```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from matplotlib.colors import LogNorm
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Conv2D,Flatten,MaxPooling2D,Dropout,BatchNormalizat
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import Adam
import os
```

```
x = pd.read_csv('/content/drive/MyDrive/CV/fer2013 (1).csv')
print (x.values.shape)
x.head()
```

```
(35887, 3)
```

	emotion	pixels	Usage
0	0	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...	Training
1	0	151 150 147 155 148 133 111 140 170 174 182 15...	Training
2	2	231 212 156 164 174 138 161 173 182 200 106 38...	Training
3	4	24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 1...	Training
4	6	4 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84...	Training

```
y = x.values[:,0]
pixels = x.values[:, 1]
print (type(pixels))
print (len(pixels))
print (len(pixels[0]))
print (pixels[10][10])
```

```
<class 'numpy.ndarray'>
35887
8287
3
```

```
p = pixels[10].split(' ')
print(len(p))
```

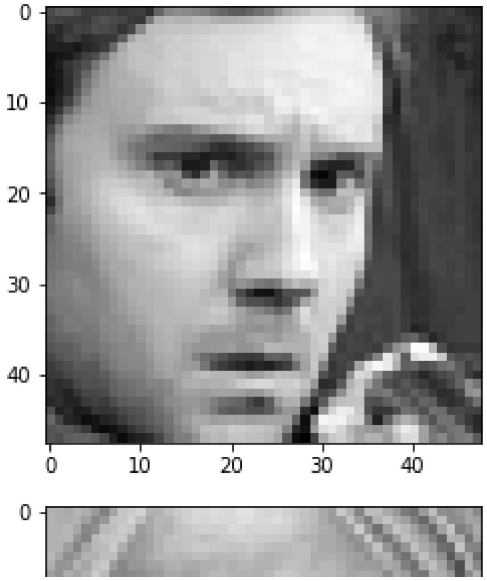
```
2304
```

```
X = np.zeros((pixels.shape[0], 48*48))
```

```
for ix in range(X.shape[0]):
    p = pixels[ix].split(' ')
    for iy in range(X.shape[1]):
        X[ix, iy] = int(p[iy])


temp=X


for ix in range(4):
    plt.figure(ix)
    plt.imshow(temp[ix].reshape((48, 48)), interpolation='none', cmap='gray')
plt.show()
```



X

```
array([[ 70.,  80.,  82., ..., 106., 109.,  82.],
       [151., 150., 147., ..., 193., 183., 184.],
       [231., 212., 156., ...,  88., 110., 152.],
       ...,
       [ 17.,  17.,  16., ..., 154., 133., 113.],
       [ 30.,  28.,  28., ...,  35.,  30.,  28.],
       [ 19.,  13.,  14., ..., 189., 199., 201.]])
```

y

```
array([0, 0, 2, ..., 0, 3, 2], dtype=object)
```

```
X=X/255
```

X

```
array([[0.2745098 , 0.31372549, 0.32156863, ..., 0.41568627, 0.42745098,
        0.32156863],
       [0.59215686, 0.58823529, 0.57647059, ..., 0.75686275, 0.71764706,
        0.72156863],
       [0.90588235, 0.83137255, 0.61176471, ..., 0.34509804, 0.43137255,
        0.59607843],
       ...,
       [0.06666667, 0.06666667, 0.0627451 , ..., 0.60392157, 0.52156863,
        0.44313725],
       [0.11764706, 0.10980392, 0.10980392, ..., 0.1372549 , 0.11764706,
        0.10980392],
       [0.0745098 , 0.05098039, 0.05490196, ..., 0.74117647, 0.78039216,
        0.78823529]])
```

```
X_train = X[0:30000, :]
Y_train = y[0:30000]
print (X_train.shape, Y_train.shape)
```

```
X_test = X[30000:32300,:]
Y_test = y[30000:32300]
print (X_test.shape, Y_test.shape)
```

```
(30000, 2304) (30000,)
(2300, 2304) (2300,)
```

```
X_train = X_train.reshape((X_train.shape[0], 48, 48,1 ))
X_test  = X_test.reshape((X_test.shape[0], 48, 48,1))
```

```
Y_train = to_categorical(Y_train)
Y_test  = to_categorical(Y_test)
print(Y_train.shape)
print(Y_test.shape)
```

```
(30000, 7)
(2300, 7)
```

```
from tensorflow.keras.callbacks import ReduceLROnPlateau
lr_reduce = ReduceLROnPlateau(monitor='val_acc', factor=0.1, epsilon=0.0001, patience=1, verb
```

```
WARNING:tensorflow:`epsilon` argument is deprecated and will be removed, use `min_delta
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
        featurewise_center=False,
        samplewise_center=False,
        featurewise_std_normalization=False,
        samplewise_std_normalization=False,
        zca_whitening=False,
        rotation_range=10,
        zoom_range = 0.0,
        width_shift_range=0.1,
        height_shift_range=0.1,
        horizontal_flip=False,
        vertical_flip=False)

datagen.fit(X_train)
```

```
model = Sequential()

model.add(Conv2D(64, kernel_size=(3, 3), activation='relu', input_shape=(48,48,1)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
```

```
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.22))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(7, activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='adam' ,
              metrics=['acc'])

print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 46, 46, 64)        640
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 44, 44, 64)        36928
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 22, 22, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 22, 22, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 20, 20, 128)       73856
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 10, 10, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 8, 8, 128)         147584
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
dropout_1 (Dropout)          (None, 4, 4, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 512)               1049088
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 7)                 1799
=================================================================
Total params: 1,441,223
Trainable params: 1,441,223
```

```
Non-trainable params: 0
_________________________________________________________________
None
```

```
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-7) ,
              metrics=['acc'])
```

```
/usr/local/lib/python3.7/dist-packages/tensorflow/python/keras/optimizer_v2/optimizer_v
  "The `lr` argument is deprecated, use `learning_rate` instead.")
```

```
batch_size = 64
epochs = 40
```

```
history = model.fit(X_train, Y_train,
                    batch_size=batch_size,
                    validation_data=(X_test, Y_test),
                    epochs = epochs,
                    shuffle=True,
                    verbose = 2)
```

```
Epoch 1/40
469/469 - 546s - loss: 1.8243 - acc: 0.2463 - val_loss: 1.8055 - val_acc: 0.2561
Epoch 2/40
469/469 - 552s - loss: 1.7420 - acc: 0.2865 - val_loss: 1.5376 - val_acc: 0.3974
Epoch 3/40
469/469 - 549s - loss: 1.5200 - acc: 0.4039 - val_loss: 1.3872 - val_acc: 0.4809
Epoch 4/40
469/469 - 549s - loss: 1.4048 - acc: 0.4583 - val_loss: 1.3109 - val_acc: 0.4952
Epoch 5/40
469/469 - 548s - loss: 1.3367 - acc: 0.4855 - val_loss: 1.2140 - val_acc: 0.5257
Epoch 6/40
469/469 - 547s - loss: 1.2835 - acc: 0.5073 - val_loss: 1.2023 - val_acc: 0.5352
Epoch 7/40
469/469 - 546s - loss: 1.2356 - acc: 0.5271 - val_loss: 1.1713 - val_acc: 0.5509
Epoch 8/40
469/469 - 546s - loss: 1.2072 - acc: 0.5409 - val_loss: 1.1631 - val_acc: 0.5526
Epoch 9/40
469/469 - 544s - loss: 1.1775 - acc: 0.5515 - val_loss: 1.1626 - val_acc: 0.5591
Epoch 10/40
469/469 - 546s - loss: 1.1499 - acc: 0.5643 - val_loss: 1.1280 - val_acc: 0.5696
Epoch 11/40
469/469 - 545s - loss: 1.1208 - acc: 0.5747 - val_loss: 1.1182 - val_acc: 0.5735
Epoch 12/40
469/469 - 543s - loss: 1.0990 - acc: 0.5845 - val_loss: 1.1062 - val_acc: 0.5796
Epoch 13/40
469/469 - 543s - loss: 1.0797 - acc: 0.5930 - val_loss: 1.0898 - val_acc: 0.5791
Epoch 14/40
469/469 - 545s - loss: 1.0569 - acc: 0.5970 - val_loss: 1.0995 - val_acc: 0.5865
Epoch 15/40
469/469 - 542s - loss: 1.0466 - acc: 0.6046 - val_loss: 1.1097 - val_acc: 0.5783
Epoch 16/40
```

```
469/469 - 541s - loss: 1.0279 - acc: 0.6127 - val_loss: 1.0877 - val_acc: 0.6000
Epoch 17/40
469/469 - 541s - loss: 1.0076 - acc: 0.6198 - val_loss: 1.0748 - val_acc: 0.5909
Epoch 18/40
469/469 - 542s - loss: 0.9942 - acc: 0.6260 - val_loss: 1.0767 - val_acc: 0.5965
Epoch 19/40
469/469 - 542s - loss: 0.9728 - acc: 0.6340 - val_loss: 1.0804 - val_acc: 0.5896
Epoch 20/40
469/469 - 541s - loss: 0.9629 - acc: 0.6374 - val_loss: 1.0701 - val_acc: 0.5974
Epoch 21/40
469/469 - 541s - loss: 0.9452 - acc: 0.6434 - val_loss: 1.0845 - val_acc: 0.5930
Epoch 22/40
469/469 - 540s - loss: 0.9301 - acc: 0.6485 - val_loss: 1.0804 - val_acc: 0.5970
Epoch 23/40
469/469 - 541s - loss: 0.9095 - acc: 0.6583 - val_loss: 1.1016 - val_acc: 0.5961
Epoch 24/40
469/469 - 540s - loss: 0.9048 - acc: 0.6614 - val_loss: 1.0783 - val_acc: 0.6039
Epoch 25/40
469/469 - 540s - loss: 0.8849 - acc: 0.6697 - val_loss: 1.0719 - val_acc: 0.6065
Epoch 26/40
469/469 - 540s - loss: 0.8709 - acc: 0.6724 - val_loss: 1.0867 - val_acc: 0.5978
Epoch 27/40
469/469 - 540s - loss: 0.8609 - acc: 0.6771 - val_loss: 1.0859 - val_acc: 0.6026
Epoch 28/40
469/469 - 540s - loss: 0.8528 - acc: 0.6848 - val_loss: 1.0883 - val_acc: 0.5965
Epoch 29/40
469/469 - 540s - loss: 0.8368 - acc: 0.6860 - val_loss: 1.0779 - val_acc: 0.6004
Epoch 30/40
```

```
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```
model.save('emotion.h5')
```

0s completed at 2:44 PM