Facial keypoints detection using Neural Network

Shutong Zhang
Stanford University
zhangst@cs.stanford.edu

Chenyue Meng
Stanford University
chenyue@cs.stanford.edu

Abstract

Nowadays, facial keypoints detection has become a very popular topic and its applications include Snapchat, How old are you, have attracted a large number of users. The objective of facial keypoints detection is to find the facial keypoints in a given face, which is very challenging due to very different facial features from person to person. The idea of deep learning has been applied to this problem, such as neural network and cascaded neural network. And the results of these structures are significantly better than state-of-the-art methods, like feature extraction and dimension reduction algorithms.

In our project, we would like to locate the keypoints in a given image using deep architectures to not only obtain lower loss for the detection task but also accelerate the training and testing process for real-world applications. We have constructed two basic neural network structures, one hidden layer neural network and convolutional neural network as our baselines. And we have proposed an approach to better locate the coordinates of facial keypoints with introduced features other than raw input. Specifically, we use a block of pretrained Inception Model to extract intermediate features and using different deep structures to compute the final output vector. The experiments results have shown the effectiveness of deep structures for facial keypoints detection tasks, and using the pretrained Inception Model has slightly improved the performance of detection compared to baseline methods.

1. Introduction

With the fast development in computer vision area, more and more research works and industry applications are focused on facial keypoints detection. Detecting keypoints in a given face image would act as a fundamental part for many applications, including facial expression classification, facial alignment, tracking faces in videos and also applications for medical diagnosis. Thus, how to detect facial keypoints both fast and accurately to use it as a preprocessing procedure has become a big challenge.

There are two main challenges for facial keypoints detection, one is that facial features have great variance between different people and different external factors, the other is that we have to reduce time complexity to achieve real-time keypoints detection.

1. **Facial features are very different**

   Facial features are very different from people to people, which result in the difficulty of training the regression model. And similar as object detection task, under different illumination conditions, positions, sizes, detecting facial keypoints would be very challenging.

2. **Detecting keypoints has to be fast**

   Detecting keypoints in face images is one of the first few steps for many applications as we mentioned before, for example analyzing facial expression or detecting faces in images and videos. Moreover if we would like to fit the detection procedure to a real-time mobile app, we have to complete the detection within seconds [1]. Therefore, the computation complexity of keypoints detection have to be lower than traditional image classification tasks. Unlike other image classification task that we only evaluate the accuracy, we have to focus on the time that we spend on the task [2]. When using traditional deep structures the training and testing process tend to be much slower, which is not we want for facial keypoints detection.

There are two main kinds of state-of-arts methods for detecting facial keypoints, the first type uses feature extraction algorithms, using Gabor features [10] and other features, and the other one is using probability graphic model [9] [5] to specify the relationship between pixels and its neighbors. With the development of deep learning, many deep structures designed for this task have been explored recently. Different deep structures have been proposed for facial keypoints detection, like deep convolutional cascade network [7], which can better deal with two main challenges we mentioned before.

Our objective is to locate 15 sets of facial keypoints when given a raw facial image. The input is a set of 96×96 raw facial images with only the grayscale pixel values, and the output is a 30-dimentional vector, which indicate the (x, y) coordinates of 15 sets of facial keypoints.

In our project, we are going to use deep structures for facial keypoints detection, which can learn well from different faces and overcome the variance between faces of different person or of different conditions to a great extent. Two widely-used model, One Hidden Layer Neural Network and Convolutional Neural Network, are designed as baselines in our project. Most importantly, we have used the pretrained Inception Model[8] to explore some techniques to reduce the computation complexity for detecting facial keypoints. With sparsely connected layers and different number of different size of filters, Inception Model have the ability to better capture local features and reduce computation complexity at the same time. When compared to our baseline models, we can see a great improvement when using pretrained inception models to predict the location of facial keypoints.

The contributions of this paper includes:

1. Explore the performance of different deep structures on the task of detecting facial keypoints, and evaluate the effectiveness using MSE loss.

2. Use the recently proposed Inception Model when detecting facial keypoints. Although the Inception Model is trained on image classification task on ImageNet, the experiments results have shown the intermediate features can adapted to facial keypoints detection task well.

3. Conduct experiments on real-world datasets from kaggle challenge, and our model can be easily extended to other facial detection task.

The remainder of this paper is organized as follows. In Section 2, we provide a literature review for the field of facial keypoints detection, where the state-of-methods includes feature extraction algorithms and different deep structures. In Section 3, we describe our baseline methods and Inception Model in detail. The dataset from kaggle challenge we use for this paper is analyzed in Section 4. The experimental results are summarized in Section 5, where we can see the effectiveness of adapting the Inception Model to the task of detecting facial keypoints compared of traditional deep structures. We conclude our work in Section 6.

2. Related Work

Traditional methods have explored feature extraction strategies includes texture-based and shape-based features and different types of graphic models to detect facial keypoints.

[10] proposed a methods that uses Gabor wavelet feature based boosted classifiers which can detect 20 different facial keypoints with limited head rotation and different illumination conditions. [4] also use Gabor features for detecting facial keypoints. Using a sample log Gabor response of a facial point, the authors have shown that the locations on the test facial images that are similar to the sample facial points can be detected. [6] focused on keypoints detection for Textured 3D Face Recognition. Features are extracted by fitting a surface to the neighborhood of a keypoint and sampling it on a uniform grid. And then PCA are used for dimensional reduction and then projected and matched from a probe and gallery face.

Other methods have designed probability graphic models to capture the relationship between different pixels and features to detect facial keypoints. Using Markov Random Fields, [9] exploits the constellations that facial points can form. Also the authors use boosted regression to learn the mappings from the pixels appearance of the area around the keypoints and the location of the keypoints. These models have reached high performance on aligned faces but still need improvements for faces under different environment conditions. [5] have proposed a model to detect the target location by local evidence aggregation. By aggregating the estimates obtained from stochastically selected local appearance information into a single robust prediction other than focus only on the target locaion only, the proposed model have solved the regression problem from a new prospective.

Recent works have focused on deep architectures for this detection task since these structures can better capture the high-level features of a image, in our problem a given face. [7] proposed a three-level carefully designed convolutional networks, which at first capture the global high-level features, and then refine the initiliation to locate the positions of keypoints. On the other hand, [3] used pretrained DBN on surrounding feed-forward neural network with linear Gaussian output layer to detect facial keypoints.

Different from our work, the deep structures mentioned before have not focused on the time complexity but only on the correctness of the detected keypoints. But using Inception Model we can train a much more complexity model in less time, thus the detection process would be faster than the traditional deep structures considering the number of parameters, which can be adapted to this task.

3. Approach

In this section, we'll first formally formulate the keypoints detection problem and introduce performance metrics. To solve this problem, we build two models as baselines, which are realized based on simple neural network and convolutional neural network respectively. Then as the major part of this section, the Inception Model [8] will be introduced and analyzed in detail. Especially, we'll demonstrate its impressive advantages over traditional cnn models, and apply this model to extract low-level features as input of following networks.

3.1. Problem Formulation

We have split our dataset into three parts X_{train}, X_{val}, X_{test} and we define the ground truth corresponding to these three sets to be y_{train}, y_{val}, y_{test}.

Given a training dataset X_{train} we have

$$X_{train} = \{X_{train}^{(1)}, \cdots, X_{train}^{(n_{train})}\}$$

$$y_{train} = \{y_{train}^{(1)}, \cdots, y_{train}^{(n_{train})}\}$$

where $X_{train}^{(i)}$ represents a given image and $y_{train}^{(i)}$ represents the keypoints vector for $X_{train}^{(i)}$.

We will train our model M on (X_{train}, y_{train}) and evaluate M on (X_{val}, y_{val}) to tune parameters. After the training process, we would evaluate our model M on the test set (X_{test}, y_{test}).

The performance of our network models is evaluated based on two major metrics. The first one is detection accuracy, which is represented by regression loss. To be more specific, in our case, we quantize the loss using mean squared error (MSE) between the ground truth y and predicted keypoints vectors $\hat{y}$. The second one is time consumption in both training and testing phase.

3.2. One Hidden Layer Neural Network

First, we use *one hidden layer neural network* model as a baseline. We reshape the 96×96 image to a 9216×1 vector as the input of our network. The neuron number of the hidden layer is 100. And the output layer outputs 15 sets of coordinates of the 15 keypoints, giving 30 numbers in total.

The loss function is defined as the Euclidean distance between the ground truth and output keypoints vectors. To converge faster, we use Nesterov momentum as the update rule for gradient descent. The batch size is 1 and we iterate for 400 epochs during the training phase.

3.3. Convolutional Neural Network

To achieve better keypoints detection accuracy (lower loss), we build a *convolution neural network* model as an advanced baseline. The architecture diagram of our CNN is shown in Fig. 1. The number shown above each layer demonstrates the size of its corresponding activation, and the description below each layer describes its function. The hyperparameters of those convolutional layers in the network are illustrated in Table. 1. Both hidden layers have 500 neurons and the output layers generate a 30-element vector, same as the former *one hidden layer neural network*, representing the coordinates of the keypoints.

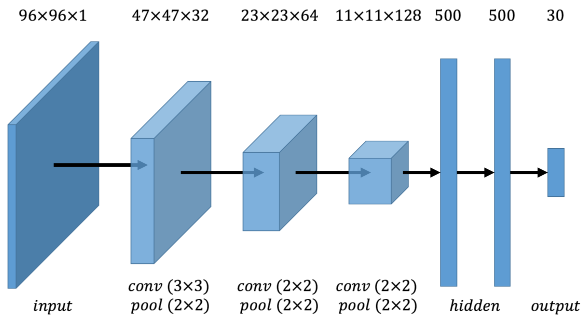

Figure 1. Convolution Neural Network Architecture

Table 1. Hyperparameters of CNN

	conv layer 1		conv layer 2		conv layer 3	
	conv	pool	conv	pool	conv	pool
filter	3×3	2×2	2×2	2×2	2×2	2×2
stride	1	2	1	2	1	2
pad	0	0	0	0	0	0

3.4. Insights for Baselines of Facial Keypoints Detection

As we mentioned before, one of the biggest challenges for facial keypoints detection is the computation complexity. In our experiments, we find that even though cnn achieves better performance, which will be illustrated in detail in Sec. 5.3, it takes much more time to compute in each epoch (35s vs 0.30s). This phenomenon reflects a major drawback of traditional cnn architectures that they use a dramatic amount of computational resources, especially when the network size increases. Another nature weakness of cnn is its proneness to overfitting due to its large number of parameters.

3.5. Inception Model

As is mentioned above, one inevitable weakness of multi-layer cnn architecture is its unnecessarily large amount of parameters. Due to increased layer and parameter size, the network is inclined to overfit and fails to generalize. Besides, a huge number of parameters causes dramatically large computational resource consumption as well.

To overcome such drawbacks illustrated above, one fundamental way of solving this issue is to make some ultimate adjustments from the architecture-level. Szegedy et al proposed such a model named Inception Model [8], which prefers sparsely connected architectures over fully connected ones, even inside the convolutions. This method not only solves the large parameter issue fundamentally, but also is more consistent from a biological perspective.

However, this idea may not work as well as we expected in practice because of the optimization approach in hardware-level. When it comes to numerical computation on non-uniform sparse data structures, traditional computing infrastructures are not very inefficient. This is also the reason why we need to put so many efforts in the research of efficient operations on sparse data structures, e.g. sparse matrix multiplication. In such a case, to keep the balance between computational resource consumption and computational efficiency, an optimal structure seems to be like a globally sparse convolutional neural network, which is composed of dense structures locally. Global sparsity guarantees the number of parameters is limited while local density provides efficient computation.

A typical block of the Inception Model is shown in Fig. 2. In each layer, we use multiple filters with different sizes 1×1, 3×3 and 5×5 instead of one filter in traditional cnn layers. This approach achieves so-called global sparsity since we split a huge amount of parameters from one-size filter up into several groups corresponding to different filter sizes. Meanwhile, it guarantees local density as well because each convolution operation can be realized in a traditional way.

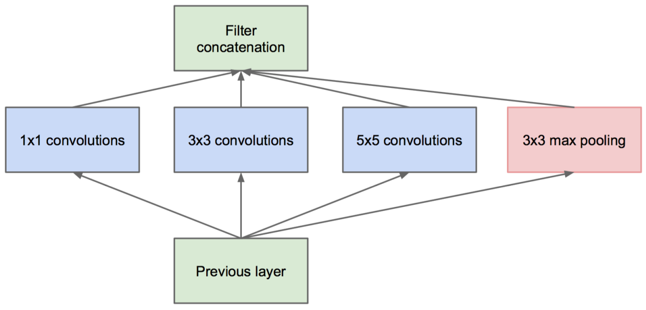

Figure 2. Naive inception module

Nevertheless, the above naive version of Inception Model block is still inefficient in computation. Assume the input size is $S \times S \times C$ where $S \times S$ is the image size and C represents the channel number, then for each filter 1×1, 3×3 and 5×5, the number of channels are all the same C. A large C will lead to a large number of computations due to 5×5 filter; while if C is reduced to a moderate value, then there will be less point in using 1×1 filter since it may not generate well-generalized features. In a nutshell, this block needs further adjustments to vary the number of channels of different size filters.

To realize such an architecture illustrated above, the authors in [8] bring up an advanced block shown in Fig. 3, which is used in the final Inception Model. The main difference from the structure in Fig. 2 is that 1×1 convolutional filters are inserted in front of 3×3 and 5×5 filters (and after max pooling). In such a way, we first use 1×1 convolutions to compute reductions before expensive convolutions like 3×3 and 5×5. These 1×1 convolutions can also be used as rectified linear activations. As a result, we bear more channels when computing 3×3 and fewer for 5×5 since it is more computationally consumptive.

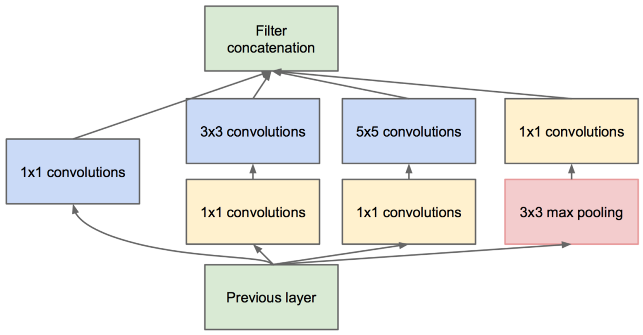

Figure 3. Inception module with dimension reductions

Here in Table. 2 displays some specific numbers of depths in each filter. The 3×3 and 5×5 reductions represent the two 1×1 convolutions inserted before them. The input depth is 128, which is reduced to 96 and 16 before 3×3 and 5×5 filters. The numbers in bold represent the depths of convolutional outputs, which add up to 256 (= 64 + 128 + 32 + 32) in total.

Table 2. Inception model instance

1×1	3×3 reduce	3×3	5×5 reduce	5×5	pool proj	output size
64	96	**128**	16	**32**	**32**	**256**

Another impressive advantage of Inception Model is its flexibility to be adapted to different layers. In lower layers, which we focus more on local regions of the input image, like edge information, what matters more is actually smaller size filters like 1×1 and 3×3. On the other hand, however, in higher layers, where features like texture are more inclined to be captured, we probably need to use larger size filters like 5×5. Using such a model block, we can adjust the number of depths among different filters according to our demands at different layer levels.

In the following experiments, we didn't build a network using Inception Model blocks. Instead, we extracted features from a pretrained Inception Model for image classification, and input those features to three different networks, 1) *one hidden layer neural network* and 2) *cnn* whose architectures are quite similar to the former baselines, and 3) *cnn* with dropout.

In general, we compare performance (loss and train/test time) among those 5 networks to evaluate the effectiveness of those extracted features compared to raw image.

4. Dataset Description

The dataset we use is from kaggle's facial keypoints detection competition. There are 7049 images in total, each of which is a 96 $\times$ 96 pixels image. Among all the images, 2140 out of them have ground truth positions for all 15 facial keypoints, which are used to as our dataset to train, validate and test the network.

The dataset is randomly divided into training set, validation set and test set. First around 20% of all the 2140 images are chosen to be the test set (440), where 80% of the left images (1700) are training set (1360) and the rest of them are used for validation (340).

The 15 facial keypoints on a given face are listed below:

Table 3. All 15 facial keypoints

left_eye_center	right_eye_center
left_eye_inner_corner	left_eye_outer_corner
right_eye_inner_corner	right_eye_outer_corner
left_eyebrow_inner_end	left_eyebrow_outer_end
right_eyebrow_inner_end	right_eyebrow_outer_end
nose_tip	mouth_left_corner
mouth_right_corner	mouth_center_top_lip
mouth_center_bottom_lip	

5. Experiment

In this section, we will demonstrate how we conduct experiments on both baseline models and inception models in detail. First, we introduce the evaluation metrics to assess the performance of all the models in both detection accuracy and time complexity. Then experimental results will be shown qualitatively and quantitatively.

5.1. Evaluation Metrics

For our experiments, as our evaluation metrics for regression loss, we use mean squared error (MSE) between the ground truth keypoint coordinate vector and the predicted one.

Given the ground truth $y = \{y_1, \cdots, y_i, \cdots, y_n\}$ and our estimate $\hat{y} = \{\hat{y_1}, \cdots, \hat{y_i}, \cdots, \hat{y_n}\}$, MSE loss is defined as the average of the square of all of the error, which can be represented as:

$$loss_{\text{MSE}} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

In our case, the coordinates of keypoints are in the range of $[0, 95] \times [0, 95]$. To better correspond to normal distribution initialization ($\mu = 0$) of weights in network, we rescale those coordinates to $[-1, 1] \times [-1, 1]$.

As for evaluating the execution time of each model, we compare the time cost in both training phase (per epoch) and testing phase (per image).

5.2. Experimental Setup

We conduct experiments on 5 models in total, 2 of which are baseline models and the left 3 are extensions from baseline using Inception Model.

The experimental platform is a 2-core 2.7GHz CPU + 8GB 1867MHz memory laptop. And the framework we used for building and training cnn is Lasagne and Theano, implemented in python.

5.2.1 Baseline models

The two baselines are *one hidden layer* neural network and *5 layer* cnn, which are well defined in Section. 3.2 and Section. 3.3 respectively. Those two networks are the most typical deep architectures and serve as baselines.

5.2.2 Inception models

Different from the baseline models, we insert Inception Model block to extract features first, and input these high-level features to following deep architectures which the same as baselines.

To be more specific, we use a pretrained Inception Model [8] to extract low-level features given human faces. And we take those features as inputs to train the former two baseline models. Basically, all we need to do is to adjust the input interface (change the input dimension of the networks). And we add dropout in the cnn model as a third Inception model to overcome overfitting.

5.3. Experimental Results

The experimental results consist of two parts as well. First, we focus on the detection accuracy by comparing training, validation and testing loss among all the 5 models illustrated above. Second, we evaluate the time complexity of those 5 models by comparing their time consumption in both training and testing phases.

5.3.1 Detection accuracy

To evaluate our networks, we compare loss after a certain number of epochs (400). The curves of decreasing loss are shown as follows in Fig. 4. Dashed curves represent training loss while solid curves are validation loss.

Both training and validation loss in all the 5 models decrease as epoch number increases. Specific numbers can be found in Table. 4.

- For *one hidden layer* model, which is the simplest deep architecture among the 5 approaches, achieve the worst accuracy ($loss_{test} = 3.88$[1]) as expected. Besides, it overfits in the training phase and a huge gap can be seen between the two green curves in Fig. 4.

[1] 10^{-3} is omitted here and in the following loss numbers.

Table 4. Regression loss (MSE) of training, validation and testing set

iter $/10^{-3}$	hidden			cnn			inception_hidden			inception_cnn			incp_cnn_dropout		
	train	valid	test	train	valid	test	train	valid	test	train	valid	test	train	valid	test
100	5.25	6.27	/	4.22	4.11	/	2.18	2.87	/	1.50	1.78	/	3.90	4.38	/
200	3.76	4.96	/	3.79	3.78	/	1.50	2.44	/	1.02	1.42	/	3.56	4.14	/
300	2.97	4.24	/	3.37	3.43	/	1.16	2.26	/	0.77	1.29	/	3.17	3.75	/
400	2.50	3.82	**3.88**	2.94	3.09	**3.05**	0.95	2.15	**2.07**	0.56	1.18	**1.13**	2.50	2.96	**3.12**

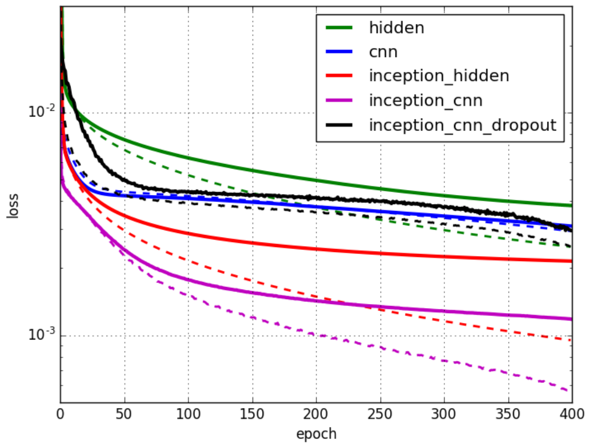

Figure 4. Regression loss (MSE) versus epochs

- For *cnn* model, the results are shown in blue in Fig. 4. Compared to the one hidden layer model, the loss is slightly better (3.09) And the training and validation loss are very close, meaning there's not much overfitting.
- For the Inception one hidden layer model, we use Inception features as input and double the hidden layer size. It achieves better accuracy (2.07) but suffers from overfitting severely (red curves).
- For the Inception cnn model, the loss can be further decreased to 1.13 But it shares the same drawback that the network overfits. One of the possible reasons is that there are too many parameters.
- To overcome overfitting, we use dropout as regularization in the last Inception cnn dropout model. Even though the network is no longer overfitting, its performance drops dramatically to 3.12. However, from the black curves in Fig. 4, we can see the trend of loss continuing decreasing as epochs increase. Since dropout makes the network converge slower, it needs more epochs to reach the same loss as the Inception cnn model.

Here we display 6 sets of detected keypoints using the 5 networks, compared to the ground truth keypoints from the dataset. Shown in Fig. 5, the keypoints computed with less loss is intuitively much closer to the ground truth.

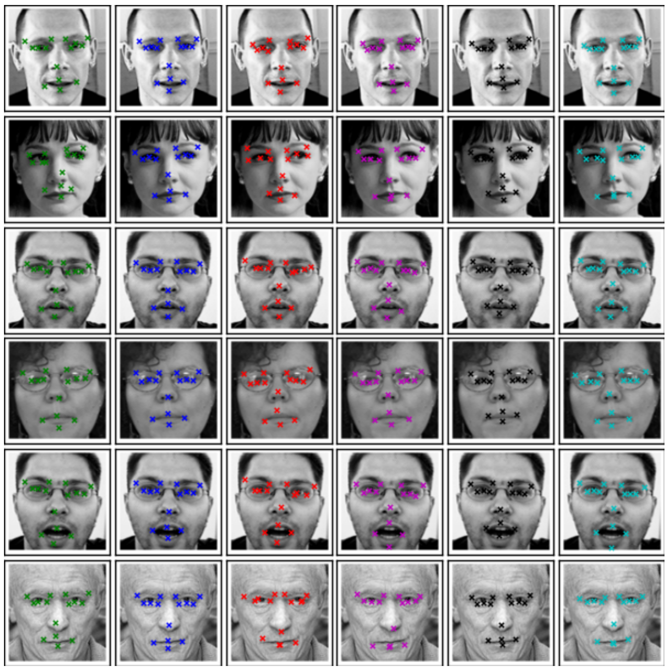

Figure 5. Keypoints detection results. From left to right, one hidden layer network (red), cnn (blue), inception hidden(red), inception cnn (magenta), inception cnnn with dropout (black) and groudtruth (cyan).

5.3.2 Time complexity

Another important metric to evaluate the performance of a network model is its time cost in both training and testing. The results are shown in Table. 5, where *training* time represents the time cost for one epoch (1700 images) and *testing* time represents the time cost for predicting the keypoints in one test image.

Table 5. Time consumption in training and testing

	hid	cnn	incp_hid	incp_cnn	incp_cnn_drpt
train / s	0.30	35	12	28	29
test / ms	0.105	7.67	4.77	8.50	8.86

For the basic hidden layer architecture, we can actually verify the results theoretically. The original input is a 96 ×

$96 = 9216$ dimension vector, and the hidden layer size is 100. Hence the total amount of multiplication operations is around 10^6 (ignoring the impact from additions). Using Inception features results in an dramatic increase in input from 9216 to $28 \times 28 \times 256 = 200704$. And we also increase the hidden layer size from 100 to 200. So the total number of multiplications is around $2 \times 10^5 \times 200 \approx 4 \times 10^7$. The amount of multiplications needs computing increases 40x, as a result of which, the time consumption should be 40x as well. Real numbers show 40.52(=12.155/0.30) and 45.62(=4.767/0.1045).

Convolutions are much more complicated due to matrix operation optimization in numpy. Linearly increased amount of multiplications may not cause linear increase in time cost. In our case, we use $28 \times 28 \times 256$ rather than $96 \times 96 \times 1$ as the input of Inception models. Even though the actual computation amount rises dramatically, the actual time consumption drops in the training phase, and increases slightly in the test phase. In other words, bearing slightly higher time cost, the detection accuracy improves dramatically from 3.05 to 1.13.

6. Conclusion and Future Work

In this paper, we have focused on the task of detecting facial keypoints when given raw facial images. Specifically, for a given $96 * 96$ image, we would predict 15 sets of (x, y) coordinates for facial keypoints. Two traditional deep structures, One Hidden Layer Neural Network and Convolutional Neural Network, are implemented as our baselines. We further explored a sparsely connected Inception Model to reduce computational complexity to fit the requirements for detecting facial keypoints. Experiments which conducted on real-world kaggle dataset have shown the effectiveness of deep structures, especially Inception Model.

As for our future work, we can explore from these few aspects,

1. We have already shown the effectiveness of the Inception Model when used as the pretrained model, but the performance has a chance to improve if we train from the scratch.

2. As we can see from the results, using deep structures can increase time complexity when compared to other start-of-art methods but the results have been shown to improve a lot much. What we can do in the future is to design a deep structure specifically for this task to further improve the performance.

3. Different resolution can greatly affect the results of the facial keypoints detection, thus what we can do is try to reduce the resolution of our given raw images to see the variance of the performance to further evaluate our model.

References

[1] M. Dantone, J. Gall, G. Fanelli, and L. Van Gool. Real-time facial feature detection using conditional regression forests. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2578–2585. IEEE, 2012.

[2] M. Gargesha and S. Panchanathan. A hybrid technique for facial feature point detection. In *Image Analysis and Interpretation, 2002. Proceedings. Fifth IEEE Southwest Symposium on*, pages 134–138. IEEE, 2002.

[3] M. Haavisto et al. Deep generative models for facial keypoints detection. 2013.

[4] E.-J. Holden and R. Owens. Automatic facial point detection. In *Proc. Asian Conf. Computer Vision*, volume 2, page 2, 2002.

[5] B. Martinez, M. F. Valstar, X. Binefa, and M. Pantic. Local evidence aggregation for regression-based facial point detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(5):1149–1163, 2013.

[6] A. S. Mian, M. Bennamoun, and R. Owens. Keypoint detection and local feature matching for textured 3d face recognition. *International Journal of Computer Vision*, 79(1):1–12, 2008.

[7] Y. Sun, X. Wang, and X. Tang. Deep convolutional network cascade for facial point detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3476–3483, 2013.

[8] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[9] M. Valstar, B. Martinez, X. Binefa, and M. Pantic. Facial point detection using boosted regression and graph models. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2729–2736. IEEE, 2010.

[10] D. Vukadinovic and M. Pantic. Fully automatic facial feature point detection using gabor feature based boosted classifiers. In *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, volume 2, pages 1692–1698. IEEE, 2005.