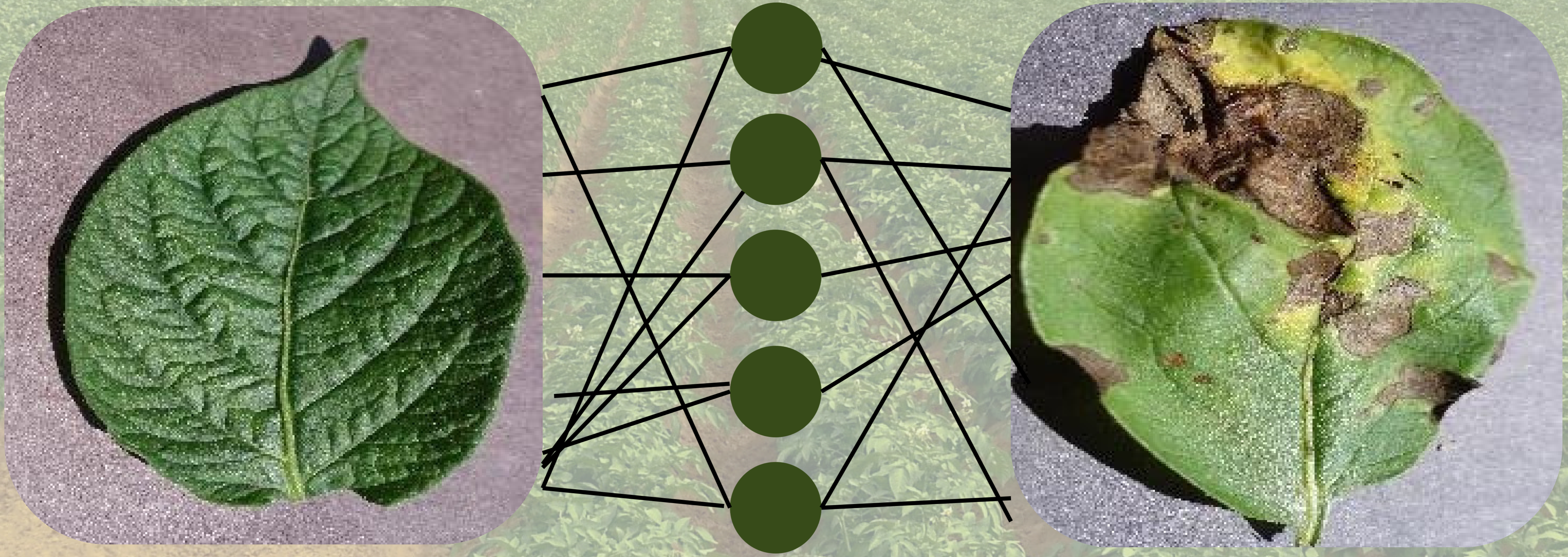
Plant Disease Classification using CNN

Early Blight and Late Blight is a common disease found in plants like tomato and potato, using Convolution Neural Network, we are trying to predit this disease by analyzing the leaves of a plant and classifying them into three differnt class.

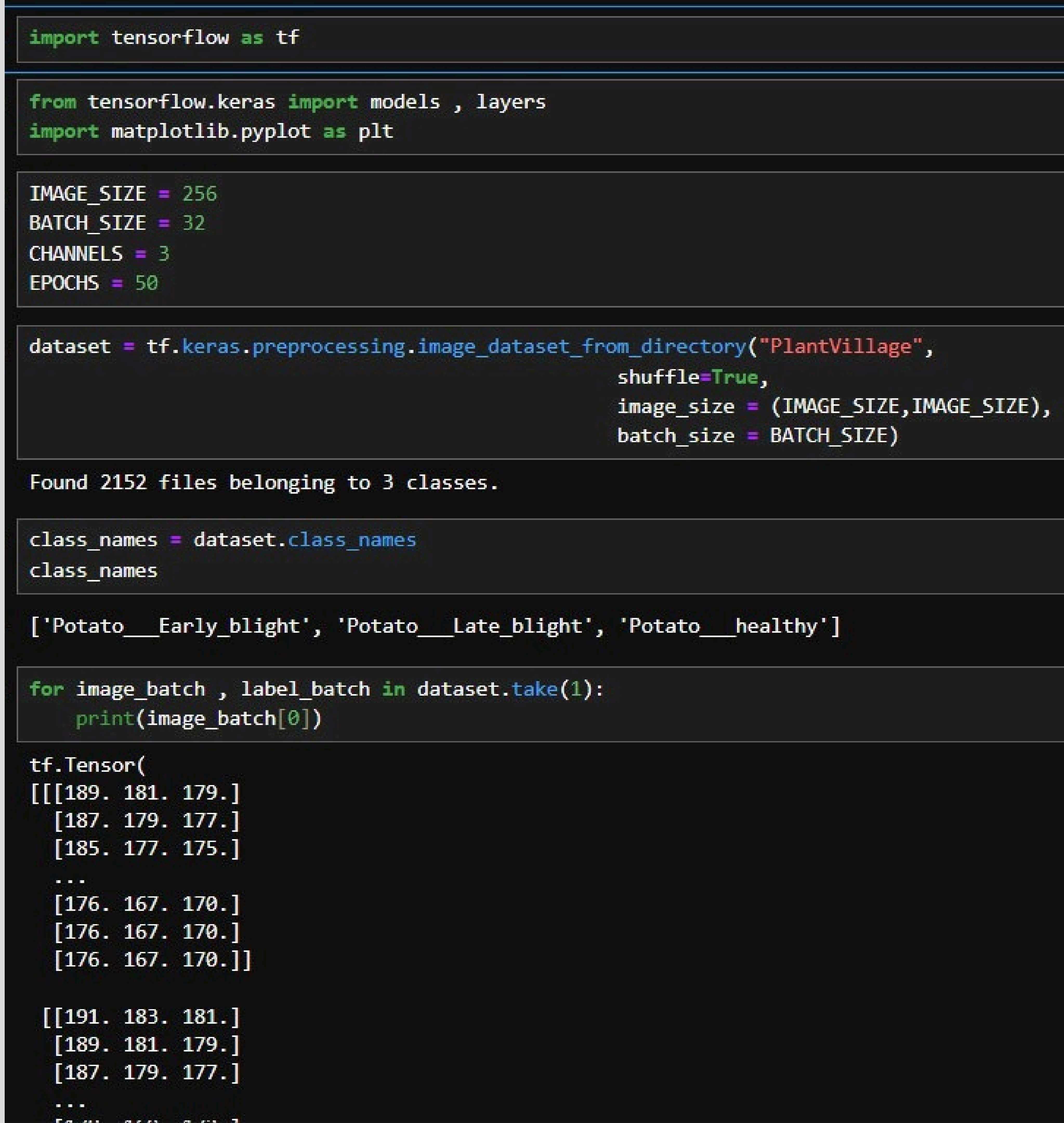


Tools that are used:

- Tensorflow
- Keras
- Python
- Kaggle
- Google Cloud Platform(GCP)
- Gradio

In the initial part of the we’ve imported all the essential libraries dataset from our directory(PlantVillage) and checked our Data Sample by printing it which currently is in the form of Tensor.

After performing some basic EDA , splitting of the data we tried to introduce cache and prefetch for better time optimization.



```
import tensorflow as tf
```

```
from tensorflow.keras import models , layers
import matplotlib.pyplot as plt
```

```
IMAGE_SIZE = 256
BATCH_SIZE = 32
CHANNELS = 3
EPOCHS = 50
```

```
dataset = tf.keras.preprocessing.image_dataset_from_directory("PlantVillage",
                                                              shuffle=True,
                                                              image_size = (IMAGE_SIZE,IMAGE_SIZE),
                                                              batch_size = BATCH_SIZE)
```

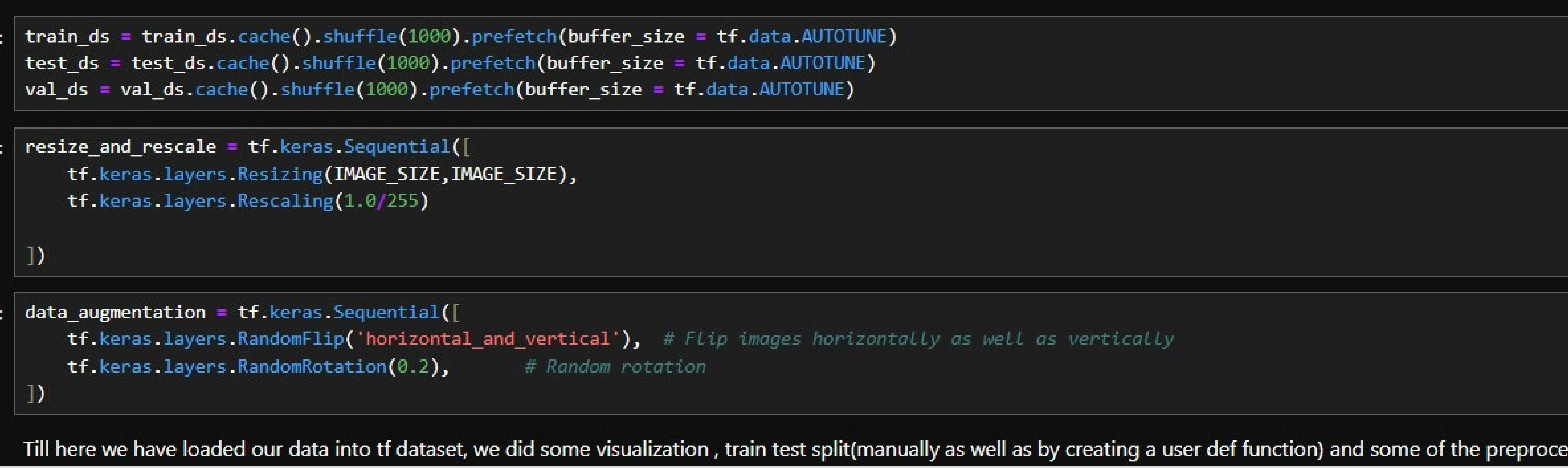
```
Found 2152 files belonging to 3 classes.
```

```
class_names = dataset.class_names
class_names
```

```
['Potato___Early_blight', 'Potato___Late_blight', 'Potato___healthy']
```

```
for image_batch , label_batch in dataset.take(1):
    print(image_batch[0])
```

```
tf.Tensor(
[[[189. 181. 179.]
  [187. 179. 177.]
  [185. 177. 175.]
  ...
  [176. 167. 170.]
  [176. 167. 170.]
  [176. 167. 170.]]

 [[191. 183. 181.]
  [189. 181. 179.]
  [187. 179. 177.]
  ...
```



```
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size = tf.data.AUTOTUNE)
test_ds = test_ds.cache().shuffle(1000).prefetch(buffer_size = tf.data.AUTOTUNE)
val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size = tf.data.AUTOTUNE)
```

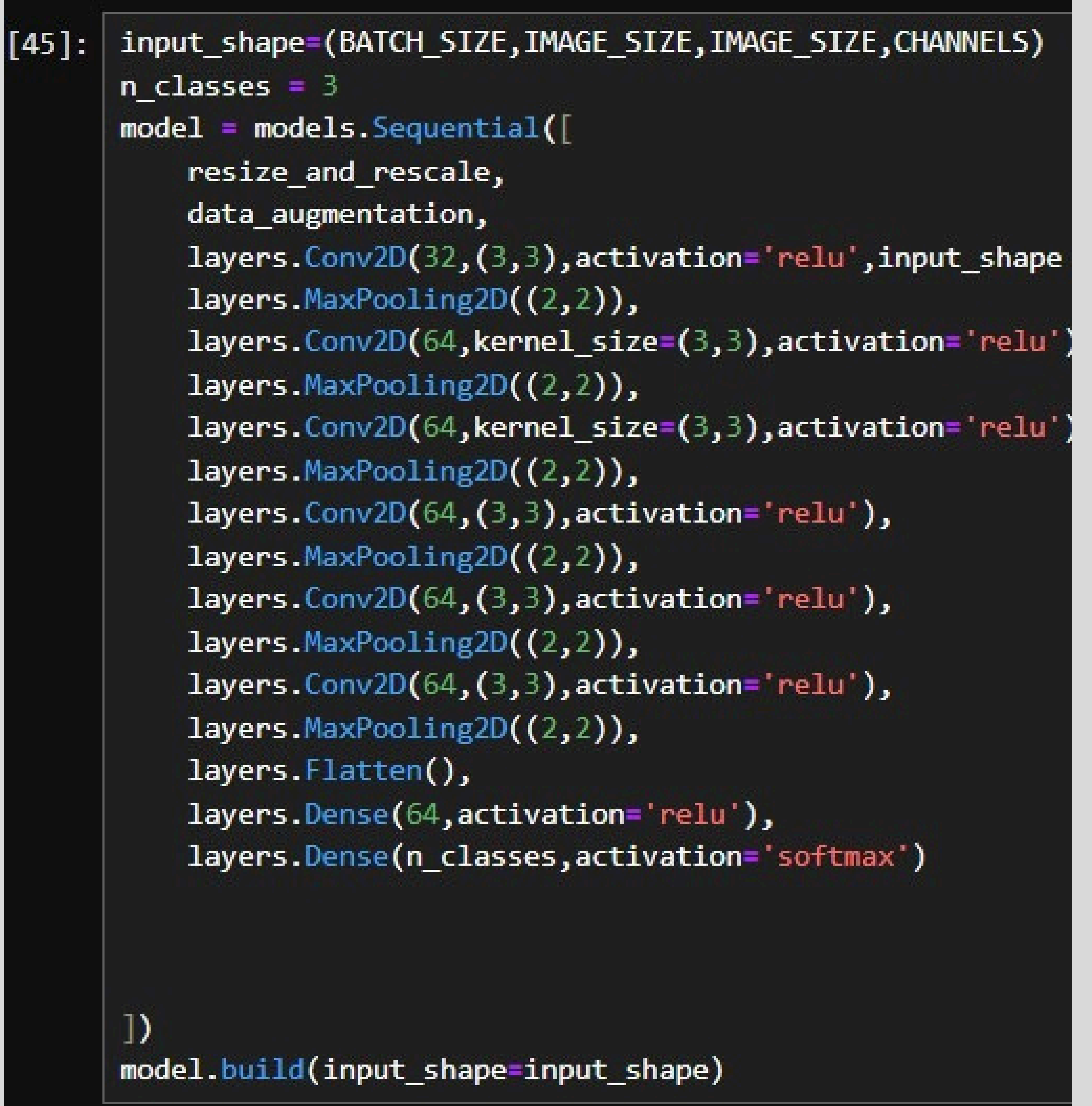
```
resize_and_rescale = tf.keras.Sequential([
    tf.keras.layers.Resizing(IMAGE_SIZE,IMAGE_SIZE),
    tf.keras.layers.Rescaling(1.0/255)


])
```

```
data_augmentation = tf.keras.Sequential([
    tf.keras.layers.RandomFlip('horizontal_and_vertical'),  # Flip images horizontally as well as vertically
    tf.keras.layers.RandomRotation(0.2),       # Random rotation
])
```

Till here we have loaded our data into tf dataset, we did some visualization , train test split(manually as well as by creating a user def function) and some of the preproce

Here we have used cache and prefetch method to improve the efficiency and time complexity of the model.
Data Augmentation is the method where we do little bit of changes like flip and rotation to the original data samples to increase the robustness and generate and extra sample, for the same purpose we can use ImageDataGenerator.

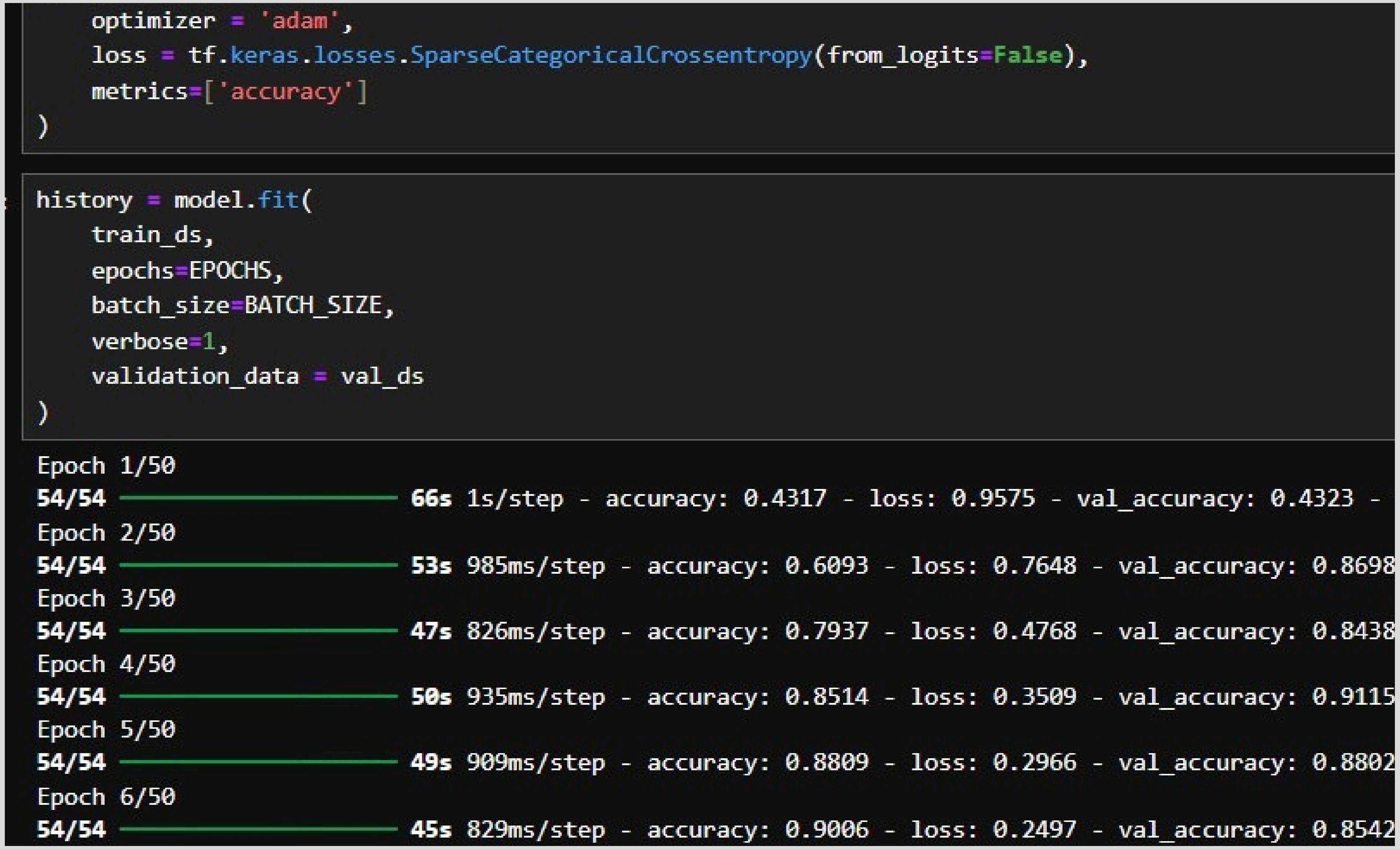
After performing some EDA, we specified our layers that are to be present in our model.
Layers parameters specifies the number of neurons per layer, activation Function which in this case is ReLu for all the hidden layer and Softmax for the output Layer.



```
[45]: input_shape=(BATCH_SIZE,IMAGE_SIZE,IMAGE_SIZE,CHANNELS)
      n_classes = 3
      model = models.Sequential([
          resize_and_rescale,
          data_augmentation,
          layers.Conv2D(32,(3,3),activation='relu',input_shape
          layers.MaxPooling2D((2,2)),
          layers.Conv2D(64,kernel_size=(3,3),activation='relu')
          layers.MaxPooling2D((2,2)),
          layers.Conv2D(64,kernel_size=(3,3),activation='relu')
          layers.MaxPooling2D((2,2)),
          layers.Conv2D(64,(3,3),activation='relu'),
          layers.MaxPooling2D((2,2)),
          layers.Conv2D(64,(3,3),activation='relu'),
          layers.MaxPooling2D((2,2)),
          layers.Conv2D(64,(3,3),activation='relu'),
          layers.MaxPooling2D((2,2)),
          layers.Flatten(),
          layers.Dense(64,activation='relu'),
          layers.Dense(n_classes,activation='softmax')



      ])
      model.build(input_shape=input_shape)
```

In this block of code we specified the optimizer that we are using which is Adam in our case and mentioned the type of loss which will be used to optimize our model.

By fitting the model we state the dataset, Epochs, batch size, verbose, validation data.

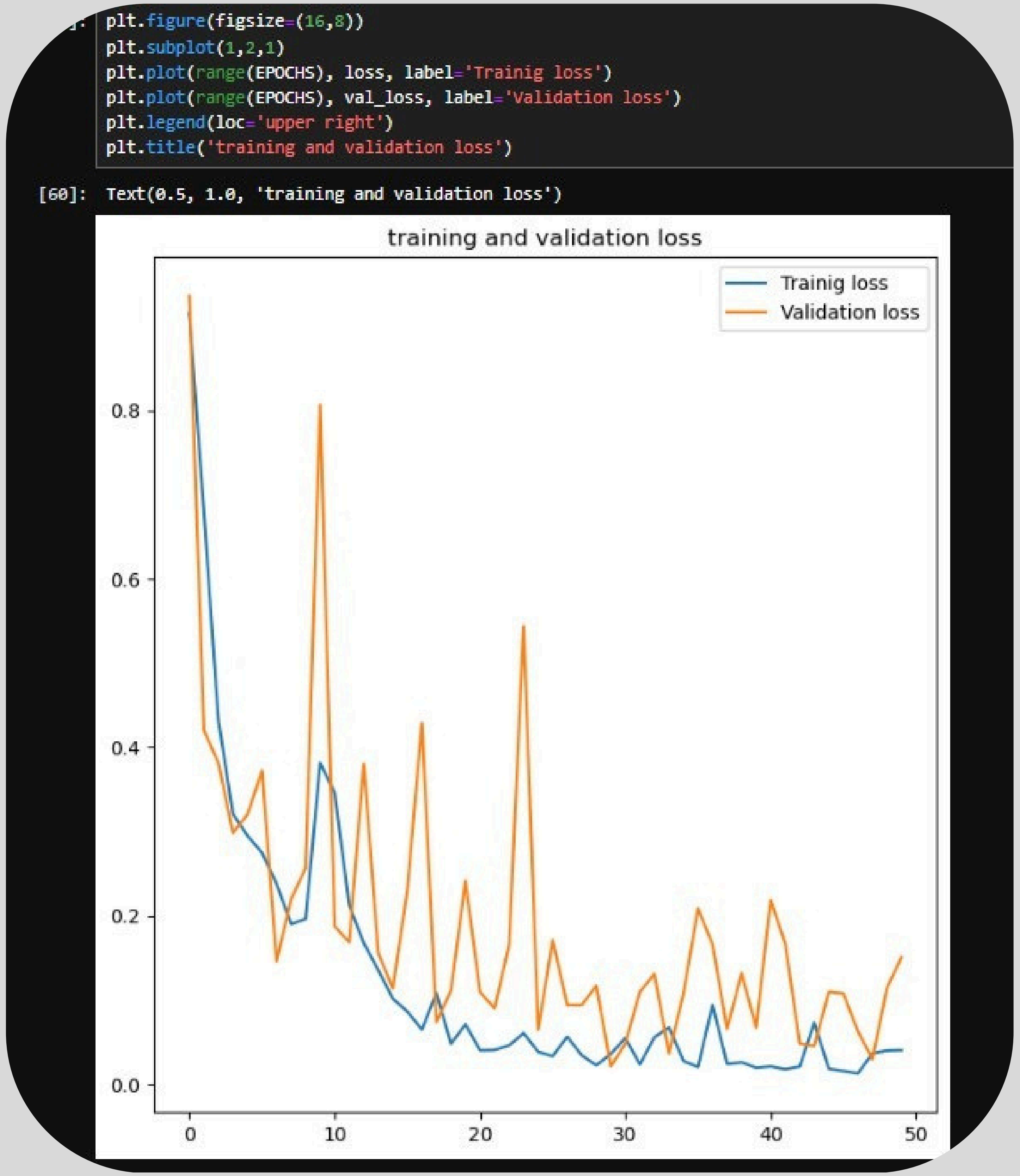


```
    optimizer = 'adam',
    loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)
```

```
history = model.fit(
    train_ds,
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    verbose=1,
    validation_data = val_ds
)
```

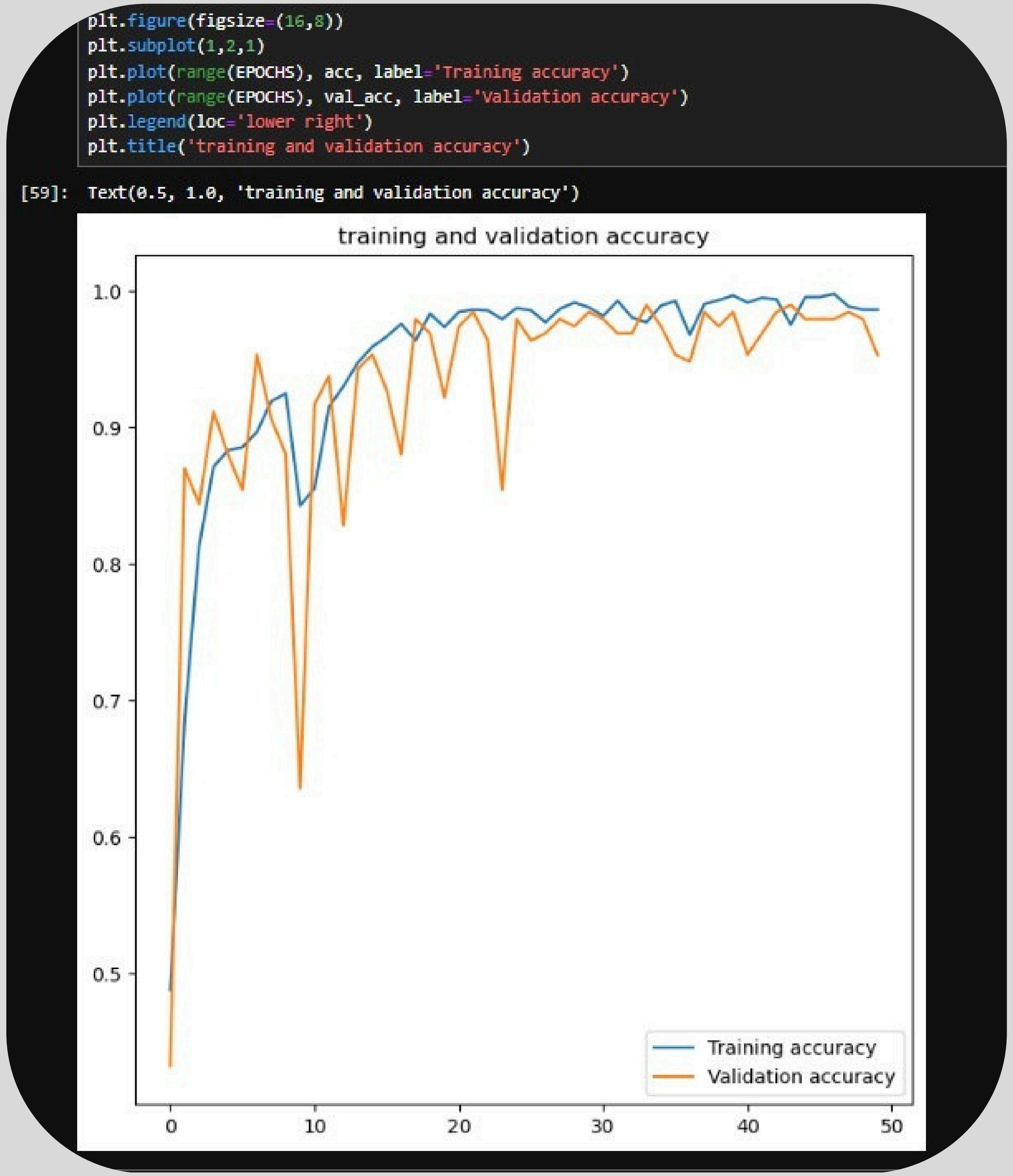
```
Epoch 1/50
54/54 ━━━━━━━━━━━━━━━━━━━━ 66s 1s/step - accuracy: 0.4317 - loss: 0.9575 - val_accuracy: 0.4323 -
Epoch 2/50
54/54 ━━━━━━━━━━━━━━━━━━━━ 53s 985ms/step - accuracy: 0.6093 - loss: 0.7648 - val_accuracy: 0.8698
Epoch 3/50
54/54 ━━━━━━━━━━━━━━━━━━━━ 47s 826ms/step - accuracy: 0.7937 - loss: 0.4768 - val_accuracy: 0.8438
Epoch 4/50
54/54 ━━━━━━━━━━━━━━━━━━━━ 50s 935ms/step - accuracy: 0.8514 - loss: 0.3509 - val_accuracy: 0.9115
Epoch 5/50
54/54 ━━━━━━━━━━━━━━━━━━━━ 49s 909ms/step - accuracy: 0.8809 - loss: 0.2966 - val_accuracy: 0.8802
Epoch 6/50
54/54 ━━━━━━━━━━━━━━━━━━━━ 45s 829ms/step - accuracy: 0.9006 - loss: 0.2497 - val_accuracy: 0.8542
```

Training/validation "Loss" VS Training/validation "accuracy"



```
plt.figure(figsize=(16,8))
plt.subplot(1,2,1)
plt.plot(range(EPOCHS), loss, label='Trainig loss')
plt.plot(range(EPOCHS), val_loss, label='Validation loss')
plt.legend(loc='upper right')
plt.title('training and validation loss')
```

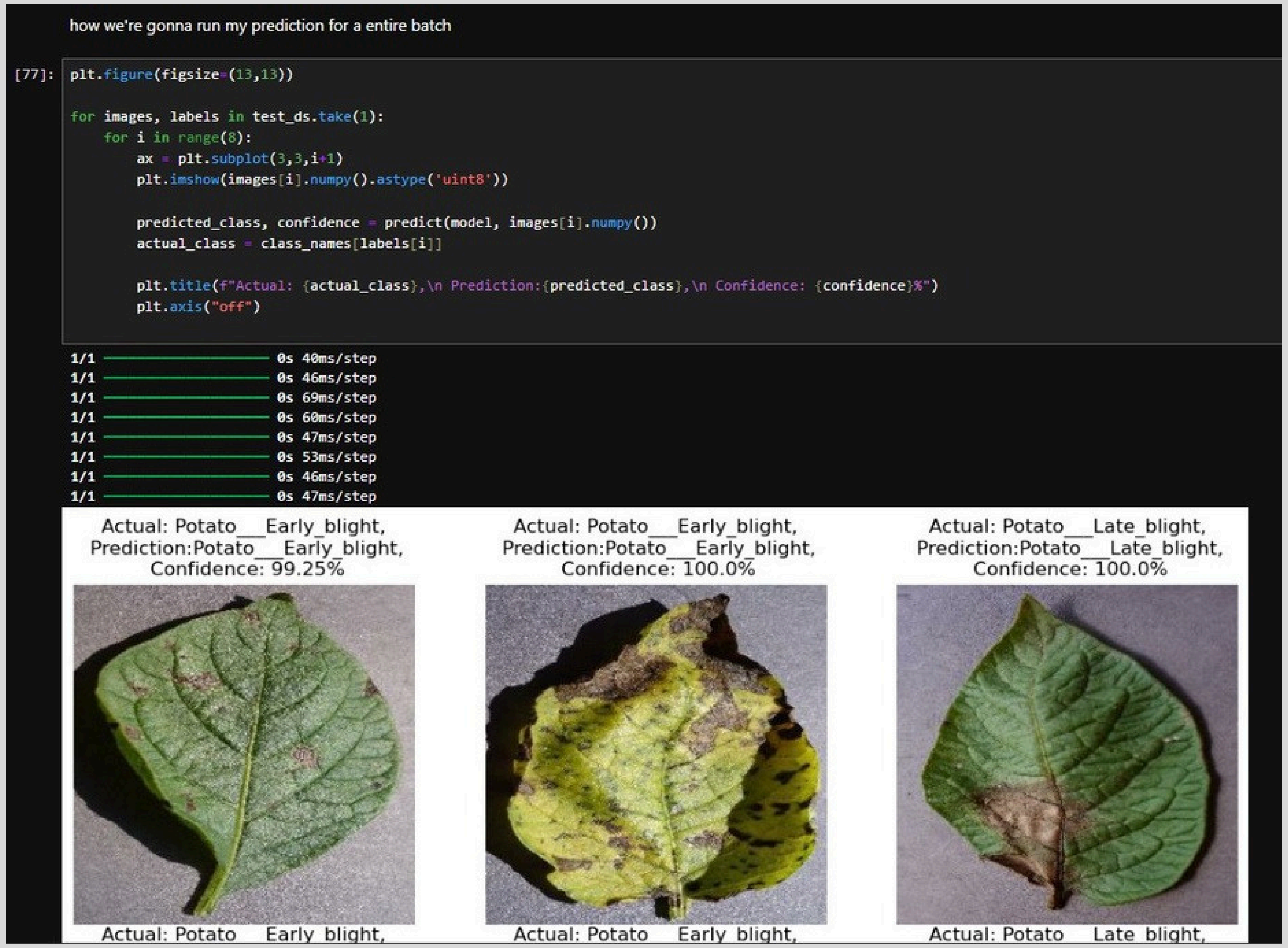
```
[60]:  Text(0.5, 1.0, 'training and validation loss')
```



```
plt.figure(figsize=(16,8))
plt.subplot(1,2,1)
plt.plot(range(EPOCHS), acc, label='Training accuracy')
plt.plot(range(EPOCHS), val_acc, label='Validation accuracy')
plt.legend(loc='lower right')
plt.title('training and validation accuracy')
```

```
[59]:  Text(0.5, 1.0, 'training and validation accuracy')
```

Prediction of Entire Batch

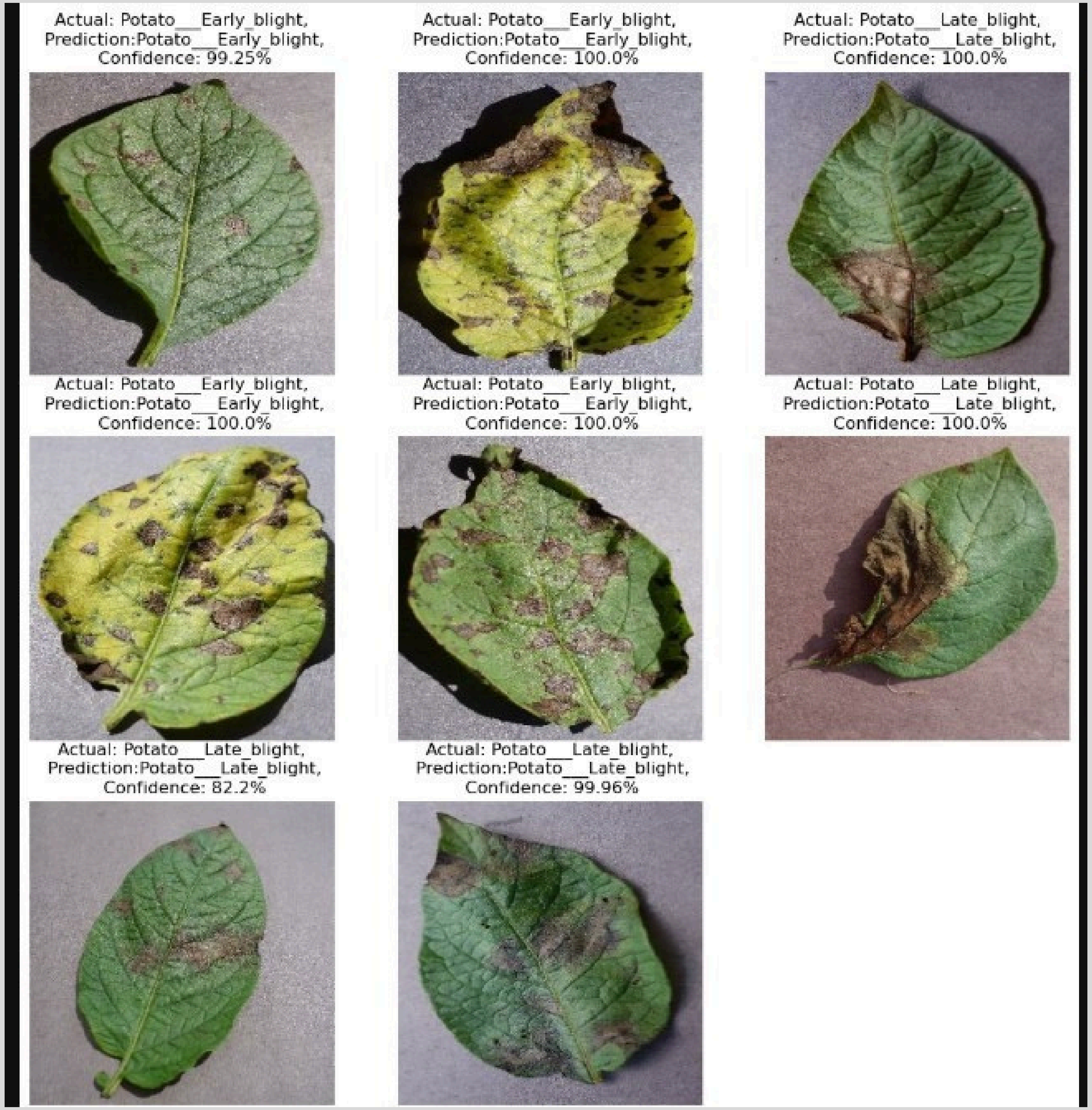


how we're gonna run my prediction for a entire batch

```
[77]: plt.figure(figsize=(13,13))

      for images, labels in test_ds.take(1):
          for i in range(8):
              ax = plt.subplot(3,3,i+1)
              plt.imshow(images[i].numpy().astype('uint8'))

              predicted_class, confidence = predict(model, images[i].numpy())
              actual_class = class_names[labels[i]]

              plt.title(f"Actual: {actual_class},\n Prediction:{predicted_class},\n Confidence: {confidence}%")
              plt.axis("off")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 60ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
```

Actual: Potato___Early_blight, Prediction:Potato___Early_blight, Confidence: 99.25%
Actual: Potato___Early_blight, Prediction:Potato___Early_blight, Confidence: 100.0%
Actual: Potato___Late_blight, Prediction:Potato___Late_blight, Confidence: 100.0%
Actual: Potato___Early_blight,
Actual: Potato___Early_blight,
Actual: Potato___Late_blight,



Actual: Potato___Early_blight, Prediction:Potato___Early_blight, Confidence: 99.25%
Actual: Potato___Early_blight, Prediction:Potato___Early_blight, Confidence: 100.0%
Actual: Potato___Late_blight, Prediction:Potato___Late_blight, Confidence: 100.0%
Actual: Potato___Early_blight, Prediction:Potato___Early_blight, Confidence: 100.0%
Actual: Potato___Early_blight, Prediction:Potato___Early_blight, Confidence: 100.0%
Actual: Potato___Late_blight, Prediction:Potato___Late_blight, Confidence: 100.0%
Actual: Potato___Late_blight, Prediction:Potato___Late_blight, Confidence: 82.2%
Actual: Potato___Late_blight, Prediction:Potato___Late_blight, Confidence: 99.96%

lets save our model:

```
model_version = 1
model.save(f"../models/{model_version}.keras")
```

Purpose of the project:

- **Early Disease Detection** – Farmers can identify plant diseases before they spread, reducing crop losses and ensuring healthier yields.
- **Reduced Dependence on Pesticide**s – Accurate disease identification means targeted treatments, cutting down unnecessary pesticide use and saving costs.
- **Optimized Crop Management** – With real-time disease insights, farmers can adapt their farming practices to prevent outbreaks and improve productivity.
- **Higher Yields & Profits** – Healthier plants result in better yields, leading to increased profitability in agricultural markets.
- **Sustainable Farming Practices** – Efficient disease management supports eco-friendly farming by reducing waste and chemical exposure.

Presented by:

Likit S Poojary

PGA-33(Andheri)