Noise\_Remover project is an introductory example to demonstrate how to filter periodic noise from an image using CNN.

To download this project, use the below commands.

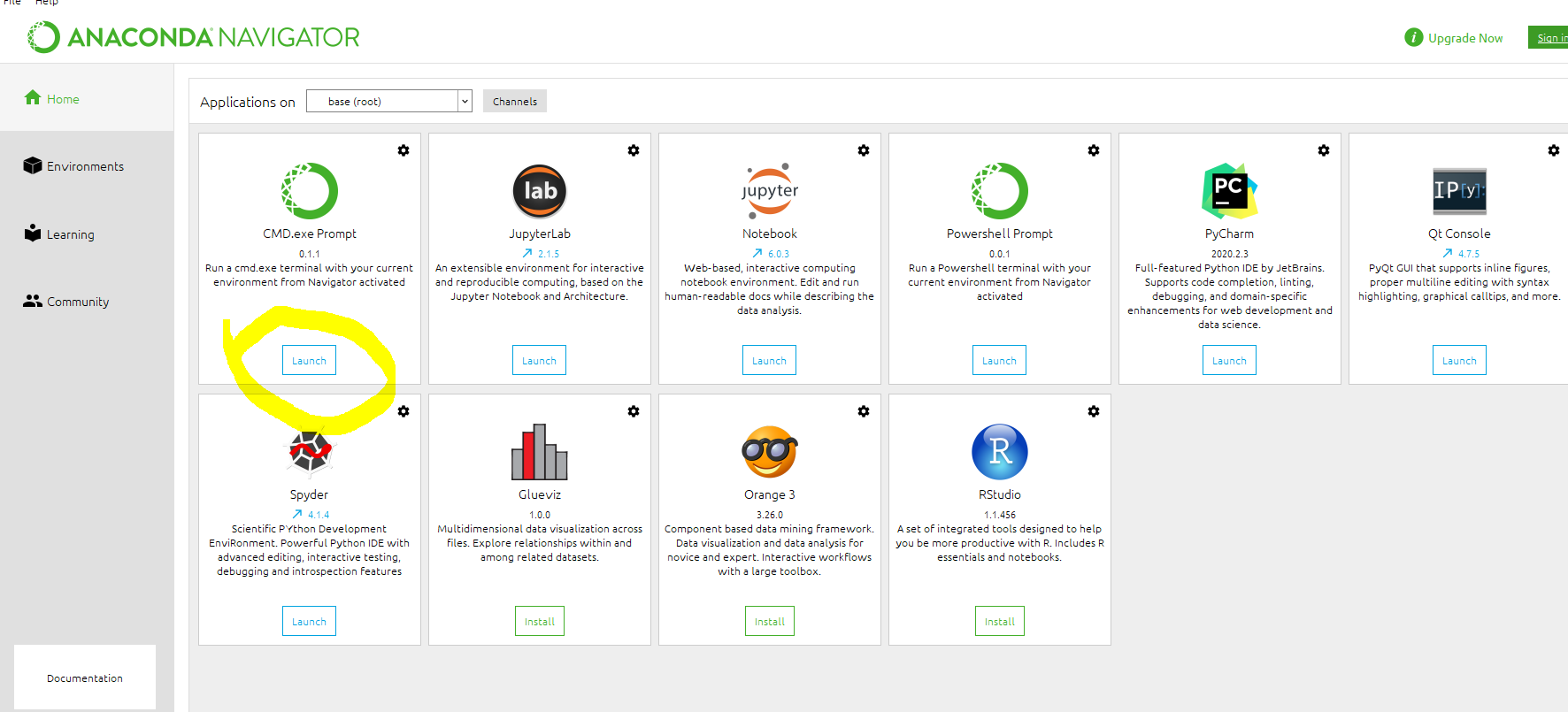
Git clone <https://github.com/jeccec51/Neural_Noise_Remover.git>

I recommend downloading and using anaconda to setup packages and environment. You may use pip as well.

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

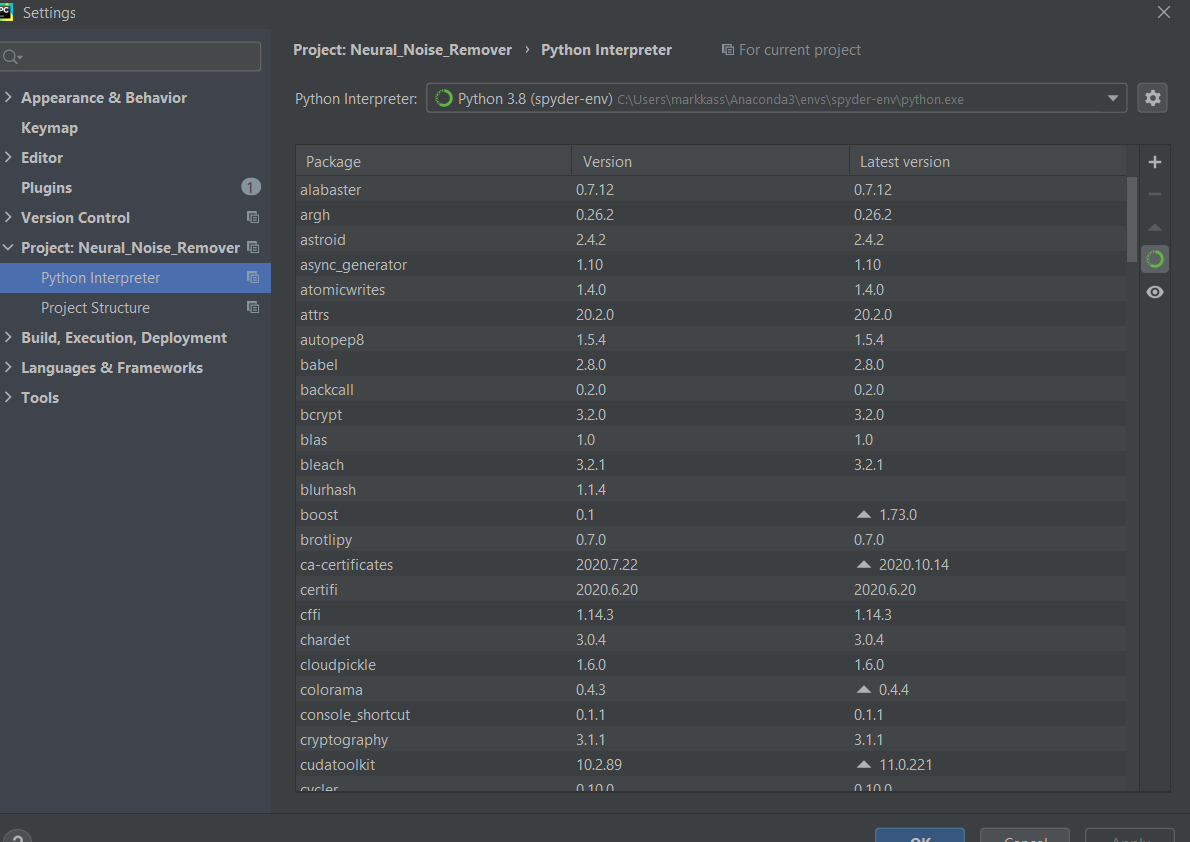
I am attaching list of packages that need to be added to this environment. To install a package, you can use conda install packagename.

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



# Noise Removal

Periodic noise is repetitive, fast moving patterns present in an image along with foreground and background. On example is provided below.



Lena Image has Lena in the foreground, studio wall in the background, and undesirable set of repetitive patterns in the image as well. For most of the image processing applications such a noise highly undesirable. One of the pre-processing techniques we use is to remove this noise.

In this introductory project, I am trying to show how can we remove such repetitive patterns using the learned feature maps in CNN.

This project is deeply inspired by the below research paper.

<https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Gatys_Image_Style_Transfer_CVPR_2016_paper.pdf>

In this paper the made use of the learned feature to transfer the artistic style of an image to foreground of another image.

This idea comes from 2 aspects.

1. The foreground related information is stored very deep in an CNN, probably the last or second last layers.
2. By studying the similarity between the feature maps, across different layers, we can get an estimate of the style of an image. The artistic styles are repetitive, spread throughout the image.

We are interested in point number 2. Style is nothing but a periodic noise kind of component.

The idea is to weigh each of the CNN layers correlation matrix between the vectorized feature maps. O, wait, this too complicated……..

Ok … lets break it down.

1. Weigh---🡪 In this example we use a VGG net. WE all know that VGG19 has 16 deep conv layers. So, each of the layer is weighed with respect to a empirically set factor
2. Feature Maps-🡪 Each of the convolution layer produce number of feature maps, where this number is equal to depth of that layer.
3. Vectorized feature maps -🡪 Lets say layer\_1 produces 16 feature maps. Now we unroll each of the feature maps in to a 1D array, its size will be product of width and height of the image.
4. Now we place each of this unrolled 1D arras as rows of a matrix. So, if a layer has 16 feature maps of 112 x 112 features, the column matrix described in point 4 will have a dimension of 16 X 12544.
5. At this point we need to calculate the cross-correlation between images, so the cc matrix will be of dimension 16 x 16 in the example mentioned in point 4 (in paper this is termed as grammatrix)
6. Now in this way we will calculate the gram matrices for 1\_1, 2\_1, 3\_1, 4\_1, 5\_1 layers and add them as per the weight defined in point 1.
7. We will start with a random noise image, and will calculate the loss as mean squared error between the final matrix obtained from point 6 and noise image. The pixels of the noise image will be adjusted using back propagation , and finally we get an image similar to gram matrix. The gram matrix is nothing but a representation of repeating similar patterns in the image.
8. In the next step this periodic noise image will be subtracted from original to remove major chunk of periodic noise in the image.

get\_vgg19\_model()

This function returns a pretrained vgg19 from torchvision

def get\_device():

This function selects appropriate device (GPU/CPU) depending on the availability

load\_image\_as\_tensor(img\_path, shape=None):

This function loads input image as tensors. (Image to be filtered)

def im\_convert(tensor):

This function convert the processed tensor back to image for visualization

get\_features(image, model, layers=None):

This function extracts the feature maps form the layers of a CNN. If layers are not mentioned, it will be defaulted to

'0': 'conv1\_1',  
 '5': 'conv2\_1',  
 '10': 'conv3\_1',  
 '19': 'conv4\_1',  
 '28': 'conv5\_1'

It returns a dictionary with feature maps against each layer

def find\_gram\_matrix(tensor):

Calculates the cross-correlation gram matrix for the feature maps

def generate\_random\_image():

Generates a random noise image using PIL

def train\_image(model, optimizer, steps, rand\_image, image\_grams, show\_every=500):

Train the image. Optimizer is Adams. loss is mean squared error