IMAGE CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORKS

Shawn Diaz

ABSTRACT

Given a baseline convolutional neural network, the goal of this experiment was to adjust the baseline network hyper-parameters to improve performance. How hyper-parameters are adjusted can affect the performance of the network to a notable degree. Therefore, it is important to use an approach that allows for us to see the effects of changing each parameter in specific ways. By adjusting hyper-parameters methodically and analyzing the results, a better understanding can be achieved about how each adjustment affects the performance of the network. After running the tests and adjusting hyper-parameters from the baseline network, the adjusted network showed notably improved performance. Given these results, conducting experiments like this one can be beneficial for further solidifying the understanding that exists about fine-tuning neural networks.

1. INTRODUCTION

Neural networks contain hyper-parameters which can be adjusted to fine-tune how well a network performs. These parameters are adjusted according to certain factors, such as the type of data the model is trained on. In this case, the CIFAR-10 data-set was used. This data-set consists of 60,000 images (50,000 for training and 10,000 for testing) with an image size of $32 \times 32 \times 3$, and it can be used to train a model for classifying ten items: airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks. Hyper-parameters are also adjusted based on how well the model performs during training. Various metrics can be used to analyze the performance of a model, and subsequently fine-tune it. With this experiment, the goal is to improve the given baseline network (outlined in section 3) in ways that decrease loss and improve accuracy when trained with the CIFAR-10 data-set. In turn, the well-known methods used to achieve this goal can be further tested and validated, both for learning purposes, as well as for review.

2. RELATED WORKS

There is a lot of research surrounding the CIFAR-10 data-set and the development of highly optimized neural networks that perform well on the data-set. One example of such works was published in 2012 by Krizhevsky et al. [1]. They were able to achieve an accuracy of 89% on the CIFAR-10 data-set using dropout in the first two fully connected layers of their model. This regularization technique combats overfitting by randomly ignoring a number of neurons during each training pass. With each pass, there will be an amount of randomly selected neurons which will not contribute to the activation or update of weights. This results in a model that is capable of better generalization.

Springenberg et al. [2] were able to achieve an accuracy of 95.59% on the CIFAR-10 data-set. They managed to do this using a network architecture that replaces the max pooling layers of the convolutional network with convolutional layers that have a larger stride. In order for them to analyze the network, they introduced a version of the deconvolution approach that allowed them to analyze how the network learned in the absence of pooling layers. It was found that this approach yielded equal, if not better, performance when compared to other state-of-the-art architectures.

Another interesting work, from DeVries et al. [3], investigates a simple regularization technique called "cutout". Cutout is an easy to implement regularization technique that can be used to improve the robustness and overall performance of a convolutional neural network. It works by occluding sections of input images as a form of data augmentation. The end result is a model that is forced to look at the data in a more generalized sense, thus improving the performance of the model.

A fourth work, by Cubuk et al. [4] attained an accuracy of 98.52% on the CIFAR-10 data-set using a new data augmentation technique named AutoAugment. This technique uses a search algorithm to automatically augment data in order to maximize the validation accuracy on a data-set. The algorithm works by defining data augmentation policies and uses the most effective ones as the model is trained. It was found that this technique can be applied to many different network architectures to improve accuracy.

Looking at regularization techniques, another technique was proposed by Yamada et al. [5]. Their technique, named ShakeDrop, was inspired by another similar regularization technique named Shake-Shake, proposed by Gastaldi et al. [6]. Shake-Shake worked by using a random variable to disturb the forward pass, and similarly, with a different variable, to disturb the backwards pass of their neural network. With ShakeDrop, they were able to overcome the challenge of applying the Shake-Shake regularization to ResNet, as well as achieve stability during training by using RandomDrop. Ran-

domDrop drops layers, similar to how dropout drops neurons in a neural network pass. The results of the ShakeDrop regularization technique were an improvement in performance, attaining an accuracy of 97.69% on the CIFAR-10 data-set.

3. METHOD

The baseline network is a convolutional neural network with 7 layers. This is illustrated in Figure 1. The first four layers are convolutional layers, and the last three are fully connected layers. Each of the four convolutional layers use a filter size of 3×3, with 32, 64, 128, and 256 filters, respectively. The first two convolutional layers are followed by max pooling layers with a pool size of 2×2. The two fully connected layers have 2048 units each, before the output layer. The output layer is a softmax regression with 10 classes, reflecting the 10 classes of the CIFAR-10 data-set. The network is trained over 25 epochs with a batch size of 128 using the Adam optimizer with a learning rate of 0.001. Finally, categorical cross-entropy is used for the loss function. This baseline implementation yields an accuracy of 68.53%.

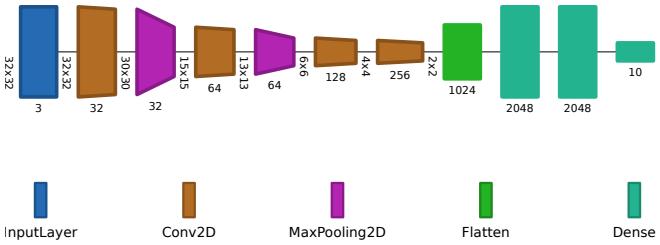

Fig. 1. An illustration using Net2Vis[7] of the architecture of the baseline neural network.

4. EXPERIMENTS AND RESULTS

For the first experiment, the only hyper-parameter that was adjusted initially was the number of epochs. The one that resulted in the best accuracy with the baseline network is what was used for the rest of the experiments. First, the baseline network was trained over one epoch, and then five epochs. Both of these experiments yielded a model that did not converge, or reach a point where it could not learn any further. Therefore, the same baseline network was trained using 10 epochs. This showed better results, where the model stopped learning at around 69% accuracy; This was a little better than the baseline network, but with less epochs. Finally, the network was trained over 15 epochs. This resulted in the highest accuracy at 70.11%, informing the epoch amount to use for the rest of the experiments. These results are summarized in Table 1.

Next, various optimizers were tested. First, the Adam optimizer was tested with different learning rates: 0.1, 0.01, and 0.001. It turned out that the higher learning rates barely wanted to learn at all, and the lowest rate did the best. Next, the stochastic gradient descent (SGD) optimizer was tested. SGD seemed to perform best with the highest learning rate of 0.1 during training. However, at only 62% accuracy, it was not an improvement over the Adam optimizer. It is also worth noting that using SGD optimizer with a lower learning rate might have had better results with higher epoch size. Finally, root mean squared propagation, or RMSprop, was tested. This optimizer, similar to the Adam optimizer, only showed learning with the lowest learning rate; It achieved an accuracy of 67.5% after 15 epochs.

Number of Epochs	Loss	Accuracy (%)
1	1.3812	48.76
5	0.9621	67.73
10	1.0936	69.34
15	**1.5826**	**70.11**
25	1.9888	68.53

Table 1.
Comparison between the baseline network loss and accuracy given various epoch amounts. These comparisons informed how many epochs to use for the subsequent experiments.

Looking at Figure 2, the baseline network showed the highest accuracy with 15 epochs. However, the network had a problem of overfitting. Overfitting occurs when the model learns the training data-set too well. This means that when it is given data not in the training set (i.e. the test-set), it under performs. This issue could also explain why the network performed slightly worse with 25 epochs.

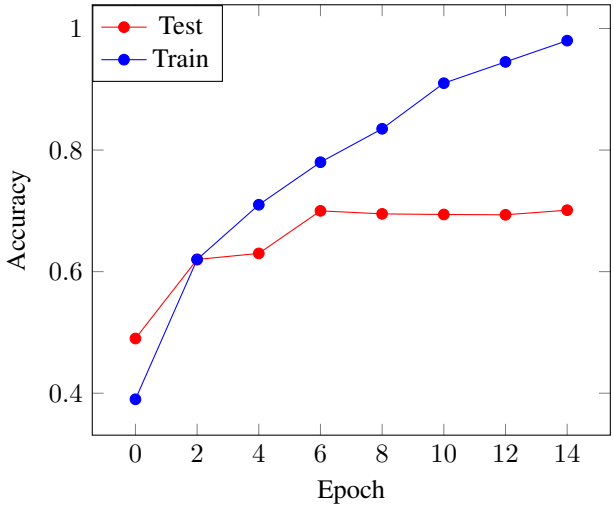

Fig. 2. The learning curve for the baseline network over 15 epochs. Here, overfitting can be observed where the test set stops learning around 69% accuracy, while the training set continues to learn on the data.

To combat the overfitting issues of the baseline network, regularization techniques were introduced. First, dropout with a rate of 0.2 was added to the two fully connected layers. This adjustment increased the network accuracy to 71.% on the test set. Then, L1 regularization was tested instead of dropout. With only L1 regularization, the network did not do very well. This could be due to the technique not being suited for the given problem. However, L2 regularization worked well and improved the performance of the network. With dropout and L2 regularization together, the network saw the best performance.

The fine-tuned network that achieved the best accuracy with 15 epochs and a batch size of 128 used two forms of regularization: dropout with a rate of 0.2 and L2 regularization with a factor of 0.01. The learning curve for this network can be seen in Figure 3. Looking at this figure, the fine-tuned network appears to fit the data well. It achieves an accuracy of 74% the training set, and an accuracy of 72% on the testing set.

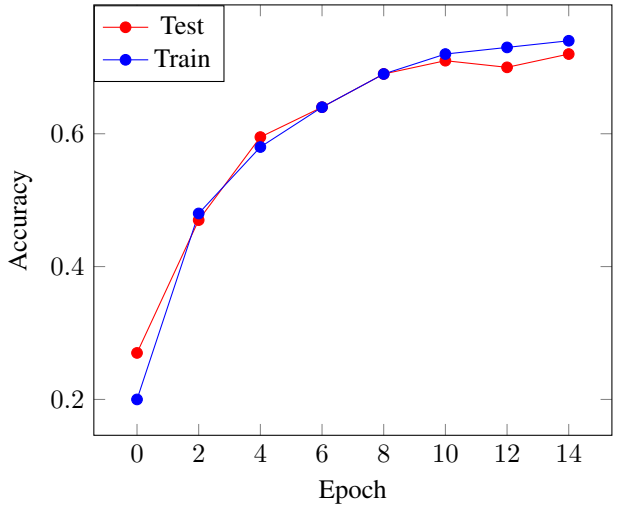

Fig. 3. The learning curve for the fine-tuned network over 15 epochs. In this case, there is very little overfitting, and the model seems to be well fit.

Even though these results are an improvement over the baseline network, the model did not seem to reach convergence. Therefore, it made sense to try training the the same fine-tuned network, but using 25 epochs; This would possibly allow the model to learn further if it could. With these adjustments, an accuracy of 77% was achieved on the training set, with an accuracy 73.5% on the test set. Here, the accuracy on the test set improved, but not without some overfitting. Figure 4 shows the learning curve of the fine-tuned network with 25 epochs.

Ignoring some of the overfitting issues, the network was trained using a larger batch size of 256 instead of the baseline batch size of 128. This gave an accuracy of 72.5% on the test set. Here, the accuracy on the test set did not improve,

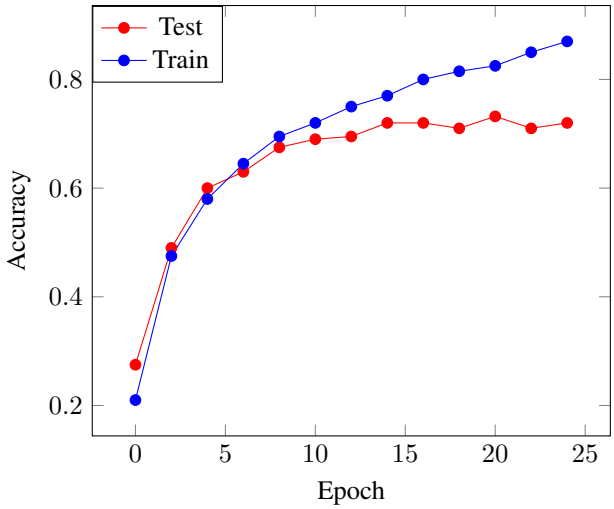

Fig. 4. The learning curve for the fine-tuned network over 25 epochs. The test set for this model shows more overfitting when compared to the 15 epoch version. However, it had a higher overall test accuracy at 73.5%.

and overfitting was still present. With a batch size of 512, the accuracy decreased to 67.5%. Therefore, using L2 regularization, dropout, and a batch size of 128, gave the highest accuracy on the test set of all the experiments that were conducted. These hyper-parameters were the final fine-tuned network that was concluded with.

Although it is outside of the scope of this experiment, adjusting the dropout rate or the regularization factor further would probably help with some of the overfitting issues that were still present in the final fine-tuned network. Table 2 shows the final fine-tuned network compared to state-of-the-art works using the CIFAR-10 data-set. These works utilize many more advanced techniques to achieve accuracy well into the 80% and 90% ranges. With that said, an accuracy of 73.5% on the final-fine tuned network is an improvement over the baseline network, even if not competitive with more advanced works.

Method	**Accuracy (%)**
Baseline network	68.53
Fine-tuned network	**73.50**
Network In Network[8]	91.20
Hyper Residual Network 40-2[9]	92.77
EfficientNet[10]	98.90
GPipe[11]	99.00
Big Transfer (BiT-L)[12]	99.37

Table 2. Comparisons between the accuracy of the final fine-tuned network to that of state-of-the-art works using the CIFAR-10 data-set.

5. CONCLUSION

The results of this experiment show that fine-tuning a network for image classification is not a trivial task. When looking at the performance of the final fine-tuned network on the CIFAR-10 dataset in comparison to state-of-the-art works, there is a lot that can be done outside of the scope of this experiment to further optimize the network and improve performance. However, when compared to the baseline network, there was a noticeable increase in accuracy, and the model was better fit. Through the process of making small changes to the hyper-parameters and analyzing the results, it starts to become more clear what is useful and not for training a model to do image classification.

6. REFERENCES

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, may 2017.

[2] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller, "Striving for simplicity: The all convolutional net," 2015.

[3] Terrance DeVries and Graham W. Taylor, "Improved regularization of convolutional neural networks with cutout," 2017.

[4] Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le, "Autoaugment: Learning augmentation policies from data," 2019.

[5] Yoshihiro Yamada, Masakazu Iwamura, Takuya Akiba, and Koichi Kise, "Shakedrop regularization for deep residual learning," *IEEE Access*, vol. 7, pp. 186126–186136, 2019.

[6] Xavier Gastaldi, "Shake-shake regularization," 2017.

[7] Alex Bäuerle, Christian van Onzenoodt, and Timo Ropinski, "Net2vis – a visual grammar for automatically generating publication-tailored cnn architecture visualizations," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 6, pp. 2980–2991, 2021.

[8] Min Lin, Qiang Chen, and Shuicheng Yan, "Network in network," 2014.

[9] David Ha, Andrew Dai, and Quoc V. Le, "Hypernetworks," 2016.

[10] Mingxing Tan and Quoc V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2020.

[11] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," 2019.

[12] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby, "Big transfer (bit): General visual representation learning," 2020.