Using cascaded shape regression for face alignment

Author CandNo:215816

Abstract
Research in facial landmark detection has been increasing in the last decade. With the rapid evolution of Artificial Intelligence (AI), there have been many attempts to solve this task such as, regression [1], supervised descent [2] and convolutional neural networks [3]. The basic framework for these is extracting features from an image and predicting the human face shape (or any other object of interest) based on the extracted features and the ground truth labels of the training data. However, one common problem with this approach is that some shapes, can be very varied from the average human face shape, which causes the model to make mistakes in its prediction. In this paper we show a simple approach to solve face alignment using cascaded shape regression. Our method was heavily inspired by [4]. Our model can predict the landmarks for unseen images in under 1 second, with under 3 units of distance between the ground truth and predicted shape, and after half an hour of training with 3000 images and 46 landmarks. Our model has also been shown to predict accurately for some complex data.

Key words: Facial landmarks; Human face shape; Cascaded shape regression.

1. Introduction

Detecting facial landmarks, is essential for tasks such as face tracking, face segmentation and adding graphical effects to faces. According to Cao, X., Wei, Y., Wen, F. and Sun, J., (2014, p.532)

> The purpose of face alignment is given an initial shape S, with l landmarks and a face image, predict S to try and minimise the distance from the ground truth shape $\hat{S}$
>
> $$argmin(\hat{S} - S) = 0$$
>
> The calculation above is used to evaluate the performance of the training data, however for the testing data $\hat{S}$ is unknown

The way S is estimated in our research, is by cascaded regression. The purpose of cascaded regression is to estimate the location of the face shape from previous predictions of the model, instead of predicting the exact location of the points in 1 iteration. In cascaded regression, after the first iteration the model predicts where the face is, and the rest of iterations focuses on solving detail.

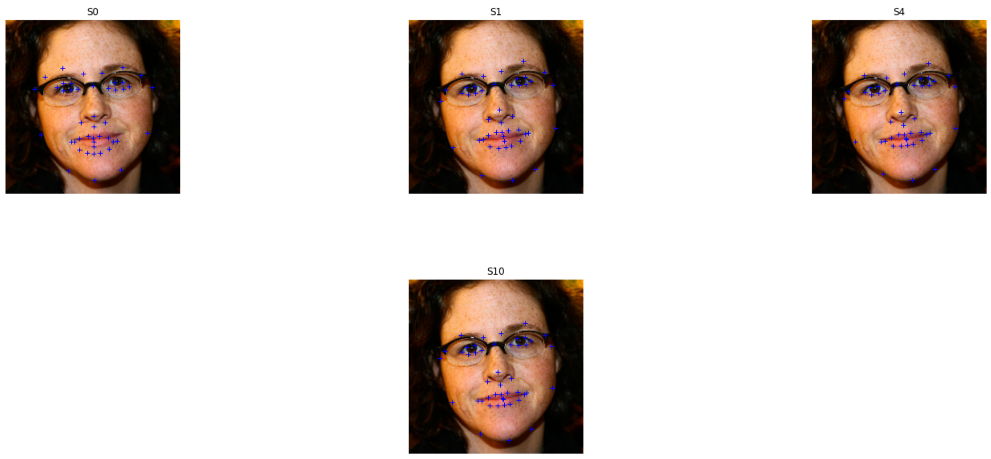


Fig. 1. Illustration of cascaded regression after 10 iterations starting from an initial shape S_0 which is the average face shape from the training data.

2. Methods

Our methods include pre processing the images, such as converting the images to grey scale and lowering the resolution of the images, to avoid over-fitting. A feature extractor function, which uses SIFT descriptors in the interesting areas of the image, and a cascaded regression algorithm, which we later applied K-fold cross-validation on to reduce over-fitting.

2.1 Pre processing

At this stage our main focus was to transform the training data, to facilitate the work for the feature extraction and regression stages. We first halved the resolution of the training images from 236x236 to 118x118 and transformed the images using Principal Component Analysis (PCA), to remove and transform the number of features. We also converted the images to grey scale, to remove the RGB dimension and equalized the grey scale image, in order to distinguish the edges of the face better.

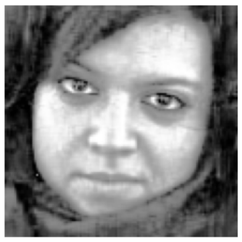


Fig. 2. Illustration of image after lowering the resolution to 118x118 and PCA.

2.2 Extracting features

Before starting to extract features from the image, we first decided to locate the interesting points of the image. Assuming that most human faces are similar, we decided to calculate the average face shape of the training data, and use the positions of the landmark l as our interesting points. We then used SIFT descriptors and placed them on the interesting points of the image, to extract the features. When choosing the range of features to extract from a key point, we decided to extract more features from the eyes, nose and mouth, because we think that most human faces will differ in the shape and sizes of these, meaning that the model may need more information when predicting these landmarks.
Below is an illustration of the code used to extract the features from the image.

```
function featureExtractor($S_0$,Image) //$S_0$ is the average face shape
1.      interestingPoints = []
2.      for landmark in $S_0$
3.              interestingPoints[landmark] = SIFTCompute(Image,landmark)
4.      return interestingPoints
```

After acquiring the descriptors from the image, we can now use these descriptors to fit our regression algorithm.

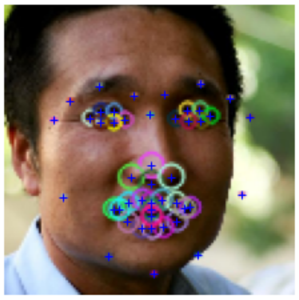


Fig. 3. Illustration of interesting points in an image, landmarks inside the face are larger than the landmarks which create the face shape.

2.3 Cascaded regression

For cascaded regression to work properly, we must restrict our model from correctly predicting the output on the first iteration. We initially attempted to use a damping factor, inspired by Cao, X., Wei, Y., Wen, F. and Sun, J., 2014. *Face alignment by explicit shape regression* [4], which involved in multiplying the prediction output for each iteration of the regressor by a random number between 0 and 1. However, we observed that it was not improving its prediction after 1 iteration.

Instead we decided to regularize our model, using Least Absolute Shrinkage and Selection Operator (LASSO), which prevented over-fitting and was getting closer to the $\hat{S}$ shape, yet, it increased the training time.

Below we provide the pseudocode for our model

```
function cascadedRegressionTrain(Images[N],$\hat{S}$[N],iterations) //N is the number of training data
1.      trainedRegressors = []
```

```
2.      for i in iterations
3.          predictedShapes = []
4.          descriptors = []
5.          target = []
6.          for n in N
7.              if i > 1
8.                  predictedShapes[n] = cascadedRegressionTest(Images[n],trainedRegressors)
9.                  descriptors[n] = featureExtractor(predictedShapes[n],Images[n])
10.                 target[n] = featureExtractor(predictedShapes[n],Images[n])
11.             else
12.                 descriptors[n] = featureExtractor($S_0$,Images[n])
13.                 target[n] = featureExtractor($S_0$,Images[n])
14.         regressors[i] = Regressor.fit(descriptors,target)
15.     return regressors
```

```
    function cascadedRegressionTest(Image,Regressors) //Regressors are obtained from training the model
1.      predictedShapes = [$S_0$] //Use average face shape as initial face shape
2.      for k in Regressors
3.          descriptor = featureExtractor(predictedShapes[k],Image)
4.          prediction = Regressors[k].predict(descriptor)
5.          newShape = predictedShapes[k] + prediction
6.          predictedShapes[k] = newShape
7.      return newShape
```

Finally we used used K-fold cross validation, in order to create a validation set for our model. We decided to shuffle the training data, and divide it into 3 folds. After dividing the data into 3 folds, we tested the accuracy of our data by calculating the average euclidean distance from each landmark and the ground truth shape for each image in the validation set.

3. Results

3.1 Face alignment

After doing k-fold cross validation, we achieved a results of 2 units of distance between the ground truth shape and predicted shape, when trained and tested on histogram equalized training images. We achieved similar results when passing different pre processed images, however we observed that the model predicts with more accuracy when passing low resolution images.
Figure 4 below shows a graph of the error rate after passing different images at different pre processing stages. We believe that the reason it performs at its best when passing low resolution images, is because it keeps most of the features, which aids the regression algorithm in having more information about the location of a landmark.

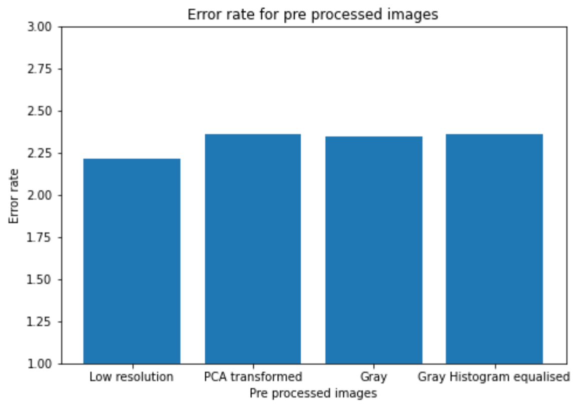


Fig. 4. Illustration of error rate, as we can clearly observe, the model can still predict the landmarks accurately for every pre processing stage. However, the model predicts at its peak when passing in low resolution images.

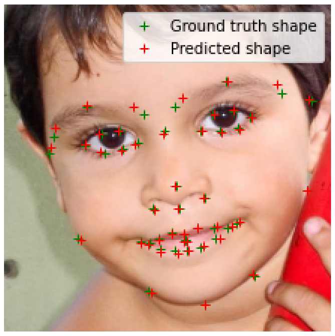


Fig. 5. Illustration of the ground truth and predicted landmarks for any pre processed image, because the error rate amongst all pre processed images is so similar, visually it seems that the predicted and ground truth points are almost aligned.

We also calculated the cumulative density for our final model (predicted on low resolution images) to see the percentage of our images with 2 -3 units of distance between the ground truth and the predicted shape.

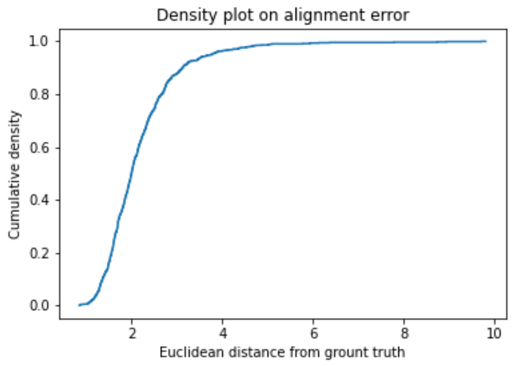


Fig. 6. We can clearly see that 80% of our validation set is 4 units away from the ground truth shape.

3.2 Face segmentation

After predicting the facial landmarks on an image, we decided to build a simple face segmentation program. Firstly we extracted the predicted coordinates for each landmark excluding eyes, nose and mouth and reorganized them to create a polygon which covered the face in the image. We then extracted the polygon (using a method from the opencv library, cv2.fillPoly()) and pasted it on an empty background. The figure below shows the pseudocode and the result of the segmentation function.

```
    function faceSegmentation(Image,predictedShape)
1.     reorganisedShape = extractOuterFaceLandmarks(predictedShape) //Get the outer face landmarks and reorganise them
2.     emptyImage = EmptyImage(Image.size) //Create an empty image the same size as the image in the parameter
3.     Image = cv2.fillPoly(reorganisedShape,emptyImage)
4.     Image = cv2.binary(Image) //We want to be able to see a segmented face, if we don't call this methods we would see a white face shape in a black background.
4.     return Image
```

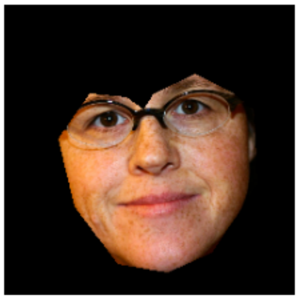


Fig. 7. Illustration of face segmentation on an example image, we connected all of the facial landmarks located on the edges of the face, and then pass a black mask on the image, removing all of the features outside of the polygon.

3.3 Graphical effects

We decided to create a function to add graphical effects to our images. Using the PIL library, we pasted a hat on to our image, and by extracting the 2 landmarks which represented the temples and by calculating the distance between them, we were able to resize and place the hat in the correct place. We also used some basic trigonometry to rotate the hat at the correct angle. Moreover, we segmented the face of the image, using our face segmentation code, and changed the skin colour to blue.
The figure below illustrates the pseudocode and the results of graphical effects.

```
    function addGraphicalEffect(Image,predictedShape,PastedImage)
1.      PastedImage = ImageTransform(predictedShape) // Resizes and rotates the image according to the face shape
2.      PIL(Image.paste(PastedImage)) //Using PIL library to paste image
4.      return Image
```

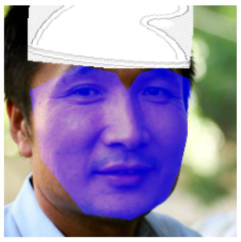


Fig. 8. Illustration of the graphical effects used on an image, calculating the distance from the landmarks located at the temples, we can place and resize the hat, and using segmentation, we can change the colour within the segmented shape.

3.4 Failure cases

Our model predicts the landmarks with an average of 2.5 units of distance between the ground truth shape and the predicted shape for our validation set, however, there are some important failure cases to discuss. The figures below illustrate 2 images which the model cannot predict the landmarks properly for.

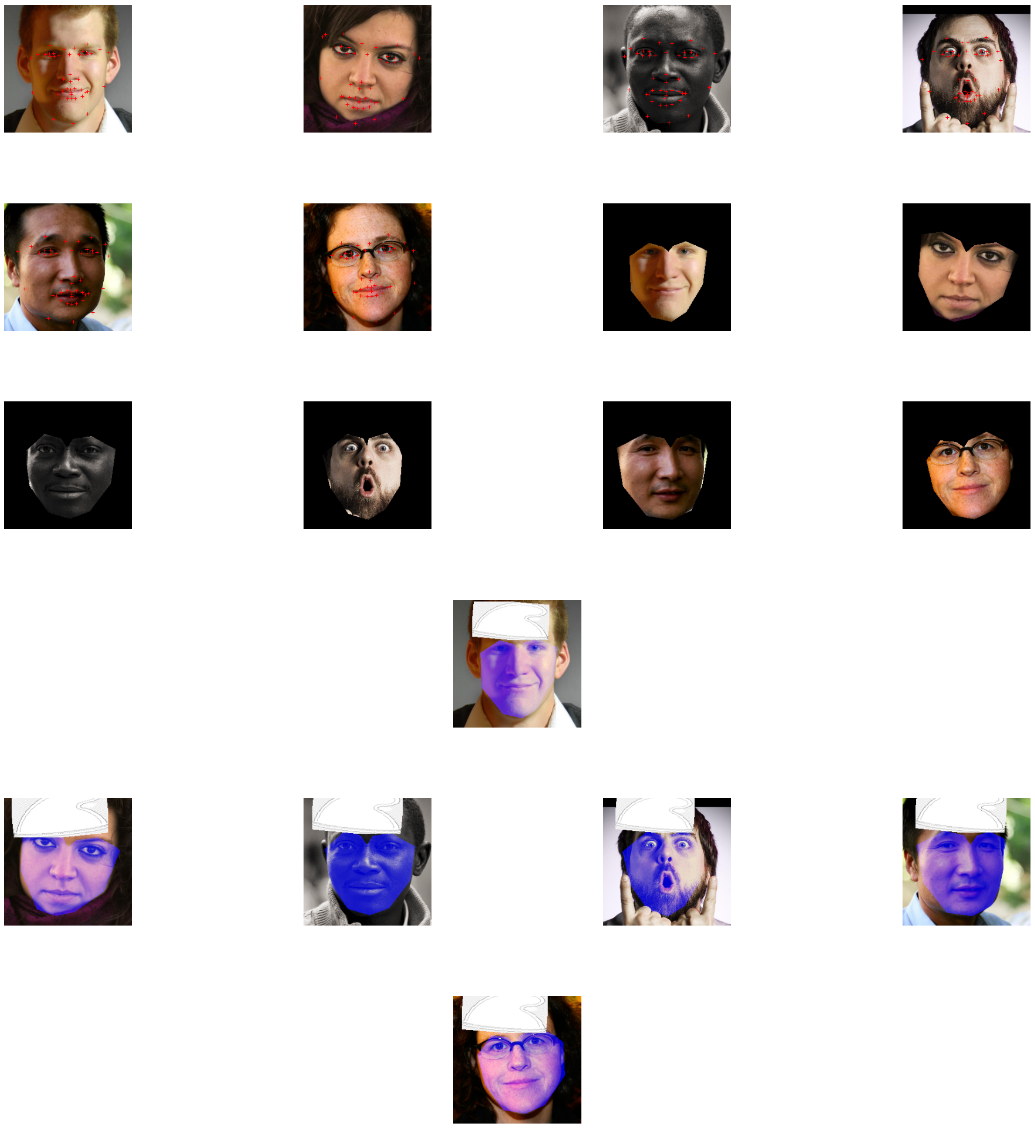
Fig. 9. 2 figures where the model miss-places the facial landmarks

For the image illustrated on the left hand side in Fig 9, we believe that the reason it cannot predict the facial landmarks for the nose and mouth is because the model is not trained on enough examples for face images with different expressions. We believe that a simple solution to this problem would be to augment the training data with more training images with different facial expressions.

For the image on the right hand side in Fig 9, we believe it is making a small alignment error on the dark side of the face because of the quality of the image. We suggest applying some pre-processing methods on the training images, such as making it the image brighter to avoid small alignment errors.

Failure cases for the face segmentation and the graphical effects depend very much on how accurate face alignment was performed, as the function on the landmarks of the face shapes.

4. Qualitative examples: example images



5. References

[1] Tzimiropoulos G (2015) Project-out cascaded regression with an application to face alignment. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, , pp 3659–3667.

[2] Xiong X, De la Torre F (2013) Supervised descent method and its applications to face alignment. *Proceedings of the IEEE conference on computer vision and pattern recognition*, , pp 532–539.

[3] Kowalski M, Naruniec J, Trzcinski T (2017) Deep alignment network: A convolutional neural network for robust face alignment. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, , pp 88–97.

[4] Cao X, Wei Y, Wen F, Sun J (2014) Face alignment by explicit shape regression. *International Journal of Computer Vision* 107(2):177–190.