CIFAR-10 Classifcation Using a Convolutional Neural Network

Written by: Michael Silicia, Joe Lyons, Griffin McMann, Dalton Rothenberger

1 Introduction

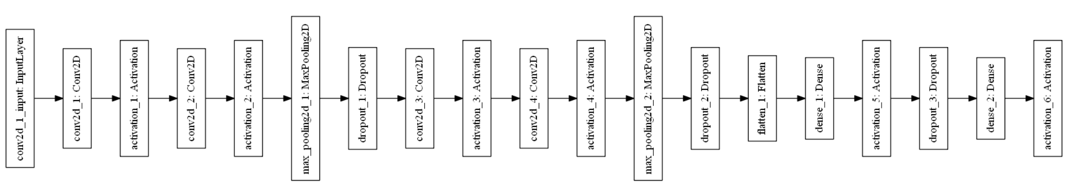
We tackle the CIFAR-10 classification problem with a machine learning model in Python using the Keras neural network library. The CIFAR-10 data-set consists of 60,000 32x32 color images. The data-set is broken into 10 classes, each with 6,000 images. Our goal was to modify components of an existing model, linked here, in order to gain a better understanding about how these components effect the accuracy of the model when classifying images in the CIFAR-10 data-set.

2 Methodology

For testing our model, we modified three components of our convolutional neural network (CNN): Activation Function, Batch Size, and Learning Rate. Each component is tested independently with a total of 27 runs. To expedite the process we run our model using GPU acceleration and use a bash script to change the desired components automatically between runs. Data from each run is organized in excel sheets for comparison.

3 Model Description

The model consisted of four convolutional layers, one flattening layer, and two dense layer. For all test, the final dense layer used a softmax activation. The convolutional layers, the flattening layer, and the other dense layer were the layers that had their activation changed during the testing. The loss function used was categorical cross entropy and the optimizer used was RMSprop. Also, the images were normalized before being trained on. For a more detailed look into the actual code we have a GitHub Repo linked here containing the files from our experiment. Below is a diagram from Keras of our model that shows the various layers.



Submitted to 33rd Conference on Neural Information Processing Systems (NeurIPS 2019). Do not distribute.

4 Data

Note: All tests were run with 100 epochs.

RELU

Batch	Learning Rate	Accuracy
128	.001	.6634
128	.01	.1000
128	.1	.1000
64	.001	.4603
64	.01	.1000
64	.1	.1000
32	.001	.2731
32	.01	.1000
32	.1	.1000

Leaky RELU

Batch	Learning Rate	Accuracy
128	.001	.8440
128	.01	.1000
128	.1	.1000
64	.001	.8102
64	.01	.1000
64	.1	.1000
32	.001	.7403
32	.01	.1000
32	.1	.1000

SOFTMAX

Batch	Learning Rate	Accuracy
128	.001	.4464
128	.01	.4039
128	.1	.1000
64	.001	.5984
64	.01	.3892
64	.1	.1000
32	.001	.6661
32	.01	.1993
32	.1	.1000

5 Data Analysis

RELU and Leaky RELU follow the trend that batch size 128 in tandem with .001 learning rate yields the highest accuracy value. RELU scored a .6634 accuracy and Leaky RELU scored a .8440 accuracy. This trend does not follow for SOFTMAX, where instead batch size 32 and .001 learning rate yields the highest accuracy value at .6661. Interestingly however, SOFTMAX has the highest average accuracy of .3337. Leaky RELU has an average accuracy of .3327 and RELU has an average accuracy of .2218. Despite having the highest achieved accuracy, Leaky RELU as well as RELU have such low average accuracy. This is because at a larger learning rate, .01 and .1, the model resulted in a .1 accuracy. With only 10 classes, the model essentially classifies at random at larger learning rates when using the Leaky RELU and RELU functions. SOFTMAX however was able to make use of .001 and .01 learning rates. This caused the results for SOFTMAX to have less instances where the accuracy was .1, and allowed for a higher average accuracy than either Leaky RELU or RELU.

6 Conclusion

In the case of CIFAR-10, Leaky RELU proved more accurate when compared to Softmax and standard RELU. In all tests utilizing a learning rate of .001, Leaky RELU significantly outperforms.

Even with the lowest batch size, Leaky RELU outperforms even the best runs of both other activation functions making it an excellent tool for image classification. One point of data which we did not account for was timing of the runs. If we were to re-run our model, we would record the timing to allow further investigation of the advantages provided by our modifications.

7 Delineation of Work

Michael was responsible for creating a bash script that automatically called the python program with varying parameters and then outputted the results to a spreadsheet for readability. As well as modifying the program to take command line arguments so various parameters could be tested. Dalton was responsible for running the experiments, some debugging to get the experiments running, and writing the model description section of the paper. Griffin was responsible for writing the introduction, data analysis, as well as manually creating a spreadsheet for SOFTMAX. Joe was responsible for the Methodology section, the Data section, the Conclusion, and putting Leaky RELU data into a spreadsheet.