

Image Based Car Classification

Submitted by:
Haider Ali Khaliq
Zamna Razzaq
Muhammad Nihaal Asif

Introduction:

The classification of Cars based on their images is an important task in today's world. It can be used in various industries such as Traffic control or Law enforcement. Correctly classifying a car based on images provided of said car can go a long way to ensuring safety and security. A task made increasingly feasible through the power of machine learning and deep learning algorithms.

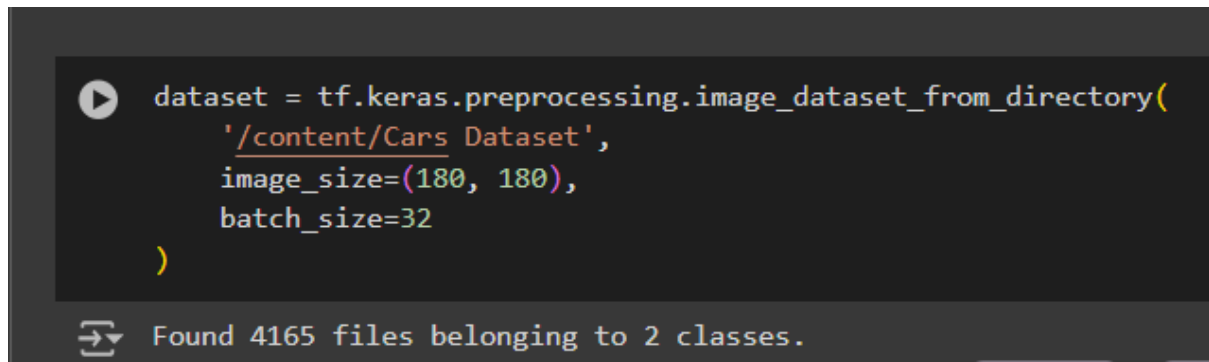
This report delves into the realm of automotive classification using a dataset comprising images of various car models belonging to seven distinct classes. Leveraging the prowess of machine learning and deep learning techniques, particularly Multilayer Perceptron (MLP), Convolutional Neural Networks (CNN), and the VGG architecture, the objective is to develop robust models capable of accurately classifying these vehicles.

Objective:

The primary objective of this report is to explore the efficiency of different machine learning and deep learning models in classifying a diverse range of car models. By harnessing the power of image classification algorithms, we aim to develop models that can accurately identify and categorise vehicles into their respective classes based on visual cues extracted from images.

1. Dataset Overview:

The dataset used in this study comprises images of vehicles from seven distinct classes: Audi, Hyundai Creta, Mahindra Scorpio, Rolls Royce, Swift, Tata Safari, and Toyota Innova. Each class represents a unique make and model of a car, providing a diverse and comprehensive collection for training and testing the classification models. The total images in the dataset are 4165.



```
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    '/content/Cars Dataset',
    image_size=(180, 180),
    batch_size=32
)
```

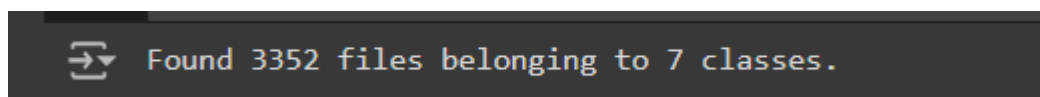
Found 4165 files belonging to 2 classes.

Moreover The dataset is divided into two parts.

1. Training Data
2. Testing Data

1.1 Training Data:

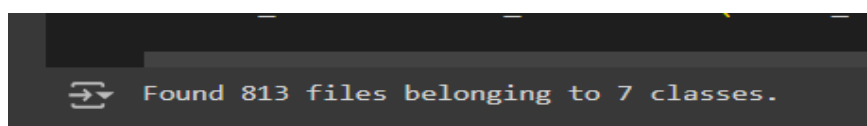
The training data comprises of 3352 images divided among 7 classes.



```
Found 3352 files belonging to 7 classes.
```

1.2 Testing data:

The testing Data comprises of 813 images belonging to 7 classes



```
Found 813 files belonging to 7 classes.
```

2. Preprocessing Data:

For our Project we have used a total of 525 images from the Training dataset and 140 images for Testing .The reason for this is that working with images requires a lot of computation power. At the current level we did not have enough computation power available to us so we decided to reduce the dataset accordingly. We further applied some preprocessing techniques on our dataset.

2.1 Tensor Conversion:

Converting images to tensors is a common practice in deep learning and machine learning workflows, and it serves several purposes:

- **Compatibility with TensorFlow:**

TensorFlow, being a popular deep learning framework, operates efficiently with tensors. Converting images to tensors allows seamless integration with TensorFlow's computation graph, enabling operations like convolution, pooling, and matrix multiplication to be performed directly on image data.

- **Efficient Data Processing:**

Tensors are multi-dimensional arrays that facilitate efficient numerical computations. By converting images to tensors, we can leverage TensorFlow's optimizer numerical operations for tasks such as normalisation, augmentation, and model training.

- **GPU Acceleration:**

Modern deep learning frameworks like TensorFlow are optimised to harness the computational power of GPUs for accelerated training. GPUs are highly efficient at processing numerical data stored in tensors, making tensor-based representations ideal for training deep learning models.

2.2 Gray-scale Images:

Gray-scaling images is a very common step in preprocessing data. It serves many benefits like Dimensionality Reduction etc. Gray-scaling of images reduces the image to a single channel whereas RGB images have

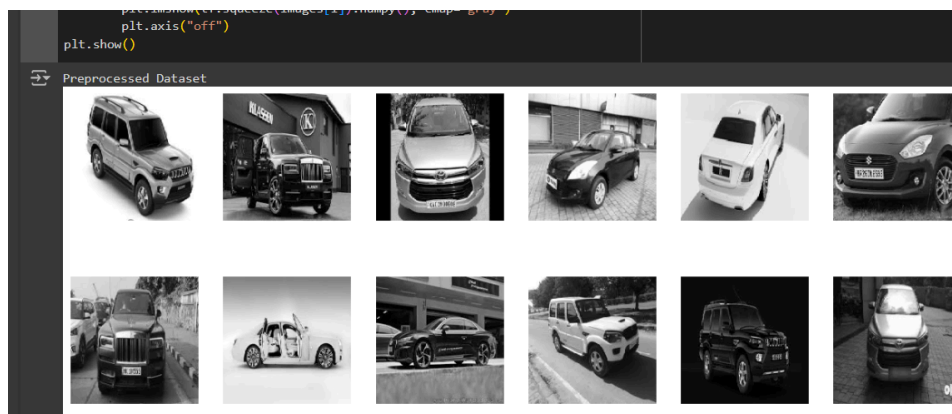
3 channels. By doing so we retain the important information of the images such as their texture, shape and the contrast of objects in them. These features are usually enough for any Deep learning classifiers. In simpler terms by Gray-scaling an image all the important information is retained while also making the data more efficient for the models. We applied the Gray-scaling techniques on our data as a part of preprocessing so that our data becomes more efficient for computation.

2.3 Normalisation:

Normalisation is applied on images for various reasons. We applied it mainly because it helps in convergence in Training of the data. Normalisation of images Normalising pixel values to a specific range, typically $[0, 1]$ or $[-1, 1]$, can help accelerate model convergence during training. Normalised data reduces the scale of input features, mitigating the issues of vanishing or exploding gradients commonly encountered in deep neural networks.

```
def convert_to_grayscale(image, label):  
    grayscale_image = tf.image.rgb_to_grayscale(image)  
    return grayscale_image, label  
  
[ ] normalization_layer = tf.keras.layers.Rescaling(1./255)  
    grayscale = dataset_train.map(convert_to_grayscale)  
  
    preprocessed_dataset = grayscale.map(lambda x, y: (normalization_layer(x), y))
```

2.4 Preprocessed Dataset:

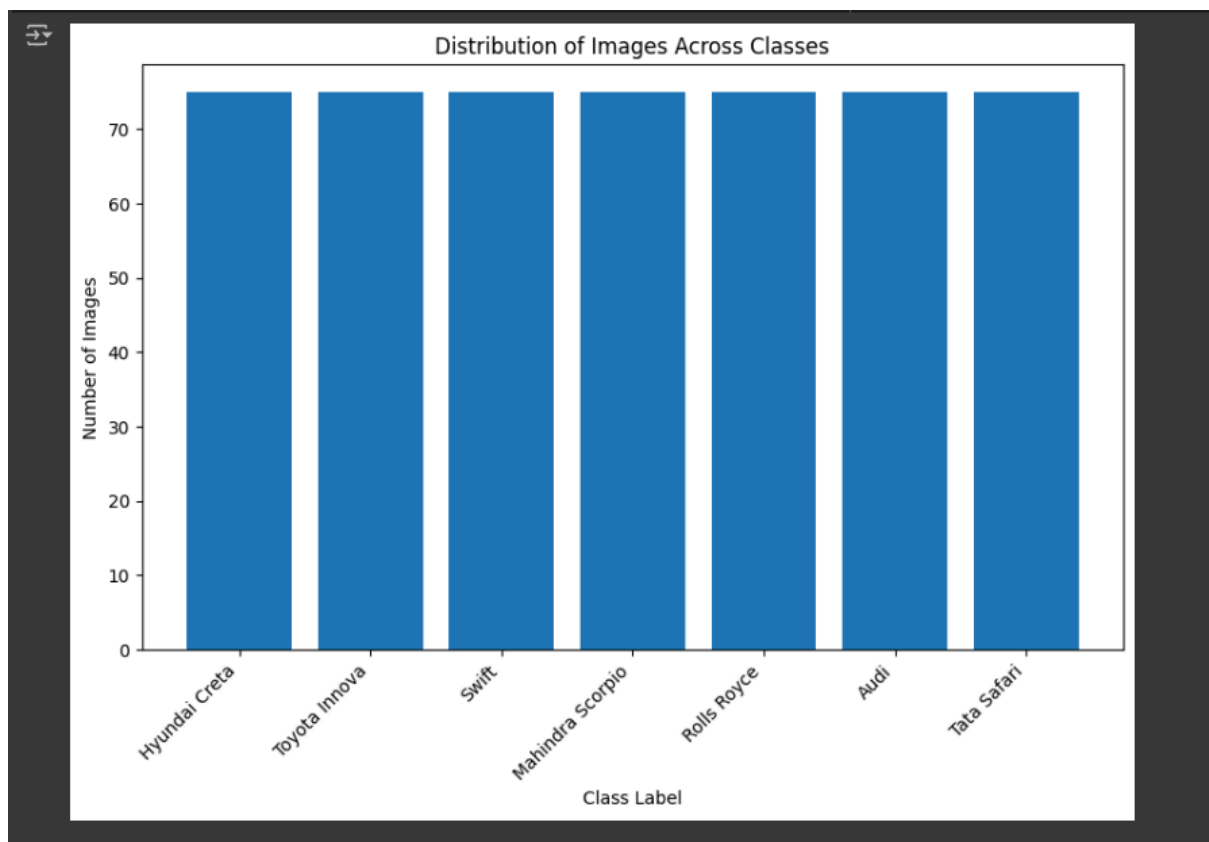


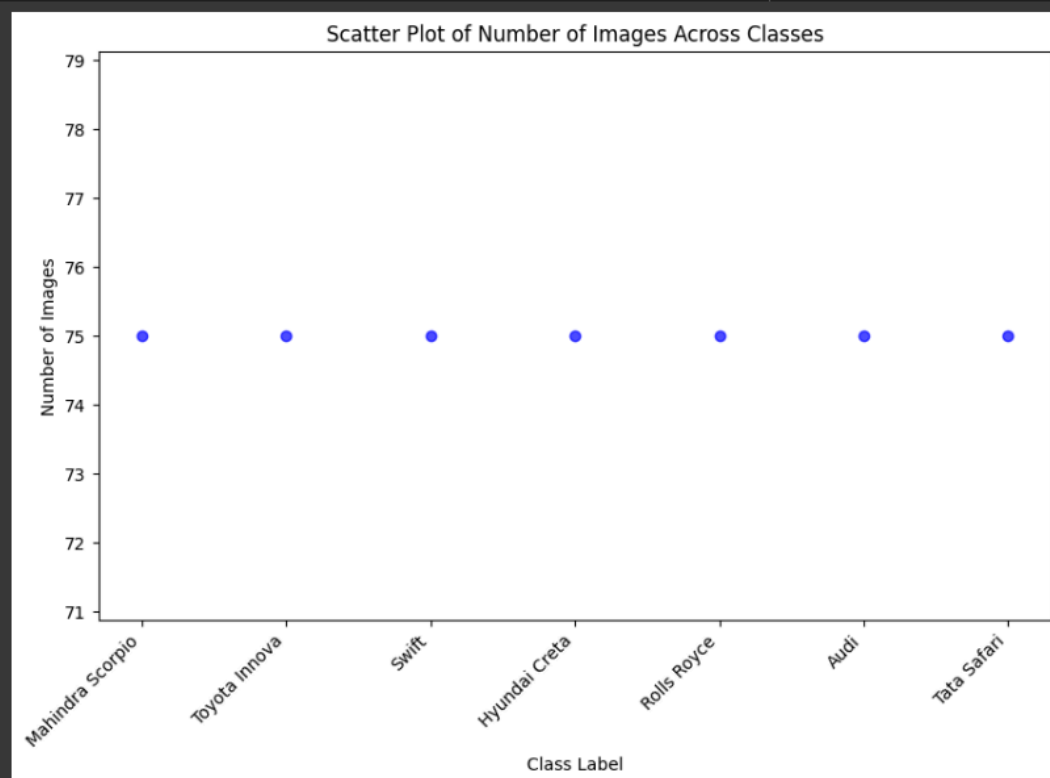
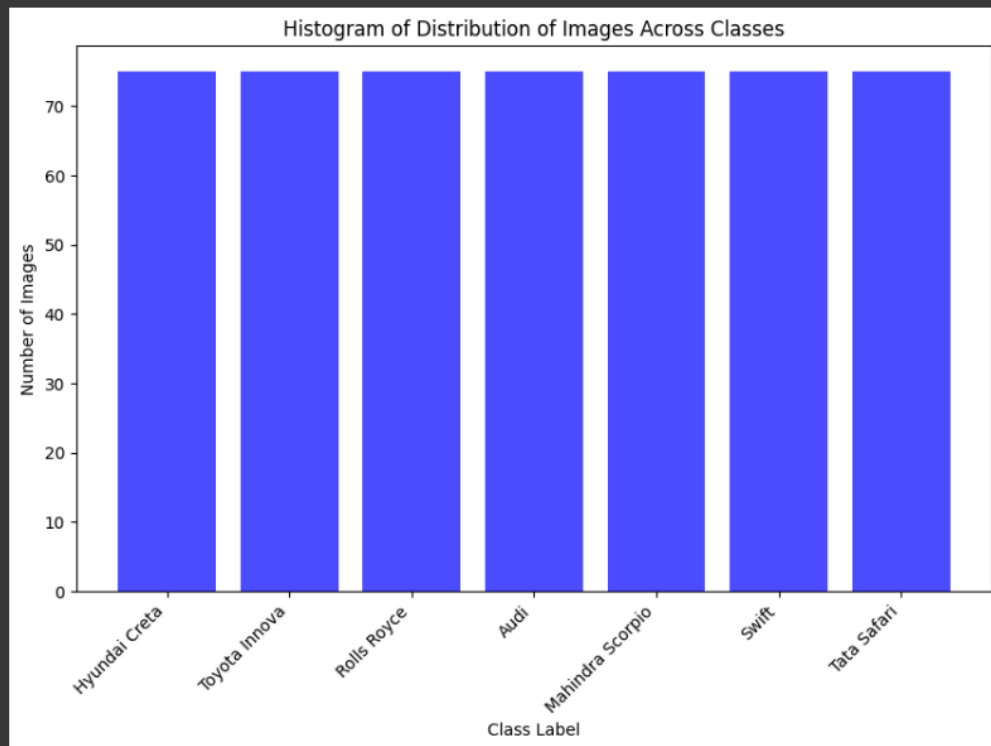
3. Visualization:

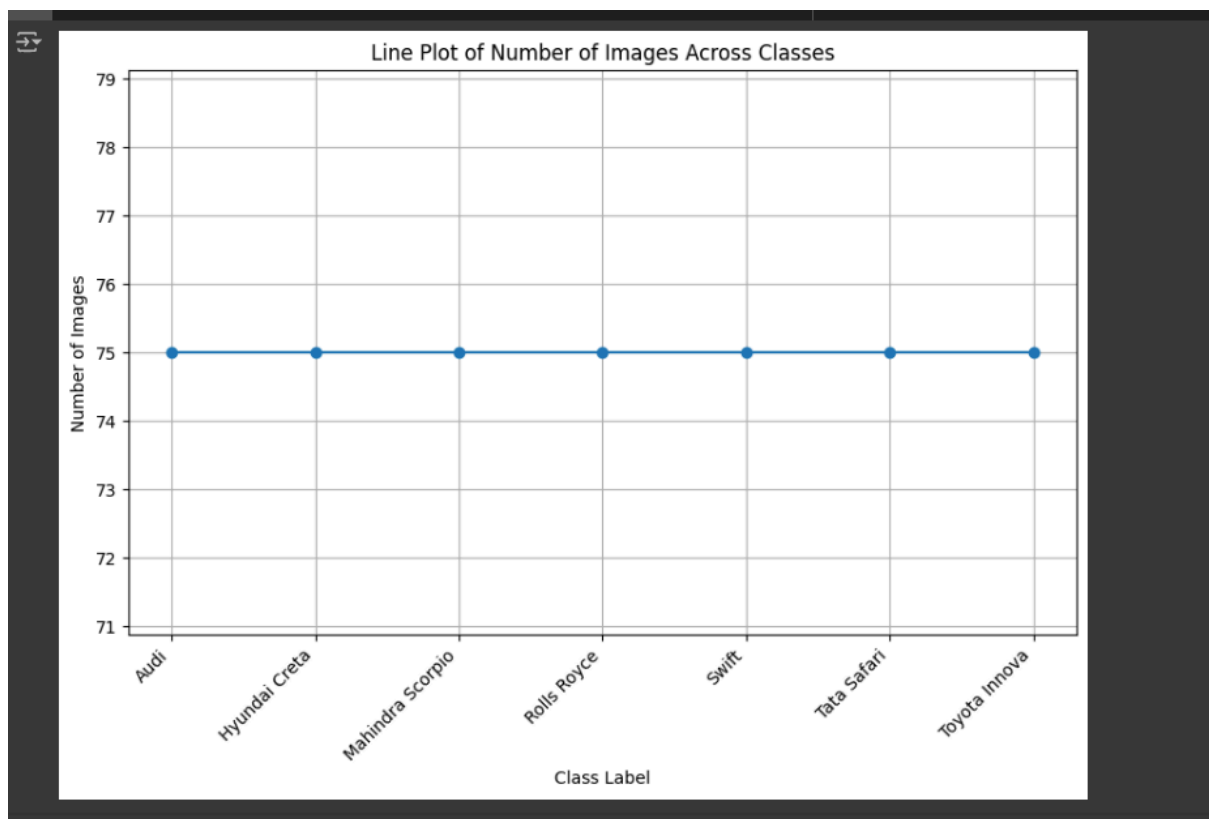
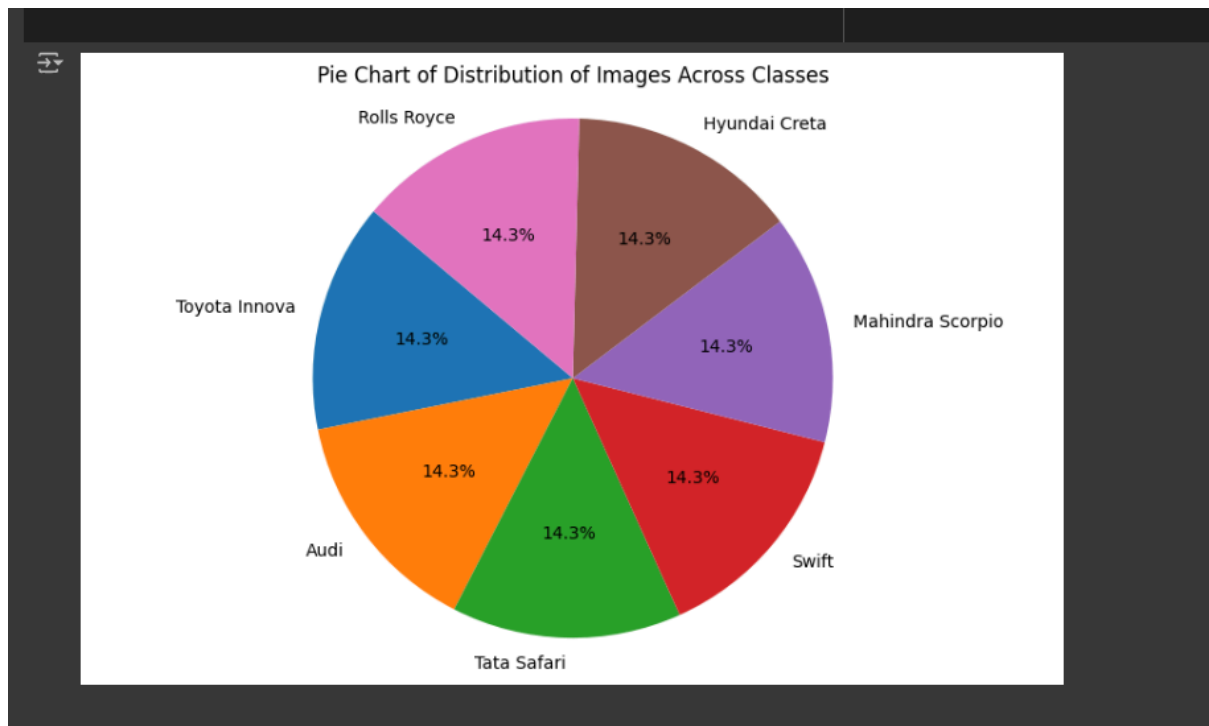
We applied 5 Visualization Techniques as was instructed to us namely:

- Bar plot
- Histogram
- Line plot
- Pie chart
- Scatter plot

The results are as follows:



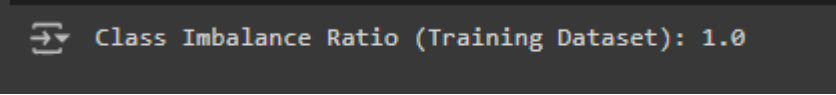




The visualisation techniques show that we have selected 75 images each from the 7 classes.

4. Class imbalance:

The next step was to check for class imbalances. Our dataset did not show any class imbalances since we acquired equal amounts of data from all classes hence, Class imbalance was not a problem in our case.



```
↩ Class Imbalance Ratio (Training Dataset): 1.0
```

5. Classification Models:

For Classification we have applied 3 Models Multi-Layer perceptron(MLP) , Convolutional Neural Network(CNN) and Transfer Learning(VGG).

5.1 Multi layer perceptron(MLP) :

The MLP model can be explained by breaking it into two main components. “The Model Architecture” and “The Model Compilation”

Model Architecture:

- **Input Layer (Flatten):** The input layer flattens the input data, which is a 180x180 grayscale image (1 channel), into a one-dimensional array. This layer serves as the input to the subsequent dense layers.
- **Dense Layer (128 units, ReLU activation):** The first dense layer consists of 128 units with a rectified linear unit (ReLU) activation function. This layer performs a linear transformation of the input data followed by the application of the ReLU activation function, introducing non-linearity to the model.
- **Dense Layer (64 units, ReLU activation):** The second dense layer consists of 64 units with a ReLU activation function. Similar to the previous layer, it applies a linear transformation followed by the ReLU activation function

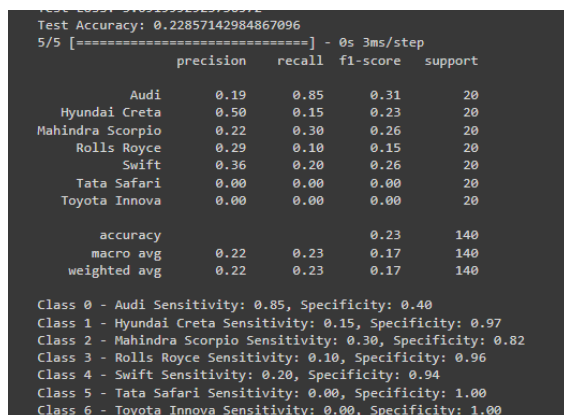
- **Output Layer (Dense):** The output layer consists of a dense layer with a softmax activation function. This layer has as many units as the number of unique classes in the dataset (determined by ``len(np.unique(y_train))``). The softmax activation function converts the raw output scores into probabilities, representing the likelihood of each class.

Model Compilation:

- **Optimizer:** The model is compiled with the Adam optimizer, a popular choice for deep learning tasks due to its adaptive learning rate properties and computational efficiency.
- **Loss Function:** The sparse categorical cross entropy loss function is specified, suitable for multi-class classification tasks where the labels are integers (as opposed to one-hot encoded vectors). This loss function computes the cross-entropy between the true labels and the predicted probabilities.
- **Metrics:** The model is evaluated based on the accuracy metric during training. Accuracy represents the proportion of correctly classified samples in the dataset.

Reporting the Results:

The Mlp performed adequately on our Data it provided an Accuracy of 23%. Whereas this accuracy is nowhere near the model accuracy It can still be justified. There was a number of reasons why our model performed poorly. It performed this way mainly because of the lack of data it was provided, Our model was given very less training data causing it to perform poorly. We could not provide more data since we were bound computational wise. Moreover we provided more metrics other than Accuracy. Below is a screenshot of those metrics.



```

Test Accuracy: 0.22857142984867096
5/5 [=====] - 0s 3ms/step

```

	precision	recall	f1-score	support
Audi	0.19	0.85	0.31	20
Hyundai Creta	0.50	0.15	0.23	20
Mahindra Scorpio	0.22	0.30	0.26	20
Rolls Royce	0.29	0.10	0.15	20
Swift	0.36	0.20	0.26	20
Tata Safari	0.00	0.00	0.00	20
Toyota Innova	0.00	0.00	0.00	20
accuracy			0.23	140
macro avg	0.22	0.23	0.17	140
weighted avg	0.22	0.23	0.17	140

Class 0 - Audi	Sensitivity: 0.85, Specificity: 0.40
Class 1 - Hyundai Creta	Sensitivity: 0.15, Specificity: 0.97
Class 2 - Mahindra Scorpio	Sensitivity: 0.30, Specificity: 0.82
Class 3 - Rolls Royce	Sensitivity: 0.10, Specificity: 0.96
Class 4 - Swift	Sensitivity: 0.20, Specificity: 0.94
Class 5 - Tata Safari	Sensitivity: 0.00, Specificity: 1.00
Class 6 - Toyota Innova	Sensitivity: 0.00, Specificity: 1.00

Precision Explained for Each Class

- **Audi : Precision: 0.19**
- **Hyundai Creta: Precision: 0.50**
- **Mahindra Scorpio :Precision: 0.22**
- **Rolls Royce :Precision: 0.29**
- **Swift : Precision: 0.36**
- **Tata Safari : Precision: 0.00**
- **Toyota Innova : Precision: 0.00**

Out of all the classes the precision for Hyundai-Creta is the relatively highest one. But generally the precision is low. This is due to the fact that the Training data provided was not enough for the classifier but due to computational limitations we could not provide more Training data

Recall Explained for Each Class:

Recall means that how much of the actual data has been correctly identified. It is also known as sensitivity.

- **Audi :Recall: 0.85**
- **Hyundai Creta: Recall: 0.15**
- **Mahindra Scorpio: Recall: 0.30**
- **Rolls Royce :Recall: 0.10**
- **Swift : Recall: 0.20**
- **Tata Safari : Recall: 0.00**
- **Toyota Innova : Recall: 0.00**

The recall, same as precision, is relatively low. Except for Audi which has 85% of its samples correctly classified. The reason for a low recall was the same as the precision. The training dataset was not enough for the classifier

General Application to Classes:

F1 Score: Indicates whether class is well-predicted with a good balance between precision and recall or not.

Support :Indicates a larger number of examples of that class in the dataset, which usually leads to more reliable estimates of precision, recall, and hence the F1 score.

High Specificity: Suggests that the classifier is good at not labelling negatives as this class, which is especially important in applications where false positives are more detrimental than false negatives.

In our case we had low F1-Score same as recall and precision but our Specificity was High for most of the classes indicating that the model is good at not labelling negatives.

Fine tuning the hyper parameters:

Tuning hyperparameters is essential for improving the accuracy and generalisation of the model for car recognition. By adjusting hyperparameters such as learning rate, batch size, and model architecture, we can enhance the model's ability to learn relevant features from the dataset. This optimization process helps prevent overfitting or underfitting, ensuring that the model performs well on unseen data and effectively captures the nuances of different cars. Ultimately, fine-tuning hyperparameters is crucial for maximising the model's accuracy while efficiently utilising computational resources.

To fine tune the hyper parameters we provided these params:

```
param_grid = {  
    'hidden_layer_sizes': [(128,), (200,)],  
    'activation': ['relu', 'tanh'],  
    'alpha': [0.001], # Regularization term  
    'batch_size': [32], # Batch size for mini-batch training  
    'max_iter': [100] # Maximum number of iterations or epochs  
}
```

The reason for providing less params was again computational limitations. Nonetheless, we saw a decrease in the overall accuracy of the model To 17%.

```

Classification Report:
              precision    recall  f1-score   support

     0       0.14       0.20       0.16         20
     1       0.15       0.10       0.12         20
     2       0.18       0.15       0.16         20
     3       0.16       0.20       0.18         20
     4       0.18       0.15       0.16         20
     5       0.29       0.20       0.24         20
     6       0.16       0.20       0.18         20

 accuracy          0.17         140
 macro avg          0.18         140
 weighted avg       0.18         140

Confusion Matrix:
[[4 3 3 3 1 3]
 [4 2 5 3 4 1]
 [2 4 3 1 5 1]
 [6 1 2 4 0 2]
 [3 1 0 6 3 1]
 [4 1 4 4 1 4]
 [6 1 0 4 1 4]]
Class 0 - Sensitivity (Recall): 0.20, Specificity: 0.74
Class 1 - Sensitivity (Recall): 0.10, Specificity: 0.90
Class 2 - Sensitivity (Recall): 0.15, Specificity: 0.87
Class 3 - Sensitivity (Recall): 0.20, Specificity: 0.79
Class 4 - Sensitivity (Recall): 0.15, Specificity: 0.87
Class 5 - Sensitivity (Recall): 0.20, Specificity: 0.91
Class 6 - Sensitivity (Recall): 0.20, Specificity: 0.79
Validation set score: 0.17

```

Suggesting that the original hyper-parameters given to the model were optimum in this case.

5.2 Convolutional Neural Network(CNN) :

The CNN model can be explained by breaking it into two main components. “The Model Architecture” and “The Model Compilation”

Model Architecture:

- **Convolutional Layers:** - The first convolutional layer (`Conv2D`) consists of 32 filters with a kernel size of (3, 3) and ReLU activation function. It operates on input images with a shape of (180, 180, 1) (grayscale images). A max-pooling layer (`MaxPooling2D`) follows, which reduces the spatial dimensions of the feature maps by taking the maximum value in each 2x2 region. The second convolutional layer consists of 64 filters with a kernel size of (3, 3) and ReLU activation function. Another max-pooling layer follows. The third convolutional layer consists of 64 filters with a kernel size of (3, 3) and ReLU activation function.
- **Flattening Layer:** After the convolutional layers, the output is flattened into a one-dimensional vector using the `Flatten` layer. This prepares the data for input into the dense layers.

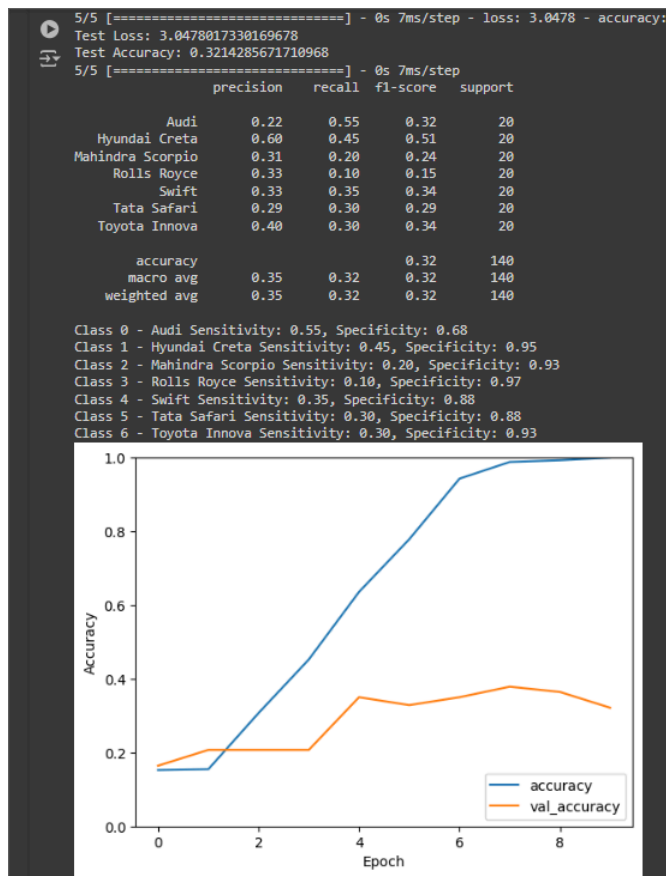
- **Dense Layers:** Two dense layers follow the flattened layer, each with ReLU activation functions. The first dense layer consists of 64 units, and the output layer consists of as many units as the number of unique classes in the dataset. The output layer uses the softmax activation function to output class probabilities.

Model Compilation:

- **Optimizer:** The model is compiled with the Adam optimizer, a popular choice for gradient-based optimization algorithms in deep learning.
- **Loss Function:** The sparse categorical cross entropy loss function is specified, suitable for multi-class classification tasks where the labels are integers.
- **Metrics:** The model is evaluated based on the accuracy metric during training and validation.

Reporting the Results:

The CNN performed better on our Data than on MLP it provided an Accuracy of 32%. Whereas this accuracy is nowhere near the model accuracy it can still be justified. There was a number of reasons why our model performed poorly. It performed this way mainly because of the lack of data it was provided, Our model was given very less training data causing it to perform poorly. We could not provide more data since we were bound computational wise. Moreover we provided more metrics other than Accuracy. Below is a screenshot of those metrics.



Precision Explained for Each Class

- **Audi : Precision: 0.22**
- **Hyundai Creta: Precision: 0.60**
- **Mahindra Scorpio :Precision: 0.31**
- **Rolls Royce :Precision: 0.33**
- **Swift : Precision: 0.33**
- **Tata Safari : Precision: 0.29**
- **Toyota Innova : Precision: 0.40**

Out of all the classes the precision for Hyundai-Creta is the relatively highest one. But generally the precision is low. This is due to the fact that the Training data provided was not enough for the classifier but due to computational limitations we could not provide more Training data. Moreover in the case of CNN tata safari and toyota innova also have some amount of precision whereas in the case of MLP they were 0.00.

Recall Explained for Each Class:

Recall means that how much of the actual data has been correctly identified. It is also known as sensitivity.

- **Audi :Recall: 0.55**
- **Hyundai Creta: Recall: 0.45**
- **Mahindra Scorpio: Recall: 0.20**
- **Rolls Royce :Recall: 0.10**
- **Swift : Recall: 0.35**
- **Tata Safari : Recall: 0.30**
- **Toyota Innova : Recall: 0.30**

The recall, same as precision, is relatively low. In CNN the recall for AUDI also decreased but the same as precision Tata safari and Toyota innova have increased recall The reason for a low recall was the same as the precision. The training dataset was not enough for the classifier

General Application to Classes:

F1 Score: Indicates whether class is well-predicted with a good balance between precision and recall or not.

Support :Indicates a larger number of examples of that class in the dataset, which usually leads to more reliable estimates of precision, recall, and hence the F1 score.

High Specificity: Suggests that the classifier is good at not labelling negatives as this class, which is especially important in applications where false positives are more detrimental than false negatives.

CNN has a relatively higher F1 score for each class than MLP also the specificity is also better. This suggests that overall CNN was a better model for classification of our dataset than MLP.

Fine tuning the hyper parameters:

Tuning hyperparameters is essential for improving the accuracy and generalisation of the model for car recognition. By adjusting hyperparameters such as learning rate, batch size, and model architecture, we can enhance the model's ability to learn relevant features from the dataset. This optimization process helps prevent overfitting or underfitting, ensuring that the model

performs well on unseen data and effectively captures the nuances of different cars. Ultimately, fine-tuning hyperparameters is crucial for maximising the model's accuracy while efficiently utilising computational resources.

To fine tune the hyper parameters we provided these params:

```
# Define the parameter grid for grid search
param_grid = {
    'filters': [32, 64], # Different numbers of filters
    'kernel_size': [(3, 3)], # Size of the kernel in the convolutional layers.
    'activation': ['relu'], # Activation function to be used.
    'dropout_rate': [0.5] # Dropout rate to prevent overfitting.
}
```

The reason for providing less params was again computational limitations. Nonetheless, we saw a decrease in the overall accuracy of the model to 19%.

```
Classification Report:
              precision    recall  f1-score   support

     0         0.00         0.00         0.00         17
     1         0.00         0.00         0.00         17
     2         0.29         0.44         0.35         18
     3         0.00         0.00         0.00         16
     4         0.31         0.27         0.29         15
     5         0.20         0.47         0.28         15
     6         0.08         0.14         0.10         14

 accuracy          0.19
 macro avg         0.12
 weighted avg      0.12
```

```
Confusion Matrix:
[[0 1 6 0 0 3 7]
 [0 0 4 0 0 6 7]
 [1 0 8 0 2 4 3]
 [0 2 4 0 5 5 0]
 [0 0 2 2 4 3 4]
 [2 0 0 1 2 7 3]
 [0 1 4 0 0 7 2]]
```

```
Sensitivity and Specificity by Class:
Class 0: Sensitivity = 0.00, Specificity = 0.00
Class 1: Sensitivity = 0.00, Specificity = 0.00
Class 2: Sensitivity = 0.44, Specificity = 0.29
Class 3: Sensitivity = 0.00, Specificity = 0.00
Class 4: Sensitivity = 0.27, Specificity = 0.31
Class 5: Sensitivity = 0.47, Specificity = 0.20
Class 6: Sensitivity = 0.14, Specificity = 0.08
```

Like the MLP Model CNN also suggests that the original hyper parameters given to the model were better.

Conclusion:

The conclusion to the hyper-parameter tuning is that in both our models CNN and MLP tuning the hyper parameters resulted in a lower accuracy than the original models with the original hyper parameters. Now this is a very unorthodox result. The reason for this result is that we only gave both the models a small chunk of hyper-parameters to tune. Whereas in reality if given all of the combinations of the hyper-parameters both the models would have performed substantially. We did not give all the combinations due to the computational limitations we were facing. Even if we did tune the hyper-parameters to the optimum it would have been difficult to achieve an accuracy of 90-95% or more the reason for that would be the Dataset reduction. Our dataset was reduced from its original size as mentioned in the start as well due to the reduction of the Training dataset the models could not accurately learn it and hence performed poorly.

5.3 Transfer Learning(VGG) :

VGG, which stands for Visual Geometry Group, is a convolutional neural network (CNN) architecture proposed by researchers at the University of Oxford. It is renowned for its simplicity and effectiveness in image classification tasks. The key idea behind VGG is to use a deep network with very small (3x3) convolutional filters, which are stacked together to increase the depth of the network while keeping the receptive fields of the neurons relatively small.

Architecture:

- **Convolutional Layers:** The core building blocks of VGG are convolutional layers. VGG typically consists of multiple blocks of convolutional layers followed by max-pooling layers. Each convolutional layer uses small receptive fields (e.g., 3x3) with a stride of 1, and the number of filters increases with the depth of the network. The use of small filters enables VGG to capture detailed patterns in the input images.
- **Pooling Layers:** After each block of convolutional layers, max-pooling layers are used to reduce the spatial dimensions of the feature maps

while retaining the most important information. Max-pooling is performed over small windows (e.g., 2x2) with a stride of 2, effectively downsampling the feature maps.

- **Fully Connected Layers:** Towards the end of the network, fully connected layers are added to perform high-level reasoning and make predictions. The fully connected layers take the flattened output from the last convolutional or pooling layer and transform it into class probabilities.

VGG Variants:

- **VGG16:** The original VGG architecture with 16 weight layers, including 13 convolutional layers and 3 fully connected layers.
- **VGG19:** Similar to VGG16, but with 19 weight layers, offering slightly improved performance at the cost of increased computational complexity.

Working Principle:

- **Feature Extraction:** The initial layers of VGG perform feature extraction by convolving the input image with learnable filters. Each convolutional layer captures different features, such as edges, textures, and shapes, at different levels of abstraction. As the network deepens, the convolutional layers learn increasingly complex features, representing higher-level semantics of the input images.
- **Hierarchical Representation:** Through multiple layers of convolutions and pooling, VGG builds a hierarchical representation of the input images. Lower layers capture low-level features, while deeper layers capture more abstract and high-level features.
- **Classification:** The hierarchical feature representation obtained from the convolutional layers is then flattened and fed into fully connected layers for classification. - The final layer typically uses softmax activation to produce class probabilities, enabling VGG to predict the most likely class label for the input image.

Advantages:

- **Simplicity:** VGG's architecture is simple and easy to understand, making it widely used and studied in the field of computer vision.

- **Effectiveness:** Despite its simplicity, VGG achieves competitive performance on various image classification benchmarks, demonstrating the effectiveness of deep convolutional neural networks.

Limitations:

- **Computational Complexity:** VGG has a large number of parameters, leading to high computational and memory requirements during training and inference.
- **Overfitting:** With a large number of parameters, VGG is prone to overfitting, especially when trained on small datasets.

Implementation:

Our model is not VGG16 or VGG19. It is a custom model based on the VGG architecture. While it utilises some concepts from VGG, such as the use of convolutional layers, max-pooling layers, and fully connected layers, it does not strictly adhere to the exact architecture of either VGG16 or VGG19.

In Our custom model:

- we start with a base model using the pre-trained weights of VGG16.
- Then, we add custom layers on top of the base model, including a Flatten layer, Dense layer with ReLU activation, and another Dense layer with softmax activation for classification.
- This custom architecture allows you to leverage the feature extraction capabilities of the pre-trained VGG16 model while adding your own layers to adapt it to your specific classification task.

Results:

Our VGG model has a total accuracy of 52% The VGG model has performed better than the CNN model and the Mlp model. This is due to the fact that the VGG model has a vast Training dataset on which it is already pre trained. Due to this pre training the VGG model already has learnt complex patterns and in result it performs better than the other models. In our case VGG has better Accuracy,precision,recall,sensitivity and specificity All due to the fact that it was already pre-trained using the imagenet dataset.

```

5/5 [=====] - 54s 10s/step - loss: 1.4666
Test Accuracy: 0.5142857432365417
5/5 [=====] - 55s 11s/step

Classification Report:
      precision    recall  f1-score   support

   Audi           0.57       0.40       0.47        20
 Hyundai Creta    0.50       0.80       0.62        20
 Mahindra Scorpio 0.65       0.55       0.59        20
  Rolls Royce     0.55       0.60       0.57        20
    Swift         0.56       0.50       0.53        20
   Tata Safari    0.39       0.35       0.37        20
 Toyota Innova    0.42       0.40       0.41        20

 accuracy          0.51       0.51       0.51       140
  macro avg        0.52       0.51       0.51       140
 weighted avg        0.52       0.51       0.51       140

Confusion Matrix:
[[ 8  3  1  3  0  5  0]
 [ 0 16  0  1  2  1  0]
 [ 0  2 11  2  2  1  2]
 [ 2  1  1 12  1  1  2]
 [ 0  5  1  1 10  0  3]
 [ 2  3  1  0  3  7  4]
 [ 2  2  2  3  0  3  8]]

```

Sensitivity by Class:

Class Audi: 0.40
 Class Hyundai Creta: 0.80
 Class Mahindra Scorpio: 0.55
 Class Rolls Royce: 0.60
 Class Swift: 0.50
 Class Tata Safari: 0.35
 Class Toyota Innova: 0.40

Specificity by Class:

Class Audi: 0.95
 Class Hyundai Creta: 0.87
 Class Mahindra Scorpio: 0.95
 Class Rolls Royce: 0.92
 Class Swift: 0.93
 Class Tata Safari: 0.91
 Class Toyota Innova: 0.91

Precision Explained for Each Class

- Audi : Precision: 0.57
- Hyundai Creta: Precision: 0.50
- Mahindra Scorpio :Precision: 0.65
- Rolls Royce :Precision: 0.55
- Swift : Precision: 0.56
- Tata Safari : Precision: 0.39
- Toyota Innova : Precision: 0.42

Out of all the classes the precision for the Mahindra scorpio is the relatively highest one. But generally the precision is normal. This is due to the fact that the Training data provided was not enough but the model performed better than the other two models because it had a vast dataset on which it was pre-trained . Moreover in the case of VGG all 7 of the classes have a normal amount of precision

Recall Explained for Each Class:

Recall means that how much of the actual data has been correctly identified. It is also known as sensitivity.

- **Audi :Recall: 0.40**
- **Hyundai Creta: Recall: 0.80**
- **Mahindra Scorpio: Recall: 0.55**
- **Rolls Royce :Recall: 0.60**
- **Swift : Recall: 0.50**
- **Tata Safari : Recall: 0.35**
- **Toyota Innova : Recall: 0.40**

The recall, same as precision, is relatively better In VGG. The recall for All the classes has significantly increased in the VGG model. This goes to show that the pre-training or the acquisition of the pre-trained weights from the imagenet dataset has significantly improved the working of the VGG model even when given a dataset which is less compared to the normal standard.

Evaluation metrics:

The rest of the evaluation metrics namely specificity and f1 score have also shown an increase when compared to the previous models this goes to show that pre-trained weights have gone a long way in bettering the output of our models. This also reinforces our previous claim that, Our models are not performing as well as they should because we are providing less Data for training. Where as in this case VGG was provided by more Training data so it performed better.

6. Bonus Models:

We implemented a bonus model as was instructed to us. The model that we implemented was the Inception Model.

6.1 Inception (V3) :

The Inception model, specifically InceptionV3, is a convolutional neural network (CNN) architecture designed by Google. It is widely known for its efficiency in both training and inference while achieving high accuracy on image classification tasks.

Model Architecture:

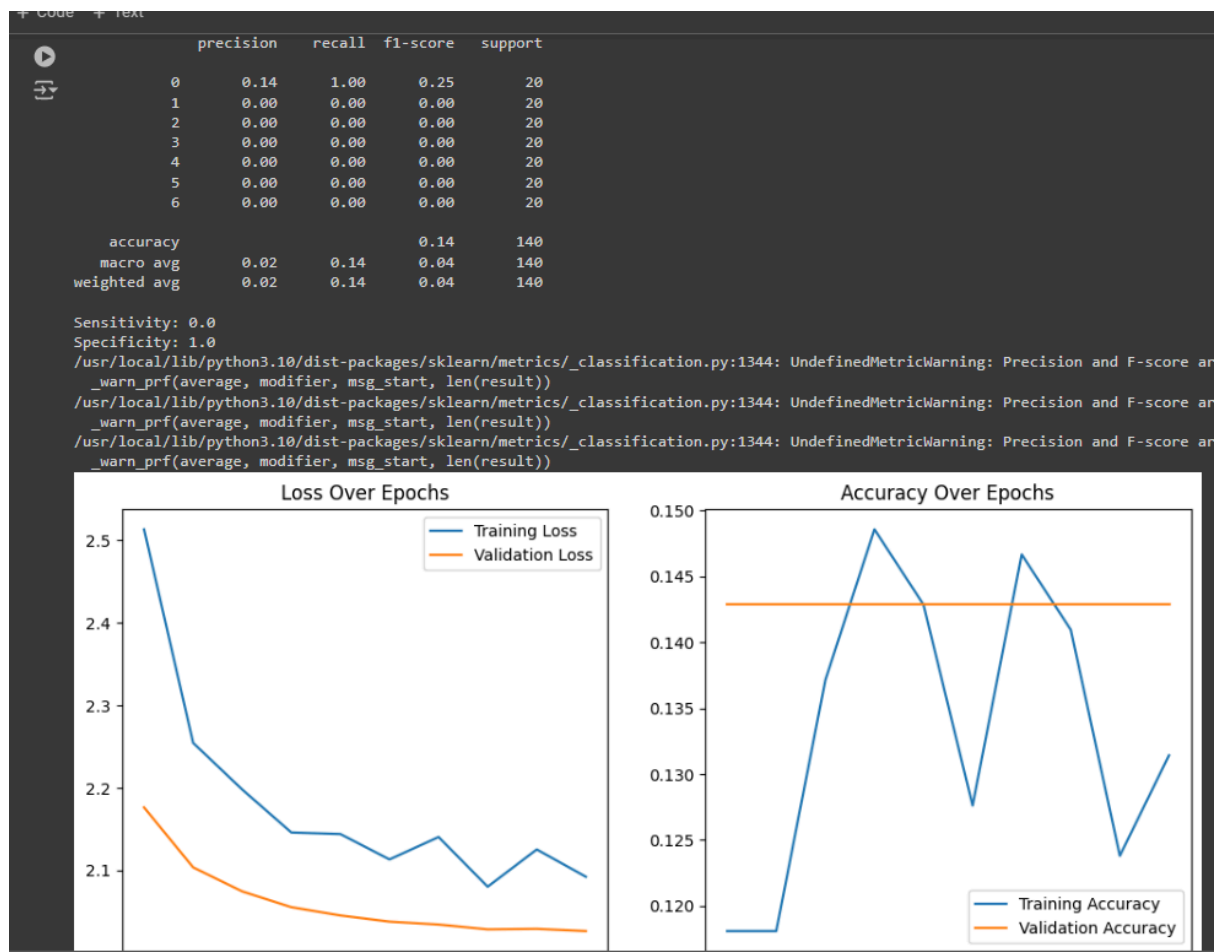
- **Base InceptionV3 Model:** InceptionV3 is a CNN architecture designed by Google. It's characterised by its unique inception modules, which allow for efficient use of computational resources by performing multiple convolutions at different scales within the same layer. The base InceptionV3 model is loaded with pre-trained weights from the ImageNet dataset. These weights have been learned from a large dataset of images and capture generic features like edges, textures, and shapes.
- **Custom Layers on Top:** On top of the base InceptionV3 model, additional layers are added to adapt it to the specific task of binary image classification.
- **Flatten Layer:** This layer is used to convert the 3D feature maps obtained from the convolutional layers of the base model into a 1D vector, which can be fed into the subsequent dense layers.
- **Dense Layers:** Two dense layers are added for feature extraction. The first dense layer has 1024 neurons and uses the ReLU activation function, allowing the model to learn complex patterns and relationships in the data. The second dense layer introduces a dropout regularization technique, which helps prevent overfitting by randomly deactivating some neurons during training.
- **Output Layer:** The final dense layer consists of a single neuron with a sigmoid activation function, suitable for binary classification tasks. It predicts the probability of an image belonging to the positive class (e.g., class 1).

Model Compilation:

- **Optimizer:** The RMSprop optimizer is chosen for training the model. It's an adaptive learning rate optimization algorithm that adjusts the learning rates of individual parameters based on their historical gradients. The learning rate is set to 0.0001, which controls the step size during optimization.
- **Loss Function:** Since this is a binary classification task (distinguishing between two classes), binary cross-entropy loss is used. It measures the difference between the predicted probabilities and the actual labels, penalising incorrect predictions and encouraging the model to output high probabilities for the correct class.
- **Metrics:** The model's performance is evaluated using accuracy, which measures the proportion of correctly classified images over the total number of images. It's a common metric for classification tasks and provides insight into the model's overall performance.

Reporting Results:

The results of the V3 model were not substantial. There are a number of reasons for this poor performance. Although the V3 inception model was trained on the imagenet dataset, the **data quantity** that we provided was very less. Deep learning models require large amounts of data to perform better. Secondly, we didn't **hyper tune** the parameters of the inception model. This can also serve as a reason for the poor performance of the model. Lastly, the model is doing **underfitting** since it has not been provided with an adequate amount of data due to computational limitations.



7. Overall Conclusion:

The Overall conclusion of the Project would be that we implemented 3 models namely MLP,CNN and VGG out of which MLP and CNN did not perform up to the mark. The Reason for this as we discussed above was that we could not provide the model enough data to learn proper patterns and classify accordingly. This was due to the computational limitations in which we had to work. Other than that we followed the proper procedures. We implemented pre-processing techniques e.g: Graysaciling of images and normalisation techniques. Furthermore, we Also checked for Class imbalancing which was not present in our case. After hyper-tuning the models we could not reach the potential the Models were capable of. But nonetheless our claim was proven to be true in the next phase when we implemented the VGG model. Had it not been for the computational limitations our models would have performed way better This was proven in our implementation of the VGG model. The VGG model is pretrained on the imagenet dataset. It has the ability to perform at a higher accuracy since it

already has learnt a lot of patterns. The VGG model provided us with a higher accuracy and other evaluation metrics compared to the other models.