Coursework 2 Report: Traffic Light Classification Using CNN

ID: 2270587

1. Introduction

Accurately recognizing traffic lights is extremely important in the development of autonomous vehicles (AVs), as it is very impactful on the safety and efficiency of their operation. A major target in AVs is making sure no accidents happen due to red light running, which may lead to extremely bad consequences such as crashes, injuries and fatalities. In this project, the main goal is to design and implement an image classification system that can identify the two classes of traffic lights (red and green). Using a deep learning model like Convolutional Neural Network (CNN), this project aims to achieve high accuracy in its classification task. This model is made to be deployed in real world scenarios where AVs are relied upon to make good accurate decisions based on the colour of traffic lights.

2. Data Preparation

Before the model is built, it is very important to prepare the dataset and clean it out. The dataset consists of images of traffic lights that need to be split into two classes of green and red. The cleaning of data enables the model to have access to clean data, different and varied data to learn from, and it also helps to improve the model's ability to predict from unseen images.

2.1 Data Augmentation and Dataset Splitting

First step to clean the data was the implementation of data augmentation, this is done to increase the diversity of the data and prevent overfitting. As the data may not be particularly large or diverse, data augmentation helps simulate different versions or scenarios that the model could come across in real world cases. The transformations made are:

- Rotation: this is done so that the model is not dependant on the traffic light being upright as it may very well not be in the real world, so rotation of the images by small angles was done.
- Flipping: flipping the image horizontally was applied to make the model constant to the left-right orientation of the traffic light.
- Zooming and Shifting: Random shifting and zooming was done to cover different distances and positions that the traffic lights may be in the frame.
- Brightness and Contrast Adjustments: Even though this was not included in the implementation of the project, this transformation could be implemented for potential future improvements, as it helps the model adapt to different weather conditions.

3. Model Design

The model used in this project was a Convolutional Neural Network (CNN), it is a deep learning model that is very effective for image classification tasks specifically. CNNs use convolutional layers to learn features from images automatically without the need for manual feature extraction. Which is why CNNs are a very good fit for tasks like object detection and classification.

3.1 CNN Architecture

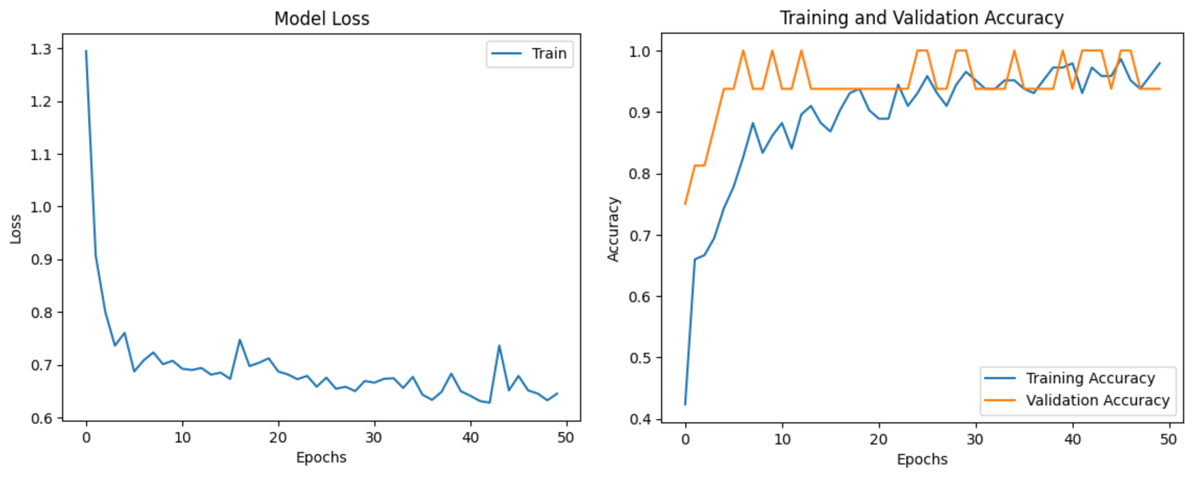
The CNN architecture was designed with several key layers, including:

- Convolutional Layers: Convolutional layers add various filters to the input images, this allows models to detect edges, textures, and more advanced patters such as traffic lights.
- Max-Pooling Layers: Max pooling layers reduce the spatial dimensions of the feature maps, this helps in decreasing computational cost and also extract important features.
- Fully Connected Layers: These layers connect the extracted features to the final classification output. In context of this project, the model outputted two classes of red and green.
- Dropout Layer: To prevent overfitting, a dropout layer was added to randomly disable a fraction of the neurons during training. This encouraged the model to learn more robust features.
- Softmax Activation Function: For the final layer, the softmax activation function to make probabilities for each class. The class with the highest probability was selected as the predicted label.

4. Training the Model

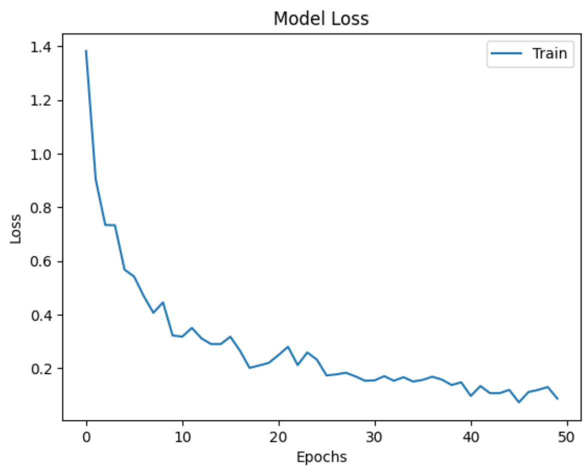
After the data preparation and the design of the model were complete, the next step is to train the model. The training process works by feeding the model the training images and adjusting the weights of the model through backpropagation to minimize the loss function. The loss function that was used is the categorical cross-entropy, which is very commonly used for multi-class classification tasks. At each epoch during the training the model's performance was evaluated, and through these evaluations, adjustments were made to the model's parameters to improve its accuracy.

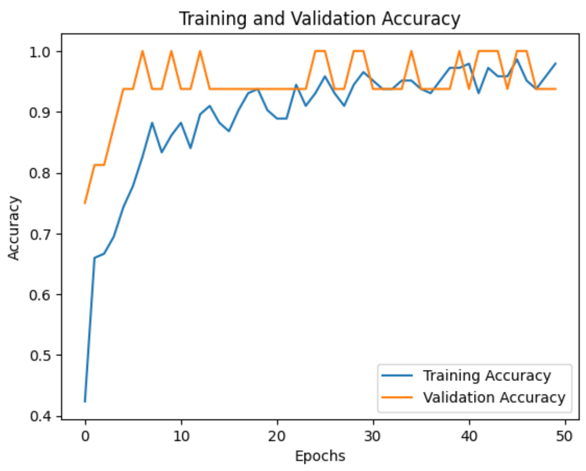
Figure 1 - Model Loss, and Accuracy Graphs Before Improvements



4.1 VGG Implementation

To improve on the model, the first implementation done was VGG architecture. It is known for its simplicity and depth, as it uses smaller convolutional kernels to effectively get patterns. The model was then fine tuned with the dataset to enhance its learning features. Implementing VGG has shown good improvements in the loss which shows how suitable it is for image classification. On the other hand, it took more time to train and still required more computational power which are negatives to be considered for its use.





4.2 Hyperparameter Tuning

Hyperparameter tuning was implemented to make sure the model performed as required. This process involved the testing of different combinations of hyperparameters to identify the best configuration for the model. The hyperparameters that were tested included:

- Batch Size: This is the number of images processed at once in each training step. The batch sizes that were tested are 16, 32, and 64. They were tested to find the best balance between the computational efficiency and model performance.
- Epochs: The number of times the training dataset was passed through the model. The higher the number of epochs, the more that the model learns more from the data. However, if the epochs are too high the model can be exposed to overfitting.
- Learning Rate: The rate in which the model's weights are adjusted during the training process. Learning rates of 0.001 and 0.0001 were both tested to find between them

the best rate that allowed the model to converge quickly without looking over the optimal solution.

After testing all the combinations, the best hyperparameters were found to be a batch size 16, 20 epochs, and a learning rate of 0.001. With these applied, the model was able to achieve a validation accuracy score of 82.5%.

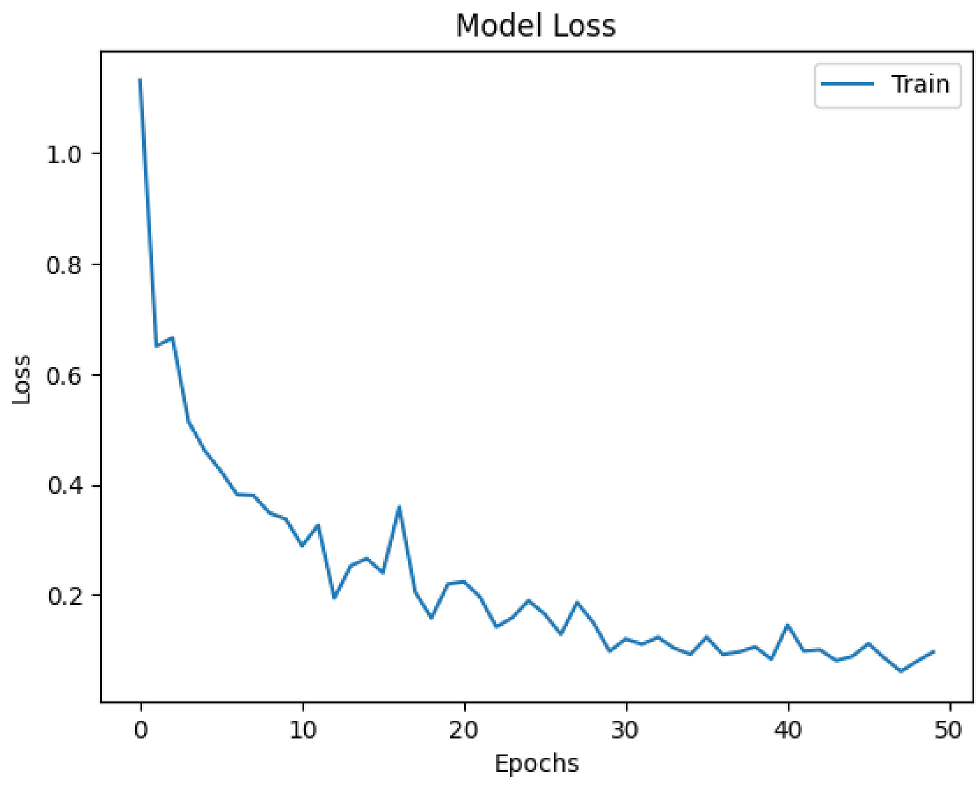
5. Model Evaluation

After training, the model was evaluated on the test dataset to assess its performance on unseen data. The evaluation metrics used were accuracy and loss, which provide insights into the model's overall performance and its ability to classify traffic light images correctly.

5.1 Accuracy and Loss Analysis

Training and validation accuracies were tracked over multiple epochs to evaluate the performance of the model during training. This enabled the observation of the model's progress and determining whether the model was under or overfitting.

At the end of the training, the model achieved an accuracy of 82.5% on the validation dataset. The loss curve shown below also showed a consistent decrease in the loss over time, this shows that the model was successfully learning to classify the traffic light images.



5.2 Testing the Model

To test the model's ability to generalize, five images have been selected for testing that were not part of the training dataset. These images were chosen to evaluate how well the model is able to classify unseen data. The following are the predicted labels for each of the images (Images used were 15, 17, 19, 13, and 10):

- Image 1: Predicted label: Green
- Image 2: Predicted label: Red
- Image 3: Predicted label: Green
- Image 4: Predicted label: Red
- Image 5: Predicted label: Green

The model successfully predicted the traffic light colour for all five test images, which shows its ability to generalize well to new data.

6. Suggestions for Further Improvement

Even though the model has done well, there are still areas of improvement that can be tackled in a few ways.

6.1 Advanced Data Augmentation

What was implemented are more basic methods such as rotation, zoom, flipping, there are more techniques more advanced that could be implemented. For example, implementing brightness and contrast adjustments or adding Gaussian noise could allow the model to be more accurate in different lighting conditions and environmental factors. These additional augmentations would ensure that the model can handle a lot of real-world conditions, where traffic lights may be seen under different lighting or weather conditions.

6.2 Model Architecture Improvements

CNN is a perfectly excellent architecture to be used for this project, however further improvements could be made by using more advanced or deeper architectures. The use of pre-trained models such as VGG16 and ResNet50 which have been trained on large image datasets such as ImageNet. These models have the ability to transfer the features they have learned to new tasks, such as traffic light classification, which would lead to higher accuracy with less training epochs. Fine-tuning a pre-trained model could significantly enhance performance, especially when the dataset is relatively small.

7. Conclusion

In this project, a Convolutional Neural Network (CNN) model has been built successfully in order to classify traffic light images into two classes (red and green). This model was trained with a dataset consisting of traffic light images, and after dataset cleaning and preparation, designing the model, and hyperparameter tuning, the model was able to score a validation accuracy of 82.5%. Moreover, the model has been able to accurately classify images that are unseen, showing its generalization ability.

However, the model can still be improved further using more advanced methods of data augmentation and also using pre-trained models. These two ideas being implemented would positively impact the accuracy score and the model's robustness, which would make this model more suitable to be utilized in real AV systems.