Assignment 1 Part 2A

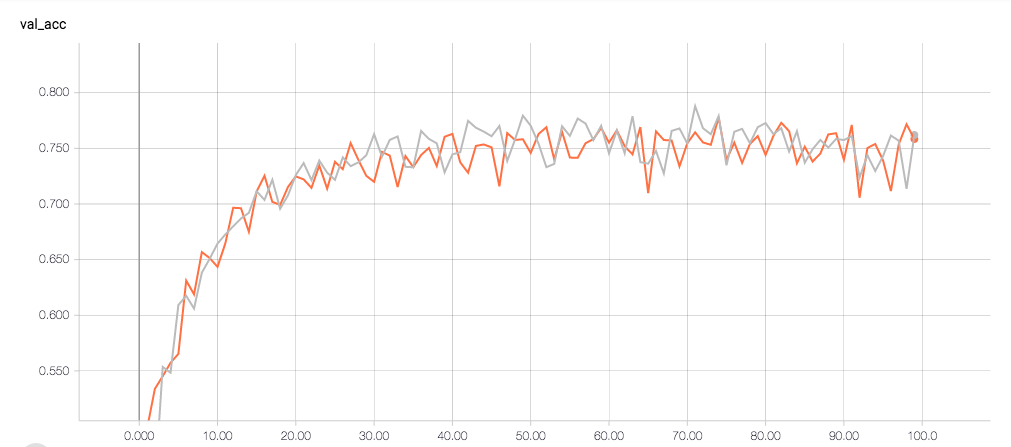
Team 8

INFO 7374 Spring 2019

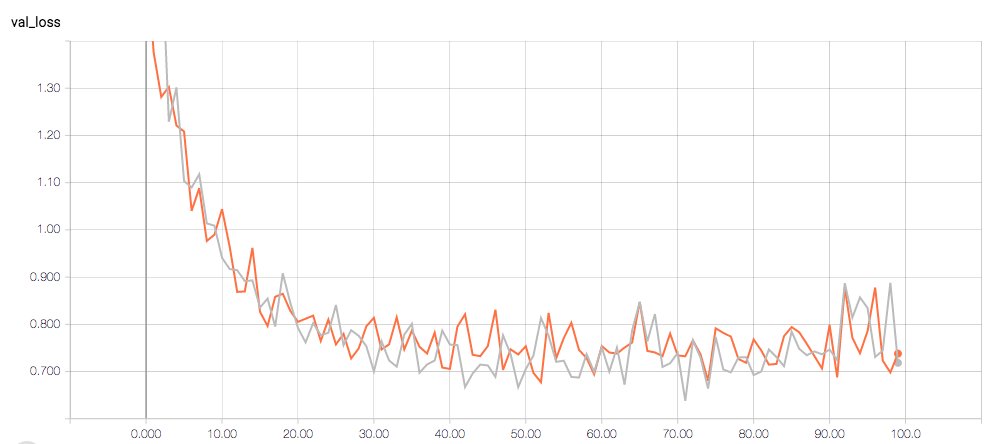
**CPU vs GPU**

In comparing training a model on a CPU versus a GPU, we trained on the same machine with the only difference being whether or not the GPU was used. We found that the GPU did not affect the accuracy and loss. Both final values were within 0.02 of each other and the validation accuracy and loss plots indicate this difference is noise. However the hardware change sped up the training time by a factor of 6.625, with an average training time of 1.66 ms per image on the CPU and 0.25 ms per image on the GPU with batch sizes of 32 images. These results are summarized in the table and figures below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Experimental Parameter** | **Accuracy** | **Loss** | **Total Runtime (min)** | **Training time per image (ms)** |
| CPU | 0.762 | 0.718 | 138.48 | 1.66 |
| GPU | 0.758 | 0.738 | 22.19 | 0.25 |



Validation Accuracy for CPU vs GPU. CPU: gray, GPU: orange

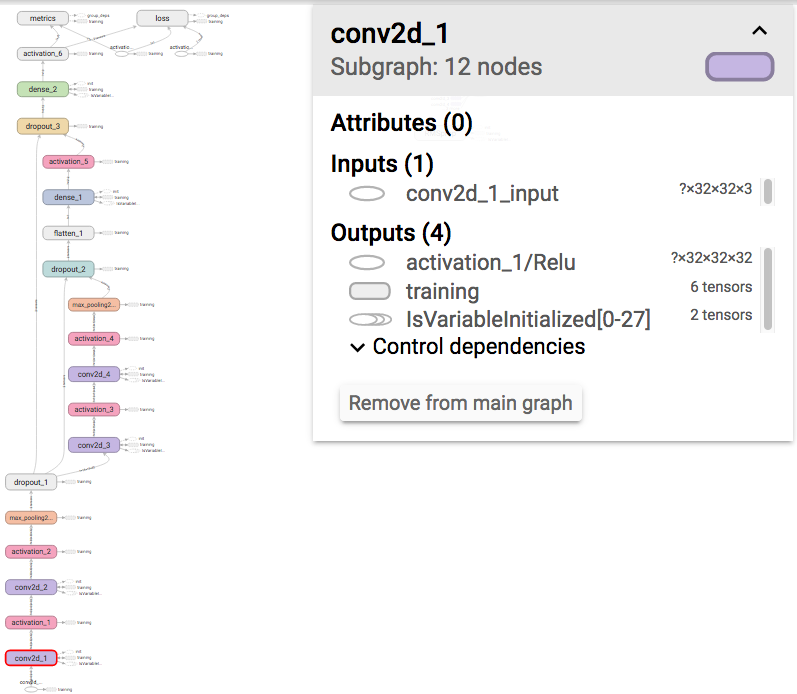


Validation Loss for CPU vs GPU. CPU: gray, GPU: orange

**Callbacks**

**TensorBoard**

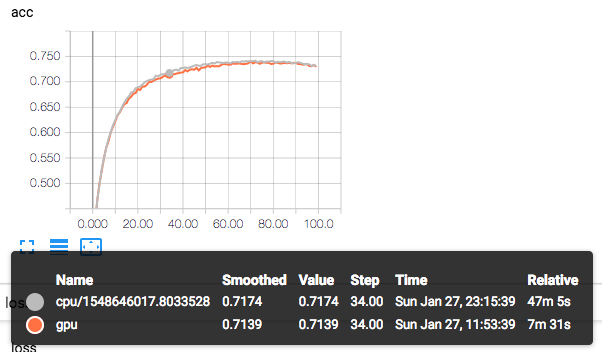
The TensorBoard Keras callback requires that you specify the directory that logs should be saved to. While the model is trained, it writes data about the training including accuracy and loss scores to the log file, which can then be viewed by running a TensorBoard server. Through the dashboard you can access interactive plots of training and validation accuracy and loss and the learning rate, view graphs of your models, and run a debugger, among other options. It makes it easy to overlay data from multiple experiments, and you can see the scores at individual timepoints.



Example interactive model graph

|  |  |
| --- | --- |
|  |  |
|  |  |

Example TensorBoard plots



Example interactive comparison

**TerminateOnNaN**

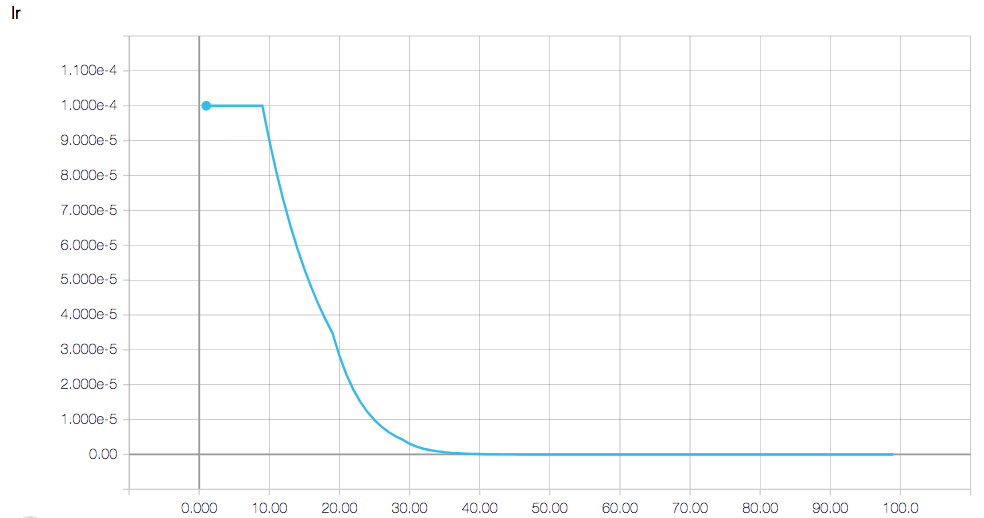
The terminate on NaN callback causes the model to stop training if a NaN loss is encountered rather than throwing an error. This can happen due to NaNs in the training data or if the learning rate is too high. The callback does not take any arguments. We were unable to recreate the conditions under which this occurs, as our model simply failed to train at all with NaNs in the data, as the tensorboard plots demonstrate.

|  |  |
| --- | --- |
|  |  |
|  |  |

Accuracy and Loss for Data with NaNs

**LearningRateScheduler**

The learning rate scheduler callback allows you to write a custom function to change the learning rate over time. Typically, it is used to decay the learning rate. The function must take two arguments, the epoch number and the current learning rate. The rate of decay can be constant or be a function of the number of epochs. We wrote a function where the rate of decay increases every 10 epochs, returning lr \* 0.9floor((1+epochs)/10). The learning rate scheduler takes two arguments, the schedule function and a flag for verbose mode. As the plots show, the effect of this function is for the model to effectively stop training as the learning rate approaches zero.



Learning Rates Set by the LearningRateScheduler

|  |  |
| --- | --- |
|  |  |

Train Accuracy and Validation Accuracy with LearningRateScheduler

**Custom Callbacks**

The API supports writing custom callbacks that inherit from keras.callbacks.Callback. The class can implement the same functions as the lambdas callback – on epoch begin/end, on batch begin/end and on train begin/end. We tested out two custom callbacks, the loss history recording from the documentation example (<https://keras.io/callbacks/>), and a learning rate history recording for keeping track of the learning rates generated by the learning rate scheduler. Both write values to a list that can be accessed as a property of the callback object.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

Accuracy and Loss Curves for Custom Callbacks.

Green: Loss History. Red: Learning Rate History with LearningRateScheduler