University of Kentucky

IBM Deep Learning Workshop

Jason Crites
IBM Healthcare and Life Sciences Solution Leader

Dr. Anthony Skjellum
Director UT Chattanooga Sim Center

Chekuri Choudary
Cognitive Systems Specialist

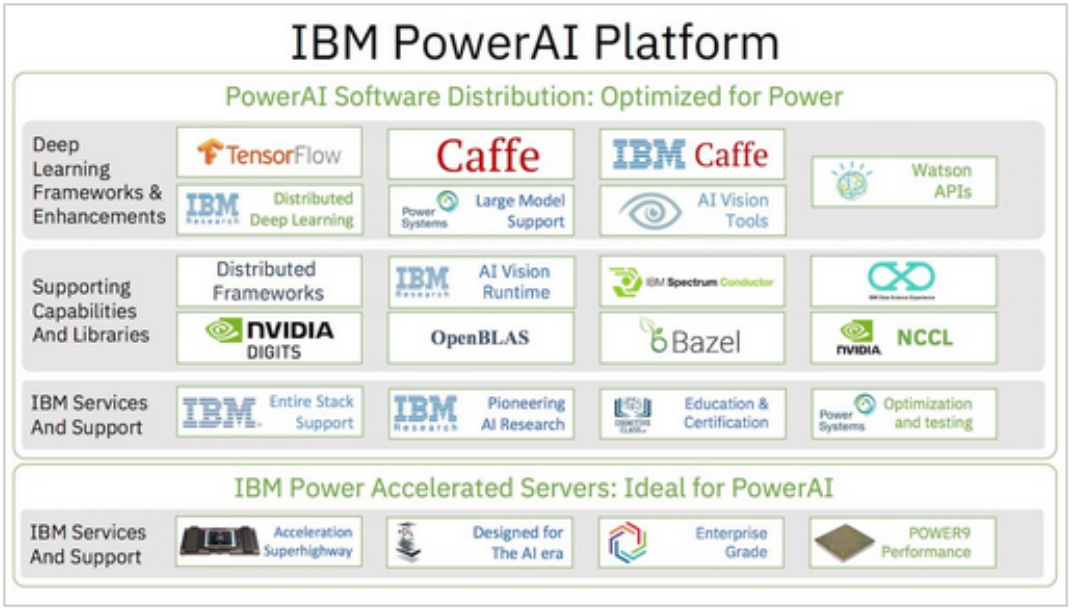
James Nash
Deep Learning Technical Specialist

July 23-24th - 9-4:30PM

---

http://172.17.150.18/PowerAILab/

Hands-on Lab



Abstract

In this lab, users will divide up into 16 teams We have four AC922 Power Systems severs, each with four NVIDIA Tesla V100 GPUs. By allocating one GPU on each system to a container, we have 16 discrete user spaces for lab work.

Each team will have a unique user ID with authorization to instantiate a container. Each of the tasks outlined below will be executed with this container, allowing the teams to operate independently, without impacting each other.

Outline

Section 1 How to connect to an AC922 Power System and instantiate a container with one (1) GPU.

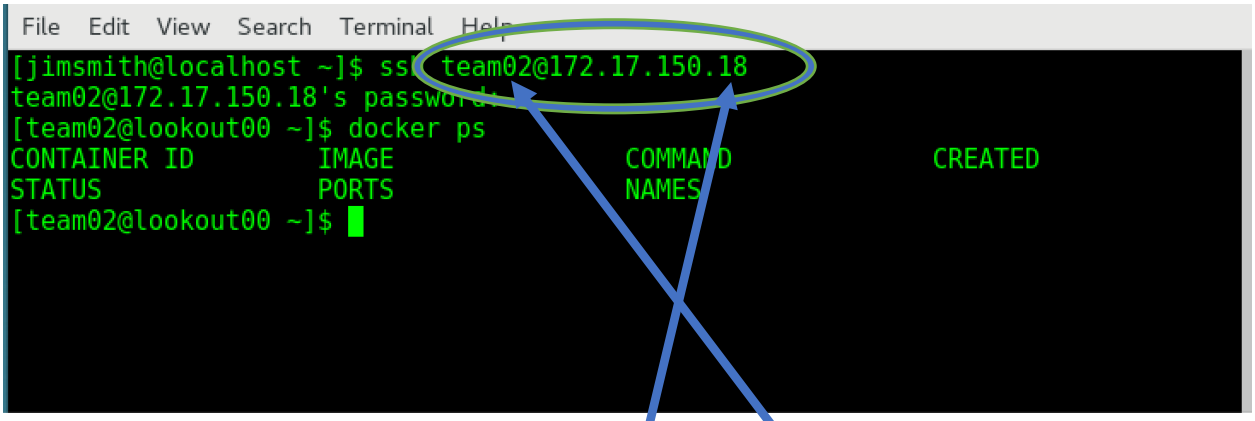
Get a teamxx assignment and Sign in!

1. If you do not already have one, get a lab slip from the instructor:

team01

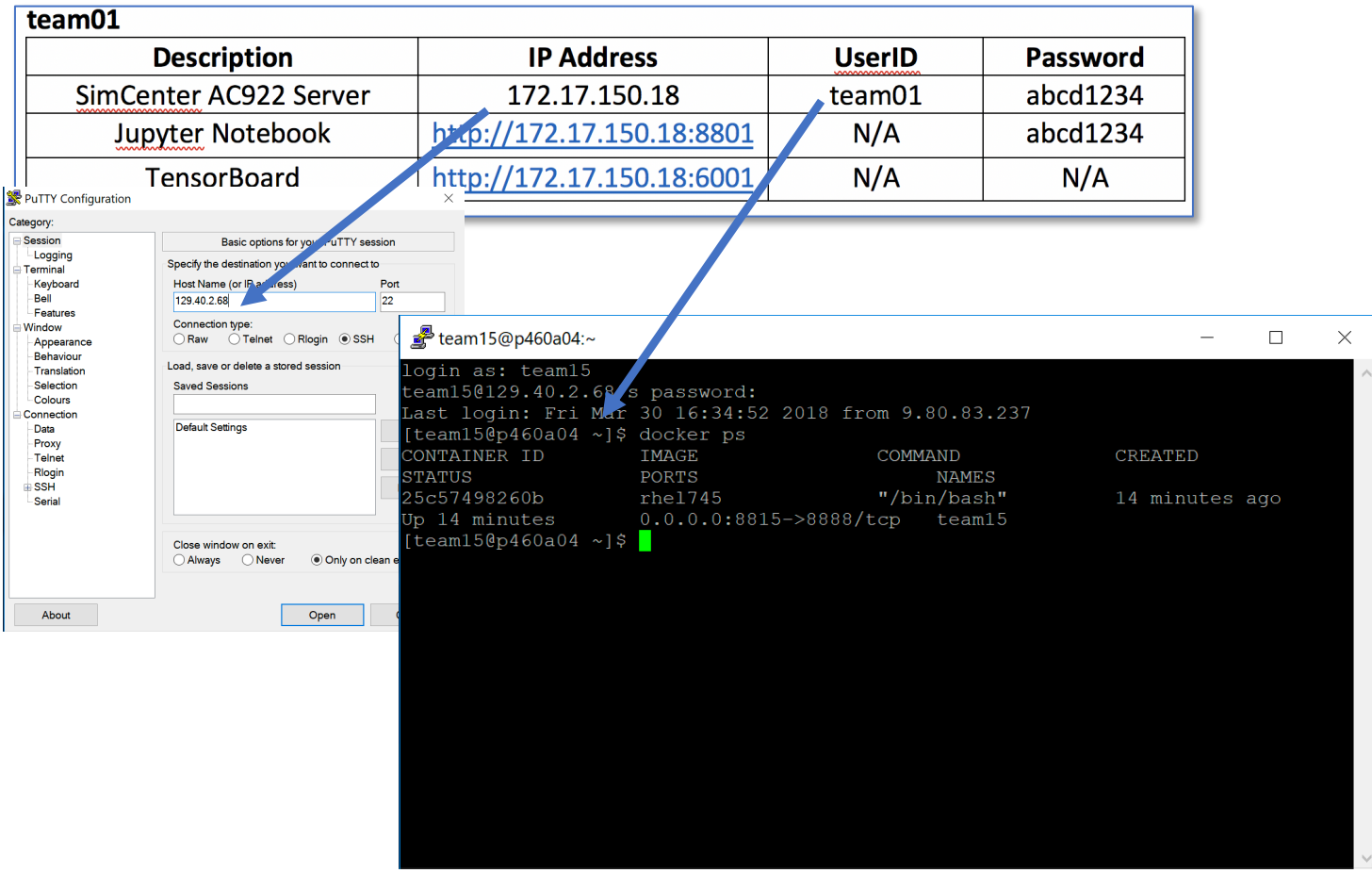
Description	IP Address	UserID	Password
SimCenter AC922 Server	172.17.150.18	team01	abcd1234
Jupyter Notebook	http://172.17.150.18:8801	N/A	abcd1234
TensorBoard	http://172.17.150.18:6001	N/A	N/A

If you're using Mac OS or Linux, Terminal is a good choice, or on Windows, Putty works well. Sign on to your assigned host using the IP Addess, UserID, and Password.



team01

Description	IP Address	UserID	Password
SimCenter AC922 Server	172.17.150.18	team01	abcd1234
Jupyter Notebook	http://172.17.150.18:8801	N/A	abcd1234
TensorBoard	http://172.17.150.18:6001	N/A	N/A



3. List all containers on your host

Once you have signed on to your assigned system, check to see if there are any containers running by using the "**docker ps**" command:

```
[teamxx@p460a04 ~]$ docker ps

CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
```

Notice in this output, that there are no containers running currently on the system. Therefore, you're free to proceed to the next step.

If there are containers listed, then you'll want to ensure that they're all for other teams. Consider the following output:

```
[team13@p460a04 ~]$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS                    NAMES
0add051f7aed        rhel745             "/bin/bash"         34 seconds ago
Up 33 seconds       0.0.0.0:8814->8888/tcp   team14
0e0c8c2e39ec        rhel745             "/bin/bash"         About a minute
ago   Up About a minute   0.0.0.0:8816->8888/tcp   team16
25c57498260b        rhel745             "/bin/bash"         About an hour
ago    Up About an hour    0.0.0.0:8815->8888/tcp   team15
```

This time notice that containers are running for team14, team15 and team16, (Note: the output has wrapped as the terminal display is not as wide as the command output).

If you are team13, then no worries! However, if you are team14-16, then something is wrong, you are ready to start a container and one is already running for your team, seek assistance from the instructor!

4. Start your container

Listing the files in your home directory you will see that there is only one, an executable script to start your container. Type "**./lab-start**" to start and enter your container.

```
[team01@lookout00 ~]$ ll
total 4
-rwxr-xr-x 1 root root 189 Jun  2 08:58 lab-start

[team01@lookout00 ~]$ ./lab-start
[root@0fcdb7b107f3 /]#
```

5. You're now root in your own Docker container!

Notice that you shell prompt changes from a default Bash $ to the all-powerful root #! The number after the root@ prefix is your container ID by the way.

```
[team01@lookout00 ~]$ ll
total 4
-rwxr-xr-x 1 root root 189 Jun  2 08:58 lab-start

[team01@lookout00 ~]$ ./lab-start
[root@0fcdb7b107f3 /]#
```

Don't let it go to your head, while you're the super-user of your container, you're still nothing more than a simple user to the hosting OS.

6. Let's test the GPU

One GPU is allocated to each container for this lab, so that each team can install and test PowerAI. Let's ensure our GPU is here with us in our container, and working properly.

Issue an **"nvidia-smi"** command to see what GPUs we have access to:

```
[root@0fcdb7b107f3 /]# nvidia-smi
Sat Jun  2 11:06:36 2018
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 396.26                 Driver Version: 396.26                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  Off  | 00000004:04:00.0 Off |                    0 |
| N/A   31C    P0    36W / 300W |     10MiB / 15360MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
[root@0fcdb7b107f3 /]#
```

If you're output looks similar to this, then Great News!!! Proceed to the next step. If not, then seek assistance from the instructor.

Section 2 Image Classification using Caffe

This tour is designed to introduce you to image classification using Caffe. You'll use a Jupyter Notebook for this, and you'll observe how GPUs speed up image classification.

The Jupyter Notebook consists of a server which you'll start on the system we provided, and a client that runs in your laptop's web browser. You interact with the server through the GUI presented by the client.

Caffe is an open-source deep learning framework from the Berkeley Vision and Learning Center. In this tour you'll walk through a deep learning 'image classification' example using an open-source model already trained on images from the ImageNet public dataset. (More open source pre-trained models can be found at the Caffe Model Zoo. More open source example code can be found at Github.)

Image classification is the process whereby you input an image (i.e. a photograph of something) and expect the system to analyze and recognize the image. The system is able to do this (or not) based on training of its neural network beforehand. In other words, before asking the system to perform image classification, you train the system's neural network to recognize certain images. For example, you might feed the neural network thousands of images of house cats so it can train itself to recognize random images of house cats.

The details of the training algorithm are far beyond the scope of this PoT, and most people don't even care, and don't need to care. You can consider it a blackbox. You train it by supplying thousands and thousands, if not millions, of known images; and when 'enough' training has been done, you can then submit individual, unknown images and ask the system to identify them. In the exercise below, the training has already occurred. The exercise simply asks the system to identify a certain input image (i.e. to perform image classification), and it also demonstrates how the use of GPUs can greatly speed up image classification.

If you're currently logged into your container, proceed below! If not, then use the steps in **Section 1** to log in and start your container.

Configuring your Jupyter Notebook

For many of the lab sections we'll be using Jupyter Notebooks. WIthin the **/files/PowerAILab** directory, there is a script for configuring your Jupyter session for this lab. Once this session is launched, you will point your browser to an address and open the notebook for each lab section which will step you through the lab you're doing.

Run "**cfgjptr_caffe.sh 8890**" followed by "**source ~/.bashrc**":

```
[root@0fcdb7b107f3 /]# cfgjptr_caffe.sh  8890
Writing default config to: /root/.jupyter/jupyter_notebook_config.py
[root@0fcdb7b107f3 /]#

[root@0fcdb7b107f3 /]# source ~/.bashrc
[root@0fcdb7b107f3 /]#
```

Use **caffe-activate** to set some environment variables

```
[root@0fcdb7b107f3 /]# source /opt/DL/caffe/bin/caffe-activate
[root@0fcdb7b107f3 /]#
```

Start a Jupyter Notebook

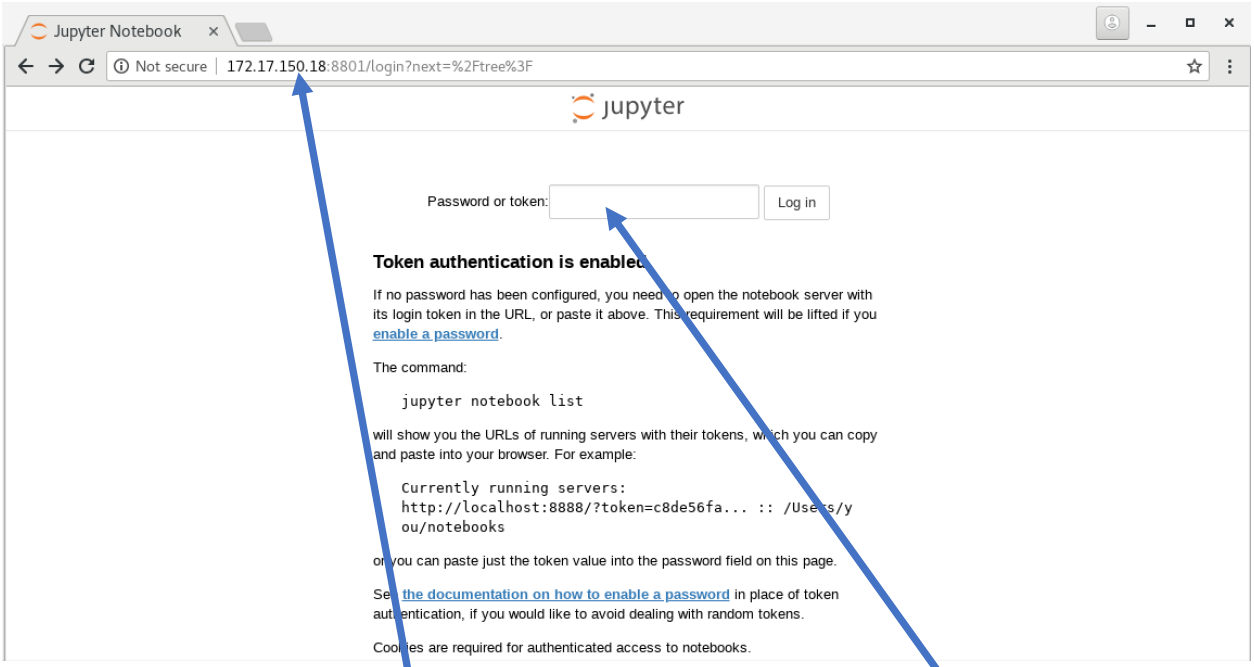
```
[root@0fcdb7b107f3 /]# jupyter notebook --no-browser --allow-root
[I 11:24:44.772 NotebookApp] Writing notebook server cookie secret to
/root/.local/share/jupyter/runtime/notebook_cookie_secret
[I 11:24:45.434 NotebookApp] JupyterLab beta preview extension loaded from
/usr/local/anaconda2/lib/python2.7/site-packages/jupyterlab
[I 11:24:45.434 NotebookApp] JupyterLab application directory is
/usr/local/anaconda2/share/jupyter/lab
[I 11:24:45.439 NotebookApp] Serving notebooks from local directory:
/opt/DL/caffe/examples
[I 11:24:45.439 NotebookApp] 0 active kernels
[I 11:24:45.440 NotebookApp] The Jupyter Notebook is running at:
[I 11:24:45.440 NotebookApp] http://0.0.0.0:8890/?token=...
[I 11:24:45.440 NotebookApp] Use Control-C to stop this server and shut
down all kernels (twice to skip confirmation).
```

Open a Browser and point it at your Jupyter Notebook

Our next steps will all be hosted from the Jupyter Notebook we've just brought up in our container.

In the URL field of your favorite Browser, enter your assigned IP address and port number:

**Hint the IP address and port number is on your slip



team01

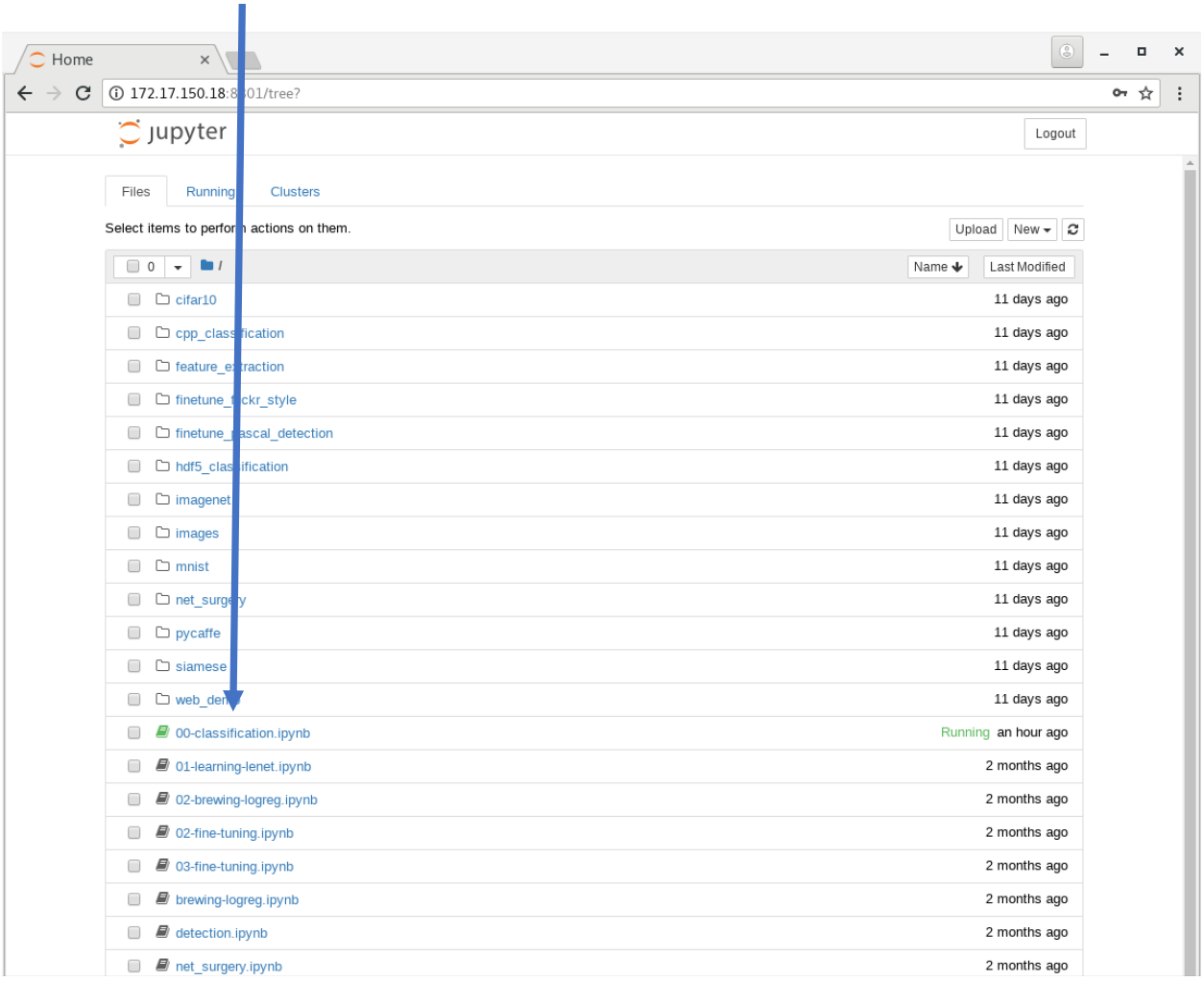
Description	IP Address	UserID	Password
SimCenter AC922 Server	172.17.150.18	team01	abcd1234
Jupyter Notebook	http://172.17.150.18:8801	N/A	abcd1234
TensorBoard	http://172.17.150.18:6001	N/A	N/A

You'll be prompted for a password or token, this is also on your slip!

After entering (your super-secret password abcd1234) you will get a page with a link for the Lab Jupyter Notebook. Hurray!!

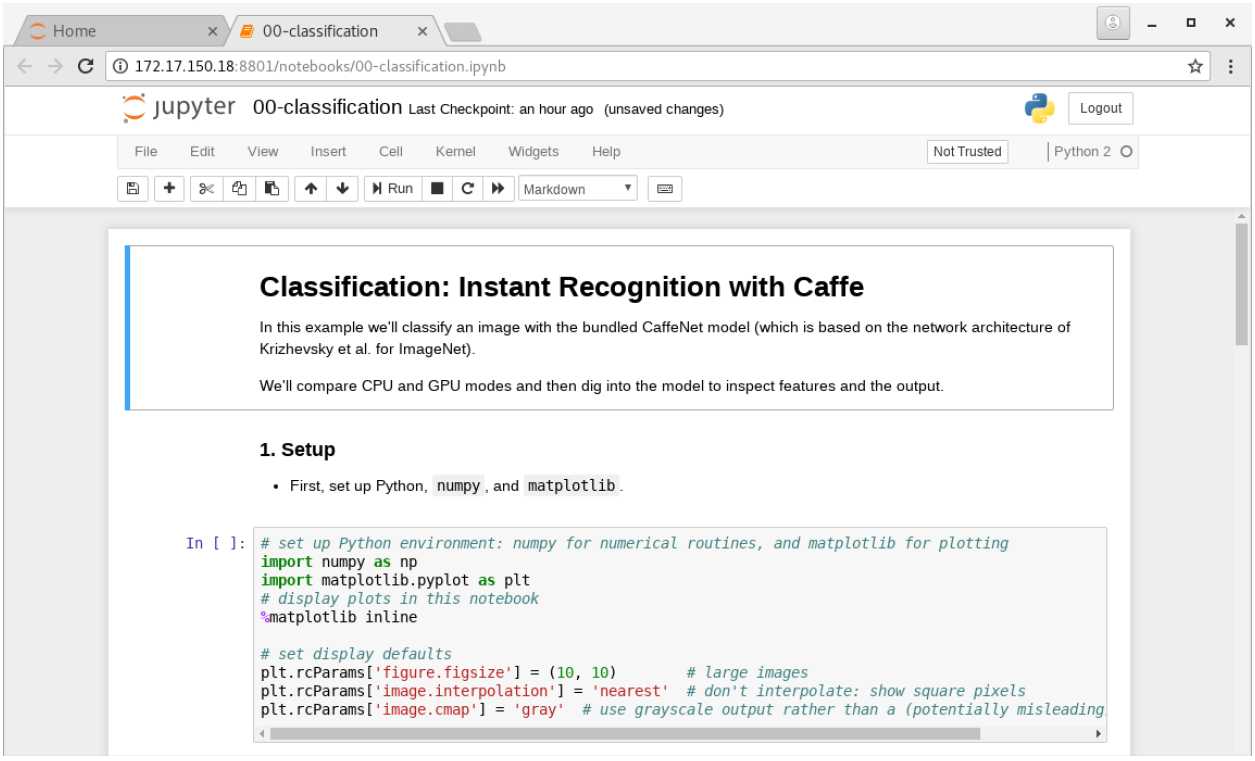
You'll see the Jupyter Notebook home page which looks like this, scroll down to find:

00-classification.ipynb and click it:



Before proceeding, let's explain a few things.

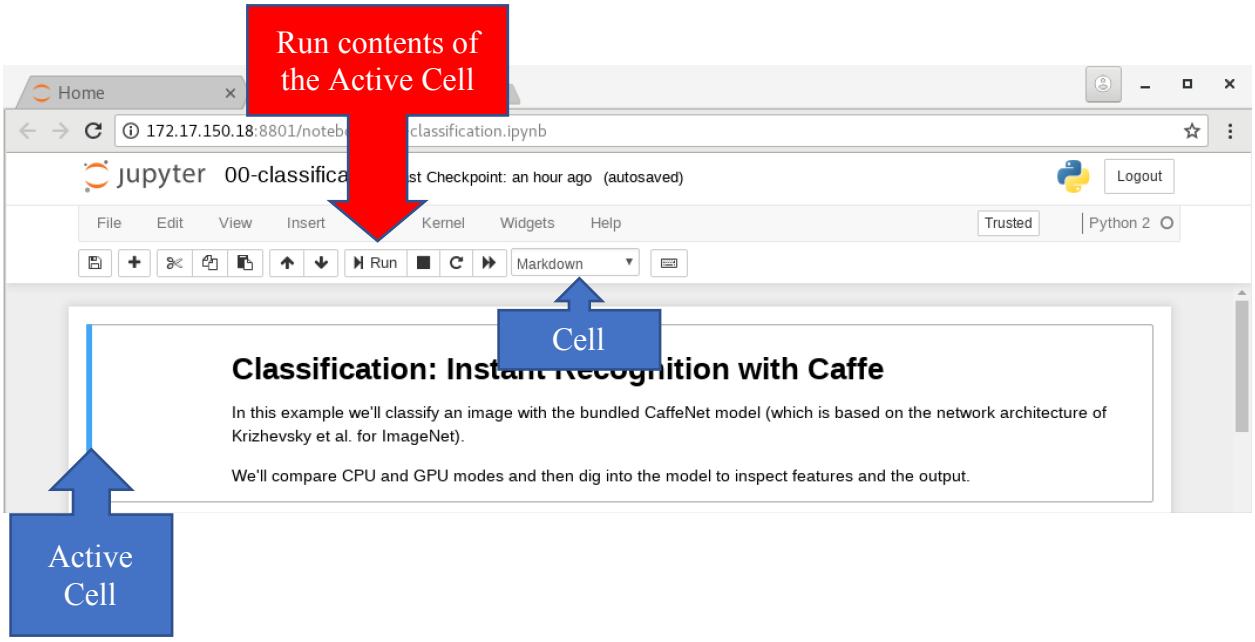
The 00-classification.ipynb file comes from /opt/DL/caffe/examples on your container. It's an 'ipynb' (or IPython Notebook) file created by Jupiter Notebook members (an open-source community) to demonstrate image classification using Caffe, and to show how GPUs speed up image classification. These notebook files are designed to be created, viewed, edited, manipulated, and run in the Jupyter client within a web browser. Notebook files contain various 'cells' - Heading cells, Markdown cells, Code cells, and Raw cells.



If you're interested in what ipynb files look like, you can open another PuTTY ssh session to your container and inspect 00-classification.ipynb. For additional information on these 'notebook' files, take a look at: http://jupyternotebook.readthedocs.io/en/latest/notebook.html

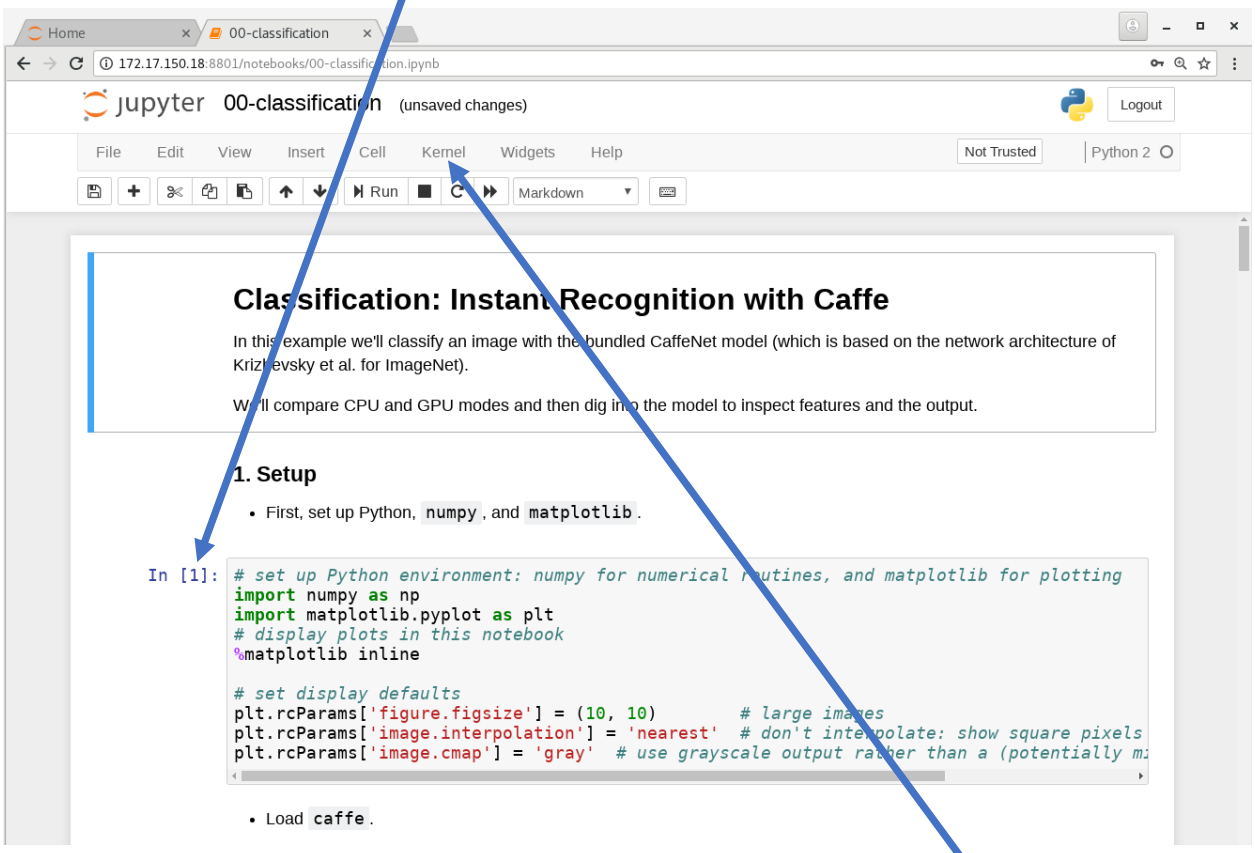
Note, however, that .ipynb files are not typically created by hand; they are created using the Jupyter Notebook.

In the screen shot below, the blue bar on the left highlights the active cell you are in, and to the right of the control buttons along the top, you can see that this particular cell is a 'Markdown' cell, which is really just a text cell used to document things in the notebook. The run button will step through code, section by section.



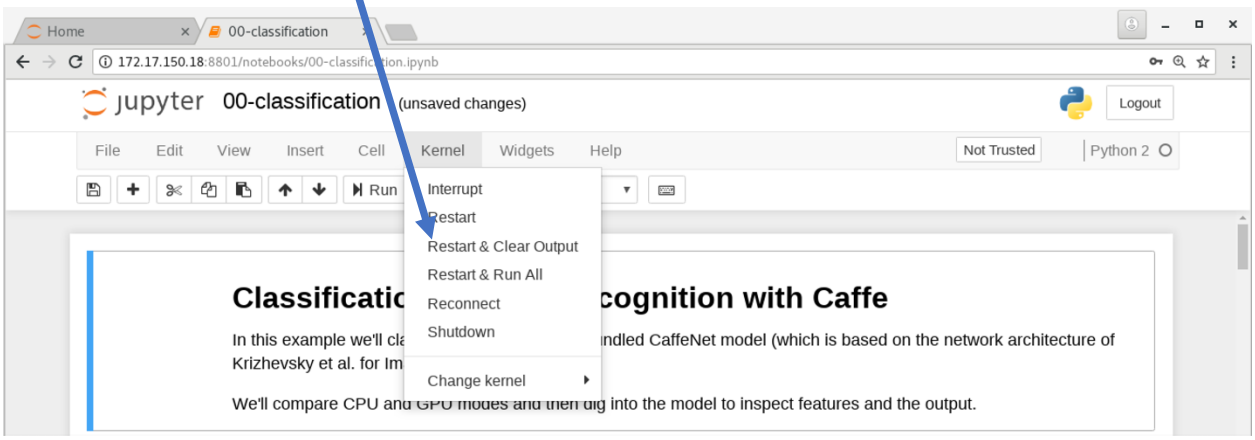
When you start up the 00-classification.ipynb file, it may have output from a previous invocation. (Jupyter by default automatically save Notebooks, so this is how output from a previous session might be present.)

If there is a number in the **In [ ]** box, the Notebook has previous output present.

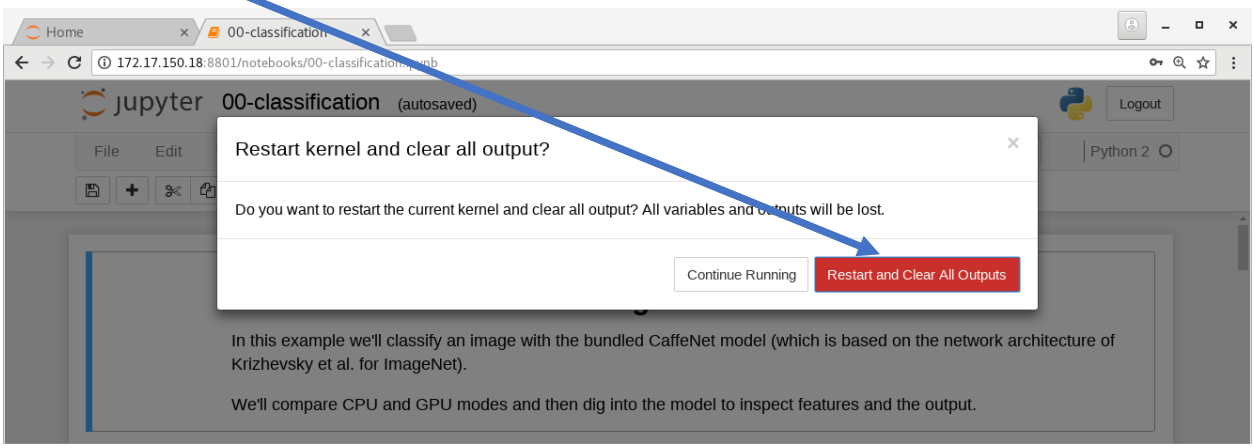


Not to worry! We can **Restart & Clear Output** by clicking on the Kernel tab.

Then click **Restart & Clear Output**.

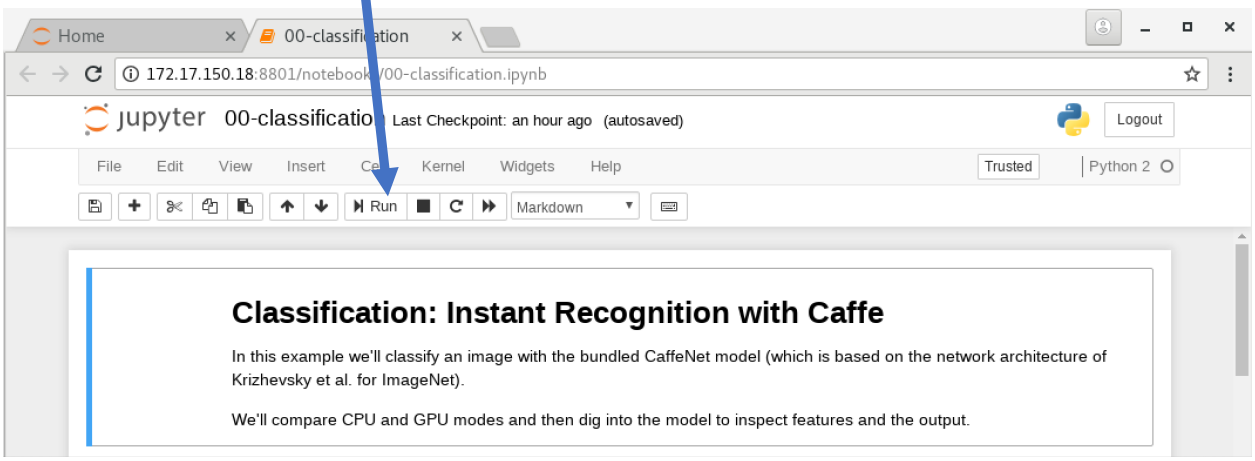


To confirm that we want to restart the kernel and clear all output we need to click the **Restart and Clear All Outputs button**.



Step through the Notebook

Go ahead and click the Run button once to see how it advances you to the next cell.



Ok, now that you understand the mechanics of the Jupyter Notebook, you're ready to step through 00-classification.ipynb. As you do this, you'll see that Markdown cells simply get highlighted, and Code cells run. And the output of Code cells that generate output will appear inline in the notebook.

However, don't simply click, click, click through the notebook!

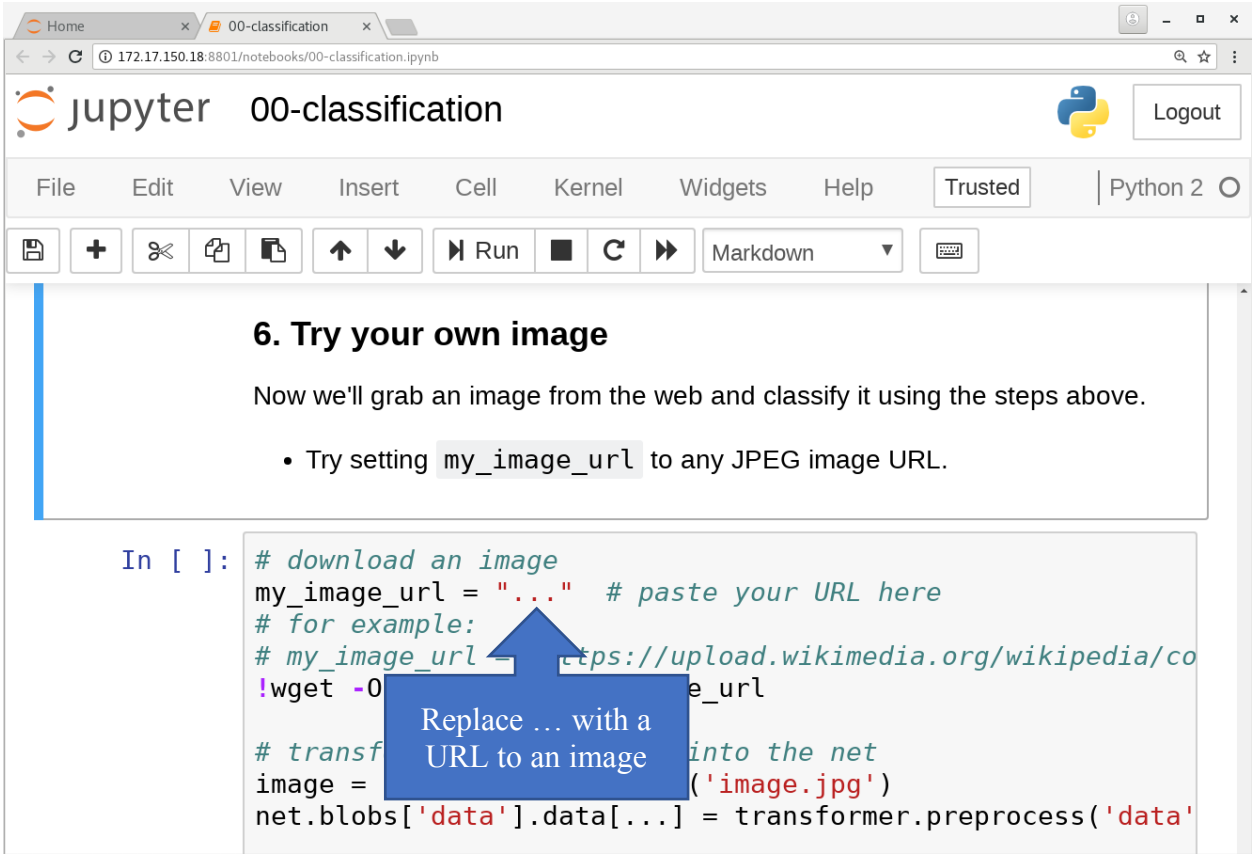
Take your time to read the Markdown text to understand what the notebook is attempting to demonstrate. If you don't do this, you'll get to the end and ask yourself "what did I just do, and why? what did I learn?".

One more thing to notice: When you first arrive at a Code cell, you will probably see an Input indicator like In [5] to the left of the Code cell. This is left over from the last time someone stepped through the notebook.

When you click the button to then run Code cell yourself, the Input indicator will change to In [*], sometimes for only a instant, and sometimes for several seconds or minutes. When the asterisk in the brackets is replaced by a number (e.g. In [12]), the code in that cell has finished running, and you can click to move to the next cell. (While you can click ahead without waiting, it's less confusing if you wait for Code cells to complete before stepping ahead.)

We are performing classification using the system's CPU vs. a GPU! The system's CPU takes seconds to complete the task; whereas the GPU complete image classification in milliseconds.

When you arrive at step 6, you can try this by pasting a URL for a JPG here.



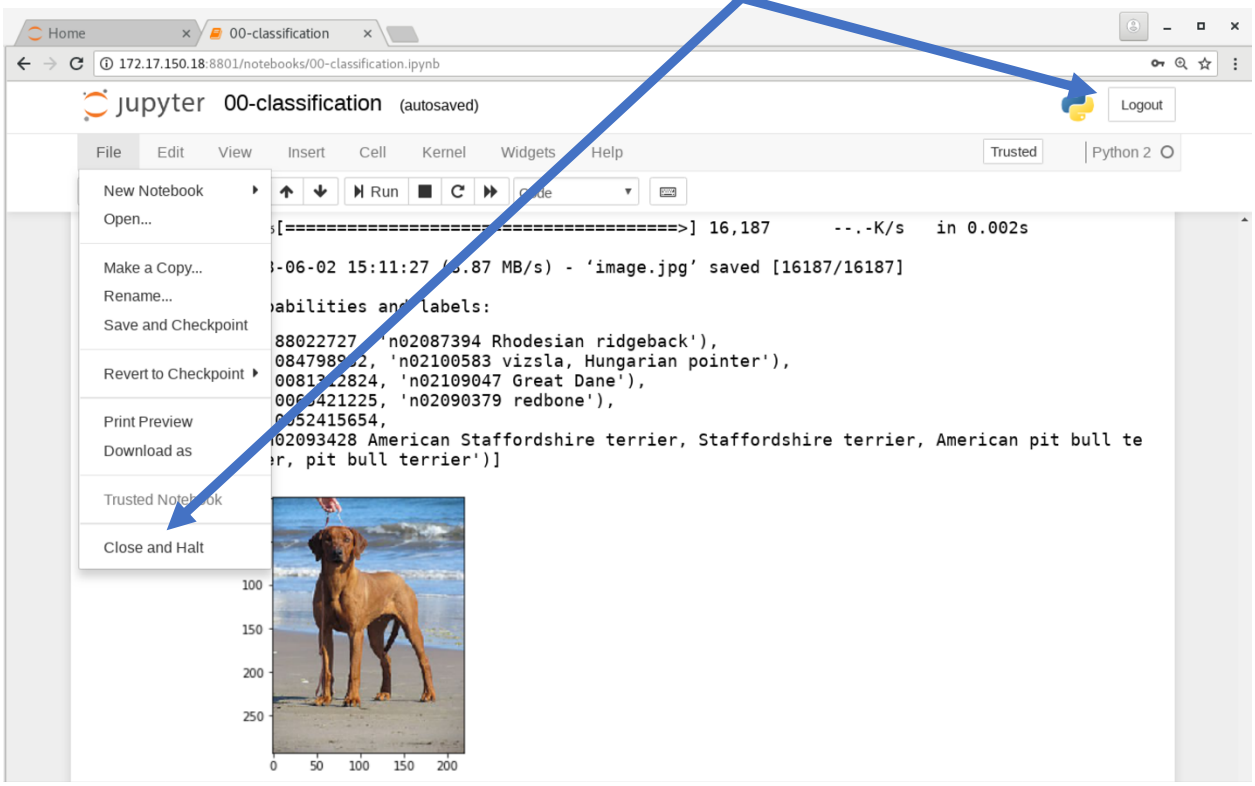
Here are some examples you can try:

https://upload.wikimedia.org/wikipedia/commons/thumb/b/ba/Male_Rhodesian_ridgeback_IMG_6800.JPG/220px-Male_Rhodesian_ridgeback_IMG_6800.JPG

https://s-media-cache-ak0.pinimg.com/originals/7a/85/74/7a857483e16e60cf1c2d54f978414d86.jpg

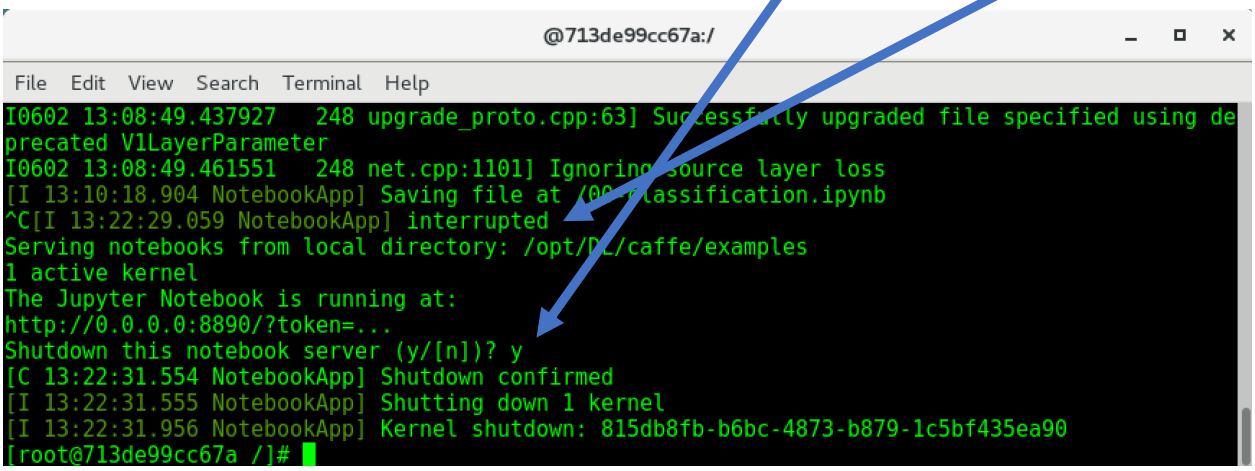
Try other images, note that our pre-trained CaffeNet module works pretty well with images of animals. You may notice it is not trained very well for individuals or logos today. Check-out “other learning activities” below to understand how to train a framework.

After completing step "6 Try your own image", click **File > Close and Halt** to close and halt the notebook. Then click **Logout**:



Conclude this Section

Find your ssh session where you started the Jupyter Notebook server, and type Ctrl-C to interrupt the notebook server. Then immediately enter 'y' to shutdown the notebook server:



Note: If you don't enter 'y' within 5 seconds, the notebook server resumes. If this happens, just type Ctrl-C again and enter 'y' quicker than last time! :)

Optionally, enter 'exit' to terminate your container, and then enter 'exit' again to end your ssh session. If you'll be doing additional lab sections, no need to exit your container

This concludes **Section 2 Image Classification using Caffe**. In this section you learned a bit about the Jupyter Notebook and IPython Notebook files; you used Caffe to perform image classification; and you observed how beneficial GPUs are for speeding up tasks like image classification.

Section 3 Using MNIST and TensorFlow

This lab is intended to give users an example of normal (albiet simple) process by which one could use TensorFlow to help identify handwritten characters. This will also expose you to TensorBoard which could be used to visualize or troubleshoot potential problems.

MNIST is a simple computer vision dataset. It consists of images of handwritten digits like these:

It also includes labels for each image, telling us which digit it is. For example, the labels for the images above are 5, 0, 4, and 1.

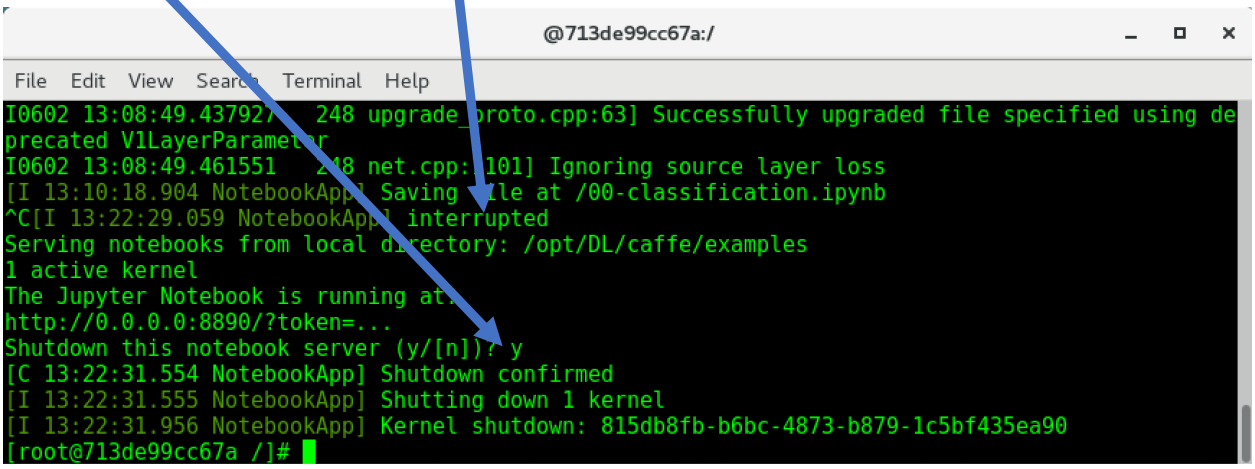
In this tutorial, we're going to train a model to look at images and predict what digits they are. Our goal isn't to train a really elaborate model that achieves state-of-the-art performance -- although we'll give you code to do that later! -- but rather to dip a toe into using TensorFlow. As such, we're going to start with a very simple model, called a Softmax Regression.

The actual code for this tutorial is very short, and all the interesting stuff happens in just three lines. However, it is very important to understand the ideas behind it: both how TensorFlow works and the core machine learning concepts. Because of this, we are going to very carefully work through the code.

GOALS:

- **Learn about the MNIST data and softmax regressions**
- **Create a function that is a model for recognizing digits, based on looking at every pixel in the image**
- **Use TensorFlow to train the model to recognize digits by having it "look" at thousands of examples (and run our first TensorFlow session to do so)**
- **Check the model's accuracy with our test data**
- **Look at the model using TensorBoard**
- **Understand how this would run on a GPU (20x faster on a Tesla P100)**

If you're currently logged into your container, proceed below! If not, then use the steps in Section 1 to log in and start your container. If your running a Jupyter Notebook from a prior section, cancel it with Ctrl-C to interrupt the notebook server. Then immediately enter 'y' to shutdown the notebook server.



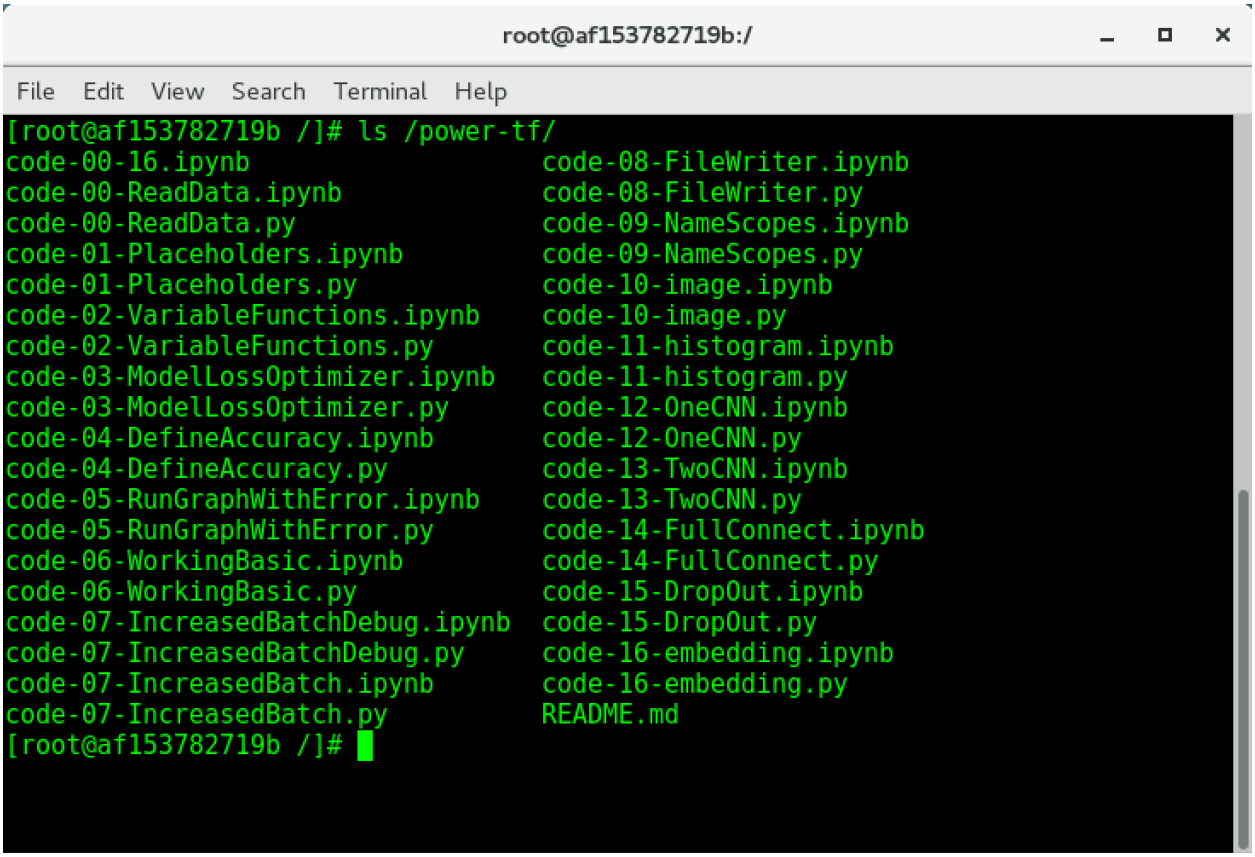
Note: If you don't enter 'y' within 5 seconds, the notebook server resumes. If this happens, just type Ctrl-C again and enter 'y' quicker than last time! :)

Git clone the python code for this lab

We've put the python code that we use in the lab section out on GitHub. Ensure you're in the root directory of your container, and issue a " git clone https://github.com/jjnash/power-tf" command":

```
[root@af153782719b /]# cd /
[root@af153782719b /]# git clone https://github.com/jjnash/power-tf
Cloning into 'power-tf'...
remote: Counting objects: 53, done.
remote: Total 53 (delta 0), reused 0 (delta 0), pack-reused 53
Unpacking objects: 100% (53/53), done.
[root@af153782719b /]#
```

The file we need for this lab are now in /power-tf



Configuring your Jupyter Notebook

Again for this lab section we'll have the option of using Jupyter Notebooks, but with a specific configuration. WIthin the **/files/PowerAILab** directory, there is a script for configuring your Jupyter session for this lab. Once this session is launched, you will point your browser to an address and open the notebook for each lab section which will step you through the lab you're doing.

Use **tensorflow-activate** to set appropriate environment variables **source**

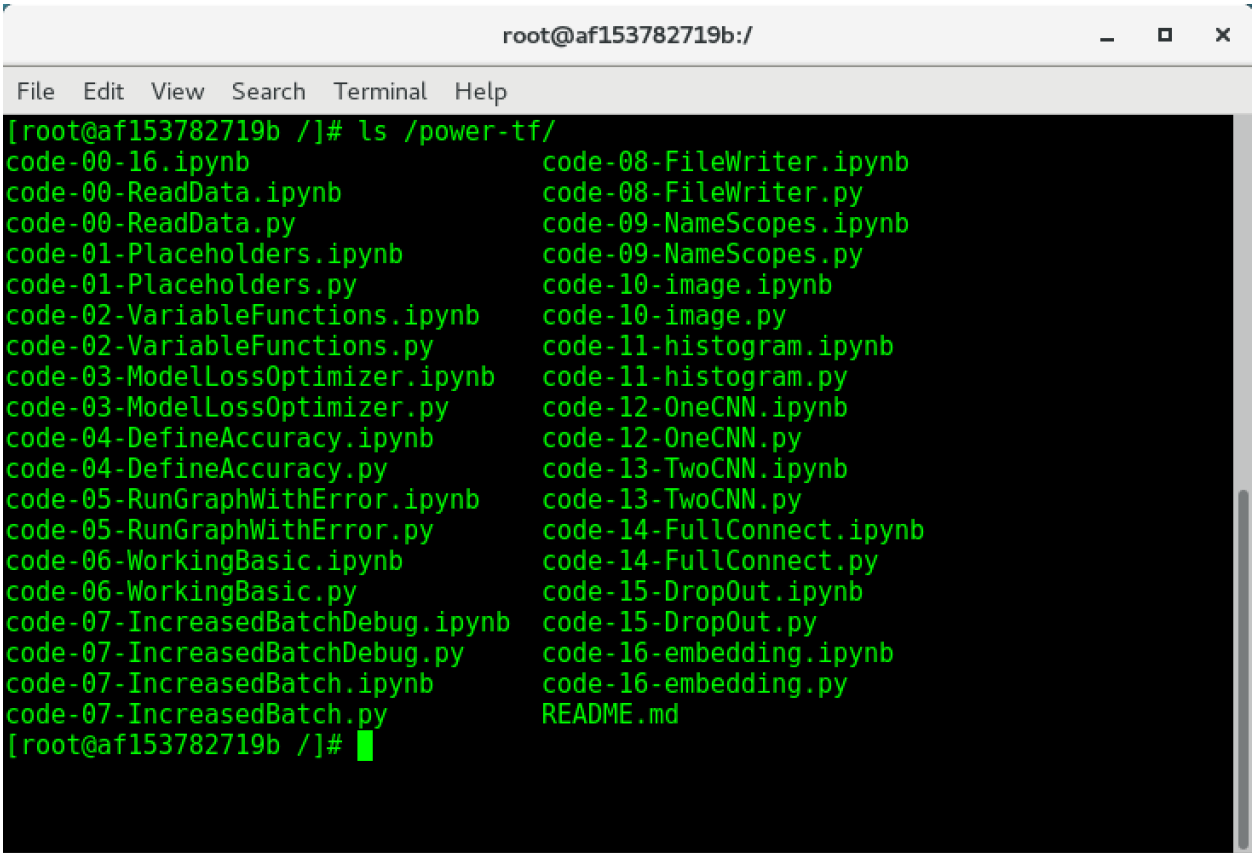
```
[root@af153782719b /]# source /opt/DL/tensorflow/bin/tensorflow-
activate
[root@af153782719b /]#
```

Run "**cfgjptr_mnist.sh 8890**" followed by "**source ~/.bashrc**":

Note: If you already have a Jupyter Notebook configuration file from another lab, you will be prompted to replace it.

```
[root@af153782719b /]# cfgjptr_mnist.sh  8890
Overwrite /root/.jupyter/jupyter_notebook_config.py with default config?
[y/N]y
Writing default config to: /root/.jupyter/jupyter_notebook_config.py
[root@af153782719b /]# source ~/.bashrc
[root@af153782719b /]#
```

There are 17 code samples (in two different formats) we will walk through each. We will discuss each progressively and end with **code-16-embedding.py**, a fully functional TF program.



You have **TWO** options for executing this lab section, (**choose one**).

Option 1: Execute sample code from the bash command-line.

```
[root@3fc208a32134 /]# cd /power-tf/
[root@3fc208a32134 power-tf]# python code-00-ReadData.py
[root@3fc208a32134 power-tf]# python code-01-Placeholders.py # etc …
```

Option 2: Execute sample code via a Jupyter Notebook

```
[root@3fc208a32134 power-tf]# jupyter notebook --no-browser --allow-root
[I 16:07:52.555 NotebookApp] Writing notebook server cookie secret to
/root/.local/share/jupyter/runtime/notebook_cookie_secret
[I 16:07:53.218 NotebookApp] JupyterLab beta preview extension loaded from
/usr/local/anaconda2/lib/python2.7/site-packages/jupyterlab
[I 16:07:53.219 NotebookApp] JupyterLab application directory is
/usr/local/anaconda2/share/jupyter/lab
[I 16:07:53.224 NotebookApp] Serving notebooks from local directory:
/power-tf
[I 16:07:53.224 NotebookApp] 0 active kernels
[I 16:07:53.224 NotebookApp] The Jupyter Notebook is running at:
[I 16:07:53.224 NotebookApp] http://0.0.0.0:8890/?token=...
[I 16:07:53.224 NotebookApp] Use Control-C to stop this server and shut
down all kernels (twice to skip confirmation).
```

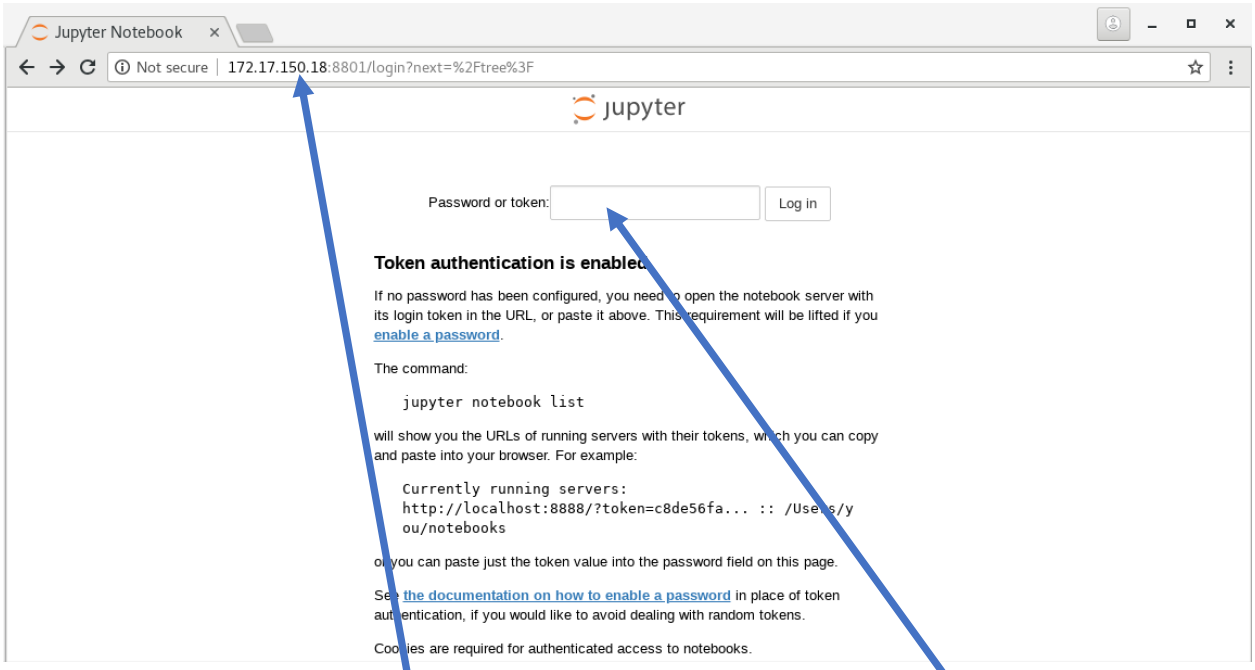
Leaving your SSH window open (with Jupyter running)

Continuation of Option 2:

Open a Browser and point it at your Jupyter Notebook

Our next steps will all be hosted from the Jupyter Notebook we've just brought up in our container.

In the URL field of your favorite Browser, enter your assigned IP address and port number:
**Hint the IP address and port number is on your slip



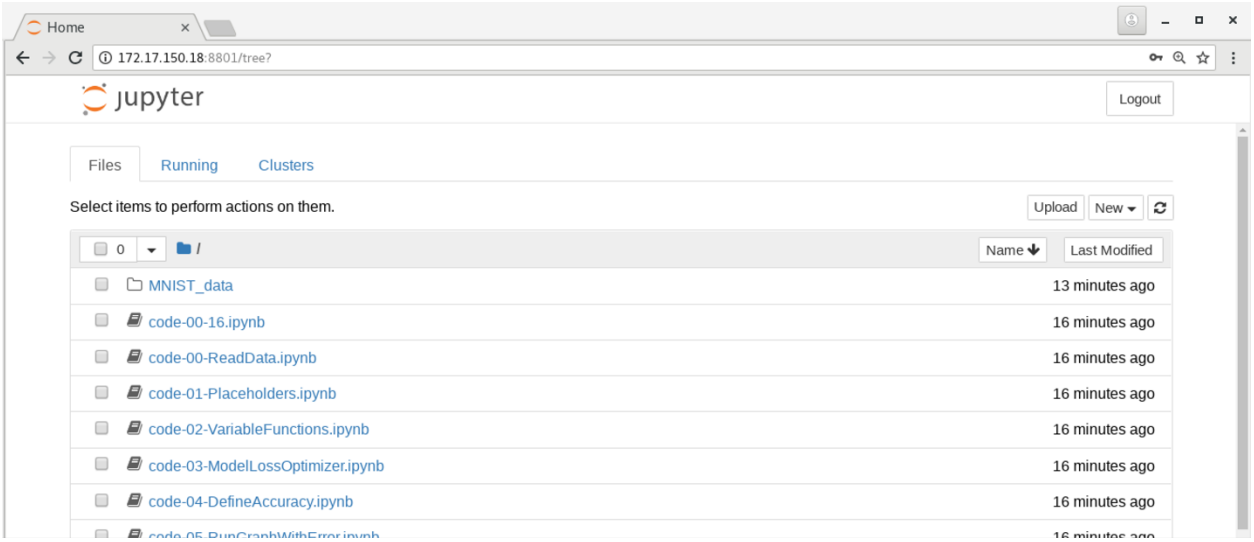
team01

Description	IP Address	UserID	Password
SimCenter AC922 Server	172.17.150.18	team01	abcd1234
Jupyter Notebook	http://172.17.150.18:8801	N/A	abcd1234
TensorBoard	http://172.17.150.18:6001	N/A	N/A

You'll be prompted for a password or token, this is also on your slip!

After entering (your super-secret password abcd1234) you will get a page with a link for the Lab Jupyter Notebook. Hurray!!

You'll see the Jupyter Notebook home page which looks like this, scroll down to find each version of the code starting with **code-00-16.ipynb,** and click it:



This Notebook has all 16 version of the Python program in individual ipynb notebooks, allowing us to step through each revision.

Brief details explaining Jupyer:

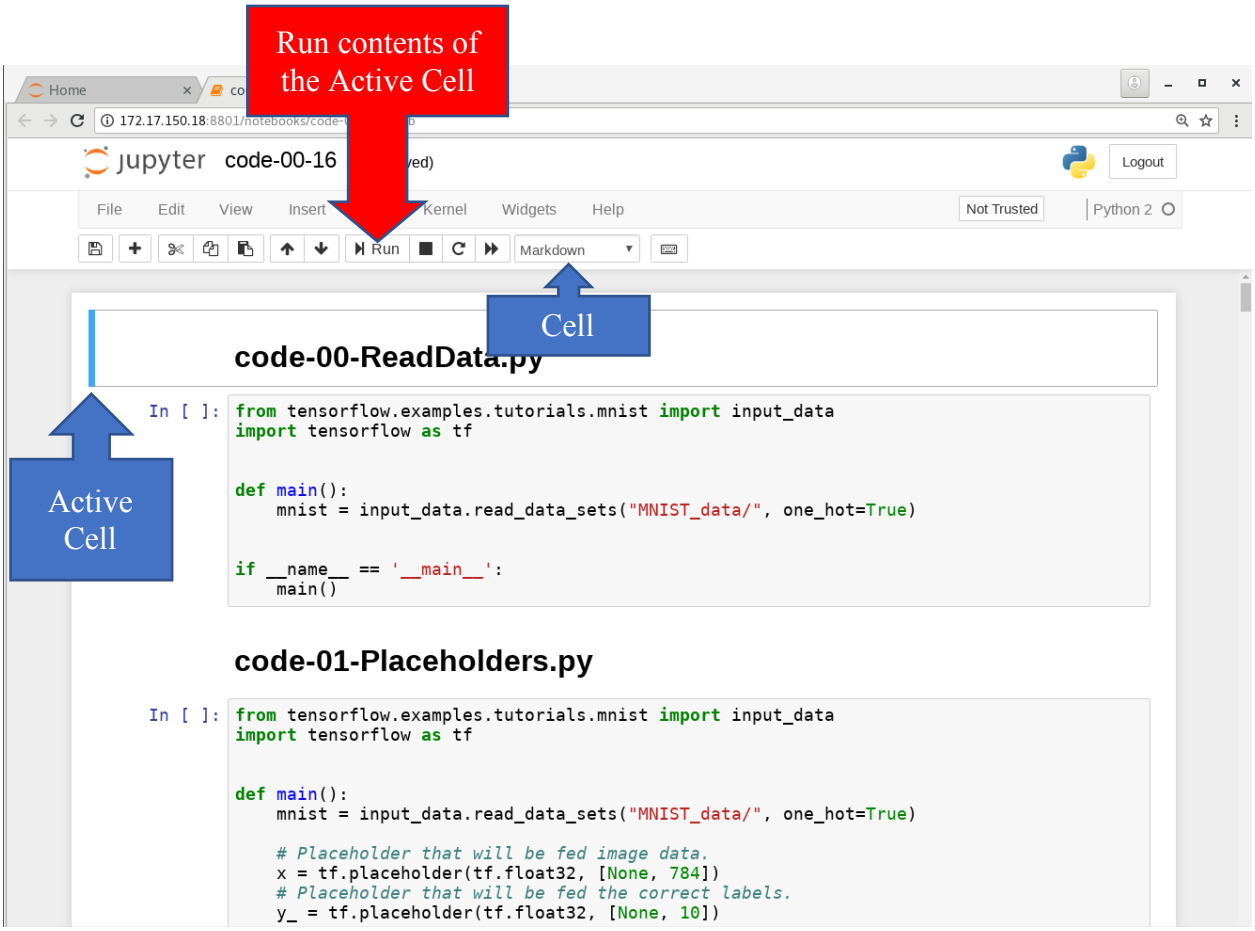
The ipynb files in /opt/DL/tensorflow are IPython Notebook files created for use by Jupyter notebooks. These notebook files are in JSON format to be created, viewed, edited, manipulated,and run in the Jupyter client within a web browser. Notebook files can contain markup code as well as python commands.

If you're interested in what ipynb files look like, you can open another ssh session to your container and inspect 00-classification.ipynb. For additional information on these 'notebook' files, take a look at:

https://github.com/tritemio/jupyter_notebook_beginner_guide

Note, however, that .ipynb files are not typically created by hand; they are created using the Jupyter Notebook.

In the screen shot below, the blue bar on the left highlights the active cell you are in, and to the right of the control buttons along the top, you can see that this particular cell is a 'Markdown' cell, which is really just a text cell used to document things in the notebook. The run button will step through code, section by section.



Regardless of Option 1 or Option 2 we will now execute some Python code:

code-00-ReadData.py / code-00-ReadData.ipynb

```
[root@3fc208a32134 /]# cd /power-tf/
[root@3fc208a32134 power-tf]# python code-00-ReadData.py
```

or

Click  **in the Jupyter Notebook**


```
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf


def main():
    mnist = input_data.read_data_sets("MNIST_data/",
one_hot=True)
```

This imports MNIST data so it can be accessed and used by TensorFlow.

code-01-Placeholders.py

(New code added is displayed in **BOLD**)

This new code defines a Placeholder for input: image and label

```
import input_data
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf


def main():
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

    # Placeholder that will be fed image data.
    x = tf.placeholder(tf.float32, [None, 784])
    # Placeholder that will be fed the correct labels.
    y_ = tf.placeholder(tf.float32, [None, 10])


if __name__ == '__main__':
    main()
```

code-02-VariableFuctions.py / code-02-VariableFuctions.ipynb

Define Variables for model – weight and bias:

```
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf


def weight_variable(shape):
  """Generates a weight variable of a given shape."""
  initial = tf.truncated_normal(shape, stddev=0.1)
  return tf.Variable(initial)


def bias_variable(shape):
  """Generates a bias variable of a given shape."""
  initial = tf.constant(0.1, shape=shape)
  return tf.Variable(initial)


def main():
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

    # Placeholder that will be fed image data.
    x = tf.placeholder(tf.float32, [None, 784])
    # Placeholder that will be fed the correct labels.
    y_ = tf.placeholder(tf.float32, [None, 10])

    # Define weight and bias.
    W = weight_variable([784, 10])
    b = bias_variable([10])


if __name__ == '__main__':
    main()
```

code-03-ModelLossOptimizer.py / code-03-ModelLossOptimizer.ipynb

Define Loss and Optimzer functions.

```
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf


def weight_variable(shape):
  """Generates a weight variable of a given shape."""
  initial = tf.truncated_normal(shape, stddev=0.1)
  return tf.Variable(initial)


def bias_variable(shape):
  """Generates a bias variable of a given shape."""
  initial = tf.constant(0.1, shape=shape)
  return tf.Variable(initial)


def main():
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

    # Placeholder that will be fed image data.
    x = tf.placeholder(tf.float32, [None, 784])
    # Placeholder that will be fed the correct labels.
    y_ = tf.placeholder(tf.float32, [None, 10])

    # Define weight and bias.
    W = weight_variable([784, 10])
    b = bias_variable([10])

    # Here we define our model which utilizes the softmax regression.
    y = tf.nn.softmax(tf.matmul(x, W) + b)

    # Define our loss.
    cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y),
reduction_indices=[1]))

    # Define our optimizer.
    train_step =
tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

if __name__ == '__main__':
    main()
```

code-04-DefineAccuracy.py / code-04-DefineAccuracy.ipynb

Add Accuracy Calculation.

```
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf

...

def main():
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

    # Placeholder that will be fed image data.
    x = tf.placeholder(tf.float32, [None, 784])
    # Placeholder that will be fed the correct labels.
    y_ = tf.placeholder(tf.float32, [None, 10])

    # Define weight and bias.
    W = weight_variable([784, 10])
    b = bias_variable([10])

    # Here we define our model which utilizes the softmax regression.
    y = tf.nn.softmax(tf.matmul(x, W) + b)

    # Define our loss.
    cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y),
reduction_indices=[1]))

    # Define our optimizer.
    train_step =
tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

    # Define accuracy.
    correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
    correct_prediction = tf.cast(correct_prediction, tf.float32)
    accuracy = tf.reduce_mean(correct_prediction)


if __name__ == '__main__':
    main()
```

code-05-RunGraphWithError.py / code-05-RunGraphWithError.ipynb

Connect to runtime and run a training graph.

```
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf

...
def main():
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
...

    # Define accuracy.
    correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
    correct_prediction = tf.cast(correct_prediction, tf.float32)
    accuracy = tf.reduce_mean(correct_prediction)

    # Launch session.
    sess = tf.InteractiveSession()

    # Do the training.
    for i in range(1100):
        batch = mnist.train.next_batch(100)
        sess.run(train_step, feed_dict={x: batch[0], y_: batch[1]})

    # See how model did.
    print("Test Accuracy %g" % sess.run(accuracy, feed_dict={x:
mnist.test.images, y_: mnist.test.labels}))


if __name__ == '__main__':
    main()
```

code-06-WorkingBasic.py / code-06-WorkingBasic.ipynb

Fix error: initialize variables

```
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf

...
def main():
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
...

    # Launch session.
    sess = tf.InteractiveSession()

    # Initialize variables.
    tf.global_variables_initializer().run()

    # Do the training.
    for i in range(1100):
        batch = mnist.train.next_batch(100)
        sess.run(train_step, feed_dict={x: batch[0], y_: batch[1]})

    # See how model did.
    print("Test Accuracy %g" % sess.run(accuracy, feed_dict={x:
mnist.test.images, y_: mnist.test.labels}))


if __name__ == '__main__':
    main()
```

code-07-IncreasedBatch.py / code-07-IncreasedBatch.ipynb

Try a larger batch of images.

```
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf

...
def main():
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
...

# Launch session.
    sess = tf.InteractiveSession()

    # Initialize variables.
    tf.global_variables_initializer().run()

    # Do the training.
    for i in range(1100):
        batch = mnist.train.next_batch(100)
        if i % 100 == 0:
            train_accuracy = sess.run(accuracy, feed_dict={x:batch[0], y_:
batch[1]})
            print("Step %d, Training Accuracy %g" % (i,
float(train_accuracy)))
        sess.run(train_step, feed_dict={x: batch[0], y_: batch[1]})

    # See how model did.
    print("Test Accuracy %g" % sess.run(accuracy, feed_dict={x:
mnist.test.images,
                                                              y_:
mnist.test.labels}))


if __name__ == '__main__':
    main()
```

code-08-FileWriter.py / code-08-FileWriter.ipynb

Add FileWriter to visualize with TensorBoard

```
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf

LOGDIR = './tensorflow_logs/mnist_deep'

...
def main():
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
...

    # Launch session.
    sess = tf.InteractiveSession()

    # Initialize variables.
    tf.global_variables_initializer().run()

    # Create summary writer
    writer = tf.summary.FileWriter(LOGDIR, sess.graph)

    # Do the training.
    for i in range(1100):
        batch = mnist.train.next_batch(100)
        if i % 100 == 0:
            train_accuracy = sess.run(accuracy, feed_dict={x:batch[0], y_:
batch[1]})
            print("Step %d, Training Accuracy %g" % (i,
float(train_accuracy)))
        sess.run(train_step, feed_dict={x: batch[0], y_: batch[1]})

    # See how model did.
    print("Test Accuracy %g" % sess.run(accuracy, feed_dict={x:
mnist.test.images, y_: mnist.test.labels}))

    # Close summary writer
    writer.close()


if __name__ == '__main__':
    main()
```

code-09-NameScopes.py / code-09-NameScopes.ipynb

Add names and name scope to make it easier to read the graph.

```
...
def weight_variable(shape):
  """Generates a weight variable of a given shape."""
  initial = tf.truncated_normal(shape, stddev=0.1)
  return tf.Variable(initial, name='weight')


def bias_variable(shape):
  """Generates a bias variable of a given shape."""
  initial = tf.constant(0.1, shape=shape)
  return tf.Variable(initial, name='bias')


def main():
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

    # Placeholder that will be fed image data.
    x = tf.placeholder(tf.float32, [None, 784], name='x')
    # Placeholder that will be fed the correct labels.
    y_ = tf.placeholder(tf.float32, [None, 10], name='labels')

    # Define weight and bias.
    W = weight_variable([784, 10])
    b = bias_variable([10])

    # Here we define our model which utilizes the softmax regression.
    with tf.name_scope('softmax'):
        y = tf.nn.softmax(tf.matmul(x, W) + b, name='y')

    # Define our loss.
    with tf.name_scope('loss'):
        cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y),
reduction_indices=[1]), name='cross_entropy')

    # Define our optimizer.
    with tf.name_scope('optimizer'):
        train_step =
tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy,
name='train_step')

    # Define accuracy.
    with tf.name_scope('accuracy'):
        correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
        correct_prediction = tf.cast(correct_prediction, tf.float32,
name='correct_prediction')
        accuracy = tf.reduce_mean(correct_prediction, name='accuracy')
...
```

code-10-image.py / code-10-image.ipynb

Viewing images in TensorBoard

```
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf

...
def main():
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
...

    # Define weight and bias.
    W = weight_variable([784, 10])
    b = bias_variable([10])

    with tf.name_scope('reshape'):
        x_image = tf.reshape(x, [-1, 28, 28, 1])
        tf.summary.image('input', x_image, 4)

    # Here we define our model which utilizes the softmax regression.
    with tf.name_scope('softmax'):
        y = tf.nn.softmax(tf.matmul(x, W) + b, name='y')

...
# Initialize variables.
    tf.global_variables_initializer().run()

    # Merge all the summary data
    merged = tf.summary.merge_all()

    # Create summary writer
    writer = tf.summary.FileWriter(LOGDIR, sess.graph)

    # Do the training.
    for i in range(1100):
        batch = mnist.train.next_batch(100)
        if i % 5 == 0:
            summary = sess.run(merged, feed_dict={x: batch[0], y_:
batch[1]})
            writer.add_summary(summary, i)
        if i % 100 == 0:
...
```

code-11-histogram.py / code-11-histogram.ipynb

View line graphs and histograms of variables

```
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf

...
def main():
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
...

    # Define weight and bias.
    W = weight_variable([784, 10])
    tf.summary.histogram('weight', W)
    b = bias_variable([10])
    tf.summary.histogram('bias', b)
...

    # Here we define our model which utilizes the softmax regression.
    with tf.name_scope('softmax'):
        y = tf.nn.softmax(tf.matmul(x, W) + b, name='y')
        tf.summary.histogram('softmax', y)

    # Define our loss.
    with tf.name_scope('loss'):
        cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y),
reduction_indices=[1]), name='cross_entropy')
        tf.summary.scalar('loss', cross_entropy)
...

    # Define accuracy.
    with tf.name_scope('accuracy'):
        correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
        correct_prediction = tf.cast(correct_prediction, tf.float32,
name='correct_prediction')
        accuracy = tf.reduce_mean(correct_prediction, name='accuracy')
        tf.summary.scalar('accuracy', accuracy)

    # Launch session.
    sess = tf.InteractiveSession()
...
```

code-12-OneCNN.py / code-12-OneCNN.ipynb

Create the first Convolutional Layer in the Neural Network (CNN)

```
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf

...
def main():
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

    # Placeholder that will be fed image data.
    x = tf.placeholder(tf.float32, [None, 784], name='x')
    # Placeholder that will be fed the correct labels.
    y_ = tf.placeholder(tf.float32, [None, 10], name='labels')

    # Define weight and bias.
    W = weight_variable([784, 10])
    tf.summary.histogram('weight', W)
    b = bias_variable([10])
    tf.summary.histogram('bias', b)

    # Reshape to use within a convolutional neural net.
    # Last dimension is for "features" - there is only one here, since
images are
    # grayscale -- it would be 3 for an RGB image, 4 for RGBA, etc.
    with tf.name_scope('reshape'):
        x_image = tf.reshape(x, [-1, 28, 28, 1])
        tf.summary.image('input', x_image, 4)

    # Convolutional layer - maps one grayscale image to 32 features.
    with tf.name_scope('conv1'):
        W_conv1 = weight_variable([5, 5, 1, 32])
        b_conv1 = bias_variable([32])
        x_conv1 = tf.nn.conv2d(x_image, W_conv1, strides=[1, 1, 1, 1],
padding='SAME')
        h_conv1 = tf.nn.relu(x_conv1 + b_conv1)

    # Pooling layer - downsamples by 2X.
    with tf.name_scope('pool1'):
        h_pool1 = tf.nn.max_pool(h_conv1, ksize=[1, 2, 2, 1],
                                 strides=[1, 2, 2, 1], padding='SAME')

    # After downsampling, our 28x28 image is now 14x14
    # with 32 feature maps.
    with tf.name_scope('flatten'):
        h_pool_flat = tf.reshape(h_pool1, [-1, 14*14*32])

    # Map the features to 10 classes, one for each digit
    with tf.name_scope('fc-classify'):

        W_fc2 = weight_variable([14*14*32, 10])
        b_fc2 = bias_variable([10])
        y = tf.matmul(h_pool_flat, W_fc2) + b_fc2
```

```
    # Here we define our model which utilizes the softmax regression.
    with tf.name_scope('softmax'):
        y = tf.nn.softmax(tf.matmul(x, W) + b, name='y')
        tf.summary.histogram('softmax', y)

    ############################
    # Define our loss.
    with tf.name_scope('loss'):
        # Use more numerically stable cross entropy.
        cross_entropy = tf.reduce_mean(
            tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y),
            name='cross_entropy'
        )
        tf.summary.scalar('loss', cross_entropy)

    # Define our optimizer.
    with tf.name_scope('optimizer'):
        train_step =
tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy,
name='train_step')

    # Define accuracy.
    with tf.name_scope('accuracy'):
        correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
        correct_prediction = tf.cast(correct_prediction, tf.float32,
name='correct_prediction')
        accuracy = tf.reduce_mean(correct_prediction, name='accuracy')
        tf.summary.scalar('accuracy', accuracy)

...
```

code-13-TwoCNN.py / code-13-TwoCNN.ipynb

Create the second Convolution Layer in the Neural Network (CNN)

```
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf

...
def main():
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
...

    # Pooling layer - downsamples by 2X.
    with tf.name_scope('pool1'):
        h_pool1 = tf.nn.max_pool(h_conv1, ksize=[1, 2, 2, 1],
                                 strides=[1, 2, 2, 1], padding='SAME')

    # Second convolutional layer -- maps 32 feature maps to 64.
    with tf.name_scope('conv2'):
        W_conv2 = weight_variable([5, 5, 32, 64])
        b_conv2 = bias_variable([64])
        x_conv2 = tf.nn.conv2d(h_pool1, W_conv2, strides=[1, 1, 1, 1],
padding='SAME')
        h_conv2 = tf.nn.relu(x_conv2 + b_conv2)

    # Second pooling layer.
    with tf.name_scope('pool2'):
        h_pool2 = tf.nn.max_pool(h_conv2, ksize=[1, 2, 2, 1],
                                 strides=[1, 2, 2, 1], padding='SAME')

    # After 2 rounds of downsampling, our 28x28 image
    # is down to 7x7 with 64 feature maps.
    with tf.name_scope('flatten'):
        h_pool_flat = tf.reshape(h_pool2, [-1, 7*7*64])

    # Map the features to 10 classes, one for each digit
    with tf.name_scope('fc-classify'):
        W_fc2 = weight_variable([7*7*64, 10])
        b_fc2 = bias_variable([10])
        y = tf.matmul(h_pool_flat, W_fc2) + b_fc2

# Define our optimizer.
    with tf.name_scope('optimizer'):
        train_step =
tf.train.AdamOptimizer(0.0001).minimize(cross_entropy, name='train_step')
...
```

code-14-FullConnect.py / code-14-FullConnect.ipynb

Create the Fully connected layer in the Neural Network

```
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf

...
def main():
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
...

    # After 2 rounds of downsampling, our 28x28 image
    # is down to 7x7 with 64 feature maps.
    with tf.name_scope('fc1'):
        h_pool_flat = tf.reshape(h_pool2, [-1, 7*7*64])
        W_fc1 = weight_variable([7*7*64, 1024])
        b_fc1 = bias_variable([1024])
        h_fc1 = tf.nn.relu(tf.matmul(h_pool_flat, W_fc1) + b_fc1)

    # Map the features to 10 classes, one for each digit
    with tf.name_scope('fc-classify'):
        W_fc2 = weight_variable([1024, 10])
        b_fc2 = bias_variable([10])
        y = tf.matmul(h_fc1, W_fc2) + b_fc2
...
```

code-15DropOut.py / code-15DropOut.ipynb

Add the dropout layer in the neural network to control overfitting.

```
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf
...
def main():
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
...

# After 2 rounds of downsampling, our 28x28 image
    # is down to 7x7 with 64 feature maps.
    with tf.name_scope('fc1'):
        h_pool_flat = tf.reshape(h_pool2, [-1, 7*7*64])
        W_fc1 = weight_variable([7*7*64, 1024])
        b_fc1 = bias_variable([1024])
        h_fc1 = tf.nn.relu(tf.matmul(h_pool_flat, W_fc1) + b_fc1)

    # Dropout - controls the complexity of the model, prevents co-
adaptation of
    # features.
    with tf.name_scope('dropout'):
        keep_prob = tf.placeholder(tf.float32)
        h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

    # Map the features to 10 classes, one for each digit
    with tf.name_scope('fc-classify'):
        W_fc2 = weight_variable([1024, 10])
        b_fc2 = bias_variable([10])
        y = tf.matmul(h_fc1_drop, W_fc2) + b_fc2
...
    # Do the training.
    for i in range(1100):
        batch = mnist.train.next_batch(100)
        if i % 5 == 0:
            summary = sess.run(merged, feed_dict={x: batch[0], y_:
batch[1], keep_prob: 1.0})
            writer.add_summary(summary, i)
        if i % 100 == 0:
            train_accuracy = sess.run(accuracy, feed_dict={x:batch[0], y_:
batch[1], keep_prob: 1.0})
            print("Step %d, Training Accuracy %g" % (i,
float(train_accuracy)))
        sess.run(train_step, feed_dict={x: batch[0], y_: batch[1],
keep_prob: 0.5})

    # See how model did.
    print("Test Accuracy %g" % sess.run(accuracy, feed_dict={x:
mnist.test.images,
                                                              y_:
mnist.test.labels,
                                                              keep_prob:
1.0}))

...
```

code-16-embedding.py / code-16-embedding.ipynb

Add full visualization for all the layers.

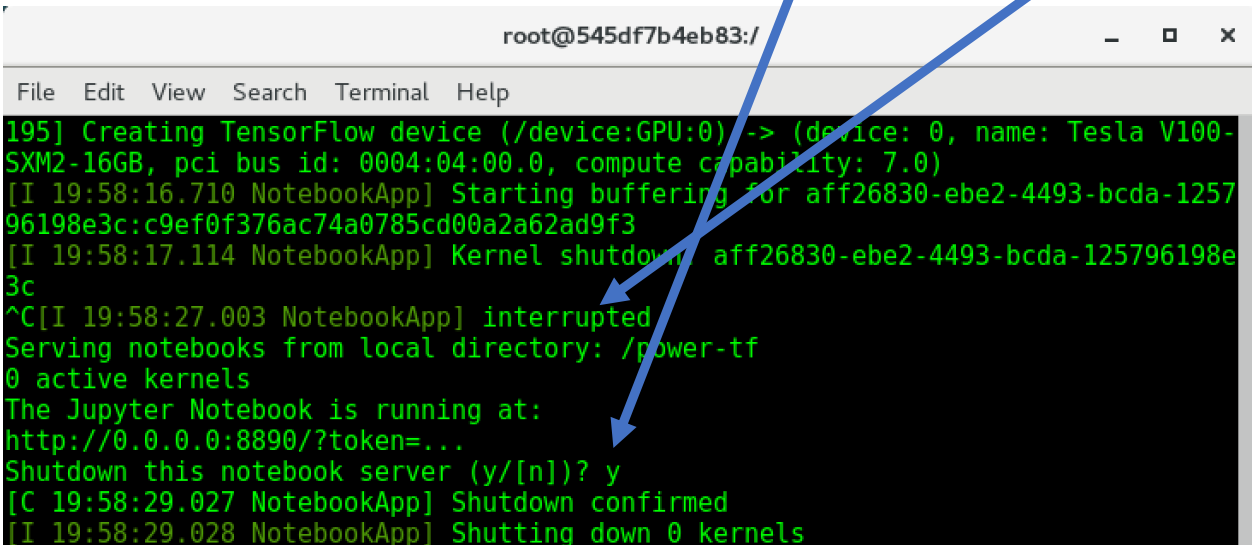
```
from tensorflow.examples.tutorials.mnist import input_data
import os
import tensorflow as tf
import sys
import urllib

if sys.version_info[0] >= 3:
  from urllib.request import urlretrieve
else:
  from urllib import urlretrieve

LOGDIR = './tensorflow_logs/mnist_deep'

...
def main():
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
...

# Convolutional layer - maps one grayscale image to 32 features.
    with tf.name_scope('conv1'):
        W_conv1 = weight_variable([5, 5, 1, 32])
        b_conv1 = bias_variable([32])
        x_conv1 = tf.nn.conv2d(x_image, W_conv1, strides=[1, 1, 1, 1],
padding='SAME')
        h_conv1 = tf.nn.relu(x_conv1 + b_conv1)
        tf.summary.histogram("weights", W_conv1)
        tf.summary.histogram("biases", b_conv1)
        tf.summary.histogram("activations", h_conv1)

    # Pooling layer - downsamples by 2X.
    with tf.name_scope('pool1'):
        h_pool1 = tf.nn.max_pool(h_conv1, ksize=[1, 2, 2, 1],
                                 strides=[1, 2, 2, 1], padding='SAME')
        # Display the image after max pooling on tensorboard
        h_pool1_image = tf.reshape(h_pool1, [-1, 14, 14, 1])
        tf.summary.image('conv1', h_pool1_image, 4)

    # Second convolutional layer -- maps 32 feature maps to 64.
    with tf.name_scope('conv2'):
        W_conv2 = weight_variable([5, 5, 32, 64])
        b_conv2 = bias_variable([64])
        x_conv2 = tf.nn.conv2d(h_pool1, W_conv2, strides=[1, 1, 1, 1],
padding='SAME')
        h_conv2 = tf.nn.relu(x_conv2 + b_conv2)
        tf.summary.histogram("weights", W_conv2)
        tf.summary.histogram("biases", b_conv2)
        tf.summary.histogram("activations", h_conv2)

    # Second pooling layer.
    with tf.name_scope('pool2'):
        h_pool2 = tf.nn.max_pool(h_conv2, ksize=[1, 2, 2, 1],
```

```
                        strides=[1, 2, 2, 1], padding='SAME')
        # Display the image after max pooling on tensorboard
        h_pool2_image = tf.reshape(h_pool2, [-1, 7, 7, 1])
        tf.summary.image('conv2', h_pool2_image, 4)

    # After 2 rounds of downsampling, our 28x28 image
    # is down to 7x7 with 64 feature maps.
    with tf.name_scope('fc1'):
        h_pool_flat = tf.reshape(h_pool2, [-1, 7*7*64])
        W_fc1 = weight_variable([7*7*64, 1024])
        b_fc1 = bias_variable([1024])
        h_fc1 = tf.nn.relu(tf.matmul(h_pool_flat, W_fc1) + b_fc1)
        tf.summary.histogram("weights", W_fc1)
        tf.summary.histogram("biases", b_fc1)
        tf.summary.histogram("activations", h_fc1)

...
    # Create summary writer
    writer = tf.summary.FileWriter(LOGDIR, sess.graph)

    # Get sprite and labels file for the embedding projector
    GITHUB_URL ='https://raw.githubusercontent.com/mamcgrath/TensorBoard-
TF-Dev-Summit-Tutorial/master/'
    urlretrieve(GITHUB_URL + 'labels_1024.tsv', os.path.join(LOGDIR,
'labels_1024.tsv'))
    urlretrieve(GITHUB_URL + 'sprite_1024.png', os.path.join(LOGDIR,
'sprite_1024.png'))

    # Setup embedding visualization
    embedding = tf.Variable(tf.zeros([1024, 1024]), name="test_embedding")
    assignment = embedding.assign(h_fc1_drop)
    saver = tf.train.Saver()

    config = tf.contrib.tensorboard.plugins.projector.ProjectorConfig()
    embedding_config = config.embeddings.add()
    embedding_config.tensor_name = embedding.name
    embedding_config.sprite.image_path = 'sprite_1024.png'
    embedding_config.metadata_path = 'labels_1024.tsv'
    # Specify the width and height of a single thumbnail.
    embedding_config.sprite.single_image_dim.extend([28, 28])
    tf.contrib.tensorboard.plugins.projector.visualize_embeddings(writer,
config)

    # Do the training.
    for i in range(1100):
        batch = mnist.train.next_batch(100)
        if i % 5 == 0:
            summary = sess.run(merged, feed_dict={x: batch[0], y_:
batch[1], keep_prob: 1.0})
            writer.add_summary(summary, i)
        if i % 100 == 0:
            train_accuracy = sess.run(accuracy, feed_dict={x:batch[0], y_:
batch[1], keep_prob: 1.0})
            print("Step %d, Training Accuracy %g" % (i,
float(train_accuracy)))
        if i % 500 == 0:
```

```
            sess.run(assignment, feed_dict={x: mnist.test.images[:1024],
y_: mnist.test.labels[:1024], keep_prob: 1.0
})
            saver.save(sess, os.path.join(LOGDIR, "model.ckpt"), i)
        sess.run(train_step, feed_dict={x: batch[0], y_: batch[1],
keep_prob: 0.5})

    # See how model did.
    print("Test Accuracy %g" % sess.run(accuracy, feed_dict={x:
mnist.test.images,
                                                              y_:
mnist.test.labels,
                                                              keep_prob:
1.0}))

    # Close summary writer
    writer.close()


if __name__ == '__main__':
    main()
```

Conclude this Section

Find your ssh session where you started the Jupyter Notebook server, and type Ctrl-C to interrupt the notebook server. Then immediately enter 'y' to shutdown the notebook server:



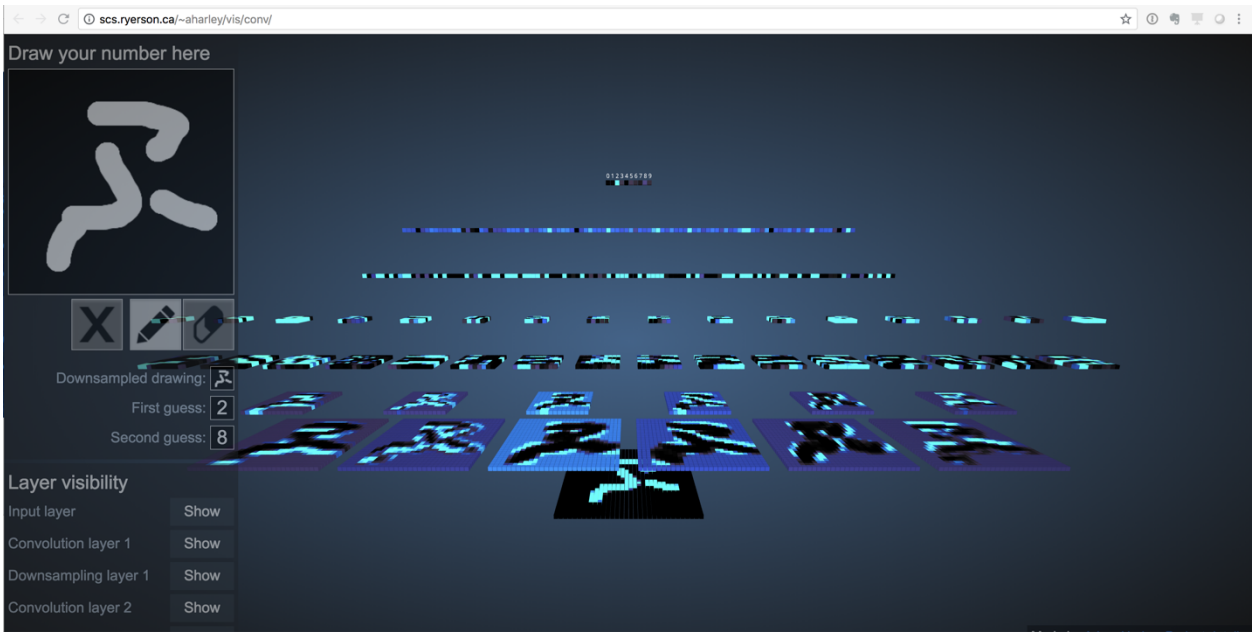
Note: If you don't enter 'y' within 5 seconds, the notebook server resumes. If this happens, just type Ctrl-C again and enter 'y' quicker than last time! :)

Do Not enter 'exit' to terminate your container, you'll need the files you create in /power-tf/tensorflow_logs/mnist_deep in **code-16-embedding.py / code-16-embedding.ipynb** for the next lab, **Section 4 TensorBoard – Visualization.**

Further Exploration

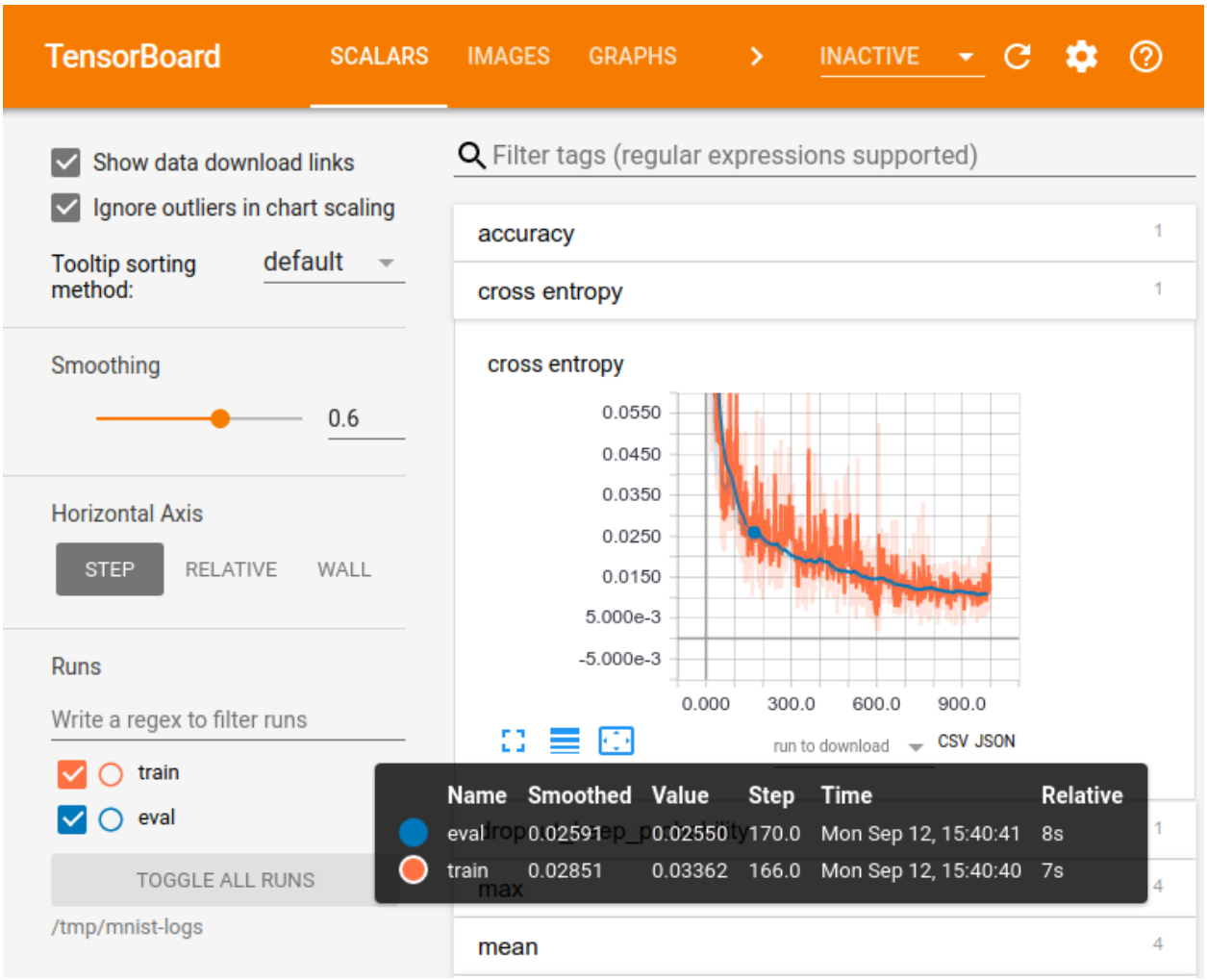
Here's an example of using the MNIST data set in an interactive way. Give it a try!

http://scs.ryerson.ca/~aharley/vis/conv/



Section 4 TensorBoard – Visualization

The computations you'll use TensorFlow for - like training a massive deep neural network - can be complex and confusing. To make it easier to understand, debug, and optimize TensorFlow programs, they've included a suite of visualization tools called TensorBoard. You can use TensorBoard to visualize your TensorFlow graph, plot quantitative metrics about the execution of your graph, and show additional data like images that pass through it. When TensorBoard is fully configured, it looks like this:



There is a 30-minute tutorial is intended to get you started with simple TensorBoard usage. It assumes a basic understanding of TensorFlow:

https://www.youtube.com/watch?time_continue=4&v=eBbEDRsCmv4

There are other resources available as well! The TensorBoard GitHub has a lot more information on using individual dashboards within TensorBoard including tips & tricks and debugging information.

Installing TensorFlow via pip should also automatically install TensorBoard.

TensorBoard operates by reading TensorFlow events files, which contain summary data that you can generate when running TensorFlow. Here's the general lifecycle for summary data within TensorBoard.

First, create the TensorFlow graph that you'd like to collect summary data from, and decide which nodes you would like to annotate with summary operations.

For example, suppose you are training a convolutional neural network for recognizing MNIST digits. You'd like to record how the learning rate varies over time, and how the objective function is changing. Collect these by attaching tf.summary.scalar ops to the nodes that output the learning rate and loss respectively. Then, give each scalar_summary a meaningful tag, like 'learning rate' or 'loss function'.

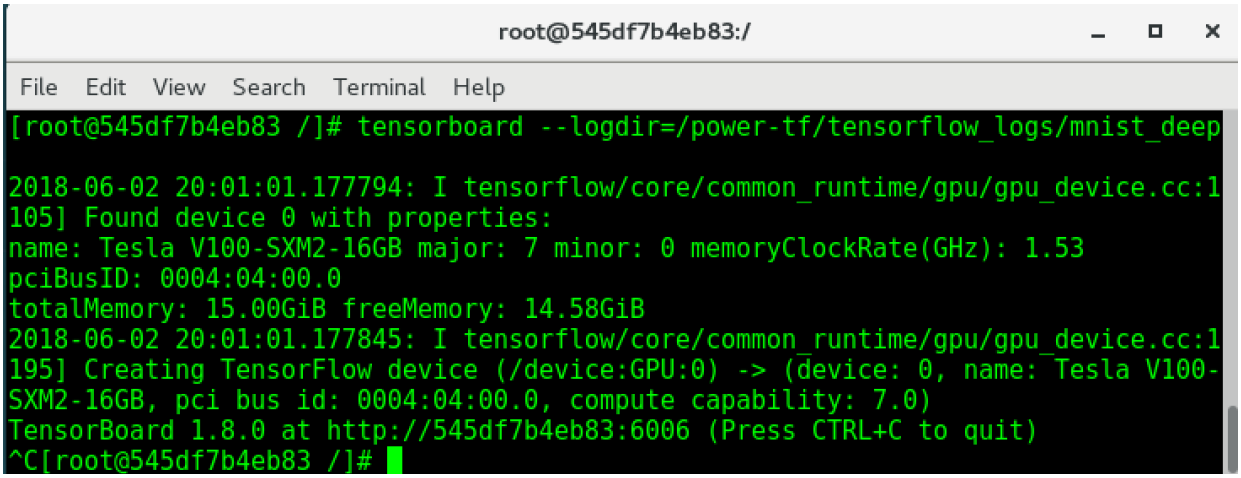
In **Section 3 Using MNIST and TensorFlow** you included **tf.summary.histogram()** calls in **code-16-embedding.py / code-16-embedding.ipynb** which wrote gradient and weights outputs to the

/power-tf/tensorflow_logs/mnist_deep

directory, so you'll be able to visualize statistics, such as how the weights or accuracy varied during training.

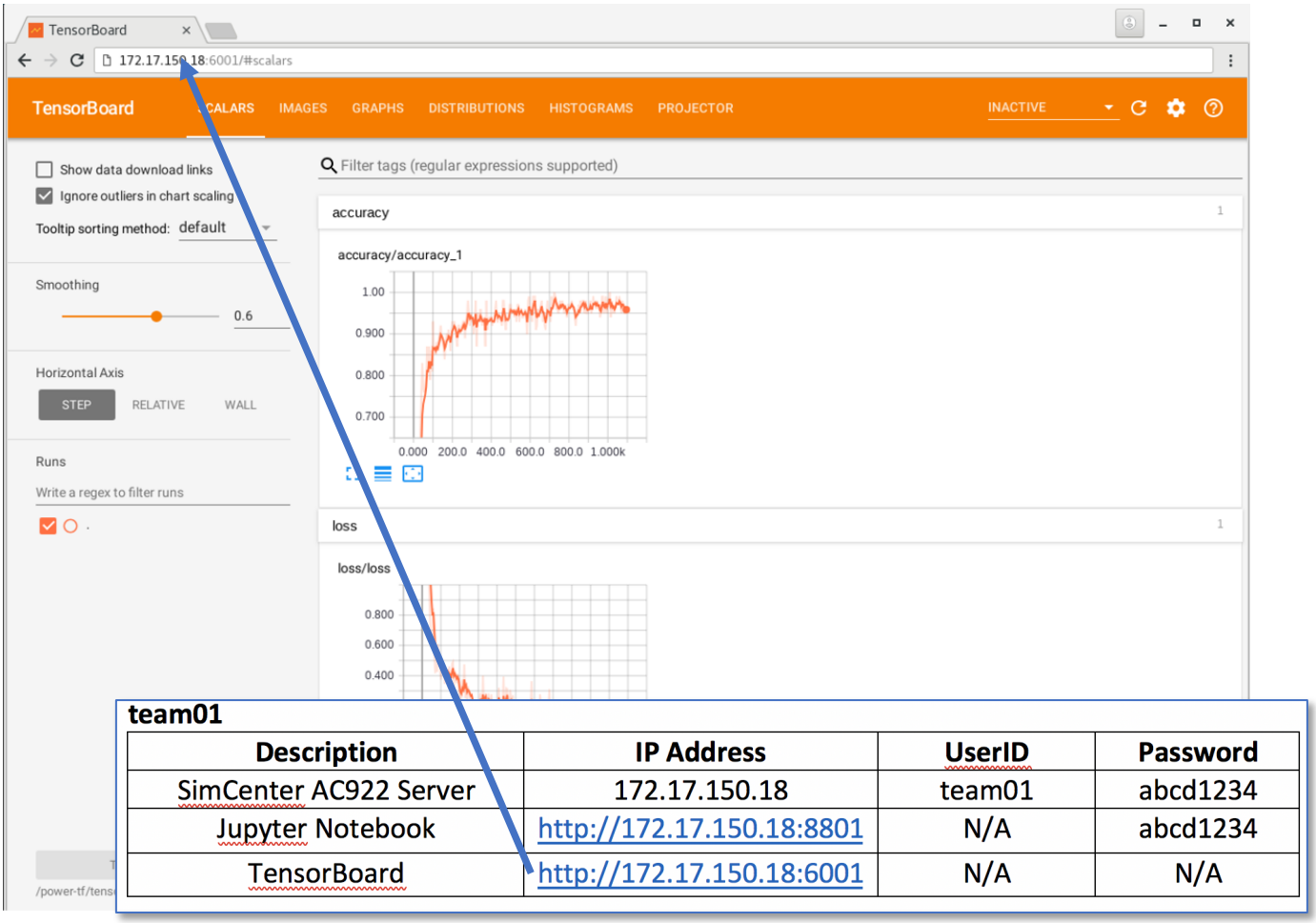
In the same ssh session you used in the prior lab section, issue a **tensorboard --logdir=/power-tf/tensorflow_logs/mnist_deep** command:

```
[root@545df7b4eb83 /]# tensorboard --logdir=/power-
tf/tensorflow_logs/mnist_deep
2018-06-02 20:01:01.177794: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1105] Found device 0 with
properties:
name: Tesla V100-SXM2-16GB major: 7 minor: 0 memoryClockRate(GHz): 1.53
pciBusID: 0004:04:00.0
totalMemory: 15.00GiB freeMemory: 14.58GiB
2018-06-02 20:01:01.177845: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1195] Creating TensorFlow
device (/device:GPU:0) -> (device: 0, name: Tesla V100-SXM2-16GB, pci bus
id: 0004:04:00.0, compute capability: 7.0)
TensorBoard 1.8.0 at http://545df7b4eb83:6006 (Press CTRL+C to quit)
```



Open a Browser and point it at TensorBoard

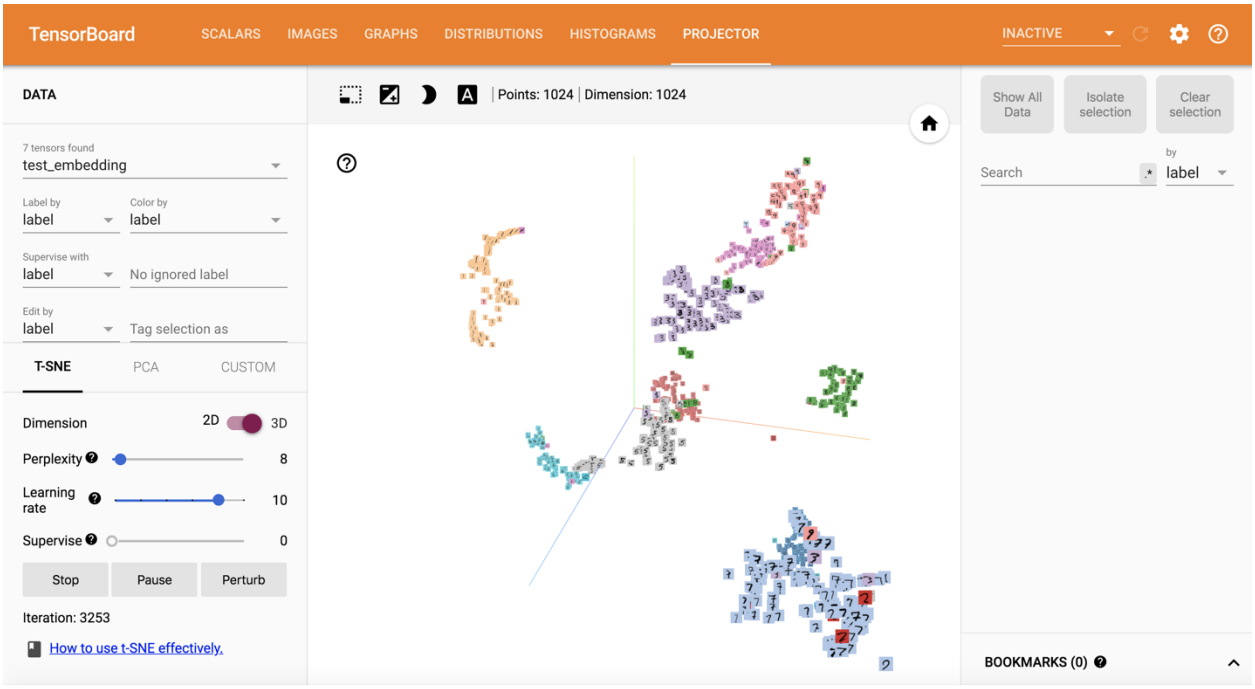
In the URL field of your favorite Browser, enter your assigned IP address and port number: **Hint the IP address and port number is on your slip



team01

Description	IP Address	UserID	Password
SimCenter AC922 Server	172.17.150.18	team01	abcd1234
Jupyter Notebook	http://172.17.150.18:8801	N/A	abcd1234
TensorBoard	http://172.17.150.18:6001	N/A	N/A

Look at the various tabs, there's an incredible amount of information available. We'll have a discussion and take of TensorBoard as time permits.

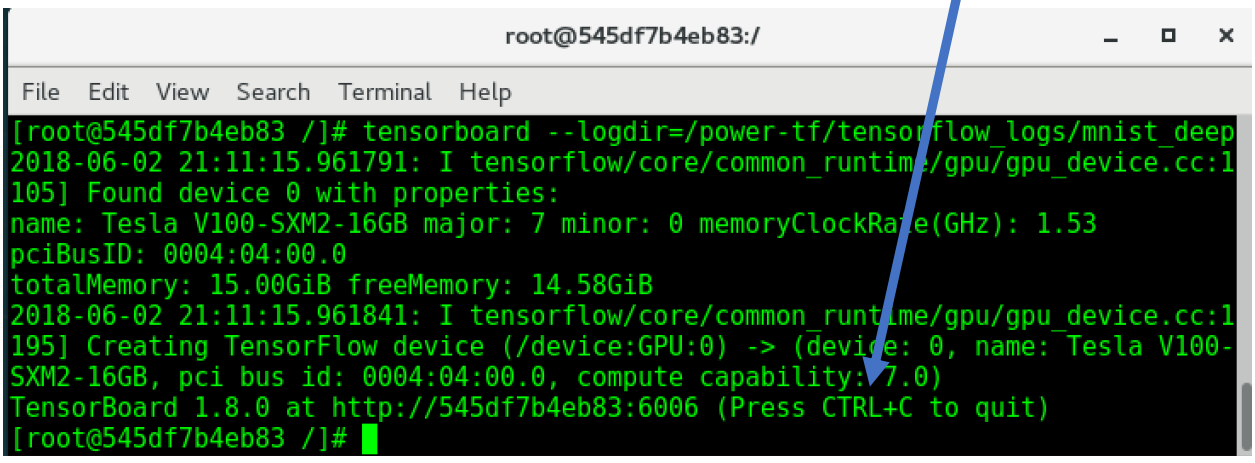


You can also get much more detail here:

https://www.tensorflow.org/get_started/summaries_and_tensorboard

Conclude this Section

Find your ssh session where you started Ten sorBoard, and type Ctrl-C to interrupt the notebook server.



Optionally, enter 'exit' to terminate your container, and then enter 'exit' again to end your ssh session. If you'll be doing additional lab sections, no need to exit your container

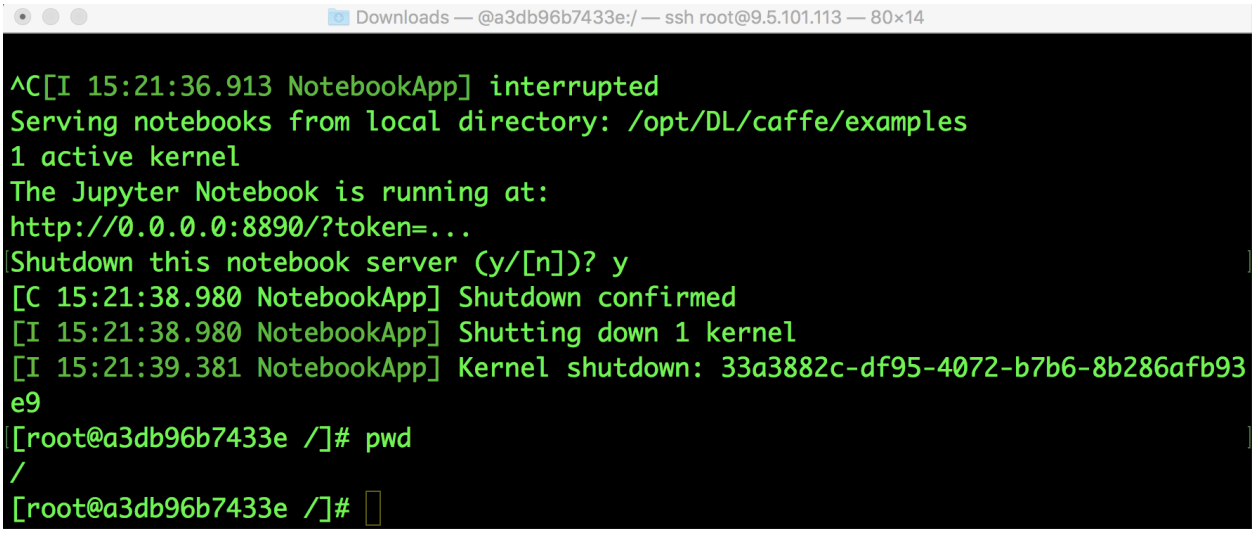
This concludes **Section 4 TensorBoard – Visualization**. In this section you learned a bit TensorBoard; you used it to get deeper insight into the execution characteristics of the model you built in **Section 3 Using MNIST and TensorFlow code-16-embedding.py / code-16-embedding.ipynb**.

Section 5 Tensorflow-Deep Learning to Solve Titanic

This lab is intended to give users an example of normal (albiet simple) process by which one could use TensorFlow to help identify handwritten characters. This will also expose you to TensorBoard which could be used to visualize or troubleshoot potential problems.

If you're currently logged into your container, proceed below! If not, then use the steps in Section 1 to log in and start your container. If your running a Jupyter Notebook from a prior section, cancel it with Ctrl-C to interrupt the notebook server. Then immediately enter 'y' to shutdown the notebook server.

Note: If you don't enter 'y' within 5 seconds, the notebook server resumes. If this happens, just type Ctrl-C again and enter 'y' quicker than last time! :)



Git clone the python code for this lab

We've put the python code that we use in the lab section out on GitHub. Ensure you're in the root directory of your container, and issue a " git clone https://github.com/jjnash/titanic" command":

```
[root@a3db96b7433e /]# cd /
[root@a3db96b7433e /]# git clone https://github.com/jjnash/titanic
Cloning into 'titanic'...
remote: Counting objects: 30, done.
remote: Compressing objects: 100% (25/25), done.
remote: Total 30 (delta 8), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (30/30), done.
[root@a3db96b7433e /]#
```

The files we need for this lab are now in /power-tf

```
[root@545df7b4eb83 /]# ls /titanic/notebooks/
notebook.ipynb
[root@545df7b4eb83 /]#
```

Configuring your Jupyter Notebook

Again for this lab section we'll have the option of using Jupyter Notebooks, but with a specific configuration. WIthin the **/files/PowerAILab** directory, there is a script for configuring your Jupyter session for this lab. Once this session is launched, you will point your browser to an address and open the notebook for each lab section which will step you through the lab you're doing.

Use **tensorflow-activate** to set appropriate environment variables **source**

```
[root@af153782719b /]# source /opt/DL/tensorflow/bin/tensorflow-
activate
[root@af153782719b /]#
```

Run "**cfgjptr_titanic.sh 8890**" followed by "**source ~/.bashrc**":

Note: If you already have a Jupyter Notebook configuration file from another lab, you will be prompted to replace it.

```
[root@a3db96b7433e /]# cfgjptr_titanic.sh  8890
Overwrite /root/.jupyter/jupyter_notebook_config.py with default config?
[y/N]y
Writing default config to: /root/.jupyter/jupyter_notebook_config.py
[root@a3db96b7433e /]#

[root@68715c8c0974 PowerAILab]# source ~/.bashrc
[root@68715c8c0974 PowerAILab]#
```

Start a Jupyter Notebook

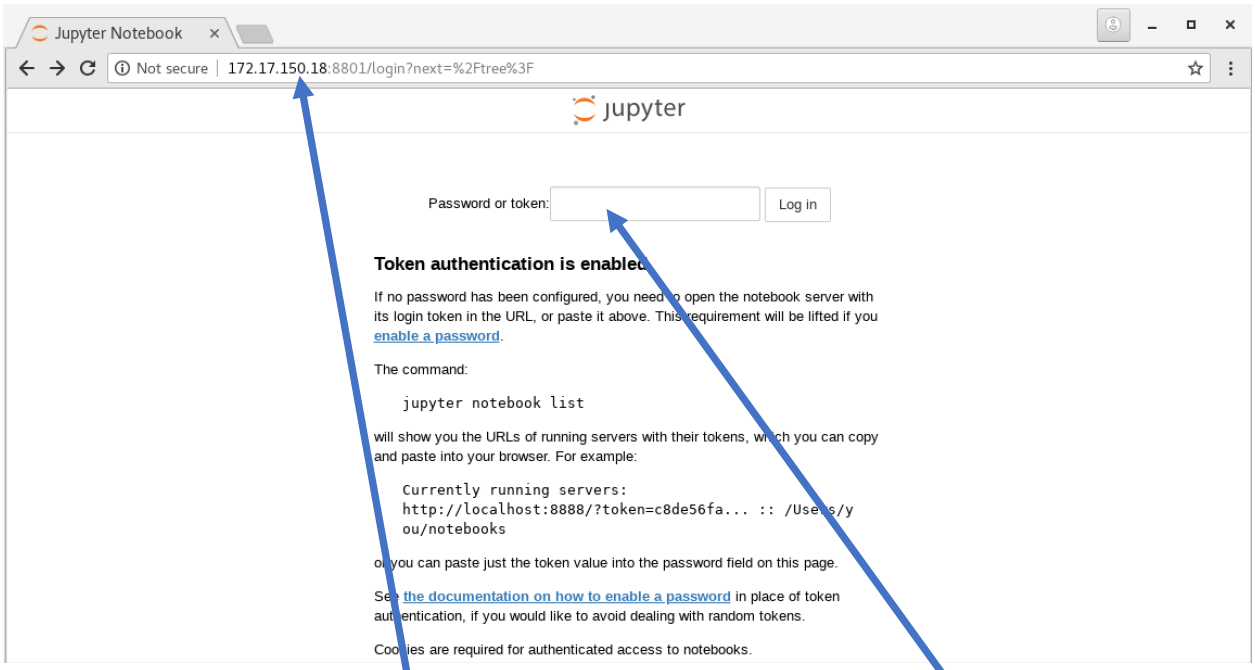
```
[root@545df7b4eb83 /]# jupyter notebook --no-browser --allow-root
[I 22:18:40.591 NotebookApp] JupyterLab beta preview extension loaded from
/root/anaconda2/lib/python2.7/site-packages/jupyterlab
[I 22:18:40.591 NotebookApp] JupyterLab application directory is
/root/anaconda2/share/jupyter/lab
[I 22:18:40.596 NotebookApp] Serving notebooks from local directory:
/titanic
[I 22:18:40.596 NotebookApp] 0 active kernels
[I 22:18:40.596 NotebookApp] The Jupyter Notebook is running at:
[I 22:18:40.596 NotebookApp] http://0.0.0.0:8890/?token=...
[I 22:18:40.596 NotebookApp] Use Control-C to stop this server and shut
down all kernels (twice to skip confirmation).
```

Open a Browser and point it at your Jupyter Notebook

Our next steps will all be hosted from the Jupyter Notebook we've just brought up in our container.

In the URL field of your favorite Browser, enter your assigned IP address and port number:

**Hint the IP address and port number is on your slip



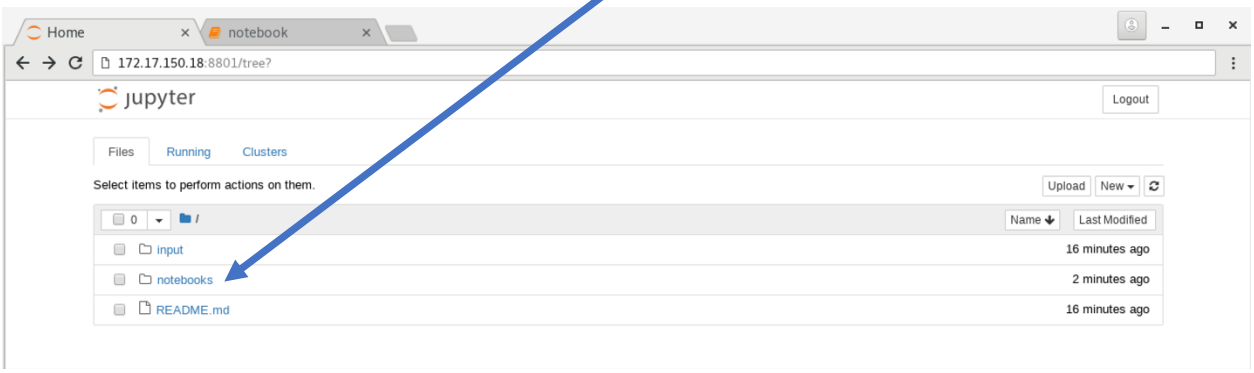
team01

Description	IP Address	UserID	Password
SimCenter AC922 Server	172.17.150.18	team01	abcd1234
Jupyter Notebook	http://172.17.150.18:8801	N/A	abcd1234
TensorBoard	http://172.17.150.18:6001	N/A	N/A

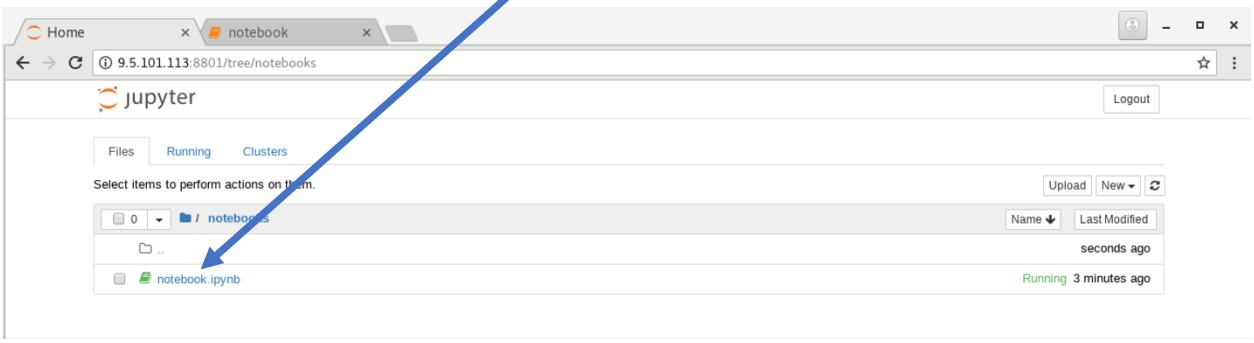
You'll be prompted for a password or token, this is also on your slip!

After entering (your super-secret password abcd1234) you will get a page with a link for the Lab Jupyter Notebook. Hurray!!

You'll see the Jupyter Notebook home page which looks like this, scroll down to find each version of the code starting with **notebooks,** and click it:



You'll see the Jupyter Notebook page which looks like this, scroll down to find each version of the code starting with **notebook.ipynb,** and click it:



Brief details explaining Jupyer:

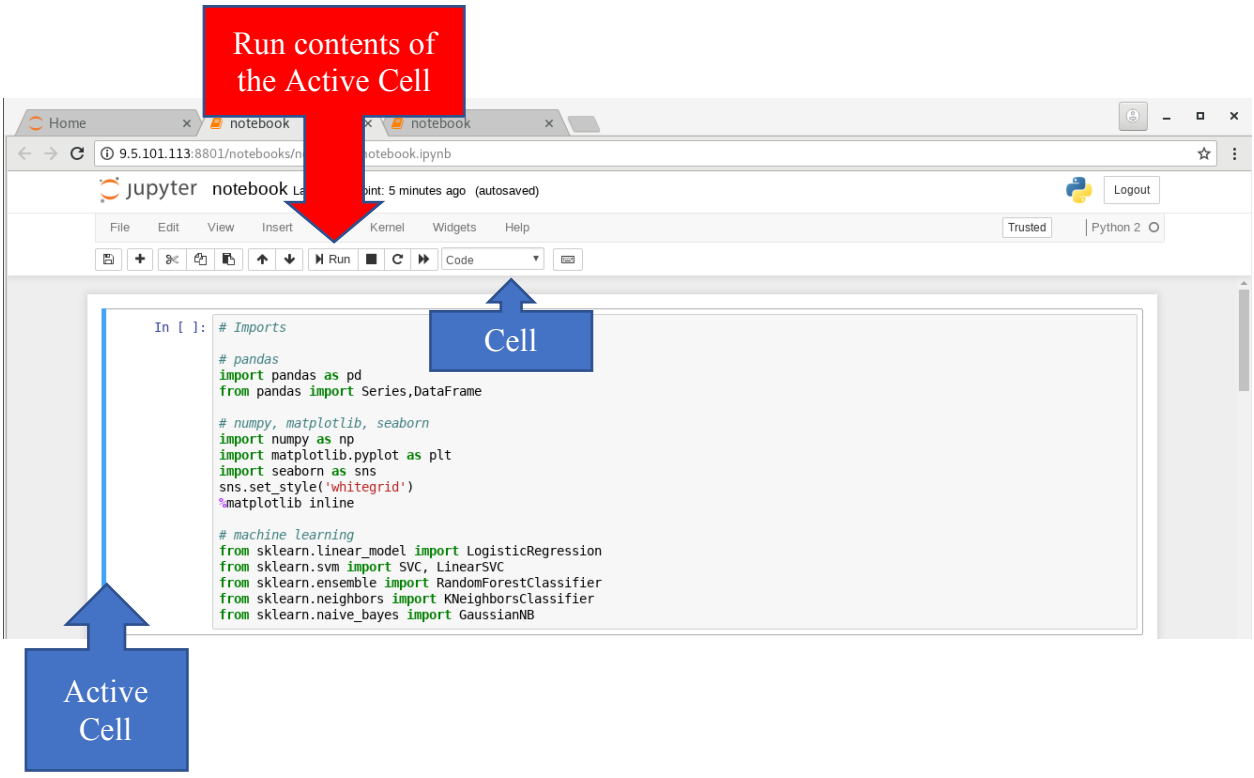
The ipynb files in /opt/DL/tensorflow are IPython Notebook files created for use by Jupyter notebooks. These notebook files are in JSON format to be created, viewed, edited, manipulated,and run in the Jupyter client within a web browser. Notebook files can contain markup code as well as python commands.

If you're interested in what ipynb files look like, you can open another ssh session to your container and inspect 00-classification.ipynb. For additional information on these 'notebook' files, take a look at:

https://github.com/tritemio/jupyter_notebook_beginner_guide

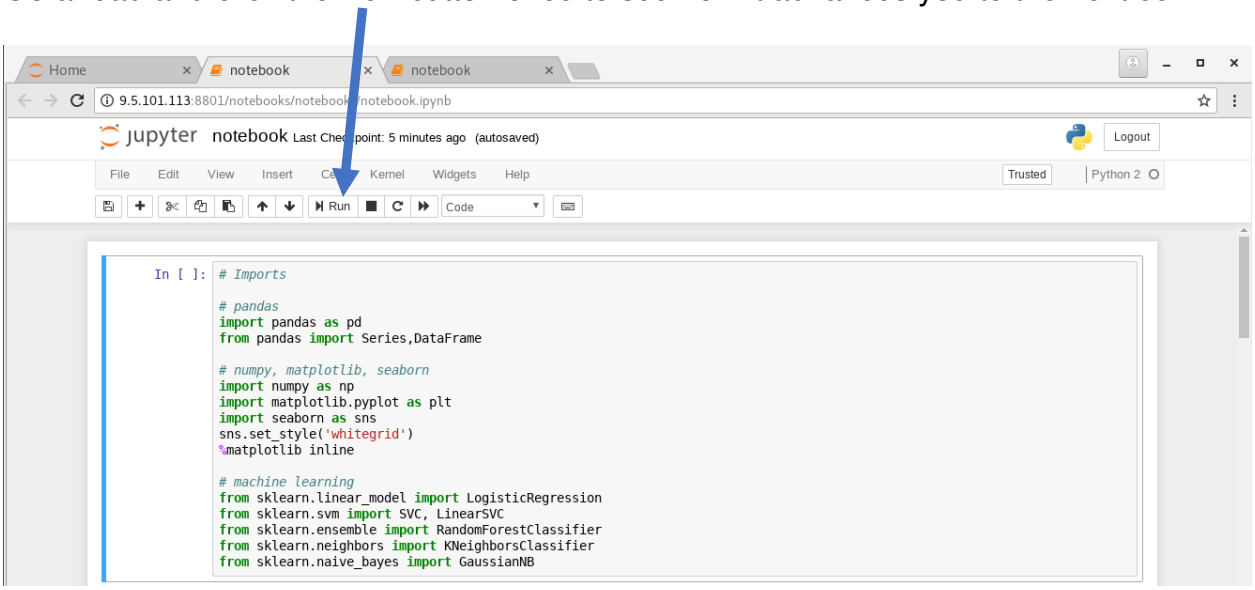
Note, however, that .ipynb files are not typically created by hand; they are created using the Jupyter Notebook.

In the screen shot below, the blue bar on the left highlights the active cell you are in, and to the right of the control buttons along the top, you can see that this particular cell is a 'Markdown' cell, which is really just a text cell used to document things in the notebook. The run button will step through code, section by section.



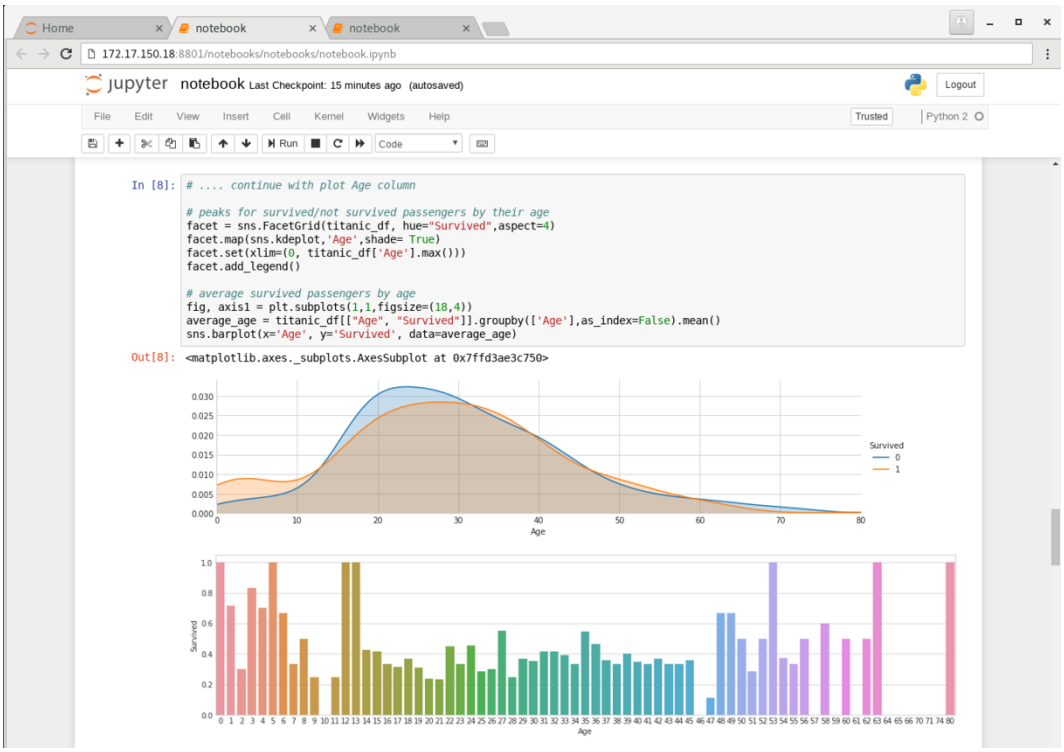
Step through the Notebook

Go ahead and click the Run button once to see how it advances you to the next cell.



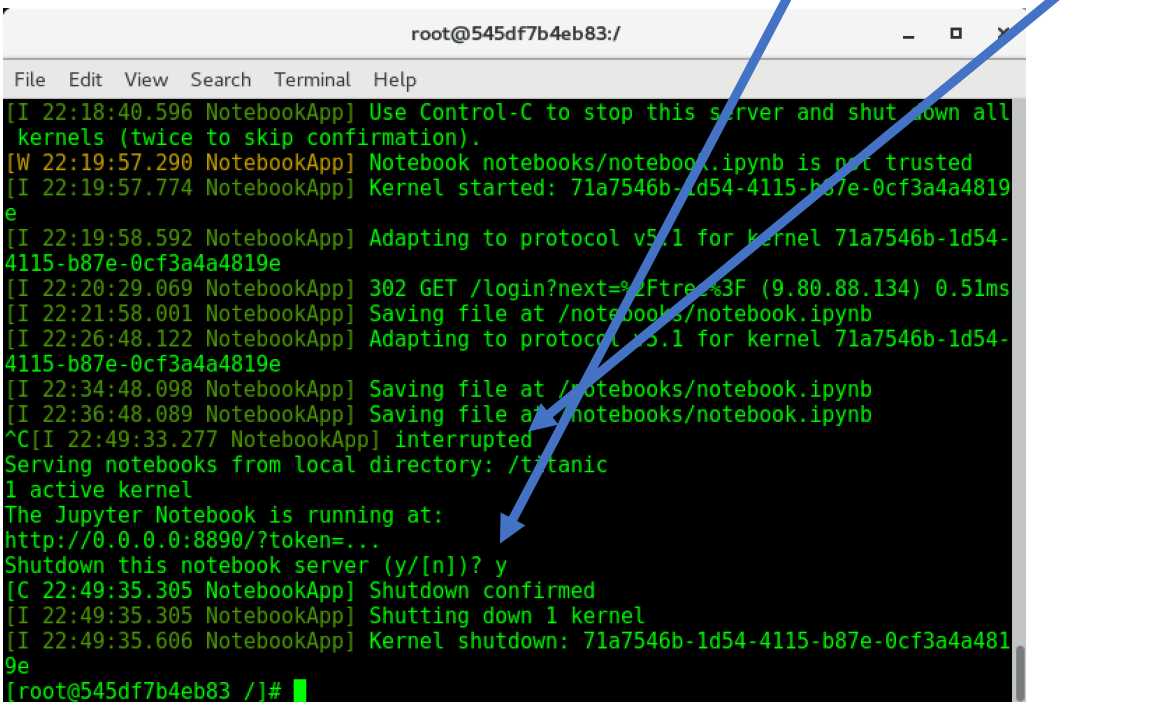
Ok, now that you understand the mechanics of the Jupyter Notebook, you're ready to step through **notebook.ipynb**. There are no Markdown cells in this notebook, only Code cells. And the output of Code cells that generate output will appear inline in the notebook.

We'll talk though this notebook and make some observations. Feel free to poke about, running the code cells and reviewing the output.



Conclude this Section

Find your ssh session where you started the Jupyter Notebook server, and type Ctrl-C to interrupt the notebook server. Then immediately enter 'y' to shutdown the notebook server:



Note: If you don't enter 'y' within 5 seconds, the notebook server resumes. If this happens, just type Ctrl-C again and enter 'y' quicker than last time! :)

Optionally, enter 'exit' to terminate your container, and then enter 'exit' again to end your ssh session. If you'll be doing additional lab sections, no need to exit your container

This concludes the **Section 5 Tensorflow-Deep Learning to Solve Titanic**.