Coursework 4 : Convolutional Neural Network in CIFAR-10

Aldy Syahdeini – s1575408

March 21, 2017

---

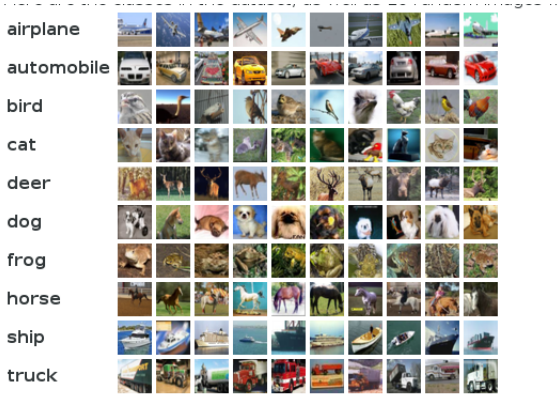
1 INTRODUCTION

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

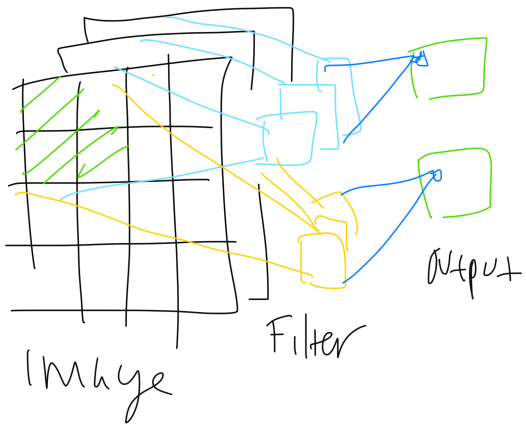


Figure 2: Convolutional layer

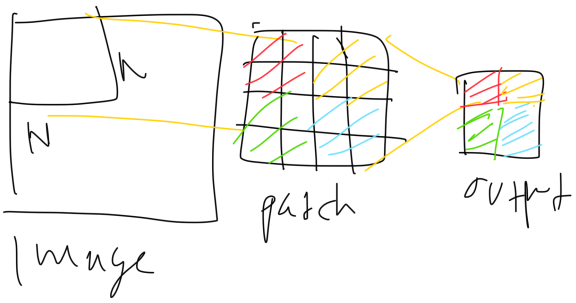


Figure 3: Convolutional layer

- 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

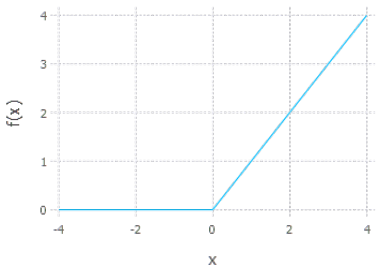


Figure 4: Relu function

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

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]

As it seen in Figure 4 and figure 4. Relu will result the dot product between the weight (w) and input (x) if the output is more than 0, otherwise the function will clapped it to 0. 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.

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.

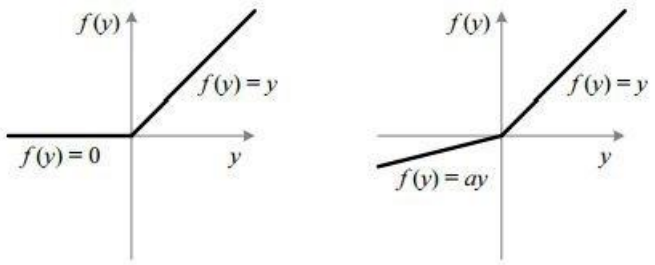


Figure 6: Different between relu (left) and leaky relu (right)

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

Figure 7: Leaky Relu formula

- 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 `http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html#43494641522d3130` 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.

2.3 Problem and research question

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 14, there is local optima and global optima, and we wish our hyper-parameter make as start at a good point 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 8,9,10 and 11 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.

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.

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 same , 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.

Initial learning rate is 0.001 and Adam as an optimizer. We initialize our weight using gaussian distribution with mean 0 and standart deviation 1.0.

3.2.2 Experiment objective

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

3.2.3 Result

Result show that the error is very high but it's decreasing significanlty, so I think that the weight initialization is not right. The result shows interesting result, where model with only a layer and 4 layers shows a very bad performance compared to the others. 2-layer seems to overfit the data. If we investigate figure 15 and 16. We will see there is a local minima at epoch 10^{th}. 3-layer and layer-t move near local minima but able to avoid it. But 5-Layer seems to stuck at the local minimath.

3.3 Using ImageNet Architecture to improve our model

3.4 ImageNet

ImageNet architecture is proposed by hinton et. all. (2012) [4], 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, Dropout and Data augmentation we will overcome the overfitting and they will learn a feature from the image. In the paper they introduce Local Response Normalization (LRN) which we already explain in the introduction.

3.5 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

We use tf.nn.lrn for the LRN function and tf.nn.dropout for the dropout function.

3.6 Result

The result in Figure 8 and 9 shows that this is so far the best model we are having with the accuracy 76% and 1.6% error for validation set. From figures 8 we can see that after epoch 38

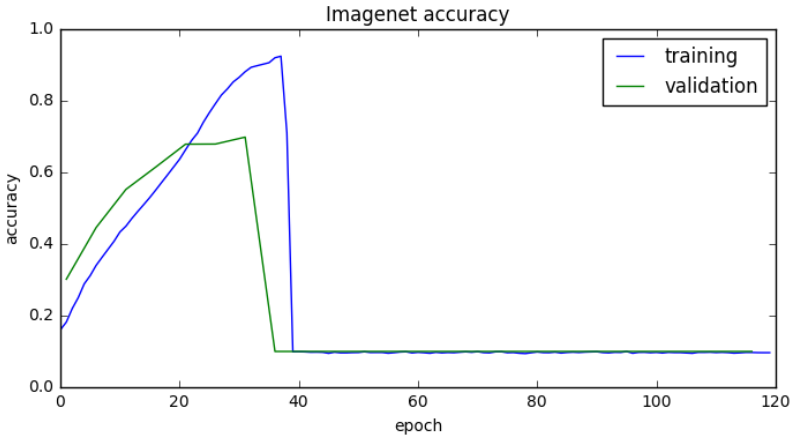


Figure 8: Image Net accuracy

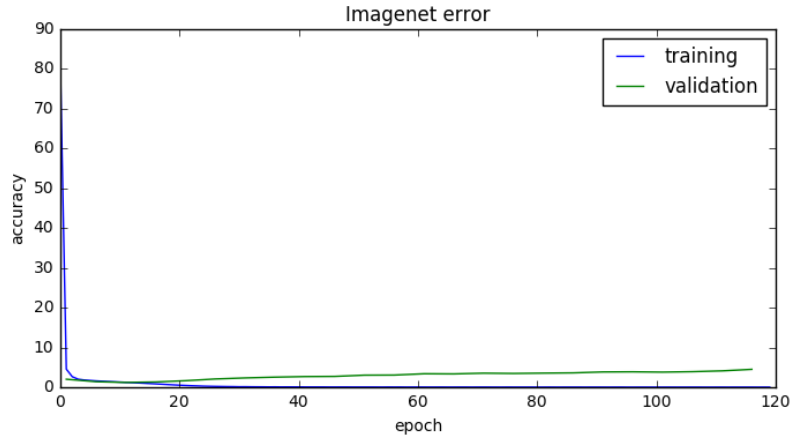


Figure 9: Image Net accuracy

the performance of our model is drop significantly, at this point the best soluion is to use early stopping. We stop training our model after we found the best performance and use the model with 38 epoch.

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

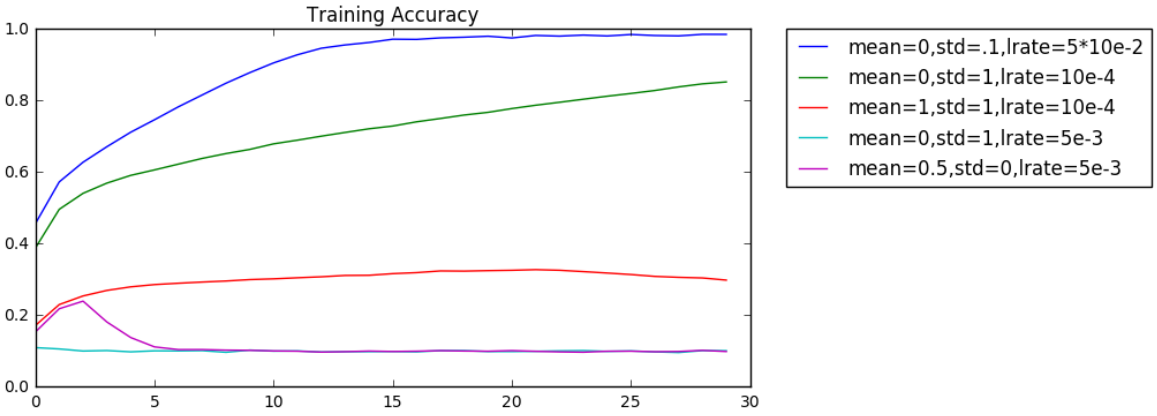


Figure 10: Hyper-parameter training Accuracy

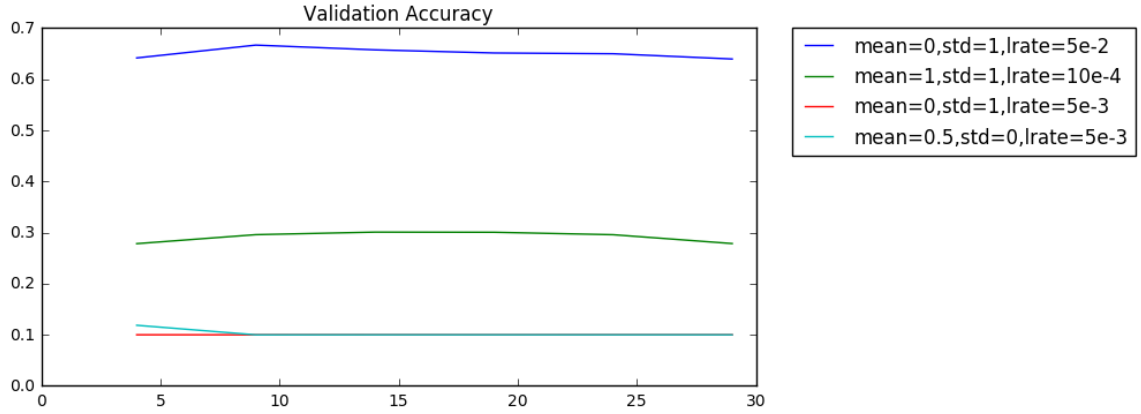


Figure 11: Hyper-parameter training Error

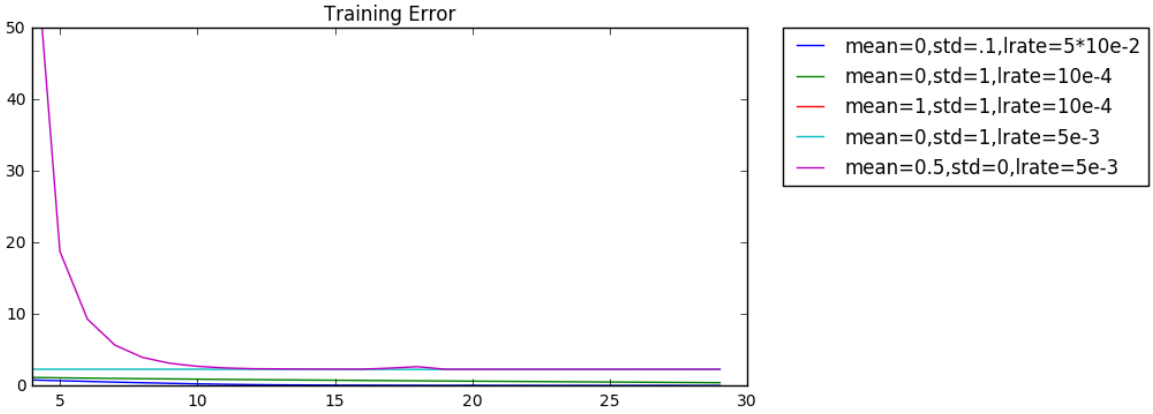


Figure 12: Hyper-parameter training Error

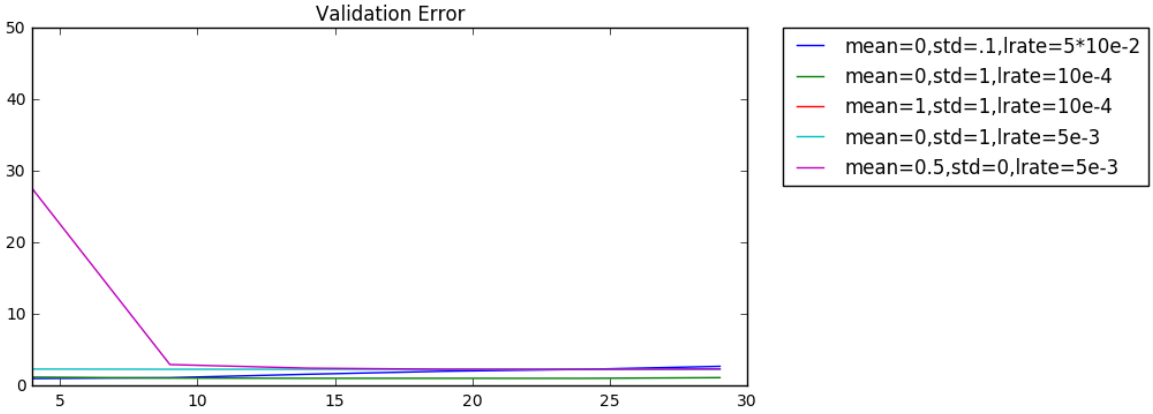


Figure 13: Hyper-parameter validation Error

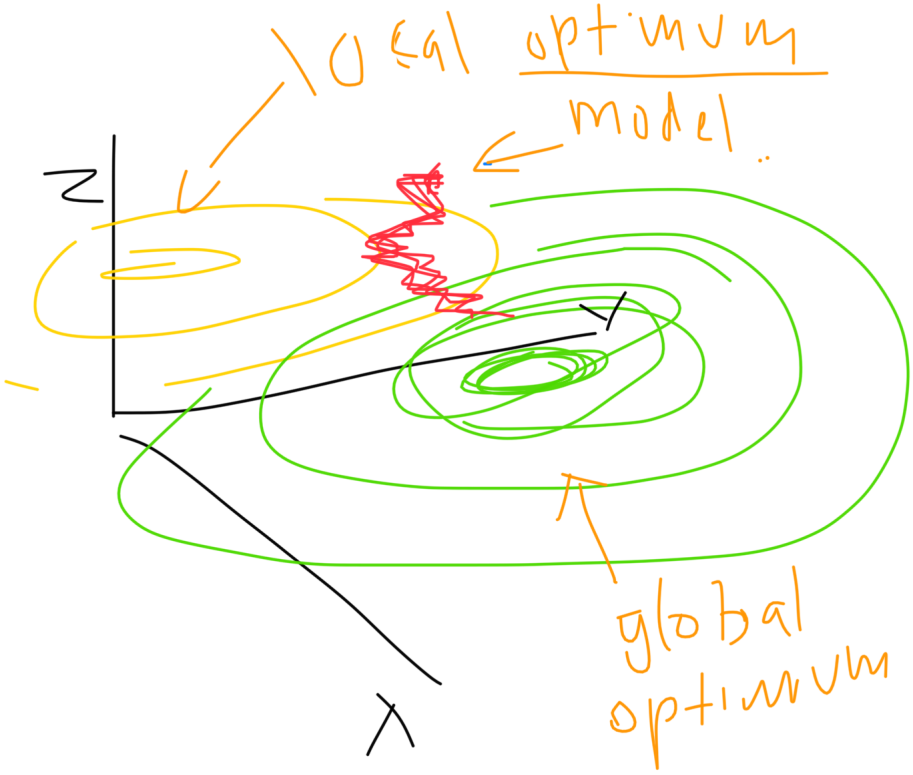


Figure 14: Solution space

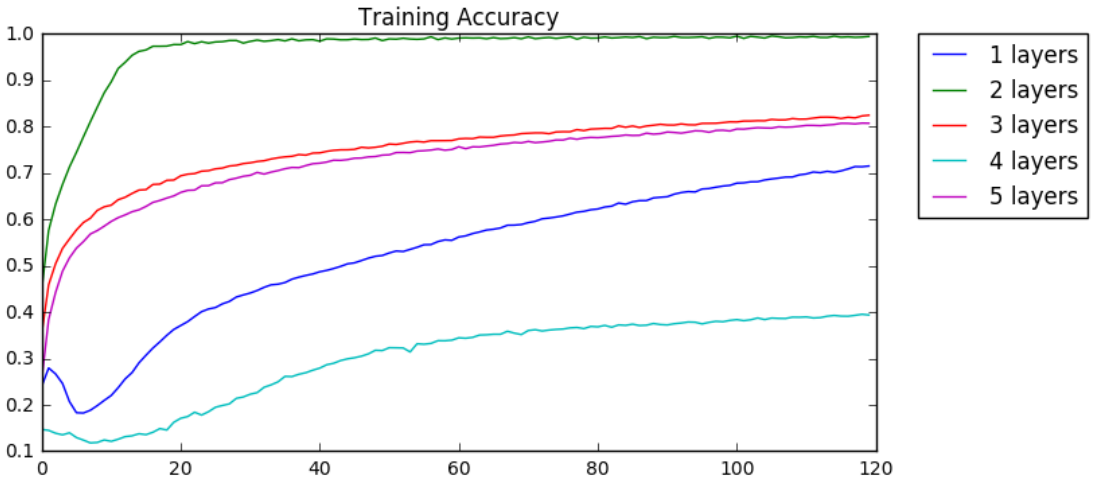


Figure 15: Hyper-parameter training Accuracy

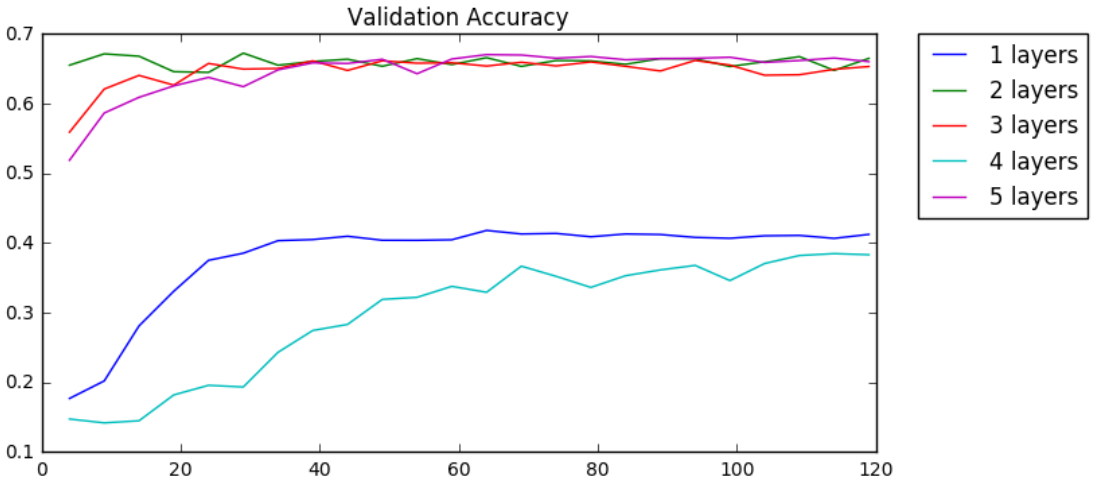


Figure 16: Hyper-parameter training Accuracy