In [33]:

```
import glob
from shutil import copyfile

emotions = ["neutral", "anger", "contempt", "disgust", "fear", "happy", "sadness", "surprise"] #Define emotion order

participants = glob.glob("source_emotion/*") #Returns a list of all folders with participant numbers

for x in participants:
    part = "%s" %x[-4:] #store current participant number

    for sessions in glob.glob("%s/*" %x): #Store list of sessions for current participant
        for files in glob.glob("%s/*" %sessions):
            current_session = files[20:-30]

            file = open(files, 'r')
            emotion = int(float(file.readline())) #emotions are encoded as a float, readline as float, then convert to intege
            sourcefile_emotion = glob.glob("source_images/%s/%s/*" %(part, current_session))[-1] #get path for last image in
            sourcefile_neutral = glob.glob("source_images/%s/%s/*" %(part, current_session))[0] #do same for neutral image
            dest_neut = "sorted_set/neutral/%s" %sourcefile_neutral[25:] #Generate path to put neutral image
            dest_emot = "sorted_set/%s/%s" %(emotions[emotion], sourcefile_emotion[25:]) #Do same for emotion containing image
            copyfile(sourcefile_neutral, dest_neut) #Copy file
            copyfile(sourcefile_emotion, dest_emot) #Copy file
```

In [34]:
```
import cv2
import glob

faceDet = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
faceDet_two = cv2.CascadeClassifier("haarcascade_frontalface_alt2.xml")
faceDet_three = cv2.CascadeClassifier("haarcascade_frontalface_alt.xml")
faceDet_four = cv2.CascadeClassifier("haarcascade_frontalface_alt_tree.xml")

emotions = ["neutral", "anger", "contempt", "disgust", "fear", "happy", "sadness", "surprise"] #Define emotions


def detect_faces(emotion):
    files = glob.glob("sorted_set/%s/*" %emotion) #Get list of all images with emotion
    filenumber = 0

    for f in files:

        frame = cv2.imread(f) #Open image
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) #Convert image to grayscale

        #Detect face using 4 different classifiers
        face = faceDet.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=10, minSize=(5, 5), flags=cv2.CASCADE_SCALE_IMAGE
        face_two = faceDet_two.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=10, minSize=(5, 5), flags=cv2.CASCADE_SCA
        face_three = faceDet_three.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=10, minSize=(5, 5), flags=cv2.CASCADE_
        face_four = faceDet_four.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=10, minSize=(5, 5), flags=cv2.CASCADE_S

        #Go over detected faces, stop at first detected face, return empty if no face.
        if len(face) == 1:
            facefeatures = face
        elif len(face_two) == 1:
            facefeatures = face_two
        elif len(face_three) == 1:
            facefeatures = face_three
        elif len(face_four) == 1:
            facefeatures = face_four
        else:
            facefeatures = ""

        #Cut and save face
        for (x, y, w, h) in facefeatures: #get coordinates and size of rectangle containing face
            print ("face found in file: %s" %f)
            gray = gray[y:y+h, x:x+w] #Cut the frame to size
            try:
                out = cv2.resize(gray, (350, 350)) #Resize face so all images have same size
                cv2.imwrite("dataset/%s/%s.jpg" %(emotion, filenumber), out) #Write image
            except:
               pass #If error, pass file

        filenumber += 1 #Increment image number


for emotion in emotions:
    detect_faces(emotion) #Call function
```

```
face found in file: sorted_set/neutral/28_011_00000009.png
face found in file: sorted_set/neutral/38_005_00000009.png
face found in file: sorted_set/neutral/37_005_00000026.png
face found in file: sorted_set/neutral/30_013_00000009.png
face found in file: sorted_set/neutral/37_001_00000006.png
face found in file: sorted_set/neutral/27_001_00000014.png
face found in file: sorted_set/neutral/16_001_00000013.png
face found in file: sorted_set/neutral/38_001_00000001.png
face found in file: sorted_set/neutral/09_003_00000009.png
face found in file: sorted_set/neutral/11_001_00000009.png
face found in file: sorted_set/neutral/30_001_00000008.png
face found in file: sorted_set/neutral/15_004_00000002.png
face found in file: sorted_set/neutral/26_008_00000025.png
face found in file: sorted_set/neutral/51_002_00000004.png
face found in file: sorted_set/neutral/38_007_00000006.png
face found in file: sorted_set/neutral/16_007_00000014.png
face found in file: sorted_set/neutral/34_004_00000003.png
face found in file: sorted_set/neutral/25_008_00000007.png
face found in file: sorted_set/neutral/27_010_00000009.png
```

In [35]:
```
import cv2
import glob
import random
import numpy as np

emotions = ["neutral", "anger", "contempt", "disgust", "fear", "happy", "sadness", "surprise"] #Emotion list
fishface = cv2.face.FisherFaceRecognizer_create()
data = {}


def get_files(emotion): #Define function to get file list, randomly shuffle it and split 80/20
    files = glob.glob("dataset/%s/*" %emotion)
    random.shuffle(files)
    training = files[:int(len(files)*0.8)] #get first 80% of file list
    prediction = files[-int(len(files)*0.2):] #get last 20% of file list
    return training, prediction


def make_sets():
    training_data = []
    training_labels = []
    prediction_data = []
    prediction_labels = []

    for emotion in emotions:
        training, prediction = get_files(emotion)
        #Append data to training and prediction list, and generate labels 0-7
        for item in training:
            image = cv2.imread(item) #open image
            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) #convert to grayscale
            training_data.append(gray) #append image array to training data list
            training_labels.append(emotions.index(emotion))
        for item in prediction: #repeat above process for prediction set
            image = cv2.imread(item)
            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            prediction_data.append(gray)
            prediction_labels.append(emotions.index(emotion))

    return training_data, training_labels, prediction_data, prediction_labels


def run_recognizer():
    training_data, training_labels, prediction_data, prediction_labels = make_sets()
    print("training fisher face classifier")
    print("size of training set is:", len(training_labels), "images")
    fishface.train(training_data, np.asarray(training_labels))
    print("predicting classification set")
    cnt = 0
    correct = 0
    incorrect = 0

    for image in prediction_data:
        pred, conf = fishface.predict(image)
        if pred == prediction_labels[cnt]:
            correct += 1
            cnt += 1
        else:
            incorrect += 1
            cnt += 1
    return ((100*correct)/(correct + incorrect))


#Now run it
metascore = []
for i in range(0,10):
    correct = run_recognizer()
    print("got", correct, "percent correct!")
    metascore.append(correct)

print("\n\nend score:", np.mean(metascore), "percent correct!")
```

```
training fisher face classifier
size of training set is: 521 images
predicting classification set
got 25.984251968503937 percent correct!
training fisher face classifier
size of training set is: 521 images
predicting classification set
got 19.68503937007874 percent correct!
training fisher face classifier
size of training set is: 521 images
predicting classification set
got 17.322834645669293 percent correct!
training fisher face classifier
size of training set is: 521 images
predicting classification set
got 14.960629921259843 percent correct!
training fisher face classifier
```

```
training fisher face classifier
size of training set is: 521 images
predicting classification set
```

In [38]:
```
#Change from:
emotions = ["neutral", "anger", "contempt", "disgust", "fear", "happy", "sadness", "surprise"]
#To:
emotions = ["neutral", "anger", "disgust", "fear", "happy", "surprise"]
```

In [39]:
```
def run_recognizer():

    training_data, training_labels, prediction_data, prediction_labels = make_sets()

    print("training fisher face classifier")
    print("size of training set is:", len(training_labels), "images")

    fishface.train(training_data, np.asarray(training_labels))

    print("predicting classification set")

    cnt = 0
    correct = 0
    incorrect = 0

    for image in prediction_data:
        pred, conf = fishface.predict(image)

        if pred == prediction_labels[cnt]:
            correct += 1
            cnt += 1
        else:
            cv2.imwrite("difficult/%s_%s_%s.jpg" %(emotions[prediction_labels[cnt]], emotions[pred], cnt), image) #<-- this o

    return ((100*correct)/(correct + incorrect))

run_recognizer()
```

```
training fisher face classifier
size of training set is: 485 images
predicting classification set
```

Out[39]:
```
26.89075630252101
```