CPRE 487/587 Lab 1: DNN Intro (Tensorflow)

Introduction: In this lab, we will be exploring the very basics of the several different libraries we will be using to help complement our usage of the Tensorflow Python library in CPRE 487/587. We will also begin to learn some of the basic operations of the framework to better understand the foundation of deep neural networks for image processing inside of machine learning.

Module 3.3 - Working With the Data-set:

One of the most important reasons for having a larger amount of data points to use (in this case, for training and validation) is to make sure that there is a larger amount of data to use as a check for accuracy and precision when it comes to making predictions. Though a smaller data set may allow for small peaks in performance and storage size, a larger data set can ensure that the statistics of the check for a model are more accurately trained.

Our data set is not considerably small. We have 10,000 images in our validation set and 100,000 images in our training set. The large size lets us know how much accuracy needs to be put into each step of the process.

Module 3.4 – Model Exploration

As the image becomes processed down the pipeline of the feature maps, we can see that the input starts to become a smaller grid of cells, which follows the principles of the decreasing mapping size relationship from lectures (output = input – feature_map + unit_step...).

As we can see, the deeper the feature maps in the pipeline, we can see that overall the number of parameters in a given channel (and even sometimes the number of channels) tends to decrease and condense the amount of data in given channels. For example, we begin with a general (?, 60, 60, 32) shape for the first conv2d layer, and then we reduce to (? 8, 8, 128). This means that we convert from a 60x60 pixel image down to an 8x8 pixel image, thus proving the order of the feature map correlates to the number of pixels it is handling. A “higher number” layer might also been seen deeper in the layer pipeline, for example, conv2d vs conv2d_5.

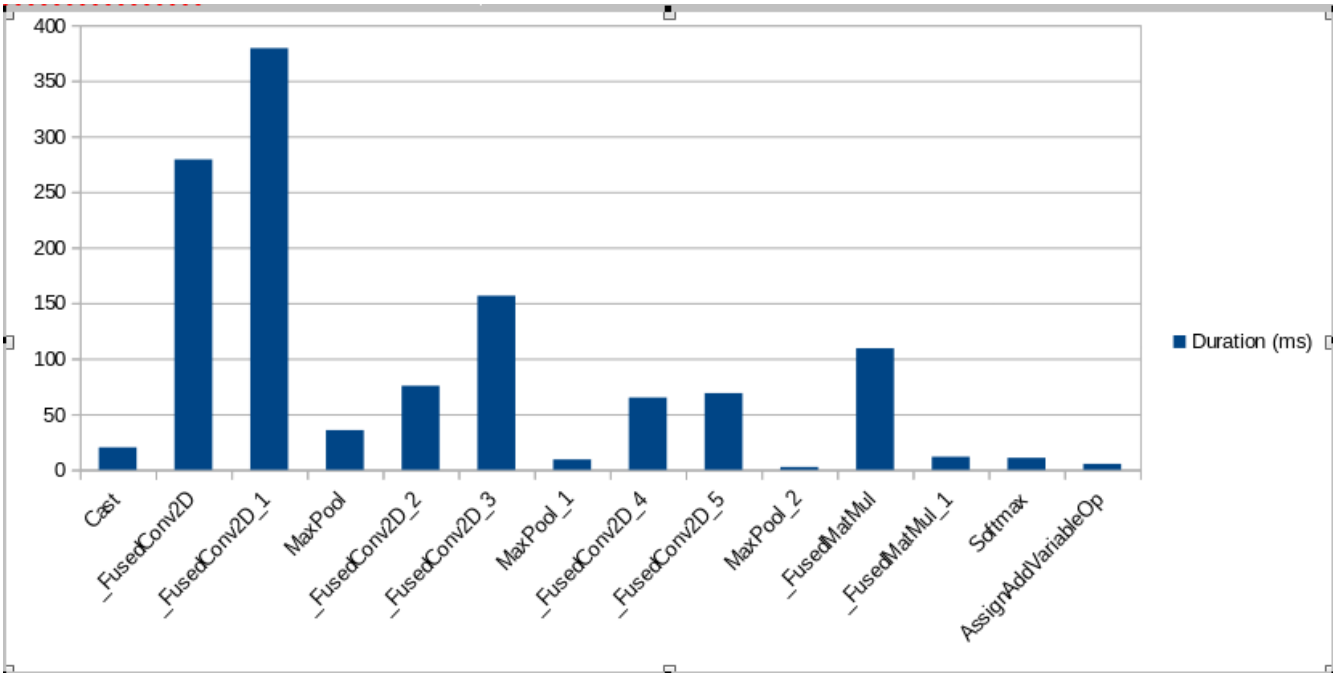
Module 3.5 – Model Analytics/Metrics with Tensorboard

(Part 1 – Single Online Inference Profiling)

When learning how to do this portion of the lab, we started by understanding how to capture the end-to-end latency of a single example image:

The end-to-end latency for the execution for one example batch image is approximately 1.151ms long, which includes the time the process takes to work through all the layers of the inference. So far, we can see that the second FusedConv2D (Conv2D1) layer is taking the longest in the execution pipeline. We

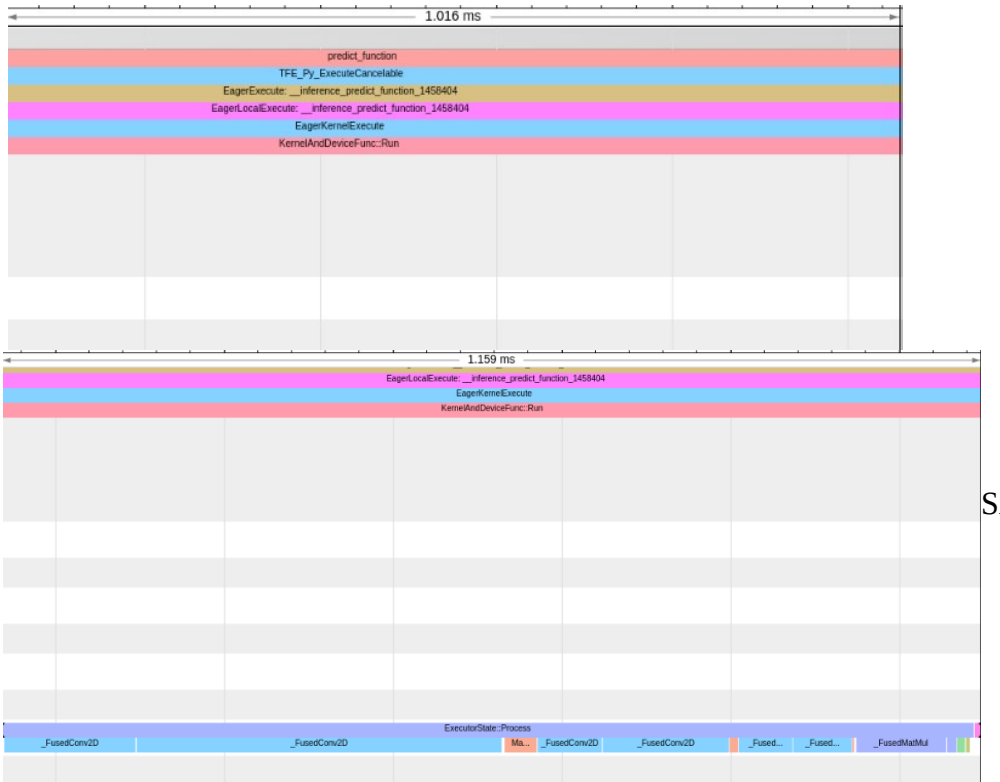
believe that this step in the pipeline is taking the longest because it has the largest depth for it's input compared to the other layers in the pipeline.



Above is a graph we took comparing the process layer name and it's respective timing to prove our recent conclusion.

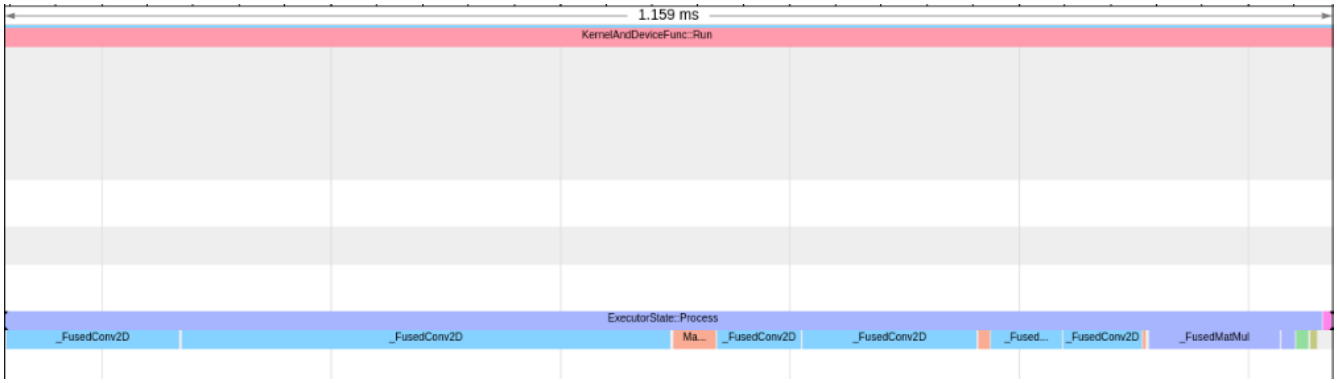
Now, we can conduct the same experiment but now for our three sample images that we took earlier in this lab:

SAMPLE IMAGE 1:



SAMPLE IMAGE 2:

SAMPLE IMAGE 3:



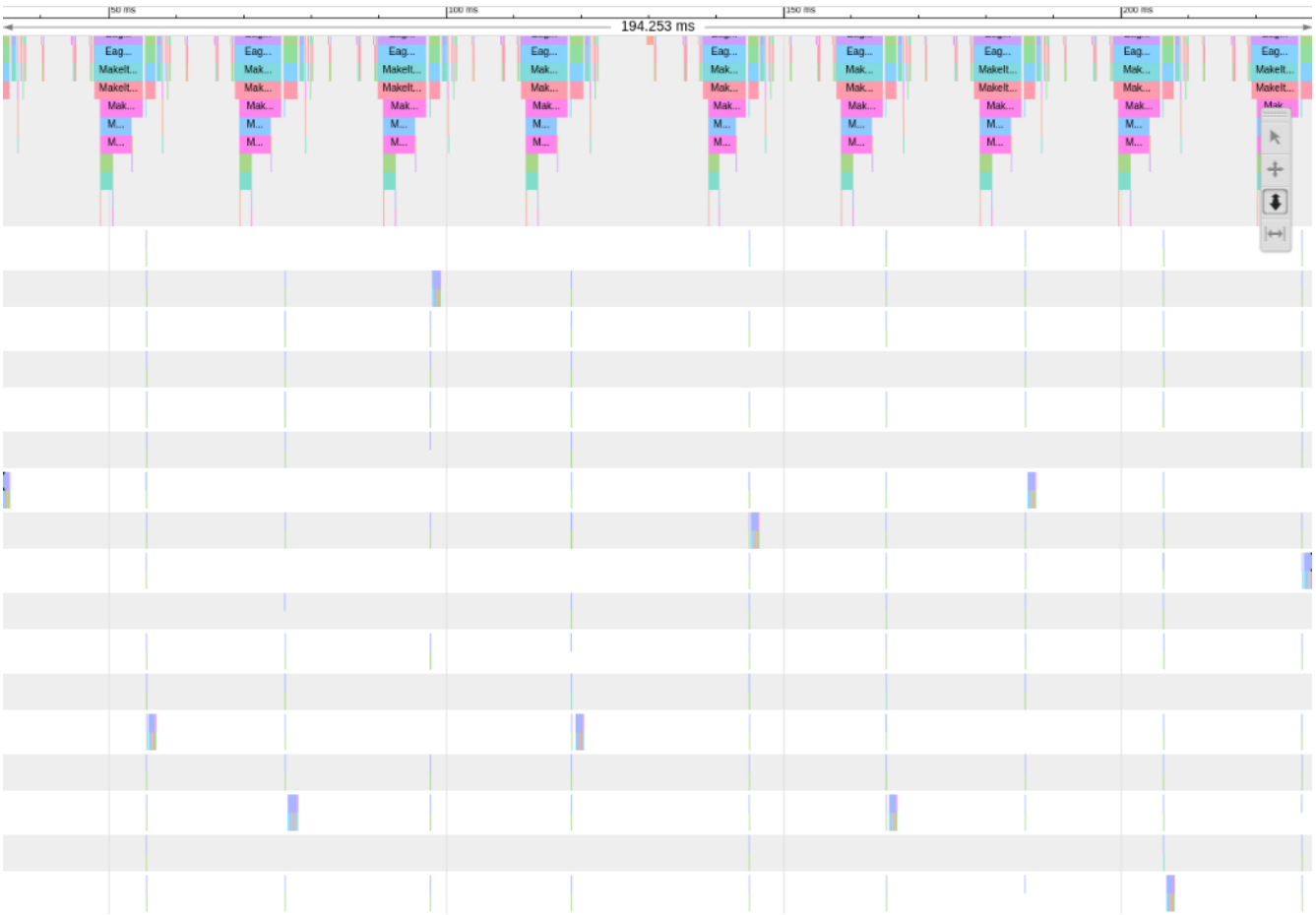
In total between our *three sample images*, we have a total operation runtime of 3.334ms. We acquire this calculation by adding the sum of the three total images (1.016ms + 1.159ms + 1.159ms = 3.334ms)

Our assumption still holds that the kernel taking the longest (or otherwise most critical) that the second CONV2D layer is taking the longest time out of the execution. Optimizing this portion could see the potential largest efficiency difference.

(Part 2 – Multiple Online Inference Profiling)

In this next portion, we are asked to perform online inference for separate sets of 10, 100, and 100 images.

SET OF 10 IMAGES:



TOTAL TIME: 194.253ms or about 0.194s.

SET OF 100 IMAGES:

TOTAL TIME: 2.136s

SET OF 1000 IMAGES:

TOTAL TIME: 18.595s

For this section, it was hard to calculate an exact portion of time it took to perform the total among every different set of images.

(Part 3 – Batch Inference Profiling)

For this, we now need to use batch inference for 1000 images of the validation data set using batch sizes of 20, 40, 100, and 200 images.

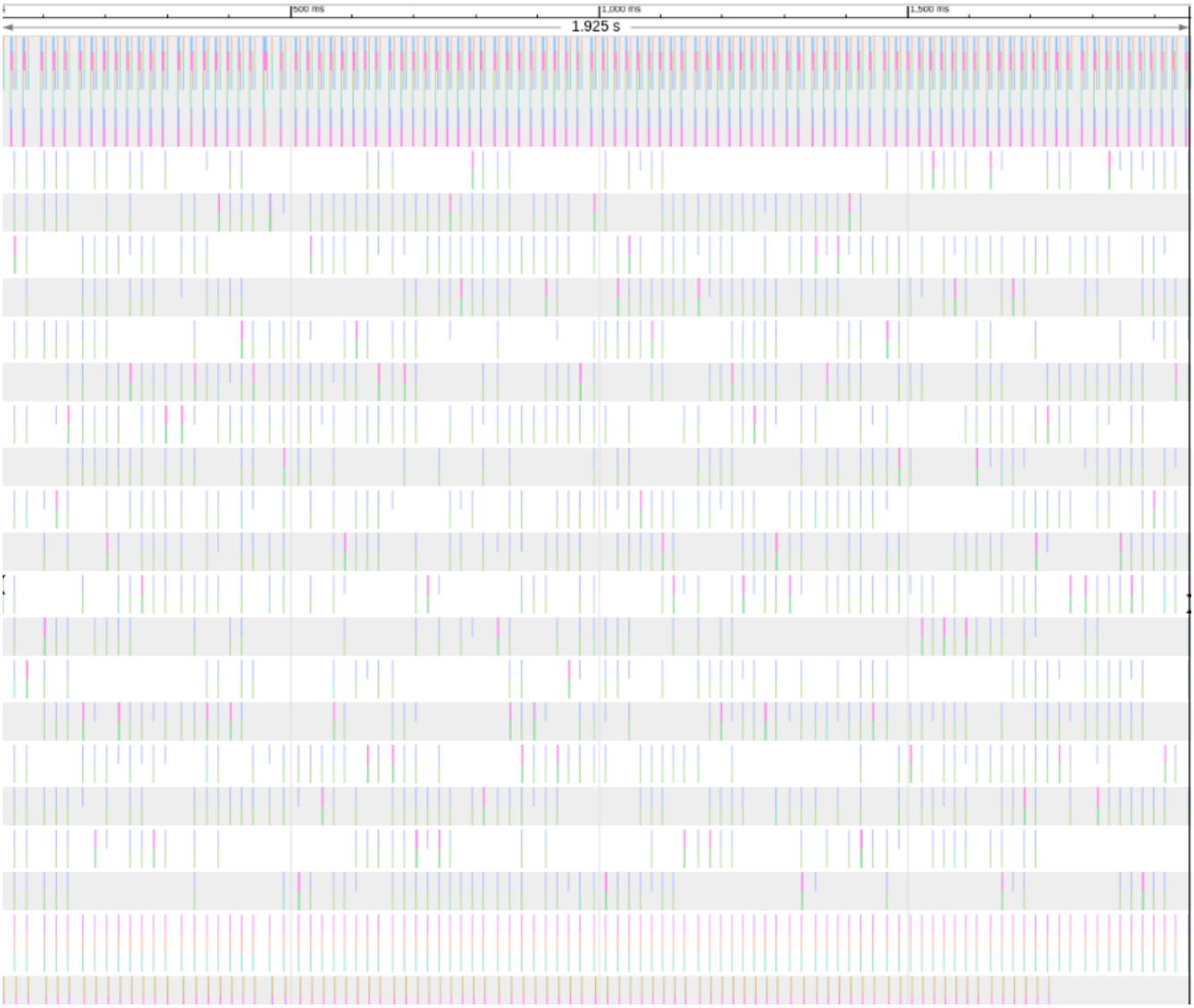
BATCH SIZE 20:

TOTAL TIME: 977.843ms

BATCH SIZE 40:

TOTAL TIME: 4.869s

BATCH SIZE 100:



TOTAL TIME: 1.925s

BATCH SIZE 200:

TOTAL TIME: 9.149s

From these observations, we are able to see that there is a correlation between choosing an associate batch size along with an arbitrary number to select from our validation set.

The kernels most critical for optimization still seem to be expensive in the conv2d layers of the execution pipeline. This is because the convolution layers are having to do most of the hard computational processes in order to transform, or in other words “convolve” the data to be processed for the next layer to come.

Module 3.7 – Training

We noticed that given a higher number of epochs (assuming our code doesn't early-stop), our accuracy tends to grow a bit larger due to more iterations of allowing our model to train. Therefor, we see an increase in the validation accuracy since the training accuracy for those higher epoch iterations tend to grow as well.

The minimum number of training epochs we found to help achieve the best possible validation set accuracy seemed to start leveling out around 3 epochs. We see slight gains by possibly allowing further than three epochs, but on average, most of our runs for training would terminate after 3, meaning that the code detects little to no gain from our learning rate.

When we use a large number of epochs, our validation accuracy tends to increase, but rarely makes it all the way through the large number of epochs due to our early stopping catch. Using a larger number of epochs generally allows the computer to keep working the model towards an improved training set by further calculating the weights and setting them to more accurate values to get optimal results.