School of Chemical and Biomolecular Engineering

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

CH4244

Numerical Methods & Data Analytics

Name: Ng Yu Xuan (U1921991J)

Date of Submission: 17 November 2022

Table of Contents

[Python Problem]

<u>Question A:</u>

a) Load the data and then preprocess them with ImageDataGenerator().

<u>Summary</u>

<u>Loading and Preprocessing of Train, Validation and Test Data</u>

In the given cars dataset, the dataset has already been split for us into 3 categories under train, valid and test folders in a ratio of 4:1:1.

- **<u>Train:</u>** The model being utilised will learn from the training dataset during training
- **<u>Validation:</u>** The data in this category will be used to test for overfitting/underfitting as well as any form of fine-tuning
- **<u>Test:</u>** Test data will be used to analyse the performance of the model by evaluating the results of the multiple metrics used for the test dataset

<u>Data Preprocessing</u>

- Data is required to be normalized in order for train data images to be more equally considered when the model is training and updating its weights
- Neural networks are notorious for resulting in easily overfitting both training and validation datasets.
- If data augmentation is considered, applying it on training and validation dataset will give rise to variations in such datasets which is considered a type of regularization and reduces overfitting. <u>(Preferably only on training set to increase diversity of training set which is done in this assignment)</u>

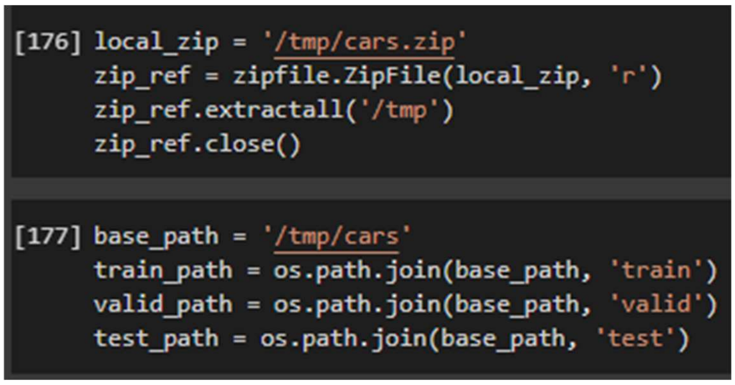
<u>- Data Preprocessing specifics:</u>

- **<u>Training Set</u>**: tf.keras.applications.vgg16.preprocess_input + **<u>further data augmentation</u>**

- **<u>Validation/Test Set</u>:** tf.keras.applications.vgg16.preprocess_input only

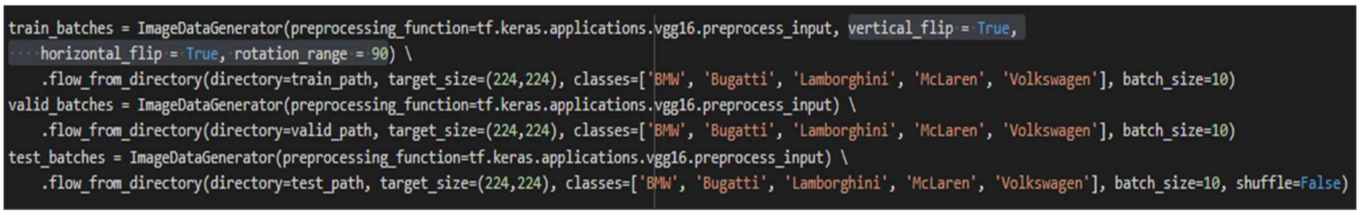
Answer:

Firstly, the zip file which contains all the necessary data for training, validation and test must be downloaded. Imported modules are also required for further usage downstream.



```
[176] local_zip = '/tmp/cars.zip'
      zip_ref = zipfile.ZipFile(local_zip, 'r')
      zip_ref.extractall('/tmp')
      zip_ref.close()

[177] base_path = '/tmp/cars'
      train_path = os.path.join(base_path, 'train')
      valid_path = os.path.join(base_path, 'valid')
      test_path = os.path.join(base_path, 'test')
```

As shown above, the zip file is unzipped and the image files in the *cars* dataset given undergoes extraction.



```
train_batches = ImageDataGenerator(preprocessing_function=tf.keras.applications.vgg16.preprocess_input, vertical_flip = True,
    horizontal_flip = True, rotation_range = 90) \
    .flow_from_directory(directory=train_path, target_size=(224,224), classes=['BMW', 'Bugatti', 'Lamborghini', 'McLaren', 'Volkswagen'], batch_size=10)
valid_batches = ImageDataGenerator(preprocessing_function=tf.keras.applications.vgg16.preprocess_input) \
    .flow_from_directory(directory=valid_path, target_size=(224,224), classes=['BMW', 'Bugatti', 'Lamborghini', 'McLaren', 'Volkswagen'], batch_size=10)
test_batches = ImageDataGenerator(preprocessing_function=tf.keras.applications.vgg16.preprocess_input) \
    .flow_from_directory(directory=test_path, target_size=(224,224), classes=['BMW', 'Bugatti', 'Lamborghini', 'McLaren', 'Volkswagen'], batch_size=10, shuffle=False)
```

As observed from the above image, the data loaded from the cars dataset are then utilised in the *ImageDataGenerator()* function so as be loaded and preprocessed. From the argument that is shown in *ImageDataGenerator()*, preprocessing function is set to tf.keras.applications.vgg16.preprocess_input which converts the targeted images from RGB to BGR, while having each colour channel zero-centered with respect to the ImageNet dataset.

Further Data Augmentation on training set as shown above: Vertical Flip, Horizontal Flip and Rotation Range of 90°

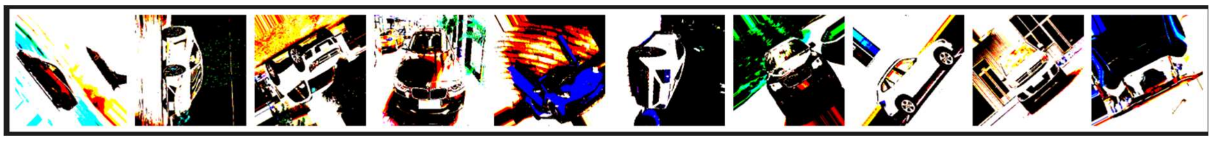


Figure 1: Images from training data set which underwent Data Augmentation that was applied as stated above

Question B:

b) Build a convolutional neural network either from scratch or existing models.

Considerations

Strategy: Transfer learning is utilised to build our very own neural network to save training time, achieve better performance while assisting in training the built neural network with relatively low amount of data

Choice of Model: VGG16 – One of the most popular pre-trained transfer learning models which uses an object detection and classification algorithm

Loss function: categorical_crossentropy - Number of classes in the *cars* dataset is considerably small which allows our data labels to be one-hot encoded

Optimizer: Adam Optimizer – Algorithm is utilised to accelerate the gradient descent algorithm by taking into consideration the 'exponentially weighted average' of the gradients

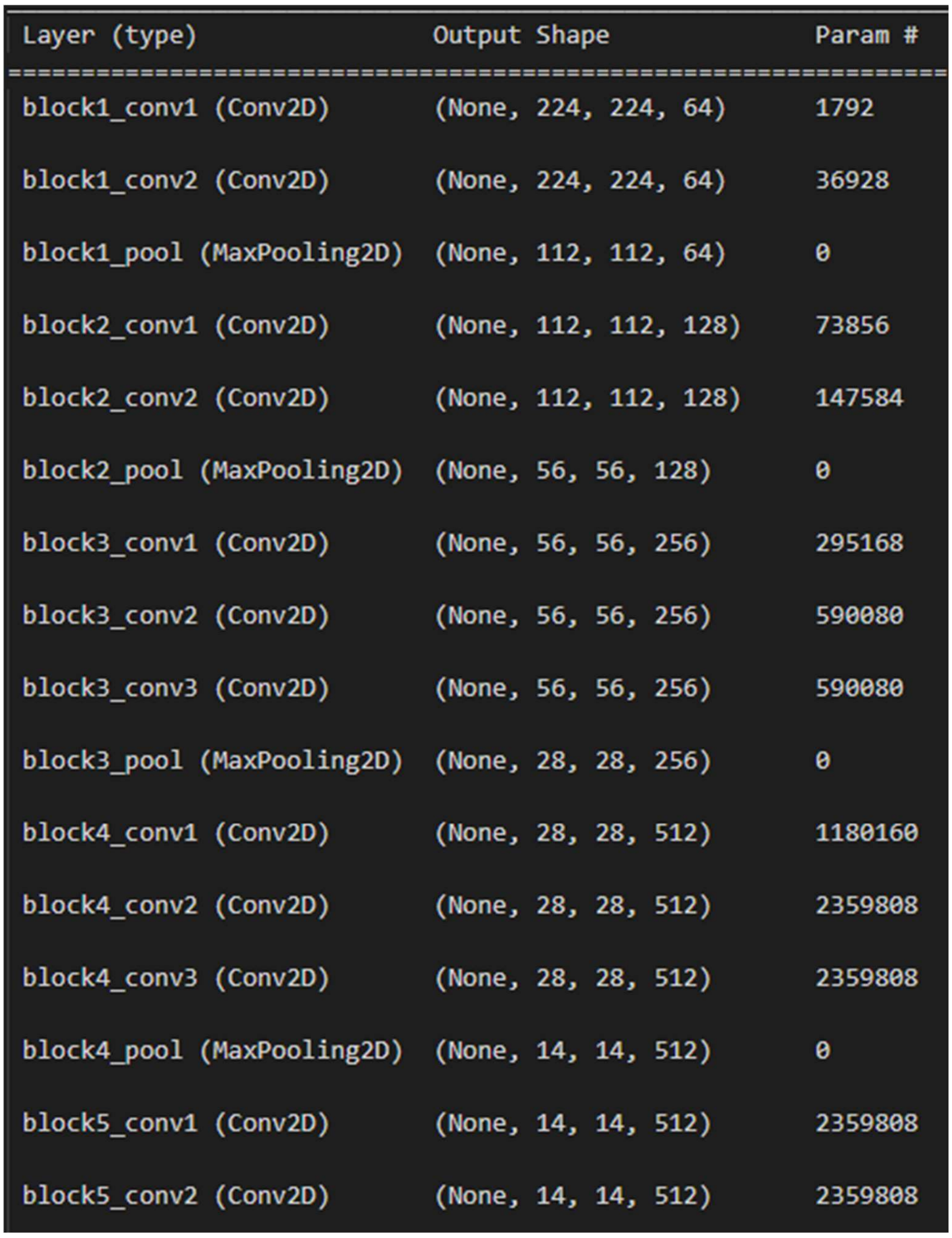
Built Model Architecture

1) Input Layer
2) Pre-trained VGG16 layers
3) Fully-connected layers
4) Output(softmax) layer with 5 categories to the new model

Transfer Learning Alterations

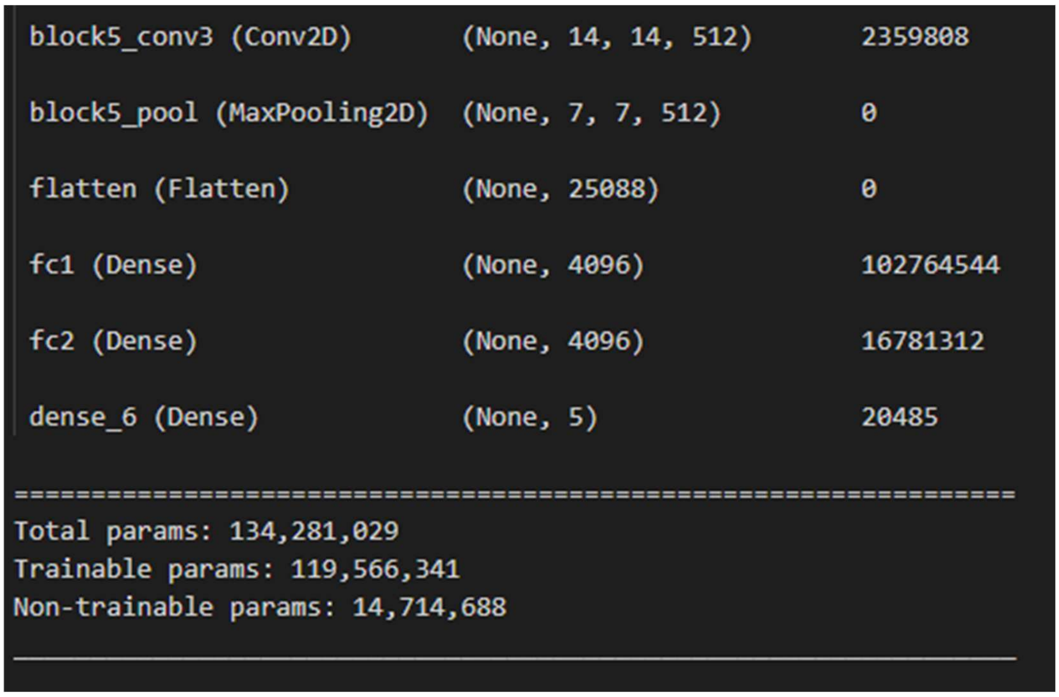
- As the output layer is formed, the layer is initialized in a random way while the layers existing before it are pre-trained VGG16 layers. Hence, warming up of the newly created layers is required so that pre-trained weights will not be lost.
- **Fine-Tuning** is conducted where a fraction of VGG16 pre-trained layers will be used to retrain the newly formed full-connected (FC) layers.

Model Architecture:



```
 Layer (type)                Output Shape              Param #
=================================================================
 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808
```

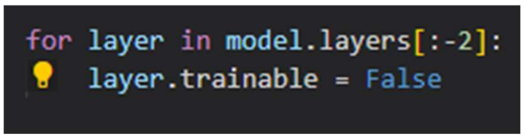
Figure 2: First half of current-built model



```
 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 fc1 (Dense)                 (None, 4096)              102764544

 fc2 (Dense)                 (None, 4096)              16781312

 dense_6 (Dense)             (None, 5)                 20485

=================================================================
Total params: 134,281,029
Trainable params: 119,566,341
Non-trainable params: 14,714,688
_________________________________________________________________
```

Figure 3: Second half of current-built model

As Shown in figure 3, Fine-tuning is executed for layers before FC1 and FC2 for training purposes where the layers before dense FC1 and FC2 layers are frozen to not lose their weight while they are being used for training of FC1, FC2 and classification header for increased relevancy for the scenario given in this particular assignment.



```
for layer in model.layers[:-2]:
    layer.trainable = False
```

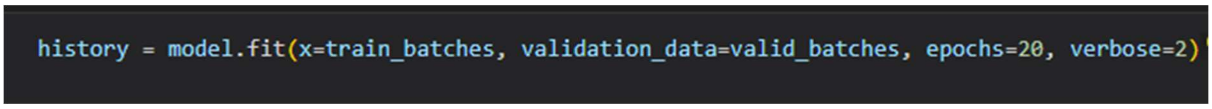
Figure 4: Freezing of all layers before FC1 and FC2

Question C:

c) Train the neural network, evaluate the model prediction, and obtain the evaluation metrics, including prediction accuracy and confusion matrix.

Network Training

As depicted in figure 4, training is being executed and the arguments for model.fit() are as follows.



```
history = model.fit(x=train_batches, validation_data=valid_batches, epochs=20, verbose=2)
```

Figure 5: Training of newly built neural network model

Evaluate Model Prediction

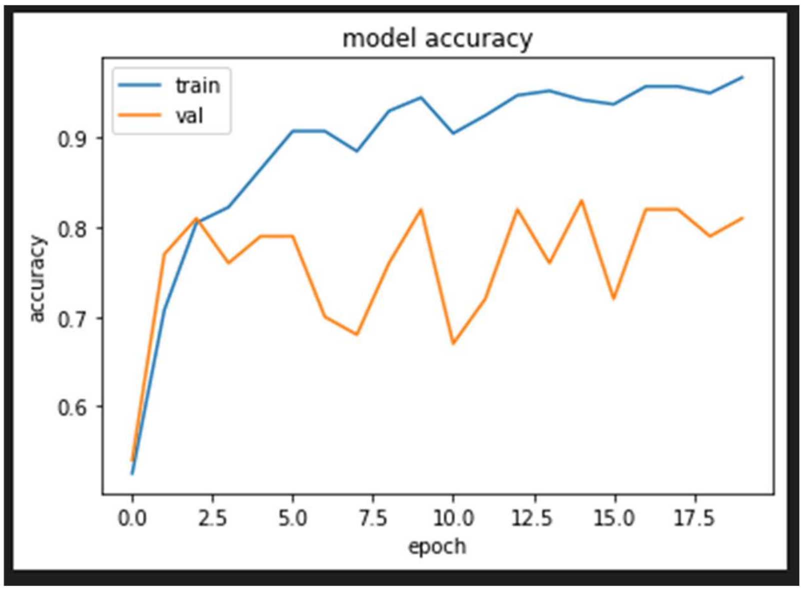


Figure 6: Results for current-built Fine-Tuned VGG16 Model

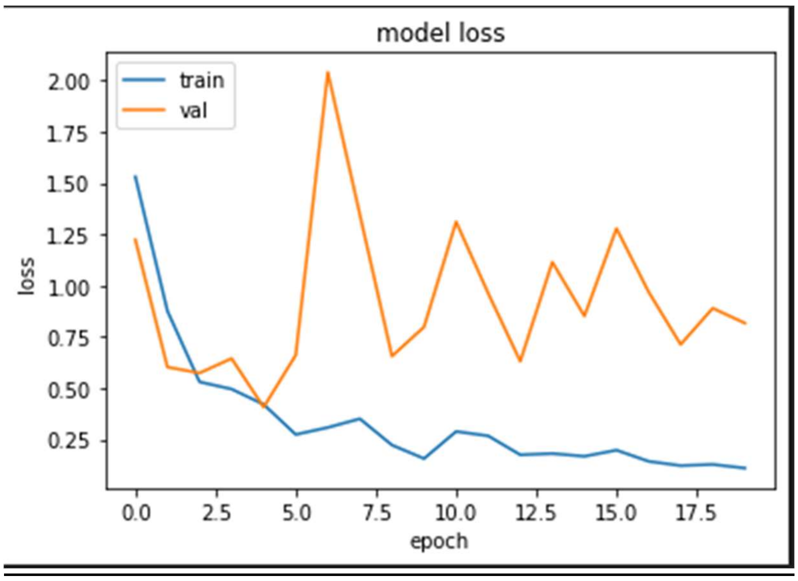
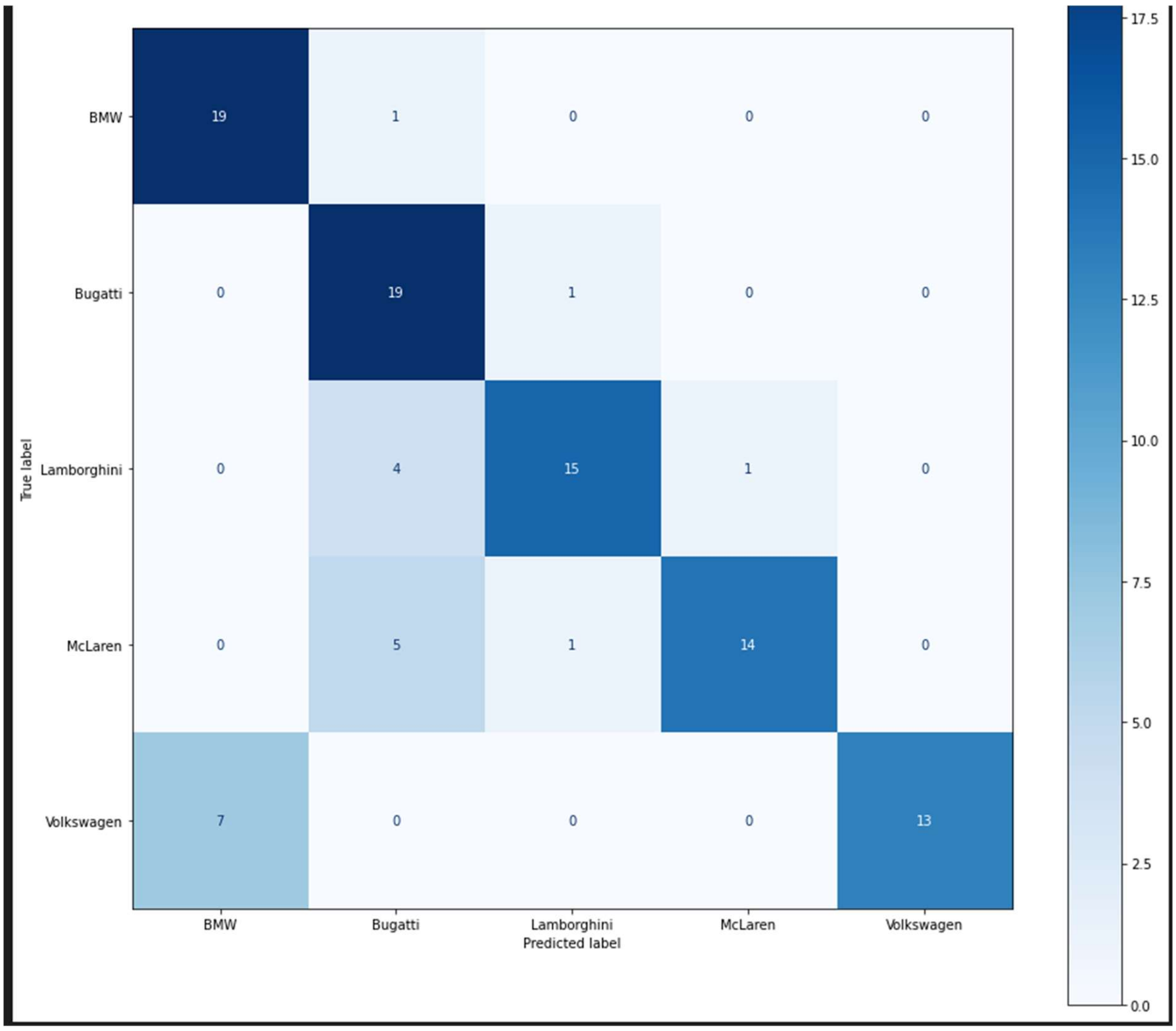


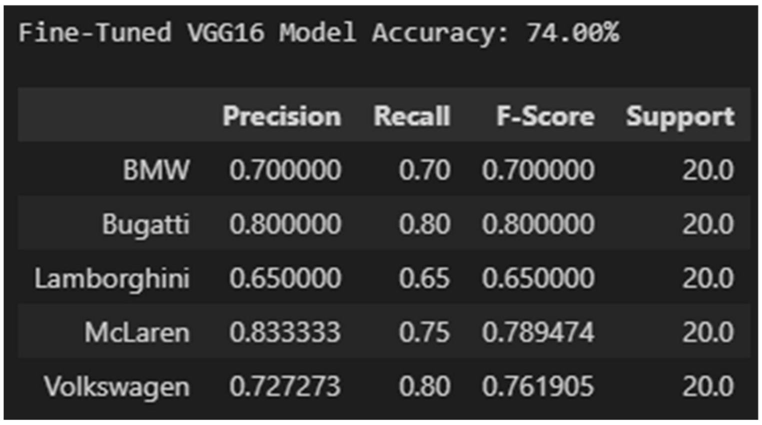
Figure 7: Results for current-built Fine-Tuned VGG16 Model loss



True label \ Predicted label	BMW	Bugatti	Lamborghini	McLaren	Volkswagen
BMW	19	1	0	0	0
Bugatti	0	19	1	0	0
Lamborghini	0	4	15	1	0
McLaren	0	5	1	14	0
Volkswagen	7	0	0	0	13

Figure 8: Confusion Matrix for current-built Fine-Tuned VGG16 Model

Evaluation Metrics



```
Fine-Tuned VGG16 Model Accuracy: 74.00%
```

	Precision	Recall	F-Score	Support
BMW	0.700000	0.70	0.700000	20.0
Bugatti	0.800000	0.80	0.800000	20.0
Lamborghini	0.650000	0.65	0.650000	20.0
McLaren	0.833333	0.75	0.789474	20.0
Volkswagen	0.727273	0.80	0.761905	20.0

Figure 9: Evaluation Metrics of current-built Fine-Tuned VGG16 Model

As observed from Figure 7 and Figure 8, our current built Fine-Tuned VGG16 Model has an accuracy of 74.00% which is considered effective at predicting test data to a certain extent. The model managed to score relatively well in precision, recall and F-Scores for all classes, proving its effectiveness to a certain extent.

Question D:

d) Based on error analysis and online materials, suggest a proper direction that can lead to better performance of your model in your presentation document (If you are referring to any online material, do provide a valid link to that reference).

Error Analysis

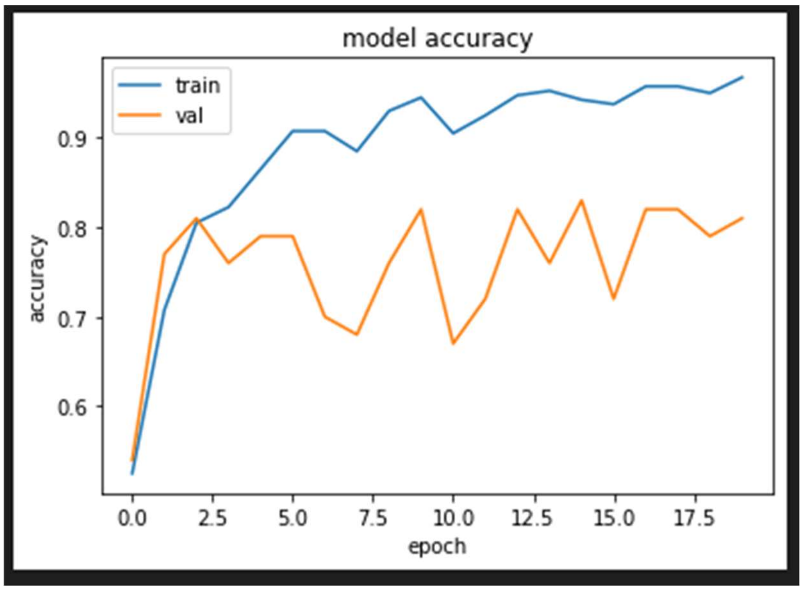


Figure 9: Repeat of Figure 6 regarding results for current-built Fine-Tuned VGG16 Model

Referring to Figure 9 as depicted above, the validation accuracy is relatively lower than training accuracy, coupled with the fact that neural networks in general are notorious for overfitting issues, this might indicate some form of overfitting in our model as well which can be remedied in a few ways (Brownlee, 2020).

Possible Overfitting Causes:

1. The data utilised for training is not properly cleaned and contains unnecessary values. The model irrelevant values in the training data and fails to generalize the model's learning.
2. The model possesses a high variance.
3. The training data size is insufficient, hence the model trains on limited training data for several epochs.
4. The architecture of the model has several neural layers stacked together. Deep neural networks are complex and need a significant amount of time to train, commonly leading to overfitting the training set. (Baheti, 2022)

Direction

A few steps were taken when building the next model for Question E to ensure overfitting is reduced to increase performance of our current-built Fine-Tuned VGG16 model.

Early stopping:

This method aims to pause the model's training before memorizing fluctuations from the data. Epochs are also increased for the upcoming updated built version of the neural network since there is early stoppage. There can be a risk that the model stops training too soon, leading to underfitting. Early stopping allows the model to train till an optimum time/iterations, effectively reducing the occurrence of overfitting. (Baheti, 2022)

Adding Dropout Layers:

Large weights in a neural network signifies that there is a more complex network at work. Randomly dropping out nodes in the network is an efficient method to reduce overfitting. For regularization purposes, a quantity of layer outputs is randomly ignored or "*dropped out*" to decrease the complexity of the model. (Baheti, 2022). For this particular assignment under question E, the layer of FC2 is removed and a dropout layer is added after FC1 for this specific purpose.

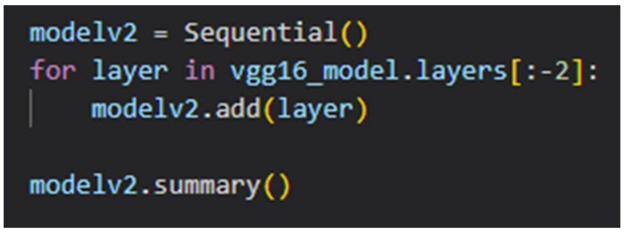
Fine-Tuning:

Unfreeze a few of the top layers of a frozen model base and train both the newly-added classifier layers and the last layers of the base model. This enables us to "fine-tune" the higher-order feature representations in the base model to ensure more contextualisation for the specific task. (Transfer learning and fine-tuning, n.d.) Previously, FC1 and FC2 are unfrozen while the layers before them were frozen. The next model will have FC1 staying unfrozen as well.

Question E:

e) Implement the new direction suggested in d) and demonstrate better evaluation metrics in your prediction outcome.

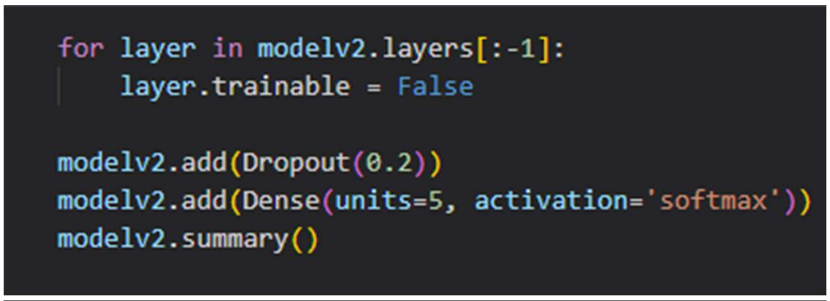
Steps taken for the next-built model



```
modelv2 = Sequential()
for layer in vgg16_model.layers[:-2]:
    modelv2.add(layer)

modelv2.summary()
```

Figure 10: Removal of FC2 layer as shown in index [:-2] for vgg16_model.layers

FC2 is removed as shown above as part of the dropout method being implemented in the current model.



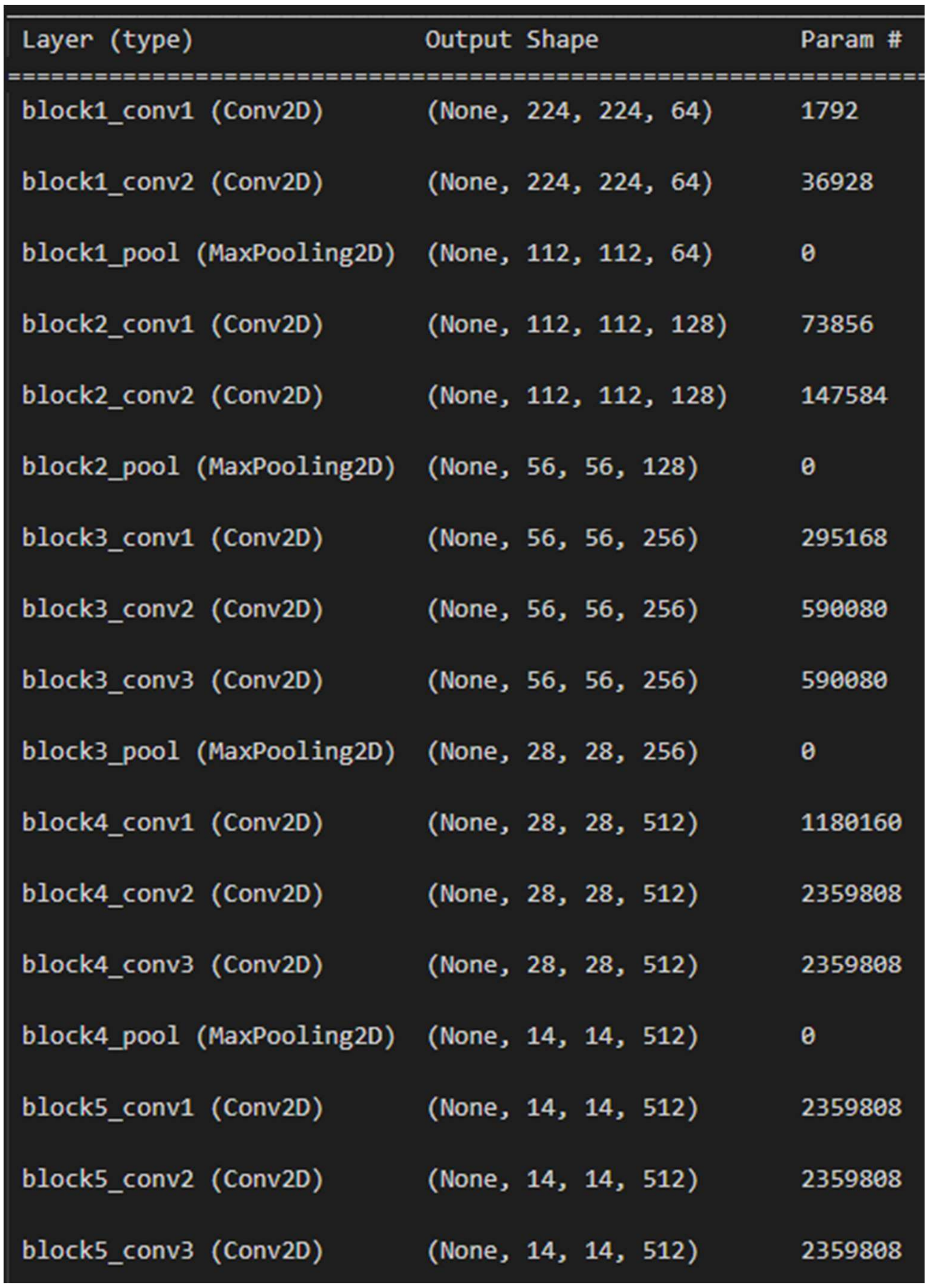
```
for layer in modelv2.layers[:-1]:
    layer.trainable = False

modelv2.add(Dropout(0.2))
modelv2.add(Dense(units=5, activation='softmax'))
modelv2.summary()
```

Figure 11: Dropout layer added after FC1 layer

A dropout layer is added as shown above as continuation of the dropout method being utilised in the current model. A value of 0.2 is given in the dropout function which indicates a probability of 20% for nodes to be dropped during each weight update cycle. (Godalle, 2022) FC1 stays unfrozen while layers before FC1 are frozen to maintain their weights during training as part of the Fine-Tuning process.

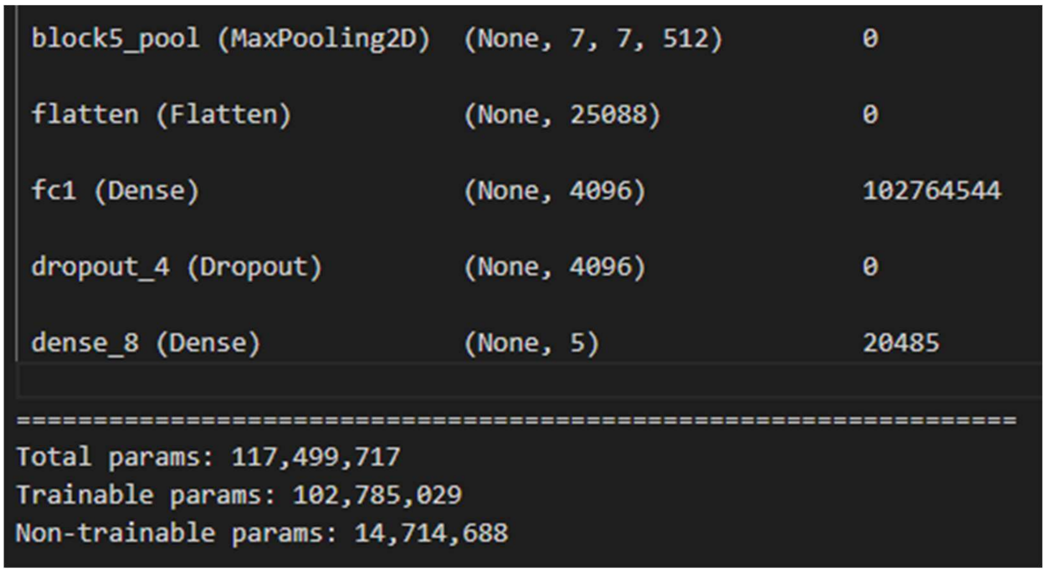
Model Architecture:

Below depicts the model architecture of the next-built model.



```
 Layer (type)                Output Shape              Param #
=================================================================
 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808
```

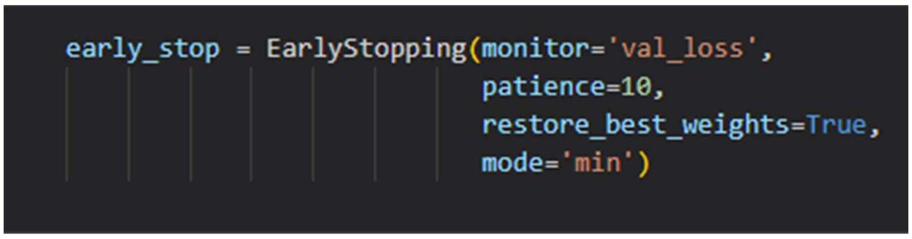
Figure 12: First half of updated-built model



```
block5_pool (MaxPooling2D)  (None, 7, 7, 512)        0

flatten (Flatten)           (None, 25088)            0

fc1 (Dense)                 (None, 4096)             102764544

dropout_4 (Dropout)         (None, 4096)             0

dense_8 (Dense)             (None, 5)                20485

=================================================================
Total params: 117,499,717
Trainable params: 102,785,029
Non-trainable params: 14,714,688
```

Figure 13: Second half of updated-built model

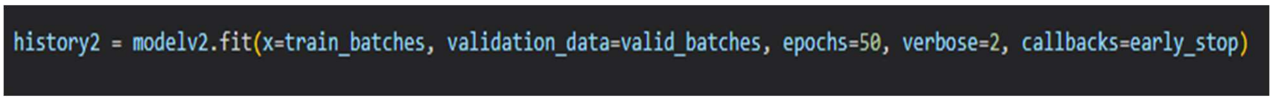
Early stopping is also applied for our updated model as shown below.



```
early_stop = EarlyStopping(monitor='val_loss',
                           patience=10,
                           restore_best_weights=True,
                           mode='min')
```

Figure 14: Application of Early Stopping in updated-built model

Epochs are raised to 50 as shown below with early stopping being implemented during fitting of model.



```
history2 = modelv2.fit(x=train_batches, validation_data=valid_batches, epochs=50, verbose=2, callbacks=early_stop)
```

Figure 15: Indication of Epochs raised

Evaluation Metrics

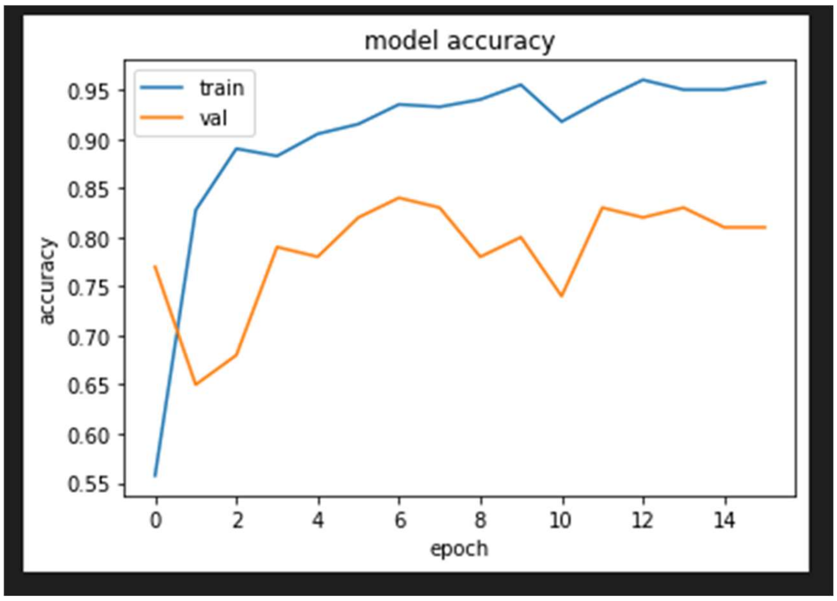


Figure 16: Results for Updated-built Fine-Tuned VGG16 Model

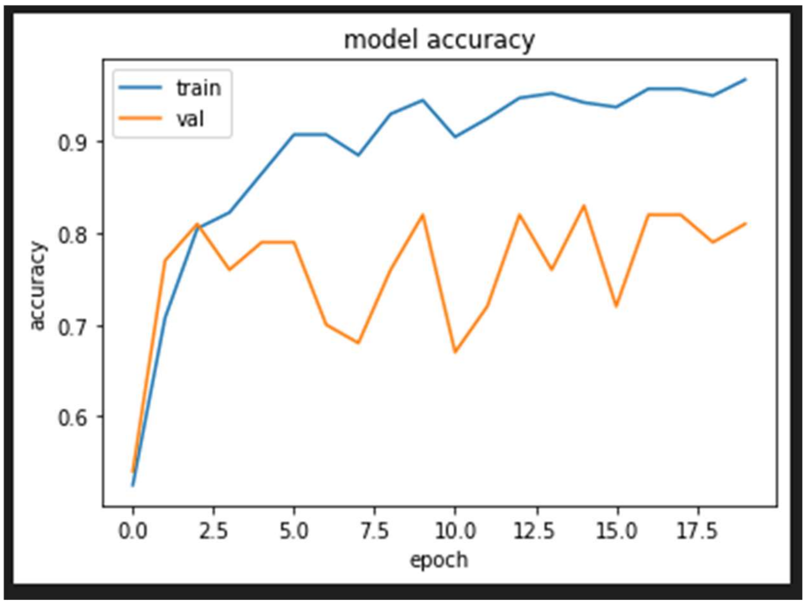


Figure 17: Results for previously-built Fine-Tuned VGG16 Model as shown in Figure 6

As observed above, the newly updated-built Fine-tuned VGG16 model has a slightly better curve and accuracy values as seen in the y-axis for its validation data component in terms of model accuracy as compared to the previously built fine-tuned model as shown above, effectively showing how overfitting is reduced and performance is increased in the current model as compared to the previous one.

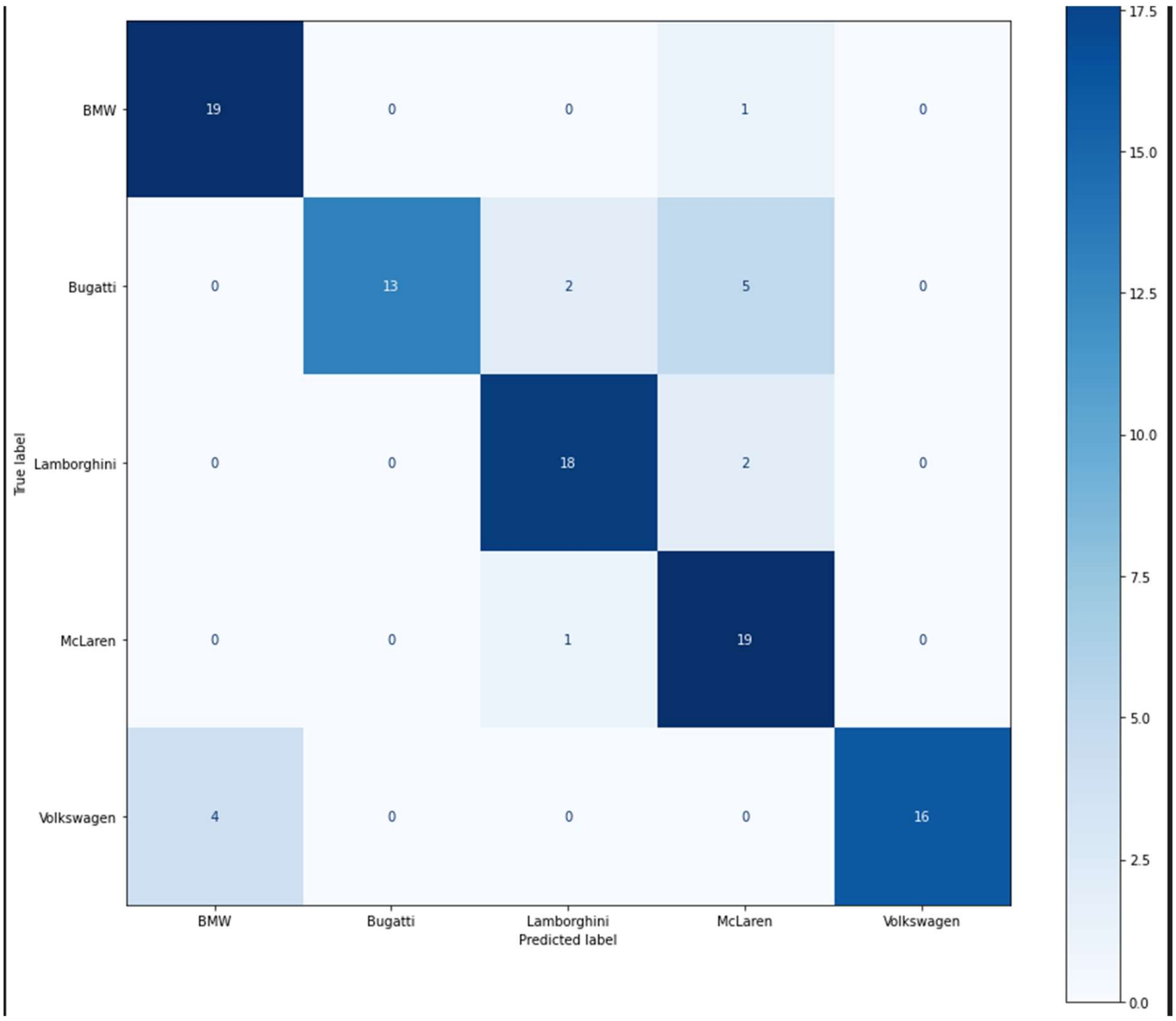
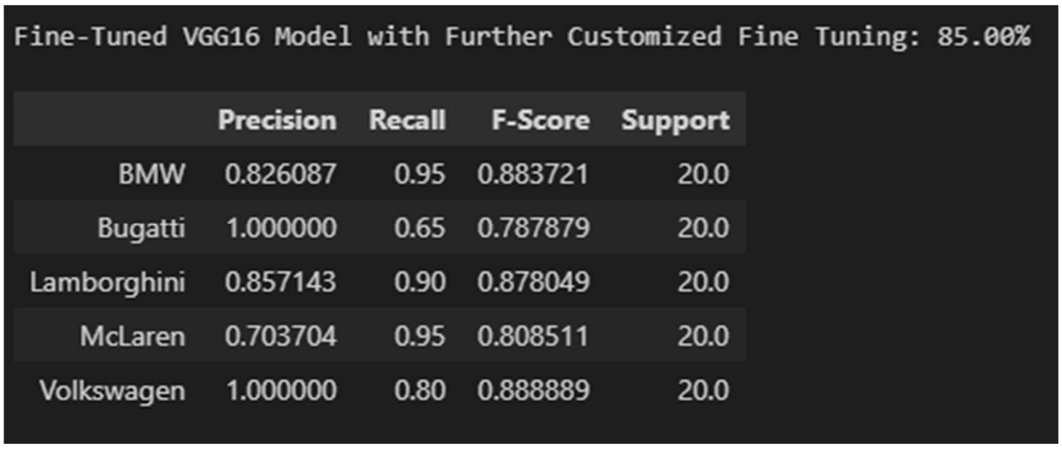


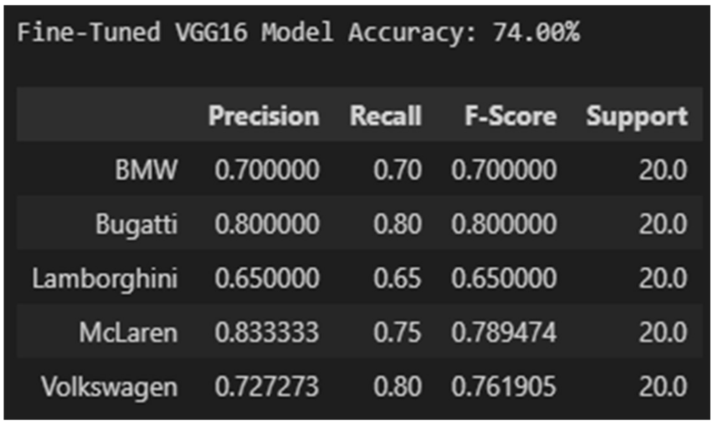
Figure 18: Confusion Matrix for Updated-built Fine-Tuned VGG16 Model



```
Fine-Tuned VGG16 Model with Further Customized Fine Tuning: 85.00%
```

	Precision	Recall	F-Score	Support
BMW	0.826087	0.95	0.883721	20.0
Bugatti	1.000000	0.65	0.787879	20.0
Lamborghini	0.857143	0.90	0.878049	20.0
McLaren	0.703704	0.95	0.808511	20.0
Volkswagen	1.000000	0.80	0.888889	20.0

Figure 19: Evaluation Metrics of updated-built Fine-Tuned VGG16 Model



```
Fine-Tuned VGG16 Model Accuracy: 74.00%
```

	Precision	Recall	F-Score	Support
BMW	0.700000	0.70	0.700000	20.0
Bugatti	0.800000	0.80	0.800000	20.0
Lamborghini	0.650000	0.65	0.650000	20.0
McLaren	0.833333	0.75	0.789474	20.0
Volkswagen	0.727273	0.80	0.761905	20.0

Figure 20: Evaluation Metrics of updated-built Fine-Tuned VGG16 Model

As observed from Figure 19 and Figure 20, our updated built Fine-Tuned VGG16 Model has an accuracy of 85.00% which is considered way more effective at predicting test data as compared to the previous model. The model also managed to score relatively way better in precision, recall and F-Scores for all classes as well, effectively proving its superiority in predicting test data.

Conclusion

As shown from the new evaluation metrics for the updated-built Fine-Tuned VGG16 model, the methods employed which include Early Stopping, adding dropout layers and fine-tuning have been very effective in achieve relatively better evaluation metrics results. Overfitting is also reduced to a certain extent which led to the creation of a better neural network model in the end.

Potential Improvements

Hyperparameters can also be considered when in comes to optimizing and building machine learning models. Hyperparameters are defined as parameters that can control the learning process of a neural network model.

- Type of optimizers used
- Value of Learning Rate
- Probability of Dropout
- Quantity of neurons in each layer within hidden layers

Other kinds of convolutional neural network models can also be used as well to build a new neural network architecture for experimentation which might potentially achieve greater accuracy in predicting test data. Examples include:

- ResNet50
- Inception
- VGG19

References

Baheti, P. (3 October, 2022). *What is Overfitting in Deep Learning [+10 Ways to Avoid It]*. Retrieved from V7labs: https://www.v7labs.com/blog/overfitting#:~:text=We%20can%20solve%20the%20problem,of%20epochs%20is%20set%20high

Brownlee, J. (11 November, 2020). *How to Identify Overfitting Machine Learning Models in Scikit-Learn*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/overfitting-machine-learning-models/

Godalle, E. (25 July, 2022). *What is a drop out rate in keras?* Retrieved from ProjectPro: https://www.projectpro.io/recipes/what-is-drop-out-rate-keras

Transfer learning and fine-tuning. (n.d.). Retrieved from TensorFlow: https://www.tensorflow.org/tutorials/images/transfer_learning