UKRAINIAN CATHOLIC UNIVERSITY

BACHELOR THESIS

Lane segmentation for advanced driver-assistance systems

Author:
Maryana TEMNYK

Supervisor:
Orest VARHA

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Department of Computer Sciences
Faculty of Applied Sciences

Lviv 2020

Declaration of Authorship

I, Maryana TEMNYK, declare that this thesis titled, "Lane segmentation for advanced driver-assistance systems" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"It is our choices, Harry, that show what we truly are, far more than our abilities."

Albus Dumbledore

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Bachelor of Science

Lane segmentation for advanced driver-assistance systems

by Maryana TEMNYK

Abstract

Lane detection is an image analysis problem of detecting lane markings on the road. Recently it became one of the crucial parts of most Advanced diver-assistance systems. With the current progress in deep learning, plenty of new methods have been developed for lane detection. However, even novel powerful CNN architectures are struggling to give a satisfactory result along with acceptable performance. Recent studies show that developing a fast, robust and accurate model for real-time lane detection is a hard task due to external factors such as weather conditions, lighting, traffic, complex marking shapes, thin or invisible lanes on rough and hilly roads, etc.

The main focus of this thesis is to investigate, compare, and develop lane detection methods that can be used in real-time ADAS applications. Current demonstration of the work presented in this thesis can be found here.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Orest VARHA, who guided me through this thesis with patience and great advice and to my supportive friends and family, who relieved any pressure through all these years. Also, I want to thank my first ever machine learning team for valuable comments and interesting discussions.

Finally, I want to thank the Ukrainian Catholic University and the Faculty of Applied Sciences for a wonderful Computer Science program that I am proud to be a part of...

Contents

List of Figures

List of Tables

List of Abbreviations

ADAS	**A**dvanced **D**river-**A**ssistance **S**ytems
DCNN	**D**eep **C**onvolutional **N**eural **N**etwork
DL	**D**eep **L**earning
ANN	**A**rtificial **N**eural **N**etwork
CNN	**C**onvolutional **N**eural **N**etwork
FC layer	**F**ully **C**onnected layer
SAD	**S**elf- **A**ttention **D**istillation
BCE	**B**alanced **C**ross-**e**ntropy
CV	**C**omputer **V**ision

Chapter 1

Introduction

Lane detection is a problem of finding lane markup on the road image. Autonomous driving is impossible without lane detection. It is widely applied in various advanced driver-assistance systems(ADAS) and can be used in other fields such as sports and robotics. In particular, such applications as parking systems, sports analytics systems, cars/robots/drones navigation, adaptive cruise control, lane departure warning, and lane-keeping assist systems, etc. can benefit from an effective solution to the lane detection problem.

The nature of the problem itself introduces a number of issues. First of all, lanes are tiny and thin objects, and they occupy 2-3% of an image, which makes them hard for a model to learn. Second of all, lanes have a very basic shape - line, which can be present anywhere on a real-life image, for instance, on buildings, vans, traffic lights, etc. This simplicity of the shape forces the model to confuse lane markings with sunlight reflections, pipes, or other line-like objects.

In computer vision, lane detection is mostly formulated as a semantic segmentation problem. Semantic segmentation is a task of classifying pixels of an image. Nowadays, this task is mainly solved with deep convolutional neural networks(DCNN) that outperform previous approaches in terms of accuracy and robustness. However, DCNN-based methods require a significant amount of computational power, which is only possible using special processors with hardware accelerators(e.g., GPU). Such automotive processors are expensive, and they are usually busy running multiple other algorithms(e.g., obstacle detection). Thus it is important to research how to design a lane detection algorithm that is both accurate and efficient in terms of computation.

1.1 Goal setting

As the main application for lane detection, we chose to implement a lane departure warning system, detailed in Chapter 5. Therefore the main task for the thesis is to train a robust, accurate, and fast model for real-time usage in the system mentioned above. Another challenge is that we want to make our model robust against missing or damaged road lines and generalize in this case from the surrounding scene. As in real life, not all roads are marked, and sometimes drivers understand where the marking should be by tire traces or by visually analyzing a context of a driving scene.

1.2 Method proposal

- Combine existing lightweight CNN architectures, and segmentation pipelines with self-attention based training to achieve the needed speed and accuracy, as detailed in Chapter 4

- Force the model to learn where the markings should be, by removing or blurring markup on training images, also detailed in Chapter 4. To our knowledge, no one tried this yet.

1.3 Thesis structure

- **Chapter 2. Background theory**

  This chapter contains background information on key machine learning concepts such as neural networks and convolutional neural networks.

- **Chapter 3. Related work**

  In this chapter, a short overview of recently introduced semantic segmentation methods based on deep learning can be found. Also, here we examine the research around the lane detection problem before and after deep learning.

- **Chapter 4. Methodology**

  In this chapter, you can find a thorough description of the proposed method.

- **Chapter 5. Implementation and application**

  In this chapter, all implementation details, frameworks, and hardware are outlined. Also, a lane departure warning system and the role of the method in it are described.

- **Chapter 7. Experiments and Results**

  This chapter is a collection of experiment descriptions and an analysis of their results.

- **Chapter 8. Summary**

  This chapter concludes the thesis and contains a summary of the work done and a future work outline.

Chapter 2

Background information

The performance of information processing systems changed drastically, with deep learning gaining more and more popularity. Artificial Neural networks showed impressive results in solving a wide range of problems, and their future potential is yet to discover. One of the most impacted fields by DL is image analysis, where Convolutional Neural Networks(CNNs) caused tremendous progress.

2.1 Artificial Neural Networks

The idea behind Artificial Neural Networks was to reconstruct the work of the animal neural tissue. Similarly to natural nervous tissue, ANN consists of mathematical representations of neurons, the biological cells, arranged in layers, see figure 2.1. The simplest form of an artificial neuron, known as a multi-layer perceptron, receives multiple inputs(signals from other neurons), calculates a weighted sum of inputs, adds bias, and applies the activation function, which produces neuron output. The neuron output value is propagated to the input of the neuron in the next layer. An activation function is usually non-linear, monotonically increasing, and differentiable. The neuron learns how to process input signals by changing the weights. The right configuration of weights in each neuron is what we try to achieve when training the network. Neurons can be arranged in different types of layers, the simplest of which is a dense layer, also known as a fully connected(FC) layer, where neurons and connections form a complete graph. A typical simple network has an input layer, hidden layers, and an output layer. A deep neural network is a network that has many hidden layers in it. An area of research on deep neural networks is called Deep Learning.

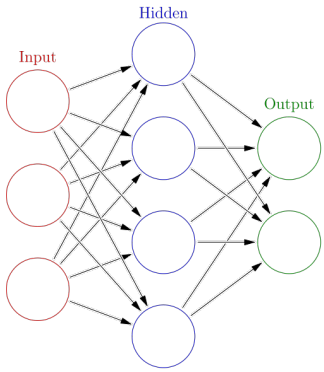


FIGURE 2.1: An illustration of a dense neural network with input, output, and a hidden layer [Source]

So how do we train a network? The training process is carried out using an optimization method called gradient descent. First of all, we need some function that evaluates how good is the output of the model in comparison to the label. This type of function is called loss function or sometimes error function. We determine the output of the network by computing an output of each neuron starting from the input layer and on, creating a chain of calculations. Finally, we compute the value of loss function. This process is called forward propagation. When we know the value of the loss function, we can start computing how we should update the weights. Since the loss function is a combination of previous functions, all of which are differentiable, we can compute gradients and use the chain rule to determine how much and in which direction we should change our inner functions, and therefore weights of the last layer. In this manner, we can calculate gradients for every layer, moving from the output to input determining how to change each weight of each neuron. This process is called backpropagation. By propagating the network forward and backward many times, we update our weights so that the result is more and more similar to the label according to the loss function.

2.2 Convolutional Neural Networks

The simple dense(fully connected) neural network has a serious limitation in image analysis - it heavily depends on the placement of an object. In other words, it will not find an object on an image if it changes its location. It also has other drawbacks, such as a huge number of parameters, which makes training and inference slow, and causes the risk of overfitting. Convolutional neural networks, introduced by Lecun et al. [20], are invariant to the placement of an object. They contain convolutional layers that try to find some features in the input matrix using filters - smaller matrices of weights (Figure 2.1). The input image is multiplied by the filter in a sequential manner in order to create a feature map. Each neuron in a convolutional layer tries to learn weights for a filter, therefore finding features on an image. The next convolutional layer finds features in the output of the previous one, thereby building a network.

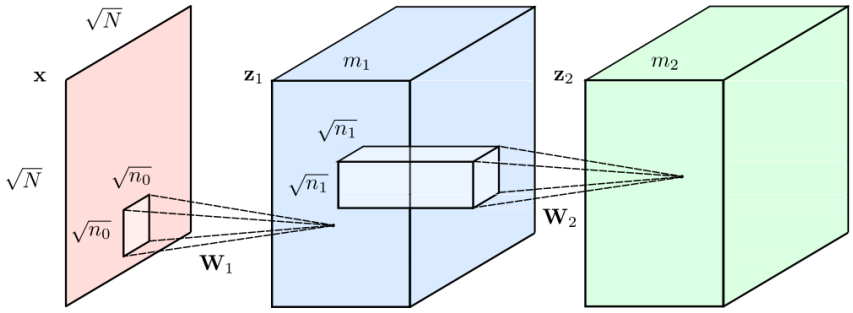


FIGURE 2.2: Convolutional layers [Source]

In the simplest case, CNNs consist of three kinds of layers - mentioned above convolutional, pooling, and dense, described in section 2.1. Pooling layers have no learnable parameters in them. They provide invariance to the object placement by reducing the size of the feature map and creating the so-called perception area for each neuron. The bigger the perception area also called the receptive field, the better the image analysis that the network performs. By reducing the size of the feature map, we also reduce the number of parameters and computation in the network. Dense layers are usually used as the last layers of the network.

Over the last years, the CNNs found their application practically in all computer vision tasks, superseding most of the existing approaches. Starting from AlexNet's [17] success on ImageNet dataset [12] numerous novel architectures, like Vgg [31], Resnet [14], Inception [32], Xception [11], ResNeXt and others were constructed for image classification task. From there, the existing architectures were modified, and new state of the art architectures were built to solve other image analysis problems, object detection and image segmentation in particular.

Chapter 3

Related works

3.1 Image Segmentation

3.1.1 Overview

An image segmentation task is a problem of assigning a class label to each pixel of an image. Before DL, the majority of approaches to the segmentation problem usually were based on handcrafted features and custom rules e.g., Conditional Random Fields, 2001 [19]. These methods were neither robust nor accurate enough to produce plausible results.

One of the first neural network-based approaches to image segmentation, FCN, was introduced by Lang et al. [23] in 2014. Authors proposed to remove fully connected layers from an existing VGG [31], GoogLeNet [33], AlexNet [17] architectures and place convolutional blocks instead to obtain a small segmentation map. To upsample the output of the last layers, the authors used bilinear interpolation and backward strided convolution, also known as deconvolution.

A year later SegNet [3] was presented with its encoder-decoder architecture. Upsampling is performed using previously-stored max-pooling indices. This process is known as unpooling. The decoder block is followed by a final pixel-wise softmax classification layer. In 2016 transpose convolutions were put instead of unpooling operations.

One of the most notable architectures was UNet by Ronneberger et al. [29] that used skip-connections to pass feature map tensors from encoder blocks to respective decoder blocks. Corresponding feature maps were concatenated in the decoder, thus keeping information about small objects when downsampling. Transpose convolution is used for upsampling along with bilinear interpolation for image smoothing.

Later numerous attempts were made to create an accurate, multiscale model for multiclass segmentation, with a large number of classes. Bright examples: FRRN [28] PSPNet [39], Deeplab(s) [7] [9] [8], RefineNet [21].

3.1.2 Light architectures for segmentation

Most of the aforementioned approaches focus on accuracy and robustness letting their models be as heavy as needed. However real-time applications demand lightweight solutions for segmentation problem and lane detection is one of them. Efficient neural network (Enet) [27], used in this thesis is a deep network specifically designed for real-time usage. Its structure, shown in Figure 3.1 is heavily influenced by that of ResNets [14] and optimized for faster inference and high accuracy.

Among other relatively lightweight architectures are LinkNet [6] a fully convolutional architecture with around 11.5 million parameters, that works well with a

Name	Type	Output size
initial		$16 \times 256 \times 256$
bottleneck1.0	downsampling	$64 \times 128 \times 128$
$4\times$ bottleneck1.x		$64 \times 128 \times 128$
bottleneck2.0	downsampling	$128 \times 64 \times 64$
bottleneck2.1		$128 \times 64 \times 64$
bottleneck2.2	dilated 2	$128 \times 64 \times 64$
bottleneck2.3	asymmetric 5	$128 \times 64 \times 64$
bottleneck2.4	dilated 4	$128 \times 64 \times 64$
bottleneck2.5		$128 \times 64 \times 64$
bottleneck2.6	dilated 8	$128 \times 64 \times 64$
bottleneck2.7	asymmetric 5	$128 \times 64 \times 64$
bottleneck2.8	dilated 16	$128 \times 64 \times 64$
Repeat section 2, without bottleneck2.0		
bottleneck4.0	upsampling	$64 \times 128 \times 128$
bottleneck4.1		$64 \times 128 \times 128$
bottleneck4.2		$64 \times 128 \times 128$
bottleneck5.0	upsampling	$16 \times 256 \times 256$
bottleneck5.1		$16 \times 256 \times 256$
fullconv		$C \times 512 \times 512$

FIGURE 3.1: Enet architecture

multiclass problem and ICnet [38] that efficiently combines and fuses information from different scales.

3.2 Lane detection

3.2.1 Earlier approaches

Before deep learning, lane detection was done mostly by using handcrafted features and geometric parameters to obtain lane segments [18]. These methods require complex image preprocessing and are not robust and accurate enough to give good results in complicated driving scenarios. As the main algorithm, different variations of Hough transform [13] were used, and for preprocessing such methods as Sobel operator[16], Inverse Perspective Mapping (also known as Bird's eye view), Adaptive Thresholding , Canny edge detection [5] were popular.

3.2.2 DCNN approaches

With the recent growth in deep learning, new architectures of CNNs were introduced in image classification, object detection, and image segmentation. Therefore a lot of novel approaches to this task were suggested in the last years.

Xingang Pan et al. [26] proposed to learn the shapes of the segmented lanes better by introducing Spatial CNN. The network can learn spatial relationships between pixels by using not just traditional layer by layer convolutions but also slice by slice convolutions within the feature maps. Qin Zou et al. [40] proposed to combine CNNs with Recurrent neural networks to make use of information about lanes in consecutive video frames as opposed to performing segmentation on a single image that has no information about continuousness of the lane. The approaches described above concentrate mostly on the accuracy and robustness of the models without taking into account the speed and weight of the models.

Several approaches take into account the efficiency of the proposed methods. For instance Davy Neven et al. [25] proposed an end to end pipeline for real-time lane detection with no additional postprocessing needed. He et al. combine binary segmentation with embeddings clustering in a network called LaneNet to achieve accurate enough semantic segmentation without using heavy networks Figure 3.2. As for post-processing he et al. proposed to use a light curve fitting network on the

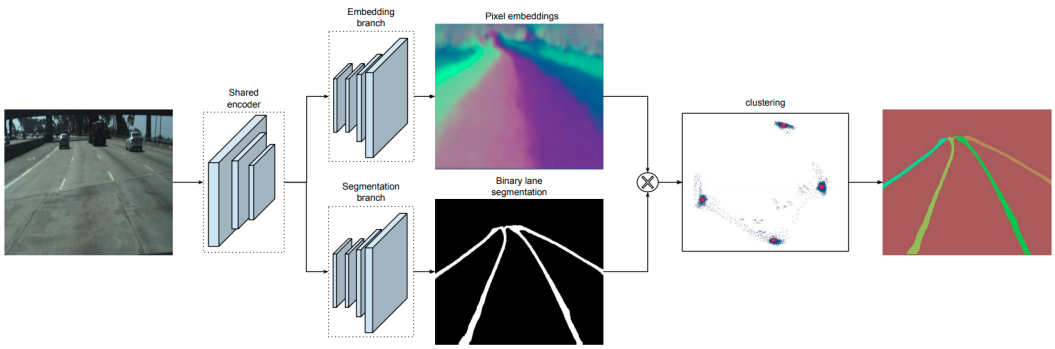


Fig. 2. LaneNet architecture. It consists of two branches. The segmentation branch (bottom) is trained to produce a binary lane mask. The embedding branch (top) generates an N-dimensional embedding per lane pixel, so that embeddings from the same lane are close together and those from different lanes are far in the manifold. For simplicity we show a 2-dimensional embedding per pixel, which is visualized both as a color map (all pixels) and as points (only lane pixels) in a xy grid. After masking out the background pixels using the binary segmentation map from the segmentation branch, the lane embeddings (blue dots) are clustered together and assigned to their cluster centers (red dots).

FIGURE 3.2: Lanenet architecture [Source]

segmentation result. This network, H-net, learns a perspective transformation in a way that benefits the curve fitting. All in all the proposed pipeline scheme is the following: Figure 3.3

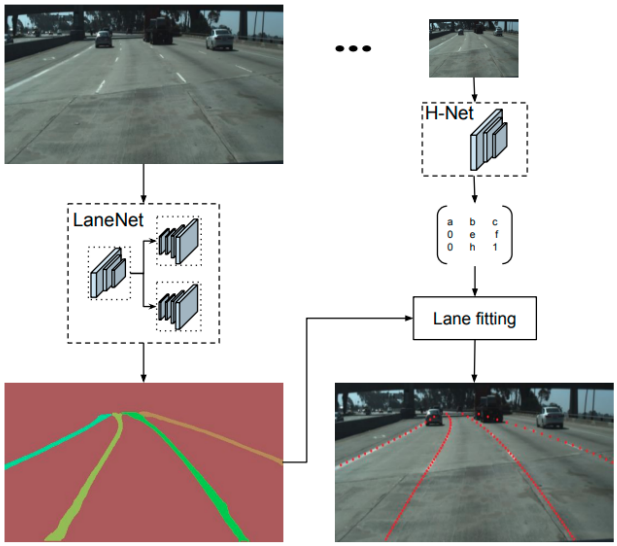


Fig. 1. System overview. Given an input image, LaneNet outputs a lane instance map, by labeling each lane pixel with a lane id. Next, the lane pixels are transformed using the transformation matrix, outputted by H-Net which learns a perspective transformation conditioned on the input image. For each lane a 3rd order polynomial is fitted and the lanes are reprojected onto the image.

FIGURE 3.3: End-to-end lane detection pipeline [Source]

Yuenan Hou et al. [15] proposed a new method of training the lightweight models for lane detection with so-called self-attention distillation, which forces the model to capture useful information from itself. Authors chose Enet architecture to transfer attention between its blocks. This method does not require any modifications to the model on the inference stage. Self-attention is only used while training, which makes the technique quite appealing to try.

As a result of related works study, the two aforementioned papers were chosen as a basis for the proposed solution. We propose to develop a binary segmentation model, Enet, trained with self-attention distillation on variously augmented data. Later this model can be used as a binary branch of LaneNet to achieve semantic segmentation, and the curve fitting network H-net can be applied afterward as a form of postprocessing. In this thesis, we will focus on the development and research of the binary branch for this approach. The details can be found in the next chapter.

Chapter 4

Methodology

4.1 Data

The datasets for segmentation usually consist of an image and a binary mask pairs. As for lane detection problems, masks are usually generated with polylines from JSON or txt files with coordinates of the lanes, where vertical coordinates are joint for the whole dataset to ease the work of the annotators.

4.1.1 Sources

With the recent growth of interest in applying machine learning to ADAS problems, a few datasets for lane detection were collected. In our experiments, we are interested in detecting borders between lanes more than in the accurate placement of each road marking like arrows or crosswalks, which determined the choice of our datasets. The two most popular datasets were used - Tusimple [35] dataset and CuLane [36] dataset.

The Tusimple dataset was created for lane detection as well as velocity estimation. The lane detection part of the dataset was generated from 7 000 road video clips of 20 frames each. The train/validation part includes 3626 video clips where one of the 20 frames is annotated, and a test part has 362 videoclips with no labels available. Collected images usually feature relatively fine weather conditions with sunny weather and excellent visibility, various daytime, traffic conditions, and a different number of lanes. The Tusimple dataset is relatively small and easy for a model to learn, but also has all basic features for lane detection in it, which makes it really convenient to try different methods on it.

The CULane is a bit newer and a much bigger dataset for lane detection. It was extracted by six drivers on different vehicles in Beijing. The dataset consists of 88 880 and 9 675 labeled training and validation images, respectively, and 34 680 unlabeled test images. The CULane dataset has plenty of data with complicated real-life conditions on the road like bad lighting, traffic, snowy-rainy weather, sun reflections on the window, etc. In cases where the markings are unseen, the annotations were made as if the lane border is visible, with the intention for algorithms to be able to distinguish between lanes even if the border is occluded by a road object. During preprocessing, we found out that the data is not as clean as we hoped, so after preparation, we got 78 421 images in the training set and 8 936 in the validation set.

4.1.2 Augmentation

Deep neural networks require a considerable amount of representative data to achieve satisfactory results. To increase data volume and introduce more real-life features to

the existing dataset, researchers usually use image augmentation. Image augmentation is a set of processing techniques for existing images that makes some features more visible or introduces new desired features to a picture.

FIGURE 4.1: Augmentation results on TuSimple dataset

Commonly used augmentation

In this work, we applied some commonly used methods for augmentation of real-life photos. Due to the specifics of our task, we cannot use destructive augmentation on our images. Therefore such operations as rotation, flip, perspective transform were applied to imitate possible camera incline, and ISO noise for camera noise imitation is used. To accomplish such augmentation, we used python albumentations [1] library.

Weather conditions imitation

Most cases of poor road visibility are caused by unwanted weather conditions. Hence it would be useful to be able to imitate common weather-related obstacles when augmenting the data. After analyzing the driving process, we came to the conclusion that the most common obstacles for lane visibility on a camera are solar flares, fogs, and shadows on the road. To create such augmentation, we used the aforementioned albumentations library, which in turn uses a custom library for road images augmentation called AutomoId [30]. This library also contains some augmentation

techniques that can be used to imitate other weather conditions on the road. The examples of the augmented data used in our experiments are shown in Figure 4.1.

Lane markings deletion

When we investigated the process of driving, we noticed that the substantial amount of the roads in our country is not marked or marked a long time ago, and at present, road lines are not visible. Thus most of the available training data is not representative in this case. So we came up with one more augmentation method - deletion or blurring the existing lane markings on the image. This problem is essentially an image inpainting task - we need to make regions with lanes on them similar to the rest of the road, and we have a binary mask that indicates the lane placement. This is solved with Telea [34] inpainting method. The implementation of this method in OpenCV [4] library was used on data preparation step to obtain results shown in Figure 4.2.

FIGURE 4.2: Lane deletion example

4.2 Architectures

4.2.1 Classical CV approach

When starting any work with deep learning, the first thing that should be done is checking non-deep learning methods to know for sure that a deep learning approach

is worth starting. As mentioned in the Related works, early lane detection was done with classical computer vision methods, which require a significant amount of pre- and postprocessing but are used in real vehicles. The simplest pipeline of classical CV methods for lane detection is the following. A picture is filtered with yellow and white(colors of road markup) color filters, after that, the region of interest is defined, in our case, it is a lower part of the picture, where the road is located, other parts of the image are of no use. On a filtered image, Canny edge detection is applied to obtain an edge mask. Supposedly on this stage, the only straight lines on the mask are lane markings, in practice though it rarely happens. To find straight lines on this mask Hough lines algorithm is used, and the result is obtained. Hough lines algorithm usually finds a large number of lines. Thus the final stage should perform line clustering, outlier removal, etc.

4.2.2 Unet

Unet [29] is a known architecture that became an extremely popular solution in the segmentation domain, and as such, was chosen as a first architecture to try on our data for experimental purposes. Firstly introduced for bioimages segmentation, this fully convolutional network has the following architecture 4.3.

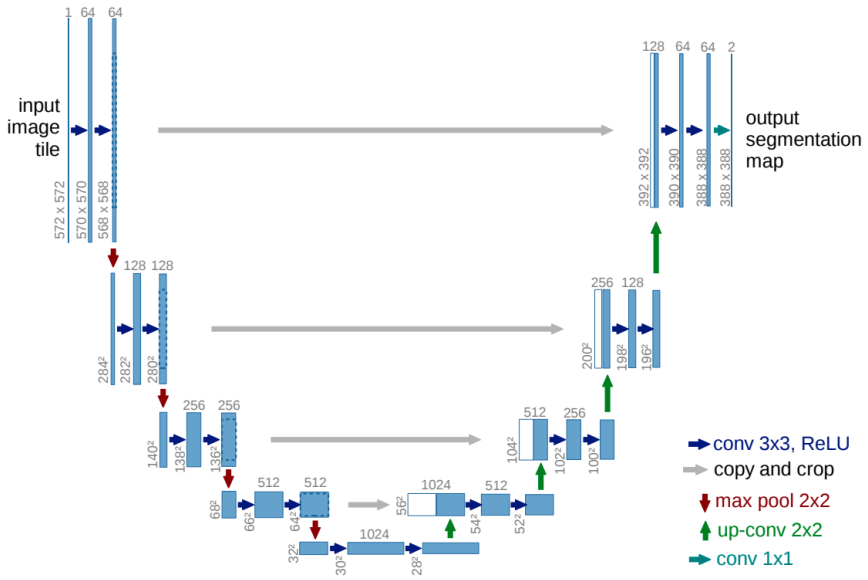


FIGURE 4.3: Unet architecture. From [29]

The blocks in encoder consist of pooling and convolutional layers, in decoder - deconvolution upsampling with bilinear interpolation for smoothing and convolution. The main aim was to prevent the network from losing information about small objects while pooling, by introducing skip-connections. Through those connections, feature maps from encoder blocks to decoder blocks of corresponding resolution are passed. In the encoder, the feature maps are concatenated - white and blue tensors in the decoder in Figure 4.3.

While Unet usually produces good results, the model drawback is that it contains too many parameters, requires significant computation resources, thus not suitable for real-time applications.

4.2.3 Enet

When it comes to lightweight solutions in segmentation Enet [27] architecture is definitely a choice. The architecture of the whole network can be seen in Figure 3.1. This encoder-decoder network consists of the following blocks: Figure 4.4. The

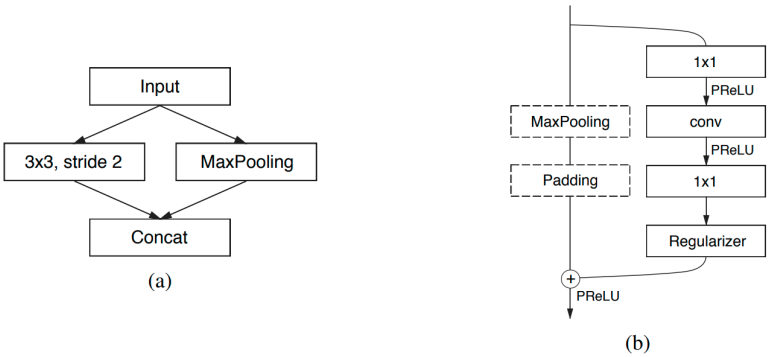


Figure 2: (a) ENet initial block. MaxPooling is performed with non-overlapping 2×2 windows, and the convolution has 13 filters, which sums up to 16 feature maps after concatenation. This is heavily inspired by [28]. (b) ENet bottleneck module. conv is either a regular, dilated, or full convolution (also known as deconvolution) with 3×3 filters, or a 5×5 convolution decomposed into two asymmetric ones.

FIGURE 4.4: Enet blocks [Source]

bottleneck blocks are essentially branches that separate from the main branch and then merge back by element-wise addition. The encoder consists of 3 stages, with 5 bottlenecks, stage three has no downsampling. the decoder consists of 2 stages with 3 and 2 bottlenecks respectively, followed by a fully convolutional layer for the final output. For downsampling, a max-pooling layer is used in the encoder, and a 2×2 convolution with stride$= 2$ is used instead of 1×1 convolution.

In this work, two variations of Enet architecture are used. The main difference is in upsampling strategies utilized in the main branches of the decoder bottleneck blocks. The first implementation uses unpooling as an upsampling mechanism, while the second one uses naive upsampling. In the original paper, zero-padding was used. The Enet variant with naive upsampling is evaluated with the intention to port the model to Google Coral embedded NN accelerator, which supports only basic layers and thus lacks the support of unpooling.

4.2.4 Self-attention distillation

Self-attention Distillation (SAD) was proposed by Yuenan Hou et al. [15], as a method for improving the training of lightweight models. He et al. proposed to utilize the information from attention maps derived from different layers of the same network. Attention maps show on which spatial areas of the input image the network focuses most. Knowledge distillation is a well-known concept to transfer the knowledge from the teacher(large network) to a student (small network). Self-attention distillation(SAD) proposes to transfer knowledge not from separate teacher network, but instead from the layers of the same network. Special branches called attention generators are incorporated into the network to derive activation-based attention maps from different layers of the network. To get a better understanding of how they work, assume the output of the internal layer, for which the attention map is generated is a tensor of size $m \times n \times k$. An attention map is $m \times n$ representation of this tensor. In the original paper, attention generation was implemented as follows. To

squeeze the initial tensor to $m \times n$, the authors tested a few functions. The best result was produced by using the sum of squares on the depth axis k. After that, bilinear interpolation was applied when needed to upsample the map, and the final step was normalization by spatial softmax function. In our implementation, we used L_2 normalization as the last step, and the rest remained the same.

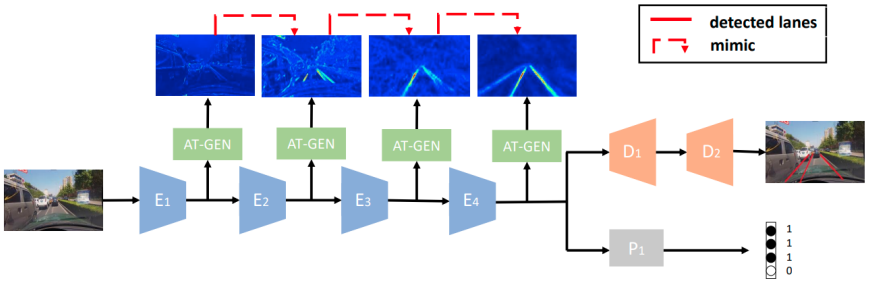


Figure 3. An instantiation of using SAD. $E_1 \sim E_4$ comprise the encoder of ENet [17], D_1 and D_2 comprise the decoder of ENet. Following [16], we add a small network to predict the existence of lanes, denoted as P_1. AT-GEN is the attention generator.

FIGURE 4.5: Enet-SAD architecture, from [15]

Such attention generator branches can be placed after any layer of the network. The way self-attention works is by optimizing the output of attention generator of one layer to the attention map of the other layer, or a label. A special distillation loss term has to be added to the loss function on the training step to enable such optimization. One of the paths in which the attention can be transferred in Enet is illustrated in Figure 4.5. One can easily define other ways of attention distillation within the network.

4.3 Loss functions

4.3.1 Segmentation losses and penalties

When training segmentation models, particular losses should be considered. Balanced cross-entropy loss (BCE) is one of the most common losses for segmentation problems. This is a variation of simple cross-entropy loss with weights for positive and negative examples that are used for unbalanced data. The loss has a serious drawback - it causes the model to be "unsure" on the edges of the binary map. In order to prevent this, some penalties are used alongside with BCE loss to improve predictions. Among such penalties, Jaccard and Dice losses are used, both of them add overlap measures to existing loss. The Jaccard coefficient adds intersection-over-union(IoU) overlap, while Dice adds an F1-score measure. In this work, BCE loss with the Dice penalty is used in most of the experiments to train the networks.

4.3.2 SAD loss

As was mentioned in section 4.2.4 to enable SAD within the network special loss term should be added. In the original paper [15] the distillation loss was formulated as follows:

$$\mathcal{L}_{\text{distill}}(A_m, A_{m+1}) = \sum_{m=1}^{M-1} \mathcal{L}_{\text{d}}(\Psi(A_m), \Psi(A_{m+1}))$$

Where $\Psi(A_m)$ denotes Attention generator output from layer A_m, and L_d is L_2 loss. The impact of each attention map on the distillation loss term is adjusted with coefficients. In experiments with SAD, our overall loss is comprised of BCE loss with a Dice penalty and a distillation loss, and the impact of each term is regulated with coefficients as well.

Chapter 5

Implementation and application

5.1 Frameworks and computational resources

Python 3.6 is used for all of the work. Keras [10] with TensorFlow [2] backend is used as the main framework for neural network building and training. Although it is a very high-level library for deep learning, with the use of some hacks, it was successfully utilized for self-attention distillation implementation, and all of the architectures used for the Experiments section in this thesis. For the manipulations with images albumentations [1] and OpenCV [4] was used. The implementation of Enets with upsampling and unpooling was taken from [24].

Most of the models were trained on Nvidia Geforce 1080 TI, with 11 GB of graphic memory. Enet with unpooling took approximately 1 minute per epoch on 78 421 images of size 256×256 with batch-size 16. In the same setting, one epoch for Unet took approximately 3 minutes.

5.2 Future application

5.2.1 Lane departure warning system

All of the work mentioned was done for future application in a lane departure warning system as a Proof of Concept project. Figure 5.1 provides a general scheme for such a system. The main algorithm is going to be run on Raspberry PI 4 Model B (4GB RAM, 1.5 GHZ quad-core) with acceleration on Google Coral Edge TPU.

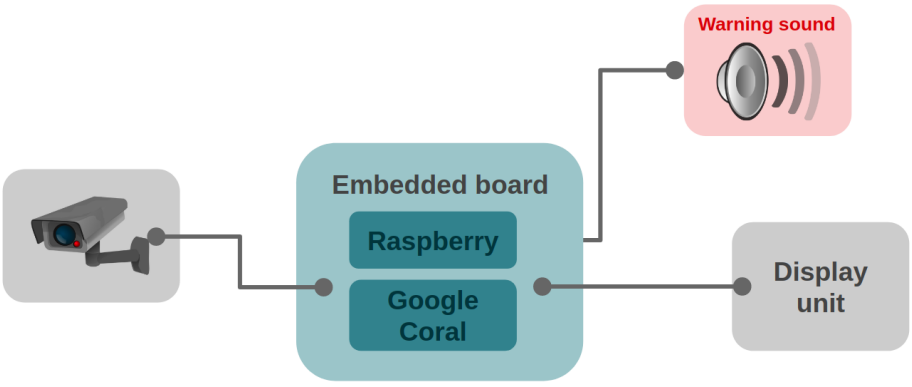


FIGURE 5.1: Lane departure warning system

The end-to-end algorithm for lane detection can be seen in Figure 5.2. As illustrated, a LaneNet [25] pipeline is a basis for this algorithm. As was mentioned before

Enet-SAD will be incorporated as a binary segmentation branch, but with shared encoder, as shown in the scheme.

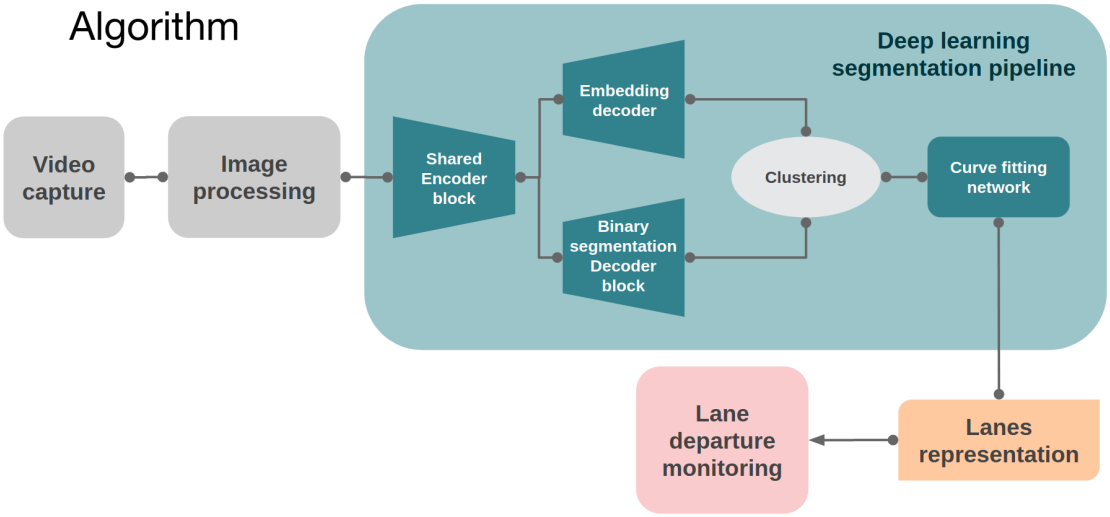


FIGURE 5.2: End-to-end real-time lane detection

5.2.2 Hardware and limitations

There are some requirements and limitations for the model to be run on Edge TPU. It must be converted to the TF Lite format and compiled by the Edge TPU compiler. Some specific operations are not supported by the converter and compiler. Therefore they should be replaced or removed.

Due to those limitations Enet SAD with unpooling could not be converted for real-time usage, but after minor modifications to Enet with naive upsampling, the team was able to run its quantized version on the specified hardware in real-time with prediction-time 12 milliseconds, 0.5965 F1-score(Dice) and 0.9826 pixel-wise accuracy. The measures provided here are evaluated on a 362-image validation set extracted from the original Tusimple train set, as mentioned in section 4.1.1.

Chapter 6

Experiments and results

Before diving into experiments, description evaluation metrics should be outlined. Some of the standard metrics for segmentation tasks that we used in this thesis are F1-score, or Dice coefficient, and pixel-wise accuracy. The metrics shown here are evaluated on the validation sets described in 4.1.1. While experimenting, we found out that the evaluation of the models on datasets is not the best criterion for judging their success as the data is not diverse enough. With the intention to find the data that differs from CULane and TuSimple, we tested our models on videos of Kyiv roads and pacific coast roads for a better understanding of how each model performs. In this section, you can find plenty of prediction visualizations on particular frames of the videos that picture various scenarios.

To train all our models, we used Adam optimizer and a ReduceLROnPlateau callback with a multiplying factor of 0.75, the minimum learning rate of 1e-6, 10 epochs of patience, and 1e-4 of minimum delta for Dice metric change. All our models are trained roughly to 150 epochs, the reason being that no significant improvement or change of trend is seen further on. With more training and finetuning, ENet-SAD unpooling architecture shows a 0.6239 dice coefficient with pixel-wise accuracy of 0.984, and Unet shows 0.6289 and 0.984, respectively.

6.1 Classical computer vision methods

FIGURE 6.1: Classical CV methods result

To test them on some images featuring different road conditions, we applied yellow and white color filters, defined a region of interest, then applied Canny edge detection followed by Hough line detection and postprocessing for filtering out the unnecessary lines and making corrections based on previous frames of the video.

The algorithm ended up failing even in relatively simple scenarios, as shown in Figure 6.1.

Although the method is easy to implement and does not require any additional learning it did not show satisfactory accuracy or robustness, and needs tons of post-processing to work properly, hence in the rest of the practical part of this work two segmentation neural network architectures were given a try - Unet and various modifications of Enet.

6.2 Unet

For this experiment, Unet was trained with BCE loss and dice penalty, starting at the learning rate of 1e-4. The model has approximately 31 million parameters, which is relatively heavy. This model also was converted for real-time usage, but the speed was too low for its usage. The results and process of training can be seen in Figure 6.2. The model generalizes quite good and is able to accurately place lanes even if they are occluded by a sequence of cars, but tends to catch a larger amount of false positives than Enet.

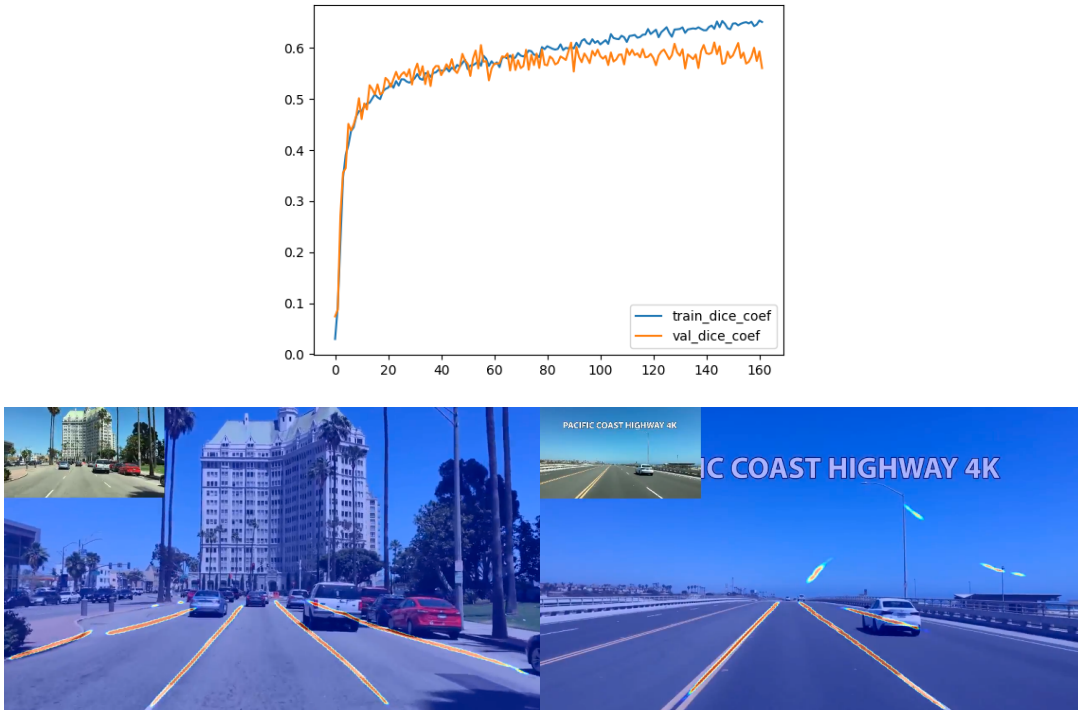


FIGURE 6.2: Unet results. Training process and Predictions

6.3 Enet unpooling

As a baseline for the set of experiments with this architecture, Enet was trained with BCE loss, a Dice penalty, and a starting learning rate of 1e-3. The results are the following, Figure 6.3

The comparison to Unet in terms of metrics can be seen in Table **??**. The difference with Unet predictions can be seen as well when comparing vdeoresults. Enet catches

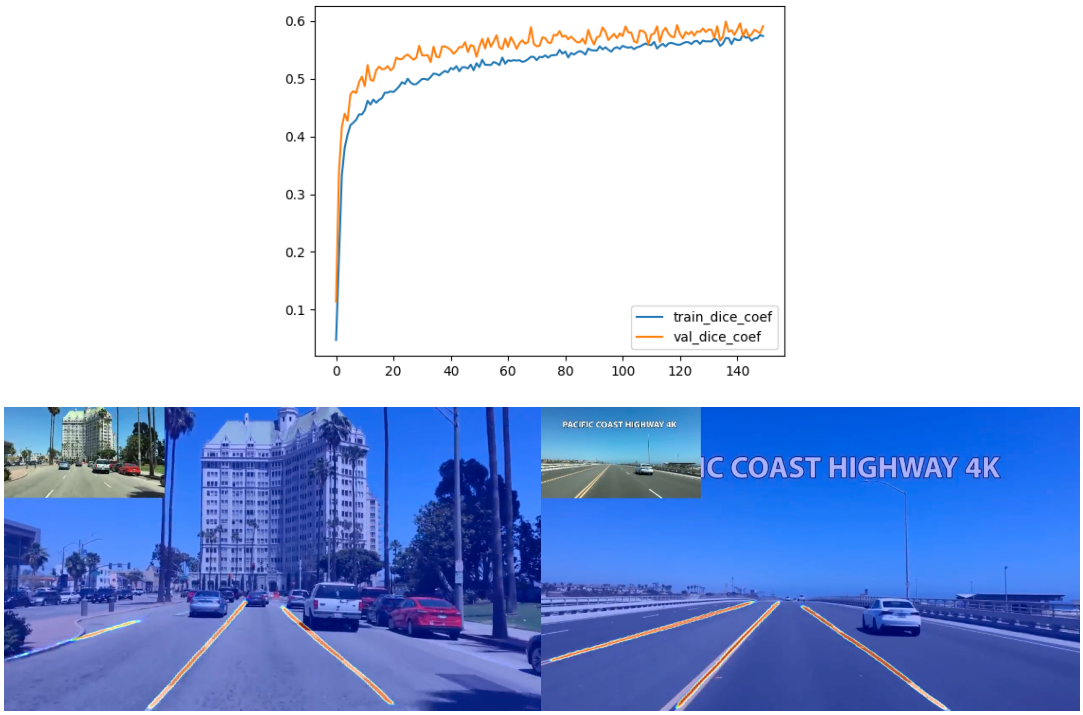


FIGURE 6.3: Enet-unpooling results. Training process and Predictions

a significant amount of false positives as well, but much less than Unet, it seems to have worse generalization abilities than Unet, however.

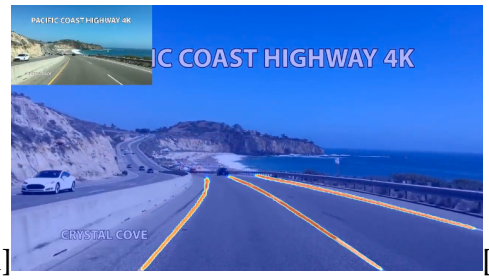
	TuSimple		**CULane**	
	Dice	Accuracy	Dice	Accuracy
Unet	0.6108	0.9835	0.4781	0.9804
Enet-unpooling	0.5992	0.9834	0.4674	0.9792

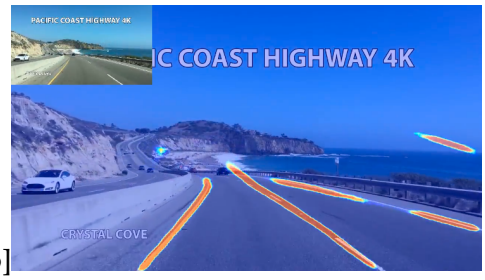
TABLE 6.1: Unet and Enet unpooling comparison.

6.3.1 Experiments with lane thickness

While experimenting with our data, we found that all models have quite a big false-positive rate. It mostly happens because of the simplicity of what the models try to learn as the features of a very thin white line are quite easily found on other objects. Therefore we tried to make our labels thicker to make the model more focused on the needed areas. The label width was increased from 5 pixels to 15 pixels so that the lines are significantly thicker, but the shapes of the lines are not distorted. The rest of the experiment setting is the same as the Enet-unpooling baseline. The results are in Figure 6.4. As can be seen, the f1-score is bigger, but in this case, it does not indicate any success as this metric is heavily impacted by the size of the labels. To see the results, we compare the two models by analyzing predictions on the image with lots of objects on the background.

As we can see, the performance is quite poor. The false-positive rate has only increased, the learning curves (Figure 6.4) show that increasing lane thickness caused the model to converge much faster and overfit very easily.

[a] 

[b] 

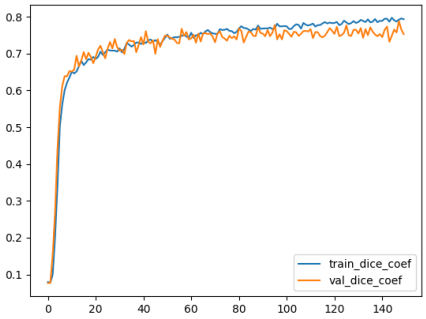
[c] 

FIGURE 6.4: [a] The result of the model trained on a dataset without changes. [b] The result of the model trained on a dataset with thicker labels. [c] The learning curves for a train with thick labels.

6.3.2 lane deletion

For lane deletion, a mask with a label width of 25 pixels was used. The reason for this is the fact that labels do not intersect with lane markings on the image as accurate as needed for proper lane deletion. With increasing the width of the line on a binary mask, we increase a percentage of markings deleted. The drawback of this is sufficient image distortion. Bigger width is used only for the lane deletion process, in the dataset itself, the labels have standard 5-pixel width.

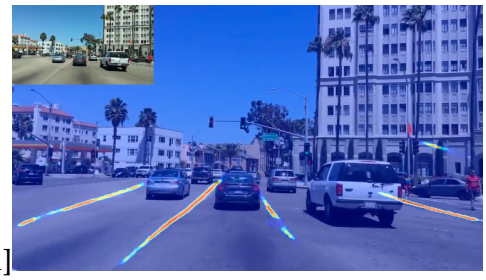
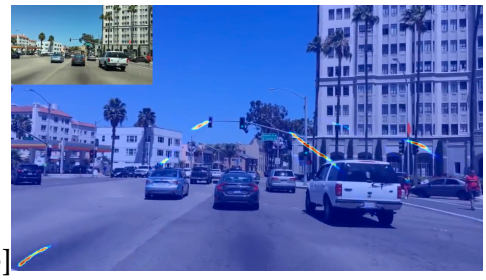
[a] 
[b] 

FIGURE 6.5: [a] The result of the model trained on a dataset without changes. [b] The result of the model trained on a dataset with deleted lanes.

The resulting dataset consists of both normal images and corresponding images with deleted lanes (3626×2). The validation set is the same as before. The rest of the experiment setting is the same as of Enet-unpooling baseline. The results are shown in Figure 6.5. To see the effectiveness of this augmentation, we decided to compare models by analyzing their predictions on the images with the unmarked road. As can be seen in the figure, the model did not capture any new global information about the lane placement and overfitted to the dataset instead.

6.3.3 Self-attention distillation

For this set of experiments, Attention generators were incorporated in Enet-unpooling architecture after the initial block and 1, 2, 3 Encoder bottlenecks. In the original paper [15], Enet architecture was modified, and also a small network for predicting the existence of the lanes was added. But we were interested in how much SAD influences training on its own, therefore apart from SAD, no changes to the network are made.

Chained attention path

In earlier Figure 4.5 you can see how chained SAD looks like on Enet. Each block of the encoder mimics the successive block. We set the parameters so that each attention generator has the same impact on training as the rest and Distillation loss influences the training the same amount as BCE loss with the Dice penalty. The rest of the setting is the same as in the Enet-unpooling experiment. The results are as follows: 6.6

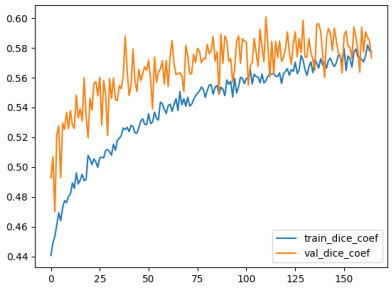


FIGURE 6.6: Enet-SAD-unpooling results with chained attention path. Training process and Predictions

Mimicking labels

This experiment differs from the previous only in the paths for attention distillation. While in the previous experiment, each attention map is compared to that of the next block, here, all of them are compared directly with labels (like deep supervision). The following results are received: 6.7

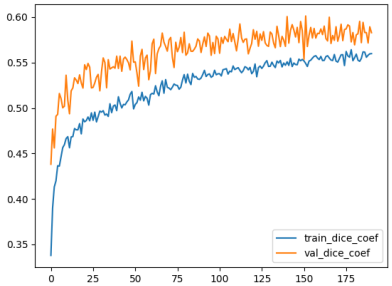


FIGURE 6.7: Enet-SAD-unpooling results with label mimicking. Training process and Predictions

Comparison

Even though the metrics results of SAD training are not as promising as in the original paper, multiple train sessions proved the stable positive difference in dice coefficient of approximately 0.01-0.1 and 0.005-0.02 difference in accuracy while training on TuSimple dataset comparing to the model without SAD. Table 6.2

TABLE 6.2: The performance of Enet without and with SAD variations on TuSimple dataset

	Accuracy	**Dice**
Enet-unpooling	0.9834	0.5992
SAD chained mimicking	0.9831	0.6008
SAD label mimicking	0.9826	0.6017

This does not seem to be much until the models are compared on videos. As we can see from Figure 6.8, the signal gets distilled the best when attention orients on labels. The small difference in metrics results can be explained with the nature of the TuSimple dataset. Most images of this dataset contain little to no city objects on the background, and therefore no false positives are predicted with or without SAD.

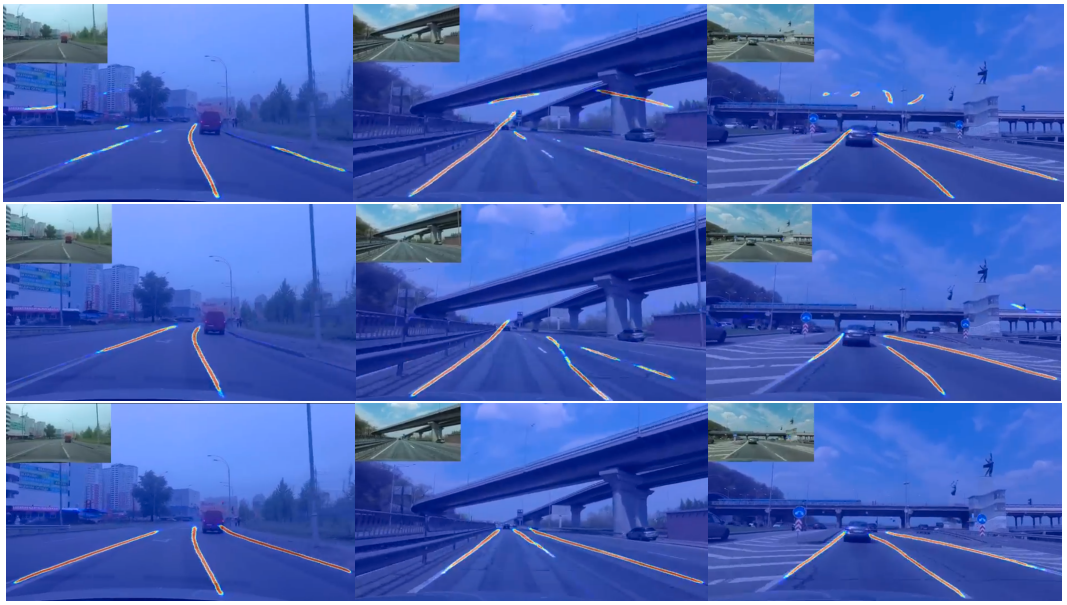


FIGURE 6.8: from top to bottom: Enet-unpooling predictions, chained-SAD predictions, label-mimicking SAD predictions. Prediction performed on Kyiv streets

6.4 Enet upsampling

TABLE 6.3: Enet with unpooling and upsampling performance on TuSimple dataset

	Accuracy	**Dice**
Enet-unpooling	0.9834	0.5992
Enet-upsamling	0.9823	0.5922

For the sake of the possibility of real-time usage, the unpooling mechanism is changed for naive upsampling. The rest of the experiment setting is the same as in the Enet-unpooling experiment. As can be seen in the results below 6.9, this results in some decrease in performance. SAD can be easily incorporated into this network as well. Learning curves (Figure 6.9) show that the model converges faster than Enet-unpooling with lower metrics numbers (table 6.3

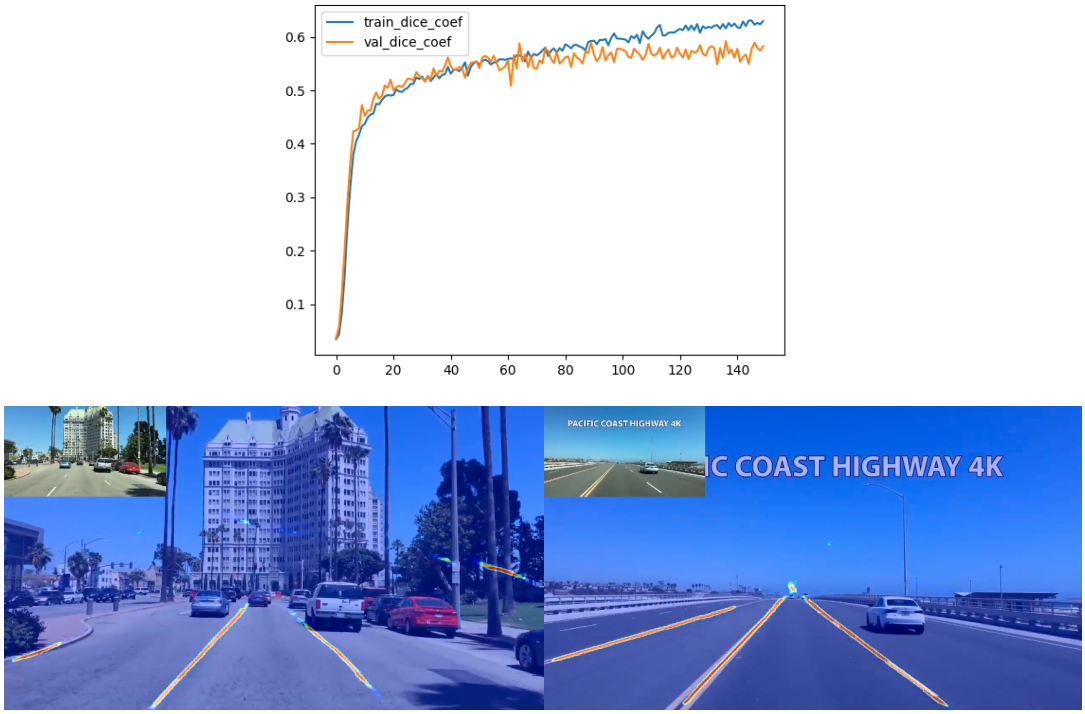


FIGURE 6.9: Enet-upsampling results. Training process and Predictions

Chapter 7

Summary

7.1 Contribution

- A new way of augmentation for road lanes is tried out.
- Self-attention distillation is incorporated to Enet architecture in Keras framework.
- An influence of SAD on training the models was investigated with a chained mimicking and label mimicking.
- the performance of Enet with unpooling and upsampling was investigated and compared.

7.2 Possible improvement

- Improve Enet. To investigate the influence of the various factors on training, we took the Enet model as-is, with no additional improvements. However, if we make the improvements suggested in [15], the results should be better.
- Try other architectures. Enet is not the only solution for lightweight segmentation, such architectures as ICnet or LinkNet should be considered for possible improvement. Also, better architectures for capturing global context when no lanes on the road are visible can be found.
- Augment better and get more data. New ways of weather imitation can be tried, as well as changing the structure of the road. For example, this [22] resource provides a notable example of such augmentation. Besides, more data can be added. For example, BDD 100K dataset [37] has lane markings classification labels, which can be adjusted for our task. A small testing dataset can be created from Lviv streets photos as well. The variety of road structures in this city can be a valuable testing material.
- Future work. A binary branch should be incorporated into the main lane detection pipeline, described in Future Application section 5.2.

Bibliography

[1] E. Khvedchenya V. I. Iglovikov A. Buslaev A. Parinov and A. A. Kalinin. "Albumentations: fast and flexible image augmentations". In: *ArXiv e-prints* (2018). eprint: 1809.06839.

[2] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[3] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. *SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation*. 2015. arXiv: 1511.00561 [cs.CV].

[4] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[5] J Canny. "A Computational Approach to Edge Detection". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 8.6 (June 1986), 679–698. ISSN: 0162-8828. DOI: 10.1109/TPAMI.1986.4767851. URL: https://doi.org/10.1109/TPAMI.1986.4767851.

[6] Abhishek Chaurasia and Eugenio Culurciello. "LinkNet: Exploiting encoder representations for efficient semantic segmentation". In: *2017 IEEE Visual Communications and Image Processing (VCIP)* (2017). DOI: 10.1109/vcip.2017.8305148. URL: http://dx.doi.org/10.1109/VCIP.2017.8305148.

[7] Liang-Chieh Chen et al. *DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs*. 2016. arXiv: 1606.00915 [cs.CV].

[8] Liang-Chieh Chen et al. *Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation*. 2018. arXiv: 1802.02611 [cs.CV].

[9] Liang-Chieh Chen et al. *Rethinking Atrous Convolution for Semantic Image Segmentation*. 2017. arXiv: 1706.05587 [cs.CV].

[10] François Chollet et al. *Keras*. https://keras.io. 2015.

[11] François Chollet. *Xception: Deep Learning with Depthwise Separable Convolutions*. 2016. arXiv: 1610.02357 [cs.CV].

[12] J. Deng et al. "ImageNet: A Large-Scale Hierarchical Image Database". In: *CVPR09*. 2009.

[13] Richard O. Duda and Peter E. Hart. "Use of the Hough Transformation to Detect Lines and Curves in Pictures". In: *Commun. ACM* 15.1 (Jan. 1972), 11–15. ISSN: 0001-0782. DOI: 10.1145/361237.361242. URL: https://doi.org/10.1145/361237.361242.

[14] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[15] Yuenan Hou et al. *Learning Lightweight Lane Detection CNNs by Self Attention Distillation*. 2019. arXiv: 1908.00821 [cs.CV].

[16] Nick Kanopoulos, Nagesh Vasanthavada, and Robert L Baker. "Design of an image edge detection filter using the Sobel operator". In: *IEEE Journal of solid-state circuits* 23.2 (1988), pp. 358–367.

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'12. Lake Tahoe, Nevada: Curran Associates Inc., 2012, 1097–1105.

[18] Ammu Kumar and Philomina Simon. "Review of Lane Detection and Tracking Algorithms in Advanced Driver Assistance System". In: *International Journal of Computer Science and Information Technology* 7 (Aug. 2015), pp. 65–78. DOI: 10.5121/ijcsit.2015.7406.

[19] John Lafferty, Andrew Mccallum, and Fernando Pereira. "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data". In: Jan. 2001, pp. 282–289.

[20] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[21] Guosheng Lin et al. *RefineNet: Multi-Path Refinement Networks for High-Resolution Semantic Segmentation*. 2016. arXiv: 1611.06612 [cs.CV].

[22] Ming-Yu Liu. *Unsupervised Image-to-Image Translation*. https://github.com/mingyuliutw/UNIT. 2018.

[23] Jonathan Long, Evan Shelhamer, and Trevor Darrell. *Fully Convolutional Networks for Semantic Segmentation*. 2014. arXiv: 1411.4038 [cs.CV].

[24] Pavlos Melissinos. *ENet-keras*. https://github.com/PavlosMelissinos/enet-keras. 2017.

[25] Davy Neven et al. *Towards End-to-End Lane Detection: an Instance Segmentation Approach*. 2018. arXiv: 1802.05591 [cs.CV].

[26] Xingang Pan et al. *Spatial As Deep: Spatial CNN for Traffic Scene Understanding*. 2017. arXiv: 1712.06080 [cs.CV].

[27] Adam Paszke et al. *ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation*. 2016. arXiv: 1606.02147 [cs.CV].

[28] Tobias Pohlen et al. *Full-Resolution Residual Networks for Semantic Segmentation in Street Scenes*. 2016. arXiv: 1611.08323 [cs.CV].

[29] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

[30] Ujjwal Saxena. *Automold–Road-Augmentation-Library*. URL: "https://github.com/UjjwalSaxena/Automold--Road-Augmentation-Library".

[31] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. arXiv: 1409.1556 [cs.CV].

[32] Christian Szegedy et al. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV].

[33] Christian Szegedy et al. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV].

[34] Alexandru Telea. "An Image Inpainting Technique Based on the Fast Marching Method". In: *Journal of Graphics Tools* 9 (Jan. 2004). DOI: 10.1080/10867651.2004.10487596.

[35] *Tusimple benchmark*. URL: https://github.com/TuSimple/tusimple-benchmark.

[36] Ping Luo Xiaogang Wang Xingang Pan Jianping Shi and Xiaoou Tang. "Spatial As Deep: Spatial CNN for Traffic Scene Understanding". In: *AAAI Conference on Artificial Intelligence (AAAI)*. 2018.

[37] Fisher Yu et al. *BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning*. 2018. arXiv: 1805.04687 [cs.CV].

[38] Hengshuang Zhao et al. *ICNet for Real-Time Semantic Segmentation on High-Resolution Images*. 2017. arXiv: 1704.08545 [cs.CV].

[39] Hengshuang Zhao et al. *Pyramid Scene Parsing Network*. 2016. arXiv: 1612.01105 [cs.CV].

[40] Qin Zou et al. "Robust Lane Detection From Continuous Driving Scenes Using Deep Neural Networks". In: *IEEE Transactions on Vehicular Technology* 69.1 (2020), 41–54. ISSN: 1939-9359. DOI: 10.1109/tvt.2019.2949603. URL: http://dx.doi.org/10.1109/TVT.2019.2949603.