**Purpose**

The purpose of this project was to design a convolutional neural network using Python’s TensorFlow libraries to train the model to recognize 32x32 pixel images from the classical cifar10 dataset. The requirements were to train and test the model in strictly less than 8 minutes on the respective development machine with the performance criteria that it should be able to accurately predict images with an accuracy greater than 64%.

**Approach**

PyCharm IDE was used for development. Initially, several different CNN models were run varying the batch size and number of epochs as well as the CNN layers. However, the training was extremely slow lasting over 20 minutes to generate a model with an accuracy of 60% or less. Upon researching, it was soon discovered that the TensorFlow module comes in two different flavors, tensorflow (i.e. tensorflow-cpu) and tensorflow-gpu. The latter version allows the machine to leverage the gpu for processing, which is significantly faster, if the machine has the hardware to support it.

Therefore, the development machine selected for this project was a MSI brand laptop. This allowed mobility as development, training, and testing can be performed on the same machine as the one used for demoing the project. The development machine also had the specifications necessary for running tensorflow-gpu. The specifications for this machine are as follows:

***Development Machine Specifications***

**Processor:** Intel 6 Gen Core i7-6700HQ

**Graphics:** Nvidia GeForce GTX 1060

**Video Mem:** 6GB GDDR5

**System Mem:** 16GB DDR4

However, simply having the hardware specs were not enough. Several added steps were required to get the machine ready for utilizing the gpu in tensorflow. These steps included:

1. Installing/updating Nvidia GPU Drivers (Requires driver version 410.x or higher)
2. Installing CUDA Toolkit shipping CUPTI (Requires CUDA 10.0)
3. Downloading and adding cuDNN SDK (vers. 7.4.1 or better) to the CUDA Toolkit

Once this was done, tensorflow-gpu could be pip installed and imported into the project without issue. Prior to doing these steps, however, running the program would throw errors. The difference in performance was significant, with CNN models that took over 20 minutes to train and test taking less than 2 minutes after the change.

Finally, once the project was running smoothly, the remainder of the project was adjusting the parameters and CNN layers to get the best accuracy. First, this involved adjusting the number of epochs, as this counts the number of times the entire dataset is passed through the model. However, this became an issue due to overfitting as accuracy dropped at higher and higher epoch. To adjust for overfitting, dropout was used. Another method is weight regularization. Once dropout was added to every one or two layers, the number of epoch could be increased without bound to improve accuracy with diminishing return. Increasing batch size was another parameter adjusted which significantly improved training time with a small decrease to accuracy. Finally, adding more convolution layers and adjusting the various parameters within each layer was the final adjustment.

To keep track of improvements, all runs were logged to a log file named results.txt. These can be found locally along with the Python source file: cifar10cnn.py.

**Final Results**

Using the final model with 40 epochs and a batch size of 500, the training and testing on my development machine **lasted** **7.28 minutes** with a **test accuracy of** **85.45%**. The training accuracy was above 91% on the last epoch. The final CNN and details can be viewed in the Appendix. This CNN was inspired by several different sources, particularly an article written by Abhijeet Kumar referenced in sources.

**Sources**

**Kumar, Abhijeet**. “Achieving 90% Accuracy in Object Recognition Task on CIFAR-10 Dataset with Keras: Convolutional Neural Networks.” *Machine Learning in Action*, WordPress.com, 5 Dec. 2018, appliedmachinelearning.blog/2018/03/24/achieving-90-accuracy-in-object-recognition-task-on-cifar-10-dataset-with-keras-convolutional-neural-networks/.

**Appendix**

Best Run Summary:

Test accuracy: 0.8544999957084656, Test Loss:0.49447950910925864

Epoch: 40, Batch size:500

Total time: 7.28 mins

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d (Conv2D) (None, 32, 32, 32) 896

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

batch\_normalization\_v1 (Batc (None, 32, 32, 32) 128

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_1 (Conv2D) (None, 32, 32, 32) 9248

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

batch\_normalization\_v1\_1 (Ba (None, 32, 32, 32) 128

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d (MaxPooling2D) (None, 16, 16, 32) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout (Dropout) (None, 16, 16, 32) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_2 (Conv2D) (None, 16, 16, 64) 18496

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

batch\_normalization\_v1\_2 (Ba (None, 16, 16, 64) 256

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_3 (Conv2D) (None, 16, 16, 64) 36928

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

batch\_normalization\_v1\_3 (Ba (None, 16, 16, 64) 256

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_1 (MaxPooling2 (None, 8, 8, 64) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_1 (Dropout) (None, 8, 8, 64) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_4 (Conv2D) (None, 8, 8, 128) 73856

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

batch\_normalization\_v1\_4 (Ba (None, 8, 8, 128) 512

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_5 (Conv2D) (None, 8, 8, 128) 147584

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

batch\_normalization\_v1\_5 (Ba (None, 8, 8, 128) 512

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_2 (MaxPooling2 (None, 4, 4, 128) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_2 (Dropout) (None, 4, 4, 128) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten (Flatten) (None, 2048) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense (Dense) (None, 512) 1049088

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_3 (Dropout) (None, 512) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_1 (Dense) (None, 10) 5130

=================================================================

Total params: 1,343,018

Trainable params: 1,342,122

Non-trainable params: 896

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_