People's Democratic Republic of Algeria

The Ministry of Higher Education and Scientific Research

University of Mohamed El-Bachir El-Ibrahimi - Bordj Bou Arreridj

Faculty of Sciences and Technology

Department Electronic’MCIL’

Thesis

Presented to get

The master’s diploma

Faculty: Science and technology

Specialty: Industrial Electronic

By

- *Hadj Said Yahia*
- *Chihani Rostom*

Entitled

Lane Line Detection Using a Deep Learning Model

Supported on: …………………………

Before a jury composed of:

Last name & First name	Grade	Quality	Establishment
BOUDRAA Khadija	Dr.	President	Univ-BBA
SID AHMED Soumia	Dr.	supervisor	Univ-BBA
MESSALI Zoubeida	PROF.	examiner	Univ-BBA

College year 2023/2024

Acknowledgments

First, we would like to praise Allah the Almighty, the Most Gracious, and the Most Merciful for His blessing given to us during our study and in completing this thesis.

We would like to express our gratitude and sincere thanks to our principal **supervisor *Dr. SID AHMED Soumia***, for her guidance, advice, feedback, and continuous support throughout the whole project.

We are very much grateful to all teachers of the University of Mohamed Elbachir El Ibrahimi, who provided us with the tools necessary for the success of our university studies during our course.

Finally, yet importantly, we would like to thank our families for supporting us throughout our life. Without their love and encouragement, we would not have finished this thesis.

Summary:

Deep Learning, within the field of Artificial Intelligence, has emerged as a prominent field renowned for its capacity to discern complex patterns and features directly from raw data. Our project is centered on exploring techniques for detecting lane lines in self-driving cars, leveraging deep learning methodologies, specifically through the application of FCN (Fully Convolutional Network). We aim to conduct a comparative analysis between deep learning approaches and traditional computer vision methods utilizing OpenCV, shedding light on the strengths and limitations of each approach in the context of lane line detection for autonomous vehicles.

Sommaire :

Le Deep Learning, dans le domaine de l’intelligence artificielle, est devenu un domaine de premier plan réputé pour sa capacité à discerner des modèles et des caractéristiques complexes directement à partir de données brutes. Notre projet est centré sur l'exploration de techniques de détection de lignes de voie dans les voitures autonomes, en tirant parti des méthodologies d'apprentissage profond, notamment grâce à l'application du FCN (Fully Convolutional Network). Notre objectif est de mener une analyse comparative entre les approches d'apprentissage profond et les méthodes traditionnelles de vision par ordinateur utilisant OpenCV, mettant en lumière les forces et les limites de chaque approche dans le contexte de la détection de lignes de voie pour les véhicules autonomes.

ملخص:

لقد برز التعلم العميق، ضـــمن مجال الذكاء الاصـــطناعي، كمجال بارز يشـــتهر بقدرته على تمييز الأنماط والميزات المعقدة مباشـرة من البيانات الأولية. يتمحور مشـروعنا حول اسـتكشـاف تقنيات الكشـف عن خطوط الطرقات في السـيارات ذاتية القيادة، والاستفادة من منهجيات التعلم العميق، وتحديدًا من خلال تطبيق FCN (الشبكة التلافيفية بالكامل). نحن نهدف إلى إجراء تحليل مقارن بين أساليب التعلم العميق وطرق الرؤية الحاسوبية التقليدية باستخدامOpenCV ، وتسليط الضوء على نقاط القوة والقيود في كل نهج في سياق اكتشاف خط المسار للمركبات ذاتية القيادة.

TALE OF CONTENTS:

LIST OF FIGURES

LIST OF TABLES

Acronyms And Abbreviations

AI : Artificial Intelligence

CNN : Artificial Neural Network

RNN: Recurrent Neural Networks

FCN : Convolutional Networks

DL : Deep Learning

ML : Machin Learning

FC : Fully Connected

GPU : Graphic Processor Unit

MSE: Mean Squared Error

NN : Neural Network

GAN : Generative Adversarial Network

GENERAL INTRODUCTION

In the fast-changing world of transportation technology, the combination of Artificial Intelligence (AI) and Deep Learning is becoming very important, reshaping the future of mobility. As we delve into the complexities of lane line detection, it becomes apparent that harnessing the capabilities of AI is crucial for navigating the intricacies of modern roadways. Deep Learning, a subset of AI, is at the forefront of this change. It offers sophisticated methods for machines to perceive, interpret, and respond to visual cues with remarkable accuracy and efficiency.

Deep Learning has attracted significant attention for its capacity to identify complex patterns and features directly from raw data. Leveraging the complex architecture of neural networks NN, Deep Learning algorithms can discern subtle nuances within visual inputs, outperforming conventional techniques in both performance and adaptability. In the context of lane line detection, this paradigm shift has opened doors to unprecedented levels of precision and reliability, fueling the quest for safer and more efficient transportation systems.

As we embark on the journey of exploring lane line detection using Deep Learning, it is paramount to understand the nuances of various methodologies and architectures. In this thesis, We conduct a detailed implementation and comparative analysis of a traditional technique using OpenCV alongside a Fully Convolutional Networks model. Through meticulous experimentation and evaluation, we aim to uncover the detailed characteristics of each method, highlighting their respective strengths and limitations. By deepening our understanding of Deep Learning techniques, this work aims to demonstrate how Deep Learning models for lane line detection can surpass conventional methods.

To achieve this goal, we have structured our thesis into three chapters:

Chapter 1: In this chapter, we will discuss the Deep Learning field and focus on the CNN architecture.

Chapter 2: In this chapter, we aim to conduct a comparative analysis between deep learning approaches and traditional computer vision methods utilizing OpenCV, shedding light on the strengths and limitations of each approach in the context of lane line detection for autonomous vehicles.

Chapter 3: This chapter contains the comparative result’s implementation of our CNN architecture, using the FCN (fully convolutional network) model.

CHAPTER 1

1.1. Introduction

Deep learning, a subset of Artificial Intelligence (AI), has gained prominence for its ability to utilize deep neural networks in tasks like image recognition and speech processing, revolutionizing the capabilities of computers in various domains. We further break down ML into its different types, including supervised learning, where algorithms are trained on labeled data to make predictions, and unsupervised learning, which involves finding hidden patterns in unlabeled data. The chapter also provides a brief overview of neural networks NN that simulate the learning mechanisms of biological organisms and covers specifically Convolutional Neural Networks (CNNs), which are designed to process grid-like data structures such as images.

1.2. Artificial intelligence

Artificial Intelligence, often abbreviated as AI, refers to the simulation of human intelligence processes by machines, typically computer systems. This field encompasses various technologies that enable machines to perform tasks that typically require human intelligence, such as visual perception, speech recognition, decision-making, and language translation. AI systems are designed to learn from data, adapt to new inputs, and perform tasks with minimal human intervention. The goal of AI is to create machines that can mimic human cognitive functions and perform tasks more efficiently and accurately. AI has applications in a wide range of industries, including healthcare, finance, transportation, and entertainment, and continues to advance rapidly with ongoing research and development efforts.[1]

1.3. Machine learning

Machine learning is a branch of artificial intelligence that focuses on developing algorithms and statistical models to enable computers to learn from and make predictions or decisions based on data without being explicitly programmed. It involves the use of algorithms that iteratively learn from data, allowing computers to find hidden insights without being explicitly programmed where to look. Machine learning is widely used in various applications such as image and speech recognition, medical diagnosis, financial forecasting, and recommendation systems. It encompasses different types of learning, including supervised learning, unsupervised learning, and reinforcement learning, each serving different purposes in training models. Overall, machine learning plays a crucial role in automating tasks and making data-driven decisions in various fields.[2]

1.3.1. Supervised learning

Supervised learning is indeed a machine learning paradigm where algorithms are trained on labeled datasets. These labels guide the algorithm in learning the relationship between the input features and the target outcome. For instance, in weather prediction, a supervised learning model would analyze historical data labeled with whether it rained or not, alongside input

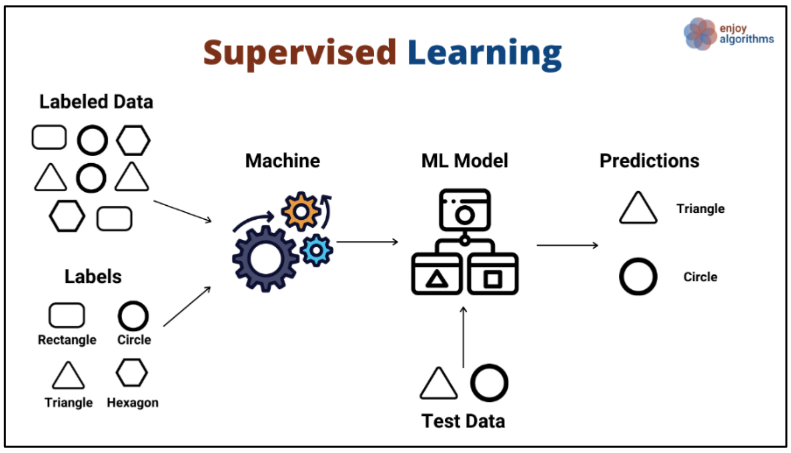


Figure 1. Supervised Learning process.

features like temperature, time, season, and atmospheric pressure. The model discerns patterns in this data to predict future rainfall. The process is often depicted in diagrams, such as the one mentioned (Figure 1).[3]

1.3.2. Unsupervised learning

Unlike supervised learning which relies on labeled data, unsupervised learning deals with unlabeled data. This means the data points don't have predefined categories or classifications. The most common use case for unsupervised learning algorithms is in clustering analysis, where the algorithm learns hidden patterns and groups in data that are not explicitly labeled.[4]

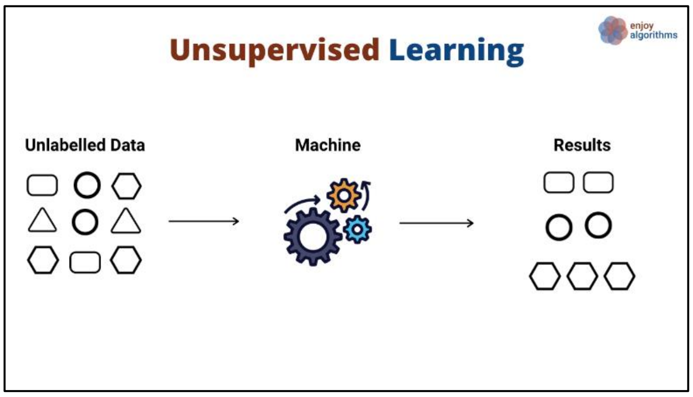


Figure 2. Unsupervised Learning process.

1.3.3. Supervised learning methods

Supervised learning methods are typically divided into two main categories, each suited for different types of prediction tasks:

- **Classification:** This method is used when the output variable is categorical. The goal is to predict which category or class the new observation belongs to. Examples of classification problems include determining whether an email is spam or not spam, or diagnosing whether a patient is sick or healthy based on medical test results.
- **Regression:** This method is applied when the output variable is a real or continuous value.It aims to predict a quantity, such as the price of a house based on its features, the amount of rainfall from weather data, or a person’s height based on genetic and dietary information.

Both methods rely on labeled datasets to learn the mapping from input variables to the target variable, enabling the algorithm to make accurate predictions on new, unseen data.

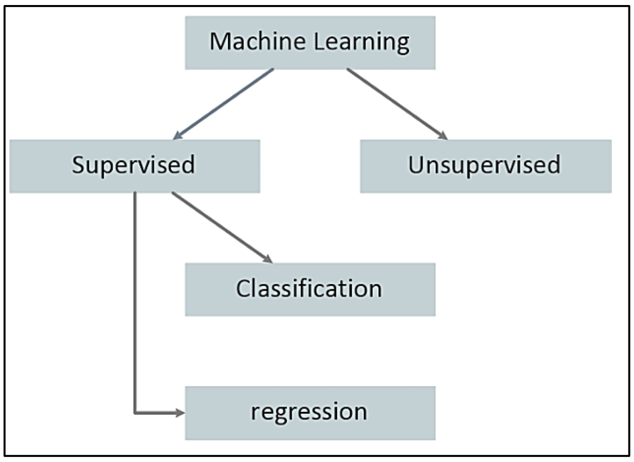


Figure 3. Hierarchy Machine Learning algorithms.

1.4. Deep learning

Deep Learning refers to a subset of machine learning methods that are based on artificial neural networks. Its algorithms are designed to model and represent complex patterns in data through multiple layers of interconnected nodes. These networks are capable of learning and making decisions in a way that mimics the human brain's functioning. Deep learning has been particularly successful in tasks such as image and speech recognition, natural language processing, and autonomous driving. It requires large amounts of data for training and is known for its ability to automatically discover intricate patterns within the data. Deep learning has revolutionized various industries by enabling advancements in areas like healthcare, finance, and technology. [5].

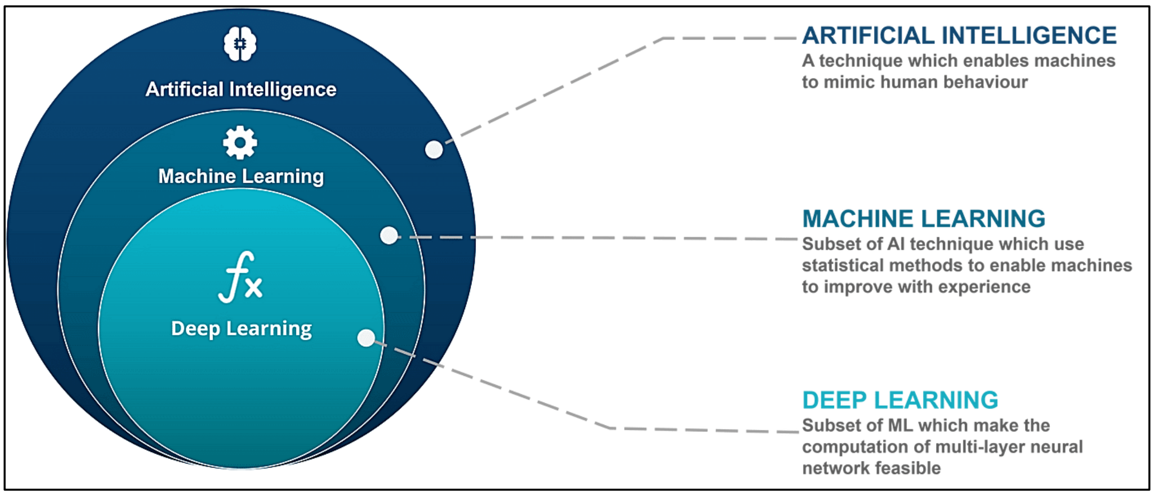

Artificial Intelligence
Machine Learning
Deep Learning
ARTIFICIAL INTELLIGENCE
A technique which enables machines to mimic human behaviour
MACHINE LEARNING
Subset of AI technique which use statistical methods to enable machines to improve with experience
DEEP LEARNING
Subset of ML which make the computation of multi-layer neural network feasible

Figure 4. Relationship between AI, ML and DL

1.4.1. Choosing between Machine Learning and Deep Learning

When deciding between Machine Learning (ML) and Deep Learning (DL), it is essential to consider their capabilities and applications. ML is a broader category of algorithms that identify patterns and make decisions based on data, while DL, a subset of ML, uses deep neural networks to extract features from complex data. Both ML and DL have been successfully applied in various fields, such as cybersecurity for threat event detection [6] and healthcare for brain tumor identification using MRI scans. DL, with its deep neural networks, has shown remarkable success in tasks like image recognition and natural language processing. Ultimately, the choice between ML and DL depends on the complexity of the data and the specific requirements of the task at hand, with DL being more suitable for intricate data structures and tasks requiring high levels of abstraction.

1.5. Neural networks

1.5.1. Neuron from Biology to AI

The transition from biological neurons to artificial neural networks marks a significant milestone in the development of artificial intelligence. Modeled after the intricate network of neurons in the human brain, artificial neurons, or perceptron, serve as the building blocks of these systems.

While biological neurons communicate through electrochemical signals, artificial neurons use mathematical functions to process and transmit information. This transformation involves abstracting the principles of neural computation, such as synaptic connections and signal propagation, into computational algorithms and architectures. Despite simplifications and abstractions, artificial neural networks emulate the essential features of their biological counterparts, enabling them to perform complex cognitive tasks and learn from data.[7]

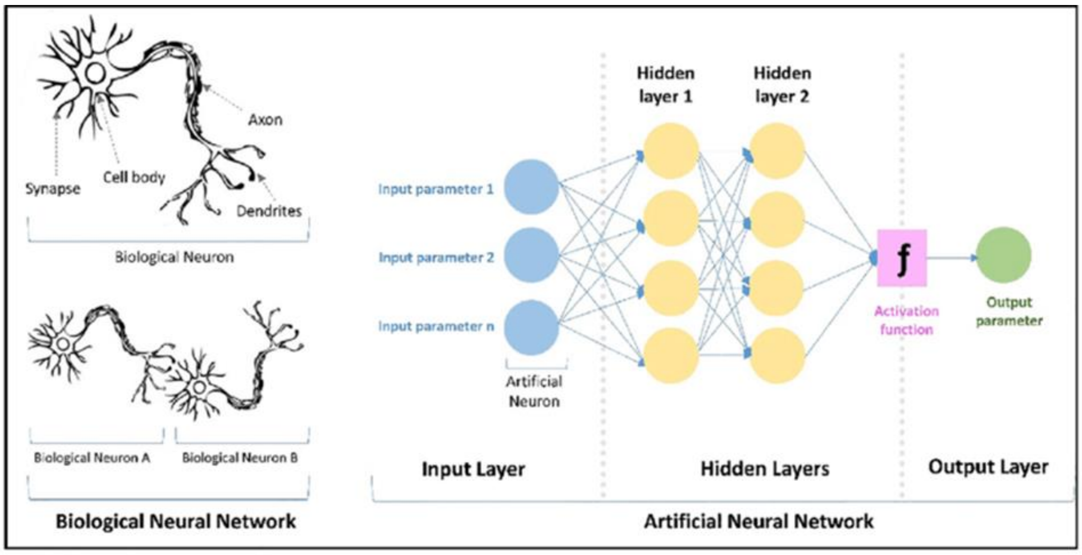


Figure 5. Bio neural network vs Ai neural network.

Artificial neural networks typically consist of an input layer, one or more hidden layers, and an output layer, with each layer comprising interconnected neurons. During training, the network adjusts its internal parameters, such as weights and biases, to minimize the difference between predicted and actual outputs. This process of learning enables neural networks to generalize patterns from training data and make accurate predictions on unseen examples, making them powerful tools for tasks such as image recognition, speech recognition, and natural language processing.

1.5.2. Neural network parameters and hyper-parameters

Neural networks are characterized by two types of parameters weights and biases. These parameters are essential for the network to learn and make predictions effectively.

1.5.2.1.Weights

Weights represent the strength of connections between neurons in adjacent layers. Each connection between neurons is associated with a weight, which determines the impact of the input on the output of a neuron. During the training process, these weights are adjusted iteratively to minimize the difference between predicted and actual outputs. The values of weights determine the network's ability to capture patterns and relationships in the data.[8]

1.5.2.2.Biases

Biases are additional parameters added to each neuron in a layer, independent of input. They allow neural networks to capture patterns that might not be directly related to the input data. Biases essentially provide flexibility to the model, enabling it to fit more complex functions. Like weights, biases are also adjusted during training to optimize the network's performance. [9]

In addition to parameters, neural networks also have hyperparameters, which are settings that govern the architecture and behavior of the network. These hyperparameters are set before training and are not adjusted during the learning process. Some common hyperparameters include:

1.5.2.3.Learning Rate

The learning rate controls the step size of parameter updates during training. It determines how much the parameters are adjusted in each iteration based on the gradient of the loss function. A higher learning rate may lead to faster convergence but risks overshooting the optimal solution, while a lower learning rate may result in slower convergence but better stability.

1.5.2.4.Batch size

Batch size refers to the number of input samples processed in one forward and backward pass. Different batch sizes can impact training accuracy, runtime, and model performance [10]. While larger batches are traditionally thought to enhance model performance, recent studies suggest that smaller batch sizes can be more beneficial, especially when dealing with data exhibiting global similarities and local differences, like electronic health records and medical imaging. Research indicates that smaller batches can improve loss performance, capture more meaningful information in latent spaces, and lead to better classification and regression results[11]. Additionally, experiments show that increasing batch size does not always guarantee higher accuracy in image classification tasks. Therefore, selecting an optimal batch size is crucial in training CNNs to achieve the desired balance between accuracy and efficiency.

1.5.2.5.Number of Layers and Neurons

The architecture of the neural network, including the number of layers and neurons in each layer, is a crucial hyperparameter. Deeper networks with more layers can capture complex relationships in the data, but they also require more computational resources and are prone to overfitting. Finding the right balance between the depth and width of the network is essential for optimal performance.

1.5.2.6.Activation Functions

Activation functions introduce non-linearity to the network, enabling it to learn complex mappings between inputs and outputs. Common activation functions include ReLU (Rectified Linear Unit), sigmoid, and tanh. Choosing the appropriate activation function can significantly impact the network's ability to model complex data distributions.[12]

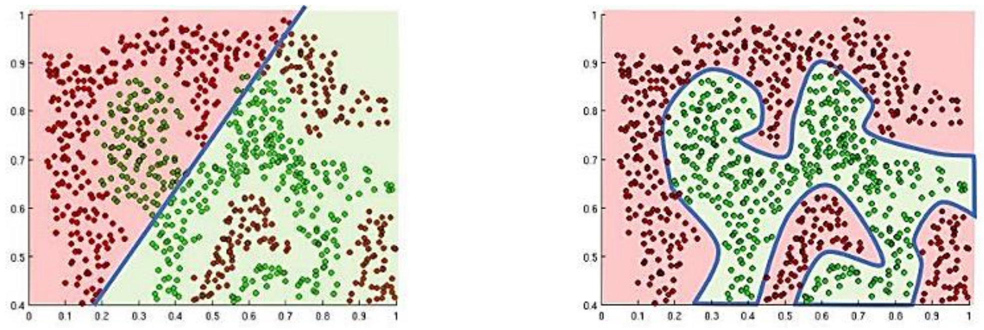


Figure 6. Comparison between linear and non-linear activation function.

1.5.2.7.Type of activation function

1.5.2.7.1. ReLU Function

ReLU stands for (Rectified Linear Unit). Although it gives the impression of a linear function, ReLU has a derivative function and allows for backpropagation while simultaneously making it computationally efficient[13]. The main catch here is that the ReLU function does not activate all the neurons at the same time. The neurons will only be deactivated if the output of the linear transformation is less than 0.

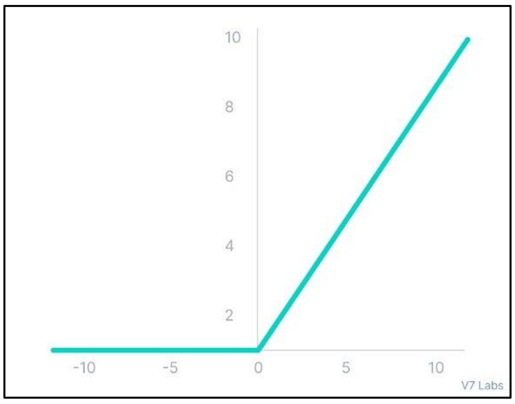


Figure 7. ReLU activation function.

$$\boldsymbol{ReLu(x) = max(x, 0)} \quad (1)$$

The advantages of using ReLU as an activation function are as follows:

- Since only a certain number of neurons are activated, the ReLU function is far more computationally efficient when compared to the sigmoid and tanh functions.

- ReLU accelerates the convergence of gradient descent towards the global minimum of the loss function due to its linear, non-saturating property.[13]

1.5.2.7.2. Sigmoid / Logistic

Sigmoid / Logistic Activation Function This function takes any real value as input and outputs values in the range of 0 to 1. The larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to 0.0, as shown below.

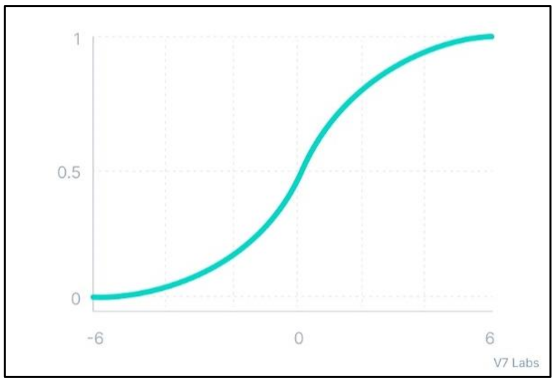


Figure 8. Sigmoid/ Logistic activation function.

$$\boldsymbol{sigmoid(x) = \frac{1}{1+e^{-x}}} \quad (2)$$

Here’s why the sigmoid/logistic activation function is one of the most widely used functions:

- ✓ It is commonly used for models where we have to predict the probability as an output. Since the probability of anything exists only between the range of 0 and 1, sigmoid is the right choice because of its range.
- ✓ The function is differentiable and provides a smooth gradient, i.e., preventing jumps in output values. This is represented by an S-shape of the sigmoid activation function.

1.5.2.8.Loss Function

Remember, our network's behavior is determined by all its weights and biases. The weights represent the strength of connections between each neuron in one layer and each neuron in the next, while each bias indicates whether its neuron tends to be active or inactive.

To start things off, initialize all the weights and biases with random numbers. This network will perform horribly on the given training example since it’s just doing something random. Anyway, it will give a prediction. A terrible one, as the model has no idea yet if the prediction is good, or how to measure how good it is.

To assess the performance of a neural network model and enhance its accuracy, we use a loss function. This function measures the disparity between the predicted values and the actual values, offering a way to gauge the quality of the predictions. By minimizing the loss function, we can steer the model toward making more precise predictions as it evolves.[14]

$$\boldsymbol{Loss\left(y(x), a^L(x)\right)} \qquad (3)$$

1.5.2.9.Types of Loss Function

Two main types of loss functions correlate to the two major types of neural networks, regression, and classification loss functions:

1.5.2.9.1. Regression Loss Function

Regression models deal with predicting a continuous value. For example, given data such as floor area, number of rooms, and size of rooms, a regression model can predict the price of a house. One of the most commonly used loss functions in regression problems is called "Mean Squared Error" (MSE), also known as "Quadratic Cost".

Mean Squared Error measures the average squared difference between the predicted values and the actual values. It is defined by the formula:

$$\boldsymbol{MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2} \qquad (4)$$

Where y_i represents the actual value and $\hat{y}_i$ represents the predicted value, and *n* is the number of data points. MSE penalizes larger errors more than smaller ones, making it a useful measure for regression models to minimize during training.

1.5.2.9.2. Classification loss function

Classification problems involve predicting a discrete class output. This involves dividing the dataset into distinct and unique classes based on different parameters so that a new and unseen record can be categorized into one of these classes. The most widely used loss function for this type of learning method is the "Binary Cross-Entropy" loss function:

$$\boldsymbol{BCE} = -\frac{\boldsymbol{1}}{\boldsymbol{n}} \sum_{i=1}^{n} [\boldsymbol{y_i} \ln(\boldsymbol{a_i^L}) + (\boldsymbol{1} - \boldsymbol{y_i}) \ln(\boldsymbol{1} - \boldsymbol{a_i^L})] \tag{5}$$

Where n is the total number of items in the training data, the sum is overall training inputs x, y_i is the corresponding desired output, and a_i^L represents the predicted output from the final layer L of the network. Binary Cross-Entropy measures the performance of a classification model whose output is a probability value between 0 and 1, guiding the model to minimize the difference between the predicted probabilities and the actual class labels.[15]

1.5.3. Training the neural network

Simply informing the neural network model what a terrible job it’s doing isn’t very helpful. Instead, it's essential to guide the model in adjusting its weights and biases to enhance its predictions. These adjustments must be directed towards reducing the loss function. The key question is: how can we determine the inputs that will minimize the loss function's value? To address this, we implement gradient descent on the loss function, which systematically modifies the inputs to achieve the lowest possible cost.

1.5.3.1.Gradient Descent (GD)

This is the fundamental algorithm for optimizing neural networks. It calculates the gradient of the loss function for each trainable parameter (weights and biases). The gradient essentially points in the direction of the steepest descent in the loss function's landscape.

The parameters are then updated in the opposite direction of the gradient by a small amount (learning rate), moving the network closer to a minimum of the loss function.[16]

$$\boldsymbol{w} = \boldsymbol{w'} - \boldsymbol{\gamma} \frac{\boldsymbol{d(C)}}{\boldsymbol{d(w)}} \tag{6}$$

$$\boldsymbol{b} = \boldsymbol{b'} - \boldsymbol{\gamma} \frac{\boldsymbol{d(C)}}{\boldsymbol{d(b)}} \tag{7}$$

where γ is a small, positive parameter (known as the learning rate), ($\frac{d(C)}{d(w)}$) partial derivative with respect to weight, ($\frac{d(C)}{d(w)}$) partial derivative with respect to bias. Also, w' and b' represent old weight and bias, w and b indicate updated weight and bias.

1.5.3.2.Stochastic Gradient Descent (SGD)

Indeed, in practical scenarios, it's computationally intensive to calculate the gradient descent for numerous layers across the full training set during each iteration of the algorithm. To make this process more efficient, a technique known as Batching is employed. Batching involves dividing the training set into smaller, manageable batches of data. These batches are then used to feed the neural network (NN). The gradient computation is not performed on the entire dataset but rather on these smaller subsets, denoted by m. This approach not only accelerates the training process but also helps in optimizing the use of memory resources and can contribute to better generalization of the model.

$$\boldsymbol{w} = \boldsymbol{w}' - \frac{\gamma}{m}\sum_j \frac{d(C_j)}{d(\boldsymbol{w})} \tag{8}$$

$$\boldsymbol{b} = \boldsymbol{b}' - \frac{\gamma}{m}\sum_j \frac{d(C_j)}{d(\boldsymbol{b})} \tag{9}$$

1.5.3.3.Backpropagation

Backpropagation is a fundamental algorithm in neural networks that calculates the gradient of the loss function with respect to the weights and biases of the neurons. The process of training a neural network involves finding the optimal set of weights that result in the best fit between the network's output and the actual data. Backpropagation facilitates this by linking the loss function's outcome to the weight parameters. Here's a simplified breakdown of the process:

- If the loss function's value decreases with the current weights, it indicates that the gradient direction is beneficial.

- Conversely, if the loss function's value increases, the gradient direction is reversed.

- This iterative process continues until the loss function reaches a minimum value or no further reduction is possible (convergence).

Upon convergence, the neural network has located a point on the loss function where any increase or decrease in weight values leads to an increase in the loss function. This point is typically a local minimum, where the network's predictions are as close as possible to the target values given the current model architecture and data.

1.5.4. Convolutional Neural Network (ConvNet/CNN)

A Convolutional Neural Network (CNN), also known as ConvNet, is a specialized type of deep learning algorithm mainly designed for tasks that necessitate object recognition, including image classification, detection, and segmentation. CNNs are employed in a variety of practical scenarios, such as autonomous vehicles, security camera systems, and others. [17]

The convolutional neural network is made of different parts that help the CNNs mimic how the human brain operates to recognize patterns and features in images. This section dives into the definition of each one of these parts.

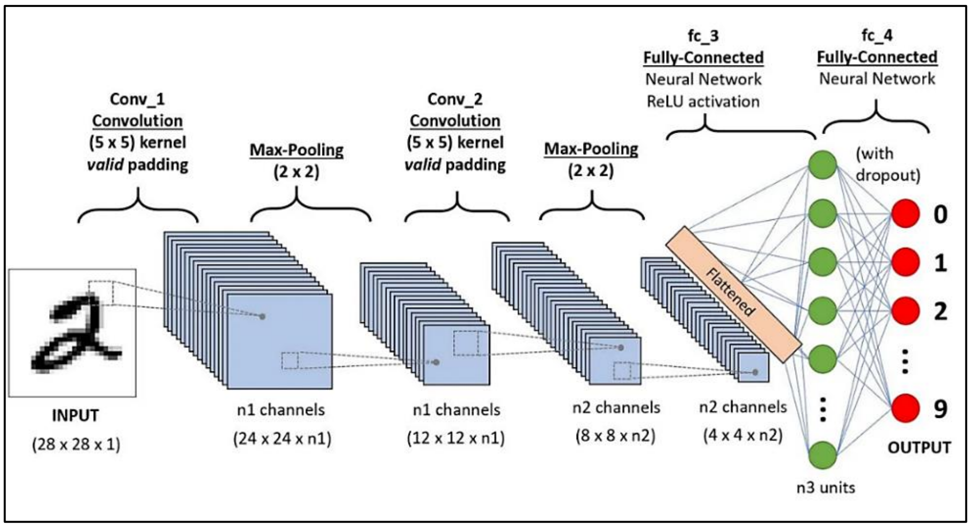


Figure 9. Architecture of the CNNs applied to digit recognition.

1.5.4.1.Convolution layers

This is the first building block of a CNN. As the name suggests, the main mathematical task performed is called convolution, which is the application of a sliding window function to a matrix of pixels representing an image. The sliding function applied to the matrix is called kernel or filter, and both can be used interchangeably.

In the convolution layer, several filters of equal size are applied, and each filter is used to recognize a specific pattern from the image, such as the curving of the digits, the edges, the whole shape of the digits, and more. Put simply, in the convolution layer, we use small grids (called filters or kernels) that move over the image. Each small grid is like a mini magnifying glass that looks for specific patterns in the photo, like lines, curves, or shapes. As it moves across the photo, it creates a new grid that highlights where it found these patterns. For example, one filter might be good at finding straight lines, another might find curves, and so on. By using several different filters, CNN can get a good idea of all the different patterns that make up the image.[18]

The more convolution layers the network has, the better the layer is at detecting more abstract features.

1.5.4.2. Hyper-parameters

In Convolutional Neural Networks (CNNs), hyperparameters play a crucial role in shaping the model's performance. Let's explore some of the key hyperparameters:

1.5.4.2.1. Kernel Size (Filter Size)

The kernel size determines the receptive field of each convolutional layer. A larger kernel size allows for the capture of more global features, thus providing a broader overview of the input data, whereas a smaller kernel size targets more localized patterns, offering detailed insights into specific regions of the input. Commonly utilized kernel sizes include 3x3, 5x5, and 7x7, each offering a balance between feature granularity and computational efficiency.

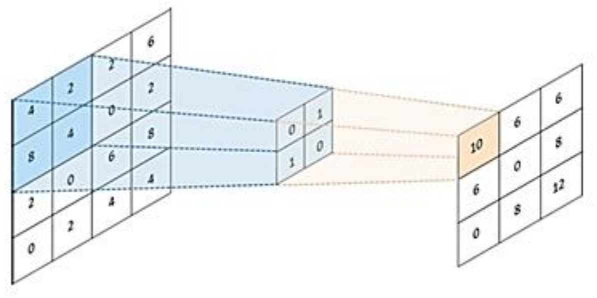


Figure 10. An illustration of the convolution of a 2 x 2 kernel moving over a 4 x 4 image, resulting in a 3 x 3.

1.5.4.2.2 Number of Filters (Channels)

The number of filters in a convolutional layer defines the depth of the output feature maps. More filters allow the network to learn complex features. However, increasing the number of filters also increases computational cost.

1.5.4.2.3. Stride

The stride specifies the step size when sliding the kernel over the input. Larger strides reduce the spatial dimensions of the output feature maps. Smaller strides preserve more spatial information.

1.5.4.2.4. Padding

The padding adds extra pixels around the input to maintain spatial dimensions after convolution. Common types include "same" (zero-padding) and "valid" (no padding).

1.5.4.3. Pooling layer

The goal of the pooling layer is to pull the most significant features from the convoluted matrix. This is done by applying some aggregation operations, which reduce the dimension of the feature map (convoluted matrix), hence reducing the memory used while training the network. Pooling is also relevant for mitigating overfitting. The most common aggregation functions that can be applied are:

- Max pooling, which is the maximum value of the feature map
- Average pooling is the average of all the values.

Below is an illustration of each of the previous examples:

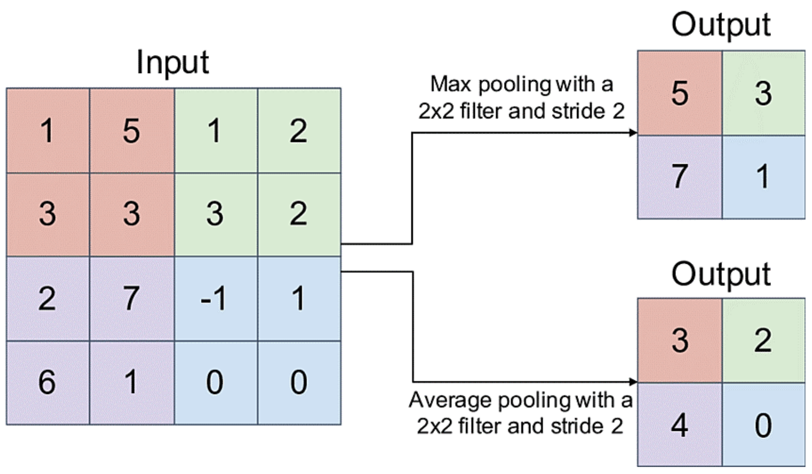

Input
1 5 1 2
3 3 3 2
2 7 -1 1
6 1 0 0
Max pooling with a 2x2 filter and stride 2
Output
5 3
7 1
Average pooling with a 2x2 filter and stride 2
Output
3 2
4 0

Figure 11. Application of max and average pooling with a stride of 2 using 2x2 filter

Also, the dimension of the feature map becomes smaller as the pooling function is applied. The last pooling layer flattens its feature map so that it can be processed by the fully connected layer.

1.5.4.4. Flattening

"Imagine we have a pooled feature map, a result of applying a pooling layer to the convolved image obtained from the initial convolution operation. The next step involves 'Flattening.'

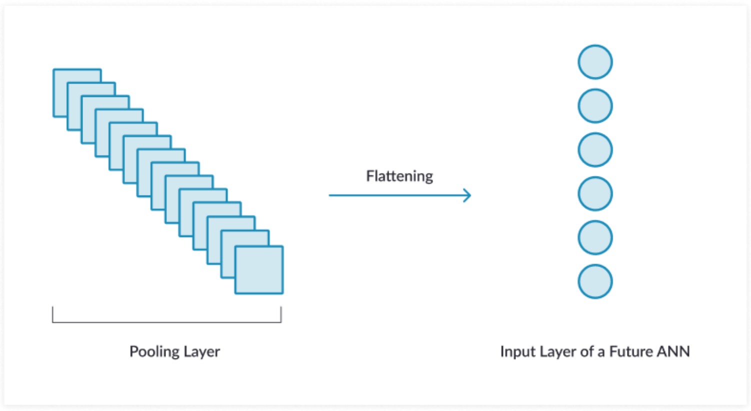


Figure 12. Illustration of the flattening step.

This process entails transforming the pooled feature map into a single, elongated column. We sequentially arrange the numerical values, row by row, into a continuous vector. The primary goal of flattening is to prepare this vector for input into a subsequent artificial neural network, specifically the fully connected (FC) layer, where it undergoes further analysis. [19]

In essence, the journey begins with an input image. We first subject it to a convolution layer, followed by a pooling layer. The output is then flattened into an extensive vector that serves as the input for the fully connected layer, or ANN, completing the preparatory process for deep learning tasks."

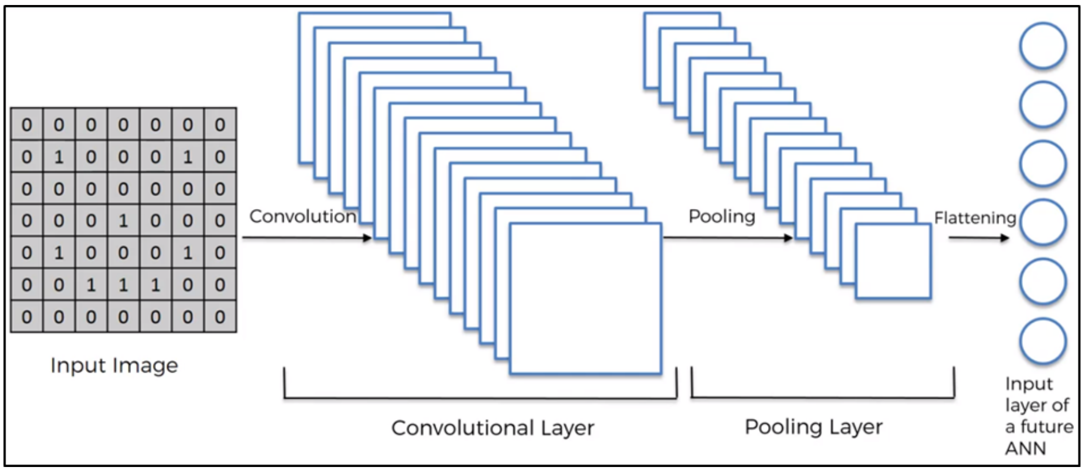


Figure 13. Illustration of features extraction process for typical CNN model.

1.5.4.5. Fully connected layers (FC)

These layers are in the last layer of the convolutional neural network, and their inputs correspond to the flattened one-dimensional matrix generated by the last pooling layer. A ReLU activations function is generally applied to them for non-linearity. Finally, the output layer.[20]

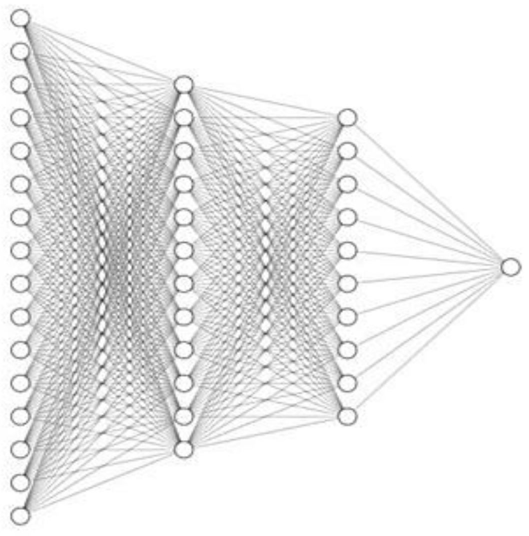


Figure 14. Fully connected layer.

1.6.Conclusion

In this chapter, we introduced the fundamentals of deep learning, which will be extremely helpful for our next phase: lane line detection. We began with a foundational overview of neural networks and then progressed to a detailed discussion of Convolutional Neural Networks (CNNs), a pivotal architecture in deep neural networks. This included an in-depth examination of each component and layer, from the convolutional layers to the final fully connected (FC) layers. Equipped with this insight, this section has come to an end, and a new beginning will be undertaken in the following chapters.

CHAPTER 02

2.1 Introduction

Lane line detection is an essential component of modern autonomous driving systems and advanced driver assistance systems (ADAS). It involves the identification and localization of lane boundaries to facilitate the safe navigation of vehicles. This technology has seen significant advancements over the years, transitioning from basic (classic) computer vision techniques to sophisticated deep-learning models like CNNs.

In this chapter of our project, we will explore the classic computer vision techniques of lane line detection, then we go deeply into deep learning techniques like CNNs, FCN, and other models, and then we delve into the advantages and disadvantages of each technique.

2.2 Lane Line Detection

The emergence of intelligent vehicles has been a transformative force in the automotive industry, promising enhanced safety and efficiency on the road. At the forefront of this technological revolution is lane detection, a system that is indispensable for autonomous driving. Lane detection algorithms are tasked with the critical function of identifying and tracking lane boundaries, which is fundamental to maintaining vehicle positioning and ensuring safe navigation. Despite advancements in perception sensors and the clarity of lane markings on roadways, lane detection remains a formidable challenge for researchers due to environmental factors that can significantly impact the performance of recognition algorithms.

The main challenges facing the development of real-time lane detection algorithms include variations in road structures, incomplete or erased lane markings, and environmental variations that cause illumination disturbances such as snow, shadows, and rain precipitation. To accomplish reliable lane detection, algorithms must be capable of reacting successfully to these challenges, regardless of external factors. This chapter will explore the intricacies of these challenges in detail, examining how they affect the detection process and the solutions that have been proposed to overcome them. We will delve into the strengths and weaknesses of various approaches, including both rule-based and learning-based methodologies, and discuss how they have evolved to meet the robustness requirements of safe and reliable autonomous driving systems.

2.3 Lane line detection algorithms

2.3.1 Classic Computer Vision method

Long before the advent of deep learning algorithms, traditional computer vision (CV) techniques were the primary tools for processing visual information. The origins of the field trace back nearly 60 years when the extraction of information from visual data gained prominence. *Lawrence Gilman Roberts*, often regarded as the father of CV, conducted pioneering experiments aimed at constructing and displaying three-dimensional representations of solid objects from two-dimensional photographs.

As image quality improved, academia further subdivided CV into various fields, with image segmentation and edge detection emerging as among the most prominent. To detect lane lines in images using computer vision we will use OpenCV-Python open-source library. By following these steps:

2.3.1.1 Canny Edge Detection

Canny edge detection is a popular edge detection algorithm developed by John F. Canny in 1986. It is widely used in computer vision and image processing for detecting the edges of objects within images. The Canny edge detection algorithm works in several steps:[21]

2.3.1.1.1 Gaussian Smoothing

The first step involves smoothing the image using a Gaussian filter to reduce noise. This step helps in removing small variations in pixel intensity that are not part of the actual edges.

2.3.1.1.2 **Gradient Calculation**

After smoothing, the algorithm calculates the gradient of the image intensity at each pixel. Typically, this is done using Sobel masks to approximate the derivatives in the x and y directions.

2.3.1.1.3 Non-Maximum Suppression

This step aims to thin out the edges obtained from the gradient calculation. For each pixel, only the local maximum gradient magnitude in the edge direction is retained, while all others are suppressed.

2.3.1.1.4 Double Thresholding

The edge pixels obtained from non-maximum suppression are then classified into strong, weak, or non-edges based on their gradient magnitudes. This is done using two thresholds: a high threshold (strong edge) and a low threshold (weak edge). Pixels with gradient magnitudes above the high threshold are considered strong edges, while those below the low threshold are considered non-edges. Pixels with gradient magnitudes between the two thresholds are labeled as weak edges.

2.3.1.1.5 Edge Tracking by Hysteresis

In this final step, weak-edge pixels that are connected to strong-edge pixels are reclassified as strong edges. This is done by tracing along the edges and connecting adjacent weak edge pixels to strong edge pixels. This helps in forming continuous.

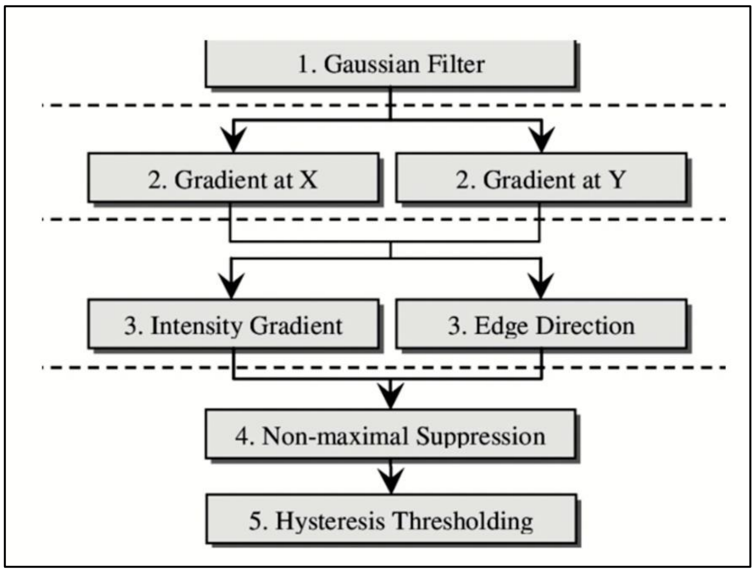


Figure 16. Canny edge detection algorithm.

2.3.1.2 Region of interest

The car's camera focuses solely on the two lanes directly within its field of view, disregarding any extraneous elements. To isolate the relevant lanes and filter out unwanted pixels, we employ a polygonal region of interest (ROI). By defining this polygon, we effectively exclude all pixels lying outside of its boundaries, ensuring that only the desired lane markings are retained for further processing.

2.3.1.3 Hough Transform

The Hough Transform is a popular technique in computer vision and image processing used primarily for detecting shapes within an image, particularly lines and curves. It was initially proposed by Paul Hough in 1962 for detecting straight lines in images and has since been extended to detect other shapes as well, such as circles and ellipses.[22]

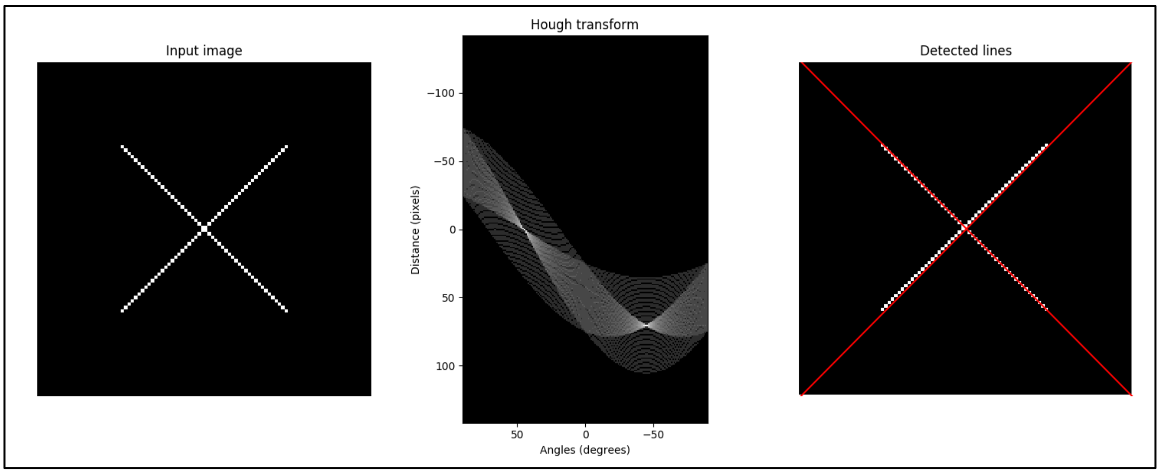


Figure 17. Detecting straight lines using Hough transform.

2.3.1.4 Steering angle

Now that we've identified the coordinates of the lane lines, our next task is to adjust the car's steering to ensure it remains within these lines. Ideally, we aim not just to keep it within the lane boundaries but also to position it at the center of the lane. This involves computing the appropriate steering angle for the car based on the detected lane lines. In the accompanying figure, the red line positioned in the middle symbolizes the computed steering angle for the car.

Traditional computer vision methods like OpenCV for lane line detection have been widely used due to their simplicity and effectiveness in certain scenarios. Here are some advantages and disadvantages of these methods: Traditional methods boast speed advantages over their deep-learning counterparts. These methods, which often employ simpler algorithms such as edge detection and color filtering, are less computationally intensive, resulting in faster processing times. Additionally, traditional methods are easier to implement and understand, making them more accessible to individuals with limited resources or expertise in machine learning. They also have lower resource requirements, allowing them to operate effectively on less powerful hardware, which is particularly advantageous for real-time applications in vehicles with limited processing capabilities. However, traditional methods can be sensitive to varying lighting conditions, weather, and road surface quality, which can lead to unreliable detection. They may not perform well in complex scenarios, such as detecting lane lines in the presence of shadows, wear and tear of lane boundaries, or congested roads. Moreover, these methods often require manual feature engineering, which can be time-consuming and may not generalize well across different environments.

2.3.2 Deep learning-based methods

Deep learning has significantly advanced the field of lane line detection, offering robust solutions to the challenges faced by traditional computer vision methods. By leveraging neural networks, particularly convolutional neural networks (CNNs), deep learning models can learn from large datasets and recognize complex patterns in lane markings with high accuracy. These models are trained on annotated images featuring various road types, lighting conditions, and weather scenarios, enabling them to detect lanes with remarkable precision. The deep learning approach to lane line detection typically involves several stages:

- **Data Preprocessing:** Images are preprocessed to enhance lane line features, which may include techniques like normalization, augmentation, and perspective transformation.
- **Feature Extraction:** CNNs are used to automatically extract relevant features from the preprocessed images without the need for manual feature engineering.
- **Lane Line Prediction**: The extracted features are then used to predict the presence and position of lane lines, often through segmentation methods or direct end-to-end prediction.
- **Post-processing**: The predicted lane lines are refined to ensure consistency and accuracy, which may involve filtering, polynomial fitting, or temporal smoothing.

In general, Deep learning models for lane line detection are trained and evaluated on various benchmarks such as CULane, TuSimple, and BDD100K, demonstrating their effectiveness in real-world conditions. These models are not only more accurate but also more adaptable to different driving environments compared to traditional methods. They continue to evolve, with ongoing research focusing on improving their efficiency and reliability for deployment in autonomous vehicles. Deep learning has become the method of choice for lane line detection due to its accuracy and robustness. Some of the most commonly used deep learning models for this task include:

2.3.2.1 Encoder-decoder CNN architecture:

Encoder-decoder Convolutional Neural Networks (CNNs) are a powerful class of deep learning models particularly suited for tasks like image segmentation, which is a critical component in lane detection systems. These networks function by compressing the input image into a lower-dimensional feature space (encoding) and then reconstructing the output from this space back to the original image dimensions (decoding).

In the context of lane detection, the encoder part of the network learns to capture the essential features of the road and lane markings, while the decoder part aims to map these features back to the spatial dimensions of the input image to predict the location of lane lines. The

effectiveness of encoder-decoder CNNs in lane detection has been demonstrated in various studies, where they have been used to process sequential frames from driving scenes, enhancing the detection robustness, especially in challenging conditions.
Figure 17 shows an example of an encoder-decoder CNN architecture for lane line detection.

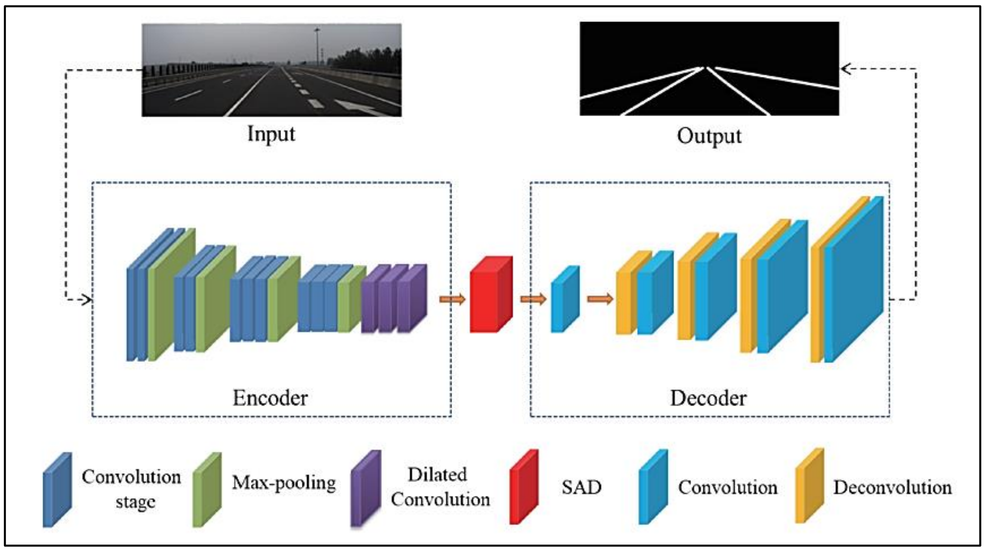


Figure 17. Encoder-decoder CNN architecture.

2.3.2.2 Fully Convolutional Networks (FCNs)

Fully Convolutional Networks (FCNs) are a specialized architecture designed primarily for semantic segmentation tasks, including lane detection. The architecture of FCNs is distinct from traditional CNNs in that it replaces fully connected layers with convolutional layers throughout the network. This design allows FCNs to handle input images of any size and produce segmentation maps that correspond in size to the input images. Here’s a simplified breakdown of the FCN architecture:

- **Convolutional Layers:** These layers extract features from the input image through filters that capture various aspects of the image, such as edges and textures.
- **Pooling Layers:** Following convolution, pooling layers reduce the spatial dimensions of the feature maps, which helps in reducing the computational load and controlling overfitting.
- **Upsampling Layers:** To construct the segmentation map, FCNs use upsampling layers to increase the resolution of the feature maps back to the size of the original image.

- **Skip Connections:** FCNs often include skip connections that combine deep, semantic information from lower layers with shallow, appearance information from higher layers to produce detailed and accurate segmentations.[23]

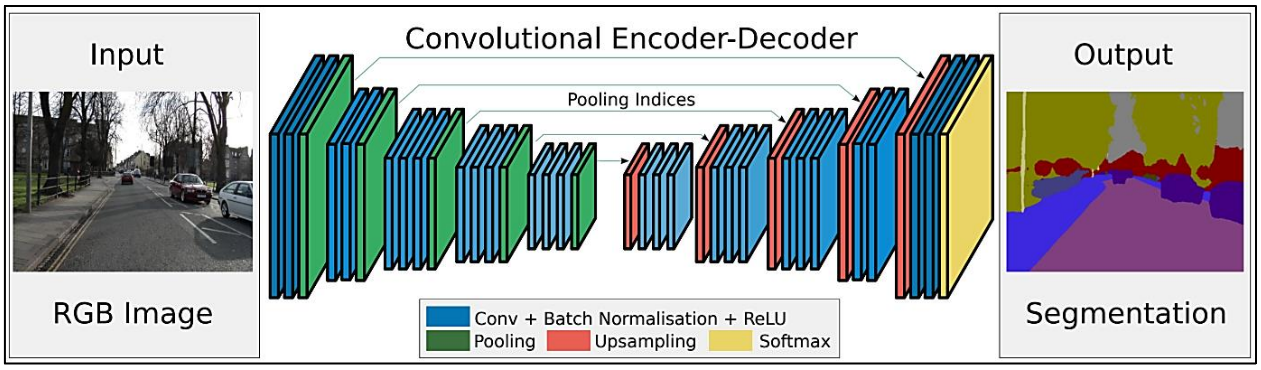


Figure 18. Fully Convolutional Networks (FCNs) architecture.

The FCN architecture is efficient in learning and inference, as it has fewer parameters due to the absence of dense layers, making it faster to train and capable of real-time processing. This makes FCNs particularly suitable for applications like lane detection, where real-time analysis is crucial.

FCNs have been successfully applied to various datasets, such as TuSimple and CULane, achieving high levels of accuracy and precision in lane detection tasks. Their ability to process large-scale datasets and produce detailed segmentations makes them a powerful tool in the development of advanced driver assistance systems and autonomous vehicles.

2.3.2.3 Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are a class of neural networks that are adept at processing sequences of data, making them ideal for tasks that involve temporal dependencies, such as lane detection in video sequences. The architecture of RNNs allows them to maintain a memory of previous inputs by incorporating feedback loops in the network. Here’s a high-level overview of the RNN architecture:

- **Input Layer:** This is where the network receives the input data, which in the case of lane detection, could be a sequence of image frames from a video.
- **Recurrent Layer:** The core of the RNN, this layer has connections that feed back into itself, allowing it to pass information from one step of the sequence to the next. This is crucial for learning and remembering the context of lane positions across frames.
- **Hidden Layers:** Depending on the complexity of the task, RNNs can have multiple hidden layers. Each neuron in these layers processes inputs from both the current time step and the previous time steps.

- **Output Layer:** The final layer produces the output, which for lane detection would be the predicted positions of lane lines in the current frame.

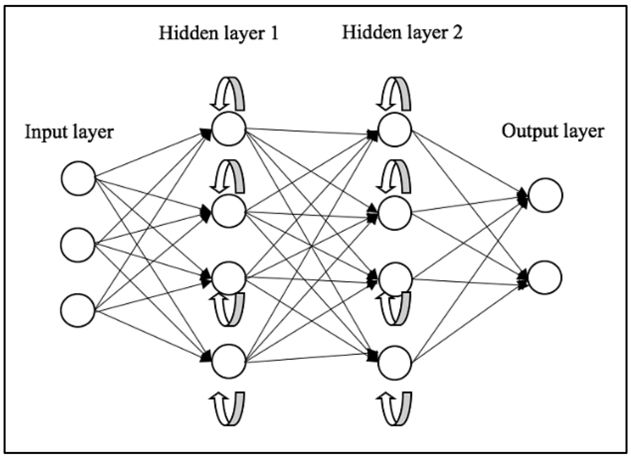


Figure 19. General form of RNNs.

One of the challenges with basic RNNs is the vanishing gradient problem, which makes it difficult for the network to learn long-range dependencies. To address this, advanced variants like Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs) have been developed. These architectures introduce gates that regulate the flow of information, allowing the network to better capture long-term dependencies and forget irrelevant data.
In the context of lane detection, RNNs can be used to predict the trajectory of lane lines over time, providing a more consistent and accurate detection, especially in scenarios where the lane markings are temporarily occluded or faded. They are often combined with CNNs to form a CNN-RNN hybrid model, where the CNN extracts spatial features from individual frames, and the RNN analyzes the temporal sequence of these features to predict lane lines with greater precision.

2.4 Conclusion

In conclusion, in this chapter, we compared traditional computer vision and deep learning for lane line detection. While computer vision methods are more interpretable and less resource-intensive, they falter in complex conditions. Deep learning excels in robustness and adaptability, offering high accuracy despite its higher computational demand. In the next chapter, we will implement the conventional method using OpenCV and an FCN model to observe the practical differences between them.

CHAPTER 3

3.1. Introduction

In the concluding chapter of our project, we will present and analyze the results of our experiments. We have implemented the classic computer vision technique explained in Chapter Two and compared the results for lane line detection using the deep learning FCN model technique

3.2. Software and Libraries for Implementation

3.2.1. Software Overview

Our project leverages Google Colab, a cloud-based platform that facilitates the execution of Python scripts and machine learning tasks using CPUs, and TPUs. Here’s a concise summary of Google Colab’s features:

- ✓ **Virtual Machines:** Google Colab utilizes virtual machines to run scripts and perform computations in the cloud. These machines are pre-equipped with various machine-learning libraries such as TensorFlow and PyTorch.
- ✓ **Kaggle Notebooks:** A Kaggle Notebook is an interactive environment provided by Kaggle, a platform for data science and machine learning competitions. These notebooks are hosted by Jupyter Notebooks which allow users to write and execute code in a web-based interface. They are equipped with various tools and resources for data analysis, visualization, and model building. Kaggle Notebooks support languages like Python and R, and they come preloaded with popular libraries and datasets, making it easy for users to start their data science projects without needing to set up their development environment.
- ✓ **Hardware Acceleration**: Google Colab offers GPU and TPU acceleration, significantly speeding up machine learning processes, enabling efficient handling of large datasets, and quicker model training.
- ✓ **Google Drive Integration:** Seamless integration with Google Drive allows for convenient storage and access to data and notebooks from anywhere. Users can mount their Google Drive within Colab for direct data access.
- ✓ **Collaborative Features:** The platform supports collaboration, enabling users to share notebooks, comment on code, and work together in real-time.
- ✓ **Code Snippets Library:** A vast collection of code snippets and examples in Google Colab simplifies the implementation of common machine learning tasks.

In essence, Google Colab stands out as a powerful and flexible tool for machine learning and data science projects, offering a suite of features to accommodate diverse workflows.

3.2.2. Programming Language

For our project within Google Colab, we chose Python as our programming language. Renowned for its high-level capabilities, Python is prevalent in various fields, including web development, data analytics, machine learning, and artificial intelligence.

3.2.3. Frameworks

- ✓ **TensorFlow:** An open-source machine learning framework by Google Brain, released in 2015. It's designed for developing, training, and deploying machine learning models across multiple platforms, with tools and resources for a broad spectrum of applications, from image/audio recognition to natural language processing.
- ✓ **Keras:** A high-level neural network API in Python, Keras operates atop TensorFlow, CNTK, Theano, or PlaidML. Developed by François Chollet of Google, it's licensed under MIT and simplifies the creation of deep learning models.
- ✓ **PyTorch:** Originating from Facebook's AI Research lab, PyTorch is an open-source Python library for machine learning. It emphasizes flexibility and speed in deep learning model development, supporting dynamic computational graphs and automatic differentiation.

3.2.4. Libraries

- ✓ **OpenCV:** A free, open-source computer vision and machine learning software library.
- ✓ **Matplotlib:** A Python library for creating static, animated, and interactive visualizations.
- ✓ **NumPy:** A Python library for large-scale multi-dimensional arrays and matrices, along with a comprehensive collection of mathematical functions.
- ✓ **Pandas:** An open-source Python package for data manipulation and analysis, offering tools for data cleaning, exploration, transformation, and efficient handling of large data sets.

3.3. Dataset

To train our FCN model we used the dataset provided by *Michael Virgo (mvirgo)*, this dataset contains:

- ✓ 21,054 total images were gathered from 12 videos (a mix of different times of day, weather, traffic, and road curvatures).

Divided based on various conditions (night vs. day, shadows, rain vs. sunshine):

Weather & Time	Percentage of data
Clear night driving	17.4%
Rainy morning driving	16.4%
Cloudy afternoon driving	66.2%

Table 1. Percentage of data according to weather & time

Divided based on straight lines and various curved lines:

Curvature	Percentage of data
Straight or mostly straight roads	26.5%
A mix of moderate curves	30.2%
Very curvy roads	43.3%

Table 2. Percentage of data according to curvature

- ✓ The roads also contain difficult areas such as construction and intersections.
- ✓ Finally, after removing all the unusable images mainly due to blurriness, hidden lines, etc.… All in all, there were a total of 12,764 usable images for training.

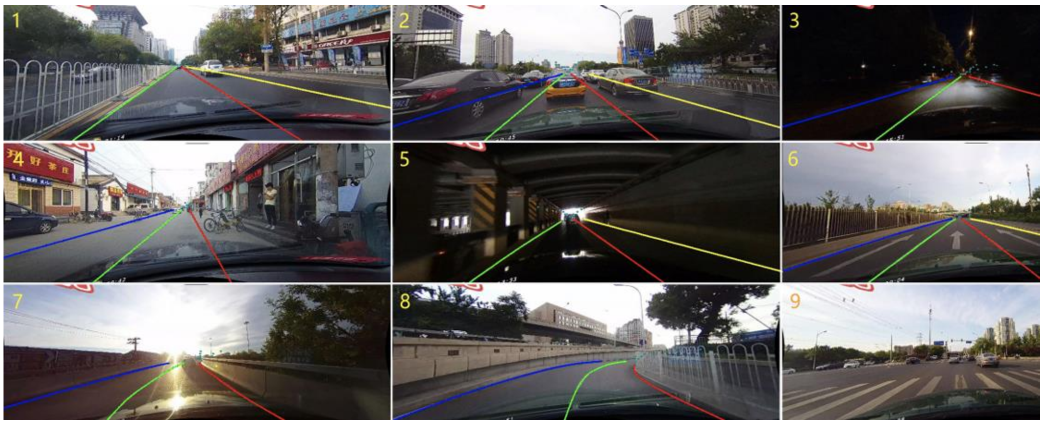


Figure 20. Various training dataset.

Table 3 displays the different road conditions represented in the images across the entire dataset.

1	2	3	4	5	6	7	8	9
Normal	Crowded	Night	No line	Tunnel	Arrows	Dazzle light	Curve	Cross road

Table 3. Dataset Scenarios

3.4. Baseline Model Selection

We had chosen the MLND-Capstone project by *Michael Virgo* as a baseline model for our project. Because it provides us a clear documentation with all the information we need and the clarity of its codebase. Michael's project report indicates that he attained notable performance on his dataset with just 10 epochs of training. [24]

3.4.1. Network Architecture Selection

The neural network of this baseline model is structured with an Encoder-Decoder architecture, as shown in (figure 22), The Encoder segment is composed of seven (07) Convolutional layers, mixed with five (05) Dropouts and three (03) Max pooling layers. The Encoder's role is to compress the image dimensions while capturing its key features into more channels. On the other hand, the Decoder segment comprises six (06) layers of Deconv mixed with four (04) Dropouts and three (03) Upsampling layers. It up-samples the feature vectors to an image mask, which has the same height and width as the original image, on a gray-scale format. The main reason for selecting this particular architecture is to speed up training when it comes to processing our higher-resolution images. In contrast, applying alternative architectures that incorporate Convolutional Layers followed by Fully Connected Layers would necessitate the training of a very large number of large parameter sets, which would consequentially impede the neural network's efficiency.

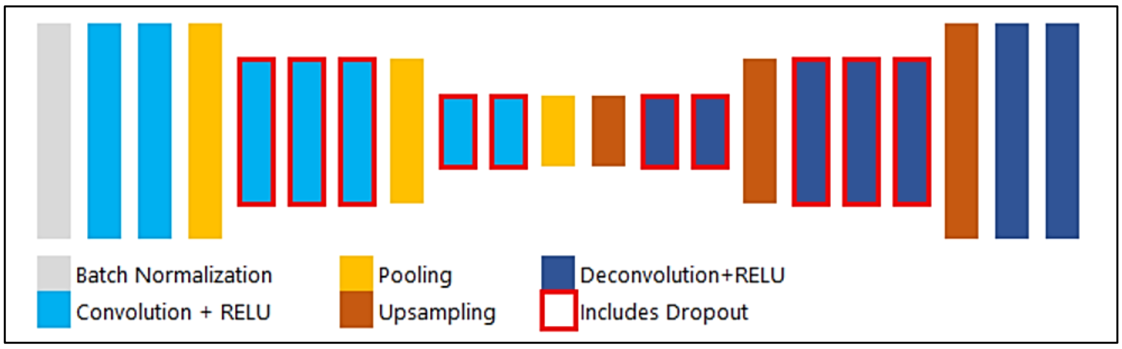


Figure 18. Baseline model fully convolutional network architecture.

3.4.2. Model Adjustments

To obtain a better lane line detection result we made some changes in the baseline model from (MLND-Capstone project by *Michael Virgo).* These are the key changes we applied:

- ✓ **Increase the number of convolutional layers:** we added more two (02) Convolutional layers; (Conv2D (128, (3, 3)) two (02) Dropouts and one Pooling layer, and two (02) Deconv layers (Conv2DTranspose (128, (3, 3)) mixed with two (02) Dropouts and one Upsampling layer, with increased filters to capture more features.
- ✓ **Using padding='same' instead:** ensures the output size matches the input size for better handling of edge cases.

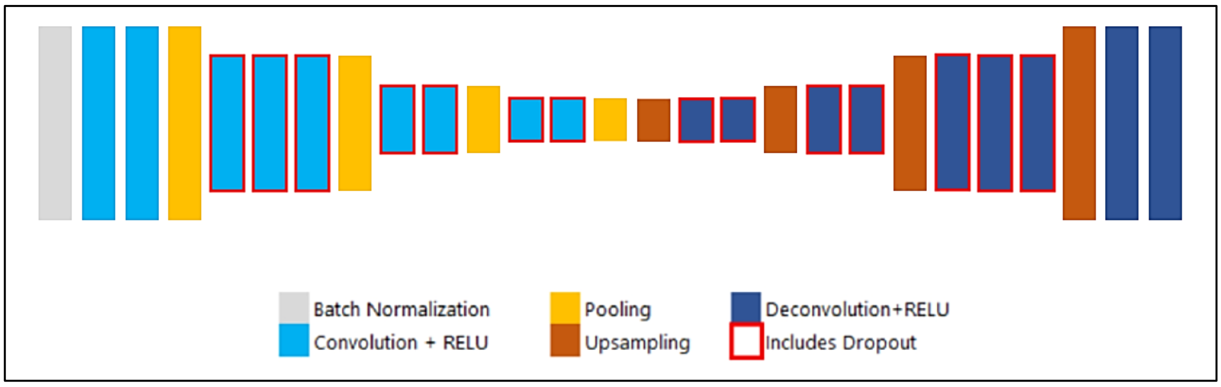


Figure 19. Final fully convolutional network architecture.

These changes aim to enhance the model's ability to accurately detect lane lines by learning more complex patterns and reducing overfitting.

3.5. Results, and Discussion

3.5.1. Evaluation Metrics

To evaluate the performance of our training model, we have applied the following evaluation metrics: Accuracy, Recall, Precision, F1 Score, and MSE (mean-squared error) to minimize the loss between the predicted pixel values of the output lane image and what the lane image label was. As presented in (Table 4).

Epochs	Accuracy	Recall	Precision	F1 Score	Loss
25	0.9612	0.9561	0.9590	0.9575	0.0051
50	0.9619	0.9695	0.9635	0.9665	0.0040
75	0.9627	0.9675	0.9668	0.9671	0.0033
100	0.9628	0.9716	0.9682	0.9700	0.0029

Table 4. Evaluation Metrics Results for Different Epoch Values.

- **Accuracy:** The accuracy slightly increases from 0.9612 at 25 epochs to 0.9628 at 100 epochs.
- **Recall:** Recall improves significantly from 0.9561 at 25 epochs to 0.9716 at 100 epochs. Higher recall means the model is getting better at identifying the actual lane lines correctly. This improvement suggests that the model is becoming more sensitive to detecting lanes as training progresses.
- **Precision:** Precision also shows an upward trend, from 0.9590 at 25 epochs to 0.9682 at 100 epochs. This indicates that the model's predictions are becoming more accurate, meaning fewer false positives (non-lane pixels predicted as lane pixels).
- **F1 Score:** The F1 score, which is the harmonic mean of precision and recall, improves from 0.9575 at 25 epochs to 0.9700 at 100 epochs. This metric shows that the model is achieving a good balance between precision and recall, making it reliable for lane detection tasks.
- **Loss:** The loss decreases from 0.0051 at 25 epochs to 0.0029 at 100 epochs. A lower loss indicates that the model's predictions are getting closer to the actual values, which implies better performance and convergence.

3.5.2. Specific Observations

All metrics show a steady improvement as the number of epochs increases. This suggests that the model continues to learn and generalize better over time without overfitting, which is a good sign of the model's robustness.

Both precision and recall are improving together, which is critical for the lane detection task. High recall ensures that most lane lines are detected, while high precision ensures that there are few false detections.

The significant improvement in metrics between 25 and 100 epochs indicates that the training process is effective. The model leverages the training data efficiently to enhance its performance.

Overall, the training appears to be progressing well, and the model is performing admirably in detecting lane lines.

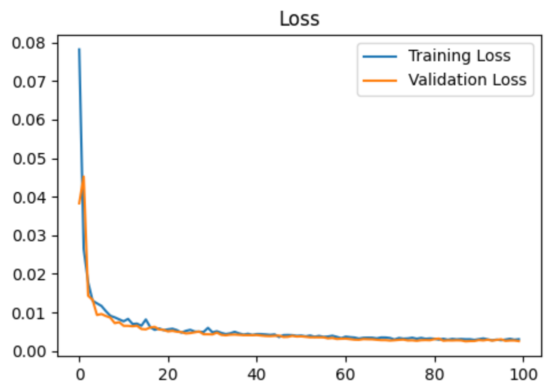


Figure 21. Training and Validation Loss

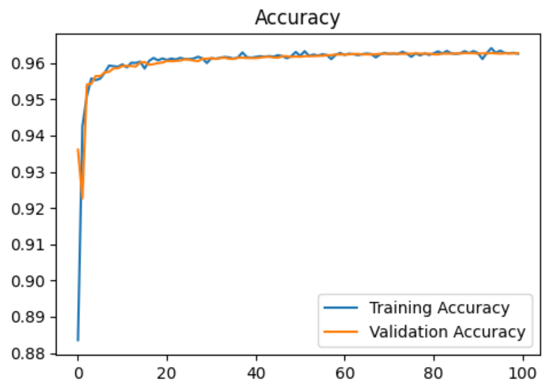


Figure 20. Training and Validation Accuracy

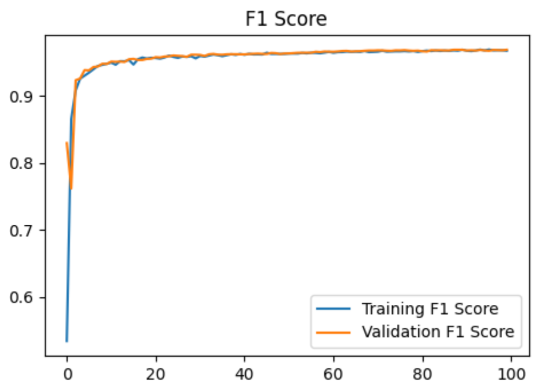


Figure 22. Training and Validation F1 Score

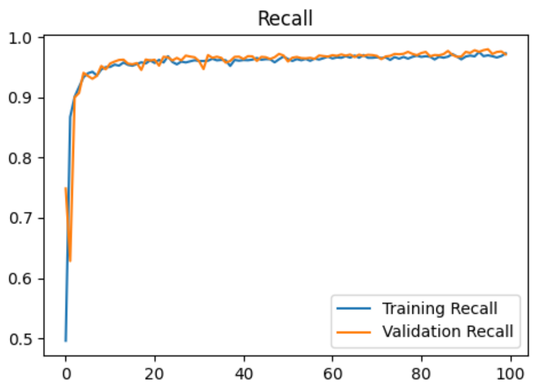


Figure 23. Training and Validation Recall

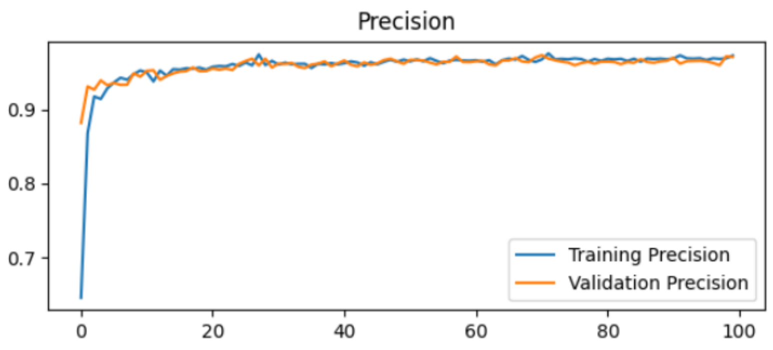
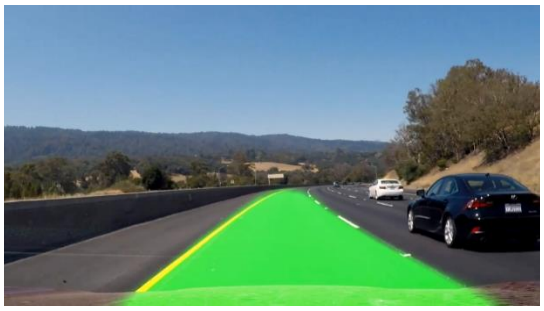


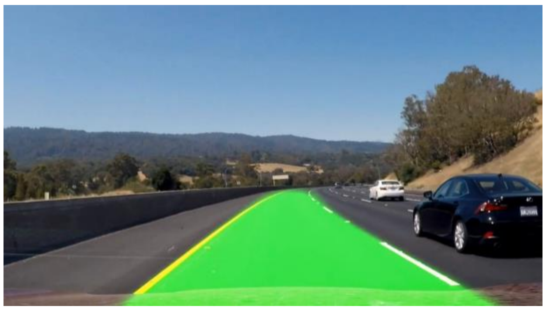
Figure 24. Training and Validation Precision

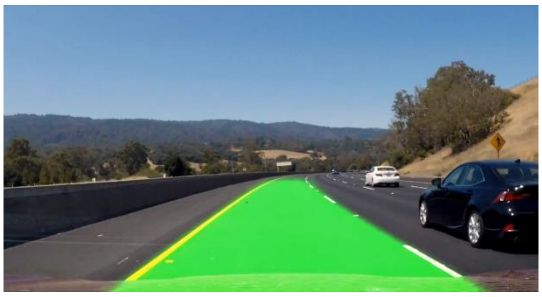
3.5.3. Our model results progression

The following images are extracted frames from a tasting video based on our trained model to detect lane lines, in different weather, time, and curvature, in four different epoch values:

3.5.3.1 Straight and clear lane lines


25 epochs


50 epochs


75 epochs

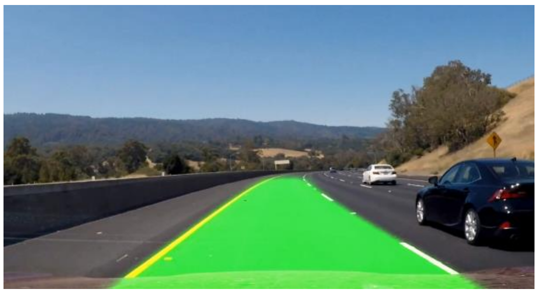
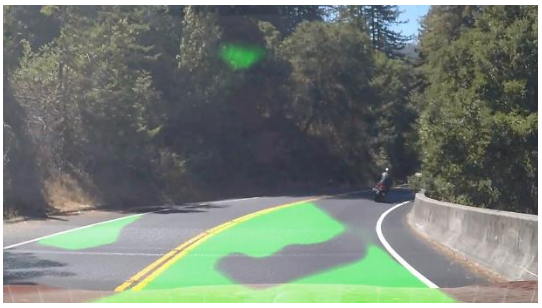
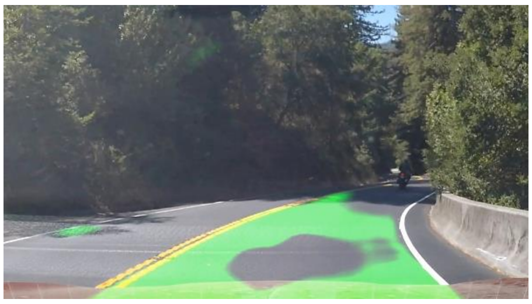

100 epochs

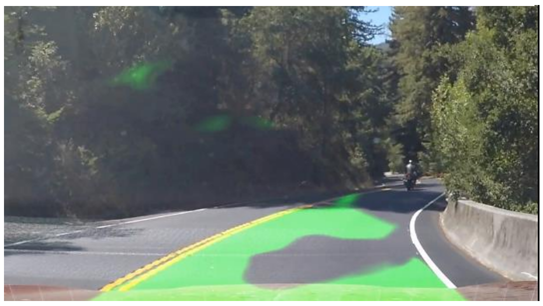
Figure 25: Model results from Straight and clear lane lines.

As we can see in the previous result the model only needs 25 epochs to detect the lane lines due to the Straight and clear lane lines.

3.5.3.2 Curvy Lane Lines


25 epochs


50 epochs


75 epochs

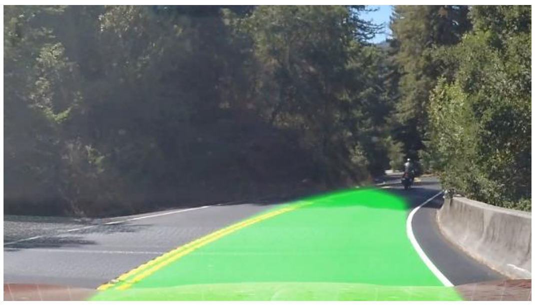
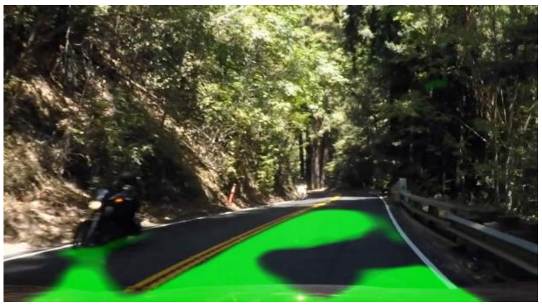

100 epochs

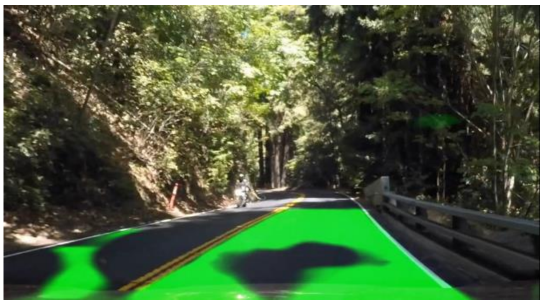
Figure 26: Model results in Curvy lane lines.

In this case, it takes the model 100 epochs to well detect the lane lines and that is because of curvy lane lines.

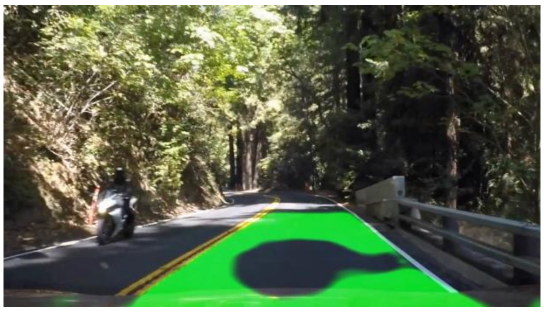
3.5.3.3 Hard shadows



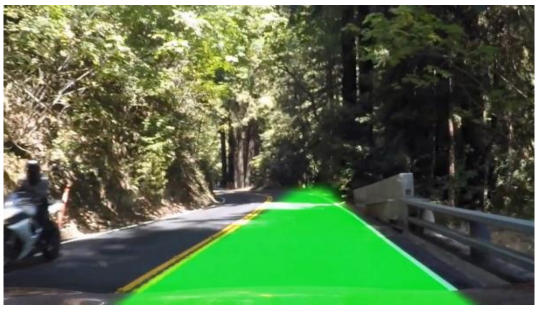
25 epochs



50 epochs



75 epochs



100 epochs

Figure 30: Model results in Hard shadows.

In this scenario, the model requires 100 epochs to accurately detect the lane lines due to the hard shadows which make it difficult to detect the lane lines.

3.5.3.4 Hard sunlight

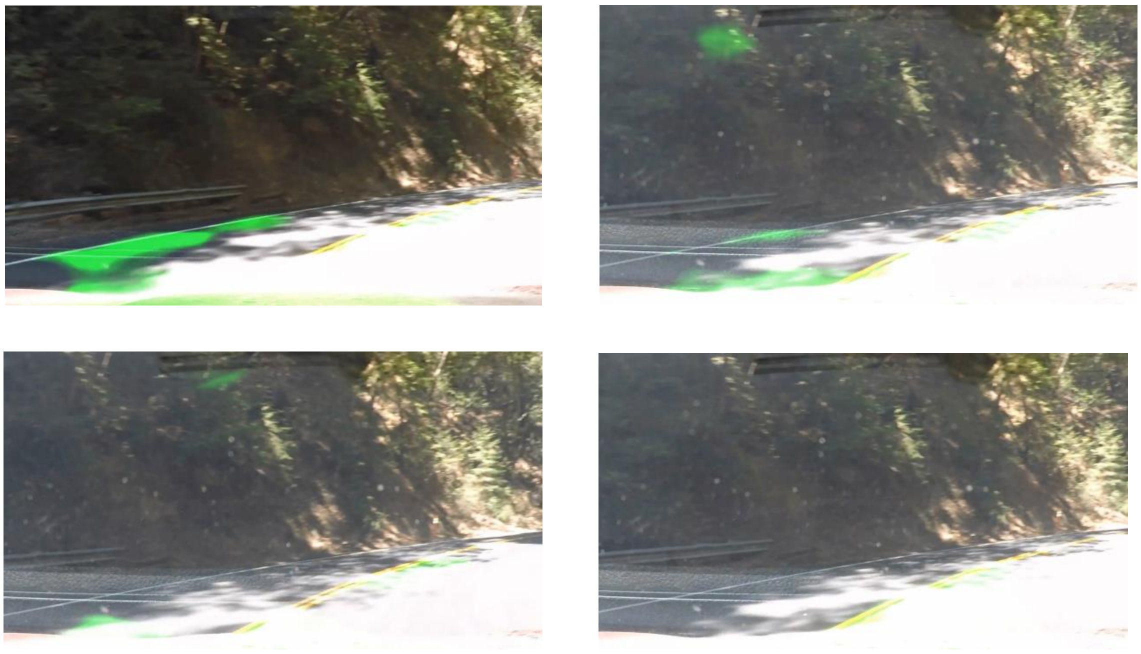


Figure 27 : Model results in Hard Sunlight.

As we can notice from the result shown, the model needs at least 100 epochs to accurately detect the lane lines due to the intense sunlight, which complicates their detection.

3.5.3.5 Blurry footage

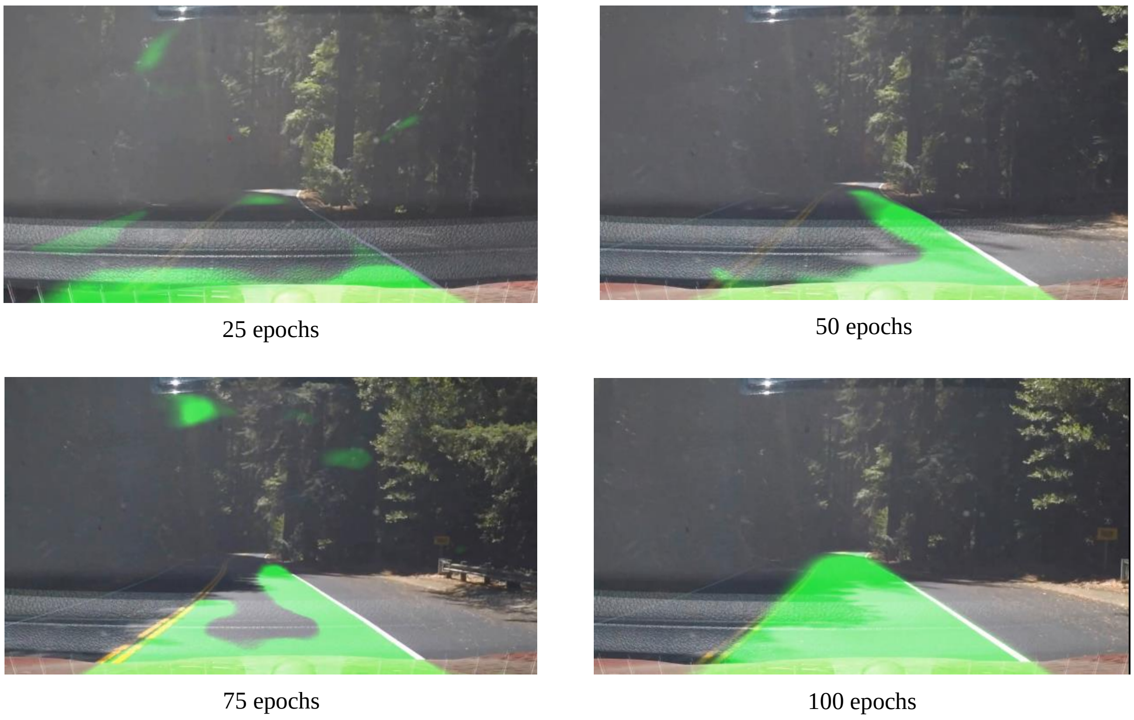


Figure 28: Model results in Blurry footage.

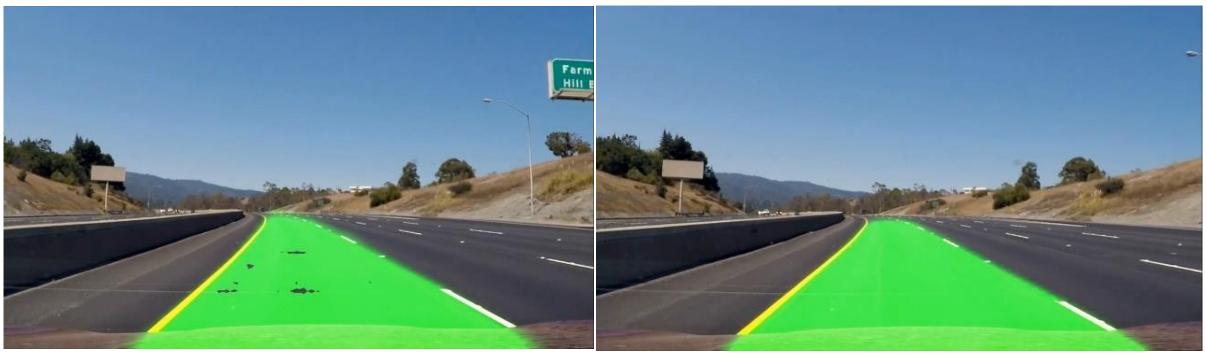
In this final scenario, where the footage is blurry which makes it hard to detect lane lines, it takes at least 100 epochs to accurately detect the lane lines.

Overall, it is evident that challenging conditions such as curvy lines, hard shadows, intense sunlight, and blurry footage significantly increase the training epochs needed for the model to detect lane lines accurately.

3.5.4. Comparison between the baseline model and Our model results

In this part, we will do a comparison between the baseline model results and our adjusted model results in four different scenarios: Straight and clear lines, curvy lines, hard shadows, hard sunlight, and blurry footage.

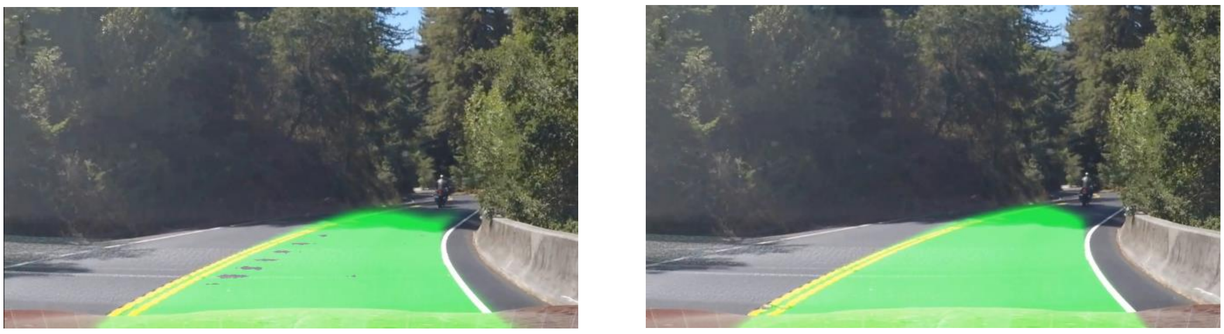
3.5.4.1 Straight and clear lane lines



Original model Developed model

Figure 29: Comparison of our model and the baseline model in straight and clear lane lines.

3.5.4.2 Curvy Lane Lines



Original model Developed model

Figure 30 : Comparison of our model and the baseline model in Curvy Lane line.

3.5.4.3 Hard shadows

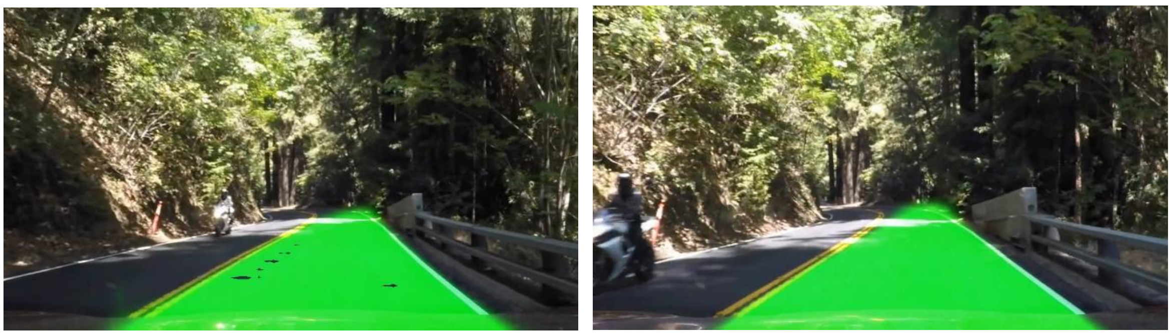

Original model Developed model

Figure 31 : Comparison of our model and the baseline model in Hard shadows.

3.5.4.4 Hard Sunlight

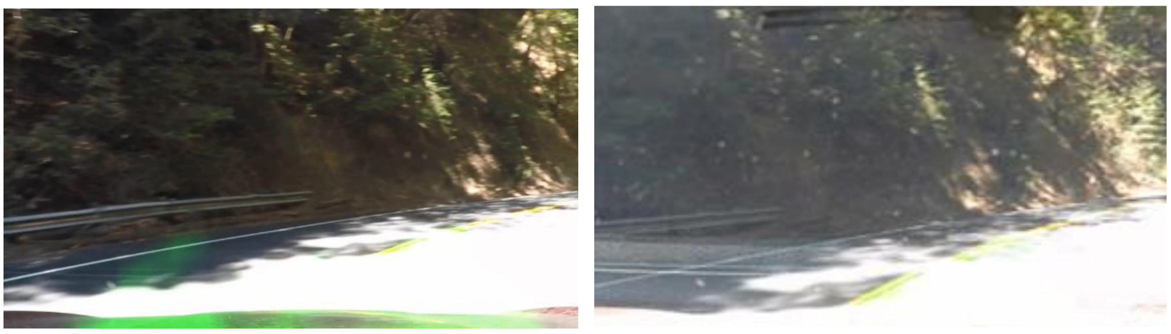

Original model Developed model

Figure 32: Comparison of our model and the baseline model in Hard Sunlight.

3.5.4.5 Blurry footage

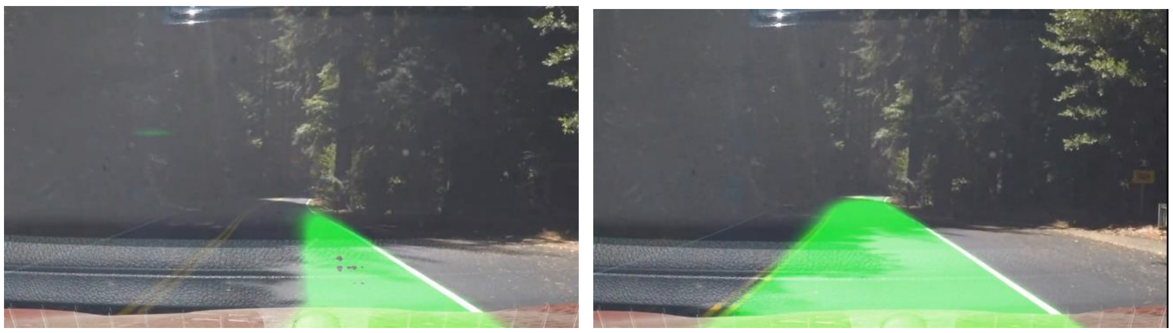

Original model Developed model

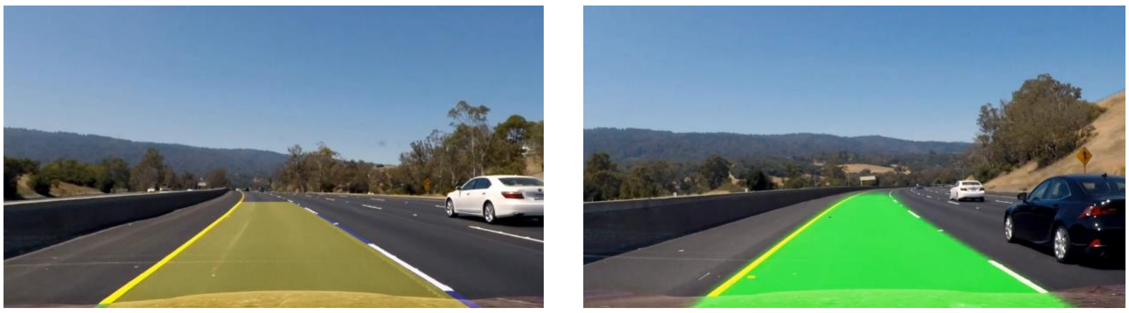
Figure 37: Comparison of our model and the baseline model in Blurry footage.

Based on the results we obtained, show that our model performs better in most scenarios; Straight and clear lines, curvy lines, hard shadows, hard sunlight, and blurry footage, this improvement can be attributed to the fact that the baseline model was trained in fewer number of epochs. On the other hand, our model was trained with an advanced architecture for 100 epochs, leading to enhanced performance outcomes.

3.5.5. Comparison between Computer vision and Deep learning

Now we will compare our FCN model output and the computer vision (OpenCV) output in different scenarios as follows:

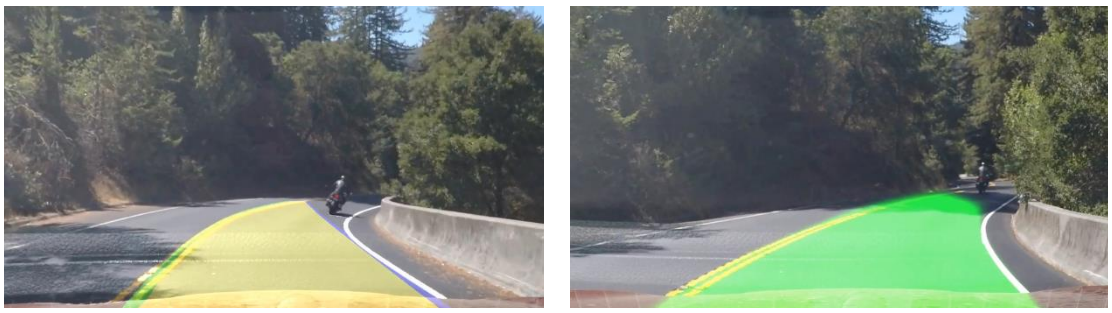
3.5.5.1 Straight and clear lane lines:



Computer vision (OpenCV) Deep learning FCN

Figure 33: Comparison between CV and DL in Straight and clear lane lines.

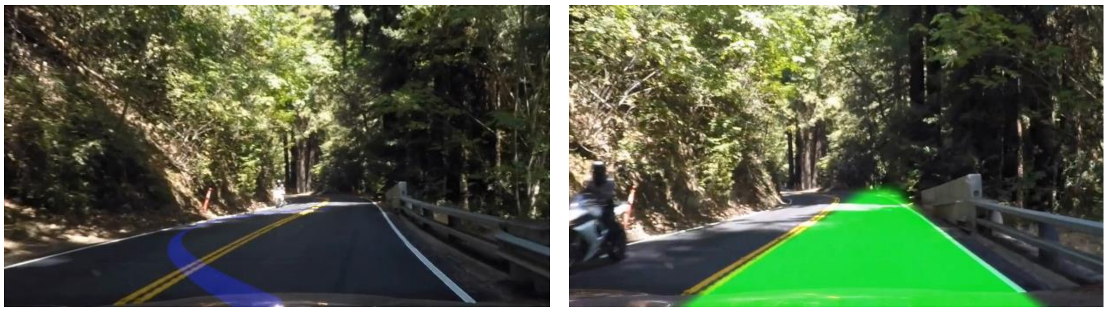
3.5.5.2 Curvy Lane lines



Computer vision (OpenCV) Deep learning FCN

Figure 34 : Comparison between CV and DL in Curvy Lane lines.

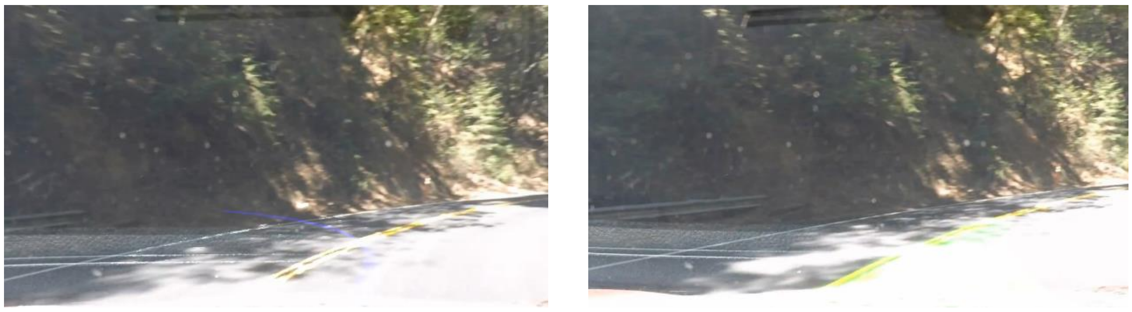
3.5.5.3 Hard shadows



Computer vision (OpenCV) Deep learning FCN

Figure 35: Comparison between CV and DL in Hard shadows.

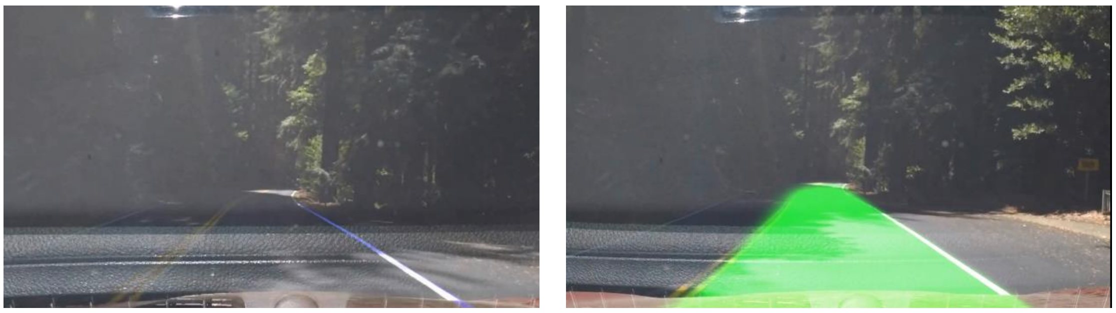
3.5.5.4 Hard Sunlight



Computer vision (OpenCV) Deep learning FCN

Figure 36: Comparison between CV and DL in Hard Sunlight.

3.5.5.5 Blurry Footage



Computer vision (OpenCV) Deep learning FCN

Figure 37: Comparison between CV and DL in Blurry footage.

In perfect conditions like Straight and clear lane lines, the traditional computer vision method shows a better result than the deep learning model. Except that the results clearly show that the Deep learning FCN model has outperformed the traditional computer vision method significantly in most of the scenarios we had, in the scenario of images that contain a curvy line we noticed that the deep learning FCN model gave much more accurate results than computer vision method, the same with scenarios that contain hard shadows and sunlight and blurred footage, Since the image processing suffers from several problems.

Both traditional image processing and deep learning approaches have their own set of advantages and limitations when it comes to lane line detection. Traditional methods are simple, efficient, and suitable for controlled environments, but they struggle with variability and require manual tuning. Deep learning methods, while computationally intensive and dependent on large datasets, offer robustness, adaptability, and high accuracy, making them ideal for the dynamic and unpredictable conditions encountered in real-world driving. As technology advances, the integration of both methods could potentially leverage their respective strengths, leading to more reliable and efficient lane detection systems.

3.6. Conclusion

In this work, we explored the application of deep learning techniques, particularly Fully Convolutional Networks (FCNs), for robust lane detection in autonomous driving scenarios. We demonstrated the effectiveness of FCNs in accurately identifying lane markings under various challenging conditions, such as different times of day, weather, and road curvatures.

Our experiments on the dataset showed promising results, with the FCN model achieving high precision, recall, and F1 scores as training progressed. The steady improvement in these metrics, coupled with the reduced loss, indicates that the model was able to generalize well without overfitting, making it a strong choice for real-world path detection applications after further development. While traditional computer vision methods like OpenCV offer simplicity and speed, deep learning approaches like FCNs have the advantage of superior performance and adaptability to complex scenarios. As autonomous driving technology continues to advance, the adoption of deep learning for lane detection will play a crucial role in ensuring safe and reliable navigation on our roads.

General Conclusion

Working on lane detection for autonomous driving using deep learning has been an enriching and challenging experience. The journey has highlighted the vast potential and current limitations of deep learning in real-world applications.

At first, there were notable gains when using deep learning models instead of conventional computer vision techniques. The ability of the model to reliably identify lanes in a variety of scenarios demonstrated the strength of Fully Convolutional Networks (FCNs). But it was soon apparent that there were still several difficult situations, including changes in light and shadow and intense glare.

One key lesson from this project is the critical importance of data diversity. While the current dataset was sufficient to outperform traditional methods, it lacked the variety needed to generalize across all possible driving conditions. This realization underscores the fundamental principle in deep learning: the quality and diversity of data are as crucial as the model architecture itself.

In summary, this project has been a profound learning experience. It has not only demonstrated the power of deep learning but also revealed its limitations and areas for improvement. By focusing on more diverse data, exploring advanced neural network architectures, and expanding detection capabilities, we can move closer to developing a truly reliable and versatile autonomous driving system. The journey ahead is filled with challenges, but with each step, we get closer to realizing the full potential of autonomous driving technology.

Potential Improvements

This is our approach to lane detection using deep learning. While it's not perfect, it is significantly more accurate than the traditional computer vision technique, but in the hard challenging scenarios still loses the lane in the transition between light and shadow.

Here are some ideas to improve our model in the future:

- More data. This is always the case in deep learning of course, but we think with an even wider array of conditions (such as transitions between light and shadow) and more different cameras, the model can get even better.
- Use of data without lane lines or only one line. This will be useful in areas outside cities where there is one line or no line on the road.

- Expanding the model to detect even more items like vehicle and pedestrian detection, we could add that on separate filters at the end. This will make driving safer in crowded cities.
- Experiment with different network architectures: While the Fully Convolutional Network (FCN) architecture performed well, exploring other deep learning models like U-Net or DeepLabV3 could potentially lead to even better results.
- Implement test-time augmentation: Applying transformations like flipping, rotating, or scaling to the input images during inference and averaging the predictions can often improve robustness and accuracy.

By incorporating these improvements, the deep learning model for lane detection could potentially achieve even higher accuracy, robustness, and generalization capabilities, making it more suitable for real-world autonomous driving applications.

BIBLIOGRAPHY:

[1] Raymond Noble & Denis Noble (June 22 2023). Artificial Intelligence.

[2] Deepti Chopra and Roopal Khurana (2023). Introduction To Machine Learning.

[3] Jitendra Sharma (May 27 2021). Introduction to Supervised Deep Learning Algorithms!

[4] kurtis Pykes (Jan 2024). Introduction to Unsupervised Learning.

[5] Frederick Douglass (June 2023) Deep Learning and Neural Networks: Methods and Applications.

[6] Janiesch, Christian & Zschech, Patrick & Heinrich, Kai. (2021). Machine learning and deep learning.

[7] Iván Sánchez Fernández (June 2023). Machine learning and deep learning in medicine and neuroimaging.

[8] Shunan Wu (May 29 2023). A Weight-Generating Approach of a Deep Neural Network for the Parameter Identification of Dynamic Systems.

[9] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, Aaron Courville Proceedings of the 36th International Conference on Machine Learning, PMLR 97:5301-5310, (2019).On the Spectral Bias of Neural Networks

[10] İbrahim Arslan (February 27 2022). Analysis on the Selection of the Appropriate Batch Size in CNN Neural Network

[11] Noor Baha Aldin (March 31 2022). Accuracy Comparison of Different Batch Size for a Supervised Machine Learning Task with Image Classification.

[12] Vipul Bansal (Feb 24 2022). Activation Functions: Dive into an optimal activation function.

[13] Pragati Baheti (May 27, 2021). Activation Functions in Neural Networks [12 Types & Use Cases].

[14] Artem Oppermann(Dec 14, 2022). How Loss Functions Work in Neural Networks and Deep Learning.

[15] Sushant Kumar (Sep 21, 2020). Common Loss functions in machine learning for Classification model.

[16] Jiawei Zhang (Mar 2019). Gradient Descent based Optimization Algorithms for Deep Learning Models Training

[17] Zoumana KEITA (Nov 2023). An Introduction to Convolutional Neural Networks (CNNs).

[18] Keiron O'Shea, Ryan Nash (Dec 2015). An Introduction to Convolutional Neural Networks

[19] Radhey Shyam (October 21, 2021). Convolutional Neural Network and its Architectures

[20] Diego Unzueta (Oct. 18, 2022) Fully Connected Layer vs. Convolutional Layer

[21] Sofiane Sahir (Jan 25, 2019). Canny Edge Detection Step by Step in Python — Computer Vision

[22] Surya Teja (Sep 27, 2019) .Karri Hough Transform

[23] Jonathan Long, Evan Shelhamer, Trevor Darrell (8 Mar 2015). Fully Convolutional Networks for Semantic Segmentation.

[24] https://github.com/mvirgo/MLND-Capstone/tree/master.