**Summer 2022 Deep Learning**

**Report of Lab #2**

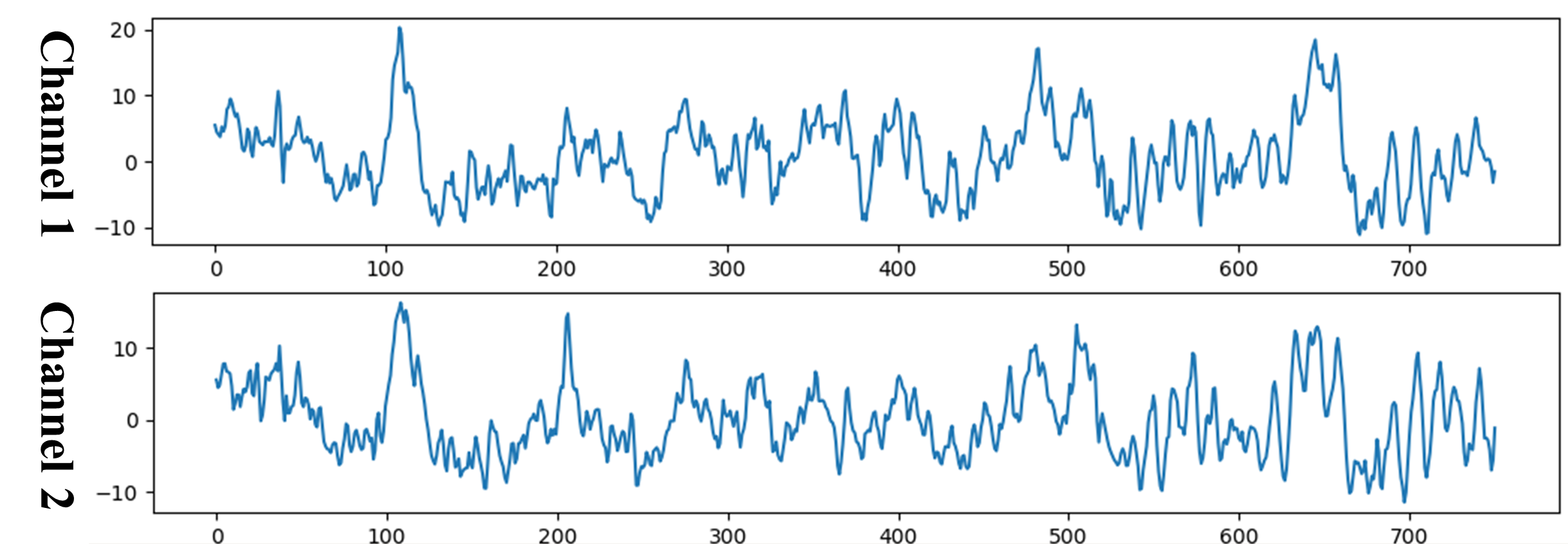
Student name: 劉子齊 Jonathan

Student ID: 311605004

**Part 1: Introduction**

In this lab, my goal is to implement EEGNet and DeepConvNet with three kinds of activation functions, which are ReLU, Leaky ReLU, and ELU, and classify the provided EEG signals into two classes. Also, we should show the highest accuracy of two models with three kinds of activation functions and plot the accuracy curve to show the learning result.

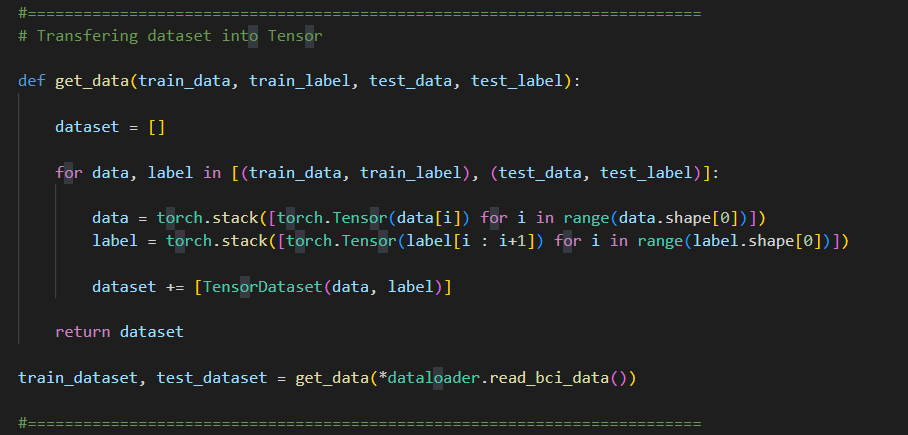
Before moving on to the next part, I would to talk about the dataset I made use of in this lab. The following is how the dataset looks. The dataset got 2 channels with 750 datapoints each. These datapoints can be classified into two labels, which are left-handed and right-handed.



**Part 2: Experiment set up**

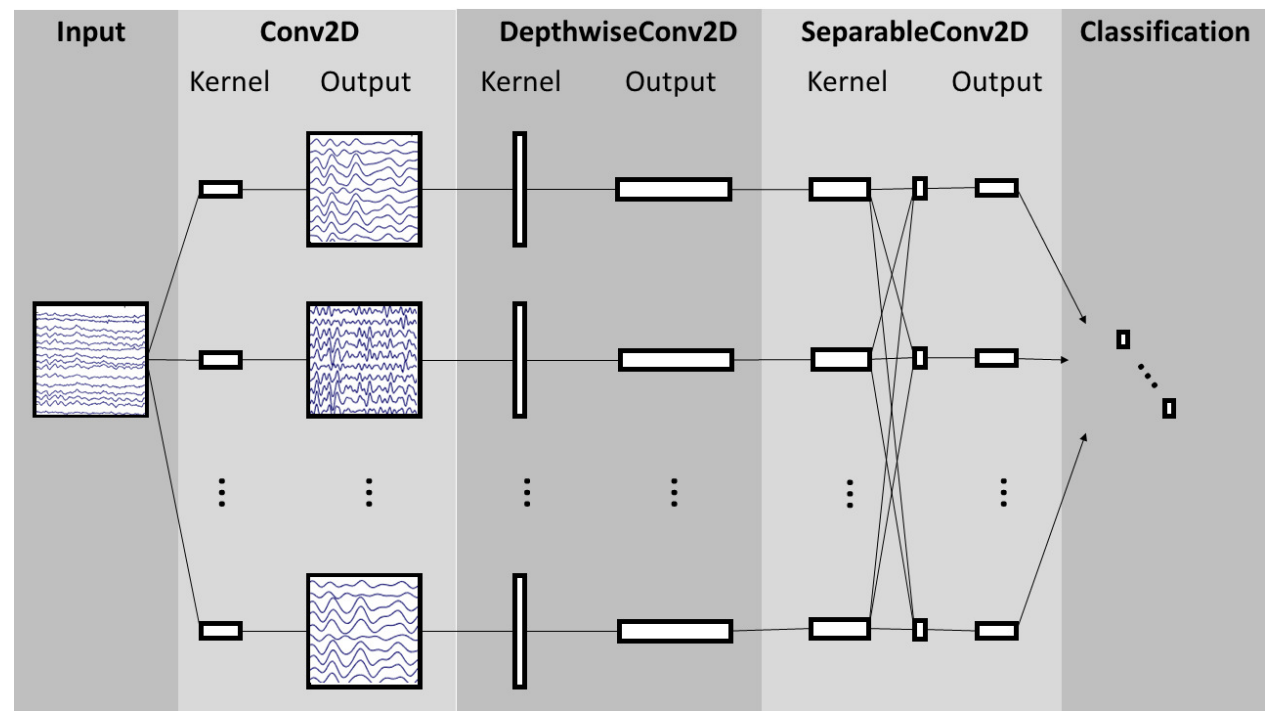
**Part 2-0: Data Setup**

To apply DataLoader from “torch.utils.data”, I convert the original dataset obtained through the “dataloader.py”, which is in the form of numpy array, into the form of TensorDataset. Converting data to TensorDataset allows me to train the model with specific batch size easily. The difference between a NumPy array and a tensor is that the tensors are backed by the accelerator memory like GPU and they are immutable, unlike NumPy arrays. Thus, every tensor can be represented as a multidimensional array or vector, but not every vector can be represented as tensors.

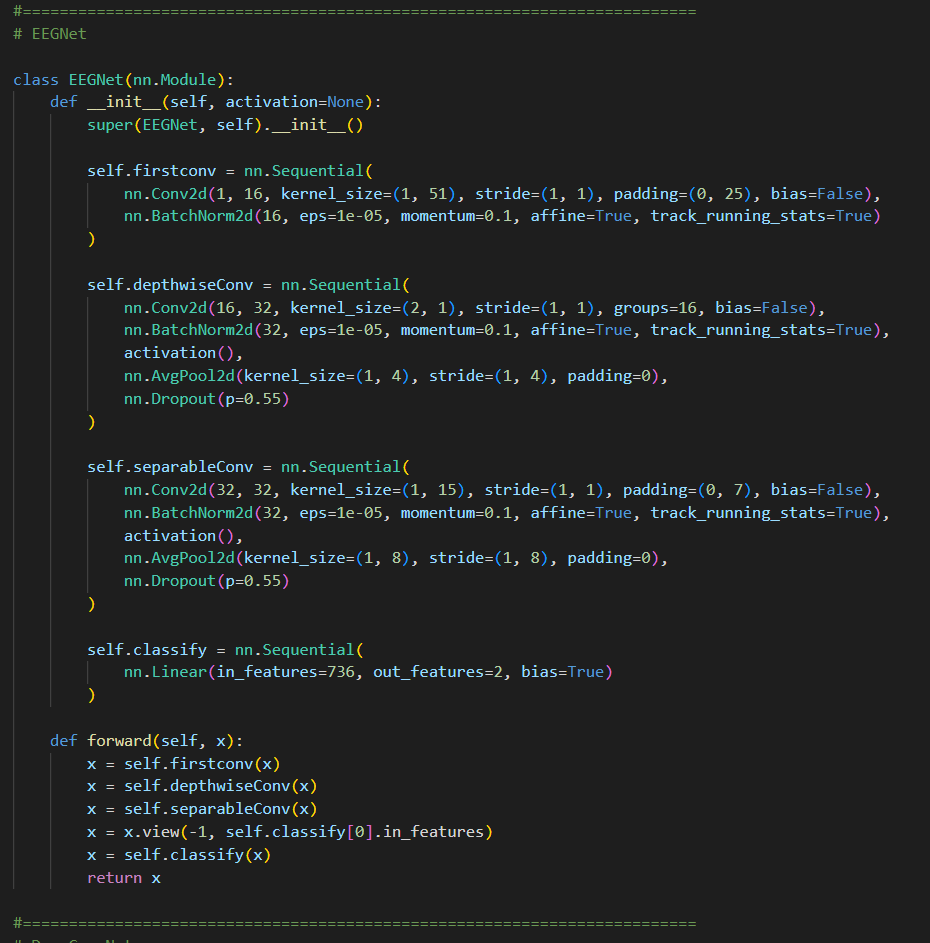


**Part 2-A: Details of the two models**

**Part 2-A-1: EEGNet**

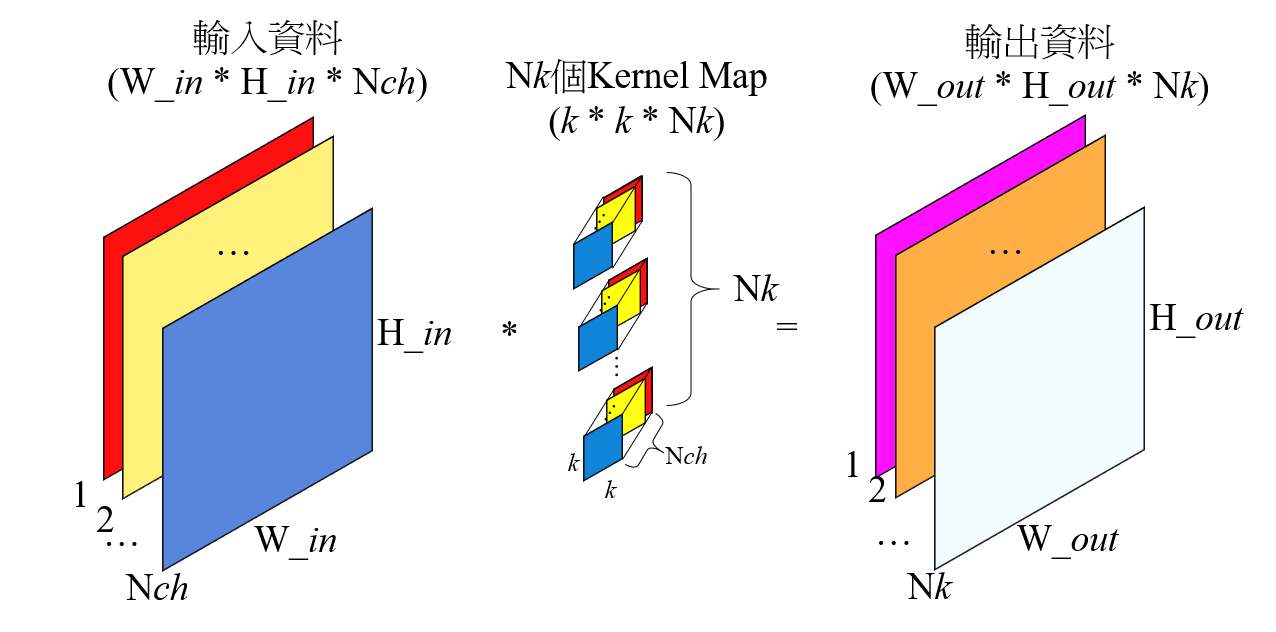


The figure above is the overall visualization of the EEGNet architecture from the reference “EEGNet: A Compact Convolutional Neural Network for EEG-based BrainComputer Interfaces.”, which is also where the EEGNet was born. The following figure is how I implemented the EEGNet architecture. In the first block, the model learns how to extract features from the EEG signals, and will output a feature map containing the EEG signals at different band-pass frequencies.



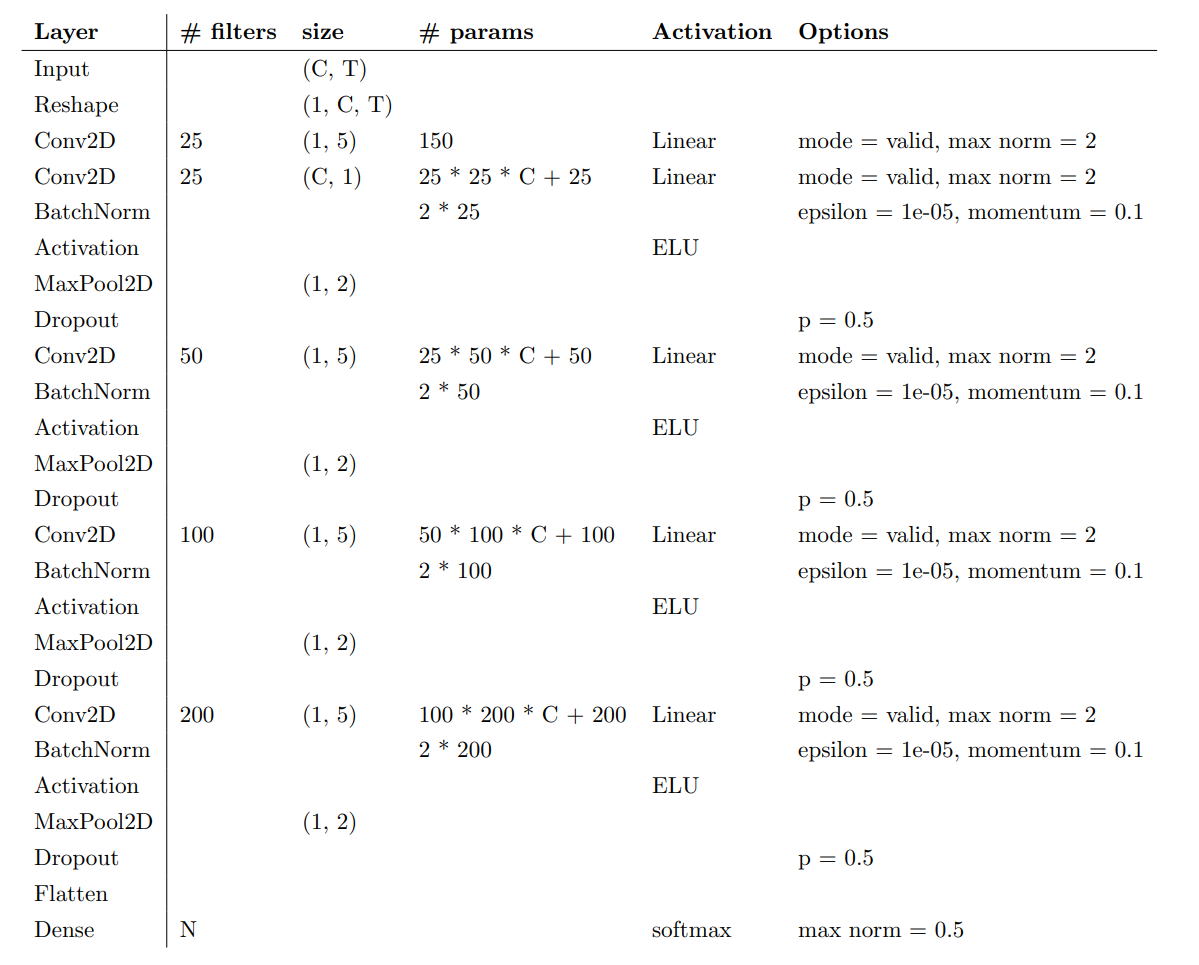
In the second block, which is the Depthwise Convolution, the main purpose of this block is to learn a spatial filter. This operation provides a direct way to learn spatial filters for each temporal filter through the not fully-connected property, thus enabling the efficient extraction of frequency-specific spatial filters. I also applied Batch Normalization here before the activation function to improve the learning rate of a neural network before applying the activation function nonlinearity. After that, I applied a 2D average pooling layer, to reduce the sampling rate of the signal. Last, I used the dropout technique and set the dropout probability to help prevent over-fitting when training on small sample sizes

In the third block, we use a Separable Convolution, which is actually a Depthwise Convolution with different size. There are two main goals here. First is to reduce the number of parameters to fit. Second is to decouple the relationship within and across feature maps through the property of Depthwise and Pointwise Convolutions, which can be seen in the figure below.

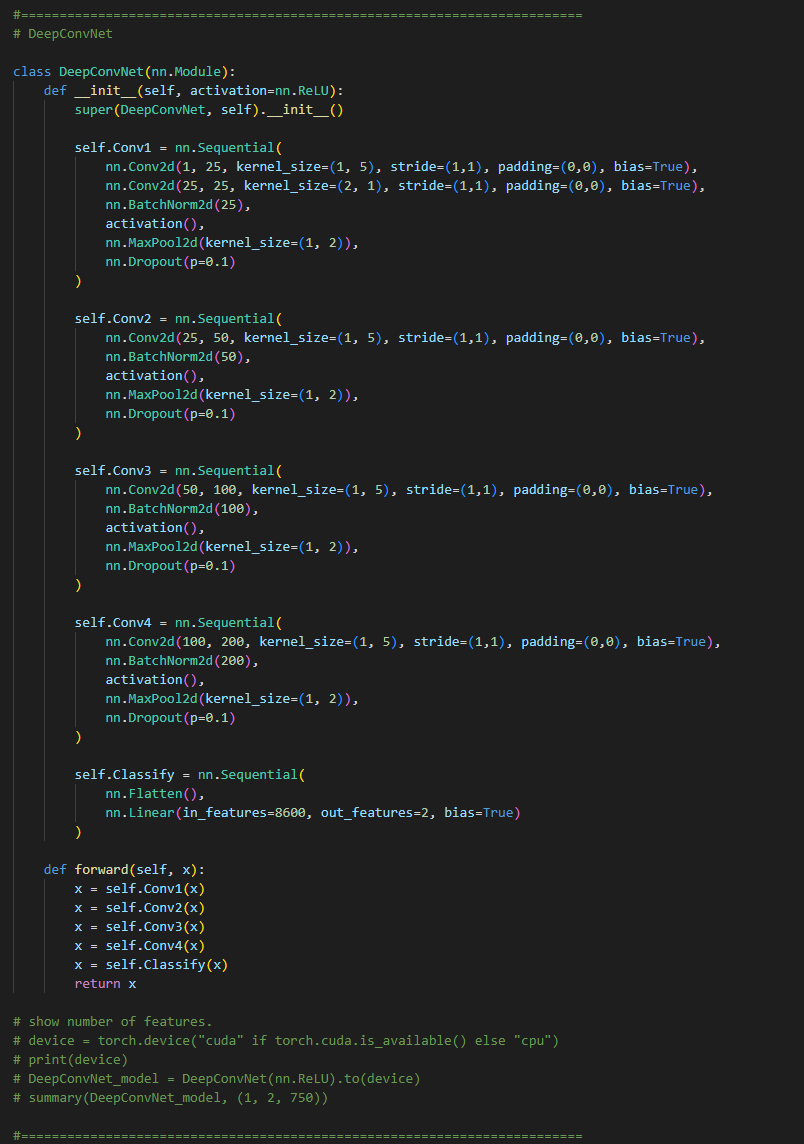


In the last block, which is the classification block, the features were send to the SoftMax classification layer. Then we will get a dictionary named “accuracy” with model names and accuracy results in it.

**Part 2-A-2: DeepConvNet**



The figure above is the of the architecture of the DeepConvNet from the reference “EEGNet: A Compact Convolutional Neural Network for EEG-based BrainComputer Interfaces.” In the paper, we can find some comparison between DeepConvNet and EGGNet. However, we can consider DeepConvNet as a normal CNN. DeepConvNet is designed to be a general-purpose architecture, which is not restricted to specific feature types. The figure below is how I implemented the DeepConvNet according to the architecture provided by the paper above.



**Part 2-B. Explain the activation function**

The following is the mathematical representation of the three activation functions:

* **Rectified Linear Units (ReLU)**



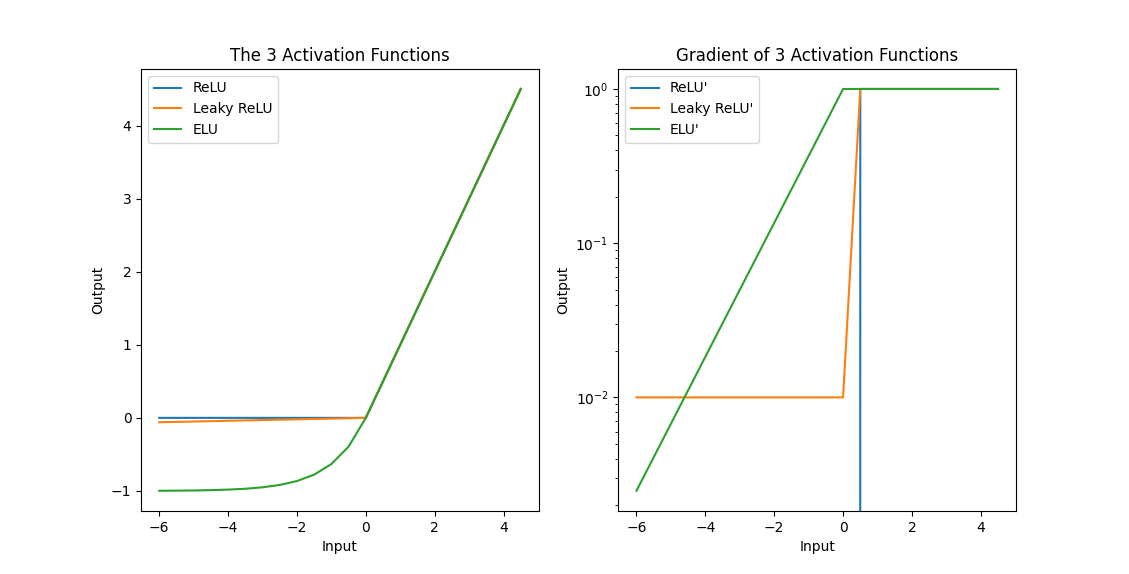
* **Leaky Rectified Linear Units (Leaky ReLU)**



* **Exponential Linear Unit (ELU)**



The figure below which I plotted the three activations in, we can easily compare them through it. We can tell that the three activation functions actually act nearly the same from the positive input value. Through the subplot on the right, which is the gradient of the functions, we can tell that ReLU gets 0 gradient whenever the input is negative, and this can probably cause serious vanishing problem to the model. Hence, Leaky ReLU and ELU were designed to avoid such problem. Both of them got values but not zero when negative input, and from the figure we can tell that ELU git greater computation volume than Leaky ReLU.



**Part 3: Experimental results**

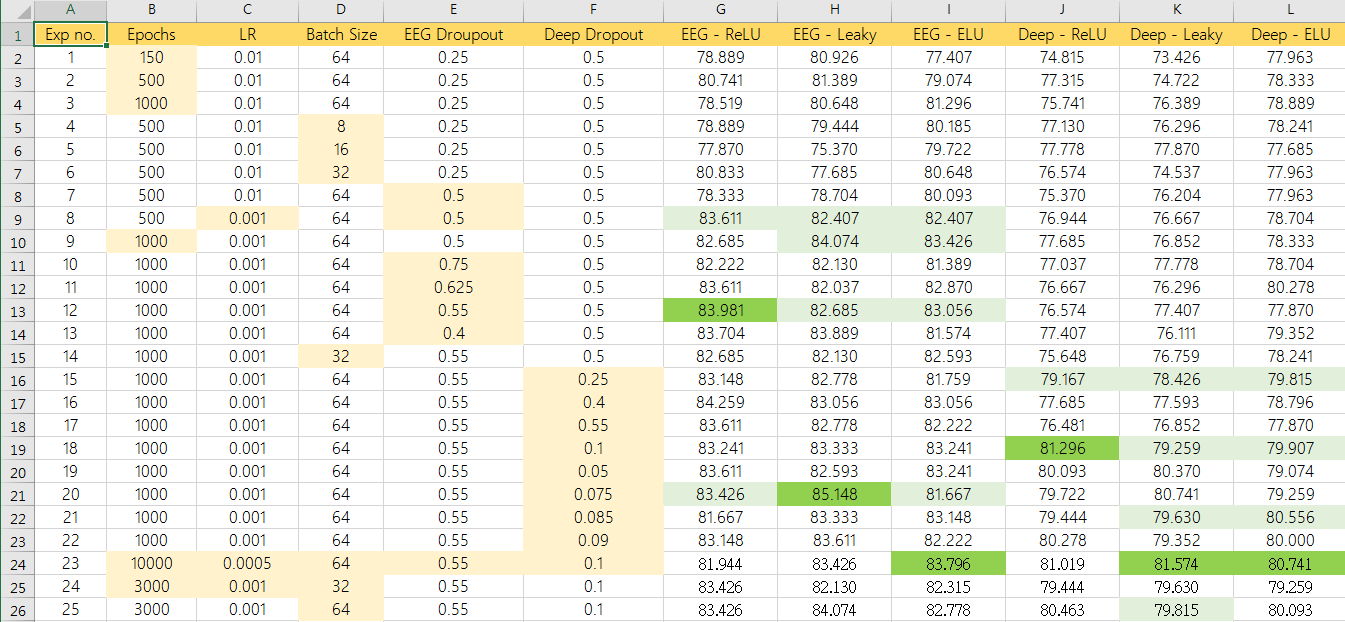
**Part 3-A: Highest testing accuracy**

For Lab 2, I totally conducted 25 experiments, the following chart is the parameters and the results of all the experiments.

The blocks that are filled with “light orange”, like this , are the parameters I modified in that experiment.

The blocks filled with “light green”, like this , are the result that had improved in comparison to the current experiment to the previous ones.

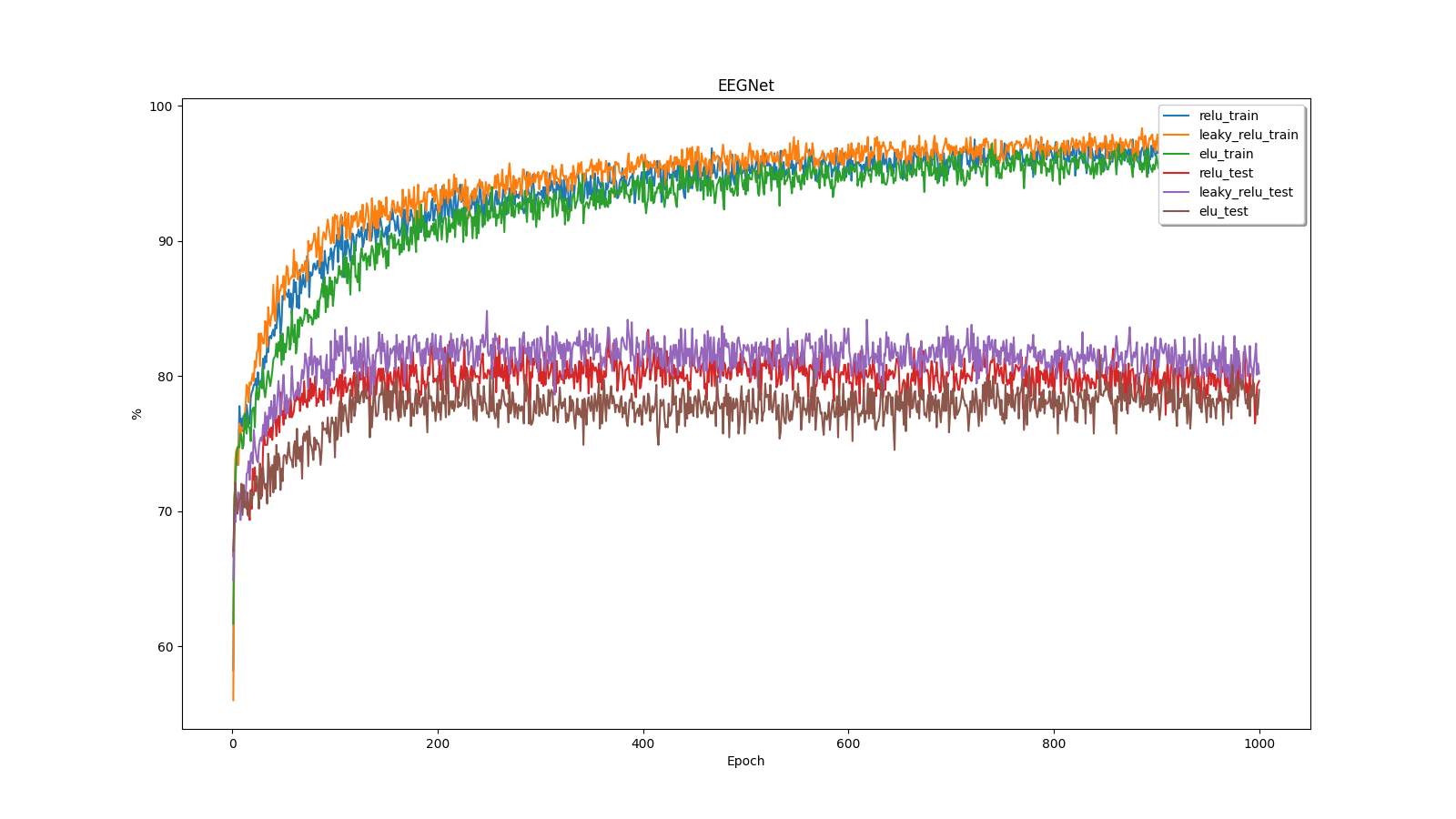
The most important, the blocks filled with “eye-catching green”, like this , are the blocks with the best accuracy result of the model and its activation function, which is noted on top of the column.



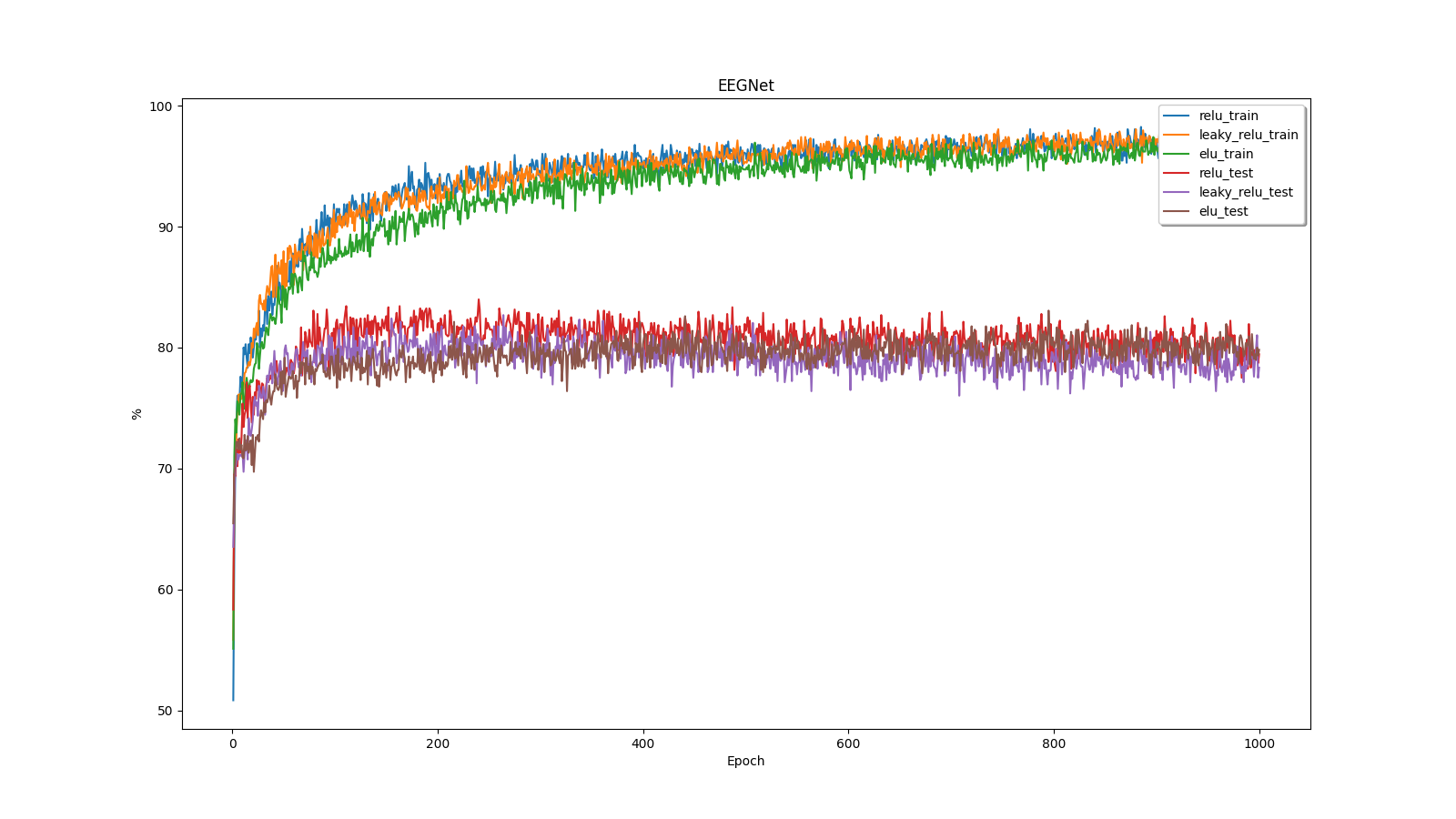
**Part 3-B: Comparison figures**

In this part, I’ll show the comparison figures of the experiments which occurred the best accuracy results I mentioned above.

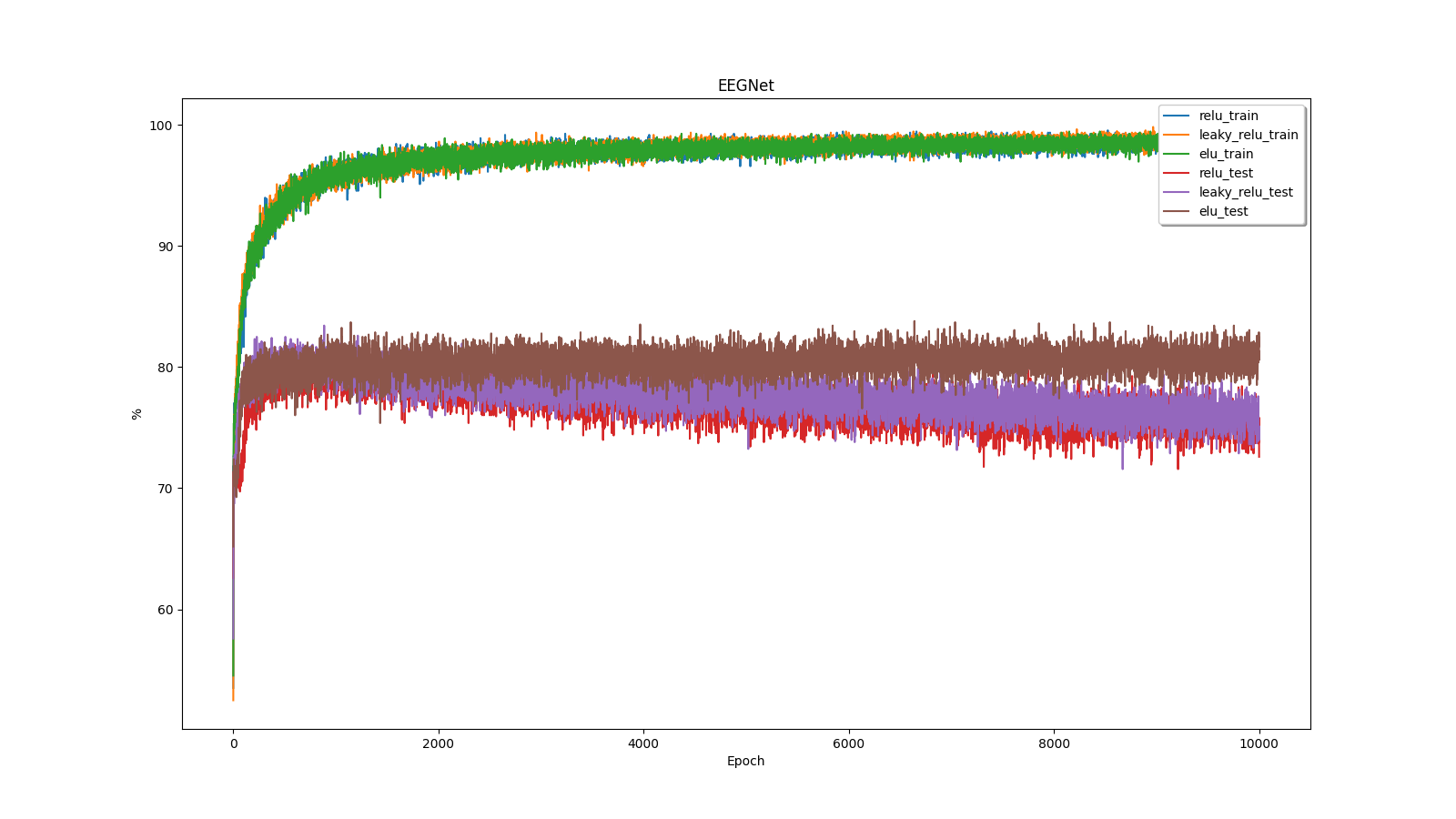
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Exp\_No. | Epochs | LR | BatchSize | EEG\_Droupout | Deep\_Dropout | EEG\_Leaky |
| 20 | 1000 | 0.001 | 64 | 0.55 | 0.075 | 85.148 % |

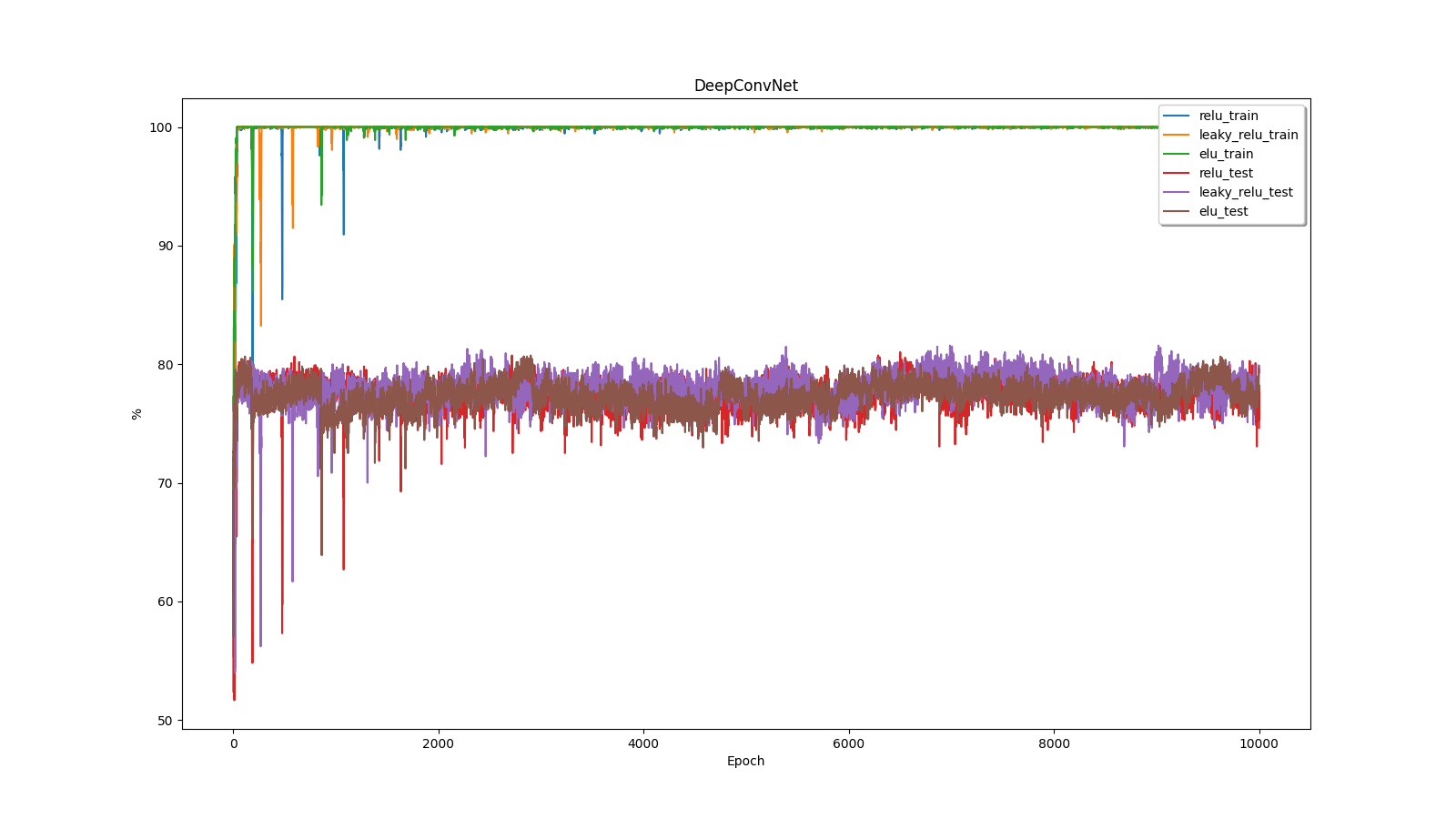


|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Exp\_No. | Epochs | LR | BatchSize | EEG\_Droupout | Deep\_Dropout | EEG\_ReLU |
| 12 | 1000 | 0.001 | 64 | 0.55 | 0.5 | 83.981 % |

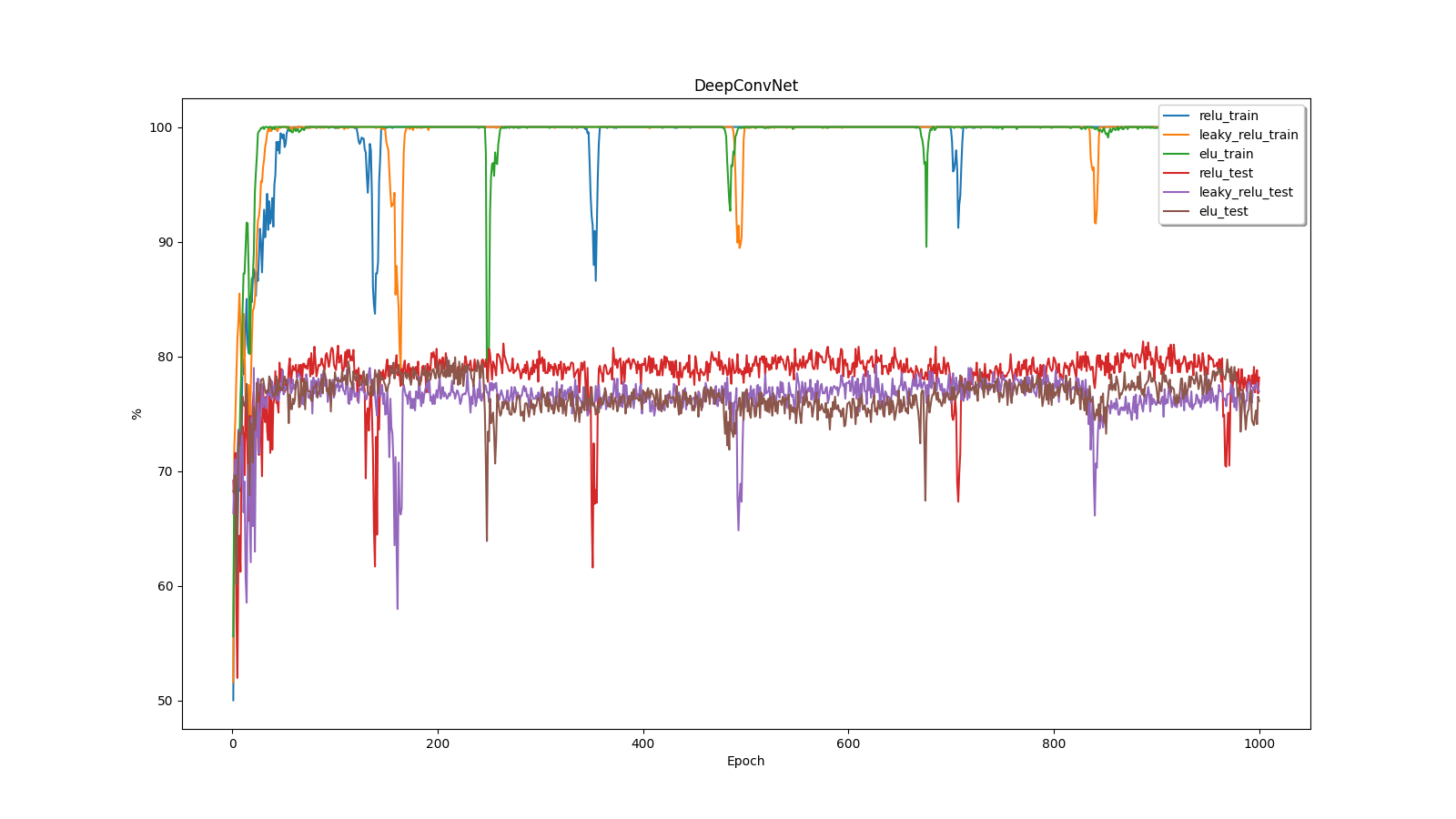


|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Exp\_No. | Epochs | LR | BatchSize | EEG\_Droupout | Deep\_Dropout | Model Name |
| 23 | 10000 | 0.0005 | 64 | 0.55 | 0.1 | 83.796 % (EEG\_ELU) |
| 23 | 10000 | 0.0005 | 64 | 0.55 | 0.1 | 81.574 % (Deep\_Leaky) |
| 23 | 10000 | 0.0005 | 64 | 0.55 | 0.1 | 80.741 % (Deep\_ELU) |





|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Exp\_No. | Epochs | LR | BatchSize | EEG\_Droupout | Deep\_Dropout | Deep\_ReLU |
| 18 | 1000 | 0.001 | 64 | 0.55 | 0.1 | 81.296 % |



**Part 4: Discussion**

**Part 4-A: Learning Rate**

Through the experiments above, we can easily find the epochs required for the neural networks to reach the loss goal increases as we lower the learning rate, which proves that higher learning rate requires less training time in the same background conditions.

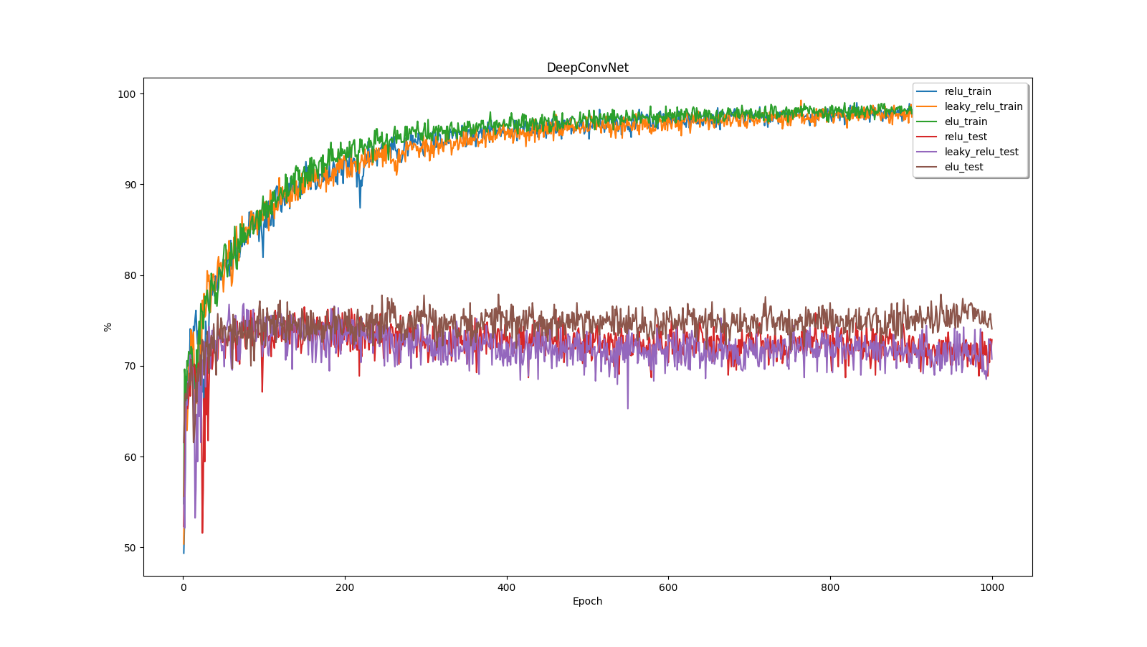
However, if we lower the learning rate to low-low value, such as the learning rate which is 0.0005 in experiment no. 23, also pulling the total epoch number to 10000, we might still get some better results, but might also overfit some of the models.

**Part 4-B: Dropouts**

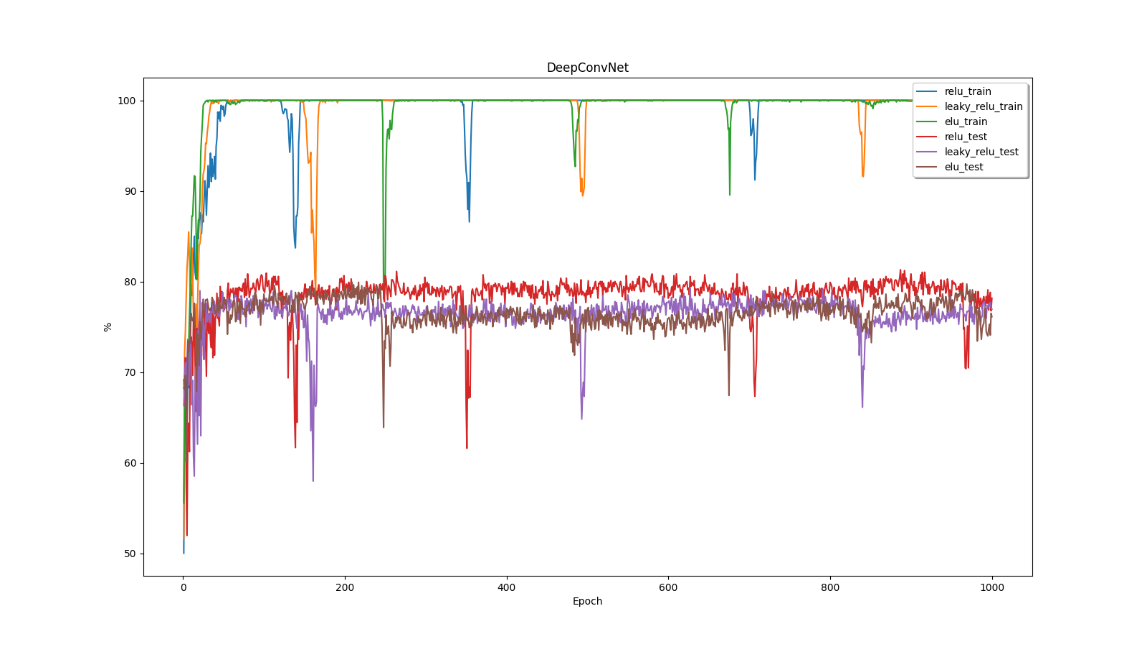
First, I would like to talk about what Dropout layers do. Dropout layers actually simply ignore neurons with the probability of “1-p” either or “p” randomly during the training phase of certain set of neurons which is chosen at random. In the ignoring process, Dropout will not ignore those neurons in part of the forward or backward pass.

The following are the accuracy curves of the experiments with different dropout probability of EEGNet (all the other parameters are the same). From the figures below, we find that when we lower the probability of dropout, the accuracy curve seems to be less stable. Since individual nodes are either dropped out of the net with probability 1-p or kept with probability p, if the probability is too low or too high, it is more likely that the model loses control.

* P = 0.55



* P = 0.1



* P = 0.05

