Explore Training ConvNets for Image Classification

Jakob Melki **Muhammed Memedi** **Seyed Hesam Ahmadi** **Mehrdad Bahmani**

Abstract

The purpose of this project was to explore training convolutional neural networks (CNNs). The problem that the CNN had to solve, was to classify images from a given dataset, CIFAR-10. The CNN was built according to a similar model of AlexNet and different training methods such as dropout, batch normalization, data augmentation and different optimizers (stochastic gradient descent and Adam) was applied. The networks test accuracy for the different experiments varied between 53 % to 63 %. The results also showed that the SGD optimizer reduced overfitting significantly compared to the Adam optimizer, but with a cost of the test accuracy. The effects from dropout and data augmentation showed to be a decrease of overfitting, while the opposite occurred with added batch normalization.

1 Introduction

Image classification has an important role in many different areas. Some examples of such fields are robotics, security systems, search engines and medical imaging. In robotics (more specifically driver-less vehicles and domestic robots) image classification are used to detect static and dynamic objects, which make it easier for the robots to interact with the environment. In security systems, image classification can be used for face recognition and in the medical area, it can be used to detect diseases, for example cancer, from x-ray images. Finally, image classification is also important for search engines, such as Google, to search and sort images.

Convolutional neural networks (CNNs) is the most popular network when it comes to image classification problems. Compared to a fully connected neural network, CNN has fewer parameters, which makes training faster, and needs less training data.

The problem that we worked on in this project was image classification. By a given initial set of images, the network was trained and after some time, it was able to classify newly seen images correctly. Another purpose of this project was to explore training CNNs for image classification. Different methods such as dropout, batch normalization and data augmentation was implemented for exploration of their effect of the CNN's performance. Furthermore, we also tested two different optimizers, Adam and stochastic gradient descent (SGD) optimzers. The results showed that we achieved less overfitting with the SGD optimizer compared with the Adam optimizer, but also with a drop of the test accuracy. The presence from dropout and data augmentation, seemed to reduce overfitting, while the opposite was shown by added batch normalization.

2 Related work

A paper that has studied similar problem is given by (1). The model in this paper is based upon the first version of the AlexNet, i.e. when it was introduced by Alex Krizhevsky for the first time. This model was at the time a large CNN and the purpose was to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest. This model outperformed its competition with a top 1 and top 5 test error rates of 37.5 % and 17 %, which was considerably better that the previous state-of-the-art. To make training faster, they also used non-saturating neurons and a very efficient GPU implementation. They also used dropout to reduce overfitting.

Another project that also explored training CNNs with the CIFAR-10 dataset is given by (2). Here, they examined how different activation units and dropout affected the CNN's performance. They compared two activation units, ReLu and sigmoid and discovered that ReLu lead to about 5 times faster training, with still the same accuracy as sigmoid at convergence. For the dropout experiment, they concluded that adding dropout leads to more regularization and less overfitting.

3 Data

The data we are working with is the well known CIFAR-10 data, which was accessed through the Keras package in python. It consists of 6000 32 x 32 colour images in 10 classes with 6000 images per class, 50000 for training and 10000 for test. The training data contains 5000 images from each class and is divided into 5 batches, each consisting of 10000 images, while the test data consists of 10000 images. Furthermore, the test data contains exactly 1000 random selected images from each class. The training batches contain the remaining images in random order, with some batches containing more of some classes than others. See figure [1] for samples of the CIFAR-10 images.

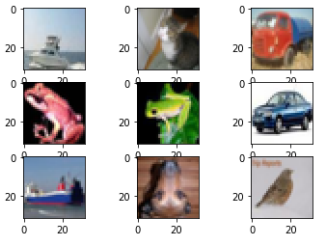

Figure 1: Random samples from the CIFAR-10 dataset.

In this project the whole data was used, 50000 for training and 10000 for test. For hyperparameter searches, the training data was split according to 70 % for training and 30 % for validation. No prepossessing or any special treatment of the data was made during this project.

There are several state of the art methods applied to the CIFAR-10 dataset. Some with the highest classification accuracy are Bit-L, GIPIPE + transfer learning, both having 99 % or above in classification accuracy. The Bit-L model uses transfer of pre-trained representations to enhance sample efficiency and simplify hyperparameter tuning. The GPIPE + transfer learning model uses a pipeline parallelism library to scale the network.

4 Methods

4.1 Convolutional neural networks

Convolutional neural netowrks, or CNNs for short, were proposed by a computer scientist Yann LeCunn in the late 90s, when he was inspired by the human visual perception of recognizing objects (5). They were developed mainly for image classification, but have achieved success in other areas too, such as text processing (7). The trend started with AlexNet in 2012 with 8 layers and has now progressed up to 152 layer with ResNet (8).

One of the strengths of CNNs is that they are very good to handle large and unstructured data, which make them suitable for problems such as image classification. They have tend to show good results on difficult classification tasks that require understanding abstract concepts in images. (8)

Another special property of CNNs, is that they do not require any special prepossessing of the data to derive features. Instead, they take the data as it is and learns feature extraction on their own. Researches have discovered that CNNs are very flexible and can exploit patterns and useful features from raw data. For example, given a set of images of dogs and cats, the CNN can learn the key features that separate the cats and dogs easily on its own. (8)

Finally, CNNs are very effective in reducing the number of parameters in a network without affecting the performance of the network. This can be very useful for deep networks that require a lot of

parameters and thus a lot of computational time. However, as mentioned, with CNNs one can overcome this problem with reducing the amounts of parameters without affecting the performance. (6)

4.2 AlexNet

In 2012, a convolutional neural network called AlexNet, named after its designer Alex Krizhevsky won the ImageNet Large Scale Visual Recognition Challenge. The network gained popularity for its good performance in image classification.

A reason why AlexNet performs so well is because it uses the ReLu activation function instead of the tanh. For a performance of 25% error on the CIFAR-10 dataset, using ReLu instead of tanh, which was the standard at the time, is up to 6 times faster.

Because of the many layers, AlexNet had 60 million parameters. To avoid overfitting, dropout was introduced into the network. Forcing random neurons to turn off also forces the network to become robust. Another reason for AlexNet's success is that it uses overlap in pooling. This technique makes it harder for the network to overfit. (4).

In this section, we base our model on the AlexNet CNN and perform various experiments on it. The base model is presented in table 1.

Table 1: Layerwise description of our similar AlexNet model. Padding: same refers to adding an amount of padding such that the input shape is not changed.

Layer type	Details	Output (width, height, depth)
Conv	96 filters, size 3 x 3 at stride 4, pad 0	(8, 8, 96)
Max pool	2 x 2 regions at stride 2	(4, 4, 96)
Norm	Batch normalization layer	(4, 4, 96)
Conv	256 filters, size 5 x 5 at stride 1, padding: same	(4, 4, 256)
Max pool	2 x 2 regions at stride 2	(2, 2, 256)
Norm	Batch normalization layer	(2, 2, 256)
Conv	384 filters, size 3 x 3 at stride 1, padding: same	(2, 2, 384)
Conv	384 filters, size 3 x 3 at stride 1, padding: same	(2, 2, 384)
Conv	256 filters, size 3 x 3 at stride 1, padding: same	(2, 2, 256)
Max pool	2 x 2 regions at stride 2	(1, 1, 256)
Norm	Batch normalization layer	(1, 1, 256)
Fully connected	4096 neurons	(4096)
Fully connected	4096 neurons	(4096)
Fully connected	10 neurons (class scores)	(10)

5 Experiments

We built a CNN according to table 1 and trained it for 20 epochs with batch size 100. We used 50000 samples for training set and 10000 samples for the test set.

First the model was compiled with an Adam optimizer with learning rate $\eta = 0.001$, and parameters $\beta_1 = 0.9, \beta_2 = 0.999$. See figures 2 and 3 for the accuracy and loss plots for this part.

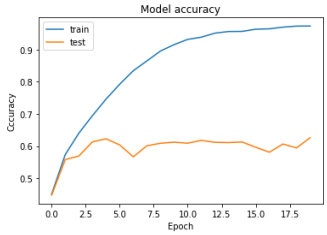

Figure 2: Accuracy plot of the base model with Adam optimizer.

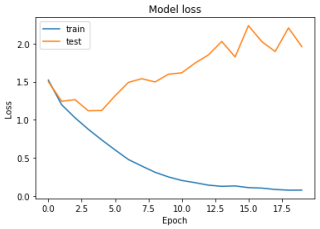

Figure 3: Loss plot of the base model with Adam optimizer.

Thereafter, we explored how dropout, batch normalization, SGD optimizer and data augmentation affected the CNN by implementing them one at a time.

For the dropout experiment, we added dropout for the first and second fully connected layer with 0.4 dropout rate. See the results in figure 4 and 5. Note that the optimizer here, was the previous Adam optimizer.

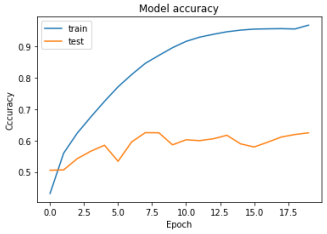

Figure 4: Accuracy plot of the base model with Adam optimizer and added dropout.

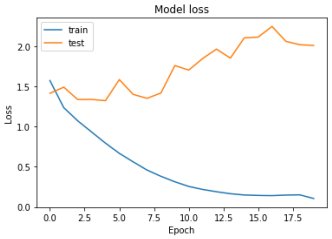

Figure 5: Loss plot of the base model with Adam optimizer and added dropout.

To see how batch normalization affected the CNN, we added batch normalization for the first and second fully connected layer. See the results in figure 6 and 7. Note that there was no dropout in this part and that the optimizer was Adam.

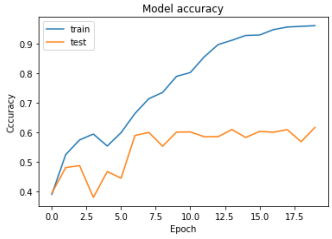

Figure 6: Accuracy plot of the base model with Adam optimizer and added normalization layers.

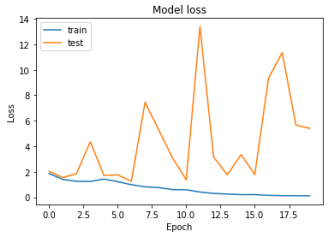

Figure 7: Loss plot of the base model with Adam optimizer and added normalization layers.

Then, we switched to the SGD optimizer with a learning rate of $\eta = 0.001$, with Nesterov's accelerated gradient and no decay. We repeated the exact same experiments as with Adam optimizer, but now with the SGD optimizer instead. See figures 8 - 13 for results.

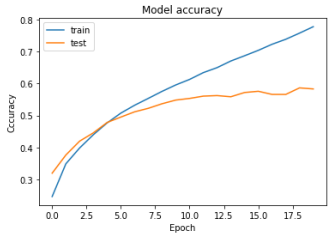

Figure 8: Accuracy plot of the base model with SGD optimizer.

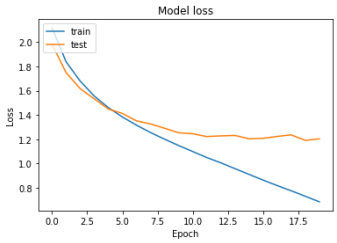

Figure 9: Loss plot of the base model with SGD optimizer.

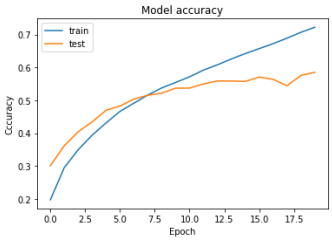

Figure 10: Accuracy plot of the base model with SGD optimizer and dropout.

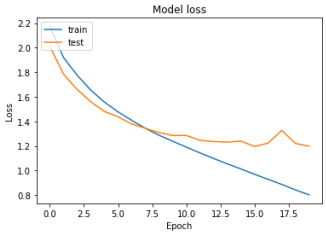

Figure 11: Loss plot of the base model with SGD optimizer and dropout.

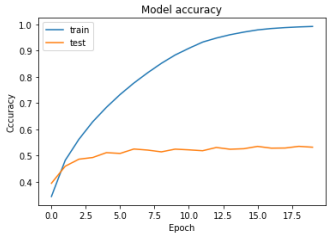

Figure 12: Accuracy plot of the base model with SGD optimizer and normalization.

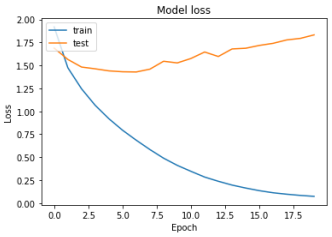

Figure 13: Loss plot of the base model with SGD optimizer and normalization.

Next, we introduced data augmentation to make the model more robust. This consisted of four transformations: 0-20 degrees of rotation, horizontal flipping, width shifting and height shifting. Each transformations was applied to random images in the dataset. See figure 14 for some random images were data augmentation was applied. Data augmentation was implemented on the model with SGD optimizer and dropout. See figures 15 and 16 for results.

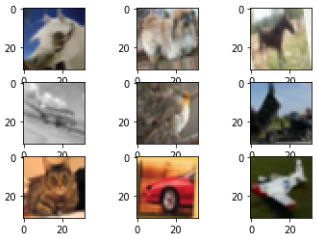

Figure 14: Random samples from the augmented data.

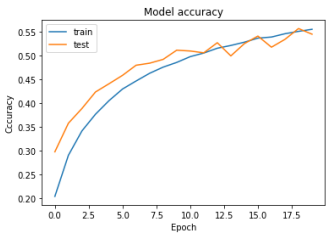

Figure 15: Accuracy plot of the base model with SGD optimizer, dropout and data augmentation.

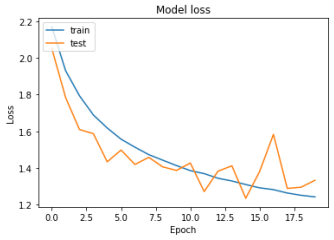

Figure 16: Loss plot of the base model with SGD optimizer, dropout and data augmentation.

Finally, we also did a random learning rate search to find a good value on the learning rate. For this experiment, we trained the SGD + dropout + data augmentation model with the learning rates 0.0005, 0.00075, 0.001, 0.00125 and 0.0015. Since big learning rates achieve higher accuracy faster than smaller learning rates, we extended the learning to 30 epochs, so as to see the performance of the models when they are converging. For time reasons, the training data and validation data was shrunk to 25000 and 6250 samples, respectively.

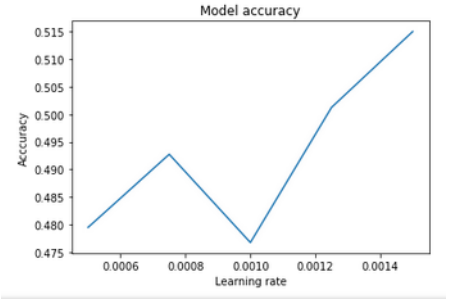

Figure 17: Validation accuracy as a function of learning rate.

The learning rate corresponding to the highest accuracy shown in figure 17 will be used to test our enhanced model. See figure 18 and 19 for the results of our enhanced model with learning rate $\eta = 0.00075$.

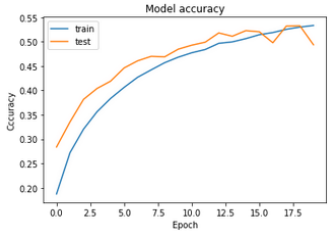

Figure 18: Accuracy plot of the base model with SGD optimizer, dropout and data augmentation with learning rate $\eta = 0.00075$

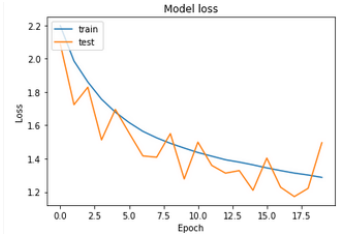

Figure 19: Loss plot of the base model with SGD optimizer, dropout and data augmentation with learning rate $\eta = 0.00075$

To summarize all results from our experiments, we wrote down the final test accuracy and loss for each experiment, see the table below.

Table 2: Test accuracy and loss of the different models. lr = learning rate, DA = data augmentation, Norm = batch normalization.

Model type	Test loss	Test accuracy
Base + Adam	1.959	0.626
Base + Adam + Norm	5.407	0.616
Base + Adam + Dropout	2.013	0.625
Base + SGD	1.203	0.583
Base + SGD + Norm	1.833	0.532
Base + SGD + Dropout	1.984	0.585
Base + SGD + Dropout + DA	1.265	0.550
New lr + SGD + Dropout + DA	1.506	0.479

6 Conclusions

From the experiments with the Adam optimizer, we can see in figures 2, 4 and 6 that the network gets overfitted to the training data. Even though dropout should reduce overfitting according to the theory, one does not see any significant reduce of overfitting in the experiment with Adam and dropout. However, we can also observe a slight higher accuracy on the test data with the Adam optimizer compared to the SGD optimizer.

From the experiments with the SGD optimizer, we can see that we get less overfitting and smoother curves (better convergence) compared to the Adam optimizer. The downside, is that the test accuracy drops a little bit. This is seen for example in figures 2 and 8.

Furthermore, we can see from the dropout experiments with the SGD optimizer, that the dropout indeed contributes to a reduce of overfitting. However, the opposite (increase of overfitting) and a slight reduce of test accuracy is achieved with added batch normalization, which can be seen from the results with SGD and added batch normalization, figure 12. In addition, we can see that it also make the convergence worse for the Adam optimizer, illustrated in figure 6 and 7.

From the results of the implementation of data augmentation, figure 15 and **??**, we can see that the network is not overfitted at all and still has a good test accuracy even if it dropped a little bit.

Thus from our observations above, we draw the following conclusions for our model. The SGD optimizer works better than the Adam optimizer since it leads to a much better convergence and reduce of overfitting. Dropout and data augmentation leads to less overfitting, while batch normalization increases overfitting. However, from our findings, it also seems that there is a trade of between overfitting and test accuracy in our case. If we reduce the overfitting, there is a risk that the test accuracy also drops a little bit.

From the learning rate experiment in figure 17 we see that the performance has a local maximum at $\eta = 0.00075$ and decreases at our base $\eta = 0.001$. After $\eta = 0.001$ the performance increases with the learning rate. One explanation to this could be that the performance is measured at epoch 30, so that small learning rates are simply learning too slow by that time. Comparing figure 15 with $\eta = 0.001$ and figure **??** with $\eta = 0.00075$ shows slightly worse performance on the latter learning rate. This could be because the small learning rate caused the learning to get stuck in a local minima or its simply just a coincidence due to the stochastic nature in the learning process.

References

[1] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural net-works. InNIPS, 2012

[2] Wang, Chen. "Convolutional Neural Network for Image Classification." (2015).

[3] Image Classification on CIFAR-10, Paperswithcode. https://paperswithcode.com/sota/image-classification-on-cifar-10.

[4] J.Wey. AlexNet: The Architecture that Challenged CNNs. towards data science. 3 Jul 2019. https://towardsdatascience.com/alexnet-the-architecture-that-challenged-cnns-e406d5297951

[5] K. Maladkar. Overview Of Convolutional Neural Network In Image Classification. Analytics India Magazine. 25 Jan 2018. https://analyticsindiamag.com/convolutional-neural-network-image-classification-overview/

[6] P. Mishra. Why are Convolutional Neural Networks good for image classification?. Medium. 27 May 2019. https://medium.com/datadriveninvestor/why-are-convolutional-neural-networks-good-for-image-classification-146ec6e865e8

[7] S. Bhatt. Understanding the basics of CNN with image classification. Becoming Human. 4 Oct 2019. https://becominghuman.ai/understanding-the-basics-of-cnn-with-image-classification-7f3a9ddea8f9

[8] R. Bhatia. Why Convolutional Neural Networks Are The Go-To Models In Deep Learning. Analytics India Magazine. 25 Sep 2018. https://analyticsindiamag.com/why-convolutional-neural-networks-are-the-go-to-models-in-deep-learning/