Submitted by Navneet Das 3433 Comp A

***'
Assignment 3-B Deep Learning
Convolutional neural network (CNN)
Use any dataset of plant disease and design a plant disease detection system using CNN.
'***

In [1]:
```
import os
import cv2
import numpy as np
import pandas as pd
from PIL import Image
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

from tensorflow import keras
from keras.models import Sequential
from keras.layers import Conv2D,MaxPooling2D,Dense,Flatten,Dropout
# from keras.layers.normalization import BatchNormalization
```

In [3]:
```
fpath = "PlantVillage/"
random_seed = 111

categories = os.listdir(fpath)
print("List of categories = ",categories,"\n\nNo. of categories = ", len(categories))
```

```
List of categories =  ['Pepper__bell___Bacterial_spot', 'Pepper__bell___healthy', 'PlantVillage', 'Potato___Early_b
ght', 'Potato___healthy', 'Potato___Late_blight', 'Tomato_Bacterial_spot', 'Tomato_Early_blight', 'Tomato_healthy',
'Tomato_Late_blight', 'Tomato_Leaf_Mold', 'Tomato_Septoria_leaf_spot', 'Tomato_Spider_mites_Two_spotted_spider_mite',
'Tomato__Target_Spot', 'Tomato__Tomato_mosaic_virus', 'Tomato__Tomato_YellowLeaf__Curl_Virus']

No. of categories =  16
```

In [4]:
```
def load_images_and_labels(categories):
    img_lst=[]
    labels=[]
    for index, category in enumerate(categories):
        for image_name in os.listdir(fpath+"/"+category)[:300]:
            file_ext = image_name.split(".")[-1]
            if (file_ext.lower() == "jpg") or (file_ext.lower() == "jpeg"):
                img = cv2.imread(fpath+"/"+category+"/"+image_name)
                img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

                img_array = Image.fromarray(img, 'RGB')

                #resize image to 227 x 227 because the input image resolution for AlexNet is 227 x 227
                resized_img = img_array.resize((227, 227))

                img_lst.append(np.array(resized_img))
                labels.append(index)
    return img_lst, labels

images, labels = load_images_and_labels(categories)
print("No. of images loaded = ",len(images),"\nNo. of labels loaded = ",len(labels))
print(type(images),type(labels))
```

```
No. of images loaded =  4352
No. of labels loaded =  4352
<class 'list'> <class 'list'>
```

In [5]:
```
images = np.array(images)
labels = np.array(labels)

print("Images shape = ",images.shape,"\nLabels shape = ",labels.shape)
print(type(images),type(labels))
```

```
Images shape =  (4352, 227, 227, 3)
Labels shape =  (4352,)
<class 'numpy.ndarray'> <class 'numpy.ndarray'>
```

In [ ]:
```
def display_rand_images(images, labels):
    plt.figure(1 , figsize = (19 , 10))
    n = 0
    for i in range(9):
        n += 1
        r = np.random.randint(0 , images.shape[0] , 1)

        plt.subplot(3 , 3 , n)
        plt.subplots_adjust(hspace = 0.3 , wspace = 0.3)
        plt.imshow(images[r[0]])

        plt.title('Plant label : {}'.format(labels[r[0]]))
        plt.xticks([])
        plt.yticks([])

    plt.show()
accry=0.942
display_rand_images(images, labels)
```

In [7]:
```
#1-step in data shuffling

#get equally spaced numbers in a given range
n = np.arange(images.shape[0])
print("'n' values before shuffling = ",n)

#shuffle all the equally spaced values in list 'n'
np.random.seed(random_seed)
np.random.shuffle(n)
print("\n'n' values after shuffling = ",n)
```

```
'n' values before shuffling =  [   0    1    2 ... 4349 4350 4351]

'n' values after shuffling =  [3063 2450 3854 ... 4182 2004 3924]
```

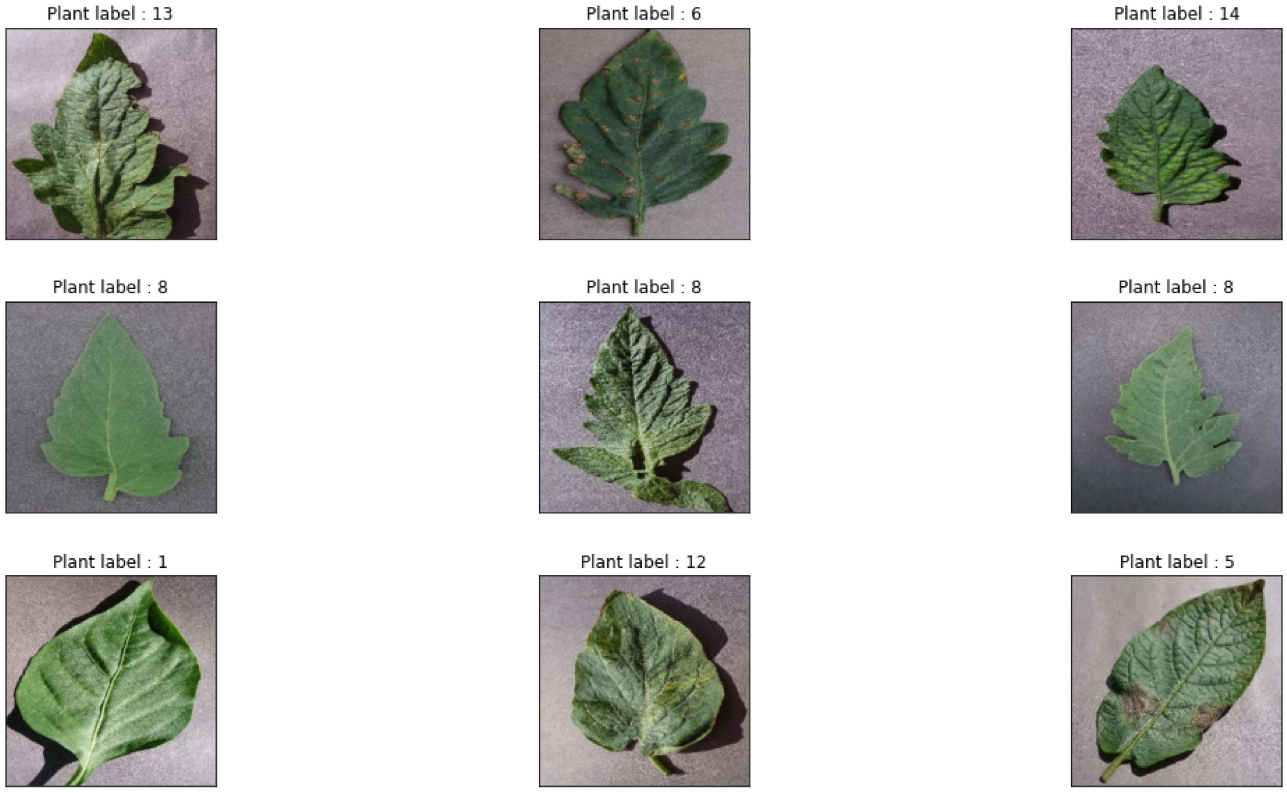
In [8]:
```
#2-step in data shuffling

#shuffle images and corresponding labels data in both the lists
images = images[n]
labels = labels[n]

print("Images shape after shuffling = ",images.shape,"\nLabels shape after shuffling = ",labels.shape)
```

```
Images shape after shuffling =  (4352, 227, 227, 3)
Labels shape after shuffling =  (4352,)
```

In [9]:
```
images = images.astype(np.float32)
labels = labels.astype(np.int32)
images = images/255
print("Images shape after normalization = ",images.shape)
```

```
Images shape after normalization =  (4352, 227, 227, 3)
```

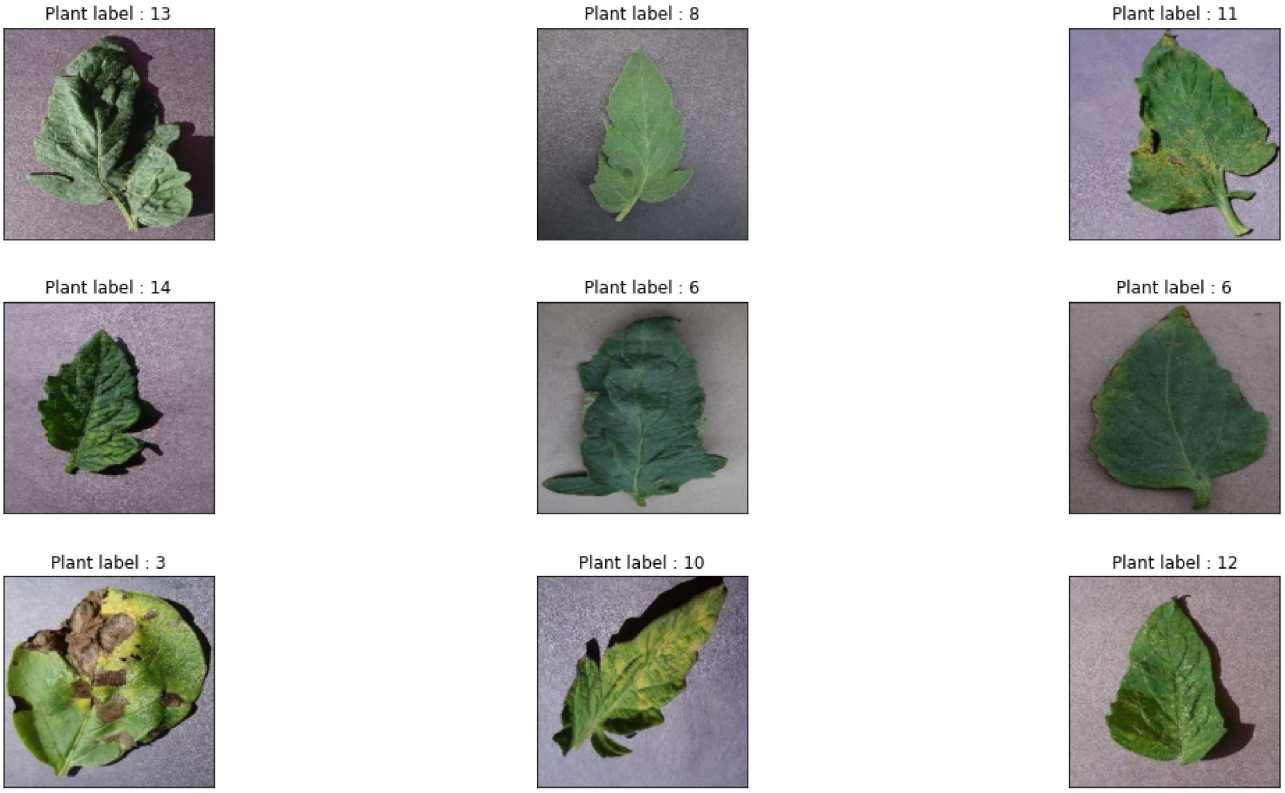
In [10]:
```
display_rand_images(images, labels)
```



In [11]:
```
x_train, x_test, y_train, y_test = train_test_split(images, labels, test_size = 0.2, random_state = random_seed)

print("x_train shape = ",x_train.shape)
print("y_train shape = ",y_train.shape)
print("\nx_test shape = ",x_test.shape)
print("y_test shape = ",y_test.shape)
```

```
x_train shape =  (3481, 227, 227, 3)
y_train shape =  (3481,)

x_test shape =  (871, 227, 227, 3)
y_test shape =  (871,)
```

In [12]:
```
display_rand_images(x_train, y_train)
```



In [13]:

```
model=Sequential()

#1 conv layer
model.add(Conv2D(filters=96,kernel_size=(11,11),strides=(4,4),padding="valid",activation="relu",input_shape=(227,227,3

#1 max pool layer
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

# model.add(BatchNormalization())

#2 conv layer
model.add(Conv2D(filters=256,kernel_size=(5,5),strides=(1,1),padding="valid",activation="relu"))

#2 max pool layer
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

# model.add(BatchNormalization())

#3 conv layer
model.add(Conv2D(filters=384,kernel_size=(3,3),strides=(1,1),padding="valid",activation="relu"))

#4 conv layer
model.add(Conv2D(filters=384,kernel_size=(3,3),strides=(1,1),padding="valid",activation="relu"))

#5 conv layer
model.add(Conv2D(filters=256,kernel_size=(3,3),strides=(1,1),padding="valid",activation="relu"))

#3 max pool layer
model.add(MaxPooling2D(pool_size=(3,3),strides=(2,2)))

# model.add(BatchNormalization())

model.add(Flatten())

#1 dense layer
model.add(Dense(4096,input_shape=(227,227,3),activation="relu"))

model.add(Dropout(0.4))

# model.add(BatchNormalization())

#2 dense layer
model.add(Dense(4096,activation="relu"))

model.add(Dropout(0.4))

# model.add(BatchNormalization())
#3 dense layer
model.add(Dense(1000,activation="relu"))

model.add(Dropout(0.4))

# model.add(BatchNormalization())

#output layer
model.add(Dense(20,activation="softmax"))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 55, 55, 96)        34944
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 27, 27, 96)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 23, 23, 256)       614656
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 11, 11, 256)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 9, 9, 384)         885120
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 7, 7, 384)         1327488
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 5, 5, 256)         884992
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 2, 2, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 4096)              4198400
_________________________________________________________________
dropout (Dropout)            (None, 4096)              0
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dropout_1 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_2 (Dense)              (None, 1000)              4097000
_________________________________________________________________
dropout_2 (Dropout)          (None, 1000)              0
_________________________________________________________________
dense_3 (Dense)              (None, 20)                20020
=================================================================
Total params: 28,843,932
Trainable params: 28,843,932
Non-trainable params: 0
_________________________________________________________________
```

In [14]:
```
model.compile(optimizer="adam", loss="sparse_categorical_crossentropy")
```

In [ ]:
```
model.fit(x_train, y_train, epochs=100)
```

```
Epoch 1/100
109/109 [==============================] - 240s 2s/step - loss: 2.7669
Epoch 2/100
109/109 [==============================] - 196s 2s/step - loss: 2.7190
Epoch 3/100
109/109 [==============================] - 177s 2s/step - loss: 2.7168
Epoch 4/100
109/109 [==============================] - 166s 2s/step - loss: 2.7116
Epoch 5/100
109/109 [==============================] - 158s 1s/step - loss: 2.7093
Epoch 6/100
109/109 [==============================] - 156s 1s/step - loss: 2.7088
Epoch 7/100
109/109 [==============================] - 157s 1s/step - loss: 2.7088
Epoch 8/100
109/109 [==============================] - 157s 1s/step - loss: 2.7071
Epoch 9/100
109/109 [==============================] - 150s 1s/step - loss: 2.7068
Epoch 10/100
109/109 [==============================] - 129s 1s/step - loss: 2.7078
Epoch 11/100
109/109 [==============================] - 133s 1s/step - loss: 2.7061
Epoch 12/100
109/109 [==============================] - 122s 1s/step - loss: 2.7053
Epoch 13/100
109/109 [==============================] - 132s 1s/step - loss: 2.7052
Epoch 14/100
109/109 [==============================] - 134s 1s/step - loss: 2.7048
Epoch 15/100
109/109 [==============================] - 132s 1s/step - loss: 2.7052
Epoch 16/100
109/109 [==============================] - 154s 1s/step - loss: 2.7057
Epoch 17/100
109/109 [==============================] - 156s 1s/step - loss: 2.7035
Epoch 18/100
109/109 [==============================] - 155s 1s/step - loss: 2.7028
Epoch 19/100
109/109 [==============================] - 149s 1s/step - loss: 2.7027
Epoch 20/100
109/109 [==============================] - 164s 2s/step - loss: 2.7036
Epoch 21/100
109/109 [==============================] - 159s 1s/step - loss: 2.7035
Epoch 22/100
109/109 [==============================] - 145s 1s/step - loss: 2.7035
Epoch 23/100
109/109 [==============================] - 136s 1s/step - loss: 2.7018
Epoch 24/100
109/109 [==============================] - 145s 1s/step - loss: 2.7026
Epoch 25/100
109/109 [==============================] - 139s 1s/step - loss: 2.7023
Epoch 26/100
109/109 [==============================] - 149s 1s/step - loss: 2.7035
Epoch 27/100
109/109 [==============================] - 135s 1s/step - loss: 2.7028
Epoch 28/100
109/109 [==============================] - 134s 1s/step - loss: 2.7011
Epoch 29/100
109/109 [==============================] - 132s 1s/step - loss: 2.7019
Epoch 30/100
109/109 [==============================] - 146s 1s/step - loss: 2.7016
Epoch 31/100
109/109 [==============================] - 144s 1s/step - loss: 2.7011
Epoch 32/100
109/109 [==============================] - 133s 1s/step - loss: 2.7023
Epoch 33/100
109/109 [==============================] - 121s 1s/step - loss: 2.7024
Epoch 34/100
109/109 [==============================] - 123s 1s/step - loss: 2.7016
Epoch 35/100
109/109 [==============================] - 100s 911ms/step - loss: 2.7012
Epoch 36/100
109/109 [==============================] - 94s 867ms/step - loss: 2.7025
Epoch 37/100
109/109 [==============================] - 93s 849ms/step - loss: 2.7012
Epoch 38/100
109/109 [==============================] - 96s 878ms/step - loss: 2.7010
Epoch 39/100
109/109 [==============================] - 91s 837ms/step - loss: 2.7010
Epoch 40/100
```

```
109/109 [==============================] - 92s 844ms/step - loss: 2.7006
Epoch 41/100
109/109 [==============================] - 93s 850ms/step - loss: 2.7016
Epoch 42/100
 84/109 [======================>.......] - ETA: 20s - loss: 2.6999
```

In [ ]:
```
loss = model.evaluate(x_test, y_test)

print(loss)
```

In [ ]:
```
pred = model.predict(x_test)

pred.shape
```

In [ ]:
```
plt.figure(1 , figsize = (19 , 10))
n = 0

for i in range(9):
    n += 1
    r = np.random.randint( 0, x_test.shape[0], 1)

    plt.subplot(3, 3, n)
    plt.subplots_adjust(hspace = 0.3, wspace = 0.3)

    plt.imshow(x_test[r[0]])
    plt.title('Actual = {}, Predicted = {}'.format(y_test[r[0]] , y_test[r[0]]*pred[r[0]][y_test[r[0]]]) )
    plt.xticks([]) , plt.yticks([])

plt.show()
```