Face Recognition Using TensorFlow Pre-Trained Model & OpenCV

Swastik Somani
Jan 16 · 3 min read

Hi, I'm Swastik Somani, a machine learning enthusiast. Today I will share you how to create a face recognition model using TensorFlow pre-trained model and OpenCv used to detect the face.

> *Hope you will like my content!!!!*

This blog divided into four parts-

1. Introduction of Face recognition.
2. Detect the Face using OpenCV.
3. Create the Face Recognition Model.
4. Convert the TensorFlow Model(.pb) into TensorFlow Lite(.tflite).

ntroduction of Face Recognition

Face Recognition system is used to identify the face of the person from image or video using the face features of the person. We create the face recognition model using the deep learning algorithm. It uses Convolution Neural Network to detect the face of the person.

How CNN Works??

Convolution neural network inspired by the biological process in the connectively pattern of neurons. CNN Face Classification takes input as a image, then it will process it and classify into the different categories which is stored in the our dataset. The hidden layer of CNN consist Convolutional layer, Activation Function(ReLu, Sigmoid & any other), pooling layers, fully connected layers and normalization layers.

> *Lets do coding!!!*

Detect the Face using OpenCV

Install the OpenCV using the cmd

```
pip install opencv-python
```

Or

```
pip3 install opencv-python
```

Import the libraries -

```
import cv2
import numpy as np
import os
```

face.py hosted with ♡ by GitHub view raw

First, we need to collect the images from the directory.

```
def dataset():
    images = []
    labels = []
    labels_dic = {}
    people = [person for person in os.listdir("people/")]
    for i, person in enumerate(people):
            labels_dic[i] = person
            for image in os.listdir("people/" + person):
                    images.append(cv2.imread("people/" + person + '/' + image, 0))
                    labels.append(person)

    return (images, np.array(labels), labels_dic)

images, labels, labels_dic = collect_dataset()
```

face.py hosted with ♡ by GitHub view raw

Now, we are using the “haarcascade_frontalface_default.xml” file you can easy download this xml file from the google.

Now detect the faces from the images

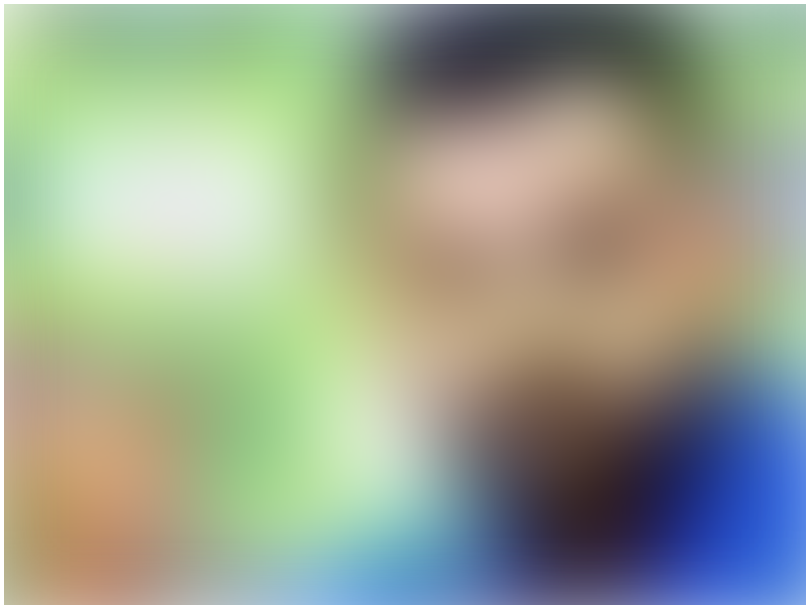
```
class FaceDetector(object):
    def __init__(self, xml_path):
        self.classifier = cv2.CascadeClassifier(xml_path)

    def detect(self, image, biggest_only=True):
        scale_factor = 1.2
        min_neighbors = 5
        min_size = (30, 30)
        biggest_only = True
        faces_coord = self.classifier.detectMultiScale(image,
                                                       scaleFactor=scale_factor,
                                                       minNeighbors=min_neighbors,
                                                       minSize=min_size,
                                                       flags=cv2.CASCADE_SCALE_IMAGE)
        return faces_coord

def cut_faces(image, faces_coord):
    faces = []

    for (x, y, w, h) in faces_coord:
        w_rm = int(0.3 * w / 2)
        faces.append(image[y: y + h, x + w_rm: x + w - w_rm])

    return faces

def resize(images, size=(224, 224)):
    images_norm = []
    for image in images:
        if image.shape < size:
            image_norm = cv2.resize(image, size,
                                    interpolation=cv2.INTER_AREA)
        else:
            image_norm = cv2.resize(image, size,
                                    interpolation=cv2.INTER_CUBIC)
        images_norm.append(image_norm)

    return images_norm




def normalize_faces(image, faces_coord):

    faces = cut_faces(image, faces_coord)
    faces = resize(faces)

```

```
46      return faces
47
48  for image in images:
49      detector = FaceDetector("haarcascade_frontalface_default.xml")
50      faces_coord = detector.detect(image, True)
51      faces = normalize_faces(image ,faces_coord)
52      for i, face in enumerate(faces):
53              cv2.imwrite('%s.jpeg' % (count), faces[i])
54              count += 1
```

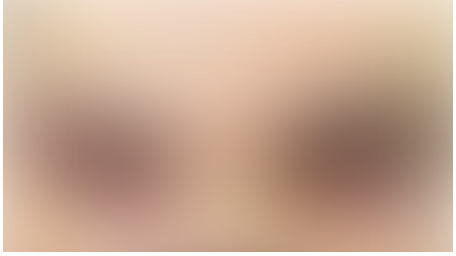
face.py hosted with ♡ by GitHub view raw

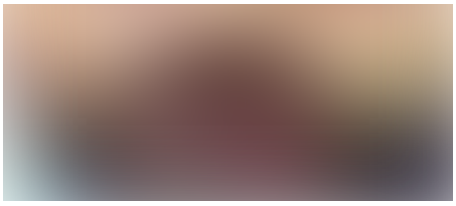
Run the python code -

```
python face.py
```



Before





After

> *Yeahhh we detect the face from the images!!!*

Create the Face Recognition Model

We create the our face recognition model by using the mobilenet pre-trained model.

You can download the retrain.py by using this link-

tensorflow/hub

A library for transfer learning by reusing parts of TensorFlow models. - tensorflow/hub

github.com

After downloading file, use this cmd to retrained the model

```
python retrain.py \
  --bottleneck_dir=tf_files/bottlenecks \
  --how_many_training_steps=500 \
  --model_dir=tf_files/models/ \
  --summaries_dir=tf_files/training_summaries/mobilenet_0.50_224 \
  --output_graph=tf_files/retrained_graph.pb \
  --output_labels=tf_files/retrained_labels.txt \
  --architecture=mobilenet_0.50_224 \
  --image_dir=tf_files/people
```

After running this cmd, it gives tensorflow model after re-training of the model. Now you have your own face recognition model 😮 😃.

To run the TensorBoard, run this commond

```
tensorboard --logdir tf_files/training_summaries &
```

TensorBoard Accuracy graph for the trained model

The orange line shows the accuracy of the model on the training data. While the blue line shows the accuracy on the test set.

Convert the TensorFlow Model(.pb) into TensorFlow Lite(.tflite).

You can also run one simple cmd to create tensorflow lite file.

```
tflite_convert \
  --graph_def_file=tf_files/retrained_graph.pb \
  --output_file=tf_files/optimized_graph.lite \
  --input_format=TENSORFLOW_GRAPHDEF \
  --output_format=TFLITE \
  --input_shape=1,224,224,3 \
  --input_array=input \
  --output_array=final_result \
  --inference_type=FLOAT \
  --input_data_type=FLOAT
```

After running this cmd, it convert tensorflow file(.pb) into tensorflow lite file(.tflite). You can use this tflite file into your android and ios phone.

Machine Learning Face Recognition Opencv TensorFlow Deep Learning