BeardIT

Nastya Merkulova
Optimization Class Project. MIPT

Introduction

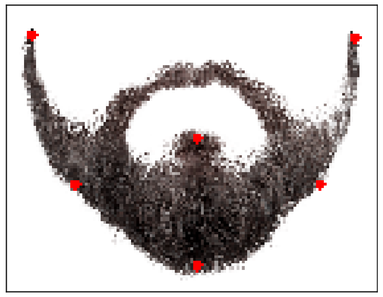
"How would I look with a beard?" - this agonizing question must be familiar to any man considering the momentous decision of growing it all off. This project presents a method of attaching a beard based on the spotting of the feature points on the face.

Face Recognition

- **Step 1: Finding all the Faces**
  We are going to use a method invented in 2005 called Histogram of Oriented Gradients (HOG). To find faces in an image, we will start by making our image black and white. Then we will look at every single pixel in our image one at a time. We want to draw an arrow showing in which direction the image is getting darker. We will break up the image into small squares of 16x16 pixels each. In each square, we will count up how many gradients point in each major direction (how many point up, point up-right, point right, etc). Then we will replace that square in the image with the arrow directions that were the strongest. To find faces in this HOG image, all we have to do is find the part of our image that looks the most similar to a known HOG pattern that was extracted from a bunch of other training faces.
- **Step 2: Posing and Projecting Faces**
  We will try to warp each picture so that the eyes and lips are always in the sample place in the image. To do this, we are going to use an algorithm called face landmark estimation. The basic idea is we will come up with 68 specific points (called landmarks) that exist on every face - the top of the chin, the outside edge of each eye, the inner edge of each eyebrow, etc. Then we will train a machine learning algorithm to be able to find these 68 specific points on any face. Now that we know were the eyes and mouth are, we will simply rotate, scale and shear the image so that the eyes and mouth are centered as best as possible. We will not do any fancy 3d warps because that would introduce distortions into the image.
- **Step 3: Encoding Faces**
  What we need is a way to extract a few basic measurements from each face. Then we could measure our unknown face the same way and find the known face with the closest measurements. For example, we might measure the size of each ear, the spacing between the eyes, the length of the nose, etc. The solution is to train a Deep Convolutional Neural Network. But instead of training the network to recognize pictures objects, we are going to train it to generate 128 measurements for each face.
- **Step 4: Finding the persons name from the encoding**
  All we have to do is find the person in our database of known people who has the closest measurements to our test image. You can do that by using any basic machine learning classification algorithm. We will use a simple linear SVM classifier, but lots of classification algorithms could work. All we need to do is train a classifier that can take in the measurements from a new test image and tells which known person is the closest match.

Beard attachment algorithm

In advance, the feature points on the beard were marked:



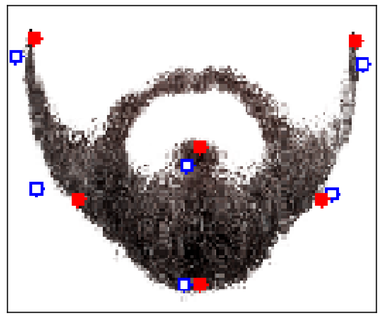
Further for the processed image there are feature points and a frame in which it is necessary to place a beard. Further, the following loss functions are optimized:

```
def f(x):
    err = 0
    coef_w = x[0]
    coef_h = x[1]
    angle  = x[2]
    for face, beardd in zip(face_fp, beard_fp):
        b_x = beardd[0] * coef_w
        b_y = beardd[1] * coef_h

        [b_x_new, b_y_new] = np.array([[np.cos(angle), np.sin(angle)],
                                       [-np.sin(angle), np.cos(angle)]]) @ np.array([b_x, b_y])
        err += np.linalg.norm(face - [b_x_new, b_y_new] + [x_opt, y_opt])
    return err
```

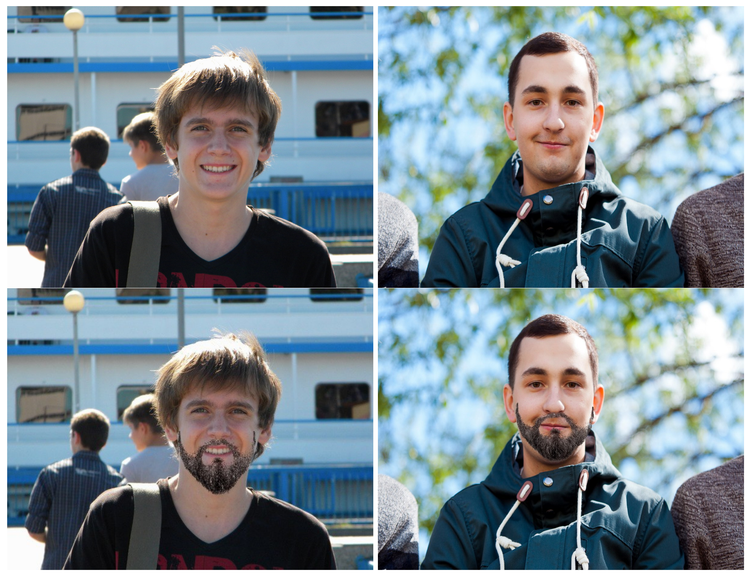
```
def f(x):
    err = 0
    for face, b in zip(face_fp, beard_fp):
        err += np.linalg.norm(face - b + x)
    return err
```

1 optimizes the beard extension (width, length), 2 finds the optimal point - the coordinates of the upper left border.



The loss function is the root of the sum of squares of distances between the corresponding feature points of the face and beard.

Results



Conclusion

The method of face and feature points recognition was studied and implemented. Then the beard attachment algorithm was proposed and implemented based on minimizing the loss function, which is the root of the sum of squares of distances between the points that form lines on the face and on the beard.

References

1. Adam Geitgey. Machine Learning is Fun! Part 4: Modern Face Recognition with Deep Learning. Medium. 2017
2. Florian Schroff, Dmitry Kalenichenko, James Philbin. FaceNet: A Unified Embedding for Face Recognition and Clustering. Google Inc. 2015
3. Code: https://colab.research.google.com/drive/1Zq8cmSkOpK11\JhGitLesttTLDKheo1Aa