Assignment 1: CIFAR-100 Image Classification

Jasmine Sun

j.h.sun@wustl.edu

UCF Computer Vision REU

May 27, 2024

1 Introduction

In this assignment, I designed a sequential convolutional model in Python using the Keras API and other relevant libraries to classify the CIFAR-100 dataset. CIFAR-100 consists of 60000 32x32 images in 100 classes, with 600 images in each. The goal was to understand the process of processing and formatting data to be trained, create a model with as high accuracy as possible, train the model, evaluating its performance, and visualizing the results.

2 Method

My final sequential convolutional model consists of 6 convolution layers with Batch Normalization, Dropout, MaxPooling, Flattening, and Dense layers, which are visualized in Figure 1. The next section will cover my reasoning for these layers.

Parameters:

- Optimizer: SGD
- Number of Epochs: 88
- Batch Size: 128
- Learning rate: 0.01

Figure 1

3 Experiment

My strategy for creating a model largely consisted of testing out different models that have been used for both CIFAR-100 and other datasets such as CIFAR-10 and MNIST as a baseline, and then I tested removing/adding certain layers and tweaking parameters to see their effects. The sections below outline the main things I tested while developing my model.

3.1 First Attempt

I implemented VGG-16 architecture as my model. As for parameters, I had a batch size of 128, 5 epochs, 0.01 learning rate, and the Adam optimizer. This resulted in an accuracy of about 0.01, with little improvement between epochs. I noticed that as I removed layers, my accuracy improved to about 0.08, which was unexpected. Because of this, I decided to remove all layers and play around with the parameters instead.

3.2 Optimizer

With a bit of research, I found that other people had found that the SGD optimizer works better than Adam with VGG-16. Even though I had already removed those layers, I decided to implement SGD instead, which increased my accuracy to about 0.28.

3.3 Number of Epochs

A main issue with 5 epochs was that the model was not able to reach constant accuracy because it was not given enough repetitions. To fix this problem, I changed the number of epochs to 100. With early stopping implemented, I could now see more data, which helped me fix overfitting issues.

3.4 Batch Size

I found that decreasing batch size to a much smaller size caused my model to run very slowly, if at all, sometimes using up all available RAM. On the other hand, increasing batch size sacrificed accuracy, so 128 seemed to be a sweet spot.

3.5 Learning Rate

I adjusted the learning rate and concluded that the default setting of 0.01 worked best. At 0.1, the validation accuracy jumped around a lot, producing a jagged graph rather than a smooth curve. At 0.001, it required far too many epochs to produce results.

3.6 Layers

Without layers, I had a problem with overfitting; test accuracy came up to be 0.7-0.9, while validation accuracy still hovered around 0.25-0.35. Starting with convolution layers and MaxPooling modeled after VGG-16 architecture, I found that 6 convolution layers was an optimal number to use – more caused accuracy to decrease. Next, I implemented Batch Normalization after each convolution layer to help with overfitting. Dropout helped as well, which was placed largely by guess and check. Lastly, my output layer, also modeled after VGG-16 architecture, consists of three Dense layers. Like the convolution layers, I found that placing Batch Normalization and Dropout between the Dense layers improved accuracy and overfitting issues.

4 Results

My model produced 1.47 test loss and 0.60 test accuracy. Below are graphs visualizing my results. In blue are validation loss and accuracy, respectively. In orange are training loss and accuracy, respectively.

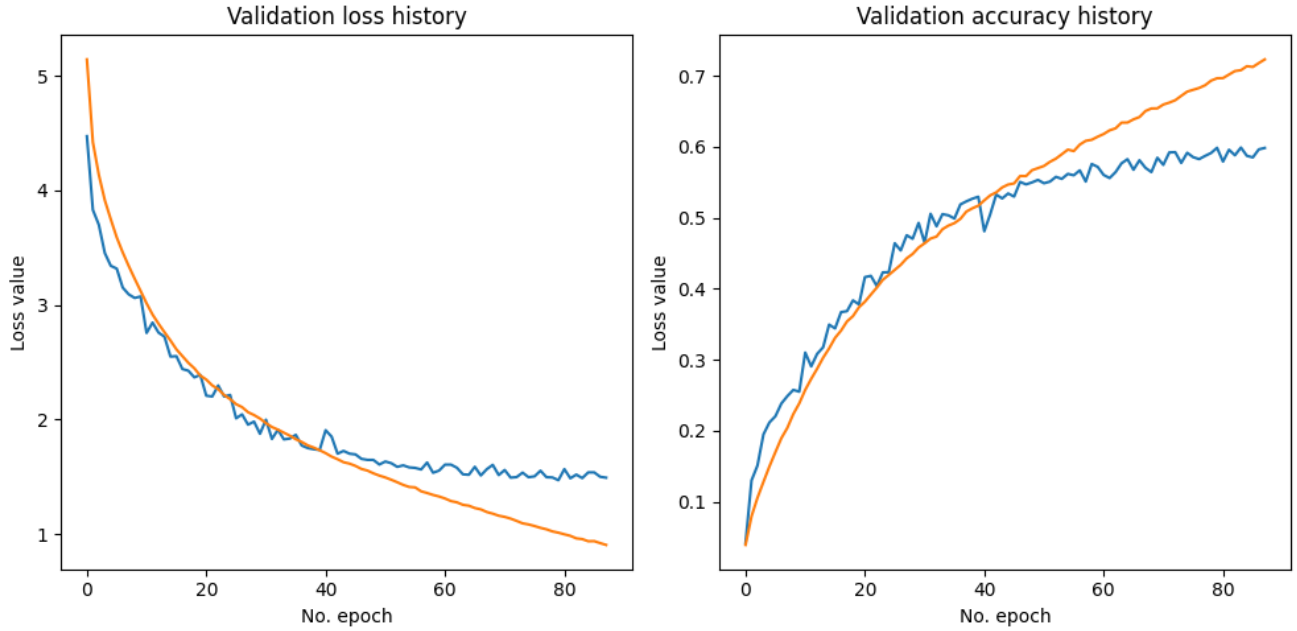

Figure 2

5 Conclusion

By experimenting with different layers of the model, I found that Batch Normalization, Dropout, and early stopping helped most with overfitting. While overfitting is still present after around epoch 40 in my final model, it has been improved upon greatly since my first few models. With more time and a more powerful and unlimited GPU, I could use a smaller batch size and increase layers, which could potentially increase accuracy and decrease overfitting. Furthermore, more time would allow for more in-depth exploration of other functions in the Keras API and other libraries. Beyond these, I could look at other ways to improve my model such as through data augmentation, regularization, or implementing a learning rate schedule.