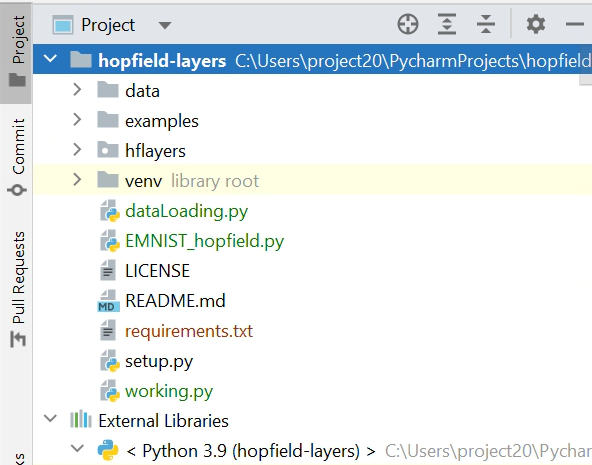
**Code Execution:**

This project was run as a PyCharm project and is based on the Hopfield layer implementation that can be found here:

<https://github.com/ml-jku/hopfield-layers>

The steps for setting up the project are:

1. From PyCharm, clone git repository by:
   1. From the upper left corner click on **VCS** and pick **Get from Version Control** from the dropdown list.
   2. In the pop-up box, make sure the **Version Control** field is set to “Git”.
   3. Specify the URL of the remote repository that you want to clone - <https://github.com/ml-jku/hopfield-layers.git>
   4. Chose the directory in which the project will open and press the **Clone** button.
   5. After repository has been successfully cloned, we now have a project set with a virtual environment and the implementation of the Hopfield layer.
2. In the **Project** window (left side), under **Hopfield-layers** add the attachment “working.py”. it should look like this:



1. Open “working.py” and run.

**Code structure:**

This code is composed of 3 main parts:

1. Data acquisition and augmentation (lines 40-73) –

The EMNIST dataset is imported from torchvision’s datasets library. Using the “transform” attribute, the data is augmented with a random rotation of up to and random vertical and horizontal shifts of up to 20% of the image size.

1. Auxiliary functions –
2. Train\_epoch (lines 79-128) – This function executes for each batch in an epoch the forward propagation, loss and gradients calculations, and finally back propagation.
3. Eval\_iter (lines 129-159) – This function evaluates the current model, returning the loss and accuracy.
4. Eval\_test\_set (lines 160-189) – This function evaluates the test set and returns the predictions along with the ground-truth label.
5. Operate (lines 190-224) – This function gets as input the dataset, optimizer and architecture and utilizes the other functions to train the model. It returns 2 Pandas DataFrames that contain the loss and accuracy of each epoch.
6. Set\_seed (line 224-237) – sets random seed.
7. Plot\_performance (line 238-263) – This function plots and saves the loss and accuracy curves.
8. Main (lines 272-end) –

In this section we use Pytorch layers to assemble the architecture, define hyper-parameters for optimization, and run the training and evaluating in a loop. Notice the hyper-parameters we used are not just the ones specified in the lines of code. Since optimization takes a generous amount of time, certain parameters were optimized together and certain apart.

**Result**

For each model trained, 3 files are saved –

1. PDF file containing loss and accuracy plots as a function of the epochs for both the training and the validation data.
2. PNG file containing the confusion matrix for the 26 labels. The header includes the test set accuracy and loss.
3. Model.pth contains the saved model along with its weights for later uses.

Because the size of the “model.pth” files take up a lot of memory, we have attached only the best model. The rest of the models are saved locally on our computers and can be sent to you if needed.