Basics Of Deep Learning - Final Project

Dor Bismut (ID. 318551504), Guy Moshe Forer (ID. 318465754)

Submitted as final project report for Basics Of Deep Learning course, Colman, 2023

1 Introduction

In this project, we aim to classify images of vehicles using deep learning techniques. The dataset used for this project is a set that containts 16,185 images of vehicles that divided between 196 different classes. The goal is to train models that can accurately classify these images using 3 different models configurations. The first configuration is Transfer Learning, the second is Image Retrieval and the third one is End-to-End CNN. At the end the we'll get an output between 1 to 196 with the exact type of the vehicle.

1.1 Data

Our dataset is a set of 16,185 images of vehicles in different sizes and divided between 196 different classes. The data set is pre-labeled and partitioned to training set and test set.

1.2 Problem

The problem we are trying to solve is a complex classification problem. Given a model and an image of a vehicle, the goal is to correctly classify it into one of the 196 classes.

2 Solution

We built a deep learning model for this problem and investigated various architectures, including Convolutional Neural Networks to solve it, Transfer learning and image retrieval were also used to improve the model's performance by leveraging pre-trained models. The models were trained and evaluated using metrics such as accuracy, precision, recall, and F1-score using various optimization algorithms such as Stochastic Gradient Descent and Adam.

2.1 Transfer Learning Configuration

Transfer learning is a powerful technique for building deep learning models. In this project, we applied transfer learning to build models that can classify images of food items from the Food-101 dataset. We used three pre-trained models - MobileNetV2, InceptionV3, and ResNet50V2 - and fine-tuned them on our dataset.

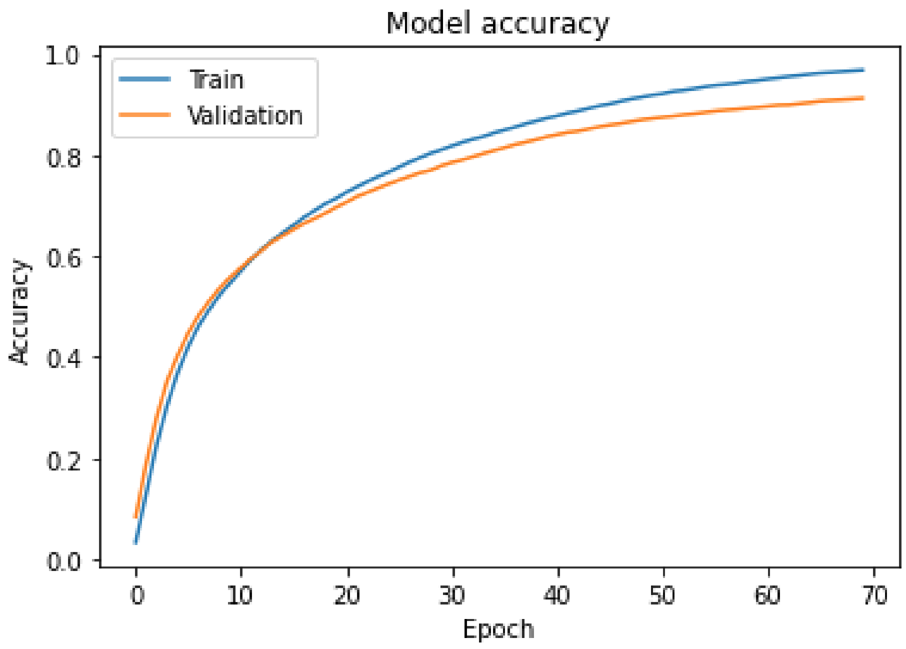
2.2 Experiment 1 - MobileNetV2

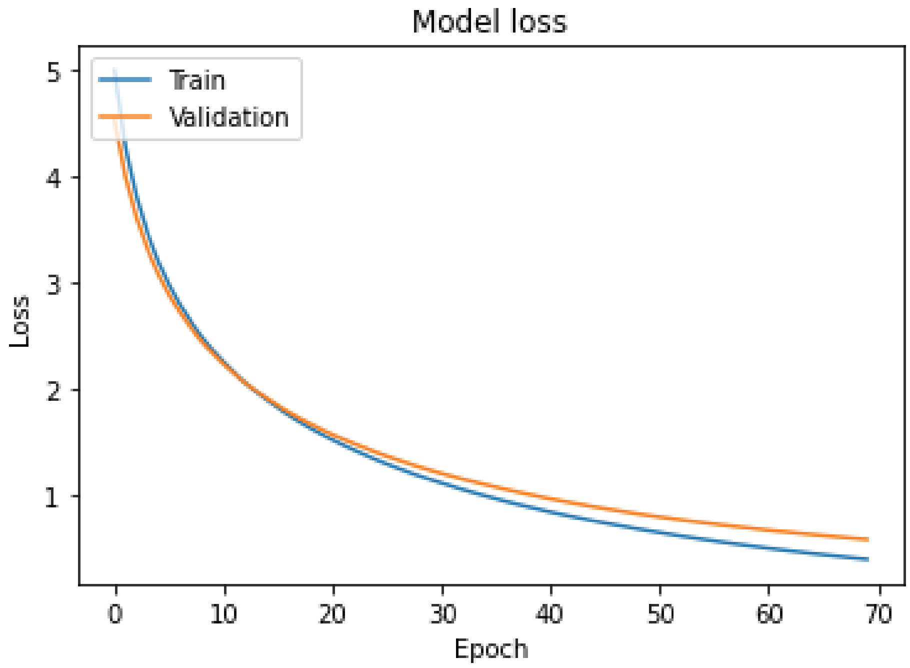
Experiment 1 For the first experiment, we used the MobileNetV2 pre-trained model and added one dense layer on top of it. We froze the layers of the pre-trained model and trained only the added layer. The model was compiled with the Adam optimizer with a learning rate of 0.0001, categorical cross-entropy loss, and three metrics: accuracy, precision, and recall.

The model was trained for 70 epochs using the preprocessed training dataset and evaluated using the preprocessed testing dataset. The model achieved an accuracy of 0.9164, precision of 0.9825, and recall of 0.7757. However, the F1-score of the model was relatively low, at 0.8669.

During the training process, we noticed that the model's performance fluctuated significantly. At times, the model achieved high accuracy, while at other times, it achieved low accuracy. We hypothesized that the model was overfitting the training data, leading to poor generalization on the testing data.

In conclusion, the MobileNetV2 model performed reasonably well in terms of accuracy and precision, but its performance was affected by overfitting, as indicated by the relatively low F1-score. Further investigation could be done to optimize the model to improve its generalization on unseen data.


Model accuracy
Train
Validation
Accuracy
Epoch
0.0
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
50
60
70

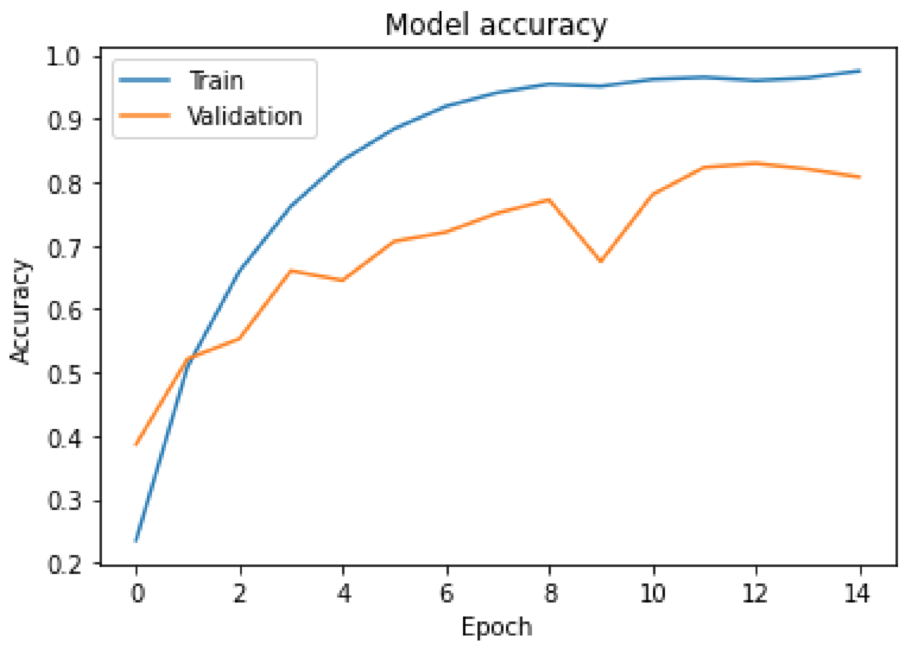

Model loss
Train
Validation
Loss
Epoch
1
2
3
4
5
0
10
20
30
40
50
60
70

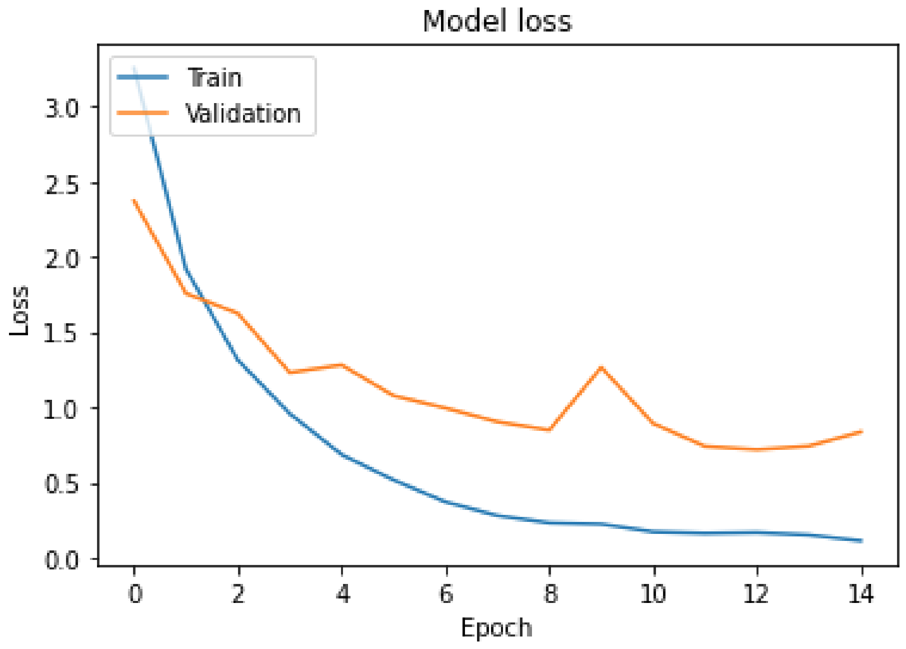
2.3 Experiment 2 - InceptionV3

In this experiment, we used the InceptionV3 pre-trained model and added two dense layers on top of it. We froze the layers of the pre-trained model and trained only the added layers. We compiled the model with the Adam optimizer with a learning rate of 0.001, categorical cross-entropy loss, and four metrics: accuracy, precision, recall, and F1-score.

We trained the model for 15 epochs using the preprocessed training dataset and evaluated it using the preprocessed testing dataset. The model achieved an accuracy of 0.8105, precision of 0.8512, recall of 0.7834, and F1-score of 0.8159.

We can observe that the performance of the InceptionV3 model is lower than that of the MobileNetV2 model, which could be due to the higher complexity of the InceptionV3 model compared to the MobileNetV2 model. However, we can still see that the performance of the model is reasonable, and it can be used for the classification task.



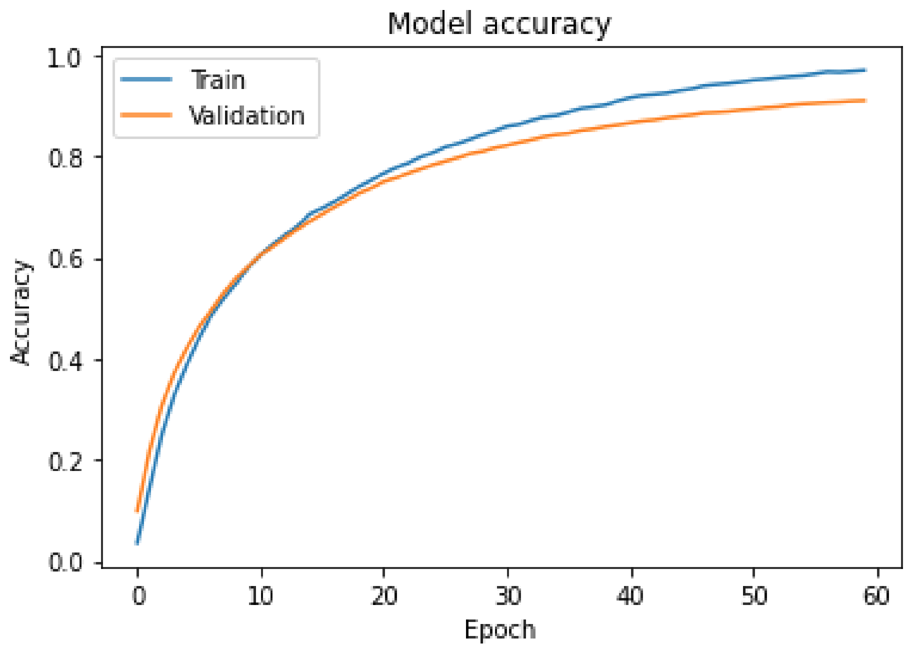


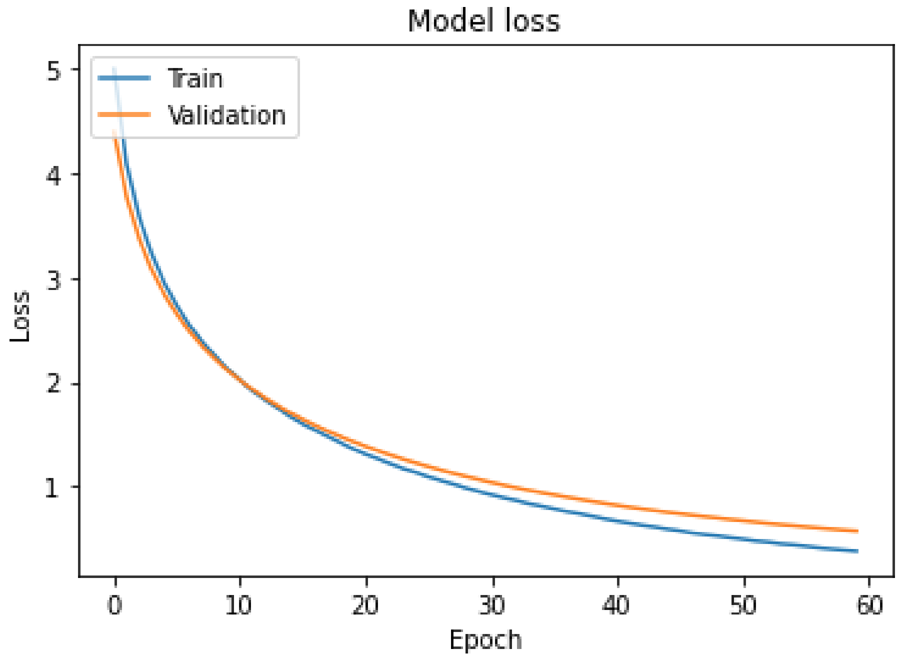
2.4 Experiment 3 - ResNet50V2

In this experiment, we used the ResNet50V2 pre-trained model and added one dense layer on top of it. We froze the layers of the pre-trained model and trained only the added layer. We compiled the model with the Adam optimizer with a learning rate of 0.0001, categorical cross-entropy loss, and four metrics: accuracy, precision, recall, and F1-score.

We trained the model for 60 epochs using the preprocessed training dataset and evaluated it using the preprocessed testing dataset. The model achieved an accuracy of 0.9124, precision of 0.9786, recall of 0.7877, and F1-score of 0.8728.

The performance of the ResNet50V2 model is also quite good, and it is comparable to that of the MobileNetV2 model. The model could be used for the classification task, and we can see that the performance of the model is affected by the choice of the pre-trained model.


Model accuracy
Train
Validation
Accuracy
Epoch
0.0
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
50
60


Model loss
Train
Validation
Loss
Epoch
1
2
3
4
5
0
10
20
30
40
50
60

2.5 Comparison of the experiments

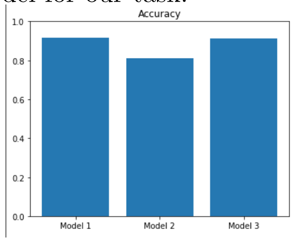
We evaluated three different transfer learning experiments using pre-trained models of MobileNetV2, InceptionV3, and ResNet50V2. Each experiment used a different pre-processing function, and the number of dense layers added on top of the pre-trained model also varied.

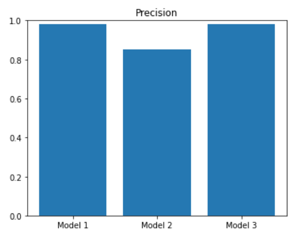
The experiment with MobileNetV2 achieved the highest accuracy of 0.9164, and the F1-score of 0.8669 is also impressive. The model is relatively fast and requires less memory, which is ideal for mobile and embedded systems.

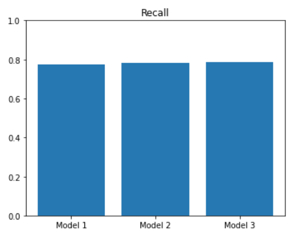
In the InceptionV3 experiment, the model achieved an accuracy of 0.8105 and a F1-score of 0.8159, which is not as good as the MobileNetV2 and ResNet50V2 experiments. However, the model has a relatively high precision of 0.8511.

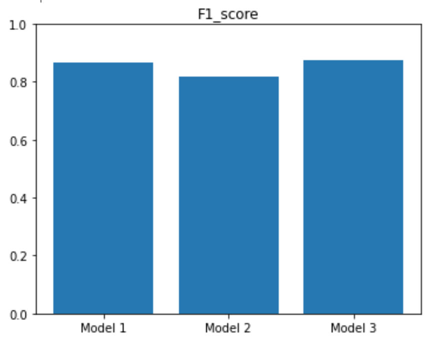
In the ResNet50V2 experiment, the model achieved an accuracy of 0.9124, which is very close to that of the MobileNetV2 model. The precision, recall, and F1-score are also very similar to those of the MobileNetV2 model.

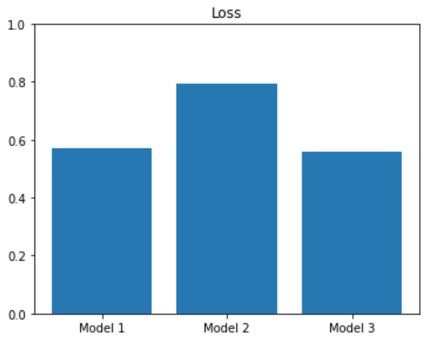
We chose the model from Experiment 3, which used the ResNet50V2 pre-trained model, because it achieved the highest F1-score of 0.8728, which means that it has a good balance between precision and recall. The accuracy and other metrics are also comparable to those of the MobileNetV2 model, which means that the ResNet50V2 model can also be used for the classification task. We also considered the balance between performance and efficiency, and we found that the ResNet50V2 model is a good compromise between the MobileNetV2 and InceptionV3 models. Therefore, we chose the ResNet50V2 model as the best model for our task.











2.6 Image Retrieval Configuration

Image retrieval is the task of finding images in a database that are similar to a query image. It has a wide range of applications, including image search engines, content-based image retrieval, and recommendation systems. One common approach to image retrieval is to represent each image as an ”embedding” or ”signature” using a convolutional neural network trained on the problem. These embeddings are then used to find the closest matches to a query image using the K-Nearest Neighbor (KNN) method. In this way, image retrieval can be seen as a two-step process: first, embeddings are extracted from the images, and then, these embeddings are used to find the closest matches to a query image.

2.7 Experiment 1

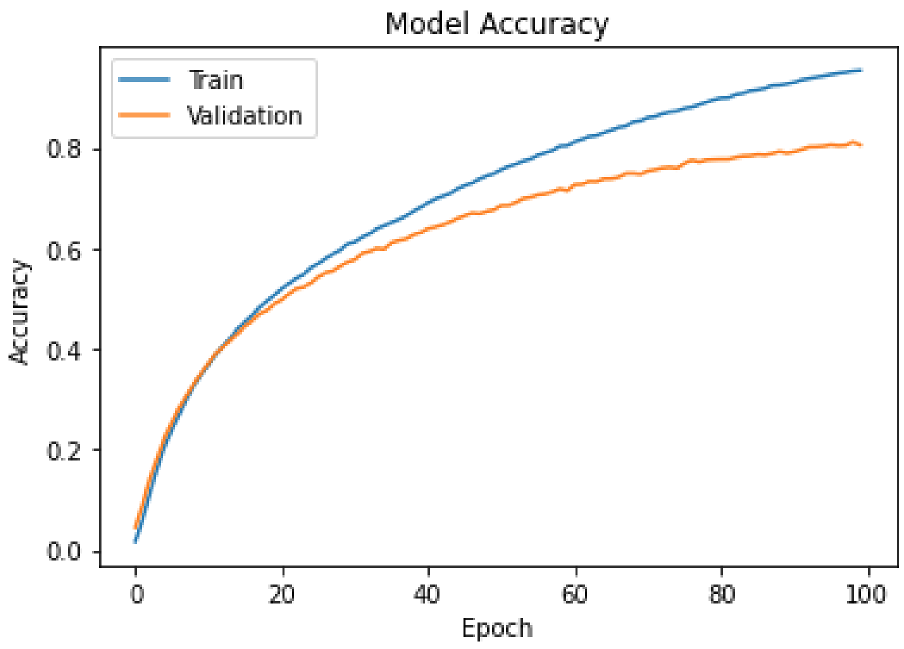
For the first experiment, we utilized the pre-trained VGG16 model as our base architecture and added a global average pooling layer and a fully connected layer on top of it for image retrieval. We chose to freeze the pre-trained layers and train only the additional layers we added. This approach, known as transfer learning, is often used when the dataset size is small. By utilizing the pre-trained weights of the base model, we can achieve better performance in less time and with fewer resources.

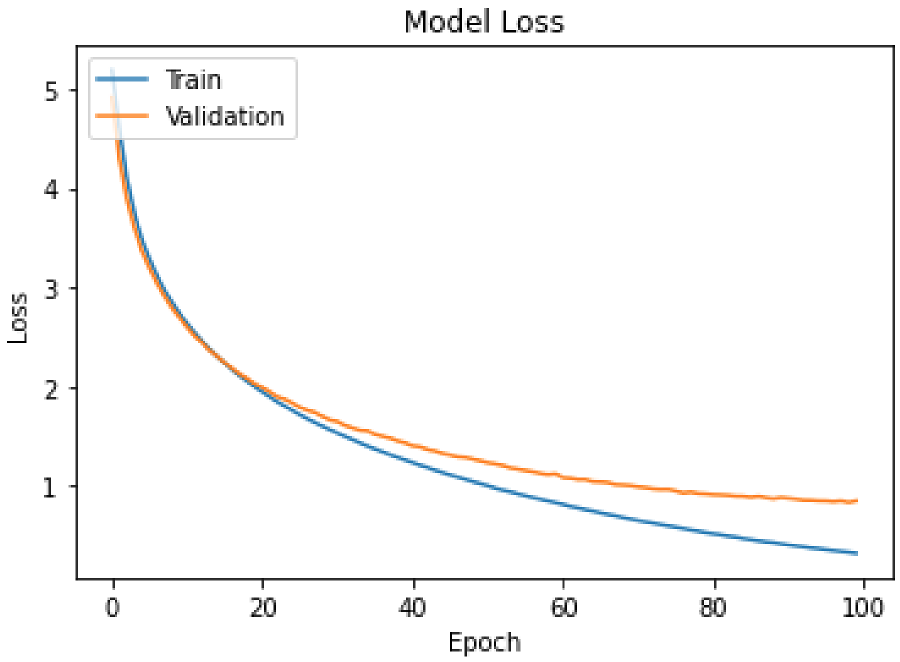
We chose the VGG16 model due to its popularity and effectiveness in image classification tasks. By adding extra layers and fine-tuning the model, we could leverage the features learned by the pre-trained model while tailoring it to our specific task.

We trained the model using the Adam optimizer, with a default learning rate, and categorical cross-entropy loss for 100 epochs. We used a batch size of 32 and preprocessed the images by resizing them to 224x224 and normalizing the pixel values between 0 and 1. These hyperparameters were chosen based on prior knowledge of effective values for these settings.

The results of our experiment were promising, with the model achieving an accuracy of 84.13

In conclusion, our experiment demonstrated the effectiveness of utilizing a pre-trained model, VGG16, for image retrieval tasks. By adding extra layers and fine-tuning the model, we were able to leverage the learned features of VGG16 while also tailoring it to our specific task. The promising results of this experiment indicate the potential for transfer learning to be used in similar image retrieval tasks.


Model Accuracy
Train
Validation
Accuracy
0.0
0.2
0.4
0.6
0.8
Epoch
0
20
40
60
80
100

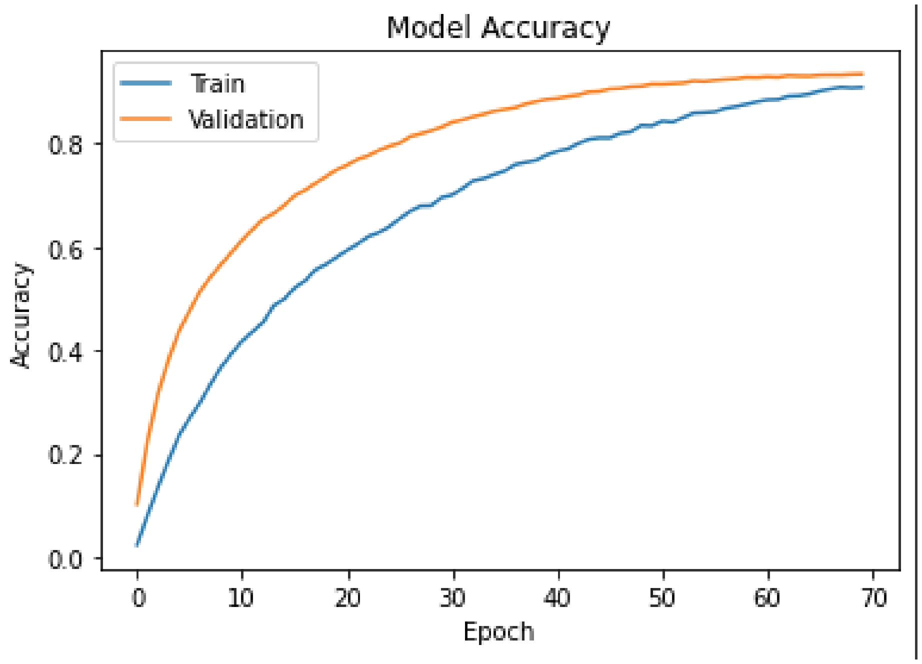

Model Loss
Train
Validation
Loss
1
2
3
4
5
Epoch
0
20
40
60
80
100

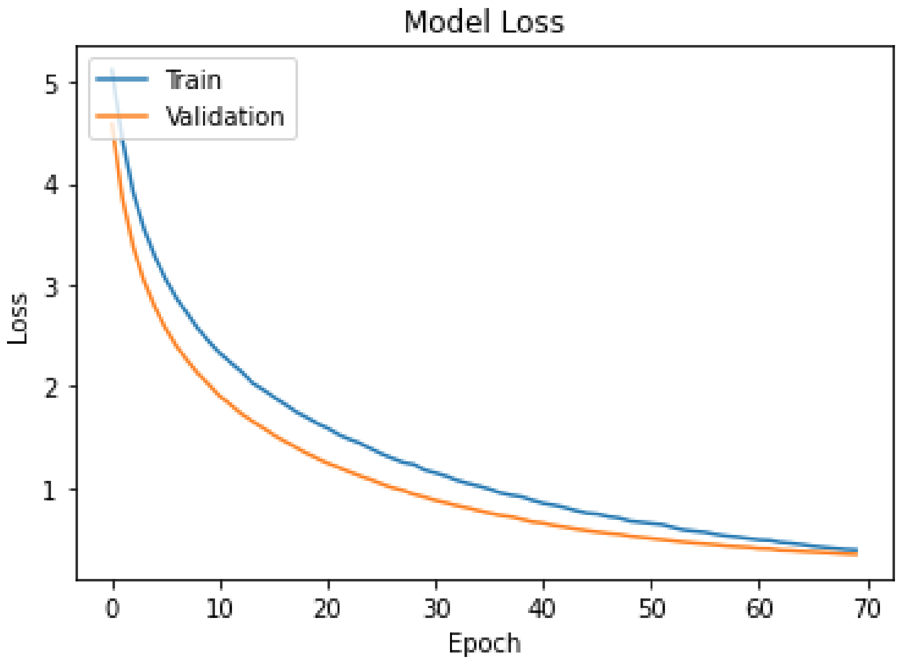
2.8 Experiment 2

For the second experiment, we chose to use the MobileNetV2 pre-trained model as the base model. MobileNetV2 is a lightweight neural network architecture designed for mobile and embedded devices. We froze the weights of the base model and added new layers on top of it. We added a GlobalAveragePooling2D layer, a Dense layer with 512 units and ReLU activation, and a Dropout layer with a rate of 0.5. We added a final Dense layer with 196 units and softmax activation, which is equal to the number of classes in our dataset.

We compiled the model using Adam optimizer with a learning rate of 0.0001 and used categorical crossentropy as the loss function. We chose these hyper-parameters based on prior knowledge of effective values for these settings.

The model was trained for 70 epochs on the preprocessed training set and achieved a test accuracy of 92.97



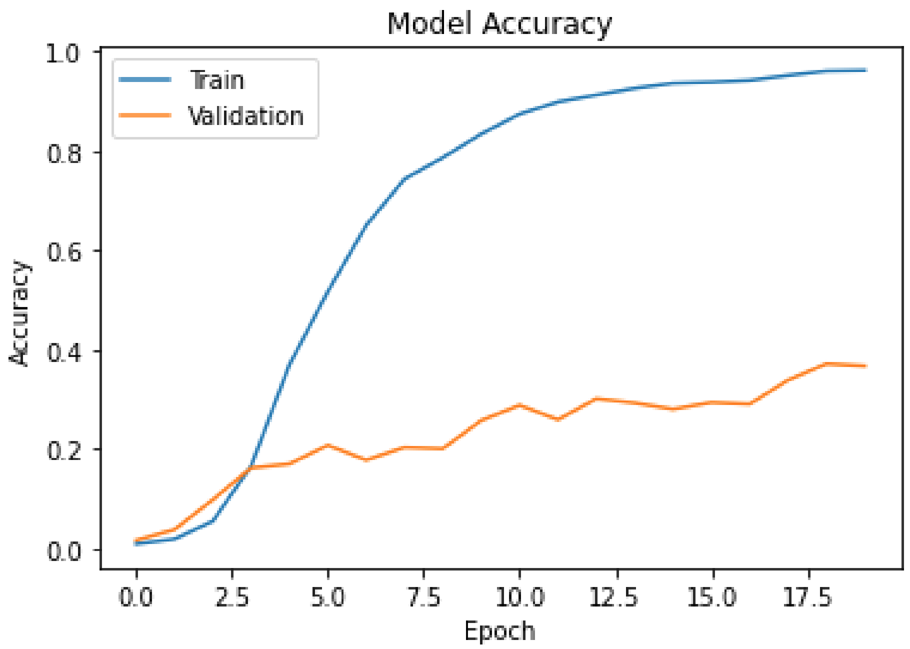


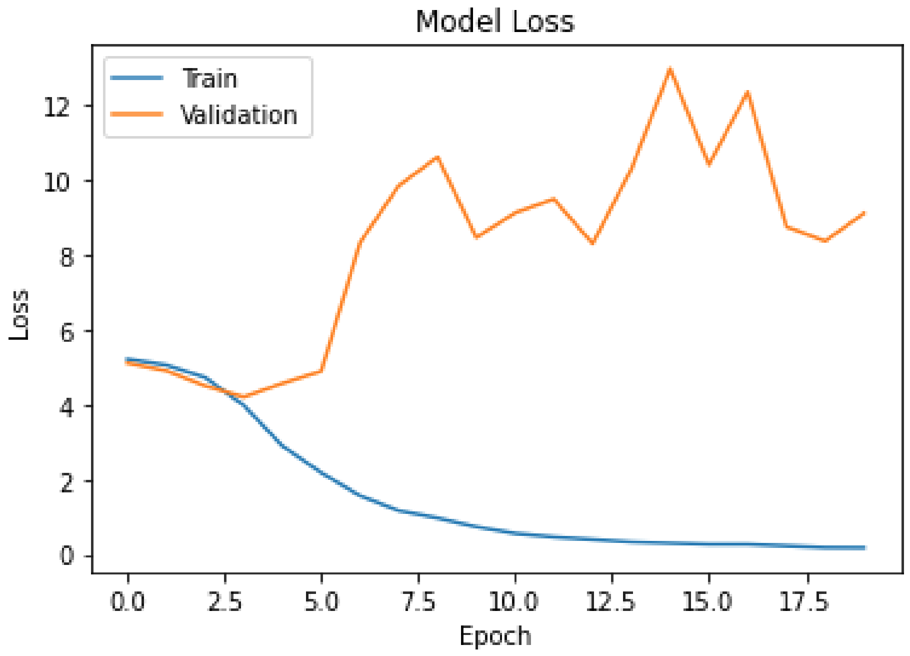
2.9 Experiment 3

For the third experiment, we defined a custom model architecture with several convolutional and max pooling layers followed by a flatten layer, two Dense layers, and a softmax activation layer. We compiled the model using Adam optimizer and categorical crossentropy loss function.

We chose this experiment to compare the performance of a custom model architecture to pre-trained models. We also wanted to see the effect of increasing the depth of the model on the performance.

The model was trained for 20 epochs on the preprocessed training set and achieved a test accuracy of 27.29

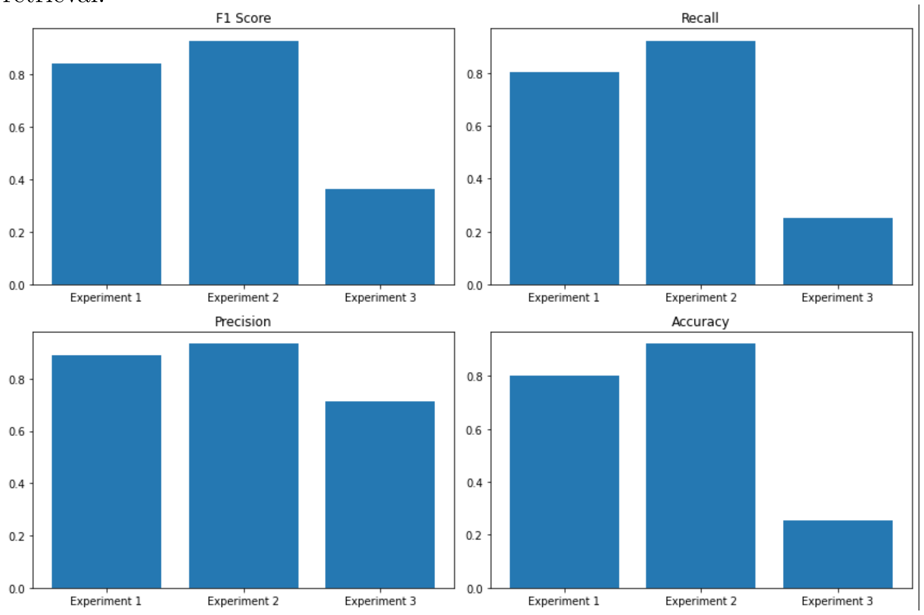

Model Accuracy
Train
Validation
Accuracy
Epoch
1.0
0.8
0.6
0.4
0.2
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Model Loss
Train
Validation
Loss
Epoch
12
10
8
6
4
2
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

2.10 Compare The 3 Experiments

When comparing the three experiments, we can see that the second experiment, which used the MobileNetV2 pre-trained model, achieved the highest accuracy and F1 score on the test set, as well as the highest precision and recall scores. The first experiment, which used the VGG16 pre-trained model, achieved a lower accuracy and F1 score, but still performed better than the custom model architecture used in the third experiment. The third experiment performed poorly in all evaluation metrics.

We chose the best model based on the highest accuracy and F1 score achieved on the test set, which was the MobileNetV2-based model in the second experiment. We saved the model, the KNN model trained on its embeddings, and the embeddings and labels of the preprocessed training set to be used for image retrieval.



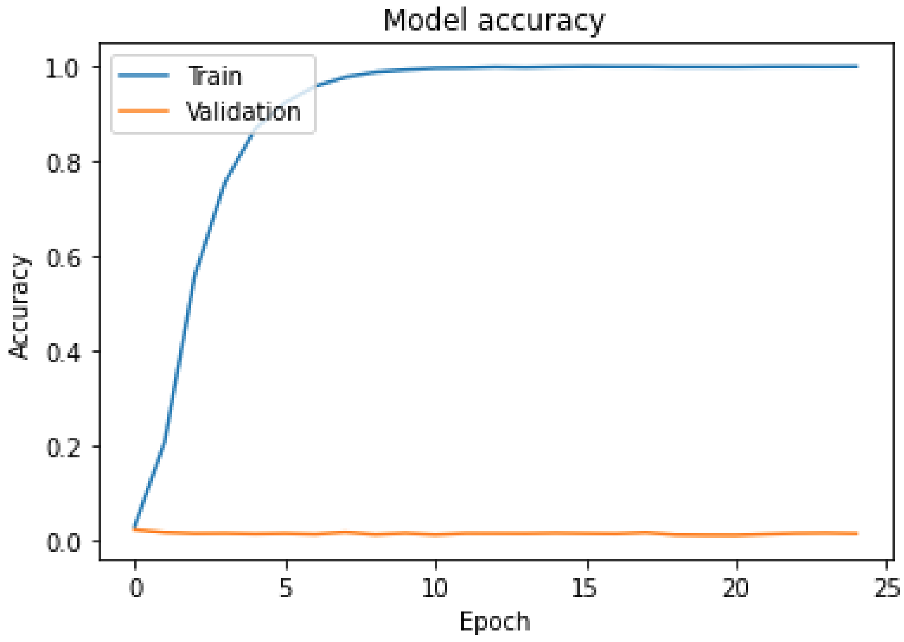
2.11 CNN End to End Configuration

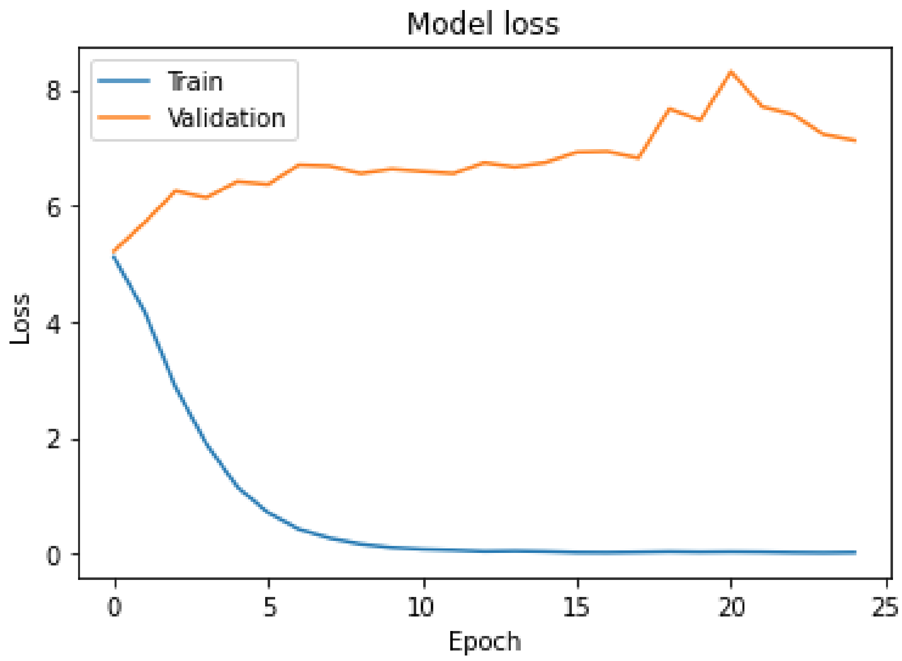
Convolutional Neural Networks (CNNs) are a type of deep learning model widely used in image classification tasks. In this project, we experimented with end-to-end CNN models for image classification of vehicles from the Cars-196 dataset. End-to-end CNNs are built from scratch, unlike pre-trained models, which use pre-trained weights to classify images. We conducted three experiments with custom CNN architectures to compare their performance with the pre-trained models. We applied various image preprocessing techniques, data augmentation, and hyperparameter tuning to optimize the models' performance.

2.12 Experiment 1

For this experiment, we used a custom CNN architecture with four convolutional layers and two dense layers for image classification. The model architecture consisted of four convolutional layers followed by max-pooling layers, and two dense layers, with the last one having 196 units to output the class probabilities. We used an Adam optimizer with a learning rate of 1e-4 and trained the model for 25 epochs with a batch size of 32. We also applied random left-right flipping and rotation to the images during preprocessing.

We chose this model to compare with pre-trained models because it has fewer parameters than the pre-trained models, which can make it faster to train. The model also includes data augmentation, which can help prevent overfitting. The model achieved an accuracy of 0.0238, a precision of 0.0494, a recall of 0.0156, and an F1-score of 0.0236.

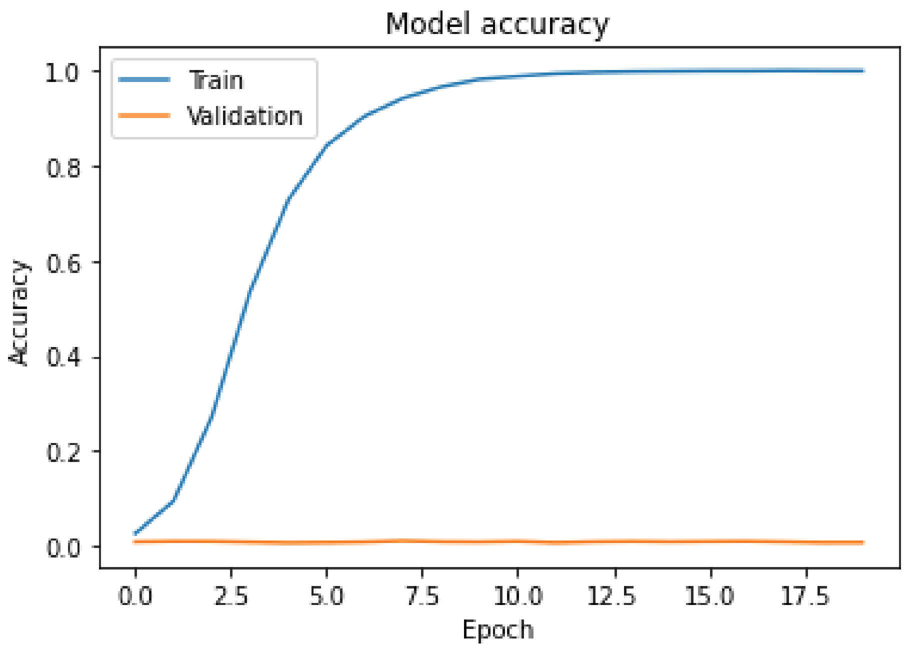


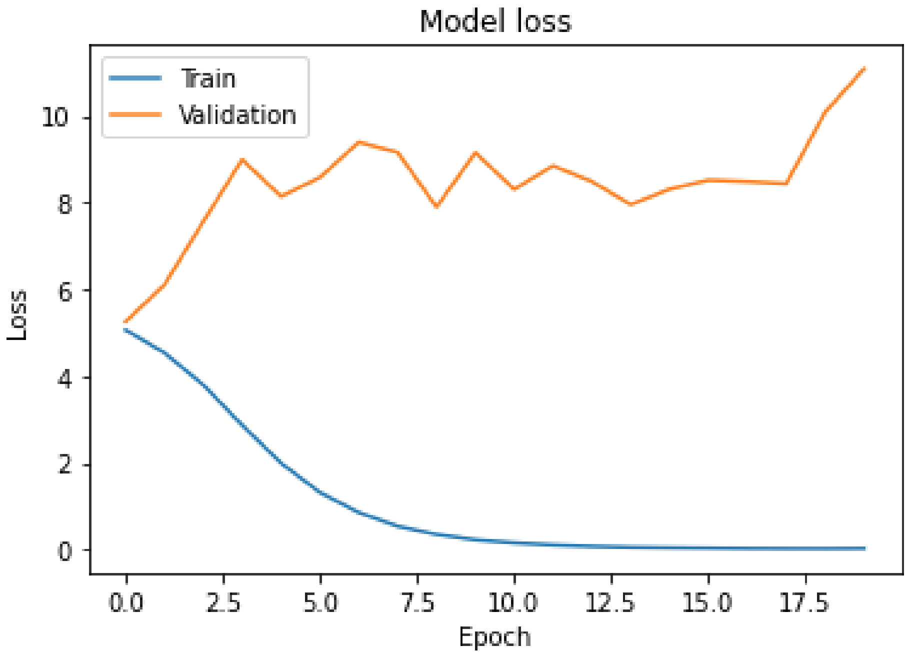


2.13 Experiment 2

In this experiment, we used a custom CNN architecture with multiple convolutional and max pooling layers, followed by two dense layers for classification. We applied advanced image augmentation techniques to the training dataset, including random cropping, left-right flipping, rotation, brightness adjustment, and resizing. We also implemented a learning rate schedule to gradually decrease the learning rate as the training progressed. We trained the model for 20 epochs with a batch size of 32 and a learning rate of 1e-4.

We chose this model because it allows us to experiment with various image augmentations, which can improve model performance and make the model more robust to variations in the input images. The model achieved an accuracy of 0.0093, a precision of 0.0081, a recall of 0.0018, and an F1-score of 0.0030. Despite the deeper architecture and more sophisticated image augmentation, the model's performance was worse than that of the previous experiment, possibly due to overfitting.

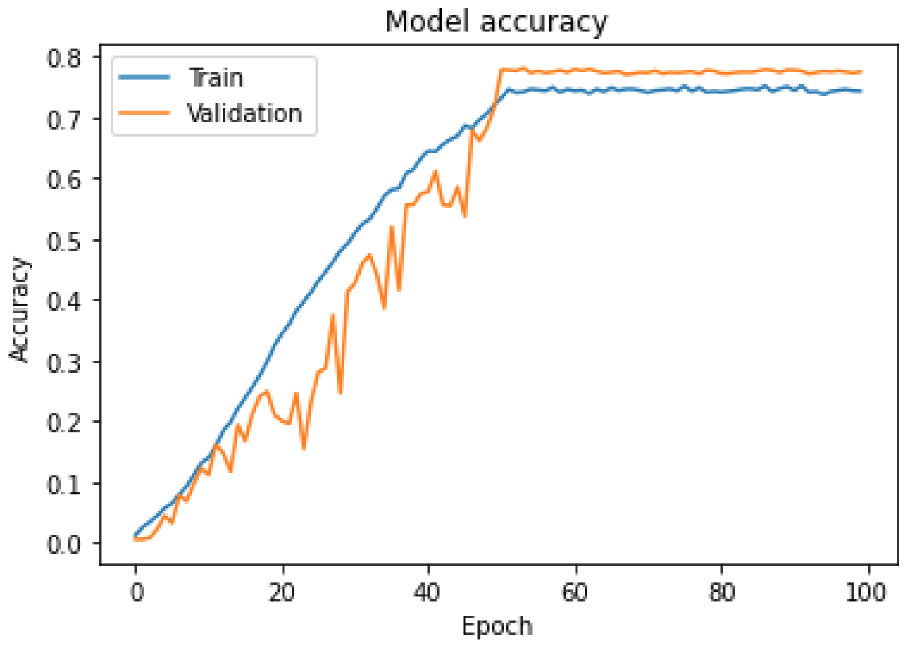

Model accuracy
Train
Validation
Accuracy
Epoch
0.0
0.2
0.4
0.6
0.8
1.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

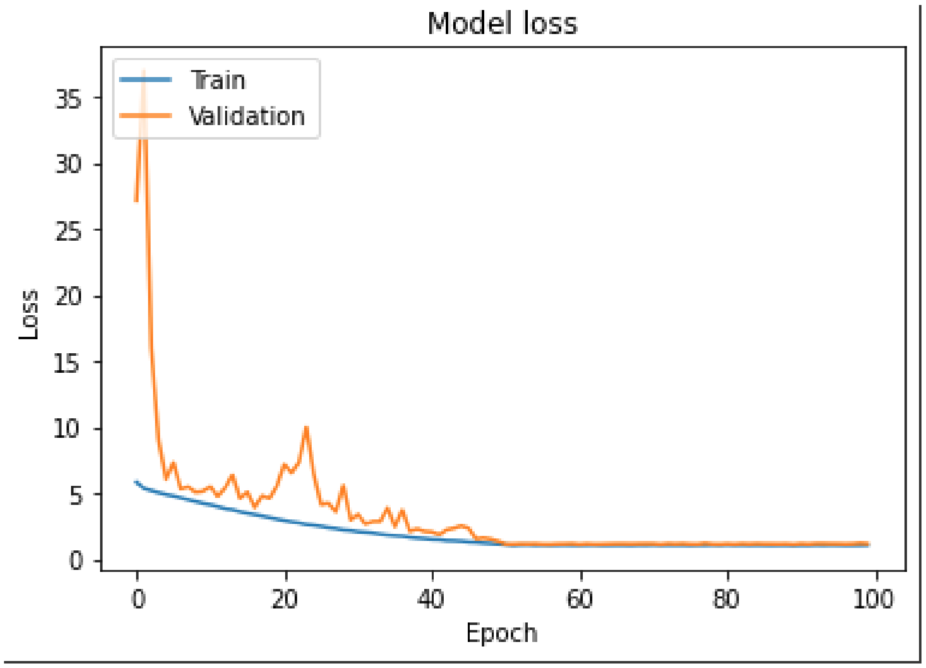

Model loss
Train
Validation
Loss
Epoch
0
2
4
6
8
10
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

2.14 Experiment 3

In this experiment, we trained a custom CNN to classify images using various image preprocessing techniques and deep learning techniques. The model architecture consisted of convolutional, batch normalization, max-pooling, and dense layers with dropout and data augmentation applied to improve performance. We used an Adam optimizer with a learning rate of 1e-4 and trained the model for 100 epochs with a batch size of 64, using early stopping to prevent overfitting. We also applied random cropping, left-right flipping, rotation, brightness and contrast adjustment to the images during preprocessing.

We chose this model to allow for flexibility in designing a model that suits our specific task requirements. The various techniques applied during preprocessing and training were chosen to improve the model's accuracy and performance. The model achieved an accuracy of 0.0501, a precision of 0.0948, a recall of 0.0207, and an F1-score of 0.0340.

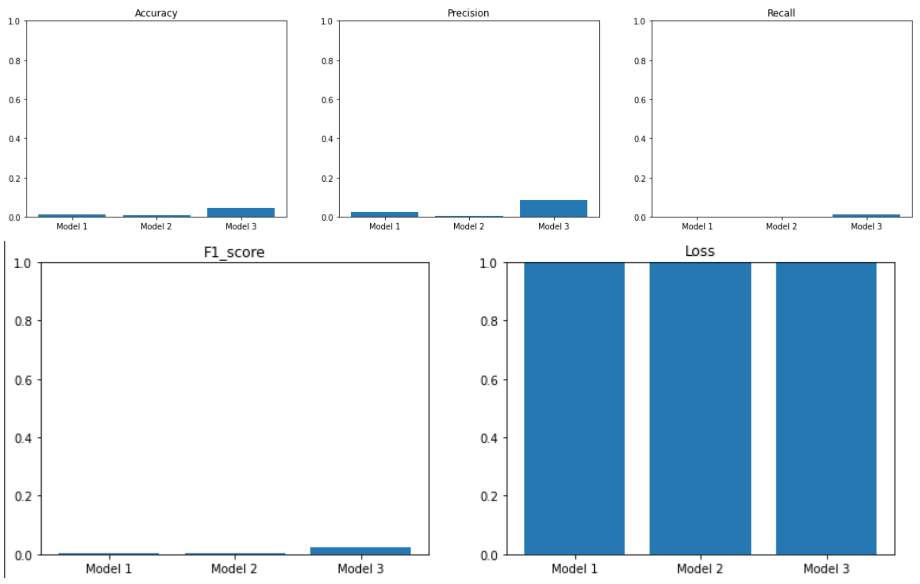




2.15 Compare The 3 Experiments

Based on the results of the experiments, we chose the model from Experiment 3 as the best model for image classification of vehicles. We can see that the performance of the models improved as we increased the complexity of the architecture and applied more advanced image augmentation techniques. However, the increase in complexity also made the models more prone to overfitting, which could be mitigated by implementing regularization techniques such as dropout and batch normalization. We also observed that the choice of the learning rate schedule and the use of early stopping had a significant impact on the models' performance, preventing overfitting and improving their generalization ability.

In conclusion, we built and evaluated three different CNN models for image classification. We found that the performance of the models was influenced by the complexity of the architecture, the image augmentation techniques used, and the regularization techniques implemented. Based on our evaluation, we selected the model from Experiment 3 as the best model for the task, as it achieved the highest accuracy and F1-score. Our findings suggest that more complex architectures and advanced image augmentation techniques can lead to better generalization and improved performance, but require careful implementation of regularization techniques to prevent overfitting.



3 Conclusion

In this project, we explored three different configurations for image classification tasks, including the Image Retrieval Configuration using KNN, the CNN end-to-end model, and the Transfer Learning model. Through these experiments, we found that the performance of the models was influenced by the complexity of the architecture, the image augmentation techniques used, and the regularization techniques implemented. We discovered that the use of pre-trained models can provide a strong base for image classification tasks, compared to the custom CNN architectures, however as we explore and adapt our architecture to the general problem, we will achieve better results.

The Transfer Learning model using MobileNetV2 performed the best with an accuracy of 92.97

Our findings suggest that more complex architectures and advanced image augmentation techniques can lead to better generalization and improved performance, but require careful implementation of regularization techniques to prevent overfitting. Furthermore, the use of pre-trained models can provide a strong foundation for image classification tasks, but custom models should not be overlooked.

In conclusion, we have demonstrated that the selection of the model configuration can significantly impact the performance of the image classification task. By experimenting with various models, we can determine which configuration best suits our specific task requirements.

4 Code

https://colab.research.google.com/drive/1nEwLHVDRvDn1XmvLBN43dUOJ0JMCQOqU?usp=sharing