Real-Time Lane Detection for Driving Analysis

Aleman Mihnea, Moga Florian

1. Context

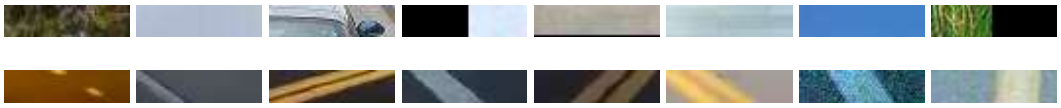
Lane detection enables cars to self-drive and not drive off the surface of the road. In addition to that, lane detection and classification would mean that the road would be a safer place for drivers and pedestrians, if drivers that illegally cross lanes would automatically be identified by Live Traffic Cameras. Schools would benefit from such cameras, making the area safer for pupils.

2. Proposed solution

Our approach consists of a lightweight neural network with a dataset composed of royalty free images downloaded from the internet and image manipulation techniques. The lane detection model will be used for autonomous vehicles.

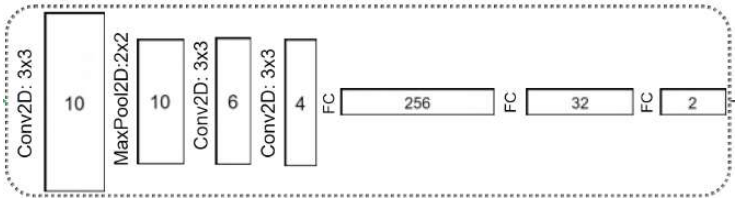
2.1 Dataset creation

We used 2 techniques to create our dataset. Firstly, we reshape our image to the proposed 768 x 1024 size and with a sliding window we capture non overlapping 16 x 64 patches, excluding the patches that are black. Secondly, with a Tkinter app that captures patches with the proposed size, we collected patches from various images and driving videos from the internet. Patches created differ from perfect conditions, to rain, night and strong shadows. Different dataset sizes are used for the binary classification (lane and background), starting from brief 100 images/class. In the end, we roughly captured 2600 patches for each image category. We augmented the lanes with horizontal, vertical flip and also by adding Gaussian Noise.

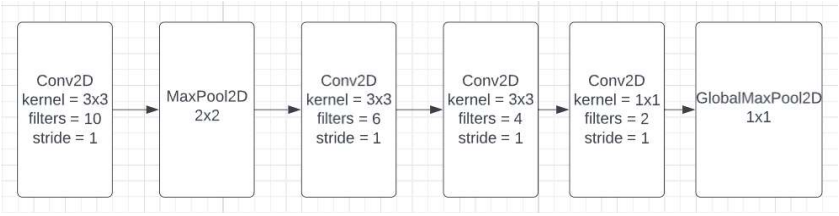


2.2 Model creation

To test our idea we implemented two models in pytorch . The models are shown in the image below. Both models have the same main building block, which consists of 3 convolutional layers and one max pooling layer. For the first model, we chose to use 3 final dense layers for our final prediction. We used the stochastic gradient descent as our optimizer. We obtained the best metrics with a 70/30 split, 89% accuracy and 0.4251 loss with 50 epochs, lr = 10*-3, weight decay of 10**-4 and CrossEntropyLoss criterion.



For our second model we chose to implement a fully convolutional network.We obtained the best metrics with a 80/20 split, 93% accuracy and 0.3175 loss with 25 epochs, lr = 10*-4, weight decay of 10**-3 and CrossEntropyLoss criterion.



We also added BatchNormalization and Dropout layers between all convolutional Layers to help prevent overfitting. To get a probability distribution, we used a softmax activation function on the last layer.

2.3 Feature extraction

Each image classified as a lane is converted to black and white and the mean white pixel value is extracted. This is achieved by clustering the colors in the image with the K-means algorithm with 2-3 clusters, and the cluster that is closest to the 255 value is considered to be the lane, and then, the reference position, $(x_r; y_r)$, is added to find the exact location in the original image.

2.4 Points clustering

We propose a clustering method that groups together points that belong to the same lane boundary. This method utilizes both Euclidean distance and angle as the criteria for grouping points. Unlike the traditional K-means clustering algorithm which groups points based on proximity to a central point, our method groups points based on their proximity to an imagined curve.

As illustrated in Figure 3b, the search starts by selecting a point, p_i in row_{n-1} from the last row of P , follow by calculating the Euclidean distance and angle between p_i and all the points in row_{n-2}, which will be compared with a predefined distance and angle thresholds, as shown in Figure 3a. If d and α are in the desired range, then the point is considered to belong to the same cluster as p_i. The process continues for points in the next rows until the top row is reached as shown in Figure 3c.

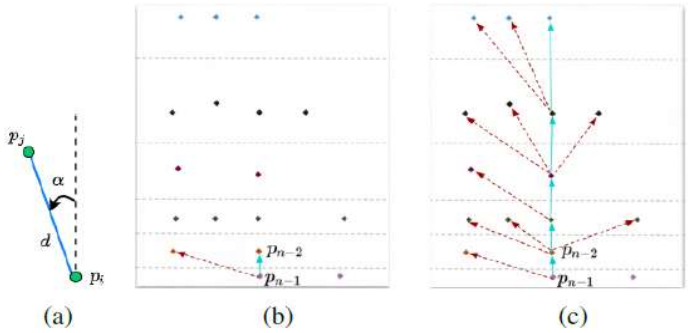


Fig. 3: (3a) The clustering criteria: Euclidean distance, d , and angle, α , between points p_i and p_j for $i \neq j$, (5b) finding a point in row_{n-2} that belongs to the same lane cluster as p_{n-1}, (5c) selected points belonging to the same lane boundary shown by cyan arrow.

After finding the clusters, we refine the results to eliminate any overlapping clusters. This is done by sorting the clusters based on the number of points they contain. The cluster with the most points is considered a strong candidate for representing a lane boundary and is used as a reference for finding other lane boundaries. This is achieved by utilizing information about lane width, L_w, and assuming that lane lines are parallel. The remaining lane boundaries are searched for by defining a search region from the reference cluster, as shown below in Fig 4.

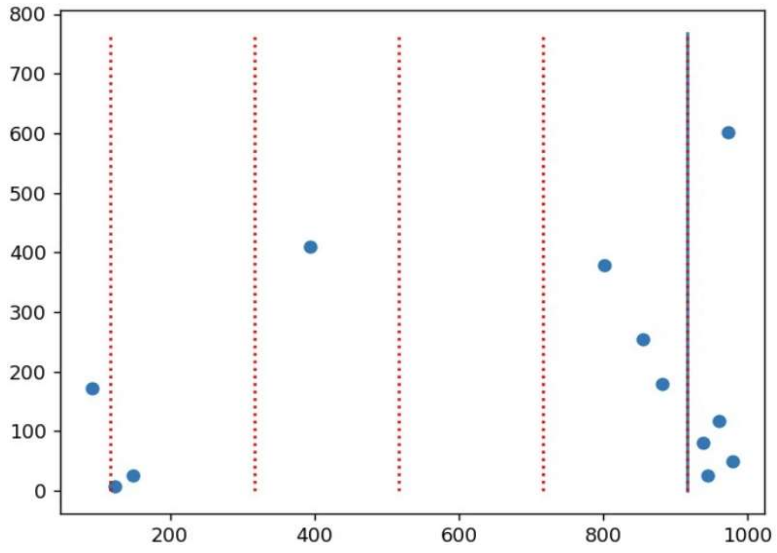


Fig 4.

3. Performance Evaluation

To evaluate our model, we created ground truth data from the KITTI Road dataset (Fig 5.), urban marked section. After that, we fit our resulting clusters with numpy.polyfit 2nd degree polynomials. We tested our curve against the ground truth with a hausdorff distance. If a cluster sits with a threshold of 15 pixels (the width of a double lane) against the ground truth, it is considered a TP, else a FP. FN are considered lanes that are not identified.

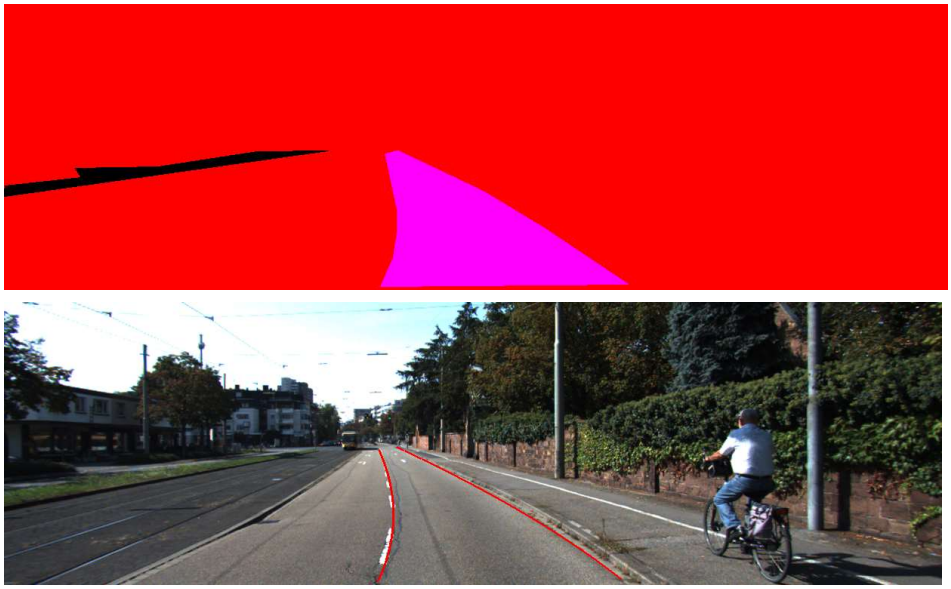


Fig 5.

For our first model, our result fell short due to the lack of right lane marking and from our clustering based on a bottom up approach, but mostly to the difficult conditions present in the dataset. Our model would classify strong shadows as lanes. As for the run time efficiency, we tested it on several images and this solution isn't suited for real time inference, an image being processed for several seconds, consisting of 768 patches. The training took about 30 minutes.

TP	FP	FN
0.274	0.370	0.355

For the second model, the results were consistently better, not only the training was faster,because there were fewer parameters, but also the accuracy improved. With a 8 core Ryzen 9 4000 Series processor, we manage to prepare this model for live inference, having a speed of 5 fps. Also since this model contained patch classification, it would classify wrong certain white parts of random objects, but they would be taken out of the calculation due to clustering and picking the region of interest. The training took about 10-15 minutes.

TP	FP	FN

0.404	0.328	0.268

I also trained and tested MobileNetv2. Although the accuracy was better, the training took significantly longer due to limited resources and the size of the network. While training, this model took about 3 seconds per batch of 32 images, and about 10 sec to predict the frame for live inference.

4. Literature Review

In our study, we evaluated several lane detection approaches and chose a proposed method due to its lightweight nature and ability to perform live inference. While there are other approaches that may provide more accurate results, they are typically slower and more computationally expensive. For example, instance segmentation models or transformers-based models can provide more precise road segmentation, but may not be suitable for real-time applications.
For example, LLDNet uses a lightweight encoder and decoder architecture [2], which can detect lanes on defected roads, especially ones with blurry lane lines, no lane lines, and cracked pavements. Although it manages to predict at 30 fps, it is used together with 32 GB RAM, Intel core i9-11900k @ 3.50 GHz CPU processor, and NVIDIA Geforce RTX 3080Ti graphics card, which we will never get this much computation power in a car.

Our decision to use the proposed method was based on the specific needs of our application, which required a lightweight and fast approach for live inference. That is why we went for a similar approach that is used in [1].To minimize the effect of environmental factors on the detector they developed a data driven approach (using CNNs) for feature extraction and a fast post-processing block for even more filtering out the outliers. They also achieved a very decent performance, 29 fps on an Intel NUC, Intel Core i7-6770HQ with 8 cores.

5. Conclusion

In conclusion, it is worth noting that our methodology has yielded encouraging outcomes for the purpose of real-time lane detection. Given that it does not necessitate a significant amount of computational power, it is well-suited for deployment in vehicles. It is imperative to ensure that the model is trained in accordance with the specific environmental conditions it is expected to operate in. For instance, our model was trained utilizing images that incorporated various background elements, including trees, houses, and other vehicles. Consequently, the placement of the camera on the vehicle may potentially impact the accuracy of the model. In some instances, optimal results may be achieved if the camera is positioned in front of the car's grill, as the camera's field of view may primarily comprise the road's surface. Moreover, capturing images while driving may facilitate the detection of an improved region of interest and the road's angle.

6. References

Tesfamchael Getahun, Ali Karimoddini, Priyantha Mudalige (2021). **A Deep Learning Approach for Lane Detection | IEEE Conference Publication. [1]**

Md. Al-Masrur Khan, Md Foysal Haque, Kazi Rakib Hasan, Samah H. Alajmani, Mohammed Baz , Mehedi Masud and Abdullah-Al Nahid. (2022) **LLDNet: A Lightweight Lane Detection Approach for Autonomous Cars Using Deep Learning. [2]**

Himanshu Rawlani. (2020) **Understanding and implementing a fully convolutional network (FCN).** [3]

Rudra N. Hota, Shahanaz Syed, Subhadip Bandyopadhyay, P. Radhakrishn. (2009) **A Simple and Efficient Lane Detection using Clustering and Weighted Regression** [4]

Grigory Serebryakov, Satya Mallick. (2020) **Fully Convolutional Network For Image Classification on Arbitrary Sized Image [5]**