Motive behind Version:

This version of the code runs a similar neural network as the origninal code. However the data used only consists of 9 classes (typed phrases/passwords). Additionally this dataset only contains high quality thermal images to emulate a more controlled setup. This is done to validate the efficiency of the neural network itself and conclude that the poorer accuracy rates in the original version is a result of the data and the not the quality of the Neural Network

Part 1 - Importing Necessary libraries

Here we import all the necessary libraries

```
#Load libraries
import os
import numpy as np
import torch
import glob
import torch.nn as nn
from torchvision.transforms import transforms
from torch.utils.data import DataLoader
from torch.optim import Adam
from torch.autograd import Variable
import torchvision
import pathlib
from pandas import read_csv
from __future__ import print_function
import pandas as pd
import shutil
import os
import sys
from os.path import join
```

Part 2 - Importing the necessary image data and CSV files

This section has the following functions -

- First we get the labels of the data from the csv file.
- The obtained labels are then separated into Training and testing sets (8:2)
- A small check is done to see whether the training and testing sets have the same number of unique classes.
- The images are now separated into two different folders based on training and testing lists.

```python
map_file = read_csv('/content/drive/MyDrive/Colab Notebooks/Thermal
Image/Version 4 - Small Dataset/Train.csv')
length = len(map_file)
print(length)
labels = set()
for i in range(len(map_file)):
    # convert spaced separated tags into an array of tags
    type = map_file['Password'][i].split('\n')
    # add tags to the set of known labels
    labels.update(type)
#print(labels)
labels = list(labels)
labels.sort()
classes = labels
classes
```

```
126
```

```
['1?wb:J5c6X[-5az7',
 '3}fW8&tR4bmdY_b7',
 'End by win a pun!',
 'Rh(3q4aRmWx!R)e',
 'The big dwarf only jumps?',
 'The world is a stage!',
 'gareth_king@post.cd',
 'gwen_king@post.com',
 'kyle_king@post.uk']
```

```python
train_dir =r'/content/drive/MyDrive/Colab Notebooks/Thermal
Image/Version 4 - Small Dataset/Image Data'
test_dir =r'/content/drive/MyDrive/Colab Notebooks/Thermal
Image/Version 4 - Small Dataset/Image Data'
#Please make sure to change the path in the above line during testing
when testing
train_path = r"train_label"
test_path =  r"test_label"
if not os.path.exists(train_path):
    os.mkdir(train_path)

if not os.path.exists(test_path):
    os.mkdir(test_path)
```

Disclaimer during testing : Please check the path of the training and testing directory, and the CSV files. As mentioned in the user guide, please copy the path from your drive for the directory containing the images and the CSV files.

##**Subpart 1**

The images are divided into training and testing directories depending on the training and testing labels.

```
for i in range(2):
  ran = np.random.RandomState()
  training_labels = map_file.sample(frac=0.8, random_state=ran)
  testing_labels =
map_file.loc[~map_file.index.isin(training_labels.index)]
  training_label_class = []
  testing_label_class = []
  for File_Name, Class_Name in training_labels.values:
    if Class_Name not in training_label_class:
        training_label_class.append(Class_Name)


  for File_Name, Class_Name in testing_labels.values:
    if Class_Name not in testing_label_class:
        testing_label_class.append(Class_Name)

  for File_Name, Class_Name in training_labels.values:
    # Create subdirectory with `Class_Name`
    if not os.path.exists(train_path + '/'+ str(Class_Name)):
        os.mkdir(train_path + '/'+ str(Class_Name))
    src_path = train_dir + '/'+ str(File_Name) + '.jpg'
    dst_path = train_path + '/'+ str(Class_Name) + '/' +
str(File_Name) + '.jpg'
    try:
        shutil.copy(src_path, dst_path)
        print("File Transfer Successful")
    except:
        print('Error')

  for File_Name, Class_Name in testing_labels.values:
    # Create subdirectory with `Class_Name`
    if not os.path.exists(test_path + '/'+ str(Class_Name)):
        os.mkdir(test_path + '/'+ str(Class_Name))
    src_path = test_dir + '/'+ str(File_Name) + '.jpg'
    dst_path = test_path + '/'+ str(Class_Name) + '/' + str(File_Name)
+ '.jpg'
    try:
        shutil.copy(src_path, dst_path)
        print("File Transfer Successful")
    except:
        print('Error')
```

```
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
```

File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful

```
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
```

```
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
```

File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful

```
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
File Transfer Successful
```

Part 3 - Designing the Neural Network

Now we start designing the Neural Network. We chose to use Convolutional Neural Network which was designed using the PyTorch Library.

Subpart 1

- We define a transfomer and that resizes the image and transforms the file to tensor
- We also check if the device is running on cuda or CPU

```
use_cuda = True
device = torch.device("cuda" if (use_cuda and 
torch.cuda.is_available()) else "cpu")
print("device type : ",device)
transformer=transforms.Compose([
    transforms.Resize((64,64)),
    transforms.ToTensor(),
])
```

```
device type :  cpu
```

Subpart 2

We now load the data

```
Dataloader_training=DataLoader(
    
torchvision.datasets.ImageFolder(train_path,transform=transformer),
    batch_size=8, shuffle=True
)
Dataloader_testing=DataLoader(
    torchvision.datasets.ImageFolder(test_path,transform=transformer),
    batch_size=4, shuffle=True
)
```

Subpart 3 - The CNN Network

Unlike the network with the original dataset, this one only classifies 9 labels.

```
class ConvNet(nn.Module):
    def __init__(self,num_classes=9):
        super(ConvNet,self).__init__()
        
        
self.conv1=nn.Conv2d(in_channels=3,out_channels=10,kernel_size=3,strid
e=1,padding=1)
        self.bn1=nn.BatchNorm2d(num_features=10)
        self.relu1=nn.ReLU()
        
        self.pool=nn.MaxPool2d(kernel_size=2)
```

```
self.conv2=nn.Conv2d(in_channels=10,out_channels=24,kernel_size=3,stri
de=1,padding=1)
        self.relu2=nn.ReLU()


self.conv3=nn.Conv2d(in_channels=24,out_channels=32,kernel_size=3,stri
de=1,padding=1)
        self.bn3=nn.BatchNorm2d(num_features=32)
        self.relu3=nn.ReLU()
        self.fc=nn.Linear(in_features=32 **3,out_features=num_classes)

        #Feed forwad function

    def forward(self,input):
        output=self.conv1(input)
        output=self.bn1(output)
        output=self.relu1(output)

        output=self.pool(output)

        output=self.conv2(output)
        output=self.relu2(output)

        output=self.conv3(output)
        output=self.bn3(output)
        output=self.relu3(output)

        output=output.view(-1,32**3)

        output=self.fc(output)

        return output
```

```
model=ConvNet(num_classes=9).to(device)
```

```
import torch.optim as optim
optimizer = optim.SGD(model.parameters(), lr=0.001)
loss_fn = nn.CrossEntropyLoss()
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sn
import pandas as pd
epoch_count = 50


train_count=len(glob.glob(train_path+'/**/*.jpg'))
test_count=len(glob.glob(test_path+'/**/*.jpg'))
```

Part 4 - Training and Testing the Network

The network is trained and tested 50 times and the best result in terms of accuracy is retained.

```
best_accuracy=0.0
predlist_final=[]
lbllist_final=[]

for epoch in range(epoch_count):
    predlist=[]
    lbllist=[]
    #Evaluation and training on training dataset
    model.train()
    train_accuracy=0.0
    train_loss=0.0

    for i, (images,labels) in enumerate(Dataloader_training):
        if torch.cuda.is_available():
            images=Variable(images.cuda())
            labels=Variable(labels.cuda())

        optimizer.zero_grad()

        outputs=model(images)
        loss=loss_fn(outputs,labels)
        loss.backward()
        optimizer.step()


        train_loss+= loss.cpu().data*images.size(0)
        _,prediction=torch.max(outputs.data,1)

        train_accuracy+=int(torch.sum(prediction==labels.data))

    train_accuracy=train_accuracy/train_count
    train_loss=train_loss/train_count


    # Evaluation on testing dataset
    model.eval()

    test_accuracy=0.0
    for i, (images,labels) in enumerate(Dataloader_testing):
        if torch.cuda.is_available():
            images=Variable(images.cuda())
            labels=Variable(labels.cuda())

        outputs=model(images)
        _,prediction=torch.max(outputs.data,1)
```

```
        test_accuracy+=int(torch.sum(prediction==labels.data))
        predlist.extend(prediction)
        lbllist.extend(labels.data)
    test_accuracy=test_accuracy/test_count


    print('Iteration: '+str(epoch)+'\nTrain Loss: '+str(train_loss)+'\
nTrain Accuracy: '+str(train_accuracy)+'\nTest Accuracy: 
'+str(test_accuracy))
    print("\n\n")
    #Save the best model
    if test_accuracy>best_accuracy:
        torch.save(model.state_dict(),'best_checkpoint.model')
        best_accuracy=test_accuracy
        predlist_final.clear()
        lbllist_final.clear()
        predlist_final = predlist[:]
        lbllist_final = lbllist
```

```
Iteration: 0
Train Loss: tensor(2.9589)
Train Accuracy: 0.08943089430894309
Test Accuracy: 0.19148936170212766



Iteration: 1
Train Loss: tensor(2.3936)
Train Accuracy: 0.2032520325203252
Test Accuracy: 0.1276595744680851



Iteration: 2
Train Loss: tensor(2.0895)
Train Accuracy: 0.2845528455284553
Test Accuracy: 0.1276595744680851



Iteration: 3
Train Loss: tensor(1.7864)
Train Accuracy: 0.43089430894308944
Test Accuracy: 0.2978723404255319



Iteration: 4
Train Loss: tensor(1.6785)
```

```
Train Accuracy: 0.4715447154471545
Test Accuracy: 0.7021276595744681



Iteration: 5
Train Loss: tensor(1.3284)
Train Accuracy: 0.6422764227642277
Test Accuracy: 0.5531914893617021



Iteration: 6
Train Loss: tensor(1.2943)
Train Accuracy: 0.6178861788617886
Test Accuracy: 0.6808510638297872



Iteration: 7
Train Loss: tensor(1.0907)
Train Accuracy: 0.7317073170731707
Test Accuracy: 0.6170212765957447



Iteration: 8
Train Loss: tensor(0.9347)
Train Accuracy: 0.7804878048780488
Test Accuracy: 0.7659574468085106



Iteration: 9
Train Loss: tensor(0.8870)
Train Accuracy: 0.8130081300813008
Test Accuracy: 0.8297872340425532



Iteration: 10
Train Loss: tensor(0.7545)
Train Accuracy: 0.8373983739837398
Test Accuracy: 0.851063829787234



Iteration: 11
```

```
Train Loss: tensor(0.7028)
Train Accuracy: 0.8211382113821138
Test Accuracy: 0.8085106382978723



Iteration: 12
Train Loss: tensor(0.6368)
Train Accuracy: 0.8617886178861789
Test Accuracy: 0.8085106382978723



Iteration: 13
Train Loss: tensor(0.5817)
Train Accuracy: 0.8699186991869918
Test Accuracy: 0.9361702127659575



Iteration: 14
Train Loss: tensor(0.5013)
Train Accuracy: 0.926829268292683
Test Accuracy: 0.7659574468085106



Iteration: 15
Train Loss: tensor(0.5109)
Train Accuracy: 0.9186991869918699
Test Accuracy: 0.8936170212765957



Iteration: 16
Train Loss: tensor(0.4389)
Train Accuracy: 0.9186991869918699
Test Accuracy: 0.9361702127659575



Iteration: 17
Train Loss: tensor(0.3593)
Train Accuracy: 0.975609756097561
Test Accuracy: 0.9148936170212766
```

```
Iteration: 18
Train Loss: tensor(0.3187)
Train Accuracy: 0.975609756097561
Test Accuracy: 0.8723404255319149

Iteration: 19
Train Loss: tensor(0.3309)
Train Accuracy: 0.959349593495935
Test Accuracy: 0.8723404255319149

Iteration: 20
Train Loss: tensor(0.3003)
Train Accuracy: 0.943089430894309
Test Accuracy: 0.9361702127659575

Iteration: 21
Train Loss: tensor(0.2681)
Train Accuracy: 0.991869918699187
Test Accuracy: 0.9148936170212766

Iteration: 22
Train Loss: tensor(0.2396)
Train Accuracy: 0.991869918699187
Test Accuracy: 0.9361702127659575

Iteration: 23
Train Loss: tensor(0.2174)
Train Accuracy: 0.991869918699187
Test Accuracy: 0.9361702127659575

Iteration: 24
Train Loss: tensor(0.2156)
Train Accuracy: 0.991869918699187
Test Accuracy: 0.9361702127659575
```

```
Iteration: 25
Train Loss: tensor(0.1869)
Train Accuracy: 0.991869918699187
Test Accuracy: 0.9361702127659575

Iteration: 26
Train Loss: tensor(0.1832)
Train Accuracy: 0.991869918699187
Test Accuracy: 0.9361702127659575

Iteration: 27
Train Loss: tensor(0.1786)
Train Accuracy: 0.991869918699187
Test Accuracy: 0.9361702127659575

Iteration: 28
Train Loss: tensor(0.1596)
Train Accuracy: 1.0
Test Accuracy: 0.9361702127659575

Iteration: 29
Train Loss: tensor(0.1477)
Train Accuracy: 0.991869918699187
Test Accuracy: 0.9361702127659575

Iteration: 30
Train Loss: tensor(0.1417)
Train Accuracy: 1.0
Test Accuracy: 0.9361702127659575

Iteration: 31
Train Loss: tensor(0.1223)
Train Accuracy: 1.0
Test Accuracy: 0.9361702127659575
```

```
Iteration: 32
Train Loss: tensor(0.1335)
Train Accuracy: 1.0
Test Accuracy: 0.9361702127659575


Iteration: 33
Train Loss: tensor(0.1174)
Train Accuracy: 1.0
Test Accuracy: 0.9361702127659575


Iteration: 34
Train Loss: tensor(0.1065)
Train Accuracy: 1.0
Test Accuracy: 0.9361702127659575


Iteration: 35
Train Loss: tensor(0.1083)
Train Accuracy: 1.0
Test Accuracy: 0.9361702127659575


Iteration: 36
Train Loss: tensor(0.1009)
Train Accuracy: 1.0
Test Accuracy: 0.9361702127659575


Iteration: 37
Train Loss: tensor(0.0981)
Train Accuracy: 1.0
Test Accuracy: 0.9361702127659575


Iteration: 38
Train Loss: tensor(0.1009)
Train Accuracy: 1.0
Test Accuracy: 0.9361702127659575
```

```
Iteration: 39
Train Loss: tensor(0.0992)
Train Accuracy: 1.0
Test Accuracy: 0.9361702127659575



Iteration: 40
Train Loss: tensor(0.0993)
Train Accuracy: 1.0
Test Accuracy: 0.9361702127659575



Iteration: 41
Train Loss: tensor(0.0816)
Train Accuracy: 1.0
Test Accuracy: 0.9361702127659575



Iteration: 42
Train Loss: tensor(0.0814)
Train Accuracy: 1.0
Test Accuracy: 0.9361702127659575



Iteration: 43
Train Loss: tensor(0.0755)
Train Accuracy: 1.0
Test Accuracy: 0.9361702127659575



Iteration: 44
Train Loss: tensor(0.0820)
Train Accuracy: 1.0
Test Accuracy: 0.9361702127659575



Iteration: 45
Train Loss: tensor(0.0679)
Train Accuracy: 1.0
```

```
Test Accuracy: 0.9361702127659575


Iteration: 46
Train Loss: tensor(0.0760)
Train Accuracy: 1.0
Test Accuracy: 0.9361702127659575


Iteration: 47
Train Loss: tensor(0.0716)
Train Accuracy: 1.0
Test Accuracy: 0.9361702127659575


Iteration: 48
Train Loss: tensor(0.0672)
Train Accuracy: 1.0
Test Accuracy: 0.9361702127659575


Iteration: 49
Train Loss: tensor(0.0669)
Train Accuracy: 1.0
Test Accuracy: 0.9361702127659575
```

Part 5 - Results and Visualization

This part is divided into the following part -

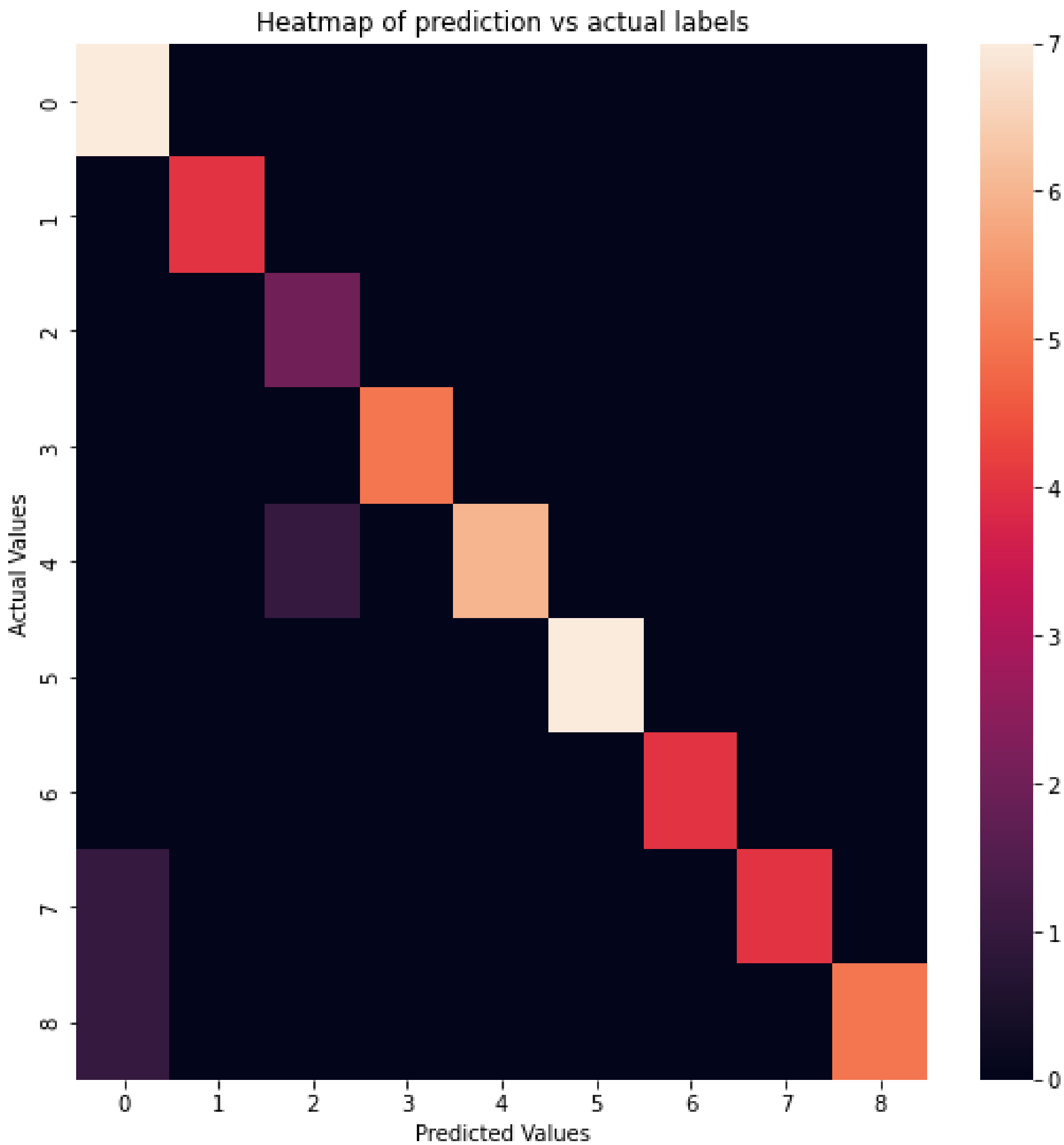
- Display accuracy
- Show confusion matrix
- Calculate Precision and Recall
- Display all results

```
print("The highest accuracy among all iteration is " +
str(best_accuracy*100) + " %")
```

```
The highest accuracy among all iteration is 93.61702127659575 %
```

```python
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix,
ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import seaborn as sns
conmat = confusion_matrix(lbllist_final, predlist_final)
#disp = ConfusionMatrixDisplay(confusion_matrix=conmat)
#disp.plot()
#plt.show()
print(conmat)
fig, ax = plt.subplots(figsize=(9,9))
sns.heatmap(conmat)
plt.xlabel('Predicted Values')
plt.ylabel('Actual Values')
ax.set_title('Heatmap of prediction vs actual labels')
plt.show()
```

```
[[7 0 0 0 0 0 0 0 0]
 [0 4 0 0 0 0 0 0 0]
 [0 0 2 0 0 0 0 0 0]
 [0 0 0 5 0 0 0 0 0]
 [0 0 1 0 6 0 0 0 0]
 [0 0 0 0 0 7 0 0 0]
 [0 0 0 0 0 0 4 0 0]
 [1 0 0 0 0 0 0 4 0]
 [1 0 0 0 0 0 0 0 5]]
```




```
FP = conmat.sum(axis=0) - np.diag(conmat)
FN = conmat.sum(axis=1) - np.diag(conmat)
TP = np.diag(conmat)
TN = conmat.sum() - (FP + FN + TP)

print(FP)
print(FN)
print(TP)
print(TN)
```

```
[2 0 1 0 0 0 0 0 0]
[0 0 0 0 1 0 0 1 1]
[7 4 2 5 6 7 4 4 5]
[38 43 44 42 40 40 43 42 41]
```

```python
def Precision (TP, FP):
  ans = 0
  total = 0
  for i in range(len(TP)):
    total +=1
    ans += (TP[i])/(TP[i]+FP[i])
  return ans/total

def Recall (TP, FN):
  ans = 0
  total = 0
  for i in range(len(TP)):
    total +=1
    ans += (TP[i])/(TP[i]+FN[i])
  return ans/total

print("Accuracy of the best model : ", str(best_accuracy*100) + " %")
print("Average Precision of best model :", str(Precision(TP,FP)*100) +
"%")
print("Average Recall of best model :", str(Recall(TP,FN)*100), "%")
```

```
Accuracy of the best model :  93.61702127659575 %
Average Precision of best model : 93.82716049382717%
Average Recall of best model : 94.33862433862434 %
```

Disclaimer - Due to the Training and Testing Set Split being randomized, the accuracy will differ slightly during different uses of this model.

Citations

This code was created with the help of several online tutorials and books which are listed below -

- Sewak, Mohit, Md Rezaul Karim, and Pradeep Pujari. Practical convolutional neural networks: implement advanced deep learning models using Python. Packt Publishing Ltd, 2018.
- https://www.youtube.com/watch?v=9OHlgDjaE2I
- https://numpy.org/doc/1.16/reference/generated/numpy.random.RandomState.html
- https://www.youtube.com/watch?v=7q7E91pHoW4&list=PLqnslRFeH2UrcDBWF5mfPGpqQDSta6VK4&index=11&ab_channel=PythonEngineer

- https://www.youtube.com/watch?v=pDdP0TFzsoQ&list=PLqnslRFeH2UrcDBWF5mfPGpqQDSta6VK4&index=14&ab_channel=PythonEngineer