 (1986), pp. 533–536. URL: https://api.semanticscholar.org/CorpusID:205001834.

[28] Xingjian Shi et al. *Convolutional lstm network: a machine learning approach for precipitation nowcasting*. 2015. arXiv: 1506.04214 [cs.CV].

[29] Connor Shorten and Taghi Khoshgoftaar. "A survey on image data augmentation for deep learning." In: *Journal of big data* 6 (July 2019). DOI: 10.1186/s40537-019-0197-0.

[30] Mennatullah Siam et al. "A comparative study of real-time semantic segmentation for autonomous driving." In: *Proceedings of the ieee conference on computer vision and pattern recognition (cvpr) workshops*. June 2018.

[31] Mennatullah Siam et al. "Video object segmentation using teacher-student adaptation in a human robot interaction (hri) setting." In: *2019 international conference on robotics and automation (icra)*. 2019, pp. 50–56. DOI: 10.1109/ICRA.2019.8794254.

[32] Karen Simonyan and Andrew Zisserman. *Very deep convolutional networks for large-scale image recognition*. 2015. arXiv: 1409.1556 [cs.CV].

[33] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting." In: *Journal of machine learning research* 15 (June 2014), pp. 1929–1958.

[34] Robin Strudel et al. "Segmenter: transformer for semantic segmentation." In: *Proceedings of the ieee/cvf international conference on computer vision*. 2021, pp. 7262–7272.

[35] Yuan Tian, Tao Guan, and Cheng Wang. "Real-time occlusion handling in augmented reality based on an object tracking approach." In: *Sensors* 10.4 (2010), pp. 2885–2900.

[36] Sercan Türkmen. "Scene understanding through semantic image segmentation in augmented reality." MA thesis. S. Türkmen, 2019.

[37] Sepehr Valipour et al. *Recurrent fully convolutional networks for video segmentation*. 2016. arXiv: 1606.00487 [cs.CV].

[38] Sepehr Valipour et al. "Recurrent fully convolutional networks for video segmentation." In: *2017 ieee winter conference on applications of computer vision (wacv)*. 2017, pp. 29–36. DOI: 10.1109/WACV.2017.11.

[39] Greg Van Houdt, Carlos Mosquera, and Gonzalo Nápoles. "A review on the long short-term memory model." In: *Artificial intelligence review* 53 (Dec. 2020). DOI: 10.1007/s10462-020-09838-1.

[40] Ashish Vaswani et al. "Attention is all you need." In: 2017. URL: https://arxiv.org/pdf/1706.03762.pdf.

[41] Hao Wang, Weining Wang, and Jing Liu. "Temporal memory attention for video semantic segmentation." In: *2021 ieee international conference on image processing (icip)*. IEEE. 2021, pp. 2254–2258.

[42] Yang Wang et al. *Occlusion aware unsupervised learning of optical flow*. 2018. arXiv: 1711.05890 [cs.CV].

[43] Wenhui Zhang and Tejas Mahale. "End to end video segmentation for driving: lane detection for autonomous car." In: *Arxiv preprint arxiv:1812.05914* (2018).