Topic : Neural Network for Image Classification

Objective for this template:

1. To introduce participants to the basic pipeline for Image classification using a multinomial logistic regression.
2. Use tensorflow to build a simple sequential neural network that implements a multinomial logistic regression.
3. Demonstrate the process of training the model and evaluating its performance

Designed By: *Rodolfo C. Raga Jr.*

Step 1:

```
#KERAS
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation, Flatten
from keras.layers.convolutional import Convolution2D, MaxPooling2D
from tensorflow.keras.optimizers import SGD,RMSprop,Adam
from keras.utils import np_utils

import numpy as np
import matplotlib.pyplot as plt
import matplotlib
import os
from PIL import Image
from numpy import *
# SKLEARN
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix

# input image dimensions
img_rows, img_cols = 200, 200

# number of channels
img_channels = 1

from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

Step 2: Setup directory for raw and resized data

```
path1 = "/content/gdrive/My Drive/face_data"    #path of folder of 
images
path2 = "/content/gdrive/My Drive/processed_faces"  #path of folder to 
save images
print("Directory path is set")
```

```
Directory path is set
```

Step 3 : Check number of raw images stored in input directory

```
listing = os.listdir(path1)
num_samples=size(listing)
print ("Total number of raw images is {}".format(num_samples))
```

```
Total number of raw images is 400
```

Step 4: Resize images and convert to grayscale

```
for file in listing:
    im = Image.open(path1 + '/' + file)
    img = im.resize((img_rows,img_cols))
    gray = img.convert('L')
                #need to do some more processing here
    gray.save(path2 +'/' +  file, "JPEG")

imlist = os.listdir(path2)
print ("Raw images converted to following filenames 
{}".format(imlist))
```

```
Raw images converted to following filenames ['happy_00012.JPG', 
'happy_00024.JPG', 'happy_00007.JPG', 'happy_00017.JPG', 
'happy_00021.JPG', 'happy_00001.JPG', 'happy_00022.JPG', 
'happy_00004.JPG', 'happy_00020.JPG', 'happy_00018.JPG', 
'happy_00015.JPG', 'happy_00003.JPG', 'happy_00011.JPG', 
'happy_00019.JPG', 'happy_00026.JPG', 'happy_00027.JPG', 
'happy_00023.JPG', 'happy_00005.JPG', 'happy_00016.JPG', 
'happy_00025.JPG', 'happy_00014.JPG', 'happy_00006.JPG', 
'happy_00013.JPG', 'happy_00009.JPG', 'happy_00002.JPG', 
'happy_00010.JPG', 'happy_00008.JPG', 'happy_00032.JPG', 
'happy_00037.JPG', 'happy_00042.JPG', 'happy_00038.JPG', 
'happy_00041.JPG', 'happy_00047.JPG', 'happy_00043.JPG', 
'happy_00051.JPG', 'happy_00035.JPG', 'happy_00030.JPG', 
'happy_00049.JPG', 'happy_00029.JPG', 'happy_00039.JPG', 
'happy_00046.JPG', 'happy_00044.JPG', 'happy_00031.JPG', 
'happy_00036.JPG', 'happy_00033.JPG', 'happy_00050.JPG', 
'happy_00028.JPG', 'happy_00048.JPG', 'happy_00034.JPG', 
'happy_00040.JPG', 'happy_00045.JPG', 'happy_00077.JPG', 
'happy_00064.JPG', 'happy_00052.JPG', 'happy_00071.JPG', 
'happy_00068.JPG', 'happy_00062.JPG', 'happy_00078.JPG', 
'happy_00063.JPG', 'happy_00057.JPG', 'happy_00080.JPG', 
```

```
'happy_00059.JPG', 'happy_00054.JPG', 'happy_00058.JPG',
'happy_00069.JPG', 'happy_00074.JPG', 'happy_00066.JPG',
'happy_00061.JPG', 'happy_00060.JPG', 'happy_00056.JPG',
'happy_00070.JPG', 'happy_00073.JPG', 'happy_00075.JPG',
'happy_00072.JPG', 'happy_00053.JPG', 'happy_00076.JPG',
'happy_00081.JPG', 'happy_00065.JPG', 'happy_00079.JPG',
'happy_00067.JPG', 'happy_00055.JPG', 'happy_00092.JPG',
'happy_00096.JPG', 'happy_00082.JPG', 'happy_00086.JPG',
'happy_00099.JPG', 'neutral_00002.JPG', 'happy_00087.JPG',
'happy_00084.JPG', 'neutral_00003.JPG', 'happy_00083.JPG',
'neutral_00004.JPG', 'neutral_00005.JPG', 'neutral_00001.JPG',
'happy_00097.JPG', 'happy_00091.JPG', 'happy_00095.JPG',
'happy_00094.JPG', 'happy_00090.JPG', 'happy_00093.JPG',
'happy_00085.JPG', 'happy_00088.JPG', 'happy_00089.JPG',
'happy_00098.JPG', 'neutral_00006.JPG', 'happy_00100.JPG',
'neutral_00030.JPG', 'neutral_00019.JPG', 'neutral_00010.JPG',
'neutral_00023.JPG', 'neutral_00025.JPG', 'neutral_00028.JPG',
'neutral_00024.JPG', 'neutral_00013.JPG', 'neutral_00035.JPG',
'neutral_00009.JPG', 'neutral_00031.JPG', 'neutral_00016.JPG',
'neutral_00011.JPG', 'neutral_00034.JPG', 'neutral_00027.JPG',
'neutral_00012.JPG', 'neutral_00033.JPG', 'neutral_00029.JPG',
'neutral_00018.JPG', 'neutral_00032.JPG', 'neutral_00039.JPG',
'neutral_00015.JPG', 'neutral_00008.JPG', 'neutral_00017.JPG',
'neutral_00036.JPG', 'neutral_00037.JPG', 'neutral_00022.JPG',
'neutral_00020.JPG', 'neutral_00040.JPG', 'neutral_00038.JPG',
'neutral_00021.JPG', 'neutral_00014.JPG', 'neutral_00026.JPG',
'neutral_00007.JPG', 'neutral_00051.JPG', 'neutral_00047.JPG',
'neutral_00061.JPG', 'neutral_00049.JPG', 'neutral_00058.JPG',
'neutral_00074.JPG', 'neutral_00063.JPG', 'neutral_00055.JPG',
'neutral_00060.JPG', 'neutral_00059.JPG', 'neutral_00073.JPG',
'neutral_00065.JPG', 'neutral_00069.JPG', 'neutral_00041.JPG',
'neutral_00054.JPG', 'neutral_00064.JPG', 'neutral_00048.JPG',
'neutral_00044.JPG', 'neutral_00043.JPG', 'neutral_00070.JPG',
'neutral_00052.JPG', 'neutral_00050.JPG', 'neutral_00053.JPG',
'neutral_00071.JPG', 'neutral_00045.JPG', 'neutral_00072.JPG',
'neutral_00046.JPG', 'neutral_00057.JPG', 'neutral_00068.JPG',
'neutral_00062.JPG', 'neutral_00056.JPG', 'neutral_00042.JPG',
'neutral_00067.JPG', 'neutral_00066.JPG', 'neutral_00100.JPG',
'sad_00005.JPG', 'sad_00012.JPG', 'sad_00015.JPG',
'neutral_00092.JPG', 'sad_00010.JPG', 'neutral_00085.JPG',
'sad_00013.JPG', 'sad_00011.JPG', 'neutral_00080.JPG',
'neutral_00099.JPG', 'neutral_00094.JPG', 'sad_00008.JPG',
'neutral_00091.JPG', 'neutral_00083.JPG', 'neutral_00096.JPG',
'sad_00007.JPG', 'neutral_00095.JPG', 'neutral_00097.JPG',
'neutral_00086.JPG', 'sad_00001.JPG', 'neutral_00084.JPG',
'neutral_00078.JPG', 'neutral_00081.JPG', 'neutral_00075.JPG',
'neutral_00098.JPG', 'sad_00014.JPG', 'neutral_00087.JPG',
'neutral_00090.JPG', 'sad_00004.JPG', 'neutral_00079.JPG',
'neutral_00077.JPG', 'sad_00002.JPG', 'sad_00003.JPG',
'neutral_00089.JPG', 'neutral_00088.JPG', 'sad_00009.JPG',
```

```
'neutral_00093.JPG', 'sad_00006.JPG', 'neutral_00082.JPG',
'neutral_00076.JPG', 'sad_00019.JPG', 'sad_00024.JPG',
'sad_00020.JPG', 'sad_00029.JPG', 'sad_00017.JPG', 'sad_00018.JPG',
'sad_00026.JPG', 'sad_00021.JPG', 'sad_00022.JPG', 'sad_00027.JPG',
'sad_00028.JPG', 'sad_00030.JPG', 'sad_00023.JPG', 'sad_00031.JPG',
'sad_00025.JPG', 'sad_00016.JPG', 'sad_00044.JPG', 'sad_00041.JPG',
'sad_00054.JPG', 'sad_00040.JPG', 'sad_00036.JPG', 'sad_00047.JPG',
'sad_00049.JPG', 'sad_00051.JPG', 'sad_00043.JPG', 'sad_00048.JPG',
'sad_00056.JPG', 'sad_00034.JPG', 'sad_00035.JPG', 'sad_00033.JPG',
'sad_00032.JPG', 'sad_00042.JPG', 'sad_00037.JPG', 'sad_00039.JPG',
'sad_00052.JPG', 'sad_00045.JPG', 'sad_00038.JPG', 'sad_00053.JPG',
'sad_00046.JPG', 'sad_00050.JPG', 'sad_00055.JPG', 'sad_00088.JPG',
'sad_00057.JPG', 'sad_00077.JPG', 'sad_00089.JPG', 'sad_00070.JPG',
'sad_00060.JPG', 'sad_00073.JPG', 'sad_00064.JPG', 'sad_00087.JPG',
'sad_00084.JPG', 'sad_00061.JPG', 'sad_00072.JPG', 'sad_00086.JPG',
'sad_00076.JPG', 'sad_00069.JPG', 'sad_00080.JPG', 'sad_00078.JPG',
'sad_00081.JPG', 'sad_00085.JPG', 'sad_00071.JPG', 'sad_00062.JPG',
'sad_00065.JPG', 'sad_00067.JPG', 'sad_00059.JPG', 'sad_00074.JPG',
'sad_00083.JPG', 'sad_00066.JPG', 'sad_00082.JPG', 'sad_00058.JPG',
'sad_00063.JPG', 'sad_00068.JPG', 'sad_00075.JPG', 'sad_00079.JPG',
'surprised_00003.JPG', 'sad_00093.JPG', 'surprised_00021.JPG',
'surprised_00019.JPG', 'sad_00095.JPG', 'sad_00098.JPG',
'sad_00092.JPG', 'sad_00094.JPG', 'sad_00097.JPG',
'surprised_00005.JPG', 'surprised_00008.JPG', 'sad_00099.JPG',
'surprised_00017.JPG', 'surprised_00022.JPG', 'sad_00091.JPG',
'surprised_00001.JPG', 'surprised_00007.JPG', 'surprised_00002.JPG',
'surprised_00016.JPG', 'surprised_00011.JPG', 'sad_00096.JPG',
'surprised_00009.JPG', 'sad_00090.JPG', 'surprised_00020.JPG',
'surprised_00013.JPG', 'surprised_00015.JPG', 'surprised_00014.JPG',
'sad_00100.JPG', 'surprised_00018.JPG', 'surprised_00012.JPG',
'surprised_00006.JPG', 'surprised_00004.JPG', 'surprised_00010.JPG',
'surprised_00043.JPG', 'surprised_00025.JPG', 'surprised_00050.JPG',
'surprised_00063.JPG', 'surprised_00039.JPG', 'surprised_00042.JPG',
'surprised_00053.JPG', 'surprised_00065.JPG', 'surprised_00028.JPG',
'surprised_00056.JPG', 'surprised_00046.JPG', 'surprised_00057.JPG',
'surprised_00040.JPG', 'surprised_00047.JPG', 'surprised_00037.JPG',
'surprised_00029.JPG', 'surprised_00054.JPG', 'surprised_00031.JPG',
'surprised_00044.JPG', 'surprised_00030.JPG', 'surprised_00068.JPG',
'surprised_00041.JPG', 'surprised_00035.JPG', 'surprised_00049.JPG',
'surprised_00048.JPG', 'surprised_00062.JPG', 'surprised_00051.JPG',
'surprised_00067.JPG', 'surprised_00027.JPG', 'surprised_00052.JPG',
'surprised_00069.JPG', 'surprised_00058.JPG', 'surprised_00064.JPG',
'surprised_00055.JPG', 'surprised_00032.JPG', 'surprised_00061.JPG',
'surprised_00033.JPG', 'surprised_00066.JPG', 'surprised_00034.JPG',
'surprised_00023.JPG', 'surprised_00038.JPG', 'surprised_00026.JPG',
'surprised_00060.JPG', 'surprised_00024.JPG', 'surprised_00045.JPG',
'surprised_00036.JPG', 'surprised_00059.JPG', 'surprised_00071.JPG',
'surprised_00093.JPG', 'surprised_00097.JPG', 'surprised_00082.JPG',
'surprised_00098.JPG', 'surprised_00089.JPG', 'surprised_00079.JPG',
'surprised_00070.JPG', 'surprised_00088.JPG', 'surprised_00084.JPG',
```

```
'surprised_00080.JPG', 'surprised_00075.JPG', 'surprised_00073.JPG',
'surprised_00085.JPG', 'surprised_00091.JPG', 'surprised_00086.JPG',
'surprised_00095.JPG', 'surprised_00090.JPG', 'surprised_00076.JPG',
'surprised_00072.JPG', 'surprised_00077.JPG', 'surprised_00092.JPG',
'surprised_00094.JPG', 'surprised_00087.JPG', 'surprised_00096.JPG',
'surprised_00074.JPG', 'surprised_00083.JPG', 'surprised_00078.JPG',
'surprised_00081.JPG', 'surprised_00099.JPG', 'surprised_00100.JPG']
```

Step 5 : Check number of resized images for use as input

```
im1 = array(Image.open(path2 + '/'+ imlist[0]))
# open one image to get size
m,n = im1.shape[0:2] # get the size of the images
imnbr = len(imlist) # get the number of images

print(imnbr)
print ("Total number of processed images is {}".format(imnbr))
```

```
400
Total number of processed images is 400
```

Step 6: Apply one hot encoding and generate label the images

```
# create matrix to store all flattened images
immatrix = array([array(Image.open(path2+ '/' + im2)).flatten() for
im2 in imlist],'f')

print("Matrix shape is {}".format(immatrix.shape))
print(immatrix)

label=np.ones((num_samples,),dtype = int)
label[0:100]=0
label[100:200]=1
label[200:300]=2
label[300:]=3

print("Label shape is {}".format(label.shape))
print(label)
```

```
Matrix shape is (400, 40000)
[[ 29.  73. 142. ...  25.  67.  93.]
 [240. 232. 218. ...   5.   6.   6.]
 [  0.   0.   0. ...  27.  30.  32.]
 ...
 [131. 131. 131. ... 140. 144. 147.]
 [247. 236. 224. ...  53.  76.  87.]
 [237. 212. 168. ... 246. 227. 213.]]
Label shape is (400,)
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

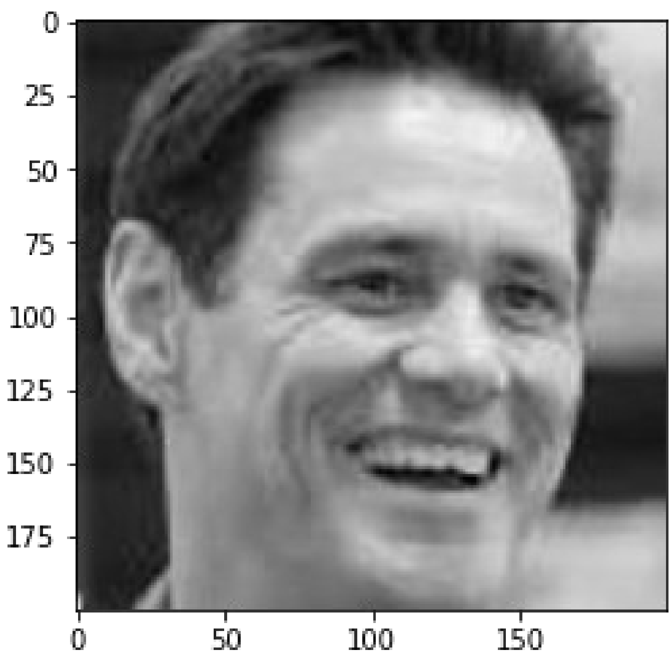
```
0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2
 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3]
```

Step 7: Attach labels to data matrix

```
data,Label = shuffle(immatrix,label, random_state=2)
train_data = [data,Label]

img=immatrix[67].reshape(img_rows,img_cols)
plt.imshow(img)
plt.imshow(img,cmap='gray')
print (train_data)
```

```
[array([[  7.,  14.,  23., ..., 204., 205., 205.],
       [140., 140., 140., ...,   7.,   7.,   7.],
       [  2.,   1.,   0., ..., 230., 231., 231.],
       ...,
       [130., 127., 126., ...,  85.,  85.,  85.],
       [105., 107., 110., ..., 198., 194., 192.],
       [ 82.,  82.,  82., ...,  60.,  60.,  60.]], dtype=float32),
array([0, 0, 2, 1, 3, 0, 0, 1, 3, 2, 1, 0, 1, 2, 1, 1, 1, 3, 3, 0, 2,
3,
       0, 3, 0, 3, 0, 2, 3, 1, 0, 3, 0, 2, 0, 2, 3, 0, 0, 3, 2, 3, 1,
1,
       1, 3, 1, 3, 3, 1, 1, 2, 1, 3, 1, 0, 1, 2, 2, 0, 1, 3, 2, 2, 1,
2,
       0, 3, 0, 2, 2, 2, 0, 1, 2, 1, 2, 0, 2, 1, 2, 3, 0, 2, 1, 3, 2,
1,
       3, 2, 2, 0, 2, 3, 1, 1, 0, 0, 1, 3, 2, 2, 0, 1, 2, 1, 1, 1, 1,
2,
       3, 1, 2, 0, 2, 1, 0, 3, 3, 0, 1, 2, 2, 3, 3, 0, 1, 3, 2, 1, 2,
0,
       1, 3, 3, 3, 0, 2, 3, 0, 2, 1, 1, 2, 1, 1, 2, 2, 3, 3, 0, 1, 0,
2,
```

```
        3, 0, 1, 2, 0, 1, 0, 1, 2, 3, 0, 0, 1, 2, 2, 1, 0, 2, 1, 2, 1,
1,
        0, 2, 3, 1, 2, 0, 3, 0, 3, 3, 0, 0, 0, 2, 2, 0, 3, 3, 3, 1, 2,
1,
        3, 1, 3, 1, 3, 3, 1, 2, 0, 1, 1, 1, 0, 1, 2, 1, 0, 3, 1, 0, 3,
1,
        1, 2, 2, 1, 2, 3, 3, 1, 2, 3, 0, 3, 0, 3, 2, 2, 3, 1, 3, 0, 0,
3,
        0, 3, 2, 3, 3, 0, 0, 0, 3, 3, 1, 0, 1, 1, 1, 2, 2, 2, 1, 2, 1,
3,
        3, 1, 0, 3, 3, 3, 1, 2, 0, 3, 2, 3, 3, 0, 0, 3, 3, 3, 2, 3, 2,
1,
        2, 3, 1, 3, 0, 0, 2, 0, 0, 0, 3, 2, 1, 1, 1, 2, 3, 3, 1, 1, 2,
0,
        2, 2, 3, 1, 1, 3, 0, 2, 3, 2, 3, 3, 0, 1, 0, 0, 3, 3, 0, 0, 0,
3,
        2, 1, 3, 0, 2, 1, 2, 0, 0, 2, 0, 0, 0, 2, 1, 0, 2, 2, 2, 2, 1,
2,
        0, 2, 3, 3, 1, 0, 2, 0, 0, 0, 1, 2, 3, 3, 1, 0, 3, 3, 2, 0, 0,
0,
        3, 0, 3, 2, 1, 1, 1, 2, 1, 1, 2, 2, 3, 0, 2, 3, 0, 1, 2, 3, 0,
2,
        0, 0, 0, 1]))]
```



Step 8: Generate training and testing data

```
(X, y) = (train_data[0],train_data[1])
print(X)
print(y)
X_train, X_test, y_train, y_test = train_test_split(X, y,
```

```python
test_size=0.2, random_state=4)

print('X_train :', X_train)
print('y_train :', y_train)
print('X_test :', X_test)
print('y_test :', y_test)

X_train = X_train.reshape(X_train.shape[0], img_rows, img_cols, 1)
X_test = X_test.reshape(X_test.shape[0], img_rows, img_cols, 1)

X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

X_train /= 255
X_test /= 255

print('X_train shape:', X_train.shape)
print('y_train shape:', y_train.shape)
print('X_test shape:', X_test.shape)
print('y_test shape:', y_test.shape)
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')
```

```
[[  7.  14.  23. ... 204. 205. 205.]
 [140. 140. 140. ...   7.   7.   7.]
 [  2.   1.   0. ... 230. 231. 231.]
 ...
 [130. 127. 126. ...  85.  85.  85.]
 [105. 107. 110. ... 198. 194. 192.]
 [ 82.  82.  82. ...  60.  60.  60.]]
[0 0 2 1 3 0 0 1 3 2 1 0 1 2 1 1 1 3 3 0 2 3 0 3 0 3 0 2 3 1 0 3 0 2 0
2 3
 0 0 3 2 3 1 1 1 3 1 3 3 1 1 2 1 3 1 0 1 2 2 0 1 3 2 2 1 2 0 3 0 2 2 2
0 1
 2 1 2 0 2 1 2 3 0 2 1 3 2 1 3 2 2 0 2 3 1 1 0 0 1 3 2 2 0 1 2 1 1 1 1
2 3
 1 2 0 2 1 0 3 3 0 1 2 2 3 3 0 1 3 2 1 2 0 1 3 3 3 0 2 3 0 2 1 1 2 1 1
2 2
 3 3 0 1 0 2 3 0 1 2 0 1 0 1 2 3 0 0 1 2 2 1 0 2 1 2 1 1 0 2 3 1 2 0 3
0 3
 3 0 0 0 2 2 0 3 3 3 1 2 1 3 1 3 1 3 3 1 2 0 1 1 1 0 1 2 1 0 3 1 0 3 1
1 2
 2 1 2 3 3 1 2 3 0 3 0 3 2 2 3 1 3 0 0 3 0 3 2 3 3 0 0 0 3 3 1 0 1 1 1
2 2
 2 1 2 1 3 3 1 0 3 3 3 1 2 0 3 2 3 3 0 0 3 3 3 2 3 2 1 2 3 1 3 0 0 2 0
0 0
 3 2 1 1 1 2 3 3 1 1 2 0 2 2 3 1 1 3 0 2 3 2 3 3 0 1 0 0 3 3 0 0 0 3 2
1 3
 0 2 1 2 0 0 2 0 0 0 2 1 0 2 2 2 2 1 2 0 2 3 3 1 0 2 0 0 0 1 2 3 3 1 0
3 3
```

```
 2 0 0 0 3 0 3 2 1 1 1 2 1 1 2 2 3 0 2 3 0 1 2 3 0 2 0 0 0 1]
X_train : [[120. 120. 120. ...  40.  39.  37.]
 [162. 163. 162. ... 178. 179. 180.]
 [254. 254. 254. ... 226. 226. 226.]
 ...
 [141. 135. 100. ... 249. 242. 247.]
 [106. 107. 108. ...  55.  59.  61.]
 [196. 196. 196. ...  25.  18.  14.]]
y_train : [2 1 2 2 0 1 0 3 3 0 0 1 0 1 1 2 2 1 3 3 1 0 1 2 1 0 3 3 2 1
1 2 1 3 0 2 0
 3 2 3 2 1 2 2 0 3 0 2 0 2 0 1 1 1 3 0 2 0 3 3 3 2 1 1 3 0 2 1 1 2 0 3
1 2
 3 0 3 3 3 0 2 2 0 0 2 3 1 3 0 2 0 0 2 0 2 2 2 0 0 0 2 0 3 3 0 0 0 0 3
2 1
 1 2 3 3 2 0 0 3 3 1 2 1 2 2 0 3 0 2 1 1 1 1 3 1 0 0 3 3 2 0 3 3 3 2 2
1 3
 0 2 2 0 3 3 3 3 2 2 1 0 3 0 0 0 2 0 2 3 1 0 2 3 1 3 2 0 3 2 0 0 2 1 3
2 3
 1 0 0 1 1 3 3 2 0 2 2 2 3 1 1 1 1 3 1 2 3 0 0 3 1 1 0 1 0 1 1 1 3 2 2
2 3
 2 3 0 2 3 2 1 2 3 1 0 2 2 2 3 0 2 2 3 3 3 1 0 0 1 3 0 3 2 1 0 2 3 3 0
1 1
 3 3 3 3 1 0 2 0 2 2 1 0 0 1 0 0 1 1 1 1 1 1 1 2 1 1 3 2 2 0 1 0 0 0 3
1 1
 3 3 0 0 1 1 0 1 0 2 3 1 1 2 0 1 2 2 1 0 2 1 1 2]
X_test : [[236. 236. 236. ... 199. 202. 203.]
 [ 58.  55.  50. ... 194. 194. 193.]
 [246. 246. 246. ... 116. 216. 255.]
 ...
 [213. 216. 224. ...  94. 114. 122.]
 [231. 231. 231. ...  65.  66.  67.]
 [ 91.  87.  87. ...  27.  22.  19.]]
y_test : [3 0 3 1 2 2 0 0 0 2 3 3 2 3 3 2 0 2 0 0 1 3 1 2 1 0 2 2 0 2
0 0 3 2 2 1 1
 2 2 2 3 1 0 3 3 3 3 3 2 3 2 1 3 1 1 0 1 1 0 1 2 0 1 1 0 3 0 0 3 3 1 1
1 3
 1 3 2 3 3 3]
X_train shape: (320, 200, 200, 1)
y_train shape: (320,)
X_test shape: (80, 200, 200, 1)
y_test shape: (80,)
320 train samples
80 test samples
```

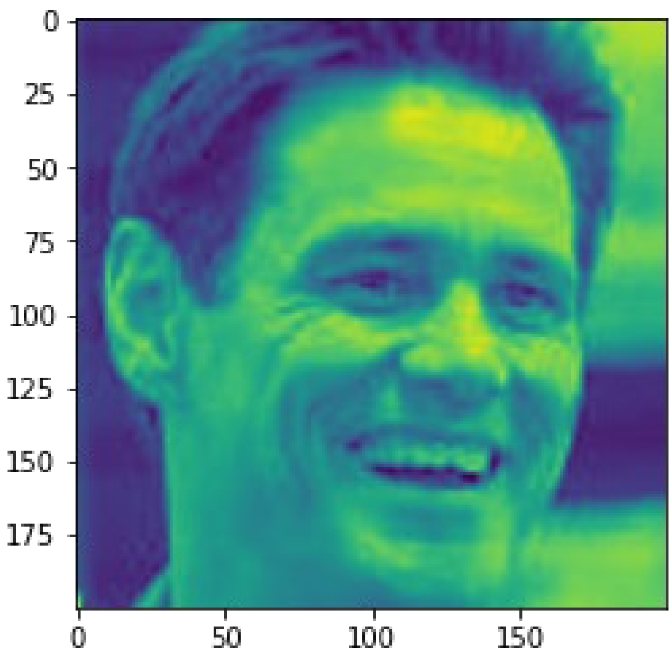
Step 9: Convert output to categorical form and test input data

```
print(y_train)
print(y_test)
nb_classes=4
# convert class vectors to binary class matrices
Y_train = np_utils.to_categorical(y_train, nb_classes)
```

```
Y_test = np_utils.to_categorical(y_test, nb_classes)

print(Y_train)
print(Y_test)


i = 10
img=immatrix[67].reshape(img_rows,img_cols)
plt.imshow(img)
print("label of this image is: ", Y_train[i,:])

[2 1 2 2 0 1 0 3 3 0 0 1 0 1 1 2 2 1 3 3 1 0 1 2 1 0 3 3 2 1 1 2 1 3 0
2 0
 3 2 3 2 1 2 2 0 3 0 2 0 2 0 1 1 1 3 0 2 0 3 3 3 2 1 1 3 0 2 1 1 2 0 3
1 2
 3 0 3 3 3 0 2 2 0 0 2 3 1 3 0 2 0 0 2 0 2 2 2 0 0 0 2 0 3 3 0 0 0 0 3
2 1
 1 2 3 3 2 0 0 3 3 1 2 1 2 2 0 3 0 2 1 1 1 1 3 1 0 0 3 3 2 0 3 3 3 2 2
1 3
 0 2 2 0 3 3 3 3 2 2 1 0 3 0 0 0 2 0 2 3 1 0 2 3 1 3 2 0 3 2 0 0 2 1 3
2 3
 1 0 0 1 1 3 3 2 0 2 2 2 3 1 1 1 1 3 1 2 3 0 0 3 1 1 0 1 0 1 1 1 3 2 2
2 3
 2 3 0 2 3 2 1 2 3 1 0 2 2 2 3 0 2 2 3 3 3 1 0 0 1 3 0 3 2 1 0 2 3 3 0
1 1
 3 3 3 3 1 0 2 0 2 2 1 0 0 1 0 0 1 1 1 1 1 1 1 2 1 1 3 2 2 0 1 0 0 0 3
1 1
 3 3 0 0 1 1 0 1 0 2 3 1 1 2 0 1 2 2 1 0 2 1 1 2]
[3 0 3 1 2 2 0 0 0 2 3 3 2 3 3 2 0 2 0 0 1 3 1 2 1 0 2 2 0 2 0 0 3 2 2
1 1
 2 2 2 3 1 0 3 3 3 3 3 2 3 2 1 3 1 1 0 1 1 0 1 2 0 1 1 0 3 0 0 3 3 1 1
1 3
 1 3 2 3 3 3]
[[0. 0. 1. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 ...
 [0. 1. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]]
[[0. 0. 0. 1.]
 [1. 0. 0. 0.]
 [0. 0. 0. 1.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 1. 0.]
 [1. 0. 0. 0.]
 [1. 0. 0. 0.]
 [1. 0. 0. 0.]
 [0. 0. 1. 0.]
```

```
[0. 0. 0. 1.]
[0. 0. 0. 1.]
[0. 0. 1. 0.]
[0. 0. 0. 1.]
[0. 0. 0. 1.]
[0. 0. 1. 0.]
[1. 0. 0. 0.]
[0. 0. 1. 0.]
[1. 0. 0. 0.]
[1. 0. 0. 0.]
[0. 1. 0. 0.]
[0. 0. 0. 1.]
[0. 1. 0. 0.]
[0. 0. 1. 0.]
[0. 1. 0. 0.]
[1. 0. 0. 0.]
[0. 0. 1. 0.]
[0. 0. 1. 0.]
[1. 0. 0. 0.]
[0. 0. 1. 0.]
[1. 0. 0. 0.]
[1. 0. 0. 0.]
[0. 0. 0. 1.]
[0. 0. 1. 0.]
[0. 0. 1. 0.]
[0. 1. 0. 0.]
[0. 1. 0. 0.]
[0. 0. 1. 0.]
[0. 0. 1. 0.]
[0. 0. 1. 0.]
[0. 0. 0. 1.]
[0. 1. 0. 0.]
[1. 0. 0. 0.]
[0. 0. 0. 1.]
[0. 0. 0. 1.]
[0. 0. 0. 1.]
[0. 0. 0. 1.]
[0. 0. 0. 1.]
[0. 0. 1. 0.]
[0. 0. 0. 1.]
[0. 0. 1. 0.]
[0. 1. 0. 0.]
[0. 0. 0. 1.]
[0. 1. 0. 0.]
[0. 1. 0. 0.]
[1. 0. 0. 0.]
[0. 1. 0. 0.]
[0. 1. 0. 0.]
[1. 0. 0. 0.]
[0. 1. 0. 0.]
```

```
 [0. 0. 1. 0.]
 [1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 1. 0. 0.]
 [1. 0. 0. 0.]
 [0. 0. 0. 1.]
 [1. 0. 0. 0.]
 [1. 0. 0. 0.]
 [0. 0. 0. 1.]
 [0. 0. 0. 1.]
 [0. 1. 0. 0.]
 [0. 1. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 0. 1.]
 [0. 1. 0. 0.]
 [0. 0. 0. 1.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]
 [0. 0. 0. 1.]
 [0. 0. 0. 1.]]
label of this image is:  [1. 0. 0. 0.]
```



Step 10.1 Using Feedforward NN model

```
model = tf.keras.Sequential()

layer_0 = tf.keras.layers.Flatten(input_shape=(1, img_rows, img_cols))
layer_1 = tf.keras.layers.Dense(units=128, activation="relu")
layer_2 = tf.keras.layers.Dense(units=128, activation="relu")
layer_3 = tf.keras.layers.Dense(units=128, activation="relu")
```

```
layer_4 = tf.keras.layers.Dense(units=10, activation="softmax")

model.add(layer_0)
model.add(layer_1)
model.add(layer_2)
model.add(layer_3)
model.add(layer_4)
model.summary()
model.compile(optimizer=keras.optimizers.Adadelta(learning_rate=0.0001
),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Step 10.2: Using Convolutional Neural Network model

```
# number of convolutional filters to use
nb_filters = 32
# size of pooling area for max pooling
nb_pool = 2
# convolution kernel size
nb_conv = 3

#batch_size to train
batch_size = 32
# number of output classes
nb_classes = 4
# number of epochs to train
nb_epoch = 20

model = Sequential()

model.add(Convolution2D(nb_filters, nb_conv, nb_conv, padding='valid',
input_shape=(img_rows, img_cols, 1))) #, activation = 'relu',
data_format='channels_first'))
convout1 = Activation('relu')
model.add(convout1)
model.add(Convolution2D(nb_filters, nb_conv, nb_conv))
convout2 = Activation('relu')
model.add(convout2)
model.add(MaxPooling2D(pool_size=(nb_pool, nb_pool)))
model.add(Dropout(0.5))

model.add(Flatten())
model.add(Dense(128))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(nb_classes))
model.add(Activation('softmax'))
model.compile(optimizer=keras.optimizers.Adadelta(learning_rate=0.0001
),
```

```
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Step 12: Train the model and gather training history data

```
hist = model.fit(X_train, Y_train, batch_size=batch_size,
epochs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
Epoch 1/20
10/10 [==============================] - 2s 128ms/step - loss: 1.3989
- accuracy: 0.2219 - val_loss: 1.3928 - val_accuracy: 0.2125
Epoch 2/20
10/10 [==============================] - 1s 104ms/step - loss: 1.3969
- accuracy: 0.2375 - val_loss: 1.3928 - val_accuracy: 0.2125
Epoch 3/20
10/10 [==============================] - 1s 103ms/step - loss: 1.3917
- accuracy: 0.2469 - val_loss: 1.3928 - val_accuracy: 0.2125
Epoch 4/20
10/10 [==============================] - 1s 103ms/step - loss: 1.3995
- accuracy: 0.2406 - val_loss: 1.3928 - val_accuracy: 0.2125
Epoch 5/20
10/10 [==============================] - 1s 105ms/step - loss: 1.3927
- accuracy: 0.2688 - val_loss: 1.3928 - val_accuracy: 0.2125
Epoch 6/20
10/10 [==============================] - 1s 104ms/step - loss: 1.3988
- accuracy: 0.2375 - val_loss: 1.3928 - val_accuracy: 0.2125
Epoch 7/20
10/10 [==============================] - 1s 103ms/step - loss: 1.3847
- accuracy: 0.2781 - val_loss: 1.3928 - val_accuracy: 0.2125
Epoch 8/20
10/10 [==============================] - 1s 107ms/step - loss: 1.4015
- accuracy: 0.2531 - val_loss: 1.3928 - val_accuracy: 0.2125
Epoch 9/20
10/10 [==============================] - 1s 104ms/step - loss: 1.3941
- accuracy: 0.2500 - val_loss: 1.3927 - val_accuracy: 0.2125
Epoch 10/20
10/10 [==============================] - 1s 103ms/step - loss: 1.3930
- accuracy: 0.2719 - val_loss: 1.3927 - val_accuracy: 0.2125
Epoch 11/20
10/10 [==============================] - 1s 104ms/step - loss: 1.3941
- accuracy: 0.2375 - val_loss: 1.3927 - val_accuracy: 0.2125
Epoch 12/20
10/10 [==============================] - 1s 104ms/step - loss: 1.3936
- accuracy: 0.2281 - val_loss: 1.3927 - val_accuracy: 0.2125
Epoch 13/20
10/10 [==============================] - 1s 103ms/step - loss: 1.3812
- accuracy: 0.2781 - val_loss: 1.3927 - val_accuracy: 0.2125
Epoch 14/20
10/10 [==============================] - 1s 103ms/step - loss: 1.3828
- accuracy: 0.2969 - val_loss: 1.3927 - val_accuracy: 0.2125
Epoch 15/20
```

```
10/10 [==============================] - 1s 103ms/step - loss: 1.3860
- accuracy: 0.3187 - val_loss: 1.3927 - val_accuracy: 0.2125
Epoch 16/20
10/10 [==============================] - 1s 104ms/step - loss: 1.3986
- accuracy: 0.2438 - val_loss: 1.3927 - val_accuracy: 0.2125
Epoch 17/20
10/10 [==============================] - 1s 104ms/step - loss: 1.3908
- accuracy: 0.2875 - val_loss: 1.3927 - val_accuracy: 0.2125
Epoch 18/20
10/10 [==============================] - 1s 104ms/step - loss: 1.4013
- accuracy: 0.2562 - val_loss: 1.3927 - val_accuracy: 0.2125
Epoch 19/20
10/10 [==============================] - 1s 102ms/step - loss: 1.4010
- accuracy: 0.2094 - val_loss: 1.3927 - val_accuracy: 0.2125
Epoch 20/20
10/10 [==============================] - 1s 103ms/step - loss: 1.3941
- accuracy: 0.2844 - val_loss: 1.3927 - val_accuracy: 0.2125
```

Step 13: Display results of performance assessment metrics

```
score = model.evaluate(X_test, Y_test, verbose=0)
print(model.metrics_names)
print(score)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

#print(Y_test)
y_pred = model.predict(X_test)
print(Y_pred)
#y_pred = np.argmax(Y_pred, axis=1)
print(y_pred)
```

```
['loss', 'accuracy']
[1.392669677734375, 0.21250000596046448]
Test loss: 1.392669677734375
Test accuracy: 0.21250000596046448
[[0.26892036 0.2590206  0.2386332  0.23342583]
 [0.26774833 0.24728121 0.2444792  0.24049123]
 [0.26880264 0.26598272 0.23566483 0.2295498 ]
 [0.26344118 0.2543834  0.24343583 0.2387396 ]
 [0.2604505  0.24889766 0.24527867 0.24537316]
 [0.25302827 0.24874498 0.2497913  0.24843553]
 [0.25988567 0.25238517 0.23616162 0.2515675 ]
 [0.26031256 0.25251025 0.24695686 0.24022032]
 [0.26700738 0.24961326 0.24468544 0.23869392]
 [0.26610038 0.2576046  0.24451956 0.23177542]
 [0.25900164 0.25245756 0.24242896 0.2461119 ]
 [0.25658327 0.25427067 0.24719785 0.2419482 ]
 [0.2808625  0.25049707 0.23502363 0.23361683]
 [0.2721354  0.26834476 0.2348994  0.22462043]
 [0.2607615  0.25166377 0.24435477 0.2432199 ]
```

```
[0.2659855  0.25074223 0.23999962 0.24327262]
[0.2576252  0.253557   0.2503136  0.23850416]
[0.2581709  0.24860786 0.25050768 0.24271354]
[0.26207337 0.2558005  0.2433112  0.23881502]
[0.27075776 0.24872302 0.23521614 0.24530306]
[0.27092424 0.25741506 0.23953494 0.23212579]
[0.27133527 0.25398302 0.24338107 0.2313007 ]
[0.26705417 0.2605341  0.24365492 0.22875682]
[0.26297104 0.2510649  0.23794597 0.24801812]
[0.26585534 0.24928106 0.24366187 0.24120173]
[0.26125708 0.2548123  0.24364205 0.24028859]
[0.27111024 0.2563407  0.23717889 0.23537017]
[0.268959   0.2534303  0.24286495 0.23474574]
[0.27631152 0.25969765 0.23891282 0.22507794]
[0.26855356 0.25665748 0.2469694  0.22781949]
[0.26647955 0.25277933 0.24242027 0.2383208 ]
[0.26567188 0.26746094 0.2429748  0.22389233]
[0.269087   0.25066933 0.24278964 0.23745407]
[0.26007912 0.2523349  0.24355622 0.2440298 ]
[0.26399827 0.25587812 0.24439919 0.23572446]
[0.2592145  0.25801638 0.2508627  0.23190643]
[0.27025014 0.24929056 0.24329498 0.23716426]
[0.25130746 0.25470856 0.24918899 0.24479501]
[0.27245677 0.25639793 0.23687439 0.23427087]
[0.26465315 0.2535436  0.24717008 0.23463313]
[0.26610807 0.2502349  0.24278703 0.24087003]
[0.27090442 0.2554239  0.23729274 0.23637897]
[0.26618463 0.25301123 0.24803273 0.23277147]
[0.2585817  0.26155674 0.24798414 0.23187746]
[0.26628286 0.25382012 0.24052086 0.23937608]
[0.27226302 0.2477012  0.23928615 0.24074963]
[0.26746613 0.25377065 0.24677399 0.2319892 ]
[0.26424813 0.25100127 0.24447133 0.2402792 ]
[0.2665034  0.25408697 0.24347277 0.23593687]
[0.2616545  0.25401813 0.2421642  0.24216323]
[0.26712275 0.2530072  0.24533871 0.23453134]
[0.26226446 0.25666314 0.24651806 0.23455438]
[0.26469886 0.25387996 0.24735667 0.23406449]
[0.2689511  0.2593136  0.24355529 0.22818005]
[0.2721393  0.2534322  0.23701903 0.2374094 ]
[0.25732055 0.2556812  0.2523384  0.23465979]
[0.26461744 0.2546788  0.24012065 0.2405831 ]
[0.26976278 0.25655103 0.23546255 0.23822358]
[0.26373905 0.25878116 0.24893086 0.22854882]
[0.26488447 0.25508448 0.24309714 0.236934  ]
[0.26546448 0.25284258 0.24411432 0.23757863]
[0.2699552  0.25157347 0.24013487 0.23833646]
[0.26313508 0.25469044 0.24719699 0.23497744]
[0.26349565 0.25807482 0.24813953 0.23029   ]
[0.27150863 0.25568092 0.2461656  0.22664487]
```

```
 [0.26785284 0.2610592  0.24888304 0.22220494]
 [0.26501623 0.2562681  0.24238749 0.23632818]
 [0.2719258  0.25605246 0.23957692 0.23244475]
 [0.2558977  0.2553462  0.2512201  0.2375359 ]
 [0.26301408 0.25719154 0.24357992 0.2362145 ]
 [0.25819507 0.25350025 0.25072733 0.23757741]
 [0.2716642  0.25735906 0.24009952 0.23087727]
 [0.2676851  0.26116684 0.23943223 0.2317158 ]
 [0.26974303 0.25427127 0.24722297 0.22876275]
 [0.26613623 0.2523351  0.25176975 0.22975896]
 [0.26225597 0.25543183 0.24559878 0.23671345]
 [0.27004886 0.25410128 0.24184698 0.23400286]
 [0.2717629  0.26528376 0.24008569 0.22286764]
 [0.26491317 0.27621126 0.229694   0.22918159]
 [0.26789367 0.26251242 0.23660529 0.23298863]]
[[0.26892036 0.2590206  0.2386332  0.23342583]
 [0.26774833 0.24728121 0.2444792  0.24049123]
 [0.26880264 0.26598272 0.23566483 0.2295498 ]
 [0.26344118 0.2543834  0.24343583 0.2387396 ]
 [0.2604505  0.24889766 0.24527867 0.24537316]
 [0.25302827 0.24874498 0.2497913  0.24843553]
 [0.25988567 0.25238517 0.23616162 0.2515675 ]
 [0.26031256 0.25251025 0.24695686 0.24022032]
 [0.26700738 0.24961326 0.24468544 0.23869392]
 [0.26610038 0.2576046  0.24451956 0.23177542]
 [0.25900164 0.25245756 0.24242896 0.2461119 ]
 [0.25658327 0.25427067 0.24719785 0.2419482 ]
 [0.2808625  0.25049707 0.23502363 0.23361683]
 [0.2721354  0.26834476 0.2348994  0.22462043]
 [0.2607615  0.25166377 0.24435477 0.2432199 ]
 [0.2659855  0.25074223 0.23999962 0.24327262]
 [0.2576252  0.253557   0.2503136  0.23850416]
 [0.2581709  0.24860786 0.25050768 0.24271354]
 [0.26207337 0.2558005  0.2433112  0.23881502]
 [0.27075776 0.24872302 0.23521614 0.24530306]
 [0.27092424 0.25741506 0.23953494 0.23212579]
 [0.27133527 0.25398302 0.24338107 0.2313007 ]
 [0.26705417 0.2605341  0.24365492 0.22875682]
 [0.26297104 0.2510649  0.23794597 0.24801812]
 [0.26585534 0.24928106 0.24366187 0.24120173]
 [0.26125708 0.2548123  0.24364205 0.24028859]
 [0.27111024 0.2563407  0.23717889 0.23537017]
 [0.268959   0.2534303  0.24286495 0.23474574]
 [0.27631152 0.25969765 0.23891282 0.22507794]
 [0.26855356 0.25665748 0.2469694  0.22781949]
 [0.26647955 0.25277933 0.24242027 0.2383208 ]
 [0.26567188 0.26746094 0.2429748  0.22389233]
 [0.269087   0.25066933 0.24278964 0.23745407]
 [0.26007912 0.2523349  0.24355622 0.2440298 ]
 [0.26399827 0.25587812 0.24439919 0.23572446]
```

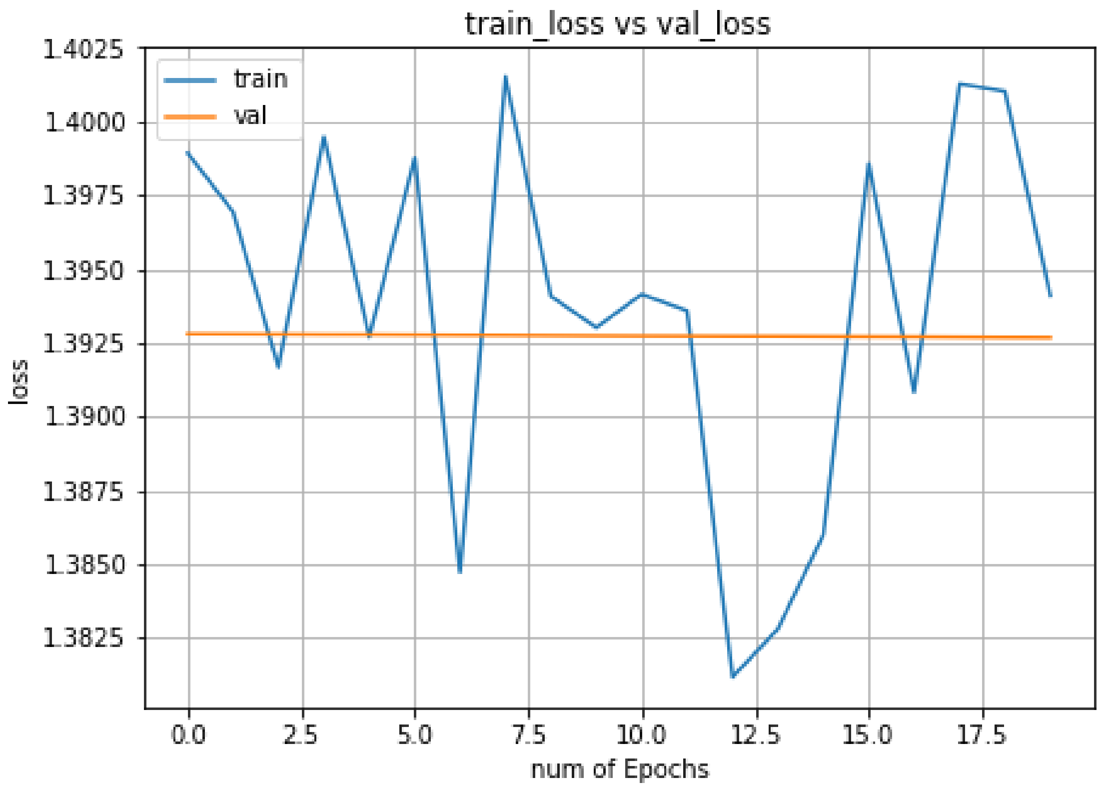
```
 [0.2592145  0.25801638 0.2508627  0.23190643]
 [0.27025014 0.24929056 0.24329498 0.23716426]
 [0.25130746 0.25470856 0.24918899 0.24479501]
 [0.27245677 0.25639793 0.23687439 0.23427087]
 [0.26465315 0.2535436  0.24717008 0.23463313]
 [0.26610807 0.2502349  0.24278703 0.24087003]
 [0.27090442 0.2554239  0.23729274 0.23637897]
 [0.26618463 0.25301123 0.24803273 0.23277147]
 [0.2585817  0.26155674 0.24798414 0.23187746]
 [0.26628286 0.25382012 0.24052086 0.23937608]
 [0.27226302 0.2477012  0.23928615 0.24074963]
 [0.26746613 0.25377065 0.24677399 0.2319892 ]
 [0.26424813 0.25100127 0.24447133 0.2402792 ]
 [0.2665034  0.25408697 0.24347277 0.23593687]
 [0.2616545  0.25401813 0.2421642  0.24216323]
 [0.26712275 0.2530072  0.24533871 0.23453134]
 [0.26226446 0.25666314 0.24651806 0.23455438]
 [0.26469886 0.25387996 0.24735667 0.23406449]
 [0.2689511  0.2593136  0.24355529 0.22818005]
 [0.2721393  0.2534322  0.23701903 0.2374094 ]
 [0.25732055 0.2556812  0.2523384  0.23465979]
 [0.26461744 0.2546788  0.24012065 0.2405831 ]
 [0.26976278 0.25655103 0.23546255 0.23822358]
 [0.26373905 0.25878116 0.24893086 0.22854882]
 [0.26488447 0.25508448 0.24309714 0.236934  ]
 [0.26546448 0.25284258 0.24411432 0.23757863]
 [0.2699552  0.25157347 0.24013487 0.23833646]
 [0.26313508 0.25469044 0.24719699 0.23497744]
 [0.26349565 0.25807482 0.24813953 0.23029   ]
 [0.27150863 0.25568092 0.2461656  0.22664487]
 [0.26785284 0.2610592  0.24888304 0.22220494]
 [0.26501623 0.2562681  0.24238749 0.23632818]
 [0.2719258  0.25605246 0.23957692 0.23244475]
 [0.2558977  0.2553462  0.2512201  0.2375359 ]
 [0.26301408 0.25719154 0.24357992 0.2362145 ]
 [0.25819507 0.25350025 0.25072733 0.23757741]
 [0.2716642  0.25735906 0.24009952 0.23087727]
 [0.2676851  0.26116684 0.23943223 0.2317158 ]
 [0.26974303 0.25427127 0.24722297 0.22876275]
 [0.26613623 0.2523351  0.25176975 0.22975896]
 [0.26225597 0.25543183 0.24559878 0.23671345]
 [0.27004886 0.25410128 0.24184698 0.23400286]
 [0.2717629  0.26528376 0.24008569 0.22286764]
 [0.26491317 0.27621126 0.229694   0.22918159]
 [0.26789367 0.26251242 0.23660529 0.23298863]]
```

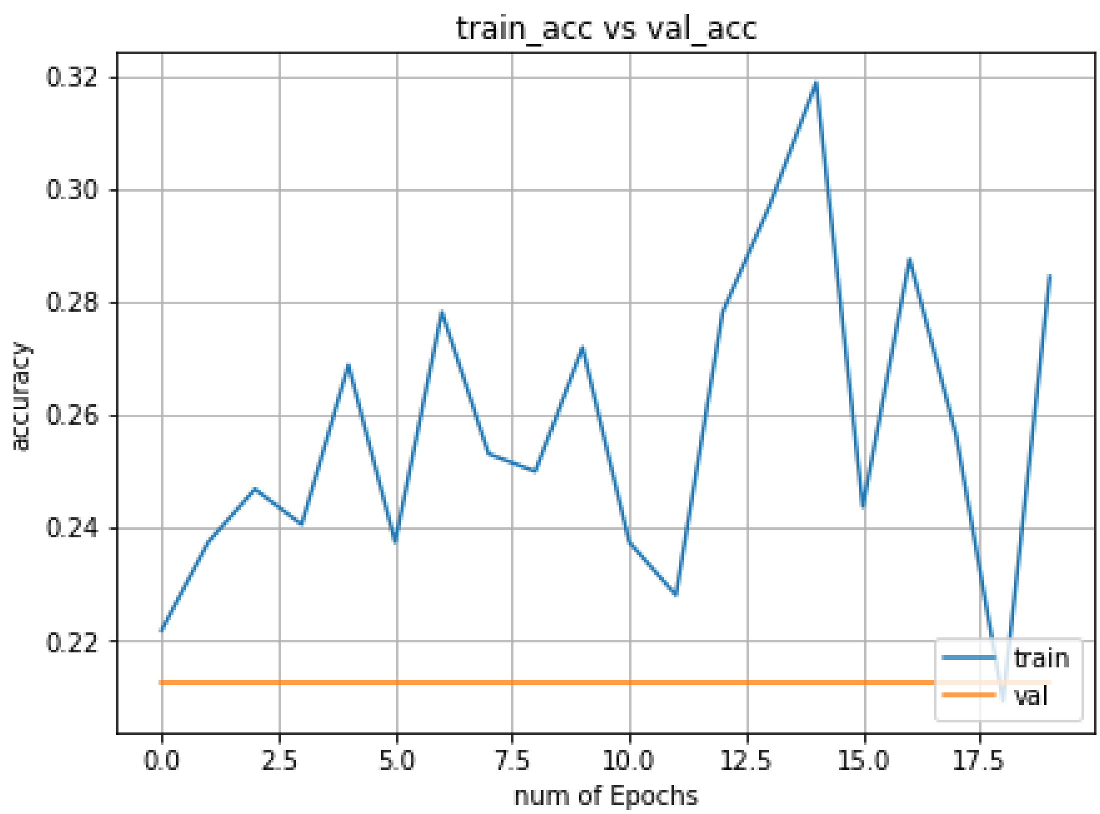
Other things we can do:

Analyze training statistics

```python
train_loss=hist.history['loss']
val_loss=hist.history['val_loss']
train_acc=hist.history['accuracy']
val_acc=hist.history['val_accuracy']
xc=range(nb_epoch)

plt.figure(1,figsize=(7,5))
plt.plot(xc,train_loss)
plt.plot(xc,val_loss)
plt.xlabel('num of Epochs')
plt.ylabel('loss')
plt.title('train_loss vs val_loss')
plt.grid(True)
plt.legend(['train','val'])
#print (plt.style.available # use bmh, classic,ggplot for big
pictures)
#plt.style.use(['classic'])

plt.figure(2,figsize=(7,5))
plt.plot(xc,train_acc)
plt.plot(xc,val_acc)
plt.xlabel('num of Epochs')
plt.ylabel('accuracy')
plt.title('train_acc vs val_acc')
plt.grid(True)
plt.legend(['train','val'],loc=4)
#print plt.style.available # use bmh, classic,ggplot for big pictures
#plt.style.use(['classic'])
```

```
<matplotlib.legend.Legend at 0x7f44431bdd50>
```


train_loss vs val_loss
train
val
loss
1.4025
1.4000
1.3975
1.3950
1.3925
1.3900
1.3875
1.3850
1.3825
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
num of Epochs


train_acc vs val_acc
accuracy
0.32
0.30
0.28
0.26
0.24
0.22
train
val
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
num of Epochs