```
import numpy as np
import pickle
import cv2
import os
import matplotlib.pyplot as plt
from os import listdir
from sklearn.preprocessing import LabelBinarizer
from keras.models import Sequential
from keras.layers import BatchNormalization, Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator, img_to_array
from sklearn.model_selection import train_test_split
```

```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("emmarex/plantdisease")

print("Path to dataset files:", path)
```

```
Warning: Looks like you're using an outdated `kagglehub` version, please consider updating (latest version: 0.3.4)
Path to dataset files: /root/.cache/kagglehub/datasets/emmarex/plantdisease/versions/1
```

```
# Dimension of resized image
DEFAULT_IMAGE_SIZE = tuple((256, 256))

# Number of images used to train the model
N_IMAGES = 100

# Path to the dataset folder
train_dir = os.path.join(path, 'train')
val_dir = os.path.join(path, 'val')
```

```
def convert_image_to_array(image_dir):
    try:
        image = cv2.imread(image_dir)
        if image is not None:
            image = cv2.resize(image, DEFAULT_IMAGE_SIZE)
            return img_to_array(image)
        else:
            return np.array([])
    except Exception as e:
        print(f"Error : {e}")
        return None
```

```
image_list, label_list = [], []

try:
    print("[INFO] Loading images ...")
    plant_disease_folder_list = listdir(train_dir)

    for plant_disease_folder in plant_disease_folder_list:
        print(f"[INFO] Processing {plant_disease_folder} ...")
        plant_disease_image_list = listdir(f"{train_dir}/{plant_disease_folder}/")

        for image in plant_disease_image_list[:N_IMAGES]:
            image_directory = f"{train_dir}/{plant_disease_folder}/{image}"
            if image_directory.endswith(".jpg")==True or image_directory.endswith(".JPG")==True:
                image_list.append(convert_image_to_array(image_directory))
                label_list.append(plant_disease_folder)

    print("[INFO] Image loading completed")
except Exception as e:
    print(f"Error : {e}")

# Transform the loaded training image data into numpy array
np_image_list = np.array(image_list, dtype=np.float16) / 225.0
print()

# Check the number of images loaded for training
image_len = len(image_list)
print(f"Total number of images: {image_len}")
```

```
[INFO] Loading images ...
[INFO] Processing Soybean___healthy ...
[INFO] Processing Cherry_(including_sour)___Powdery_mildew ...
[INFO] Processing Corn_(maize)___Northern_Leaf_Blight ...
[INFO] Processing Tomato___healthy ...
[INFO] Processing Tomato___Target_Spot ...
```

```
[INFO] Processing Potato___Early_blight ...
[INFO] Processing Pepper,_bell___Bacterial_spot ...
[INFO] Processing Peach___healthy ...
[INFO] Processing Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot ...
[INFO] Processing Corn_(maize)___healthy ...
[INFO] Processing Cherry_(including_sour)___healthy ...
[INFO] Processing Tomato___Leaf_Mold ...
[INFO] Processing Apple___Cedar_apple_rust ...
[INFO] Processing Potato___healthy ...
[INFO] Processing Squash___Powdery_mildew ...
[INFO] Processing Strawberry___healthy ...
[INFO] Processing Orange___Haunglongbing_(Citrus_greening) ...
[INFO] Processing Apple___healthy ...
[INFO] Processing Grape___Leaf_blight_(Isariopsis_Leaf_Spot) ...
[INFO] Processing Strawberry___Leaf_scorch ...
[INFO] Processing Tomato___Spider_mites Two-spotted_spider_mite ...
[INFO] Processing Tomato___Tomato_Yellow_Leaf_Curl_Virus ...
[INFO] Processing Pepper,_bell___healthy ...
[INFO] Processing Potato___Late_blight ...
[INFO] Processing background ...
[INFO] Processing Raspberry___healthy ...
[INFO] Processing Corn_(maize)___Common_rust_ ...
[INFO] Processing Apple___Black_rot ...
[INFO] Processing Peach___Bacterial_spot ...
[INFO] Processing Grape___healthy ...
[INFO] Processing Tomato___Septoria_leaf_spot ...
[INFO] Processing Apple___Apple_scab ...
[INFO] Processing Blueberry___healthy ...
[INFO] Processing Grape___Black_rot ...
[INFO] Processing Tomato___Bacterial_spot ...
[INFO] Processing Grape___Esca_(Black_Measles) ...
[INFO] Processing Tomato___Late_blight ...
[INFO] Processing Tomato___Tomato_mosaic_virus ...
[INFO] Processing Tomato___Early_blight ...
[INFO] Image loading completed

Total number of images: 3900
```

```
label_binarizer = LabelBinarizer()
image_labels = label_binarizer.fit_transform(label_list)

pickle.dump(label_binarizer,open('plant_disease_label_transform.pkl', 'wb'))
n_classes = len(label_binarizer.classes_)

print("Total number of classes: ", n_classes)
```

```
Total number of classes:  39
```

```
augment = ImageDataGenerator(rotation_range=25, width_shift_range=0.1,
                             height_shift_range=0.1, shear_range=0.2,
                             zoom_range=0.2, horizontal_flip=True,
                             fill_mode="nearest")
```

```
print("[INFO] Splitting data to train and test...")
x_train, x_test, y_train, y_test = train_test_split(np_image_list, image_labels, test_size=0.2, random_state = 42)
```

```
[INFO] Splitting data to train and test...
```

```
EPOCHS = 25
STEPS = 100
LR = 1e-3
BATCH_SIZE = 32
WIDTH = 256
HEIGHT = 256
DEPTH = 3
```

```
model = Sequential()
inputShape = (HEIGHT, WIDTH, DEPTH)
chanDim = -1

if K.image_data_format() == "channels_first":
    inputShape = (DEPTH, HEIGHT, WIDTH)
    chanDim = 1

model.add(Conv2D(32, (3, 3), padding="same",input_shape=inputShape))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(3, 3)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(128, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(Conv2D(128, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(1024))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(n_classes))
model.add(Activation("softmax"))

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 256, 256, 32)      896
_________________________________________________________________
activation_1 (Activation)    (None, 256, 256, 32)      0
_________________________________________________________________
batch_normalization_1 (Batch (None, 256, 256, 32)      128
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 85, 85, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 85, 85, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 85, 85, 64)        18496
_________________________________________________________________
activation_2 (Activation)    (None, 85, 85, 64)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 85, 85, 64)        256
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 85, 85, 64)        36928
_________________________________________________________________
activation_3 (Activation)    (None, 85, 85, 64)        0
_________________________________________________________________
batch_normalization_3 (Batch (None, 85, 85, 64)        256
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 42, 42, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 42, 42, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 42, 42, 128)       73856
_________________________________________________________________
activation_4 (Activation)    (None, 42, 42, 128)       0
_________________________________________________________________
batch_normalization_4 (Batch (None, 42, 42, 128)       512
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 42, 42, 128)       147584
_________________________________________________________________
activation_5 (Activation)    (None, 42, 42, 128)       0
_________________________________________________________________
batch_normalization_5 (Batch (None, 42, 42, 128)       512
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 21, 21, 128)       0
_________________________________________________________________
dropout_3 (Dropout)          (None, 21, 21, 128)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 56448)             0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              57803776
_________________________________________________________________
activation_6 (Activation)    (None, 1024)              0
_________________________________________________________________
batch_normalization_6 (Batch (None, 1024)              4096
_________________________________________________________________
dropout_4 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 39)                39975
_________________________________________________________________
```

```
# Initialize optimizer
opt = Adam(lr=LR, decay=LR / EPOCHS)

# Compile model
model.compile(loss="binary_crossentropy", optimizer=opt, metrics=["accuracy"])
```

```
# Train model
print("[INFO] Training network...")
history = model.fit_generator(augment.flow(x_train, y_train, batch_size=BATCH_SIZE),
                              validation_data=(x_test, y_test),
                              steps_per_epoch=len(x_train) // BATCH_SIZE,
                              epochs=EPOCHS,
                              verbose=1)
```

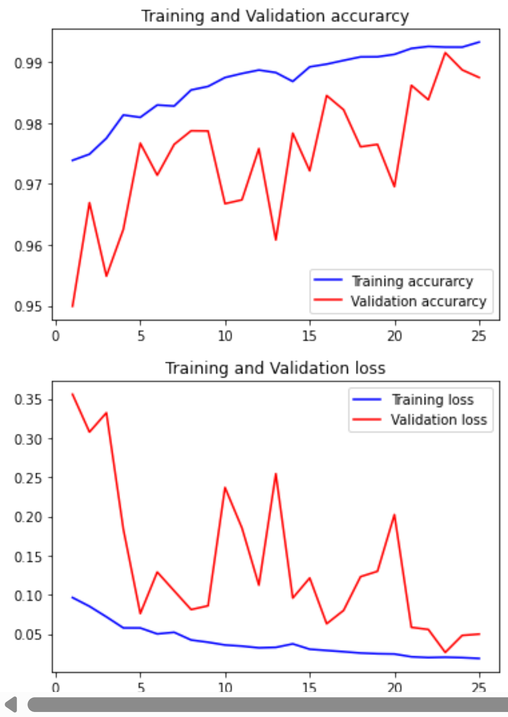
```
[INFO] Training network...
Epoch 1/25
97/97 [==============================] - 792s 8s/step - loss: 0.0964 - accuracy: 0.9739 - val_loss: 0.3557 - val_accuracy: 0.9499
Epoch 2/25
97/97 [==============================] - 778s 8s/step - loss: 0.0852 - accuracy: 0.9749 - val_loss: 0.3076 - val_accuracy: 0.9669
Epoch 3/25
97/97 [==============================] - 793s 8s/step - loss: 0.0718 - accuracy: 0.9775 - val_loss: 0.3322 - val_accuracy: 0.9549
Epoch 4/25
97/97 [==============================] - 788s 8s/step - loss: 0.0576 - accuracy: 0.9813 - val_loss: 0.1837 - val_accuracy: 0.9626
Epoch 5/25
97/97 [==============================] - 791s 8s/step - loss: 0.0575 - accuracy: 0.9809 - val_loss: 0.0760 - val_accuracy: 0.9767
Epoch 6/25
97/97 [==============================] - 802s 8s/step - loss: 0.0500 - accuracy: 0.9830 - val_loss: 0.1289 - val_accuracy: 0.9714
Epoch 7/25
97/97 [==============================] - 795s 8s/step - loss: 0.0520 - accuracy: 0.9828 - val_loss: 0.1049 - val_accuracy: 0.9765
Epoch 8/25
97/97 [==============================] - 802s 8s/step - loss: 0.0422 - accuracy: 0.9854 - val_loss: 0.0811 - val_accuracy: 0.9787
Epoch 9/25
97/97 [==============================] - 796s 8s/step - loss: 0.0395 - accuracy: 0.9860 - val_loss: 0.0859 - val_accuracy: 0.9787
Epoch 10/25
97/97 [==============================] - 804s 8s/step - loss: 0.0359 - accuracy: 0.9875 - val_loss: 0.2368 - val_accuracy: 0.9668
Epoch 11/25
97/97 [==============================] - 809s 8s/step - loss: 0.0345 - accuracy: 0.9881 - val_loss: 0.1852 - val_accuracy: 0.9674
Epoch 12/25
97/97 [==============================] - 802s 8s/step - loss: 0.0323 - accuracy: 0.9887 - val_loss: 0.1122 - val_accuracy: 0.9758
Epoch 13/25
97/97 [==============================] - 806s 8s/step - loss: 0.0328 - accuracy: 0.9883 - val_loss: 0.2544 - val_accuracy: 0.9608
Epoch 14/25
97/97 [==============================] - 804s 8s/step - loss: 0.0373 - accuracy: 0.9868 - val_loss: 0.0958 - val_accuracy: 0.9783
Epoch 15/25
97/97 [==============================] - 810s 8s/step - loss: 0.0305 - accuracy: 0.9892 - val_loss: 0.1213 - val_accuracy: 0.9722
Epoch 16/25
97/97 [==============================] - 796s 8s/step - loss: 0.0289 - accuracy: 0.9897 - val_loss: 0.0630 - val_accuracy: 0.9845
Epoch 17/25
97/97 [==============================] - 807s 8s/step - loss: 0.0273 - accuracy: 0.9903 - val_loss: 0.0800 - val_accuracy: 0.9822
Epoch 18/25
97/97 [==============================] - 829s 9s/step - loss: 0.0256 - accuracy: 0.9909 - val_loss: 0.1230 - val_accuracy: 0.9761
Epoch 19/25
97/97 [==============================] - 822s 8s/step - loss: 0.0247 - accuracy: 0.9909 - val_loss: 0.1299 - val_accuracy: 0.9765
Epoch 20/25
97/97 [==============================] - 814s 8s/step - loss: 0.0243 - accuracy: 0.9913 - val_loss: 0.2022 - val_accuracy: 0.9696
Epoch 21/25
97/97 [==============================] - 820s 8s/step - loss: 0.0210 - accuracy: 0.9922 - val_loss: 0.0584 - val_accuracy: 0.9862
Epoch 22/25
97/97 [==============================] - 824s 8s/step - loss: 0.0198 - accuracy: 0.9926 - val_loss: 0.0557 - val_accuracy: 0.9838
Epoch 23/25
97/97 [==============================] - 825s 9s/step - loss: 0.0203 - accuracy: 0.9925 - val_loss: 0.0264 - val_accuracy: 0.9916
Epoch 24/25
97/97 [==============================] - 818s 8s/step - loss: 0.0197 - accuracy: 0.9925 - val_loss: 0.0481 - val_accuracy: 0.9887
Epoch 25/25
97/97 [==============================] - 813s 8s/step - loss: 0.0187 - accuracy: 0.9933 - val_loss: 0.0497 - val_accuracy: 0.9875
```

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)

# Train and validation accuracy
plt.plot(epochs, acc, 'b', label='Training accurarcy')
plt.plot(epochs, val_acc, 'r', label='Validation accurarcy')
plt.title('Training and Validation accurarcy')
plt.legend()

plt.figure()

# Train and validation loss
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and Validation loss')
plt.legend()
plt.show()
```



```
print("[INFO] Calculating model accuracy")
scores = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {scores[1]*100}")
```

```
[INFO] Calculating model accuracy
780/780 [==============================] - 52s 66ms/step
Test Accuracy: 98.74754548072815
```

```
# Dump pickle file of the model
print("[INFO] Saving model...")
pickle.dump(model,open('plant_disease_classification_model.pkl', 'wb'))
```

```
[INFO] Saving model...
```

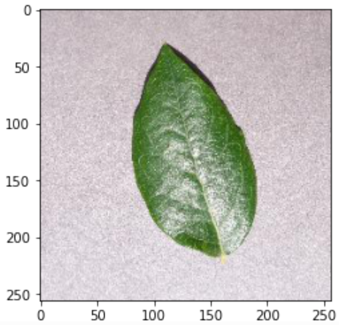
```
# Dump pickle file of the labels
print("[INFO] Saving label transform...")
filename = 'plant_disease_label_transform.pkl'
image_labels = pickle.load(open(filename, 'rb'))
```

```
[INFO] Saving label transform...
```

```
def predict_disease(image_path):
    image_array = convert_image_to_array(image_path)
    np_image = np.array(image_array, dtype=np.float16) / 225.0
    np_image = np.expand_dims(np_image,0)
    plt.imshow(plt.imread(image_path))
    result = model.predict_classes(np_image)
    print((image_labels.classes_[result][0]))
```

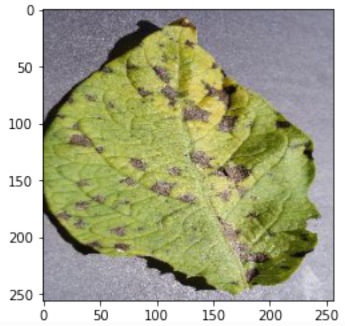
```
predict_disease('/content/PlantVillage/val/Blueberry___healthy/008c85d0-a954-4127-bd26-861dc8a1e6ff___RS_HL 2431.JPG')
```

```
Blueberry___healthy
```



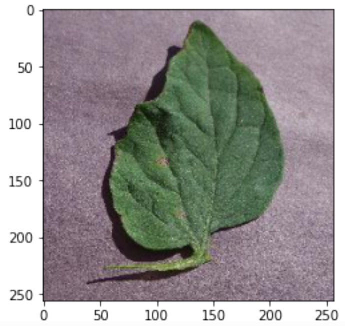
```
predict_disease('/content/PlantVillage/val/Potato___Early_blight/03b0d3c1-b5b0-48f4-98aa-f8904670290f___RS_Early.B 7051.JPG')
```

```
Potato___Early_blight
```



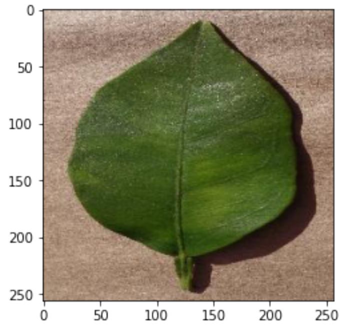
```
predict_disease('/content/PlantVillage/val/Tomato___Target_Spot/1006b3dd-22d8-41b8-b83d-08bf189fcdaa___Com.G_TgS_FL 8118.JPG')
```

```
Tomato___Target_Spot
```



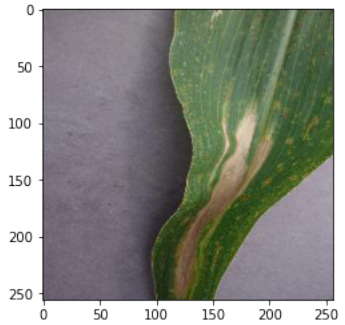
```
predict_disease('/content/PlantVillage/val/Orange___Haunglongbing_(Citrus_greening)/02459e0c-a189-4dc9-a0dc-0548e36d0efb___CREC_HLB 571
```

```
Orange___Haunglongbing_(Citrus_greening)
```



```
predict_disease('/content/PlantVillage/val/Corn_(maize)___Northern_Leaf_Blight/028159fc-995e-455a-8d60-6d377580a898___RS_NLB 4023.JPG')
```

```
Corn_(maize)___Northern_Leaf_Blight
```



Start coding or generate with AI.