A DEEP MOTION VECTOR APPROACH TO VIDEO OBJECT SEGMENTATION

A Thesis

by

VINEET GARG

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Chao Tian
Committee Members,	Anxiao Jiang
	Ulisses Braga-Neto
	Zixiang Xiong
Head of Department,	Miroslav M. Begovic

May 2019

Major Subject: Electrical Engineering

ABSTRACT

Video object segmentation is gaining increased research and commercial importance in recent times from no checkout lines in Amazon Go stores to autonomous vehicles operating on roads. Efficient operation for such use cases require segmentation inference in real time. Even though there has been significant research in image segmentation, both semantic and instance, there is still much scope for improvement in video segmentation. Video segmentation is a direct extension of image segmentation, except that there is temporal relation between neighboring frames of videos. Exploiting this temporal relation in an efficient way is one of the most important challenges in video segmentation. This temporal relation has a lot of redundancy involved and many of the prevalent state-of-the-art techniques do not exploit this redundancy.

Optical flow is one of the approaches for exploiting temporal redundancies. Intermediate feature maps of previous frames are interpolated using this information and rest of the segmentation operation is performed. However, optical flow provides motion resolution on a pixel level. There is not enough motion between consecutive frames to warrant motion estimation on pixel level. Instead we can divide a frame into multiple blocks and estimate the movement of their centroids in consecutive video frames.

Based on this idea, we present a motion vector approach to video semantic segmentation. Additionally, we also propose an adaptive technique to select keyframes during inference. We show that our proposed algorithm can bring down the computational complexity during inference by as much as 50% with only a 2-3% drop in the accuracy metric. Our algorithm can operate at as high as 136 frames per second indicating that it can easily handle real time inference.

DEDICATION

To my parents

ACKNOWLEDGMENTS

I wish to express my sincere gratitude to my advisor Dr. Chao Tian for providing guidance and support throughout my graduate studies. I also wish to thank Dr. Anxiao Jiang, Dr. Ulisses Braga-Neto and Dr. Zixiang Xiong for serving on my thesis committee.

A special thanks to my friends Arnav and Palash for many of the intense and deep discussions on deep learning and computer vision which significantly increased my understanding of the subject. Finally, I wish to thank all my family and friends in College Station and back in India for their constant support and encouragement.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supervised by a thesis committee consisting of Professor Chao Tian and Professor Zixiang Xiong and Professor Ulisses Braga Neto of the Department of Electrical and Computer Engineering and Professor Anxiao Jiang of the Department of Computer Science and Engineering.

Funding Sources

Graduate study was supported by a non-resident tuition waiver from the Department of Electrical and Computer Engineering at Texas A&M University.

Many of the simulations were performed on GPU servers graciously provided by Dr. Anxiao Jiang of the Department of Computer Science and Engineering.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION AND LITERATURE REVIEW

Image and Video Segmentation has been an active research for past 2 decades. The research in both these areas was more of theoretical in nature due to absence of enough computation resources. Many of the deep learning ideas were available even in 90s, but active research never happened due to lack of computational resources. However, progress made in GPU architectures, combined with active research in deep learning, has significantly propelled the advances in both image and video segmentation (especially image segmentation) in past decade. Even though video segmentation can be thought of as a direct extension of image segmentation, except that video frames have temporal relations between them, there is still a significant scope of improvement.

We know that neighboring frames in videos are highly correlated. Using this information, we can interpolate intermediate feature maps to perform final segmentation. In this thesis presentation, we present techniques which can bring down the computational complexity in video segmentation. Specifically, we explore the use of motion vectors, used in video encoding, combined with deep learning architectures to perform video semantic segmentation. We show how the use of motion vector can significantly bring down the redundancy and time complexity for the video segmentation tasks with a little drop in accuracy metric.

1.1 Deep Learning

We first present some background on the recent deep learning research to better understand the advances in video segmentation. Deep learning gained prominence when AlexNet [1] achieved sub 15% error rate and over 10% improvement with respect to existing techniques on the ImageNet dataset [2]. Earlier classical machine learning techniques for image classification and object detection used to focus on hand-crafted features. However, deep learning operations such as convolutional nets combined with proper initialization (Xavier [3], He [4]) and activations (e.g. ReLU) can efficiently extract spatial features in the images

and perform classification and detection task very efficiently.

There have been multiple architectures which later improved over AlexNet's performance e.g. VGG16 [5], ResNet [6], GoogleNet [7] to name a few. Since all of these architectures are built for classification, they have a fully connected softmax layer on top of extracted feature maps. Most of the image and video segmentation networks remove the last fully connected layer and use the previous layers as a backbone component to extract feature maps. Essentially, they are built on top of these popular backbone architectures.

1.2 Image Segmentation

There are 2 type of segmentations prevalent in the recent literature semantic segmentation and instance segmentation. In instance segmentation, we detect each occurrence of an object class uniquely. The detection can be in terms of a bounding box or on a pixel level. One of the earliest attempts towards instance segmentation was R-CNN [8] where authors generated bounding box proposals for every image using selective search [9]. These bounding boxes are passed to a modified version of AlexNet followed by an SVM classifier to detect a particular object. The object detection is posed as a regression problem to tighten the bounding box proposals. It is evident that this architecture is passing every region proposal through the complete network leading to redundancy. An improved version Fast R-CNN [10] was proposed to decrease the redundancy. Authors proposed running the backbone architecture only once on the complete image under consideration and later use the feature maps to generate bounding box proposals. Fast R-CNN also combines feature extractor CNN layers, classifier and regressor into a single network. An improved version of Fast R-CNN, Faster R-CNN [11] was proposed which focused on accelerating the region proposal step.

Both R-CNN and Fast R-CNN used Selective Search to generate region proposals. However, it was observed that Selective Search is acting as bottleneck for the whole process. A region proposal network (RPN) is added to generate regions of different aspect ratio for segmentation. All the algorithms discussed yet give the instance segmentation in a bounding box format. Finally, a pixel level extension of Faster R-CNN was proposed. Mask R-CNN

[12] adds a parallel branch to the Faster R-CNN architecture. The parallel branch takes the region proposals of feature map as input in a fully convolutional layer and outputs a binary mask indicating whether a particular pixel belongs to object or background. Since, the shape of feature map and original image is different, use of bilinear interpolation is suggested to avoid rounding off the pixels leading to precise instance level object segmentation.

In semantic segmentation, we give class-wise segmentation on a pixel level. Every unique occurrence of an object class is not characterized uniquely, instead unique occurrences of same class are labelled as belonging to the same class. Because of this, in semantic segmentation, we do not need a separate region proposal network. Semantic segmentation can be thought of multi-label classification on a pixel level. Intermediate feature maps are significantly downsampled as compared to the original image resolution due to the pooling operations and final output mask needs to be in the dimension as the image. Semantic segmentation algorithms employ a variety of upsampling algorithms to reach the same resolution as input image.

First of such algorithms was the Fully Convolutional Networks (FCN) [13] which uses deconvolutional layers to upsample the intermediate feature maps. Additionally, it uses skip connections from relatively higher resolution feature maps to aid the process of upsampling. SegNet [14] stored the feature map indices during pooling operations and used this information to upsample (un-pool). Later [15] introduced the concept of dilated convolutions (atrous convolution) to increase the receptive field during upsampling. Finally, there have been a series of 4 papers referred to as DeepLab which are considered as benchmark algorithms for semantic segmentation. The final paper in this series DeepLabv3+ [16] uses an atrous spatial pyramid pooling (ASPP) module combined with a skip connection from initial block of the backbone network to output the final segmentation result.

1.3 Video Segmentation

Multiple algorithms, both classical and deep learning based, enforce temporal relations in neighboring frames. One such paper is ObjectFlow [17] which uses optical flow to prop-

agate and estimate object segmentation for every mask. Since optical flow estimates can be incorrect especially at the object boundaries, it proposed an iterative scheme to estimate optical flow and perform object segmentation. Segmentation is performed by propagating foreground labels through a graphical model consisting of pixels and superpixels. [18] was the first work (also referred as MaskTrack) to show that a deep learning network designed for semantic image segmentation can be used for video segmentation task as well. It feeds current frame's estimated segmentation mask to the next frame to enforce temporal consistency. The mask estimate acts as a guide for next frame's segmentation. MaskTrack is a semi-supervised approach towards segmentation. In semi-supervised problems, ground truth is available for only the first frame during validation and testing phases. The network is trained during the training phase in an offline manner. Later, the network is fine tuned for the first frame's ground truth during validation and testing.

During the same time-frame, another semi-supervised technique OSVOS [19] was proposed. OSVOS argues against enforcing any kind of temporal relation during segmentation. Instead, it proposes to treat every frame as a standalone image and perform segmentation on that. It was observed that temporal relation is achieved as a by-product. Because this approach does not exploit temporal information, it performs really well on cases when objects reappear in video sequence or when objects are occluded.

It can be clearly observed that both OSVOS and MaskTrack do not exploit the temporal information efficiently, instead they both compute feature maps for every frame in the sequence. Most of the image or video segmentation networks have 2 parts: a backbone network and a recognition network. Backbone Network such as ResNet101 acts as a feature extractor and computes high level features associated with the frame. Recognition Network performs upsampling and final segmentation on the computed feature maps. Backbone Network takes majority chunk of the computational resources consumed in the whole process.

Deep Feature Flow [20] (DFF) proposed an algorithm along these lines. Referring to Figure 1.1 [20], it can be observed that spatial relations are maintained even in the low-resolution

feature maps. DFF utilizes this information by interpolating feature maps in neighboring frames using the optical flow information. A separate network FlowNet [21] is designed for estimating optical flow. This approach brings down the computational complexity significantly with a little drop in the accuracy metric.

During the same time, we were conducting our research, [22] proposed an approach similar to our algorithm. However, their work does not address the adaptive keyframe selection problem and fusion of low level features during upsampling.

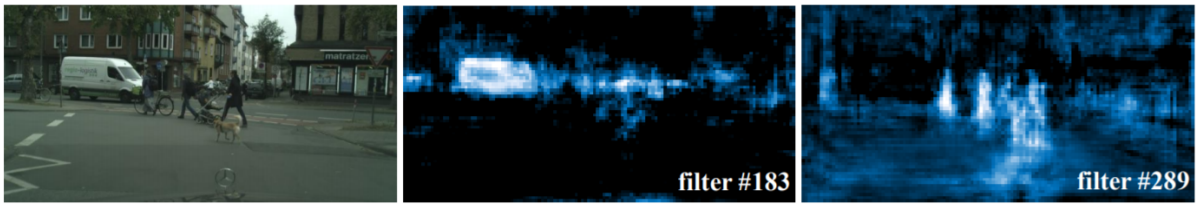


Figure 1.1: Illustration of feature map filters

[23] proposed a modulator-based approach to perform semi-supervised segmentation. It uses a modulator network to adjust the intermediate layers given a generic segmentation mask (object of interest). Effectively, it acts like meta-leaning algorithms which uses meta-data from previous experiences to adapt to new situations with a limited number of gradient updates. Finally, we talk about another recent algorithm [24] which attempts to fully enforce the temporal relation in neighboring frames. It uses Convolutional LSTM model to perform segmentation. Even though it can selectively propagate features to future time steps, their current approach does not bring down the redundancy in any way.

2. PROPOSED APPROACH

Optical flow-based approach DFF is significantly bringing down the computational complexity for segmentation tasks. Instead of optical flow, we propose the use of motion vector. Motion Vector operates on block level (collection of pixels) to estimate movement of block centroids. The motivation behind using motion vector is that there is not enough motion between consecutive frame which warrant motion estimation on a pixel level. Additionally, deep learning architectures are powerful enough to compensate for any loss in motion information due to the use of motion vectors. In subsequent subsections, we explain the preliminaries followed by a detailed description of our approach

2.1 Motion Estimation

We have established earlier that many of the video segmentation techniques perform a significant amount of redundant computations. The reason can be attributed to inefficient exploitation of temporal information in consecutive video frames. Many of the motion estimation techniques can come handy to exploit the temporal redundancy. DFF [20] proposed a flow estimation-based algorithm to perform segmentation. They estimate the optical flow with a neural network and used this information to interpolate the intermediate feature maps. This approach significantly brought down the computational complexity with a little drop in accuracy metric. The question now is: can we further bring down the time complexity during inference operations. We propose to perform segmentation by motion estimation using motion vectors

2.2 Motion Vector

There are two types of redundancies in video coding – spatial redundancy and temporal redundancy. Spatial redundancy is addressed by transmitting significant DCT (Discrete Cosine Transform) coefficients. Temporal redundancy is addressed by motion estimation algorithms. Motion estimation refers to estimation of motion vectors between consecutive

frames. Talking in context of H.264 MPEG video compression standard, videos have three type of frames – I, P and B frames. I frame is least compressible followed by P and B frames. I frame is compressed in itself meaning it does not need any other frame's information for decoding. P frames require previous frame's information for decoding and B frames requires both previous and next frame's information for decoding.

Assuming we are encoding a frame I_t of size HxW. It is divided into blocks of equal dimension NxN pixels represented by $B_t^k(x, y)$ where x, y refers to centroid coordinates of k^{th} block in the frame at time t. Motion estimation finds best match in the reference frame such that the error $B_t^k(x, y) - B_{t-1}^k(x + m_x^k, y + m_y^k)$ is minimized. The vector m_x^k,m_y^k is referred to as the motion vector. In simple words, motion vector refers to the displacement between current frame's block and reference frame's block such that error between them is minimized. Thus, during the video encoding only the error term and estimated motion vectors are used for transmission. Figure 2.1[1] shows an illustration of estimated motion vector and transmitted error term of P-frame.

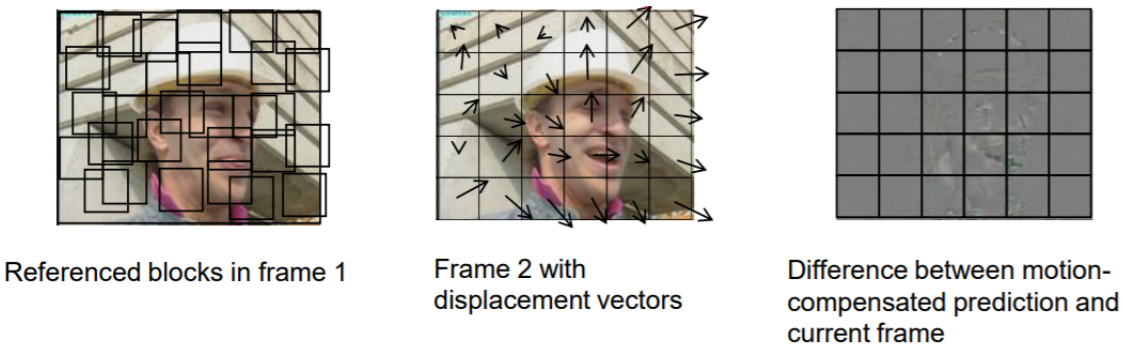


Figure 2.1: Illustration of Motion vector in neighboring frames

There are multiple ways to estimate motion vector, most common of them being block

[1] credits:https://web.stanford.edu/class/ee398a/handouts/lectures/EE398a_MotionEstimation_2012.pdf

matching algorithms. In block-matching a neighborhood is defined around the block to be matched. Figure 2.2[2] shows an illustration of the neighborhood search space defined in the block matching algorithm. Exhaustive search can be a very inefficient time-consuming process. For our experiments, we use Three Step Search [25] approach to estimate the motion vectors.

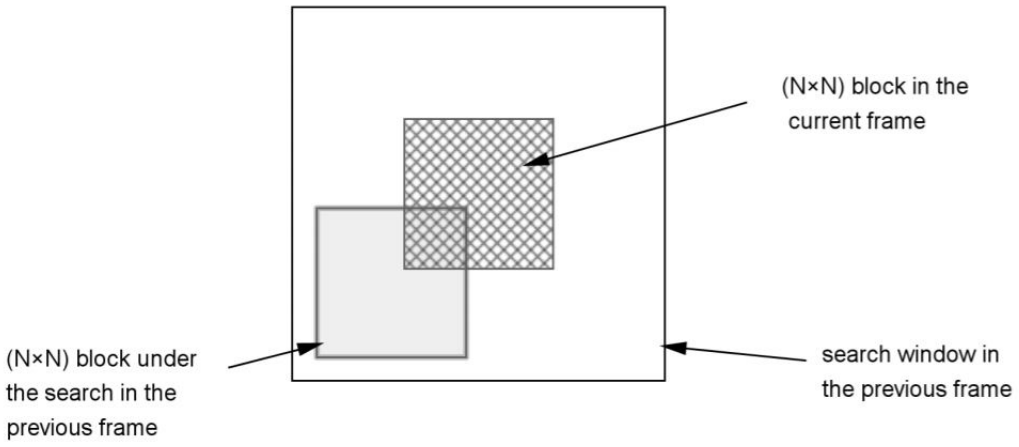


Figure 2.2: Illustration of Block Matching algorithm to estimate Motion Vector

2.3 Algorithm

Now that we have covered all the preliminaries, we will explain our algorithm in detail. Intermediate feature maps in image segmentation network captures high level features, computation of which consumes most of the computational power. We argue that it is not necessary to run the backbone network on neighboring frames. There is not enough motion between neighboring frames which merit computing intermediate feature maps for every frame in the sequence. We believe that even the segmentation using optical flow estimation would still have too much redundancy.

[2]credits: https://courses.engr.illinois.edu/ece417/fa2017/ece417fa2017lecture23.pdf

Motion vector can capture the movement in high level features very well and further bring down the computational complexity. Taking this as inspiration, we propose the use of motion vector for interpolating the intermediate feature maps in neighboring frames. The benefits are two folds:

- Motion vectors are already encoded in the incoming video stream, so we do not need to compute motion vector explicitly unlike optical flow
- If we are performing segmentation on a sequence of video frames (images), motion vector computation would be much cheaper as compared to optical flow computation. The reason being motion vector operates on block level, whereas optical flow operates on pixel level.

Let us assume we have two neighboring frames I_t and I_{t+1} with ground truth segmentation mask as S_t and S_{t+1} and intermediate feature maps as z_t and z_{t+1} respectively. Let I_t be I-frame and I_{t+1} be the P-frame. Let the backbone network be a function $f(.)$ and recognition network be a function $g(.)$. When we run the full segmentation network on I_t, estimated segmentation mask is $\hat{S}_t = g(f(I_t))$. Let the motion vector transformation be a function $h(.)$ (Equation 2.1). Thus, the interpolated feature map would be $\hat{z}_{t+1} = h(f(I_t))$ and estimated segmentation mask would be $\hat{S}_{t+1} = g(\hat{z}_{t+1}) = g(h(f(I_t)))$. As we will explain later, training data is such that ground truth is available after every 20^{th} frame. So, we perform training such that P-frame will have the ground truth. We use weighted cross-entropy loss as the loss function. The loss would be $L(S_{t+1}, \hat{S}_{t+1})$. Since the ground truth is not available for frame I_t, the gradient flows from $\hat{S}_{t+1}$ to $\hat{z}_{t+1}$ and later to z_t through the function $h(.)$ ensuring all the learnable parameters (weights) of the network are updated. It should be noted that the function $h(.)$ is similar to bilinear interpolation and has no learnable parameter.

2.4 Motion Vector Interpolation

We use a block size of 16 with a neighborhood of 16 for block matching. With a frame of dimension 512x512x3, the dimension of motion vector $m = (m_x, m_y)$ would be 32x32.

Since, motion vector captures the movement between the block centroids in neighboring frames, the coordinates would satisfy $m_x, m_y \in [-16, 16]$. We normalize the motion vector by the block size implying $m_x, m_y \in [-1, 1]$

The interpolation is in feature map space whose resolution is 16 times lower than the original input frame size. We also transform the motion vector from relative coordinates to absolute coordinates with respect to upper left corner of the feature map. Effectively, every pixel in feature space represent one block of the original input frame. The procedure for estimating the pixel (x_0, y_0) of $\hat{z}_{t+1}$ with the corresponding motion vector as (m_{x_0}, m_{y_0}) is as follows:

$$
\begin{aligned}
x_1 &= floor(m_{x_0}), y_1 = floor(m_{y_0}) \\
x_2 &= x_1 + 1, y_2 = y_1 + 1 \\
t_1 &= z_t(x_1, y_1)\frac{x_2 - m_{x_0}}{x_2 - x_1} + z_t(x_2, y_1)\frac{m_{x_0} - x_1}{x_2 - x_1} \\
t_2 &= z_t(x_1, y_2)\frac{x_2 - m_{x_0}}{x_2 - x_1} + z_t(x_2, y_2)\frac{m_{x_0} - x_1}{x_2 - x_1}
\end{aligned}
$$

$$
\hat{z}_{t+1}(x_0, y_0) = t_1\frac{y_2 - m_{y_0}}{y_2 - y_1} + t_2\frac{m_{y_0} - y_1}{y_2 - y_1} \tag{2.1}
$$

2.5 Information Loss

It should be noted that in a way we are discarding the frames (P-frames) on which the full segmentation network is not run. The only information we are using is the motion vector which captures motion between blocks of neighboring frames. There can be cases when objects are moving really fast in the neighboring frames. Additionally, low level features also change in consecutive frames. Thus, using only the motion vector information might lead to unsmooth boundaries in the segmentation mask. We use the DeepLabv3+ architecture [16] which introduces a skip connection from initial blocks of the backbone network to end of the

segmentation network. In case of ResNet101 backbone, the skip connection would be from output of block 1 of the backbone. This skip connection would preserve some information of the frames on which full segmentation network is not run leading to smooth boundaries in the final segmentation mask.

2.6 Key Frame Selection

In many of the video encodings such as H.261, H.264, motion vector information is already encoded in the video stream and we would not have to explicitly compute them leading to further drop in computational complexity. There are many available datasets, where only static images of video sequences are available. In those cases, we will need to compute motion vectors which will still be cheaper than optical flow computation. For such cases, we need to decide on what frames (Keyframes) to run the full segmentation network. DFF [20] fixed the duration of keyframes meaning they run the full segmentation network for every Nth frame (N can be any fixed integer). It is an inefficient scheme because the content of the frames in video can change at a varying rate. We propose to design an auxiliary network which can decide on-the-fly if we need to run the full segmentation network or we can just interpolate the intermediate feature maps from the previous frames and perform segmentation. As established in the section 2.5, we are extracting low level features from every frame, we pass them through auxiliary network to learn the change in content between 2 frames.

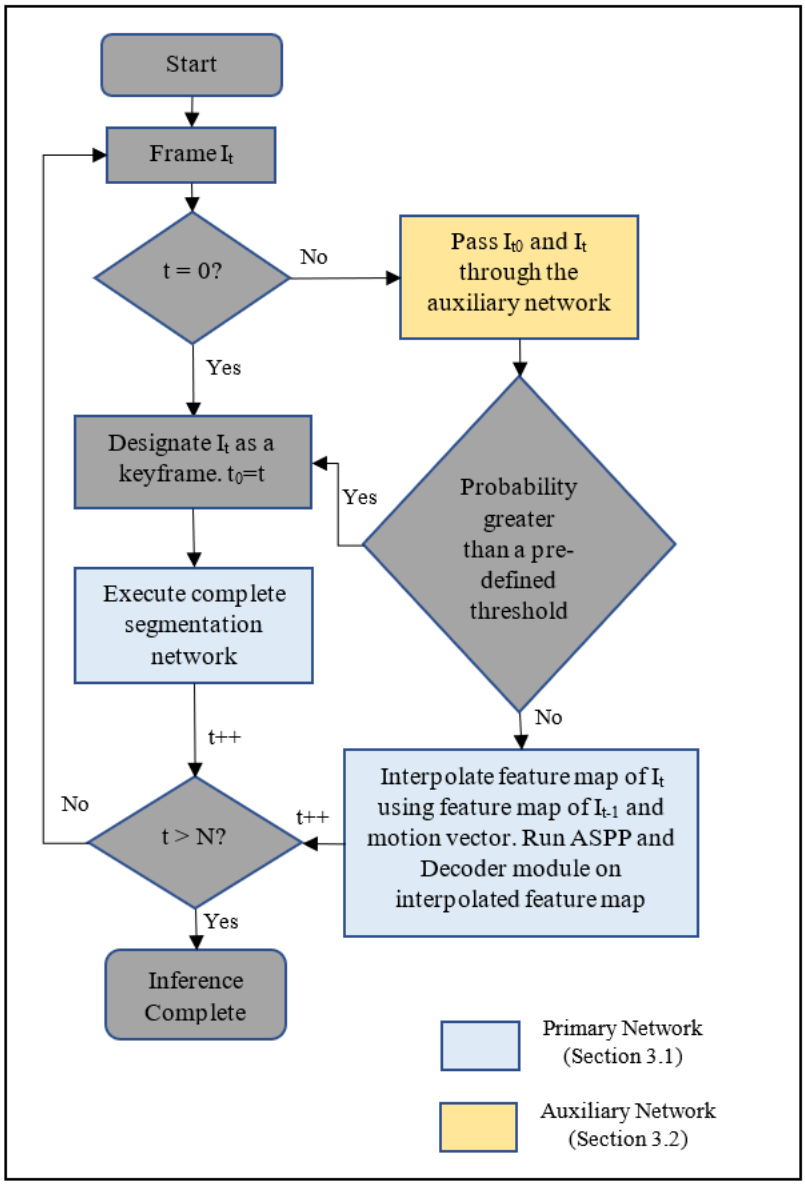


Figure 2.3: Inference Flowchart

2.7 Inference

Figure 2.3 illustrates the inference flowchart used for our proposed algorithm.

- Sequence of video frames coming as input. If it is the first frame, designate it as a keyframe.

- Run full segmentation network on the keyframe and store the intermediate feature map.
- Go to the next frame
- Pass the previous keyframe and the current frame through the auxiliary network which indicates if there is enough motion between them. If probability is below a certain threshold, interpolate previous frame's feature map using motion vector, run rest of the network and go to step 3
- Otherwise designate current frame as a keyframe and go to step 2

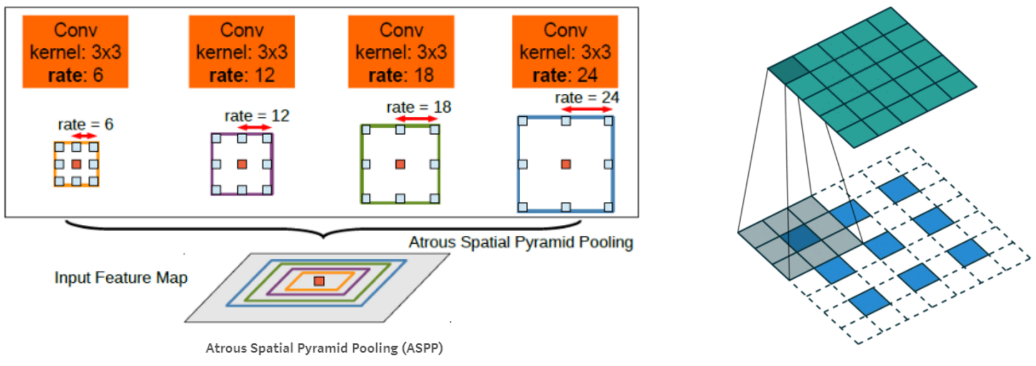
2.8 Extension to Instance Segmentation

The proposed approach can be extended to instance segmentation task as well. We just need to the replace the upsampling module of the network with a Region Proposal Network similar to Faster R-CNN [11] or Mask R-CNN[12]. The feature map interpolation using motion vector will remain exactly the same. Thus, our algorithm is segmentation agnostic.

3.1 Primary Network

We use a modified version of DeepLabv3+ architecture for our experiments which is a benchmark network for semantic understanding tasks. It has 3 modules namely Backbone (Encoder), ASPP and decoder. Backbone network also referred to as encoder uses ResNet101 architecture. It has 4 blocks of multiple residual layers with each block reducing the resolution by 2. Thus, if the input resolution is HxW, the feature map resolution would be $H/16$x$W/16$. Each block is a series of convolution, ReLU, BatchNorm and skip connections. Backbone network is followed by the ASPP module. Referring to Figure 3.1(a) [26], ASPP module has 4 parallel operations performing deconvolution at different dilatation rates (Figure 3.1(b) [27]) and a global averaging pool operation. Output of these 5 operations is stacked together in filter dimension and passed through a convolution operation. The decoder module has a skip connection from first block of the backbone network. It stacks the output of ASPP module and first block of backbone along the filter dimension and passes it through multiple convolution and upsampling operations to output the final segmentation mask.

The output size is HxWxC where C is the number of classes. A softmax operation is



(a) ASPP Module

(b) Dilated Deconvolution

Figure 3.1: Upsampling Module

performed to select the class for every pixel. We use cross entropy loss function for training the network. Since dataset can be biased in terms of proportion of different classes, we train the network with weighted cross entropy loss function as well. We experiment with both the Adam optimizer and Stochastic Gradient Descent (SGD) optimizer. We use weight decay and polynomial learning rate scheduler to adaptively vary the learning rate.

The above explained network architecture is executed only for I-frames. Referring to Figure 3.2, we can see that for P-frames, only block 1 of the backbone architecture is executed. Later high level features of previous frame are interpolated using the motion vector information followed by ASPP and Decoder module. Low level features of P-frames are fused with the Decoder module to incorporate information of P-frame. This operation avoids discarding of full information of P-frames.

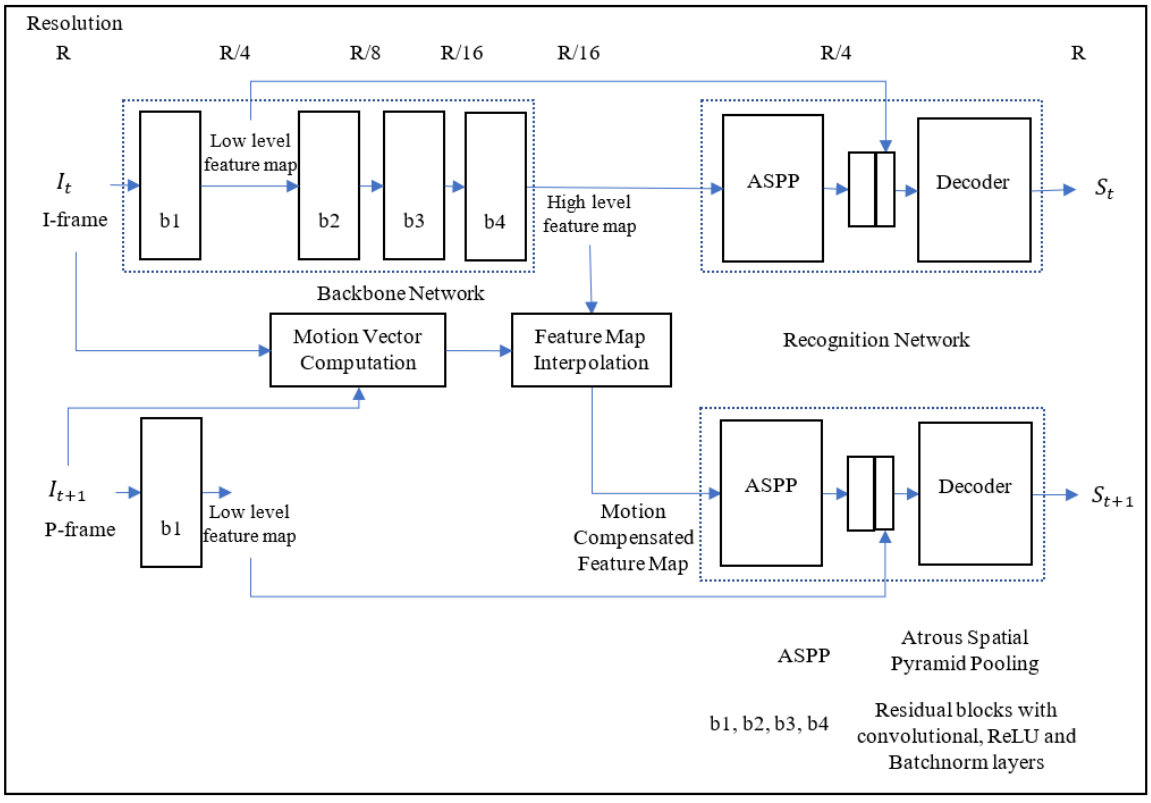


Figure 3.2: Primary Network

3.2 Auxiliary Network

As we discussed earlier that we are using an auxiliary network to adaptively select keyframes. Since every frame goes through first block of ResNet101 irrespective of whether it is a keyframe or not. Output of the first block can capture lower level features such as edges, boundaries etc. We form a pair of neighboring frames and calculate their low-level features and pass them through the auxiliary network. Low-level features are stacked across the filter dimension and passed through a series of convolution, batch norm, ReLU, global average pooling and convolution operation. Finally, softmax of the output is calculated to return the probability of whether the current frame should be a keyframe or not. In terms of convolution operations, we are adding only 2 layers on top of the original primary network. Since the output is also binary in nature, we use cross entropy loss alongside an SGD optimizer.

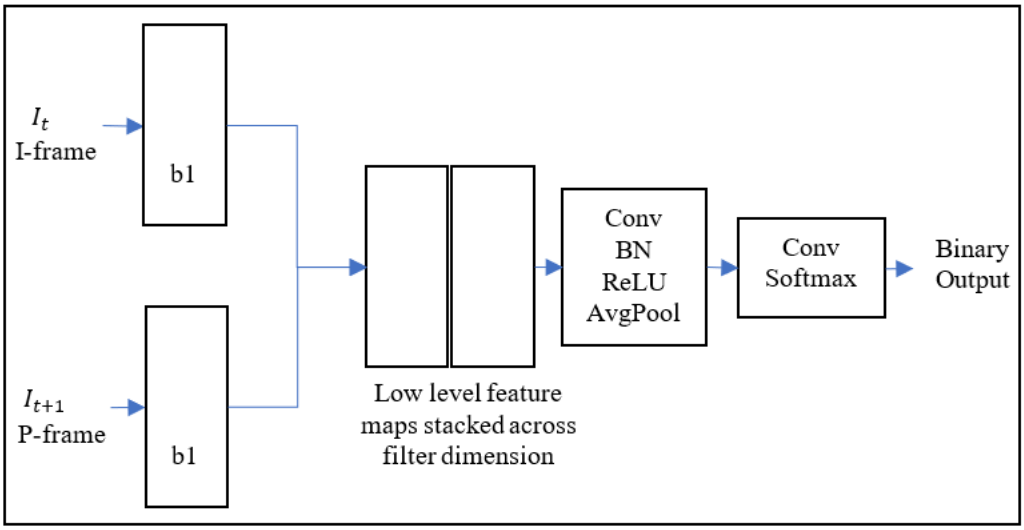


Figure 3.3: Auxiliary Network

3.3 Training Procedure

DeepLabv3+ architecture has a ResNet101 backbone followed by an ASPP (atrous spatial pyramid pooling) module and decoder. ResNet101 is pretrained on ImageNet [2]. We divide training of our Segmentation Network in two phases. In first phase, we train our model on all the standalone images in the dataset. We use Adam optimizer with a learning rate of 0.007 and a decay rate of 0.0001. We use a batch size of 8 with 2 Nvidia P4000 GPU modules. We also use many data augmentation techniques such as random horizontal flipping and rotation of input images, random cropping, gaussian blurring and gaussian normalization. This leads to much more generalization on validation set and reduces overfitting. Additionally, we train the ASPP and decoder modules at 10 times higher learning rate as compared to the backbone module. We use input frame resolution as 512x512.

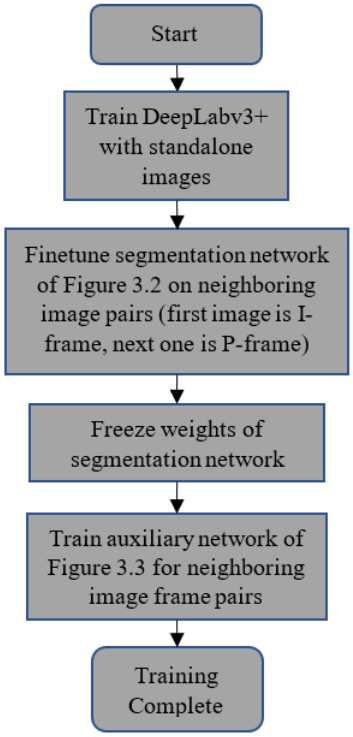


Figure 3.4: Training Flowchart

When our model reaches the highest validation accuracy metric, we move to next phase of training. In second phase, we finetune the network for the video sequence. We use a block size of 16 with a neighborhood search space as 16 too. It should be noted that ground truth is available for every 20^{th} frame only. To overcome this, we train the network for a pair of images. Second image is the one for which the ground truth is available. Effectively, first image in the pair is an I-frame on which full segmentation network is executed and second one is a P-frame for which intermediate feature maps are interpolated and low level feature maps are computed using block 1 of the backbone.

Gradient can flow from feature map of second image to feature map of first image. Motion vector transform has no learnable parameter. Training is performed with a relatively lower learning rate of 0.0002. We could have kept the backbone parameters non-trainable. However, since the feature maps capture high level features and high-level features change very slowly in neighboring frames, keeping backbone parameters might not capture the appropriate change in the low level features.

When the segmentation network is fully trained, we train the auxiliary network to adaptively select the keyframes. We generate the ground truth for every frame using the trained primary segmentation network and create pairs of neighboring frames. Then, we generate binary labels for these image pairs which indicate whether there is enough motion between them or not. We divide the frame in 9 segments and calculate relative motion between corresponding segments of the image pair. We take the maximum deviation and compare it against a pre-defined threshold. The reason for dividing the image in 9 segments is to address the cases where there is enough motion in some parts and no motion in some other parts leading to averaging.

The input to auxiliary network is the low level feature maps of the block 1 of the ResNet101 backbone. We freeze the segmentation network completely to avoid any gradient flow from the auxiliary network. We use cross entropy loss with an SGD optimizer to train the auxiliary network. Figure 3.4 shows flowchart of the complete training procedure.

4. EXPERIMENTS

In this section, we outline details of conducted experiments alongside datasets and obtained results.

4.1 Dataset

There are many available video segmentation datasets. Since it is a semantic segmentation task, we perform all our experiments with Cityscapes dataset [28]. Cityscapes is a semantic understanding dataset consisting of urban scenes as seen from a car driving in various cites of Germany. It has 30 classes such as road, car, person, bridge etc. to name a few. The dataset consists of 2975 training and 500 validation labelled ground truth images. The labelled ground truth corresponds to 20^{th} frame of 30 frame snippets.

4.2 Performance Metrics

4.2.1 Accuracy Metric

For segmentation purposes, there are 2 commonly used metrics for performance (accuracy) namely mean intersection over union $mIoU$ and mean average precision mAP. For semantic segmentation, $mIoU$ metric is used. IoU is defined as the ratio of true positives TP and sum of true positives, false negatives FN and false positives FP. $mIoU$ is the mean value across all classes in the dataset.

$$IoU = 100\frac{TP}{TP + FN + FP} \tag{4.1}$$

4.2.2 Inference Time

We also use inference time as a metric. We assume that during inference, a single frame is processed at any given instance. With this assumption, we pass the frame sequence through the network. We compare our proposed algorithm's inference time against the case when there is no interpolation and complete segmentation network is executed for every frame.

4.3 Results

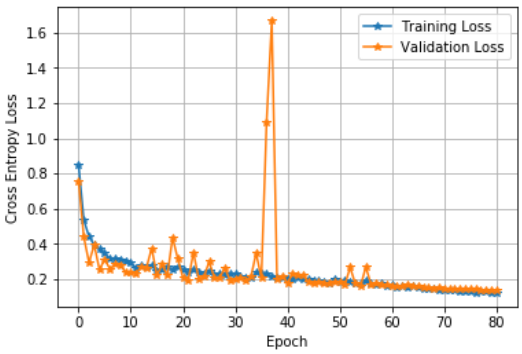
4.3.1 Loss and Accuracy Metric

We compare performance of 4 cases to evaluate our algorithm. First, we consider the DeepLabv3+ architecture's performance on standalone images as our benchmark. Secondly, we evaluate our algorithm which employs motion compensation and adaptive keyframe selection. Referring to Table 4.1, we can observe a drop of 3 points between the benchmark DeepLabV3+ and our algorithm with motion vector interpolation and adaptive keyframe selection. Next, we experiment with separate ASPP and Decoder module for I and P frame. This leads to an improvement of 1 point to 0.655. This makes sense because for P-frame, ASPP and Decoder act on a noisy feature map (interpolated feature map) and using separate modules can capture the noise introduced by motion compensation. Finally, we consider when low level features are not computed for P-frames and not fused during upsampling. This leads to a significant drop of 15 points on the accuracy metric. This points to the observation that motion vector might not be able to capture the complete motion between the neighboring frames and we need low level features to incorporate P-frame information. This also indicates that there is relatively more change in low level features as compared to high level features in consecutive frames.

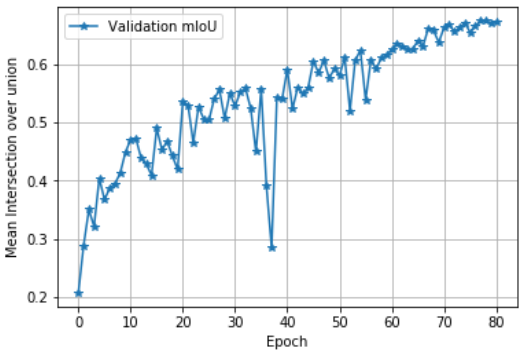
Algorithm	mIoU
DeepLabV3+	0.674
Ours (Motion Compensation and Adaptive Keyframe selection)	**0.646**
Ours (Separate upsampling module for I and P frame)	**0.655**
Ours (No low level features)	0.523

Table 4.1: Accuracy Metric

For DeepLabV3+, even though the highest reported metric (mIoU) is as high as 0.8, we could achieve only 0.675 as the highest metric. The primary reason being that this is highly

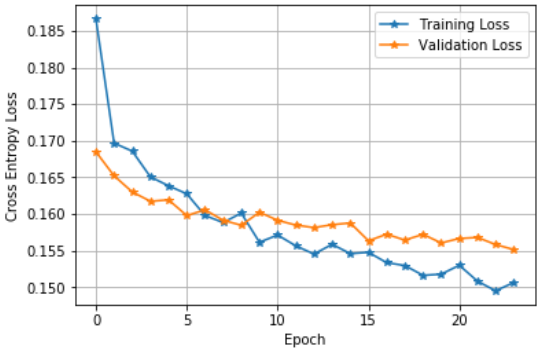


(a) Loss

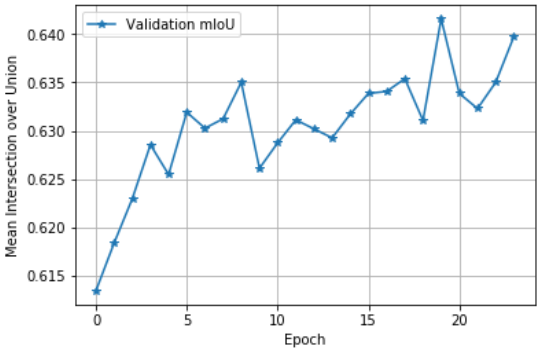


(b) Mean Intersection over Union

Figure 4.1: Training on standalone images



(a) Loss



(b) Mean Intersection over Union

Figure 4.2: Finetuning on video sequence

dependent on hyper-parameter settings and available computational resources. Referring to Figure 4.1, we can observe that loss is decreasing and mIoU is increasing. If we train furthermore, these metrics will continue to improve. However, considering the limitation of resources, we have limited the number of epochs to 80. This constitutes the phase 1 of training. In phase 2, we finetune the learned weights of previous phase at a much lower learning rate for the video sequence. Referring to Figure 4.2, both the loss and mIoU metrics are improving. After finetuning for 24 epochs, we have reached an mIoU metric of 0.646 which is an approximately 3 point drop from our benchmark.

4.3.2 Inference Time Metric

We have already explained our assumptions for comparing the inference time metric. We take a sequence of 90 images for performing the inference. We compare the performance of our proposed algorithm with different keyframe durations against the case when full inference is run on every frame.

Algorithm	KeyFrame Duration	Time (seconds)
Ours	2	**0.66**
Ours	3	**0.53**
Ours	4	**0.49**
Ours	Adaptive	**0.57**
Ours (No low level features)	Adaptive	0.51
Ours	0	0.99
DeepLabV3+	-	0.99

Table 4.2: Inference Time Metric

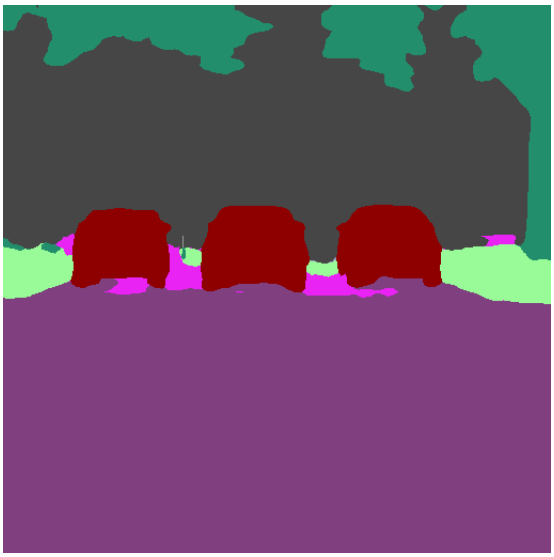
When keyframe duration is 2 (meaning every 3^{rd} frame is a keyframe), we are observing a drop of approximately 34% in the inference time. We are getting a segmentation rate of 136 fps which indicates that we can easily perform inference in real time scenarios. When the keyframe duration goes to 4, inference time comes down as much as 52%. In case of adaptive keyframe selection, inference time drops by 42%. When low level features are not computed for every frame, a further drop in inference time is observed. However, since the accuracy metric is too low (0.523, ref Table 4.1) for this case, reduction in inference time does not mean much.

These results combined with accuracy metric clearly proves the real world scenario feasibility of our proposed algorithm. When motion compensation is not performed and every frame is treated as a keyframe, inference time is similar to the DeepLabV3+ inference case.

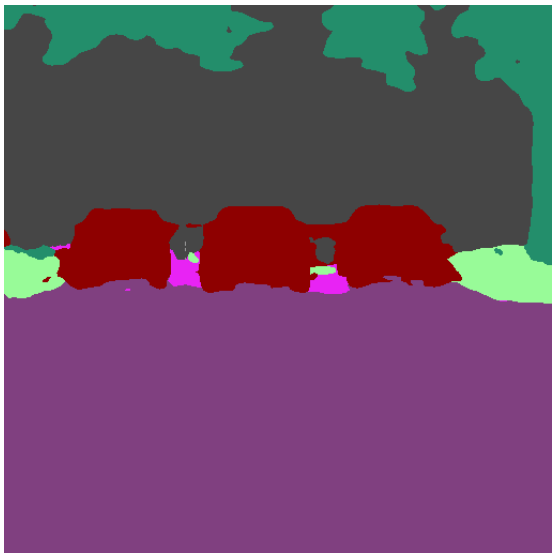
(a) P-frame

(b) Ground Truth

Figure 4.3: Input frame and ground truths



(a) Prediction using full segmentation network inference



(b) Prediction using our proposed approach

Figure 4.4: Predictions

4.3.3 Segmentation visualization

In this section, we show some of the segmented frames using our proposed approach. Figure 4.3(a) and 4.3(b) has input P-frame and ground truth respectively. In one case, we run the full segmentation network on the frame (treating it as a keyframe) shown in Figure 4.4(a). In the other case (which is our proposed algorithm), we interpolate the intermediate map of the previous frame and run rest of the segmentation network. The prediction with the full inference is more closer to the actual ground truth. However, prediction with our proposed algorithm is not too far. As per the mIoU metric, for this particular frame, it lags by approximately 3%.

4.4 Other Experimental Observations

In this section, we lay down some of the experiment we conducted which did not yield expected results. This analysis is important to understand our presented results in a more meaningful context.

- We attempted to enforce temporal consistency on low level features. In consecutive frames, there is relatively more change in low level features as compared to high level features. This hypothesis is backed up by the observations in section 4.3.1. LSTMs can capture long term temporal consistency in time series very well. LSTMs can decide what low features to propagate to next frames for processing. However, the vanilla version of LSTM turned out to be a computationally expensive operation.

- The dataset is relatively biased towards some of the classes. To balance out the bias, we experimented with weighted cross entropy loss. However, it did not yield us desired results.

- Referring to Figure 4.1 (a) and (b), we observe a large fluctuation in loss and accuracy metric. The reason can be attributed to a relatively smaller batch size of 4. This can significantly increase the batch variance and move the gradient in wrong direction

- Referring to Figure 4.5, achieved accuracy metrics are highly dependent on initialization. Even though, we are initializing the convolutional layer weights with the He initialization [4], the rate of convergence still varies significantly for different runs.

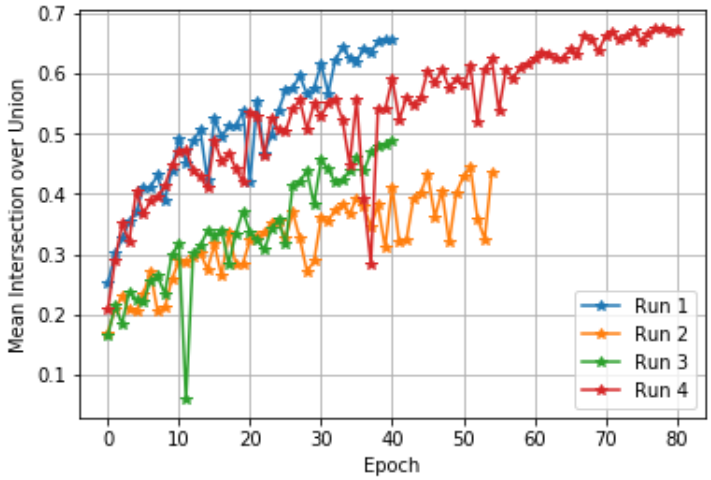


Figure 4.5: Rate of convergence of accuracy metric for different runs

5. SUMMARY AND CONCLUSIONS

We discussed a motion vector based approach for video semantic segmentation. Many of the existing techniques do not exploit the redundancy in the video frame sequences induced by the temporal relations which can lead to these algorithms becoming infeasible for real world use cases. We addressed this redundancy by interpolating intermediate feature maps with the help of motion vectors. This has led to upto 50% drop in the inference time making our algorithm much more feasible for real world production environments. This was at the expense of a small drop of 2-3% in the accuracy metric. In terms of fps, we can process upto 136 frames per second which is much higher than any of the streaming rates proving our algorithm should not suffer from any kind of bottlenecks during real time inference. Additionally, our algorithm can be extended to instance segmentation cases as well by replacing the upsampling module with a region proposal network.

We also proposed an auxiliary network which can decide on the fly whether we need to run the full segmentation on a frame. This network takes away the cases where there is hard scheduling of keyframes. Instead it adaptively decides whether a frame will be a keyframe or not.

5.1 Immediate Extensions

Our work has some immediate extensions which we enlist in this section.

- We have assumed till now that motion vector is available in video encoding. In cases, when it is not available, we are using Three Step Search (TSS) for estimating motion vector. Implementing TSS on CUDA can be a challenge and might lead to bottlenecks. Instead, we can design a neural network architecture to estimate it for the cases when only the individual frames are available. We actually designed a network on lines of Flownet [21] for this task. However, the learned model has relatively high mean square error in motion vector estimation. We believe that this error can brought down

by tweaking network and hyperparameters.

- We are using bilinear interpolation for estimating feature map of non keyframes. Instead we can use spatial transformer networks [29] for this task, which is much more generic and fast for warping operations.

- We have exclusively used DeepLabv3+ architecture for the semantic segmentation task. However, we can experiment with other architectures available in literature to bring down the computational complexity with similar accuracy metrics.

5.2 Future Work

We have shown that motion vectors can significantly bring down the computational complexity with a reasonable drop in the accuracy metric. This property can be utilized for many other video related applications. One such area would be activity recognition in videos which is an active area of research in computer vision now. Many of the prevalent techniques do not exploit the temporal redundancy efficiently and rely on LSTM based approaches for encoding the video sequence into feature maps. Some approaches use 3D convolutional for collection of video frames to estimate the feature maps. There has been some work on efficient frame selection which can adequately capture the activity in the video sequence. We believe that frame selection can be performed in the feature space whereas feature maps can be computed using our motion vector-based interpolation.

Additionally, we wish to work on computationally efficient ways to enforce temporal consistency on low level features. Using temporal relations in these features should further boost the accuracy score and enforce the temporal consistency even more strongly. In terms of architecture, we will explore LSTMs or Recurrent Nets.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[3] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (Y. W. Teh and M. Titterington, eds.), vol. 9 of *Proceedings of Machine Learning Research*, (Chia Laguna Resort, Sardinia, Italy), pp. 249–256, PMLR, 13–15 May 2010.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, (Washington, DC, USA), pp. 1026–1034, IEEE Computer Society, 2015.

[5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *arXiv prepring arXiv:1506.01497*, 2015.

[7] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[8] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '14, (Washington, DC, USA), pp. 580–587, IEEE Computer Society, 2014.

[9] J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders, "Selective search for object recognition," *International Journal of Computer Vision*, 2013.

[10] R. Girshick, "Fast r-cnn," in *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, (Washington, DC, USA), pp. 1440–1448, IEEE Computer Society, 2015.

[11] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, (Cambridge, MA, USA), pp. 91–99, MIT Press, 2015.

[12] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proceedings of the International Conference on Computer Vision (ICCV)*, 2017.

[13] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, pp. 640–651, Apr. 2017.

[14] V. Badrinarayanan, A. Handa, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for robust semantic pixel-wise labelling," *arXiv preprint arXiv:1505.07293*, 2015.

[15] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *CoRR*, vol. abs/1511.07122, 2015.

[16] L. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," *CoRR*, vol. abs/1802.02611, 2018.

[17] Y.-H. Tsai, M.-H. Yang, and M. J. Black, "Video segmentation via object flow," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[18] F. Perazzi, A. Khoreva, R. Benenson, B. Schiele, and A.Sorkine-Hornung, "Learning video object segmentation from static images," in *Computer Vision and Pattern Recognition*, 2017.

[19] S. Caelles, K.-K. Maninis, J. Pont-Tuset, L. Leal-Taixé, D. Cremers, and L. Van Gool, "One-shot video object segmentation," in *Computer Vision and Pattern Recognition (CVPR)*, 2017.

[20] X. Zhu, Y. Xiong, J. Dai, L. Yuan, and Y. Wei, "Deep feature flow for video recognition," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4141–4150, 2017.

[21] P. Fischer, A. Dosovitskiy, E. Ilg, P. Häusser, C. Hazirbas, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox, "Flownet: Learning optical flow with convolutional networks," *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 2758–2766, 2015.

[22] S. Jain and J. E. Gonzalez, "Fast semantic segmentation on video using motion vector-based feature interpolation," *CoRR*, vol. abs/1803.07742, 2018.

[23] L. Yang, Y. Wang, X. Xiong, J. Yang, and A. K. Katsaggelos, "Efficient video object segmentation via network modulation," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[24] N. Xu, L. Yang, Y. Fan, J. Yang, D. Yue, Y. Liang, B. L. Price, S. Cohen, and T. S. Huang, "Youtube-vos: Sequence-to-sequence video object segmentation," *CoRR*, vol. abs/1809.00461, 2018.

[25] R. Li, B. Zeng, and M. L. Liou, "A new three-step search algorithm for block motion estimation," *IEEE Trans. Cir. and Sys. for Video Technol.*, vol. 4, pp. 438–442, Aug. 1994.

[26] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *CoRR*, vol. abs/1606.00915, 2016.

[27] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," 2016. cite arxiv:1603.07285.

[28] M. Cordts, M. O. S. Ramos, M. Enzweiler, R. Benenson, U. Franke, S. Roth, B. Schiele, D. A. R, T. Darmstadt, M. Informatics, and T. Dresden, "The cityscapes dataset."

[29] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, "Spatial transformer networks," *CoRR*, vol. abs/1506.02025, 2015.