

This report will be broken down into the following sections:

1. Setup
2. MNIST assignment
3. CIFAR10 assignment
4. References

Setup

For MNIST, Python PyCharm was used to run the code. This was done in Python 3.x with Anaconda and Tensorflow 1.5.1. MNIST can be run from inside PyCharm using a CPU without issue.

“mnist_with_summaries.py” is the original tutorial code which can be run to obtain the baseline accuracy for the assignment. “matthewbitterHW9.py” is the modified MNIST script which reports an improved accuracy. “cnn_mnist.py” is the other original tutorial code and does not need to be run, it is included simply as reference to compare to “matthewbitterHW9.py”. convert.py simply downloads the dataset and changes it to the required file format. A discussion of the results will come in a later section.

For CIFAR10, Google Colab and PyCharm were used to run the code. Python 3.x with a Tesla GPU were used. This code cannot be run in a reasonable amount of time using a CPU. To run the code, you will need to create an account with Google Colab and execute the script “GoogleColab_hw9part2.ipynb” which sets up the Google Colab environment (a copy of the Jupyter Notebook in .py format is also included for easy viewing in a text editor). The Google Colab script will run “cifar10_train.py” which runs both eval and train. A detailed discussion of results will be done in a later section.

When running the Google Colab script, os level file paths will need to be changed to reflect your own person google drive file structure.

Lastly, all python files attached to this report include at the top a detailed list of changes made including specific line numbers of the changes.

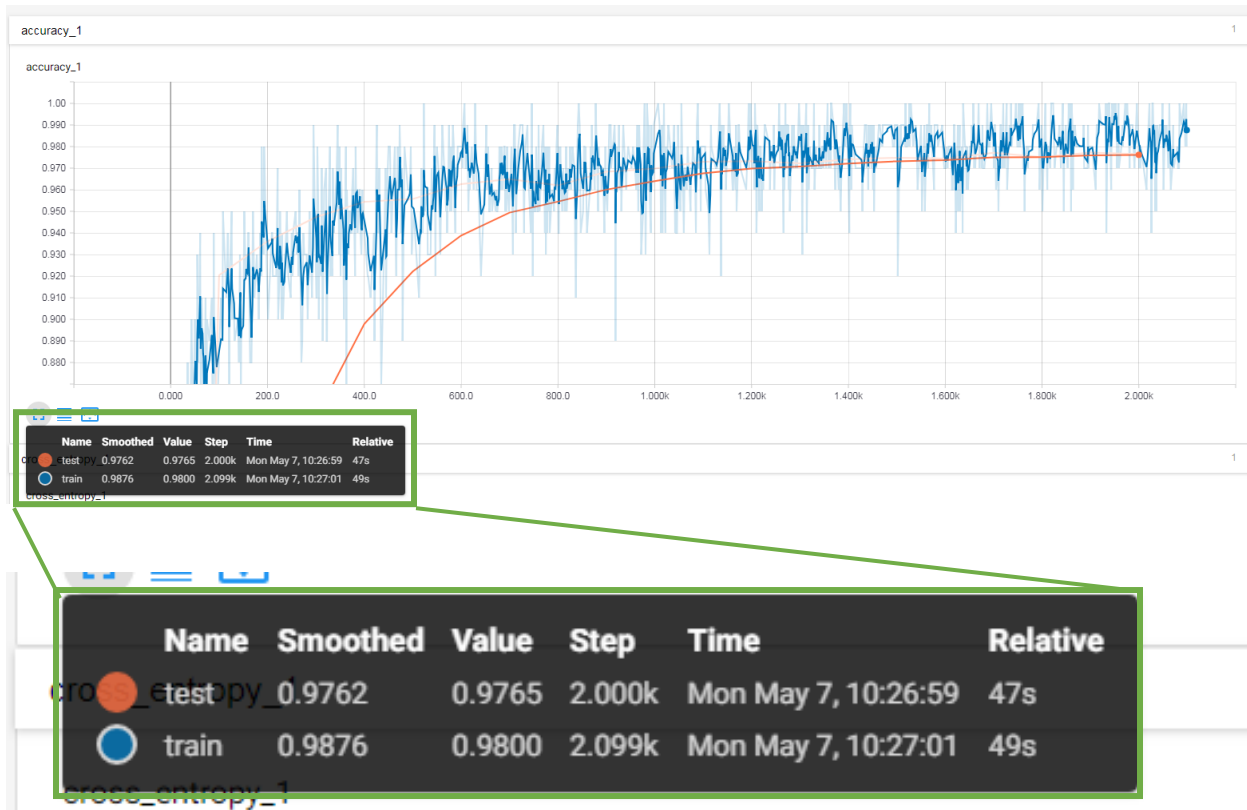
The following files are included:

- MNIST
 - matthewbitterHW9.py – CHANGED, details in code comments
 - mnist_with_summaries.py – CHANGED, details in code comments
 - convert.py – No change, from github which is listed in my references
 - cnn_mnist.py – No change, from original tutorial
- CIFAR10
 - cifar10.py - CHANGED, details in code comments
 - cifar10_eval.py - CHANGED, details in code comments
 - cifar10_input.py – No change, from original tutorial
 - cifar10_train.py - CHANGED, details in code comments
 - GoogleColab_hw9part2.ipynb - CHANGED, details in code comments
 - GoogleColab_hw9part2.py - CHANGED, details in code comments

MNIST Part A - Original

Tensorflow 1.5.1 was used for the MNIST tutorial. The original code from the “mnist_with_summaries.py” was used to generate the line chart below with slight modifications to the logging frequency and number of steps till stop. Specific line numbers of changes are in the python files.

The original MNIST “mnist_with_summaries.py” code achieved an accuracy of 98% of train and 97.65% on test at 2100 and 2000 steps respectively. The architecture of the original MNIST code “mnist_with_summaries.py” was a simple 1 hidden layer fully connected network using the AdamOptimizer.



Conversely, the “cnn_mnist.py” code from the tutorial used a completely different network/graph architecture as well as tensorflow objects. Specifically, it used estimators and tf.layers which makes it significantly easier to create deeper layers of pooling and convolutions because tf.layers offers higher level functions to define pooling and convolutions with a single call. Compared to the “mnist_with_summaries.py” tutorial that uses tf.nn and matrix multiplication which requires the user to create pooling and convolutions from scratch.

Name	Smoothed	Value	Step	Time	Relative
eval/accuracy	0.8664	0.8664	2.100k	Mon May 7, 17:12:23	0s

The accuracy above (86.64% and 2100 steps) is from the “cnn_mnist.py” file. The original tutorial only reported accuracy at the end of the run. The architecture and the logging frequency was improved upon in MNIST Part B.

MNIST Part B – Improvement

The improved code for MNIST is in “matthewbitterHW9.py”. I chose to modify the “cnn_mnist.py” file rather than the “mnist_with_summaries.py” because the former uses tf.layers which was significantly easier to modify and improve the accuracy on v.s. the lower level tf.nn functions and matrix multiplication of weights and biases directly.

I will be highlighting the differences that relate to the CNN architecture specifically between my improved file and the cnn_mnist tutorial in the section below.



The most significant improvement to the MNIST accuracy was changing the optimizer to the Adam Method rather than the original gradient decent that was used in the cnn_mnist tutorial. This is because the Adam method uses a feature called momentum to speed up the training of the networks parameters by increasing the step size dynamically when going down a steep gradient. The downside of this is it requires more computation v.s. simple gradient decent which was evident in the training duration of the “cnn_mnist.py” (5min 16sec) that uses simple gradient decent and my improved script that uses the Adam optimizer (9min 49sec).

Additional improvements were also made to the model. Specifically, a 3rd convolution was added to the model, between the last two pools, which **improved the accuracy by 1.5% versus prior runs of the model.**

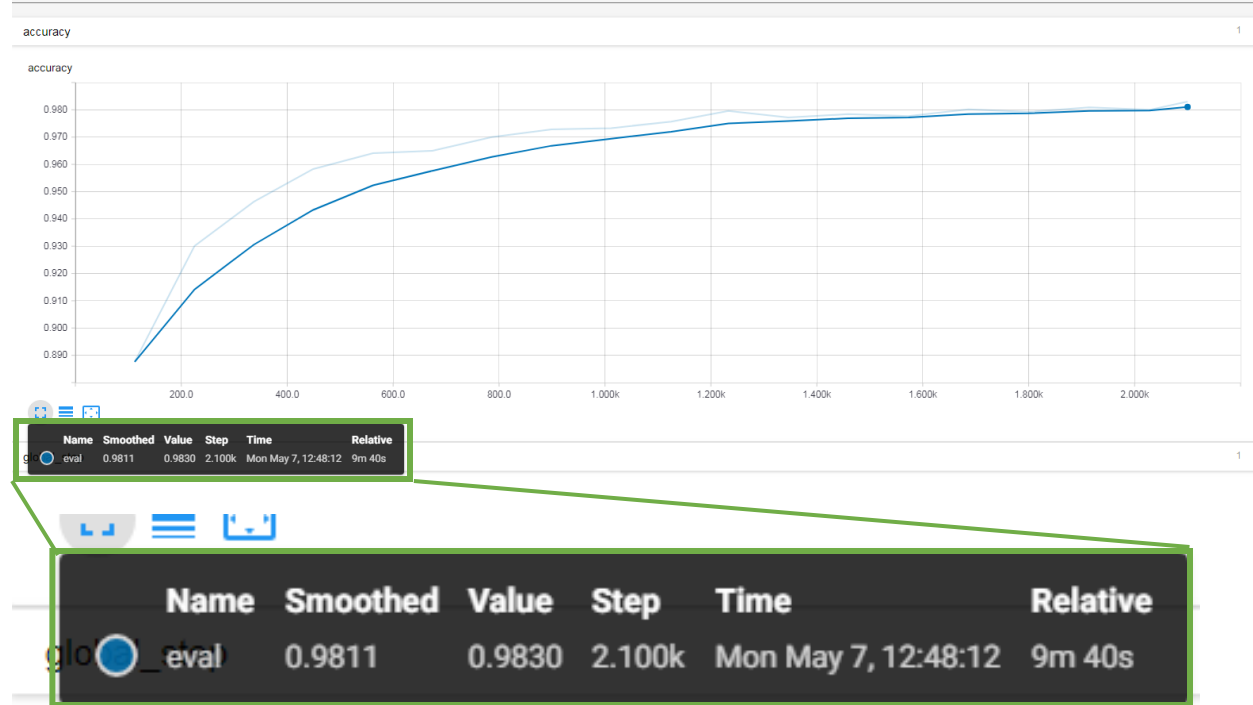
Technical changes were also made to the architecture of the model to support evaluations being run every 100 steps or less. Specifically, “tf.estimator.train_and_evaluate” was used to run training and evaluation in an alternating pattern as shown in the line charts below using the parameter “throttle_secs=25” and logging checkpoints every 100 steps.

The test/evaluation accuracy of the **improved model is 98.11% and test training accuracy is 99%.** This is an **improvement** over both the cnn_mnist script and the mnist_with_summaries script.

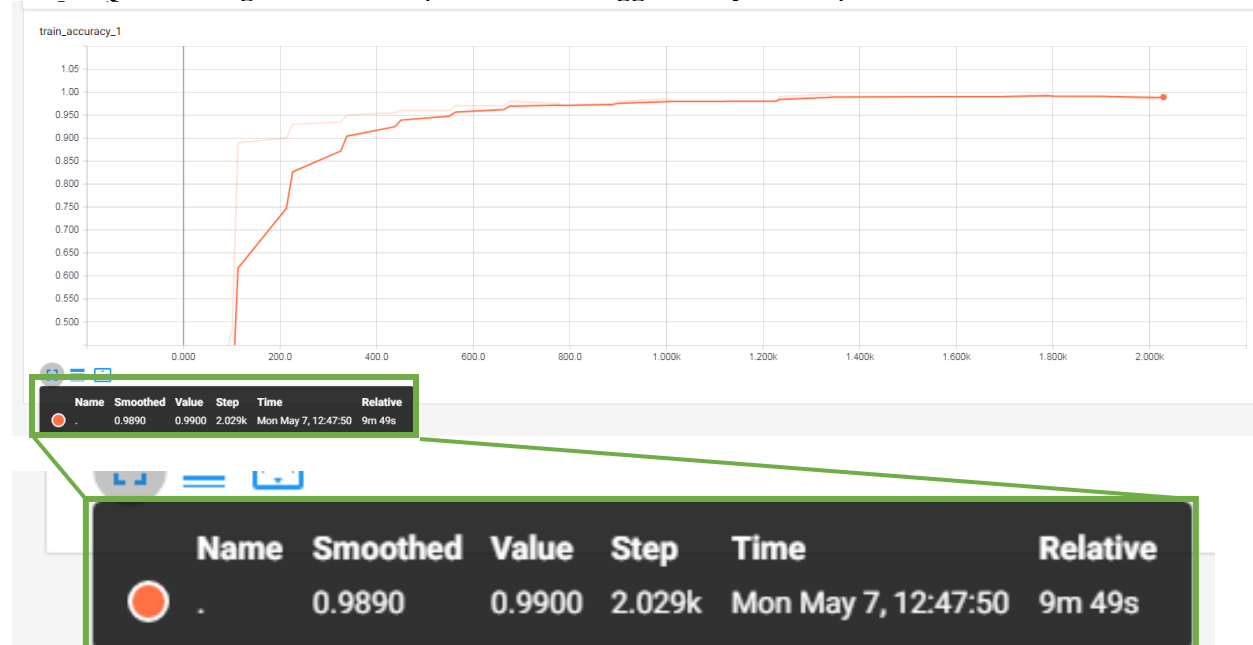
Loss of cnn_mnist.py highlighting the 5min 16sec run time using simple gradient decent.

	Name	Smoothed	Value	Step	Time	Relative
	.	0.9259	0.8674	2.001k	Mon May 7, 17:12:02	5m 16s
	eval	0.5576	0.5576	2.100k	Mon May 7, 17:12:23	0s

Accuracy of Test/Evaluation of MNIST Improved model logged every 100 steps



Accuracy of training of MNIST Improved model logged every 100 steps



CIFAR10 Part A - Original

For the CIFAR10 assignment I used Google Colab for GPU processing because it would take 20hours+ using my home CPU. The GPU cut down the time to less than 30minutes per run.

The performance of the original tutorial script for 2300 steps was around 71.38% as seen in the orange highlighted box in the picture from Part B. The combined line chat in section B shows the test/evaluation accuracy every 100 steps with many different model architectures. I modified the `cifar10_eval` and `cifar10_train` scripts to support running the evaluation every 100 steps.

As I mentioned in the code comments I did not have available to me `save_checkpoint_steps` function due to being on TF 1.4 however I designed an effective solution using timings instead and ensured that the data was evaluated at least every 100 steps by changing the timings of the evaluations. Specifically, I called `cifar10_eval.evaluate()` inside the `cifar10_train.train()` function to accomplish this and changed the `cifar10_eval.evaluate()` to run only once per call.

CIFAR10 Part B – Improvement

To improve the model for CIFAR10 I tried many different approaches, of which only a few showed better accuracy results compared to the original tutorial code. I will summarize these findings inside a table and discuss the best model in more detail. Additionally, the code submitted is only for the best model and the original tutorial code modified for 100 step evaluations. The Model Name heading references the ledger of line charts below.

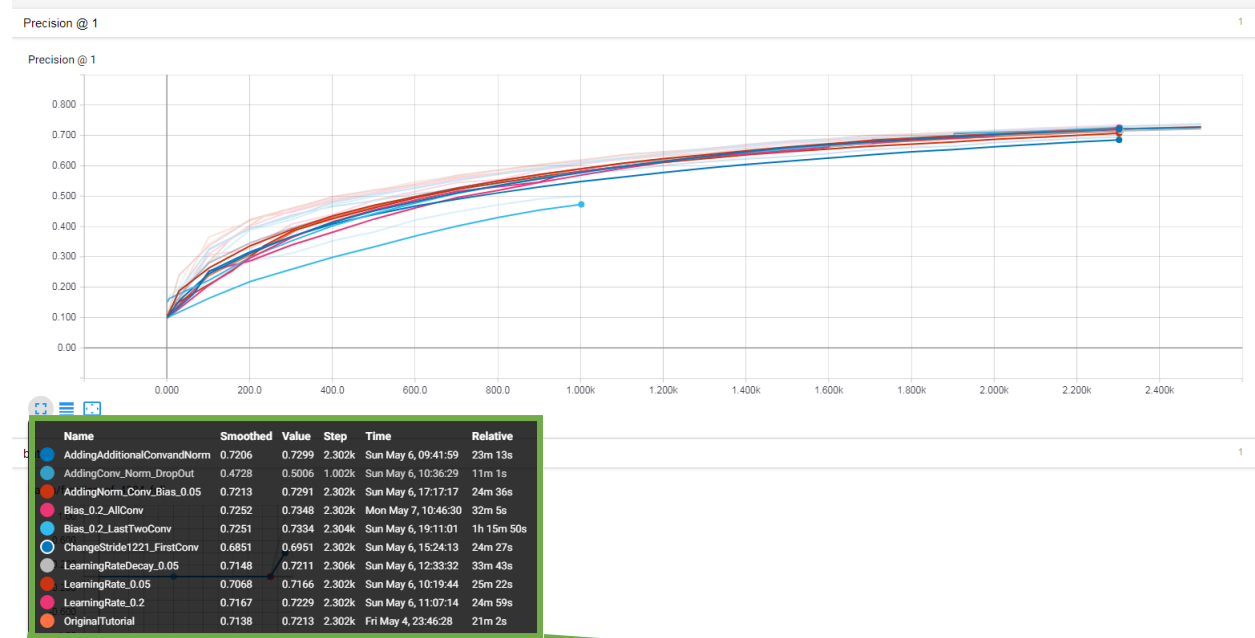
Model Name	Brief description of changes versus original tutorial code	Performance Discussion
AddingAdditionalConvandNorm	Adding an additional convolution and normalization to the graph	This showed slightly better performance than baseline however, within the margin of error
AddingConv_Norm_DropOut	Same as above however I also included dropout logic of 40% to account for overfitting	This performed extremely poorly probably due to the drop rate being too high. A lower dropout percentage can be tested in the future.
AddingNorm_Conv_Bias_0.05	Added a conv and norm layer and lowered the initial bias from 0.1 to 0.05 for all convolutions.	This showed slightly better performance than baseline however, within the margin of error
Bias_0.2_AllConv	Added a conv and norm layer and increased the initial bias from 0.1 to 0.2 for all convolutions and dense layers.	This showed the best performance increase of all the models. Detailed discussion below.
Bias_0.2_LastTwoConv	Added a conv and norm layer and increased the initial bias from 0.1 to 0.2 for last two convolutions only.	Basically, the same as the model above, so increasing the initial bias showed good results regardless of which layer it is implemented in.

ChangeStride1221_FirstConv	Changed the window of first conv layer from [1,1,1,1] to [1,2,2,1]	Extremely poor performance due to information loss from summarizing a larger window into a single pixel.
LearningRateDecay_0.05	Reduced the learning rate decay from 0.1 to 0.05	Showed same performance as original code however not as high as the best model due to learning rate being too high in later steps which causes each step to be too large.
LearningRate_0.05	Reduced the learning rate from .1 to 0.05	Poor performance due to the model taking too small of a step and not learning quickly enough.
LearningRate_0.2	Increased learning rate from .1 to .2	Showed same performance as original code.
OriginalTutorial	Original tutorial code unchanged except for logging and step number	N/A

The model with the title Bias_0.2_AllConv showed the best results with an accuracy increase of **1.35%** over the original tutorial code keeping the number of steps constant between comparisons, namely **2300 steps**.

Detailed explanation of the changes to the best improved CIFAR10 model are below:

- The initial learning rate was increased from 0.1 to 0.2. I do not believe this had a significant impact on improving the accuracy in later steps, however, it did offer a boost to accuracy in the earlier steps of the model. Typically, the learning rate is changed in orders of magnitude so that is another reason why I believe this did not have a significant contribution to accuracy.
- Increased the initial bias of all convolutional layers and the dense fully connected layers from 0.1 to 0.2. I believe this was one of the main contributors of the improved accuracy. Specifically, it allowed the activation function to shift entirely in the direction of the initial bias constant which was the correct direction based on the dataset and so achieved a higher accuracy with the same number of steps compared to the original tutorial. In a deeper network or a network that was trained for more steps I believe this improvement would be negligible however for this specific case it increased the accuracy.
- Lastly, I added an additional convolutional layer and normalizing layer to the graph to improve the flexibility of model. I believe this was the sweet spot in balancing overfitting and bias as adding additional convolutional layers showed a decrease in test accuracy.

Evaluation/Test Accuracy of all the models including the original tutorial code for CIFAR10.

Name	Smoothed	Value	Step	Time	Relative
AddingAdditionalConvandNorm	0.7206	0.7299	2.302k	Sun May 6, 09:41:59	23m 13s
AddingConv_Norm_DropOut	0.4728	0.5006	1.002k	Sun May 6, 10:36:29	11m 1s
AddingNorm_Conv_Bias_0.05	0.7213	0.7291	2.302k	Sun May 6, 17:17:17	24m 36s
Bias_0.2_AllConv	0.7252	0.7348	2.302k	Mon May 7, 10:46:30	32m 5s
Bias_0.2_LastTwoConv	0.7251	0.7334	2.304k	Sun May 6, 19:11:01	1h 15m 50s
ChangeStride1221_FirstConv	0.6851	0.6951	2.302k	Sun May 6, 15:24:13	24m 27s
LearningRateDecay_0.05	0.7148	0.7211	2.306k	Sun May 6, 12:33:32	33m 43s
LearningRate_0.05	0.7068	0.7166	2.302k	Sun May 6, 10:19:44	25m 22s
LearningRate_0.2	0.7167	0.7229	2.302k	Sun May 6, 11:07:14	24m 59s
OriginalTutorial	0.7138	0.7213	2.302k	Fri May 4, 23:46:28	21m 2s

References

Explanation on how to use tf.layers

<https://stackoverflow.com/questions/49201832/how-to-use-tensorboard-and-summary-operations-with-the-tf-layers-module>

Basic explanation of tensorflow architecture

<http://www.jbencina.com/blog/2017/05/31/getting-started-tensorflow/>

Additional examples of tensorflow architecture

<https://github.com/wagonhelm/Visualizing-Convnets/blob/master/visualizingConvnets.ipynb>

How to use logging hooks

<https://stackoverflow.com/questions/45353389/printing-extra-training-metrics-with-tensorflow-estimator>

How to use tensor board with TF estimator

<https://stackoverflow.com/questions/43782767/how-can-i-use-tensorboard-with-tf-estimator-estimator>

Foundation of “matthewbitterHW9.py” file. Explained how to report accuracy using estimator and tf.layers.

<https://github.com/tensorflow/models/blob/5ddd7e55c9ab6ae10a1459afc6309a2a417398d9/official/mnist/mnist.py#L211>

Original MNIST tensorflow tutorial

<https://github.com/tensorflow/models/blob/master/official/mnist/mnist.py>

How to use Google Colab

<https://medium.com/deep-learning-turkey/google-colab-free-gpu-tutorial-e113627b9f5d>

Google Colab with CIFAR10 set up

<https://medium.com/deep-learning-turkey/deep-learning-lab-episode-2-cifar-10-631aea84f11e>

No module found using tensorboard with TF 1.7

<https://stackoverflow.com/questions/46579682/tensorboard-modulenotfound-no-module-named-tensorflow-tensorboard>

Differences between GD and AdamGD

<https://stats.stackexchange.com/questions/184448/difference-between-gradientdescentoptimizer-and-adamoptimizer-tensorflow>

MNIST Tutorial

<https://www.tensorflow.org/tutorials/layers>

Tensorboard Tutorial

https://www.tensorflow.org/programmers_guide/summaries_and_tensorboard

CIFAR10 Tutorial

https://www.tensorflow.org/tutorials/deep_cnn