Factors Analysis of Convolutional Neural Network on CIFAR-10

Zipei Wei and Samantha Kerkhoff

Abstract

Neural networks are currently one of the most popular topics in machine learning due to their ability to model datasets with high-dimensionality. They can achieve a high degree of accuracy on many types of problems from image classification to language recognition. In this project, we used a convolutional neural network to categorize images from the CIFAR-10 dataset. We explore variations of Convolutional neural network model, such as changing the batch size and adding data augmentation during data preprocessing, then compare the results and features of these changes. In some cases, our results showed significant changes in accuracy. In others, there may be little change in accuracy, but instead, change how long the algorithm takes to run. We discuss these results and which changes made the biggest difference using our dataset and algorithm.

Introduction

Convolutional Neural Network are one of the most popular neural nets for training in these days, besides using the Convolutional Neural Network for image classification, CNN had a large implication on video analysis, natural language processing and drug discovery (Wikipedia, 2018). CNN is on average outperformance compared to other classification methods such as k-mean, cluster analysis. With a relatively large dataset, CNN can overall performances well in both supervised and unsupervised learning.

Like many other neural networks, Convolutional neural network was inspired by biology processes, specifically the receptive field (Wikipedia, 2018). The model of Convolutional neural network is relatively new compared to the long history of computer architecture. In 1994, it was Yann Lecun first introduced the idea of Convolutional neural network [Figure 1] and the model was named LeNet5 (Yann Lecun, 1998). The LeNet5 model contains three main architecture of Convolutional neural network: convolutional layer, pooling layer and non-linearity layer or hidden layer. Each type of layers function differently, the convolutional layer applied an activation function with weight and bias for spatial extract, then the pooling layer for identifying the most significant features and subsampling. The hidden layer is targeted for handling non-linearity input. Besides that, Convolutional neural network also applies a loss function to find the optimized gradient descent.

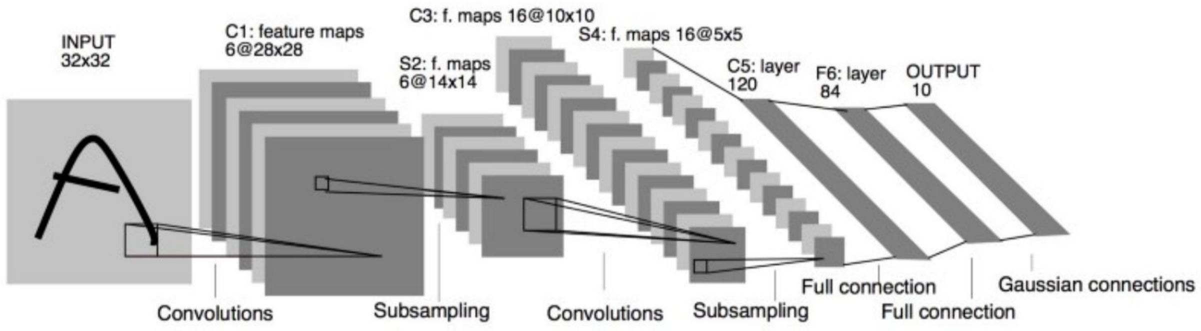


Figure 1. LeNet5

The structure of the convolutional neural network seems clear and easy to foresee. However, how each factor in the Convolutional neural network played a role to affect the output performance seems to be debated frequently. For instance, what is the optimized number of layers of Convolutional neural network? And how many layers will affect the computation efficiency? Besides that, is there correlation exists between certain factors or a joint influence affecting the performance of the Convolutional neural network? Thus, for these reasons, we are particularly interested in how different factors affect model performance. If there is, what's the tradeoff of this particular factor? Rather than obtain a higher accuracy of a particular dataset, this paper hopes to provide guidelines regarding the construction of the convolutional neural network.

To derive a factor analysis of convolutional neural network, we will start with the paired comparison analysis for the four factors that we found the most interested, which is: 1) Number of step size for gradient descent 2) Data augmentation for input 3) Number of input (batch size) 4) Different calculation for Learning Rate. By using Tensorflow provided by Google team, we first construct a LeNet5 5 layers convolutional network without data augmentation, as well as 128 batch size, 10k step size, and 0.05 standard learning rate). Derived from the original model, we will change the factors separately for each run to perform the paired comparison. Later on, we even train the Tensorflow model using CIFAR-10 dataset, which is a multi-layers Convolutional neural network with a step size of 100k. The results of the factors analysis will be discussed in the results part.

The CIFAR-10 data is one of the most used datasets for image classification, which consists of 60,000 32x32 color images. These images are divided up into ten classes, such as airplane, cat, or ship, and with 50,000 for training and 10,000 for testing. The data augmentation is one of the techniques that used a lot for improving the performance of Convolutional neural network, which is the image preprocessing techniques such as flip the images, change the brightness of the images. In this article, by applying the paired comparison, each factor will be examined and individually and will be verified if it indeed improves the performance of convolutional neural network no matter what type of the dataset you are selected or how the differences of each variation of Convolutional neural network.

Background

As CIFAR-10 is one of the most popular datasets for training and testing for deep learning, there are several published paper regarding the training of Cifar-10. The original model that was trained on Tensorflow website has referenced the structure from the published article “Convolutional deep belief networks on Cifar-10” by Alex Krizhevsky (Alex Krizhevsky, 2010). In his paper, he obtained an accuracy of 74.5 % using 2 convolutional layers with weight decay for learning rate. Later on, his paper was referenced by Tensorflow team and they applied multilayer Convolutional network and Tensorflow team obtained an accuracy of 86%. Thus one of the interesting point that we were interesting is how the weight decay would affect the results of the Convolutional neural network. Besides that, Tensorflow also applied the data augmentation before the training.

Besides the Google team for training the CIFAR-10 dataset, Microsoft team “Deep Residual Learning for Image Recognition” also trained the dataset with a residual learning framework called ImageNet to propose a visualization of the neural net during the training stage (Kaiming He, 2015). Both the Google team and Microsoft team constructed the Convolutional neural net using the Relu as the activation function and the Softmax function in the final layer for classification. Derived from the above two examples, we decided to apply the Relu as the activation function and the softmax for the final layer classification.

Approach

In order to train the dataset using Convolutional neural network, we need to use the Tensorflow framework. In the official website of Tensorflow, there are specific pages for how each variable works in Tensorflow. The Cifar-10 also being trained by Tensorflow team with a multi-layer Convolutional Neural Network with an accuracy rate at 86%. Besides that, Tensorflow is varied by GPU, CPU version as well as different operating systems. For instance, for Mac users, the only available Tensorflow is the CPU version, thus is not computation efficiency. Our team references the data structure from the Tensorflow website with a few modifications. It takes us 2 hours to train the model on a CPU version with a step size of 10k. Thus, later on, our team uploaded the model to the Amazon Web Service (AWS) to obtain computation efficiency. By training with a p2.xlarge AWS System with an Nvidia k80 GPU, the model can finish running with a 5 layer Convolutional Neural Network in 10 minutes. it also only takes less than 2 hours to train a Convolutional neural network with a 100k step size.

After setup the Tensorflow on our system, what we want to do next is downloaded the dataset from the Cifar-10 website. In the Cifar-10 website, there are two types of data types for the Cifar-10 dataset. The first one is the numpy version where data are stored as a 100000*3072 numpy array. The second one is the binary version where data are stored in 5 train data batch and 1 test data batch. In each batch, the images and class label is saved as binary conversion, where each input is 3073 bytes. The first byte is the class label, and the rest 3072 bytes is the image itself [Figure 2]. Thus the input will extract as a length of 3073 and assigned the first 3072 as an image, the last bytes as the class label. Later on, each image will be reshaped to suit for training.

The next step is we input each image, the batch size is the number of images we input into the neural network each time. By reference the basic structure from Tensorflow, which contains two convolutional layers, two max-pooling layers, two normalization layers, followed by two fully connected layers with rectified linear activation function and the final classification layer using softmax function (Tensorflow, 2015). The first layer of the convolutional neural network is the conv layer, which the activate function Relu will be applied. Compared to other activate functions, Relu seems to be our first choice due to the Relu function had the tendency to help the data to converge faster. Then after one conv layer, we normalized the layer for the pooling

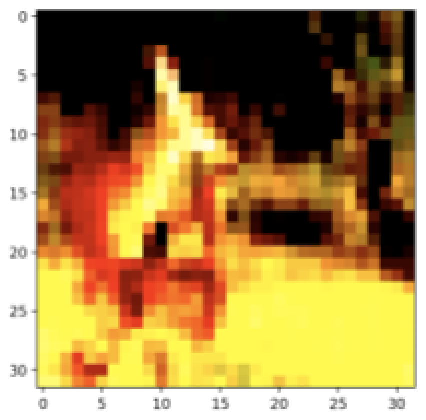


Image Height 32 * Image Width * RGB 3 = 3072 bytes

Figure 2. Reading images from binary files

layer. As discussed previously, the pooling layer mainly for feature recognition and subsampling. After that, we applied the conv layer again followed by the next max pooling layer. The final layer is the softmax for classification. The training model can be visualized by tensor board [Figure 3].

Besides that, there are a few things we changed from the Tensorflow model. The first factor is the calculation of weights. In Tensorflow model, they added a weight decay function for penalizes large weight. For our model, we simply apply the regular learning rate without weight decay. The next thing is, at the beginning of the project, the model is trained on a smaller step size 10k, where Tensorflow model is trained on an expensive GPU Nvidia k40 for 8 hours, thus the 100k step size is not computation friendly for individual’s GPU. The third modification we did is to not include the data augmentation in our base model since the data augmentation will be added for comparison purpose later on.

There are four main parts of our algorithm that we wanted to adjust to seeing what made the biggest difference in our accuracy. These are the number of steps, adding data augmentation, the batch size, and the learning rate. For the number of steps and the batch size, changing these values simply requires changing a variable in our code. In our testing, we ran examples from 5k to 100k steps to find at what point the model is no longer improved. For Batch size, we try from 32 to 512 samples per batch and examine the effect this can

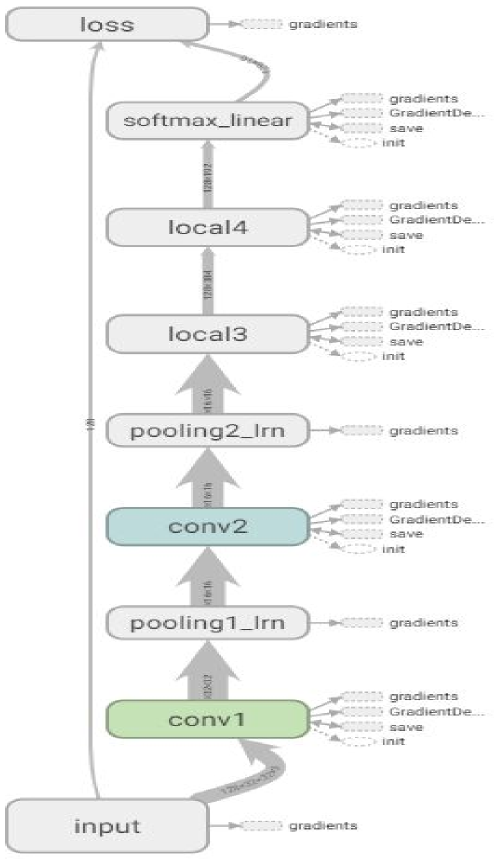


Figure 3. Training model

cause. Adding data augmentation required more thought in how it would change our code and existing algorithm and from our testing, we discuss if it is worth adding. Changing the learning rate can be more complicated, depending on the approach. In our algorithm, we use the same learning rate through the entire training process, which means our results should be straightforward.

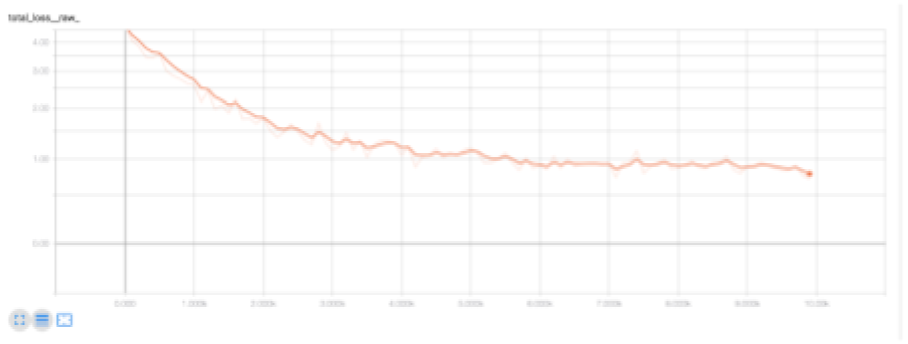
Below we will discuss in further detail what we were looking for with each of these sets of changes. We provide accuracy numbers to show the results of our testing and we examine what these numbers mean and what they imply about the changes to our models. We also consider changes or combinations of tests that may provide better results.

Results

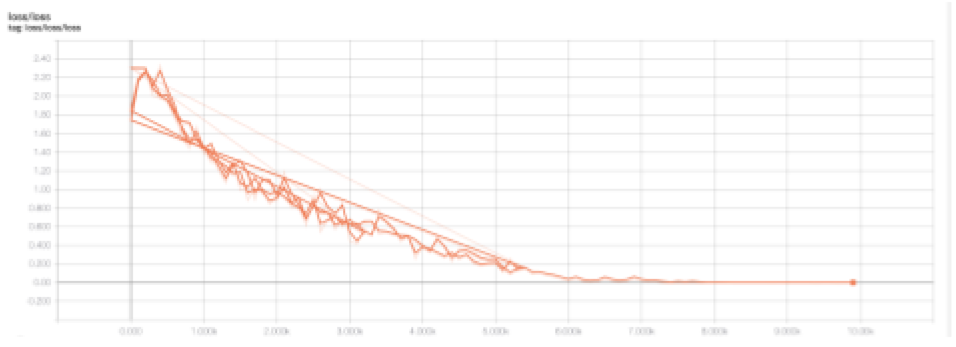
A. Number of steps

The first factor we compared is the number of steps, with the base model discussed above, We obtained a 78.1% accuracy with a 10k step, no data augmentation and no weight decay. The loss of the function almost reached 0 at a 10k step for the base model and the computation time is about 2 hours on a CPU version and about 20 minutes on AWS. With a batch size of 128 and a 10k step size, it is very obvious the neural network is not friendly with smaller computation power. Thus, after training the model on the CPU version, we decided to move the model to AWS for further exploration.

By setting up the model on AWS, we chose the p2.xlarge machine with an Nvidia



a). Tensorflow Loss at 10k Step Size



a). Base Model Loss at 10k Step Size

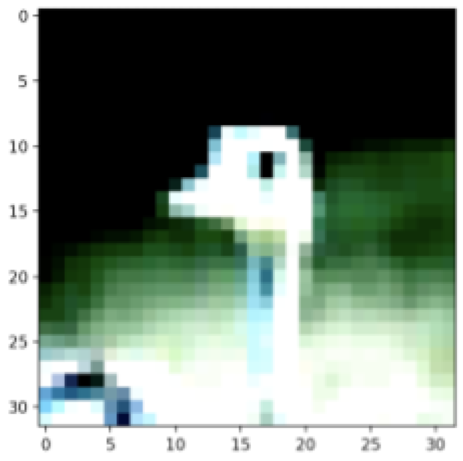
Figure 4. Loss at 10k

k80 to train the base model for the 100k step. It takes about 2 hours to train the base model in 100k with a loss of 0. In Figure 4 is the loss chart of the base model and Tensorflow model. Both of these models have the zig-zag shapes to show the loss gradually decrease as the step size increase. The difference is, at 10k, the base model is already reached 0. When we tried to increase the step size, the accuracy will stay the same. Thus, later on, other factors are not all trained in 100k step size.

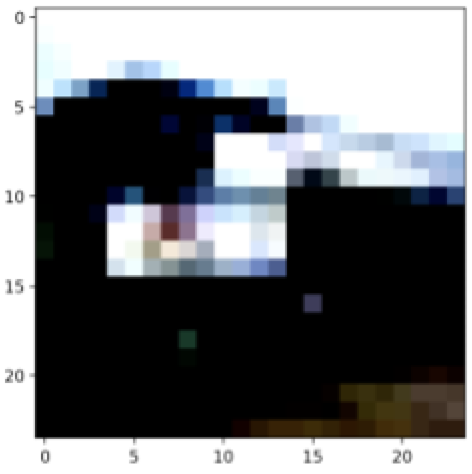
Thus, later on, other factors are not all trained in 100k step size. The results are quite different from the Tensorflow model. In the 10k step size, the Tensorflow model still has a loss of 1.0. Even after training for 100k, the Tensorflow model still has a loss of 0.51. Later on, Tensorflow team trained for 1000k stepsize to make the loss goes to 0, is not explored in this paper.

B. Data augmentation

The next factor we compared is the data augmentation effects, as many articles stated by altering the images can increase the model's accuracy, we implemented the data augmentation in our base model and trained with 10k step. The data techniques that we used are a) Randomly crop for a 24 by 24 area. b) Randomly flip each input image from left to right. c). Randomly increase the brightness of the images d) Randomly changes the contrast of the image. Figure 5 is the comparison results for images with data augmentation and without data augmentation.



a) Without Data Augmentation



a) With Data Augmentation

Figure 5. Comparison of Data Augmentation

Also, all the images must be standardized before an input for training by the requirement structure of Tensorflow. By training with both versions in the 10k step, it is surprised the training model with data augmentation actually has lower accuracy. One explanation for these is it depends on dataset specifically. Using the Cifar-10 dataset, the image itself is smaller enough, thus by cropping additional area of its actually placed a negative effect on the output. Another finding is the Tensorflow model without data augmentation is not performing well compared to the Tensorflow model with data augmentation. With data augmentation, the Tensorflow model's accuracy improved around 1.5% on both 10k and 100k step size. One assumption can be made from this result, the weight decay that was implemented actually has combined positive results for the model output. By penalizing the larger weight with data augmentation, it actually performs better than using them separately.

C. Batch Size

When working with a large dataset, it may not be possible to pass the entire contents through the network all at once. In order to process it, the dataset must be broken up into batches that the machine that's processing the data can handle. However, one of the downsides to splitting the data up is that fewer inputs can lead to underfitting, which means the model may not be as good. One of the simplest features we decided to examine was the size of the batches that our network was processing.

We ran tests with batch sizes from 32 samples to 512 samples and found that there was a slight variation in the accuracy we produced. Our best accuracy was 0.778 and was achieved when we used a batch size of 128. While we found that using more than 128 samples did not improve our accuracy, the losses were not as significant as when we decreased the batch size.

One big difference the batch size made was in how long it took to process a single batch. As could be expected, the larger batches took longer to process and past a certain point the machine would be unable to handle a larger size. Smaller sizes could achieve results much more quickly. Because of the limited improvement past a certain size, it is not necessarily best to choose the largest batch size that a machine can handle. This is something that should be chosen based on the machine performing the processing and what is reasonable given the data input.

Below is the results of our batch size testing. These tests were all performed with a learning rate of 0.05 and 10,000 steps.

D. Learning Rate

As mentioned earlier in this report, our neural network model uses a gradient descent algorithm for the optimizer. In a gradient descent algorithm, the learning rate controls how quickly weight values can change from one step to the next. The advantage of a high learning rate is that the algorithm can improve quickly. However, if the learning rate is too high, then the changes may happen too quickly and the algorithm could "overshoot" the most optimal choice.

Using our basic starting point of a 10,000 step run with a batch size of 128, we tested a few learning rates that would remain the same for the duration of the run. We found that with a learning rate of around 0.05 to 0.10, there was little change in accuracy. However, as we lowered the learning rate, we found that the accuracy decreased along with it. Similarly, increasing the learning rate too far decreased our performance.

These results match what we expected. Because the number of steps was the same in each test run, it is likely the runs with lower learning rates were unable to converge by the time they completed. With more steps, it is possible we would see better results for these lower learning rates. This is another case where the values must be tuned to the data and is the reason why many algorithms will dynamically adjust the learning rate, to gain the benefit of both higher and lower selections.

Factors Analysis Comparison Chart

Factors	**Loss**		**Accuracy (%)**	
	Base Model	**Tensorflow Model**	**Base Model**	**Tensorflow Model**
Number of Steps 10k	0.0003	1.0000	78.1	81.5
Number of Steps 100k	0.0000	0.5100	78.4	84.5
Without Data Augmentation	0.0003	0.7700	78.1	80.6
With Data Augmentation	0.0005	1.0000	74.4	81.5
Standard weight	0.0003	\	78.1	\
Weight Decay	\	1.0000	\	81.5
Standard Batch Size (128)	0.0002	\	77.8	\
Decrease Batch Size (64)	0.1255	\	75.3	\
Increase Batch Size (256)	0.0004	\	77.1	\
Learning Rate (0.005)		\	56.0	\
Learning Rate (0.02)		\	75.7	\
Learning Rate (0.05)		\	77.8	\
Learning Rate (0.1)		\	77.2	\
Learning Rate (0.2)		\	75.5	\

Figure 6. Factor Analysis Comparison Chart

Conclusion

Through our testing, we were able to see what type of effect each of our changes had on the loss and accuracy of our model. When we consider the learning rate, we see that the optimal selection can have a huge effect on the accuracy of the testing set. Although we did not implement weight decay in our model, we can see from the Tensorflow model that it could provide higher accuracy than what we achieved. However, this accuracy would have caused the loss to converge more slowly, which would have increased our computation time significantly.

When we added data augmentation, we did not find the improvement in our results that we expected. This shows that although some feature changes can be helpful, they are not necessarily helpful in all cases. In some cases, changes to one attribute must be combined with changes to another in order to see improved results. For instance, if something slows down the time to converge, we must increase the number of steps to account for it.

In a few cases, we considered how changes affected the computational complexity of our model. When looking at the batch size, we found that training took a lot longer as we increased the batch size, however, the accuracy did not necessarily improve. This is important because even if there are no computational limits, we do not want to increase the complexity of the model if there is no improvement in the final results.

Overall, our results showed that training models should be examined on a case-by-case basis. Many factors cannot be changed alone and the whole model must be considered. Even when looking at similar models, changes may affect one differently from the other. We hope our findings can provide a guideline for how some features affect the results of a model and what to expect when making changes to seek improvement.

References

1. Wikipedia contributors. (2018, December 6). Convolutional neural network. In *Wikipedia, The Free Encyclopedia*. Retrieved 07:27, December 10, 2018, from https://en.wikipedia.org/w/index.php?title=Convolutional_neural_network&oldid=872264841
2. Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, Nov. 1998.
3. Krizhevsky, Alex. (2012). Convolutional Deep Belief Networks on CIFAR-10.
4. K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, 2016, pp. 770-778.
5. Abadi, Martin & Agarwal, Ashish & Barham, Paul & Brevdo, Eugene & Chen, Zhifeng & Citro, Craig & Corrado, G.s & Davis, Andy & Dean, Jeffrey & Devin, Matthieu & Ghemawat, Sanjay & Goodfellow, Ian & Harp, Andrew & Irving, Geoffrey & Isard, Michael & Jia, Yangqing & Kaiser, Lukasz & Kudlur, Manjunath & Levenberg, Josh & Zheng, Xiaoqiang. (2015). TensorFlow : Large-Scale Machine Learning on Heterogeneous Distributed Systems.