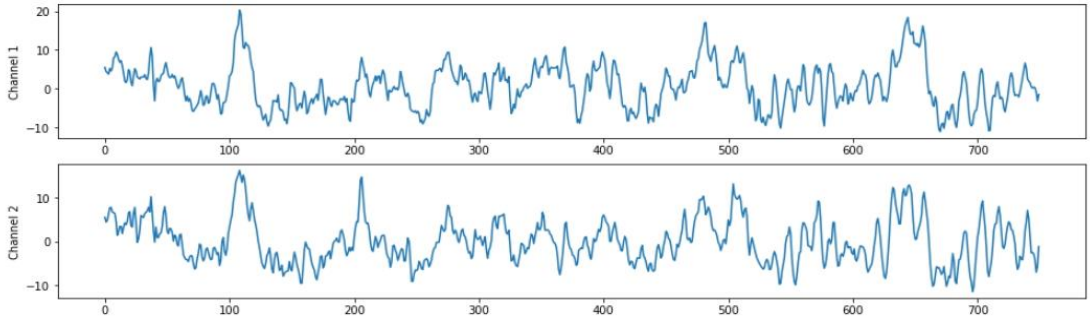
1. Introduction

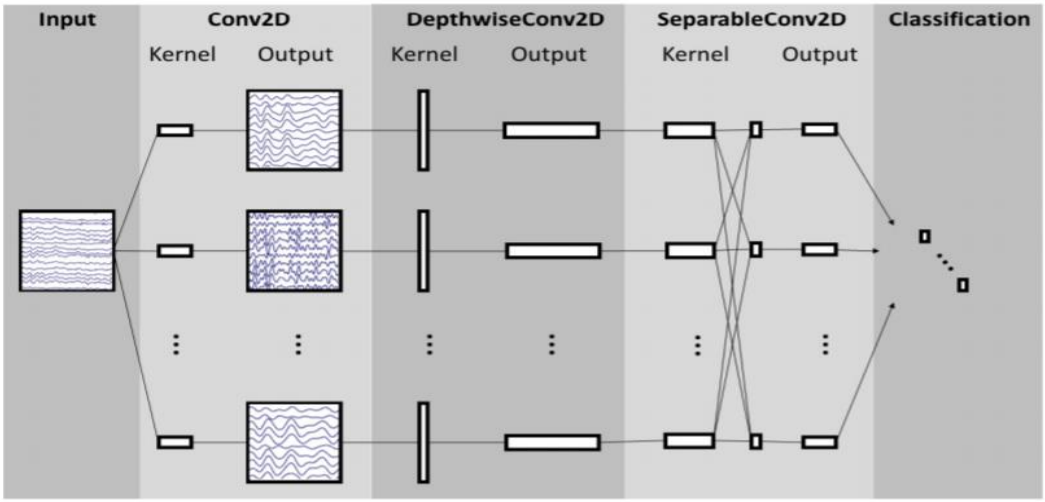
We use Non-stationary (i.e. time-varying) Brain computer interfaces (BCI) data to design two different models (EEGNet, DeepConvNet) to learn the classification (left or right). The picture below are the BCI data which were from our brain signal :

```
from dataloader import read_bci_data

train_data, train_label, test_data, test_label = read_bci_data()
fig, ax = plt.subplots(2, 1, figsize=(16, 5))
ax[0].set_ylabel("Channel 1")
ax[0].plot(np.arange(750), train_data[0,0,0,:])
ax[1].set_ylabel("Channel 2")
ax[1].plot(np.arange(750), train_data[0,0,1,:])
```

```
(1080, 1, 2, 750) (1080,) (1080, 1, 2, 750) (1080,)

[<matplotlib.lines.Line2D at 0x2032125fd30>]
```



According to the data, there are 1080 training data and 1080 test data, each training/testing data have 2 channels and each channel has 750 time samples. We will use EEGNet, a compact convolutional neural network for EEG-based BCIs and DeepConvNet.



Layer	# filters	size	# params	Activation	Options
Input		(C, T)			
Reshape		(1, C, T)			
Conv2D	25	(1, 5)	150	Linear	mode = valid, max norm = 2
Conv2D	25	(C, 1)	25 * 25 * C + 25	Linear	mode = valid, max norm = 2
BatchNorm			2 * 25		epsilon = 1e-05, momentum = 0.1
Activation				ELU	
MaxPool2D		(1, 2)			
Dropout					p = 0.5
Conv2D	50	(1, 5)	25 * 50 * C + 50	Linear	mode = valid, max norm = 2
BatchNorm			2 * 50		epsilon = 1e-05, momentum = 0.1
Activation				ELU	
MaxPool2D		(1, 2)			
Dropout					p = 0.5
Conv2D	100	(1, 5)	50 * 100 * C + 100	Linear	mode = valid, max norm = 2
BatchNorm			2 * 100		epsilon = 1e-05, momentum = 0.1
Activation				ELU	
MaxPool2D		(1, 2)			
Dropout					p = 0.5
Conv2D	200	(1, 5)	100 * 200 * C + 200	Linear	mode = valid, max norm = 2
BatchNorm			2 * 200		epsilon = 1e-05, momentum = 0.1
Activation				ELU	
MaxPool2D		(1, 2)			
Dropout					p = 0.5
Flatten					
Dense	N			softmax	max norm = 0.5

2. Experiment set up
   A. The detail of my model
      - EEGNet

```
EEGNet(
  (firstConv): Sequential(
    (0): Conv2d(1, 16, kernel_size=(1, 51), stride=(1, 1), padding=(0, 25), bias=False)
    (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (depthwiseConv): Sequential(
    (0): Conv2d(16, 32, kernel_size=(2, 1), stride=(1, 1), groups=16, bias=False)
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ELU(alpha=1.0)
    (3): AvgPool2d(kernel_size=(1, 4), stride=(1, 4), padding=0)
    (4): Dropout(p=0.3)
  )
  (separableConv): Sequential(
    (0): Conv2d(32, 32, kernel_size=(1, 15), stride=(1, 1), padding=(0, 7), bias=False)
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ELU(alpha=1.0)
    (3): AvgPool2d(kernel_size=(1, 8), stride=(1, 8), padding=0)
    (4): Dropout(p=0.3)
  )
  (classify): Sequential(
    (0): Linear(in_features=736, out_features=2, bias=True)
  )
)
```

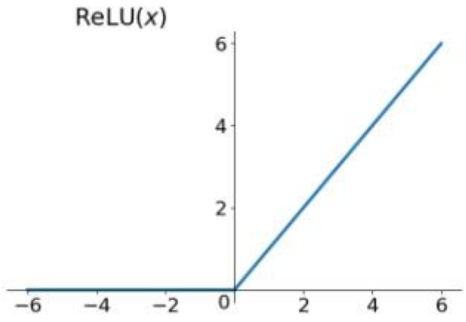
      - DeepConvNet

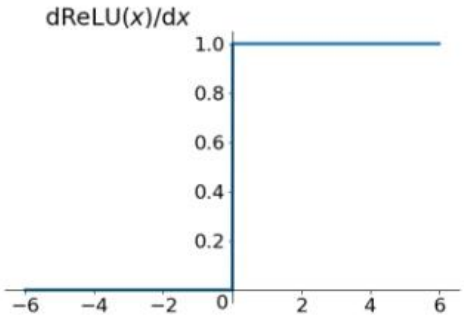
```
DeepConvNet(
  (firstLayer): Sequential(
    (0): Conv2d(1, 25, kernel_size=(1, 5), stride=(1, 1), padding=(0, 2), bias=False)
    (1): Conv2d(25, 25, kernel_size=(2, 1), stride=(1, 1), bias=False)
    (2): BatchNorm2d(25, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): ELU(alpha=1.0)
    (4): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (5): Dropout(p=0.3)
  )
  (secondLayer): Sequential(
    (0): Conv2d(25, 50, kernel_size=(1, 5), stride=(1, 1), padding=(0, 2), bias=False)
    (1): BatchNorm2d(50, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ELU(alpha=1.0)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.3)
  )
  (thirdLayer): Sequential(
    (0): Conv2d(50, 100, kernel_size=(1, 5), stride=(1, 1), padding=(0, 2), bias=False)
    (1): BatchNorm2d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ELU(alpha=1.0)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.3)
  )
  (fourthLayer): Sequential(
    (0): Conv2d(100, 200, kernel_size=(1, 5), stride=(1, 1), padding=(0, 2), bias=False)
    (1): BatchNorm2d(200, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ELU(alpha=1.0)
    (3): MaxPool2d(kernel_size=(1, 2), stride=(1, 2), padding=0, dilation=1, ceil_mode=False)
    (4): Dropout(p=0.3)
  )
  (classify): Sequential(
    (0): Linear(in_features=9200, out_features=2, bias=True)
  )
)
```

B. Explain the activation function (ReLU, Leaky ReLU, ELU)

ReLU

$$\text{ReLU} = \max(0, x)$$





Advantage:

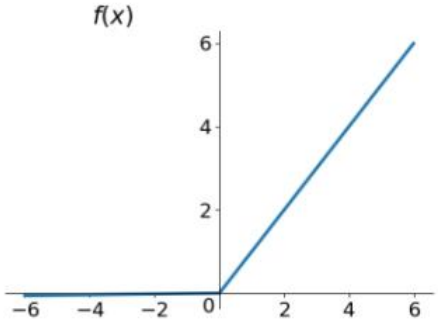
1 . Avoid gradient vanishing problem (in positive)

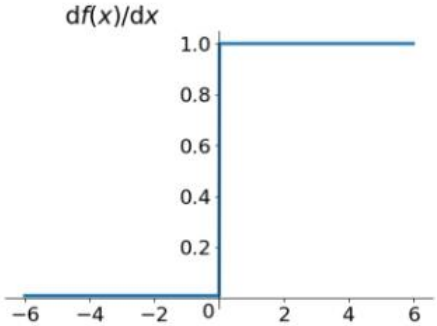
2 . Easy for compute

3 . Converge quickly than sigmoid / tanh function.

Leaky ReLU

$$f(x) = \max(0.01x, x)$$



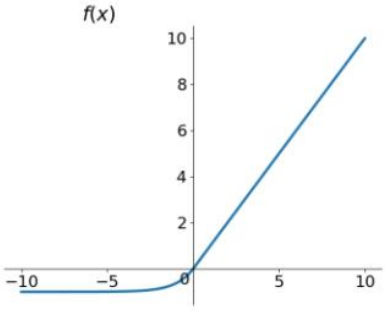


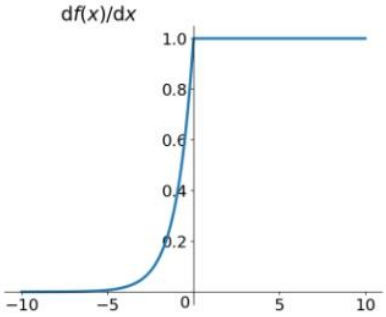
Advantage:

1 . Solve dead ReLU problem

ELU (Exponential Linear Units)

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha(e^x - 1), & \text{otherwise} \end{cases}$$

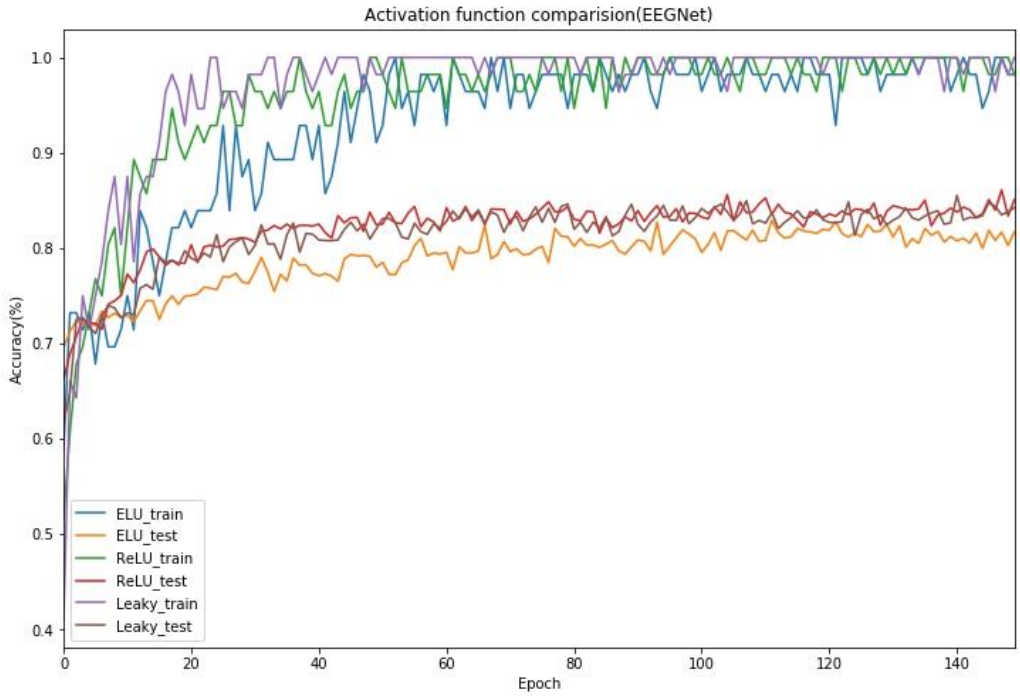


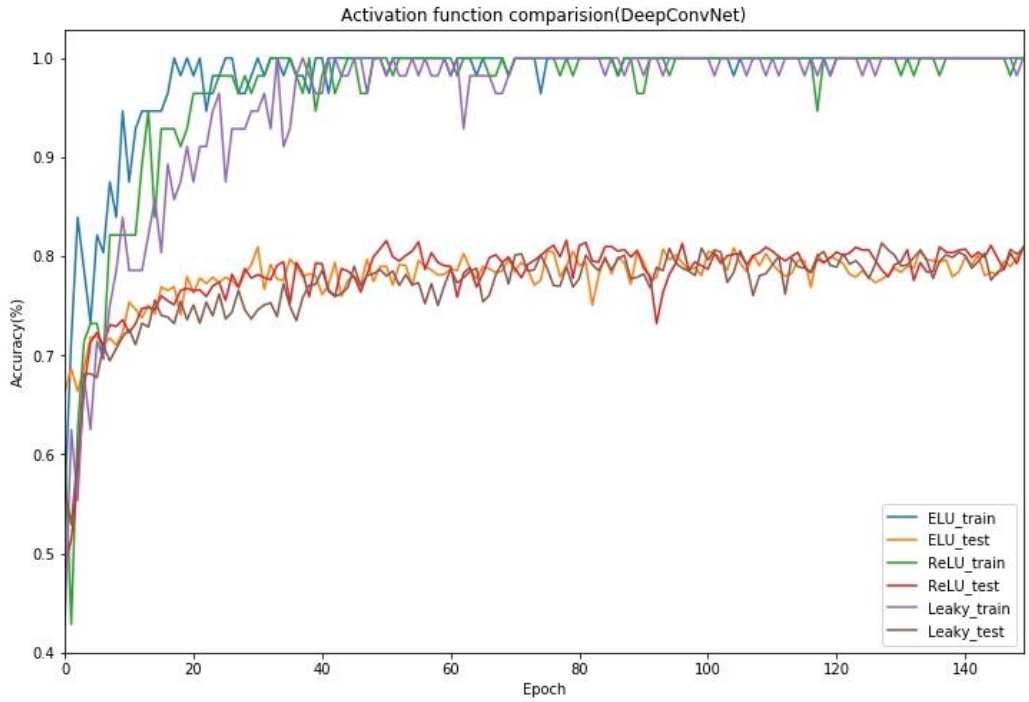


Advantage:

1 . Avoid dead ReLU problem

3. Experimental results
   A. The highest testing accuracy
      - Screenshot with two models





   B. Comparison figures

	ELU	ReLU	Leaky ReLU
EEGNet	84.03%	**86.06%**	85.60%
DeepConvNet	80.96%	**82.23%**	81.23%

In my experiment ReLU function is better than ELU function all the time, but sometimes the results are close to Leaky ReLU function.

4. Discussion

A. Try different dropout, learning rate, activation function. In my experiment, I found that when dropout is higher the accuracy will increase , but when dropout > 0.5 the acc will start to decrease.

B. From the table given in homework, the filter number is the out channel in pytorch function.

C. If we want to fixed our output tensor size from conv2d , the padding size should follow the (kernel_size – 1) / 2

D. I save the model when I complete each epoch, but each of my models are not same as the model when I was training. (I have saved the model's sate_dict() and optimizer.state_dict() but the result is quite weird)

Reload model and test each epoch

```
array([0.56238839, 0.51551339, 0.58705357, 0.66216518, 0.69263393,
       0.68337054, 0.70993304, 0.70089286, 0.70368304, 0.71618304,
       0.69620536, 0.71696429, 0.71696429, 0.71573661, 0.70714286,
       0.68828125, 0.71774554, 0.72712054, 0.72633929, 0.73694196,
       0.73225446, 0.74006696, 0.74162946, 0.74520089, 0.73928571,
       0.74598214, 0.75892857, 0.74475446, 0.77243304, 0.73102679,
       0.77667411, 0.77198661, 0.74162946, 0.77287946, 0.75223214,
       0.75658482, 0.77042411, 0.73292411, 0.76763393, 0.76517857,
       0.76049107, 0.77790179, 0.74988839, 0.775    , 0.75770089,
       0.75569196, 0.78214286, 0.75457589, 0.78013393, 0.76640625,
       0.77578125, 0.78482143, 0.75368304, 0.78169643, 0.74899554,
       0.76908482, 0.78247768, 0.74866071, 0.79508929, 0.77265625,
       0.77254464, 0.79040179, 0.74665179, 0.78482143, 0.79977679,
       0.74196429, 0.79274554, 0.76707589, 0.78404018, 0.78761161,
       0.76674107, 0.79196429, 0.79274554, 0.78125  , 0.78515625,
       0.79709821, 0.75881696, 0.78404018, 0.79386161, 0.77611607,
       0.79352679, 0.75524554, 0.79229911, 0.77410714, 0.79308036,
       0.77890625, 0.78828125, 0.7890625 , 0.77176339, 0.79308036,
       0.78794643, 0.78359375, 0.80926339, 0.77533482, 0.77578125,
       0.78995536, 0.79229911, 0.79140625, 0.79854911, 0.77767857,
       0.80446429, 0.78046875, 0.76595982, 0.79542411, 0.76986607,
       0.79419643, 0.79497768, 0.76517857, 0.80022321, 0.76830357,
       0.78950893, 0.79029018, 0.77935268, 0.78761161, 0.75535714,
       0.78716518, 0.78794643, 0.79107143, 0.78872768, 0.78169643,
       0.79308036, 0.79229911, 0.79341518, 0.80290179, 0.79185268,
       0.80133929, 0.79464286, 0.79542411, 0.78404018, 0.80011161,
       0.77935268, 0.80323661, 0.79107143, 0.78716518, 0.79575893,
       0.77622768, 0.78872768, 0.77890625, 0.79308036, 0.80837054,
       0.80167411, 0.81227679, 0.79107143, 0.80290179, 0.7796875 ,
       0.79620536, 0.7765625 , 0.79810268, 0.78515625, 0.79263393])
```

NOT reload model and test each epoch

```
array([0.6624504 , 0.6889881 , 0.70696925, 0.72544643, 0.72172619,
       0.72085813, 0.71478175, 0.74107143, 0.74466766, 0.75012401,
       0.77281746, 0.76376488, 0.77703373, 0.79588294, 0.79923115,
       0.7905506 , 0.78224206, 0.78695437, 0.78397817, 0.78422619,
       0.80344742, 0.78509425, 0.80121528, 0.80295139, 0.80121528,
       0.80208333, 0.80902778, 0.80964782, 0.8110119 , 0.80952381,
       0.80605159, 0.81783234, 0.8188244 , 0.82366071, 0.82018849,
       0.82514881, 0.82056052, 0.82390873, 0.82390873, 0.82316468,
       0.82514881, 0.81634425, 0.81039187, 0.83407738, 0.8265129 ,
       0.83184524, 0.83209325, 0.81758433, 0.8375496 , 0.82452877,
       0.82514881, 0.8375496 , 0.8265129 , 0.82576885, 0.83668155,
       0.84362599, 0.82242063, 0.8312252 , 0.82738095, 0.81795635,
       0.84238591, 0.82824901, 0.83184524, 0.84362599, 0.82899306,
       0.83841766, 0.82477679, 0.84102183, 0.84077381, 0.84040179,
       0.82514881, 0.83866567, 0.82800099, 0.83060516, 0.83556548,
       0.8421379 , 0.84858631, 0.83804563, 0.83841766, 0.84337798,
       0.83023313, 0.82862103, 0.83866567, 0.83792163, 0.81572421,
       0.83147321, 0.82142857, 0.82539683, 0.83271329, 0.82862103,
       0.83928571, 0.82800099, 0.83705357, 0.84449405, 0.83258929,
       0.83320933, 0.84126984, 0.82750496, 0.83680556, 0.83643353,
       0.83258929, 0.83060516, 0.84325397, 0.83531746, 0.85602679,
       0.82837302, 0.8484623 , 0.83705357, 0.83792163, 0.84647817,
       0.85255456, 0.83531746, 0.84598214, 0.8375496 , 0.83581349,
       0.82948909, 0.82204861, 0.83816964, 0.83556548, 0.83184524,
       0.83581349, 0.83358135, 0.83891369, 0.83494544, 0.84102183,
       0.84040179, 0.83692956, 0.84709821, 0.82403274, 0.83841766,
       0.84449405, 0.84250992, 0.8406498 , 0.84015377, 0.84102183,
       0.85032242, 0.8234127 , 0.83469742, 0.83668155, 0.84188988,
       0.83767361, 0.84300595, 0.84015377, 0.83209325, 0.83606151,
       0.85081845, 0.83990575, 0.8609871 , 0.83382937, 0.85081845])
```