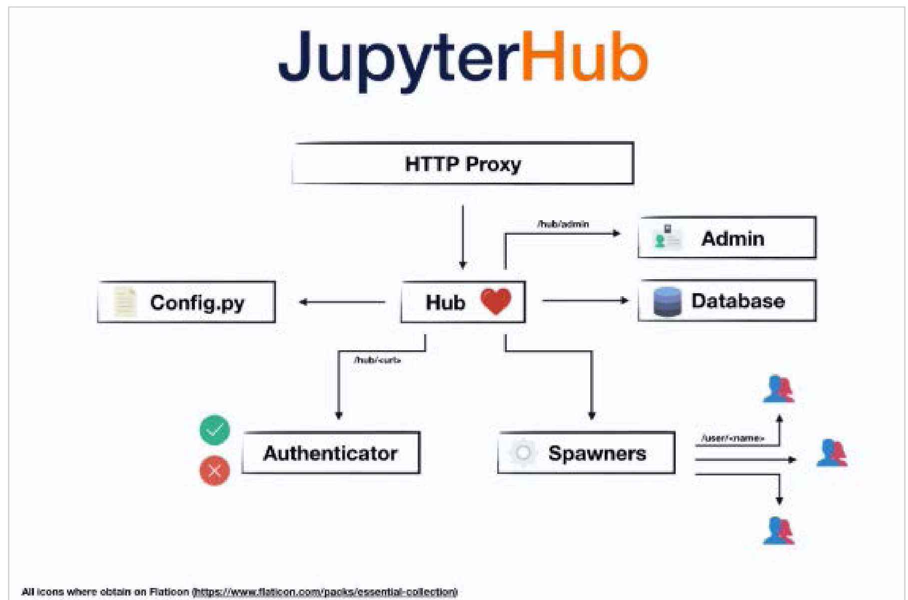
Trellix

Data Science Tips and Tricks

1. Running Jupyter notebooks on a shared Linux/ Unix machine

If you're working on anything related to data science, you are likely to use Jupyter notebooks. Even better, if you have a team of data scientists, you are likely to use JupyterHub. This amazing server lets you create a shared space for different users.



Installation is pretty simple, documented elaborately on the official guide. However, one of the issues we first ran into when we got the latest Nvidia GPU was how to run JupyterHub for multiple users. Running it is pretty simple, using the straightforward "jupyterhub" command. By default, it is configured to have just one instance for all users. However, what if you have multiple users who each want their own instance of JupyterHub? There are workarounds, such as a sudospawner, but the easiest way to have an isolation between users in this case is to run it as sudo. Hence, the command is "`sudo jupyterhub`". As this Github issue clearly states, "*The guide uses sudo, but never to become root. It sets up restricted sudo permissions to allow the Hub user to become specific other users only to launch their notebook servers, nothing else.*"

2. Activation function: sigmoid v/s softmax

This point is in context to activation, or, in other words, transfer functions which are typically used as the last layer (sometimes also between layers) of a neural network to map its outputs to a decision. Typical activation functions include tanh, sigmoid, linear, ReLU, LeakyReLU, softmax, softplus and many more. Let's focus on sigmoid and softmax – they are very similar, yet very different.

If you are aware of their equations, you know the most striking difference between the two is that while both the functions convert numbers to probability scores, the output of sigmoid does not sum up to 1, whereas softmax naturally forces the probability values such that they sum up to 1. Meaning, for an input to have a higher probability value, another input's probability value must be reduced accordingly such that they both sum up to 1. This is exactly what tells us that when to use them.

Another interesting fact is that for a binary classification problem, theoretically, it has been *proven* that sigmoid is a special case of softmax when the number of inputs is 2.

Hence, if the task at hand does not require the inputs to be mutually exclusive, or, if it is the task of binary classification, use sigmoid. Else, if the task at hand requires the classes to be mutually exclusive, or, it is a multiclass classification problem, use softmax—that way, the output will be forced to be the class with the highest probability.

3. Working with images

Working with images for the first time can be poles apart than working with tabular data. This is because, with csv-like data, you are likely to use Pandas and dataframes, whereas for images, numpy comes to the rescue elegantly. This awesome fast-computation math package is a true boon for ML folks. Here are a couple of pointers which are best kept handy.

The first step is generally to learn your data's natural distribution. The shape (size) of your data will depend on the number of images and their size.

For example, imagine you have 100 images, each 32 pixels wide in black and white. The shape will be as follows, if you are using *Tensorflow (channels-last by default):*

```
(100, 32, 32, 1) à (batch_size, height, width, num_channels)
```

If you have the same number of images but in RGB, it will look like the below:

```
(100, 32, 32, 3) à (batch_size, height, width, num_channels)
```

For Keras applications, the terminology used is channels last (corresponding to 32, 32, 1) or channels first (1, 32, 32). You will be able to view your default setting in your keras.json file, typically located in `~/.keras`.

Now, imagine you built a convolution neural network with the above configuration and achieved great results.

However, you'd like to know how good it is at classifying this one image. How do you reshape your data such that takes in just image, with a dimension of (32, 32, 3)? You reshape it as follows. If,

```
np.shape(image) = (32, 32, 3) then,
new_image = np.reshape(image, (-1, 32, 32, 3))
```

Your CNN should be able to predict new_image just fine.

Say you had your dream dataset that looks very much like CIFAR-100. It is manually not possible to go through each of those 6000-odd images. Additionally, a rule-of-thumb is to always normalize images. So, here is a simple 4-liner script for reading RGB images and normalizing them between [0,1] using Open CV2. Open CV2 is yet another life blood for those working with images. You can modify it for a bunch of images placed in a directory too by creating a loop in Python.

```
image = cv2.imread (img_name, cv2.IMREAD_COLOR)
image = cv2.resize (image,(image_size, image_size))
image = cv2.cvtColor (image,cv2.COLOR_BGR2RGB)
new_image = image/255
```

Since 8-bit images are in the range [0, 255] (because 2^8 = 256), dividing by the highest value normalizes them between [0,1]. It makes sense when one has to work with activation functions which lie in the same range. Another quick pointer: As mentioned above, 8-bit images are in the range [0, 255]. What is the darkest and brightest image? Values closer to 0 are bright and values closer to 255 are darker.

4. Having difficulty displaying huge numpy arrays?

Another JupyterHub/Jupyter notebook hack. Older versions of JupyterHub by default have a data rate limit set to display outputs for a cell. I was working with a pickle file that had a significant number of entries (164397) and was trying to display it. Below is the error I got:

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.
```

The default data rate limit is 1000000, which you can find in your Jupyter notebook's config file. But this is not adequate when you are trying to display significantly sized numpy arrays.

Alas, here is the idea that helped. Use the following line:

Option – 1:

Change is to a higher value by appending a 0 to it. This option will work depending on the size of your data. If

you still get the error and need more space, the next option will definitely work.

Option – 2:

```
import numpy as np
np.set_printoptions(threshold=sys.maxsize)
```

Before:

```
[1 1 1 ... 1 1 1]
```

I know I don't have all 1's in my dataset.

After:

```
In [22]:  1  print(Y_encoded)
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1
 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1
 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 0 1 1 1 1 0 0 1
 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1
```

All thanks to the value sys.maxsize, which, per Python's documentation, is:

"An integer giving the maximum value a variable of type Py_ssize_t can take. It's usually `2**31 -1` *on a 32-bit platform and* `2**63 - 1` *on a 64-bit platform"*

Essentially, it is the largest size that any data structure such as lists, dictionaries, etc. can have in Python. Since I work on a 64-bit machine, this value for me, is `2**63 - 1 = 9223372036854775807`.

5. How to get pre-trained models:

Per Occam's Razor, a good model always has a simple design. One will thus turn to pre-trained models that have achieved a state-of-the-art result on massive datasets, such as ImageNet, ResNet, and the likes. Now, the obvious and easy way, is to include the models from Keras and let it do the dirty work of downloading, unpacking, and loading the downloaded weights for you. Unfortunately, I was unable to use the below line in my code:

```
keras.applications.inception_resnet_
v2.InceptionResNetV2(include_top=True, weights='imagenet',
input_tensor=None,
input_shape=None, pooling=None, classes=1000)
```

which is the Keras pre-trained Inception-ResNetV2 model. I could download and place it in my working directory, but what if I wanted to use it for another model placed elsewhere? I was not willing to make multiple copies of a humungous model. After hours of looking for a work around, I found a brilliant hack: download whichever model you require directly from the source. Then, manually place it in the folder with the path: `~/.keras/models`.

Next, simply define model architecture and the model should load using `load_model('model_name.h5')`.

This directory is also where you should see any other models you previously downloaded through Keras applications.

6. Awesome Open CV2 installation for Linux

Amongst the many OpenCV installations for Linux out there, this one is the most concise and a combination of two installation tutorials.

https://medium.com/@debugvn/installing-opencv-3-3-0-on-ubuntu-16-04-lts-7db376f93961

7. GPU sharing is caring

If you have multiple teammates using a single GPU server, you will know the value of sharing resources. One of the best ways to do this is through Tensorflow or Keras. If your data seems like it will occupy one entire GPU, just mention the available CUDA device in it. These are well-known. However, what if it gives OOM errors while you are trying to augment it? The data sometimes increases exponentially, especially when dealing with images. In this case, a smart alternative is to reduce your batch size. Instead of training on 128 images in one batch, train on 64. That way, even after the data is augmented, it still fits in memory and your training proceeds uninterrupted.

8. GAN training

Albeit a tad old now, while I initially started experimenting on GAN's, I found this amazing resource to achieve better training.

9. Model analysis

Everybody knows that the only way to gauge a model's performance is by analyzing it's results on a test set.

There are multiple ways to achieve this task today, like Tensorboard which offers pretty graphs and great visualization, and, scikit-learn libraries which make computing metrics look too easy. However, sometimes one would like to calculate more than just one metric. Personally, I sometimes tend to dump output predictions from my model along with the ground truth into a text file to compute accuracy metrics (like false and true positives).

This gives me a holistic view of how my model performed and better analyze which examples from my dataset are performing better than others. I found this stack overflow link super-useful while writing to text files and I keep it handy. The difference between w, w+ and a, a+ is what is most helpful about it. https://stackoverflow.com/questions/1466000/python-open-built-in-function-difference-between-modes-a-a-ww-and-r

10. Exploratory data analysis

When you work with large datasets, it is possible that you have multiple examples belonging to multiple classes.

If you're looking to quickly count the number of unique class data points you have in your training (to check for class imbalance), here is a simple one-liner from numpy:

If y represents the distribution of examples you wish to uniquely count:

```
Import numpy as np
unique, counts = np.unique(y, return_counts=True)
```

The output will be something as follows:

```
Unique: [0 1]
Counts: [5353 159044]
```

Meaning, I have two classes, namely, 0 and 1, and, the count of class 0 → 5353 and count of class 1 → 159044.

Thus, my data is highly imbalanced. You can always make it pretty to display it as a dictionary.

Visit Trellix.com to learn more.

Trellix

About Trellix
Trellix is a global company redefining the future of cybersecurity. The company's open and native extended detection and response (XDR) platform helps organizations confronted by today's most advanced threats gain confidence in the protection and resilience of their operations. Trellix's security experts, along with an extensive partner ecosystem, accelerate technology innovation through machine learning and automation to empower over 40,000 business and government customers.

Copyright © 2022 Musarubra US LLC

042022-01