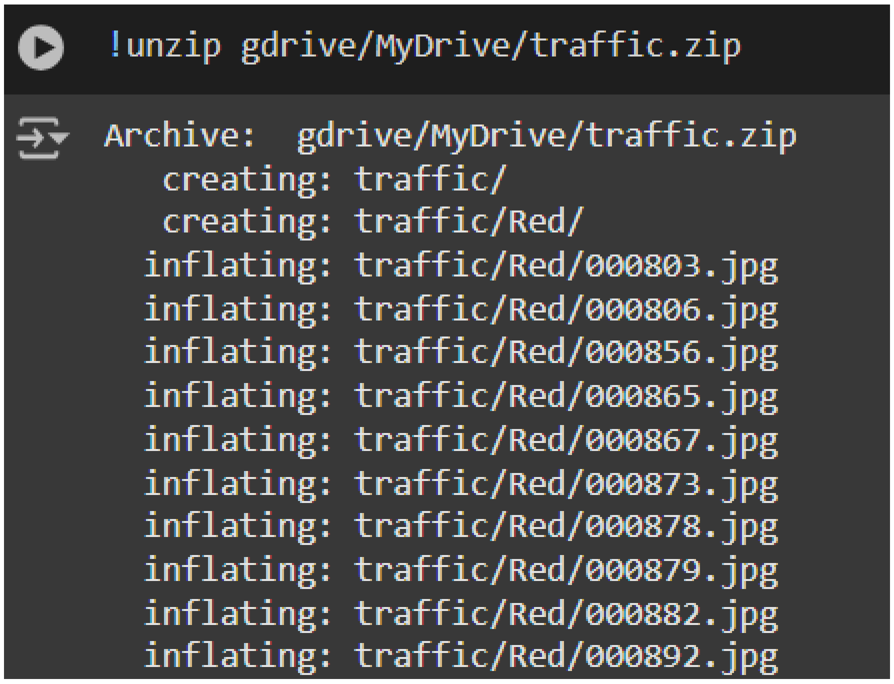
COURSEWORK 2 FOR AIFP

ID-2293815

In this coursework, we are training a model to predict the traffic signals. Our dataset is uploaded in Google drive, we mounted the drive to access the files for using it. 0 is labelled for Green and 1 is for Red.

Question 1: 2.1. Prepare the dataset for training (Data augmentation, splitting the dataset in training and testing datasets…).

After Mounting the drive, we unzipped the folder as shown in Figure 1.

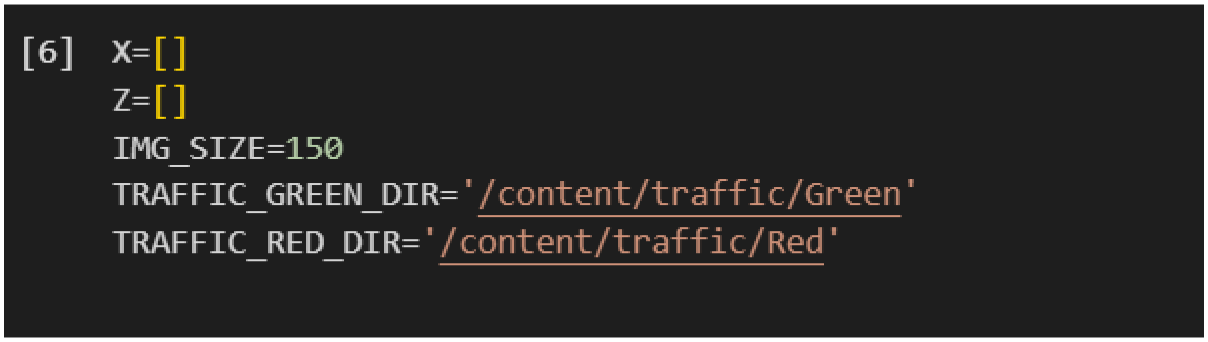


```
!unzip gdrive/MyDrive/traffic.zip
```

```
Archive:  gdrive/MyDrive/traffic.zip
   creating: traffic/
   creating: traffic/Red/
  inflating: traffic/Red/000803.jpg
  inflating: traffic/Red/000806.jpg
  inflating: traffic/Red/000856.jpg
  inflating: traffic/Red/000865.jpg
  inflating: traffic/Red/000867.jpg
  inflating: traffic/Red/000873.jpg
  inflating: traffic/Red/000878.jpg
  inflating: traffic/Red/000879.jpg
  inflating: traffic/Red/000882.jpg
  inflating: traffic/Red/000892.jpg
```

Figure 1: Unzipping the photo folder

For preparing my dataset I used 150*150 resolution for the model to capture more details of the photos as shown in Figure 2.

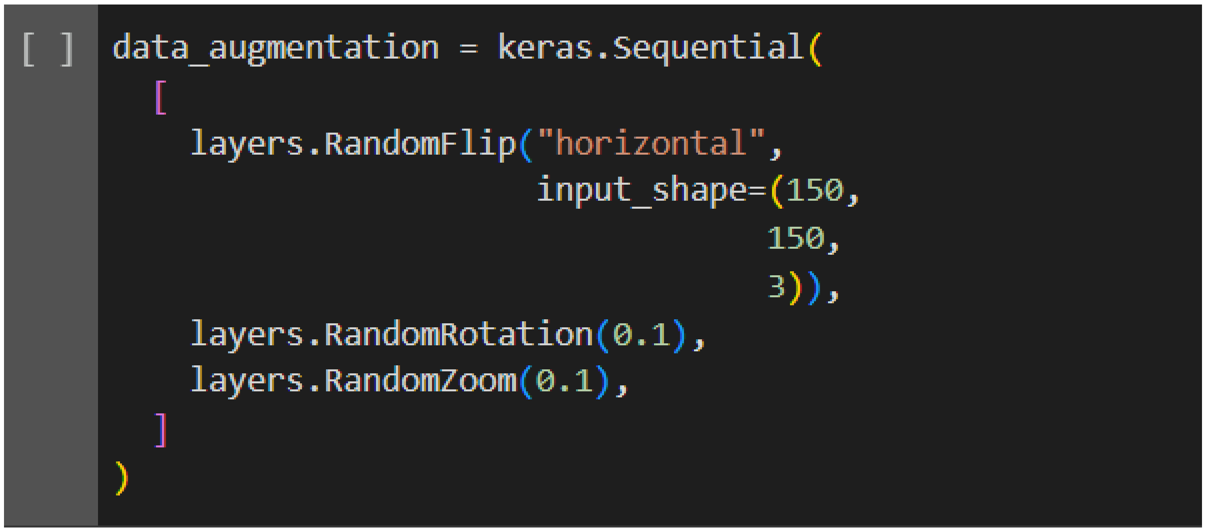


```
[6] X=[]
    Z=[]
    IMG_SIZE=150
    TRAFFIC_GREEN_DIR='/content/traffic/Green'
    TRAFFIC_RED_DIR='/content/traffic/Red'
```

Figure 2: Resizing the images

Then I assigned green and red labels to the directory as training data.

Then I split the training and testing data in 80-20 percent ratio and augmented the dataset. As shown in Figure 3, this data augmentation randomly flips the data horizontally (left to right), rotates and zooms the input by 0.1 factor randomly. This will help to generalize the model by teaching the model different aspects of the pictures.

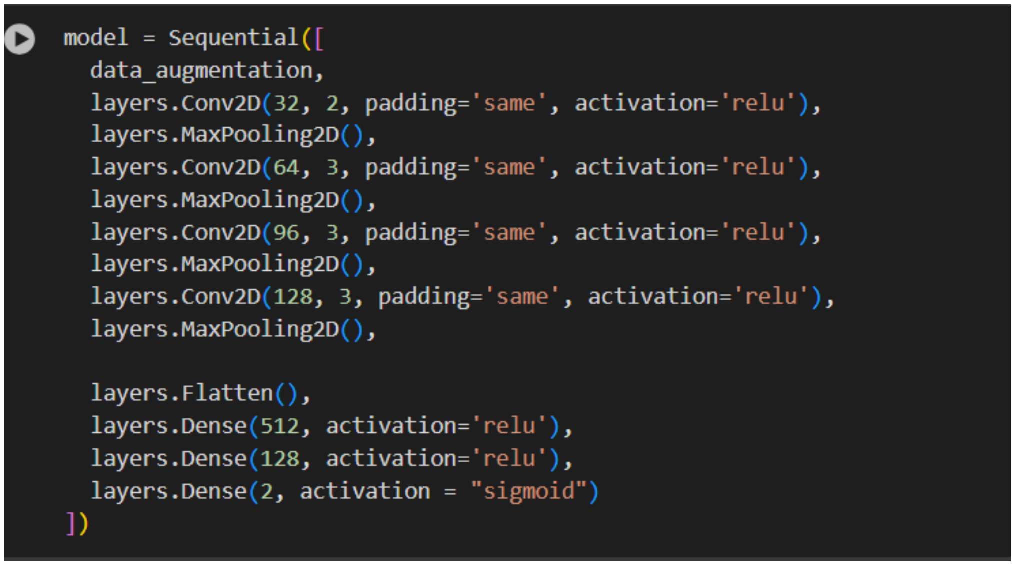


```
[ ] data_augmentation = keras.Sequential(
      [
        layers.RandomFlip("horizontal",
                          input_shape=(150,
                                       150,
                                       3)),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.1),
      ]
    )
```

Figure 3: Data Augmentation

Question 2: 2.2. Design a 2D convolutional neural network to classify these images.

After preparing the dataset, I designed a 2D convolutional neural network to classify these images as shown in Figure 4.



```
model = Sequential([
  data_augmentation,
  layers.Conv2D(32, 2, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(96, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(128, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),

  layers.Flatten(),
  layers.Dense(512, activation='relu'),
  layers.Dense(128, activation='relu'),
  layers.Dense(2, activation = "sigmoid")
])
```

Figure 4: Design of 2D CNN

The specifications of this CNN are as follows-

- First, the data augmentation layer is added;
- Then Conv2D and Maxpooling2D layers following the Conv2D are added;
- Conv2D layers extract spatial features and MaxPooling2D reduces the dimension while retaining important features;
- 1st Conv2D layer was of 32 filter and a 2x2 kernel to detect patterns in the input image;
- Afterwards the Conv2D layers used 3x3 kernels for colored images and used 64, 96, 128 filters consecutively;
- Then I flattened the output of the con2D layer to 1D vector for giving it to the dense layers.
- I had 3 dense layers which had 512, 128 and 2 neurons respectively;
- The last layer with 2 neurons is for the output layer with binary classification;
- ReLu activation function introduces non-linearity in the model.

Then I compiled the model and trained it with 50 epochs and 10% validation data.

Question 3: 2.3. Provide the training loss and accuracy plots and report the accuracy of the model.

Figure 5 and Figure 6 shows the Loss and Accuracy curves of the model.

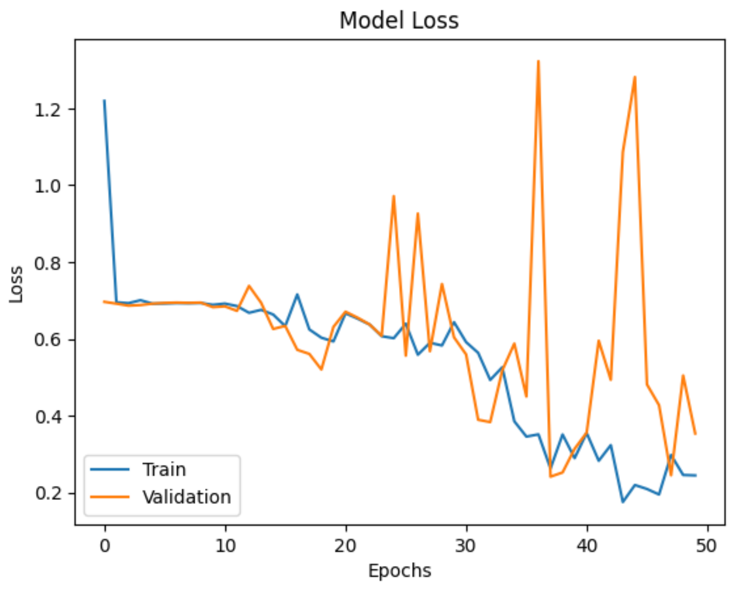


Figure 5: Loss Curve of the Conv2D Model

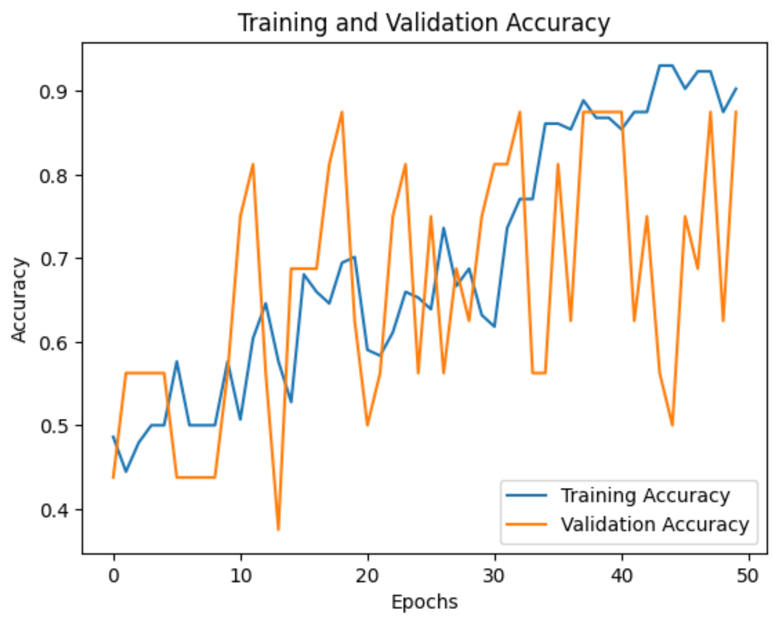
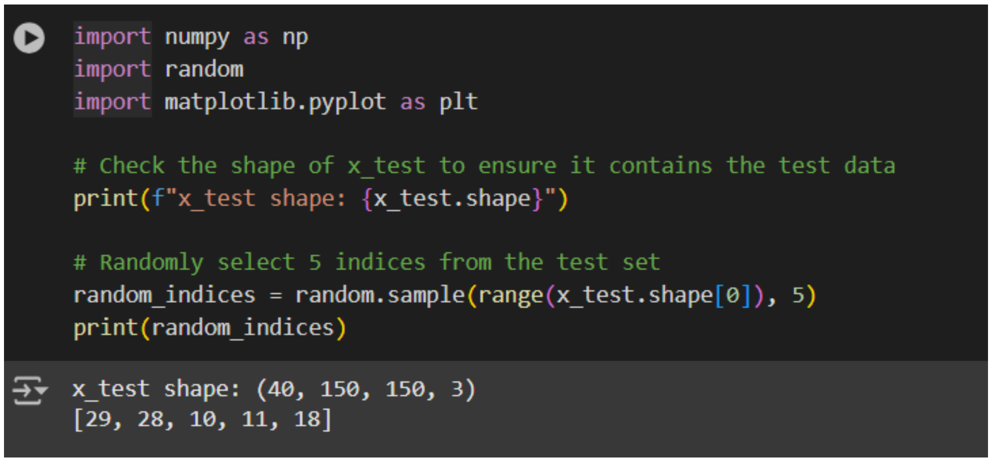


Figure 6: Accuracy Curve of the Conv2D Model

According to the pictures, although the training accuracy is much high after increasing to almost 90%, the validation accuracy is struggling to improve and fluctuates a lot. This suggests potential overfitting, meaning that the model is not generalized.

Question 4: 2.4. Select five testing images (The images should not be in your training dataset). Please report the predicted labels for all these selected images.

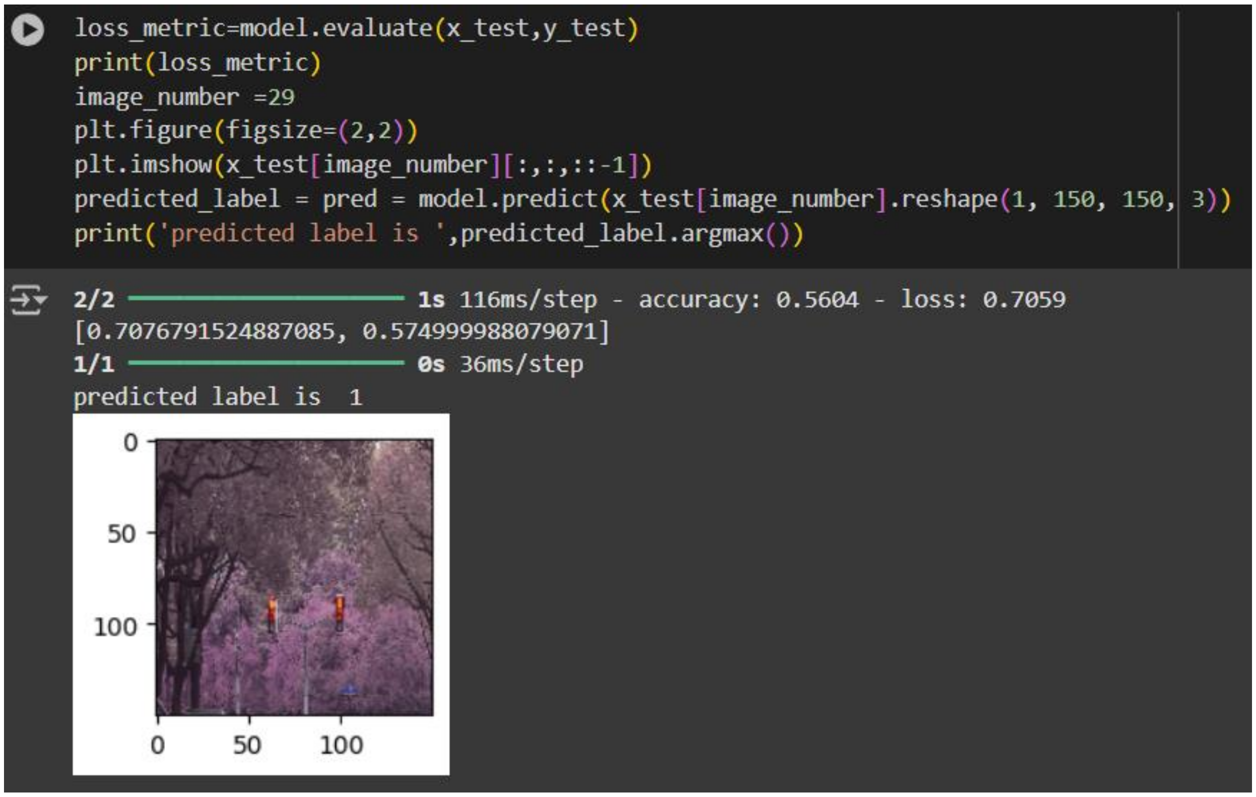
For testing, I first took random 5 indices from the testing dataset shown in Figure 7, then with each index, I predicted the labels.



```
import numpy as np
import random
import matplotlib.pyplot as plt

# Check the shape of x_test to ensure it contains the test data
print(f"x_test shape: {x_test.shape}")

# Randomly select 5 indices from the test set
random_indices = random.sample(range(x_test.shape[0]), 5)
print(random_indices)
```

```
x_test shape: (40, 150, 150, 3)
[29, 28, 10, 11, 18]
```

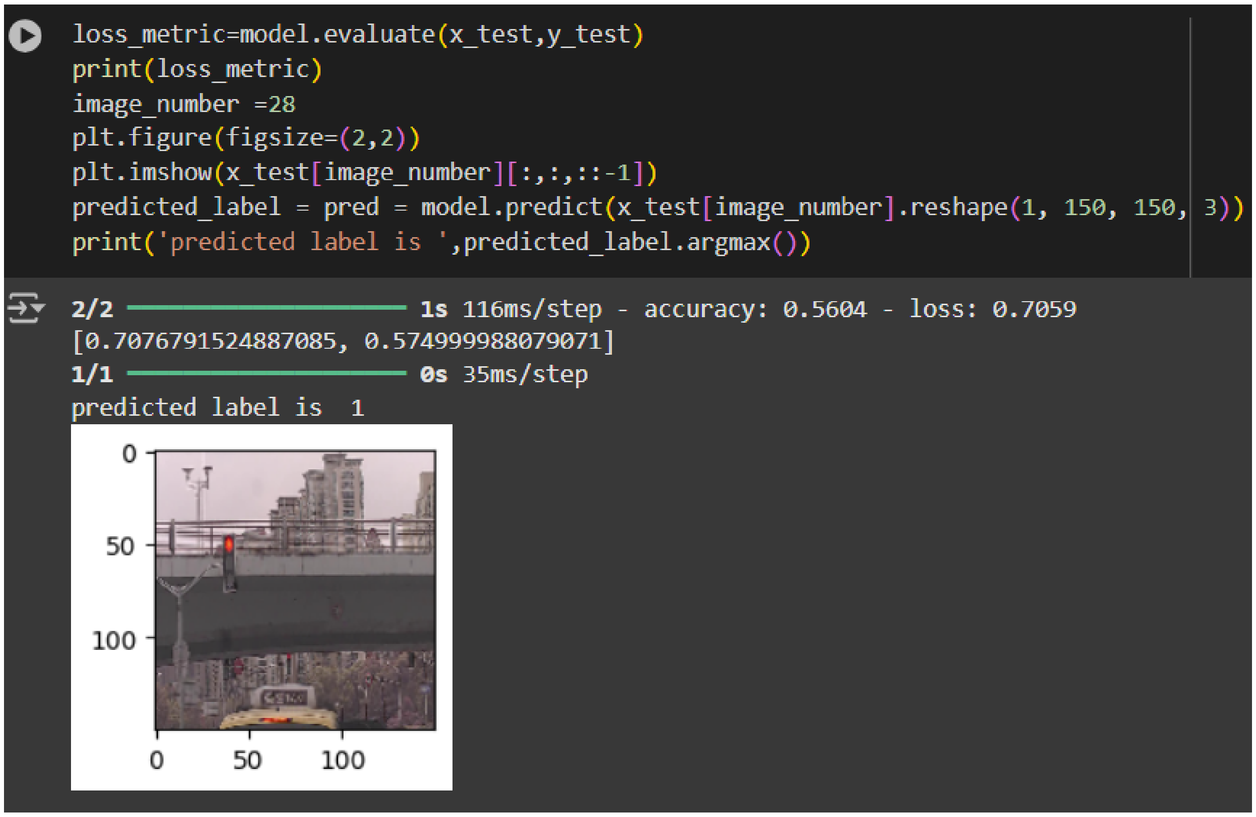
Figure 7: Selecting Test image indices from the Testing set



```
loss_metric=model.evaluate(x_test,y_test)
print(loss_metric)
image_number =29
plt.figure(figsize=(2,2))
plt.imshow(x_test[image_number][:,:,::-1])
predicted_label = pred = model.predict(x_test[image_number].reshape(1, 150, 150, 3))
print('predicted label is ',predicted_label.argmax())
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 116ms/step - accuracy: 0.5604 - loss: 0.7059
[0.7076791524887085, 0.574999988079071]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
predicted label is  1
```

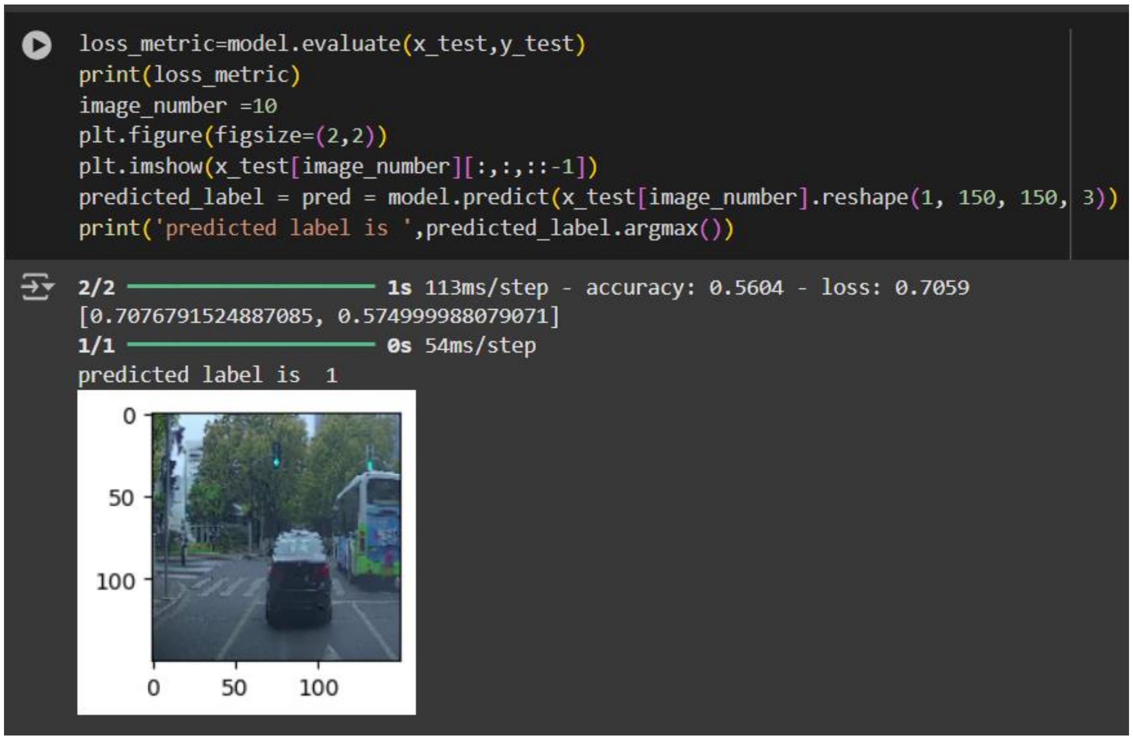
Figure 8: *Predicting labels from the testing dataset (1)*



```
loss_metric=model.evaluate(x_test,y_test)
print(loss_metric)
image_number =28
plt.figure(figsize=(2,2))
plt.imshow(x_test[image_number][:,:,::-1])
predicted_label = pred = model.predict(x_test[image_number].reshape(1, 150, 150, 3))
print('predicted label is ',predicted_label.argmax())
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 116ms/step - accuracy: 0.5604 - loss: 0.7059
[0.7076791524887085, 0.574999988079071]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
predicted label is  1
```

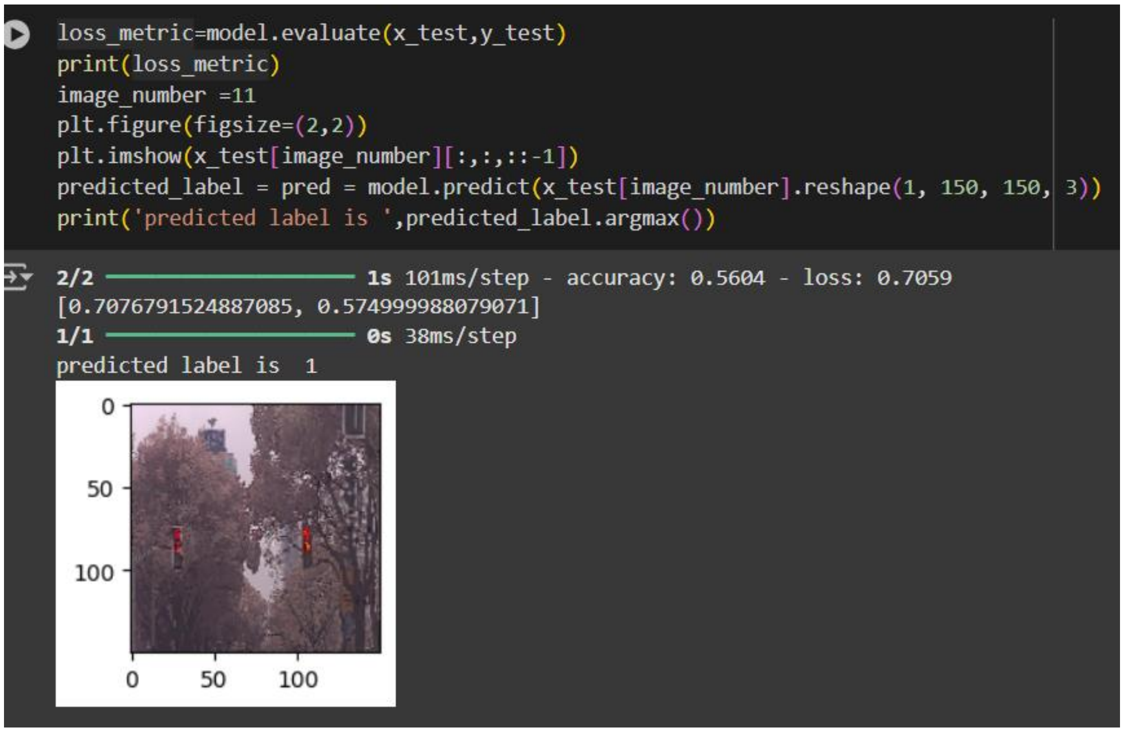
Figure 9: *Predicting labels from the testing dataset (2)*



```
loss_metric=model.evaluate(x_test,y_test)
print(loss_metric)
image_number =10
plt.figure(figsize=(2,2))
plt.imshow(x_test[image_number][:,:,::-1])
predicted_label = pred = model.predict(x_test[image_number].reshape(1, 150, 150, 3))
print('predicted label is ',predicted_label.argmax())
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 113ms/step - accuracy: 0.5604 - loss: 0.7059
[0.7076791524887085, 0.574999988079071]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
predicted label is  1
```

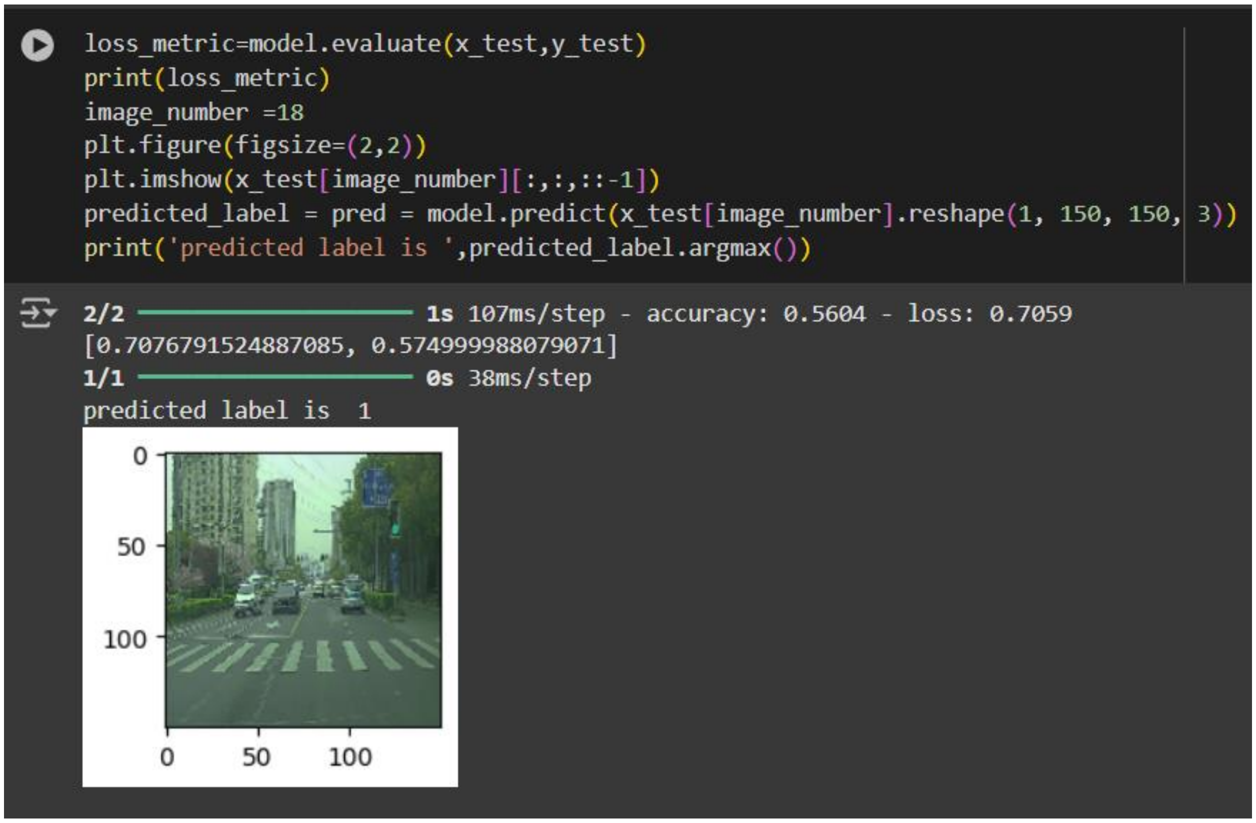
Figure 10: Predicting labels from the testing dataset (3)



```
loss_metric=model.evaluate(x_test,y_test)
print(loss_metric)
image_number =11
plt.figure(figsize=(2,2))
plt.imshow(x_test[image_number][:,:,::-1])
predicted_label = pred = model.predict(x_test[image_number].reshape(1, 150, 150, 3))
print('predicted label is ',predicted_label.argmax())
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 101ms/step - accuracy: 0.5604 - loss: 0.7059
[0.7076791524887085, 0.574999988079071]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
predicted label is  1
```

Figure 11: Predicting labels from the testing dataset (4)



```
loss_metric=model.evaluate(x_test,y_test)
print(loss_metric)
image_number =18
plt.figure(figsize=(2,2))
plt.imshow(x_test[image_number][:,:,::-1])
predicted_label = pred = model.predict(x_test[image_number].reshape(1, 150, 150, 3))
print('predicted label is ',predicted_label.argmax())
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 107ms/step - accuracy: 0.5604 - loss: 0.7059
[0.7076791524887085, 0.574999988079071]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
predicted label is  1
```

Figure 12: Predicting labels from the testing dataset (5)

From Figure 8, Figure 9, Figure 10, Figure 11 and Figure 12, it is visible that the model is predicting every picture as label 1, suggesting the model is overfitted. Decoding the accuracy scores in the pictures also makes sense, it can only correctly predict 50% of the data.

Question 5: 2.5. Please provide two ways (with evidence) to further improve the accuracy of your classifier.

Way 01: Including Data Augmentation before

In this way I included the data augmentation step before splitting the data and kept everything else the way it was. Doing this, improved the accuracy till almost 90%.

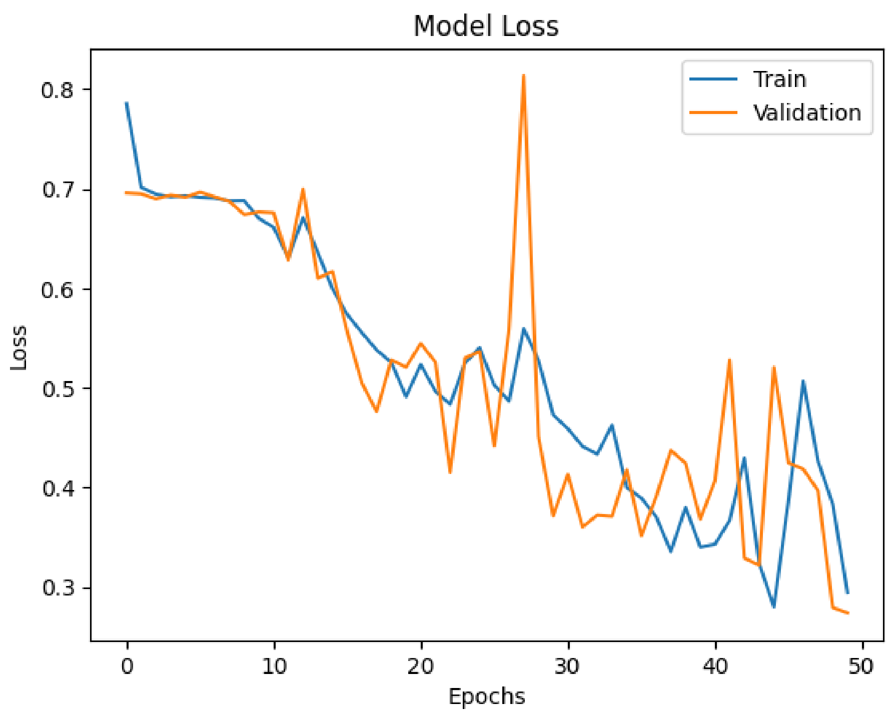


Figure 13: Loss Curve of the Data Augmentation Mode

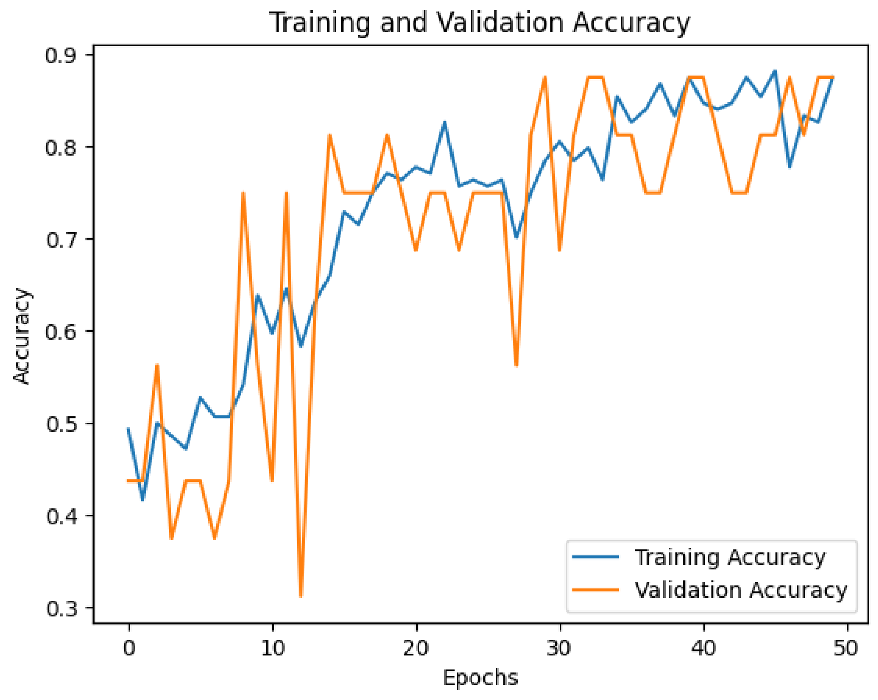
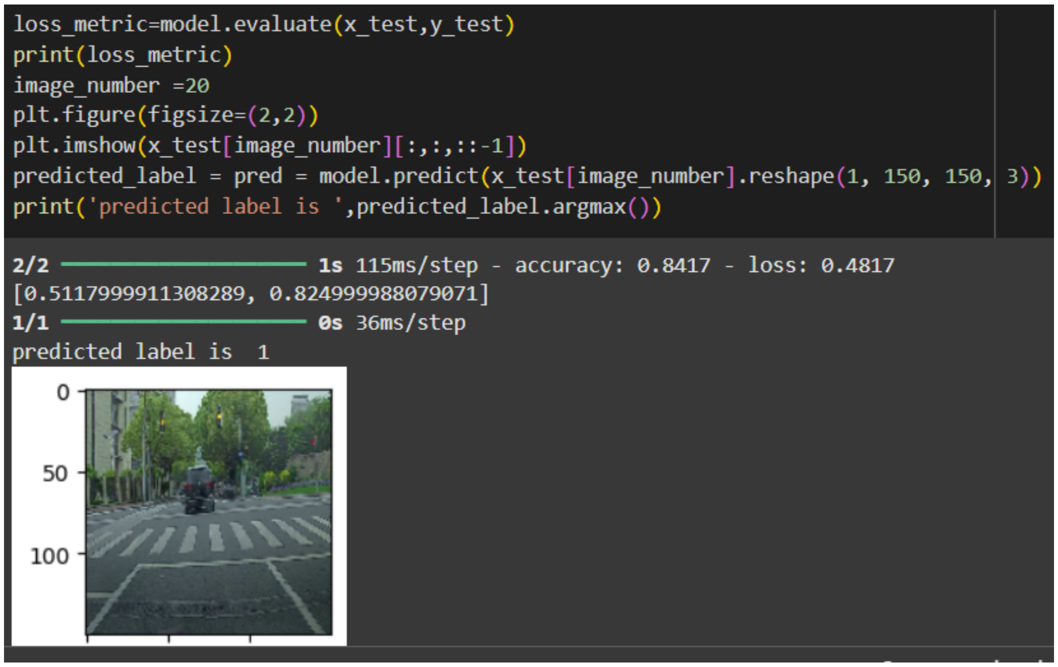


Figure 14: Accuracy Curve of the Data Augmentation Model

From Figure 13 and Figure 14 it can be said that, the model is learning with time, and the fluctuations in the validation curve suggests any difficult pictures hard to predict, although the generalization in quite good.

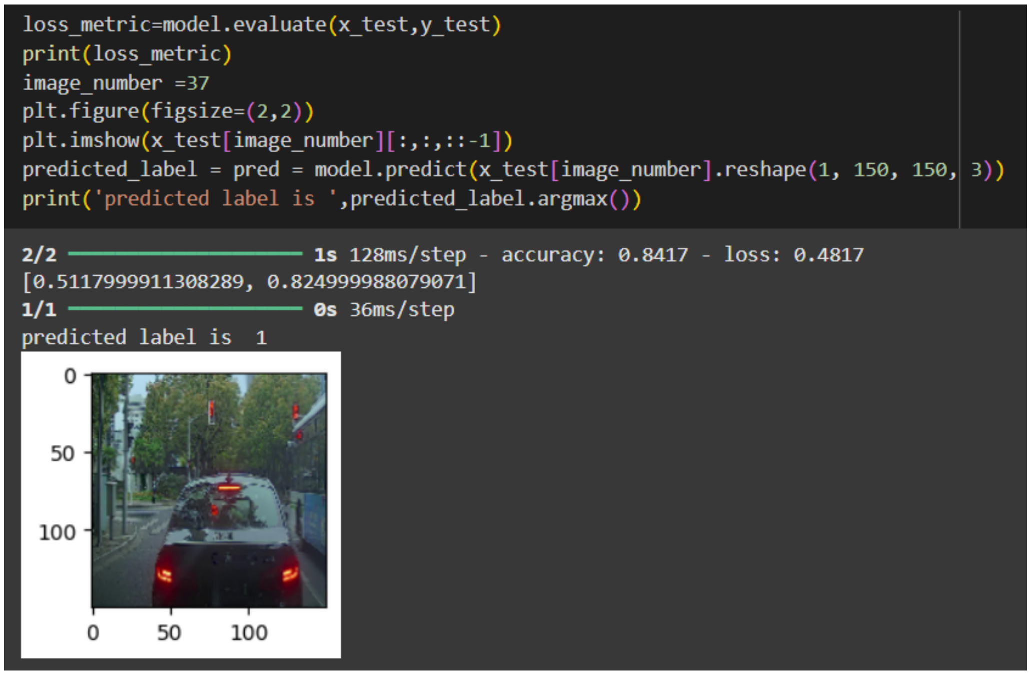
Then as before, I tested with 5 testing images shown in Figure 15, Figure 16, Figure 17, Figure 18, and in Figure 19.



```
loss_metric=model.evaluate(x_test,y_test)
print(loss_metric)
image_number =20
plt.figure(figsize=(2,2))
plt.imshow(x_test[image_number][:,:,::-1])
predicted_label = pred = model.predict(x_test[image_number].reshape(1, 150, 150, 3))
print('predicted label is ',predicted_label.argmax())
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 115ms/step - accuracy: 0.8417 - loss: 0.4817
[0.5117999911308289, 0.824999988079071]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
predicted label is  1
```

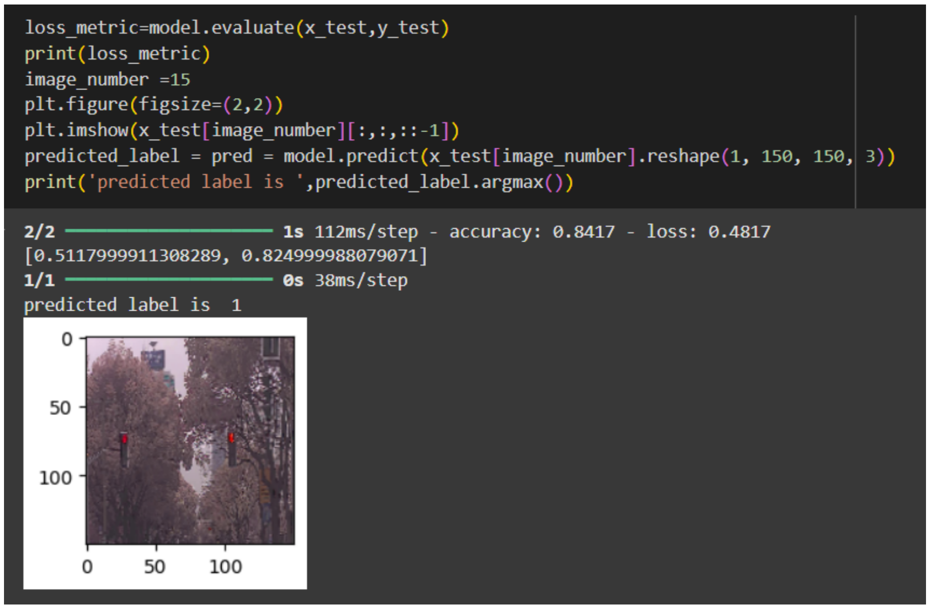
Figure 15: Predicting labels from the testing dataset with Data Augmentation (1)



```
loss_metric=model.evaluate(x_test,y_test)
print(loss_metric)
image_number =37
plt.figure(figsize=(2,2))
plt.imshow(x_test[image_number][:,:,::-1])
predicted_label = pred = model.predict(x_test[image_number].reshape(1, 150, 150, 3))
print('predicted label is ',predicted_label.argmax())
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 128ms/step - accuracy: 0.8417 - loss: 0.4817
[0.5117999911308289, 0.824999988079071]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
predicted label is  1
```

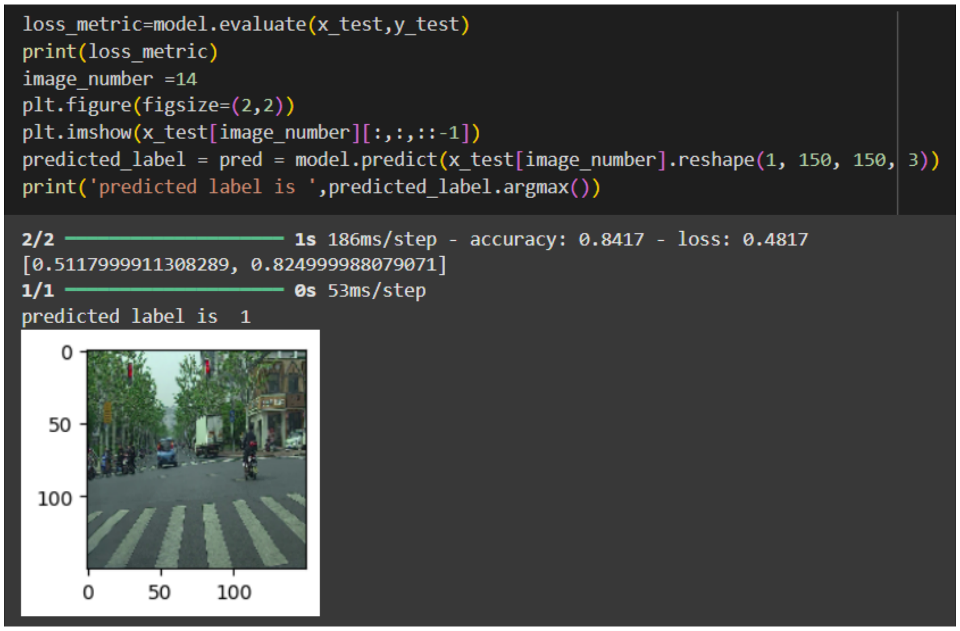
Figure 16: Predicting labels from the testing dataset with Data Augmentation (2)



```
loss_metric=model.evaluate(x_test,y_test)
print(loss_metric)
image_number =15
plt.figure(figsize=(2,2))
plt.imshow(x_test[image_number][:,:,::-1])
predicted_label = pred = model.predict(x_test[image_number].reshape(1, 150, 150, 3))
print('predicted label is ',predicted_label.argmax())
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 112ms/step - accuracy: 0.8417 - loss: 0.4817
[0.5117999911308289, 0.824999988079071]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
predicted label is  1
```

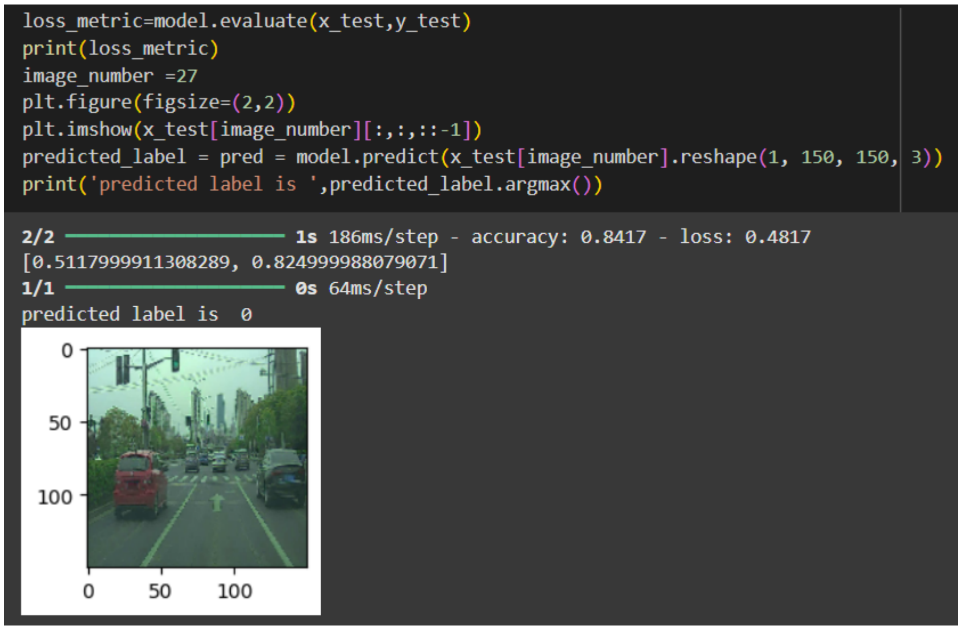
Figure 17: Predicting labels from the testing dataset with Data Augmentation (3)



```
loss_metric=model.evaluate(x_test,y_test)
print(loss_metric)
image_number =14
plt.figure(figsize=(2,2))
plt.imshow(x_test[image_number][:,:,::-1])
predicted_label = pred = model.predict(x_test[image_number].reshape(1, 150, 150, 3))
print('predicted label is ',predicted_label.argmax())
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 186ms/step - accuracy: 0.8417 - loss: 0.4817
[0.5117999911308289, 0.824999988079071]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
predicted label is  1
```

Figure 18: Predicting labels from the testing dataset with Data Augmentation (4)



```
loss_metric=model.evaluate(x_test,y_test)
print(loss_metric)
image_number =27
plt.figure(figsize=(2,2))
plt.imshow(x_test[image_number][:,:,::-1])
predicted_label = pred = model.predict(x_test[image_number].reshape(1, 150, 150, 3))
print('predicted label is ',predicted_label.argmax())
```

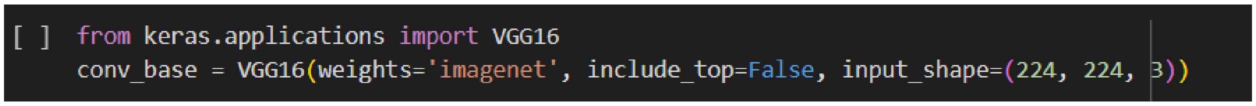
```
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 186ms/step - accuracy: 0.8417 - loss: 0.4817
[0.5117999911308289, 0.824999988079071]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step
predicted label is  0
```

Figure 19: Predicting labels from the testing dataset with Data Augmentation (5)

In terms of predicting, with data augmentation done first, the model only wrongly predicted 1 picture amongst the five with 84.17% accuracy, which is a clear improvement from the previous model.

Way 02: Transfer Learning with VGG

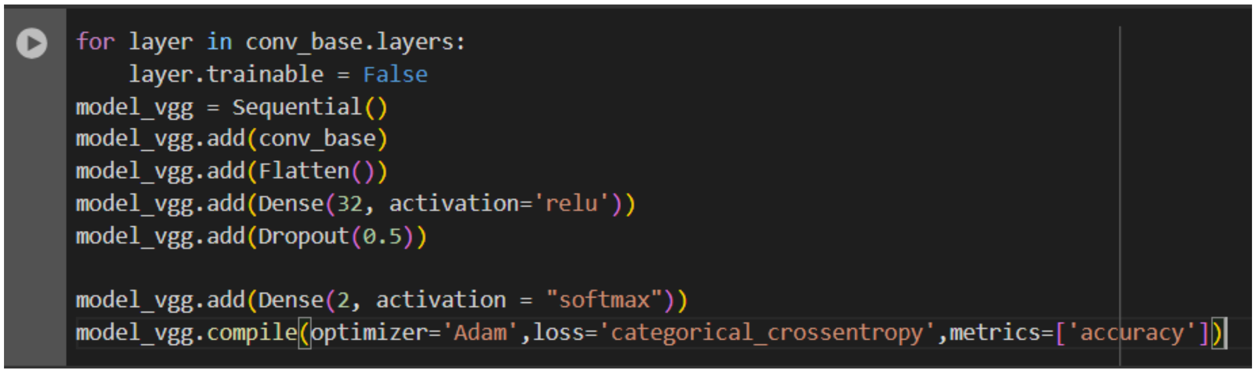
For transfer learning, I prepared the data like I did previously, only here I changed the resolution of the images to 244x244 to get a better grasp of the details by the model.



```
[ ]  from keras.applications import VGG16
     conv_base = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
```

Figure 20: Initiating the VGG model

Like Figure 20, I initiated the VGG model.



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

Figure 21: Building VGG for Transfer Learning

As shown in Figure 21, I prepared the VGG model for training. I added the dropout layer to prevent from overfitting. Then I trained the model with 10% validation split and generated the loss and accuracy curves.

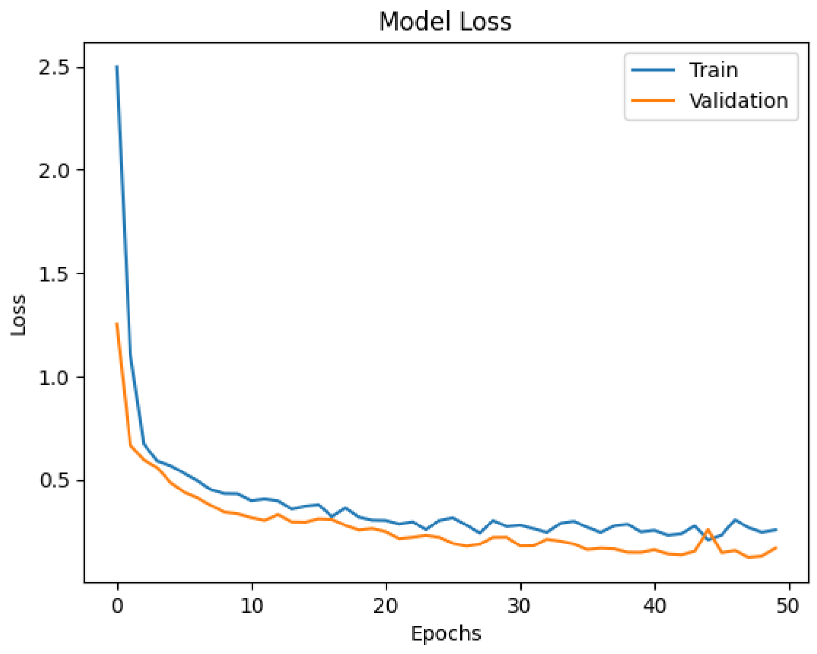


Figure 22: Loss Curve for Transfer Learning

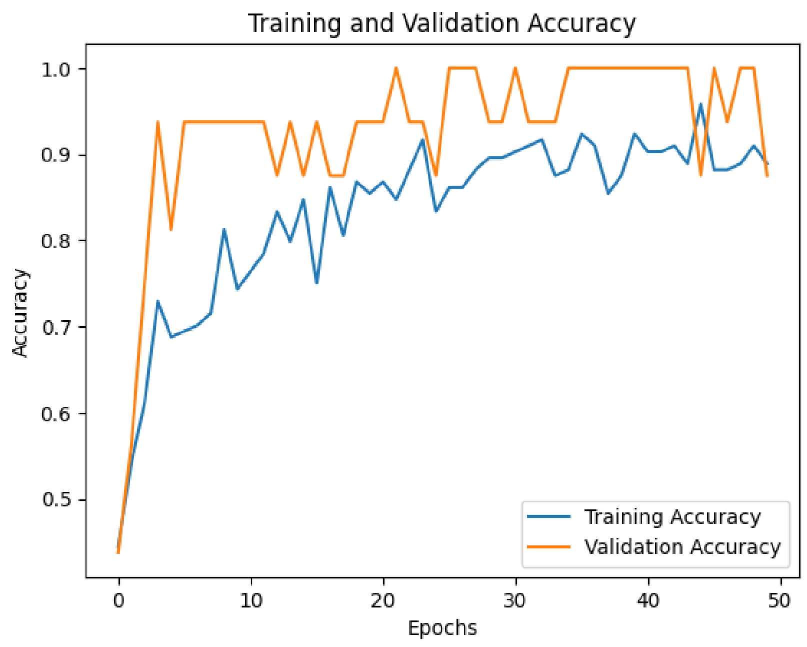
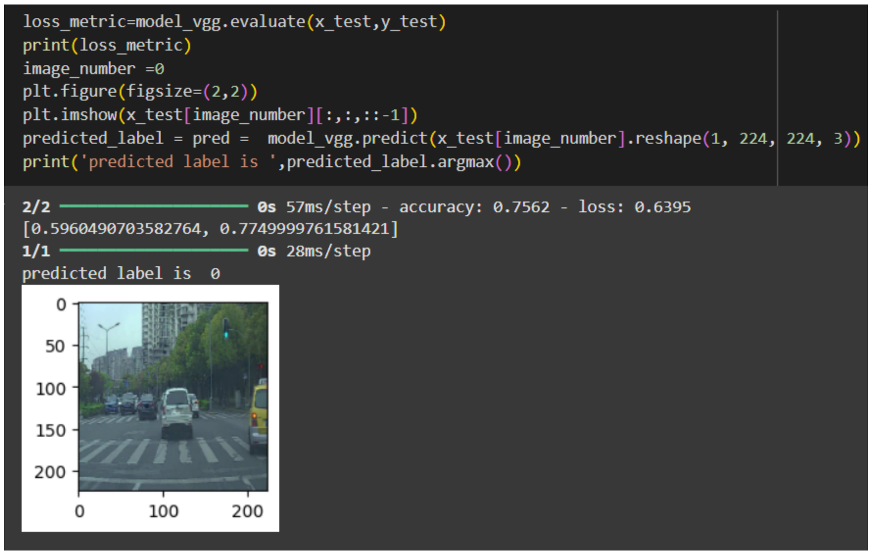


Figure 23: Accuracy Curve for Transfer Learning

From Figure 22 and Figure 23, it can be seen that the model is being learned on the higher epoch, while facing some issues in the last epochs, suggesting a possibility of overfitting. Although the model has done quite a good job on training, it struggles a little bit on the validation. But this sure is an improvement from the initial Conv2D model.

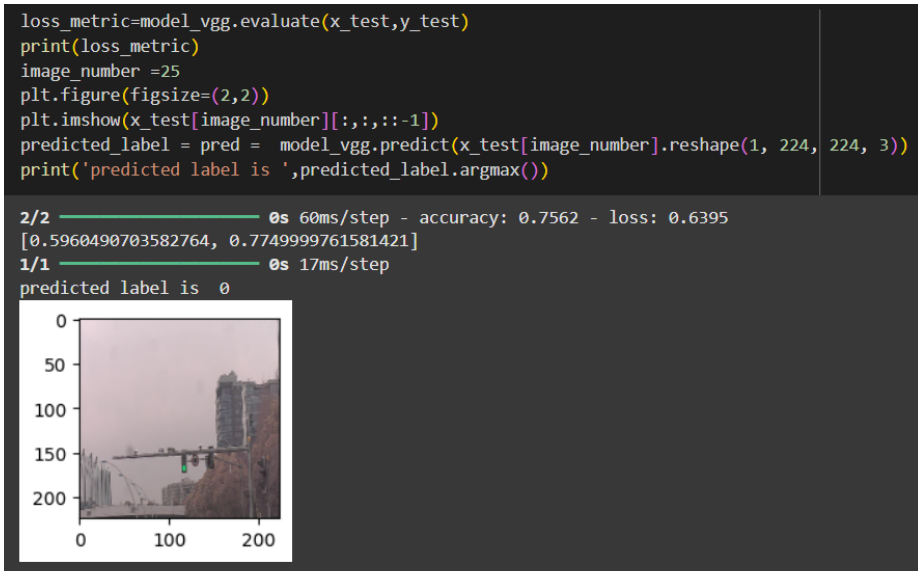
After this, I predicted pictures from the testing dataset shown in Figure 24, Figure 25, Figure 26, Figure 27, and Figure 28.



```
loss_metric=model_vgg.evaluate(x_test,y_test)
print(loss_metric)
image_number =0
plt.figure(figsize=(2,2))
plt.imshow(x_test[image_number][:,:,::-1])
predicted_label = pred =  model_vgg.predict(x_test[image_number].reshape(1, 224, 224, 3))
print('predicted label is ',predicted_label.argmax())
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - accuracy: 0.7562 - loss: 0.6395
[0.5960490703582764, 0.7749999761581421]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
predicted label is  0
```

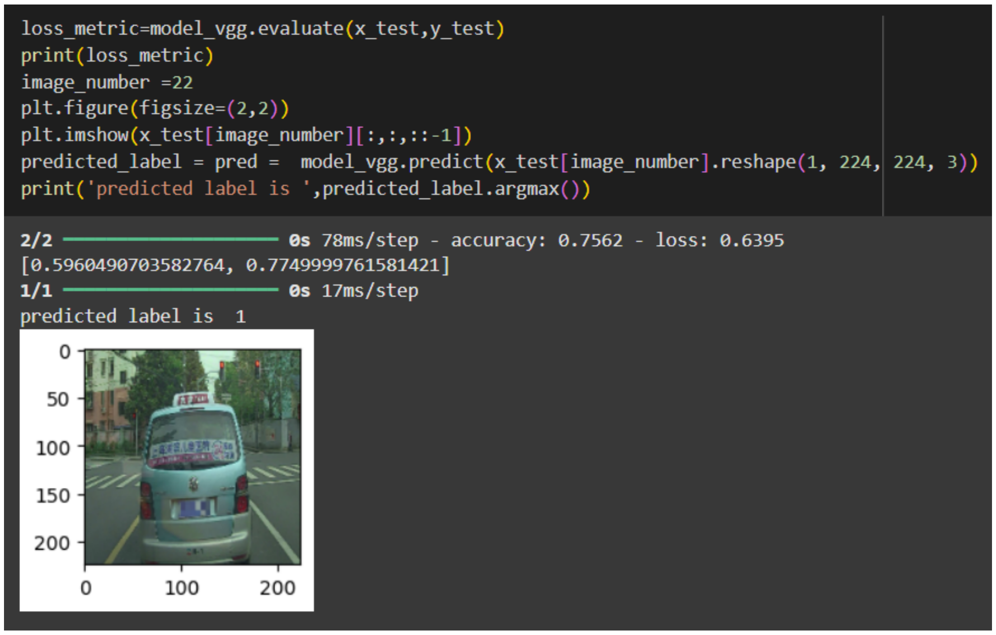
Figure 24: Predicting labels from the testing dataset with VGG (1)



```
loss_metric=model_vgg.evaluate(x_test,y_test)
print(loss_metric)
image_number =25
plt.figure(figsize=(2,2))
plt.imshow(x_test[image_number][:,:,::-1])
predicted_label = pred =  model_vgg.predict(x_test[image_number].reshape(1, 224, 224, 3))
print('predicted label is ',predicted_label.argmax())
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step - accuracy: 0.7562 - loss: 0.6395
[0.5960490703582764, 0.7749999761581421]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
predicted label is  0
```

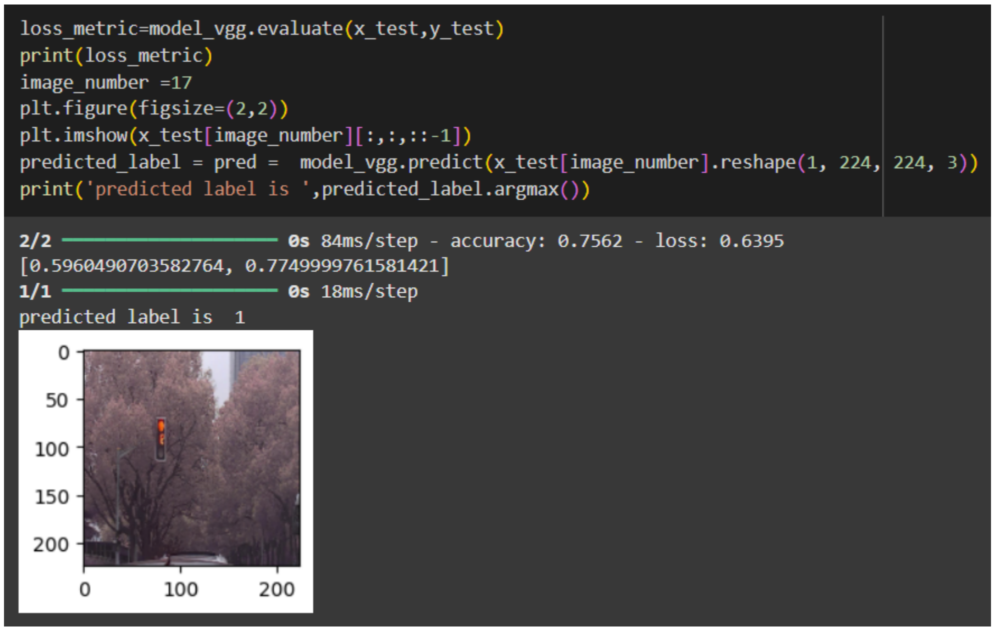
Figure 25: Predicting labels from the testing dataset with VGG (2)



```
loss_metric=model_vgg.evaluate(x_test,y_test)
print(loss_metric)
image_number =22
plt.figure(figsize=(2,2))
plt.imshow(x_test[image_number][:,:,::-1])
predicted_label = pred =  model_vgg.predict(x_test[image_number].reshape(1, 224, 224, 3))
print('predicted label is ',predicted_label.argmax())
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 78ms/step - accuracy: 0.7562 - loss: 0.6395
[0.5960490703582764, 0.7749999761581421]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
predicted label is  1
```

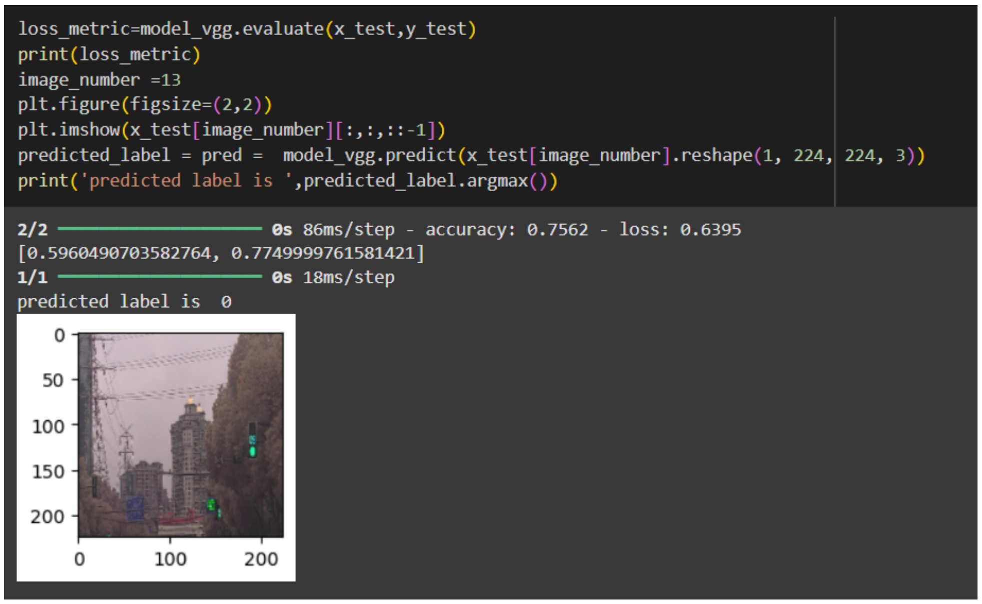
Figure 26: Predicting labels from the testing dataset with VGG (3)



```
loss_metric=model_vgg.evaluate(x_test,y_test)
print(loss_metric)
image_number =17
plt.figure(figsize=(2,2))
plt.imshow(x_test[image_number][:,:,::-1])
predicted_label = pred =  model_vgg.predict(x_test[image_number].reshape(1, 224, 224, 3))
print('predicted label is ',predicted_label.argmax())
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 84ms/step - accuracy: 0.7562 - loss: 0.6395
[0.5960490703582764, 0.7749999761581421]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
predicted label is  1
```

Figure 27: Predicting labels from the testing dataset with VGG (4)



```
loss_metric=model_vgg.evaluate(x_test,y_test)
print(loss_metric)
image_number =13
plt.figure(figsize=(2,2))
plt.imshow(x_test[image_number][:,:,::-1])
predicted_label = pred =  model_vgg.predict(x_test[image_number].reshape(1, 224, 224, 3))
print('predicted label is ',predicted_label.argmax())
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 86ms/step - accuracy: 0.7562 - loss: 0.6395
[0.5960490703582764, 0.7749999761581421]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
predicted label is  0
```

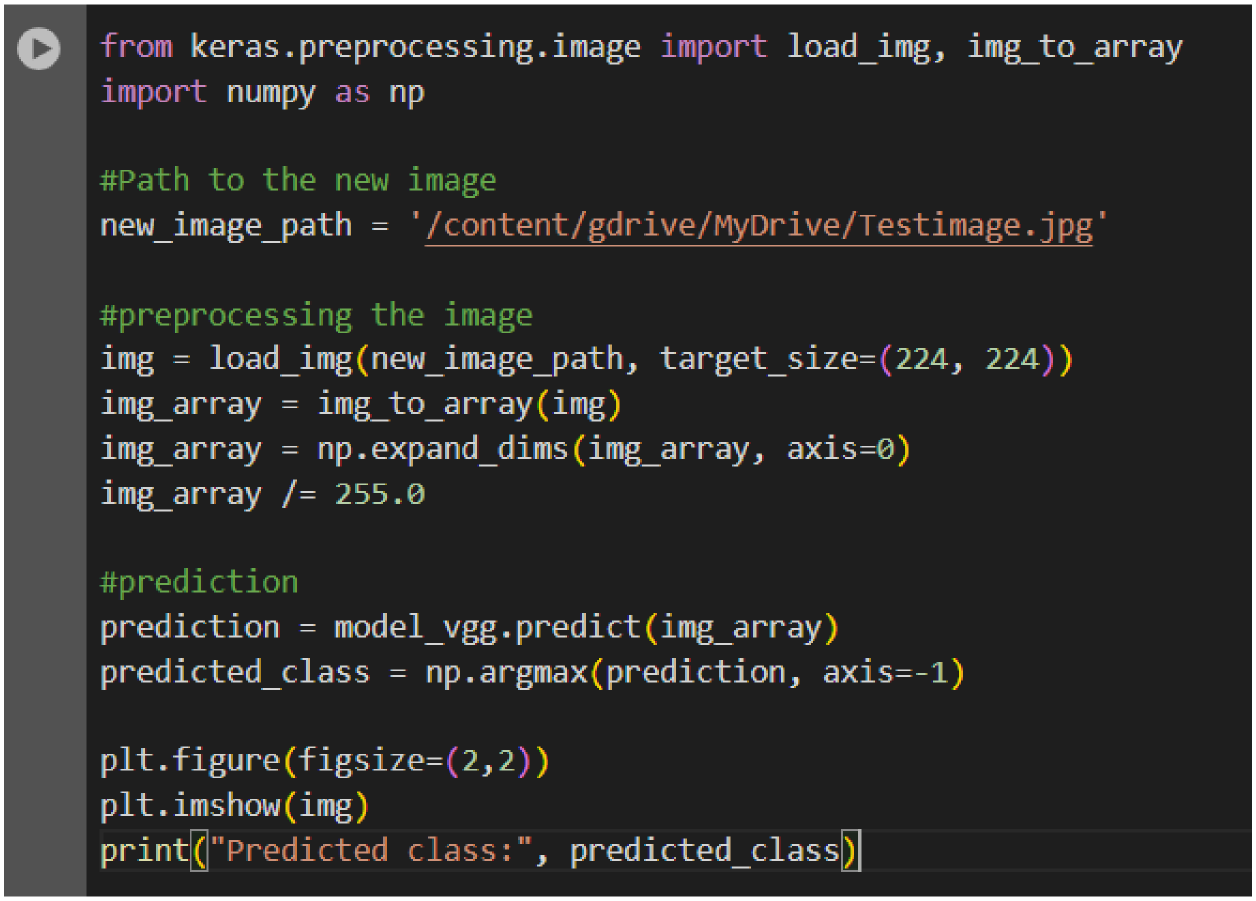
Figure 28: Predicting labels from the testing dataset with VGG (5)

From the above predictions, it is clear that, VGG predicted all the testing pictures correctly with an accuracy of 77.75% accuracy and VGG is the best model so far.

To further prove the generalization of the VGG model, I downloaded 5 images from the web and tested with the model for prediction.

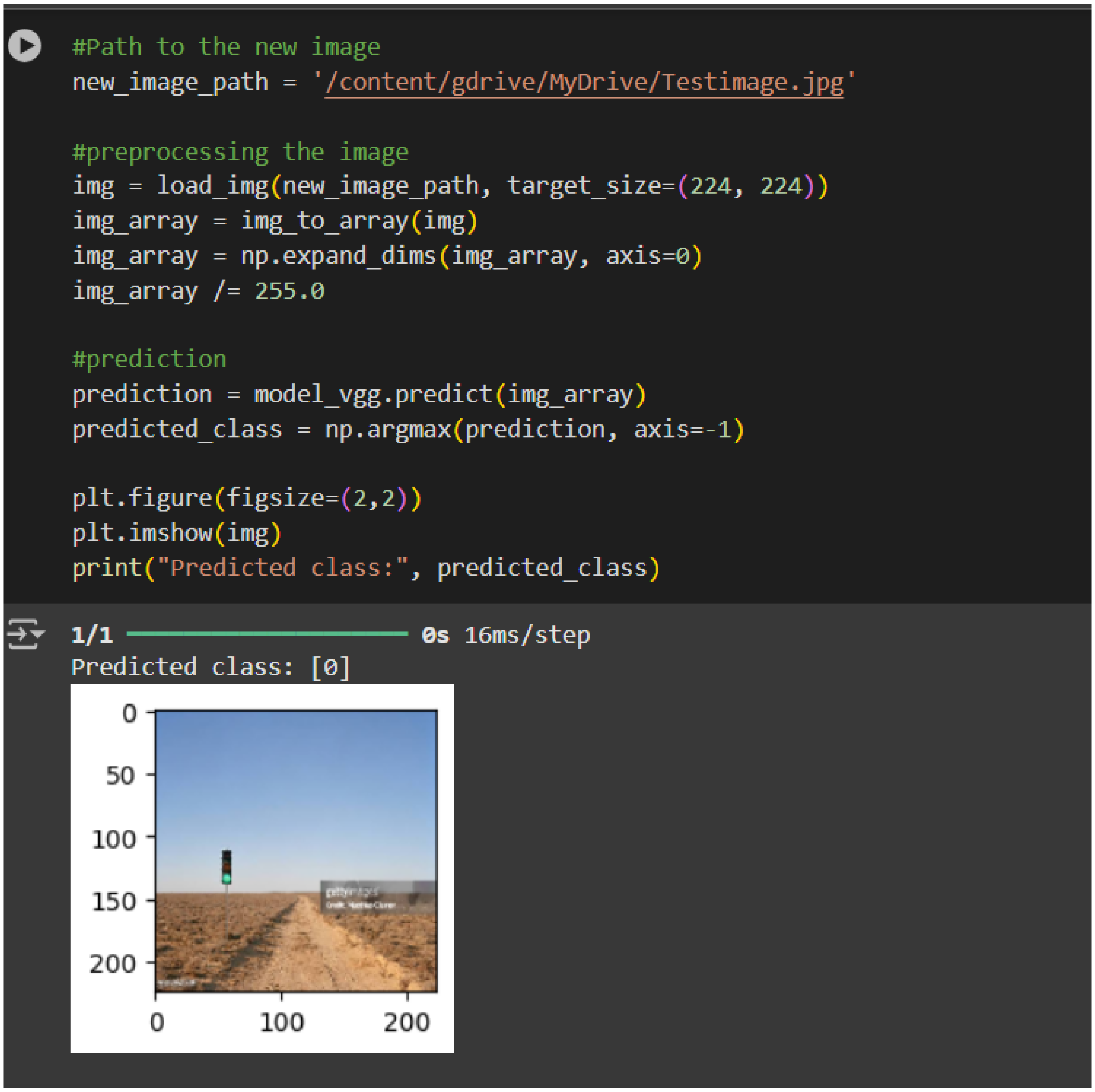
Showing the Generalization of the VGG using downloaded pictures from the Web *(EXTRA)*

I used this code snippet in Figure 29 for the picture in my google drive for predicting the images from web.



```
from keras.preprocessing.image import load_img, img_to_array
import numpy as np

#Path to the new image
new_image_path = '/content/gdrive/MyDrive/Testimage.jpg'

#preprocessing the image
img = load_img(new_image_path, target_size=(224, 224))
img_array = img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array /= 255.0

#prediction
prediction = model_vgg.predict(img_array)
predicted_class = np.argmax(prediction, axis=-1)

plt.figure(figsize=(2,2))
plt.imshow(img)
print("Predicted class:", predicted_class)
```

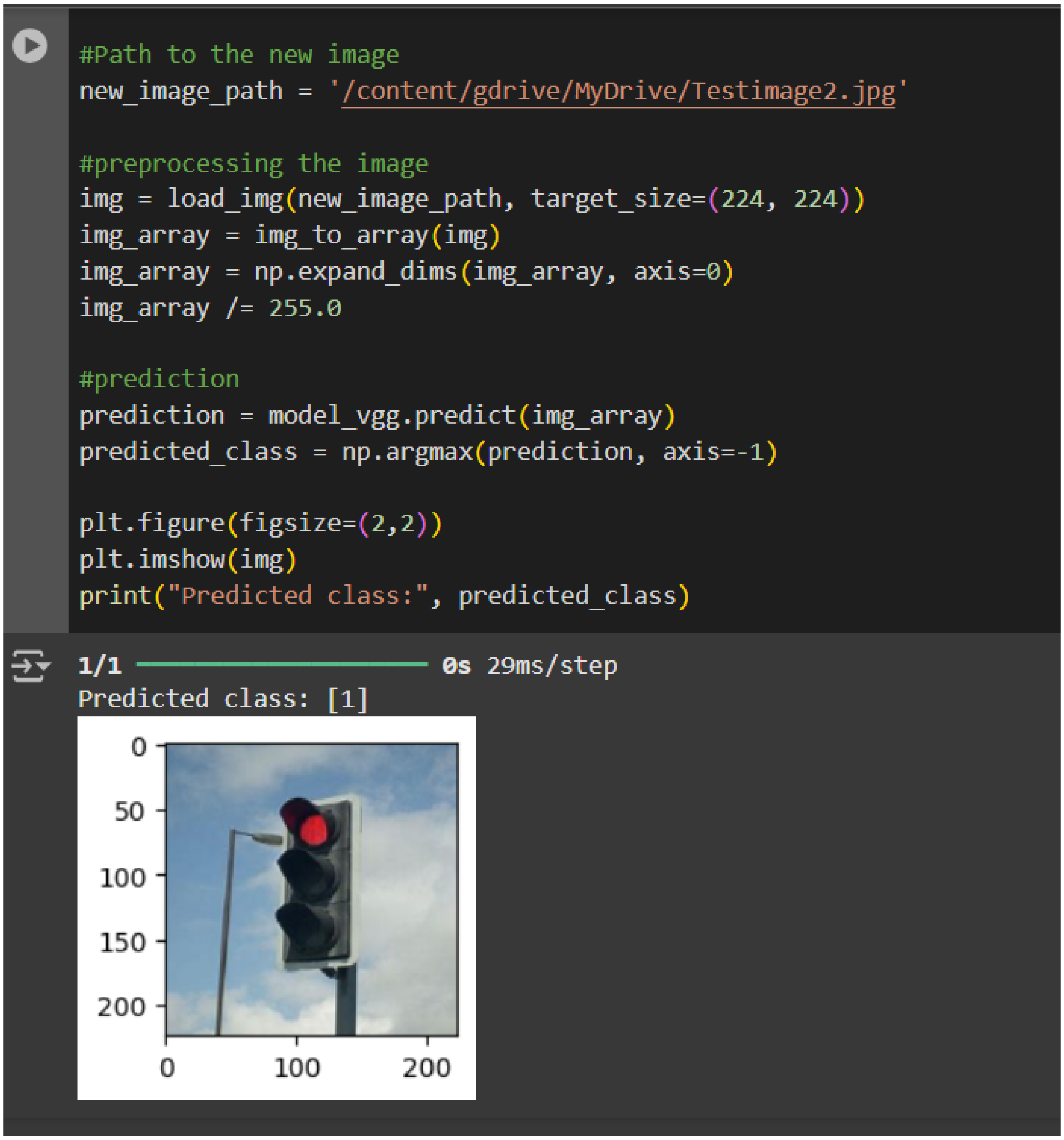
Figure 29: Predicting labels from the Web (Code)



```
#Path to the new image
new_image_path = '/content/gdrive/MyDrive/Testimage.jpg'

#preprocessing the image
img = load_img(new_image_path, target_size=(224, 224))
img_array = img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array /= 255.0

#prediction
prediction = model_vgg.predict(img_array)
predicted_class = np.argmax(prediction, axis=-1)

plt.figure(figsize=(2,2))
plt.imshow(img)
print("Predicted class:", predicted_class)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
Predicted class: [0]
```

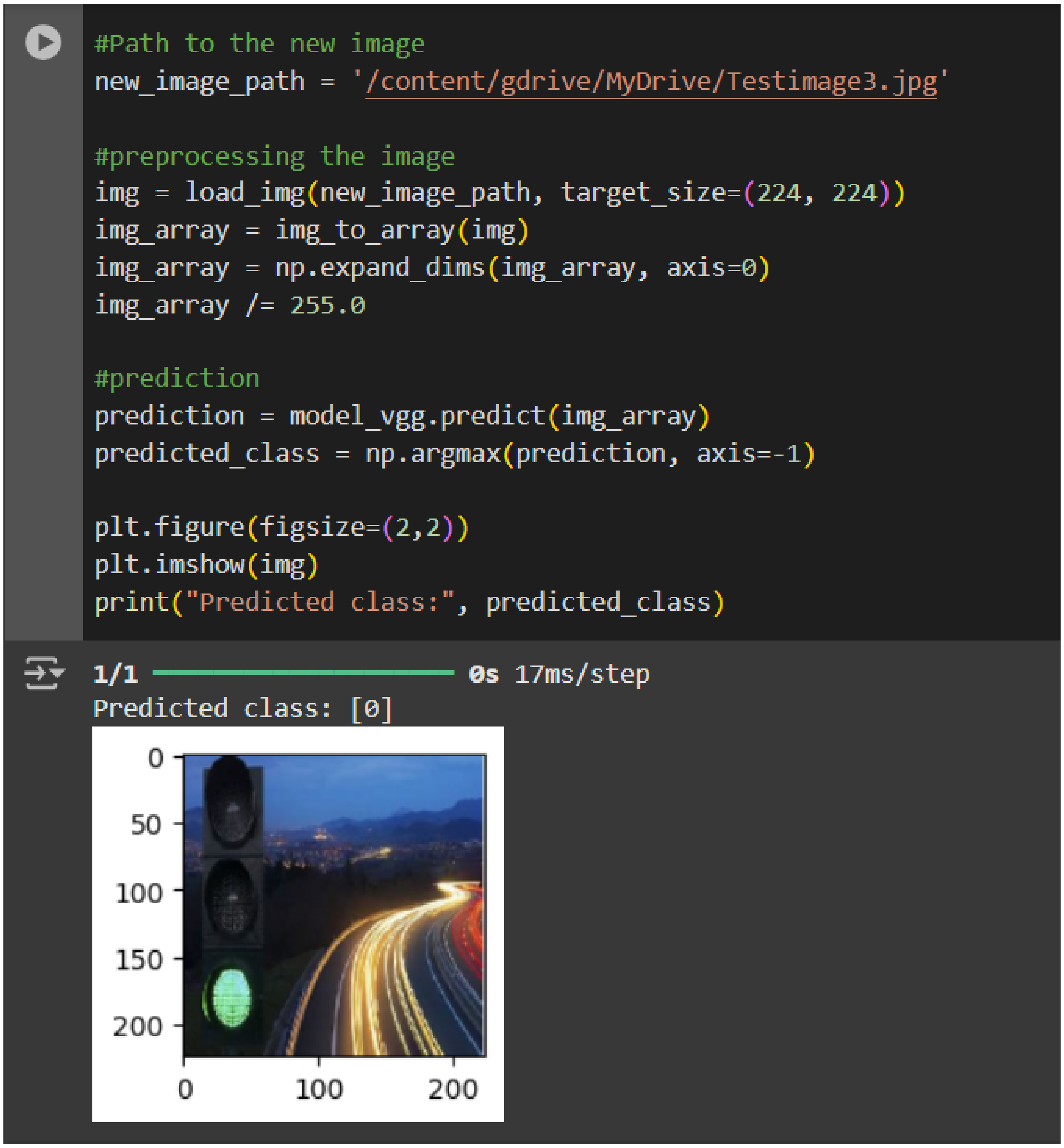
Figure 30: Predicting labels from the Web (1)



```python
#Path to the new image
new_image_path = '/content/gdrive/MyDrive/Testimage2.jpg'

#preprocessing the image
img = load_img(new_image_path, target_size=(224, 224))
img_array = img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array /= 255.0

#prediction
prediction = model_vgg.predict(img_array)
predicted_class = np.argmax(prediction, axis=-1)

plt.figure(figsize=(2,2))
plt.imshow(img)
print("Predicted class:", predicted_class)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
Predicted class: [1]
```

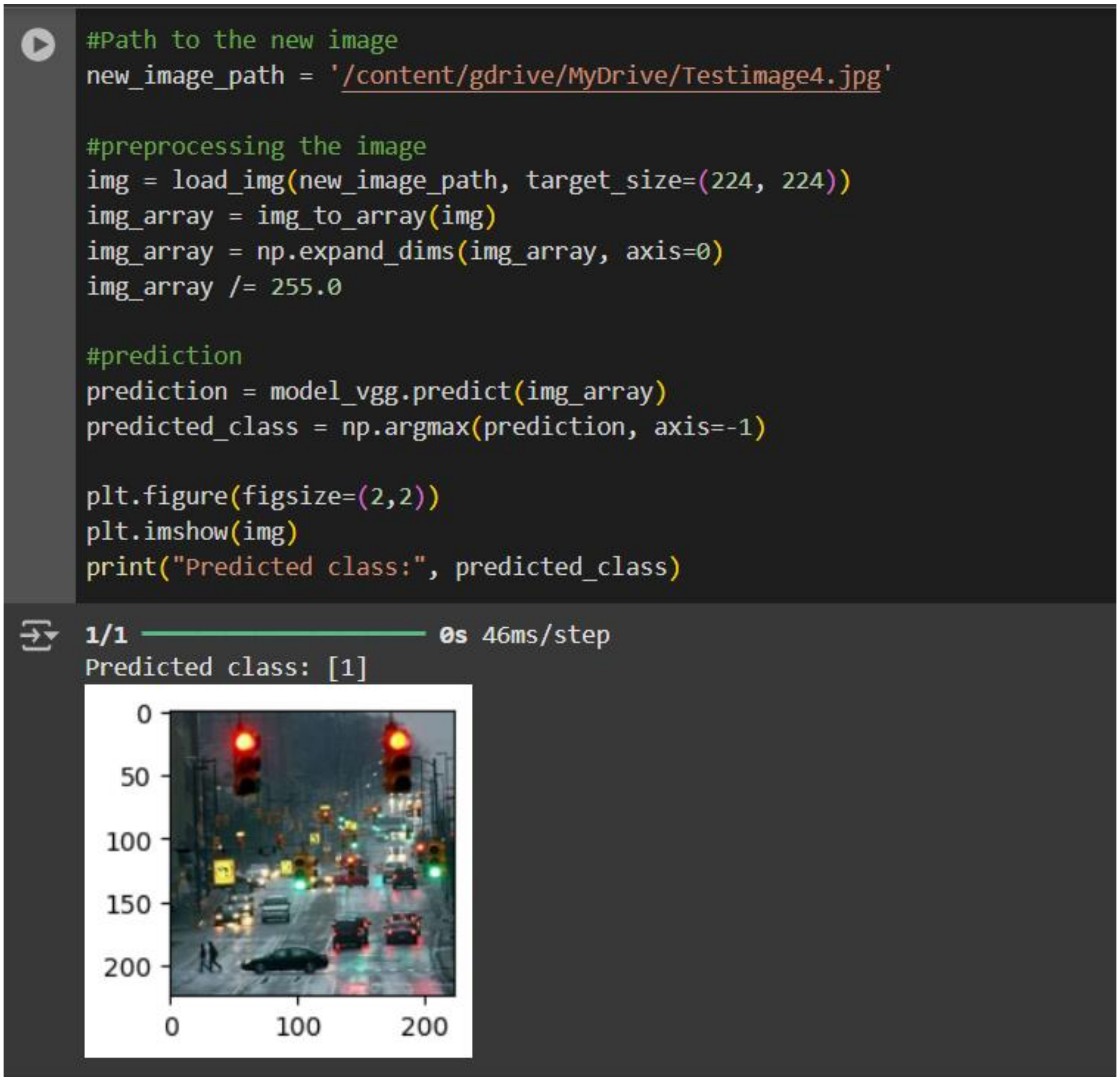
Figure 31: Predicting labels from the Web (2)



```
#Path to the new image
new_image_path = '/content/gdrive/MyDrive/Testimage3.jpg'

#preprocessing the image
img = load_img(new_image_path, target_size=(224, 224))
img_array = img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array /= 255.0

#prediction
prediction = model_vgg.predict(img_array)
predicted_class = np.argmax(prediction, axis=-1)

plt.figure(figsize=(2,2))
plt.imshow(img)
print("Predicted class:", predicted_class)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
Predicted class: [0]
```

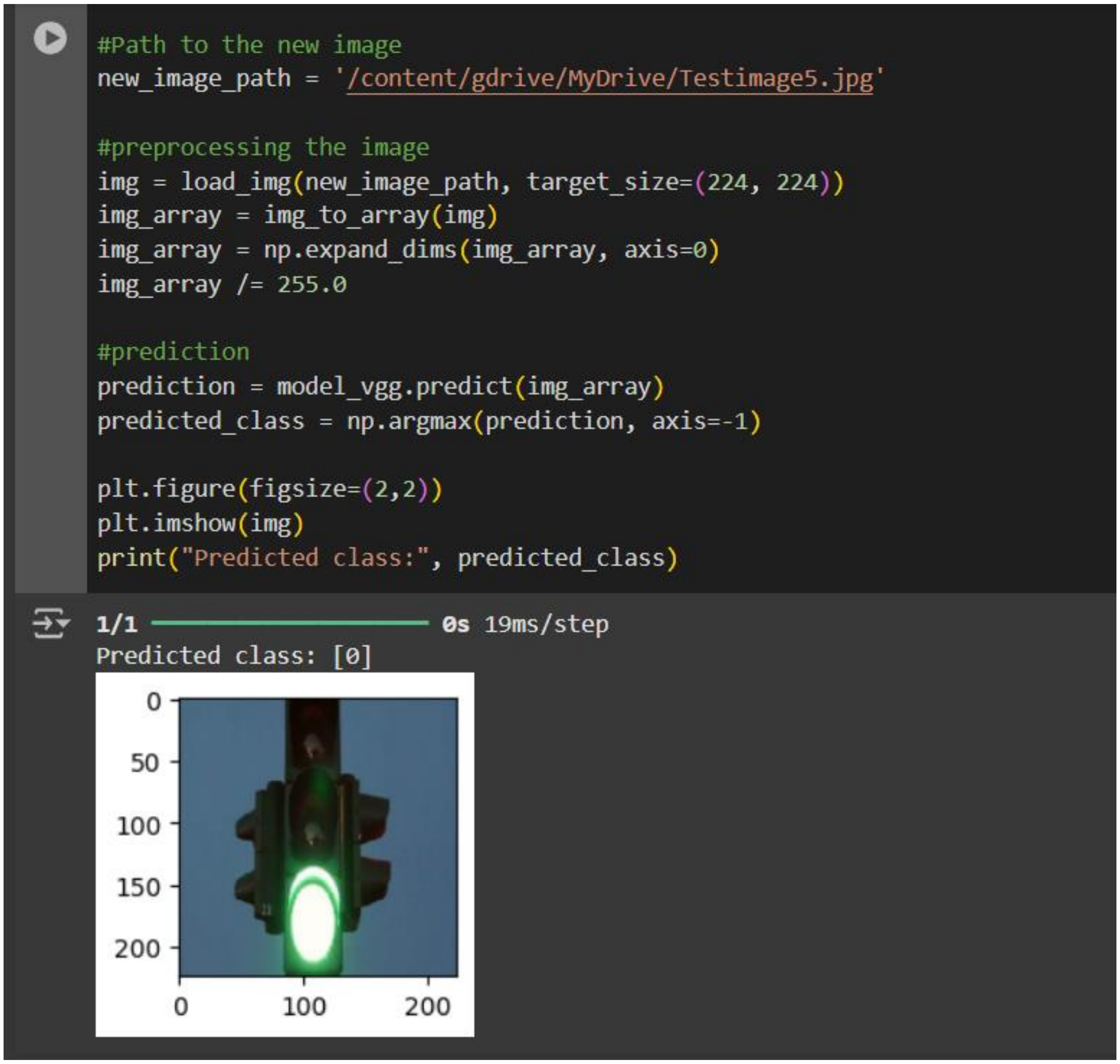
Figure 32: Predicting labels from the Web (3)



```
#Path to the new image
new_image_path = '/content/gdrive/MyDrive/Testimage4.jpg'

#preprocessing the image
img = load_img(new_image_path, target_size=(224, 224))
img_array = img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array /= 255.0

#prediction
prediction = model_vgg.predict(img_array)
predicted_class = np.argmax(prediction, axis=-1)

plt.figure(figsize=(2,2))
plt.imshow(img)
print("Predicted class:", predicted_class)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step
Predicted class: [1]
```

Figure 33: Predicting labels from the Web (4)



```
#Path to the new image
new_image_path = '/content/gdrive/MyDrive/Testimage5.jpg'

#preprocessing the image
img = load_img(new_image_path, target_size=(224, 224))
img_array = img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array /= 255.0

#prediction
prediction = model_vgg.predict(img_array)
predicted_class = np.argmax(prediction, axis=-1)

plt.figure(figsize=(2,2))
plt.imshow(img)
print("Predicted class:", predicted_class)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
Predicted class: [0]
```

Figure 34: Predicting labels from the Web (5)

As shown in Figure 29, Figure 30, Figure 31, Figure 32, Figure 33, and Figure 34, the model predicted all the images correctly although they were not that much easy.