

# Comparison of Facial Landmark Detection Algorithms

Haomin He, Zhan Li, Qiqi Hou  
CS545  
Fall Term, 2018

## Abstract

In our project, we compare different algorithms on task of facial landmark detection. Three methods are implemented and compared with each other: random fern, matrix learning and CNN (Convolutional Neural Network).

1. For the random fern method, we select ESR (Explicit Shape Regression) proposed in [1]. ESR is an algorithm which is based on the random fern. With the features extracted around the facial landmarks, ESR will regress to the final result using random fern as the regressor.
2. For the matrix learning method, we select the LBF (Local Binary Features) proposed in [2]. LBF is an algorithm which is based on the matrix learning. It firstly extracts the features based on the local facial marks. Then all the local features are concatenated together as the global features. Then a mapping matrix is learned based on the global feature.
3. For the CNN (Convolutional Neural Network) method, we implement the method TCDCN (Tasks-Constrained Deep Convolutional Network) proposed in [3]. TCDCN is a CNN-based algorithm. It does not only takes the facial landmark, but also takes the facial attributes into the consideration.

We conduct our experiments on the 300 Face in-the-Wild(300-W) dataset. We supposed that CNN based method achieved the best result, while it is the slowest method, ESR and LBF get slightly worse result but much faster. However, our experiments show that the TCDCN is slowest and worst algorithm. ESR is the fastest algorithm and LBF is the most accurate algorithm.

# Introduction

Facial landmark detection or face alignment is to find the semantic key points, e.g. eye, nose, mouth, in the face. It plays an important role in many face analysis tasks such as facial verification, face recognition. Recently, more and more applications are employing facial landmark detection as their backbone. Instagram, Facebook, Tiktok and Meitu help users to beautify their photos with the help of the face landmark detection.

In this work, we make a comprehensive exposition of 3 carefully selected face landmark detection algorithms by comparing the accuracy and speed between them. We select ESR for random fern, LBF to matrix learning and TCDCN for CNN .

1. ESR [1] is an algorithm which is based on the random fern. With the features extracted around the facial landmarks, ESR will regress to the final result using random fern as the regressor. This paper didn't publish their official codes and models. Zhan Li re-implements this algorithm based on (<https://github.com/soundsilence/FaceAlignment> [7], [https://github.com/GentleZhu/Face\\_Alignment](https://github.com/GentleZhu/Face_Alignment) [8]).
2. LBF [2] is an algorithm which is based on the matrix learning. It firstly extracts the features based on the local facial marks. Then all the local features are concatenated together as the global features. Then a mapping matrix is learned based on the global feature. This paper didn't publish their official codes and models. Haomin He re-implements this algorithm based on (<https://github.com/freesouls/face-alignment-at-3000fps> [9], <https://github.com/luoyetx/face-alignment-at-3000fps> [10]).

3. TCDCN [3] is a CNN-based algorithm. It will not only take the facial landmark, but also the facial attributes into the consideration. This paper didn't release their official codes and models. Qiqi Hou re-implements this algorithm in tensorflow based on (<https://github.com/zhzhanp/TCDCN-face-alignment> [11], <https://github.com/Clanatia/Tensorflow-TCDCN> [12])

We conduct our experiments on the 300 Face in-the-Wild(300-W) dataset. It is created from existing datasets including LFPW, AFW, Helen, XM2VTS, and IBUG. Our training set consisting of AFW, LFPW, and Helen has a total of 3148 images. Our testing set consisting of IBUG, LFPW, and the testing set of Helen has a total of 689 images.

While being the slowest, we hypothesise that the CNN based method will achieve the best result unlike their counterparts ESR and LBF will be much faster but will have worst result. We compare results among these three algorithms on the same datasets.

The experiments show that the LBF's result is the best but slightly slower than ESR. ESR is the fastest while it's accuracy is slightly worse than LBF. TCDCN is slowest and worst algorithm.

The report is organized as follows: we firstly introduce our work in the introduction section.

We describe the ESR, LBF and TCDCN in the methods section and in the experiments section, we introduce our dataset and implementation details. We also compared these three methods with respect to the accuracy and speed and we conclude the report in the conclusion section.

# Methods

## ESR

For the random fern methods, we employ the ESR to detect face landmarks. One of the feature their paper mentions is to use the shape constraint. The paper’s task is face alignment. Human’s face has common shape constraint which could be useful for alignment. The shape constraint can be the “relationship between landmarks”[1]. The ESR paper doesn’t use parametric shape models, it uses “all facial landmarks are regressed jointly in a vectorial output”[1]. ESR paper takes all the landmarks into consideration to get a regressed shape generated from all training shapes. ESR employs the global features from the whole image for all landmarks. This is more ‘discriminative’ than parametric local model for separated landmarks.

The main task in ESR is to learn a series of regressors  $R$ , totally  $T$  weak regressors. Noted as  $R^1, R^2, R^3, R^T$ . The relationship between facial shape  $S^t$ , facial image  $I$ , regressor  $R^t$ , are:

$$S^t = S^{t-1} + R^t(I, S^{t-1}), t = 1, \dots, T,$$

ESR’s new shape  $S^t$  is the sum of previous shape  $S^{t-1}$  and Regressor  $R^t$ .  $R^t$  is learnt by minimizing the sum of alignment errors [1]. Given  $N$  training examples with the true ground

shapes  $\left\{ (I_i, \hat{S}_i) \right\}_{i=1}^N$ , each regressor  $R^t$  is learnt by minimizing the sum of alignment errors till:

$$R^t = \underset{R}{\operatorname{argmin}} \sum_{i=1}^N \|\hat{S}_i - (S_i^{t-1} + R(I_i, S_i^{t-1}))\|$$

where alignment errors is  $\|S - \hat{S}\|_2$   $S$  is estimated shape and  $\hat{S}$  is true shape.

To make it work, ESR paper points out that it is very difficult to get a proper regressor  $R^t$ . With simple weak regressor, regressing hundreds of landmarks is poor in efficiency. To overcome this problem, ESR uses a “two-level cascaded regression”[1]. The hierarchy is that each regressor  $R^t$  is learned by a series of secondary boosted regressor  $r^k$ . The relationship between  $r^k$  and  $R^t$  is:

$$R^t = (r^1, \dots, r^k, r^K)$$

If the first level and second level regressors are learnt from same implementation, the two-level won't work well. In ESR, the difference between first level and second level regressor is that feature indexing in second level is fixed. This difference has benefits that it makes the process of learning stable and fast.

We denote the second level regressor  $r^k$  as  $r$ , called primitive regressor, similar to what the ESR paper does. ESR paper uses fern as primitive regressor  $r$ . As the paper mentions, Fern was used for regression in [4]. ESR uses a fern which has 5 features and thresholds that divide the feature space into  $2^F$  bins. Each bin  $b$  is associated with a regression output  $\delta S_b$  that minimizes the alignment error of training samples  $\Omega_b$  falling into the bin [1].

$$\delta S_b = \underset{\delta S}{\operatorname{argmin}} \sum_{i \in \Omega_b} ||\hat{S}_i - (S_i + \delta S)||$$

, where  $S_i$  denotes the estimated shape. The solution is the mean of shape differences :

$$\delta S_b = \frac{\sum_{i \in \Omega_b} (\hat{S}_i - S_i)}{\Omega_b}$$

According to the paper, a shrinkage is applied to overcome the over-fitting:

$$\delta S_b = \frac{1}{1 + \beta/|\Omega_b|} \frac{\sum_{i \in \Omega_b} (\hat{S}_i - S_i)}{|\Omega_b|}$$

where  $\beta$  is a free shrinkage parameter. If  $|\Omega_b|$  is large enough (there are sufficient training data),

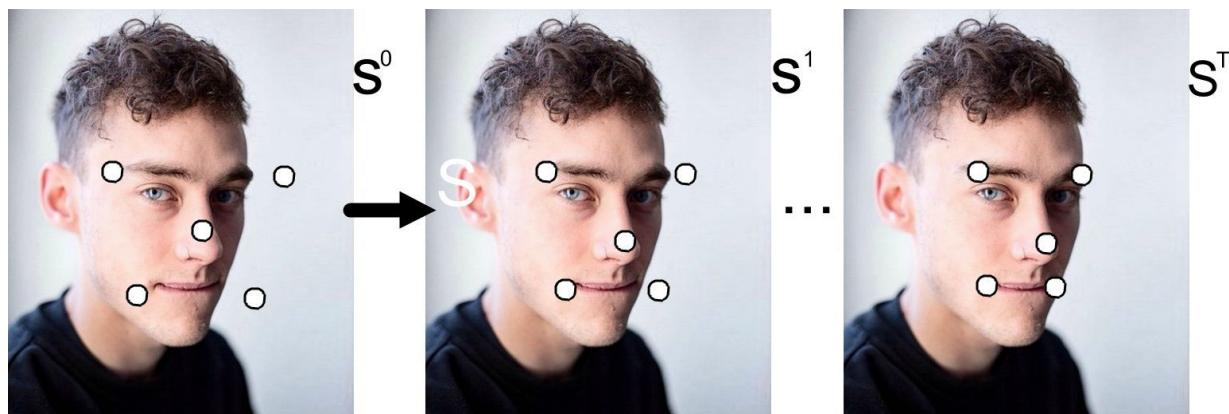
$\beta$  can be neglected. Otherwise, the output  $S_b$  is reduced.

ESR achieves non-parametric shape constraint. The final shape  $S$  can be expressed as the initial shape  $S^0$  plus the linear combination of all training shapes [1]:

$$S = S^0 + \sum_{i=1}^N w_i \hat{S}_i$$

If  $S^0$  satisfies the constraint, the estimated shape  $S$  we get from the linear combination of training shapes should also satisfies the shape constraints. Although we still don't know the what the exact shape constraint is. Our estimated shape  $S$  must satisfy the constraints generated from training data. That is “adaptively determined during the learning” [1].

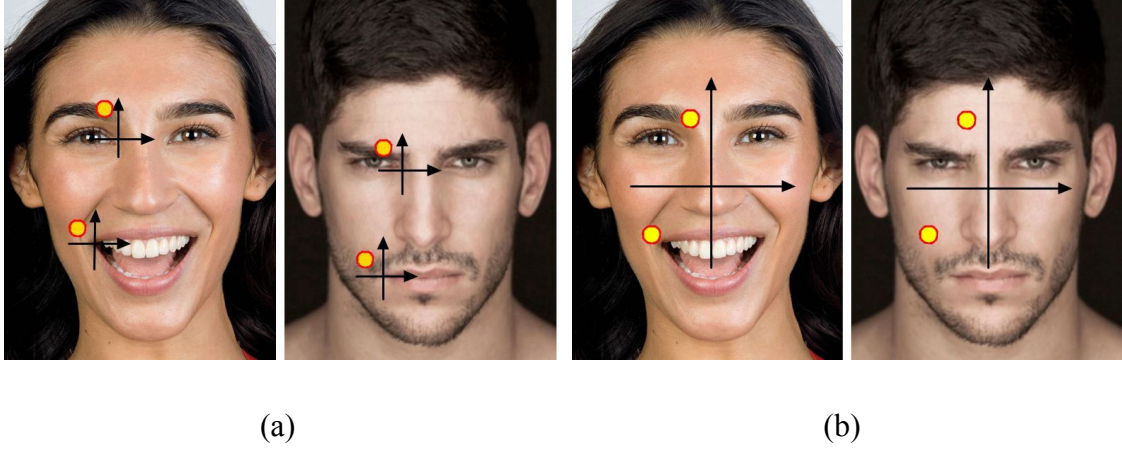
Principal Component Analysis (PCA) is what the paper performs to show the adaptive shape constraint. PCA is a statistical method under the broad title of factor analysis [5].



**Figure 1.** Face shape regression

Figure 1 shows how the shape constraint is learned from  $S^0$  to  $S^T$ . The shape  $S$  is gradually fitting the true face shape more closely.

One important thing that makes ESR paper excellent is the use of pixel-difference features. For example, the intensity difference of two pixels in the image, where it improves the speed of computing features to get regression. How to index the pixel in the images is an important thing to consider. The indexing way also influence the speed of converging for regression. The paper claims that pixel indexed by the same local coordinates have the same semantic meaning [1].



**Figure 2.** Pixels with same local coordinates (a) vs pixels with same global coordinates (b)

Figure 2 shows the difference. The part (a) is using local coordinates while part(b) is using global coordinates. We can clearly find that in the right part two pixel don't share the same feature with same global coordinates.

## LBF

For the matrix learning methods, we consult with paper "Face Alignment at 3000 FPS via Regressing Local Binary Features (LBF)" [2]. We employ the LBF to detect face landmarks.

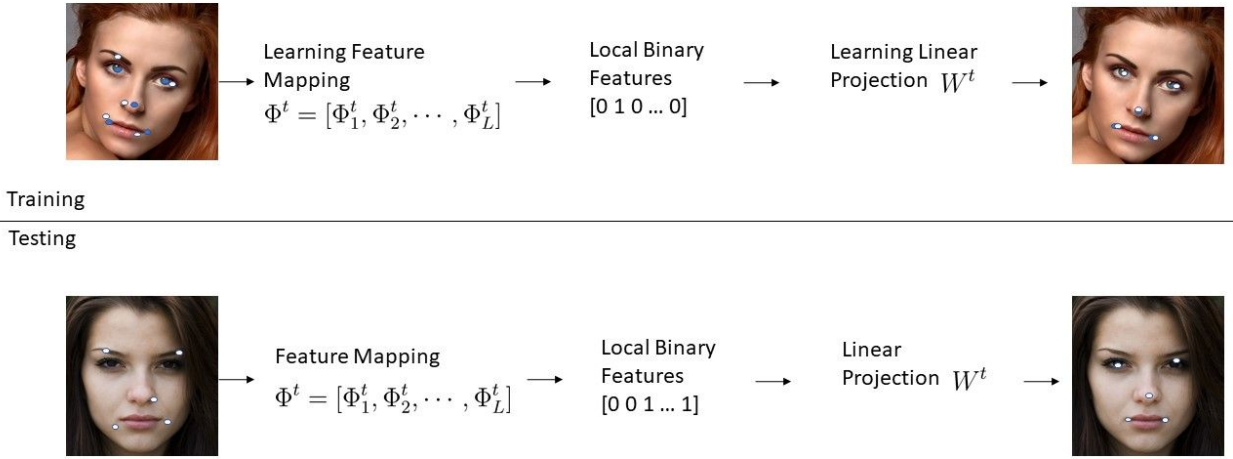
The paper introduces an efficient discriminative shape regression approach for face alignment, which contains a set of local binary features and a locality principle for learning those features.

This shape regression approach begins with an initial shape  $S^0$ ,  $S$  is progressively refined by estimating a shape increment  $\Delta S$  stage-by-stage. A shape increment  $\Delta S^t$  at stage  $t$  is  $\Delta S^t = W^t \Phi^t(I, S^{t-1})$  where  $I$  is the input image,  $S^{t-1}$  is the shape from the previous stage,  $\Phi^t$  is a feature mapping function (depends on both  $I$  and  $S^{t-1}$ ), and  $W^t$  is a linear regression matrix. The regression goes to the next stage by adding  $\Delta S^t$  to  $S^{t-1}$  [2].



The feature mapping function  $\Phi^t$  is designed by learning with a locality principle.  $\Phi^t$  is decomposed into a set of independent local feature mapping functions,  $\Phi^t = [\Phi_1^t, \Phi_2^t, \dots, \Phi_L^t]$ , where  $L$  is the number of landmarks on the facial image. The principle is: learn intrinsic features to encode the local texture for each landmark independently in the corresponding local region, then perform joint regression to incorporate the shape context.

- Estimated Shape  $S^{t-1}$
- Ground Truth Shape  $\hat{S}$



**Figure 3.** Overview of Local Binary Features approach.

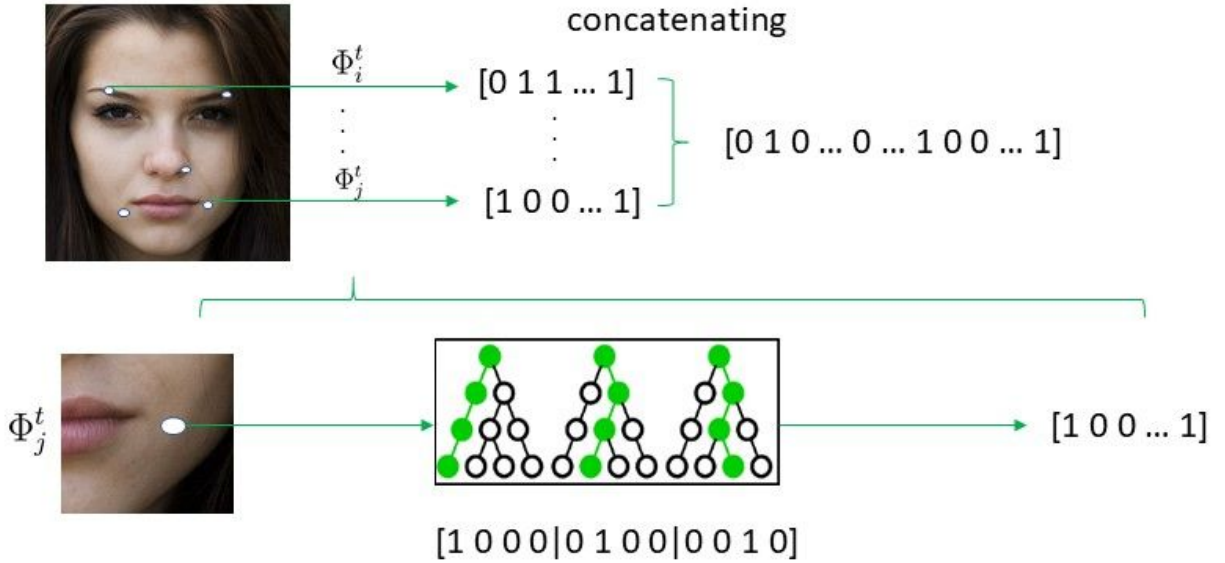
Figure 3 shows the overview of LBF: in the training phase, we first learn a local feature mapping function  $\Phi^t(I_i, S_i^{t-1})$  to generate local binary features for each landmark, and concatenate all local features to get  $\Phi^t$ . Then we learn linear regression matrix projection  $W^t$  with given features and target shape increments  $\left\{ \Delta \hat{S}_i^t = \hat{S}_i - S_i^{t-1} \right\}$ . This learning process is repeated

stage-by-stage. In the testing phase, the shape increment is directly predicted and applied to update the current estimated shape.

In order to learn local binary features  $\Phi^t$ , we need to learn each local feature mapping function independently by using standard regression random forest,  $\Phi^t = [\Phi_1^t, \Phi_2^t, \dots, \Phi_L^t]$ . The regression target for learning  $\Phi_l^t$  is the ground truth shape increment  $\Delta \hat{S}^t$ :

$$\min_{w^t, \phi_l^t} \sum_{i=1} \left\| \pi_l \circ \Delta \hat{S}_i^t - w_l^t \phi_l^t(I_i, S_i^{t-1}) \right\|_2^2,$$

where  $i$  iterates over all training samples, operator  $\pi_l$  extracts two elements  $(2l-1, 2l)$  from the vector  $\Delta \hat{S}_i$ , and  $\pi_l \circ \Delta \hat{S}_i$  is the ground truth 2D-offset of  $l$ th landmark in  $i$ th training sample [2].



**Figure 4.** Local binary features.

Figure 4 shows how local binary features are built. First, the local feature mapping function  $\Phi_l^t$  encodes the corresponding local region into a binary feature. Second, all local binary features are concatenated to form high dimensional binary features. Third, use random forest as the local mapping function. As the paper indicates: The split nodes in the trees are trained using the pixel-difference feature. LBF samples pixel features in a local region around the landmark that is estimated. To train each split node, we test 500 randomly sampled features and pick the feature that gives rise to maximum variance reduction. After training, each leaf node stores a 2D offset vector that is the average of all the training samples in the leaf. In the training, the optimal region size is estimated in each stage via cross validation. Each extracted binary feature indicates whether the input image contains some local patterns or not.

The process of extracting local binary features: a sample traverses the trees until it reaches one leaf node for each tree. The output of the random forest is the summation of the outputs stored in these leaf nodes. Supposing the total number of leaf nodes is  $D$ , the output can be rewritten as:

$w_l^t \Phi_l^t(I_i, S_i^{t-1})$ , where  $w_l^t$  is a 2-by- $D$  matrix in which each column is the 2D vector stored in the corresponding leaf node, and  $\Phi_l^t$  is a  $D$ -dimensional binary vector [2] (as shown in figure 4).

For each dimension in  $\Phi_l^t$ , its value is one if the test sample reaches the corresponding leaf node, and its value is zero if the test sample doesn't reach the corresponding leaf node. As you can see from the figure above,  $\Phi_l^t$  is a sparse binary vector, so we use a dual coordinate descent method [6] to deal with such a large-scale sparse linear system. We call  $\Phi_l^t$  local binary features.

Next, we concatenate all the binary features  $\Phi_l^t$  into a global feature mapping function  $\Phi^t$  and minimize the objective function

$$\min_{W^t} \sum_{i=1}^N \left\| \Delta \hat{S}_i^t - W^t \Phi^t(I_i, S_i^{t-1}) \right\|_2^2 + \lambda \|W^t\|_2^2$$

in order to learn a global linear projection  $W^t$ . In the objective function, the first term is the regression target, the second term is a L2 regularization on  $W^t$ , and  $\lambda$  controls the regularization strength [2]. With regularization, we can avoid overfitting. The local output we get from random forest is noisy because of possible insufficient number of training samples in a leaf node.

However, the global linear regression  $W^t$  can effectively enforce a global shape constraint and also reduce local errors caused by ambiguous local appearance at the same time.

## TCDCN

For the CNN methods, we employ the TCDCN to detect face landmarks [3]. Instead of treating the face alignment task as a single and independent problem, TCDCN treats the problem as a multi-task learning. It optimizes facial landmark detection together with related tasks, including the head pose estimation and facial attribute inference.



**Figure 5.** Different attributes of face can reflect the different distribution of the facial landmarks.

Suppose we have a total of  $T$  tasks and the training data for the  $t$ -th task are denoted as  $(x_i^t, y_i^t)$ , where  $t = \{1, \dots, T\}, i = \{1, \dots, N\}$ . Our aim is to optimize the main task  $r$ , which is facial landmark detection, with the related tasks  $a \in A$ . We can formulate the problem as follows:

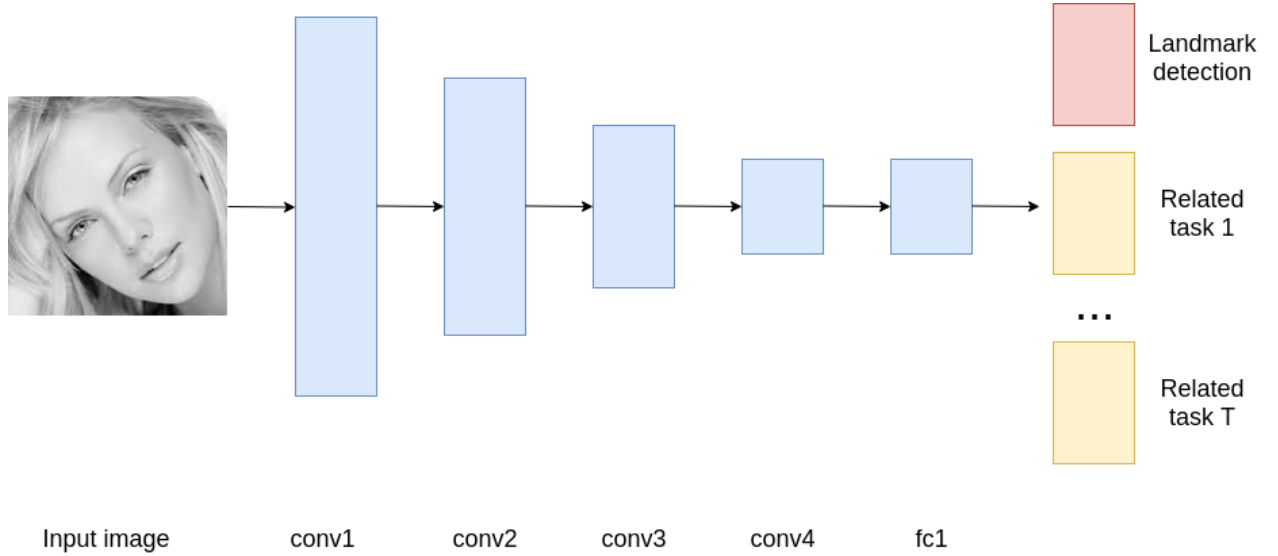
$$\arg \min_{W^r, \{W^a\}_{a \in A}} \sum_{i=1}^N l^r(y_i^r, f(x_i, W^r)) + \sum_{i=1}^N \sum_{a \in A} \lambda^a l^a(y_i^a, f(x_i, W^a)),$$

where  $\lambda^a$  indicates the weights of the  $a$ -th task's error,  $W^r$  indicates the model's of the task  $r$ ,  $W^a$  indicates the model of the related task  $a$ ,  $y^r$  indicates the label of the  $r$ -task,  $y^a$  indicates the label of the  $a$ -task.

In TCDCN, given a face image  $x^0$ , it will projects it to a higher level representation gradually by learning a sequence of non-linear mappings.

$$x^0 \xrightarrow{\sigma((W^1)^T x^0)} x^1 \xrightarrow{\sigma((W^2)^T x^1)} x^2 \xrightarrow{\dots} \xrightarrow{\sigma((W^l)^T x^{l-1})} x^l,$$

where  $\sigma$  and  $W$  indicate the non-linear activation function and the filters needed to be learned in the layer  $l$ . Note that the features are shared representation between the main task  $r$  and the related tasks  $A$ . So in TCDCN, the main task learns the shared space the latter optimizes the tasks with respect to this space. Then TCDCN propagated back the errors of the tasks to refine the space. It repeats this learning iteration until it converges.



**Figure 6.** Network structure of TCDCN

As shown in the figure 6, the TCDCN is composed of four convolutional layers and a fully connected layer. Every convolutional layers is followed by a max pooling layer. The input images are resized and converted to 40\*40 grayscale face image. Note that in the TCDCN, the

filter weights are not spatially shared, which means that the filters are different with others in the different locations. Absolute tangent function is used as the activation function for the activation layer. TCDCN uses the max-pooling on non-overlap regions of the feature map. For the fully connected layer, it contains 100 filters. The multiple tasks share the same feature vector produced by the fully connected layer.

In the TCDCN, the author also proposed a task-wise early stopping learning approach to avoid the overfitting in the related tasks and thus harming the main task. It can also accelerate the training process of the network. The main idea is that certain auxiliary task is no longer beneficial to the main task after they reach their peak performance, its learning process thus will be stopped.

Let  $E_{val}^a$  and  $E_{tr}^a$  indicate the loss function of task a on the validation set and training set respectively. The task will be stopped if its measure exceeds a threshold  $\alpha$  as below

$$\frac{k * med_{j=t-k}^t E_{tr}^a(j)}{\sum_{j=t-k}^t E_{tr}^a(j) - k * med_{j=t-k}^t E_{tr}^a(j)} \cdot \frac{E_{val}^a(t) - \min_{j=1, \dots, t} E_{tr}^a(j)}{\lambda^a \cdot \min_{j=1, \dots, t} E_{tr}^a(j)} > \alpha,$$

where t indicates the current iteration and k controls a training strip of the length k. This strategy means that if the training error of the first k steps drops quickly, which means that task is still valuable. We still need to keep the task. Otherwise, the task will be more likely to be closed. It can also compare the generalization error with the training error to get a better and stabilized model. In the TCDCN, they employs the stochastic gradient descent to update the weights of the tasks and the filters of the network.

For the prediction, we firstly need to project the test image  $x^0$  to the shared space to get  $x^l$ . Then we predict the landmark positions and the results of the auxiliary tasks. This is efficient.

## Experiments

### Dataset

We conduct our experiment on the 300-W dataset. 300-W (68 landmarks) is short for 300 Faces in-the Wild. It is created from existing datasets, including LFPW, AFW, Helen , XM2VTS, and IBUG. We split their training data into two parts for our own training and testing. Our training set consists of AFW, the training sets of LFPW, and the training sets of Helen, with 3148 images in total. Our testing set consists of IBUG, the testing set of LFPW, and the testing set of Helen, with 689 images in total. We do not use images from XM2VTS. Note that we only takes the 5 landmarks into consideration for the fair comparison. For the TCDCN, we used MFCL dataset to pretrain the model and then finetune the network in the 300-W dataset.

### Evaluation matrix

We use the inter-pupil distance normalized landmark error. For each dataset we report the error averaged over all landmarks and images. Note that the error is represented as a percentage of the pupil-distance, and we drop the notation % in the reported results for clarity.



## Implementations

For the ESR, we implement the algorithm in the Matlab. We modified a public source code and fit it into our dataset. We use the same settings for the model ( $T=10$ ,  $K=500$ ) as the paper, which indicates that we have 10 stages and 500 small local regressors. Every small regressors, we use 5 level random fern. However, for the fair comparison, in our experiment, we use the mean shape as the initial shape, which is different from the random initial set of the paper.

For the LBF, we implement the algorithm in the C++. We modified the codes to our dataset. Following the paper instruction, we use  $T=5$ ,  $N=1200$ ,  $D=7$ , which indicates that we have 5 stages. Every stage there are 1200 7-depth random tree to extract the binary features. For the fair comparison, we use the mean shape as the initial shape.

For the TCDCN, we implement the algorithm in the Tensorflow. We modified the released codes to test on our dataset. We follow the paper to set up the parameters of the models. The input size of the network is  $40*40$ . The convolutional layers are the local convolutional layers. However, we uses the SGD with TF default settings in our model, which is different from the paper. To train the model of the TCDCN, we have to use some data from their official site because it contains attributes information. Then we finetune the network on the 300-W dataset. We run 30,000 iterations in the total. Our mini batch size is set to 50. We use NVIDIA 1080-Ti to train our model.

## Comparison and Discussion

As shown in Table 1, we firstly compare the training time of each of the algorithm. We can find that ESR is the fastest, LBF is slower than ESR. TCDCN is the slowest.

**Table 1.** Training time comparison between ESR, LBF TCDCN

Method	Training time
ESR	31 mins
LBF	60 mins
TCDCN	84 mins

ESR only contains the training process of the random ferns, while the LBF contains two sections: the training process of the random forest and the training process of the matrix learning. Furthermore, there are 5,000 ( $10 * 500$ ) random ferns but the LBF contains 6,000( $5 * 1200$ ) random trees. By the way, the depth of the random fern (5) in ESR is shallower than the random trees (7) in LBF.

TCDCN is the slowest because it makes use of CNN of its backbone. The complexity of TCDCN is much higher than the ESR and LBF. ESR employs the shape index features and LBF uses the local binary features. Both features are based on the pixels and no need for the extra-processing and computation. However, TCDCN uses the CNN as the feature extractor to get its feature. The complexity is very high for the convolutional layers. The local convolutional layers in the TCDCN make the situation even worse because the convolutional kernels are different at every position, which make it hard to speed up based on the BLAS libraries and CUDNN. Although we use the fastest device (NVIDIA-1080 TI), the training process still costs a lot of time.

**Table 2.** Accuracy and speed comparison between ESR, LBF TCDCN

Method	RMSE	Time per image
ESR	6.3	3.14 ms
LBF	5.7	4.60 ms
TCDCN	13.5	52.32 ms

In the Table 2, we compare the accuracy and speed for the testing. LBF achieves the lowest error and ESR’s error is relatively lower. TCDCN’s result is the worst. For the speed ESR is the fastest algorithm, LBF is slower, and TCDCN is the slowest.

For the accuracy, we can find that LBF’s result is better than ESR. We believe that LBF takes the advantages of the random forest and the matrix learning which help it achieve a higher performance. LBF uses the random forest to extract the feature and then feed it into the matrix learning while the ESR only takes the random fern’s regression result. The result of TCDCN really surprised us because we supposed that CNN would get the highest performance. Maybe the following reasons lead to the result.

- (1) TCDCN only takes 40\*40 image as input while LBF and ESR take a bigger input images.  
For example, most ESR’s input images are higher than 300\*300. The low resolution of the input image hurts the result.
- (2) TCDCN is trained on the MDFL dataset. We finetune it to the 300-W dataset. Different dataset may also hurt the performance.
- (3) We don’t implement all the details of the paper, such as the early-stop method, which may also lead to the bad result of the TCDCN.

For the speed, we supposed that the LBF should be slightly faster than ESR. However, we find that ESR is faster than LBF. There are some reasons that may lead to the result

- (1) LBF contains more computation steps than ESR. LBF has 6,000( $5 * 1200$ ) depth-7 random trees while ESR only has 5,000 ( $10 * 500$ ) depth-5 random ferns. LBF needs to feed the features to the matrix, which ends up with more calculation.
- (2) Different implementation platforms. We implement the LBF in the C++ while we implement the ESR in the Matlab. The matrix calculation in the Matlab is much faster than the ones in C++ if we don't use BLAS libraries.

TCDCN is the slowest because that it is based on the CNN which needs much more calculations.

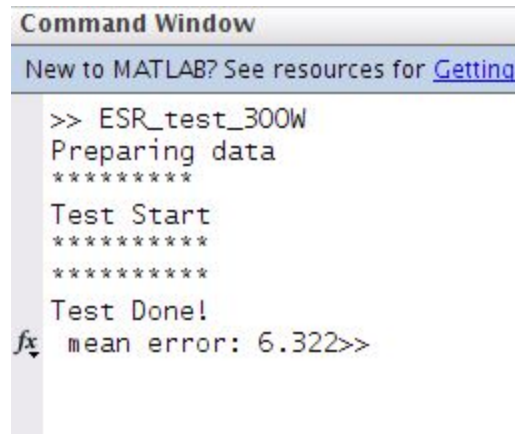
## **Conclusion and Future Work**

In this report, we compare three different facial landmark detection algorithms: ESR, LBF and TCDCN. We find that ESR based on the random fern is the fastest algorithm, LBF based on the matrix learning is slower and TCDCN based on CNN is the slowest. LBF achieve the best results, ESR's result is slightly worse and TCDCN is worst. In the future, we will improve our implementations. All the current result is worse than the result claimed on their papers.

# Reference

- [1] Cao, Xudong, et al. "Face Alignment by Explicit Shape Regression." *International Journal of Computer Vision* 107.2 (2014): 177-190.
- [2] Ren, Shaoqing, et al. "Face Alignment at 3000 FPS via Regressing Local Binary Features." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014.
- [3] Zhang, Zhanpeng, et al. "Facial Landmark Detection by Deep Multi-task Learning." *European Conference on Computer Vision*. Springer, Cham, 2014.
- [4] D. Horn. *GPU Gems 2*, chapter Stream Reduction Operations for GPGPU Applications, pages 621–636. Addison Wesley, 2005.
- [5] Face Recognition using Principle Component Analysis Kyungnam Kim Department of Computer Science University of Maryland, College Park MD 20742, USA
- [6] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *The Journal of Machine Learning Research*, 2008.
- [7] Soundsilence. "Soundsilence/FaceAlignment." GitHub, 23 May 2015, [github.com/soundsilence/FaceAlignment](https://github.com/soundsilence/FaceAlignment).
- [8] GentleZhu. "GentleZhu/Face\_Alignment." GitHub, 21 Feb. 2018, [github.com/GentleZhu/Face\\_Alignment](https://github.com/GentleZhu/Face_Alignment).
- [9] Freesouls. "Freesouls/Face-Alignment-at-3000fps." GitHub, [github.com/freesouls/face-alignment-at-3000fps](https://github.com/freesouls/face-alignment-at-3000fps).
- [10] Luoyetx. "Luoyetx/Face-Alignment-at-3000fps." GitHub, 18 Oct. 2016, [github.com/luoyetx/face-alignment-at-3000fps](https://github.com/luoyetx/face-alignment-at-3000fps).
- [11] Zhzhanp. "Zhzhanp/TCDCN-Face-Alignment." GitHub, 25 Oct. 2016, [github.com/zhzhanp/TCDCN-face-alignment](https://github.com/zhzhanp/TCDCN-face-alignment).
- [12] Clanatia. "Clanatia/Tensorflow-TCDCN." GitHub, [github.com/Clanatia/Tensorflow-TCDCN](https://github.com/Clanatia/Tensorflow-TCDCN).

# Appendix

A screenshot of the MATLAB Command Window. The title bar reads "Command Window". Below the title bar, there is a link: "New to MATLAB? See resources for [Getting](#)". The command prompt shows the execution of the script "ESR\_test\_300W". The output of the script is as follows: "Preparing data", followed by a line of seven asterisks "\*\*\*\*\*", then "Test Start", another line of seven asterisks "\*\*\*\*\*", and a third line of seven asterisks "\*\*\*\*\*". Finally, it displays "Test Done!" and "mean error: 6.322>>". A small icon with the letter "fx" is visible to the left of the final line of output.

```
Command Window
New to MATLAB? See resources for Getting

>> ESR_test_300W
Preparing data
*****
Test Start
*****
*****
Test Done!
fx mean error: 6.322>>
```

**Appendix 1.** ESR Testing Screenshot

```
optimization finished, #iter = 9
Objective value = -0.007125
nSV = 36360
[12/06/18 - 21:20:09] train 67th landmark

optimization finished, #iter = 9
Objective value = -0.009584
nSV = 36360

optimization finished, #iter = 9
Objective value = -0.002842
nSV = 36360

optimization finished, #iter = 9
Objective value = -0.002511
nSV = 36360

optimization finished, #iter = 9
Objective value = -0.003022
nSV = 36360

optimization finished, #iter = 9
Objective value = -0.012536
nSV = 36360

optimization finished, #iter = 9
Objective value = -0.003168
nSV = 36360

optimization finished, #iter = 9
Objective value = -0.002282
nSV = 36360

optimization finished, #iter = 9
Objective value = -0.006907
nSV = 36360
[12/06/18 - 21:20:13] end of train global regression of 4th stage, costs 68.3999 s
[12/06/18 - 21:20:35] Train 4th stage Done with Error = 0.084188
[12/06/18 - 21:20:35] Dump model of stage 4
[12/06/18 - 21:20:35] Train Model Down, cost 3618.1293 s
[12/06/18 - 21:20:35] Write 0th stage
[12/06/18 - 21:20:35] Write 1th stage
[12/06/18 - 21:20:35] Write 2th stage
[12/06/18 - 21:20:35] Write 3th stage
[12/06/18 - 21:20:35] Write 4th stage
```

## Appendix 2. LBF Testing Screenshot

```
landmark error : 1342531.0  
Hole error : nan  
idle: 59499 ,err : nan  
landmark error : 1319252.0  
Hole error : nan  
idle: 59599 ,err : nan  
landmark error : 1411683.1  
Hole error : nan  
idle: 59699 ,err : nan  
landmark error : 1352043.0  
Hole error : nan  
idle: 59799 ,err : nan  
landmark error : 1376927.5  
Hole error : nan  
idle: 59899 ,err : nan  
landmark error : 1351985.5
```

**Appendix 3.** TCDCN Testing Screenshot