UNIVERSITY PARTNER

UNIVERSITY OF WOLVERHAMPTON

HERALD COLLEGE KATHMANDU

Image Classification of Car Brand Logos Using Convolutional Neural Networks (6CS012)

Student Id	: 2330746
Student Name	: Prajin Singh
Group	: L6CG6
Tutor	: Mr. Shiv Kumar Yadav
Reader	: Mr. Siman Giri
Cohort	: 9
Submitted on	: 11/05/2025

Abstract

The objective of the project was to design a Convolution Neural Network to classify various car brands logos. The project uses 3 models and two variants of optimizer for training and testing. The first model was a baseline model with 3 convolution layers, a flatten layer and 3 dense layers. It was trained using the Adam optimizer and categorical cross-entropy loss, with data augmentation to improve the model's generalization on unseen data. The baseline model achieved an accuracy of 0.6075 and a loss of 1.5264 on the test data.

The second model was a deeper version of the baseline model, its architecture doubled the number of layers and incorporated regularization techniques such as L2 regularization, Batch Normalization, and Dropout to address the overfitting issues observed in the baseline model.

The third model utilized transfer learning with the VGG16 architecture, where the original layers were frozen to prevent weight updates during training and custom layers were added for fine-tuning.

As model complexity increased, both efficiency and execution time improved, while overfitting issues decreased. All the models performed well on unseen data, corresponding to their architectural design. Future work could include increasing the dataset size and introducing more variation in the data distribution.

Contents

Table of Figures

1. Introduction

In the real world, recognizing car brand logo can be valuable for autonomous traffic monitoring and vehicle tracking. However, accurately identifying logos under various conditions is challenging due to differences in logo design, image quality, and environmental factors such as lighting and background noise.

The project focuses on addressing such challenges by classifying car brand logos from images using Convolution Neural Network (CNN). CNNs are especially effective for image classification because they can capture spatial features using feature maps, making them ideal for detecting patterns in car logos.

A recent study by Saeed R. Saeed, Alyaa Hashem Mohammed, Shahad Khalid Khaleel, Aqeel Ali Al-Hilali (Saeed R. Saeed1, 2023) focused on using deep learning to recognize car logos. They worked with a dataset containing 15 car brand classes, initially with 400 images per class, later augmented to 800 images to prevent overfitting. To train and predict the logo, they utilized two models ResNet50, a pretrained model on ImageNet, was used to extract features. The model is modified by adding four ResNet convolution layers with ReLU activation, a dense layer with 0.5 dropout and a sigmoid output for binary classification. The main classification task is operated using a VGGNet-based CNN which has 16-19 layers of 3*3 convnet and 2*2 max pooling. The system works in two phases, first to preprocess video frames, locate license plates and logos and extract ROIs and classification using the VGGNet model. The fine-tuned VGGNet achieves over 10% higher average precision (AP) than the original VGGNet model, demonstrating enhanced accuracy and robustness in recognizing car logos.

The goal of the project is to design three different deep learning models like baseline, deeper model with regularization and transfer learning that can accurately classify car brand logos from images. While the scope of the project is to classify logos from images using a public dataset, testing three models and comparing their performance.

2. Dataset

The dataset used for this project is publicly available in Kaggle (source: https://www.kaggle.com/datasets/volkandl/car-brand-logos?select=Car_Brand_Logos) and was created by Volkan Özdemir. It contains a total of 2,913 images categorized into 8 car brand classes. The training set includes:

- 302 Hyundai,
- 301 Lexus,
- 317 Mazda,
- 342 Mercedes,
- 301 Opel,
- 314 Skoda,
- 306 Toyota,
- 330 Volkswagen

The test set consists of 50 images per class.

The resolutions in the dataset vary with a maximum resolution of 6000 width and 4000 heights and a minimum of 64*64.

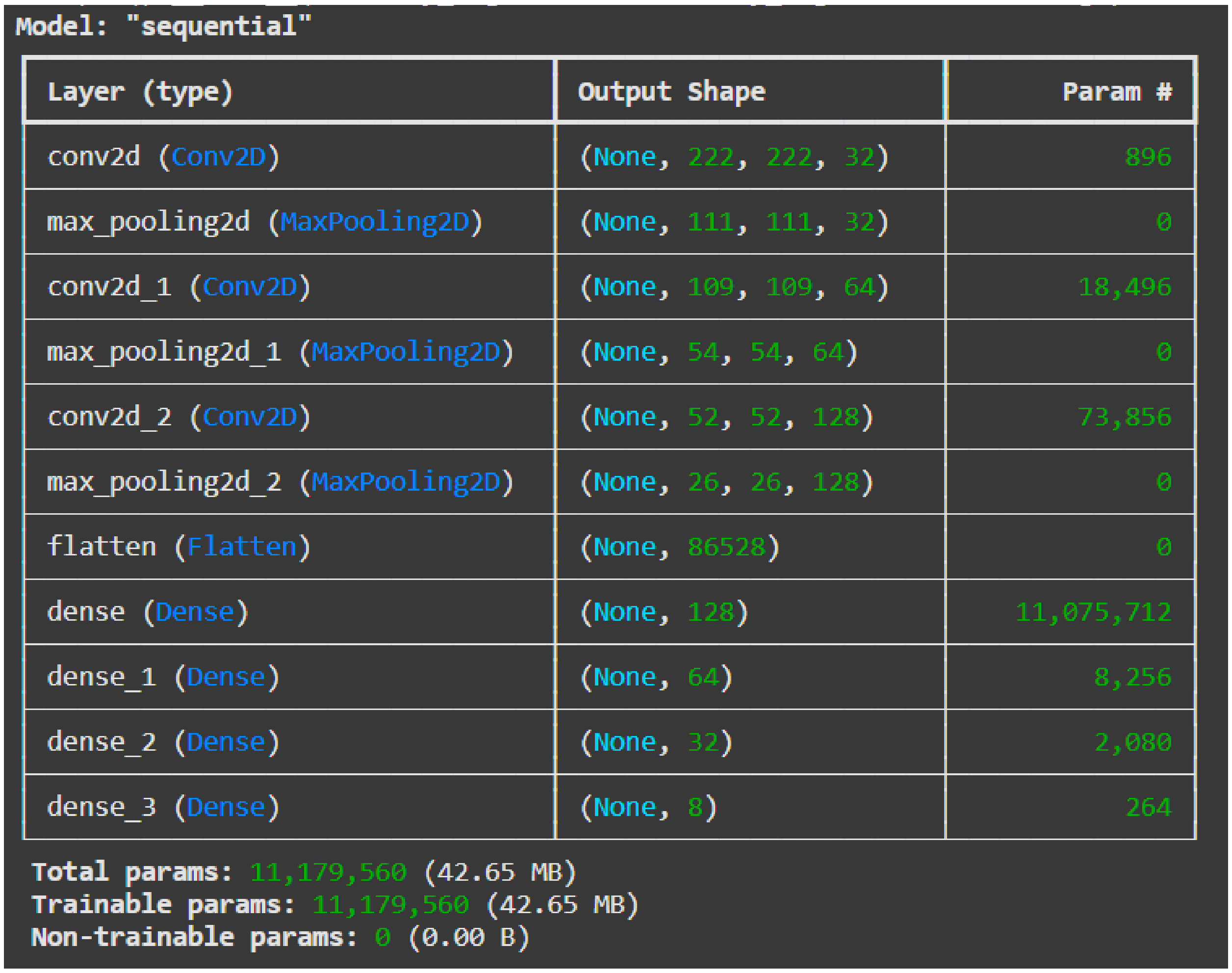
Several preprocessing techniques were applied to prepare the data for training:

- Verifying the images to eradicate the corrupt images
- Data was augmented in the training sample to increase the data distribution
- Images were resized to 224*224 pixels
- All pixel values were normalized to a 0-1 range

3. Methodology

The project was conducted on three models and two optimizers to evaluate the performance of all the models.

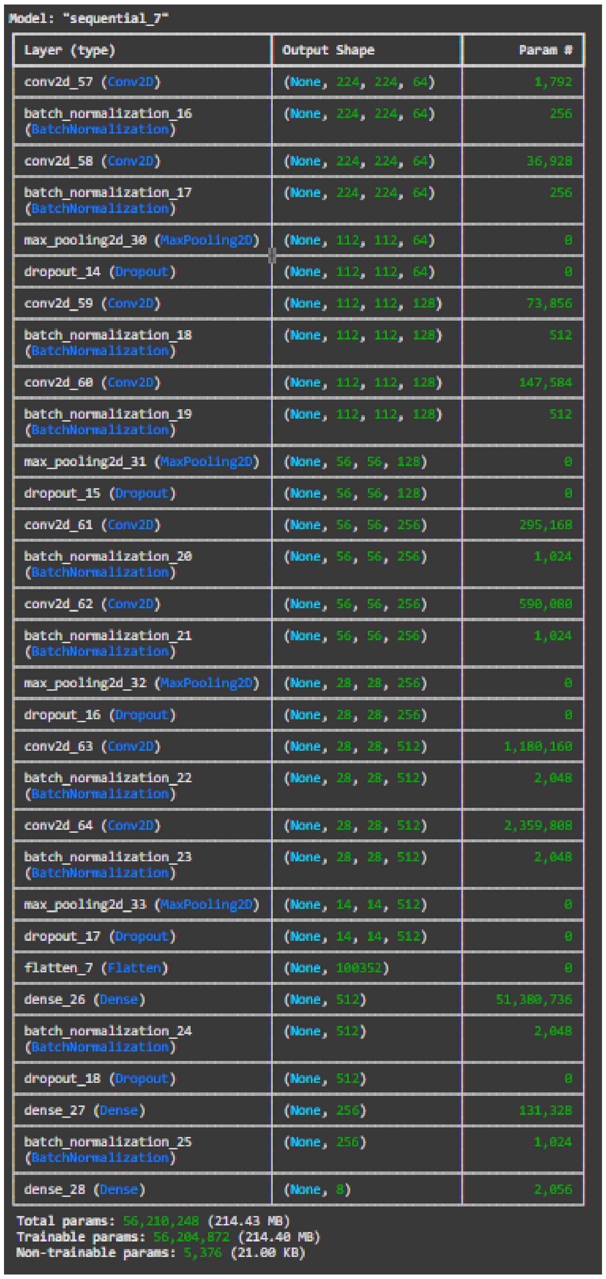
- **Baseline model:** The baseline model is the most basic model of 3 convolution layers each followed by a max pooling layer, a flatten layer and 3 fully connected network excluding the output layer. The model doesn't include regularization techniques to observe its raw performance on the dataset. The model was compiled using Adam optimizer (learning rate: 0.001) with sparse categorical cross entropy loss function for measuring loss. The model training configuration included 32 batches over 50 epochs with model checkpoint, early stopping callbacks to improve training stability. Below is the model summary with the layer-wise configuration and number of trainable parameters:



```
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ conv2d (Conv2D)                 │ (None, 222, 222, 32)   │           896 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d (MaxPooling2D)    │ (None, 111, 111, 32)   │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_1 (Conv2D)               │ (None, 109, 109, 64)   │        18,496 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_1 (MaxPooling2D)  │ (None, 54, 54, 64)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_2 (Conv2D)               │ (None, 52, 52, 128)    │        73,856 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_2 (MaxPooling2D)  │ (None, 26, 26, 128)    │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ flatten (Flatten)               │ (None, 86528)          │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense (Dense)                   │ (None, 128)            │    11,075,712 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ (None, 64)             │         8,256 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_2 (Dense)                 │ (None, 32)             │         2,080 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_3 (Dense)                 │ (None, 8)              │           264 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 11,179,560 (42.65 MB)
 Trainable params: 11,179,560 (42.65 MB)
 Non-trainable params: 0 (0.00 B)
```

Figure 1 Summary of Baseline model

- **Deeper model:** The deeper model extends the baseline model by doubling the convnets numbers of convolutional layer and adding regularization techniques like l2, batch normalization and dropout to improve the model generalization. The max filter numbers also increased from 128 to 512. While training the model was trained into two variants of optimizer which are Adam and Stochastic Gradient Descent (SGD). The separation allowed a direct comparison of how differently the optimizer affects the same network architecture in terms of convergence speed and loss reduction. Similar to base model, the deeper model also uses same loss function but additionally uses ReduceLROnPlateau in callbacks for reducing learning rate when "val_loss" metric has stopped improving. Below is the model summary with the layer-wise configuration and number of trainable parameters:



```
Model: "sequential_7"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ conv2d_57 (Conv2D)              │ (None, 224, 224, 64)   │         1,792 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_16          │ (None, 224, 224, 64)   │           256 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_58 (Conv2D)              │ (None, 224, 224, 64)   │        36,928 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_17          │ (None, 224, 224, 64)   │           256 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_30 (MaxPooling2D) │ (None, 112, 112, 64)   │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_14 (Dropout)            │ (None, 112, 112, 64)   │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_59 (Conv2D)              │ (None, 112, 112, 128)  │        73,856 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_18          │ (None, 112, 112, 128)  │           512 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_60 (Conv2D)              │ (None, 112, 112, 128)  │       147,584 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_19          │ (None, 112, 112, 128)  │           512 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_31 (MaxPooling2D) │ (None, 56, 56, 128)    │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_15 (Dropout)            │ (None, 56, 56, 128)    │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_61 (Conv2D)              │ (None, 56, 56, 256)    │       295,168 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_20          │ (None, 56, 56, 256)    │         1,024 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_62 (Conv2D)              │ (None, 56, 56, 256)    │       590,080 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_21          │ (None, 56, 56, 256)    │         1,024 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_32 (MaxPooling2D) │ (None, 28, 28, 256)    │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_16 (Dropout)            │ (None, 28, 28, 256)    │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_63 (Conv2D)              │ (None, 28, 28, 512)    │     1,180,160 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_22          │ (None, 28, 28, 512)    │         2,048 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_64 (Conv2D)              │ (None, 28, 28, 512)    │     2,359,808 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_23          │ (None, 28, 28, 512)    │         2,048 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_33 (MaxPooling2D) │ (None, 14, 14, 512)    │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_17 (Dropout)            │ (None, 14, 14, 512)    │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ flatten_7 (Flatten)             │ (None, 100352)         │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_26 (Dense)                │ (None, 512)            │    51,380,736 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_24          │ (None, 512)            │         2,048 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_18 (Dropout)            │ (None, 512)            │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_27 (Dense)                │ (None, 256)            │       131,328 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_normalization_25          │ (None, 256)            │         1,024 │
│ (BatchNormalization)            │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_28 (Dense)                │ (None, 8)              │         2,056 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 56,210,248 (214.43 MB)
 Trainable params: 56,204,872 (214.40 MB)
 Non-trainable params: 5,376 (21.00 KB)
```

Figure 2 Summary of Deeper Model

4. Experiments and Results

4.1 Baseline vs. Deeper architecture

In the baseline model, the validation accuracy remained stable for the first 4 epochs then the model started to oscillate from epoch 5. As training progressed, the model indicated overfitting, as seen in the gap between training and validation accuracy and loss curves. Whereas the deeper model trained with Adam optimizer not overfit but a significant fluctuation in the training and validation accuracy and loss curve were prominent. However, the loss was better optimized in the baseline model due to its lower complexity.

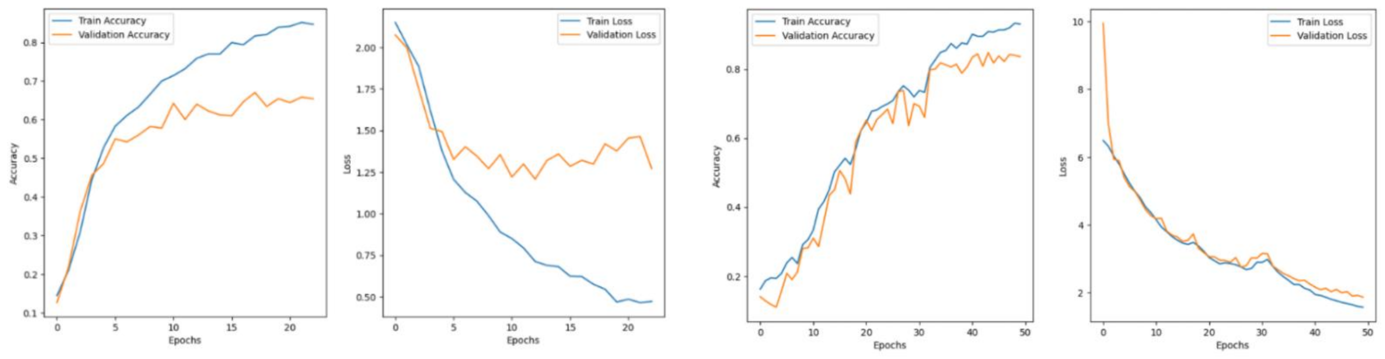
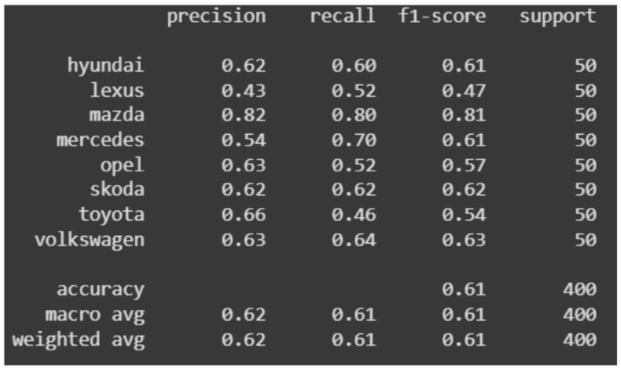


Figure 3 Graphs of training and validation accuracy and loss

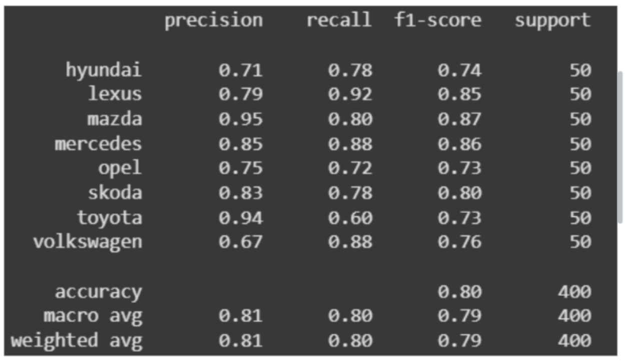
When comparing the baseline model classification report with the deeper model, a clear improvement can be noticed in the performance. The deeper model has better precision for most car brands, for instance, Hyundai improves from 0.62 to 0.71 and Lexus from 0.43 to 0.79. Recall also improved, as seen with an increase from 0.46 to 0.60 for Toyota and 0.64 to 0.88 for Volkswagen. The F1-score also shows positive incline in deeper model like Mazda inclining from 0.81 to 0.87 and Skoda from 0.62 to 0.80. Overall, the deeper model's accuracy increases from 0.61 to 0.80 which is significantly better than the base model. Hence, using more layers and filters improved the performance of the model.



```
              precision    recall  f1-score   support

     hyundai       0.62      0.60      0.61        50
       lexus       0.43      0.52      0.47        50
       mazda       0.82      0.80      0.81        50
    mercedes       0.54      0.70      0.61        50
        opel       0.63      0.52      0.57        50
       skoda       0.62      0.62      0.62        50
      toyota       0.66      0.46      0.54        50
  volkswagen       0.63      0.64      0.63        50

    accuracy                           0.61       400
   macro avg       0.62      0.61      0.61       400
weighted avg       0.62      0.61      0.61       400
```



```
              precision    recall  f1-score   support

     hyundai       0.71      0.78      0.74        50
       lexus       0.79      0.92      0.85        50
       mazda       0.95      0.80      0.87        50
    mercedes       0.85      0.88      0.86        50
        opel       0.75      0.72      0.73        50
       skoda       0.83      0.78      0.80        50
      toyota       0.94      0.60      0.73        50
  volkswagen       0.67      0.88      0.76        50

    accuracy                           0.80       400
   macro avg       0.81      0.80      0.79       400
weighted avg       0.81      0.80      0.79       400
```

Figure 4 Classification Report

4.2 Computational Efficiency

Even though the deeper model performs better than the baseline model, it is more computationally expensive. The baseline model took 21 minutes to complete the training time while the deeper model completed in 45 minutes. The increase in time is mainly due to the additional numbers of filters and layers used in the deeper model, which made the network more complex. This shows a clear trade-off between accuracy and speed. To achieve the desired results with different models, the project was trained using a T4 GPU on Google Colab.

4.3 Training with Different Optimizers.

In the SGD optimized model, the training accuracy started from 0.15 and reached above 0.75, while the validation accuracy started from 0.12 and reached around 0.6. This gap between training and validation depicts a visible overfitting. The loss in training was optimized in a stable way but in validation loss oscillated, dropping from 5.5 to 4.5.

While in the Adam optimized model, the graph showed some fluctuations, but there was no major gap between the training and validation. Both training and validation accuracy reached above 0.8, and the loss in both cases dropped below 2, showing faster and more balanced learning with better overall performance.

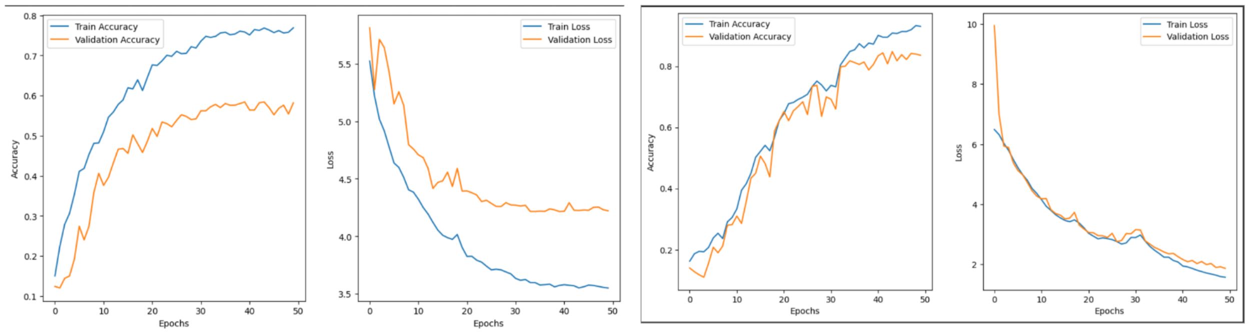
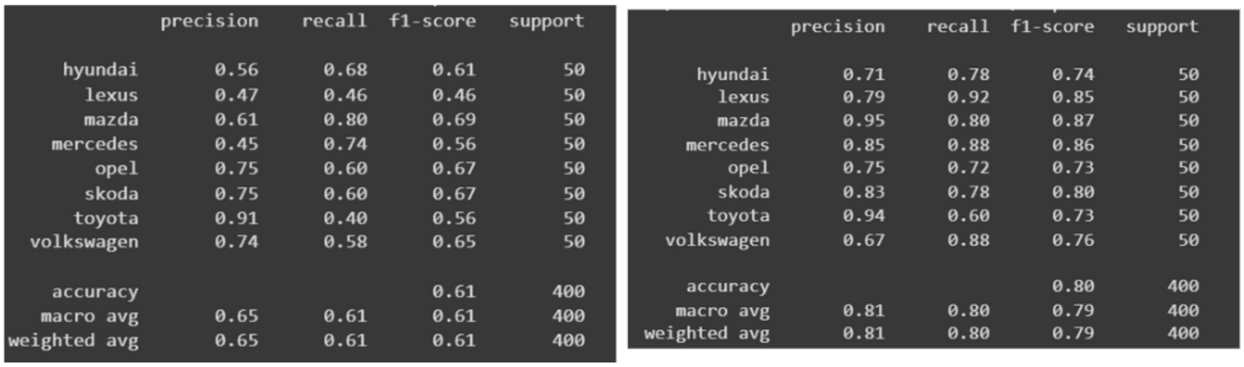


Figure 5 Graph of SGD and Adam

When comparing the classification reports, a clear difference in performance can be observed based on precision, recall, F1-score and overall accuracy across the 8 classes. The SGD model had an overall accuracy of 0.61 with macro average precision, recall and F1-score all around 0.61 – 0.65. Toyota had high precision (0.91) but lower recall (0.40) which resulted that it missed many actual Toyota samples. Other brands like Hyundai, Mazda, Skoda and Opel had balanced F1-scores (around 0.67) but the recall was still low in general.

In contrast, the Adam model performed much better as it reached 0.80 overall accuracy and it did well almost across all brands. Toyota reached 0.94 precision and 0.60 recall, Hyundai had an F1-score of 0.74 and brand like Mercedes and Skoda also performed well. The classification report highlights that Adam model predicted more correctly while missing only fewer samples.



```
              precision    recall  f1-score   support

     hyundai       0.56      0.68      0.61        50
       lexus       0.47      0.46      0.46        50
       mazda       0.61      0.80      0.69        50
    mercedes       0.45      0.74      0.56        50
        opel       0.75      0.60      0.67        50
       skoda       0.75      0.60      0.67        50
      toyota       0.91      0.40      0.56        50
  volkswagen       0.74      0.58      0.65        50

    accuracy                           0.61       400
   macro avg       0.65      0.61      0.61       400
weighted avg       0.65      0.61      0.61       400
```

```
              precision    recall  f1-score   support

     hyundai       0.71      0.78      0.74        50
       lexus       0.79      0.92      0.85        50
       mazda       0.95      0.80      0.87        50
    mercedes       0.85      0.88      0.86        50
        opel       0.75      0.72      0.73        50
       skoda       0.83      0.78      0.80        50
      toyota       0.94      0.60      0.73        50
  volkswagen       0.67      0.88      0.76        50

    accuracy                           0.80       400
   macro avg       0.81      0.80      0.79       400
weighted avg       0.81      0.80      0.79       400
```

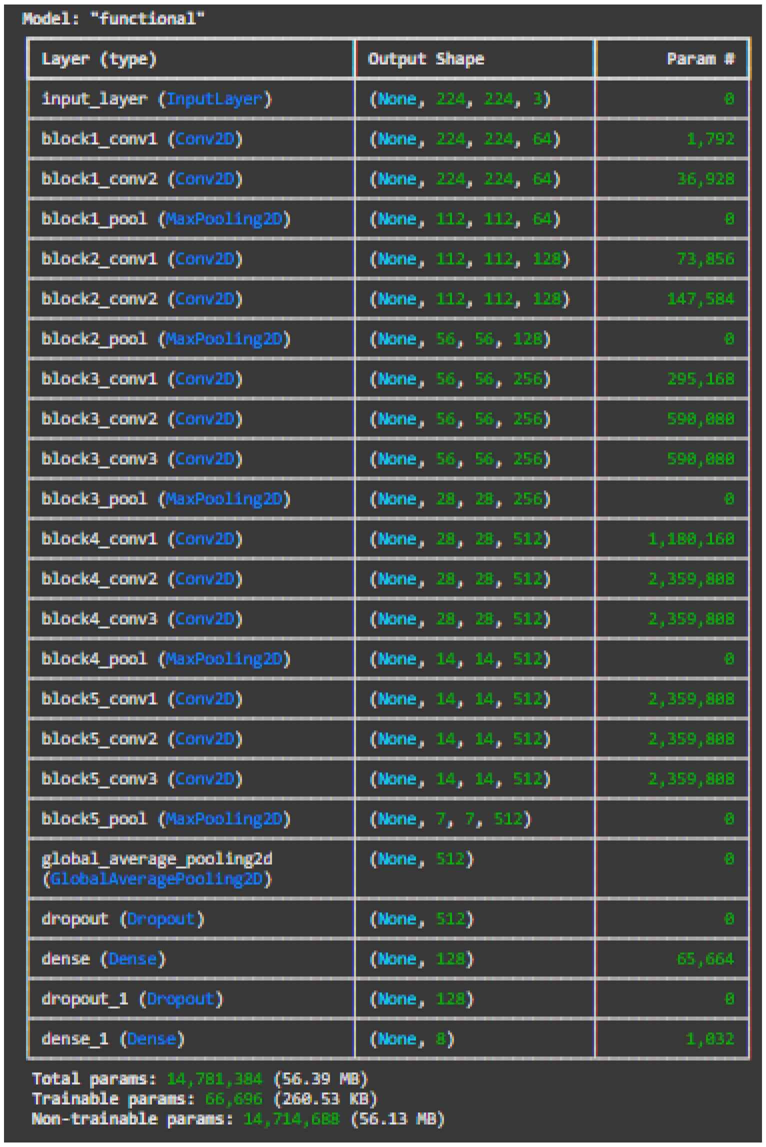
Figure 6 Classification report of SGD and Adam

4.4 Challenges in Training

Several challenges were encountered during the training process. First the dataset was not vast, which led to overfitting, especially in the baseline model. To overcome this image augmentation techniques were applied to increase the training data and improve generalization. Despite data augmentation, the baseline model still performed poorly so a deeper model was built. The deeper model handles the overfitting issue with regularization techniques and increasing the number of filters and layers. Lastly another challenging task was computational time it required, due to deeper model's complex architecture the training time compared to base model increased by double. The base model took around 21 minutes while the deeper model completed the training in 45 minutes with Adam and 50 minutes with SGD.

5. Fine – Tuning or Transfer Learning

For the third model, the project utilizes a pre-trained model known as VGG16 which was trained in the ImageNet competition. All the layers of the model were frozen, to prevent the architecture from updating weight and the original output layer was removed. Additional layers like GlobalAveragePooling, Dense and Dropout layers are added on top of base model layer with the project classification layer of 8 classes. It utilizes the same configuration as deeper model with Adam in, but the learning rate is tweaked to 1e-5. For training, the last 20 layers are unfrozen and finetuned so that the model would capture higher level features by updating their weights and learning features. Below is the fine-tuning model summary:



```
Model: "functional"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ input_layer (InputLayer)        │ (None, 224, 224, 3)    │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block1_conv1 (Conv2D)           │ (None, 224, 224, 64)   │         1,792 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block1_conv2 (Conv2D)           │ (None, 224, 224, 64)   │        36,928 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block1_pool (MaxPooling2D)      │ (None, 112, 112, 64)   │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block2_conv1 (Conv2D)           │ (None, 112, 112, 128)  │        73,856 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block2_conv2 (Conv2D)           │ (None, 112, 112, 128)  │       147,584 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block2_pool (MaxPooling2D)      │ (None, 56, 56, 128)    │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block3_conv1 (Conv2D)           │ (None, 56, 56, 256)    │       295,168 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block3_conv2 (Conv2D)           │ (None, 56, 56, 256)    │       590,080 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block3_conv3 (Conv2D)           │ (None, 56, 56, 256)    │       590,080 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block3_pool (MaxPooling2D)      │ (None, 28, 28, 256)    │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block4_conv1 (Conv2D)           │ (None, 28, 28, 512)    │     1,180,160 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block4_conv2 (Conv2D)           │ (None, 28, 28, 512)    │     2,359,808 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block4_conv3 (Conv2D)           │ (None, 28, 28, 512)    │     2,359,808 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block4_pool (MaxPooling2D)      │ (None, 14, 14, 512)    │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block5_conv1 (Conv2D)           │ (None, 14, 14, 512)    │     2,359,808 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block5_conv2 (Conv2D)           │ (None, 14, 14, 512)    │     2,359,808 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block5_conv3 (Conv2D)           │ (None, 14, 14, 512)    │     2,359,808 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ block5_pool (MaxPooling2D)      │ (None, 7, 7, 512)      │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ global_average_pooling2d        │ (None, 512)            │             0 │
│ (GlobalAveragePooling2D)        │                        │               │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout (Dropout)               │ (None, 512)            │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense (Dense)                   │ (None, 128)            │        65,664 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_1 (Dropout)             │ (None, 128)            │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ (None, 8)              │         1,032 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 14,781,384 (56.39 MB)
 Trainable params: 66,696 (260.53 KB)
 Non-trainable params: 14,714,688 (56.13 MB)
```

Figure 7 Summary of Fine Tunning

Compared to the other two models, transfer learning model's validation accuracy starts high, above 0.85, remains stable throughout training and eventually reaches above 0.90 and the validation loss also declines from a very low number of 0.5 to below 0.4. The overall accuracy increases of the model to 0.86 which is highest among all three models.

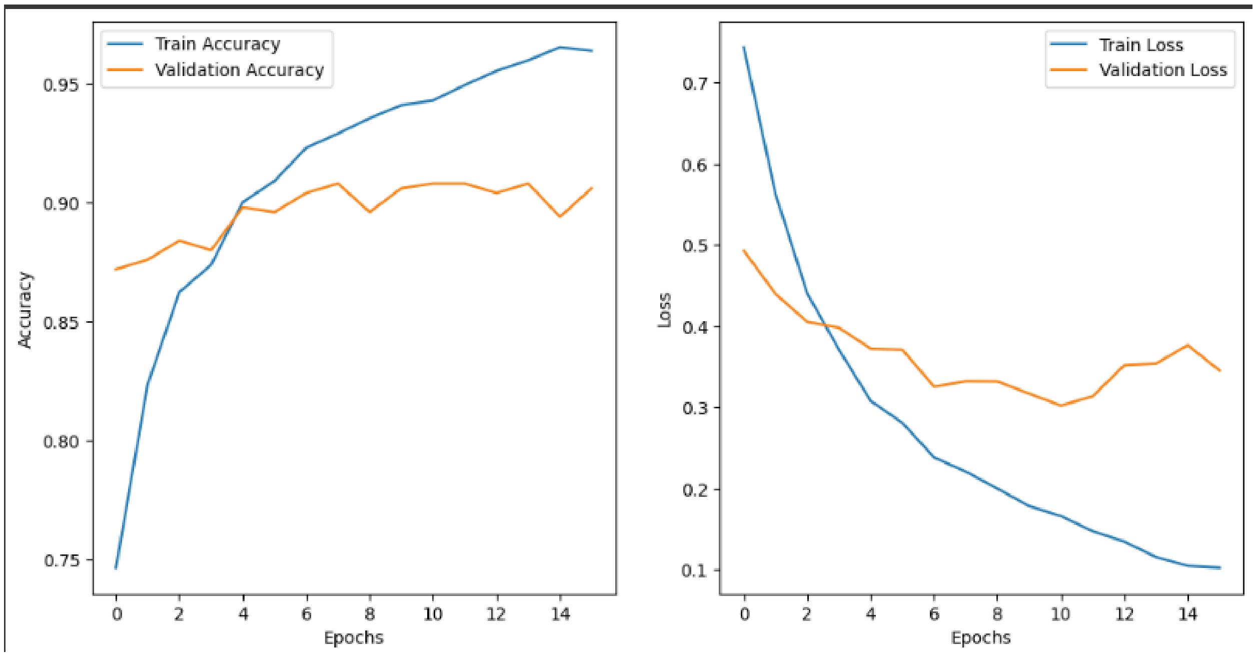
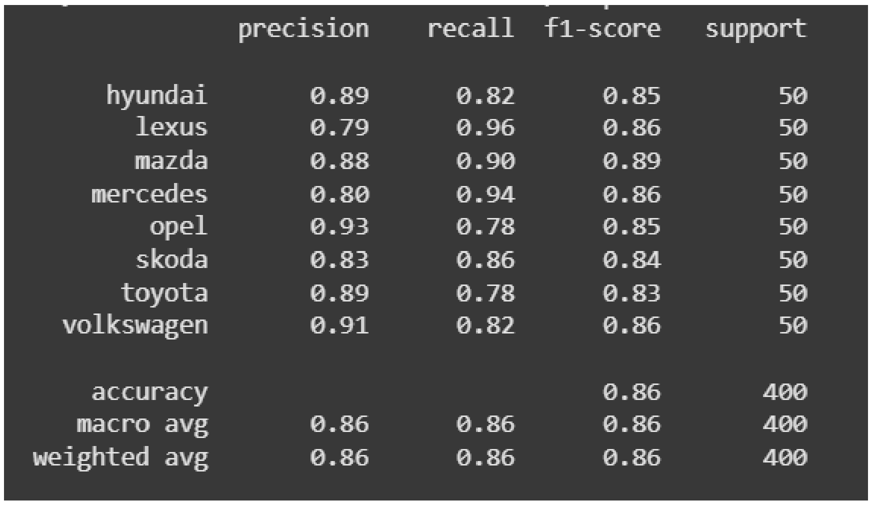


Figure 8 Transfer Learning Training and Validation Curve

The classification report also shows better performance over the precision, recall and f1 score compared to the rest of the two models.



```
              precision    recall  f1-score   support

     hyundai       0.89      0.82      0.85        50
       lexus       0.79      0.96      0.86        50
       mazda       0.88      0.90      0.89        50
    mercedes       0.80      0.94      0.86        50
        opel       0.93      0.78      0.85        50
       skoda       0.83      0.86      0.84        50
      toyota       0.89      0.78      0.83        50
  volkswagen       0.91      0.82      0.86        50

    accuracy                           0.86       400
   macro avg       0.86      0.86      0.86       400
weighted avg       0.86      0.86      0.86       400
```

Figure 9 Transfer Learning Classification Report

6. Conclusion and Future Work

Key findings show that the VGG16 transfer learning model achieved the highest accuracy at 0.86 accuracy followed by the deeper model with Adam at 0.80 and the baseline at 0.61. Trade-off exists between accuracy and computational cost, with the deeper model requiring around 45 minutes but performing well in unseen data, while the baseline trained faster but underperformed. Potential improvement involves training with larger dataset and testing with architecture like ResNet.

7. References

Saeed R. Saeed1, A. H. M. ,. S. K. K. ,. A. A. A.-H. 4., 2023. The study of deep learning for automotive logo recognition and. *Periodicals of Engineering and Natural Sciences,* XI(3), pp. 255-268.