Efficient Methods of Training Convolution Neural Network

JH Ryu

University of Kyung Hee
KHLUG
2013103902@khu.ac.kr

Abstract— **This paper introduces overall development process of training methods, to simplest activate function from complex layer architecture and optimizer. We compare existing methods with evolved methods. We propose outstanding methods and training images which are efficient in improving accuracy and time consumption.**

I. INTRODUCTION

In 2012, Alexnet[1] have emerged in ILSVRC 2012 ,Convolution Neural Network(CNN) becomes master key for any machine learning applications. Although CNN was remarkable for its improved performance, larger image quantity and resolution has resulted in increased learning time and the inability to cope with traditional methods. As a result, the alternatives to numerous traditional methods emerged, resulting in faster training speed and improved training time.

In this paper , we find out problems and improvements to existing methods. Section 2 reviews neural network architecture about Alexnet,Vgg,Googlenet and Resnet.
Section 3 describes Mobile Network[2], Efficient and Light architecture. Section 4 describes optimization functions. Section 5 describes various activation functions. Section 6 provides a brief introduction to batch normalization and methods that are not covered in detail here.
Section 7 describes experiments on ImageNet,CIFAR10 with its use cases and applications. Section 8 closes with a summary and conclusions.

[1] ImageNet Classification with Deep Convolutional Neural Networks - https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks

[2] MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications - https://arxiv.org/abs/1704.04861

II. NETWORK ARCHITECTURE REVIEWS

(1) Alexnet

Alexnet, a classical structure and the first successful CNN, is a good guide for designing network architectures for following reasons.

Alexnet contains eight learned layers, five convolutional layers, and three fully-connected layers. A typical layer structure consists of various convolution size 3x3,5x5 and 11x11, local response normalization (batch normalization), and a Relu activation function. This architecture enables the learning characteristics of the image.

(2) VGG

Vgg is a network structure with a very simple network but deeper than alexnet. Usually, deeper layer improves the results, but it makes learning difficult. But according to the vgg they fix other parameters of the architecture, and steadily increase the depth of the network by adding more convolutional layers, which is feasible due to the use of very small (3×3) convolution filters in all layers. Three-layer 3×3 convolution stack cost 81% more efficient than a single 7×7 convolution.

(3) Inception (google net)

Inception (google net) is a network that attempts to simulate the structure of a human brain that dense calculation but sparse network connection. It is implemented as a concatenation of various convolution layers called inception modules. Also inception uses 1x1 convolution to reduce dimension reductionality. There are various versions of inception, the difference being that various techniques such as kernel size reduction and residual network are applied. Also, inception has evolved into xception, which is the first time to introduce depthwise separable convolution in mobile net below.

III. MOBILE NET

MobileNet is a light and compact network announced by Google. Unlike conventional convolution, MobileNet has the following feature.

MobileNet is based on depthwise separable convolution which is a combination of depthwise convolution and pointwise convolution. The depthwise convolution applies a single filter to each input channel. The pointwise convolution applies a 1x1 convolution to combine the outputs the depthwise convolution. This creates a linear combination of the output of the depthwise convolution.

In traditional convolution, if the size of the input feature map for any network is FxF, the number of input channels is N. The size of the kernel is KxK, and the number of output channels is M. The calculation amount of the network becomes F^2NK^2M .

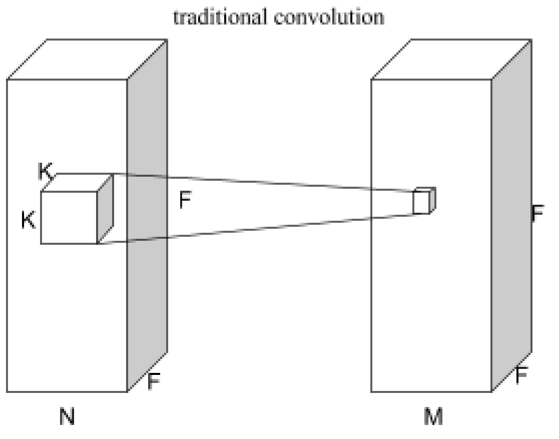


Figure:1 traditional convolutional network layer that has the computational cost of K · K · M · N · F · F

Depthwise convolution performs space direction convolution for each feature map channel. Since the convolution is not performed in the channel direction, the cost of one convolution is changed from K^2N to K^2 , therefore the calculation amount of the network is F^2NK^2 and the number of parameters is K^2N .

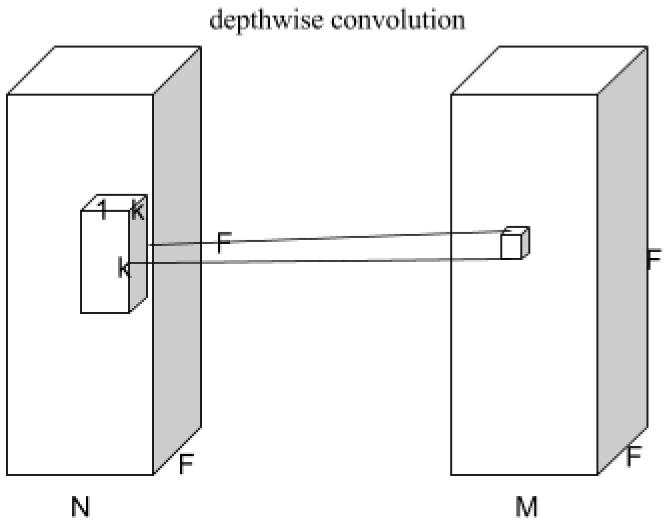


Figure:2 depthwise convolutional network layer that has the computational cost of K · K · M · F · F

Pointwise convolution is a 1x1 convolution. Every "pixel" in the feature maps at the output of the 1x1 convolution is just a linear combination of the "pixels" at the same position in the input feature maps. Convolution is not performed in the spatial direction but only in the channel direction. It is used to increase or decrease the dimension of the feature map. Since the pointwise convolution consists of K which is equal to 1. The calculation amount of this convolution is F^2NM and the number of parameters is NM .

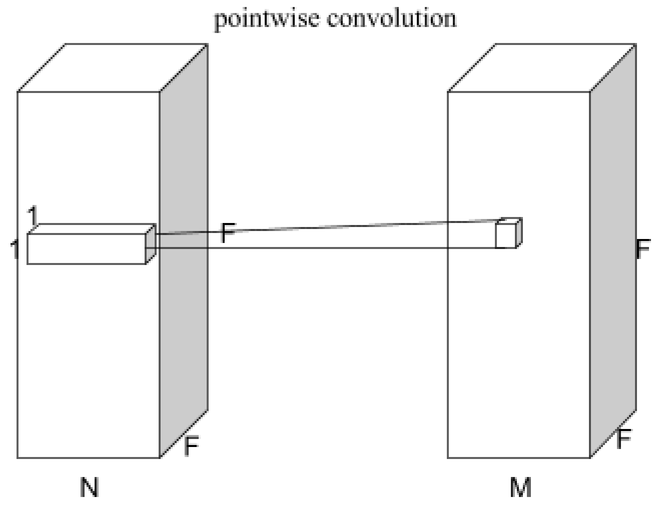


Figure:3 pointwise convolutional network layer that has the computational cost of M · N · F · F

The general convolution performs convolution simultaneously in the space direction and the channel direction of the feature map, while the pointwise convolution of the channel direction and the depthwise convolution of the spatial direction are factorized and calculated separately. This can be approximated with fewer parameters and computational complexity than the convolution which is a simultaneous convolution of spatial and channel directions. The amount of calculation is reduced from F^2NK^2M to $F^2NM+ F^2NK^2$. It is reduced to about 1/9 of formal method because reduction of computation is $\frac{1}{N}+\frac{1}{D_K{}^2}$, usually M>>K^2 .

The MobileNet performs batch normalization and two convolution operations in depthwise and pointwise in one convolution operation.

Unstructured sparse matrix operations are not typically faster than dense matrix operations until a very high level of sparsity. but 1x1 conv is general matrix multiply (GEMM) functions.

It also reduced the proportion of parameters present in the FC.

Table 2. Resource Per Layer Type

Type	Mult-Adds	Parameters
Conv 1 × 1	94.86%	74.59%
Conv DW 3 × 3	3.06%	1.06%
Conv 3 × 3	1.19%	0.02%
Fully Connected	0.18%	24.33%

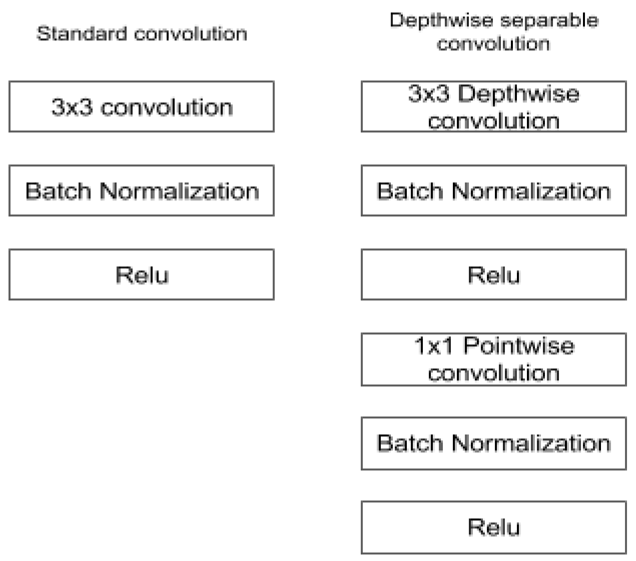


Figure:4 Left: Standard convolutional layer with batchnorm and Relu. Right Depthwise separable convolution with Depthwise and Pointwise layer also batchnorm and Relu.

IV. OPTIMIZER

The momentum function has shown a lot of improvement over SGD, but there is room for improvement. Because momentum updates the parameters equally, there is a need to make improvements by adjusting the update rate between the parameters. ADAM(Adaptive Moment Estimation) computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients v_t like Adagrad and RMSprop, Adam also keeps an exponentially decaying average of past gradients m_t, similar to momentum:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} (1-\beta_2) g^2_t$$

m_t and v_t are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively, hence the name of the method. As m_t and v_t are initialized as vectors of 0's, the authors of Adam observe that they are biased towards zero, especially during the initial time steps, and especially when the decay rates are small (i.e. β_1 and β_2 are close to 1). They counteract these biases by computing bias-corrected first and second moment estimates:

$$\widehat{m}_t = \frac{m_t}{1-\beta^t_1} \quad \widehat{v}_t = \frac{v_t}{1-\beta^t_2}$$

$f(x) =$

They then use these to update the parameters just as we have seen in Adagrad and RMSprop, which yields the Adam update rule:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\widehat{v}_t} + \varepsilon} \widehat{m}_t \quad .$$

The authors propose default values of 0.9 for β_1, 0.999 for β_2, and 10^{-8} for ϵ. They show empirically that Adam works well in practice and compares favorably to other adaptive learning-method algorithms.

V. ACTIVATION FUNCTION

a) Swish

The Relu function has been used for a long time as an activation function. However, due to the monotony of the derivative, the need for a new activation function has been consistently raised, and swish is an alternative. Swish is defined as $x \cdot \sigma(\beta x)$, where $\sigma(z) = (1 + \exp(-z))-1$ is the sigmoid function and β is either a constant or a trainable parameter. If β = 0, Swish becomes the scaled linear function $f(x) = x/2$. As $\beta \rightarrow \infty$, the sigmoid component approaches becomes like the ReLU function.
The derivative of Swish is

$$f`(x) = \sigma(\beta x) + \beta x \cdot \sigma(\beta x)(1 - \sigma(\beta x))$$
$$= \sigma(\beta x) + \beta x \cdot \sigma(\beta x) - \beta x \cdot \sigma(\beta x)\ 2$$
$$= \beta x \cdot \sigma(x) + \sigma(\beta x)(1 - \beta x \cdot \sigma(\beta x))$$
$$= \beta f(x) + \sigma(\beta x)(1 - \beta f(x))$$

A suitable β is known as a value between 1 and 2.

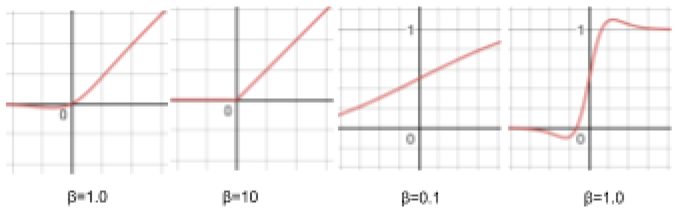
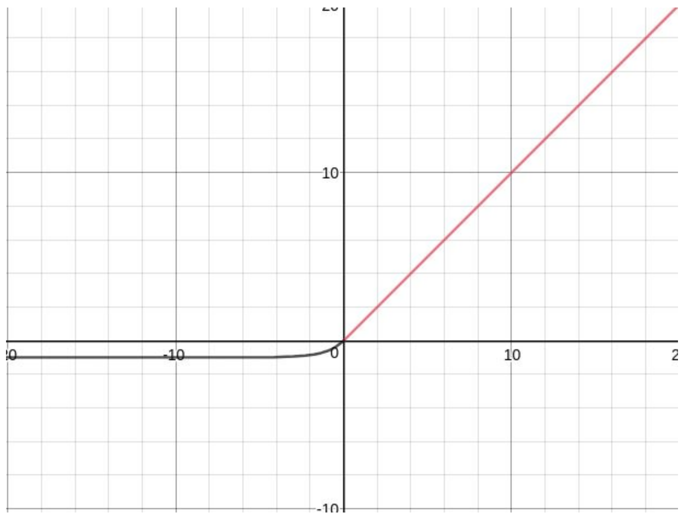


Figure 5: Left figures are Swish activation functions. Right are first derivatives of Swish.

b) Exponential Linear Unit (ELU)

Exponential Linear Unit (ELU) are alternatives to another Relu function. These functions also prevent monotony and increase nonlinearity like swish.

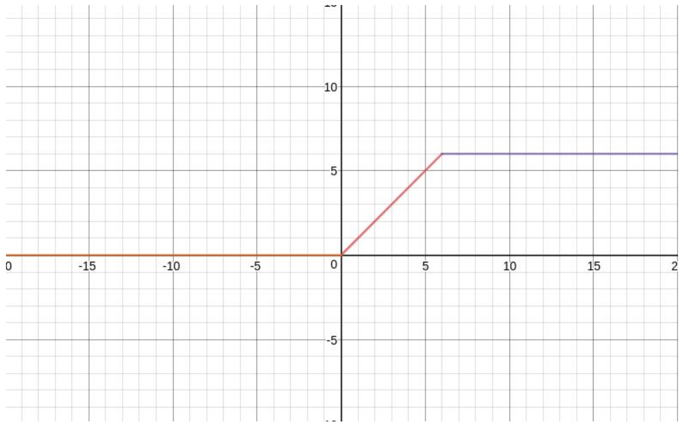


The ELU hyperparameter α controls the value to which an ELU saturates for negative net inputs. ELUs diminish the vanishing gradient effect as rectified linear units (ReLUs).

Concatenates a ReLU which selects only the positive part of the activation with a ReLU which selects only the negative part of the activation

c) Concatenated Rectified Linear Units(CLELU)

The CRelu function is a good alternative to the RELU function as well as the above function. The cRELU function simply truncates the value in the RELU function beyond a certain threshold.



VI. BATCH NORMALIZATION AND OTHERS

a) Bach normalization

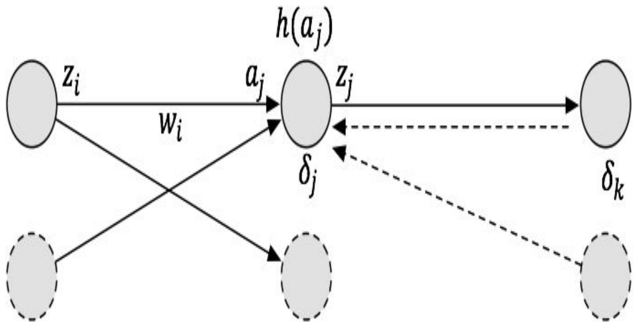
Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. It is refer to phenomenon as internal covariate shift, and address the problem by normalizing layer inputs. Batch normalization method draws its strength from making normalization a part of the model architecture and performing the normalization for each training mini-batch. Batch Normalization allows us to use much higher learning rates and be less careful about initialization, and in some cases eliminates the need for Dropout. Applied to a state of-the-art image classification model, Batch Normalization achieves the same accuracy with 14 times fewer training steps, and beats the original model by a significant margin.

b) Decoupled Neural interfaces using Synthetic Gradients

Now briefly introduce other methods. Training neural network Problems.

- Forward locking: no module can be process its incoming data before the previous nodes in the directed forward graph have executed
- update locking:no module can be updated before all dependent modules have executed in forward mode
- Backwards locking:no module can be updated before all dependent modules have executed in forward mode and backward mode

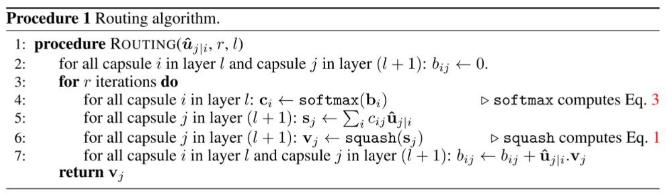
Decoupled Neural interfaces using Synthetic Gradients solved the problem of update locking.



The goal of this work is to remove update locking for neural networks. This is achieved by removing backpropagation. To update weights θi of module i we drastically approximate the function implied by backpropagation

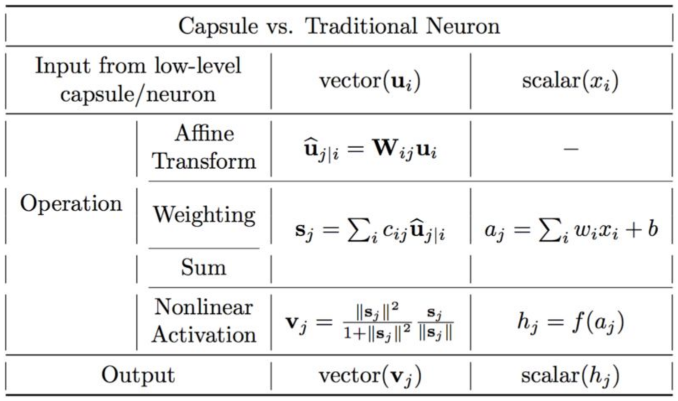
c) Dynamic Routing Between Capsules

CNNs Have Important Drawbacks cause pooling removes spatial properties. Dynamic Routing Between Capsules solves that problems. Capsules that perform some quite complicated internal computations(routing)



Procedure 1 Routing algorithm.

```
1: procedure ROUTING($\hat{\mathbf{u}}_{j|i}$, $r$, $l$)
2:     for all capsule $i$ in layer $l$ and capsule $j$ in layer $(l+1)$: $b_{ij} \leftarrow 0$.
3:     for $r$ iterations do
4:         for all capsule $i$ in layer $l$: $\mathbf{c}_i \leftarrow \texttt{softmax}(\mathbf{b}_i)$        ▷ softmax computes Eq. 3
5:         for all capsule $j$ in layer $(l+1)$: $\mathbf{s}_j \leftarrow \sum_i c_{ij}\hat{\mathbf{u}}_{j|i}$
6:         for all capsule $j$ in layer $(l+1)$: $\mathbf{v}_j \leftarrow \texttt{squash}(\mathbf{s}_j)$        ▷ squash computes Eq. 1
7:         for all capsule $i$ in layer $l$ and capsule $j$ in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i}.\mathbf{v}_j$
    return $\mathbf{v}_j$
```



Capsule vs. Traditional Neuron			
Input from low-level capsule/neuron		vector($\mathbf{u}_i$)	scalar(x_i)
Operation	Affine Transform	$\hat{\mathbf{u}}_{j\|i} = \mathbf{W}_{ij}\mathbf{u}_i$	–
	Weighting	$\mathbf{s}_j = \sum_i c_{ij}\hat{\mathbf{u}}_{j\|i}$	$a_j = \sum_i w_i x_i + b$
	Sum		
	Nonlinear Activation	$\mathbf{v}_j = \frac{\\|\mathbf{s}_j\\|^2}{1+\\|\mathbf{s}_j\\|^2}\frac{\mathbf{s}_j}{\\|\mathbf{s}_j\\|}$	$h_j = f(a_j)$
Output		vector($\mathbf{v}_j$)	scalar(h_j)

VII. EXPERIMENTS

a) Network

The following experiments were conducted using a VGG network that simple but good performance. The data set used in the experiment is a CIFAR-10 dataset, one of the well known data. The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The environment of the computer undergoing the experiment is as follows.

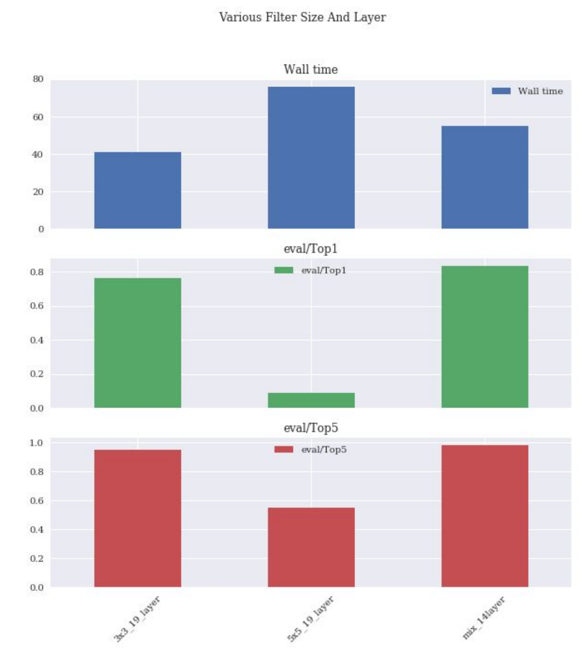
- CPU: Intel(R) Core(TM) i7-6850K CPU @ 3.60GHz
- RAM: DDR4 128GB
- GPU: Nvidia GTX 1080 TI x4

The hyperparameters for learning are as follows.

- keeping dropout probability 0.2
- image size 32
- 20000 step
- learning rate 0.0002
- optimizer ADAM

The following three experiments were conducted.

1. 19 layer using only 3x3 convolution
2. 19 layer using only 5x5 convolution
3. 14 layer that mixes various convolutions

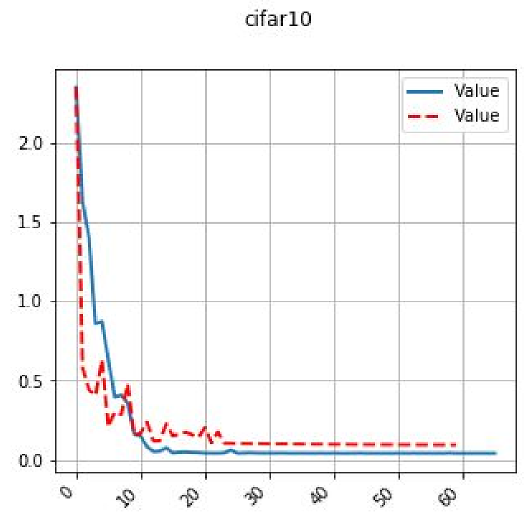


The following results can be obtained through the above experiment. As mentioned in the VGG, 3x3 operations yield sufficient results. Also, we can see that convergence does not work well when the size of the kernel increases. We reduced the depth of the layer and increased the size of the kernel to increase the accuracy, but finally confirmed that the execution time (computation cost) increased from 19 layers.

b) Optimizer

The device and dataset used in this experiment are the same as the previous experiment, and the specific hyper parameters of the learning are as follows.

- alexnet
- 300000 step
- learning rate 0.005

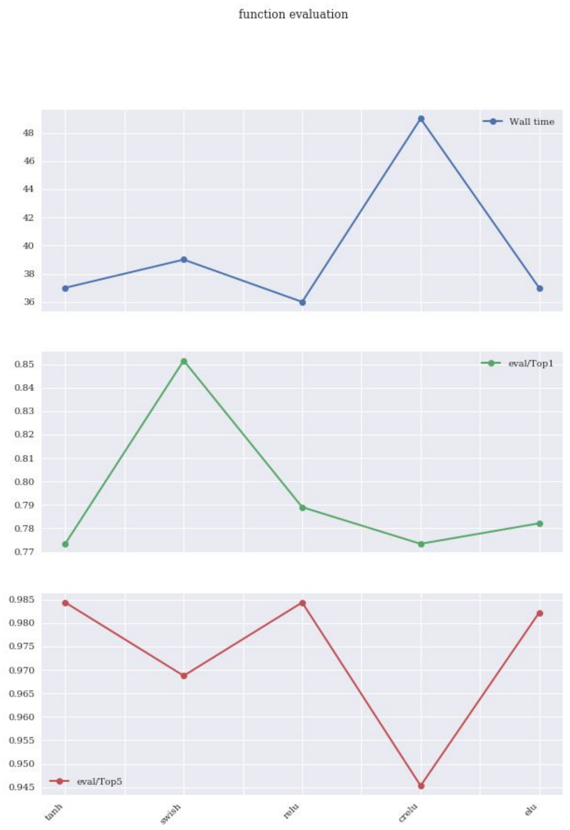


red line is momentum, blue line is Adam

The experimental results show the characteristics of the optimizer function. It can be seen that Adam (red line) converges faster and more accurately with less vibration than momentum (blue line). This phenomenon can be regarded as preventing the oscillation due to Adam's second momentum. In addition, we found that the momentum algorithm was faster in convergence at the beginning. This phenomenon is because the previous gradient is not sufficiently reflected at the beginning due to the exponential average factor beta.

c) Activation Function

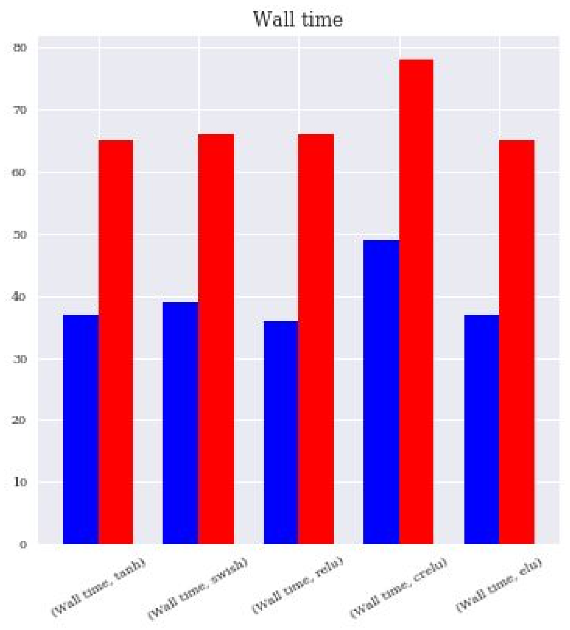
This experiment was conducted by changing only the activation function in the first 19 layer using only 3x3 convolution.



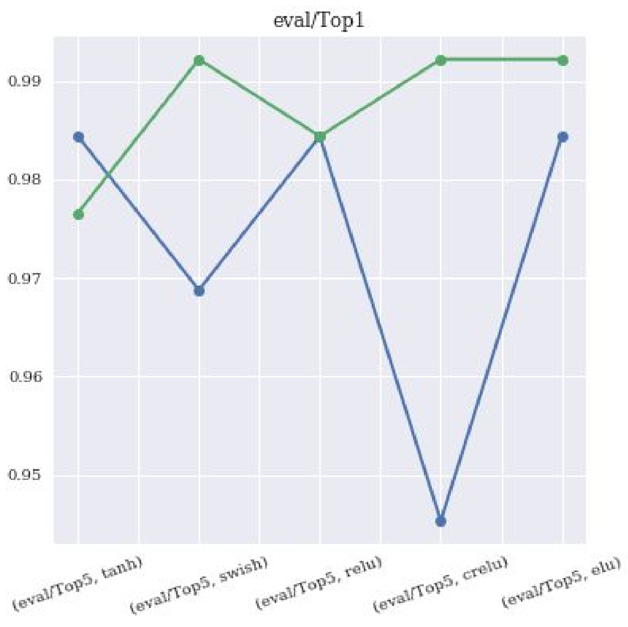
Most functions (except swish) showed worse results than Relu functions, especially the cRelu function. This show that Relu is on average the best function, but it implies that there may be other alternatives.

d) Batch Normalization

This experiment was also done in the first 19 layer using only 3x3 convolution and only the batch normalization function was added in the previous experiment.



red adopted batch norm, blue is not adopted batch norm



greed is adopted batch norm, greed is not adopted batch norm

In this experiment, we used LRN, a function used in Alexnet. The results of the experiments were not so different from the theories. The execution time increased nearly twice as a whole, but the tendency of increasing accuracy was confirmed. This tendency is evident in Crelu, which is presumably due to the internal covariance shift problem when using crelu.

VIII. SUMMARY

Since CNN is a really effective technique for image learning, many studies have been conducted to effectively perform learning. There are a number of methods, but in this paper, three representative methods are presented.
Starting from the approach of changing the structure of the network, the use of more advanced optimization functions and the use of sophisticated activation functions have confirmed that cnn learning can proceed more efficiently.

IX. REFERENCES

[1] Adam: A Method for Stochastic Optimization Diederik P. Kingma, Jimmy Ba Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015 arXiv:1412.6980
[2] An overview of gradient descent optimization algorithms Sebastian Ruder arXiv:1609.04747
[3] Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift Sergey Ioffe, Christian Szegedy arXiv:1502.03167
[4] ImageNet classification with deep convolutional neural networks Proceeding NIPS'12 Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1 Pages 1097-1105
[5] Very Deep Convolutional Networks for Large-Scale Image Recognition arXiv:1409.1556
[6] Searching for Activation Functions Prajit Ramachandran, Barret Zoph, Quoc V. Le arXiv:1710.05941
[7] Rethinking the Inception Architecture for Computer Vision Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna arXiv:1512.00567
[8] MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam arXiv:1704.04861
[9] Going Deeper with Convolutions Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich arXiv:1409.4842
[10] Dynamic Routing Between Capsules Sara Sabour, Nicholas Frosst, Geoffrey E Hinton arXiv:1710.09829
[11] Decoupled Neural Interfaces using Synthetic Gradients Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, Koray Kavukcuoglu arXiv:1608.05343
[12] Xception: Deep Learning with Depthwise Separable Convolutions François Chollet arXiv:1610.02357