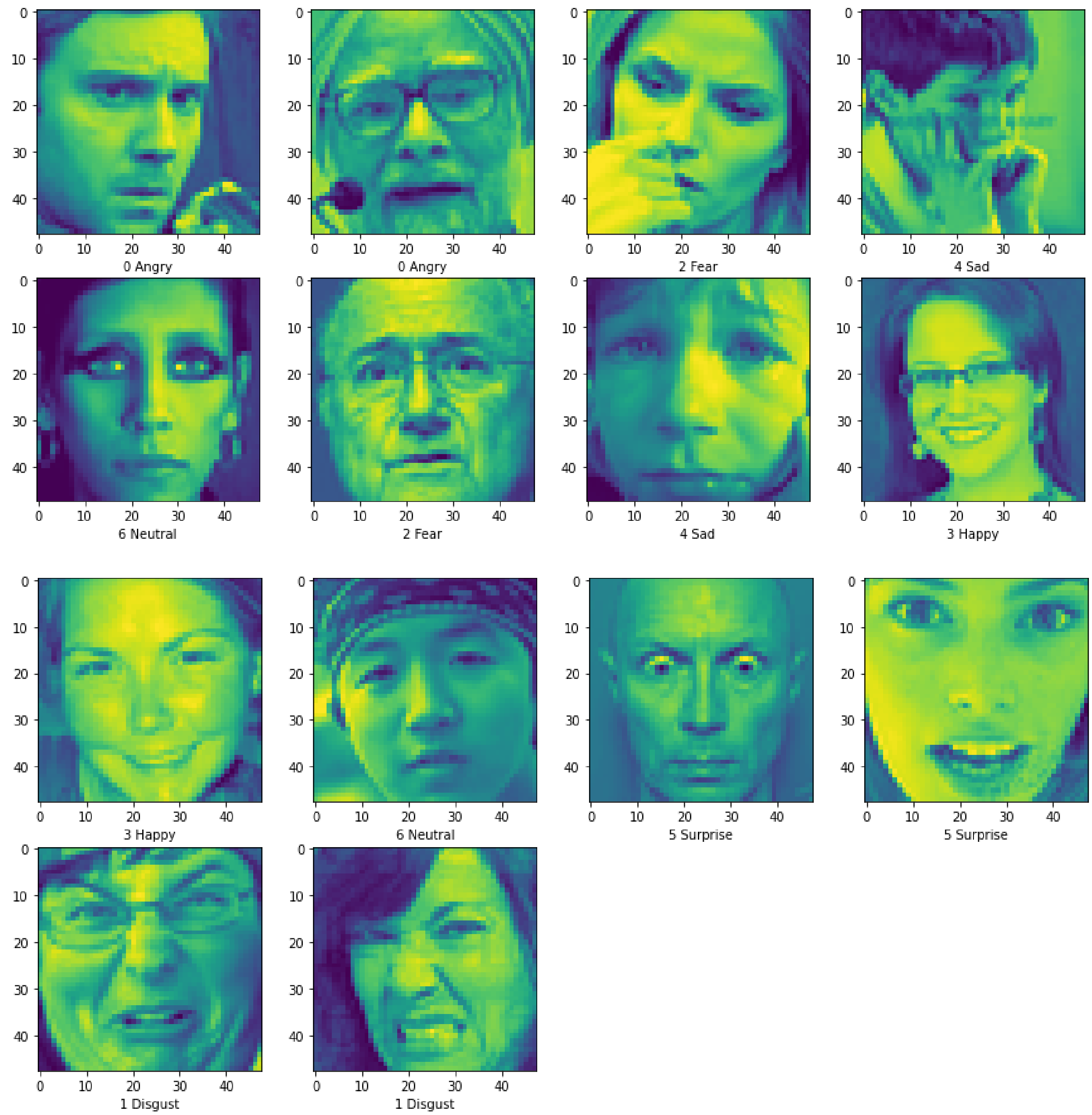
<u>CSCE 633 MACHINE LEARNING Homework-3 Report</u>

Name: Sai Namith Garapati

UIN: 832001176

Machine learning for facial emotion recognition

(a) Visualization:



Explanation of code and observations:

- As seen from the image above, here 1-2 images are randomly selected per emotion and are visualized.
- A dictionary is created for all emotions with respect to labels. List of all the label values we will be using to visualize the images is created.
- A loop is run until all the elements of list are visualized. Initially the pixel values for respective label value of each image is converted into a NumPy array.
- This array is reshaped into 48 x 48 and then the final grayscale image containing 48 x 48 is visualized.

```
#importing the libraries ad mounting the google drive
import warnings
warnings.filterwarnings("ignore")
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from tensorflow.keras.utils import to_categorical
from keras.layers import Conv2D, Flatten, MaxPooling2D
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.applications.vgg16 import VGG16
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
from google.colab import drive
drive.mount('/content/gdrive/')


#creating dataframes for training testing and validation
traindata = pd.read_csv('/content/gdrive/MyDrive/Data/Q1_Train_Data_edited
.csv')
valdata   = pd.read_csv('/content/gdrive/MyDrive/Data/Q1_Validation_Data_e
dited.csv')
testdata  = pd.read_csv('/content/gdrive/MyDrive/Data/Q1_Test_Data_edited.
csv')

#1
traindata1 = traindata.drop(columns=['emotion'])
em_list = [0,0,1,1,2,2,3,3,4,4,5,5,6,6]
em_dict = {0:"Angry", 1: "Disgust", 2: "Fear", 3:"Happy", 4:"Sad", 5:"Surp
rise", 6:"Neutral"}   #creating a dictionary for all emotions with respect
ive key values
fig, axs = plt.subplots(3, 3, figsize=(15,15))
j=0
```

```
for i,val in enumerate(traindata[traindata.columns[0]]):

  if(len(em_list)) and (val in em_list):
    em_list.remove(val)
    plt.subplot(4,4,j+1)
    plt.xlabel(str(val) + " " + em_dict[val])
    img_array = np.asfarray(traindata1.iloc[i])
        #converting the data to a numpy array
    img = np.reshape(img_array, (48,48))
        #resizing to get an image of output 48 x 48
    plt.imshow(img)
    j = j + 1
  elif (len(em_list)):
    continue
  else:
    break
```

(b)Data exploration :

Emotion Label	Emotion	Number of samples
0	Angry	3995
1	Disgust	436
2	Fear	4097
3	Happy	7215
4	Sad	4830
5	Surprise	3171
6	Neutral	4965

Explanation of code and observations :

- The count of number of samples of each emotion is obtained with the help of value_counts ( ).
- Maximum number of images account to happy emotion whereas we have the least amount of images with disgust.

```
#dataexploration
count = traindata[traindata.columns[0]].value_counts()   #getting the coun
t of images for each emotion
l = []
for i in count.keys():
  l.append((i,em_dict[i], count [i]))
l
```

(c) . Image Classification with FNN'S:

In this part, we will use a feedforward neural network (FNN) (also called \multilayer perceptron") to perform the emotion classification task. The input of the FNN comprises of all the pixels of the image.

(c.i) Implementation of FNN and finding out best hyperparameter

Implementation:

- Initially all the data of training, testing and validation is converted to a NumPy array. Then the NumPy array is reshaped to 48 x 48 to get into the shape of our required image. Now we have the data of 48 x 48 grayscale images of training, validation and testing ready
- Preprocessing of images is done by normalizing all the images. This helps us in adjusting differences in the values of attributes.
- First feed-forward model is designed with 3 hidden layers with activation function as 'relu' in all three hidden layers. First, we have a dense fully connected layer with 1000 nodes in first hidden layer and then 100 in second and 100 in third.
- Last, we have a dense fully connected layer with 7 nodes with a SoftMax function which would classify our image into seven emotions.
- Model is compiled with optimizer as 'SGD' and loss function as 'cross entropy' with metric as accuracy.
- All the images are flattened to be fed to the feed forward neural network and the model is trained for **750** epochs in which the model will learn optimal way of classifying by forward propagating and backward propagating several times.
- Cross-entropy loss and accuracy over the number of iterations during training are also plotted.
- Similarly, two other models with different hyperparameter combinations are implemented and best model is evaluated on the validation set.
- Finally, we evaluate the performance of the model we built on the test data.

Observations:

Hyper parameter combinations used:

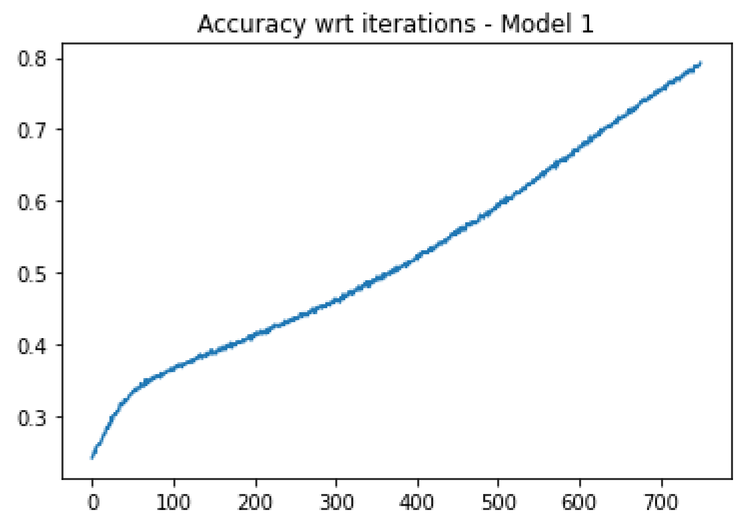
Model 1:

- Three hidden layers
- 1000 nodes in first hidden layer, 100 nodes in second hidden layer, 100 nodes in third hidden layer. 7 nodes in last layer.
- Activation function as 'ReLu' in all three hidden layers, SoftMax activation function in the last layer
- Dropout of 0.5 in first hidden layer, 0.5 in second hidden layer and 0.25 in the third layer.
- Optimizer: 'sgd', loss: 'categorical cross entropy', metric : 'accuracy'
- 750 epochs

Results of model 1:

1. Emotion classification accuracy on training set: **0.938730001449585**
2. Emotion classification accuracy on validation set: **0.4536082446575165**
(We can see that accuracy on training set is pretty high when compared to validation set, this is because I have run the FNN model for 750 epochs for better accuracy. Hence for 750 epochs it is overfitting but still showing better accuracy on validation set compared to other smaller values of epochs.)
3. Running time for training the FNN: **6min 40s (due to 750 epochs)**
4. Number of parameters for each FNN: **2,415,907**

Plot of crossentropy loss and accuracy over the number of iterations during training:



Model 2:

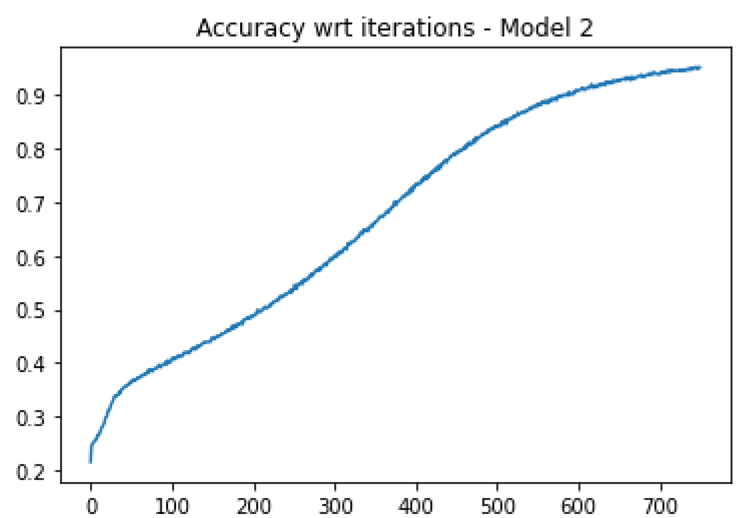
- Five hidden layers

- 1000 nodes in first hidden layer, 500 nodes in second hidden layer, 250 nodes in third hidden layer, 100 nodes in the fourth hidden layer, 100 nodes in the fifth hidden layer, 7 nodes in last layer.
- Activation function as ‘ReLu’ in all five hidden layers, SoftMax activation function in the last layer
- Dropout of 0.5 in first hidden layer and 0.25 in the third layer.
- Optimizer: ‘sgd’, loss : ‘categorical cross entropy’, metric : ‘accuracy’
- 750 epochs.

Results of model 2:

5. Emotion classification accuracy on training set: **0.9948796629905701**
6. Emotion classification accuracy on validation set: **0.4625243842601776**
7. Running time for training the FNN: **7min 9s (due to 750 epochs)**
8. Number of parameters for each FNN: **2,966,657**

Plot of cross entropy loss and accuracy over the number of iterations during training:



Model 3:

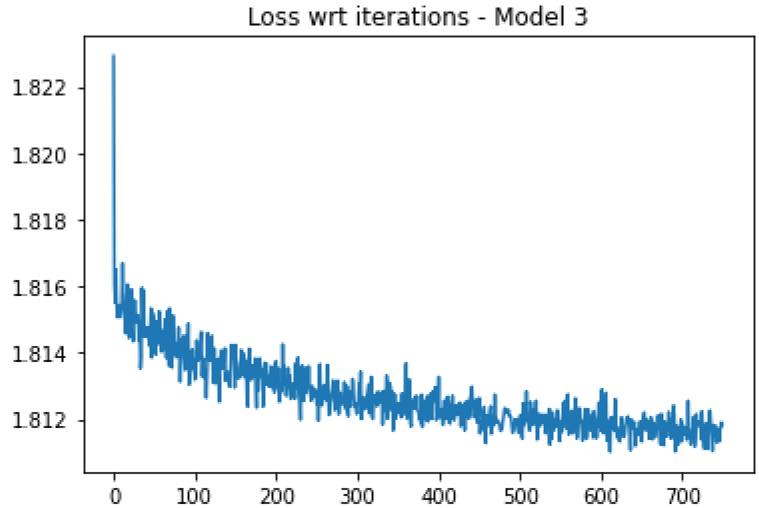
- Six hidden layers
- 1000 nodes in first hidden layer, 1000 nodes in second hidden layer, 1000 nodes in third hidden layer, 1000 nodes in the fourth hidden layer, 1000 nodes in the fifth hidden layer, 1000 nodes in the sixth hidden layer,7 nodes in last layer.
- Activation function as ‘Sigmoid’ in all five hidden layers, SoftMax activation function in the last layer
- Dropout of 0.5 in first hidden layer, 0.25 in the third layer, 0.25 in the fifth layer.
- Optimizer: ‘sgd’, loss : ‘categorical cross entropy’, metric : ‘accuracy’

Results of model 3:

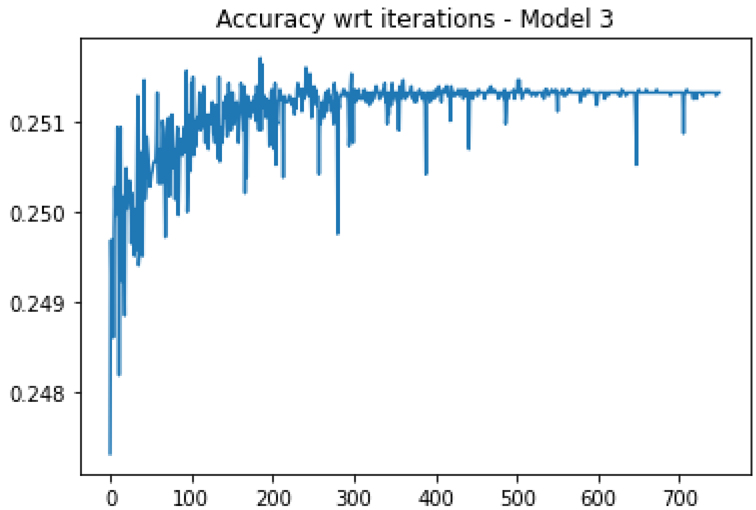
9. Emotion classification accuracy on training set: **0.2513149082660675**
10. Emotion classification accuracy on validation set: **0.24937307834625244**
11. Running time for training the FNN: **2min 28s (when done on 50 epochs)**
12. Number of parameters for each FNN: **7,317,007**

It can be observed that the model performs well when the training set is trained with more epochs. We get much better accuracy when we trained the FNN with 750 epochs instead of training the FNN with 50 epochs where we get less accuracy. The difference between accuracy of training set and validation set is considerable which means the data is overfitting, but still it performs better when performed with more epochs. Model also performs well with activation function ReLu and when we used model1 architecture, accuracy is better

Plot of cross entropy loss and over the number of iterations during training



:



(c.ii)
Upon running the three models, the best model found based on the validation set is **MODEL 2**
(Model 2 has Five hidden layers with an activation function 'ReLu' and 750 epochs.)

The emotion classification on the testing set for that model is: **0.4519364833831787**

Code for (c.i) and (c.ii) :

```
training_images = np.asfarray(traindata1)
training_images = np.reshape(training_images, (len(training_images),48,48)
)
training_labels = np.asfarray(traindata[traindata.columns[0]])
print(training_images.shape)
valdata1 = valdata.drop(columns=['emotion'])
validation_images = np.asfarray(valdata1)
validation_images = np.reshape(validation_images, (len(validation_images),
48,48))
validation_labels = np.asfarray(valdata[valdata.columns[0]])
print(validation_images.shape)
testdata1 = testdata.drop(columns=['emotion'])
testing_images = np.asfarray(testdata1)
testing_images = np.reshape(testing_images, (len(testing_images),48,48))
testing_labels = np.asfarray(testdata[testdata.columns[0]])
print(testing_images.shape)
# Preprocessing: Normalize the images.
training_images = (training_images / 255) - 0.5
validation_images = (validation_images / 255) - 0.5
testing_images = (testing_images / 255) - 0.5
import warnings
warnings.filterwarnings("ignore")  # Ignore some warning logs
```

```
#  Define a FeedForward Model with 3 hidden layers with dimensions 392 and
 196 Neurons
model1 = Sequential([
  Dense(1000, activation='relu', input_shape=(48*48,), name="firsthiddenla
yer"), Dropout(0.5),
  Dense(100, activation='relu', name="secondhiddenlayer"), Dropout(0.5),
  Dense(100, activation='relu', name="thirdhiddenlayer"), Dropout(0.25),
  Dense(7, activation='softmax'),
])

#  Validate your Model Architecture
print(model1.summary())

# Compile model
model1.compile(optimizer='sgd', loss='categorical_crossentropy',metrics=['
accuracy'],)

# Flatten the images into vectors (1D) for feed forward network
flatten_training_images = training_images.reshape((-1, 48*48))
flatten_testing_images = testing_images.reshape((-1, 48*48))
flatten_validation_images = validation_images.reshape((-1, 48*48))

# Train model

%time model1.fit(flatten_training_images, to_categorical(training_labels),
 epochs=750, batch_size= 500,)
plt.plot(model1.history.history['loss'])
plt.title('Loss wrt iterations - Model 1')
plt.figure()
plt.plot(model1.history.history['accuracy'])
plt.title('Accuracy wrt iterations - Model 1')

performance_training = model1.evaluate(flatten_training_images,to_categori
cal(training_labels))
print("Accuracy on training samples: {0}".format(performance_training[1]))
performance_validation = model1.evaluate(flatten_validation_images,to_cate
gorical(validation_labels))
print("Accuracy on Val samples: {0}".format(performance_validation[1]))

#model2
import warnings
warnings.filterwarnings("ignore")  # Ignore some warning logs
```

```
#  Define a Feed-
Forward Model with 3 hidden layers with dimensions 392 and 196 Neurons
model2 = Sequential([
  Dense(1000, activation='relu', input_shape=(48*48,), name="firsthiddenla
yer"), Dropout(0.5),
  Dense(500, activation='relu', name="secondhiddenlayer"),
  Dense(250, activation='relu', name="thirdhiddenlayer"), Dropout(0.25),
  Dense(100, activation='relu', name="fourthhiddenlayer"),
  Dense(100, activation='relu', name="fifthhiddenlayer"),
  Dense(7, activation='softmax'),
])

#  Validate your Model Architecture
print(model2.summary())
# Compile model
model2.compile(optimizer='sgd', loss='categorical_crossentropy',metrics=['
accuracy'],)

# Flatten the images into vectors (1D) for feed forward network
flatten_training_images = training_images.reshape((-1, 48*48))
flatten_testing_images = testing_images.reshape((-1, 48*48))
flatten_validation_images = validation_images.reshape((-1, 48*48))

# Train model

%time model2.fit(flatten_training_images, to_categorical(training_labels),
 epochs=750, batch_size= 500,)
plt.plot(model2.history.history['loss'])
plt.title('Loss wrt iterations - Model 2')
plt.figure()
plt.plot(model2.history.history['accuracy'])
plt.title('Accuracy wrt iterations - Model 2')

performance_testdata = model2.evaluate(flatten_testing_images, to_categori
cal(testing_labels))
performance_validation = model2.evaluate(flatten_validation_images,to_cate
gorical(validation_labels))
print("Accuracy on Val samples: {0}".format(performance_validation[1]))
print("Accuracy on Test samples: {0}".format(performance_testdata[1]))

#model3
import warnings
```

```
warnings.filterwarnings("ignore")  # Ignore some warning logs



#  Define a Feed-
Forward Model with 3 hidden layers with dimensions 392 and 196 Neurons
model3 = Sequential([
  Dense(1000, activation='sigmoid', input_shape=(48*48,), name="firsthidde
nlayer"), Dropout(0.5),
  Dense(1000, activation='sigmoid', name="secondhiddenlayer"),
  Dense(1000, activation='sigmoid', name="thirdhiddenlayer"), Dropout(0.25
),
  Dense(1000, activation='sigmoid', name="fourthhiddenlayer"),
  Dense(1000, activation='sigmoid', name="fifthhiddenlayer"), Dropout(0.25
),
  Dense(1000, activation='sigmoid', name="sixthhiddenlayer"),

  Dense(7, activation='softmax'),
])

#  Validate your Model Architecture
print(model3.summary())
# Compile model
model3.compile(optimizer='sgd', loss='categorical_crossentropy',metrics=['
accuracy'],)

# Flatten the images into vectors (1D) for feed forward network
flatten_training_images = training_images.reshape((-1, 48*48))
flatten_testing_images = testing_images.reshape((-1, 48*48))
flatten_validation_images = validation_images.reshape((-1, 48*48))

# Train model

%time model3.fit(flatten_training_images, to_categorical(training_labels),
 epochs=50, batch_size= 500,)
plt.plot(model3.history.history['loss'])
plt.title('Loss wrt iterations - Model 3')
plt.figure()
plt.plot(model3.history.history['accuracy'])
plt.title('Accuracy wrt iterations - Model 3')
performance_training = model3.evaluate(flatten_training_images,to_categori
cal(training_labels))
print("Accuracy on Val samples: {0}".format(performance_training[1]))
performance_validation = model3.evaluate(flatten_validation_images,to_cate
gorical(validation_labels))
```

```
print("Accuracy on Val samples: {0}".format(performance_validation[1]))
```

(d)Image Classification with CNNs:

Implementation:

- Initially all the data of training, testing and validation is converted to a NumPy array. Then the NumPy array is reshaped to 48 x 48 to get into the shape of our required image. Now we have the data of 48 x 48 grayscale images of training, validation and testing ready
- Preprocessing of images is done by normalizing all the images. This helps us in adjusting differences in the values of attributes.
- First CNN model is designed with first two layers as convolution layers with kernel size as 3 and with activation function as 'ReLu'. Next is a max pooling layer with pool size 2 x 2 and a dropout value of 0.25. Next, we have two sets of one convolution layer followed by a max pool layer with same parameters as before
- Finally, the outputs are connected to a dense fully connected layer with 7 nodes with the 'soft max' activation function.
- Model is compiled with optimizer as 'SGD' and loss function as 'cross entropy' with metric as accuracy.
- The CNN model is trained for 300 epochs in which the model will learn optimal way of classifying by forward propagating and backward propagating several times.
- Similarly, two other models with different hyperparameter combinations are implemented and best model is evaluated on the validation set.
- Finally, we evaluate the performance of the model we built on the test data.

Observations:

Hyper parameter combinations used:

CNN Model 1:

- Starting with 2 convolution layers each with 32 filters and filter size of 3x 3 with 'ReLu' activation function followed by a max pooling layer with pool size of (2,2) and dropout of 0.25. Convolution layer with 64 filters and and filter size of 3x 3 with 'ReLu' activation function followed by a max pooling layer with pool size of (2,2) and dropout of 0.25 is repeated twice
- Activation function as 'ReLu' in all layers, SoftMax activation function in the last dense fully connected layer with 512 nodes
- Optimizer: 'sgd', loss: 'categorical cross entropy', metric : 'accuracy'
- 300 epochs

Results of model 1:

1. Emotion classification accuracy on training set: **0.7567839345171**
2. Emotion classification accuracy on validation set: **0.5703538656234741**
3. Running time for training the CNN : **25min 9s (due to 300 epochs)**
4. Number of parameters for each CNN: **593,383**

(plot of cross entropy loss is not plotted because it not asked in the question of CNN)

CNN Model 2:

- Starting with 1 convolution layer with 32 filters and filter size of 3x 3 with 'ReLu' activation function and other convolution layer with 30 filters and filter size of 3x 3 with 'ReLu' activation function followed by a max pooling layer with pool size of (2,2) and dropout of 0.25. Convolution layer with 50 filters and filter size of 3x 3 with 'ReLu' dropout of 0.25 is next layer followed by Convolution layer with 60 filters and 10 filters each with filter size of 3x 3 and 4 x4 with 'ReLu' and dropout of 0.25. Before dense layer we have Max pooling layer with pool size (2,2) and dropout of 0.25.
- Activation function as 'ReLu' in all layers, SoftMax activation function in the last dense fully connected layer with 512 nodes
- Optimizer: 'sgd', loss: 'categorical cross entropy', metric: 'accuracy'

Results of model 2:

5. Emotion classification accuracy on training set: **0.5982345119876**
6. Emotion classification accuracy on validation set: **0.4388408958911896**
7. Running time for training the CNN : **26min 41s (due to 300 epochs)**
8. Number of parameters for each FNN: **308,217**

CNN Model 3:

- Starting with 1 convolution layer with 32 filters and filter size of 3x 3 with 'ReLu' activation function and other convolution layer with 64 filters and filter size of 3x 3 with 'ReLu' activation function followed by a max pooling layer with pool size of (2,2) and dropout of 0.25. Convolution layer with 32 filters and filter size of 3x 3 with 'ReLu' dropout of 0.25 is next layer followed by Convolution layer with 20 filters and 10 filters each with filter size of 3x 3 with 'ReLu'. Before dense layer we have Max pooling layer with pool size (2,2) and dropout of 0.25.
- Activation function as 'ReLu' in all layers, SoftMax activation function in the last dense fully connected layer with 512 nodes
- Optimizer: 'sgd', loss: 'categorical cross entropy', metric: 'accuracy'
- This model is done only for 30 epochs

Results of model 3:

9. Emotion classification accuracy on training set: **0.37698522210121155**
10. Emotion classification accuracy on validation set: **0.31468311100235**
11. Running time for training the CNN : **4mins 35s**

12. Number of parameters for each FNN: **109,677**

When observed from all models it is observed that the accuracy in validation set is better when we train the model for more epochs (300), but the difference between accuracy of training set and validation set is considerable which means the data is overfitting, but still it performs better when performed with more epochs.

In comparison with FNN's,

- CNN model has less number of parameters, since we are doing max pooling and convolution.
- The run time for CNN is observed to be higher than FNN. The runtime of CNN's for 300 epochs is more than the run time of FNN's for 750 epochs due to the multiple operations we perform in CNN.
- In CNN, the accuracy on validation is higher in comparison with FNN where it was around 0.45 in FNN and it was around 0.57 in CNN

(d.ii)

Upon running the three models, the best model found based on the validation set is model **1**
The emotion classification on the testing set for that model is: **0.5501578859834741**

Code for (d.i)and (d.ii) :

```
import warnings
warnings.filterwarnings("ignore")  # Ignore some warning logs
# Define 2 groups of layers: features layer (convolutions) and classificat
ion layer
features = [Conv2D(32, kernel_size=3, activation='relu', input_shape=(48,4
8,1)),
            Conv2D(32, kernel_size=3, activation='relu'),
            MaxPooling2D(pool_size=(2,2)), Dropout(0.25),
            Conv2D(64, kernel_size=3, activation='relu'),
            MaxPooling2D(pool_size=(2,2)), Dropout(0.25),
            Conv2D(64, kernel_size=3, activation='relu'),
            MaxPooling2D(pool_size=(2,2)), Dropout(0.25), Flatten(),]
classifier = [Dense(512, activation='relu'), Dense(7, activation='softmax'
),]

cnn_model1 = Sequential(features+classifier)

print(cnn_model1.summary())  # Compare number of parameteres against FFN
cnn_model1.compile(optimizer='sgd', loss='categorical_crossentropy',metric
s=['accuracy'],)

training_images_3d = training_images.reshape(28709,48,48,1)
```

```
testing_images_3d = testing_images.reshape(3589,48,48,1)
validation_images_3d = validation_images.reshape(3589,48,48,1)

%time cnn_model1.fit(training_images_3d, to_categorical(training_labels),
epochs=300, batch_size=256,)
performance_1 = cnn_model1.evaluate(validation_images_3d, to_categorical(v
alidation_labels))

print("Accuracy on validation samples: {0}".format(performance_1[1]))

# Define 2 groups of layers: features layer (convolutions) and classificat
ion layer
features = [Conv2D(32, kernel_size=3, activation='relu', input_shape=(48,4
8,1)),
            Conv2D(30, kernel_size=3, activation='relu'),
            MaxPooling2D(pool_size=(2,2)), Dropout(0.25),
            Conv2D(50, kernel_size=3, activation='relu'), Dropout(0.5),
            Conv2D(60, kernel_size=3, activation='relu'), Dropout(0.25),
            Conv2D(10, kernel_size=4, activation= 'relu'), Dropout(0.25),
            MaxPooling2D(pool_size=(2,2)), Dropout(0.25), Flatten(),]
classifier = [Dense(500, activation='relu'), Dense(7, activation='softmax'
),]

cnn_model2 = Sequential(features+classifier)

print(cnn_model2.summary())  # Compare number of parameteres against FFN
cnn_model2.compile(optimizer='sgd', loss='categorical_crossentropy',metric
s=['accuracy'],)

training_images_3d = training_images.reshape(28709,48,48,1)
testing_images_3d = testing_images.reshape(3589,48,48,1)
validation_images_3d = validation_images.reshape(3589,48,48,1)

%time cnn_model2.fit(training_images_3d, to_categorical(training_labels),
epochs=300, batch_size=1000,)
performance_2 = cnn_model2.evaluate(validation_images_3d, to_categorical(v
alidation_labels))

print("Accuracy on validation samples: {0}".format(performance_2[1]))

features = [Conv2D(32, kernel_size=3, activation='relu', input_shape=(48,4
8,1)),
            Conv2D(64, kernel_size=3, activation='relu'),
            MaxPooling2D(pool_size=(2,2)), Dropout(0.25),
            Conv2D(32, kernel_size=3, activation='relu'),
```

```
            Conv2D(20, kernel_size=3, activation='relu'),
            Conv2D(10, kernel_size=3, activation='relu'),
            MaxPooling2D(pool_size=(2,2)), Dropout(0.25), Flatten(),]
classifier = [Dense(100, activation='relu'), Dense(7, activation='softmax'
),]

cnn_model3 = Sequential(features+classifier)

print(cnn_model3.summary())  # Compare number of parameteres against FFN
cnn_model3.compile(optimizer='sgd', loss='categorical_crossentropy',metric
s=['accuracy'],)

training_images_3d = training_images.reshape(28709,48,48,1)
testing_images_3d = testing_images.reshape(3589,48,48,1)
validation_images_3d = validation_images.reshape(3589,48,48,1)

%time cnn_model3.fit(training_images_3d, to_categorical(training_labels),
epochs=3, batch_size=200,)
performance_3 = cnn_model3.evaluate(validation_images_3d, to_categorical(v
alidation_labels))

print("Accuracy on Test samples: {0}".format(performance_3[1]))

performance_test = cnn_model1.evaluate(testing_images_3d, to_categorical(t
esting_labels))
print("Accuracy on Test samples: {0}".format(performance_test[1]))
```

(e) Bayesian optimization for hyper-parameter tuning:

Implementation:

- Bayesian optimization is performed using hyper opt library.
- Hyperparameters such as number of layers, filters, dropout and number of neurons is varied and checked for multiple values using hp.choice
- Best hyper parameters are evaluated for multiple trails and given by fmin function.
- Using these hyper parameters, accuracy on validation data and test data is calculated.

Observations:

The best hyperparameters obtained using hyper opt library are dropout value of 2, number of filters : 0, num of layers : 1 number of neurons : 3
Accuracy on validation samples : **0.3262747283448878**
Accuracy on Test samples : **0.3134577**

(Bayesian optimization is implemented with only 50epochs)

Code for (e):

```
from hyperopt import hp, tpe, fmin, Trials, STATUS_OK
import sys
training_images_3d = training_images.reshape(28709,48,48,1)
testing_images_3d = testing_images.reshape(3589,48,48,1)
validation_images_3d = validation_images.reshape(3589,48,48,1)

space = {
    'num_layers':hp.choice('num_layers', [2, 3, 4]),
    'num_filters':hp.choice('num_filters', [12, 24, 36]),
    #'num_convolutions':hp.choice('num_convolutions', [2, 3, 4]),
    'dropout':hp.choice('dropout',[0,0.2,0.4]),
    #'kernel_size':hp.choice('kernel_size', [1,2,3]),
    'num_neurons':hp.choice('num_neurons',[100,200,500]),
}

def train_CNN_model(params):
  model_opt = Sequential()
  model_opt.add(Conv2D(filters = params['num_filters'], kernel_size=3, act
ivation='relu', input_shape=(48,48,1)))
  model_opt.add(Conv2D(filters = params['num_filters'], kernel_size=3, act
ivation='relu'))
  model_opt.add(Dropout(params['dropout']))

  if params['num_layers'] >= 3:
    model_opt.add(Conv2D(filters = params['num_filters'], kernel_size=3, a
ctivation='relu'))
    model_opt.add(Dropout(params['dropout']))
    model_opt.add(MaxPooling2D(pool_size=(2,2)))
    model_opt.add(Dropout(params['dropout']))
if params['num_layers'] == 4:
    model_opt.add(Conv2D(filters = params['num_filters'], kernel_size=3, a
ctivation='relu'))
    model_opt.add(MaxPooling2D(pool_size=(2,2)))
    model_opt.add(Dropout(params['dropout']))

  model_opt.add(Flatten())
  model_opt.add(Dense(params['num_neurons'],activation = 'relu'))
  model_opt.add(Dense(7, activation = 'sigmoid'))
  model_opt.compile(optimizer='sgd', loss='categorical_crossentropy',metri
cs=['accuracy'])
  #print(model_opt.summary())  # Compare number of parameteres against FFN

  model_opt.fit(training_images_3d, to_categorical(training_labels), epoch
s=50, batch_size=500,verbose = 0)
```

```
  performance_opt_val = model_opt.evaluate(validation_images_3d, to_catego
rical(validation_labels))
  performance_opt_test = model_opt.evaluate(testing_images_3d,to_categoric
al(testing_labels))
  print("Accuracy on Validation samples: {0}".format(performance_opt_val[1
]))
  print("Accuracy on Test samples: {0}".format(performance_opt_test[1]))
  sys.stdout.flush()
  return {'loss' : -performance_opt_val[1], 'status': STATUS_OK}
trials = Trials()
best_hyperparams = fmin(train_CNN_model,space, algo = tpe.suggest, max_eva
ls = 10, trials = trials)
print(best_hyperparams)
```

BONUS QUESTIONS:

(f) Fine tuning:

Implementation:

- The pre-trained CNN that we used to finetune on the FER Data is the vgg16 model.
- We imported layers and weights from the model, and we experiment with different models on the validation set.
- We use three models here. For the first model, we directly used the layers of vgg16
- In the second model, we flattened and added two dense layers to the preexisting vgg16 model.
- In the third model, we flattened and added three dense layers to the preexisting vgg16 model.

Observations:

Classification accuracy for on the validation set for Model 1 is **0.48983004689216614**
Classification accuracy for on the validation set for Model 2 is **0.4689328372478485**
Classification accuracy for all hyper-parameter combinations on the validation set for Model 3 is **0.4823070466518402**

The best model among the three as observed on the validation set is : **1**

The classification accuracy for the best hyper-parameter combination for Model 1 on test set is : **0.4837001860141754**

Code for (f) :

```
vgg16_model = VGG16(include_top = False, input_shape = (48,48,3))
vgg16_model.summary()
model = Sequential()
for layer in vgg16_model.layers:
  model.add(layer)
```

```
#Freeze layers
for layer in model.layers[:-2]:
  layer.trainable = False
model.add(Flatten())
model.add(Dense(7, activation = 'softmax'))
model.summary()
training_images_3d = training_images.reshape(28709,48,48,1)
testing_images_3d = testing_images.reshape(3589,48,48,1)
validation_images_3d = validation_images.reshape(3589,48,48,1)

new_train = np.zeros(shape = (training_images_3d.shape[0],training_images_
3d.shape[1],training_images_3d.shape[2],3))
#print(new.shape)
new_test = np.zeros(shape = (testing_images_3d.shape[0],testing_images_3d.
shape[1],testing_images_3d.shape[2],3))
new_val = np.zeros(shape = (validation_images_3d.shape[0],validation_image
s_3d.shape[1],validation_images_3d.shape[2],3))

for i in range(training_images_3d.shape[0]):
  img = training_images_3d[i,:,:,0]
  new_train[i,:,:,0] = img
  new_train[i,:,:,1] = img
  new_train[i,:,:,2] = img
for i in range(validation_images_3d.shape[0]):
  img = validation_images_3d[i,:,:,0]
  new_val[i,:,:,0] = img
  new_val[i,:,:,1] = img
  new_val[i,:,:,2] = img
for i in range(testing_images_3d.shape[0]):
  img = testing_images_3d[i,:,:,0]
  new_test[i,:,:,0] = img
  new_test[i,:,:,1] = img
  new_test[i,:,:,2] = img
model.compile(optimizer='sgd', loss='categorical_crossentropy',metrics=['a
ccuracy'],)
%time model.fit(new_train, to_categorical(training_labels), epochs=20, bat
ch_size=200,)
performance_finetuning = model.evaluate(new_val, to_categorical(validation
_labels))
print("Accuracy on Val samples: {0}".format(performance_finetuning[1]))
performance_finetuning_test = model.evaluate(new_test, to_categorical(test
ing_labels))
print("Accuracy on Val samples: {0}".format(performance_finetuning_test[1]
))
model_2 = Sequential()
```

```
for layer in vgg16_model.layers:
  model_2.add(layer)

#Freeze layers
for layer in model_2.layers[:-1]:
  layer.trainable = False
model_2.add(Flatten())
model_2.add(Dense(20, activation = 'relu'))
model_2.add(Dense(7, activation = 'softmax'))
model_2.summary()
model_2.compile(optimizer='sgd', loss='categorical_crossentropy',metrics=[
'accuracy'],)
%time model_2.fit(new_train, to_categorical(training_labels), epochs=10, b
atch_size=500,)
performance_finetuning_model2 = model_2.evaluate(new_val, to_categorical(v
alidation_labels))
print("Accuracy on Val samples: {0}".format(performance_finetuning_model2[
1]))
performance_finetuning_test_model2 = model_2.evaluate(new_test, to_categor
ical(testing_labels))
print("Accuracy on Test samples: {0}".format(performance_finetuning_test_m
odel2[1]))
model_3 = Sequential()
for layer in vgg16_model.layers:
  model_3.add(layer)
#Freeze layers
for layer in model_3.layers:
  layer.trainable = False
model_3.add(Flatten())
model_3.add(Dense(20, activation = 'relu'))
model_3.add(Dense(10, activation = 'relu'))
model_3.add(Dense(7, activation = 'softmax'))
model_3.summary()

model_3.compile(optimizer='sgd', loss='categorical_crossentropy',metrics=[
'accuracy'],)
%time model_3.fit(new_train, to_categorical(training_labels), epochs=30, b
atch_size=250,)

performance_finetuning_model3 = model_3.evaluate(new_val, to_categorical(v
alidation_labels))
print("Accuracy on Val samples: {0}".format(performance_finetuning_model3[
1]))
```

```
performance_finetuning_test_model3 = model_3.evaluate(new_test, to_categor
ical(testing_labels))
print("Accuracy on Test samples: {0}".format(performance_finetuning_test_m
odel3[1]))
```

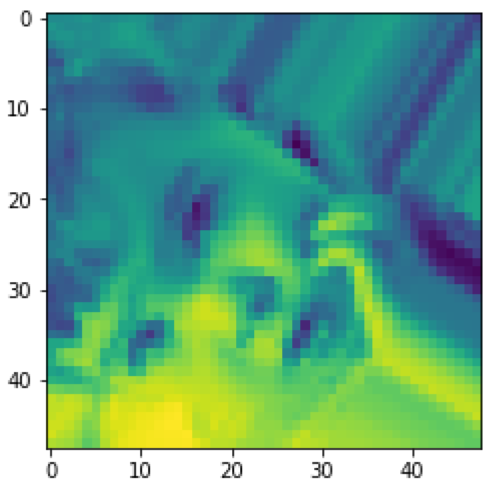
(g) Data Augmentation:

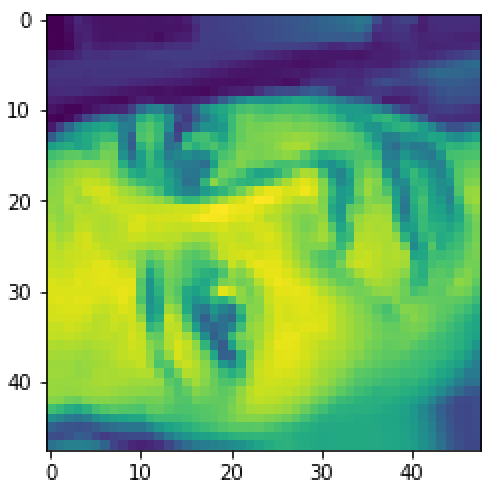
Data Augmentation is a way to increase the size of our dataset and reduce overfitting especially when we use complicated models

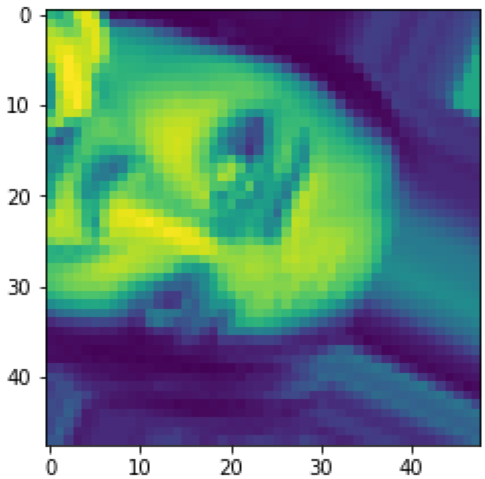
Implementation:

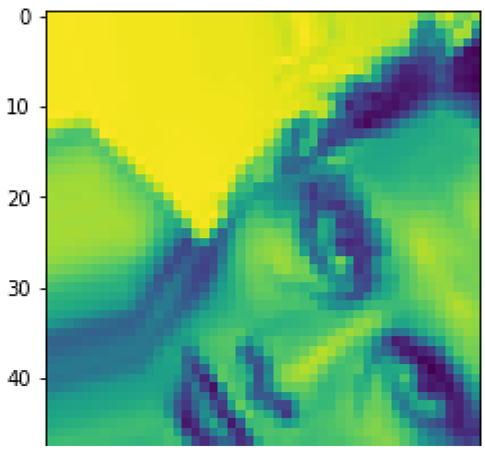
- ImageDataGenerator is imported from preprocessing
- Data generator is created using all the augmenting factors such as feature wise center, feature wise normalization, rotation range, width shift and height shift.
- Using these we now randomly create the new augmented images

Augmented FER Data :









Code for (g) :

```
from keras.preprocessing.image import ImageDataGenerator
#augmenting factors
datagen = ImageDataGenerator(
    featurewise_center=True,
    featurewise_std_normalization=True,
    rotation_range=120,
    width_shift_range=0.4,
    height_shift_range=0.1,
    horizontal_flip=True)
#Create new_data array
new_data = datagen.flow((training_images_3d, training_labels), batch_size=
1)
plt.figure()
plt.imshow(np.squeeze(new_data[1][0]))
plt.figure()
plt.imshow(np.squeeze(new_data[5][0]))
```

```
plt.figure()
plt.imshow(np.squeeze(new_data[15][0]))
plt.figure()
plt.imshow(np.squeeze(new_data[8][0]))
plt.figure()
plt.imshow(np.squeeze(new_data[6][0]))
plt.figure()
plt.imshow(np.squeeze(new_data[9][0]))
```