In [ ]:

```
import numpy as np
from os import listdir
from sklearn.preprocessing import LabelBinarizer
from keras.models import Sequential
from keras.layers.normalization import BatchNormalization
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers.core import Activation, Flatten, Dropout, Dense
from keras import backend as K
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import Adam
from keras.preprocessing import image
from keras.preprocessing.image import img_to_array
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from tensorflow.keras import datasets, layers, models
import lime
from lime import lime_image
from sklearn import preprocessing
from keras.utils.np_utils import to_categorical
from keras import optimizers
from sklearn.preprocessing import OneHotEncoder
from tensorflow.keras.optimizers import RMSprop,SGD
from keras.utils import np_utils
import keras as keras1
from sklearn.preprocessing import OneHotEncoder
import cv2
default_image_size = tuple((256, 256))
image_size = 0
directory_root = 'C:/Users/owner/Desktop/PlantVillage'
width=256
height=256
depth=3
EPOCHS = 10
INIT_LR = 1e-3
BS = 32
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 ...")
    root_dir = listdir(directory_root)
    for directory in root_dir :
        # remove .DS_Store from list
        if directory == ".DS_Store" :
            root_dir.remove(directory)


    for plant_folder in root_dir :
        plant_disease_folder_list = listdir(f"{directory_root}/{plant_folder}")

        for disease_folder in plant_disease_folder_list :
            # remove .DS_Store from list
            if disease_folder == ".DS_Store" :
                plant_disease_folder_list.remove(disease_folder)





        for image in plant_disease_folder_list[:200]:
            image_directory = f"{directory_root}/{plant_folder}/{image}"

            if image_directory.endswith(".jpg") or image_directory.endswith(".JPG"):

                    image_list.append(convert_image_to_array(image_directory))
                    label_list.append(plant_folder)
    print("[INFO] Image loading completed")
    image_size = len(image_list)



except Exception as e:
    print(f"Error : {e}")



Labelencoder = preprocessing.LabelEncoder()
target_column = Labelencoder.fit_transform(label_list)
labels=to_categorical(target_column,15)

n_classes=15
np_image_list = np.array(image_list, dtype=np.float16)/255.0


x_train, x_test, y_train, y_test = train_test_split(np_image_list,labels, test_size=0.2, random_stat
e = 42)


if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0],  256, 256,3)
    x_test = x_test.reshape(x_test.shape[0],  256, 256,3)
    inputShape = (3,256, 256)

else:
    x_train= x_train.reshape(x_train.shape[0], 256, 256,3)
    x_test= x_test.reshape(x_test.shape[0], 256, 256,3)
    inputShape = (256, 256, 3)







model = Sequential()

model.add(Conv2D(32, (3, 3), padding="same",input_shape=inputShape))
model.add(Activation("relu"))

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(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))

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(Conv2D(128, (3, 3), padding="same"))
model.add(Activation("relu"))

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.compile(optimizer=optimizers.Adam(lr=0.0003, beta_1=0.9, beta_2=0.999, epsilon=None, decay=1e-
8, amsgrad=False),
              loss='binary_crossentropy',
              metrics=['accuracy'])


aug = 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")

history = model.fit_generator(
    aug.flow(x_train, y_train, batch_size=60),
    validation_data=(x_test, y_test),
    steps_per_epoch=len(x_train) // BS,
    epochs=EPOCHS, verbose=1
    )

# acc = history.history['acc']
# val_acc = history.history['val_acc']

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



img = np.expand_dims(x_test[0], axis=0)
prediction = model.predict(img)

print(prediction.argmax(),"Predictionnnnnnnnnnn")

predictions = np.argmax(model(x_test).as_ndarray(), axis=1)

# Confusion matrix and classification report.
print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))

# Learning curve.
plt.plot(learning_curve, linewidth=3, label="train")
plt.plot(test_learning_curve, linewidth=3, label="test")
plt.title("Learning curve")
plt.ylabel("error")
plt.xlabel("epoch")
plt.legend()
plt.grid()
plt.show()


##Explainable AI

from lime.wrappers.scikit_image import SegmentationAlgorithm
explainer = lime_image.LimeImageExplainer(verbose = False)
segmenter = SegmentationAlgorithm('slic', n_segments=100, compactness=1, sigma=1)
explanation = explainer.explain_instance(x_test[0],
                                         classifier_fn = model.predict,
                                          top_labels=10, hide_color=0, num_samples=10,  segmentation
_fn=segmenter)
from skimage.color import label2rgb
temp, mask = explanation.get_image_and_mask(le_labels[0], positive_only=True, num_features=5, hide_r
est=False)
fig, (ax1, ax2) = plt.subplots(1,2, figsize = (8, 4))
ax1.imshow(label2rgb(mask,temp, bg_label = 0), interpolation = 'nearest')
ax1.set_title('Positive Regions for {}'.format(y_test[0]))
```