# Plant village image classification

References are mentioned at the end of the notebook

## 1) Import required libraries

In [ ]:
```
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 tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D,MaxPooling2D,Dense,Flatten,Dropout
from tensorflow.keras.layers import BatchNormalization

print("Loaded required libraries...")
```

```
Loaded required libraries...
```

## 2) Data loading and exploration

In [ ]:
```
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_blight', 'Potato___health
y', 'Potato___Late_blight', 'Tomato_Bacterial_spot', 'Tomato_Early_blight', 'Tomato_healthy', 'Tomato_Late_blight', 'Tomato_Leaf_Mold', 'To
mato_Septoria_leaf_spot', 'Tomato_Spider_mites_Two_spotted_spider_mite', 'Tomato__Target_Spot', 'Tomato__Tomato_mosaic_virus', 'Tomato__Tom
ato_YellowLeaf__Curl_Virus']

No. of categories =  16
```

In [ ]:
```
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"):
                #print(f"\nCategory = {category}, Image name = {image_name}")
                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 [ ]:
```
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'>
```

- Display few random images from dataset with their label

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()

display_rand_images(images, labels)
```

Plant label : 0

Plant label : 2

Plant label : 12

Plant label : 2

Plant label : 9

Plant label : 5

Plant label : 2

Plant label : 10

Plant label : 3

## 3) Prepare data for CNN model training

- Step 1 - shuffle the data loaded from the dataset

In [ ]:
```
#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 [ ]:
```
#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,)
```

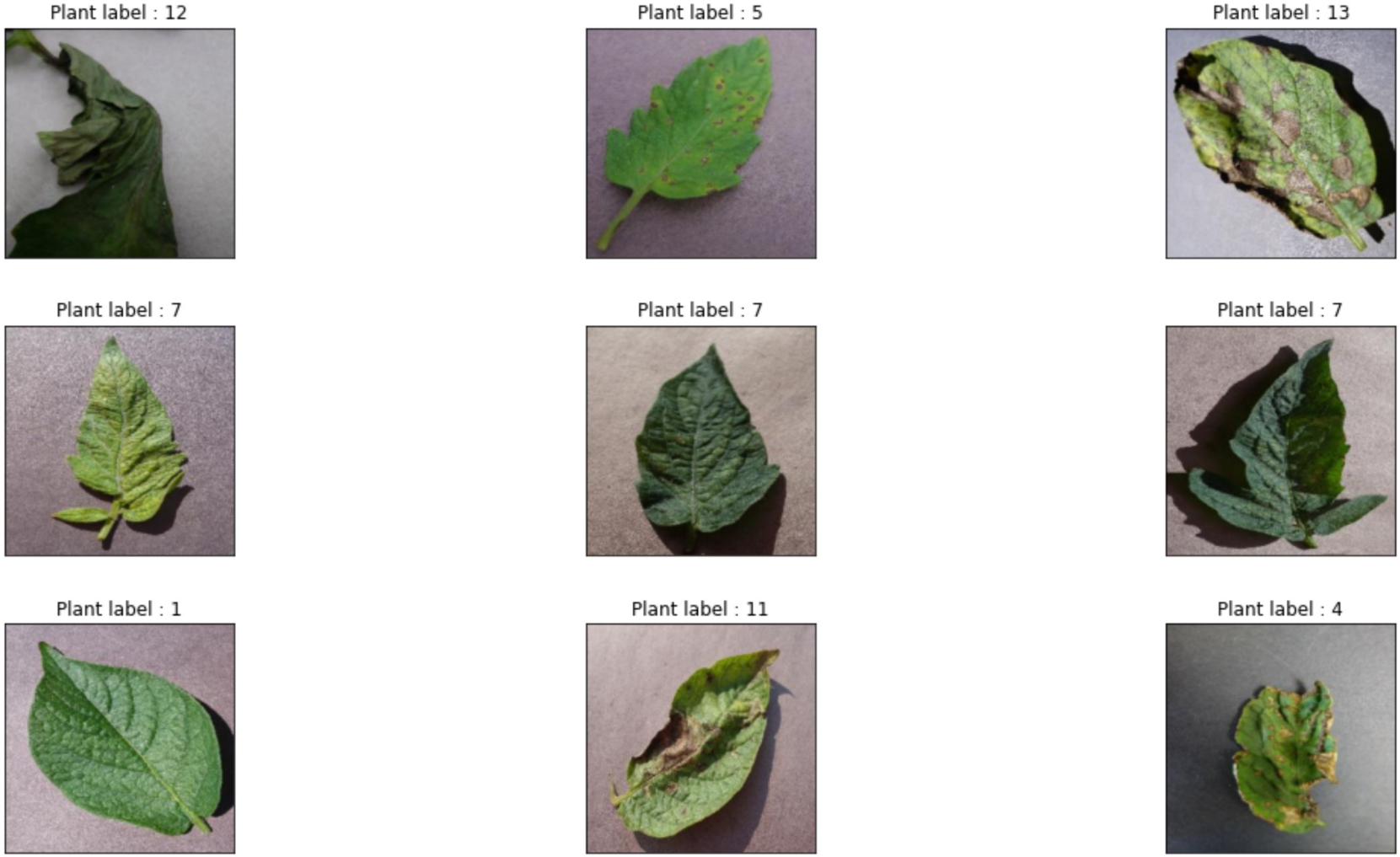
- Step 2 - Data normalization

In [ ]:
```
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)
```

- Display few random images after normalization

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




- Split dataset for training and testing

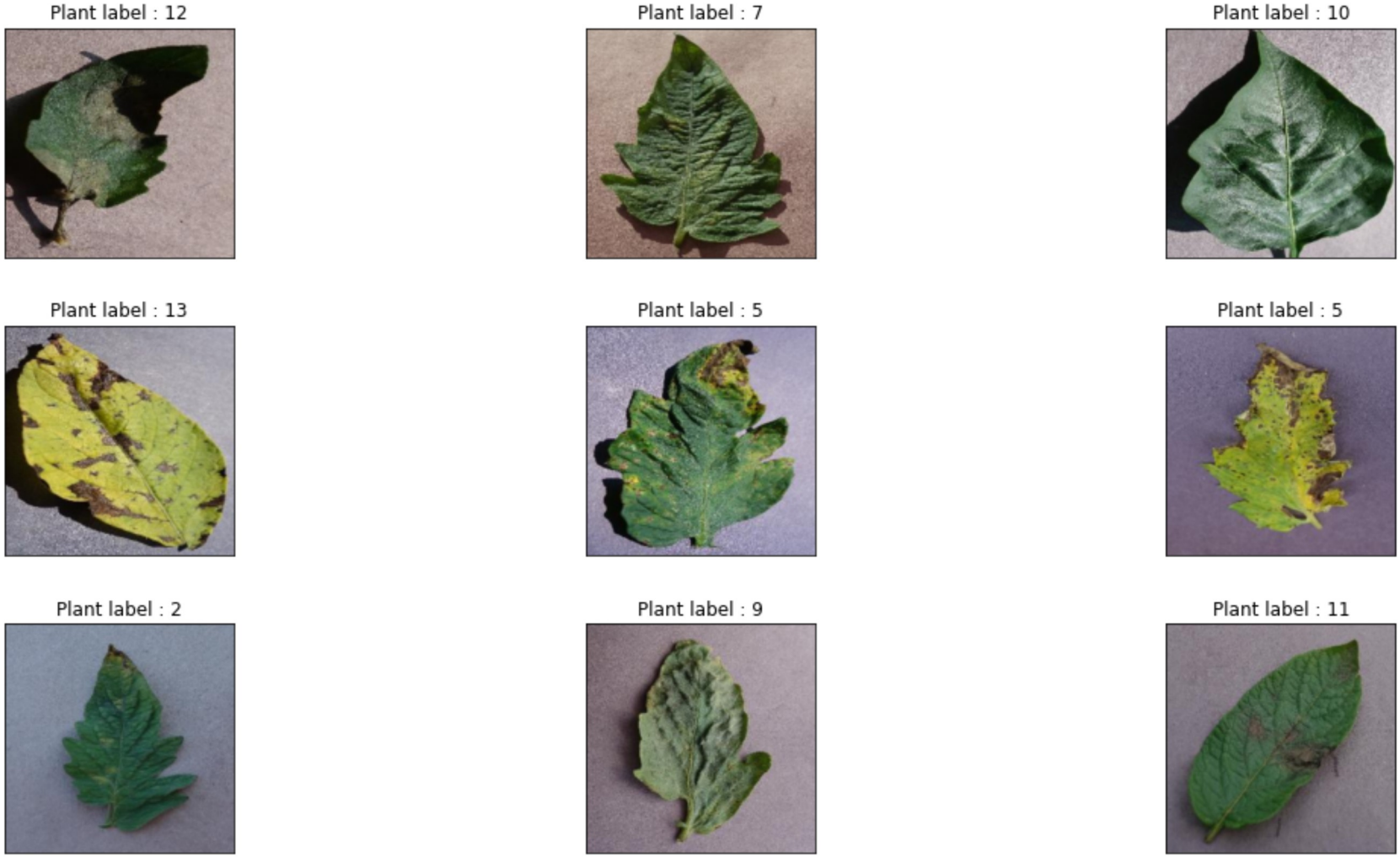
In [ ]:
```
x_train, x_test, y_train, y_test = train_test_split(images, labels, test_size = 0.2, random_state = random_seed)

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




## 4) Define CNN model (AlexNet)

In [ ]:
```
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
_________________________________________________________________
batch_normalization (BatchNo (None, 27, 27, 96)        384
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 23, 23, 256)       614656
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 11, 11, 256)       0
_________________________________________________________________
batch_normalization_1 (Batch (None, 11, 11, 256)       1024
_________________________________________________________________
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
_________________________________________________________________
batch_normalization_2 (Batch (None, 2, 2, 256)         1024
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 4096)              4198400
_________________________________________________________________
```

```
dropout (Dropout)            (None, 4096)              0
_________________________________________________________________
batch_normalization_3 (Batch (None, 4096)              16384
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dropout_1 (Dropout)          (None, 4096)              0
_________________________________________________________________
batch_normalization_4 (Batch (None, 4096)              16384
_________________________________________________________________
dense_2 (Dense)              (None, 1000)              4097000
_________________________________________________________________
dropout_2 (Dropout)          (None, 1000)              0
_________________________________________________________________
batch_normalization_5 (Batch (None, 1000)              4000
_________________________________________________________________
dense_3 (Dense)              (None, 20)                20020
=================================================================
Total params: 28,883,132
Trainable params: 28,863,532
Non-trainable params: 19,600
_________________________________________________________________
```

- Compile the defined CNN model

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

## 5) Train the model

- Fit the model using training data

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

```
Epoch 1/100
109/109 [==============================] - 7s 27ms/step - loss: 3.1355 - accuracy: 0.2086
Epoch 2/100
109/109 [==============================] - 3s 25ms/step - loss: 1.9581 - accuracy: 0.4197
Epoch 3/100
109/109 [==============================] - 3s 25ms/step - loss: 1.6485 - accuracy: 0.4981
Epoch 4/100
109/109 [==============================] - 3s 26ms/step - loss: 1.3238 - accuracy: 0.5746
Epoch 5/100
109/109 [==============================] - 3s 27ms/step - loss: 1.1764 - accuracy: 0.6192
Epoch 6/100
109/109 [==============================] - 3s 25ms/step - loss: 1.0125 - accuracy: 0.6687
Epoch 7/100
109/109 [==============================] - 3s 25ms/step - loss: 0.8640 - accuracy: 0.7046
Epoch 8/100
109/109 [==============================] - 3s 26ms/step - loss: 0.8843 - accuracy: 0.6948
Epoch 9/100
109/109 [==============================] - 3s 25ms/step - loss: 0.7443 - accuracy: 0.7563
Epoch 10/100
109/109 [==============================] - 3s 25ms/step - loss: 0.6855 - accuracy: 0.7712
Epoch 11/100
109/109 [==============================] - 3s 25ms/step - loss: 0.6411 - accuracy: 0.7795
Epoch 12/100
109/109 [==============================] - 3s 26ms/step - loss: 0.5514 - accuracy: 0.8116
Epoch 13/100
109/109 [==============================] - 3s 25ms/step - loss: 0.4595 - accuracy: 0.8372
Epoch 14/100
109/109 [==============================] - 3s 25ms/step - loss: 0.5483 - accuracy: 0.8137
Epoch 15/100
109/109 [==============================] - 3s 25ms/step - loss: 0.4187 - accuracy: 0.8508
Epoch 16/100
109/109 [==============================] - 3s 25ms/step - loss: 0.4810 - accuracy: 0.8396
Epoch 17/100
109/109 [==============================] - 3s 26ms/step - loss: 0.4158 - accuracy: 0.8460
Epoch 18/100
109/109 [==============================] - 3s 25ms/step - loss: 0.3173 - accuracy: 0.8900
Epoch 19/100
109/109 [==============================] - 3s 25ms/step - loss: 0.3642 - accuracy: 0.8786
Epoch 20/100
109/109 [==============================] - 3s 26ms/step - loss: 0.3309 - accuracy: 0.8832
Epoch 21/100
109/109 [==============================] - 3s 25ms/step - loss: 0.3034 - accuracy: 0.8982
Epoch 22/100
109/109 [==============================] - 3s 25ms/step - loss: 0.2557 - accuracy: 0.9116
Epoch 23/100
```

```
109/109 [==============================] - 3s 25ms/step - loss: 0.2943 - accuracy: 0.9034
Epoch 24/100
109/109 [==============================] - 3s 26ms/step - loss: 0.1805 - accuracy: 0.9425
Epoch 25/100
109/109 [==============================] - 3s 24ms/step - loss: 0.2327 - accuracy: 0.9225
Epoch 26/100
109/109 [==============================] - 3s 24ms/step - loss: 0.1768 - accuracy: 0.9414
Epoch 27/100
109/109 [==============================] - 3s 25ms/step - loss: 0.2509 - accuracy: 0.9160
Epoch 28/100
109/109 [==============================] - 3s 26ms/step - loss: 0.1763 - accuracy: 0.9437
Epoch 29/100
109/109 [==============================] - 3s 26ms/step - loss: 0.2314 - accuracy: 0.9247
Epoch 30/100
109/109 [==============================] - 3s 25ms/step - loss: 0.1763 - accuracy: 0.9407
Epoch 31/100
109/109 [==============================] - 3s 25ms/step - loss: 0.2444 - accuracy: 0.9196
Epoch 32/100
109/109 [==============================] - 3s 26ms/step - loss: 0.1755 - accuracy: 0.9446
Epoch 33/100
109/109 [==============================] - 3s 25ms/step - loss: 0.2734 - accuracy: 0.9128
Epoch 34/100
109/109 [==============================] - 3s 25ms/step - loss: 0.1761 - accuracy: 0.9407
Epoch 35/100
109/109 [==============================] - 3s 25ms/step - loss: 0.1631 - accuracy: 0.9472
Epoch 36/100
109/109 [==============================] - 3s 26ms/step - loss: 0.1491 - accuracy: 0.9491
Epoch 37/100
109/109 [==============================] - 3s 24ms/step - loss: 0.1429 - accuracy: 0.9547
Epoch 38/100
109/109 [==============================] - 3s 25ms/step - loss: 0.1544 - accuracy: 0.9480
Epoch 39/100
109/109 [==============================] - 3s 25ms/step - loss: 0.1160 - accuracy: 0.9640
Epoch 40/100
109/109 [==============================] - 3s 26ms/step - loss: 0.1639 - accuracy: 0.9453
Epoch 41/100
109/109 [==============================] - 3s 27ms/step - loss: 0.2049 - accuracy: 0.9346
Epoch 42/100
109/109 [==============================] - 3s 25ms/step - loss: 0.1098 - accuracy: 0.9654
Epoch 43/100
109/109 [==============================] - 3s 25ms/step - loss: 0.1603 - accuracy: 0.9438
Epoch 44/100
109/109 [==============================] - 3s 25ms/step - loss: 0.0888 - accuracy: 0.9732
Epoch 45/100
109/109 [==============================] - 3s 25ms/step - loss: 0.1404 - accuracy: 0.9470
```

```
Epoch 46/100
109/109 [==============================] - 3s 25ms/step - loss: 0.1522 - accuracy: 0.9502
Epoch 47/100
109/109 [==============================] - 3s 25ms/step - loss: 0.1005 - accuracy: 0.9674
Epoch 48/100
109/109 [==============================] - 3s 25ms/step - loss: 0.1167 - accuracy: 0.9623
Epoch 49/100
109/109 [==============================] - 3s 24ms/step - loss: 0.1134 - accuracy: 0.9615
Epoch 50/100
109/109 [==============================] - 3s 25ms/step - loss: 0.0912 - accuracy: 0.9719
Epoch 51/100
109/109 [==============================] - 3s 25ms/step - loss: 0.1125 - accuracy: 0.9619
Epoch 52/100
109/109 [==============================] - 3s 26ms/step - loss: 0.1029 - accuracy: 0.9652
Epoch 53/100
109/109 [==============================] - 3s 27ms/step - loss: 0.0825 - accuracy: 0.9710
Epoch 54/100
109/109 [==============================] - 3s 25ms/step - loss: 0.1140 - accuracy: 0.9668
Epoch 55/100
109/109 [==============================] - 3s 25ms/step - loss: 0.1322 - accuracy: 0.9599
Epoch 56/100
109/109 [==============================] - 3s 26ms/step - loss: 0.1003 - accuracy: 0.9692
Epoch 57/100
109/109 [==============================] - 3s 25ms/step - loss: 0.1286 - accuracy: 0.9644
Epoch 58/100
109/109 [==============================] - 3s 25ms/step - loss: 0.1309 - accuracy: 0.9579
Epoch 59/100
109/109 [==============================] - 3s 25ms/step - loss: 0.0922 - accuracy: 0.9732
Epoch 60/100
109/109 [==============================] - 3s 26ms/step - loss: 0.0795 - accuracy: 0.9750
Epoch 61/100
109/109 [==============================] - 3s 25ms/step - loss: 0.0719 - accuracy: 0.9727
Epoch 62/100
109/109 [==============================] - 3s 25ms/step - loss: 0.0907 - accuracy: 0.9739
Epoch 63/100
109/109 [==============================] - 3s 25ms/step - loss: 0.0998 - accuracy: 0.9678
Epoch 64/100
109/109 [==============================] - 3s 27ms/step - loss: 0.0727 - accuracy: 0.9718
Epoch 65/100
109/109 [==============================] - 3s 27ms/step - loss: 0.0776 - accuracy: 0.9764
Epoch 66/100
109/109 [==============================] - 3s 25ms/step - loss: 0.0687 - accuracy: 0.9758
Epoch 67/100
109/109 [==============================] - 3s 26ms/step - loss: 0.0604 - accuracy: 0.9768
Epoch 68/100
```

```
109/109 [==============================] - 3s 26ms/step - loss: 0.0618 - accuracy: 0.9796
Epoch 69/100
109/109 [==============================] - 3s 25ms/step - loss: 0.0945 - accuracy: 0.9696
Epoch 70/100
109/109 [==============================] - 3s 26ms/step - loss: 0.0897 - accuracy: 0.9711
Epoch 71/100
109/109 [==============================] - 3s 26ms/step - loss: 0.1039 - accuracy: 0.9679
Epoch 72/100
109/109 [==============================] - 3s 26ms/step - loss: 0.0842 - accuracy: 0.9738
Epoch 73/100
109/109 [==============================] - 3s 25ms/step - loss: 0.0448 - accuracy: 0.9847
Epoch 74/100
109/109 [==============================] - 3s 25ms/step - loss: 0.0801 - accuracy: 0.9766
Epoch 75/100
109/109 [==============================] - 3s 26ms/step - loss: 0.0901 - accuracy: 0.9684
Epoch 76/100
109/109 [==============================] - 3s 26ms/step - loss: 0.0576 - accuracy: 0.9844
Epoch 77/100
109/109 [==============================] - 3s 31ms/step - loss: 1.0804 - accuracy: 0.7546
Epoch 78/100
109/109 [==============================] - 3s 25ms/step - loss: 0.2156 - accuracy: 0.9270
Epoch 79/100
109/109 [==============================] - 3s 26ms/step - loss: 0.1270 - accuracy: 0.9557
Epoch 80/100
109/109 [==============================] - 3s 25ms/step - loss: 0.2541 - accuracy: 0.9221
Epoch 81/100
109/109 [==============================] - 3s 25ms/step - loss: 0.1073 - accuracy: 0.9645
Epoch 82/100
109/109 [==============================] - 3s 25ms/step - loss: 0.1437 - accuracy: 0.9533
Epoch 83/100
109/109 [==============================] - 3s 26ms/step - loss: 0.1149 - accuracy: 0.9579
Epoch 84/100
109/109 [==============================] - 3s 25ms/step - loss: 0.0779 - accuracy: 0.9750
Epoch 85/100
109/109 [==============================] - 3s 25ms/step - loss: 0.0724 - accuracy: 0.9785
Epoch 86/100
109/109 [==============================] - 3s 25ms/step - loss: 0.0561 - accuracy: 0.9847
Epoch 87/100
109/109 [==============================] - 3s 26ms/step - loss: 0.0602 - accuracy: 0.9797
Epoch 88/100
109/109 [==============================] - 3s 26ms/step - loss: 0.0415 - accuracy: 0.9859
Epoch 89/100
109/109 [==============================] - 3s 29ms/step - loss: 0.0551 - accuracy: 0.9844
Epoch 90/100
109/109 [==============================] - 3s 25ms/step - loss: 0.0624 - accuracy: 0.9836
```

```
Epoch 91/100
109/109 [==============================] - 3s 26ms/step - loss: 0.0357 - accuracy: 0.9883
Epoch 92/100
109/109 [==============================] - 3s 25ms/step - loss: 0.0509 - accuracy: 0.9828
Epoch 93/100
109/109 [==============================] - 3s 25ms/step - loss: 0.0583 - accuracy: 0.9801
Epoch 94/100
109/109 [==============================] - 3s 25ms/step - loss: 0.0321 - accuracy: 0.9888
Epoch 95/100
109/109 [==============================] - 3s 26ms/step - loss: 0.0565 - accuracy: 0.9801
Epoch 96/100
109/109 [==============================] - 3s 25ms/step - loss: 0.0579 - accuracy: 0.9797
Epoch 97/100
109/109 [==============================] - 3s 25ms/step - loss: 0.0712 - accuracy: 0.9761
Epoch 98/100
109/109 [==============================] - 3s 25ms/step - loss: 0.0485 - accuracy: 0.9854
Epoch 99/100
109/109 [==============================] - 3s 26ms/step - loss: 0.0783 - accuracy: 0.9778
Epoch 100/100
109/109 [==============================] - 3s 28ms/step - loss: 0.0374 - accuracy: 0.9902
```

Out[ ]: <tensorflow.python.keras.callbacks.History at 0x7f89a9f9c790>

- Metrics to evaluate accuracy and loss in test dataset

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

print(loss,accuracy)
```

```
28/28 [==============================] - 1s 16ms/step - loss: 1.1856 - accuracy: 0.8071
1.1855746507644653 0.8071182370185852
```

## 6) Predicting values using trained model

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

pred.shape
```

Out[ ]: (871, 20)

- Display few random images with actual vs predicted values of labels

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()
```

Actual = 5, Predicted = 5.0

Actual = 13, Predicted = 12.999728798866272

Actual = 13, Predicted = 12.997785449028015

Actual = 8, Predicted = 8.0

Actual = 2, Predicted = 0.3237985074520111

Actual = 4, Predicted = 3.999699592590332

Actual = 7, Predicted = 6.958945572376251

Actual = 0, Predicted = 0.0

Actual = 0, Predicted = 0.0

## 7) Save trained model, weights

In [ ]:
```
# save model in JSON format
model_json = model.to_json()
json_file = open("../working/model1.json", "w")
json_file.write(model_json)
print("Model saved in JSON format!")

# save training weights in h5 file
model.save_weights("../working/model1.h5")
print("\nModel weights saved!")
```

```
Model saved in JSON format!

Model weights saved!
```

In [ ]:
```
%cd /kaggle/working
from IPython.display import FileLink
FileLink(r'model1.h5')
#ref - https://www.kaggle.com/getting-started/168312
```

```
/kaggle/working
```

Out[ ]: model1.h5