The data consists of 48x48 pixel grayscale images of faces. The faces have been automatically registered so that the face is more or less centered and occupies about the same amount of space in each image. The task is to categorize each face based on the emotion shown in the facial expression in to one of seven categories (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral).

The dataset consists of 35,887 examples.This dataset was prepared by Pierre-Luc Carrier and Aaron Courville, as part of an ongoing research project.

We will be using Pytorch package to create the Convolutional Neural Network model to classify the images and various metrics like Confusion matrix and plots will b eused to evaluate the performance of the model and at the end we will be testing the model by few images taken from google

Ref: https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/data (https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge/data)

Importing the Necessary Libraries

In [218]:

```
import torch
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torch.nn as nn
import torch.nn.functional as F
from torchvision import datasets, transforms
from torch.utils.data import TensorDataset, DataLoader
```

In [2]:

```
df = pd.read_csv(r'C:\FER\fer2013\fer2013.csv')
```

In [3]:

```
df.head()
```

Out[3]:

	emotion	pixels	Usage
0	0	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...	Training
1	0	151 150 147 155 148 133 111 140 170 174 182 15...	Training
2	2	231 212 156 164 174 138 161 173 182 200 106 38...	Training
3	4	24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 1...	Training
4	6	4 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84...	Training

In [4]:

```
df.shape
```

Out[4]:

```
(35887, 3)
```

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35887 entries, 0 to 35886
Data columns (total 3 columns):
emotion    35887 non-null int64
pixels     35887 non-null object
Usage      35887 non-null object
dtypes: int64(1), object(2)
memory usage: 841.2+ KB
```

In [6]:

```
#converting the string values of pixels into array
def arr_conv (strg):
    b=[]
    a=strg.split(" ")
    for elements in a:
        b.append(int (elements))

    return np.array(b)
```

In [7]:

```
df["pixels_arr"]= df['pixels'].apply(arr_conv)
```

In [8]:

```
min(df["pixels_arr"][100])
```

Out[8]:

```
62
```

In [9]:

```
max(df["pixels_arr"][0])
```

Out[9]:

```
210
```

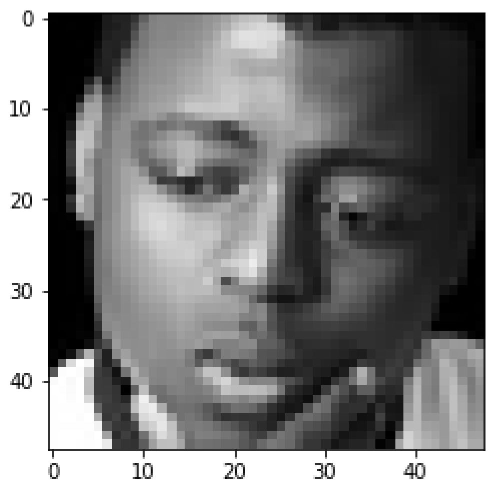
In [10]:

```
image,label = df["pixels_arr"][0],df["emotion"][0]
print('Shape:', image.shape, '\nLabel:', label)
```

```
Shape: (2304,)
Label: 0
```

In [11]:

```
plt.imshow(df["pixels_arr"][106].reshape(48,48),cmap='gray');
```



In [12]:

```
len(df['pixels_arr'][0])
```

Out[12]:

```
2304
```

In [13]:

```
type(df['pixels_arr'][0])
```

Out[13]:

```
numpy.ndarray
```

In [14]:

```
train_data_fer= torch.Tensor(df['pixels_arr']) ##important step
```

In [15]:

```
train_data_fer[0]
```

Out[15]:

```
tensor([ 70.,  80.,  82.,  ..., 106., 109.,  82.])
```

In [16]:

```
train_data_fer.shape
```

Out[16]:

```
torch.Size([35887, 2304])
```

In [17]:

```
train_data_fer= train_data_fer.reshape(35887,1,48,48) #reshaping to 1 color channel and 48x
```

In [18]:

```
train_data_fer.shape
```

Out[18]:

```
torch.Size([35887, 1, 48, 48])
```

In [19]:

```
train_data_fer[0].shape # 1st image as a tensor
```

Out[19]:

```
torch.Size([1, 48, 48])
```

In [21]:

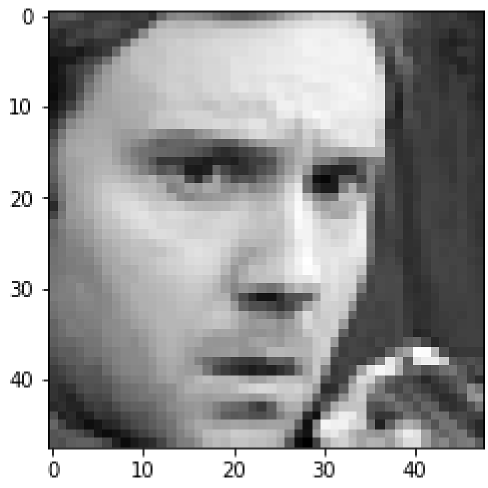
```
train_data_fer[0:5].shape #first 5 images
```

Out[21]:

```
torch.Size([5, 1, 48, 48])
```

In [23]:

```
plt.imshow(train_data_fer[0].reshape(48,48),cmap='gray');
```



In [24]:

```
class_names= ['Angry', 'Disgust','Fear', 'Happy', 'Sad', 'Surprise', 'Neutral']
```

In [25]:

```
labels= list(df['emotion'][0:5])
labels
```

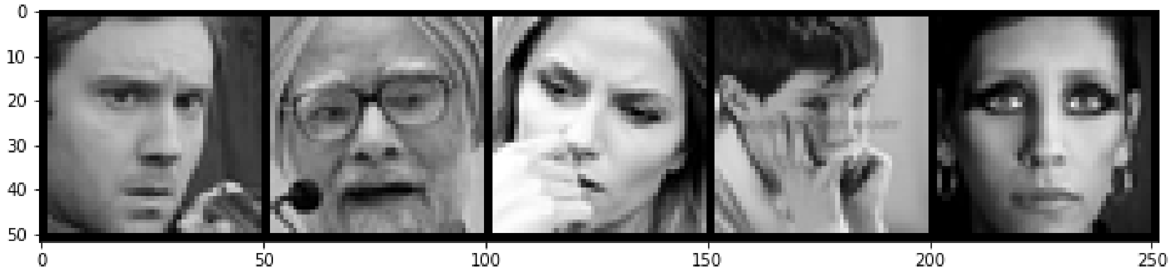
Out[25]:

```
[0, 0, 2, 4, 6]
```

In [265]:

```
np.set_printoptions(formatter=dict(int=lambda x: f'{x:5}')) # to widen the printed array
from torchvision.utils import make_grid
labels =list(df['emotion'][0:5])
print('Label:', np.array(labels))
print('Class: ', *np.array([class_names[i] for i in labels]))
images= train_data_fer[0:5]
im = make_grid(images, nrow=5)
plt.figure(figsize=(12,4))

plt.imshow(np.transpose(im, (1, 2, 0)));
```

```
Label: [    0     0     2     4     6]
Class:  Angry Angry Fear Sad Neutral
```



In [29]:

```
Y=torch.Tensor(df['emotion'])# target variable values
```

In [30]:

```
Y
```

Out[30]:

```
tensor([0., 0., 2.,  ..., 0., 3., 2.])
```

In [31]:

```
Y.shape
```

Out[31]:

```
torch.Size([35887])
```

In [32]:

```
Y = Y.type(torch.LongTensor)
```

In [33]:

```
Y.type()
```

Out[33]:

```
'torch.LongTensor'
```

In [34]:

```
Y
```

Out[34]:

```
tensor([0, 0, 2,  ..., 0, 3, 2])
```

In [35]:

```
len(Y)
```

Out[35]:

```
35887
```

Data Paritioning

In [36]:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test=train_test_split(train_data_fer, Y,test_size=0.20, random_
```

In [37]:

```
x_train[0].shape
```

Out[37]:

```
torch.Size([1, 48, 48])
```

In [38]:

```
x_train.shape
```

Out[38]:

```
torch.Size([28709, 1, 48, 48])
```

In [39]:

```
y_train.shape
```

Out[39]:

```
torch.Size([28709])
```

In [40]:

```
y_train
```

Out[40]:

```
tensor([2, 6, 4,  ..., 0, 2, 4])
```

In [41]:

```
train_dataset = TensorDataset(x_train, y_train)  #imp step
```

In [42]:

```
test_dataset = TensorDataset(x_test, y_test)  #imp step
```

In [43]:

```
train_dataset[0]
```

Out[43]:

```
(tensor([[[147., 147., 148.,  ..., 132., 133., 135.],
          [146., 146., 149.,  ..., 133., 134., 136.],
          [149., 151., 155.,  ..., 135., 136., 137.],
          ...,
          [ 49.,  42.,  36.,  ..., 157., 159., 158.],
          [ 42.,  32.,  38.,  ..., 155., 157., 158.],
          [ 37.,  35.,  35.,  ..., 155., 156., 156.]]]), tensor(2))
```

In [44]:

```
train_dataset[0][0].shape
```

Out[44]:

```
torch.Size([1, 48, 48])
```

In [45]:

```
test_dataset[0]
```

Out[45]:

```
(tensor([[[239., 239., 239.,  ..., 255., 254., 255.],
          [239., 239., 239.,  ..., 255., 254., 255.],
          [239., 239., 239.,  ..., 255., 255., 255.],
          ...,
          [ 46.,  80., 117.,  ..., 138., 207., 255.],
          [ 95., 112., 141.,  ..., 133., 217., 255.],
          [107., 108., 148.,  ..., 179., 221., 255.]]]), tensor(0))
```

In [46]:

```
test_dataset[0][0].shape
```

Out[46]:

```
torch.Size([1, 48, 48])
```

In [47]:

```
type(train_dataset)
```

Out[47]:

```
torch.utils.data.dataset.TensorDataset
```

In [48]:

```
train_dataset
```

Out[48]:

```
<torch.utils.data.dataset.TensorDataset at 0x1f9f3783cf8>
```

In [49]:

```
from torch.utils.data import TensorDataset, DataLoader
"""
Initializing train/test dataloader object which splits the training dataset into small batc
and this can be used later in the CNN model
"""
torch.manual_seed(101)
bat_sz=50
train_loader = DataLoader(train_dataset,batch_size=bat_sz,shuffle=True)
test_loader = DataLoader(test_dataset,batch_size=bat_sz,shuffle=False)
```

In [50]:

```
len(train_dataset)
```

Out[50]:

```
28709
```

In [51]:

```
len(test_dataset)
```

Out[51]:

```
7178
```

Then we have 7 emotions that we are predicting namely (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral), so we have 7 labels. We will be processing our inputs with a batch size of 50

Creating a Class from the nn.Module of Pytorch

In [52]:

```
class CONVNN(nn.Module):

    def __init__(self):

        super().__init__()
        self.conv1  = nn.Conv2d(1, 300 , kernel_size =5, stride=1,padding=2)#1 color channe
        self.norm1  = nn.BatchNorm2d(300)
        #self.drop1  = nn.Dropout2d(p=0.2)
        self.conv2  = nn.Conv2d(300, 150, 5 , 1,padding=2)
        self.norm2  = nn.BatchNorm2d(150)
        self.conv3  = nn.Conv2d(150, 75, 5 , 1,padding=2)
        self.norm3  = nn.BatchNorm2d(75)
        #self.drop2  = nn.Dropout2d(p=0.2)
        self.layer1 = nn.Linear(6*6*75,180) # we need to calculate the resulting number of
        self.drop3  = nn.Dropout2d(p=0.1)
        self.layer2 = nn.Linear(180,84)
        self.layer3 = nn.Linear(84,7) #only 7 classes of Dogs and Cats

    def forward(self,x):
                                 #self.drop1
        x= F.max_pool2d(             ( F.relu(    self.norm1(self.conv1(x)) )  )      ,2,2)

        x= F.max_pool2d(             ( F.relu(    self.norm2(self.conv2(x)) )  )      ,2,2)
        x= F.max_pool2d(             ( F.relu(    self.norm3(self.conv3(x)) )  )      ,2,2)

        x= self.drop3(F.relu(      self.layer1(  x.view(-1,6*6*75)    ) ) )      #flattening
        x= F.relu(self.layer2(x))
        x= F.log_softmax(  self.layer3(x), dim=1)          #multi class classificaiton

        return x
```

In [53]:

```
torch.manual_seed(101)
model = CONVNN()#to instansiate the model as cuda use "model = CONVNN().to(device) " or "mo
model
```

Out[53]:

```
CONVNN(
  (conv1): Conv2d(1, 300, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (norm1): BatchNorm2d(300, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
  (conv2): Conv2d(300, 150, kernel_size=(5, 5), stride=(1, 1), padding=(2,
2))
  (norm2): BatchNorm2d(150, eps=1e-05, momentum=0.1, affine=True, track_runn
ing_stats=True)
  (conv3): Conv2d(150, 75, kernel_size=(5, 5), stride=(1, 1), padding=(2,
2))
  (norm3): BatchNorm2d(75, eps=1e-05, momentum=0.1, affine=True, track_runni
ng_stats=True)
  (layer1): Linear(in_features=2700, out_features=180, bias=True)
  (drop3): Dropout2d(p=0.1, inplace=False)
  (layer2): Linear(in_features=180, out_features=84, bias=True)
  (layer3): Linear(in_features=84, out_features=7, bias=True)
)
```

In [54]:

```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(),lr=0.001)
```

Data Modelling & Training

In [55]:

```
epochs = 8
train_loss= []
test_loss= []
train_acc=[]
test_acc = []

print(f'\nConvolutional Neural Network Model Metrics:\n')
print(f'\t This CNN model configuration has {epochs} epochs with each batch size of {bat_sz
for i in range(epochs):

    train_crt_pred = 0
    test_crt_pred = 0
    conf_mat= torch.FloatTensor([])

    for b,(x_train,y_train) in enumerate (train_loader):
        b += 1
        y_pred = model.forward(x_train)
        loss= criterion(y_pred,y_train.long() )

        buffer = torch.max(y_pred.data, 1) [1]
        batch_acc = (buffer == y_train).sum()
        train_crt_pred +=  batch_acc

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if b%  int((len(train_dataset)/bat_sz)/3 )  == 0:
            print(f'Epoch{i+1:2} Batch {b:4} loss: {loss.item():5.2f} Train Accuracy: {trai

    train_loss.append(loss)  #loss after 1 epoch
    train_acc.append(train_crt_pred) # crt predictions after 1 epoch


    with torch.no_grad():  #testing after 1 complete epoch
        for b,(x_test,y_test) in enumerate (test_loader):
            b += 1
            y_eval = model(x_test)
            loss= criterion(y_eval,y_test.long())
            buffer1 = torch.max(y_eval.data, 1) [1]
            conf_mat =  torch.cat((conf_mat.float(),buffer1.float()),0)
            batch_acc = (buffer1 == y_test).sum()
            test_crt_pred +=  batch_acc

    test_loss.append(loss) #test loss after the last completed epoch
    test_acc.append(test_crt_pred) # crt predictions using the last completed epoch

    print(f'After {i+1} Epoch(s) the Train Accuracy is {(train_crt_pred.item()/len(train_da
```

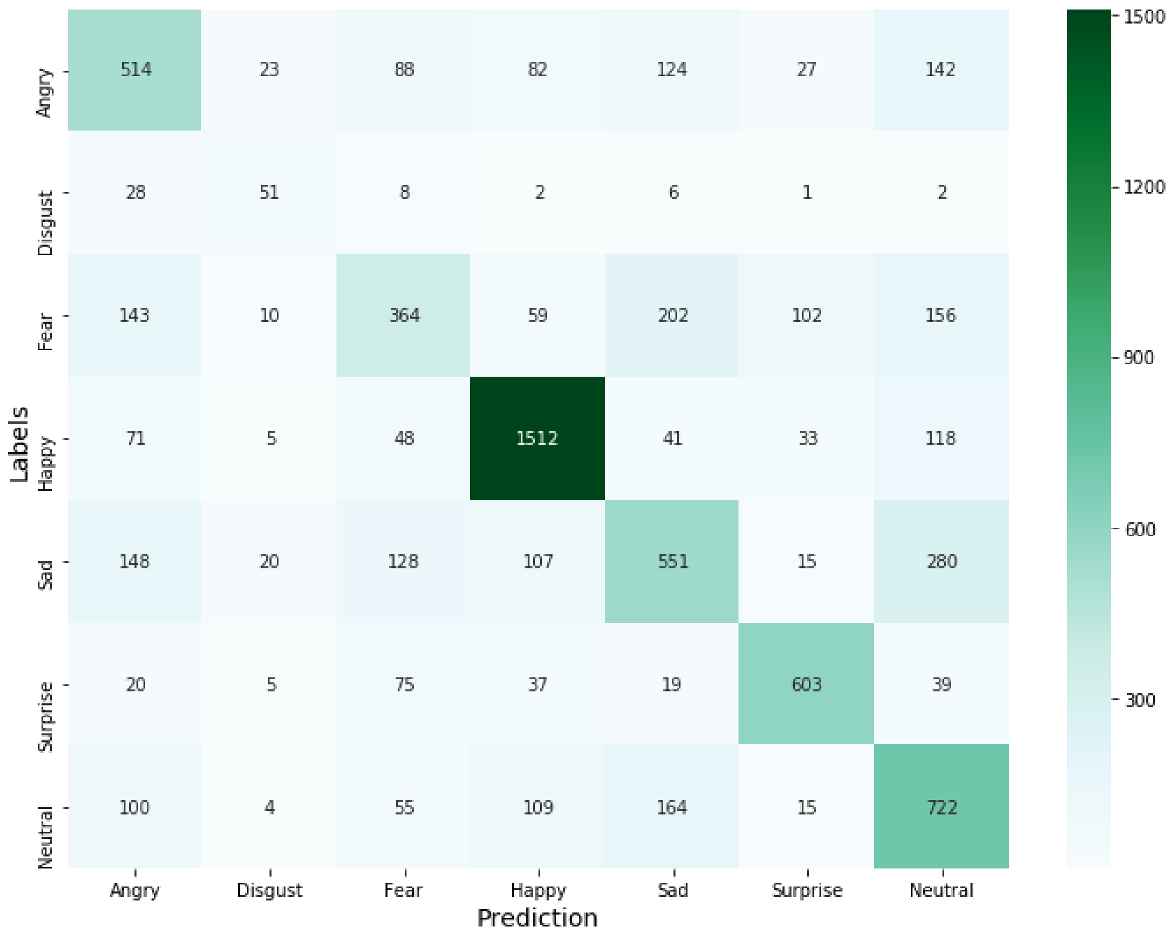
```
Convolutional Neural Network Model Metrics:

         This CNN model configuration has 8 epochs with each batch size of 5
0 images:
```

```
Epoch 1 Batch  191 loss:  1.54 Train Accuracy: 28.911%
Epoch 1 Batch  382 loss:  1.30 Train Accuracy: 34.607%
Epoch 1 Batch  573 loss:  1.37 Train Accuracy: 38.300%
After 1 Epoch(s) the Train Accuracy is 38.316% and Test Accuracy is 45.639%

Epoch 2 Batch  191 loss:  1.29 Train Accuracy: 48.555%
Epoch 2 Batch  382 loss:  1.28 Train Accuracy: 49.665%
Epoch 2 Batch  573 loss:  1.24 Train Accuracy: 50.265%
After 2 Epoch(s) the Train Accuracy is 50.270% and Test Accuracy is 53.469%

Epoch 3 Batch  191 loss:  1.28 Train Accuracy: 55.414%
Epoch 3 Batch  382 loss:  1.29 Train Accuracy: 54.817%
Epoch 3 Batch  573 loss:  1.13 Train Accuracy: 55.145%
After 3 Epoch(s) the Train Accuracy is 55.143% and Test Accuracy is 55.726%

Epoch 4 Batch  191 loss:  1.07 Train Accuracy: 59.267%
Epoch 4 Batch  382 loss:  1.17 Train Accuracy: 58.864%
Epoch 4 Batch  573 loss:  1.16 Train Accuracy: 58.970%
After 4 Epoch(s) the Train Accuracy is 58.978% and Test Accuracy is 56.534%

Epoch 5 Batch  191 loss:  1.11 Train Accuracy: 62.021%
Epoch 5 Batch  382 loss:  1.18 Train Accuracy: 62.120%
Epoch 5 Batch  573 loss:  1.00 Train Accuracy: 61.937%
After 5 Epoch(s) the Train Accuracy is 61.939% and Test Accuracy is 57.788%

Epoch 6 Batch  191 loss:  1.30 Train Accuracy: 65.236%
Epoch 6 Batch  382 loss:  0.66 Train Accuracy: 65.094%
Epoch 6 Batch  573 loss:  0.72 Train Accuracy: 64.852%
After 6 Epoch(s) the Train Accuracy is 64.844% and Test Accuracy is 58.888%

Epoch 7 Batch  191 loss:  0.54 Train Accuracy: 68.628%
Epoch 7 Batch  382 loss:  0.71 Train Accuracy: 68.764%
Epoch 7 Batch  573 loss:  0.96 Train Accuracy: 68.244%
After 7 Epoch(s) the Train Accuracy is 68.223% and Test Accuracy is 57.997%

Epoch 8 Batch  191 loss:  0.71 Train Accuracy: 71.843%
Epoch 8 Batch  382 loss:  0.74 Train Accuracy: 71.770%
Epoch 8 Batch  573 loss:  0.78 Train Accuracy: 71.836%
After 8 Epoch(s) the Train Accuracy is 71.831% and Test Accuracy is 60.142%
```

In [101]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import confusion_matrix,classification_report
import seaborn as sns
print('\nThe Confusion Matrix is plotted below:')
cfmt =pd.DataFrame(confusion_matrix(torch.Tensor([r for q,r in test_dataset]).reshape(-1,1)
plt.figure(figsize=(12,9))
sns.heatmap(cfmt,annot=True,cmap='BuGn',fmt="d")
plt.xlabel("Prediction",fontsize=14)
plt.ylabel("Labels",fontsize=14)
plt.show()
print('\nThe Classification Report is plotted below: \n\n',classification_report(torch.Tens
```

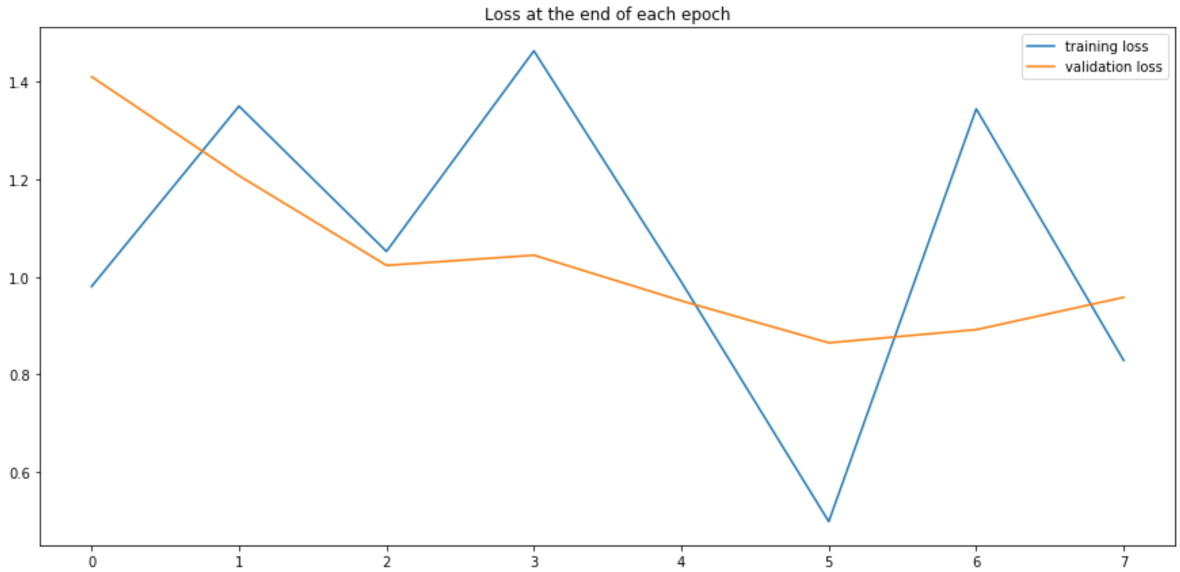
The Confusion Matrix is plotted below:



```
The Classification Report is plotted below:

              precision    recall  f1-score   support

         0.0       0.50      0.51      0.51      1000
         1.0       0.43      0.52      0.47        98
         2.0       0.48      0.35      0.40      1036
         3.0       0.79      0.83      0.81      1828
         4.0       0.50      0.44      0.47      1249
         5.0       0.76      0.76      0.76       798
         6.0       0.49      0.62      0.55      1169

    accuracy                           0.60      7178
   macro avg       0.56      0.58      0.57      7178
weighted avg       0.60      0.60      0.60      7178
```

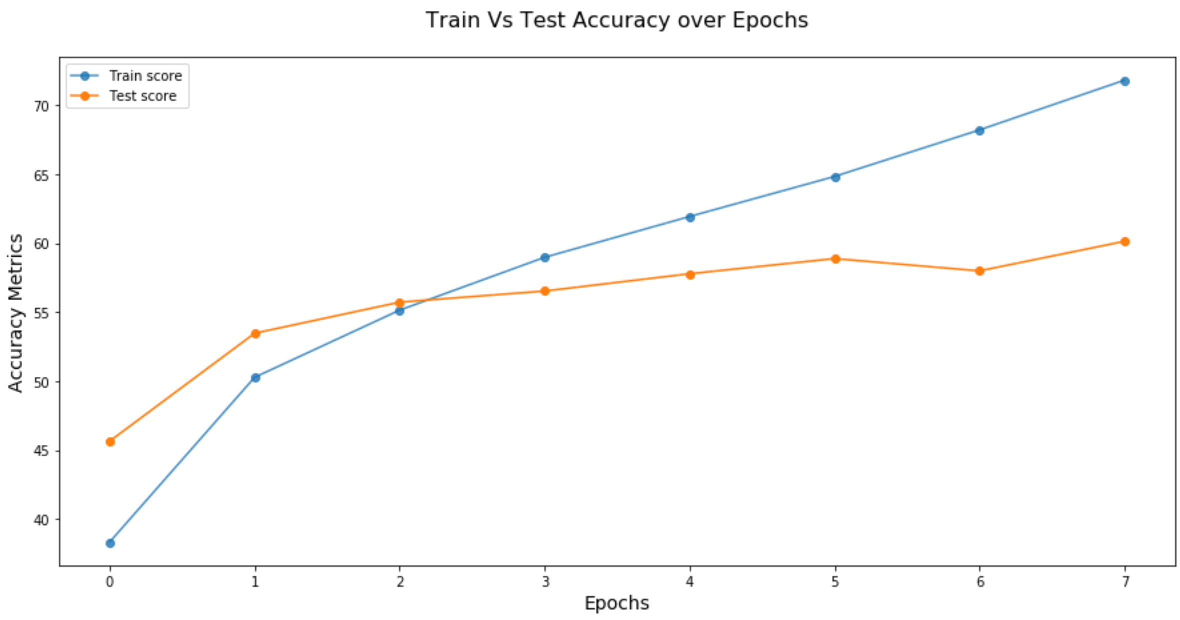
In [96]:

```
plt.figure(figsize=(15,7))
plt.plot(train_loss, label='training loss')
plt.plot(test_loss, label='validation loss')
plt.title('Loss at the end of each epoch')
plt.legend();
```



In [95]:

```
plt.figure(figsize=(15,7))
plt.plot([(t/(len(train_dataset)/100)) for t in train_acc], label='Train score',marker='o',
plt.plot([(t/(len(test_dataset)/100)) for t in test_acc], label='Test score',marker='o')
plt.title('\nTrain Vs Test Accuracy over Epochs\n',fontsize=16)
# plt.xticks(np.arange(0,20,1));
plt.xlabel('Epochs',fontsize=14)
plt.ylabel('Accuracy Metrics',fontsize=14)
plt.legend();
```



Save the Model

torch.save(model.state_dict(), 'FERCNNModel.pt')

Evaluate the model

In [170]:

```
# Evaluation DATA TRANSFORMATION
eval_transform =transforms.Compose( [
    transforms.Resize( (48,48)),
    transforms.Grayscale(num_output_channels=1),# Third transformation --Resize to 224 as t
    transforms.ToTensor(),                      # Fifth transformation -- to convert it into

] )
```

In [171]:

```
root = r'C:\Univ'
```

In [202]:

```
import os
from PIL import Image #to operate on pics or images
from IPython.display import display #Only for Jupyter Notebook
import warnings
warnings.filterwarnings('ignore')
eval_data  = datasets.ImageFolder(os.path.join(root,'expr'),transform = eval_transform)
```

In [203]:

```
eval_data
```

Out[203]:

```
Dataset ImageFolder
    Number of datapoints: 2
    Root location: C:\Univ\expr
    StandardTransform
Transform: Compose(
               Resize(size=(48, 48), interpolation=PIL.Image.BILINEAR)
               Grayscale(num_output_channels=1)
               ToTensor()
           )
```

In [204]:

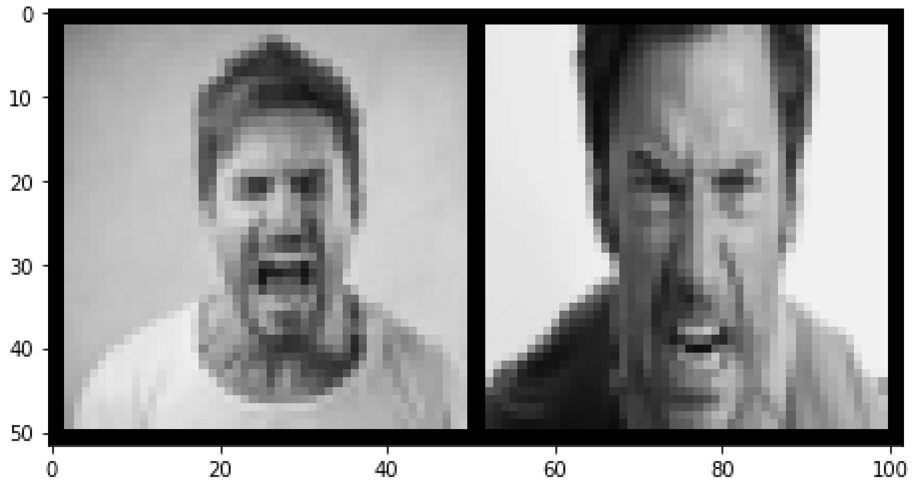
```
image, label = eval_data[0]
print('Shape:', image.shape, '\nLabel:', label)
```

```
Shape: torch.Size([1, 48, 48])
Label: 0
```

In [205]:

```
np.set_printoptions(formatter=dict(int=lambda x: f'{x:5}')) # to widen the printed array
from torchvision.utils import make_grid
# Grab the first  10 images from the first batch of training data
eval_load_all = DataLoader(eval_data, batch_size=2, shuffle=True)
for images,labels in eval_load_all:
    break

# Print the labels
print('Label:', labels.numpy())
print('Class: ', *np.array([class_names[i] for i in labels]))

# Print the images
im = make_grid(images, nrow=2)  # the default nrow is 8

plt.figure(figsize=(12,4))
plt.imshow(np.transpose(im.numpy(), (1, 2, 0)));
```

```
Label: [    0     0]
Class:  Angry Angry
```



In [206]:

```
eval_load_all = DataLoader(eval_data, batch_size=2, shuffle=False)
model.eval()
with torch.no_grad():
    correct = 0
    for X_test, y_test in eval_load_all:
        y_val = model(X_test)
        predicted = torch.max(y_val,1)[1]
        correct += (predicted == y_test).sum()

print(f'Test accuracy: {correct.item()}/{len(eval_data)} = {correct.item()*100/(len(eval_da
```

```
Test accuracy: 2/2 = 100.000%
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: