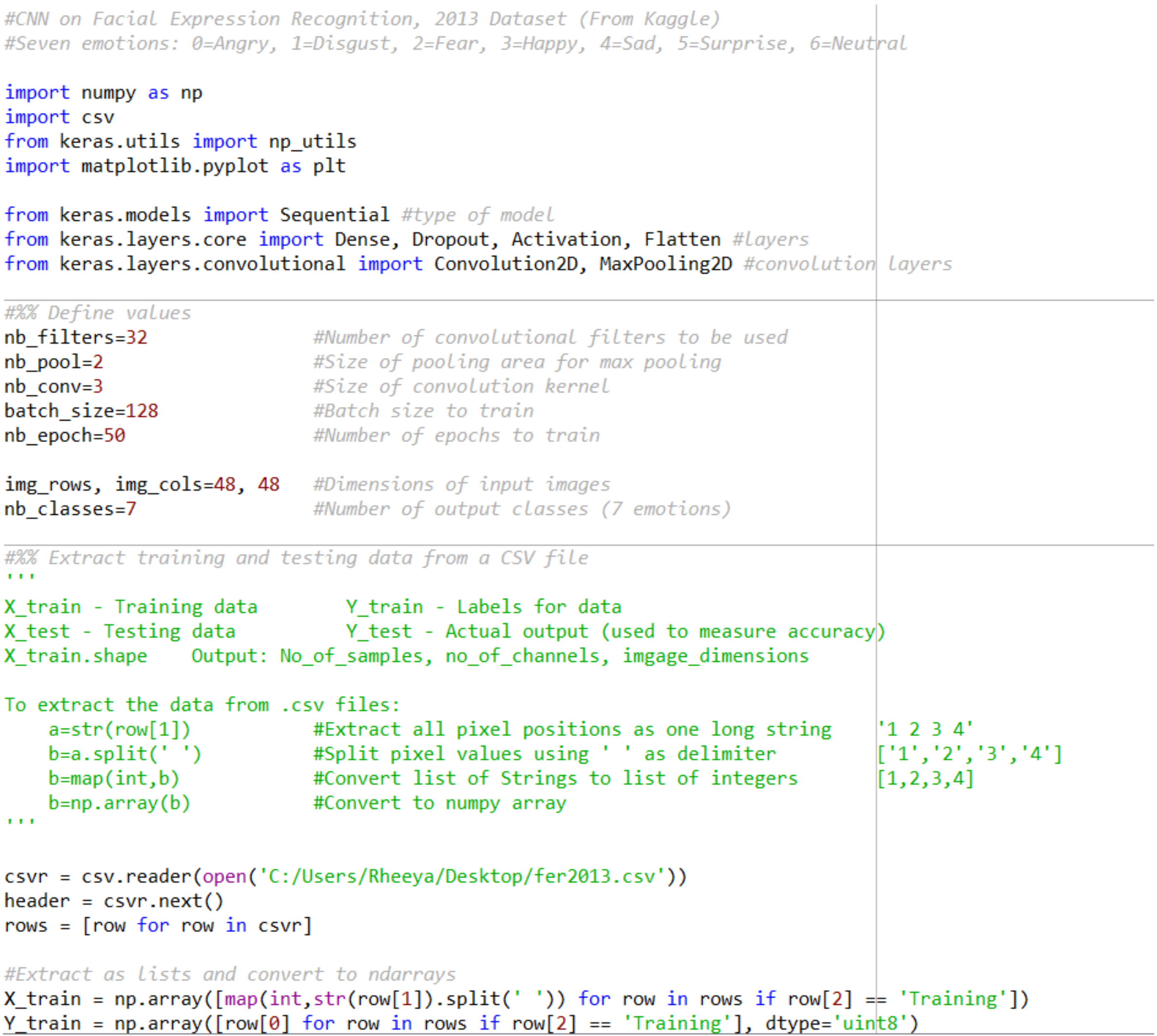
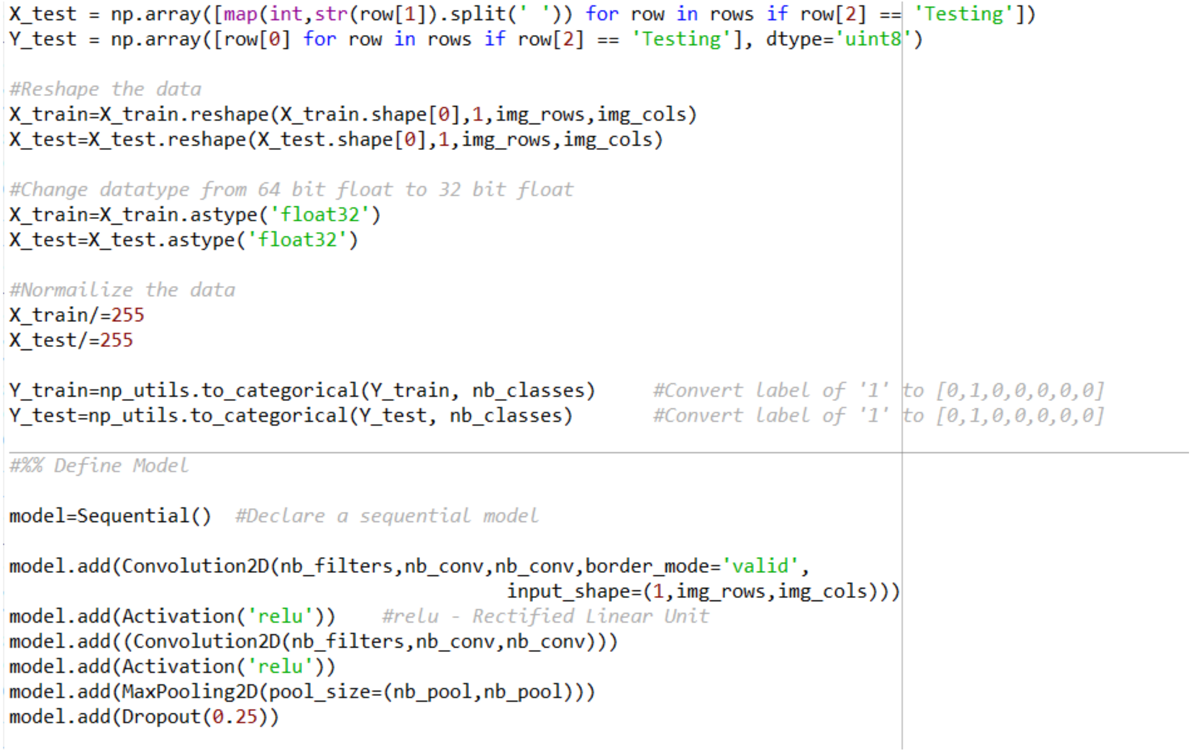
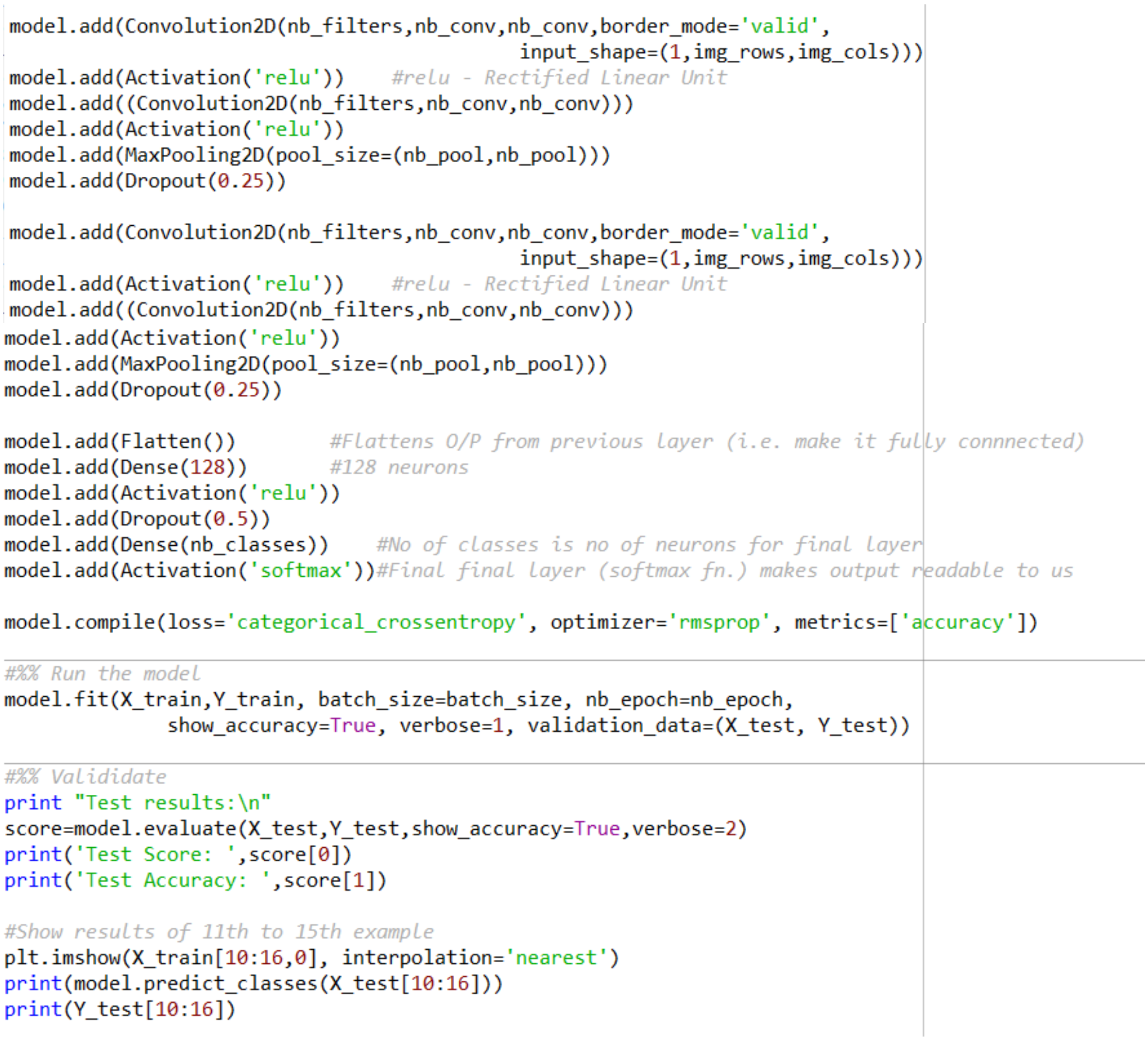


```
#CNN on Facial Expression Recognition, 2013 Dataset (From Kaggle)
#Seven emotions: 0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral

import numpy as np
import csv
from keras.utils import np_utils
import matplotlib.pyplot as plt

from keras.models import Sequential #type of model
from keras.layers.core import Dense, Dropout, Activation, Flatten #layers
from keras.layers.convolutional import Convolution2D, MaxPooling2D #convolution layers

#%% Define values
nb_filters=32               #Number of convolutional filters to be used
nb_pool=2                   #Size of pooling area for max pooling
nb_conv=3                   #Size of convolution kernel
batch_size=128              #Batch size to train
nb_epoch=50                 #Number of epochs to train

img_rows, img_cols=48, 48   #Dimensions of input images
nb_classes=7                #Number of output classes (7 emotions)

#%% Extract training and testing data from a CSV file
'''
X_train - Training data        Y_train - Labels for data
X_test - Testing data          Y_test - Actual output (used to measure accuracy)
X_train.shape    Output: No_of_samples, no_of_channels, imgage_dimensions

To extract the data from .csv files:
    a=str(row[1])           #Extract all pixel positions as one long string     '1 2 3 4'
    b=a.split(' ')          #Split pixel values using ' ' as delimiter          ['1','2','3','4']
    b=map(int,b)            #Convert list of Strings to list of integers        [1,2,3,4]
    b=np.array(b)           #Convert to numpy array
'''

csvr = csv.reader(open('C:/Users/Rheeya/Desktop/fer2013.csv'))
header = csvr.next()
rows = [row for row in csvr]

#Extract as lists and convert to ndarrays
X_train = np.array([map(int,str(row[1]).split(' ')) for row in rows if row[2] == 'Training'])
Y_train = np.array([row[0] for row in rows if row[2] == 'Training'], dtype='uint8')
```



```
X_test = np.array([map(int,str(row[1]).split(' ')) for row in rows if row[2] == 'Testing'])
Y_test = np.array([row[0] for row in rows if row[2] == 'Testing'], dtype='uint8')

#Reshape the data
X_train=X_train.reshape(X_train.shape[0],1,img_rows,img_cols)
X_test=X_test.reshape(X_test.shape[0],1,img_rows,img_cols)

#Change datatype from 64 bit float to 32 bit float
X_train=X_train.astype('float32')
X_test=X_test.astype('float32')

#Normailize the data
X_train/=255
X_test/=255

Y_train=np_utils.to_categorical(Y_train, nb_classes)     #Convert label of '1' to [0,1,0,0,0,0,0]
Y_test=np_utils.to_categorical(Y_test, nb_classes)       #Convert label of '1' to [0,1,0,0,0,0,0]

#%% Define Model

model=Sequential()  #Declare a sequential model

model.add(Convolution2D(nb_filters,nb_conv,nb_conv,border_mode='valid',
                                            input_shape=(1,img_rows,img_cols)))
model.add(Activation('relu'))     #relu - Rectified Linear Unit
model.add((Convolution2D(nb_filters,nb_conv,nb_conv)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(nb_pool,nb_pool)))
model.add(Dropout(0.25))
```



```
model.add(Convolution2D(nb_filters,nb_conv,nb_conv,border_mode='valid',
                                            input_shape=(1,img_rows,img_cols)))
model.add(Activation('relu'))    #relu - Rectified Linear Unit
model.add((Convolution2D(nb_filters,nb_conv,nb_conv)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(nb_pool,nb_pool)))
model.add(Dropout(0.25))

model.add(Convolution2D(nb_filters,nb_conv,nb_conv,border_mode='valid',
                                            input_shape=(1,img_rows,img_cols)))
model.add(Activation('relu'))    #relu - Rectified Linear Unit
model.add((Convolution2D(nb_filters,nb_conv,nb_conv)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(nb_pool,nb_pool)))
model.add(Dropout(0.25))

model.add(Flatten())        #Flattens O/P from previous layer (i.e. make it fully connnected)
model.add(Dense(128))       #128 neurons
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(nb_classes))    #No of classes is no of neurons for final layer
model.add(Activation('softmax'))#Final final layer (softmax fn.) makes output readable to us

model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy'])

#%% Run the model
model.fit(X_train,Y_train, batch_size=batch_size, nb_epoch=nb_epoch,
              show_accuracy=True, verbose=1, validation_data=(X_test, Y_test))

#%% Valididate
print "Test results:\n"
score=model.evaluate(X_test,Y_test,show_accuracy=True,verbose=2)
print('Test Score: ',score[0])
print('Test Accuracy: ',score[1])

#Show results of 11th to 15th example
plt.imshow(X_train[10:16,0], interpolation='nearest')
print(model.predict_classes(X_test[10:16]))
print(Y_test[10:16])
```