In [2]:

```
import pandas as pd
import cv2
import numpy as np
```

Imported dataset

Facial Emotion Recognition dataset

In [3]:

```
dataset_path = 'fer2013.csv'
image_size=(48,48)
```

In [4]:

```
dataset_path
```

Out[4]:

```
'fer2013.csv'
```

Load the data

In [10]:

```
def load_fer2013():
    data = pd.read_csv(dataset_path)
    pixels = data['pixels'].tolist()
    width, height = 48, 48
    faces = []
    for pixel_sequence in pixels:
        face = [int(pixel) for pixel in pixel_sequence.split(' ')]
        face = np.asarray(face).reshape(width, height)
        face = cv2.resize(face.astype('uint8'),image_size)
        faces.append(face.astype('float32'))
    faces = np.asarray(faces)
    faces = np.expand_dims(faces, -1)
    emotions = pd.get_dummies(data['emotion']).as_matrix()
    return faces, emotions
```

Preprocessed the data

Converted the dataset in the range of 0 to 1 by diving each pixel by 255

In [11]:

```
def preprocess_input(x, v2=True):
    x = x.astype('float32')
    x = x / 255.0
    if v2:
        x = x - 0.5
        x = x * 2.0
    return x
```

Model CNN

Imported all the required package Used 'relu' as an input for activation function Then used average pooling for dimension reduction. To increase the accuracy and decrease the time, we used Dropout method.

In [5]:

```
from keras.layers import Activation, Convolution2D, Dropout, Conv2D
from keras.layers import AveragePooling2D, BatchNormalization
from keras.layers import GlobalAveragePooling2D
from keras.models import Sequential
from keras.layers import Flatten
from keras.models import Model
from keras.layers import Input
from keras.layers import MaxPooling2D
from keras.layers import SeparableConv2D
from keras import layers
from keras.regularizers import l2
```

```
Using TensorFlow backend.
```

CNN model implementation

Built various CNN model by changing the parameters. Dropped out 20 percent of total connections between the layers randomly. Used softmax as last layer for making pedictions

In [6]:

```
def simple_CNN(input_shape, num_classes):

    model = Sequential()
    model.add(Convolution2D(filters=16, kernel_size=(7, 7), padding='same',
                            name='image_array', input_shape=input_shape))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=16, kernel_size=(7, 7), padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(AveragePooling2D(pool_size=(2, 2), padding='same'))
    model.add(Dropout(.2))

    model.add(Convolution2D(filters=32, kernel_size=(5, 5), padding='same'))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=32, kernel_size=(5, 5), padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(AveragePooling2D(pool_size=(2, 2), padding='same'))
    model.add(Dropout(.2))

    model.add(Convolution2D(filters=64, kernel_size=(3, 3), padding='same'))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=64, kernel_size=(3, 3), padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(AveragePooling2D(pool_size=(2, 2), padding='same'))
    model.add(Dropout(.2))

    model.add(Convolution2D(filters=128, kernel_size=(3, 3), padding='same'))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=128, kernel_size=(3, 3), padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(AveragePooling2D(pool_size=(2, 2), padding='same'))
    model.add(Dropout(.2))

    model.add(Convolution2D(filters=256, kernel_size=(3, 3), padding='same'))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=num_classes, kernel_size=(3, 3), padding='same'))
    model.add(GlobalAveragePooling2D())
    model.add(Activation('softmax',name='predictions'))
    return model

def simpler_CNN(input_shape, num_classes):

    model = Sequential()
    model.add(Convolution2D(filters=16, kernel_size=(5, 5), padding='same',
                            name='image_array', input_shape=input_shape))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=16, kernel_size=(5, 5),
                            strides=(2, 2), padding='same'))
                            strides=(2, 2), padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(.2))

    model.add(Convolution2D(filters=32, kernel_size=(5, 5), padding='same'))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=32, kernel_size=(5, 5),
                            strides=(2, 2), padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(.2))

    model.add(Convolution2D(filters=64, kernel_size=(3, 3), padding='same'))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=64, kernel_size=(3, 3),
                            strides=(2, 2), padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(.2))

    model.add(Convolution2D(filters=64, kernel_size=(1, 1), padding='same'))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=128, kernel_size=(3, 3),
                            strides=(2, 2), padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(.2))

    model.add(Convolution2D(filters=256, kernel_size=(1, 1), padding='same'))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=128, kernel_size=(3, 3),
                            strides=(2, 2), padding='same'))

    model.add(Convolution2D(filters=256, kernel_size=(1, 1), padding='same'))
    model.add(BatchNormalization())
    model.add(Convolution2D(filters=num_classes, kernel_size=(3, 3),
                            strides=(2, 2), padding='same'))

    model.add(Flatten())
    #model.add(GlobalAveragePooling2D())
    model.add(Activation('softmax',name='predictions'))
    return model

def tiny_XCEPTION(input_shape, num_classes, l2_regularization=0.01):
    regularization = l2(l2_regularization)

    # base
    img_input = Input(input_shape)
    x = Conv2D(5, (3, 3), strides=(1, 1), kernel_regularizer=regularization,
                                            use_bias=False)(img_input)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(5, (3, 3), strides=(1, 1), kernel_regularizer=regularization,
                                            use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    # module 1
    residual = Conv2D(8, (1, 1), strides=(2, 2),
                      padding='same', use_bias=False)(x)
    residual = BatchNormalization()(residual)

    x = SeparableConv2D(8, (3, 3), padding='same',
                        kernel_regularizer=regularization,
                        use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = SeparableConv2D(8, (3, 3), padding='same',
                        kernel_regularizer=regularization,
                        use_bias=False)(x)
    x = BatchNormalization()(x)

    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
    x = layers.add([x, residual])

    # module 2
    residual = Conv2D(16, (1, 1), strides=(2, 2),
    residual = Conv2D(16, (1, 1), strides=(2, 2),
                      padding='same', use_bias=False)(x)
    residual = BatchNormalization()(residual)

    x = SeparableConv2D(16, (3, 3), padding='same',
                        kernel_regularizer=regularization,
                        use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = SeparableConv2D(16, (3, 3), padding='same',
                        kernel_regularizer=regularization,
                        use_bias=False)(x)
    x = BatchNormalization()(x)

    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
    x = layers.add([x, residual])

    # module 3
    residual = Conv2D(32, (1, 1), strides=(2, 2),
                      padding='same', use_bias=False)(x)
    residual = BatchNormalization()(residual)

    x = SeparableConv2D(32, (3, 3), padding='same',
                        kernel_regularizer=regularization,
                        use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = SeparableConv2D(32, (3, 3), padding='same',
                        kernel_regularizer=regularization,
                        use_bias=False)(x)
    x = BatchNormalization()(x)

    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
    x = layers.add([x, residual])

    # module 4
    residual = Conv2D(64, (1, 1), strides=(2, 2),
                      padding='same', use_bias=False)(x)
    residual = BatchNormalization()(residual)

    x = SeparableConv2D(64, (3, 3), padding='same',
                        kernel_regularizer=regularization,
                        use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = SeparableConv2D(64, (3, 3), padding='same',
                        kernel_regularizer=regularization,
                        use_bias=False)(x)
    x = BatchNormalization()(x)

    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
    x = layers.add([x, residual])

    x = Conv2D(num_classes, (3, 3),
            #kernel_regularizer=regularization,
            padding='same')(x)
    x = GlobalAveragePooling2D()(x)
    output = Activation('softmax',name='predictions')(x)

    model = Model(img_input, output)
    return model


def mini_XCEPTION(input_shape, num_classes, l2_regularization=0.01):
    regularization = l2(l2_regularization)

    # base
    img_input = Input(input_shape)
    x = Conv2D(8, (3, 3), strides=(1, 1), kernel_regularizer=regularization,
                                            use_bias=False)(img_input)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = Conv2D(8, (3, 3), strides=(1, 1), kernel_regularizer=regularization,
                                            use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    # module 1
    # module 1
    residual = Conv2D(16, (1, 1), strides=(2, 2),
                      padding='same', use_bias=False)(x)
    residual = BatchNormalization()(residual)

    x = SeparableConv2D(16, (3, 3), padding='same',
                        kernel_regularizer=regularization,
                        use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = SeparableConv2D(16, (3, 3), padding='same',
                        kernel_regularizer=regularization,
                        use_bias=False)(x)
    x = BatchNormalization()(x)

    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
    x = layers.add([x, residual])

    # module 2
    residual = Conv2D(32, (1, 1), strides=(2, 2),
                      padding='same', use_bias=False)(x)
    residual = BatchNormalization()(residual)

    x = SeparableConv2D(32, (3, 3), padding='same',
                        kernel_regularizer=regularization,
                        use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = SeparableConv2D(32, (3, 3), padding='same',
                        kernel_regularizer=regularization,
                        use_bias=False)(x)
    x = BatchNormalization()(x)

    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
    x = layers.add([x, residual])

    # module 3
    residual = Conv2D(64, (1, 1), strides=(2, 2),
                      padding='same', use_bias=False)(x)
    residual = BatchNormalization()(residual)

    x = SeparableConv2D(64, (3, 3), padding='same',
                        kernel_regularizer=regularization,
                        use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = SeparableConv2D(64, (3, 3), padding='same',
                        kernel_regularizer=regularization,
                        use_bias=False)(x)
    x = BatchNormalization()(x)

    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
    x = layers.add([x, residual])

    # module 4
    residual = Conv2D(128, (1, 1), strides=(2, 2),
                      padding='same', use_bias=False)(x)
    residual = BatchNormalization()(residual)

    x = SeparableConv2D(128, (3, 3), padding='same',
                        kernel_regularizer=regularization,
                        use_bias=False)(x)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)
    x = SeparableConv2D(128, (3, 3), padding='same',
                        kernel_regularizer=regularization,
                        use_bias=False)(x)
    x = BatchNormalization()(x)

    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
    x = layers.add([x, residual])

    x = Conv2D(num_classes, (3, 3),
            #kernel_regularizer=regularization,
            padding='same')(x)
    x = GlobalAveragePooling2D()(x)
    output = Activation('softmax',name='predictions')(x)
    model = Model(img_input, output)
    return model

def big_XCEPTION(input_shape, num_classes):
    img_input = Input(input_shape)
    x = Conv2D(32, (3, 3), strides=(2, 2), use_bias=False)(img_input)
    x = BatchNormalization(name='block1_conv1_bn')(x)
    x = Activation('relu', name='block1_conv1_act')(x)
    x = Conv2D(64, (3, 3), use_bias=False)(x)
    x = BatchNormalization(name='block1_conv2_bn')(x)
    x = Activation('relu', name='block1_conv2_act')(x)

    residual = Conv2D(128, (1, 1), strides=(2, 2),
                      padding='same', use_bias=False)(x)
    residual = BatchNormalization()(residual)

    x = SeparableConv2D(128, (3, 3), padding='same', use_bias=False)(x)
    x = BatchNormalization(name='block2_sepconv1_bn')(x)
    x = Activation('relu', name='block2_sepconv2_act')(x)
    x = SeparableConv2D(128, (3, 3), padding='same', use_bias=False)(x)
    x = BatchNormalization(name='block2_sepconv2_bn')(x)

    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
    x = layers.add([x, residual])

    residual = Conv2D(256, (1, 1), strides=(2, 2),
                      padding='same', use_bias=False)(x)
    residual = BatchNormalization()(residual)

    x = Activation('relu', name='block3_sepconv1_act')(x)
    x = SeparableConv2D(256, (3, 3), padding='same', use_bias=False)(x)
    x = BatchNormalization(name='block3_sepconv1_bn')(x)
    x = Activation('relu', name='block3_sepconv2_act')(x)
    x = SeparableConv2D(256, (3, 3), padding='same', use_bias=False)(x)
    x = BatchNormalization(name='block3_sepconv2_bn')(x)

    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
    x = layers.add([x, residual])
    x = Conv2D(num_classes, (3, 3),
            #kernel_regularizer=regularization,
            padding='same')(x)
    x = GlobalAveragePooling2D()(x)
    output = Activation('softmax',name='predictions')(x)

    model = Model(img_input, output)
    return model


if __name__ == "__main__":
    input_shape = (64, 64, 1)
    num_classes = 7
    #model = tiny_XCEPTION(input_shape, num_classes)
    #model.summary()
    #model = mini_XCEPTION(input_shape, num_classes)
    #model.summary()
    #model = big_XCEPTION(input_shape, num_classes)
    #model.summary()
    model = simple_CNN((48, 48, 1), num_classes)
    model.summary()
```

```
WARNING:tensorflow:From C:\Users\rosha\Anaconda3\lib\site-
packages\tensorflow\python\framework\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 C:\Users\rosha\Anaconda3\lib\site-
packages\keras\backend\tensorflow_backend.py:3445: calling dropout (from
tensorflow.python.ops.nn_ops) with keep_prob is deprecated 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`.
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
image_array (Conv2D)         (None, 48, 48, 16)        800       
```

```
_________________________________________________________________
batch_normalization_1 (Batch (None, 48, 48, 16)        64        
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 48, 48, 16)        12560     
_________________________________________________________________
batch_normalization_2 (Batch (None, 48, 48, 16)        64        
_________________________________________________________________
activation_1 (Activation)    (None, 48, 48, 16)        0         
_________________________________________________________________
average_pooling2d_1 (Average (None, 24, 24, 16)        0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 24, 24, 16)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 32)        12832     
_________________________________________________________________
batch_normalization_3 (Batch (None, 24, 24, 32)        128       
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 24, 24, 32)        25632     
_________________________________________________________________
batch_normalization_4 (Batch (None, 24, 24, 32)        128       
_________________________________________________________________
activation_2 (Activation)    (None, 24, 24, 32)        0         
_________________________________________________________________
average_pooling2d_2 (Average (None, 12, 12, 32)        0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 12, 12, 32)        0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 12, 12, 64)        18496     
_________________________________________________________________
batch_normalization_5 (Batch (None, 12, 12, 64)        256       
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 12, 12, 64)        36928     
_________________________________________________________________
batch_normalization_6 (Batch (None, 12, 12, 64)        256       
_________________________________________________________________
activation_3 (Activation)    (None, 12, 12, 64)        0         
_________________________________________________________________
average_pooling2d_3 (Average (None, 6, 6, 64)          0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 6, 6, 64)          0         
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 6, 6, 128)         73856     
_________________________________________________________________
batch_normalization_7 (Batch (None, 6, 6, 128)         512       
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 6, 6, 128)         147584    
_________________________________________________________________
batch_normalization_8 (Batch (None, 6, 6, 128)         512       
_________________________________________________________________
activation_4 (Activation)    (None, 6, 6, 128)         0         
_________________________________________________________________
average_pooling2d_4 (Average (None, 3, 3, 128)         0         
_________________________________________________________________
dropout_4 (Dropout)          (None, 3, 3, 128)         0         
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 3, 3, 256)         295168    
_________________________________________________________________
batch_normalization_9 (Batch (None, 3, 3, 256)         1024      
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 3, 3, 7)           16135     
_________________________________________________________________
global_average_pooling2d_1 ( (None, 7)                 0         
_________________________________________________________________
predictions (Activation)     (None, 7)                 0         
=================================================================
Total params: 642,935
Trainable params: 641,463
Non-trainable params: 1,472
_________________________________________________________________
```

Train the model

In [7]:

```
from keras.callbacks import CSVLogger, ModelCheckpoint, EarlyStopping
from keras.callbacks import ReduceLROnPlateau
from keras.preprocessing.image import ImageDataGenerator
#from load_and_process import load_fer2013
#from load_and_process import preprocess_input
#from models.cnn import mini_XCEPTION
from sklearn.model_selection import train_test_split
```

Input Parameters

In [9]:

```
batch_size = 32
num_epochs = 35
input_shape = (48, 48, 1)
validation_split = .2
verbose = 1
num_classes = 7
patience = 50
```

In [10]:

```
data_generator = ImageDataGenerator(
                        featurewise_center=False,
                        featurewise_std_normalization=False,
                        rotation_range=10,
                        width_shift_range=0.1,
                        height_shift_range=0.1,
                        zoom_range=.1,
                        horizontal_flip=True)
```

In [11]:

```
# model parameters/compilation
model = mini_XCEPTION(input_shape, num_classes)
model.compile(optimizer='adam', loss='categorical_crossentropy',
              metrics=['accuracy'])
model.summary()
```

```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, 48, 48, 1)    0
__________________________________________________________________________________________________
conv2d_10 (Conv2D)              (None, 46, 46, 8)    72          input_1[0][0]
__________________________________________________________________________________________________
batch_normalization_10 (BatchNo (None, 46, 46, 8)    32          conv2d_10[0][0]
__________________________________________________________________________________________________
activation_5 (Activation)       (None, 46, 46, 8)    0           batch_normalization_10[0][0]
__________________________________________________________________________________________________
conv2d_11 (Conv2D)              (None, 44, 44, 8)    576         activation_5[0][0]
__________________________________________________________________________________________________
batch_normalization_11 (BatchNo (None, 44, 44, 8)    32          conv2d_11[0][0]
__________________________________________________________________________________________________
activation_6 (Activation)       (None, 44, 44, 8)    0           batch_normalization_11[0][0]
__________________________________________________________________________________________________
separable_conv2d_1 (SeparableCo (None, 44, 44, 16)   200         activation_6[0][0]
__________________________________________________________________________________________________
batch_normalization_13 (BatchNo (None, 44, 44, 16)   64          separable_conv2d_1[0][0]
__________________________________________________________________________________________________
activation_7 (Activation)       (None, 44, 44, 16)   0           batch_normalization_13[0][0]
__________________________________________________________________________________________________
separable_conv2d_2 (SeparableCo (None, 44, 44, 16)   400         activation_7[0][0]
__________________________________________________________________________________________________
batch_normalization_14 (BatchNo (None, 44, 44, 16)   64          separable_conv2d_2[0][0]
__________________________________________________________________________________________________
conv2d_12 (Conv2D)              (None, 22, 22, 16)   128         activation_6[0][0]
__________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, 22, 22, 16)   0           batch_normalization_14[0][0]
__________________________________________________________________________________________________
batch_normalization_12 (BatchNo (None, 22, 22, 16)   64          conv2d_12[0][0]
__________________________________________________________________________________________________
add_1 (Add)                     (None, 22, 22, 16)   0           max_pooling2d_1[0][0]
```

```
add_1 (Add)                     (None, 22, 22, 16)   0           max_pooling2d_1[0][0]
                                                                 batch_normalization_12[0][0]
__________________________________________________________________________________________________
separable_conv2d_3 (SeparableCo (None, 22, 22, 32)   656         add_1[0][0]
__________________________________________________________________________________________________
batch_normalization_16 (BatchNo (None, 22, 22, 32)   128         separable_conv2d_3[0][0]
__________________________________________________________________________________________________
activation_8 (Activation)       (None, 22, 22, 32)   0           batch_normalization_16[0][0]
__________________________________________________________________________________________________
separable_conv2d_4 (SeparableCo (None, 22, 22, 32)   1312        activation_8[0][0]
__________________________________________________________________________________________________
batch_normalization_17 (BatchNo (None, 22, 22, 32)   128         separable_conv2d_4[0][0]
__________________________________________________________________________________________________
conv2d_13 (Conv2D)              (None, 11, 11, 32)   512         add_1[0][0]
__________________________________________________________________________________________________
max_pooling2d_2 (MaxPooling2D)  (None, 11, 11, 32)   0           batch_normalization_17[0][0]
__________________________________________________________________________________________________
batch_normalization_15 (BatchNo (None, 11, 11, 32)   128         conv2d_13[0][0]
__________________________________________________________________________________________________
add_2 (Add)                     (None, 11, 11, 32)   0           max_pooling2d_2[0][0]
                                                                 batch_normalization_15[0][0]
__________________________________________________________________________________________________
separable_conv2d_5 (SeparableCo (None, 11, 11, 64)   2336        add_2[0][0]
__________________________________________________________________________________________________
batch_normalization_19 (BatchNo (None, 11, 11, 64)   256         separable_conv2d_5[0][0]
__________________________________________________________________________________________________
activation_9 (Activation)       (None, 11, 11, 64)   0           batch_normalization_19[0][0]
__________________________________________________________________________________________________
separable_conv2d_6 (SeparableCo (None, 11, 11, 64)   4672        activation_9[0][0]
__________________________________________________________________________________________________
batch_normalization_20 (BatchNo (None, 11, 11, 64)   256         separable_conv2d_6[0][0]
__________________________________________________________________________________________________
conv2d_14 (Conv2D)              (None, 6, 6, 64)     2048        add_2[0][0]
__________________________________________________________________________________________________
max_pooling2d_3 (MaxPooling2D)  (None, 6, 6, 64)     0           batch_normalization_20[0][0]
__________________________________________________________________________________________________
batch_normalization_18 (BatchNo (None, 6, 6, 64)     256         conv2d_14[0][0]
__________________________________________________________________________________________________
add_3 (Add)                     (None, 6, 6, 64)     0           max_pooling2d_3[0][0]
                                                                 batch_normalization_18[0][0]
__________________________________________________________________________________________________
separable_conv2d_7 (SeparableCo (None, 6, 6, 128)    8768        add_3[0][0]
__________________________________________________________________________________________________
batch_normalization_22 (BatchNo (None, 6, 6, 128)    512         separable_conv2d_7[0][0]
__________________________________________________________________________________________________
activation_10 (Activation)      (None, 6, 6, 128)    0           batch_normalization_22[0][0]
__________________________________________________________________________________________________
separable_conv2d_8 (SeparableCo (None, 6, 6, 128)    17536       activation_10[0][0]
__________________________________________________________________________________________________
batch_normalization_23 (BatchNo (None, 6, 6, 128)    512         separable_conv2d_8[0][0]
__________________________________________________________________________________________________
conv2d_15 (Conv2D)              (None, 3, 3, 128)    8192        add_3[0][0]
__________________________________________________________________________________________________
max_pooling2d_4 (MaxPooling2D)  (None, 3, 3, 128)    0           batch_normalization_23[0][0]
__________________________________________________________________________________________________
batch_normalization_21 (BatchNo (None, 3, 3, 128)    512         conv2d_15[0][0]
__________________________________________________________________________________________________
add_4 (Add)                     (None, 3, 3, 128)    0           max_pooling2d_4[0][0]
                                                                 batch_normalization_21[0][0]
__________________________________________________________________________________________________
conv2d_16 (Conv2D)              (None, 3, 3, 7)      8071        add_4[0][0]
__________________________________________________________________________________________________
global_average_pooling2d_2 (Glo (None, 7)            0           conv2d_16[0][0]
__________________________________________________________________________________________________
predictions (Activation)        (None, 7)            0           global_average_pooling2d_2[0][0]
==================================================================================================
Total params: 58,423
Trainable params: 56,951
Non-trainable params: 1,472
__________________________________________________________________________________________________
```

In [18]:

```
faces, emotions = load_fer2013()
faces, emotions
```

```
C:\Users\rosha\Anaconda3\lib\site-packages\ipykernel_launcher.py:13: FutureWarning: Method
.as_matrix will be removed in a future version. Use .values instead.
  del sys.path[0]
```

Out[18]:

```
(array([[[[ 70.],
          [ 80.],
          [ 82.],
          ...,
          [ 52.],
          [ 43.],
          [ 41.]],

         [[ 65.],
          [ 61.],
          [ 58.],
          ...,
          [ 56.],
          [ 52.],
          [ 44.]],

         [[ 50.],
          [ 43.],
          [ 54.],
          ...,
          [ 49.],
          [ 56.],
          [ 47.]],

         ...,

         [[ 91.],
          [ 65.],
          [ 42.],
          ...,
          [ 72.],
          [ 56.],
          [ 43.]],

         [[ 77.],
          [ 82.],
          [ 79.],
          ...,
          [105.],
          [ 70.],
          [ 46.]],

         [[ 77.],
          [ 72.],
          [ 84.],
          ...,
          [106.],
          [109.],
          [ 82.]]],


        [[[151.],
          [150.],
          [147.],
          ...,
          [129.],
          [140.],
          [120.]],

         [[151.],
          [149.],
          [149.],
          ...,
          [122.],
          [141.],
          [137.]],

         [[151.],
          [151.],
          [156.],
             ...,
             [109.],
             [123.],
             [146.]],

            ...,

            [[188.],
             [188.],
             [121.],
             ...,
             [185.],
             [185.],
             [186.]],

            [[188.],
             [187.],
             [196.],
             ...,
             [186.],
             [182.],
             [187.]],

            [[186.],
             [184.],
             [185.],
             ...,
             [193.],
             [183.],
             [184.]]],


           [[[231.],
             [212.],
             [156.],
             ...,
             [ 44.],
             [ 27.],
             [ 16.]],

            [[229.],
             [175.],
             [148.],
             ...,
             [ 27.],
             [ 35.],
             [ 27.]],

            [[214.],
             [156.],
             [157.],
             ...,
             [ 28.],
             [ 22.],
             [ 28.]],

            ...,

            [[241.],
             [245.],
             [250.],
             ...,
             [ 57.],
             [101.],
             [146.]],

            [[246.],
             [250.],
             [252.],
             ...,
             [ 78.],
             [105.],
             [162.]],

            [[250.],
             [251.],
             [250.],
  ...,
  [ 88.],
  [110.],
  [152.]]],


...,


[[[ 17.],
  [ 17.],
  [ 16.],
  ...,
  [ 83.],
  [114.],
  [245.]],

 [[ 18.],
  [ 17.],
  [ 16.],
  ...,
  [104.],
  [136.],
  [253.]],

 [[ 19.],
  [ 16.],
  [ 17.],
  ...,
  [128.],
  [152.],
  [255.]],

 ...,

 [[  4.],
  [ 21.],
  [ 46.],
  ...,
  [186.],
  [180.],
  [187.]],

 [[  5.],
  [ 17.],
  [ 41.],
  ...,
  [177.],
  [172.],
  [176.]],

 [[ 20.],
  [ 15.],
  [ 22.],
  ...,
  [154.],
  [133.],
  [113.]]],


[[[ 30.],
  [ 28.],
  [ 28.],
  ...,
  [ 60.],
  [ 50.],
  [ 44.]],

 [[ 30.],
  [ 27.],
  [ 28.],
  ...,
  [ 64.],
  [ 52.],
  [ 40.]],

 [[ 31.],
  [ 28.],
  [ 30.],
  ...,
  [ 61.],
  [ 54.],
  [ 37.]],

 ...,

 [[104.],
  [109.],
  [110.],
  ...,
  [ 35.],
  [ 30.],
  [ 30.]],

 [[102.],
  [105.],
  [108.],
  ...,
  [ 35.],
  [ 31.],
  [ 29.]],

 [[ 93.],
  [ 96.],
  [100.],
  ...,
  [ 35.],
  [ 30.],
  [ 28.]]],


[[[ 19.],
  [ 13.],
  [ 14.],
  ...,
  [108.],
  [ 95.],
  [ 86.]],

 [[ 16.],
  [ 17.],
  [ 15.],
  ...,
  [105.],
  [ 94.],
  [ 90.]],

 [[ 10.],
  [  9.],
  [ 10.],
  ...,
  [101.],
  [ 93.],
  [ 95.]],

 ...,

 [[ 18.],
  [ 14.],
  [ 16.],
  ...,
  [ 55.],
  [ 64.],
  [ 95.]],

 [[ 15.],
  [ 15.],
  [ 13.],
  ...,
  [123.],
  [171.],
  [192.]],

 [[ 16.],
         [ 14.],
         [ 13.],
         ...,
         [189.],
         [199.],
         [201.]]]], dtype=float32), array([[1, 0, 0, ..., 0, 0, 0],
       [1, 0, 0, ..., 0, 0, 0],
       [0, 0, 1, ..., 0, 0, 0],
       ...,
       [1, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 1, ..., 0, 0, 0]], dtype=uint8))
```

In [19]:

```
faces = preprocess_input(faces)
```

In [20]:

```
num_samples, num_classes = emotions.shape
num_samples, num_classes
```

Out[20]:

```
(35887, 7)
```

Compile the model

Splited the data into 80% as training data and 20% as testing data. Trained the model by 35 number of epochs. Used xtest data as validation dataset.

In [21]:

```
xtrain, xtest,ytrain,ytest = train_test_split(faces, emotions,test_size=0.2,shuffle=True)
model.fit_generator(data_generator.flow(xtrain, ytrain,
                                        batch_size),
                        steps_per_epoch=len(xtrain) / batch_size,
                        epochs=num_epochs, verbose=1,
                        validation_data=(xtest,ytest))
```

```
WARNING:tensorflow:From C:\Users\rosha\Anaconda3\lib\site-
packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is
deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Epoch 1/35
898/897 [==============================] - 759s 845ms/step - loss: 1.7620 - acc: 0.3348 -
val_loss: 1.7058 - val_acc: 0.4198
Epoch 2/35
898/897 [==============================] - 730s 813ms/step - loss: 1.5030 - acc: 0.4337 -
val_loss: 1.6158 - val_acc: 0.4100
Epoch 3/35
898/897 [==============================] - 732s 815ms/step - loss: 1.3918 - acc: 0.4762 -
val_loss: 1.3649 - val_acc: 0.4928
Epoch 4/35
898/897 [==============================] - 730s 813ms/step - loss: 1.3181 - acc: 0.5068 -
val_loss: 1.3208 - val_acc: 0.4974
Epoch 5/35
898/897 [==============================] - 730s 813ms/step - loss: 1.2741 - acc: 0.5232 -
val_loss: 1.2893 - val_acc: 0.5171
Epoch 6/35
898/897 [==============================] - 732s 815ms/step - loss: 1.2433 - acc: 0.5357 -
val_loss: 1.3610 - val_acc: 0.4950
Epoch 7/35
898/897 [==============================] - 730s 813ms/step - loss: 1.2144 - acc: 0.5405 -
val_loss: 1.1786 - val_acc: 0.5665
Epoch 8/35
898/897 [==============================] - 731s 814ms/step - loss: 1.1844 - acc: 0.5573 -
val_loss: 1.1845 - val_acc: 0.5699
Epoch 9/35
898/897 [==============================] - 731s 814ms/step - loss: 1.1657 - acc: 0.5615 -
```

```
val_loss: 1.1728 - val_acc: 0.5610
Epoch 10/35
898/897 [==============================] - 732s 815ms/step - loss: 1.1515 - acc: 0.5682 -
val_loss: 1.1915 - val_acc: 0.5619
Epoch 11/35
898/897 [==============================] - 730s 813ms/step - loss: 1.1350 - acc: 0.5759 -
val_loss: 1.2300 - val_acc: 0.5493
Epoch 12/35
898/897 [==============================] - 731s 814ms/step - loss: 1.1276 - acc: 0.5737 -
val_loss: 1.1354 - val_acc: 0.5715
Epoch 13/35
898/897 [==============================] - 732s 815ms/step - loss: 1.1153 - acc: 0.5816 -
val_loss: 1.1654 - val_acc: 0.5699
Epoch 14/35
898/897 [==============================] - 732s 815ms/step - loss: 1.1049 - acc: 0.5862 -
val_loss: 1.1572 - val_acc: 0.5861
Epoch 15/35
898/897 [==============================] - 731s 814ms/step - loss: 1.0943 - acc: 0.5877 -
val_loss: 1.1551 - val_acc: 0.5745
Epoch 16/35
898/897 [==============================] - 731s 815ms/step - loss: 1.0810 - acc: 0.5928 -
val_loss: 1.1195 - val_acc: 0.5933
Epoch 17/35
898/897 [==============================] - 733s 817ms/step - loss: 1.0754 - acc: 0.5976 -
val_loss: 1.1067 - val_acc: 0.5914
Epoch 18/35
898/897 [==============================] - 734s 817ms/step - loss: 1.0677 - acc: 0.5971 -
val_loss: 1.0827 - val_acc: 0.6000
Epoch 19/35
898/897 [==============================] - 734s 817ms/step - loss: 1.0621 - acc: 0.6020 -
val_loss: 1.0697 - val_acc: 0.6066
Epoch 20/35
898/897 [==============================] - 742s 826ms/step - loss: 1.0563 - acc: 0.6060 -
val_loss: 1.1106 - val_acc: 0.5906
Epoch 21/35
898/897 [==============================] - 732s 815ms/step - loss: 1.0530 - acc: 0.6040 -
val_loss: 1.0672 - val_acc: 0.6032
Epoch 22/35
898/897 [==============================] - 731s 815ms/step - loss: 1.0443 - acc: 0.6094 -
val_loss: 1.1238 - val_acc: 0.5865
Epoch 23/35
898/897 [==============================] - 731s 814ms/step - loss: 1.0375 - acc: 0.6114 -
val_loss: 1.0879 - val_acc: 0.6002
Epoch 24/35
898/897 [==============================] - 732s 815ms/step - loss: 1.0295 - acc: 0.6170 -
val_loss: 1.0691 - val_acc: 0.6052
Epoch 25/35
898/897 [==============================] - 732s 815ms/step - loss: 1.0261 - acc: 0.6148 -
val_loss: 1.1269 - val_acc: 0.5946
Epoch 26/35
898/897 [==============================] - 731s 814ms/step - loss: 1.0256 - acc: 0.6148 -
val_loss: 1.1123 - val_acc: 0.5970
Epoch 27/35
898/897 [==============================] - 732s 815ms/step - loss: 1.0204 - acc: 0.6173 -
val_loss: 1.1333 - val_acc: 0.5750
Epoch 28/35
898/897 [==============================] - 731s 814ms/step - loss: 1.0131 - acc: 0.6211 -
val_loss: 1.1156 - val_acc: 0.6004
Epoch 29/35
898/897 [==============================] - 732s 815ms/step - loss: 1.0058 - acc: 0.6248 -
val_loss: 1.1046 - val_acc: 0.5897
Epoch 30/35
898/897 [==============================] - 731s 815ms/step - loss: 1.0110 - acc: 0.6230 -
val_loss: 1.1054 - val_acc: 0.5996
Epoch 31/35
898/897 [==============================] - 730s 813ms/step - loss: 0.9985 - acc: 0.6289 -
val_loss: 1.0338 - val_acc: 0.6204
Epoch 32/35
898/897 [==============================] - 732s 815ms/step - loss: 0.9978 - acc: 0.6260 -
val_loss: 1.0474 - val_acc: 0.6106
Epoch 33/35
898/897 [==============================] - 732s 815ms/step - loss: 0.9910 - acc: 0.6312 -
val_loss: 1.0498 - val_acc: 0.6190
Epoch 34/35
898/897 [==============================] - 731s 815ms/step - loss: 0.9896 - acc: 0.6291 -
val_loss: 1.0605 - val_acc: 0.6089
Epoch 35/35
```

```
898/897 [==============================] - 733s 816ms/step - loss: 0.9897 - acc: 0.6281 -
val_loss: 1.0230 - val_acc: 0.6258
```

Out[21]:

```
<keras.callbacks.History at 0x1f313f20208>
```

In [51]:

```
xtrain.shape
```

Out[51]:

```
(28709, 48, 48, 1)
```

In [48]:

```
ytrain.shape
```

Out[48]:

```
(28709, 7)
```

In [ ]:

```

```

In [ ]:

```

```

Call model

In [12]:

```
from keras.preprocessing.image import img_to_array
import imutils
import cv2
from keras.models import load_model
import numpy as np
```

Parametersfor loading data and images

In [13]:

```
# parameters for loading data and images
detection_model_path = 'haarcascade_frontalface_default.xml'
emotion_model_path = '_mini_XCEPTION.102-0.66.hdf5'
```

In [14]:

```
# loading models
face_detection = cv2.CascadeClassifier(detection_model_path)
emotion_classifier = load_model(emotion_model_path, compile=False)
EMOTIONS = ["angry" ,"disgust","scared", "happy", "sad", "surprised",
 "neutral"]
```

Test the data using Computer vision

The dataset was tested on real time images by using computer vision. Based on the probabilities, each emotion is classified.

In [ ]:

```
# starting video streaming
cv2.namedWindow('Emotion_Recognition')
camera = cv2.VideoCapture(0)
while True:
    frame = camera.read()[1]
    #reading the frame
    frame = imutils.resize(frame,width=700)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_detection.detectMultiScale(gray,scaleFactor=1.1,minNeighbors=5,minSize=(30,30),fla
gs=cv2.CASCADE_SCALE_IMAGE)

    canvas = np.zeros((250, 300, 3), dtype="uint8")
    frameClone = frame.copy()
    if len(faces) > 0:
        faces = sorted(faces, reverse=True,
        key=lambda x: (x[2] - x[0]) * (x[3] - x[1]))[0]
        (fX, fY, fW, fH) = faces
                    # Extract the ROI of the face from the grayscale image, resize it to a fixed 2
x28 pixels, and then prepare
            # the ROI for classification via the CNN
        roi = gray[fY:fY + fH, fX:fX + fW]
        roi = cv2.resize(roi, (64, 64))
        roi = roi.astype("float") / 255.0
        roi = img_to_array(roi)
        roi = np.expand_dims(roi, axis=0)


        preds = emotion_classifier.predict(roi)[0]
        emotion_probability = np.max(preds)
        label = EMOTIONS[preds.argmax()]


        for (i, (emotion, prob)) in enumerate(zip(EMOTIONS,preds)):
                # construct the label text
            text = "{}: {:.2f}%".format(emotion, prob * 100)

                # draw the label + probability bar on the canvas
               # emoji_face = feelings_faces[np.argmax(preds)]


            w = int(prob * 300)
            cv2.rectangle(canvas, (7, (i * 35) + 5),
            (w, (i * 35) + 35), (0, 0, 255), -1)
            cv2.putText(canvas, text, (10, (i * 35) + 23),
            cv2.FONT_HERSHEY_SIMPLEX, 0.45,
            (255, 255, 255), 2)
            cv2.putText(frameClone, label, (fX, fY - 10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 0, 255), 2)
            cv2.rectangle(frameClone, (fX, fY), (fX + fW, fY + fH),
                              (0, 0, 255), 2)
#    for c in range(0, 3):
#        frame[200:320, 10:130, c] = emoji_face[:, :, c] * \
#        (emoji_face[:, :, 3] / 255.0) + frame[200:320,
#        10:130, c] * (1.0 - emoji_face[:, :, 3] / 255.0)


        cv2.imshow('your_face', frameClone)
        cv2.imshow("Probabilities", canvas)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

camera.release()
cv2.destroyAllWindows()
```

Description and Conclusion

CNN

The main advantage of CNN compared to its predecessors is that it automatically detects the important features without any human supervision. CNN is also computationally efficient. It uses special convolution and pooling operations and performs parameter sharing. Convolution The main building block of CNN is the convolutional layer. Convolution is a mathematical operation to merge two sets of information. In our case the convolution is applied on the input data using a convolution filter to produce a feature map

On the left side is the input to the convolution layer, for example the input image. On the right is the convolution filter, also called the kernel, we will use these terms interchangeably. This is called a 3x3 convolution due to the shape of the filter. We perform multiple convolutions on an input, each using a different filter and resulting in a distinct feature map. For any kind of neural network to be powerful, it needs to contain non-linearity. We again pass the result of the convolution operation through relu activation function. So the values in the final feature maps are not actually the sums, but the relu function applied to them. Stride specifies how much we move the convolution filter at each step. By default the value is 1.

Pooling

After a convolution operation we usually perform pooling to reduce the dimensionality. type of pooling is max pooling which just takes the max value in the pooling window. Average pooling which just takes the avg of all the values in the pooling window. Then we flatten the output of the final pooling layer to a vector and that becomes the input to the fully connected layer.

Fully Connected Layer(FC)

Fully connected layer involves weights, biases, and neurons. It connects neurons in one layer to neurons in another layer. It is used to classify images between different category by training.

Softmax / Logistic Layer

Softmax or Logistic layer is the last layer of CNN. It resides at the end of FC layer. Logistic is used for binary classification and softmax is for multi-classification.(gives 7 outputs in our case)

Output Layer

Output layer contains the label which is in the form of one-hot encoded.

Accuracy

The model accuracy is 63% and its highest when compared with Random Forest and Nearest Neighbor.

In [ ]: