Dog breed classifier project is an introductory example to demonstrate how to build a CNN, model from scratch, the thumb rules behind model selection and how to train the model.

To download this project, use the below commands.

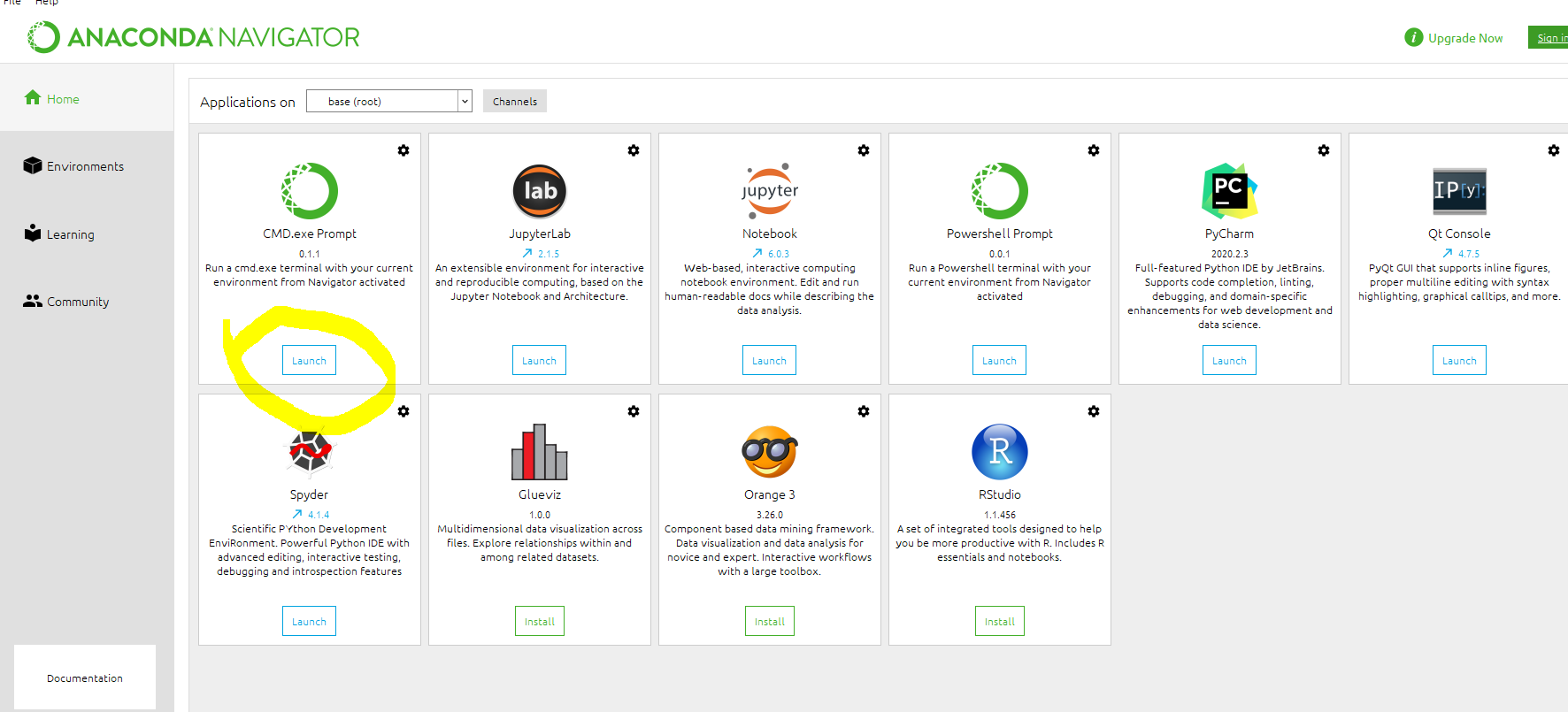
Git clone <https://github.com/jeccec51/DogBreed_CNN.git>

I recommend downloading and using anaconda.

To install latest Anaconda installer, refer to below link.

<https://docs.anaconda.com/anaconda/install/windows/>

Once anaconda is installed, I recommend creating a virtual environment. To do that run anaconda terminal from the main window. (The example is for windows users. In Linus, you can directly run conda from terminal)



Once the CMD is launched, create a new environment.

conda create --name torchenv python=3.6

This will create new virtual environment

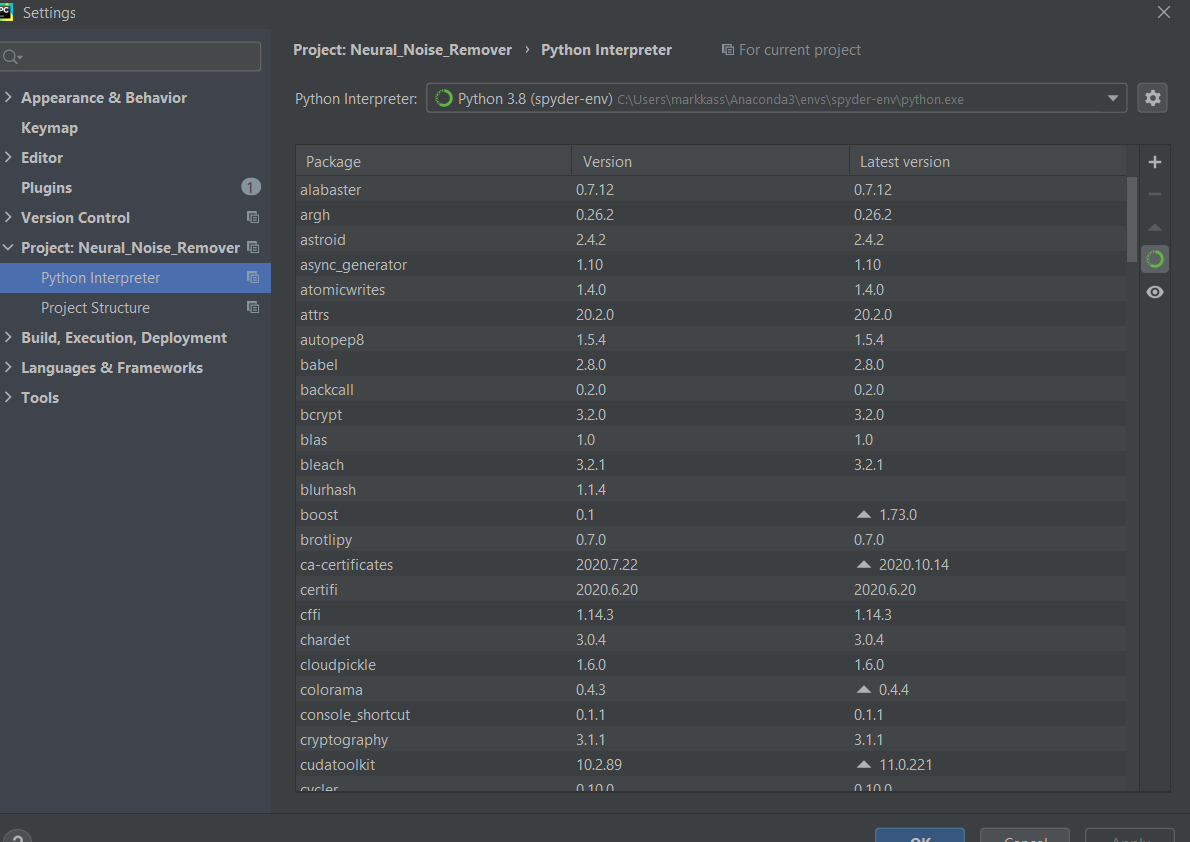
I am attaching list of packages that need to be added to this environment. To install a package, you can use conda install packagename.

This package name info is available in file trochenv\_names.txt. Use this file to install missing packages in your environment.

I Personally use pycharm, for python projects, as their memory management is efficient. You can use any slandered python editor. If you are using pycharm, I suggest changing the virtual environment to the one we created recently.

This can be done by File>Settings>Interpreter>Python Interpreter and then navigating to the newly created folder. Usually the environments are located at C:\Users\UserName\Anaconda3\envs\nameofnev\python.exe

If every step is followed correctly the code will be loaded in Pycharm without any error. Refer the screen shot below for detailed steps



The dog breed classification data can be downloaded here

<https://www.kaggle.com/c/dog-breed-identification/data>

To keep things simple, I randomly selected 20% of the total data, and put them under a validation folder. Please note that, this can be run with in data loading functions, this part will be covered in a future example.

Step 1.

# Load the Data

The data loaded from train, test, and validation folders.

Size: The input image is resized to 256 x 256. The rationale behind this is to choose a size as power of 2 so that the computation is much easy for a GPU. The higher the image, higher will be the computation time. The lower the image, lower will be our CNN depth. Higher depth is preferred for complex classification task. So as a trade-off, assuming final CNN should deliver a feature map of at least 32 x32, in size to uniquely describe each of the dog breed class, the 256-size is used.

I used augmentation (Affine, crop flip, rotation) to reduce the overfitting.

Step 2.

# Visualize Data

Data visualization functions can be used to initially visualize some training data, selected randomly, to make sure everything fine.

Step 3.

# Check for GPU

Next step is to select the correct device for training. A GPU can considerably improve the training time. getdevice function selects the gpu device if it is available

Step 4.

# Model selection.

The dogbreed classification is a challenging task due to its intra class variability. Large depth is required to adequately represent the features of each class. To begin with, I am using a depth of 4, starting from 256 x 256 x 3, at the input of layer one. At the end of each convolutional layer a pooling layer will through off the spatial pattern by reducing the feature maps to half of input size (stride is 2, filter is 2).

From the era of resnet it is proved that a 3x3 kernal is best suited for most of the complex convolution tasks, I am employing the same convention. I use a padding of 1, to make sure feature maps have same size as the input.

The depth of the convolutional maps are increased in the below order

16->32->64->128, at the outputs of 1, 2, 3, and 4th layers respectively.

This is to make sure we get a lot of possible feature maps, to ensure highest inter variability.

After CNN, I used 3 dense layers. The in and out features are as given below at the output of each layer

(128\*32\*32, 1024)

(1024, 512)

(512, 133)

After the feature maps are identified, we need to maintain high intra class variability as well. The more the dense layers we add up, the better will be intra class variability. To begin with, I am using a 3 layered nw.

I used dropout with 0.29, to avoid overfitting

Step 5

# Training

I used Cross entropy error function, as this is best suited for a classification task. I use SGD optimizer with a momentum of 0.9. Adams could considerably improv the performance. A learning rate of 0.03 was chosen. The training is done for 200 epochs, and the validation will be done at the end of each epoch. After each epoch the model will be saved if the validation loss is decreased. In this way we can avoid overfitting. A plot of training and validation loss is created at the end of training to get intuitive ideas about over and under fitting.

This example achieves around 40% accuracy, after 200 epochs. After 200 epochs its observed that the model considerably overfit.

Step 6

# Testing

Testing data set will be used for testing the model. First the best model saved during the training will be loaded. The training flag should be kept false to skip training the network in every time. We turn this flag on only when the network is trained for the first time. in subsequent run of this project for testing, training flag should be kept off. A testing accuracy of 35% is achieved with is introductory example. This is much better than a random predict, in a 120-classification problem, the probability of correct prediction in case of Radom guess is 0.0083.