Leaf Disease Detection using Alexnet

```
import os
os.listdir("../input/plant-diseases-classification-using-alexnet")
```

```
['__results__.html',
 'best_weights_9.hdf5',
 'AlexNetModel.hdf5',
 '__results___files',
 '__output__.json',
 'custom.css']
```

Tensorflow is a foundation library that can be used to create deep learning models directly or by using wrapper libraries that simplify the process built on the top of tensorflow so for making deep learning model we are installing tensorflow. TensorFlow is mainly used for: Classification, Perception, Understanding, Discovering, Prediction and Creation.

```
# ! pip install tensorflow-gpu
```

from tensorflow we are importing device_lib that will provide us the full summary or list of information about local device like computational capability, memory limit , device type etc.

```
from tensorflow.python.client import device_lib

device_lib.list_local_devices()
```

```
[name: "/device:CPU:0"
 device_type: "CPU"
 memory_limit: 268435456
 locality {
 }
 incarnation: 12930137146812964960, name: "/device:GPU:0"
 device_type: "GPU"
 memory_limit: 15900252570
 locality {
   bus_id: 1
   links {
   }
 }
 incarnation: 17279202340371761108
 physical_device_desc: "device: 0, name: Tesla P100-PCIE-16GB, pci bus id: 0000:00:04
```

Keras is a powerful and easy-to-use free open source Python library for developing and evaluating deep learning models. It wraps the efficient numerical computation libraries and allows you to define and train neural network models in just a few lines of code.

```
import os

import tensorflow as tf
from tensorflow import keras
```

```
!nvidia-smi
```

Building CNN Based On AlexNet Architecture

Here we are making the sequential model for that we are importing sequential from tensorflow. A Sequential model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor. By the use of Convolution2D it will be summing up the results into a single output pixel.Maximum pooling, or max pooling, is a pooling operation that calculates the maximum, or largest, value in each patch of each feature map. Flatten can converts the pooled feature map to a single column that is passed to the fully connected layer. Dense adds the fully connected layer to the neural network. The Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting. And hence Batch normalization accelerates training, in some cases by halving the epochs or better, and provides some regularization, reducing generalization error.

```
# Importing Keras libraries and packages
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Convolution2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import BatchNormalization
```

```
with tf.device('/device:GPU:0'):
    # Initializing the CNN
    classifier = Sequential()

    # Convolution Step 1
    classifier.add(Convolution2D(96, 11, strides = (4, 4), padding = 'valid', input_shape=

    # Max Pooling Step 1
    classifier.add(MaxPooling2D(pool_size = (2, 2), strides = (2, 2), padding = 'valid'))
    classifier.add(BatchNormalization())

    # Convolution Step 2
    classifier.add(Convolution2D(256, 11, strides = (1, 1), padding='valid', activation =

    # Max Pooling Step 2
    classifier.add(MaxPooling2D(pool_size = (2, 2), strides = (2, 2), padding='valid'))
```

```
classifier.add(MaxPooling2D(pool_size = (2, 2), strides = (2, 2), padding='valid'))
classifier.add(BatchNormalization())

# Convolution Step 3
classifier.add(Convolution2D(384, 3, strides = (1, 1), padding='valid', activation = '
classifier.add(BatchNormalization())

# Convolution Step 4
classifier.add(Convolution2D(384, 3, strides = (1, 1), padding='valid', activation = '
classifier.add(BatchNormalization())

# Convolution Step 5
classifier.add(Convolution2D(256, 3, strides=(1,1), padding='valid', activation = 'rel

# Max Pooling Step 3
classifier.add(MaxPooling2D(pool_size = (2, 2), strides = (2, 2), padding = 'valid'))
classifier.add(BatchNormalization())

# Flattening Step
classifier.add(Flatten())

# Full Connection Step
classifier.add(Dense(units = 4096, activation = 'relu'))
classifier.add(Dropout(0.4))
classifier.add(BatchNormalization())
classifier.add(Dense(units = 4096, activation = 'relu'))
classifier.add(Dropout(0.4))
classifier.add(BatchNormalization())
classifier.add(Dense(units = 1000, activation = 'relu'))
classifier.add(Dropout(0.2))
classifier.add(BatchNormalization())
classifier.add(Dense(units = 38, activation = 'softmax'))
classifier.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 54, 54, 96)        34944
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 27, 27, 96)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 27, 27, 96)        384
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 17, 17, 256)       2973952
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 256)         0
_________________________________________________________________
batch_normalization_1 (Batch (None, 8, 8, 256)         1024
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 6, 6, 384)         885120
_________________________________________________________________
batch_normalization_2 (Batch (None, 6, 6, 384)         1536
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 4, 4, 384)         1327488
_________________________________________________________________
batch_normalization_3 (Batch (None, 4, 4, 384)         1536
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 2, 2, 256)         884992
```

```
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 1, 1, 256)         0
_________________________________________________________________
batch_normalization_4 (Batch (None, 1, 1, 256)         1024
_________________________________________________________________
flatten (Flatten)            (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 4096)              1052672
_________________________________________________________________
dropout (Dropout)            (None, 4096)              0
_________________________________________________________________
batch_normalization_5 (Batch (None, 4096)              16384
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dropout_1 (Dropout)          (None, 4096)              0
_________________________________________________________________
batch_normalization_6 (Batch (None, 4096)              16384
_________________________________________________________________
dense_2 (Dense)              (None, 1000)              4097000
_________________________________________________________________
dropout_2 (Dropout)          (None, 1000)              0
_________________________________________________________________
batch_normalization_7 (Batch (None, 1000)              4000
_________________________________________________________________
dense_3 (Dense)              (None, 38)                38038
=================================================================
Total params: 28,117,790
Trainable params: 28,096,654
Non-trainable params: 21,136
_________________________________________________________________
```

Loading Weights To The Model

```
classifier.load_weights('../input/plant-diseases-classification-using-alexnet/best_weights
```

Fine Tuning By Freezing Some Layers Of Our Model

let's visualize layer names and layer indices to see how many layers we should freeze:

```
from tensorflow.keras import layers
for i, layer in enumerate(classifier.layers):
   print(i, layer.name)
```

```
0 conv2d
1 max_pooling2d
2 batch_normalization
3 conv2d_1
4 max_pooling2d_1
5 batch_normalization_1
6 conv2d_2
7 batch_normalization_2
8 conv2d_3
9 batch_normalization_3
10 conv2d_4
```

```
11 max_pooling2d_2
12 batch_normalization_4
13 flatten
14 dense
15 dropout
16 batch_normalization_5
17 dense_1
18 dropout_1
19 batch_normalization_6
20 dense_2
21 dropout_2
22 batch_normalization_7
23 dense_3
```

we chose to train the top 2 conv blocks, i.e. we will freeze the first 8 layers and unfreeze the rest:

```
print("Freezed layers:")
for i, layer in enumerate(classifier.layers[:20]):
    print(i, layer.name)
    layer.trainable = False
```

```
Freezed layers:
0 conv2d
1 max_pooling2d
2 batch_normalization
3 conv2d_1
4 max_pooling2d_1
5 batch_normalization_1
6 conv2d_2
7 batch_normalization_2
8 conv2d_3
9 batch_normalization_3
10 conv2d_4
11 max_pooling2d_2
12 batch_normalization_4
13 flatten
14 dense
15 dropout
16 batch_normalization_5
17 dense_1
18 dropout_1
19 batch_normalization_6
```

Model Summary After Freezing

trainable parameters decrease after freezing some bottom layers.

```
classifier.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 54, 54, 96)        34944
_________________________________________________________________
```

```
max_pooling2d (MaxPooling2D) (None, 27, 27, 96)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 27, 27, 96)        384
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 17, 17, 256)       2973952
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 256)         0
_________________________________________________________________
batch_normalization_1 (Batch (None, 8, 8, 256)         1024
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 6, 6, 384)         885120
_________________________________________________________________
batch_normalization_2 (Batch (None, 6, 6, 384)         1536
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 4, 4, 384)         1327488
_________________________________________________________________
batch_normalization_3 (Batch (None, 4, 4, 384)         1536
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 2, 2, 256)         884992
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 1, 1, 256)         0
_________________________________________________________________
batch_normalization_4 (Batch (None, 1, 1, 256)         1024
_________________________________________________________________
flatten (Flatten)            (None, 256)               0
_________________________________________________________________
dense (Dense)                (None, 4096)              1052672
_________________________________________________________________
dropout (Dropout)            (None, 4096)              0
_________________________________________________________________
batch_normalization_5 (Batch (None, 4096)              16384
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dropout_1 (Dropout)          (None, 4096)              0
_________________________________________________________________
batch_normalization_6 (Batch (None, 4096)              16384
_________________________________________________________________
dense_2 (Dense)              (None, 1000)              4097000
_________________________________________________________________
dropout_2 (Dropout)          (None, 1000)              0
_________________________________________________________________
batch_normalization_7 (Batch (None, 1000)              4000
_________________________________________________________________
dense_3 (Dense)              (None, 38)                38038
=================================================================
Total params: 28,117,790
Trainable params: 4,137,038
Non-trainable params: 23,980,752
_________________________________________________________________
```

Compiling the Model

```
# Compiling the Model
with tf.device('/device:GPU:0'):
    from tensorflow.keras import optimizers
    classifier.compile(optimizer=optimizers.SGD(lr=0.001, momentum=0.9, decay=0.005),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
              metrics=['accuracy'])
```

Image Preprocessing

```
# image preprocessing
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   fill_mode='nearest')

valid_datagen = ImageDataGenerator(rescale=1./255)

batch_size = 128
base_dir = "../input/new-plant-diseases-dataset/new plant diseases dataset(augmented)/New

training_set = train_datagen.flow_from_directory(base_dir+'/train',
                                                 target_size=(224, 224),
                                                 batch_size=batch_size,
                                                 class_mode='categorical')

valid_set = valid_datagen.flow_from_directory(base_dir+'/valid',
                                              target_size=(224, 224),
                                              batch_size=batch_size,
                                              class_mode='categorical')
```

```
Found 70295 images belonging to 38 classes.
Found 17572 images belonging to 38 classes.
```

Printing all the classes of training data set with index value.

```
class_dict = training_set.class_indices
print(class_dict)
```

```
{'Apple___Apple_scab': 0, 'Apple___Black_rot': 1, 'Apple___Cedar_apple_rust': 2, 'App
```

listing all the classes without index

```
li = list(class_dict.keys())
print(li)
```

```
['Apple___Apple_scab', 'Apple___Black_rot', 'Apple___Cedar_apple_rust', 'Apple___heal
```

Assigning all the training set and valid set samples to the variable

```
train_num = training_set.samples
valid_num = valid_set.samples
```

Importing checkpoint for checking the accuracy of each epoch to avoid the overfitting problem.

```
# checkpoint
from tensorflow.keras.callbacks import ModelCheckpoint

weightpath = "./best_weights_9.hdf5"
checkpoint = ModelCheckpoint(weightpath, monitor='val_acc', verbose=1, save_best_only=True
callbacks_list = [checkpoint]

#fitting images to CNN
history = classifier.fit_generator(training_set,
                         steps_per_epoch=train_num//batch_size,
                         validation_data=valid_set,
                         epochs=10,
                         validation_steps=valid_num//batch_size,
                         callbacks=callbacks_list)
#saving model
filepath="AlexNetModel.hdf5"
classifier.save(filepath)
```

```
Epoch 1/10
548/549 [============================>.] - ETA: 2s - loss: 0.0930 - acc: 0.9692
Epoch 00001: val_acc improved from -inf to 0.96305, saving model to ./best_weights_9
549/549 [==============================] - 1271s 2s/step - loss: 0.0930 - acc: 0.969:
Epoch 2/10
548/549 [============================>.] - ETA: 1s - loss: 0.0933 - acc: 0.9684
Epoch 00002: val_acc improved from 0.96305 to 0.96339, saving model to ./best_weights
549/549 [==============================] - 821s 1s/step - loss: 0.0933 - acc: 0.9685
Epoch 3/10
548/549 [============================>.] - ETA: 1s - loss: 0.0925 - acc: 0.9699
Epoch 00003: val_acc improved from 0.96339 to 0.96356, saving model to ./best_weights
549/549 [==============================] - 846s 2s/step - loss: 0.0925 - acc: 0.9699
Epoch 4/10
548/549 [============================>.] - ETA: 1s - loss: 0.0930 - acc: 0.9687
Epoch 00004: val_acc did not improve from 0.96356
549/549 [==============================] - 832s 2s/step - loss: 0.0930 - acc: 0.9687
Epoch 5/10
548/549 [============================>.] - ETA: 1s - loss: 0.0917 - acc: 0.9695
Epoch 00005: val_acc did not improve from 0.96356
549/549 [==============================] - 832s 2s/step - loss: 0.0916 - acc: 0.9696
Epoch 6/10
548/549 [============================>.] - ETA: 1s - loss: 0.0918 - acc: 0.9693
Epoch 00006: val_acc did not improve from 0.96356
549/549 [==============================] - 831s 2s/step - loss: 0.0918 - acc: 0.9693
Epoch 7/10
548/549 [============================>.] - ETA: 1s - loss: 0.0939 - acc: 0.9687
Epoch 00007: val_acc did not improve from 0.96356
549/549 [==============================] - 804s 1s/step - loss: 0.0938 - acc: 0.9688
Epoch 8/10
548/549 [============================>.] - ETA: 1s - loss: 0.0933 - acc: 0.9686
Epoch 00008: val_acc did not improve from 0.96356
549/549 [==============================] - 851s 2s/step - loss: 0.0932 - acc: 0.9686
Epoch 9/10
```

```
548/549 [============================>.] - ETA: 1s - loss: 0.0930 - acc: 0.9687
Epoch 00009: val_acc did not improve from 0.96356
549/549 [==============================] - 827s 2s/step - loss: 0.0930 - acc: 0.9688
Epoch 10/10
548/549 [============================>.] - ETA: 1s - loss: 0.0914 - acc: 0.9696
Epoch 00010: val_acc did not improve from 0.96356
549/549 [==============================] - 826s 2s/step - loss: 0.0915 - acc: 0.9696
```

Visualising Training Progress

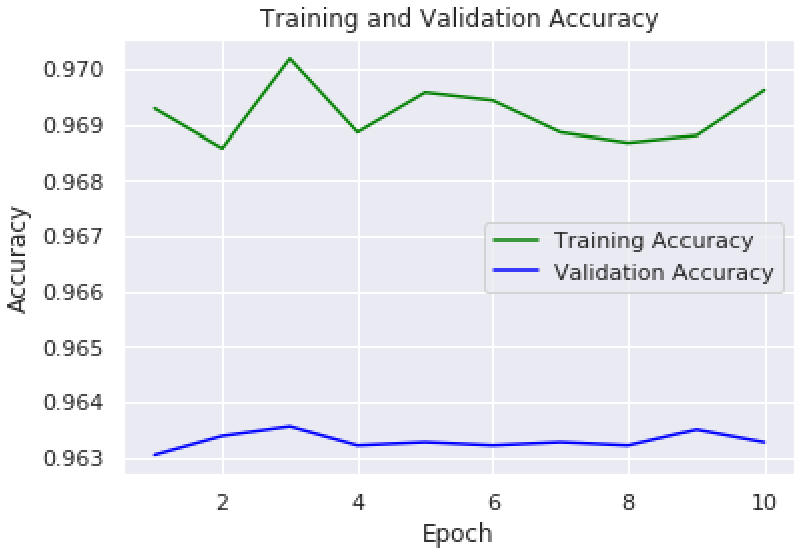
```
#plotting training values
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

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

#accuracy plot
plt.plot(epochs, acc, color='green', label='Training Accuracy')
plt.plot(epochs, val_acc, color='blue', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()

plt.figure()
#loss plot
plt.plot(epochs, loss, color='pink', label='Training Loss')
plt.plot(epochs, val_loss, color='red', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.show()
```



Predicting New Test Image(s)


```
# predicting an image
from tensorflow.keras.preprocessing import image
import numpy as np
image_path = "../input/new-plant-diseases-dataset/test/test/TomatoEarlyBlight1.JPG"
new_img = image.load_img(image_path, target_size=(224, 224))
img = image.img_to_array(new_img)
img = np.expand_dims(img, axis=0)
img = img/255

print("Following is our prediction:")
prediction = classifier.predict(img)
# decode the results into a list of tuples (class, description, probability)
# (one such list for each sample in the batch)
d = prediction.flatten()
j = d.max()
for index,item in enumerate(d):
    if item == j:
        class_name = li[index]

##Another way
# img_class = classifier.predict_classes(img)
# img_prob = classifier.predict_proba(img)
# print(img_class ,img_prob )


#ploting image with predicted class name
plt.figure(figsize = (4,4))
plt.imshow(new_img)
plt.axis('off')
plt.title(class_name)
plt.show()
```

```
Following is our prediction:
```

Tomato___Early_blight

```
new_model = tf.keras.models.load_model('./AlexNetModel.hdf5')
```

```
# predicting an image
from keras.preprocessing import image
import numpy as np
image_path = "../input/new-plant-diseases-dataset/test/test/AppleCedarRust2.JPG"
new_img = image.load_img(image_path, target_size=(224, 224))
img = image.img_to_array(new_img)
img = np.expand_dims(img, axis=0)
img = img/255

print("Following is our prediction:")
prediction = new_model.predict(img)
# decode the results into a list of tuples (class, description, probability)
# (one such list for each sample in the batch)
d = prediction.flatten()
j = d.max()
for index,item in enumerate(d):
    if item == j:
        class_name = li[index]

##Another way
# img_class = classifier.predict_classes(img)
# img_prob = classifier.predict_proba(img)
# print(img_class ,img_prob )


#ploting image with predicted class name
plt.figure(figsize = (4,4))
plt.imshow(new_img)
plt.axis('off')
plt.title(class_name)
plt.show()
```

```
Using TensorFlow backend.
Following is our prediction:
```