Emotion Recognition Photo Editor

28.05.2021

—

Parth Shukla
Human Computer Interaction Project
Computer Science
Jacobs University Bremen

Overview

The project is in essence quite simple. It detects the human emotion being portrayed by the individual in the picture and adds an element (currently, just an **Emoji**) relating to the human emotion. The project also uses live speech to text to inscribe captions on these images as well as comes with a still under-development option to perform even cooler, more experimental edits. The main implementation that I worked on are emotion detection using a Convolutional Neural Network and operations with OpenCV. Speech to Text is performed by the free Google Speech To Text API and the experimental edits are derivative of Tim Chinenov's work. Face detection done using Intel's implementation of a Haar Cascade classifier.

Specifications

The project primarily makes use of the following technologies to function:-

- Python 3.8
- Keras
- OpenCV
- NumPy
- Pandas
- Seaborn

The development and testing of this project was done on **macOs** Catalina.

Major Milestones

IDetecting Faces and Emotions

Working with OpenCV and attaching Media Tools

Adding experimental edits

I. Detecting Faces and Emotions

1. Face-Detection

Face-detection for this project is being done by a **Haar Cascade** classifier which is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

- Pick f (maximum acceptable false positive rate per layer) and d (minimum acceptable detection rate per layer)
- Lets F_{target} is target overall false positive rate
- Lets P is a set of positive examples
- Lets N is a set of negative examples
- Lets $F_0 = 1$, $D_0=1$, and $i=0$ (F_0: overall false positive rate at layer 0, D_0: acceptable detection rate at layer 0, and i: is the current layer)
- While $F_i > F_{target}$ (F_i: overall false positive rate at layer i):
  - i++ (layer increasing by 1)
  - $n_i=0$; $F_i = F_{i-1}$ (n_i: negative example i):
  - While $F_i > f*F_{i-1}$:
    - n_i ++ (check a next negative example)
    - Use P and N to train with AdaBoost to make a xml (classifier)
    - Check the result of new classifier for F_i and D_0
    - Decrease threshold for new classifier to adjust detection rate $r >= d*F_{i-1}$
  - N = empty
  - If $F_i > F_{target}$, use the current classifier and false detection to set N

Figure 1. Haar-cascade algorithm pseudo code.

In this project, an open-source classifier implemented by **Intel** was used.

2. Training the Emotion Detection model

To train the emotion detection model, the FER 2013 Dataset was used.

- **Libraries Used**

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import os

from keras.preprocessing.image import load_img, img_to_array
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import
Dense,Input,Dropout,GlobalAveragePooling2D,Flatten,Conv2D,BatchNormalization,Ac
tivation,MaxPooling2D
from keras.models import Model,Sequential
from keras.optimizers import Adam,SGD,RMSprop
```

The deep learning libraries are very helpful in creating the model. It becomes very easy to load images and convert them into an array type data format. Then, the layers required to build our CNN are also there. Additionally, some optimizers were also imported.

- **Making Training and Validation Data**

```
batch_size  = 128

datagen_train  = ImageDataGenerator()
datagen_val = ImageDataGenerator()

train_set = datagen_train.flow_from_directory(folder_path+"train",
                                              target_size = (picture_size,picture_size),
                                              color_mode = "grayscale",
                                              batch_size=batch_size,
                                              class_mode='categorical',
                                              shuffle=True)


validation_set = datagen_val.flow_from_directory(folder_path+"validation",
                                              target_size = (picture_size,picture_size),
                                              color_mode = "grayscale",
                                              batch_size=batch_size,
                                              class_mode='categorical',
                                              shuffle=False)
```

The model makes use of a batch size of 128 which means that it uses 128 training examples for 1 iteration. The “train_set” and “validation_test” contain the data being given to them by the ImageDataGenerators from the prescribed image directories. The image sizes have been standardised to 48x48 and are being studied as grayscale pictures. Since emotion recognition is a classification issue, the class_mode is also set to categorical as we would want to have differentiated categories of emotions. The training and classification sets are split in a 4:1 ratio respectively.

- **Building the CNN model**

```
no_of_classes = 7

model = Sequential()

#First Layer
model.add(Conv2D(64,(3,3),padding = 'same',input_shape = (48,48,1)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout(0.25))

#Second Layer
model.add(Conv2D(128,(5,5),padding = 'same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout (0.25))

#Third Layer
model.add(Conv2D(512,(3,3),padding = 'same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout (0.25))

#Fourth Layer
model.add(Conv2D(512,(3,3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())

#First Fully Connected Layer of the CNN
model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))
```

```
model.add(Dropout(0.25))


# Fully Connected Second Layer
model.add(Dense(512))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))

model.add(Dense(no_of_classes, activation='softmax'))



opt = Adam(lr = 0.0001)
model.compile(optimizer=opt,loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

The number of classes signifies the number of possible emotional outcomes and we're working with a **Sequential** model. Each layer is imperative in proper functioning of this Neural Network and have set parameters to help them perform well. In the first layer, we have 64 output filters operating with a 3x3 kernel size. The padding of the CNN layer is kept same from all sides. The input shape co-relates to our standardised picture sizes and since they're in grayscale, the channel is set to 1. The BatchNormalization layer helps make the CNN faster and more stable through normalization of the layers' inputs by re-centering and re-scaling. We're using the rectified linear unit activation function in our activation layer; it is likely the most commonly used activation function in the world. The rectified linear activation function overcomes the vanishing gradient problem, allowing models to learn faster and perform better. The MaxPooling2d layer extracts the most useful information from the 2x2 area. The Dropout layer prevents from overfitting the model. Later on, the Flatten layer transposes the input back into a one dimensional array.

- **Fitting The Model From Collected Data**

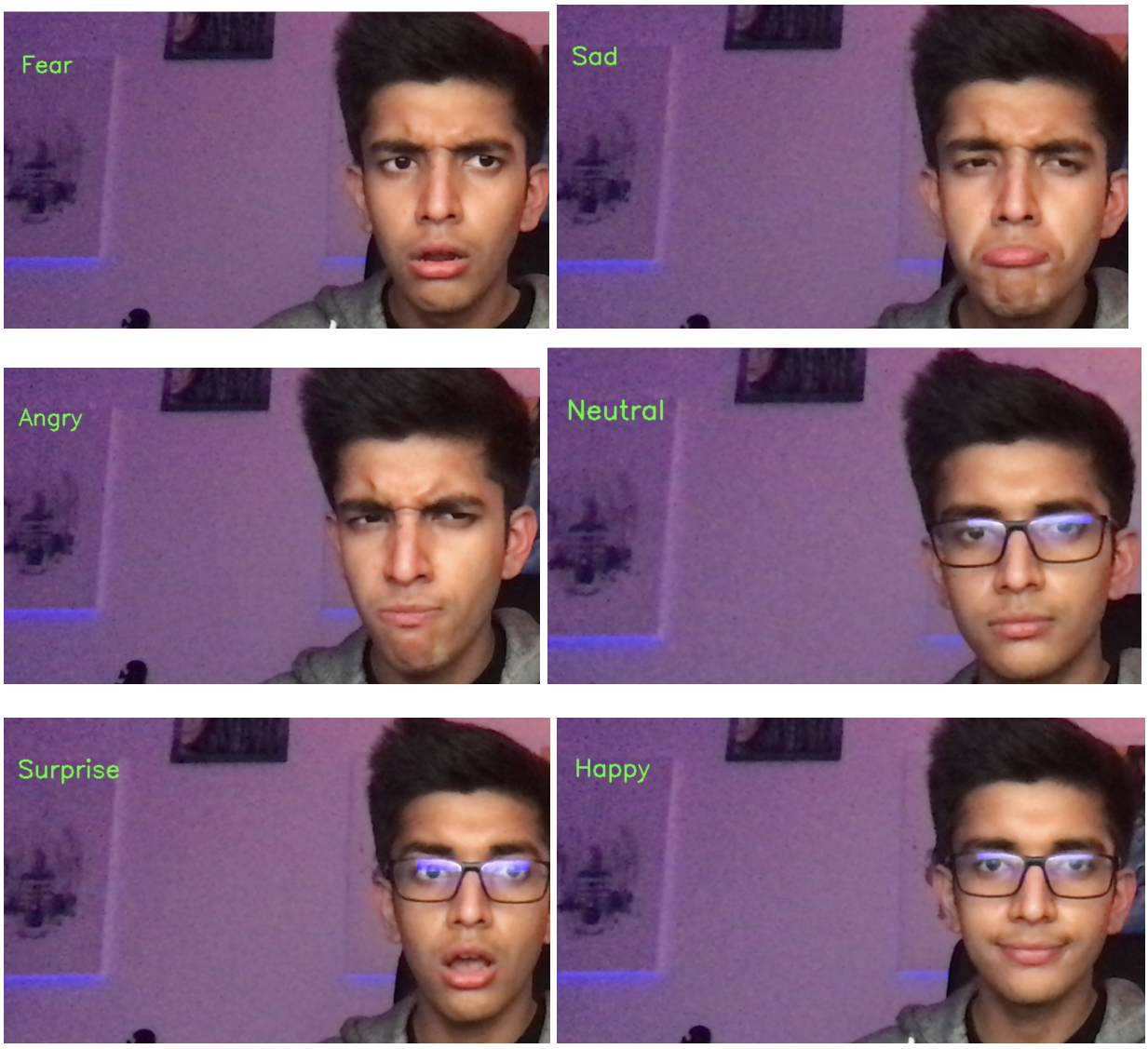
```
epochs = 48

from keras.callbacks import ModelCheckpoint

checkpoint = ModelCheckpoint("model_weights.h5", monitor='val_acc', verbose=1,
save_best_only=True, mode='max')
callbacks_list = [checkpoint]

history = model.fit_generator(generator=train_set,
                              steps_per_epoch=train_set.n//train_set.batch_size,
                              epochs=epochs,
                              validation_data = test_set,
                              validation_steps = test_set.n//test_set.batch_size,
                              callbacks=callbacks_list)
```

Results:- A model that was 68% accurate.

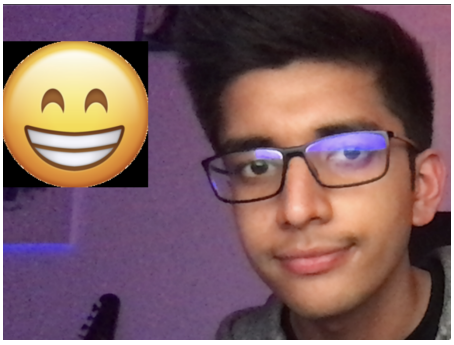


Sadly, I was unable to get the model to detect disgust from my face. Hard to do when you're looking at yourself in the screen.

II. Working with OpenCV and attaching Media Tools

Now with the model functional, I wanted to use that to make changes to the pictures I ended up seeing. I decided to add emojis that would correlate to the said emotion in the result. It was done by getting standardized sized emojis from the internet, and then I had to associate the overlay of the emoji based upon the last seen emotion on the OpenCV capture.

Overlaying the media took a surprisingly large amount of time. Had I had more time, I would work to remove the alpha channel so I could do a transparent overlay on the base media.



However, I was still not satisfied. I wanted to make this tool a little more useful. So I decided to integrate a way to add static captions to the feed. I tried a bunch of engines that would work well, both online and offline and decided to stick with the free Google Speech To Text engine to achieve it as it gave considerably better and faster results than some of the offline engines that I worked with (eg:- CMU pocketSphinx). With the caption problem solved, I wanted to think what use-case could this tool possibly fulfil and I realised the power this tool had. I could now use my tool to make **memes**.

III. Adding experimental edits

After I realised the power OpenCV and my trained models held, I started exploring what more ways could I use these tools. Turns out, there's a whole world of experimental ideas and edits that I can work with. In my project, I've integrated Tim Chinenov's Vaporware edits as an optional end for the program. This optional end is **currently buggy** and only sometimes produces beautiful works like the one seen above.

What's Next?

I hope to dig down further but also see how I can further look into emotion recognition and provide edits using these techniques that could provide the right scenery based on the emotions it recognises. But that's for the future. I hope to make use of the knowledge I gathered in the HCI class to make more fun projects in the future! Thanks for reading :)