**Lane Detection with Deep Learning**

Project Overview

When a human drive a car, the most common task is to keep the car in the traffic lane. As long as the driver has not been distracted while driving, this task is easy and possible for someone with basic training. On the other hand, for computers this task, keeping the car between its lane’s lines, is not as easy as human.

The reason behind this difficulty is that, computers do not have an ability to understand the environment they are in. The yellow and white marking lanes are not understandable for computers inherently. Thank to various kinds of techniques such as computer vision and deep learning, this task would be teachable and possible for computers to understand.

Why this matter?

Fully autonomous driving relies on the understanding the environment around the vehicle. Various perception modules are used for this understanding, and many pattern recognition and computer vision techniques are applied for these perception modules. Lane detection, which identifies the drivable area on a road, is a major perception technique required for autonomous driving.

Problem Statement

It is true that human beings can identify lane line markings while driving with basic training, but sometimes, based on the number of crashes and accidents, it is understandable that they can have a disadvantage of not always being attentive. Although, it is not that easy for computers to learn identifying lane line markings but after learning the task, there would not be any distractions for them and they have this advantages that the rate for crashes and accidents caused by distraction would be less and computers can take over this task from human driver. Using deep learning, I have used transfer learning and re-train a pre-trained model (PINet[[1]](#footnote-1)). The model is based on a Convolutional Neural Network (CNN) which perform well on image dataset. CNNs work well with images by looking at them in pixel level.

Fundamentals of The Current Model

Deep learning methods show outstanding performance for complex scenes. Among deep learning methods, Convolutional Neural Network (CNN) methods are primarily applied for feature extraction in computer vision.

Lane-mark detection and road region segmentation are the two most popular and common techniques for identifying lanes. The main goal of segmentation is to partition an image into regions. This project uses lane-mark detection for lane identification.

Semantic segmentation methods, the major research area in computer vision, are frequently applied to traffic line detection problems to make inferences about shapes and locations. Some methods use multiclass approaches to distinguish individual traffic line instances. Therefore, even though these methods can achieve outstanding performance, they can only be applied to scenes that consist of fixed numbers of traffic lines. As a solution to this problem***, instance segmentation*** methods are applied to distinguish individual instance by making clusters.

The method that has been used for this project is based on the key points estimation and instance segmentation approach. Key-points are the same thing as interest points. They are spatial locations, or points in the image that define what is interesting or what stand out in the image.

Pros of Using PINet Model

Most traditional traffic line detection methods extract low-level traffic line features using various handcraft features like color, or edges. These low-level features can be combined using a Hough transform. The combined features generate traffic line segment information. These methods are simple and can be adapted to various environments without significant modification. Still, the performance of these methods depends on condition of the testing environment such as lighting and occlusion of the dataset.

The existing methods have certain limitations. The semantic segmentation methods require the labeling or pre-processing at the pixel level for training, which is very time consuming and complicated.

These traditional methods generate and predicts so many unnecessary points since this is how semantic segmentation works. It generated plenty of classified pixel images even though just a few key points of the lane line is enough to recognize the traffic lines.

In addition, existing methods are not adaptive to various environments according to available computing power. To apply them to light systems like embedded boards, the entire architecture should be modified and trained again.

False negatives, traffic lines that the module fails to detect, do not suddenly change the control values, and correct control values can be predicted from other detected traffic lines or previous results. Some methods have higher rates of false positive. In this model, false positive has been reduced competitively since false positives (the wrong detected traffic lines by module) can cause irrecoverable circumstances.

Image Segmentation Methods

There are two types of segmentation techniques

Semantic segmentation: Semantic segmentation is the process of classifying each pixel belonging to a particular label. It doesn't different across different instances of the same object. For example, if there are 2 cars in an image, semantic segmentation gives same label to all the pixels of both cars. Therefore, this technique would not be practical for this project to identify the different lane markings.

A picture containing road, scene, outdoor, street

Description automatically generated

Fig.1. An illustration of a Semantic Segmentation.

Instance segmentation: Instance segmentation differs from semantic segmentation in the sense that it gives a unique label to every instance of a particular object in the image. It means different lane line markings are assigned different colors i.e. different labels (like clusters). With semantic segmentation all of them would have been assigned the same color.

An empty road

Description automatically generated

Fig.2. An illustration of an Instance Segmentation.

Dataset

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Dataset** | **Train** | **Test** | **Resolution** | **Type** |
| Tusimple Dataset | 3,700 | 3,000 | 1280\* 720 | Highway |

Since the dataset (images from Maryland Department of Transportation) is unlabeled, I had to use labeled public dataset (Tusimple[[2]](#footnote-2)) to retrain the model. This contains **3626** video clips of 1 sec duration each. Each of these video clips contains 20 frames of which, the last frame is annotated. These videos were captured by mounting the cameras on a vehicle dashboard.

In this dataset, we have X and Y coordinate of ground truth lane. And at most four lanes are annotated - the two **ego lanes** (two lane boundaries in which the vehicle is currently located) and the lanes to the right and left of ego lanes. All the lanes are annotated at an equal interval of height. And all input images are resized to 512 × 256 size and normalized from values of RGB of 0 ∼ 255 to values of 0 ∼ 1 before the data are fed to the proposed network in both training and testing.

Model Training

For lane detection, a neural network has been trained that consists of several hourglass modules.

An Hourglass Module is an image block module used mainly for pose estimation tasks. The design of the hourglass is motivated by the need to capture information at every scale. The hourglass is a simple, minimal design that has the capacity to capture all of the important features and bring them together to output pixel-wise predictions.

A picture containing engineering drawing

Description automatically generated

Fig.3. An illustration of a single “Hourglass” module. Each box in the figure corresponds to a residual module. The number of features is consistent across the whole hourglass.

Diagram

Description automatically generatedThe network, which has been referred to as the Point Instance Network (PINet), generates points on lanes and distinguishes predicted points into individual instance.

Fig.5. Framework with three main parts. 512\* 256 size input data is compressed by the resizing network; the compressed input is fed to the predicting network, which includes four hourglass modules. Three output branches are applied at the ends of each hourglass block; they predict confidence, offset, and embedding feature. The loss function can be calculated from the output of each hourglass block. By clipping several hourglass modules, required computing resources can be adjusted.

The confidence and offset branches predict exact points of traffic lines; loss functions are applied. The embedding branch generates the embedding features of each predicted point; the embedding feature is fed to the clustering process to distinguish each instance. The loss function of the embedding branch is inspired by an instance segmentation method.

The loss function fitting has been designed based on the contents proposed by SPGN to discriminate each instance of the predicted traffic lines.

Input RGB image size is 512×256; it is fed to the resizing network. This image is compressed to a smaller size (64 × 32) by the sequence of convolution layers in the resizing network; the output of the resizing network is fed to the predicting network.

An arbitrary number of hourglass modules can be included in the predicting network; four hourglass modules are used in this study. All hourglass modules are trained simultaneously by the same loss function. While training, user can choose how many hourglass modules to use according to the computing power, without any additional training.

Chart

Description automatically generated

Fig.6. Details of hourglass block consisting three types of bottle-neck layers: same bottlenecks, down bottlenecks, and up bottlenecks. Output branches are applied at ends of hourglass layers; confidence output is forwarded to the next block.

The resizing network output is fed to the prediction part which predicts the exact points on the traffic lines and the embedding features for instance segmentation. This network includes hourglass modules, each including an encoder, decoder and three output branches as shown in figure 6. Each colored block in the figure is a bottleneck module which have been described in fig 7.

Diagram, schematic

Description automatically generated

Fig.7. Details of bottleneck. The three kinds of bottleneck have different first layers according to their purpose.

As you can see, there are three kinds of bottle neck: same, down and up bottle necks. More details about the structure of these bottlenecks please refer to the article sourced here[[3]](#footnote-3). Each output branch has different channel (Confidence: 1, Offset:2 and Embedding: 4) and the corresponding loss function applied according to the goal of each of them.

Loss Function

For training, four loss function have been applied to each output branch of the hourglass networks. Confidence value determines if the key points of the traffic line exists. Offset value localizes the exact position of the key points predicted previously through the confidence value. And last but not least, the embedding feature is utilized to make clusters by distinguishing key points into individual instance. Therefore, three loss functions are applied to each cell of the output.

* Confidence Loss:

The confidence output branch predicts the confidence value of each cell. If a key point is present in the cell, the confidence value is close to 1 otherwise it is 0.

* Offset Loss:

From the offset branch, PINet predicts the exact location of the key points for each output cell. The output of each cell has a value between 0 and 1.

* Embedding Feature Loss:

The branch is trained to make the embedding feature of each cell closer if the embedding feature are the same in this instance.

Model Implementation

The resizing network reduce the input image’s sizes. At first, the input RGB image size is 512\*256. This network includes three convolution layers which have a filter size of 3\*3, stride 2 and padding size 1. Prelu activation and batch normalization are also applied after each convolution layer.

All images have been resized to 512\* 256 and also, they have been normalized from RGB values of 0 ~ 255 to values of 0 ~ 1 before the data are fed to the network. Various data argumentation such as shadowing, adding noise, flipping, translation, rotation and intensity changing also have been applied.

A sign on the side of a road

Description automatically generatedA car driving on a road

Description automatically generatedA car parked on the side of a road

Description automatically generated

A sign on the side of a road

Description automatically generated

A sign on the side of a road

Description automatically generatedA view of the side of a road

Description automatically generatedA view of the side of a road

Description automatically generated

Shadowing

Rotation

Gaussian Noise

Original Image

Translation

Intensity Changing

Rotation

Source code has been written in Pytorch library and I used transfer learning approach to use the pre-trained model and re-train it on Tusimple dataset.

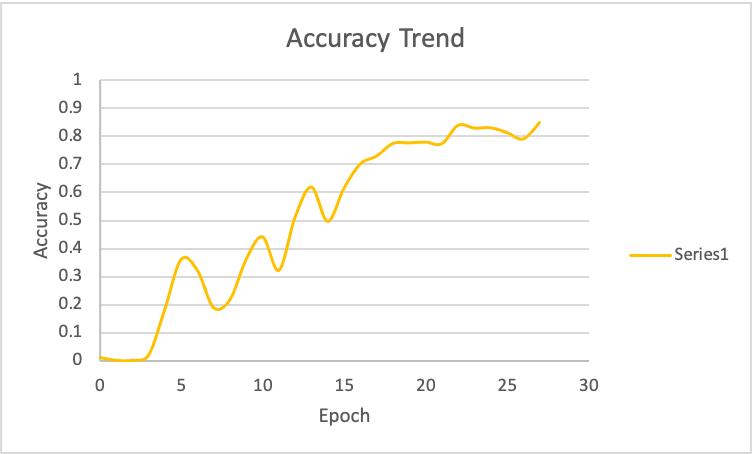
It is a very common approach to use TL (transfer learning) while working with big datasets and complicated networks in deep learning. Training CNNs from scratch is quite rare because of sample size limitations. After re-train the model on the labeled images, I am going to test the model on my unlabeled dataset (video/ images of Maryland roads).

Accuracy is the main evaluation metric of the TuSimple dataset, defined by number of points correctly predicted by the trained module for the given image divided by the number of ground-truth points in the image. The rates False Negative (number of wrongly predicted lanes divided by number of predicted lanes) and False Positive (number of missed lanes divided by number of ground truth lanes) also have been provided.

Result and Conclusion

In this project, the PINet lane detection model has been re-trained with TuSimple dataset. This lane detection method combined key point estimation and point instance segmentation methods. This model achieves high performance and a low rate of false positives; And as we know, false positives could cause major accidents and the lowest the rate the more the safety performance of the autonomous vehicle would be. The table below shows results for this project, and the graph also shows the trend for accuracy per epoch increasing.

|  |  |  |
| --- | --- | --- |
| Accuracy | FP | FN |
| 85.10 % | 0.151 | 0.17 |



The trend below also shows how the training improved within each epoch.

A picture containing cake, toy, train, birthday

Description automatically generatedA picture containing toy, food

Description automatically generatedA picture containing road, scene, outdoor, track

Description automatically generatedA picture containing indoor, road, light, colorful

Description automatically generated

Loss 5.62

Loss 1.22

Loss 3.51

Loss 7.088

A close up of a road

Description automatically generatedA picture containing road, outdoor, track, train

Description automatically generated

Loss 1.03

Loss 1.58

I tested this model on the unlabeled dataset (Images of Maryland roads) and also the attached link is the video from route MD3 in Anne Arundel County that has been tested with this model.

Future Work

Expanding the model to detect even more items at the same time would be very useful for autonomous transportation system. Detecting the lane line markings, vehicles and pedestrian at the same time is a great task for future of image segmentation, lane and object detection.

References:

1. <https://medium.com/analytics-vidhya/detecting-lanes-using-deep-neural-networks-eebf2d9e3603>
2. <https://arxiv.org/abs/2002.06604>

1. https://arxiv.org/abs/2002.06604 [↑](#footnote-ref-1)
2. https://github.com/TuSimple/tusimple-benchmark [↑](#footnote-ref-2)
3. Arxiv.org/pdf/2001.06604.pdf [↑](#footnote-ref-3)