End-to-End Lane Detection

Andreas Giannoutsos
sdi1700021

Abstract

Auto lane keeping is one of the many driver assistance technologies that are becoming more common in modern vehicles. The latter enables the car to correctly place itself inside the road lanes, which is also critical for any subsequent lane deviation or trajectory planning decision in fully autonomous vehicles. Classic lane detection methods rely on a variety of highly advanced, hand-crafted features and heuristics, which may be efficient computationally, however they are vulnerable to scalability due to the vast changes and differences in the road scenes. More recently, with the advances in machine learning and especially with the Convolutional NNs, the hand-crafted feature detectors have been replaced with deep networks to learn predictions on pixel-wise lane segmentations. In this paper, we are going to solve the lane detection problem with different methods. To be more specific, we feed a dataset of highway lane images to both methods in order to compare them. First, we try the traditional edge-detection method with hand-crafted features, and then we look at different Deep Convolutional Network architectures to address this problem. Finally, we compare these methods by using images from their output. Following an analysis of the model outputs, we evaluate the ability of the algorithms to detect the lanes on the road effectively.

1. Introduction

Lane detection is crucial in autonomous driving because lanes can serve as important cues for restricting vehicle maneuvering on roads. The goal in autonomous driving is to gain a complete understanding of the environment around the car by employing a variety of sensors and control modules. Camera-based lane recognition is a significant step toward such environmental awareness because it helps the car to align itself correctly inside the road lanes. However, due to poor lighting conditions, occlusions caused by other vehicles, irrelevant road markings, and the inherent long and thin nature of lanes, detecting lanes in the wild is challenging.

There are two types of lane detection algorithms: traditional methods and deep learning-based approaches. Traditional techniques attempt to take advantage of hand-crafted low-level or advanced features. (Sun et al., 2006) detects lanes in the HSI color representation. Meanwhile (Yu and Jain, 1997) and (Macek et al., 2004) extract lane boundaries by using the Hough Transform. They firstly extract edges on the road with a Canny edge detector (Canny, 1986), and then road lanes are obtained with the use of Hough Transforms. This pattern of edge detection continuous with the use if statistical transforms like Kallman and particle filters (Liu et al., 2010). In general, traditional approaches, need a complex selection of features and they suffer from robustness issues as a result of road scene variations that are difficult to model.

Deep networks, rather than hand-crafted feature detectors, have been used to learn dense predictions, i.e. pixel-wise lane segmentations, in more recent methods. These methods have

lately demonstrated superiority in lane detection due to their high capacity to learn lane features end-to-end. (Huval et al., 2015) were the first to try, and analyze empirically the use of a CNN on the lane detection problem. They used the output of the network as a regression tool to match the end-points of a lane. Recently the lane detection problem has been shifted to an image segmentation problem due to the success in relevant semantic segmentation work like (Long et al., 2015a). The architecture and the efficiency of these models have improved substantially (Ronneberger et al., 2015) and have been used in many other problems.

(Neven et al., 2018) have solved the lane detection problem as a segmentation with the use of convolutional auto-encoders (Badrinarayanan et al., 2016). In this article, we will test different architectures of convolutional autoencoders specialized in lane segmentation. Together will use a classic lane detector which consists of a simple Canny edge detector and a Hough transform. In the methods section, we will briefly analyze each method and finally, we will compare each method based on their final output and their complexity.

2. Methods

2.1. Canny edge detection

First and foremost, in order to extract the lanes of an image we need to detect their edges. One of the most popular edge detection methods is the Canny edge detector (Canny, 1986). It is a multi-stage algorithm and it is one of the best methods for detecting consistent edges. The algorithm can be broken down into five steps.

2.1.1. gaussian filter

Because all edge detection elements are clearly influenced by image noise, it is critical to block out the noise to avoid false detection.

$$B = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} * A$$

2.1.2. gradient intensity

Because an image edge can found in any direction, the Canny algorithm detects horizontal, vertical, and diagonal edges in the blurred image using four filters. The edge detection operator calculates the first derivative in both the horizontal G_x and vertical V_x directions G_y, and it can be calculated as follows.

$$G = \sqrt{G_x^2 + G_y^2}$$
$$\theta = \arctan 2G_x, G_y$$

2.1.3. non-maximum suppression

Lower bound cut-off suppression is used to find the places with the strongest change in pixel values. We evaluate the edge strength of the current pixel to the edge strength of the pixel in the positive and negative gradient directions. If the edge intensity of the current pixel is higher than the other pixels in the mask in the same direction, the value will be kept. Alternatively, it will be suppressed.

2.1.4. double threshold

Non-maximum suppression gets a much more adequate detection of actual edges in an image. However, noise and color variance continues to affect certain edge pixels. Selecting high and low threshold values achieves the solution. If the gradient value of an edge pixel is greater than the high threshold value but less than the low threshold value, it is classified as a strong edge pixel; if it is less than the high threshold value but greater than the low threshold value, it is classified as a weak edge pixel; and if the value is less than the low threshold, it is classified as suppressed.

2.1.5. hysteresis thresholding

Even after suspending the weak pixels two times in the row, variance and noise may still affect the determination if an edge exists. A weak edge pixel resulting from weak edges is usually connected to a strong edge pixel, whereas noise responses are not. By admitting this reality, Blob analysis then is used to track the edge connection by looking at a weak edge pixel and its eight connected neighbor pixels. There will be certainly an edge product as long as there exists a strong value pixel. Figure 1 illustrates the hysteresis thresholding (can).

2.2. Hough transform

The Hough transform is a technique used to extract features. The technique's primary objective is to use a voting procedure to find imperfect instances of objects within a given class of shapes. The lane on the road is an imperfect instance of a shape in our case. The edge detector generates points, which should be used to obtain lines.

The Hough transform is used in many model cases. The simplest one is the detection of straight lines. To begin with we need to model our shape in a set of parameters. The straight line can be modeled with Hessel normal form,

$$r = xcos(\theta) + ysin(\theta)$$

where r is the distance between the point 0 and the closest point of the line and θ is the angle between the x-axis and the previous distance line.

The linear Hough transform algorithm is used to estimate the two straight-line parameters. Each point in the transform space serves as an accumulator to detect or classify a line described by this, and the transform space has two dimensions. At each step, the algorithm iterates over all of the detected edges in the image, adding to the accumulator. The most probable lines can be obtained and their parameters can be collected by finding the containers with the highest values in the accumulator space. Depending on the nature

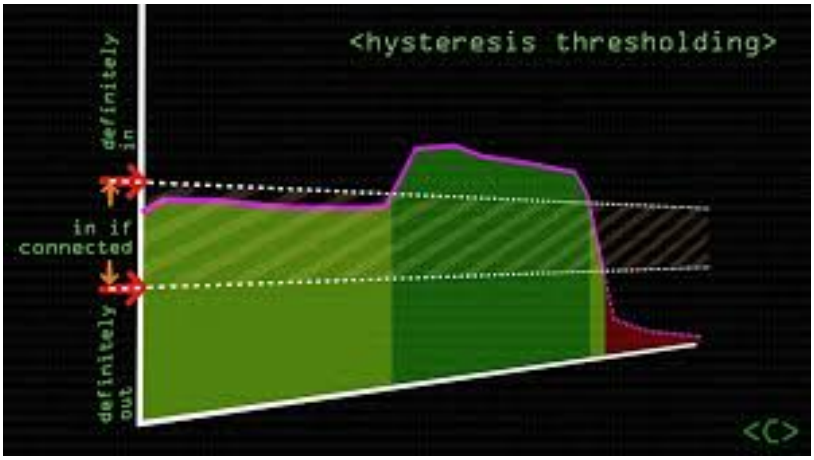

Figure 1: Hysteresis thresholding. The 2 dotted lines represent the high and low threshold. Everything that surpasses the high threshold is directly classified as a line. Next, if this line is connected and passes the high threshold, it keeps been classified as a line as long as it stays between the high and low thresholds. If it surpasses the low threshold, the line is terminated.

of the problem an optimized voting process may be utilized, however, we usually search for maxima in the accumulator space. The final result of the transform is the parameters that most likely form a line on the corresponding edge pixels.

Figure 2 clearly shows the intersection of related parameter lines in the parameter space.

In our situation, the number of lanes we need to detect should be no more than two. That could be optimized by dividing it into two bins between the left and right lanes and averaging all the parameters within these bins.

2.3. Convolutional Neural Networks

Deep learning has powerful applications in many practical fields of science and technology. Deep convolutional neural networks (deep CNNs) are an important family of artificial deep neural networks. Empirical evidence (Huval et al., 2015) suggests that convolutions help deep CNNs to efficiently learn locally shift-invariant features, allowing them to demonstrate their capabilities in texts, images, and several other types of data.

CNNs consist of multiple 2D convolutions over the images where the kernels of the filters are the trainable parameters. The expression of the 2D convolution is:

$$g(x,y) = conv\left(f(x,y)\right) \Rightarrow g(x,y) = \omega * f(x,y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} \omega(dx,dy) f(x+dx, y+dy)$$

where $g(x,y)$ is the product of the convolution and ω is the kernel. The CNNs have multiple layers and sizes of kernels ω^L. In addition every convolution output is passed

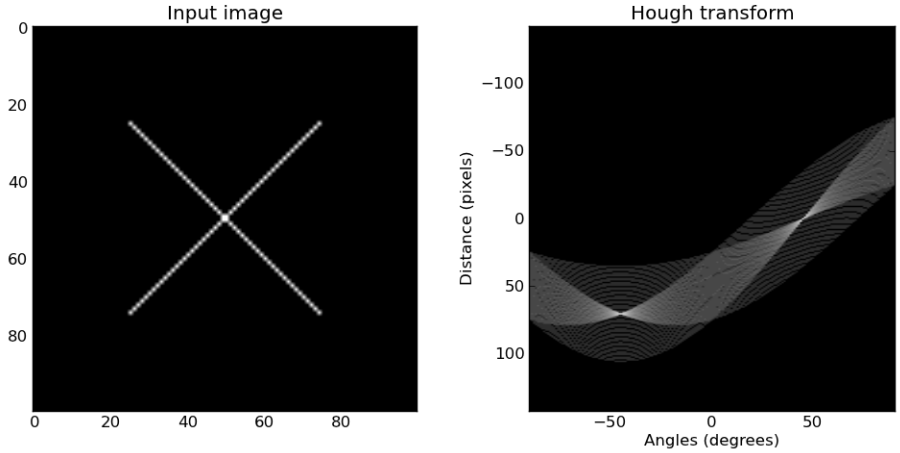

Figure 2: Hough transform accumulator space. The first figure shows the points that are detected as edges. The second figure contains all the possible parameters that form all the possible lines for the first figure. 2 intersection points are visible from figure 2. These 2 sets of parameters are the 2 lines detected in the first figure.

through a non-linear function. More preciously at the l^{th} layer of the CNN we have the following:

$$conv(a^{[l-1]}, K^(n)) = \sigma^{[l]} \sum_{i=1}^{n_H^{[l-1]}} \sum_{j=1}^{n_W^{[l-1]}} \sum_{k=1}^{n_C^{[l-1]}} K_{ijk}^{(n)} a_{x+i-1,y+j-1,k}^{[l-1]} + b_n^{[l]}$$

K^n are the kernels, a is the input of the layer, σ^l is the non-linear function or activation function at the l^{th} layer and $b_n^{[l]}$ are the biases at that layer. After several convolutional layers, the CNN model produces a predicted picture. The mean error between the predicted and target images is then calculated using the loss function, and the loss is propagated to the parameters using the back-propagation process.

2.3.1. Unet Autoencoder

Convolutional networks are often used for classification operations, where the output to an image is a single class mark. However, in many visual tasks, especially image segmentation, the desired output should provide localization, i.e., each pixel should be given a class label. Some of the most recent approaches to image segmentation have attempted to apply deep algorithms optimized for class prediction to pixel-wise labeling like (Long et al., 2015b). The findings, although promising, are not conclusive. This is large since maximum pooling and sub-sampling limits the feature map resolution.

For that reason, an alternative architecture of a convolutional network is needed, where it takes into account the localization of the class. One of the most influential networks in image segmentation, U-Net proposed by (Ronneberger et al., 2015) is an autoencoder

model. An autoencoder model consists of an encoder that encodes latent features of the image and a decoder that decodes and upsamples these features to the original image shape.

The U-Net architecture includes an encoder for context-capturing and a symmetric decoder for approximating the location. The high-resolution functions from the encoder are linked to the upsampled output for localization. Each encoder's layer has 2 3x3 convolutions and a 2x2 max pooling layer for downsampling accompanied by a ReLu activation function. After 2 downsamplings, the number of function channels is doubled. Similarly, each layer of the decoder, that upsamples the feature map, consists of an up convolution 2x2 layer and then two 3×3 convolutions with a ReLu layer. After each upsampling, the number of functional channels is halved. Skip connections are formed between the encoder and decoder by concatenating the layers with the same number of features to propagate high-resolution features to the decoder, allowing precise localization. U-Net architecture is illustrated in Figure 3.

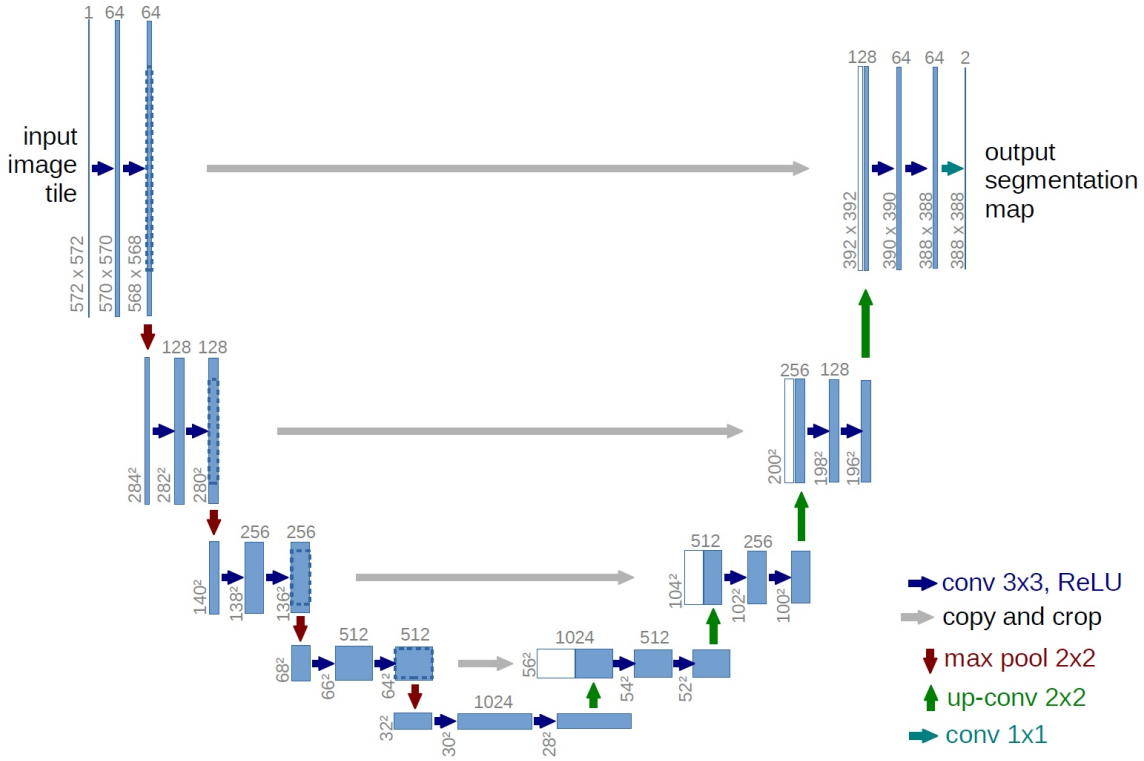

Figure 3: U-Net architecture. The blue boxes represent the convolutional blocks. The grey arrows describe the feature concatenation connecting the encoder and the decoder.

The autoencoder model performance is defined in its ability to encode as many latent features of the image as possible. In our case, we can attach to the U-Net model our

encoder with different convolutional blocks while maintaining the same skip and concatenate connections across the encoder and the decoder.

2.4. ResNet

A residual neural network (ResNet) (He et al., 2015) is a form of neural network that uses skip connections or shortcuts to move between layers. A skip connection block, building block of the ResNet, is shown in Figure 4. There are two key reasons to incorporate skip connections: to prevent the issue of vanishing gradients, or to minimize the Degradation (accuracy saturation) problem, which occurs when adding more layers to a sufficiently deep model leads to higher training error. Figure 5 demonstrates the residual model architecture.

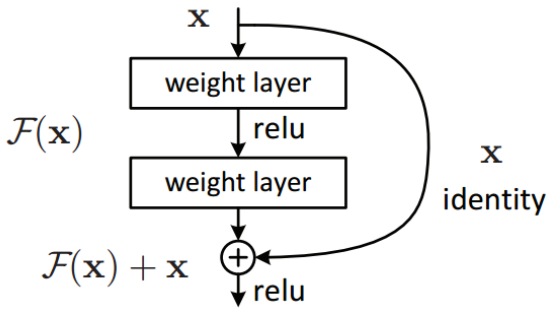

Figure 4: ResNet convolution block. The input of the block is propagated and added to the end of the block.

2.5. EfficientNet

Convolutional neural networks are typically designed with a limited resource budget and then scaled up for greater precision as more resources are available. EfficinetNet (Tan and Le, 2020) investigates model scaling in detail and discovers that carefully matching network depth, width, and resolution will contribute to improved results. Figure 6 illustrates the scaling of the network. It employs neural architecture search to develop a new baseline network and scale it up to produce a family of models known as EfficientNets, which outperform previous ConvNets in terms of precision and performance.

On ImageNet and five commonly used transfer learning datasets, the EfficientNet model can be scaled up very efficiently, surpassing state-of-the-art precision with an order of magnitude fewer parameters and FLOPS. The B0 EfficentNet, the first optimized architecture after the base model, is the one that we will be using and its width, depth and resolution are scaled by the following factors:

$$depth^{1.2}, width^{1.1}, resolution^{1.15}$$

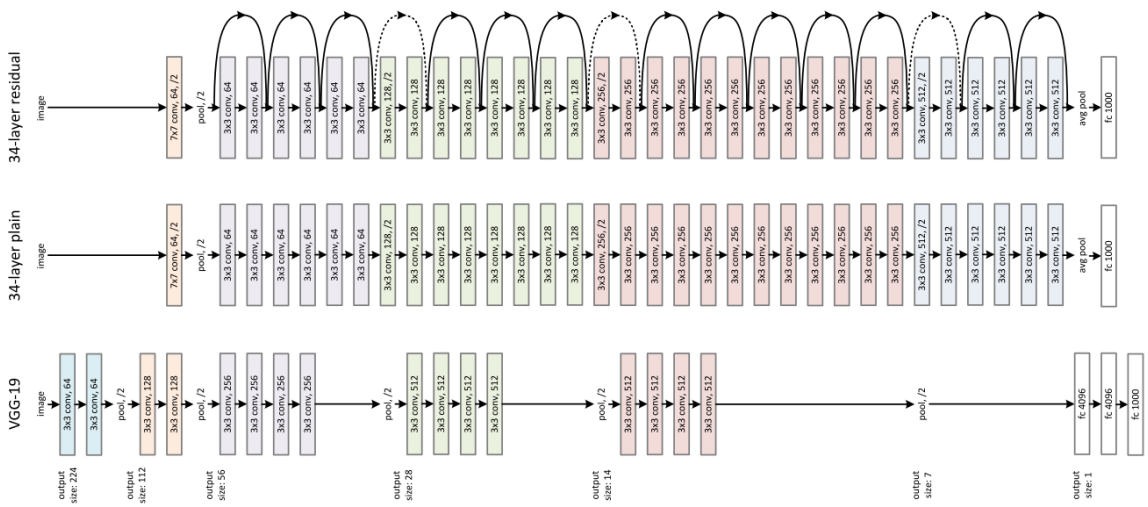

Figure 5: ResNet architecture. The VGG19 models are at the right. The middle model represents a simple network with no residual links. The residual model is the top model; it is a simple model of residual blocks.

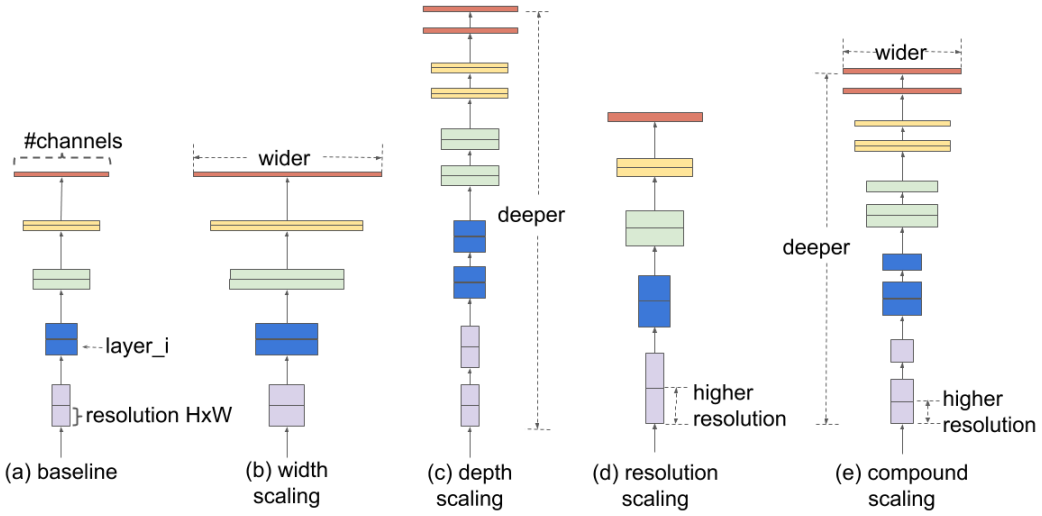

Figure 6: This figure explains the ways of scaling a model. (a) is a baseline network example. (b)-(d) are conventional scaling methods that only increases one dimension of network width, depth, or resolution. (e) is a scaling method that uniformly scales all three dimensions with a fixed ratio.

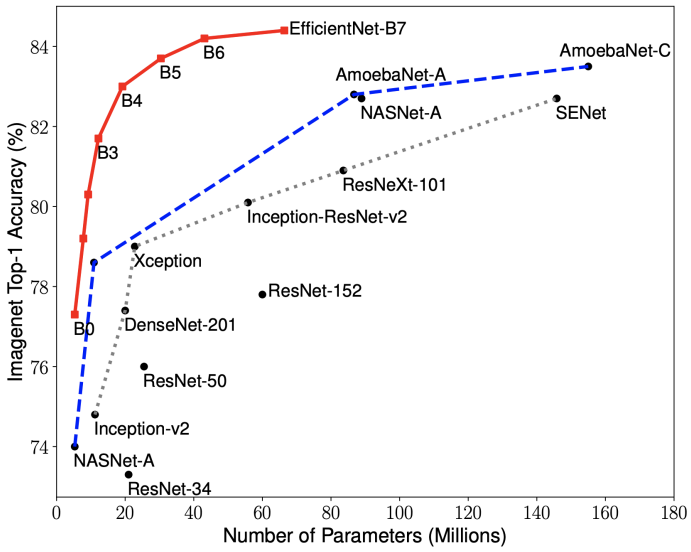

Figure 7: FLOPS vs ImageNet Accuracy with the EfficentNet baselines. This graph shows how well the other inefficient models perform in relation to their computational cost.

2.6. MobileNet

Convolutional neural networks advancements increase performance but do not actually make networks more effective in terms of size and time. Many real-world applications, such as self-driving cars, need processes to be completed quickly with limited resources. The MobileNet (Howard et al., 2017) model is designed to be lightweight and suitable for use in smartphone applications. Depthwise separable convolutions are used by MobileNet. When compared to the network with regular convolutions of the same depth in the nets, it significantly reduces the number of parameters.

Depth-wise separable convolutions were inspired by the concept that the depth and spatial component of a filter can be isolated, hence the term separable. Consider the Sobel filter.

$$Gx = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * A$$

From matrix factorisation methods, Gx can be defined as the matrix product of $[1, 2, 1]$ with $[-1, 0, 1]^T$.

We note that the filter used to have 9 parameters, but now it just has 6. This was made possible by separating the height and width measurements. The same concept can also be used to separate the depth dimension from convolutions. Following that, we apply a 1*1 filter to cover the depth dimension.

By using this type of convolutions we can drastically reduce the operations needed for a convolution. Before a 3x3 filter to produce, a 3x1 output needed 3x3x3x3 multiplications. The separable ones need only 3x3x3+3=36 multiplications. MobileNet lags about $1-2\%$ behind other popular CNN models in image detection tasks while having a fraction of their parameters. MobileNet demonstrates superior speed, size and accuracy.

3. Experiments

3.1. Dataset

The dataset used in the experiments is the TuSimple dataset https://github.com/TuSimple/tusimple-benchmark. TuSimple contains 6,408 road photos from US highways. The picture has a resolution of 1280x720. The dataset sample set is made up of 3,626 images for training, 358 for validation, and 2,782 for testing, all of which are taken in various weather conditions.

3.2. Training

For the training, the resolution of the images was reduced to 128x128 and for the deep learning methods, the number of channels was reduced from 3 to 1 due to hardware limitations. The training of the models was done in Google Colab using GPU runtime. The model parameters were the same, with the learning rate set at 10^{-4}. The batches were of size 32 and the models were trained for 15 epochs, taking about 7 hours each. In all of the cases, Adam optimizer was applied and the Binary Cross Entropy loss was used. The code can be found at https://github.com/AGiannoutsos/End-to-End-Lane-Detection and the experiments together with version of the models can be found at https://wandb.ai/andreas_giannoutsos/lane_detection. For the implementation of the models, the segmentation library (Yakubovskiy, 2020) was used.

4. Results

4.1. Classic lane detection

4.1.1. no average slopes

As we can see from the images, the results of the classic line detector without receiving the average of the line calls are not very satisfactory. In Figure 8 we can see the lines that are formed. The many different lines end up forming 2 imaginary lines that make up the 2 lines on the road. This effect, however, has been processed as the area in which lines are detected is only the triangle in front of the image. This method does not appear to be fully accurate in recognizing different dynamic environments. The 9 images in the results show different values of the hysteresis thresholding sampling. However, when the brightness in the image is increased, these different threshold values are not enough to separate the road lines. This can be seen in Figure 9, where the brightness is so high that determining the best threshold pair makes little sense.

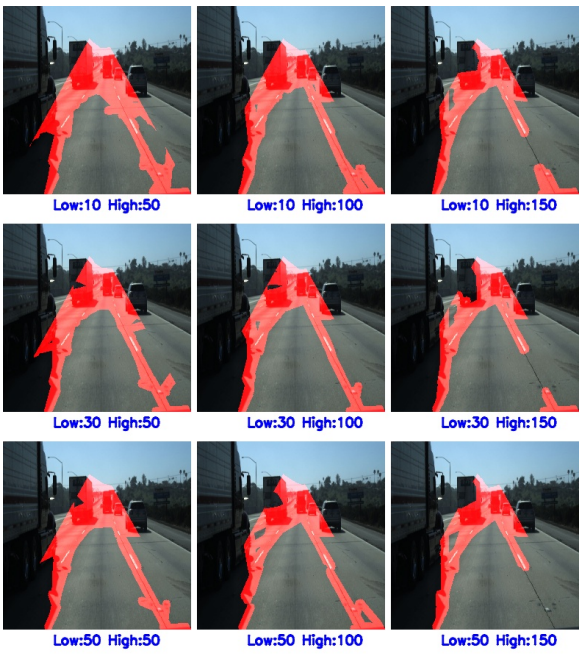

Figure 8: Classic lane detection without averaging the slopes of the detected lines. There are 9 result images in the classic lane detection result pictures. They reflect various threshold values during the hysteresis thresholding process. The low? and high? thresholds are indicated by the labels beneath the picture.

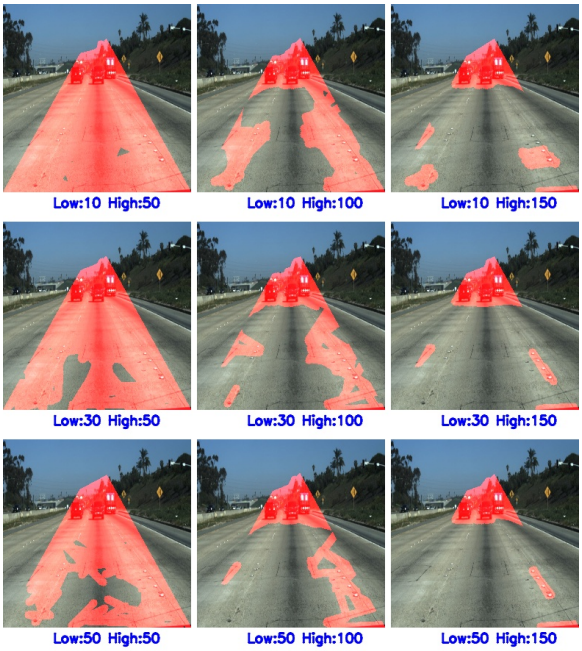

Figure 9: Classic lane detection without averaging the slopes of the detected lines but with greater brightness

4.1.2. average slopes

One way to partially reduce this problem, however, is to take the average of the slopes of the detected lines so as to create only 2 lines on the road in front of the driver. In Figure 10 we can see more satisfactory results as lines are normally formed on the road at points where we would drive. However, the peculiarity of the asphalt in some places makes the forecast problematic. Also it would be very wrong to exclude some thresholds as in some that the detector is doing well in this image it is not doing well in the previous example with the increased brightness. Moreover in Figure 11, there is an additional problem here as in the turns at many thresholds we do not have a good result at all and that is another reason why it is very difficult to choose a price. For this reason we should move away from the logic of finding the best parameters and perhaps we should let the parameters automatically adjust the street images based on.

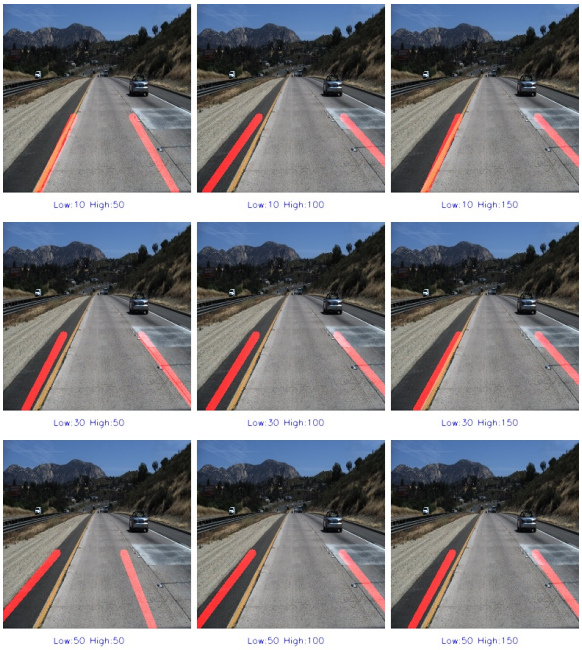

Figure 10: Classic lane detection with average slopes

4.2. Convolutional Neural Networks

As we move to the machine learning methods in Figure 12 we have immediately much better results. The results of all 4 different models of machine learning are presented there at the same time. The superiority of this method from the classical one is immediately apparent as the parameters are adjusted based on all the images in the database for training that we have available. The training curve can be found in Figure 15. We can see from this that, with the exception of the simplest model with the fewest parameters, the other models reach a very high degree of optimization.

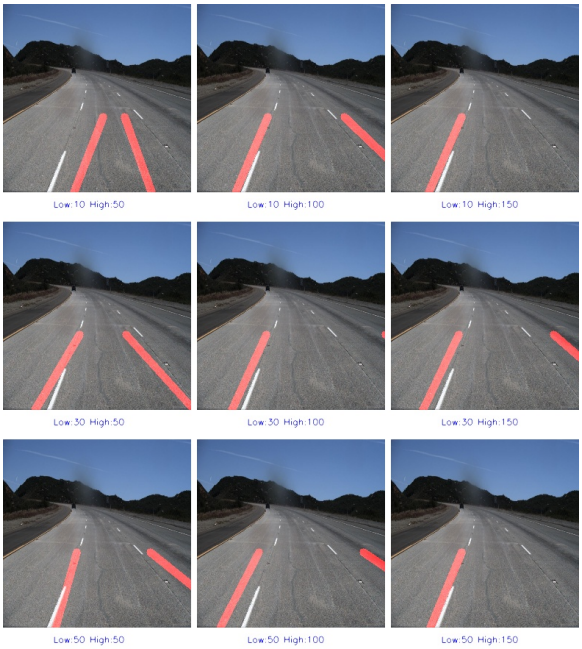

Figure 11: Classic lane detection with average slopes missing the curve

4.2.1. Simple Autoencoder

In the simplest model, which lacks the residual connections and the structure of the U-Net, the results in the straight line are very good and even lines of the other strips are detected, which is particularly useful. However in some cases like Figure 14 the prediction is noisy.

4.2.2. ResNet

The ResNet model having the largest set of parameters has the best performance of all the other models observing the characteristic forecast images. However, its superiority is not sufficiently justified by the number of its parameters as the other models achieve almost similar results with orders of magnitude smaller parameters.

4.2.3. EfficientNet

The EfficientNet having fewer parameters but a more sophisticated combination of its architecture achieves particularly good results even in Figure 14 where it predicts the line in this difficult case among all the other models.

4.2.4. MobileNet

Finally, MobileNet having the smallest number of parameters has particularly good results. With 10 times fewer parameters than ResNet, its training is easier and the prediction of lines can be done from a small computer device. Even in difficult examples such as Figure 14 or Figure 13, MobileNet can give quite good predictions.

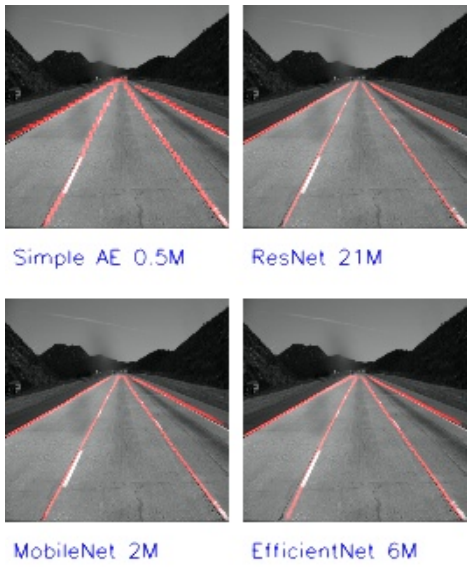

Figure 12: Autoencoder models in straight lanes. We have 4 images in the machine learning lane detection results. These are the 4 Deep Convolutional Network architectures that we tested. The number of parameters that the model has is indicated at the end of their name.

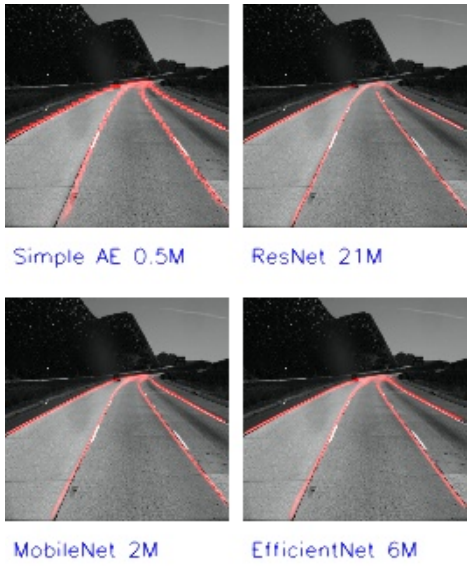

Figure 13: Autoencoder models detect curved lanes

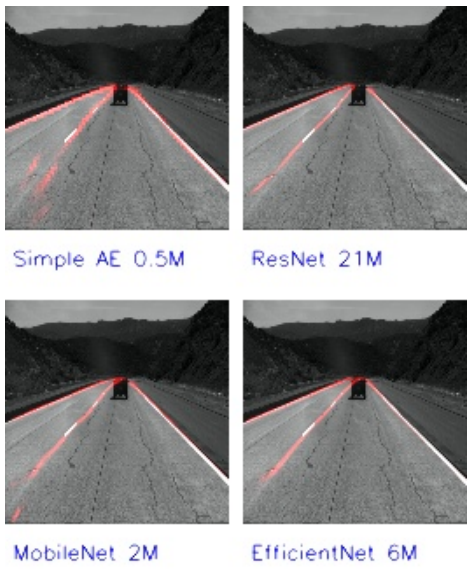

Figure 14: Autoencoder models having noisy prediction

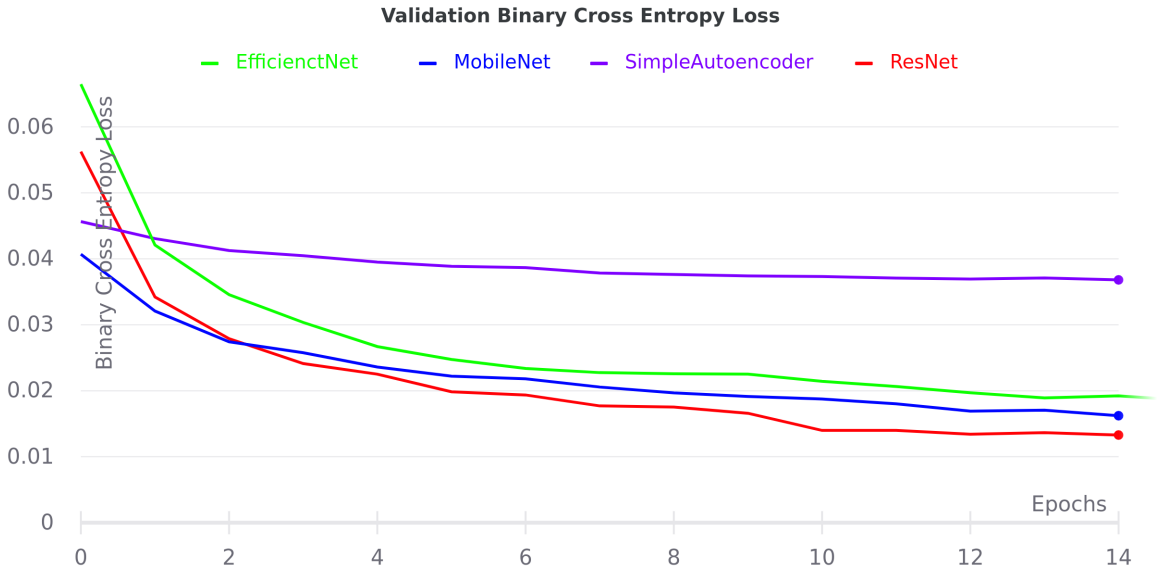

Figure 15: Binary Cross Entropy validation error of training. We experimented with four methods: a Simple Autoencoder, a ResNet, an EfficentNet, and a MobileNet. The normalized binary cross entropy loss was calculated after training for 15 epochs.

5. Further Discussion

When we compare the results of the two approaches, we can easily infer that the techniques of Convolutional Neural Networks are superior since they reliably predict the lines in ambient dynamics with many variations and are more noise resistant. However, we could have made better predictions with the classical learning models if we had used a statistical approach with particle or Kalman filters, like (Liu et al., 2010). Furthermore, these predictions are more consistent since we know exactly what happens at every point of the algorithm, as opposed to the Convolutional Neural models, which operates like a black box. Convolutional Neural Networks continue to require a large amount of data and time to train, but as we have shown, there have been many discoveries to reduce this computational cost with smarter and more powerful architectures. As a result, the distinction between these two approaches can vary depending on the application and its particulars.

6. Conclusion

In this paper, we addressed the problem of lane recognition on the road, which involves autonomous vehicles. We attempted to solve it using traditional techniques such as edge recognition and Hough transforms, as well as modern techniques such as Convolutional Neural Networks.

We compared the methods using images of their predictions, and the results were very satisfactory, as we discovered that even with simple models, we could detect lanes on the road. There is also a clear distinction between classical and modern methods, with the latter producing significantly better results than the former. Both methods, however, have advantages and disadvantages. The choice should take into account the specifics of the problem as well as the environment in which we intend to use them.

References

Canny edge detector - computerphile. URL https://www.youtube.com/watch?v=sRFM5IEqR2w&t=226s.

Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation, 2016.

John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986. doi: 10.1109/TPAMI.1986.4767851.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

Brody Huval, Tao Wang, Sameep Tandon, Jeff Kiske, Will Song, Joel Pazhayampallil, Mykhaylo Andriluka, Pranav Rajpurkar, Toki Migimatsu, Royce Cheng-Yue, Fernando

Mujica, Adam Coates, and Andrew Y. Ng. An empirical evaluation of deep learning on highway driving, 2015.

Guoliang Liu, Florentin Wörgötter, and Irene Markelić. Combining statistical hough transform and particle filter for robust lane detection and tracking. In *2010 IEEE Intelligent Vehicles Symposium*, pages 993–997, 2010. doi: 10.1109/IVS.2010.5548021.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation, 2015a.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation, 2015b.

Kristijan Macek, B. Williams, Sascha Kolski, and Roland Siegwart. A lane detection vision module for driver assistance. 01 2004.

Davy Neven, Bert De Brabandere, Stamatios Georgoulis, Marc Proesmans, and Luc Van Gool. Towards end-to-end lane detection: an instance segmentation approach, 2018.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015. URL http://arxiv.org/abs/1505.04597.

Tsung-Ying Sun, Shang-Jeng Tsai, and V. Chan. Hsi color model based lane-marking detection. In *2006 IEEE Intelligent Transportation Systems Conference*, pages 1168–1172, 2006. doi: 10.1109/ITSC.2006.1707380.

Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020.

Pavel Yakubovskiy. Segmentation models pytorch. https://github.com/qubvel/segmentation_models.pytorch, 2020.

B. Yu and A.K. Jain. Lane boundary detection using a multiresolution hough transform. In *Proceedings of International Conference on Image Processing*, volume 2, pages 748–751 vol.2, 1997. doi: 10.1109/ICIP.1997.638604.