CONTRAST OF RESNET AND DENSENET BASED ON THE RECOGNITION OF SIMPLE FRUIT DATA SET

Ding Tianye

Hangzhou Foreign Language School, Hangzhou, Zhejiang, China

ABSTRACT

In this paper, a fruit image data set is used to compare the efficiency and accuracy of two widely used Convolutional Neural Network, namely the ResNet and the DenseNet, for the recognition of 50 different kinds of fruits. In the experiment, the structure of ResNet-34 and DenseNet_BC-121 (with bottleneck layer) are used. The mathematic principle, experiment detail and the experiment result will be explained through comparison.

KEYWORDS

Deep learning, Object recognition, Computer vision, Image processing, Convolutional Neural Networks.

1. INTRODUCTION

The aim of this paper is to discover the learning efficiency and convergence rate in machine learning of ResNet and DenseNet_BC through comparable experiments. The fruit image data set [1] used in the experiment consists of images of 100*100 with black/white background and

Figure 1. one of the images in the image data set [1]

without noise interference. Including 50 kinds of different fruits, 25,100 images in total as training data and 12,700 images as testing data. In the experiment, the only consideration is how many times is needed to train the neural network so that it can have an accuracy of more than 98%. The first step in the training process is to pre-process the input images, using the Python module OpenCV to turn the images into RGB channel images, and divide each channel value by 225 so that the resultant value is in the range of 0 to 1. Combining 50 images as an input batch, determining the loss between each prediction and actual value through computing the cross

DOI: 10.5121/ijcsitce.2019.6101

entropy, $\sum_x p(x) \cdot \log(\frac{1}{q(x)})$, every time after 10 times of iterate input, compute the recognition accuracy and output as a way of visualization, and train the two networks for 2,000 times before input the testing data. Each convolutional layer uses ReLU(Rectified Linear Unit) as activation function, using a layer of batch normalization between each code block and a layer of dropout with keep probability of 0.5 to avoid the appearance of over fitting, the training of both neuron networks uses Adam Optimizer and epsilon of 0.1, and the setting of learning rate uses the method of Learning Rate Exponential

Decay ($\text{decayed learning rate} = \text{initial learning rate} * \text{decay rate}^{\frac{global\ step}{decay\ step}}$), the decay step is the same as the total training time, setting stair case as True, the initial learning rate is 1e-3, using the decay rate of 0.96. The hardware environment used in the experiment is running on GPU GTX 1070 with allocated memory of 5.0Gb, the framework used is TensorFlow [2] developed by Google in 2015.

2. DEEP LEARNING

The deep learning neuron networks are usually consisted by multiple layers, the input data of each layer is the output of the previous layer, compare with shallow learning, deep learning is usually been regarded a great from Weak AI towards Strong AI.

The Convolutional Neural Networks (CNN) can be categorized as supervised learning in deep learning, which means that the training process of the neuron networks needs to provide not only the input data, but also the actual data used to calculate the loss between prediction value and actual value, and the optimizers will use different back propagation algorithms to contribute the total loss onto each neuron and the activation functions inside each neuron will change the parameters inside each neuron, so that after enough data input to the network and trained for enough times, the neuron network will tend to find the local or global optimize and achieve a good enough performance on the particular question that it is trained for.

CNN is most widely used in the field of image recognition, a paper [3] has proven the idea that the convolutional neural networks have a better and those advanced image recognition networks like ResNet and DenseNet are developed on the base of CNN. That is one reason why fruit data set [1] is chosen as training data. The pooling layers in the CNN are used to reinforce and compress the feature in each feature map to reduce the possibility of feature disappear.

3. TWO CONVOLUTIONAL NETWORKS

In the neuron networks that were used in the computer vision and image recognition before, with the increase of the depth of the network, gradient disappear of explosion and OOM (Out of Memory) lead to the reduction in the accuracy of the neuron network has turned into a difficulty that many teams or individuals are trying to overcome.

2.1. ResNet

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 2. The parameters of different structures of ResNet, the structure used was 34-layer

In 2015, the introduction of ResNet (Residual Convolutional Network), which has won the champion of classification in the ImageNet competition, greatly narrow down the problem through the method called residual transmission- using simple code block to transfer the input and the output in the previous layer as residual to be the input data of the next layer, as shown in Figure 3, which provides a channel for the input gradient in each layer only has change in dimension but no need to process and get into the next layer of neuron network. The principle of ResNet is simple code blocks, stacking and connecting by channels, not only simplify the complexity of neuron networks, but also reduce the memory occupied by the session in the running process, which greatly reduce the probability of OOM, improve the efficiency of machine learning and the rate of gradient convergence, so that many programmers are keen on using ResNet in supervised learning.

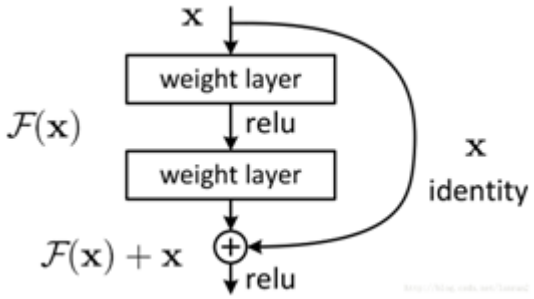


Figure 3. Visible structure of ResNet

The formula of ResNet in the Essay "*Densely Connected Convolutional Networks"* [4] is briefly introduced as "$x_l = H_l(x_{l-1}) + x_{l-1}$" where x_l represents the input data of current layer, H_l represents non-linear transformation, the input of current layer is residual formed by combining the input and output of the previous layer and transmit to the next layer. Also, each code block in the ResNet contains residual block used to convolution and the identity block used to directly transmit gradient, the resultant output of each block is $\mathrm{y} = \mathrm{F}(\mathrm{x}) + \mathrm{x}$, and $\mathrm{F}(\mathrm{x}) = y - x$ represents the part of residual. Theoretically, with the increase in the depth of the neuron network, the output of part of the residual blocks will gradually tend to be 0, but the channel through identity block is still transmitting the gradient, so that the problem of gradient disappear won't happened in the network, and the network can stay in the optimal state.

When the two adjacent blocks have a change in the convolutional kernel depth, there will be an identity block with transformation operation to change the dimension of the input tensor, so the dimension of input tensor and the output tensor will stay the same on the y-axis.

The convolution layers of the ResNet-34 in the experiment except from the very early stage and the full-connected layers are all using a kernel size of 3*3, the output number of feature maps in each stage is separately 64, 128, 256 and 512, ensuring that the computing speed and learning efficiency won't be influenced because of the output number of feature maps is too large. Normalizing the value of RGB into the range of 0 to 1 can reduce the computation complexity while training. And the labels of actual values are processed into one hot code.

During the training process, in the first 190 times of input data batch iteration, the output accuracy of the neuron network doesn't have a great increase, is about 0% to 6%. After 190 times of iteration, it appears to be a rapid increase of the recognition accuracy and when the training step reaches about 630 to 640, the accuracy first reached about 98%; in the further training, the accuracy is about 94% to 100%. After 2000 times of training, the output testing accuracy is about 96% to 98%

.2.2. DenseNet

Layers	Output Size	DenseNet-121($k=32$)	DenseNet-169($k=32$)	DenseNet-201($k=32$)	DenseNet-161($k=48$)
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	1 × 1 conv 3 × 3 conv × 6	1 × 1 conv 3 × 3 conv × 6	1 × 1 conv 3 × 3 conv × 6	1 × 1 conv 3 × 3 conv × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	1 × 1 conv 3 × 3 conv × 12	1 × 1 conv 3 × 3 conv × 12	1 × 1 conv 3 × 3 conv × 12	1 × 1 conv 3 × 3 conv × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	1 × 1 conv 3 × 3 conv × 24	1 × 1 conv 3 × 3 conv × 32	1 × 1 conv 3 × 3 conv × 48	1 × 1 conv 3 × 3 conv × 36
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	1 × 1 conv 3 × 3 conv × 16	1 × 1 conv 3 × 3 conv × 32	1 × 1 conv 3 × 3 conv × 32	1 × 1 conv 3 × 3 conv × 24
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Table 1. DenseNet architectures for ImageNet. The growth rate for the first 3 networks is $k = 32$, and $k = 48$ for DenseNet-161. Note that each "conv" layer shown in the table corresponds the sequence BN-ReLU-Conv.

Figure 4. The parameters of different DenseNet structures, the structure used was DenseNet_BC-121 from *Densely Connected Neuron Network* [4]

In CVPR 2017, the oral paper of "*Densely Connected Neuron Networks*" [4] by Gao Huang, Zhuang Liu, Kilian Q. Weinberger and Laurens van der Maaten, which firstly introduce a completely new neuron network structure- DenseNet, which focuses on solving the problem of ResNet (the gradient passing through residual might be impeded when the network gets deeper),

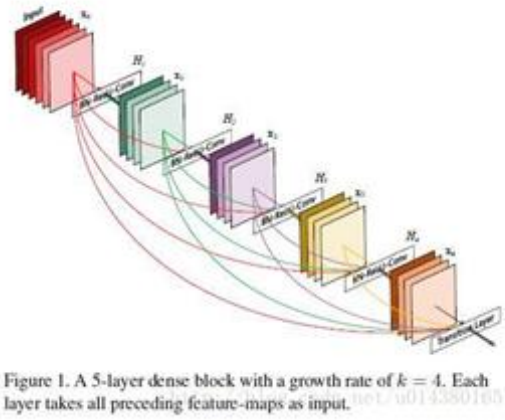


Figure 1. A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

Figure 5. the visible structure of DenseNet form *Densely Connected Neuron Networks* [4]

the paper describes the brief formula of DenseNet as "$x_l = H_l([x_0, x_1, \dots, x_{l-1}])$", this function was achieved in the experiment through stacking the output of each previous layer by the 3rd axis as input of the current layer, adopt a growth rate of 32, which means the output of each layer is 32, and drop rate of 0.5, means in the transition layer, the number of output feature maps is reduced into half of the initial number. Compare to ResNet, theoretically, because the DenseNet is narrower than ResNet, the DenseNet with same number of neuron layers will have less parameter and the because of the bottleneck layer and small number of output feature maps (every layer usually have a small convolutional kernel depth), which can save the memory used while running the session, and because input of every neuron layers is directly access to the input and output gradient of each layer, so the problem of gradient disappear won't exist with the increase in the depth of neuron network.

In the training process, the assumption is that the training of DenseNet is more speed-efficient and takes fewer steps for converging to a higher accuracy compares with the performance of ResNet. However, the output result is not as ideal as the expectation, the time taken for the network to reach an output accuracy is much long than the training process of ResNet-34, which might be cause by the depth of the neuron network, furthermore, when the training step reached about 200, there isn't an obvious improvement in the training accuracy, even after 2000 times of training, there is only an increase in accuracy about 10% to 35%, then several tries for eliminating the bottleneck layer to keep the most number of feature maps, whereas still the training result isn't as ideal as expectation.

2.3. Code Segments

In the experiment, the TensorFlow [2] framework was used, *import tensorflow as tf*, also import the module slim, *from tensorflow.contrib.slim as slim*, to simplify the complex procedure of defining a complete convolutional layer.

In both experiments, the tensor of weights and biases are randomly produced as below:

define function weights with parameter(shape)
init = tf.truncated_normal with parameter(shape, standard deviation=0.01) (tensor with shape filled with random truncated normal with standard deviation 0.01)
return variable init as the output of the function

define function biases with parameter(shape)
init = tf.constant with parameter(0.02, shape) (tensor with shape filled with 0.02)
return variable init as the output of the function

In the experiment of ResNet-34, the convolutional layers with the transformation in each stage is defined as below:

define function conv with parameters (inputs, out_size, k_size)
x_short_cut = inputs
conv1 = 2-dimensional convolution (inputs, number of output feature maps=out_size, kernel size=k_size, stride=2, padding='SAME', without activation function)
conv1_output = ReLU activation function (batch normalize (conv1 on axis=3))
conv2 = 2-dimensinal convolution (conv1_output, number of output feature maps=out_size, kernel size=k_size, stride=1, padding='SAME', without activation function)
conv2_output = batch normalize (conv2 on axis=3)
input_conv = 2-dimensinal convolution (x_short_cut, number of output feature maps =out_size, kernel size=k_size, stride=2, padding='SAME', without activation function)
input_reshape = batch normalize (input_conv on axis=3)
output = ReLU activation function (input_reshape+conv2_output)
return output as the output of the function

The identity blocks which include the channel for passing the residual of input data and the output result is defined as below:

define identity with parameters (inputs, out_size, k_size)
x_short_cut = inputs
conv1 = 2-dimensinal convolution (inputs, number of output feature maps=out_size, kernel size=k_size, stride=1, padding='SAME', without activation function)
conv1_output = ReLU activation function (batch normalize (conv1 on axis=3))
conv2 = 2-dimensinal convolution (conv1_output, number of output feature maps=out_size, kernel size=k_size, stride=1, padding='SAME', without activation function)
conv2_BN = batch normalize (conv2 on axis=3)
conv2_output = ReLU activation function (conv2_BN+x_short_cut)
return conv2_output as the output of the function

In the experiment of DenseNet_BC-121, the dense blocks with growth rate of 32 and a bottle neck layer is defined as below:

Define function dense with parameters (inputs, growth_rate=32, internal_layer=True, keep_prob)
x_input = batch normalize (inputs, axis=3)
x_relu = ReLU activation function (x_input)
# bottleneck layer
conv1 = 2-dimensinal convolution (x_relu, number of output feature maps=growth_rate*4, kernel size=1*1, stride=1, padding='SAME', without activation function)
conv1_dropout = Dropout (conv1, keep percentage=keep_prob)
conv1_relu = ReLU activation function (batch normalize (conv1_dropout on axis=3))
conv2 = 2-dimensinal convolution (conv1_relu, number of output feature maps=growth_rate, kernel size=3*3, stride=1, padding='SAME', without activation function)
conv2_dropout = Dropout (conv2, keep percentage=keep_prob)

```
    if internal_layer is True then
        output = stack ([inputs, conv2_dropout] on axis=3)
    else
        output = conv2_dropout
    return output as the output of the function
```

The transition block used to compress the dimension of input tensor is defined as below:

```
Define function transition with parameters (inputs, drop=0.5)
    x_input = batch normalize (inputs on axis=3)
    conv = 2-dimensinal convolution (x_input, number of output feature maps=the depth of tensor x_input*drop, kernel size=1*1, stride=1, padding='SAME', without activation function)
    h1 = average pooling (conv, pooling kernel size=2*2, strides=2, padding='SAME')
    return h1 as the output of the function
```

2.4. Experiment details

The procedure of tuning parameters in the experiment was mainly focus on the learning rate of the neural networks and the default decay rate in exponential decay was 0.96. The method of tuning refers to *A Practical Guide* [5], range while tuning the learning rate is set between 1E-4 and 0.7, first start with 0.7 and observe the rate of convergence through the output accuracy. Bigger learning rate was easier for the accuracy to rise to about 95%, whereas the prediction will start to move back and forth the local optimum and hardly make further improvements. Then the learning rate will be change to 0.3, but the problem still occurs. However, if the initial learning rate was too small, it will take a long time to converge since the beginning of training process. Ultimately, it turned out to be that 0.003 is the most efficient learning rate for ResNet after limited times of experiments.

2.5 Limitations

On the one hand, the neural network structure used in the experiment procedure was programmed by myself based on the understanding of the papers published by the developers of these two neural networks, rather than using the existed code segments provided by the developers. On the other hand, one factor that may cause the slow convergence rate of DenseNet compares to ResNet may caused by the depth of the neural network which was 121, much deeper than 34 layers ResNet and had more connections.

4. CONCLUSION

Through the comparison of the two experiments, it is obvious that ResNet-34 performs better than the DenseNet_BC-121 on the training and adaption of the fruit data set [1], since the efficiency of learning, the rate of fitting the data set is much faster and takes less time to return a resultant accuracy. It can be concluded that ResNet performs much better than DenseNet on simple data sets, since the ResNet avoids the disappearance of gradient through residual passing, so that the current layer is able to get both the input gradient and tensor of the previous layer and the output tensor and gradient. However, in the DenseNet, each layer needs to compute all of the input and output tensor and gradient of all the previous layers, when it is applied on those simple data sets, this might cause an increment in the computation complexity also it might blur the output processed tensor and gradient of the previous layer, so it resulted in only a limited improvement in its performance after 2000 times of training and might be the reasons why the using of DenseNet structure now is not as much as the using of ResNet structure. According to one of the developers of DenseNet [6], the structures of DenseNet are usually narrower but much deeper, so that the training process of DenseNet is more parameter-efficiency but less memory and speed-efficiency, which is mainly why the training process is slow and lack of increment in the accuracy.

ACKNOWLEDGEMENTS

It gives me great pleasure in acknowledging the support and help of Professor Juntao Ye. I would like to thank my parents for giving me support when I met difficulties. And everyone who gives me assistance when I met troubles.

REFERENCES

[1] Fruits-360, Version: 2018.07.01.0. https://www.kaggle.com/moltean/fruits. last visited on 12.07.2018.

[2] TensorFlow. https://www.tensorflow.org. last visited on 15.07.2018.

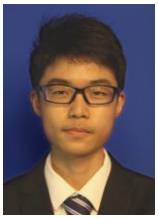
[3] M. Liang, X. Hu, Recurrent Convolutional Neural Network for Object Recognition, IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Boston, pp. 3367-3375, 2015.

[4] G. Huang, Z. Liu, K. Weinberger, L. Maaten, Densely Connected Convolutional Networks, 2017.

[5] keitakurita, Learning Rate Tuning in Deep Learning: A Practical Guide, 2018. http://mlexplained.com/2018/01/29/learning-rate-tuning-in-deep-learning-a-practical-guide/. last visited on 15.01.2019.

[6] Z. Liu, answering why DenseNet requires more memory in training, 2017. https://www.reddit.com/r/MachineLearning/comments/67fds7/d_how_does_densenet_compare_to_resnet_and/. last visited on 12.01.2019

AUTHORS



Ding Tianye, year 11 senior high student in Hangzhou Foreign Language School, has a month of summer program experience in National Laboratory of Pattern Recognition (NLPR), Institute of Automation, Chinese Academy of Sciences (CASIA). Currently the president of Developer Association, studying machine learning.