Tensorflow 2 Questions:

1. What's the structure of the network that's being used for classification?

It’s a single-hidden layer neural network with 128 nodes followed by a dropout layer for generalizability and then a dense prediction layer of 10 nodes (1 for each number).

2. How good is it?

It’s pretty good. I get 0.9758.

3. Based on what you learned in homework 4, can you beat it?

By adding another hidden layer with dropout, I was able to improve the result to 0.9776

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
```

4. Note: you'll have to make changes to the code and fix the OOM errors. Hint: what is your batch size?

A batch size of 4 was the highest I could achieve.

5. Can you improve model accuracy? Hint: are your layers frozen?

With the feature_extractor layer.trainable set to false and with 2 epochs I get:

```
Epoch 1/10
918/918 [==============================] - 302s 329ms/step - loss: 0.5011 - acc: 0.7500
Epoch 2/10
918/918 [==============================] - 298s 325ms/step - loss: 0.2496 - acc: 1.0000
Epoch 3/10
918/918 [==============================] - 364s 397ms/step - loss: 0.1816 - acc: 1.0000
Epoch 4/10
918/918 [==============================] - 287s 313ms/step - loss: 0.1377 - acc: 1.0000
```

With the feature_extractor layer.trainable set to true and with 10 epochs I get:

```
Epoch 1/10
918/918 [==============================] - 360s 392ms/step - loss: 1.2181 - acc: 0.2500
Epoch 2/10
918/918 [==============================] - 325s 354ms/step - loss: 1.0357 - acc: 0.7500
Epoch 3/10
918/918 [==============================] - 325s 354ms/step - loss: 0.8805 - acc: 0.5000
Epoch 4/10
918/918 [==============================] - 325s 354ms/step - loss: 0.8448 - acc: 0.5000
Epoch 5/10
918/918 [==============================] - 325s 354ms/step - loss: 0.8072 - acc: 1.0000
Epoch 6/10
918/918 [==============================] - 325s 354ms/step - loss: 0.8120 - acc: 0.5000
Epoch 7/10
918/918 [==============================] - 325s 354ms/step - loss: 0.6994 - acc: 1.0000
Epoch 8/10
918/918 [==============================] - 325s 354ms/step - loss: 0.7061 - acc: 1.0000
Epoch 9/10
918/918 [==============================] - 325s 354ms/step - loss: 0.6477 - acc: 0.7500
Epoch 10/10
918/918 [==============================] - 325s 355ms/step - loss: 0.6277 - acc: 0.2500
```

As seen above, the results are very unstable.

Tensorflow 1 Questions:

1. What is TensorFlow? Which company is the leading contributor to TensorFlow?

Tensorflow is an open-source library used primarily for machine learning and AI development. As its name implies, the library provides many tools to work with tensors, the basis of neural networks and deep learning. Google is the leading contributor to TensorFlow

2. What is TensorRT? How is it different from TensorFlow?

TensorRT is an SDK by NVIDIA that works with in conjunction with Tensorflow. It is a model optimizer that can reduce training time while improving accuracy and performance in production for models that are being developed on Tensorflow. One of its key abilities is to fully leverage the power of GPUs that the algorithm might be running on

3. What is ImageNet? How many images does it contain? How many classes?

ImageNet is an image database based on WordNet. In WordNet, a group of words (mostly nouns) are grouped by how well they can describe a concept. These groupings are called synsets. ImageNet provides about 1000 images per synset. There are over 100 000 synsets therefore likely 100 000 000+ images. As of April 30th, 2010, the statistics are:

- Total number of non-empty synsets: 21841
- Total number of images: 14,197,122
- Number of images with bounding box annotations: 1,034,908
- Number of synsets with SIFT features: 1000
- Number of images with SIFT features: 1.2 million

Additionally, there are 27 categories including sports, birds, animals, etc.

4. Please research and explain the differences between MobileNet and GoogleNet (Inception) architectures.

MobileNets are a family of models built on Tensorflow and optimized for mobile devices. There are two hyperparameters that you can tune that allow you to adjust between latency and accuracy. GoogLeNet is a pretrained convolutional neural network that is 22 layers deep and is something you can just plug data in. The network trained on ImageNet classifies images into 1000 object categories, such as keyboard, mouse, pencil, and many animals.

5. In your own words, what is a bottleneck?

A bottleneck is a restricting gateway or process that constrains the volume or velocity of something as it passes through.

"Bottleneck" is not used to imply that the layer is slowing down the network. We use the term bottleneck because near the output, the representation is much more compact than in the main body of the network. Every image is reused multiple times during training.

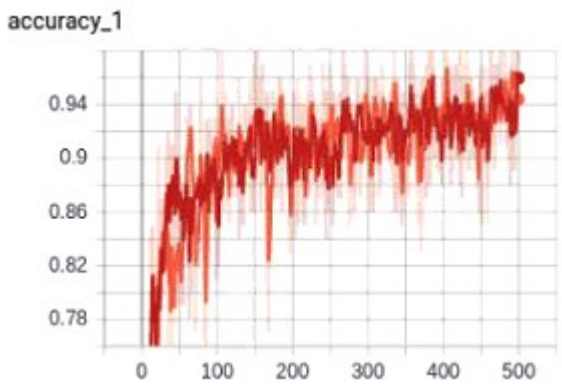
6. How is a bottleneck different from the concept of layer freezing?

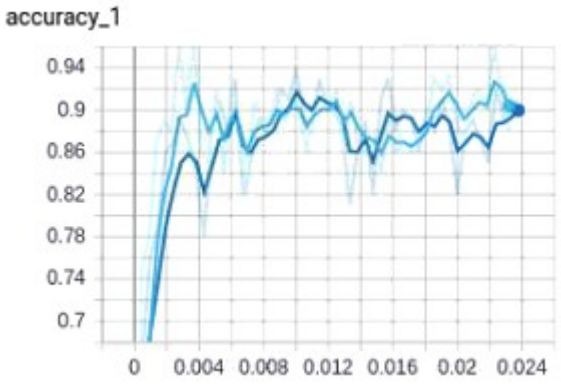
From a machine learning perspective, a bottleneck is the layer just before the final output layer that actually does the classification. Bottlenecking is when you pass data through a pretrained network/weights that are frozen where frozen implies that weights can’t be adjusted with each training iteration.

7. In this lab, you trained the last layer (all the previous layers retain their already-trained state). Explain how the lab used the previous layers (where did they come from? how were they used in the process?)

The previous layers’ weights are pretrained from a separate process. They are essentially frozen and only the final layers weights are updated through a back propagation process given a new dataset. We are essentially fine tuning the model here.

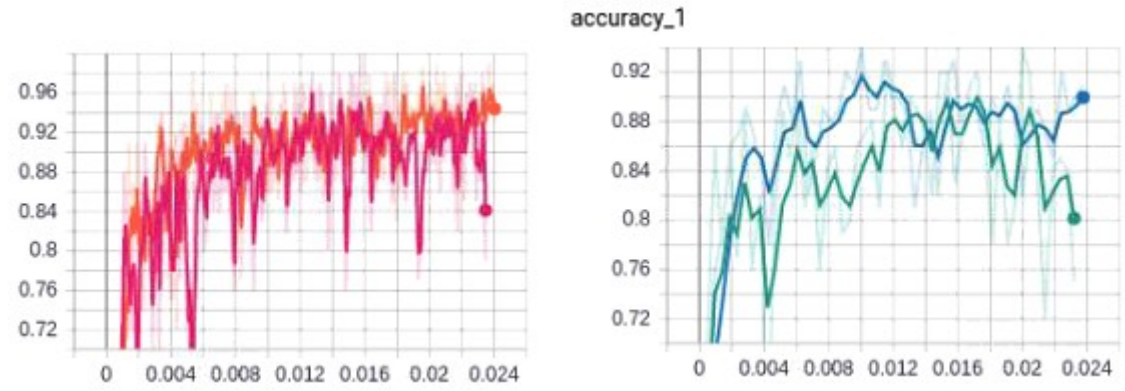
8. How does a low --learning_rate (step 7) value (like 0.005) affect the precision? How much longer does training take?





The left graph shows training data with default settings (orange) vs when the learning rate is 0.005 (red). The right graph shows the dev data with default settings (dark blue) vs when the learning rate is 0.005 (light blue). In terms of the training data, it takes about the same amount of time approximately to learn however to get a better perspective, we look at the dev data. Here we can see that lowering the learning rate leads to the model reaching a good accuracy (~0.9) earlier compared to the default settings. We can see that the saturation happens around 0.004 whereas with the default settings, the saturation happens at about 0.01.

9. How about a --learning_rate (step 7) of 1.0? Is the precision still good enough to produce a usable graph?

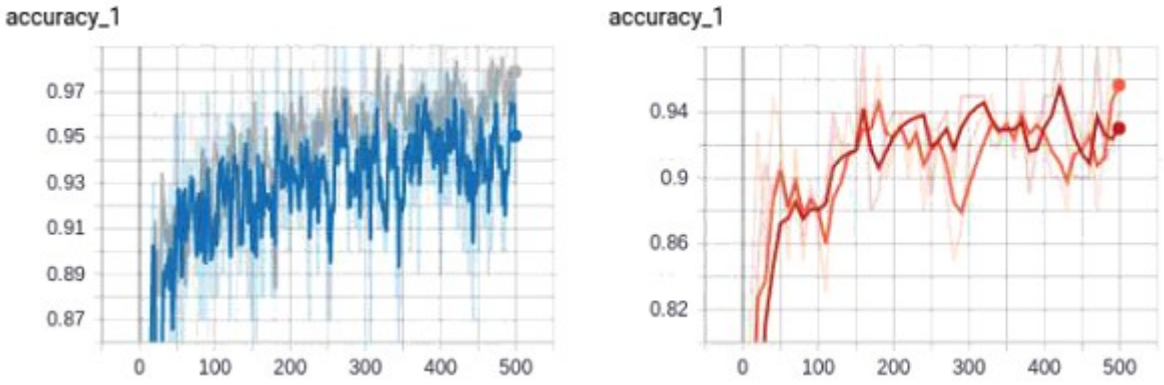


In contrast to the above, having a learning rate of 1.0 produces very questionable results. Not only is the training very unstable but looking at the development data, the overall accuracy seems to decrease with increased training. It seems to be stuck in a minima where it works well with the training data but not the development data. Ie. it's learning on the wrong features.

10. For step 8, you can use any images you like. Pictures of food, people, or animals work well. You can even use ImageNet images. How accurate was your model? Were you able to train it using a few images, or did you need a lot?

The following dataset of animals was used: https://www.kaggle.com/alessiocorrado99/animals10/data#.

It contains about 28K medium quality animal images belonging to 10 categories: dog, cat, horse, spyder, butterfly, cicken, sheep, cow, squirrel, elephant. We also take a smaller subset of 500 images per category to analyze the second question



Here the grey and orange line represent the smaller dataset whereas the dark blue and red lines represent the larger dataset. Interestingly, on training, we can achieve a higher accuracy with a smaller dataset. This could be due to increased variability with more data. Nevertheless, looking at the development data, we can see that both models achieve a similar accuracy. The range of which seems to be 0.90-0.95.

11. Run the script on the CPU (see instructions above) How does the training time compare to the default network training (section 4)? Why?

Running it using the container:

```
docker run --rm -p 6006:6006 -ti tensorflow bash`
```

the dev set reached its plateau after about 35 seconds.

Running it with the container:

```
docker run --privileged --rm -p 6006:6006 -ti w251/tensorflow:dev-tx2-4.3_b132-tf1 bash
```

the dev set reached its plateau after about 25-30 seconds

12. Try the training again, but this time do export ARCHITECTURE="inception_v3" Are CPU and GPU training times different?

Running it using the container:

```
docker run --rm -p 6006:6006 -ti tensorflow bash`
```

the dev set reached seems to have reached its plateau after about 2.5 minutes. Its important to note here that compared to question 11, the training of the bootstrap layer was significantly longer (>15 min compared to 2-3 minutes)

Running it with the container:

```
docker run --privileged --rm -p 6006:6006 -ti w251/tensorflow:dev-tx2-4.3_b132-tf1 bash
```

With the GPU, the plateau was reached in about 1.5 minutes. This is clearly a bit faster than with just CPU alone.

13. Given the hints under the notes section, if we trained Inception_v3, what do we need to pass to replace ??? below to the label_image script? Can we also glean the answer from examining TensorBoard?

The below seems to work:

```
python -m scripts.label_image --input_layer=Mul --input_height=299 --
input_width=299  --graph=tf_files/retrained_graph.pb
–image=tf_files/flower_photos/daisy/21652746_cc379e0eea_m.jpg
```