Open Innovation in Data Science

Up2053685

Part 2: Image Classification Model in Computer Vision

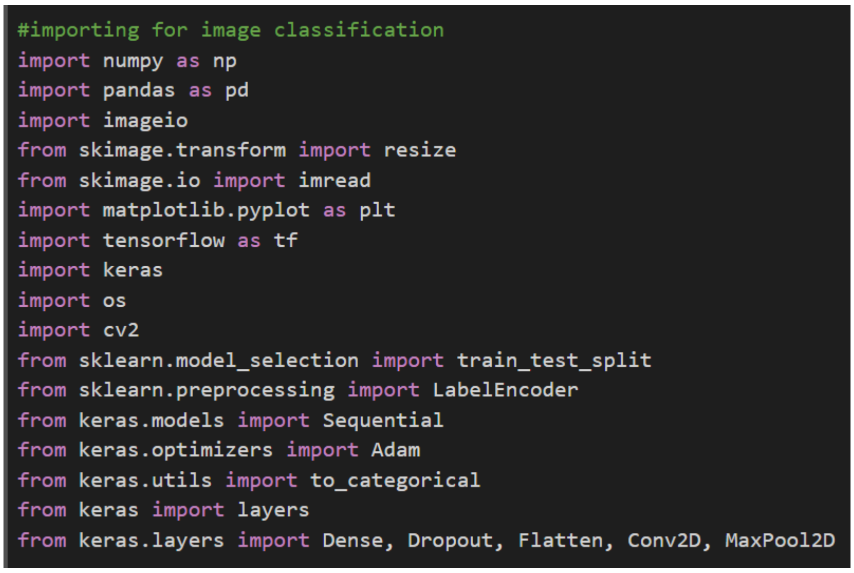
Word count = 1,098

2.1 Data Preprocessing

This is a computer vision dataset with two classes of images, vehicle and animal. This project will consist of the steps and process in classifying these two classes of images by building a 2D Convolutional neural network (CNN). The first step is to preprocess the datasets with Data Argumentation, splitting the dataset with a label encoder.

2.1.1 Data preparation

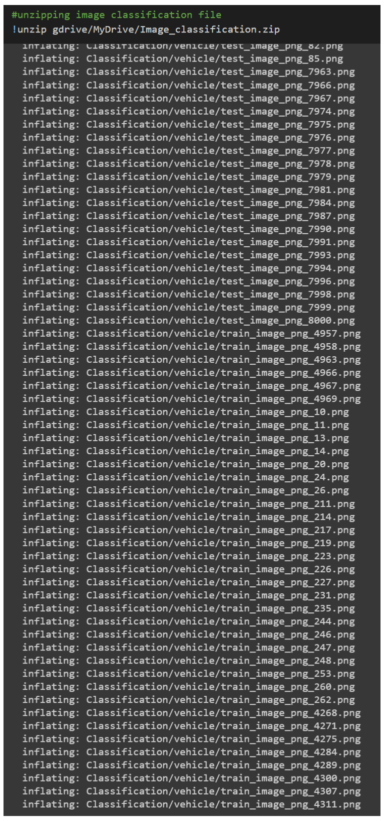
Firstly, all the necessary packages used for this project need to be imported into Google Collab as shown in Figure 1 below.



```python
#importing for image classification
import numpy as np
import pandas as pd
import imageio
from skimage.transform import resize
from skimage.io import imread
import matplotlib.pyplot as plt
import tensorflow as tf
import keras
import os
import cv2
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from keras.models import Sequential
from keras.optimizers import Adam
from keras.utils import to_categorical
from keras import layers
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
```

Figure 1- importing all necessary libraries.

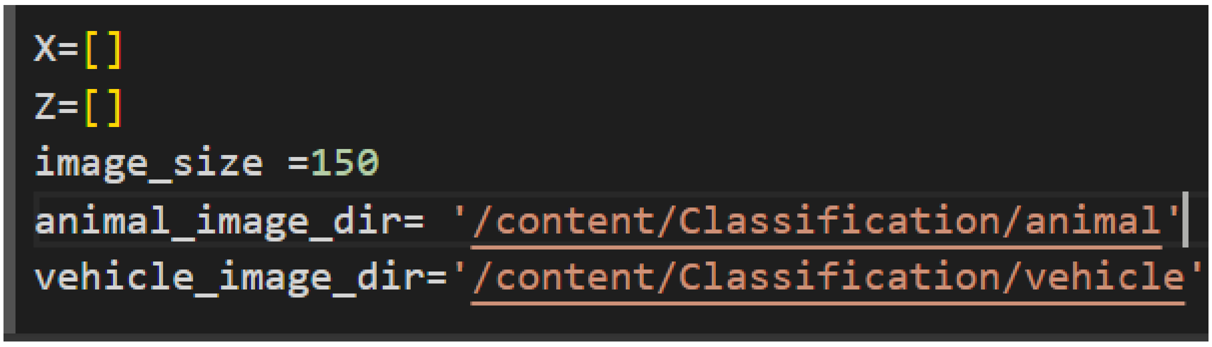
Then, the next step is to mount Google Drive and load up all the contents on the drive. ‘Imagie_classification.zip’ is our dataset and we need to unzip it as shown below in Figure 2.



```
#unzipping image classification file
!unzip gdrive/MyDrive/Image_classification.zip
  inflating: Classification/vehicle/test_image_png_82.png
  inflating: Classification/vehicle/test_image_png_85.png
  inflating: Classification/vehicle/test_image_png_7963.png
  inflating: Classification/vehicle/test_image_png_7966.png
  inflating: Classification/vehicle/test_image_png_7967.png
  inflating: Classification/vehicle/test_image_png_7974.png
  inflating: Classification/vehicle/test_image_png_7975.png
  inflating: Classification/vehicle/test_image_png_7976.png
  inflating: Classification/vehicle/test_image_png_7977.png
  inflating: Classification/vehicle/test_image_png_7978.png
  inflating: Classification/vehicle/test_image_png_7979.png
  inflating: Classification/vehicle/test_image_png_7981.png
  inflating: Classification/vehicle/test_image_png_7984.png
  inflating: Classification/vehicle/test_image_png_7987.png
  inflating: Classification/vehicle/test_image_png_7990.png
  inflating: Classification/vehicle/test_image_png_7991.png
  inflating: Classification/vehicle/test_image_png_7993.png
  inflating: Classification/vehicle/test_image_png_7994.png
  inflating: Classification/vehicle/test_image_png_7996.png
  inflating: Classification/vehicle/test_image_png_7998.png
  inflating: Classification/vehicle/test_image_png_7999.png
  inflating: Classification/vehicle/test_image_png_8000.png
  inflating: Classification/vehicle/train_image_png_4957.png
  inflating: Classification/vehicle/train_image_png_4958.png
  inflating: Classification/vehicle/train_image_png_4963.png
  inflating: Classification/vehicle/train_image_png_4966.png
  inflating: Classification/vehicle/train_image_png_4967.png
  inflating: Classification/vehicle/train_image_png_4969.png
  inflating: Classification/vehicle/train_image_png_10.png
  inflating: Classification/vehicle/train_image_png_11.png
  inflating: Classification/vehicle/train_image_png_13.png
  inflating: Classification/vehicle/train_image_png_14.png
  inflating: Classification/vehicle/train_image_png_20.png
  inflating: Classification/vehicle/train_image_png_24.png
  inflating: Classification/vehicle/train_image_png_26.png
  inflating: Classification/vehicle/train_image_png_211.png
  inflating: Classification/vehicle/train_image_png_214.png
  inflating: Classification/vehicle/train_image_png_217.png
  inflating: Classification/vehicle/train_image_png_219.png
  inflating: Classification/vehicle/train_image_png_223.png
  inflating: Classification/vehicle/train_image_png_226.png
  inflating: Classification/vehicle/train_image_png_227.png
  inflating: Classification/vehicle/train_image_png_231.png
  inflating: Classification/vehicle/train_image_png_235.png
  inflating: Classification/vehicle/train_image_png_244.png
  inflating: Classification/vehicle/train_image_png_246.png
  inflating: Classification/vehicle/train_image_png_247.png
  inflating: Classification/vehicle/train_image_png_248.png
  inflating: Classification/vehicle/train_image_png_253.png
  inflating: Classification/vehicle/train_image_png_260.png
  inflating: Classification/vehicle/train_image_png_262.png
  inflating: Classification/vehicle/train_image_png_4268.png
  inflating: Classification/vehicle/train_image_png_4271.png
  inflating: Classification/vehicle/train_image_png_4275.png
  inflating: Classification/vehicle/train_image_png_4284.png
  inflating: Classification/vehicle/train_image_png_4289.png
  inflating: Classification/vehicle/train_image_png_4300.png
  inflating: Classification/vehicle/train_image_png_4307.png
  inflating: Classification/vehicle/train_image_png_4311.png
```

Figure 2- Unzipping image classification ZIp file.

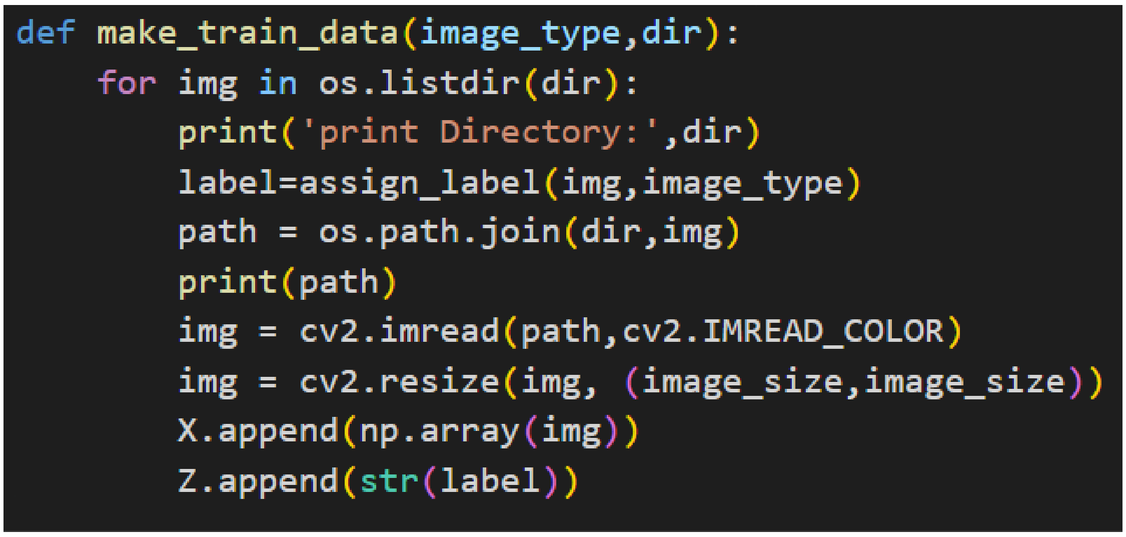
Then, once that is complete, the next step is to create an X and Y list variable to store images, then make the image size 150 and make two directories for each image class. Figure 3 shows both the animal directory and the vehicle directory.



```
X=[]
Z=[]
image_size =150
animal_image_dir= '/content/Classification/animal'
vehicle_image_dir='/content/Classification/vehicle'
```

Figure 3- code showing X and Y list with two classes directory.

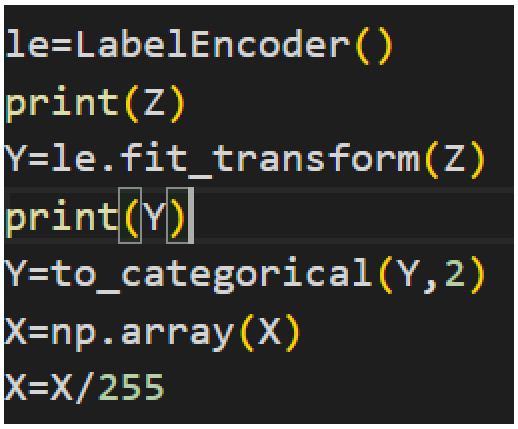
We assign each label to the image type. Then, we made a function called ‘make_train_data’. This has both the image type and the directory. Then a loop is created to process the image to the correct image type from the directory. This process is to ensure all images from a specific directory are assigned a label and stored in the correct list shown in Figure 4.



```
def make_train_data(image_type,dir):
    for img in os.listdir(dir):
        print('print Directory:',dir)
        label=assign_label(img,image_type)
        path = os.path.join(dir,img)
        print(path)
        img = cv2.imread(path,cv2.IMREAD_COLOR)
        img = cv2.resize(img, (image_size,image_size))
        X.append(np.array(img))
        Z.append(str(label))
```

Figure 4- showing the function with a loop for training data.

Lastly, the last step for preprocessing is to label encode to change to numerical features shown in Figure 5.



```
le=LabelEncoder()
print(Z)
Y=le.fit_transform(Z)
print(Y)
Y=to_categorical(Y,2)
X=np.array(X)
X=X/255
```

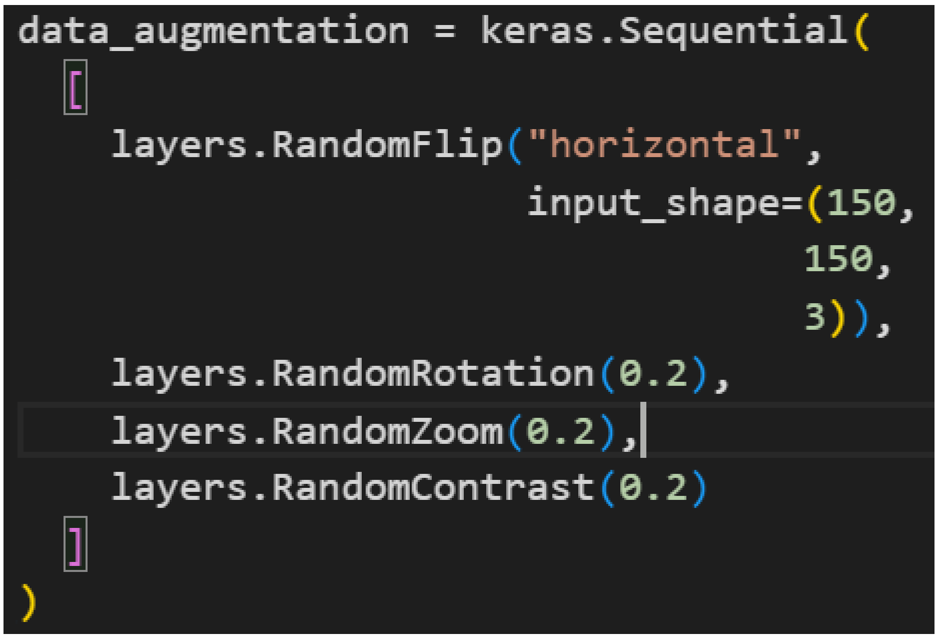
Figure 5- using Label encoder.

2.1.2 Splitting the Dataset

The next stage for data preprocessing is to split the dataset into training and testing. We used the train_test_split function to split data into 80% training and 20% testing.

2.1.3 Data Argumentation

We use data argumentation techniques to train the data appropriate to the model. We use keras with 4 layers to argumentate the images. Firstly, we use a random flip to horizontal with an input shape of (150,150,3). Then, we rotate the image by 0.2. The third layer will zoom the image by 0.2 and random contrast by 0.2. This is shown in Figure 6 below.



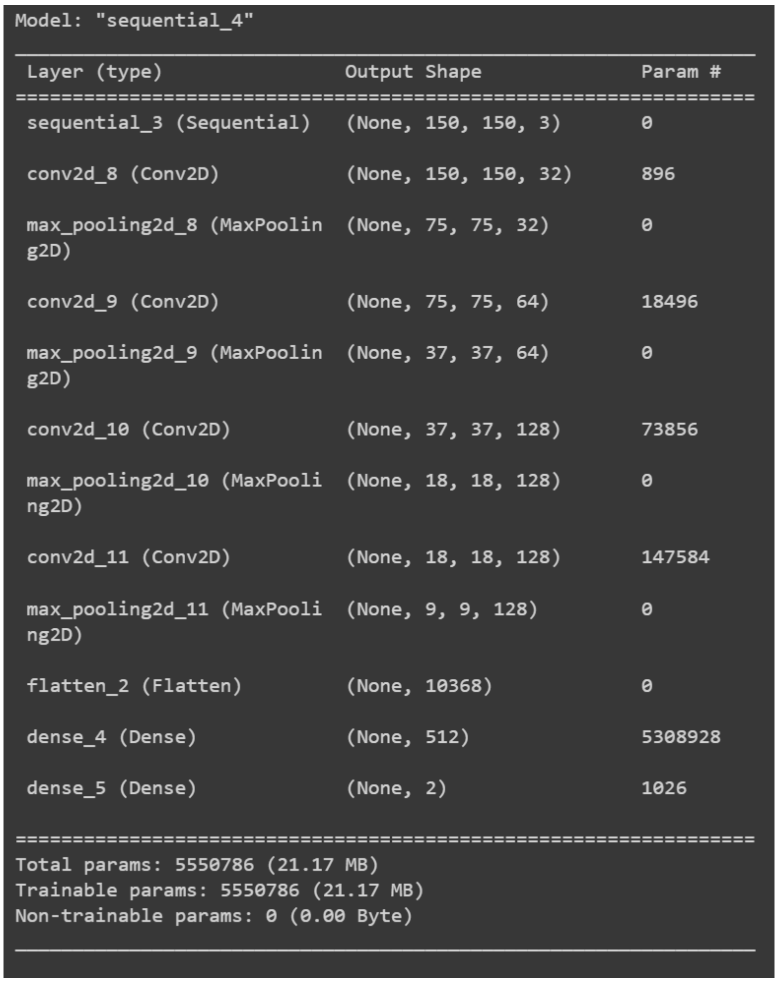
```
data_augmentation = keras.Sequential(
  [
    layers.RandomFlip("horizontal",
                      input_shape=(150,
                                   150,
                                   3)),
    layers.RandomRotation(0.2),
    layers.RandomZoom(0.2),
    layers.RandomContrast(0.2)
  ]
)
```

Figure 6- Data augmentation process.

2.2 Design a 2D Convolutional Neural Network

2.2.1 structure of the model

The structure of the model has a few convolutional layers with the uses of activation functions of ReLU and max pooling for feature extraction. The first layer has 32 neurons with a size of 3x3 with a Relu activation function. Then it is followed by a max pooling layer to reduce dimensions. The two steps are repeated with 64 filters instead followed by another convolution layer with 128 filters. The fourth layer is the same as the third layer. After the 4th layer, the 5th layer has a flatten layer to output all layers. The last two layers are dense layers with the first having 512 neurons with a ReLU activation function. The last layer has two neurons for the binary classification task and uses an activation function called sigmoid. The model summary is shown below in Figure 7.

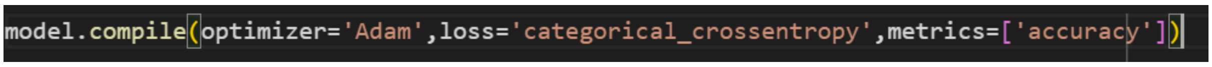


```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential_3 (Sequential)   (None, 150, 150, 3)       0

 conv2d_8 (Conv2D)           (None, 150, 150, 32)      896

 max_pooling2d_8 (MaxPoolin  (None, 75, 75, 32)        0
 g2D)

 conv2d_9 (Conv2D)           (None, 75, 75, 64)        18496

 max_pooling2d_9 (MaxPoolin  (None, 37, 37, 64)        0
 g2D)

 conv2d_10 (Conv2D)          (None, 37, 37, 128)       73856

 max_pooling2d_10 (MaxPooli  (None, 18, 18, 128)       0
 ng2D)

 conv2d_11 (Conv2D)          (None, 18, 18, 128)       147584

 max_pooling2d_11 (MaxPooli  (None, 9, 9, 128)         0
 ng2D)

 flatten_2 (Flatten)         (None, 10368)             0

 dense_4 (Dense)             (None, 512)               5308928

 dense_5 (Dense)             (None, 2)                 1026

=================================================================
Total params: 5550786 (21.17 MB)
Trainable params: 5550786 (21.17 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 7- Model Summary

2.2.2 Optimizer and Loss Function

For this model, the optimizer we use is Adam and the loss function is categorical cross-entropy. I've picked this because it's good for multi-classification models. This is shown in Figure 8.

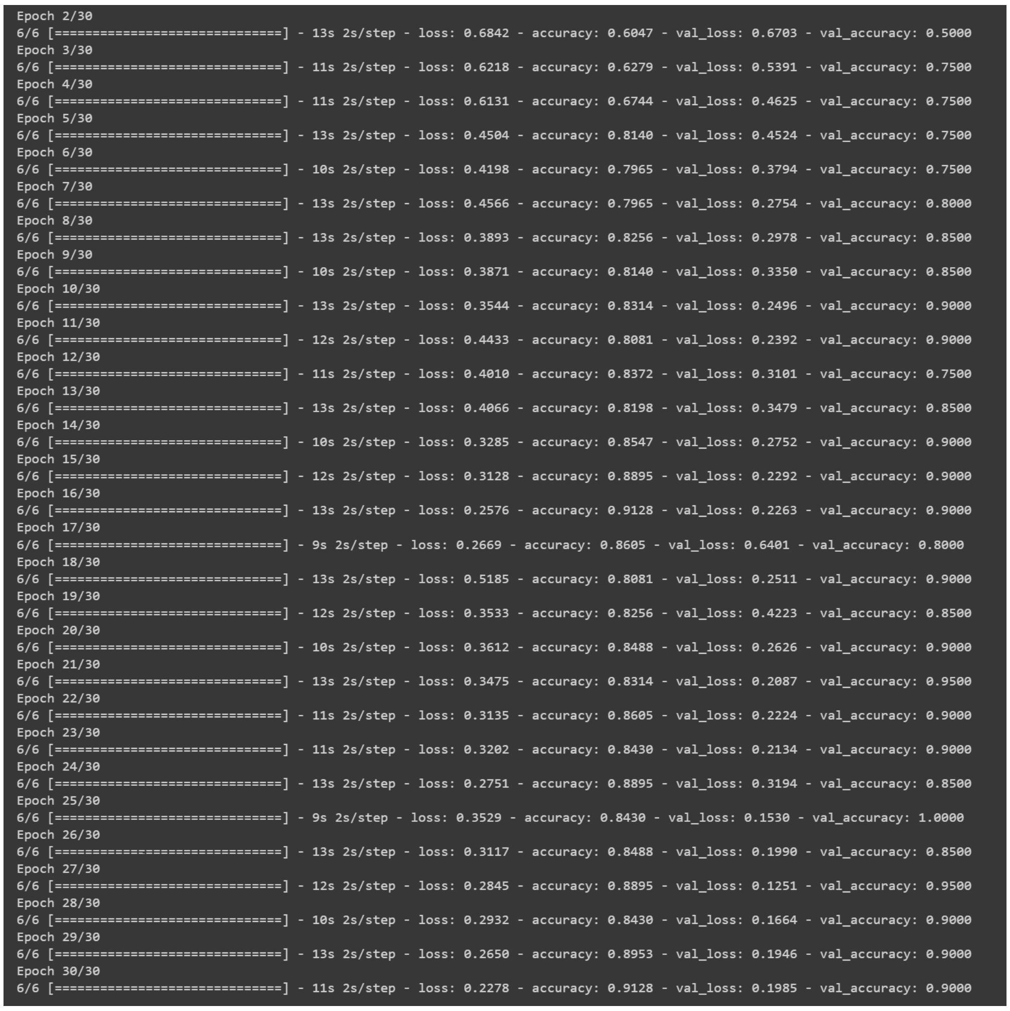


```
model.compile(optimizer='Adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

Figure 8- Adding optimizer and loss function

2.3 Training Loss and Accuracy

After testing this model, the accuracy rate is 91.3% with validation accuracy of 90%. The validation split is 10% with running 30 epochs. Figure 9 shows the output of the results.



```
Epoch 2/30
6/6 [==============================] - 13s 2s/step - loss: 0.6842 - accuracy: 0.6047 - val_loss: 0.6703 - val_accuracy: 0.5000
Epoch 3/30
6/6 [==============================] - 11s 2s/step - loss: 0.6218 - accuracy: 0.6279 - val_loss: 0.5391 - val_accuracy: 0.7500
Epoch 4/30
6/6 [==============================] - 11s 2s/step - loss: 0.6131 - accuracy: 0.6744 - val_loss: 0.4625 - val_accuracy: 0.7500
Epoch 5/30
6/6 [==============================] - 13s 2s/step - loss: 0.4504 - accuracy: 0.8140 - val_loss: 0.4524 - val_accuracy: 0.7500
Epoch 6/30
6/6 [==============================] - 10s 2s/step - loss: 0.4198 - accuracy: 0.7965 - val_loss: 0.3794 - val_accuracy: 0.7500
Epoch 7/30
6/6 [==============================] - 13s 2s/step - loss: 0.4566 - accuracy: 0.7965 - val_loss: 0.2754 - val_accuracy: 0.8000
Epoch 8/30
6/6 [==============================] - 13s 2s/step - loss: 0.3893 - accuracy: 0.8256 - val_loss: 0.2978 - val_accuracy: 0.8500
Epoch 9/30
6/6 [==============================] - 10s 2s/step - loss: 0.3871 - accuracy: 0.8140 - val_loss: 0.3350 - val_accuracy: 0.8500
Epoch 10/30
6/6 [==============================] - 13s 2s/step - loss: 0.3544 - accuracy: 0.8314 - val_loss: 0.2496 - val_accuracy: 0.9000
Epoch 11/30
6/6 [==============================] - 12s 2s/step - loss: 0.4433 - accuracy: 0.8081 - val_loss: 0.2392 - val_accuracy: 0.9000
Epoch 12/30
6/6 [==============================] - 11s 2s/step - loss: 0.4010 - accuracy: 0.8372 - val_loss: 0.3101 - val_accuracy: 0.7500
Epoch 13/30
6/6 [==============================] - 13s 2s/step - loss: 0.4066 - accuracy: 0.8198 - val_loss: 0.3479 - val_accuracy: 0.8500
Epoch 14/30
6/6 [==============================] - 10s 2s/step - loss: 0.3285 - accuracy: 0.8547 - val_loss: 0.2752 - val_accuracy: 0.9000
Epoch 15/30
6/6 [==============================] - 12s 2s/step - loss: 0.3128 - accuracy: 0.8895 - val_loss: 0.2292 - val_accuracy: 0.9000
Epoch 16/30
6/6 [==============================] - 13s 2s/step - loss: 0.2576 - accuracy: 0.9128 - val_loss: 0.2263 - val_accuracy: 0.9000
Epoch 17/30
6/6 [==============================] - 9s 2s/step - loss: 0.2669 - accuracy: 0.8605 - val_loss: 0.6401 - val_accuracy: 0.8000
Epoch 18/30
6/6 [==============================] - 13s 2s/step - loss: 0.5185 - accuracy: 0.8081 - val_loss: 0.2511 - val_accuracy: 0.9000
Epoch 19/30
6/6 [==============================] - 12s 2s/step - loss: 0.3533 - accuracy: 0.8256 - val_loss: 0.4223 - val_accuracy: 0.8500
Epoch 20/30
6/6 [==============================] - 10s 2s/step - loss: 0.3612 - accuracy: 0.8488 - val_loss: 0.2626 - val_accuracy: 0.9000
Epoch 21/30
6/6 [==============================] - 13s 2s/step - loss: 0.3475 - accuracy: 0.8314 - val_loss: 0.2087 - val_accuracy: 0.9500
Epoch 22/30
6/6 [==============================] - 11s 2s/step - loss: 0.3135 - accuracy: 0.8605 - val_loss: 0.2224 - val_accuracy: 0.9000
Epoch 23/30
6/6 [==============================] - 11s 2s/step - loss: 0.3202 - accuracy: 0.8430 - val_loss: 0.2134 - val_accuracy: 0.9000
Epoch 24/30
6/6 [==============================] - 13s 2s/step - loss: 0.2751 - accuracy: 0.8895 - val_loss: 0.3194 - val_accuracy: 0.8500
Epoch 25/30
6/6 [==============================] - 9s 2s/step - loss: 0.3529 - accuracy: 0.8430 - val_loss: 0.1530 - val_accuracy: 1.0000
Epoch 26/30
6/6 [==============================] - 13s 2s/step - loss: 0.3117 - accuracy: 0.8488 - val_loss: 0.1990 - val_accuracy: 0.8500
Epoch 27/30
6/6 [==============================] - 12s 2s/step - loss: 0.2845 - accuracy: 0.8895 - val_loss: 0.1251 - val_accuracy: 0.9500
Epoch 28/30
6/6 [==============================] - 10s 2s/step - loss: 0.2932 - accuracy: 0.8430 - val_loss: 0.1664 - val_accuracy: 0.9000
Epoch 29/30
6/6 [==============================] - 13s 2s/step - loss: 0.2650 - accuracy: 0.8953 - val_loss: 0.1946 - val_accuracy: 0.9000
Epoch 30/30
6/6 [==============================] - 11s 2s/step - loss: 0.2278 - accuracy: 0.9128 - val_loss: 0.1985 - val_accuracy: 0.9000
```

<u>Figure 9- output of each epoch with Loss, Accuracy, Validation Loss and Validation Accuracy.</u>
]

2.3.1 Training Loss

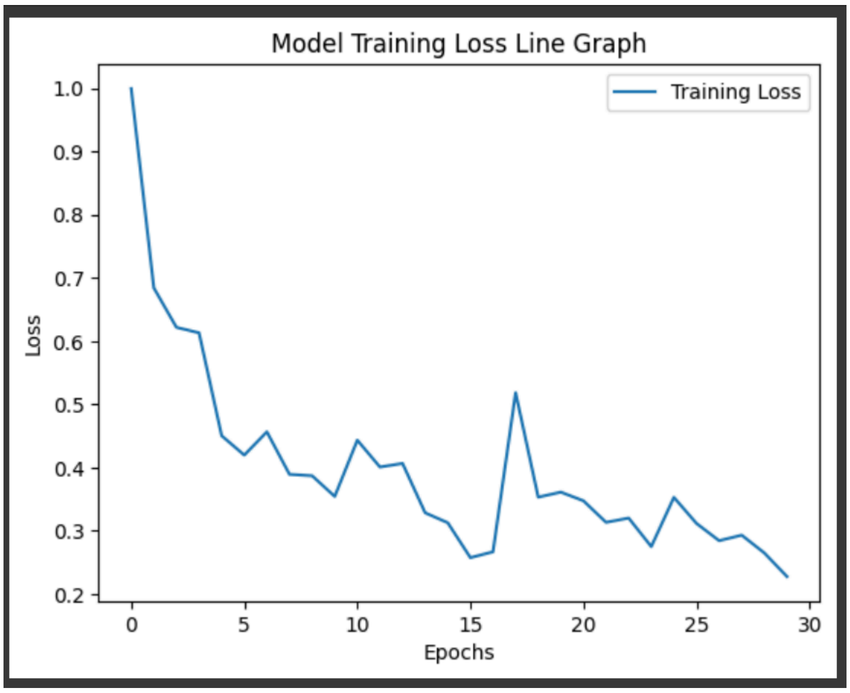


Figure 10- Line graph showing Training Loss

This graph shows a consistent training loss from the model. Even though the graph fluctuates, there is still a steady decrease from approximately 1 down to 0.25. The 17th epoch shows an anomaly spike to approximately 0.5.

2.3.2 Model Accuracy

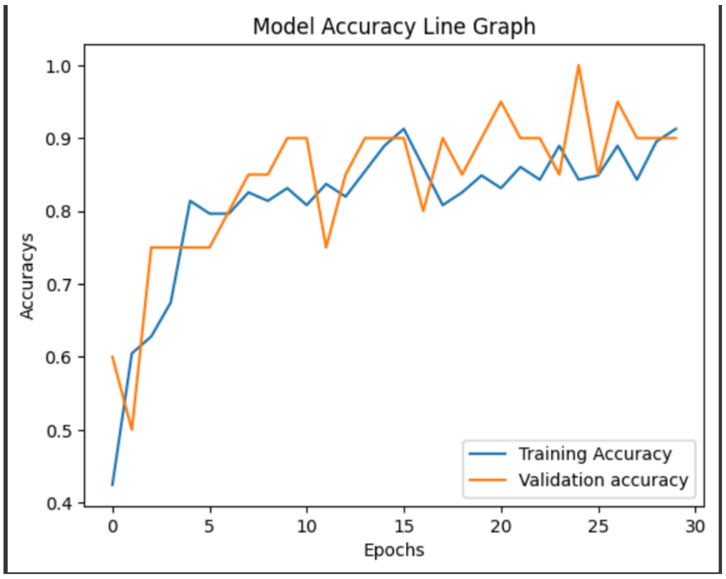


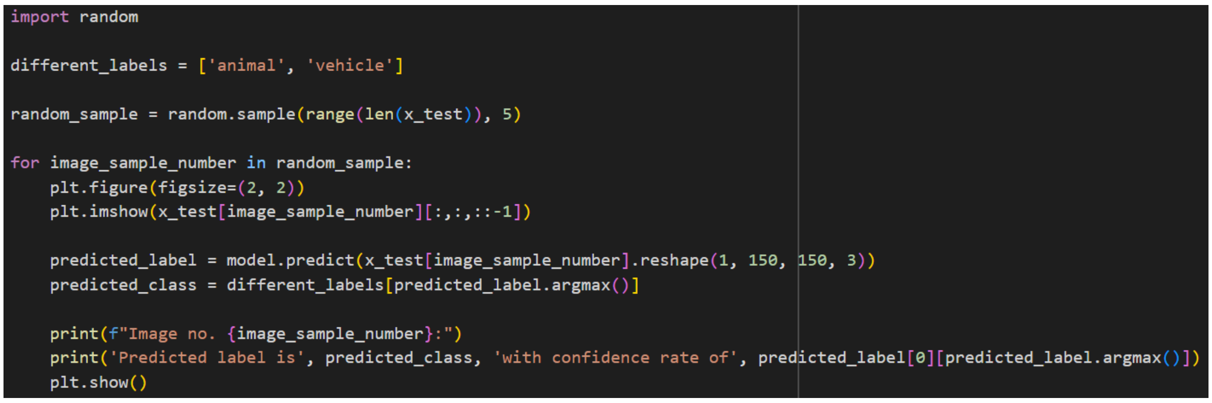
Figure 11- Line graph showing Model Accuracy Loss for Training Accuracy and Validation Accuracy

This graph shows a similar increase in both lines. The first epoch accuracy started at approximately 60% and then gradually increased to 91%. This shows a consistency in the model and lower signs of overfitting for class balance. This shows a high accuracy rate for the initial model.

2.4 Testing samples

We picked 5 samples and the model needs to predict the label.

2.4.1 Methodology

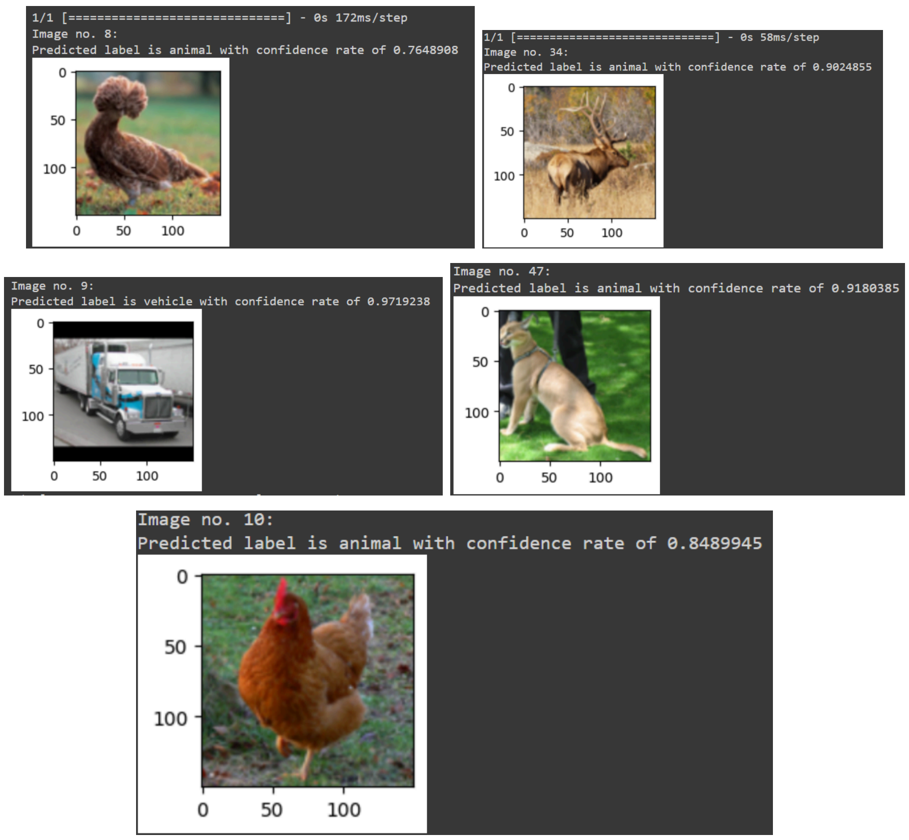


```python
import random

different_labels = ['animal', 'vehicle']

random_sample = random.sample(range(len(x_test)), 5)

for image_sample_number in random_sample:
    plt.figure(figsize=(2, 2))
    plt.imshow(x_test[image_sample_number][:,:,::-1])

    predicted_label = model.predict(x_test[image_sample_number].reshape(1, 150, 150, 3))
    predicted_class = different_labels[predicted_label.argmax()]

    print(f"Image no. {image_sample_number}:")
    print('Predicted label is', predicted_class, 'with confidence rate of', predicted_label[0][predicted_label.argmax()])
    plt.show()
```

Figure 12- Code showing the methodology of test samples

Figure 12 shows the code to generate our sample testing. Firstly, we need to import random numbers to randomly pick an image sample number. Our two labels for our classes are animal and vehicle. We use random samples in the range 1 to 5 to pick 5 distinct image sample numbers. Then, we created a loop for each sample selected to change the size. Then display the image by displaying it by reversing the RGB image. Then the code will have a predicted label and predicted class. Then we print the image number and its predicted label with a confidence rate percentage shown in Figure 13.

2.4.2 Results



```
1/1 [==============================] - 0s 172ms/step
Image no. 8:
Predicted label is animal with confidence rate of 0.7648908
```

```
1/1 [==============================] - 0s 58ms/step
Image no. 34:
Predicted label is animal with confidence rate of 0.9024855
```

```
Image no. 9:
Predicted label is vehicle with confidence rate of 0.9719238
```

```
Image no. 47:
Predicted label is animal with confidence rate of 0.9180385
```

```
Image no. 10:
Predicted label is animal with confidence rate of 0.8489945
```

Figure 13- shows the results of all 5 samples with the image number and predicted labels with confidence rate.

2.4.3 Evaluation

To conclude the testing, the majority of the test samples have an accuracy rate above 90%. Even though all the samples are correct. This shows there might be a class imbalance as more images are of animals over vehicles.

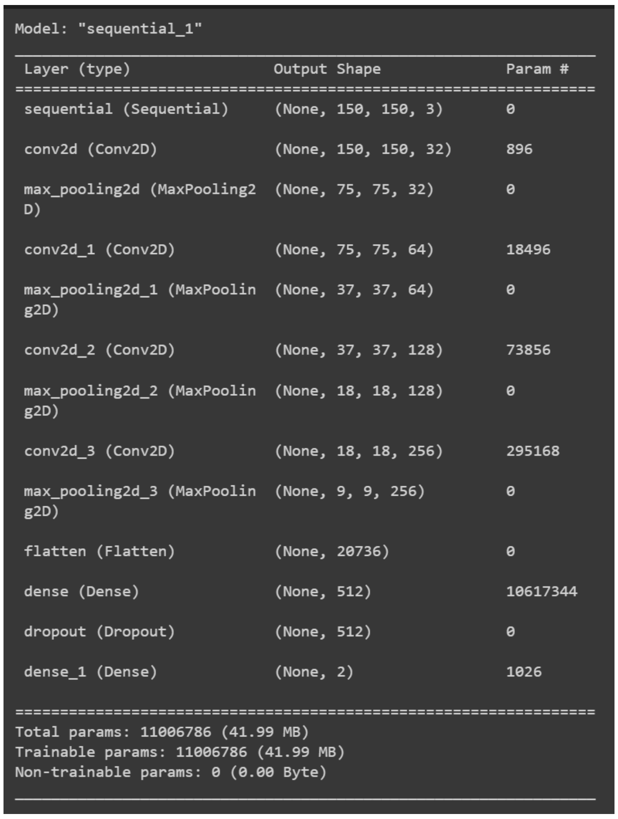
2.5 Model Improvements

Even though we have obtained a high accuracy. There are still ways to improve our model with two different methods.

2.5.1 Model Improvement 1- using Dropout

2.5.1.1 Methodology

This model is the same as the previous one but we have added a dropout with a size of 0.5 because it can reduce overfitting and improve the quality of the model. Figure 14 shows the new model summary.



```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 sequential (Sequential)     (None, 150, 150, 3)       0         
                                                                 
 conv2d (Conv2D)             (None, 150, 150, 32)      896       
                                                                 
 max_pooling2d (MaxPooling2  (None, 75, 75, 32)        0         
 D)                                                              
                                                                 
 conv2d_1 (Conv2D)           (None, 75, 75, 64)        18496     
                                                                 
 max_pooling2d_1 (MaxPoolin  (None, 37, 37, 64)        0         
 g2D)                                                            
                                                                 
 conv2d_2 (Conv2D)           (None, 37, 37, 128)       73856     
                                                                 
 max_pooling2d_2 (MaxPoolin  (None, 18, 18, 128)       0         
 g2D)                                                            
                                                                 
 conv2d_3 (Conv2D)           (None, 18, 18, 256)       295168    
                                                                 
 max_pooling2d_3 (MaxPoolin  (None, 9, 9, 256)         0         
 g2D)                                                            
                                                                 
 flatten (Flatten)           (None, 20736)             0         
                                                                 
 dense (Dense)               (None, 512)               10617344  
                                                                 
 dropout (Dropout)           (None, 512)               0         
                                                                 
 dense_1 (Dense)             (None, 2)                 1026      
                                                                 
=================================================================
Total params: 11006786 (41.99 MB)
Trainable params: 11006786 (41.99 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 14- Model Summary of Improved Model

We can see I have added a dropout after the dense layer.

2.5.1.2 Results and Evaluation

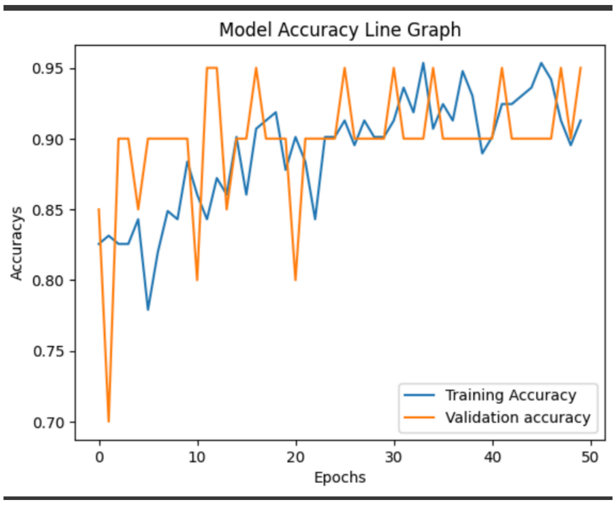


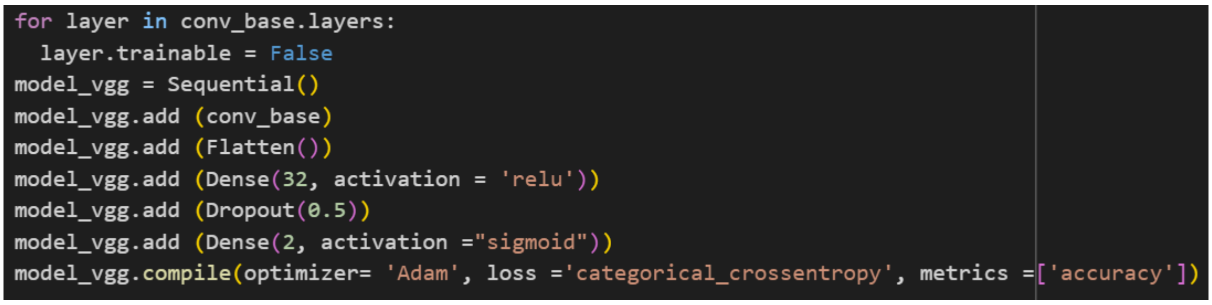
Figure 15- Line graph of Training Accuracy and Validation Accuracy

This graph shows that the highest accuracy has hit 95% in accuracy which shows an increase of approximately 4%. The last epoch is the same as the previous model, however, our validation accuracy has increased by 5%. This shows an improvement.
However, there is a limitation on this graph where there is no smooth curve. This shows that there might be an issue with overfitting as training accuracy increases, and validation decreases. Therefore, the congruence and divergence show a class imbalance.

2.5.2 Model Improvement 2- using the VGG16 model

2.5.2.1 Methodology

We use the VGG16 model to improve accuracy. This consists of importing VGG16 from the Keras application. Then we built the model as shown below in Figure 16.



```
for layer in conv_base.layers:
  layer.trainable = False
model_vgg = Sequential()
model_vgg.add (conv_base)
model_vgg.add (Flatten())
model_vgg.add (Dense(32, activation = 'relu'))
model_vgg.add (Dropout(0.5))
model_vgg.add (Dense(2, activation ="sigmoid"))
model_vgg.compile(optimizer= 'Adam', loss ='categorical_crossentropy', metrics =['accuracy'])
```

Figure 16- Code showing the structure of the VGG 16 model

First, we added a convolutional base layer and then a flatten layer after. Then, we added a dense layer with 32 neurons with a Relu activation function. The next layer is a dropout with a 0.5 rate. Lastly, we added another Dense layer with 2 neurons as it was a binary classification task. The activation we've used was Sigmoid.
Then for the optimizer and loss function, we've used Adam and categorical cross entropy.

2.5.2.2 Results and Evaluation

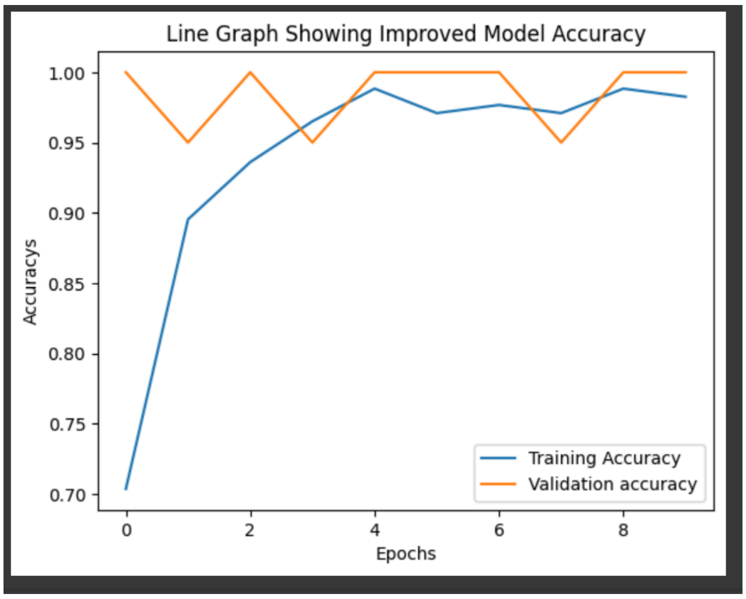


Figure 17- Line graph that shows improved model Training Accuracy and validation Accuracy

This graph shows a more consistent increase in its training accuracy reaching 98.3% this shows a great improvement compared to the original method by 7%. This shows less of a class imbalance.
However, the validation accuracy fluctuates. This is because there are still signs of class imbalance. Therefore, populating the dataset can fix this.