DEEP LEARNING

(CS6005)

MINI PROJECT

On: Convolutional Neural Networks (CNN)

IMAGE CLASSIFICATION

(with CIFAR-10 dataset)

- Aparna S S
2018103008
CSE – ‘P’ batch

Date- 20/04/2021

Problem Statement:

Traditional neural networks though have achieved appreciable performance at image classification, they have been characterized by feature engineering, a tedious process that results in poor generalization to test data. Moreover, when the images are taken in varying lighting conditions and at different angles, and since these are colored images, you will see that there are many variations in the color itself of similar objects (for example, the colour of ocean water). If we use the simple CNN architecture that we use in the basic datasets such as MNIST, we will get a low validation accuracy of around 60% which is never desirable. Hence, to improve the prediction metrics desirable for higher datasets such as CIFAR-10, I present a convolutional neural network (CNN) approach which works to achieve improved performances without feature engineering. Learnable filters and pooling layers are used to extract underlying image features. Dropout, regularization along with variation in convolution strategies are applied to reduce overfitting while ensuring increased accuracies in validation and testing. Better test accuracy with reduced overfitting is achieved with a deeper network. The predictions and testing are done using Graphical User Interface (GUI) for a better field of vision and also, when predicted through console the code has to be repeatedly executed for the desired images whereas in this case it's just a matter of a mouse click and also saves the run time.

Dataset details:

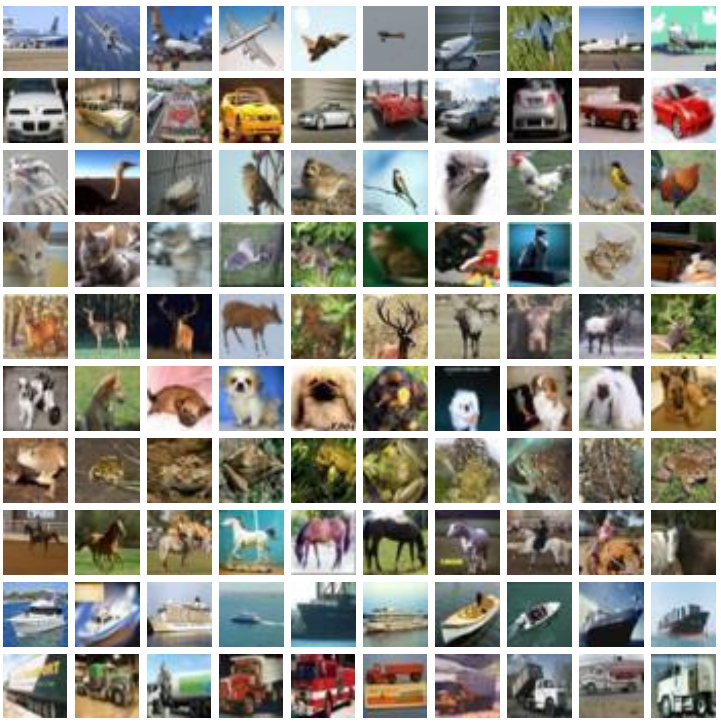
The dataset used for image classification is the **CIFAR-10** dataset.

The CIFAR-10 dataset contains 60,000 32x32 color images in 10 different classes. The 10 different classes represent airplanes, automobile, birds, cats, deer, dogs, frogs, horses, ships, and trucks. The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

CIFAR-10 is a labelled subset of the 80 million tiny images dataset.

Here are the classes in the dataset, as well as 10 random images from each:

airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck



The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.

url- https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz

Modules:

1. Loading the dataset:

- ✓ The dataset used is the CIFAR-10 dataset which is already available in the **dataset** module of **Keras**. The dataset is loaded using the **dataset.load_data()** function to the train and test parts being the x_train, y_train and x_test, y_test.

- ✓ The first few train images i.e from x_train are plotted using the **pyplot** package in the **matplotlib** library. These images are subplotted to describe the layout of the figure. The layout is organized in rows and columns, which are represented by the first and second argument. The third argument represents the index of the current plot. Here, the rows is 330, column is 1 and the index of the current plot is i(which is a loop variable used for displaying the number of images(which is 9)). This is done using **subplot** function. The plotted images are then displayed using the **imshow** function.

2. Data augmentation:

- ✓ As the model learns well when the number of images is increased, the images are augmented. This augmentation is done using the **ImageDataGenerator** class in the **Keras** module which generates batches of tensor image data with real-time data augmentation. The data will be looped over (in batches) indefinitely. The image data is generated by transforming the actual training images by rotation, crop, shifts, shear, zoom, flip, reflection, normalization etc. These augmented images are hence stored in the **datagen** variable.

- ✓ The initialised **ImageDataGenerator** class is batched using the **flow(x, y,batch_size)** method that takes numpy data & label arrays, and generates batches of augmented/normalized

data. Here, the batch size is 9, and hence 9 images will be displayed.

3. Normalization:

- ✓ As the features can be of different data types, all are converted into float data type using the **astype(data_type)** function in the **pandas** pakage. This process is done for both train and test of x split i.e for x_train and x_test.

- ✓ In order to measure how many standard deviations below or above the population mean a value is, calculate the z-score by using the formula:

$$z = (x - \mu) / \sigma$$

where, **x** is the value (x split), **μ** is the mean and **σ** is the standard deviation.

- ✓ As the inputs are vectors, they are converted into numpy array using **to_categorical(vector, num_classes)** function in the **numpy utility** library in the **Keras** module which converts a vector which has integers that represent different categories to a numpy array (or) a matrix which has binary values and has columns equal to the number of categories in the data. Here, the **number of classes is 10** (as there are 10 labels in the dataset).

4. CNN Model:

✓ Building the model-

A shallow architecture achieves only 76% accuracy. So, the idea here is to build a deep neural architecture as opposed to shallow architecture which was not able to learn features of objects accurately. The advantage of multiple layers is that they can learn features at various levels of abstraction. Multiple layers are much better at generalizing because they learn all the intermediate features between the raw input and the high-level classification. The process of building a Convolutional Neural Network majorly involves four major blocks show below.

Convolution layer=>Pooling layer=>Flattening layer=>Output layer

Here, we will build a 8 layered convolution neural network followed by flatten layer. In these 8 layers, 2 layers have the **number of filters** as **32** and the **kernel size** as **(3x3)**, 2 layers have the **number of filters** as **64** and the **kernel size** as **(3x3)** with **padding** as **valid** i.e. leaves the input image as it is and 2 layers have the **number of filters** as **128** and the **kernel size** as **(3x3)** and 2 layers have the **number of filters** as **256** and the **kernel size** as **(3x3)** with **padding** as **same** i.e. adds padding to the input image and the **activation function** as r**elu** in all the layers. After this, the **max pooling function** is done with a **pool_size** of **(2x2)**. The output layer is **dense layer** of **10 nodes** (as there are 10 classes) with **softmax activation**. In addition, these layers consist of the regularization functions:

- **Dropout** randomly drop units (along with their connections) from the neural network during training. The reduction in number of parameters in each step of training has effect of regularization. This is added after the max pooling of the layer. Here, we are using the values as **0.1 0.2 0.4 0.4** after each max pooling respectively.
- **Kernel_regularizer** allows to apply penalties on layer parameters during optimization. These penalties are incorporated in the loss function that the network optimizes. This argument in convolutional layer is nothing

but L2 regularisation of the weights. This penalizes peaky weights and makes sure that all the inputs are considered. During gradient descent parameter update, the above L2 regularization ultimately means that every weight is decayed linearly, that's why called weight decay. Here, the **weight decay is 1e-3 which is 0.001**.

- **BatchNormalization** normalizes the activation of the previous layer at each batch, i.e. applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1. It addresses the problem of internal covariate shift. Batch Normalization achieves the same accuracy with fewer training steps thus speeding up the training process.

✓ **Training the model-**

A learning rate schedule class is defined which changes the learning rate based on the number of epochs. The learning rate is set to 0.001 initially which is modified to 0.005 and 0.003 when the number of epochs is greater than 10 and 20 respectively. This is done in order to best fit the model as a very low lr can make the model underfit and high lr can make the model overfit. By changing the learning rates, the model learns variantly and hence is best fitted. The total number of **epochs** is **25.** The **batch size** is **256** i.e. the dataset is divided into 256 batches for each epoch.

The optimizers shape and mould your model into its most accurate possible form by futzing with the weights. The **optimizer** used is the **Adam** with a **decay** of **1e-6 i.e. 0.000001** which is a combination of adaptive gradient boosting and RMSProp.

The model is fit and complied with the above-mentioned parameters.

- ✓ **Performance metrics-**

To check how the model works for the test split i.e. x_test and y_test the metrics are calculated. The **metric** used here is **Accuracy**. The obtained accuracy of the model after 25 epochs is 68.5.

5. Prediction:

- ✓ **Prediction and display using console-**

A result dictionary is made to map the output classes and make predictions from the model. The image which has to be classified is opened by the **Image.open(path)** function in the **PIL** library.
The image is resized to **(32x32)** as the images in the dataset are of that dimension and the image has to be **RGB** image.
The image is converted to array and then the label or class of the image is predicted and displayed.

- ✓ **Predicting using GUI-**

The model is saved and loaded using **load_model** function in the **Keras** module. Add the label dictionary with all the 10 classes.
Initialize the GUI using **Tkinker** and use the pre-defined GUI code for developing the GUI model.
In the GUI prediction window upload the image which has to be classified by clicking the **upload image** button. On clicking the **classify image** button, the classification of the image is displayed.

CNN Model summary:

The model used is **Sequential model** and the model summary is:

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_14 (Conv2D)           (None, 30, 30, 32)        896
_________________________________________________________________
activation_14 (Activation)   (None, 30, 30, 32)        0
_________________________________________________________________
batch_normalization_14 (Batc (None, 30, 30, 32)        128
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 28, 28, 32)        9248
_________________________________________________________________
activation_15 (Activation)   (None, 28, 28, 32)        0
_________________________________________________________________
batch_normalization_15 (Batc (None, 28, 28, 32)        128
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
dropout_7 (Dropout)          (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 12, 12, 64)        18496
_________________________________________________________________
activation_16 (Activation)   (None, 12, 12, 64)        0
_________________________________________________________________
batch_normalization_16 (Batc (None, 12, 12, 64)        256
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 10, 10, 64)        36928
_________________________________________________________________
activation_17 (Activation)   (None, 10, 10, 64)        0
_________________________________________________________________
batch_normalization_17 (Batc (None, 10, 10, 64)        256
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
dropout_8 (Dropout)          (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 5, 5, 128)         73856
_________________________________________________________________
```

activation_18 (Activation)	(None, 5, 5, 128)	0
batch_normalization_18 (Batc	(None, 5, 5, 128)	512
conv2d_19 (Conv2D)	(None, 5, 5, 128)	147584
activation_19 (Activation)	(None, 5, 5, 128)	0
batch_normalization_19 (Batc	(None, 5, 5, 128)	512
max_pooling2d_9 (MaxPooling2	(None, 2, 2, 128)	0
dropout_9 (Dropout)	(None, 2, 2, 128)	0
conv2d_20 (Conv2D)	(None, 2, 2, 256)	295168
activation_20 (Activation)	(None, 2, 2, 256)	0
batch_normalization_20 (Batc	(None, 2, 2, 256)	1024
conv2d_21 (Conv2D)	(None, 2, 2, 256)	590080
activation_21 (Activation)	(None, 2, 2, 256)	0
batch_normalization_21 (Batc	(None, 2, 2, 256)	1024
max_pooling2d_10 (MaxPooling	(None, 1, 1, 256)	0
dropout_10 (Dropout)	(None, 1, 1, 256)	0
flatten_2 (Flatten)	(None, 256)	0
dense_2 (Dense)	(None, 10)	2570

Total params: 1,178,666
Trainable params: 1,176,746
Non-trainable params: 1,920

Coding Snapshots:

IMAGE CLASSIFICATION ON CIDAR-10 DATASET

Module-1

Loading dataset

```
import keras
from keras.models import Sequential
from keras.utils import np_utils
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Dense, Activation, Flatten, Dropout, BatchNormalization
from keras.layers import Conv2D, MaxPooling2D
from keras.datasets import cifar10
from keras import regularizers
from keras.callbacks import LearningRateScheduler
import numpy as np
```

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
from matplotlib import pyplot
# plot first few images
for i in range(9):
    # define subplot
    pyplot.subplot(330 + 1 + i)
    # plot raw pixel data
    pyplot.imshow(x_train[i])
# show the figure
pyplot.show()
```

Module-2

Data augmentation

```
#data augmentation
datagen = ImageDataGenerator(
    zoom_range = 0.5,
    vertical_flip=True,
    rotation_range=50,
    width_shift_range=0,
    height_shift_range=0,
    horizontal_flip=False,
    )
datagen.fit(x_train)
```

Module-3

Normalization

```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
```

```
#z-score
mean = np.mean(x_train,axis=(0,1,2,3))
std = np.std(x_train,axis=(0,1,2,3))
x_train = (x_train-mean)/(std+1e-7)
x_test = (x_test-mean)/(std+1e-7)

num_classes = 10
y_train = np_utils.to_categorical(y_train,num_classes)
y_test = np_utils.to_categorical(y_test1,num_classes)
```

Module-4

CNN model

1. Building model

```
def lr_schedule(epoch):
    lrate = 0.001
    if epoch > 15:
        lrate = 0.0005
    if epoch > 20:
        lrate = 0.0003
    return lrate
```

```
weight_decay = 1e-3
model = Sequential()
model.add(Conv2D(32, (3,3), padding='valid', kernel_regularizer=regularizers.l2(weight_decay), input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(32, (3,3), padding='valid', kernel_regularizer=regularizers.l2(weight_decay)))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.1))

model.add(Conv2D(64, (3,3), padding='valid', kernel_regularizer=regularizers.l2(weight_decay)))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, (3,3), padding='valid', kernel_regularizer=regularizers.l2(weight_decay)))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
```

```
model.add(Dropout(0.2))

model.add(Conv2D(128, (3,3), padding='same', kernel_regularizer=regularizers.l2(weight_decay)))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(128, (3,3), padding='same', kernel_regularizer=regularizers.l2(weight_decay)))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.4))

model.add(Conv2D(256, (3,3), padding='same', kernel_regularizer=regularizers.l2(weight_decay)))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(Conv2D(256, (3,3), padding='same', kernel_regularizer=regularizers.l2(weight_decay)))
model.add(Activation('relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.4))

model.add(Flatten())
model.add(Dense(num_classes, activation='softmax'))

model.summary()
```

2. Training the model

```
batch_size = 256


adam = keras.optimizers.Adam(lr=0.001)
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
hist = model.fit_generator(datagen.flow
                    (x_train, y_train, batch_size=batch_size),
                    #steps_per_epoch=x_train.shape[0],
                    epochs=25,
                    verbose=1,validation_data=(x_test,y_test),
                    callbacks=[LearningRateScheduler(lr_schedule)])
```

3. Performance metrics

```
#testing
_,acc=model.evaluate(x_test,y_test)
print("Accuracy:")
print(acc*100)
```

```
from matplotlib import pyplot as plt
```

```
#  "Accuracy"

plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()

# "Loss"
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

Module-5

Prediction

1. Predict and display using console

```
model.save("model1_cifar_25epoch.h5")
```

```
results={
   0:'aeroplane',
   1:'automobile',
   2:'bird',
   3:'cat',
   4:'deer',
   5:'dog',
   6:'frog',
   7:'horse',
   8:'ship',
   9:'truck'
}
```

```
from PIL import Image
import numpy as np
im=Image.open("aeroplane DL test 1.jpg")
# the input image is required to be in the shape of dataset, i.e (32,32,3)

im=im.resize((32,32))
im=np.expand_dims(im,axis=0)
im=np.array(im)
pred=model.predict_classes([im])[0]
print(pred,results[pred])
```

2. Predict and display using GUI

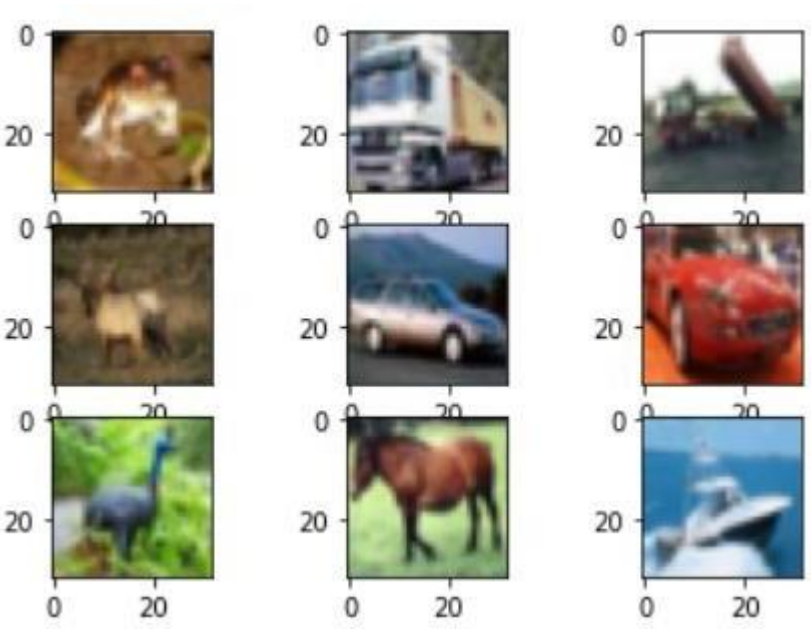
```
import tkinter as tk
from tkinter import filedialog
from tkinter import *
from PIL import ImageTk, Image
import numpy
from keras.models import load_model
model = load_model('model1_cifar_25epoch.h5')
classes = {
    0:'aeroplane',
    1:'automobile',
    2:'bird',
    3:'cat',
    4:'deer',
    5:'dog',
    6:'frog',
    7:'horse',
    8:'ship',
    9:'truck'
}
top=tk.Tk()
top.geometry('800x600')
top.title('Image Classification CIFAR10')
top.configure(background='#CDCDCD')
label=Label(top,background='#CDCDCD', font=('arial',15,'bold'))
sign_image = Label(top)
def classify(file_path):
    global label_packed
    image = Image.open(file_path)
    image = image.resize((32,32))
    image = numpy.expand_dims(image, axis=0)
    image = numpy.array(image)
    pred = model.predict_classes([image])[0]
    sign = classes[pred]
    print(sign)
```

```
    label.configure(foreground='#011638', text=sign)
def show_classify_button(file_path):
    classify_b=Button(top,text="Classify Image",
   command=lambda: classify(file_path),padx=10,pady=5)
    classify_b.configure(background='#364156', foreground='white',
font=('arial',10,'bold'))
    classify_b.place(relx=0.79,rely=0.46)
def upload_image():
    try:
        file_path=filedialog.askopenfilename()
        uploaded=Image.open(file_path)
        uploaded.thumbnail(((top.winfo_width()/2.25),
    (top.winfo_height()/2.25)))
        im=ImageTk.PhotoImage(uploaded)
        sign_image.configure(image=im)
        sign_image.image=im
        label.configure(text='')
        show_classify_button(file_path)
    except:
        pass
upload=Button(top,text="Upload an image",command=upload_image,
  padx=10,pady=5)

upload.configure(background='#364156', foreground='white',
    font=('arial',10,'bold'))
upload.pack(side=BOTTOM,pady=50)
sign_image.pack(side=BOTTOM,expand=True)
label.pack(side=BOTTOM,expand=True)
heading = Label(top, text="Image Classification CIFAR10",pady=20, font=('arial',20,'bold'))
heading.configure(background='#CDCDCD',foreground='#364156')
heading.pack()
top.mainloop()
```

Results:

Glimpse of dataset images (a plot of first few images)-



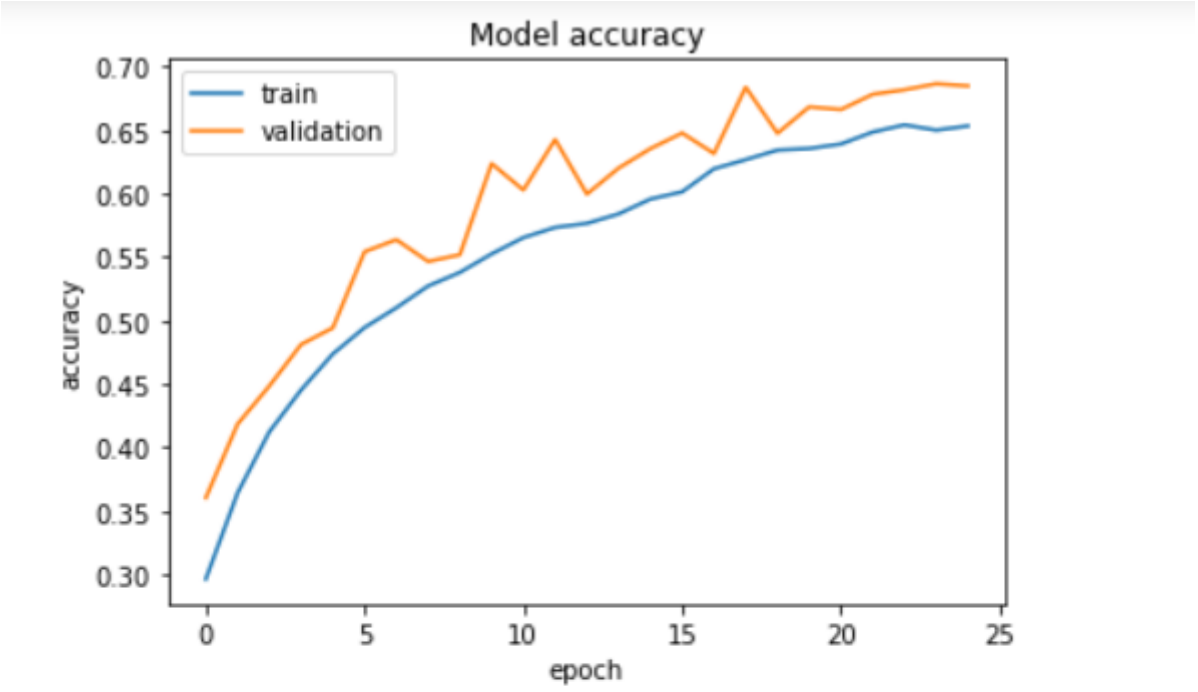
Model fit (the model is fitted with the given parameters)-

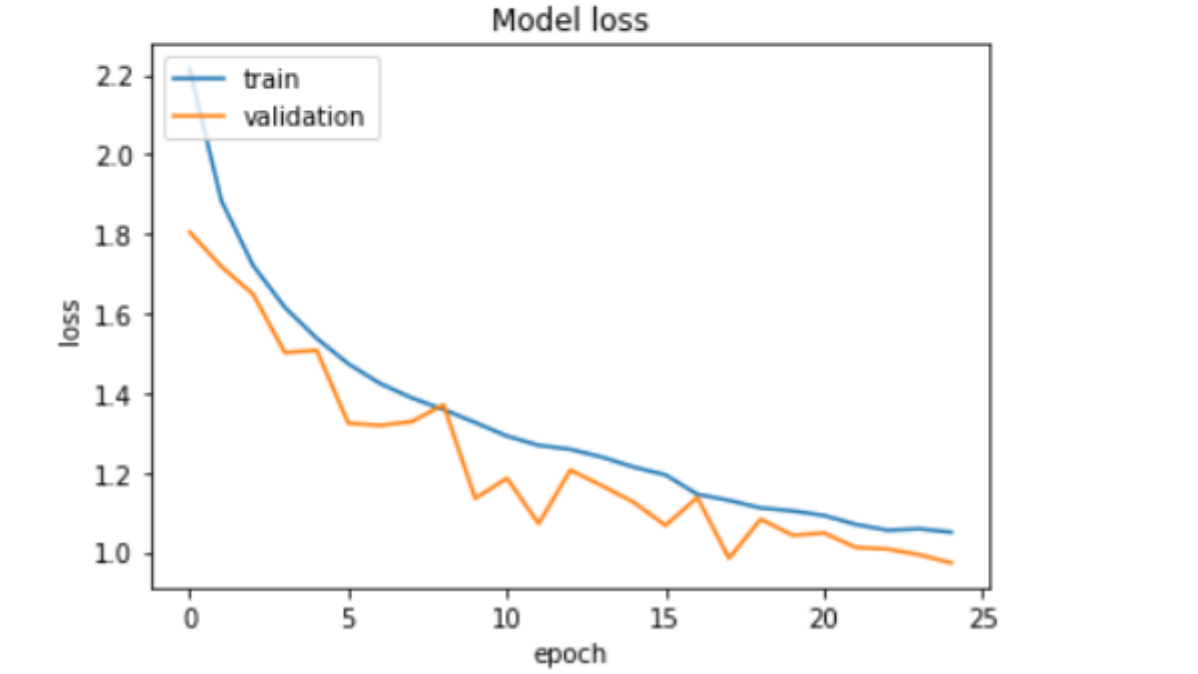
```
Epoch 1/25
196/196 [==============================] - 372s 2s/step - loss: 2.4151 - accuracy: 0.2639 - val_loss: 1.8050 - val_accuracy: 0.
3604
Epoch 2/25
196/196 [==============================] - 371s 2s/step - loss: 1.9476 - accuracy: 0.3500 - val_loss: 1.7182 - val_accuracy: 0.
4185
Epoch 3/25
196/196 [==============================] - 329s 2s/step - loss: 1.7417 - accuracy: 0.4034 - val_loss: 1.6483 - val_accuracy: 0.
4485
Epoch 4/25
196/196 [==============================] - 312s 2s/step - loss: 1.6387 - accuracy: 0.4394 - val_loss: 1.5016 - val_accuracy: 0.
4812
Epoch 5/25
196/196 [==============================] - 319s 2s/step - loss: 1.5465 - accuracy: 0.4718 - val_loss: 1.5077 - val_accuracy: 0.
4943
Epoch 6/25
196/196 [==============================] - 321s 2s/step - loss: 1.4873 - accuracy: 0.4899 - val_loss: 1.3240 - val_accuracy: 0.
5543
Epoch 7/25
196/196 [==============================] - 318s 2s/step - loss: 1.4251 - accuracy: 0.5089 - val_loss: 1.3185 - val_accuracy: 0.
5638
Epoch 8/25
196/196 [==============================] - 4960s 25s/step - loss: 1.4000 - accuracy: 0.5240 - val_loss: 1.3281 - val_accuracy:
0.5466
Epoch 9/25
196/196 [==============================] - 352s 2s/step - loss: 1.3668 - accuracy: 0.5332 - val_loss: 1.3706 - val_accuracy: 0.
5519
Epoch 10/25
196/196 [==============================] - 361s 2s/step - loss: 1.3222 - accuracy: 0.5512 - val_loss: 1.1349 - val_accuracy: 0.
6237
Epoch 11/25
196/196 [==============================] - 323s 2s/step - loss: 1.2833 - accuracy: 0.5658 - val_loss: 1.1856 - val_accuracy: 0.
6029
Epoch 12/25
196/196 [==============================] - 319s 2s/step - loss: 1.2809 - accuracy: 0.5697 - val_loss: 1.0716 - val_accuracy: 0.
6427
196/196 [==============================] - 324s 2s/step - loss: 1.2708 - accuracy: 0.5743 - val_loss: 1.2063 - val_accuracy: 0.
5997
Epoch 14/25
196/196 [==============================] - 325s 2s/step - loss: 1.2417 - accuracy: 0.5812 - val_loss: 1.1667 - val_accuracy: 0.
6202
Epoch 15/25
196/196 [==============================] - 349s 2s/step - loss: 1.2133 - accuracy: 0.5949 - val_loss: 1.1250 - val_accuracy: 0.
6355
Epoch 16/25
196/196 [==============================] - 367s 2s/step - loss: 1.1863 - accuracy: 0.6016 - val_loss: 1.0671 - val_accuracy: 0.
6478
Epoch 17/25
196/196 [==============================] - 344s 2s/step - loss: 1.1555 - accuracy: 0.6166 - val_loss: 1.1375 - val_accuracy: 0.
6317
Epoch 18/25
196/196 [==============================] - 337s 2s/step - loss: 1.1280 - accuracy: 0.6292 - val_loss: 0.9842 - val_accuracy: 0.
6839
Epoch 19/25
196/196 [==============================] - 323s 2s/step - loss: 1.1138 - accuracy: 0.6338 - val_loss: 1.0829 - val_accuracy: 0.
6476
Epoch 20/25
196/196 [==============================] - 350s 2s/step - loss: 1.1021 - accuracy: 0.6349 - val_loss: 1.0423 - val_accuracy: 0.
6684
Epoch 21/25
196/196 [==============================] - 350s 2s/step - loss: 1.0882 - accuracy: 0.6402 - val_loss: 1.0476 - val_accuracy: 0.
6663
Epoch 22/25
196/196 [==============================] - 340s 2s/step - loss: 1.0789 - accuracy: 0.6439 - val_loss: 1.0117 - val_accuracy: 0.
6783
Epoch 23/25
196/196 [==============================] - 330s 2s/step - loss: 1.0562 - accuracy: 0.6527 - val_loss: 1.0075 - val_accuracy: 0.
6819
Epoch 24/25
196/196 [==============================] - 324s 2s/step - loss: 1.0550 - accuracy: 0.6520 - val_loss: 0.9932 - val_accuracy: 0.
6866
Epoch 25/25
196/196 [==============================] - 356s 2s/step - loss: 1.0513 - accuracy: 0.6520 - val_loss: 0.9729 - val_accuracy: 0.
6849
```

Performance metrics (the accuracy of the model is calculated)-

```
313/313 [==============================] - 12s 40ms/step - loss: 0.9729 - accuracy: 0.6849
Accuracy:
68.48999857902527
```

Accuracy and loss plot-





Predictions and actual output-

```
pred = model.predict(x_test).argmax(axis=1)
pred[:10]
```

```
array([3, 8, 8, 8, 6, 6, 1, 6, 6, 1], dtype=int64)
```

```
y_test1[:10]
```

```
array([[3],
       [8],
       [8],
       [0],
       [6],
       [6],
       [1],
       [6],
       [3],
       [1]], dtype=uint8)
```

Final outputs:

1. Prediction through console-

Image-

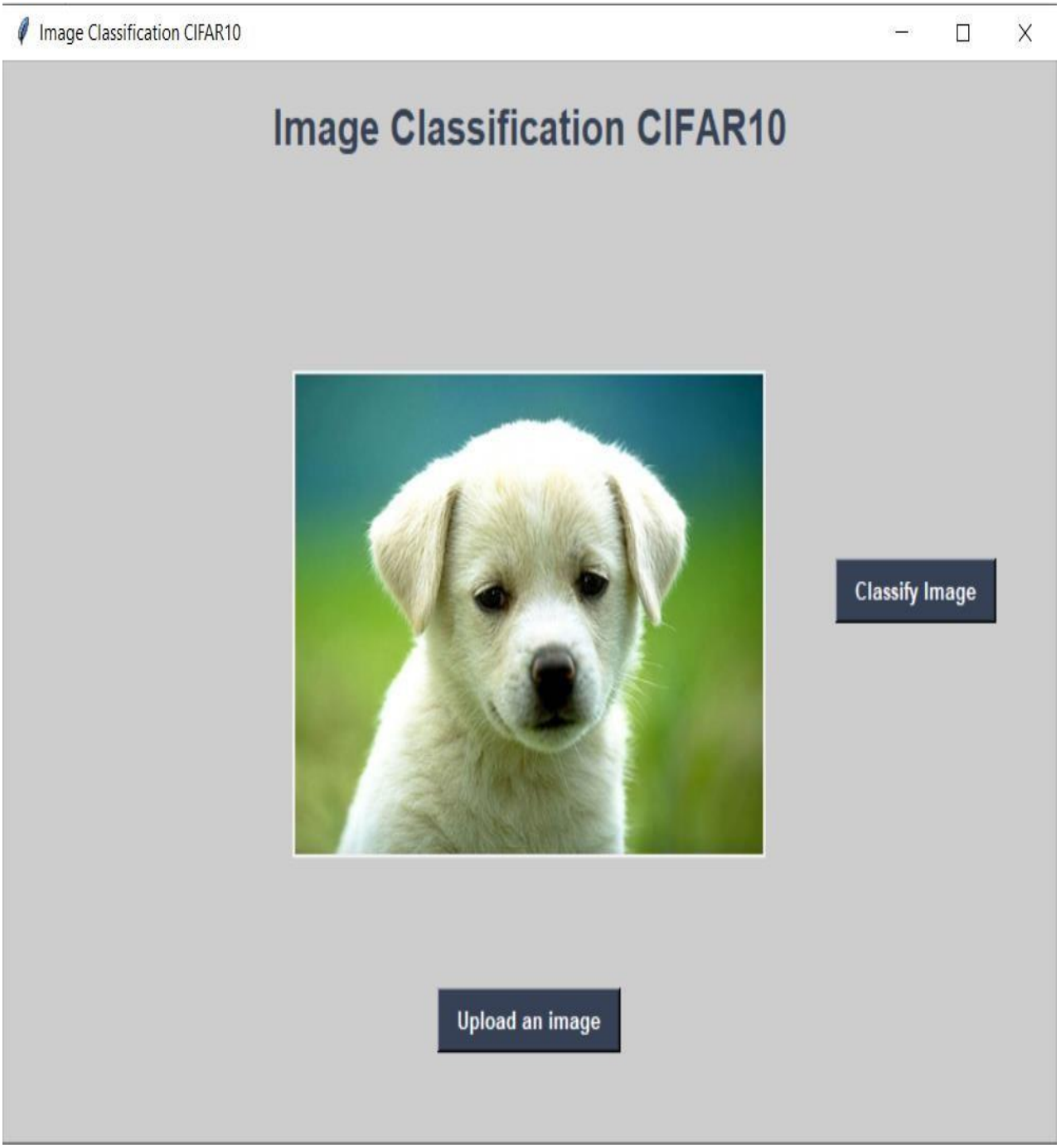
Prediction-

```
0 aeroplane
```

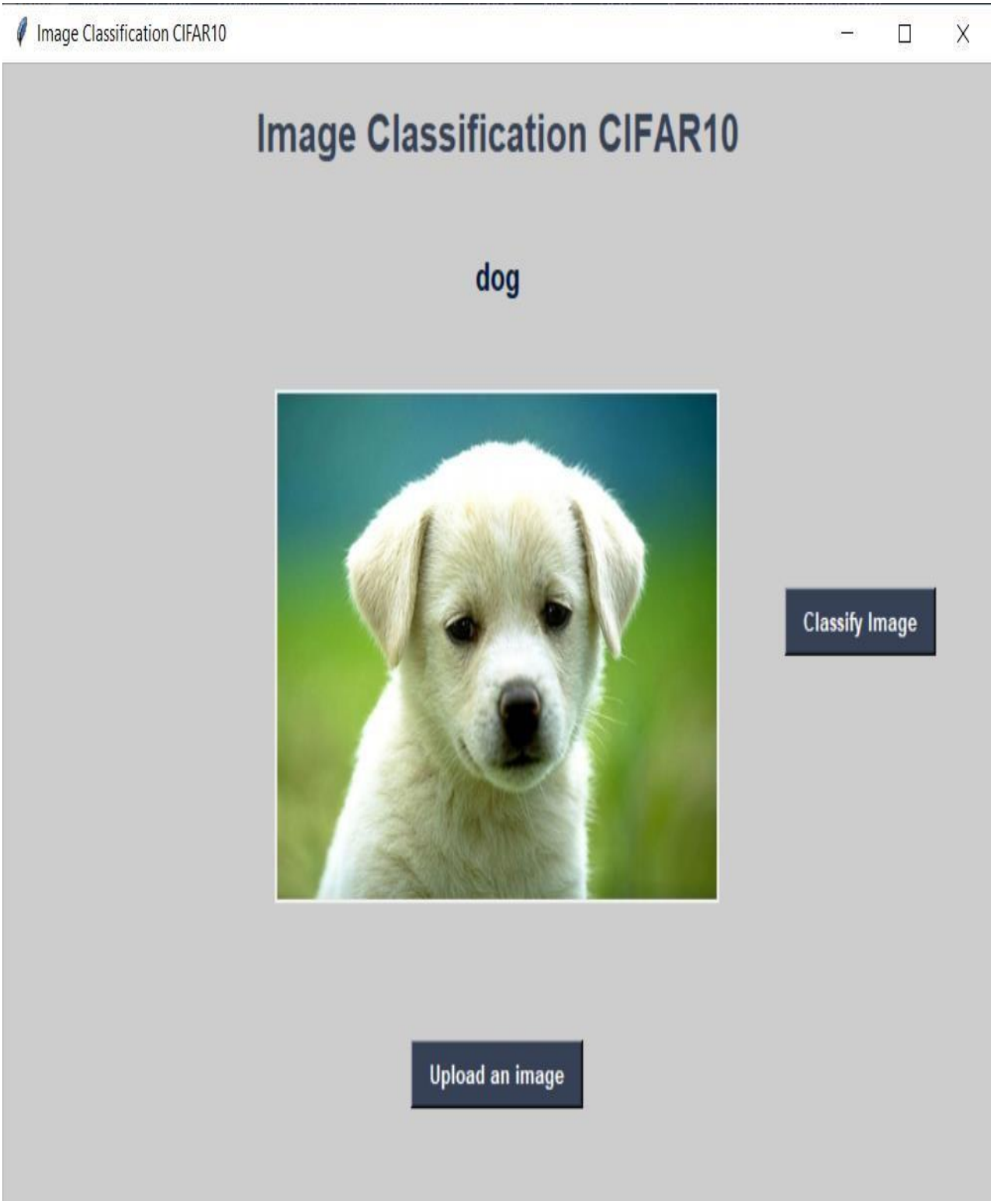
Where 0 corresponds to class and aeroplane is the predicted label

2. Prediction through GUI-

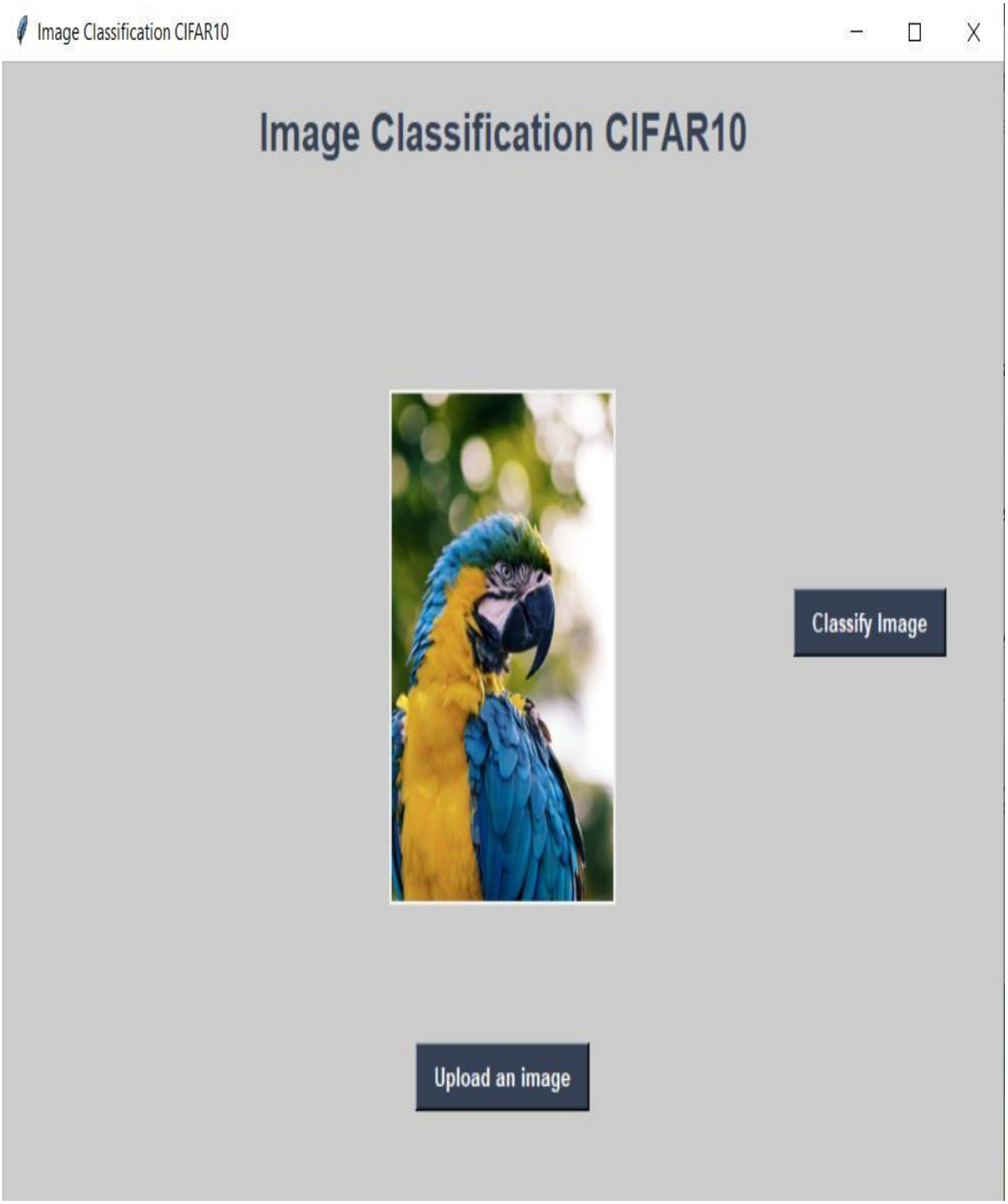
Eg-1: On uploading the image-



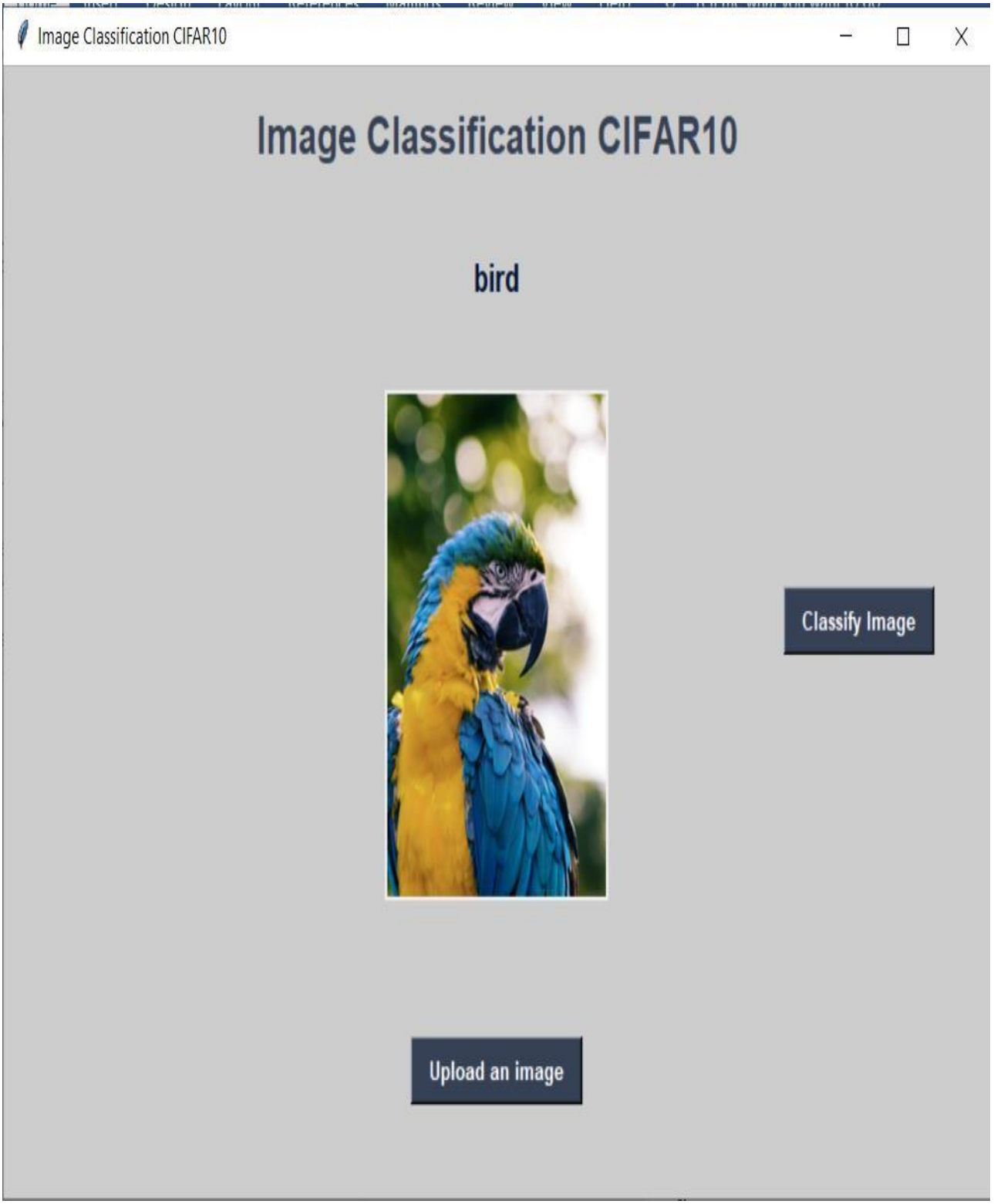
On clicking classify image-



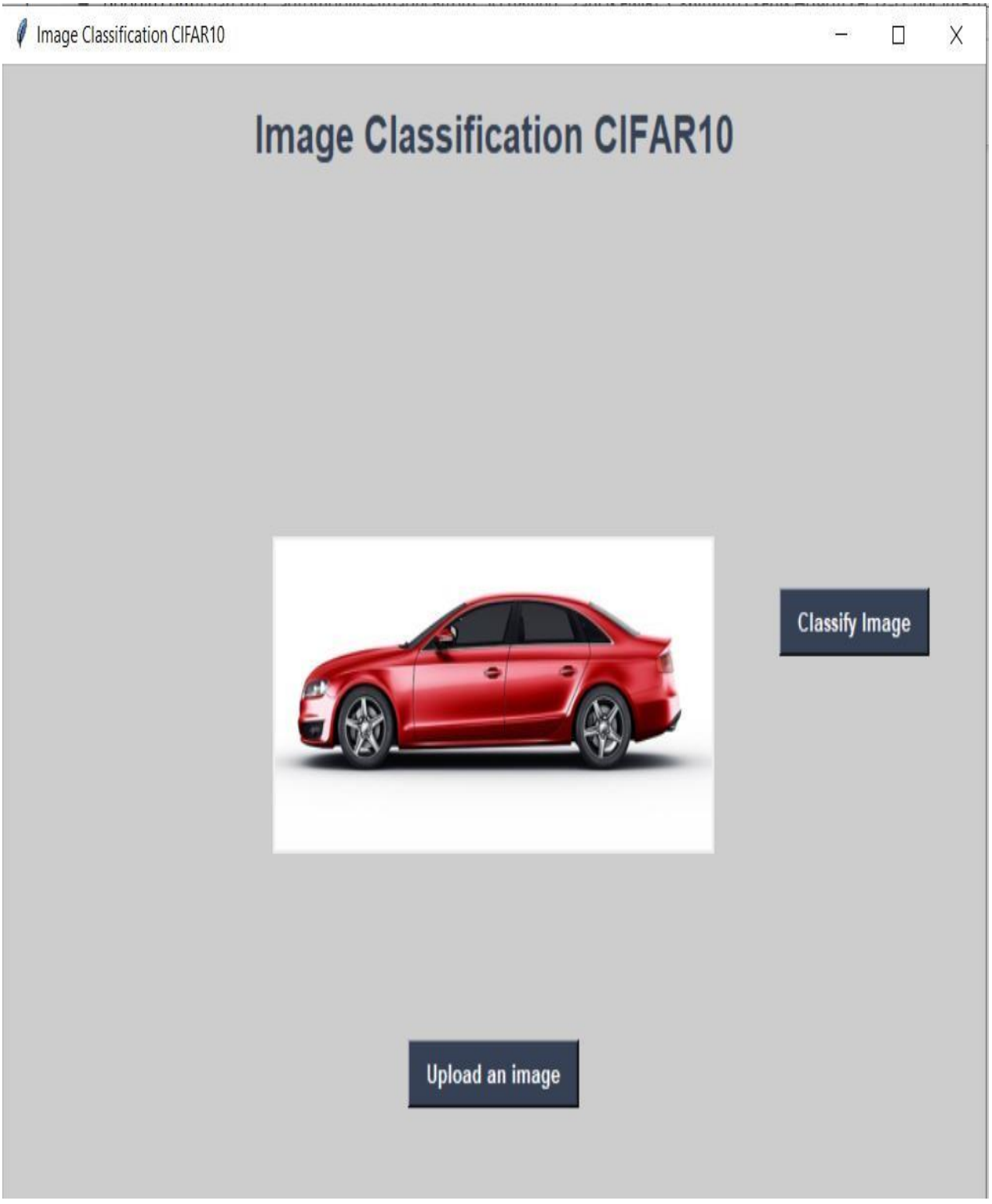
Eg-2: On uploading the image-



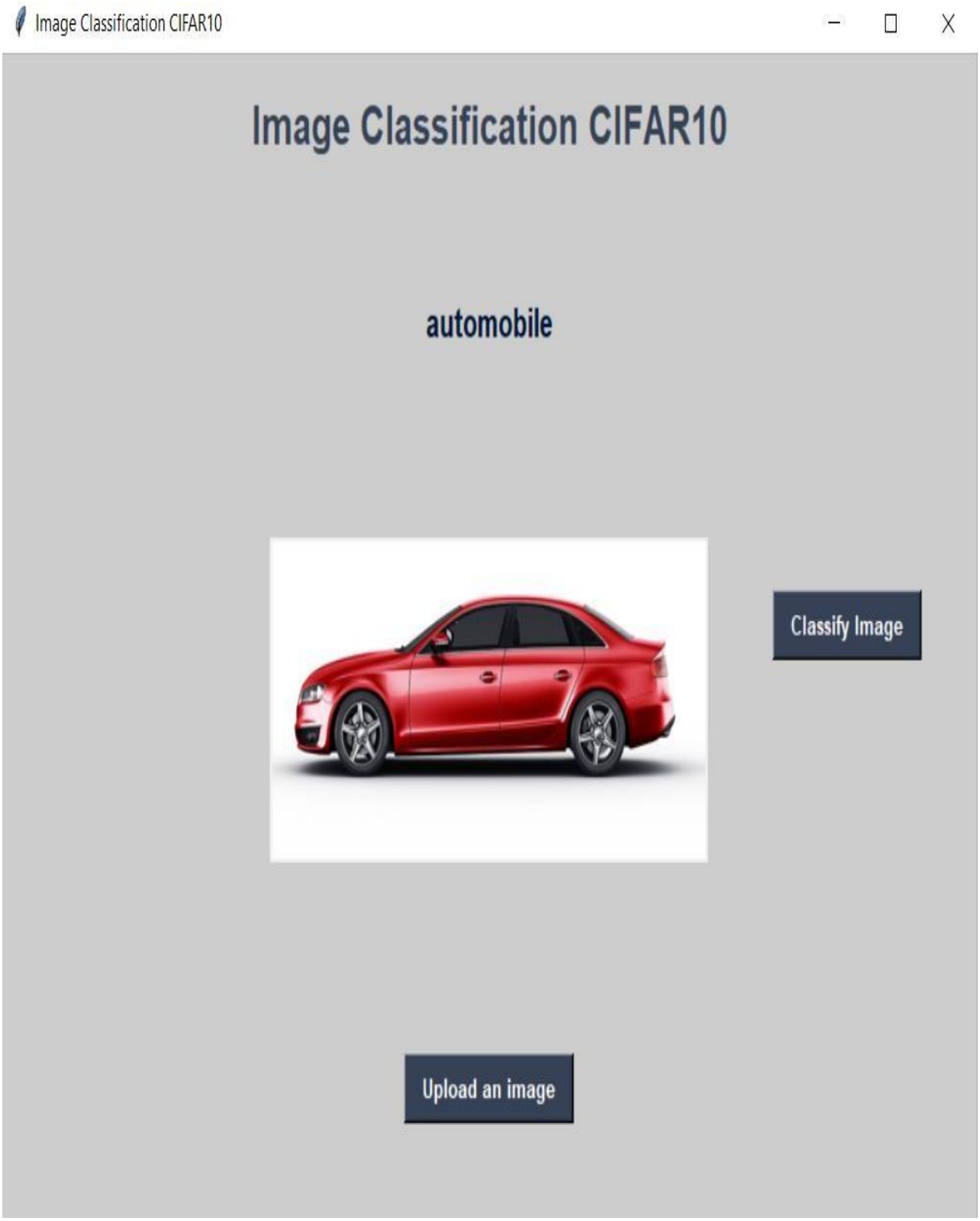
On clicking classify image-



Eg-3: On uploading the image-



On clicking classify image-



Conclusion:

The images in the 10 classes namely, airplanes, automobile, birds, cats, deer, dogs, frogs, horses, ships, and trucks of the CIFAR-10 dataset are classified into their respective classes with an accuracy of 85.73%. This accuracy can be improved by increasing the number of epochs. For instance, we can obtain an accuracy of 95% and above by increasing the number of epochs to 150 and more. But the increasing should be as such that the model is not overfitted.

References:

1. **Image Classification– CIFAR-10 Dataset** by **Tushar Jajodia** and **Pankaj Garg** published in **2020**.

2. **Improvement in Convolutional Neural Network for CIFAR-10 Dataset Image Classification** by **Suyesh Pandit** and **Sushil Kumar** published in **2020**.