Simple convolutional neural networks for image classification

Christian da Costa Eetu Vilkki

January 25, 2018

Abstract

In this study, we create a simple and fast convolutional deep neural network for image classification and see how it compares to more complex models. The used dataset is CIFAR-10, which is a well known dataset consisting of 60000 images, each belonging to one of 10 different object classes. We also consider how different hyperparameters and regularization methods affect the results. Our model achieves approximately 65% accuracy with only a 2-3 minute training time. Compared to other results, the accuracy is worse, but our model is considerably simpler and faster. The results also show room for more improvement in accuracy with further optimization and regularization.

1 Introduction

In this project we research how well relatively simple deep learning architectures can be applied to object recognition from images and how well they perform. More specifically, the goal is to create a simpler model than our reference model, ImageNet Classification with Deep Convolutional Neural Networks [Krizhevsky et al., 2012], and to achieve comparable performance. To solve this problem we also use a deep neural network, consisting of two convolutional layers with one pooling layer each and two fully connected layers. The used data is known as the CIFAR-10 dataset. It contains images of ten different object classes. The trained convolutional network takes one image as input and gives an array of floats as output that specifies the probabilities of the image belonging in each class, i.e. the probabilities that a given object is in the image. In other words, the task is to determine to which of the 10 object classes the object in the image belongs.

2 Related work

Object recognition is a fairly new area of research, with most of the models for CIFAR-10 having been developed between the years 2011 and 2016.

Therefore, it comes as no surprise that many of the prior works feature similar methods. The most common approach is deep neural networks that utilize convolutional layers and data augmentation. However, other approaches, like recurrent neural networks and Bayesian hyperparameter optimization, are used as well. [Benenson, 2016]

Our reference model, ImageNet Classification with Deep Convolutional Neural Networks [Krizhevsky et al., 2012], is fairly traditional by current standards. Its' architecture consists of five convolutional and three fully connected layers. These layers amount to 60 million parameters in total, which introduces a high risk of overfitting. To combat overfitting, the model employs two important, albeit common, methods: data augmentation and dropout. Additionally, the network uses ReLUs as its' hidden units and local response normalization after the first and second convolutional layers.

A significant distinction to other models, however, is that the model was initially developed for a subset of the ImageNet dataset, called ILSVRC-2010 [Russakovsky et al., 2015], which explains the complexity of the model. The dataset consists of roughly 1.2 million training images, 50,000 validation images, and 150,000 testing images.

Although the reference model achieves a respectable test accuracy of 89%, it still doesn't manage to compete with the top models. One of the best, if not the best, models, achieves a test accuracy of 96.53% by utilizing fractional max pooling with a convolutional network [Graham, 2014]. Fractional max pooling significantly reduces overfitting, therefore also reducing the need of using dropout.

3 Method

Our approach is to create a relatively simple convolutional neural network that is fast to train and run. Convolutional networks have historically shown great performance for computer vision and object recognition tasks, which is why this approach was chosen. The model consists of two convolutional layers with 2x2 max pooling each, followed by two fully connected layers. Finally, we have a softmax layer, which maps the output of the second fully connected layer to probabilities for the 10 classes.

We use ReLUs as our hidden units, as deep convolutional neural networks with ReLUs train significantly faster than with tanh units, for example. ReLUs have also shown to generalize well to many types of models and datasets and do not require input normalization to prevent them from saturating. ReLU is defined as

$$f(x) = max(0, x). \tag{1}$$

To improve generalization, we added Dropout after both of the fully connected layers and batch normalization after both of the convolutional layers.

Batch normalization can be thought of as preprocessing between the layers. It greatly increases the training stability, but also allows us to use higher learning rates to increase the speed of training, as the outputs of the layers get normalized. Dropout, on the other hand, is a regularization algorithm that randomly removes nonoutput units from the base network during training. It is an inexpensive approximation to training and evaluating a bagged ensemble of exponentially many neural networks [Goodfellow et al., 2016].

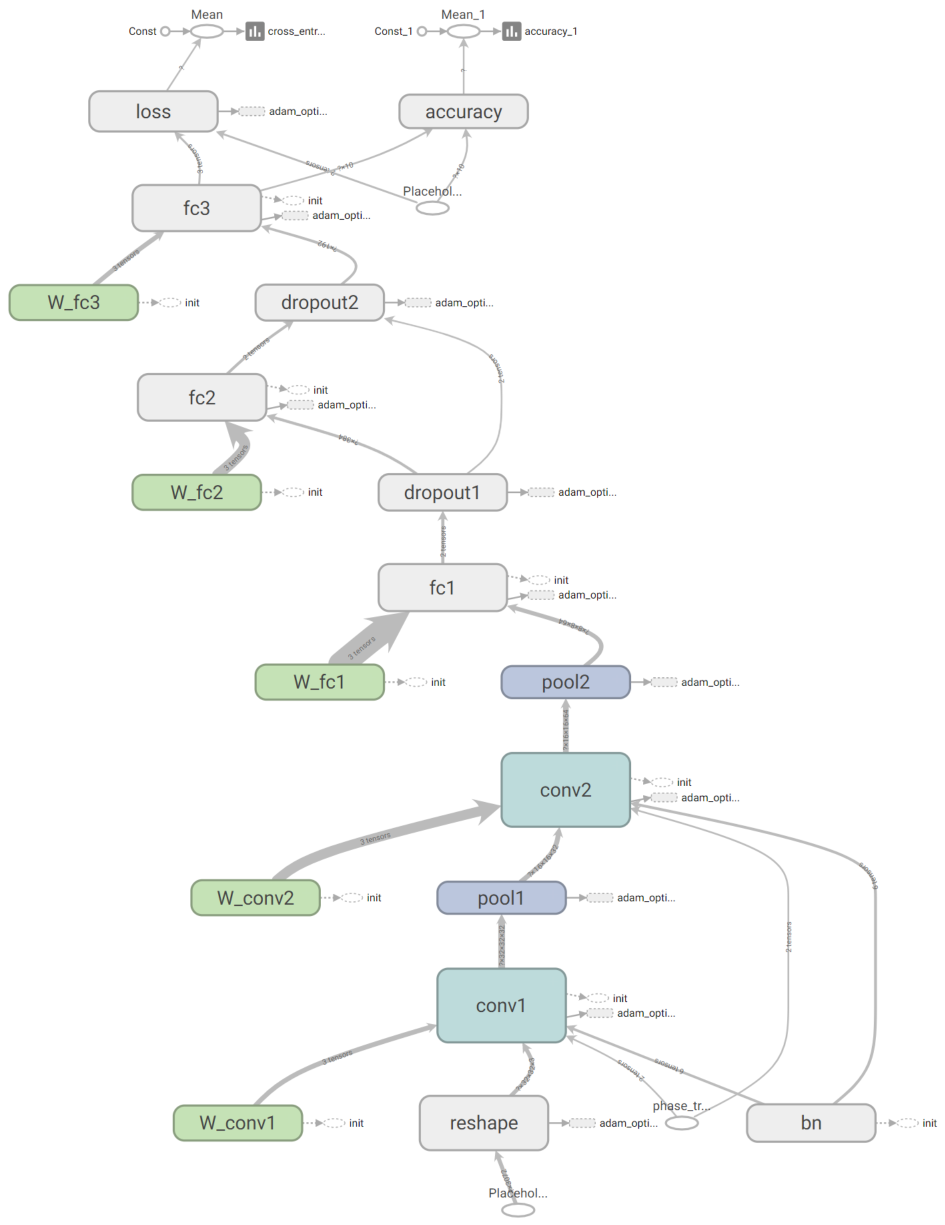

Figure 1: A Tensorboard graph showing the architecture of our deep neural network.

3.1 Cost function and optimization algorithm

Cross-entropy has become a standard cost function in classification tasks. It largely avoids slowing down the learning and manages to quite accurately estimate the difference between the true distribution and the model distribution of the outputs. This is why cross-entropy was chosen as the cost function. Cross-entropy is defined as

$$H(p, q_\theta) = E_p[-log(q_\theta)] = H(p) + D_{KL}(p||q_\theta) \tag{2}$$

where p is the true distribution and q is the model distribution parameterized with θ. $H(p)$ is the entropy and $D(p||q)$ is the KL-divergence.

Adam [Kingma and Ba, 2014] was chosen as the optimization algorithm. As with cross-entropy, Adam has recently gained a lot of favor, because it manages to combine the advantages of RMSProp and momentum algorithms in order to adapt the learning rate and also allows to use mini-batch learning. Mini-batch learning has a regularizing effect and is more efficient from a computational point of view, which increase the benefits of Adam.

3.2 Hyperparameters

Most important hyperparamaters involved in our model are learning rate, number of hidden units, convolution kernel width, dropout rate and batch size. You can find a table below that describes the values of our hyperparameters.

Learning rate is perhaps the most important hyperparameter in most models that use gradient descent based optimization algorithms, like Adam. This parameter tunes how fast or slow the weights are moved to the optimal values. If learning rate is too large, training might skip the optimal solution, and if it is too small, converging might require too many iterations or might not happen at all.

Number of hidden units is the amount of neurons in all of the hidden layers. Increasing the amount will increase the representational capacity of the model, but also increases both the time and memory cost of all the operations in the model. Additionally, too many hidden units might introduce overfitting, due to the increased complexity of the model. We use 32 and 64 hidden units for the two convolutional layers, and 384 and 192 hidden units for the two fully connected layers.

Convolution kernel width determines how narrow the ouput of the layer is. Increasing the kernel width increases the representational capacity of the model, assuming implicit zero padding is used.

Dropout rate decreases representational capacity when increased, but also increases the effects of regularization, due to dropping units more often. We use a dropout rate of 0.5, as this has proven to be the optimal rate in most cases [Srivastava et al., 2014].

Batch size is the amount of samples used for each batch when training the network with gradient descent. The optimal batch size depends heavily on the data, but has a nontrivial effect on both the training and test accuracy. Smaller batches makes the estimation of the gradient over the whole dataset harder and introduces noise, but often improves generalization for the same reason.

Learning rate	0.0005
Number of hidden units	672
Convolution kernel width	5x5
Dropout rate	0.5
Batch size	50

Table 1: Used values for hyperparameters.

4 Data

The dataset used in our research, CIFAR-10 [Krizhevsky et al., 2014] is a smaller subset of CIFAR-100 80 million image dataset. CIFAR-10 dataset (available from the official CIFAR-10 website) consists of 60000 32x32 RGB image samples from 10 different classes. The dataset is divided into 6 different batches: 5 training and 1 test batch. Each of these batches contain 10000 image samples. The training and test batches in total contain 5000 and 1000 samples of each class, respectively.
The dataset consists of the following classes:

1. airplane
2. automobile
3. bird
4. cat
5. deer
6. dog
7. frog
8. horse
9. ship
10. truck

The image data is represented with as a 3072 length integer array, with values ranging between 0 and 255. Before feeding the image data to the network, we do per image standardization to scale the image data to have a zero mean and unit norm. In addition to data preprocessing data augmentation is another option worth mentioning. As the standard CIFAR-10 dataset is quite small, data augmentation methods such as streching and scaling the images can be used to increase the size of the dataset and thus improve the performance [Krizhevsky et al., 2012].

5 Experiments

The goal for our experiments was to develop a simpler convolutional network that achieves comparable results to [Krizhevsky et al., 2012] by modifying both the architecture and hyperparameters. Initially, we started with an architecture of two convolutional layers with 2x2 max pooling each, followed by one fully connected layer. This is a standard architecture for convolutional networks, recommended by [Goodfellow et al., 2016]. Although the architecture was functional, it quickly turned out to be inadequate as even minimizing the training error was challenging. Therefore, we decided to add one more fully connected layer to increase the representational capacity. This quickly improved both the speed and stability of training the network. We also tested the network without having dropout, batch normalization or both, but those experiments resulted in poorer performance.

Smaller amounts of neurons were also tested, but the reduced representational capacity didn't seem to be able to capture the complexities of the data. However, due to limitations in time and computational power, we didn't manage to test larger amounts of neurons.

Using the deep neural network architecture described in section 3 we implemented our model using TensorFlow. Our TensorFlow model graph can be seen in Figure 1. We trained our model with various different hyperparameter values, namely batch size and learning rate. For learning rates we tried values ranging from 0.1 to 0.00001 and for batch sizes 25, 50, 100, 200 and 500. Additionally we tried training the model with different batch sizes such as 1, 5, 20 and 30+. Training the model for multiple epochs is absolutely necessary as the dataset is very small. Training the model with too few epochs results in a mediocre model, which has not converged yet. However, training the model using too many epochs resulted in overfitting. Therefore, early stopping was observed to be necessary to prevent overfitting.

6 Results

With our simple model we achieved an accuracy of - percent. Our results can be seen in the Table 2 below.

Learning rate	Batch size	Training accuracy	Test accuracy	Loss
0.01	50	0.1401	0.1771	2.304
0.01	100	0.2100	0.1616	2.135
0.0005	50	0.8800	0.6480	1.124
0.0005	100	0.8600	0.6464	1.098
0.0001	50	0.7600	0.6325	1.152
0.0001	100	0.5800	0.5700	1.114
0.00005	50	0.7200	0.6220	1.103
0.00005	100	0.7200	0.6112	1.111
0.00001	50	0.4600	0.5367	1.332
0.00001	100	0.4800	0.5137	1.401

Table 2: Results for different learning rates and batch sizes. The best result is highlighted.

From the results, we can see that having too small learning rate results in convergence happening too slow or never and having a learning rate that is too big results in the model not learning at all. The best values are in the range of 0.00005 and 0.0005. In the table above there are only two different batch sizes of which the batch size of 50 give best results. Even so these two batch sizes yield results relatively close to each other. We also tested with batch sizes of 25, 200 and 500 but for simplicity we discarded them from the table. Nevertheless, we observed that these batch sizes give significantly worse results.

Figures 2, 3 and 4 below illustrate how the training accuracies, test accuracies and loss vary over epochs and with different hyperparameters. As the descriptions are missing for individual graph lines, refer to the Table 2 above for accurate results.

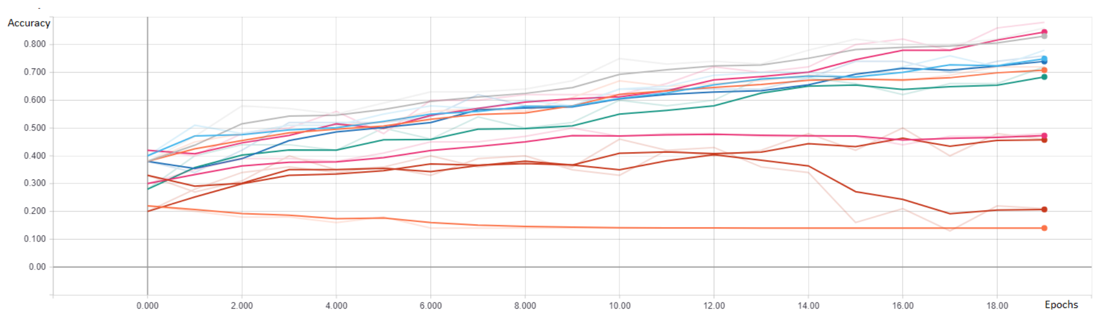

Figure 2: Training accuracies of different batch sizes and learning rates during epochs. Top most lines are the results with learning rate of 0.0005. Bottom most lines are the results with learning rate of 0.01.

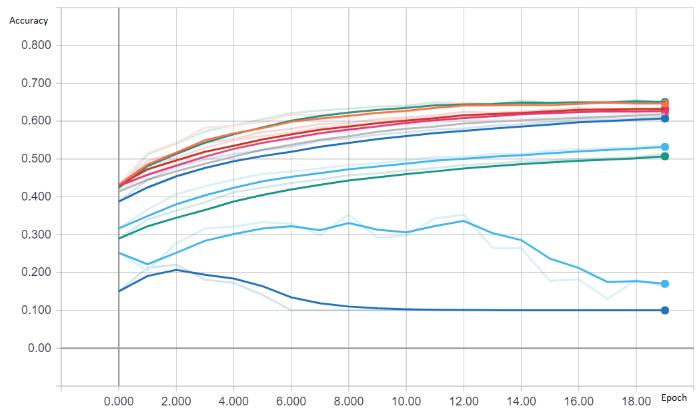

Figure 3: Test accuracies of different batch sizes and learning rates during epochs. Top most lines are the results with learning rate of 0.0005. Bottom most lines are the results with learning rate of 0.01.

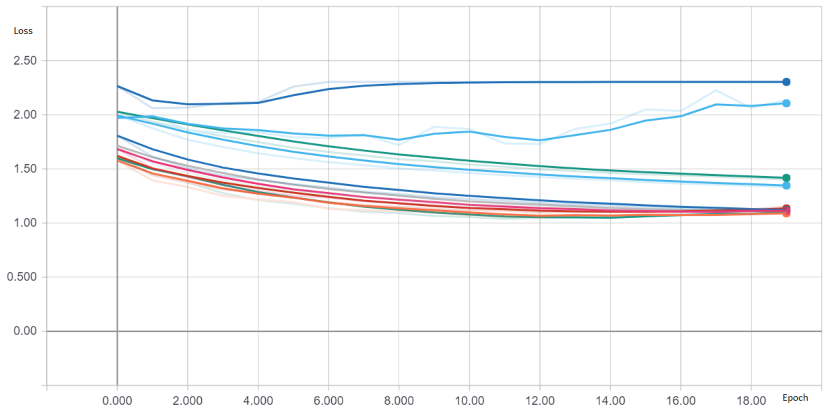

Figure 4: Cross entropy of different batch sizes and learning rates during epochs. Bottom most lines are the results with learning rate of 0.0005. Top most lines are the results with learning rate of 0.01.

7 Discussion

We achieved a test accuracy of roughly 65%, which is somewhat acceptable for a simpler model with limited training and optimization time, as the 49th best model in [Benenson, 2016] achieved a test accuracy of 76%. Nonetheless, compared to the accuracy of the reference model, 89%, our results are quite poor. One important aspect to note, though, is that our model has roughly only 3.4 million trainable parameters, which is only 5.7% of the 60 million parameters that the reference model has. Depending on the application, this might be a crucial factor, as the difference in the amount of parameters makes our model several times faster to train and evaluate, while also requiring less memory.

Although the task of classifying images into one of 10 classes is relatively easy, the CIFAR-10 dataset is considerably small. Our best test accuracy was obtained while having only 88% training accuracy. Training the network further increased the training accuracy, but test accuracy started to decrease, which indicates overfitting. To solve these problems, at least data augmentation should be used, as in the reference model and in many other approaches.

We also found that batch normalization greatly increases the stability of training, while also having a slight generalizing effect and speeding up the training. The model used in [Krizhevsky et al., 2012] doesn't use batch normalization, which could be an improvement.

8 Conclusions

All in all, we didn't spend much time training the network or optimizing other hyperparameters than learning rate and batch size. Additionally, our model started to show signs of overfitting after only a relatively small amount of training time. These factors might suggest that better results could be obtained with more timed spent on optimization and regularization.

Our model didn't manage to achieve comparable results to the reference model, because we focused more on speed and simplicity. In terms of speed, our model is superior to the reference model as our network trained in roughly 2-3 minutes (with one GTX 1080 GPU), where as it took five to six days (with two GTX 580 GPUs) of training in our reference method [Krizhevsky et al., 2012] and 4-5 hours (using Tesla K20m/K40m) of training in TensorFlows method [TensorFlow, 2015].

9 Roles of the authors

Christian worked more on data preparation and analysis as well as optimizing the hyperparameters and visualizing the results. Eetu worked more on the architecture, researching previous works and implementing batch normalization and dropout. Overall, both authors worked on all aspects of the project and the report.

References

[Benenson, 2016] Benenson, R. (2016). Classification datasets results. (Accessed on 31/01/2018).

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[Graham, 2014] Graham, B. (2014). Fractional max-pooling. *CoRR*, abs/1412.6071.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

[Krizhevsky et al., 2014] Krizhevsky, A., Nair, V., and Hinton, G. (2014). The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html.*

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc.

[Russakovsky et al., 2015] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252.

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958.

[TensorFlow, 2015] TensorFlow (2015). A binary to train cifar-10 using a single gpu. (Accessed on 31/01/2018).