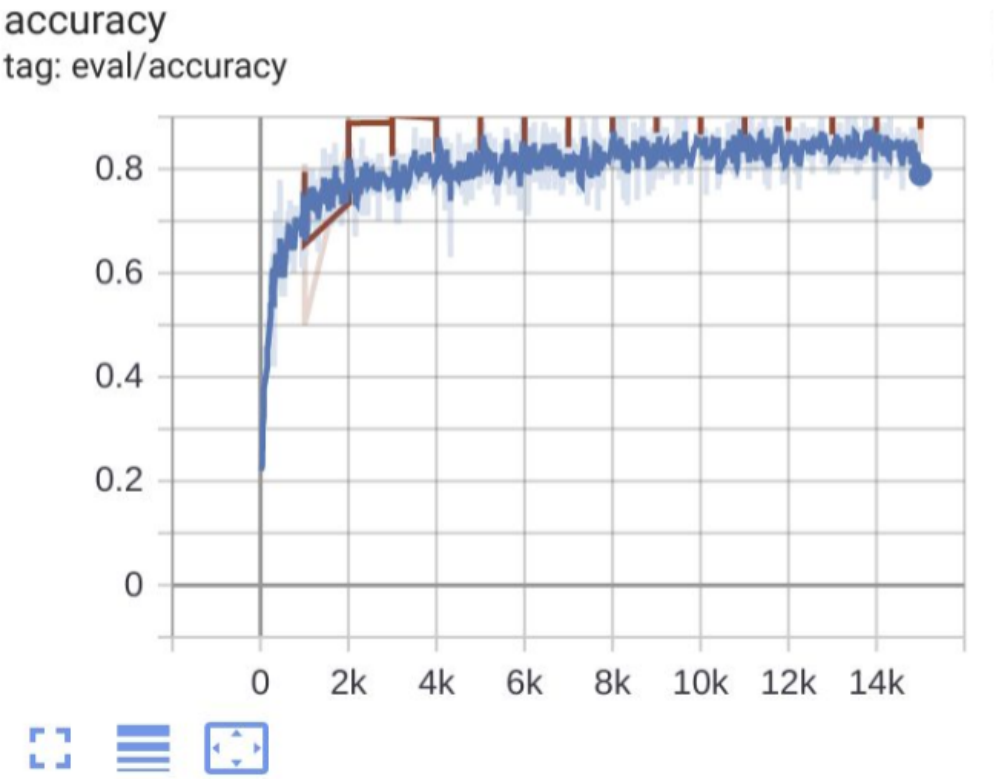
Monitoring Training in Colab

While the Keyword Spotting training script is running (for ~2 hours with the default settings and a GPU runtime), we may want to check-in and analyze how the training is performing. We've included two ways that we can do this in Colab.

TensorBoard

TensorBoard is an application that is designed to help us visualize the training process. It will output graphs of accuracy over time that look like something as shown below:



If you'd like to learn more about TensorBoard, Google has put together a nice intro Colab that explore some of its features:

https://colab.research.google.com/github/tensorflow/tensorboard/blob/master/docs/get_started.ipynb

DEBUG Mode

Unfortunately, the staff has found that it sometimes doesn't start showing data for a while (~15 minutes) and sometimes doesn't show data until training completes (and instead shows No dashboards are active for the current data set.). Therefore, we have also set the training script to run in DEBUG mode. By doing so it will print out information about the training process as it progresses. In general we see printouts at each training step showing the current accuracy that look something like this:

```
DEBUG:tensorflow:Step #25: rate 0.001000, accuracy 31.0%, cross entropy 1.342161
I1119 15:42:11.849751 140088470198144 train.py:257] Step #25: rate 0.001000, accuracy 31.0%, cross entropy 1.342161
DEBUG:tensorflow:Step #26: rate 0.001000, accuracy 33.0%, cross entropy 1.309237
I1119 15:42:12.187090 140088470198144 train.py:257] Step #26: rate 0.001000, accuracy 33.0%, cross entropy 1.309237
DEBUG:tensorflow:Step #27: rate 0.001000, accuracy 38.0%, cross entropy 1.294655
I1119 15:42:12.525735 140088470198144 train.py:257] Step #27: rate 0.001000, accuracy 38.0%, cross entropy 1.294655
DEBUG:tensorflow:Step #28: rate 0.001000, accuracy 35.0%, cross entropy 1.308346
I1119 15:42:12.882854 140088470198144 train.py:257] Step #28: rate 0.001000, accuracy 35.0%, cross entropy 1.308346
DEBUG:tensorflow:Step #29: rate 0.001000, accuracy 34.0%, cross entropy 1.343738
I1119 15:42:13.224437 140088470198144 train.py:257] Step #29: rate 0.001000, accuracy 34.0%, cross entropy 1.343738
DEBUG:tensorflow:Step #30: rate 0.001000, accuracy 38.0%, cross entropy 1.327567
I1119 15:42:13.560090 140088470198144 train.py:257] Step #30: rate 0.001000, accuracy 38.0%, cross entropy 1.327567
DEBUG:tensorflow:Step #31: rate 0.001000, accuracy 42.0%, cross entropy 1.334325
I1119 15:42:13.908692 140088470198144 train.py:257] Step #31: rate 0.001000, accuracy 42.0%, cross entropy 1.334325
DEBUG:tensorflow:Step #32: rate 0.001000, accuracy 37.0%, cross entropy 1.280344
I1119 15:42:14.248164 140088470198144 train.py:257] Step #32: rate 0.001000, accuracy 37.0%, cross entropy 1.280344
```

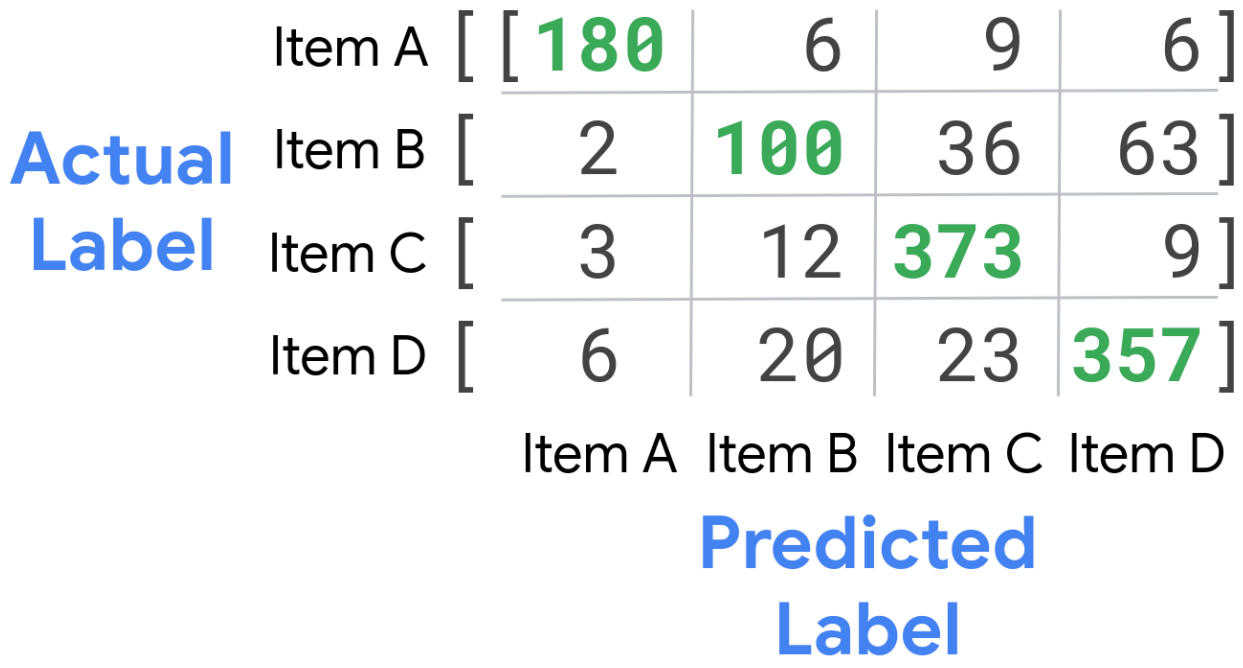
Remember because we are using a variant of stochastic gradient descent, it is natural for the error to go up and down a bit during training. The key thing to look for is that the larger trend is for the accuracy to be going up on average! For example, if we scroll down a bit on this training run we can see that ~500 steps later we now have much higher accuracy on average (so learning seems to be going well):

```
DEBUG:tensorflow:Step #552: rate 0.001000, accuracy 70.0%, cross entropy 0.783156
I1119 15:45:21.618249 140088470198144 train.py:257] Step #552: rate 0.001000, accuracy 70.0%, cross entropy 0.783156
DEBUG:tensorflow:Step #553: rate 0.001000, accuracy 76.0%, cross entropy 0.599380
I1119 15:45:21.958082 140088470198144 train.py:257] Step #553: rate 0.001000, accuracy 76.0%, cross entropy 0.599380
DEBUG:tensorflow:Step #554: rate 0.001000, accuracy 80.0%, cross entropy 0.637335
I1119 15:45:22.298068 140088470198144 train.py:257] Step #554: rate 0.001000, accuracy 80.0%, cross entropy 0.637335
DEBUG:tensorflow:Step #555: rate 0.001000, accuracy 83.0%, cross entropy 0.511202
I1119 15:45:22.637377 140088470198144 train.py:257] Step #555: rate 0.001000, accuracy 83.0%, cross entropy 0.511202
DEBUG:tensorflow:Step #556: rate 0.001000, accuracy 73.0%, cross entropy 0.677150
I1119 15:45:22.980068 140088470198144 train.py:257] Step #556: rate 0.001000, accuracy 73.0%, cross entropy 0.677150
DEBUG:tensorflow:Step #557: rate 0.001000, accuracy 73.0%, cross entropy 0.687472
I1119 15:45:23.314535 140088470198144 train.py:257] Step #557: rate 0.001000, accuracy 73.0%, cross entropy 0.687472
DEBUG:tensorflow:Step #558: rate 0.001000, accuracy 71.0%, cross entropy 0.628997
I1119 15:45:23.657741 140088470198144 train.py:257] Step #558: rate 0.001000, accuracy 71.0%, cross entropy 0.628997
DEBUG:tensorflow:Step #559: rate 0.001000, accuracy 71.0%, cross entropy 0.681584
I1119 15:45:24.164956 140088470198144 train.py:257] Step #559: rate 0.001000, accuracy 71.0%, cross entropy 0.681584
DEBUG:tensorflow:Step #560: rate 0.001000, accuracy 80.0%, cross entropy 0.560615
I1119 15:45:24.509710 140088470198144 train.py:257] Step #560: rate 0.001000, accuracy 80.0%, cross entropy 0.560615
DEBUG:tensorflow:Step #561: rate 0.001000, accuracy 71.0%, cross entropy 0.798682
```

Every 1,000 training steps we also see a **Confusion Matrix** printed out that looks something like this:

```
INFO:tensorflow:Confusion Matrix:
[[180   6   9   6]
[  2 100  36  63]
[  3  12 373   9]
[  6  20  23 357]
```

What this is describing is the number of samples that were correctly and incorrectly classified by class for the validation set at this point in the training process. Labeling the axis as shown below gives us a better idea!



The column plots the predicted item label by the model while the row plots the actual correct label. Down the diagonal in green are the number correctly labeled items. Off the diagonal we find the number of items incorrectly labeled as the various labels. For all of our Keyword Spotting models Item A is “silence”, Item B is “unknown” and then Items C, D, etc. are the various words we choose to train in the order in which we define them in the Colab. For example in the pretrained model we had:

```
WANTED_WORDS = "yes,no"
```

Which means that Item C was “yes” and Item D was “no” and the matrix was a 4x4 matrix. If we instead had the set:

```
WANTED_WORDS = "yes,no,left"
```

Then it would be a 5x5 matrix and Item C would still be “yes” and Item D would still be “no” and then there would be an Item E that was “left.”

We hope that between TensorBoard and the DEBUG output you’ll be able to understand how well your training is going and gain insights to improve your results!