Coursework 4 : Convolutional Neural Network in CIFAR-10

Aldy Syahdeini – s1575408

April 10, 2017

---

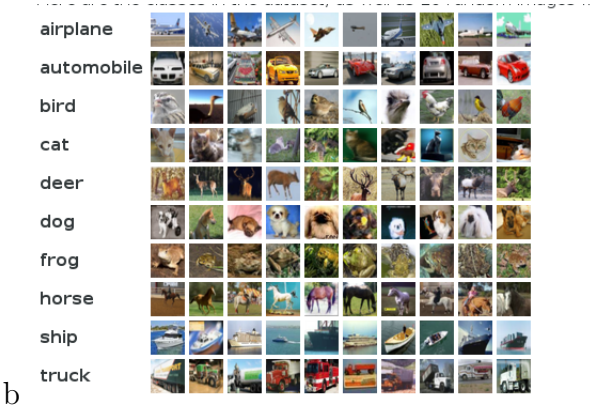
1 INTRODUCTION

1.1 CIFAR Dataset

Cifar is a dataset of images which cosist of 6000 images. Each image has 32x32 dimension with 3 channels (R,G,B). CIFAR-10 consist of 10 classes of image (airpline, cat, frog etc). In our dataset we has 40000 images data for trainig and 10000 images data for validation. The example of the images and it classes can be seen in figure 1. More information abut the dataset can be found here `https://www.cs.toronto.edu/~kriz/cifar.html`

1.2 Convolutional Neural Network

Convolutional Neural network (CNN) is a type of network architecture that is recently very popular for image recognition. CNN is basically similar to multi layer neural network, the only difference is CNN's neurons arranged in 3 dimension (width x height x channel).
there are three main properties of convolutional neural network :



b

Figure 1: CIFAR 10 Dataset

b

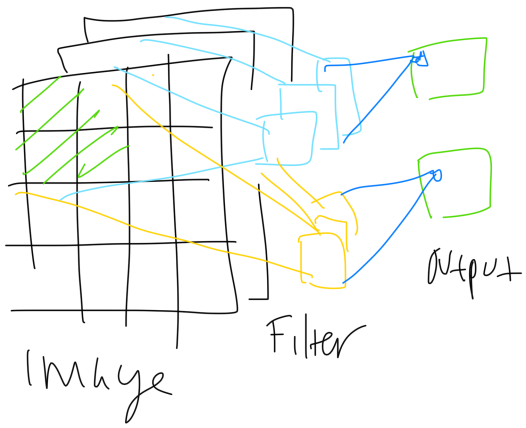


Figure 2: Convolutional layer

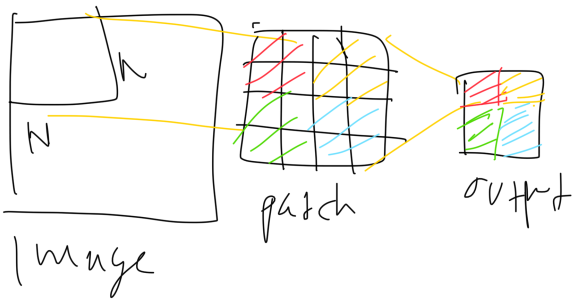


Figure 3: Convolutional layer

- Local receptive fields = hidden units connection to local path on the layer below which is important to features in an images.

- weight sharing = a way to look same feature at different point in an image, we call this as feature maps.

- Pooling = a layer to condense information from previous layer.

CNN usually consist of two main layer, Convolutional layer and pooling layer. Convolutional layer works by convoluted KxK patch of our image with KxK kernel matrix. then we slide/stride S-pixel to the next patch and put all the result in N-feature maps. Padding is also used to increase the size of the output. Example of Convolutional layer with 2 features map can be seen in Figure 2. In the figure 4x4 kernel is used for 3 channel image and it produce 2 feature map

Pooling is a way to condense an image, it operates independently on every depth slice of the input and resizes it spatially. there are two typical type of pooling layer which is average pooling and max pooling. Example of pooling layer can bee sen in Figure 3. As it seen, the pooling size is 4x4

1.3 Rectier nonlinearities (Relu)

Glorot et al. (2011) found that Relu perform so much better than sigmoid function on image recognition. Relu address the problem of vanishing gradient (Ben-gio et al., 1994) which will occur when the higher layer units are staurated at -1 or 1 which will be result in the gradient

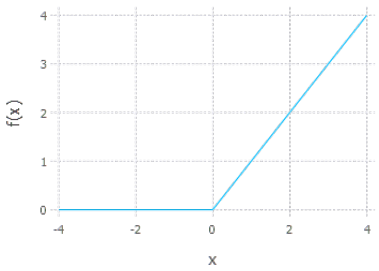


Figure 4: Relu function

$$h^{(i)} = \max(w^{(i)T}x, 0) = \begin{cases} w^{(i)T}x & w^{(i)T}x > 0 \\ 0 & \text{else} \end{cases}.$$

Figure 5: Relu formula

to be 0. This sorf of problem happened in sigmoid and tanh function which will result in covergence in a poor local minimum [1]. Relu is a simple function that will clipped the negative value to zero and pass the posive value as shown in Figure 4 and the equation in figure 5.

From the equation of the relu we can see that dot product between the weight (w) and input (x) if the output is more than 0, otherwise the function will clapped it to 0. the weight is used in the relu layer implementation.
Because we use relu as our activation function that capture the most salient feature of an object, we can think as each patch of relu as a neuron that will detect an interesting part of the image, I think its similar to the theory in cognitive science where human visual attention only focus on the most salient part of an image [2]. Relu has disadvantage because the gradient is 0 if the unit is not active (the output is below zero) which will result in the unit will never activates (Dead unit).

1.4 Leaky Relu

To overcome the Dead unit problem of relu layer. Many people try to suggest different approach [3]. One of the approach is using Leaky Relu. Leaky Relu is basically similar with Relu but instead of hammering the value to zero, this function use a which is a constant to reduce the negative value of the unit. Figure 6 shows how different leaky relu handle negative input and figure 7 show the mathematical function for leaky relu. at the function x is an input and a is a constanta.

1.5 Local response Normalization (LRN)

Because Relu have a advantage to prevent unit from saturating. and the unit produce a positve input to relu that learning will only happen in that neuron. LRN helps to add generalization

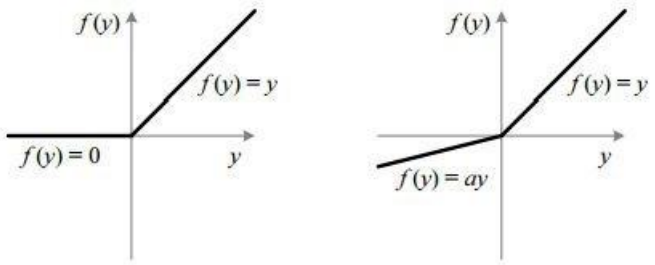


Figure 6: Different between relu (left) and leaky relu (right)

$$y_i = \begin{cases} x_i & \text{if } x_i \geq 0 \\ \frac{x_i}{a_i} & \text{if } x_i < 0, \end{cases}$$

Figure 7: Leaky Relu formula

for all the neuron by adding normalization for all the neuron. This will create competition for big activities amongst neuron outputs computed using different kernels. The LRN is a function that usually put after max-pooling inside a layer. ImageNet is one of the model that use this function. Figure 8. shows the LRN function where N is the number of kernel, and k, n, α, β is the hyper-parameter. the function works for each units of our input.

1.6 Dropout

Dropout is one of the reguralization method to overcome overfitting. Dropout works by temporarly removing out units and all its connection for the training on N^{th} epoch. The dropping out is decide with fixed probabily p independent with other units. Dropout is introduce by Hinton et all. [4], Figure 9. shows how dropout works in neural network.

2 Motivation

2.1 Previous Work

Based on the previous work in coursework three where we try different architecture of deep neural network. We use the previous work to build some prior assumption for this work.

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta}$$

Figure 8: LRN function

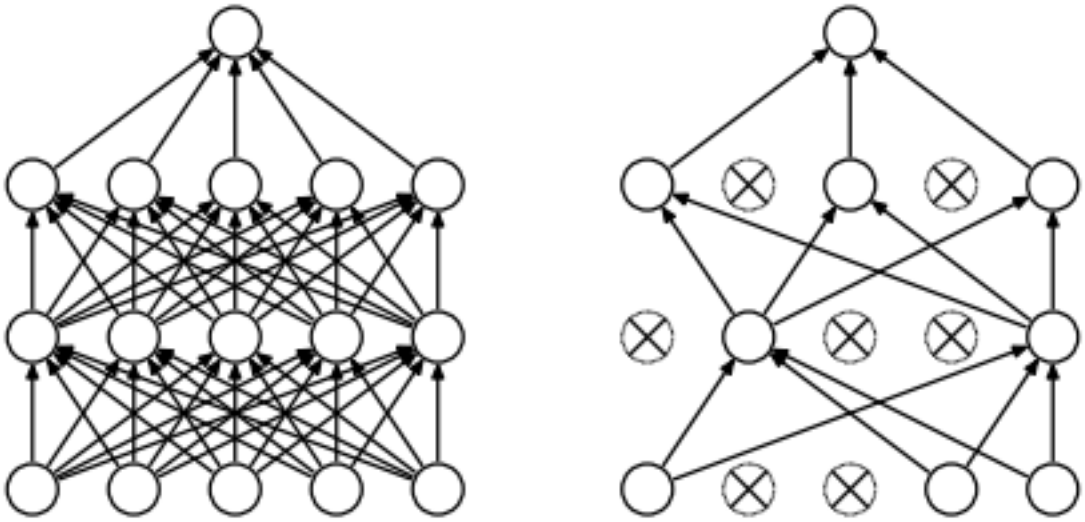


Figure 9: Dropout sample

- In the previous work, we found that too much hidden unit will have the same effect as smaller one. so at this experiment we will only use small number of hidden unit and filter (no more than 200).
- We also found that it will be better to build a more narrow network size at the end (big to small size).
- Moreover, we found that using elu instead of relu is better for the model.
- AdamOptimizer has the best performance compare to another optimizer methods
- And by putting dropout and normalization in our model it will improve their performance.

2.2 Relevant Research

There are many research has been tried to build a perfect model to predict CIFAR-10 dataset This web page provide us with the rank board of the method and its corresponding accuracy. One of the best model so far is GoogleNet which use inception as a layer. inception simply a layer that can dynamically choose the size of the patch and it can use more than one kernel in single convolution and stack it as a features.
Many of the current method use sparse convolutional neural network which transform the image into more spares matrix and also they use global normalization and whitening before processing the image.
Most of the experiment took a long time to do, so at the end this experiment we will try to use several single method from several paper and try to combine it. We will implement LeNet5 and Network in network architecture to better understand about the benefit of their Architercture over our model.

3 Experiment

3.1 Finding the right hyper-parameter

The hyper-parameter is very important as a starter point and as a properties on where our model inside the solution space. example of two dimensional solution space can be seen in Figure 19, there is local optima and global optima, and with the proper hyper-parameter we wish our hyper-parameter make a good start so we can find the global optima faster.
Finding the right hyper-parameter to use in an experiment is like finding the needle inside the haystack. There are some method available to find a perfect hyper-parameter. We will also see that different value of hyper-parameter will result in significant result.

- Manual Search: Finding the hyper-parameter based on our knowledge on the domain of the problem. Guessing and brute-force until we find the good parameter.
- Grid Search: Finding the hyper-parameter by testing a specific range of hyper-parameter. Then train the model using every combination of the hyper-parameter.
- Random Search: Like grid search we use knowledge of the problem to identify ranges for the hyperparameters. Rather than use a range we select them at random.
- Bayesian Optimization: This method is quite recent and it is proposed by Adams et al. It use the Gaussian Process (GP) to find the best hyper-parameter which use an information from the previous experiment to decide what parameter to try next.

3.1.1 Experiment objective

As we know that CNN has many hyper-parameter, so in this experiment we are trying to find the best weight initialization (mean and standard deviation) and learning rate.

3.1.2 Experiment properties

We are using grid search and we are using 2-layer convolutional because learning too complex architecture will take longer time to train and using 1-layer will get easily to overfit the data (shown in the next experiment). We also only use only 30 epoch, because at 30 epoch the plot is already differentiable and it safe times.
We use tf.nn.conv2d with kernel size 5x5 and 14 filters and padding is used.
We are evaluating :

- the learning rate from 0.0001 to 0.005.
- the initialization of weight, mean from 0 to 0.5
- the initialization of weight, standard deviation from 0.1 to 1.0.

3.1.3 Result

From figure 15,16,17 and 18 it is very clear that the optimal hyper-parameter is 0.0 for the mean, 1.0 for standard deviation and 0.005 for learning rate. if we choose 0.5 for the mean, 0 for standard deviation and 0.0005 for the learning rate for our model, we can see that from the figure that our model will have a very large error, which will mean that our weight value is very far from the global optimum. This makes our model underfit.

The 30^{th} Epoch Accuracy and Error result for validation and Training data

Weight Initialization		learning rate	Training		Validation	
mean	std dev		Accuracy	Error	Accuracy	Error
0.0	0.1	0.0005	98%	0.041	63%	2.73
0.0	0.1	0.0001	85%	0.44	64%	1.51
0.0	1.0	0.0001	29%	87.7	27%	96.27
0.0	1.0	0.005	10%	2.30	10%	2.31
0.5	1.0	0.005	9%	2.33	10%	2.37

3.1.4 Conclusion

Finding the good hyper-parameter of our model is very very important because it will deliver us to the right direction or faster movement to global optima. while choosing a wrong hyper-parameter will make our model loss it's direction to global optima or even stuck in local optima. We found that using 0.0 and 0.1 for mean and standard deviation for weight initialization and $5 * 10^{-4}$ for learning rate.

3.2 Experimenting with the size of layer

In this experiment we are addressing how many convolutional layer fits our data. This experiment are motivated by the previous coursework, we found that by using more than two layer it will make our model underfit. We try to test the hypothesis on how many convolutional layer makes our model more complex and underfit, and how many convolutional layer is optimal.

3.2.1 Experiment properties

in all this experiment we use the optimal hyper-parameter that we found in the previous experiment. Using 0.0 and 0.1 for mean and standard deviation for weight initialization and $5 * 10^{-4}$ for learning rate. Adam optimizer is also used in this experiment.
We use similar convolutional layer for all model. Each layer consist of

- Convolutional layer with 5x5 kernel with 1 stride and 14 filters.
  We use $tf.nn.conv2d(input, kernel, [1, 1, 1, 1], padding =' SAME')$, here padding is SAME because we put extra padding in our image when convoluted it with kernel.
  and for the kernel we use tf.nn.Variable with shape [5,5,14,14].
- relu layer
  Relu layer is built using function $tf.nn.relu$

- max pooling with 3x3 kernel size and 2 stride with padding
  we use function $tf.nn.max\_pool(inputs, ksize = [1, 3, 3, 1], strides = [1, 2, 2, 1], padding =' SAME')$, This layer will reduce the width and height to a half size from original.

3.2.2 Experiment objective

In this experiment our focus is to increase the number of layer until we found the perfect size for our model.

3.2.3 Result

Size of Layer	Training		Validation	
	Accuracy	Error	Accuracy	Error
2-Conv Layer	99%	0.20	66%	5.143
3-Conv Layer	82%	0.49	63%	1.26
4-Conv Layer	80%	0.53	66%	1.41
5-Conv Layer	39%	1.51	38%	1.646

The result shows that by using more than 4-convolutional layer our model become underfit. While 2,3 and 4 convolutional layer shows a great result.
Figure 20 and 21 shows an interesting result, where model with only a 1-layer and 4 layers shows a very bad performance compared to the others. It looks like the model reach convergence very slow compared to other. Also 2-Conv layer is too overfit the data. As a result 4-Convolutional layer is the best architecture for CIFAR-10

3.2.4 Result

Using too complex architecture of our model make it performance worse and take longer time to train and reach convergence. But using too simple architecture make our model easily underfit or overfit. On this point forward we use 4-Conv layer architecture because we found that it has the best performance among other

3.3 Using LeNet5 architecture to improve our model

3.3.1 leNet5

leNet is an architecture that is introduce by Yann LeCun et all. (1998) it is originally used as document recognition. Figure 10 shows the lenet5 architecture where t use 2-layer convolutional neural network and 2 fully connected network. In this experiment we use lenet5 to test if our model.

3.3.2 Inspired lenet5 Architecture

In this experiment we use the same architecture as lenet5 with 2-convolutional layer, we use relu as activation function for convolutional layer and dense layer. and use cross entropy for the last layer instead of RBF function.

Input size	Layer
32x32x3	1^{st} convolutional, 5x5 kernel 6 feature maps
32x32x6	Relu
32x32x6	Maxpooling, 2x2 kernel and 1 stride
16x16x16	2^{nd} convolutional, 5x5 kernel 16 feature maps
16x16x16	Relu
16x16x16	Maxpool, with kernel 2x2 and 2-stride
8x8x16	Dense layer (Relu) with 16 hidden units
8x8x16	Dense layer (Relu) with 120 hidden units
8x8x120	Dense layer (Relu) with 84 hidden units
8x8x12	SF

3.3.3 Result

Figure 22 and 23 shows that our model is very overfitting. at 120^{th} epoch the training accuracy and error is 97% and 0.069 respectively. While the validation accuracy and error is 58% and 4.69 respectively. the best validation accuracy it can get is 62% at epoch 60^{th}. this result is quite disappointing since the overfitting problem still happen since our last experiment, and the performance is worse. In the next experiment we will overcome this problem by using dropout.

3.4 Using ImageNet Architecture for better model

3.4.1 ImageNet

ImageNet architecture is proposed by hinton et. all. (2012) [5], they that the accuracy of the model can reach 89% for non-augmented data. I try to implement the ImageNet in our 4-layer model. In the paper they make a model with two path and combine it at the end. And from the visualization of the kernel on the first convolutional layer, it looks like each part produce different feature of the image. In this experiment we will use only one path and we hope by implementing the LRN and Dropout it will bring a better performance and it will overcome the overfitting . In the paper they introduce Local Response Normalization (LRN) which we already explain in the introduction.

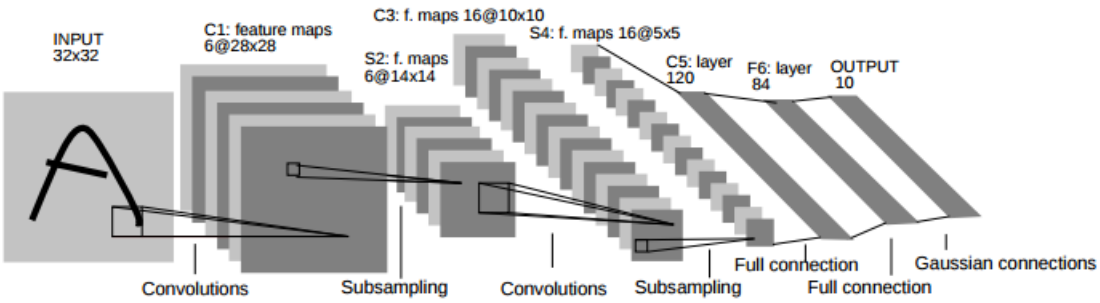


Figure 10: leNet5 architecture

3.4.2 Insipired ImageNet Architecture

Input size	Layer
32x32x3	1^{st} convolutional, 5x5 kernel 48 feature maps
32x32x48	Relu
32x32x48	LRN
32x32x48	2^{nd} convolutional, 5x5 kernel 128 feature maps
32x32x128	Relu
32x32x128	Maxpool, with kernel 3x3 and 2-stride
16x16x128	Droput, dropout probabilty 0.5
16x16x128	3^{rd} convolutional, 5x5 kernel 192 feature maps
16x16x128	Relu
16x16x128	4^{th} convolutional, 5x5 kernel 192 feature maps
16x16x128	Relu
16x16x128	Maxpool, with kernel 3x3 and 2-stride
8x8x128	Dense layer
8192x1	Softmax

For an implementation of LRN we use hyper-paramaterer from the paper [5] $k = 2, n = 5, \alpha = 10^{-4}$ and $\beta = 0.75$. Implementing it in tensor flow using function tf.nn.lrn(input, 5, 2, 0.0001, 0.75).
For dropout we use tf.nn.dropout with drop out probability 0.5 for training and we don't use dropout for validating.

3.4.3 Result

The result in Figure 24 and 25 shows that by implementing following imagenet architecture and implementing RLU and dropout make our model to have **better performance and it overcome our overfitting problem**. Our result shows the accuracy **69% and 2.30** error for validation set. Which can be seen in figure 10 and 11. From the plot we can see that after epoch 38 the performance of our model is drop significantly to 10%, at this point the best solution is to use **early stopping**. We stop training our model after we found the best performance and use the model with 38 epoch.

3.5 Using Data Augmentation and Leaky Relu in ImageNet

- Data Augmentation Data augmentation is a preprocessing our image data to add some image processing such as distortion or transpose of an image. Data Augmentation is the way to prevent the network from overfitting. The Recent paper from Baidu [6] shows that data augmentation improve performance of the deep learning. At this experiment we will try to use data augmentation before feed our first imagenet layer.

- Leaky Relu Leaky relu is already explained in introduction, here we are implementing leaky relu by 100 for alpha because it recommended in the original paper [3]. and we use

Figure 11: Original image

Figure 12: flipped image

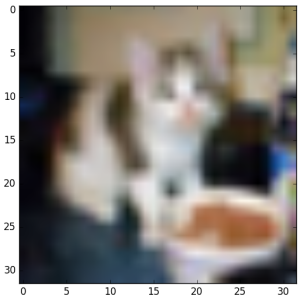


Figure 13: Original image

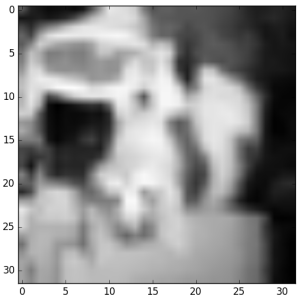


Figure 14: flipped image

$tf.maximum(1.0/alpha * inputs, inputs)$ as our leaky relu function. so the value will get clipped to $1/alpha$ rather than 0.

3.5.1 Experiment properties

For data augmentation first we convert the data into grayscale because it will make the preprocessing of the data faster. we use tf.image.rgb_to_grayscale. then we flip our image on verticaly or horizontaly. flipping an image is executed with probability 0.5 for both independently.
We use tf.reverse to flip an image.
Figure 11 and 12 show the original image and the flipped on vertical axis. and figure 13 and 14 shows the original and flipped image on horizontal axis.

3.5.2 Result

Model	Training		Validation		Epoch
	Accuracy	Error	Accuracy	Error	
Imagenet	92%	0.234	69%	1.65	38
Imagenet with data augmentation and leaky relu	93%	1.64	75%	0.193	60

From figure 26 and 27 and the table above it shows that using data augmentation improve our model performance and it make our model able to learn the data longer. We got 75% and 0.193 for the validation accuracy and error. Our model can still learn the data after 30^{th} but it

stops learning after 58^{th} epoch. To resolve this we can use early stopping.

4 Summary

In summary, In this coursework we are doing several experiment for data CIFAR-10. We found that different hyper-parameter will result in very big difference for our model. We also found that using too small layer will result in overfitting or underfitting. But with too complex layer will result in slow learning performance for the model. We also try to implement lenet5 and imagenet. we found that lenet5 is very overfit the data and implementing imagenet shows a great result, and we try to add data augmentation, dropout, local response normalization and laeky relu to our model which shows a great result in preventing overfitting and improve our model performance. Our best model so far is using imagenet architecture with data augmentation and leaky relu. we set our weight with mean = 0, standard deviation 0.1 and learning rate $5 * 10^{-4}$. we also use adam optimizer.
the best performance so far for validation set is 75% and 0.193 for the accuracy and error.

References

[1] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. *Rectier Nonlinearities Improve Neural Network Acoustic Models*. Computer Science Department, Stanford University, CA 94305 USA

[2] Neil D. B.Bruce, John K. Tsotsos *Saliency, attention, and visual search: An information theoretic approach*. Journal of vision 2009.

[3] Bing Xu, Naiyan Wang, Tianqi Chen, Mu Li *Empirical Evaluation of Recti ed Activations in Convolution Network*.

[4] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*.

[5] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton *ImageNet Classification with Deep Convolutional Neural Networks*.

[6] Ren Wu, Shengen Yan, Yi Shan, Qingqing Dang, Gang Sun *Deep Image: Scaling up Image Recognition*. Baidu Research

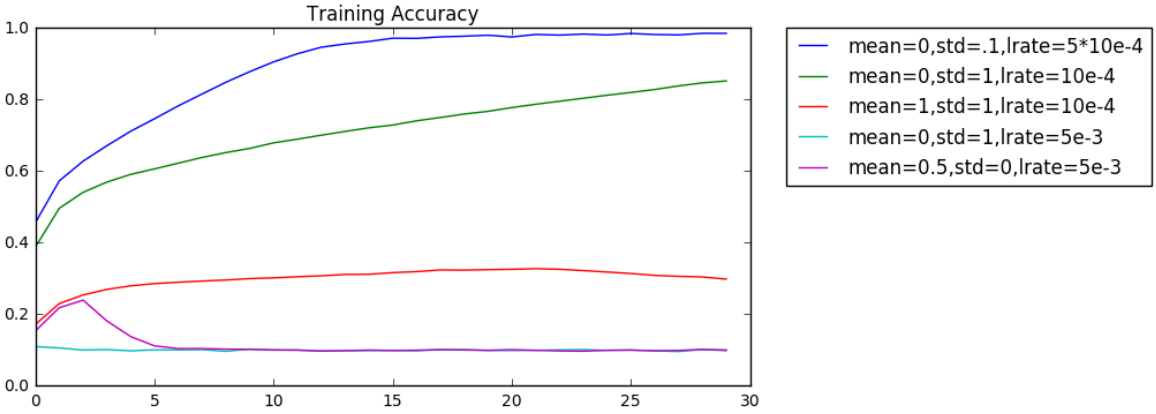


Figure 15: Hyper-parameter training Accuracy

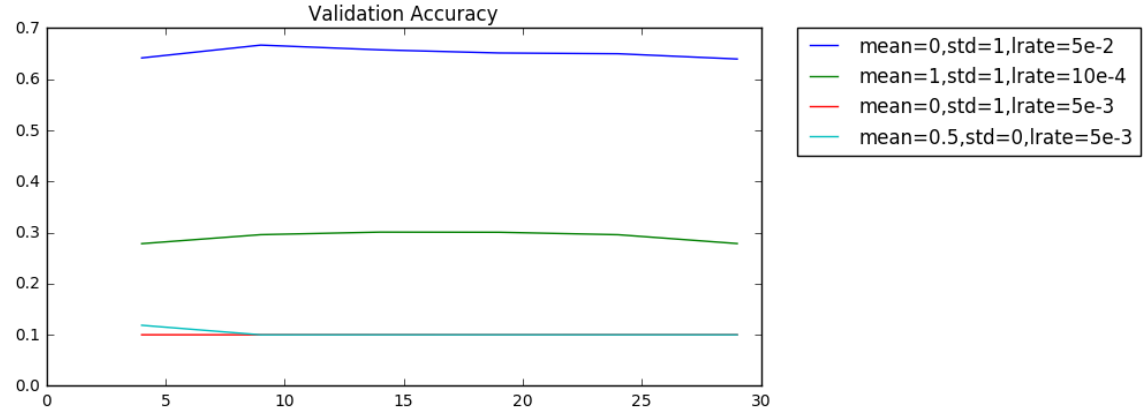


Figure 16: Hyper-parameter training Error

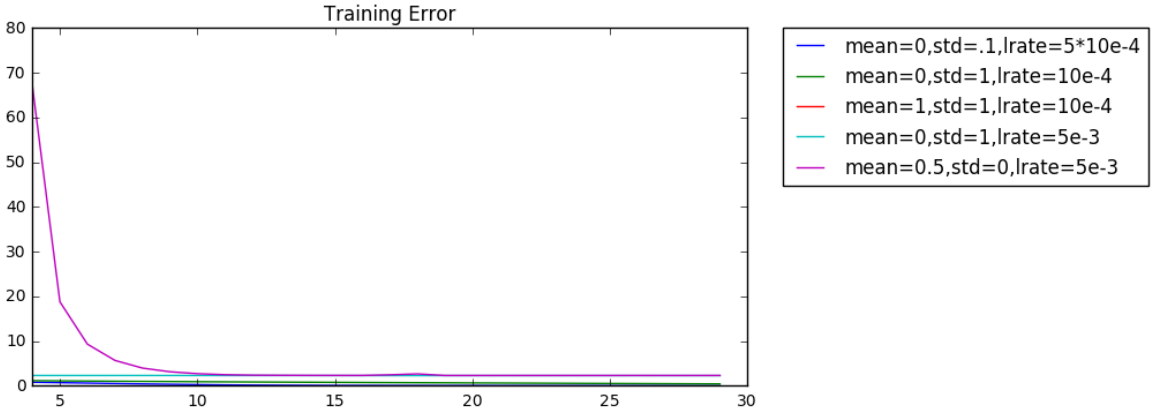


Figure 17: Hyper-parameter training Error

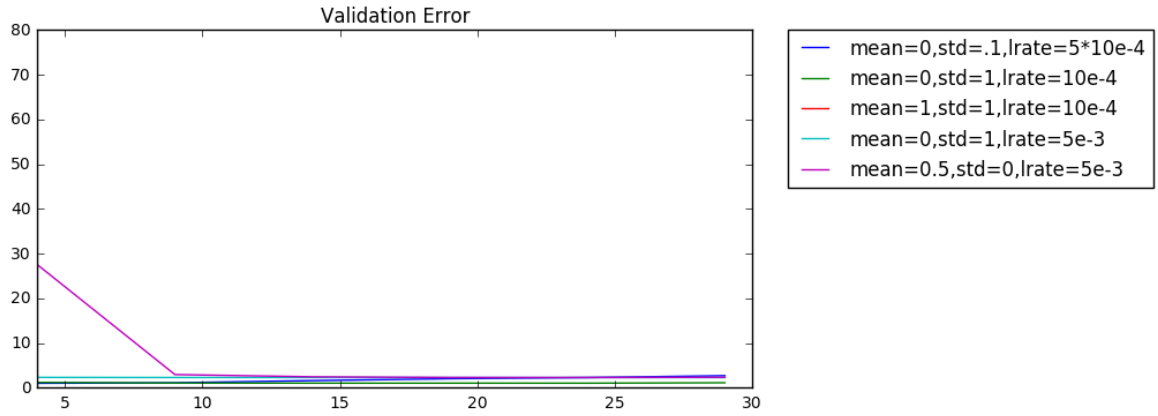


Figure 18: Hyper-parameter validation Error

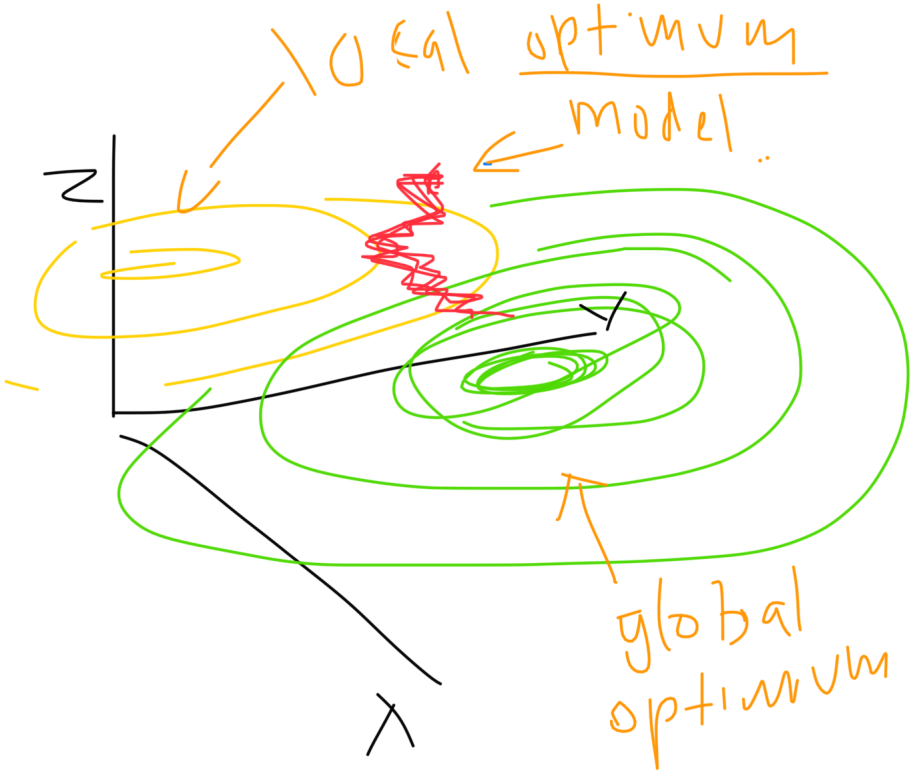


Figure 19: Solution space

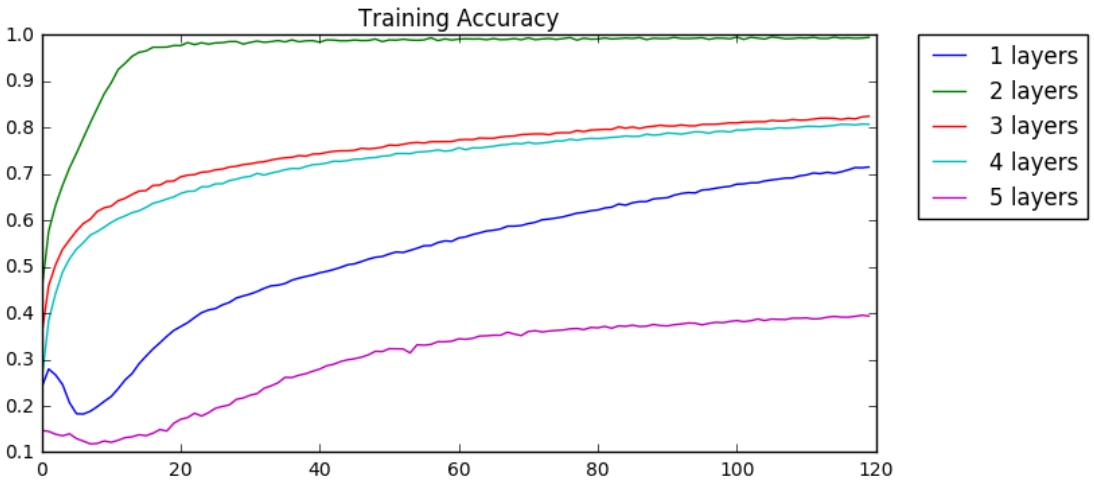


Figure 20: Layer experiment training Accuracy

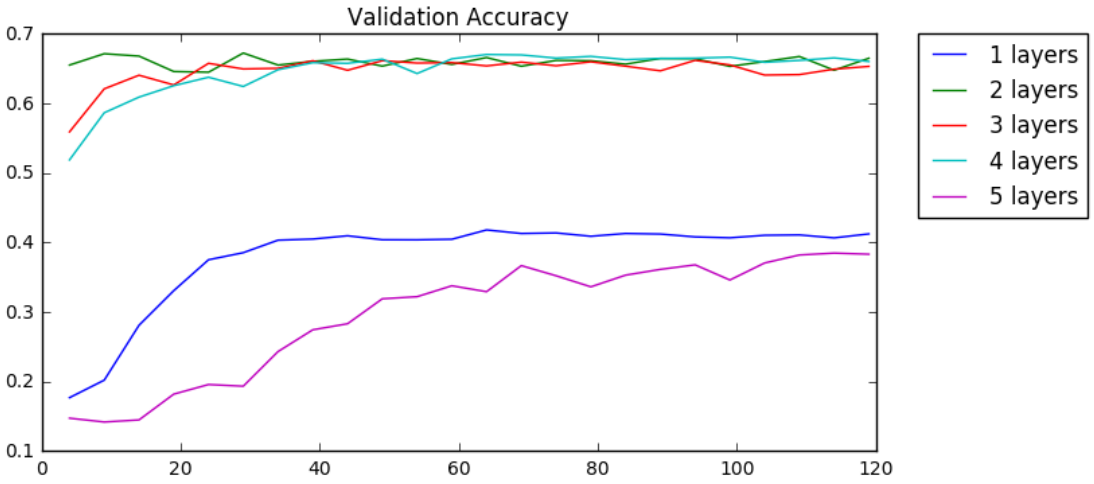


Figure 21: Layer experiment training Accuracy

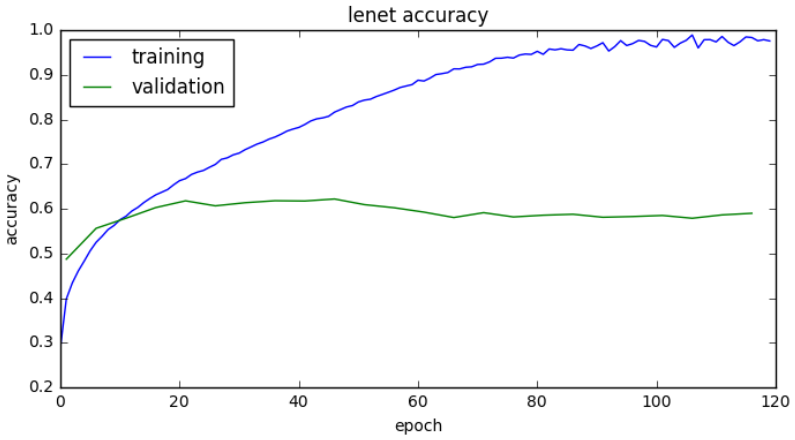


Figure 22: leNet5 accuracy

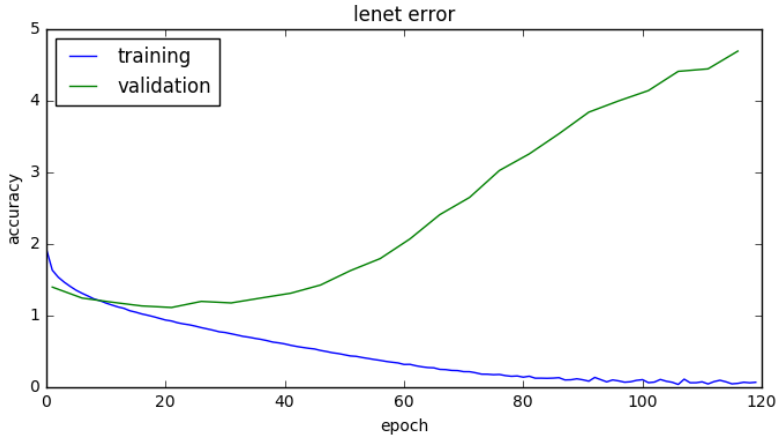


Figure 23: leNet5 error

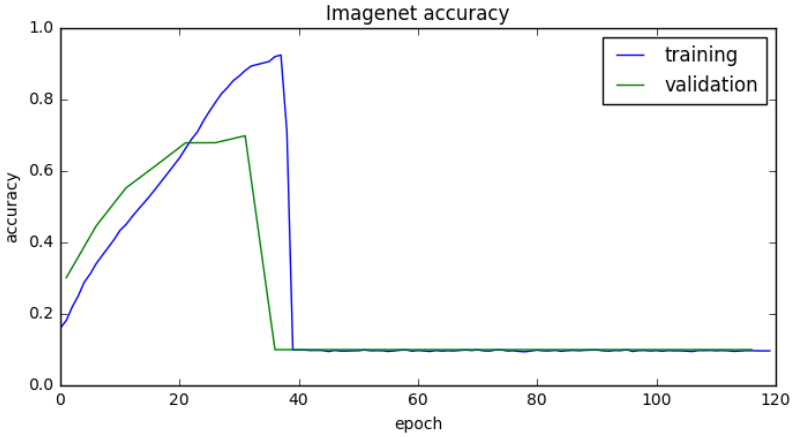


Figure 24: Image Net accuracy

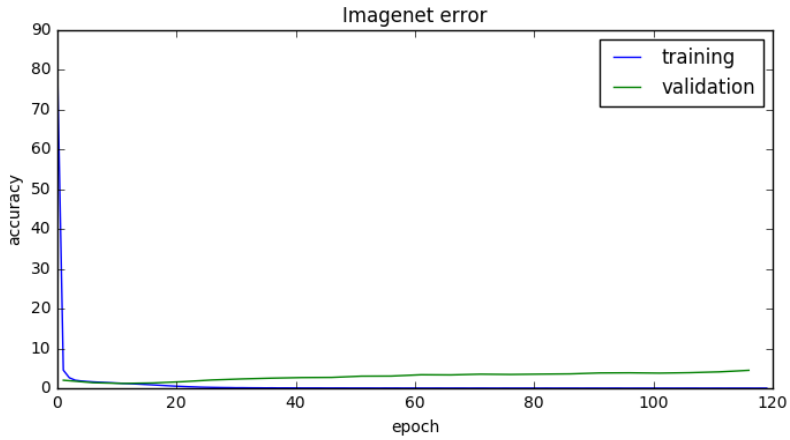


Figure 25: Image Net error

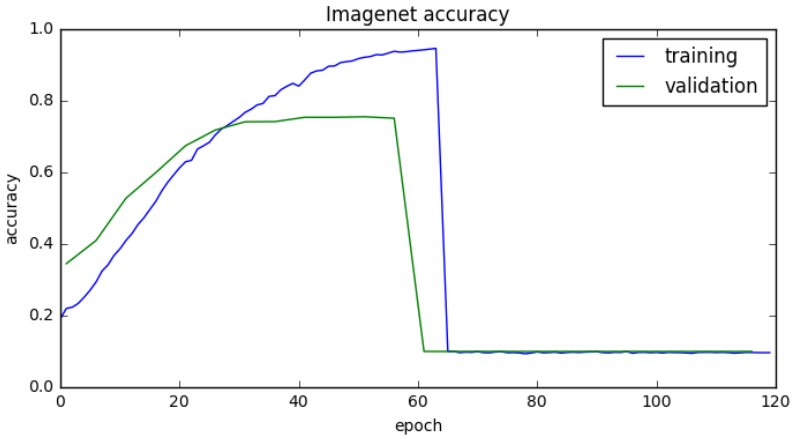


Figure 26: Enhanced imagenet accuracy

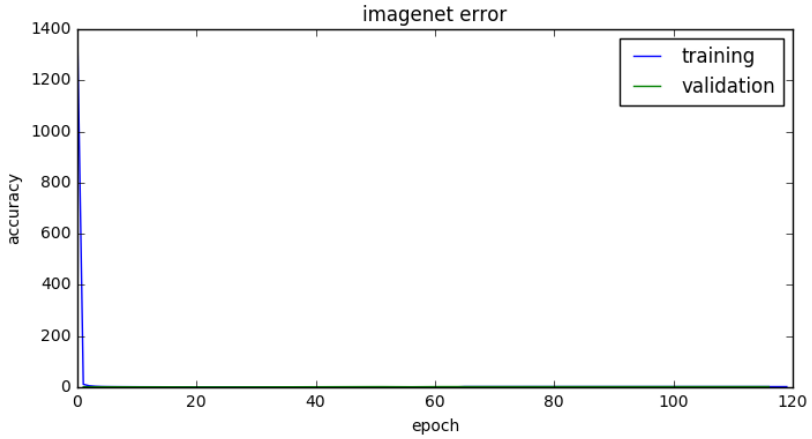


Figure 27: Enhanced imagenet error