Facial Keypoints Detection with Inception Structure

Chongzhao Mao
Electrical and Computer Engineering
University of Wisconsin-Madison
Madison, USA

Abstract—In this paper. We will tell a application of Inception Net on Kaggle dataset for Facial Keypoints Detection. We will depict some background of the convolution neural network on computer vision with the development of ImageNet. Next we will introduce the ideas of Inception Net. In the end we will do the experiment on the Facial Keypoints Detection dataset.

I. INTRODUCTION

One old network—LeNet should be the most famous neural network right now introduced by LeCun et al [1] in 1998. It starts the idea of convolutional neural network(CNN) on the image classification and recognition. About ten years later, the competition Large Scale Visual Recognition Challenge(LSVRC) from ImageNet really explodes the research on deep learning with CNN on computer vision.

ImageNet is an image database organized according to the WordNet hierarchy in which each node is depicted by hundreds and thousands of images, and right now there are around five hundreds images per node.

Right now, there are many famous CNN neural network come from the ImageNet dataset’s competition. AlexNet, introduced by Krizhevsky et al [2], was the winner of 2012 LSVRC. Szegedy et al [3] from Google, created the GoogleNet and Simonyan et al [4], with VGG Net, won the first and second prize of 2014 competition. On 2015, Microsoft Research Aisa(MSRA) used over 150 layers very deep network, called ResNet [5], won the game. The structures and networks are all trained on large scale dataset, but there are still many great ideas of designing and training CNN, while the units and layers can be used separately, which really enhanced the research in the field.

II. INCEPTION

Inception was introduced by Google, which was in Google Net and Inception-ResNet that won the first place of 2014[3] and 2016 ILSVRC classification task. The structure of Inception looks like below:

![Inception Unit Diagram](image)

Fig. 1 Inception unit

We can call each branch in the Inception unit as tower. Inception tries to use the concatenated structure and smaller filter size to optimize the deep CNN. For example, suppose the input is of dimension a * b * c and we use convolution with c filters. If we use the 11 by 11 filter size, the weight number should be

\[ c \times 11 \times 11 \times c = 121c^2 \]

If we use five 3 by 3 filters, we will have

\[ 5 \times 3 \times 3 \times c = 45c^2 \]

While in the convolution, we will find that the area and information represented by the result of convolution are the same. The 3 by 3 filter, will represent 3*3, 5*5, 7*7, 9*9, 11*11 area after each convolution in five filters. While this will be the same as a single 11 by 11 filter, which mean we are achieving the goal for dimension reduction while at the mean time reducing the weights number for calculation. So in this way we can get faster speed, more nonlinearity and less memory need in training.

As for 3 by 3 filter, we know the weight number will be

\[ 1 \times 3 \times 3 \times 3 \times c = 9c^2 \]

We can think of the difference between 2D Gaussian filter and 1D Gaussian filter. To smooth a 3D image, the 1D Gaussian will smooth along x, y, z direction, while 2D Gaussian will smooth along x-y, x-z, y-z plane. The result would be 2D Gaussian time cost is higher, and we know that 2D Gaussian is composed of two 1D Gaussian.
\[ w^{2D} = w^*w \]

So that the parameter of filter will be \( O(n^2) \) for 2D Gaussian while \( O(2^n n) = O(n) \) for 1D Gaussian. Back to the example, the composition of a 1 by 3 filter and a 3 by 1 filter should have \[ c^3*1^*c + c^1*3^*c = 6c^2 \]

While the 3 by 3 have \( 9c^2 \) number of weights. Then using this we see that we can reduce the number of weights and computing time.

One more thing is that if we can use a 1*1 filter to reduce the dimension first, then the computing time needed will be reduced also, and after computing, we can still use 1*1 filter to increase the dimension as previous look.

### III. Data

The data we use is from Kaggle’s open source datasets. The data set for this competition was provided by Dr. Yoshua Bengio of the University of Montreal.

There are 7049 gray scale face images of 96x96 pixels, where each contains the \((x, y)\) coordinates for the 15 key points that shown below.

left\_eye\_center, right\_eye\_center, left\_eye\_inner\_corner, left\_eye\_outer\_corner, left\_eyebrow\_outer\_corner, left\_eyebrow\_inner\_corner, right\_eye\_inner\_corner, right\_eye\_outer\_corner, right\_eyebrow\_outer\_corner, right\_eyebrow\_inner\_corner, nose\_tip, mouth\_left\_corner, mouth\_right\_corner, mouth\_center\_top\_lip, mouth\_center\_bottom\_lip

Here the left and right refers to the point of view of the face.

![Data Sample & Label](image)

### IV. Data Preprocessing & Augmentation

For the data given, in order to get rid of the brightness effect or other differences between images, I just normalized the pixel intensity value into \([0, 1]\) by subtracting the mean value and divide by standard deviation. For the label values, which are all in \([1, 96]\), in order to make the data augmentation easier, I just subtracted the value by 48 and then divide by 48 so they are between \([-1, 1]\).

Since we only have 7000+ images, and some images have the NA values in them, which means the face is the half-face or the key points cannot be found. After filtered those images, we only have 2000+ images for training. To increase the training set, a good way for this problem is to flip the image with right to left and left to right. Then we will see that during this process, the pixels’ column position in image will be negative compared with before since we have made the position in \([-1, 1]\). Actually, since the row value is also in \([-1, 1]\), we can flip the image upside down to augment the data, since the image we are using are all of right pose (head up mouth down), so I did not include this into the algorithm.

### V. Experiment

The deep learning tool that I use is Torch, which is in Lua. The most often used packages are ‘nn’ and ‘optim’. The code was written in iTorch of Jupyter and runs on CPU.

The learning rate was set to 0.5, while it will be updated. When the validation loss starts to increase and keep increasing for some threshold time, the learning rate will be divided by two to not miss the right place.

The optimizer is using stochastic gradient descent, which is used often. The momentum, after trying values of \([0, 0.3, 0.5, 0.7, 0.9]\), I found that the 0.7 is the best value that leads to least loss. The weight decay was set to 5e-4. The batch training is with the batch size 105. Every time when the data set have been all trained, the training data will be shuffled into another random permutation. The loss function is mean square error.

The network has 17 layers. The first several are normal spatial convolution, ReLU and max pooling. Then I used an Inception unit, consisting of three towers. First is a sequential net of 1*1, 1*3, 3*1 filters. Then I concatenated them with a 1*1 filter, which is the second tower. They together connect to a 1*1 filter and then concatenated with the third tower, a max pooling layer. After the Inception, I used the several layers of fully connection to get the output of number 30, which is the number of the coordinates of the keypoints. The net structure looks as below:

<table>
<thead>
<tr>
<th></th>
<th>Conv 3*3</th>
<th>ReLU</th>
<th>MaxPooling 2*2</th>
<th>Conv 3*3</th>
<th>ReLU</th>
<th>MaxPooling 3*3</th>
<th>Conv</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv 1*1</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>Conv 1*3</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>Conv 1*1</td>
<td>/</td>
<td>MaxPooling 3*3</td>
<td></td>
</tr>
<tr>
<td>Conv 3*1</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
</tr>
</tbody>
</table>
I also added the LeNet and AlexNet as comparison to test the performance under the same optimization parameters.

The best result right now is of the loss 0.0037, converting to the real value, it is \( \sqrt{0.0037} \times 48 = 2.9184 \). According to the range is 96, then the error rate is \( \frac{2.9184}{96} = 0.0304 = 3\% \).

VI. CONCLUSION

Convolutional neural network has broad application in computer vision. They can get rid of the process to extract the features from images, but the cost is to tuning the parameters and energy to training the neural network.

The Inception is really a good way to reduce the number of weights and computing time. With the same number of layer, the Inception will have much lower parameters.

The up-to-now error rate is 3\%, which is not enough. Due to the time limitation, the training of the network is not completed. I will keep training the network and adjust the parameters and the network to reach a better result.

REFERENCE


