application of machine learning in control vehiculs

SAFI WAIL
University of Paris Saclay Evry, France

March 2024

1 Abstract

To create an autonomous vehicle, various computer vision tasks such as lane and object detection, obstacle avoidance, and classification need to be integrated. One efficient approach to address these tasks is by training a model using data collected from a steering camera over extended periods of driving across diverse road conditions and weather scenarios. This method enables the system to learn and generalize from real-world experiences with minimal data collection.

In this project, we utilized Convolutional Neural Networks (CNNs) to train the data obtained from the Udacity simulator after an extensive driving period. Subsequently, the mapped data was fed into a neural network architecture inspired by the NVIDIA model.

2 Introduction

Convolutional Neural Networks (CNNs) are a class of deep learning models specifically designed for processing and analyzing visual data. They have revolutionized various fields, particularly computer vision, by achieving remarkable performance in tasks such as image classification, object detection, and image segmentation. Inspired by the structure and function of the visual cortex in the human brain, CNNs are composed of multiple layers, including convolutional layers, pooling layers, and fully connected layers. These layers work together to learn hierarchical representations of visual features from raw pixel data.The key component of CNNs is the convolutional layer, which applies convolution operations to input images using learnable filters or kernels. These filters slide over the input image, extracting features such as edges, textures, and patterns at different spatial scales. Through the process of convolution and subsequent non-linear activation functions, CNNs can capture complex patterns and relationships within the input data. Pooling layers are often used to downsample feature maps obtained from convolutional layers, reducing computational complexity and extracting the most salient features. Additionally, fully connected layers integrate the extracted features to make predictions, such as class labels in image classification tasks. One of the main advantages of CNNs is their ability to automatically learn hierarchical representations of features from raw data, eliminating the need for manual feature extraction. This makes CNNs highly effective in tasks where the underlying structure of the data is complex and not easily discernible by humans.

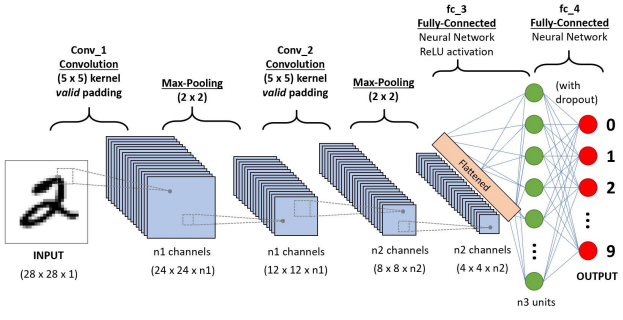


Figure 1: convolutional neural network CNN

CNNs will play a pivotal role in this project as

a core component of the model, significantly simplifying the task of autonomous driving. By leveraging CNNs, we circumvent the necessity to identify specific human-defined features like lane markings, guard rails, or other vehicles. This approach eliminates the need to construct a collection of "if, then, else" rules based on observations of these features, allowing for a more streamlined and robust autonomous driving system.

3 Basic CNN components :

1. Convolutional layer: The convolution layer is composed of a set of convolutional kernels where each neuron acts as a kernel.

2. Pooling layer: It is used to reduce the number of parameters and computational complexity of the model. The most common type of pooling operation is max-pooling.

3. Activation function: it is used to transform the output of the previous layer into a new representation that can be fed to the next layer.In literature, different activation functions such as sigmoid, tanh, maxout, SWISH, ReLU, and variants of ReLU, such as leaky ReLU, ELU, and PReLU are used to allow the model to learn complex and nonlinear relationships between the input and output data.

4. Batch normalization: The technique involves normalizing the activations of the previous layer for each mini batch of data during training, which helps to speed up the training and improve the stability of the model.

5. Dropout : It is used to improve the generalization of the network by randomly skipping some units or connections with a certain probability.

6. Fully connected layer: It is used in the later stages of a neural network where every neuron in the previous layer connected to every neuron in the current layer.

4 Tools Used In This Project:

1. Google Collab: it is a cloud-based platform that providesa free environment for writing and executing code in Python, we used it because it provides easy access to powerful hardware resources, such as GPUs.

2. tensorlfow: is an open-source machine learning library forPython that is developed by Google brain team . It is used for building and training deep neural networks.

3. Google Drive: It is a cloud-based file storage and synchronization service provided by Google. We used it to store our data.

4. Python: It is a popular high-level programming language that is widely used in a variety of fields, including artificial intelligence.

5 DAVE-2 system

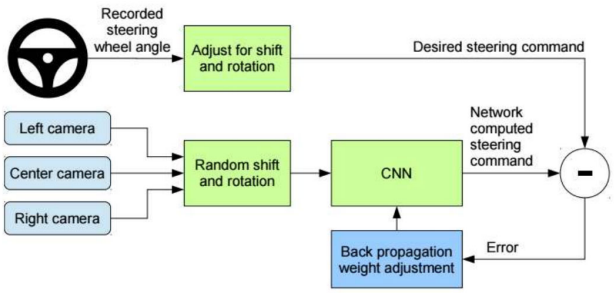
It is an advanced autonomous driving platform developed by NVIDIA that demonstrates that a single neural network can learn to steer a car directly from raw camera pixels.So It enables autonomous, optimal, and safe driving by learning how to steer based on images of different roads. As shown in Figure 1,ts architecture is based on three cameras that generate images to be fed to the neural network. The neural network then simply generates the steering angle or something equivalent. The training data is created by a human driver, and the system collects data from the cameras (training data) and from the steering wheel moved by the pilot (training labels are collected). This process is called behavioral cloning because the network is trying to clone the behavior of the human driver. Instead of using the turning radius r as the steering command, the use of the inverse (1/r) will be better for many reasons:

1. **Preventing Singularity:**

   When driving straight, the turning radius tends to infinity, which is not good for the steering command, for this reason the use of 1/r will be the best choice.

2. **smoothing transitions:**

   This also allows the change between the negative and positive values(around zero) to be very small (smooth)which creates a in the transition .



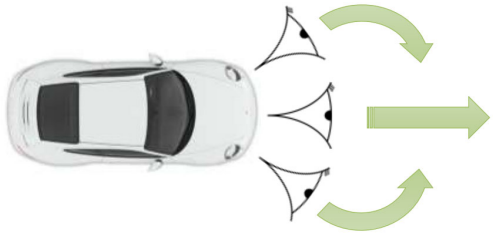


Figure 2: DAVE-2 system

data set:

the data is collected from Udacity simulator,after an extensive driving period.The problem of this method is that driving in a bad way leads to a bad decisions after training the mode. so driving in a correct way during the collection of the data should be a necessary condition.

To expedite the process, we utilized pre-existing data available on GitHub, as mentioned in the simulation file.

Upon visualizing the data, we observed that the majority of the data points cluster around a steering angle of 0. This clustering may result in under-fitting the data.

Under-fitting occurs when the model performs poorly on the training data, typically indicated by a high training loss. Its counterpart, overfitting, denotes a scenario where the model performs well on the training data but fails to generalize to unseen test or new data.

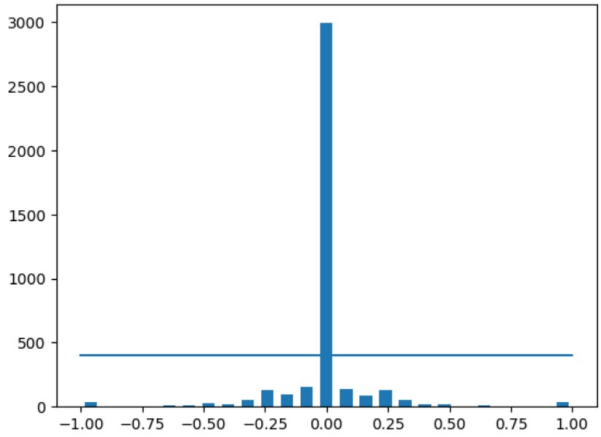


Figure 3: data set collected

To address this issue, we decided to remove some images with a steering angle of 0 (indicating the car is driving straight) from the dataset. The resulting graph(figure 4) illustrates this refinement.

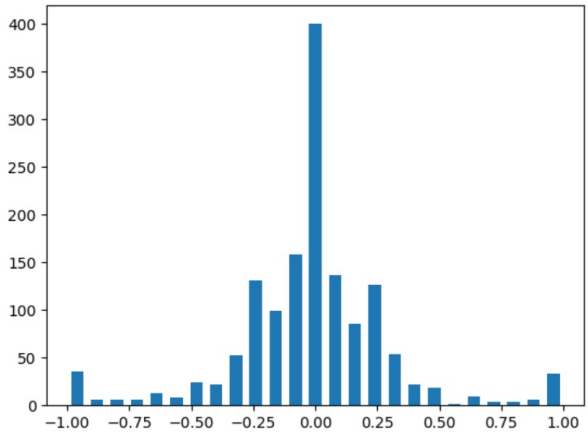


Figure 4: data set updated

And here, we have achieved a desirable distribution of data, where it appears to be evenly spread out. This balanced distribution facilitates effective

training of the model. After that we split the data into train and validation set .

training set: it is 80% of the whole data, and it is used for the training (the ones that passes through the neural network during the training stage) validation set: 20% of the data, and it is used for evaluating the model's performance before deployment

6 image prepossessing and data augmentation:

Before feeding the data the neural network we pass it through multiple image processing techniques (resizing, filtering, rotation .....), this allows the model to learn extra features and also reduce the number of calculations ..... All these techniques used in this project are described bellow.

1. **Panning:**
   Consists of moving the image in the horizontal or vertical direction.

Figure 5: the effect of panning on the image

2. **flipping:**
   in this process, we just flip the image around the axis and at the same time we multiply the steering angle by -1(the direction is reversed)

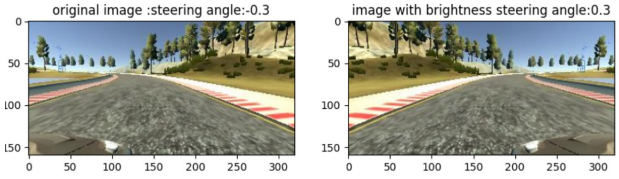


Figure 6: flipping the image

3. **RGB to YUV conversion:**
   Converting images from the RGB format to the YUV format is a common practice in various computer vision applications, including autonomous vehicles. In dynamic environments where lighting conditions can change rapidly, such as transitioning from daylight to nighttime or encountering shadows and glare, it's crucial for algorithms to adapt effectively. YUV separates luminance (brightness) from chrominance (color), providing a more robust representation of the scene. By decoupling color information from brightness, algorithms can better distinguish objects from background and handle variations in lighting conditions more effectively. This conversion enhances the reliability of tasks such as object detection, lane tracking, and obstacle avoidance in autonomous driving systems.

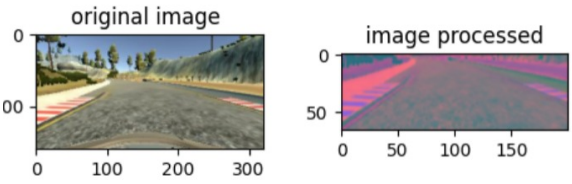


Figure 7: converting the image to UYV format

4. **Gaussian blur:**
   Gaussian blur is a popular image processing technique used to reduce noise and detail in an image while preserving edges. It works by applying a convolution operation with a Gaussian kernel to the input image

what is a gaussian filter:

A Gaussian filter is a type of low-pass filter commonly used in image processing to reduce noise and blur images. It works by attenuating high-frequency components while preserving edges.

(a) guassian filter matrix

(b) gaussian distribution

Figure 8: gaussian filter

7 Nvidea model:

before demonstrating the nvidea model we will have a look on two methods used in cnn:

1. striding: refers to the number of pixels the kernel shifts over the input matrix during convolution.

2. padding: Padding is introduced to preserve the size of the original image. It involves adding extra pixels (usually zeros) around the input image before convolution.

The model used in this project is the NVIDIA model, developed by the research team at NVIDIA. The network architecture is illustrated in the figure. The NVIDIA model comprises 5 convolutional layers, followed by 3 fully connected layers, and an additional normalization layer at the beginning, totaling 9 layers. The convolutional layers are specifically designed for feature extraction based on the results of multiple experiments conducted by the research team. The first three convolutional layers employ strided convolutions with a 2×2 stride and a 5×5 kernel, while the last two convolutional layers utilize non-strided convolutions with a 3×3 kernel size.

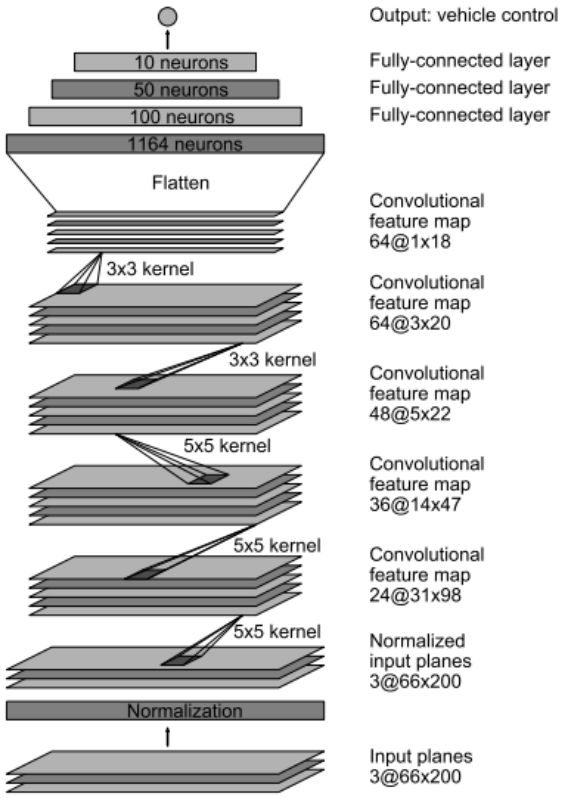


Figure 9: NVIDIA model architecture

training the model:

As previously indicated, the methodology entails feeding the data through a neural network architecture inspired by the NVIDIA model. This process involves employing the Adam optimizer algorithm to attractively adjust the model parameters during the back propagation stage of training, ensuring efficient convergence towards optimal performance by minimizing the mean square error loss.

Stochastic gradient descent (SGD) is an optimization algorithm commonly used during the training of neural networks. Its key idea involves passing a single data point (or a small batch) of the training data through the network to calculate the gradient descent and update the parameters. This process differs from batch gradient descent, which necessitates passing through all the data points before updating the parameters, making it more time-consuming.

adaptive moment estimation (adam) optimizer: a method for efficient stochastic optimization that only requires first-order gradients with little memory requirement. The method computes individual adap-

Algorithm 1 Stochastic Gradient Descent

Require: Learning rate η.
Require: Initial parameter θ.
while Stopping criterion not met **do**
  Sample a minibatch of m examples from the training set $\{x^{(1)}, \ldots, x^{(m)}\}$.
  Set $g = 0$
  for $i = 1$ to m **do**
    Compute gradient estimate:

$$g \leftarrow g + \nabla_\theta L(f(x^{(i)}; \theta)), y^{(i)}; \theta).$$

  end for
  Apply update: $\theta \leftarrow \theta - \eta g$
end while

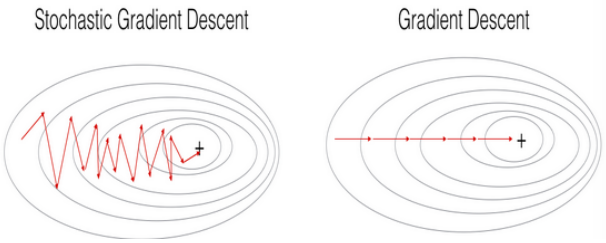


Figure 10: batch and stochastic gradient descent steps

tive learning rates for different parameters from estimates of first and second moments of the gradients. for this reason it was the best choice for our project. the loss function used during the training is the mean

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Figure 11: ADAM optimizer algorithm

square error loss(mse). mean square error: measures the average squared difference between the estimated values and the actual observed values, and the objective of the training is to minimize as much a s possible its amount(make it close to zero).

as mentioned before, the data is split into train and

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

$* n$ is the number of data points
$* Y_i$ represents observed values
$* \hat{Y}_i$ represents predicted values

Figure 12: mean square error(mse) equation

validation sets. this means that the evaluation of the system is based on the loss of both sets(training and validation losses). and acoording the their values we can decide if the system is under-fitting or over-fitting:
High tr-loss and high valid-loss → Under-fitting
low tr-loss and high valid-loss → over-fitting
low tr-loss and low valid-loss → good performance
after training the medel we get the losses and they are ploted in the graph bellow :

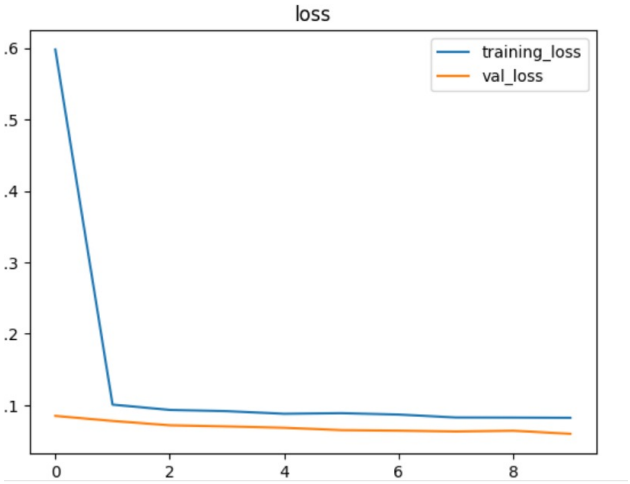


Figure 13: train and validation loss

As we can see from the graph, the trained model has an acceptable loss value (around 0.08). This value could be even lower if we had used a larger dataset.

Additionally, the validation and training losses are close to each other, indicating that there is no overfitting. The only way to test the results is to deploy our model on the Udacity simulator and let it drive the car without any intervention.
However, it's essential to remember that real-world scenarios can be much more complex and diverse than simulated environments.

Figure 14: Udacity environment

8 Conclusion

During this project, we trained a neural network designed according to the Nvidia model architecture, specifically tailored for autonomous vehicle tasks. Through our experiments and evaluations, we have demonstrated the efficacy of the convolutional neural network (CNN) in learning complex features necessary for autonomous vehicles. Our results show promising performance in both lane detection and obstacle recognition tasks, showcasing the potential of CNNs in enhancing autonomous driving systems.

9 References:

@article end to end machine learning in self driving care, URL = https://arxiv.org/abs/1604.07316,
@article Adam: A Method for Stochastic Optimization, URL = https://arxiv.org/abs/1412.6980,
@bookHands-on machine learning with scikit learn Keras and tensorflow, author = Aurélien Géron,