DD2424
May 2024

Developing a CNN From Scratch

By Tina Lagerblad, Leonardo Rezza, Alexander Åblad

Abstract

In this project we develop a convolutional neural network (CNN) from scratch, and then go through several experiments, with the goal of reaching a high accuracy in classifying images from the CIFAR-10[1] dataset. We base our basic model and first three extensions off of a detailed guide by Jason Brownlee[2]. Using this basic CNN, we manage to achieve accuracy of ~74%, which we use as a baseline for our experiments. Now our goal is to improve upon this through various methods. In our first rounds of experiments we try adding 20% dropout after each max layer, L2 weight decay with a coefficient of 0.001%, and finally augmenting the training data using x and y shifts as well as flipping images along the x-axis. Applying each of these regularization methods independently to the data resulted in accuracies of ~82%, ~73% and ~83%, respectively. So, we can infer that in our case, including dropout and data augmentation both improve the model to a similar degree, while weight decay makes accuracy slightly worse. We then tried a combination model with both dropout and data augmentation added to the base model, as well as batch normalization. This made for a classification accuracy of ~88%, making the combination model our most effective one up until this point. Various experiments were then conducted such as varying the optimizers, using learning rate scheduling, and experimenting with the regularization, where it was concluded benefits could be gained by using some, but where many may be heavily dependent on the chosen parameters.

Introduction

For our project, the task at hand was to classify images in the CIFAR-10[1] image dataset using a deep CNN, created from scratch, and do so at specified level of accuracy. A CNN is a type of artificial neural network that is commonly used for image recognition or object recognition. Next, we were tasked to make several explorations, in making additions and adjustments to our base neural network, and once again achieve the specified target results.

When we had created the basic network to use as our baseline, we experimented with adding different types of regularization to try to improve network performance. The regularization methods used were adding dropout throughout the network, applying weight decay in the form of L2 regularization, as well as data augmentation. To this end, we could follow a guide that clearly defined these steps in an article posted on Machine Learning Mastery[2]. After following this guide to create our network, and verifying that our results are as to be expected, we continue on to some explorations beyond what is defined in the article in order to further improve performance. First, we normalize the data to have values between 0 and 1 and ensure a normal distribution. Next, the stochastic gradient descent (SGD) with momentum is replaced with Adam (adaptive moment estimation) and then AdamW. We also try changing the learning rate schedulers, and the order of Dropout and BatchNorm, as well as taking one out. The goal of these experiments is to see the effect on our performance, learn about how these methods interact with our model, and hopefully find some improvements to apply.

Related Work

The guide that we follow as the basis for this project is "How to Develop a CNN From Scratch for CIFAR-10 Photo Classification" by Jason Brownlee, posted on the Machine Learning Mastery website[2]. Brownlee lays out a clear, step-by-step guide that can be followed to create a CNN, a few extensions, and results to compare to with discussions. It was a great tool both for setting up our model and experiments, but also learning more about the concepts and interpreting our own results.

It should be noted that the methods described are very widely documented, as this is a quite common project in practicing developing CNN's for image classification, so there is a lot of previous work related to this topic aside from just this article as well.

Data

The CIFAR-10 dataset is a well known and commonly used data set for developing classification algorithms in machine learning. It contains 10 classes of 6,000 small images each, for a total of 60,000 images. In our project, we use it for creating and evaluating our CNN in order to test its performance both on its own as well as with additional extensions and experiments. We use TensorFlow and Keras to load the dataset, define our model, build various extensions, and then train and evaluate everything.

Method

The code for this project was entirely written in Python, using a variety of packages centered around TensorFlow and Keras as used in the article[2] which the project is based on. The models were trained locally on different personal computers. As per the project instructions, the same baseline model from the article was implemented, along with three variations using different methods of regularization. To verify the correctness of the models, the graphs and test accuracies could be compared to the ones in from the article.

Experiments and Results

Baseline model

The baseline model using SGD + momentum and no regularization was trained for 100 epochs on the training data, where the loss and accuracy on the training and validation data for each epoch during training can be seen in figure 1. Interpreting these visuals, we can tell that the data is extremely overfit from the loss plot, and that our accuracy levels out around 70%. The final accuracy on the test data was 73.81%, which is close to the accuracy from the article[2] of around 73%.

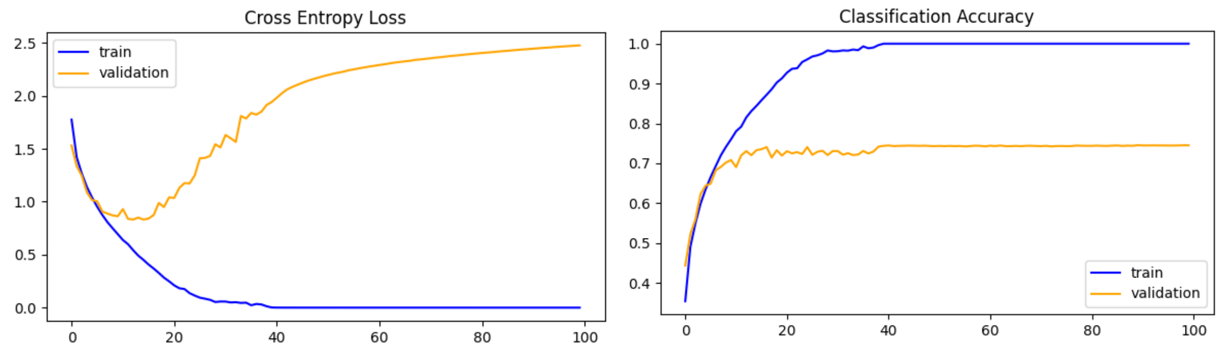

Figure 1. Loss and accuracy during training for the baseline model

Dropout

Randomly dropping nodes from our network at a fixed 20% rate after each max pooling layer and the fully connected layer resulted in our models test accuracy being improved, now at 82.12%, which is close to the article's[2] result of 83%. These results can be seen below in figure 2, which show the loss and accuracy on the training and validation data for each epoch. Here we see great improvements compared to our base model, with overfitting drastically reduced, and accuracy significantly increased.

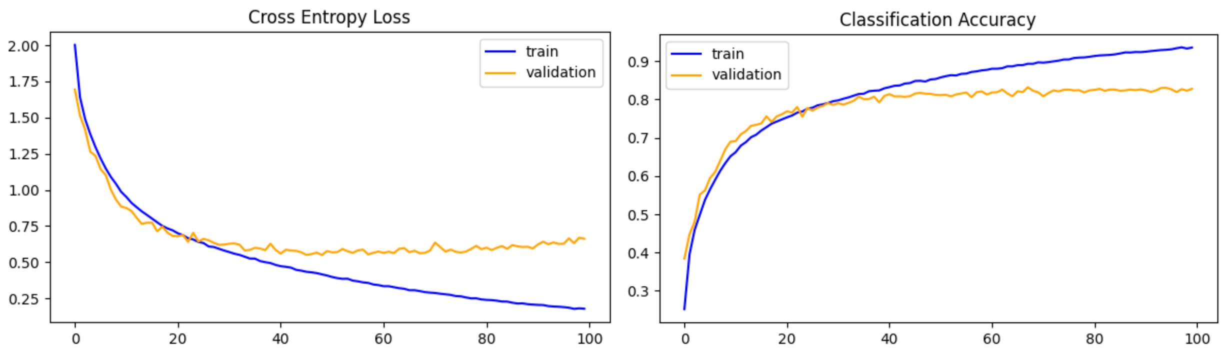

Figure 2. Loss and accuracy during training for model using dropout

Weight decay

Adding regularization in the form of L2 weight decay penalizes the model for having large weights, promoting simpler and hopefully more generalizable models. We do so by adding a term to the loss function proportional to the model weights with a coefficient of 0.001. Adding this weight decay resulted in a model accuracy of 72.95%, meaning that we actually experienced a loss in accuracy. This accuracy is close to the reported accuracy of 72% in the article[2]. Our loss and accuracy results after 100 epochs are visualized below in figure 3. Compared to the base results in figure 1, we can see that while we do reduce overfitting, it is not as effective as when we applied dropout.

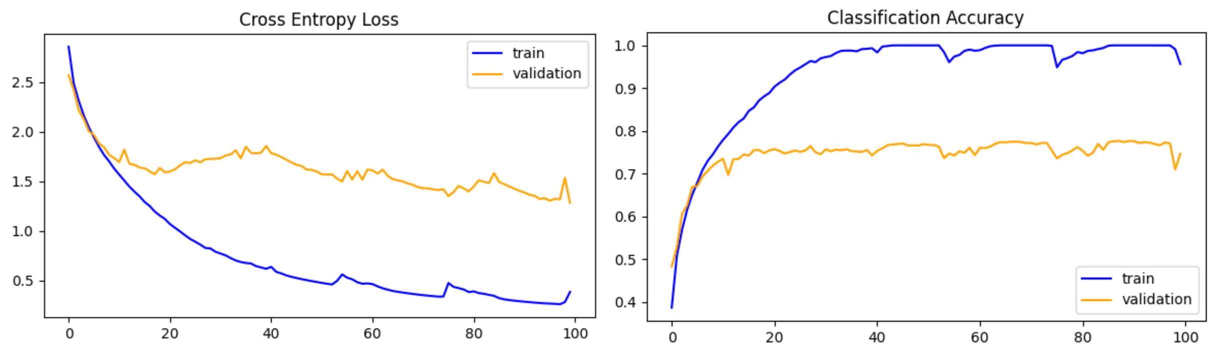

Figure 3. Loss and accuracy during training for the model using weight decay

Data augmentation

Augmenting the training dataset in terms of shifting the height and width by 10% as well as flipping along the horizontal axis is a way in which we make copies of images while not distorting to the point of losing relevant features. This is a regularization method, as the model is able to learn from a larger set of the same general features, possibly improving generalizability. Applying this method resulted in a test accuracy of 82.66%, which is an improvement upon the base model, but only slightly better than with dropout. The resulting accuracy gives us confidence in our implementation, since the article[2] reports an accuracy of 84%. The results after 100 epochs are seen in figure 4, which show loss and accuracy having very similar results to using dropout in figure 2.

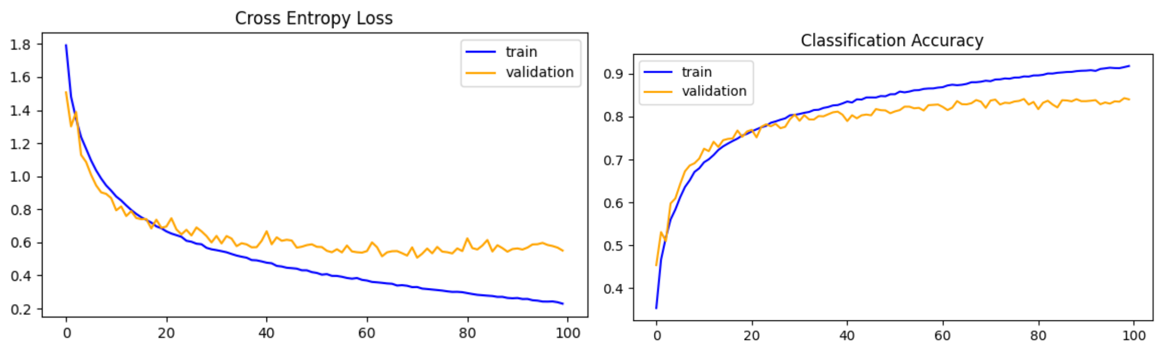

Figure 4. Loss and accuracy during training for the model using data augmentation

Model with regularization and batch normalization

This combined model was implemented with two methods of regularization in form of dropout and data augmentation. In addition to this, batch normalization is also used which required the model to train for a larger amount of epochs compared to the previous models. The model achieved an accuracy of 87.82% on the test data after training for 400 epochs, similar to the reported accuracy of 88% in the article[2]. The loss and accuracy during training can be seen in figure 5. It is worth to note that the loss and accuracy graphs for the training and validation data both look quite similar, the generalization gap during training being noticeably smaller compared to the other models.

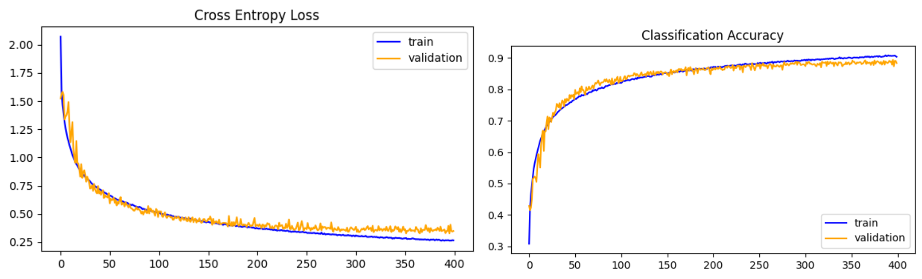

Figure 5. Loss and accuracy during training for the combined model using dropout, data augmentation and batch normalization

Centering around zero

Further experimentation was made to try to observe the impact of certain factors and varying model parameters. One such experimentation was to normalize the data to have zero mean and standard deviation 1. To directly compare the effect of this, the combined model was used and trained for 400 epochs. The final accuracy was 88.25%, which is practically the same as the previous configuration, meaning it did not have a significant impact.

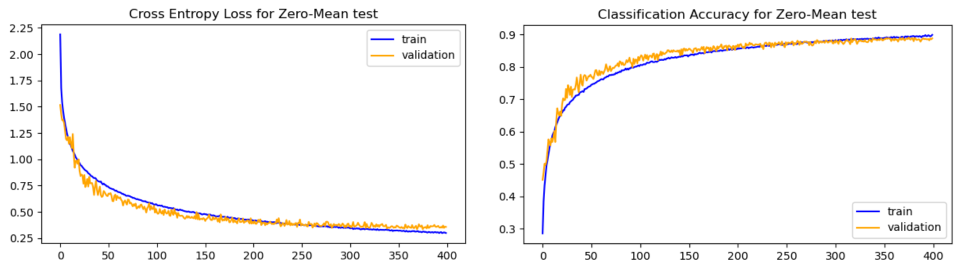

Figure 6. Loss and accuracy during training for the combined model using dropout, data augmentation and batch normalization and normalizing the data with zero mean and std 1

Adam and AdamW optimizers

Second, a couple of optimizers other than SDG were tested. These were the Adam and AdamW optimizers. Adam reached an accuracy of 86.99% and AdamW 41.3%.

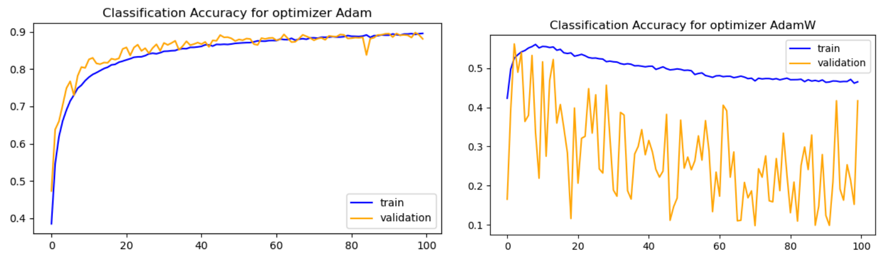

Figure 7. Accuracy during training for the combined model using the Adam and AdamW optimizers respectively

The Adam optimizer seems to reach comparable results to SGD, and perhaps even slightly quicker. The AdamW optimizer, which is the Adam optimizer with weight decay, performed poorly. Using the documentation default values of intial_learning_rate = 0.001 and weight_decay = 0.004, it seems to have dropped off too early, most likely getting stuck in a bad local minima. A larger initial learning rate could perhaps prevent this.

Learning rate scheduling

Learning rate scheduling techniques such as Learning rate warm-up + Cosine Annealing, Cosine annealing with Restarts and Step Decay were experimented with. Due to compatibility issues, the tooling configuration did not allow using warm-up with Keras learning rate scheduling. Therefore, Cosine Annealing without Warm-up was experimented on instead. One aim of Cosine Annealing is to help escape bad local minima, but here it slightly offset the accuracy to the worse, right before the end of training. Better parameter tuning could minimize this. Step decay used experimental values with boundaries = [1400, 2800, 10000] and values = [0.1, 0.01, 0.001, 0.0001]. Such specifiable parameters should instead be chosen with greater consideration and knowledge.
Final accuracies: 87.86%, 84.50% and 64.42% respectively.

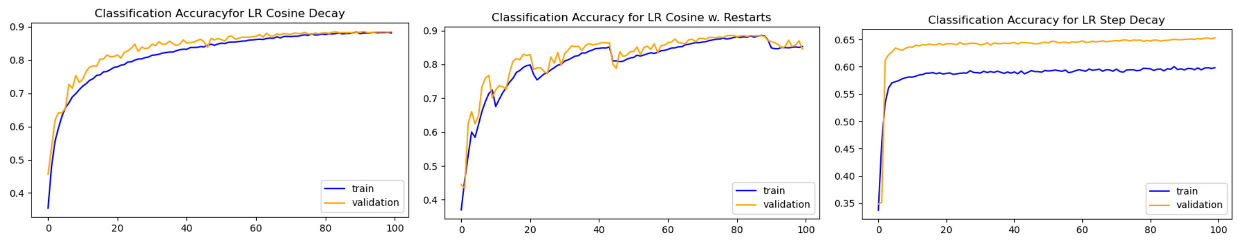

Figure 8. Accuracy during training Cosine Annealing with and without restarts, and Step Decay

Complementary impact of BatchNormalization and Dropout

Lastly, experiments trying changing order and filtering on the BatchNormalization and Dropout techniques were conducted. Switching the order of BatchNormalization and Dropout and checking the performance of only using either BatchNormalization or Dropout, carried out on the default ordering. Note that the interpretation of "switching order" was switching out one technique to the other. The project specification was not clear on this. Seemingly, switched techniques eventually converge if you let it train long enough. Reverse ordering still had a positive trend at 100 epochs. Only having Batch Normalization seems to have resulted in a slight overfitting after 30-40 epochs. As for only using Dropout, it boosted the validation accuracy relative to the training accuracy, and at 100 epochs, almost matching only using Batch Normalization. Further training could possibly improve it. Final accuracies: 59.49%, 83.94% and 80.17% respectively.

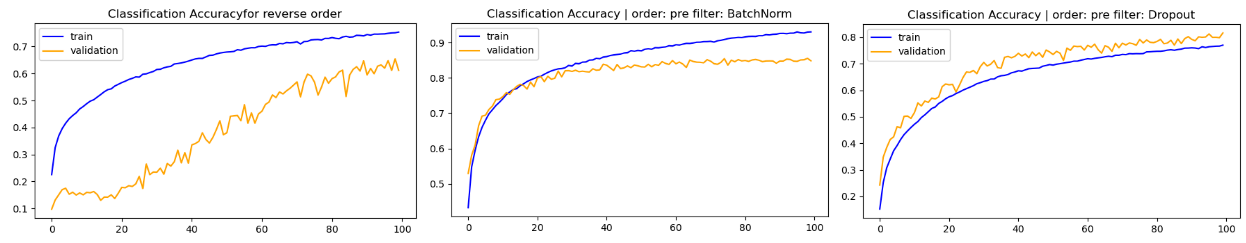

Figure 9. Accuracy during training for Reverse Ordering of Batch Normalization and Dropout, only using Batch Normalization, and only using Dropout.

Conclusion

Working on this project has given valuable experience in working with CNN and experimenting with a few basic techniques. First having implemented the model step by step and then running a few experiments has provided valuable insights into how these techniques affect the models performance. In this case, mostly default values were used during experiments. It is to future investigation suggested to try various values for those cases where the model performed poorly and see if they can work.

Works Cited

1. Krizhevsky, Alex, et al. “CIFAR-10.” *CIFAR-10 and CIFAR-100 Datasets*, www.cs.toronto.edu/~kriz/cifar.html.
2. Brownlee, Jason. “How to Develop a CNN from Scratch for CIFAR-10 Photo Classification.” *MachineLearningMastery.Com*, 27 Aug. 2020, machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/.