Tensor Flow Windows Installation Instructions and First Example Programs

Note: Although the processing “guts” of TensorFlow are written in C++, Python is the most common TensorFlow interface, especially for getting started with TensorFlow. Therefore, this document will get you up to speed on Python installation / configuration first, then get into TensorFlow from there. If you already have Python installed and are familiar with Python basics then feel free to skim or entirely skip some of the initial steps.

Table of Contents:

----

1) Uninstall Any Previous Version(s) of Python (unless you already have 3.6 or later) and remove them from PATH

It is possible to have multiple version of Python installed on one computer, but this adds additional complication and therefore won’t be covered here.

In the Windows search box at the bottom left, type “Add or remove programs”, then check the list for any previous versions of Python or anything directly related. If you have any versions of Python prior to 3.6 installed, uninstall them.

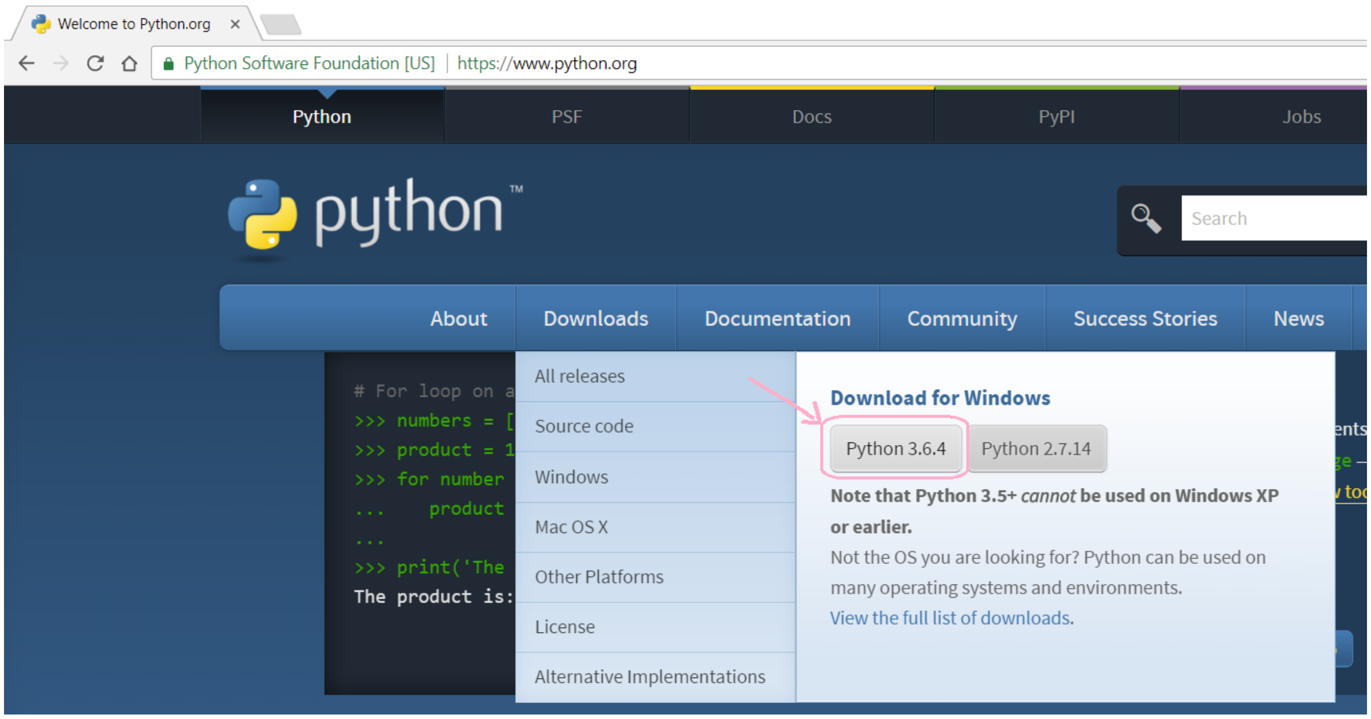
If you uninstalled a version of Python earlier than 3.6, check your PATH variable for any previous Python installs. To do this, in the Windows search box at the bottom left, type “System” then choose “Advanced system settings” -> “Advanced” tab -> “Environment Variables . . .”. Under “System variables” at the bottom, choose “Path” then “Edit” and check the list for anything referring to Python. If any paths referring to Python are found, remove them.

If you had to uninstall a previous version of Python or remove a PATH entry, reboot so these changes take effect, then continue with the next step.

2) Install Python 3.6 or later, OpenCV, and a Python Editor

You can refer to the installation instructions for TensorFlow on Windows on the official TensorFlow website https://www.tensorflow.org/install/install_windows for reference but I'll provide the necessary steps here.

Go to https://www.python.org/ and download the most recent version of Python 3, which is 3.6.4 as of this writing:


Welcome to Python.org
Python Software Foundation [US] | https://www.python.org
Python
PSF
Docs
PyPI
Jobs
python
Search
About
Downloads
Documentation
Community
Success Stories
News
All releases
Source code
Windows
Mac OS X
Other Platforms
License
Alternative Implementations
Download for Windows
Python 3.6.4
Python 2.7.14
Note that Python 3.5+ cannot be used on Windows XP or earlier.
Not the OS you are looking for? Python can be used on many operating systems and environments.
View the full list of downloads.

The TensorFlow website states that at least 3.5.x is required.

Proceed with the install. When asked, choose to add Python to the PATH for all users.

Next, reboot (do NOT skip this step), then bring up a command prompt and type "PATH", this will show the contents of your PATH variable. Verify your python location is present, i.e. "C:\Program Files\Python36\" or similar.

3) Install the necessary packages via pip

Next we will install the necessary Python packages via pip. pip is the Python package manager and installs automatically with the Python install from the previous step.

When installing packages via pip on Windows you must use an ***administrative*** command prompt or you will get very obscure errors due to non-administrative permissions.

To open an ***administrative*** command prompt, type “cmd” in the search box at the bottom left of the windows screen, then right-click on “Command Prompt” and choose “Pin to Start”. Next, choose the Start Menu icon (Windows symbol at the bottom left of the screen), then right-click on Command Prompt and choose “More” -> “Run as administrator”.

To verify that you’ve opened an ***administrative*** command prompt, verify that the title bar says “Administrator: Command Prompt” and the prompt text is “C:\WINDOWS\system32>”. Note that if you are *not* in an *administrative* command prompt the title bar will say “Command Prompt” and will default to the current user’s home directory, i.e. “C:\Users\UserNameHere”.

Since we’re using Python 3 as our only installed Python version, we can start a pip command with “pip” or “pip3”. I’ll use pip3 to make it 100% clear that we’re working with a Python 3 install.

In your ***administrative*** command prompt, enter:

```
pip3
```

If pip is successfully installed, this should show a summary of pip usage instructions and commands. Next, enter the following commands one at a time to install TensorFlow and OpenCV (we will use OpenCV in later tutorials, you may as well install it now):

```
pip3 install --upgrade tensorflow
pip3 install --upgrade opencv-python
```

The “upgrade” option will make this command update to the latest version of these packages if they were already installed, if not then this command will install the newest version. Note that pip will automatically install any packages that are dependencies of these packages.

Two other helpful pip commands are “list” and “search”. Use list now:

```
pip3 list
```

And verify that “opencv-python” (at least version 3.4) and “tensorflow” (at least version 1.5.0) are installed.

For future reference, you can use search if you have an idea what part of a package name may be but you’re not sure of the exact full name. For example, if you didn’t know the exact name of the OpenCV package was “opencv-python”, you could probably figure the package name at least contained “opencv”, so you could type:

```
pip3 search OpenCV
```

This will perform a case-insensitive search of all the packages with OpenCV in the name or the description, at which time you could check the list and find the package "opencv-python".

4) Install a Python Editor (PyCharm is recommended)

Performing the Python install above will install the editor Idle as part of the Python install, however it would be recommended to install PyCharm as it has great auto-complete, code warning, and many other modern editor features.

Other options include the Python extension within Visual Studio, and Jupyter Notebooks seem to be very popular lately, but I've found PyCharm to be the class of the field as far as Python editors. If you have another Python editor you prefer feel free to skip to the next step, but I'd really recommend installing PyCharm.

To install PyCharm, simply Google on "PyCharm" and follow the links to download the Community edition (yes, it's free). When the PyCharm installer asks you'd like to install the JetBrains Java Runtime rather than using the regular Java Runtime I suggest choosing the JetBrains Java Runtime option, I've found this will run PyCharm much more smoothly. For the other options, in most cases the default options will be fine.

5) Clone the repository containing this document

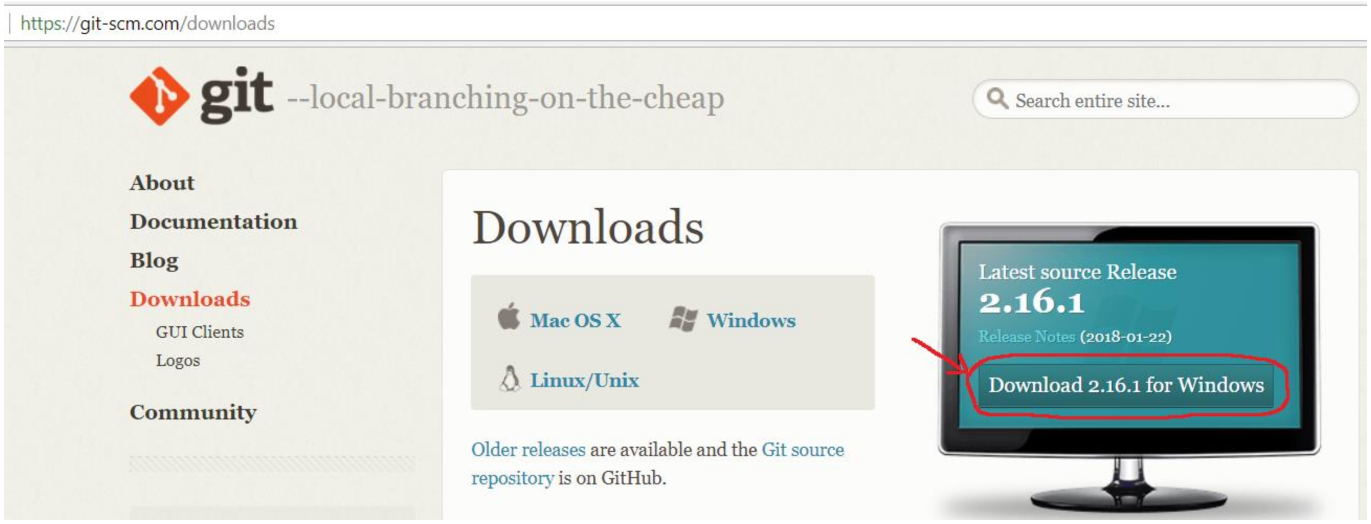
Clone the repository containing this document:

https://github.com/MicrocontrollersAndMore/TensorFlow_Tut_1_Installation_and_First_Progs

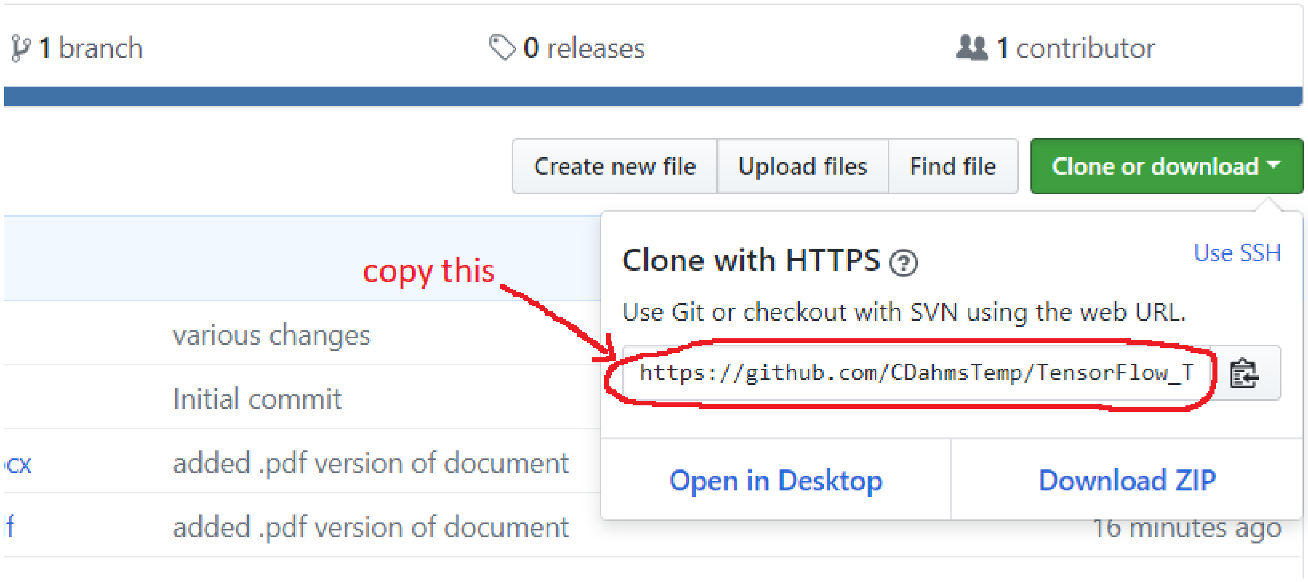
to a location you'd like to work in, for example I'll use:

C:\Users\cdahms\Documents\TensorFlow_Tut_1_Installation_and_First_Progs

Explicit instructions for those who are new to working with Git:
1) Download and install Git:


https://git-scm.com/downloads
git --local-branching-on-the-cheap
Search entire site...
About
Documentation
Blog
Downloads
GUI Clients
Logos
Community
Downloads
Mac OS X
Windows
Linux/Unix
Older releases are available and the Git source repository is on GitHub.
Latest source Release
2.16.1
Release Notes (2018-01-22)
Download 2.16.1 for Windows

2) Bring up a browser and navigate to *https://github.com/CDahmsTemp/TensorFlow_Tut_1_Installation_and_First_Progs, then towards the top-right click on "Clone or download" then copy the text in the text box:*



3) Bring up a command prompt and cd to where you'd like to clone the repo to, for example

```
cd C:\Users\YourUserNameHere\Documents
```

4) enter the following command:

```
git clone (paste text from text box above)
```

for example:

```
git clone https://github.com/CDahmsTemp/TensorFlow_Tut_1_Installation_and_First_Progs.git
```

You can clone to any directory you'd like. Going forward in this document we'll refer to the clone location of this repository as:
(repository_location)

6) Run *add_two_numbers.py* - Write a Python Script to add two numbers together

Make a file "add_two_numbers.py" and open it in your chosen Python editor. If you're using PyCharm and you attempt to make a new Python file, PyCharm will generally push you into making a project, which usually isn't necessary and further complicates things.

The easiest way around this I've found is to right-click in the directory you're working in, then choose New -> Text Document. Change the name from "New Text Document.txt" to "add_two_numbers.py" and you will then be able to open this file in PyCharm. Windows will ask you "Are you sure you want to change the file extension?", choose "Yes".

Make sure you have viewing file extensions enabled when doing this. To assure you have viewing file extensions enabled, in Windows 10 go to:

Search box at the lower left -> type “File Explorer Options” -> choose the “View” tab -> uncheck the box “Hide extensions for known file types”

Once you’ve created “add_two_numbers.py”, open it in PyCharm or your chosen Python editor and enter the following:

```
# add_two_numbers.py

# get the 2 numbers to add from the command prompt
num1 = int(input("enter num 1: "))
num2 = int(input("enter num 2: "))

# add the numbers
sum = num1 + num2

# show the result
print("sum = " + str(sum))
```

Alternatively, you can open the “add_two_numbers.py” file already present in the cloned ***(repository_location)***.

“add_two_numbers.py” is the simplest possible Python program to add two numbers together. Skim the code (it should be pretty self-explanatory), then run it and verify it works.

7) Run *add_two_numbers_via_TensorFlow.py* - Write a Python Script to add two numbers together via TensorFlow

Next we’ll add two numbers together, but via TensorFlow. In ***(repository_location)*** open “add_two_numbers_via_TensorFlow.py” in your chosen Python editor. Skim the code, then run it and verify it works.

Since this is your first TensorFlow program, it may seem very strange to perform a computational task by first defining a graph, then running the graph. If you’re new to TensorFlow and/or deep learning/neural nets in general, feel free to stare at this code for a few minutes and absorb this concept. Understanding this is essential since this is how all TensorFlow programs run (first setting up a graph, then running that graph).

8) Run *add_two_numbers_via_TensorFlow_TensorBoard.py* - Using TensorBoard to visualize your graph

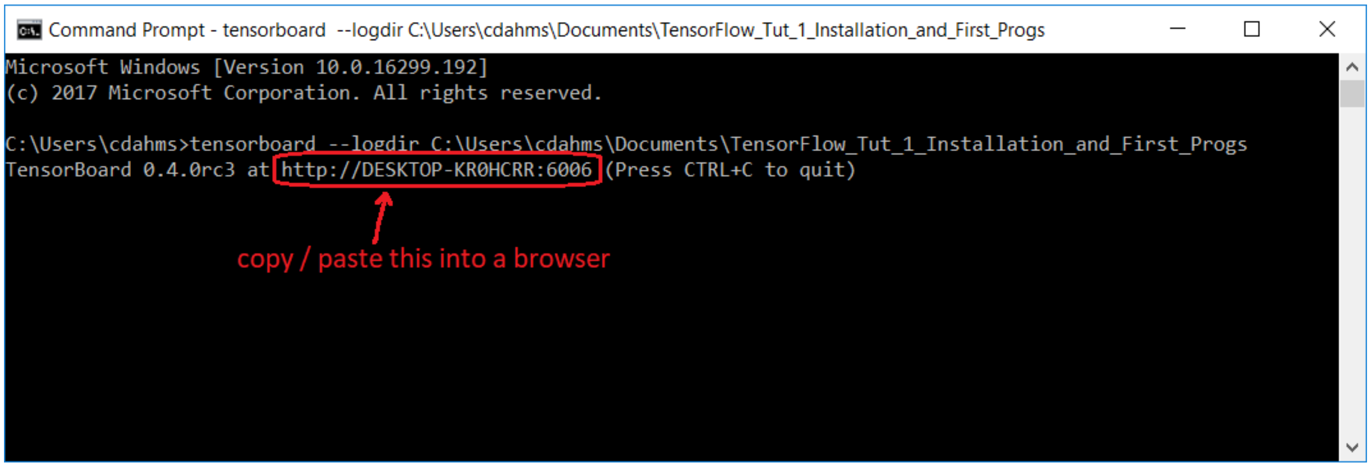
In ***(repository_location)*** open “add_two_numbers_via_TensorFlow_TensorBoard.py” in your chosen Python editor. Skim the code and verify it’s the same as the previous step, with a small number of lines modified and added at the end to facilitate using TensorBoard.

Run this script, then verify that in ***(repository_location)*** there is now a file with a name similar to “events.out.tfevents.1517075053.DESKTOP-KR0HCRR”, this is the file that will allow us to view our graph in TensorBoard.

Bring up a command prompt and enter:

tensorboard --logdir "***(repository_location)***"

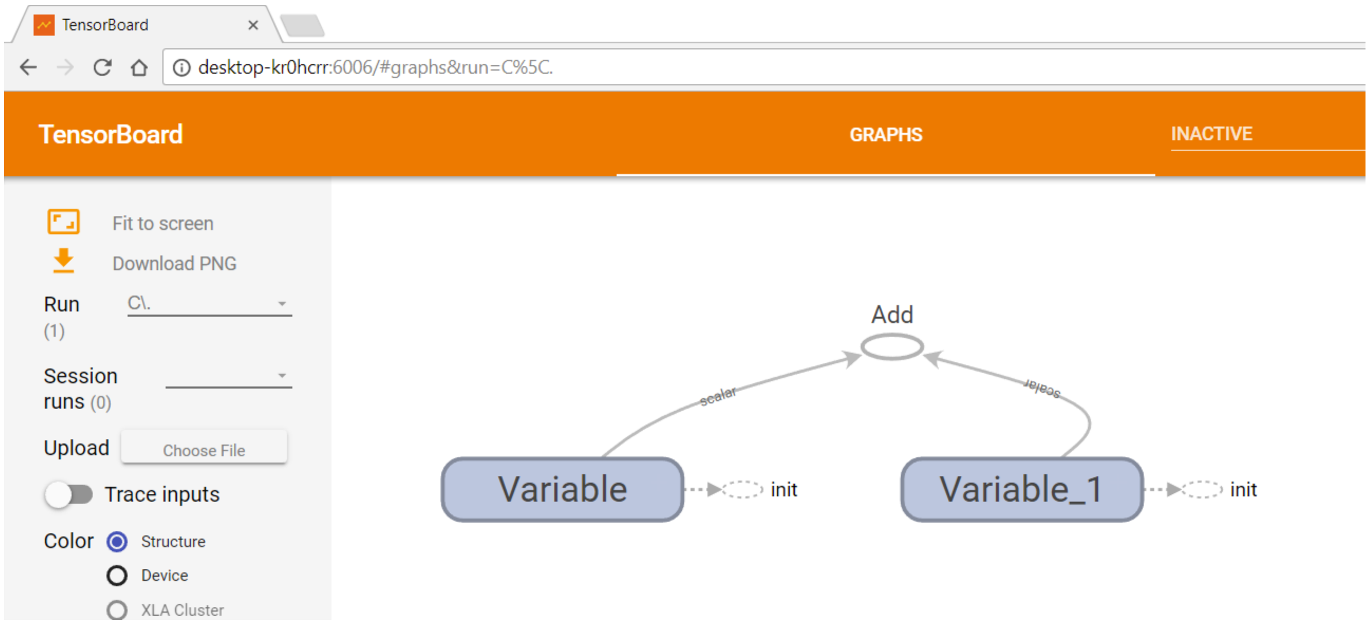
You should see something similar to this:



```
Command Prompt - tensorboard  --logdir C:\Users\cdahms\Documents\TensorFlow_Tut_1_Installation_and_First_Progs
Microsoft Windows [Version 10.0.16299.192]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\cdahms>tensorboard --logdir C:\Users\cdahms\Documents\TensorFlow_Tut_1_Installation_and_First_Progs
TensorBoard 0.4.0rc3 at http://DESKTOP-KR0HCRR:6006 (Press CTRL+C to quit)
```

Copy / paste the "http:// . . ." address text indicated above into a browser address bar and you should see TensorBoard showing your graph:



This particular graph display is not very interesting since this is about the most simple graph possible in TensorFlow. However, when developing a graph of significant complexity TensorFlow can be invaluable. TensorFlow also offers many other facilities for developing TensorFlow programs, but these are beyond the scope of this introductory document. If you'd like to learn more, Google or YouTube search on "TensorFlow TensorBoard".

9) Run ***mnist_softmax.py*** - Character recognition (using a pre-built data set)

Two of the first TensorFlow examples on Google's TensorFlow site, and many others as well, use the MNIST (Modified National Institute of Standards and Technology) hand-written character dataset. This is unfortunate because using this pre-built dataset does not demonstrate how to read in our own data and then run it through TensorFlow.

None the less, we will next take a look at these two MNIST examples, the first in this section that uses a relatively simple network and achieves about 92% accuracy, then a 2nd MNIST example that uses a Convolutional Neural Network (CNN) and achieves 98+% accuracy.

In the next two tutorial walk-throughs, classification and then object detection, respectively, we will use our own data. However, to get started let's continue with these two MNIST pre-built data set examples.

Make a new file "mnist_softmax.py", then open it in your chosen Python editor and copy/paste in the code from here:
https://github.com/tensorflow/tensorflow/blob/r1.5/tensorflow/examples/tutorials/mnist/mnist_softmax.py
or open the copy already present in ***(repository_location)***.

Go ahead and run the program, command line parameters should not be necessary. This program will download the MNIST dataset to C:\tmp automatically and then run its network on the MNIST dataset from there. If the program runs successfully the output should be similar to:

```
Extracting /tmp/tensorflow/mnist/input_data\train-images-idx3-ubyte.gz
Extracting /tmp/tensorflow/mnist/input_data\train-labels-idx1-ubyte.gz
Extracting /tmp/tensorflow/mnist/input_data\t10k-images-idx3-ubyte.gz
Extracting /tmp/tensorflow/mnist/input_data\t10k-labels-idx1-ubyte.gz
0.9205
```

The last line here is the accuracy.

There are a number of lines in this program in particular that leverage how the MNIST dataset it pre-built:

```python
from tensorflow.examples.tutorials.mnist import input_data
...
mnist = input_data.read_data_sets(FLAGS.data_dir)
...
batch_xs, batch_ys = mnist.train.next_batch(100)
...
print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))
```

These lines are what makes this example not so great; if you're using your own dataset you won't be able to use these lines at all, and developing the equivalent for your own dataset is a significant task.

Give the code a quick skim and try to get at least the gist of what the program is doing, then move on to the next example (don't worry about understanding every line for the moment).

10) Run *mnist_deep.py* - Character recognition (using a pre-built data set) via a Convolutional Neural Network (CNN)

Next make a file mnist_deep.py and open it in your chosen Python editor. Copy / paste in the code from:

https://github.com/tensorflow/tensorflow/blob/r1.5/tensorflow/examples/tutorials/mnist/mnist_deep.py
or open the copy already present in ***(repository_location)***.

So the program doesn't take too long to run, scroll down to the line:

```
for i in range(20000):
```

and change the 20000 to perhaps 2000, note that the program will still take about 5 minutes to run on a mid-range computer without a GPU.

You should be able to verify the training accuracy will get up to about 98% pretty quickly. Your output should look something like this:

```
step 1600, training accuracy 0.96
step 1700, training accuracy 0.96
step 1800, training accuracy 1
step 1900, training accuracy 0.98
test accuracy 0.9764
```

Give the code a quick skim and try to get at least the gist of what the program is doing, but don't worry about understanding every line for the moment.

In the next tutorial we'll walk-through classification (using our own images), then in the tutorial after that we'll walk-through object detection (using our own images). But first, we should cover two more topics; writing readable Python program (which surprisingly few people do) and how Python handles command line parameters (this is essential since many of the Google TensorFlow Python scripts require elaborate command line parameters be passed in).

11) 5 minute crash-course on how to write readable Python programs

Python offers a great deal of flexibility with where functions can be placed and where statements outside of functions can be placed. This can be unfortunate because many people use this flexibility in a bad way which results in programs that are exceptionally difficult to read. For example, Python will allow you to do something along the lines of the following:

```
# some_script.py

(statements here, not part of any function)

def functionA():
    (statements that are part of functionA() here)

(more statements here, not part of any function)

def main(_):
    (statements that are part of main() here)
    functionA()
    (statements that are part of main() here)
    functionB()

(more statements here, not part of any function)

def functionB():
    (statements that are part of functionB() here)

(more statements here, not part of any function)

if __name__ == '__main__':
    (statements here)
    tf.app.run(main=main, argv=[sys.argv[0]] + unparsed)
```

Clearly this is awful as the flow of the program is completely unclear and statements outside of functions are thrown in just about everywhere.

The purpose of the statement "if __name__ == '__main__':" is that if the script file containing this is being ran (i.e. if the script is "main") then whatever is after this statement will run before any of the functions. It will not, however, run before any of the statements outside the functions. For this reason it's a good idea to keep statements outside functions to a minimum. Here would be a recommended way to re-factor the above unreadable program:

```
# some_script.py

# module level variables ########################################
(module level variables here)

#################################################################
def main(_):
    (statements that are part of main() here)
    functionA()
    (statements that are part of main() here)
    functionB()
# end main

#################################################################
def functionA():
    (statements that are part of functionA() here)
# end function

#################################################################
def functionB():
    (statements that are part of functionB() here)
# end function

#################################################################
```

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

The program now clearly flows in a sensible manner from top to bottom and is easy to read. We will generally stick to this format throughout this tutorial series.

12) ***command_line_params.py*** - How TensorFlow Python scripts usually parse command line parameters

Python has a built in facility for parsing command line parameters which Google uses extensively with TensorFlow. For example, retrain.py:

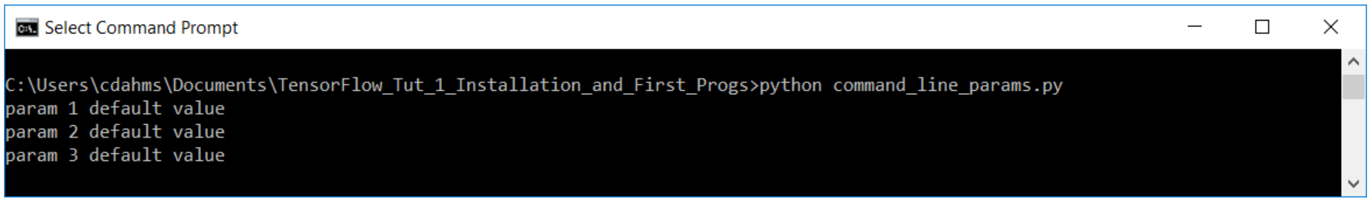
https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/image_retraining/retrain.py#L1232

which we use in the next tutorial, has 23 optional command line parameters that can be specified.

This format can be confusing at first, so we'll take a quick look at a simple example now to gain familiarity with this format.

In ***(repository_location)***, open command_line_params.py and skim the code, which should be relatively intuitive.

Run command_line_params.py from inside your chosen Python editor or from a command line without specifying any parameters, you should be able to verify output like this:



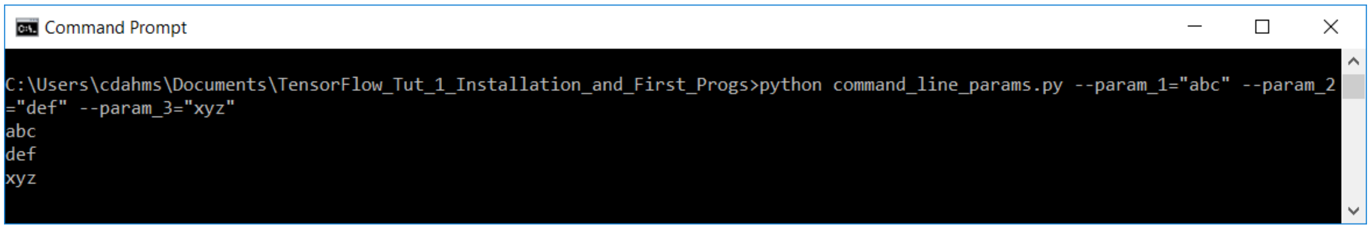
```
C:\Users\cdahms\Documents\TensorFlow_Tut_1_Installation_and_First_Progs>python command_line_params.py
param 1 default value
param 2 default value
param 3 default value
```

Since we didn't specify any command line parameters, the 3 print statement in main showed that FLAGS has the default values for the 3 params.

Now try to run command_line_params.py again, but this time specify something for each of the params as follows:

python command_line_params.py --param_1="abc" --param_2="def" --param_3="xyz"

You should be able to verify the following output:


Command Prompt

```
C:\Users\cdahms\Documents\TensorFlow_Tut_1_Installation_and_First_Progs>python command_line_params.py --param_1="abc" --param_2
="def" --param_3="xyz"
abc
def
xyz
```

Now the 3 statements in main show the params we passed in as expected.

Take some more time to familiarize yourself with this way of handling command line params as this format is frequently used in TensorFlow Python scripts.

Done!!

In the next tutorial we’ll walk-through classification, then in the 3rd tutorial we’ll walk-through object detection.