facial-expression-detection-cnn

April 1, 2024

[1]:
```
# This Python 3 environment comes with many helpful analytics libraries␣
 ↪installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/
 ↪docker-python
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list␣
 ↪the files in the input directory

import os
print(os.listdir("../input/facial-expression/fer2013/"))

# Any results you write to the current directory are saved as output.
```

```
['fer2013.csv']
```

[2]:
```
import tensorflow as tf

import keras
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, AveragePooling2D
from keras.layers import Dense, Activation, Dropout, Flatten

from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
Using TensorFlow backend.
```

[3]:
```
# get the data
filname = '../input/facial-expression/fer2013/fer2013.csv'
label_map = ['Anger', 'Disgust', 'Fear', 'Happy', 'Sad', 'Surprise', 'Neutral']
```

```
names=['emotion','pixels','usage']
df=pd.read_csv('../input/facial-expression/fer2013/fer2013.csv',names=names,␣
 ↪na_filter=False)
im=df['pixels']
df.head(10)
```

[3]:
```
   emotion    …        usage
0  emotion    …        Usage
1        0    …     Training
2        0    …     Training
3        2    …     Training
4        4    …     Training
5        6    …     Training
6        2    …     Training
7        4    …     Training
8        3    …     Training
9        3    …     Training

[10 rows x 3 columns]
```

[4]:
```
def getData(filname):
    # images are 48x48
    # N = 35887
    Y = []
    X = []
    first = True
    for line in open(filname):
        if first:
            first = False
        else:
            row = line.split(',')
            Y.append(int(row[0]))
            X.append([int(p) for p in row[1].split()])

    X, Y = np.array(X) / 255.0, np.array(Y)
    return X, Y
```

[5]:
```
X, Y = getData(filname)
num_class = len(set(Y))
print(num_class)
```

```
7
```

[6]:
```
# keras with tensorflow backend
N, D = X.shape
X = X.reshape(N, 48, 48, 1)
```

[7]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.1,␣
 ↪random_state=0)
y_train = (np.arange(num_class) == y_train[:, None]).astype(np.float32)
y_test = (np.arange(num_class) == y_test[:, None]).astype(np.float32)
```

[8]:
```
from keras.models import Sequential
from keras.layers import Dense , Activation , Dropout ,Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.metrics import categorical_accuracy
from keras.models import model_from_json
from keras.callbacks import ModelCheckpoint
from keras.optimizers import *
from keras.layers.normalization import BatchNormalization
```

[9]:
```
def my_model():
    model = Sequential()
    input_shape = (48,48,1)
    model.add(Conv2D(64, (5, 5), input_shape=input_shape,activation='relu',␣
 ↪padding='same'))
    model.add(Conv2D(64, (5, 5), activation='relu', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(128, (5, 5),activation='relu',padding='same'))
    model.add(Conv2D(128, (5, 5),activation='relu',padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Conv2D(256, (3, 3),activation='relu',padding='same'))
    model.add(Conv2D(256, (3, 3),activation='relu',padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Flatten())
    model.add(Dense(128))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(0.2))
    model.add(Dense(7))
    model.add(Activation('softmax'))

    model.compile(loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'],optimizer='adam')
    # UNCOMMENT THIS TO VIEW THE ARCHITECTURE
```

```
    #model.summary()

    return model
model=my_model()
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 48, 48, 64)        1664
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 48, 48, 64)        102464
_________________________________________________________________
batch_normalization_1 (Batch (None, 48, 48, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 24, 24, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 24, 24, 128)       204928
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 24, 24, 128)       409728
_________________________________________________________________
batch_normalization_2 (Batch (None, 24, 24, 128)       512
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 12, 12, 128)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 12, 12, 256)       295168
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 12, 12, 256)       590080
_________________________________________________________________
batch_normalization_3 (Batch (None, 12, 12, 256)       1024
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 6, 256)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 9216)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               1179776
_________________________________________________________________
batch_normalization_4 (Batch (None, 128)               512
_________________________________________________________________
activation_1 (Activation)    (None, 128)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 7)                 903
_________________________________________________________________
activation_2 (Activation)    (None, 7)                 0
=================================================================
```

```
Total params: 2,787,015
Trainable params: 2,785,863
Non-trainable params: 1,152
_________________________________________________________________
```

[10]:
```
path_model='model_filter.h5' # save model at this location after each epoch
K.tensorflow_backend.clear_session() # destroys the current graph and builds a␣
 ↪new one
model=my_model() # create the model
K.set_value(model.optimizer.lr,1e-3) # set the learning rate
# fit the model
h=model.fit(x=X_train,
            y=y_train,
            batch_size=64,
            epochs=20,
            verbose=1,
            validation_data=(X_test,y_test),
            shuffle=True,
            callbacks=[
                ModelCheckpoint(filepath=path_model),
            ]
            )
```

```
Train on 32298 samples, validate on 3589 samples
Epoch 1/20
32298/32298 [==============================] - 20s 622us/step - loss: 1.6698 -
acc: 0.3533 - val_loss: 1.7831 - val_acc: 0.2756
Epoch 2/20
32298/32298 [==============================] - 14s 441us/step - loss: 1.3242 -
acc: 0.4926 - val_loss: 1.3562 - val_acc: 0.4737
Epoch 3/20
32298/32298 [==============================] - 14s 437us/step - loss: 1.1302 -
acc: 0.5764 - val_loss: 1.1762 - val_acc: 0.5492
Epoch 4/20
32298/32298 [==============================] - 14s 438us/step - loss: 1.0033 -
acc: 0.6245 - val_loss: 1.1480 - val_acc: 0.5573
Epoch 5/20
32298/32298 [==============================] - 14s 437us/step - loss: 0.8746 -
acc: 0.6794 - val_loss: 1.0692 - val_acc: 0.6010
Epoch 6/20
32298/32298 [==============================] - 14s 437us/step - loss: 0.7291 -
acc: 0.7326 - val_loss: 1.1269 - val_acc: 0.5979
Epoch 7/20
32298/32298 [==============================] - 14s 436us/step - loss: 0.5656 -
acc: 0.7965 - val_loss: 1.2335 - val_acc: 0.6018
Epoch 8/20
32298/32298 [==============================] - 14s 438us/step - loss: 0.4108 -
acc: 0.8574 - val_loss: 1.2776 - val_acc: 0.6018
```

```
Epoch 9/20
32298/32298 [==============================] - 14s 437us/step - loss: 0.2881 -
acc: 0.9007 - val_loss: 1.5797 - val_acc: 0.5804
Epoch 10/20
32298/32298 [==============================] - 14s 435us/step - loss: 0.2052 -
acc: 0.9328 - val_loss: 1.5014 - val_acc: 0.6108
Epoch 11/20
32298/32298 [==============================] - 14s 437us/step - loss: 0.1676 -
acc: 0.9432 - val_loss: 1.7935 - val_acc: 0.6063
Epoch 12/20
32298/32298 [==============================] - 14s 438us/step - loss: 0.1285 -
acc: 0.9596 - val_loss: 1.8073 - val_acc: 0.6082
Epoch 13/20
32298/32298 [==============================] - 14s 443us/step - loss: 0.1086 -
acc: 0.9657 - val_loss: 1.8007 - val_acc: 0.6135
Epoch 14/20
32298/32298 [==============================] - 14s 436us/step - loss: 0.1115 -
acc: 0.9637 - val_loss: 1.9739 - val_acc: 0.6060
Epoch 15/20
32298/32298 [==============================] - 14s 436us/step - loss: 0.1040 -
acc: 0.9651 - val_loss: 1.9873 - val_acc: 0.6055
Epoch 16/20
32298/32298 [==============================] - 14s 442us/step - loss: 0.0952 -
acc: 0.9691 - val_loss: 1.9692 - val_acc: 0.6035
Epoch 17/20
32298/32298 [==============================] - 15s 459us/step - loss: 0.0935 -
acc: 0.9691 - val_loss: 2.0034 - val_acc: 0.6149
Epoch 18/20
32298/32298 [==============================] - 15s 450us/step - loss: 0.0794 -
acc: 0.9748 - val_loss: 2.1322 - val_acc: 0.6258
Epoch 19/20
32298/32298 [==============================] - 14s 448us/step - loss: 0.0843 -
acc: 0.9725 - val_loss: 2.0265 - val_acc: 0.6211
Epoch 20/20
32298/32298 [==============================] - 15s 450us/step - loss: 0.0724 -
acc: 0.9763 - val_loss: 2.2204 - val_acc: 0.6163
```

[11]:
```
objects = ('angry', 'disgust', 'fear', 'happy', 'sad', 'surprise', 'neutral')
y_pos = np.arange(len(objects))
print(y_pos)
```

```
[0 1 2 3 4 5 6]
```

[12]:
```
def emotion_analysis(emotions):
    objects = ['angry', 'disgust', 'fear', 'happy', 'sad', 'surprise',␣
 ↪'neutral']
    y_pos = np.arange(len(objects))
```

```
    plt.bar(y_pos, emotions, align='center', alpha=0.9)
    plt.tick_params(axis='x', which='both', pad=10,width=4,length=10)
    plt.xticks(y_pos, objects)
    plt.ylabel('percentage')
    plt.title('emotion')

plt.show()
```

[13]:
```
y_pred=model.predict(X_test)
#print(y_pred)
y_test.shape
```

[13]:
```
(3589, 7)
```

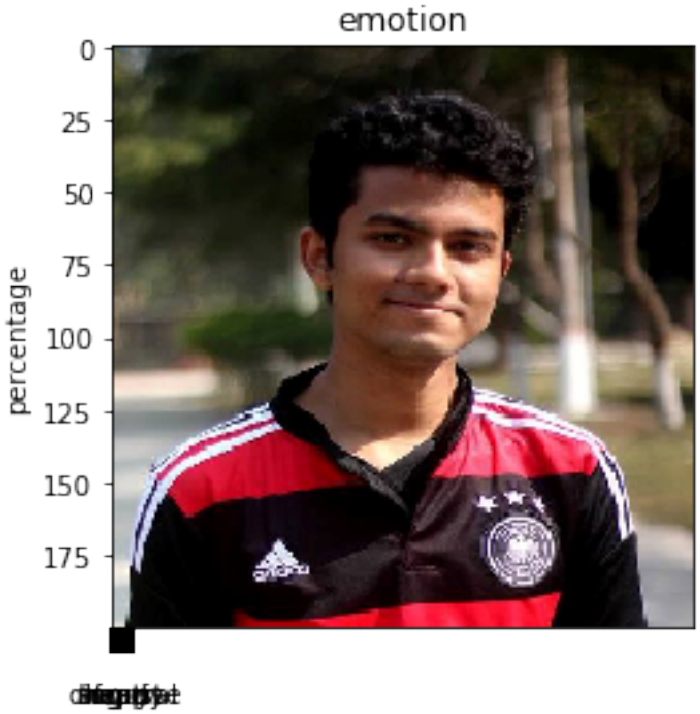
[14]:
```
#import seaborn as sn
#import pandas as pd
#import matplotlib.pyplot as plt
#import numpy as np
#from sklearn.metrics import confusion_matrix
#%matplotlib inline
#cm = confusion_matrix(np.where(y_test == 1)[1], y_pred)
#cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
#df_cm = pd.DataFrame(cm, index = [i for i in "0123456"],
                     #columns = [i for i in "0123456"])
#plt.figure(figsize = (20,15))
#sn.heatmap(df_cm, annot=True)
```

Real Time Expression Prediction

[15]:
```
from skimage import io
img = image.load_img('../input/myimage/Shawon.jpg', grayscale=True,␣
 ↪target_size=(48, 48))
show_img=image.load_img('../input/myimage/Shawon.jpg', grayscale=False,␣
 ↪target_size=(200, 200))
x = image.img_to_array(img)
x = np.expand_dims(x, axis = 0)

x /= 255

custom = model.predict(x)
#print(custom[0])
emotion_analysis(custom[0])

x = np.array(x, 'float32')
x = x.reshape([48, 48]);

plt.gray()
```

```
plt.imshow(show_img)
plt.show()

m=0.00000000000000000001
a=custom[0]
for i in range(0,len(a)):
    if a[i]>m:
        m=a[i]
        ind=i

print('Expression Prediction:',objects[ind])
```

```
/opt/conda/lib/python3.6/site-packages/keras_preprocessing/image.py:492:
UserWarning: grayscale is deprecated. Please use color_mode = "grayscale"
  warnings.warn('grayscale is deprecated. Please use '
```




```
Expression Prediction: happy
```

[16]:
```
from skimage import io
img = image.load_img('../input/testimages/wallpaper2you_443897.jpg',␣
 ↪grayscale=True, target_size=(48, 48))
show_img=image.load_img('../input/testimages/wallpaper2you_443897.jpg',␣
 ↪grayscale=False, target_size=(200, 200))
x = image.img_to_array(img)
```

```
x = np.expand_dims(x, axis = 0)

x /= 255

custom = model.predict(x)
#print(custom[0])
emotion_analysis(custom[0])

x = np.array(x, 'float32')
x = x.reshape([48, 48]);

plt.gray()
plt.imshow(show_img)
plt.show()

m=0.00000000000000000001
a=custom[0]
for i in range(0,len(a)):
    if a[i]>m:
        m=a[i]
        ind=i

print('Expression Prediction:',objects[ind])
```

`Expression Prediction: angry`

Live Demo of Production Level Project

Facial Expression Detection Web App