Detect Emotion, Age, Gender in Image!

sarehj@kth.se

Objective and Requirements

Age and gender, two of the key facial attributes, play a very foundational role in social interactions, making age and gender estimation from a single face image an important task in intelligent applications.

This Solution is able to detect faces in an image and can predict the Age, Gender, and Emotion of the person. It can also work in condition if there are multiple faces present in the image.
We will use a custom VGG16 model developed and trained on open source facial datasets downloaded from Kaggle and IMDB. OpenCV, dlib & keras were used to aid facial detection. The final system can detect the emotion, age and gender of people in any given image and by using a webcam.

Design of the system

1- VGG network: Creating a facial recognition model from scratch is a daunting task. You need to find, collect and then annotate a ton of images to have any hope of building a decent model. Hence using a pretrained model in this domain makes a lot of sense. VGG-Face is a dataset that contains 2,622 unique identities with more than two million faces.
The VGG network architecture was introduced by Simonyan and Zisserman in their 2014 paper, Very Deep Convolutional Networks for Large Scale Image Recognition. This network is characterized by its simplicity, using only 3×3 convolutional layers stacked on top of each other in increasing depth. Reducing volume size is handled by max pooling. Two fully-connected layers, each with 4,096 nodes are then followed by a softmax classifier.[1]

2-Wide Residual Networks are a variant on ResNets where we decrease depth and increase the width of residual networks.that as the network gets deeper, its performance saturates or even starts to degrade.Result in improved performance, far fewer layers, and faster training. WRN Design Rules The basic design rules for WRNs, as laid out in this seminal paper by Sergey Zagoruyko and Nikos Komodakis, are as follows:

1. Convolution filters should be no larger than 3x3 pixels for best performance.
2. Follow the convention that when you downsample by 2, you double the number of feature maps (filter channels).
3. Three important factors are introduced:
    - l, the deepening factor, which represents the number of convolutional layers in a ResNet block (often l = 2 is optimal);
    - N, the number of ResNet blocks (repeats; mini blocks) within a given convolutional group (wherein all layers in a group have the same fixed number of filter channels);
    - K, the widening factor, which multiplies the number of feature channels in each convolution layer. [2]

3-Dlib:
Our face has several features that can be identified, like our eyes, mouth, nose, etc. When we use DLib algorithms to detect these features we actually get a map of points that surround each feature.
We used a Dlib function called get_frontal_face_detector(), pretty intuitive. There is a caveat though, this function will only work with grayscale images, so we will have to do that first with OpenCV.

4-Prediction:
A class prediction is: given the finalized model and one or more data instances, predict the class for the data instances. We do not know the outcome classes for the new data. That is why we need the model in the first place. We can predict the class for new data instances using our finalized classification model in scikit-learn using the predict() function. For example, we have one or more data instances in an array called Xnew. This can be passed to the predict() function on our model in order to predict the class values for each instance in the array.

Technologies used

- OpenCv : OpenCV-Python is a library of Python bindings designed to solve computer vision problems.
- Dlib : is an advanced machine learning library that was created to solve complex real-world problems.
- Numpy : NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
- Keras : Keras is an API designed for human beings, not machines. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases.
- Python : Python is an interpreted, high-level and general-purpose programming language.
- Jupyter notebook : Project Jupyter exists to develop open-source software, open-standards, and services for interactive computing across dozens of programming languages.

Architecture

Transfer learning is a concept according to which we can transfer the learning of other pre-trained models to our data. Instead of training our own custom neural network, we can use other popular pre-trained models and pass our data to those models and ultimately get the features for our images.

We used part of a model graph named VGG16.VGG16 is a popular neural network architecture and Keras makes it easy to get a model.VGG-16 network contains 16 layers out of which 13 layers are convolution layers. This neural network is well trained on an image-net dataset which contains millions of images.

In this project we have trained a CNN using Keras(miniature version of VGG16) and used it with Opencv3 library to detect a person's face emotion. The model trained can detect up to six different emotions.These six emotions are: Happy, Sad, Angry, Surprise, Fear and Neutral.

pre-trained VGG-16

```
In [7]: from keras.applications.vgg16 import VGG16

In [8]: model=VGG16(weights='imagenet', include_top=False)

In [9]: model.summary()
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, None, None, 3)]   0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, None, None, 64)    1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, None, None, 64)    36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, None, None, 64)    0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, None, None, 128)   73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, None, None, 128)   147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, None, None, 128)   0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, None, None, 256)   295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, None, None, 256)   590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, None, None, 256)   590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, None, None, 256)   0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, None, None, 512)   1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, None, None, 512)   0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, None, None, 512)   0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
```

now lets build our model

In [17]:
```
model = Sequential()

model.add(Conv2D(32, (3, 3), padding = 'same', kernel_initializer="he_normal",
                 input_shape = (img_rows, img_cols, 1)))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Conv2D(32, (3, 3), padding = "same", kernel_initializer="he_normal",
                 input_shape = (img_rows, img_cols, 1)))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

# Block #2: second CONV => RELU => CONV => RELU => POOL
# Layer set
model.add(Conv2D(64, (3, 3), padding="same", kernel_initializer="he_normal"))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Conv2D(64, (3, 3), padding="same", kernel_initializer="he_normal"))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

# Block #3: third CONV => RELU => CONV => RELU => POOL
# Layer set
model.add(Conv2D(128, (3, 3), padding="same", kernel_initializer="he_normal"))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3), padding="same", kernel_initializer="he_normal"))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

# Block #4: third CONV => RELU => CONV => RELU => POOL
# Layer set
model.add(Conv2D(256, (3, 3), padding="same", kernel_initializer="he_normal"))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Conv2D(256, (3, 3), padding="same", kernel_initializer="he_normal"))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

# Block #5: first set of FC => RELU layers
model.add(Flatten())
model.add(Dense(64, kernel_initializer="he_normal"))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

# Block #6: second set of FC => RELU layers
model.add(Dense(64, kernel_initializer="he_normal"))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

# Block #7: softmax classifier
model.add(Dense(num_classes, kernel_initializer="he_normal"))
model.add(Activation("softmax"))

print(model.summary())
```

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 48, 48, 32)	320
activation_1 (Activation)	(None, 48, 48, 32)	0
batch_normalization_1 (Batch	(None, 48, 48, 32)	128
conv2d_2 (Conv2D)	(None, 48, 48, 32)	9248
activation_2 (Activation)	(None, 48, 48, 32)	0
batch_normalization_2 (Batch	(None, 48, 48, 32)	128
max_pooling2d_1 (MaxPooling2	(None, 24, 24, 32)	0
dropout_1 (Dropout)	(None, 24, 24, 32)	0
conv2d_3 (Conv2D)	(None, 24, 24, 64)	18496
activation_3 (Activation)	(None, 24, 24, 64)	0
batch_normalization_3 (Batch	(None, 24, 24, 64)	256
conv2d_4 (Conv2D)	(None, 24, 24, 64)	36928
activation_4 (Activation)	(None, 24, 24, 64)	0
batch_normalization_4 (Batch	(None, 24, 24, 64)	256
max_pooling2d_2 (MaxPooling2	(None, 12, 12, 64)	0
dropout_2 (Dropout)	(None, 12, 12, 64)	0
conv2d_5 (Conv2D)	(None, 12, 12, 128)	73856
activation_5 (Activation)	(None, 12, 12, 128)	0
batch_normalization_5 (Batch	(None, 12, 12, 128)	512
conv2d_6 (Conv2D)	(None, 12, 12, 128)	147584
activation_6 (Activation)	(None, 12, 12, 128)	0
batch_normalization_6 (Batch	(None, 12, 12, 128)	512
max_pooling2d_3 (MaxPooling2	(None, 6, 6, 128)	0
dropout_3 (Dropout)	(None, 6, 6, 128)	0
conv2d_7 (Conv2D)	(None, 6, 6, 256)	295168
activation_7 (Activation)	(None, 6, 6, 256)	0
batch_normalization_7 (Batch	(None, 6, 6, 256)	1024
conv2d_8 (Conv2D)	(None, 6, 6, 256)	590080
activation_8 (Activation)	(None, 6, 6, 256)	0
batch_normalization_8 (Batch	(None, 6, 6, 256)	1024
max_pooling2d_4 (MaxPooling2	(None, 3, 3, 256)	0
dropout_4 (Dropout)	(None, 3, 3, 256)	0
flatten_1 (Flatten)	(None, 2304)	0
dense_1 (Dense)	(None, 64)	147520
activation_9 (Activation)	(None, 64)	0
batch_normalization_9 (Batch	(None, 64)	256
dropout_5 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 64)	4160
activation_10 (Activation)	(None, 64)	0
batch_normalization_10 (Batc	(None, 64)	256
dropout_6 (Dropout)	(None, 64)	0
dense_3 (Dense)	(None, 6)	390
activation_11 (Activation)	(None, 6)	0

Total params: 1,328,102
Trainable params: 1,325,926
Non-trainable params: 2,176

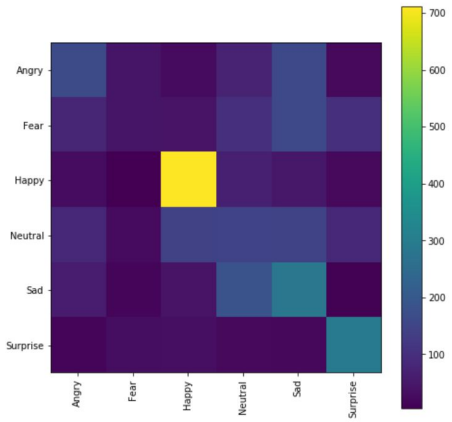
None

Training our model:

```
Found 3534 images belonging to 6 classes.
Confusion Matrix
[[167  43  25  75 160  21]
 [ 79  45  42 103 161  98]
 [ 27   4 711  65  49  23]
 [ 85  24 142 145 148  82]
 [ 59  15  40 186 285   9]
 [ 13  31  32  22  19 299]]
Classification Report
              precision    recall  f1-score   support

       Angry       0.39      0.34      0.36       491
        Fear       0.28      0.09      0.13       528
       Happy       0.72      0.81      0.76       879
     Neutral       0.24      0.23      0.24       626
         Sad       0.35      0.48      0.40       594
    Surprise       0.56      0.72      0.63       416

   micro avg       0.47      0.47      0.47      3534
   macro avg       0.42      0.44      0.42      3534
weighted avg       0.44      0.47      0.44      3534
```



Test set

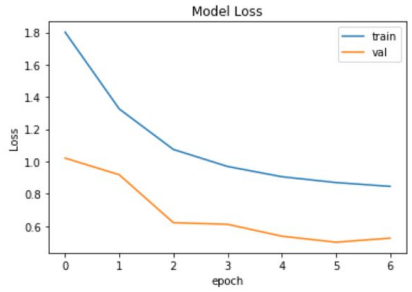
Then we trained both of the models. Below are the results we obtained on the test set.

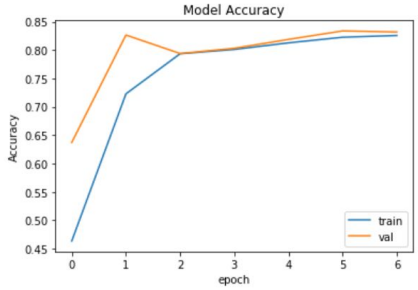
```
]: scores = model1.evaluate(pt_test, ytest_lab)
   print('%s: %.2f%%'% ("Accuracy", scores[1]*100))

   489/489 [==============================] - 0s 167us/step
   Accuracy: 79.96%
```

```
scores = model.evaluate(x_test, ytest_lab)
print('%s: %.2f%%'% ("Accuracy", scores[1]*100))

489/489 [==============================] - 0s 532us/step
Accuracy: 75.66%
```





Conclusion

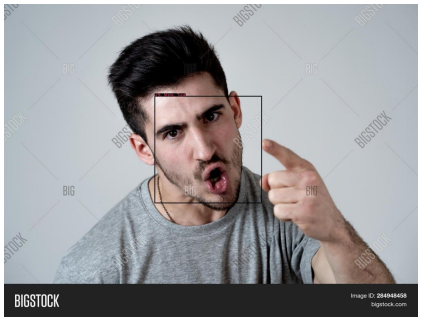
We compared our own modeling with VGG16 and we saw that VGG16 was more accurate and faster. VGG16 is better at detecting faces and is able to give the results in a more correct way.

The program is able to detect faces in an image or in a webcam video and then recognises its gender and its age and also its emotion. So we reached our goal and later we would like to make it more accurate.

---

[1]https://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/
[2]https://arxiv.org/abs/1512.03385
[3]https://numpy.org
[4]https://keras.io
[5]http://dlib.net
[6]https://opencv.or

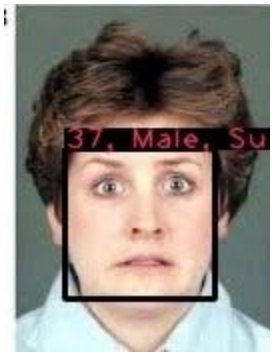
Results

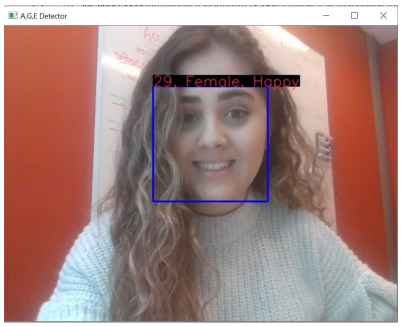

BIGSTOCK

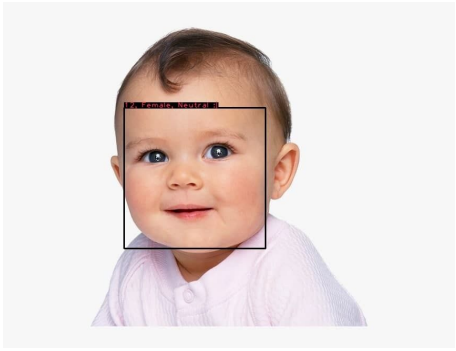

26, Female, Neutral :|

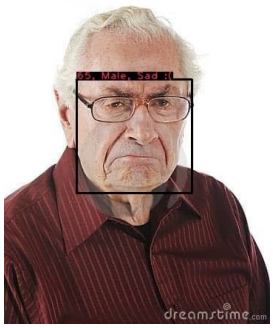

28, Female, Happy :)

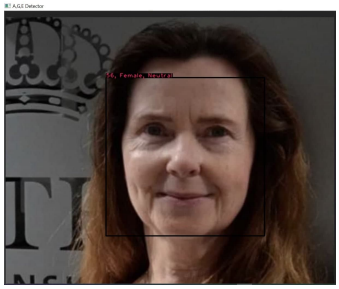

32, Female, Sad :(


37, Male, Su


A.G.E Detector
29, Female, Happy




56, Male, Sad :(
dreamstime.com


A.G.E Detector