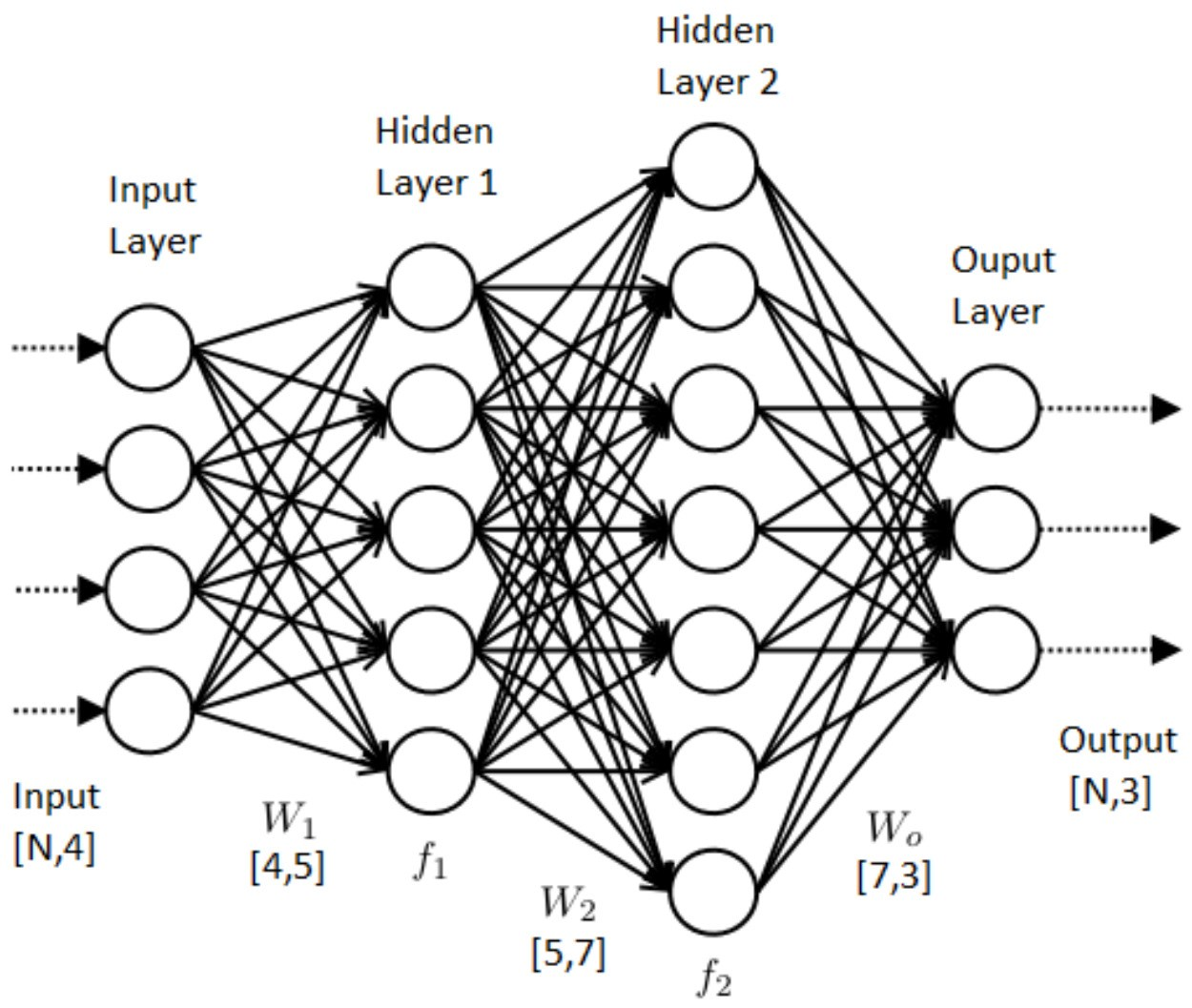
Traffic Sign Recognition

**Dataset**

**Convolutional Neural Network using PyTorch**

In this first week, I learned how **Neural Networks** work in general by the example of digits recognition. First, NN has its idea from the way our brains work. Secondly, it is called deep learning because NN consists of layers and each layer gives information to the next one and we keep going deeper to reach our desired result.

There are 3 main types of layers:



1. The input layer: It’s always one later and it is the entry to the neural network where we feed it with information.
2. The hidden layer: one or more. Let’s say for now that it is where the calculations happen to get the final answer.
3. Output layer: Also, always one layer and it has the answers we want.

In addition, we have neurons which are the circles in the above figure and they are responsible for holding information (we better say actually that a neuron is a function which we call activation).

I will explain briefly and example of how recognizing digits work. Let’s say we have number 3, so the input will consist of 3 \* 512 \* 512 where 3 stands for RGB values for each pixel and 512 \* 512 are the dimensions of the image we are feeding to our neural network. Now in each layer we hold a set of pixels and as we go to the next later, we merge them with another set, so let’s say we have number 3 and 9, they both have the half circle part at the top which will contribute to adding a value to the neurons of 9 and 3 in the last layer (they will be a total of 10 in our case and each will store the probability of seeing digit x in our picture from 0 to 9).

Well, also these lines which connect the neurons with each other have a value called Weight for each which will also determine how it will contribute to the next layer as we mergin the sets of pixels, we should get a new value to the merged set.

In the last layer, we pick the number which corresponds to the brightest neuron (the neuron which has the max probability among others in that layer).

I didn’t get into math here because it will make this report very long and assuming that it is just a report not a tutorial.

Now getting back to what was learnt from the articles after understanding the base knowledge...

**Conventional Neural Network** is a neural network which is optimized for image recognition purposes. Since we are going using it for more specific type of tasks, we have more options and ways for optimization. As a result, they contain better ways of merging these sets of neurons as we go from a layer to another. In addition, neurons have better abilities dealing with pixels such handling many pixels in one neuron and faster next layer activation calculations such as metrices multiplications. A simple ConvNet architecture could have the architecture [INPUT-CONV-RELU-POOL-FC]:

1. INPUT holds the input image as a 3-D array of pixel values.
2. CONV computes dot product between the kernel and sub-array of an input image. keeps summing all the values resulted from the dot product starting from one-pixel level until the whole input image is covered and for all the kernels.
3. RELU applies activation function max(0,x) on all the pixel values of an output.
4. POOL layer down samples the image
5. FC (Fully-Connected) computes the class score for each of the classification category.

(for example, how likely is it a cat or a dog)

**Datasets:**

In genral, we have 3 types of datasets:

1. Training Dataset: which we use to let the neural network learn the values of the weights.
2. Validation Dataset: We use this layer to know how should we adjust the hyperparameters of our network. (Sometimes we can skip using it)
3. Test Dataset: We check how good is our neural network with this dataset and it should be different from the first two. Also, never let your NN learn from this one because it will just give good results and that is a wrong measure about real life behavior of our network.

We should also make the training dataset balanced (for example number of cat images should be as much close as possible to the number of Dog pictures). If this balance was not guaranteed, then we will get in the trouble of Overfitting because we trained our NN in the wrong way.

In the German Traffic Signs Dataset, there were many folders which contain the signs only (snipped images) in addition to real life pictures of the streets outside these folders. Images had the format of ppm which depends on the header:

If the PPM magic identifier is "P6" then the image data is stored in byte format, one byte per color component (r,g,b).

If it is "P3" then the image is given as ASCII text, the numerical value of each pixel ranges from 0 to the maximum value given in the header. The lines should not be longer than 70 characters.

"P6" image files are obviously smaller than "P3" and much faster to read. Note that "P6" PPM files can only be used for single byte colors. Furthermore, the dataset is attached with a Python code to read the images into a NN.

**PyTorch:**

Now talking about PyTorch since I did some exploration with it. It has many functions and useful things such as MNIST which is a dataset of digits images. The main data structure is called a Tensor which can hold an image as a matrix in our case but what is weird about it is that the dimensions are 1 by x by y to store x by y matrix. As a result, we have to be very careful with dimensions to meet the functions requirements. Another thing is that we can write our Net Class in which we can adjust how information is passed by neurons if we don’t want to use the ready libraries functionality. Consequently, this gives us powerful possibilities to adjust the NN or (CNN in our case) as we wish especially for research purposes.