Sentiment Classification using Images of Faces

By Ann Joseph (251061872) Meher Pooja Pranavi Punyamanthula (251056788)

This is a private kaggle dataset consisting of about 28,000 rows and 2 columns. The two columns are "label" and "features". The "features" column consist of pixels that make is a 48x48 grayscale image and the "label" column is a number from 0 to 6 depending on how this grayscale image was classified where: 0 - Angry 1 - Disgust 2 - Fear 3 - Happy 4 - Sad 5 - Surprise 6 - Neutral

In this assignment, we built a Convolutional Neural Network (CNN) to classify images in our test data and compared it with the its real labels to obtain an accucary for our CNN.

The following are the steps we did in this assignment:

1. Importing Data
2. Image visualization
3. Data Preprocessing
4. Reshaping the data
5. Data Splitting
6. Building the CNN
7. Model Training
8. Prediction using the Test Set

1. Importing Data

In [20]:
```
import random
import tensorflow as tf
import keras as ks
import pandas as pd
from matplotlib.pyplot import imshow
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import *
from keras.optimizers import *
from PIL import Image
#from __future__ import print_function
import numpy as np
```

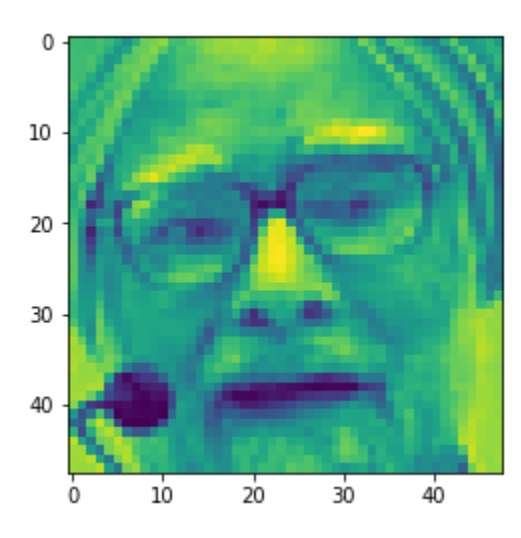
In [13]:
```
filname='sentiment_data.csv'
data = pd.read_csv("sentiment_data.csv")
label_map = ['Angry', 'Disgust', 'Fear', 'Happy', 'Sad', 'Surprise', 'Neutral']
```

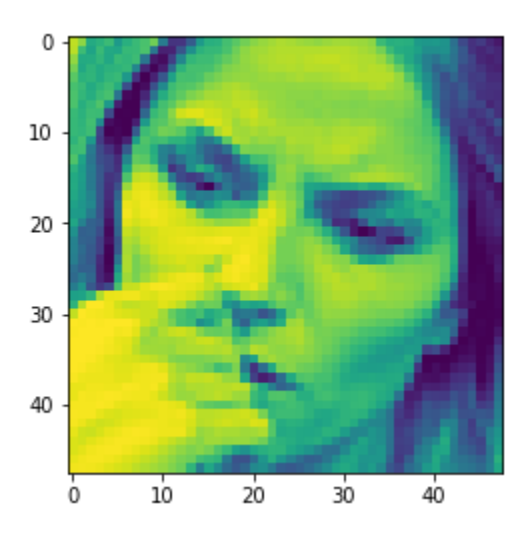
2. Image visualization

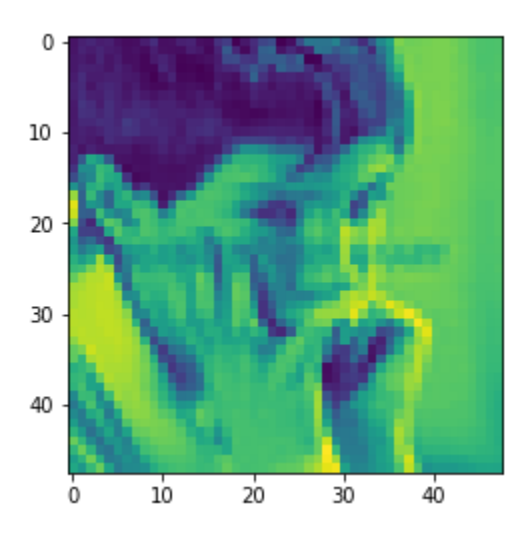
Below is a list of the labels of the first 10 images in our dataset and what the grayscale images actually look like.

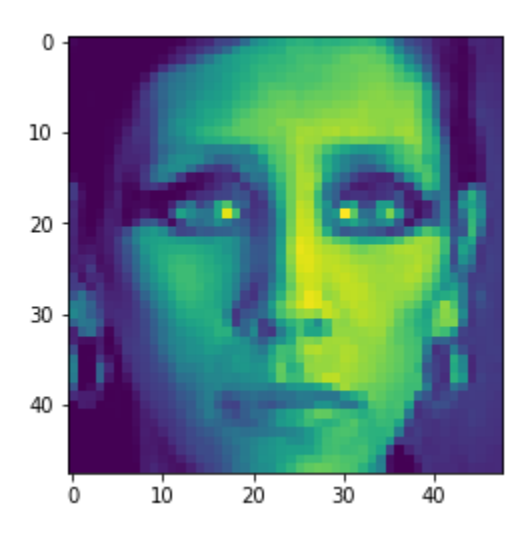
In [21]:
```
labels= {0:"angry", 1:"disgust", 2:"fear", 3:"happy", 4:"sad", 5:"surprise", 6:"neutral"}
images=[]
for image_pixels in data.iloc[1:11,1]:
    image_string = image_pixels.split(' ') #pixels are separated by commas.
    image_data = np.asarray(image_string, dtype=np.uint8).reshape(48,48)
    img = Image.fromarray(image_data)
    images.append(img)
#imshow(np.asarray(img))
#print(data.iloc[1,0])

for ima in images:
    plt.figure()
    plt.imshow(np.asarray(ima))
for label in data.iloc[1:11,0]:
    print(labels[label])
```

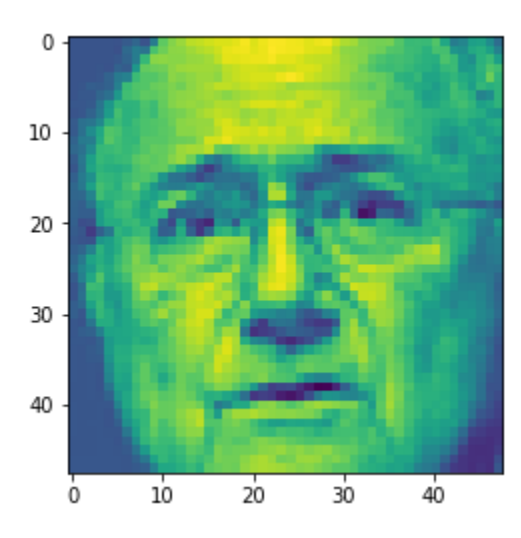
```
angry
fear
sad
neutral
fear
sad
happy
happy
fear
angry
```

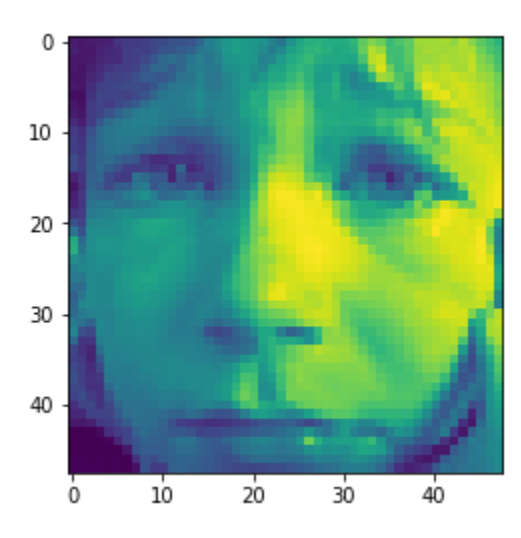


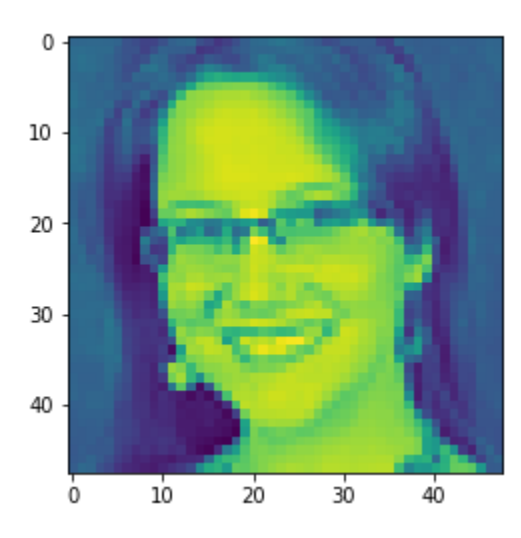


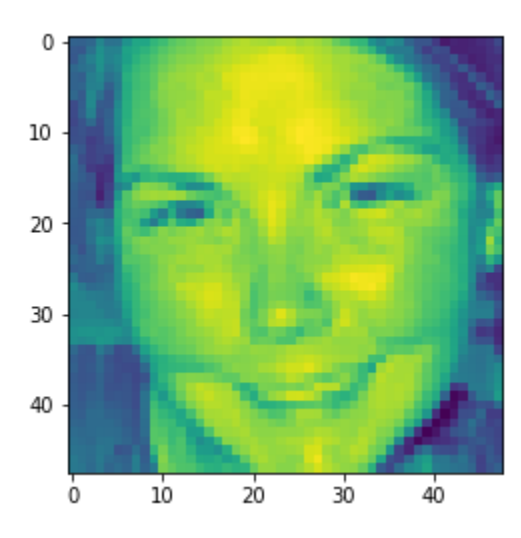


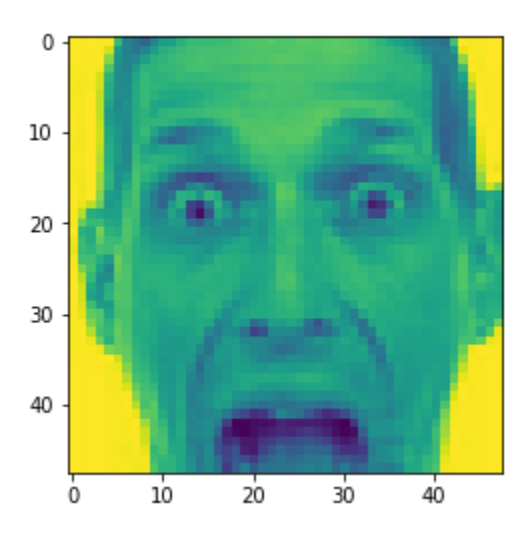


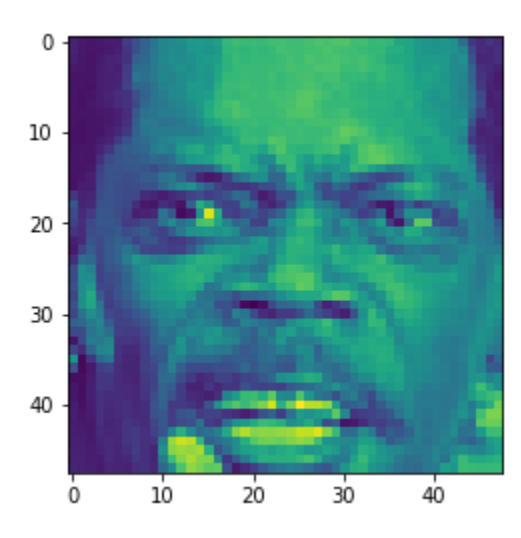












3. Data Preprocessing

Since the pixels in our data were given as a single string, we define a function below to split this string into a list of pixels for each image in our dataset.

In [22]:
```
def getData(filname):
    # images are 48x48
    # N = 35887
    Y = []
    X = []
    first = True
    for line in open(filname):
        if first:
            first = False
        else:
            row = line.split(',')

            Y.append(int(row[0]))

            X.append([int(p) for p in row[1].split()])

    X, Y = np.array(X) / 255.0, np.array(Y)
    return X, Y
```

In [23]:
```
X, Y = getData(filname)
num_class = len(set(Y))

# To see number of training data point available for each label
def balance_class(Y):
    num_class = set(Y)
    count_class = {}
    for i in range(len(num_class)):
        count_class[i] = sum([1 for y in Y if y == i])
    return count_class
balance = balance_class(Y)
```

4. Reshaping the data

Since we now have a list of pixels for each image of our dataset, we have to take those pixels and reshape it into a 48x48 matrix so that it represents a 48x48 grayscale image.

In [24]:
```
# keras with tensorflow backend
N, D = X.shape
X = X.reshape(N, 48, 48, 1)
```

5. Data Splitting

Since this dataset is large with around 28,000 rows, we will be splitting the data into 3 sets: a training set, a validation set and a test set in the ratio 80:10:10.

In [25]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.1, random_state=0)
y_train = (np.arange(num_class) == y_train[:, None]).astype(np.float32)
y_test = (np.arange(num_class) == y_test[:, None]).astype(np.float32)
```

6. Building the CNN

In [26]:
```
from keras.models import Sequential
from keras.layers import Dense , Activation , Dropout ,Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.metrics import categorical_accuracy
from keras.models import model_from_json
from keras.optimizers import *
from keras.layers.normalization import BatchNormalization
batch_size = 128
epochs = 124
```

To build our CNN, below we define a function that specifies 4 layers. As you can see, we have used the Rectified Linear Unit or ReLU as our activation function in this classification and the Max Pooling method to resize images before they are inputted to the next layer.

In [27]:
```
#Main CNN model with four Convolution layer & two fully connected layer
def baseline_model():
    # Initialising the CNN
    model = Sequential()
    # 1 - Convolution
    model.add(Conv2D(64,(3,3), border_mode='same', input_shape=(48, 48,1)))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))
    # 2nd Convolution layer
    model.add(Conv2D(128,(5,5), border_mode='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))
    # 3rd Convolution layer
    model.add(Conv2D(512,(3,3), border_mode='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))
    # 4th Convolution layer
    model.add(Conv2D(512,(3,3), border_mode='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))
    # Flattening
    model.add(Flatten())
    # Fully connected layer 1st layer
    model.add(Dense(256))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(0.25))
    # Fully connected layer 2nd layer
    model.add(Dense(512))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(0.25))
    model.add(Dense(num_class, activation='sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=[categorical_accuracy])
    return model
```

Below is a function to calculate weights in the CNN and a final CNN model with all weights initialised is returned.

In [28]:
```
def baseline_model_saved():
    #load json and create model
    json_file = open('model_4layer_2_2_pool.json', 'r')
    loaded_model_json = json_file.read()
    json_file.close()
    model = model_from_json(loaded_model_json)
    #load weights from h5 file
    model.load_weights("model_4layer_2_2_pool.h5")
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=[categorical_accuracy])
    return model
is_model_saved = True
```

7. Model Training

The final CNN model is then train using the validation set as shown below.

In [29]:
```
if(is_model_saved==False ):
    # Train model
    model = baseline_model()
    # Note : 3259 samples is used as validation data &   28,709  as training samples
    model.fit(X_train, y_train,
                    batch_size=batch_size,epochs=epochs,verbose=2,validation_split=0.1111)
    model_json = model.to_json()
    with open("model_4layer_2_2_pool.json", "w") as json_file:
        json_file.write(model_json)
    # serialize weights to HDF5
    model.save_weights("model_4layer_2_2_pool.h5")
    print("Saved model to disk")

else:
    # Load the trained model
    print("Load model from disk")
    model = baseline_model_saved()
```

```
Load model from disk
WARNING:tensorflow:From /home/pranavi/.local/lib/python3.6/site-packages/tensorflow/python/framew
ork/op_def_library.py:263: colocate_with (from tensorflow.python.framework.ops) is deprecated and
will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /home/pranavi/.local/lib/python3.6/site-packages/keras/backend/tensorflow
_backend.py:3445: calling dropout (from tensorflow.python.ops.nn_ops) with keep_prob is deprecate
d and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
```

8. Prediction using the Test Set

Finally, the CNN is used to classify images in the test set and the accuracy is calculated.

In [30]:
```
# Model will predict the probability values for 7 labels for a test image
score = model.predict(X_test)
print (model.summary())
new_X = [ np.argmax(item) for item in score ]
y_test2 = [ np.argmax(item) for item in y_test]
# Calculating categorical accuracy taking label having highest probability
accuracy = [ (x==y) for x,y in zip(new_X,y_test2) ]
print(" Accuracy on Test set : " , np.mean(accuracy)*100)
```

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 48, 48, 64)	640
batch_normalization_1 (Batch	(None, 48, 48, 64)	256
activation_1 (Activation)	(None, 48, 48, 64)	0
max_pooling2d_1 (MaxPooling2	(None, 24, 24, 64)	0
dropout_1 (Dropout)	(None, 24, 24, 64)	0
conv2d_2 (Conv2D)	(None, 24, 24, 128)	204928
batch_normalization_2 (Batch	(None, 24, 24, 128)	512
activation_2 (Activation)	(None, 24, 24, 128)	0
max_pooling2d_2 (MaxPooling2	(None, 12, 12, 128)	0
dropout_2 (Dropout)	(None, 12, 12, 128)	0
conv2d_3 (Conv2D)	(None, 12, 12, 512)	590336
batch_normalization_3 (Batch	(None, 12, 12, 512)	2048
activation_3 (Activation)	(None, 12, 12, 512)	0
max_pooling2d_3 (MaxPooling2	(None, 6, 6, 512)	0
dropout_3 (Dropout)	(None, 6, 6, 512)	0
conv2d_4 (Conv2D)	(None, 6, 6, 512)	2359808
batch_normalization_4 (Batch	(None, 6, 6, 512)	2048
activation_4 (Activation)	(None, 6, 6, 512)	0
max_pooling2d_4 (MaxPooling2	(None, 3, 3, 512)	0
dropout_4 (Dropout)	(None, 3, 3, 512)	0
flatten_1 (Flatten)	(None, 4608)	0
dense_1 (Dense)	(None, 256)	1179904
batch_normalization_5 (Batch	(None, 256)	1024
activation_5 (Activation)	(None, 256)	0
dropout_5 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 512)	131584
batch_normalization_6 (Batch	(None, 512)	2048
activation_6 (Activation)	(None, 512)	0
dropout_6 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 7)	3591