Lab2 report

1. Introduction

在這個 lab 我們主要要做以下事件

(1) 實現 EEGNet 和 DeepConvNet 分類模型。

(2) 使用 BCI 競賽數據集進行訓練和測試。

(3) 測試不同 activation function 的 accuracy，包括『ReLU』、『Leaky ReLU』和『ELU』

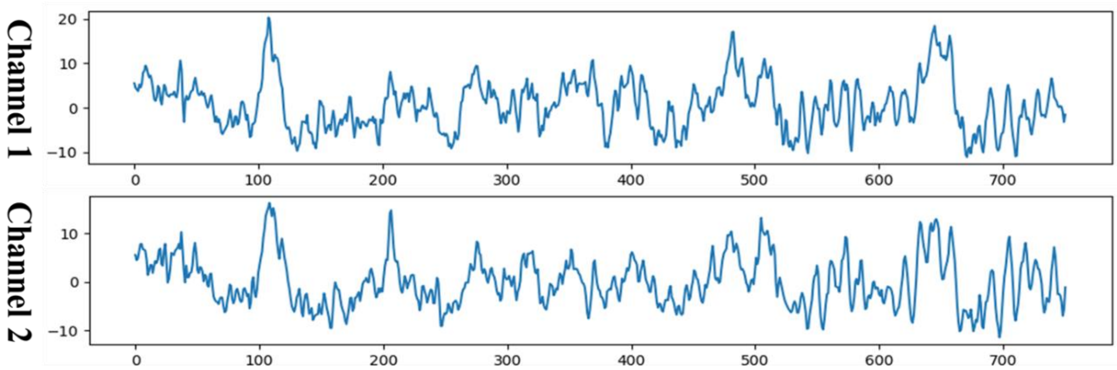
(4) Dataset 如下，每個 dataset 都有 2 個 channel 和 750 個 points

BCI Competition III – IIIb

[2 classes, 2 bipolar EEG channels]

Training data: S4b_train.npz, X11b_train.npz

Testing data: S4b_test.npz, X11b_test.npz



2. Experiment set up

A. The detail of your model

◆ **EEGNet**

```python
class EEGNet(nn.Module):
    def __init__(self, activation="relu"):
        super(EEGNet, self).__init__()
        self.activation = activation

        self.firstConv = nn.Sequential(
            nn.Conv2d(1, 16, kernel_size=(1, 51), stride=(1, 1), padding=(0, 25), bias=False),
            nn.BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )

        self.depthwiseConv = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size=(2, 1), stride=(1, 1), groups=16, bias=False),
            nn.BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
            self.get_activation(),
            nn.AvgPool2d(kernel_size=(1, 4), stride=(1, 4), padding=0),
            nn.Dropout(p=0.25)
        )

        self.separableConv = nn.Sequential(
            nn.Conv2d(32, 32, kernel_size=(1, 15), stride=(1, 1), padding=(0, 7), bias=False),
            nn.BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
            self.get_activation(),
            nn.AvgPool2d(kernel_size=(1, 8), stride=(1, 8), padding=0),
            nn.Dropout(p=0.25)
        )

        self.classify = nn.Sequential(
            nn.Linear(in_features=736, out_features=2, bias=True)
        )

    def get_activation(self):
        if self.activation == "relu":
            return nn.ReLU()
        elif self.activation == "leakyrelu":
            return nn.LeakyReLU()
        elif self.activation == "elu":
            return nn.ELU(alpha=1.0)

    def forward(self, x):
        x = self.firstConv(x)
        x = self.depthwiseConv(x)
        x = self.separableConv(x)
        x = x.view(x.size(0), -1)
        x = self.classify(x)
        return x
```

上圖的程式碼是按照助教給的 pdf 裡面的 EEGNet implementation details 實做出來的，我多增加了 get_activation()，來幫助我可以選擇要用哪種 activation function

- **DeepConvNet**

```python
class DeepConvNet(nn.Module):
    def __init__(self, activation="relu"):
        super(DeepConvNet, self).__init__()
        self.activation = activation

        self.cov1 = nn.Sequential(
            nn.Conv2d(1, 25, kernel_size=(1, 5), stride=(1, 1)),
            nn.Conv2d(25, 25, kernel_size=(2, 1), stride=(1, 1)),
            nn.BatchNorm2d(25),
            self.get_activation(),
            nn.MaxPool2d(kernel_size=(1, 2)),
            nn.Dropout(p=0.5)
        )
        self.cov2 = nn.Sequential(
            nn.Conv2d(25, 50, kernel_size=(1, 5), stride=(1, 1)),
            nn.BatchNorm2d(50),
            self.get_activation(),
            nn.MaxPool2d(kernel_size=(1, 2)),
            nn.Dropout(p=0.5)
        )
        self.cov3 = nn.Sequential(
            nn.Conv2d(50, 100, kernel_size=(1, 5), stride=(1, 1)),
            nn.BatchNorm2d(100),
            self.get_activation(),
            nn.MaxPool2d(kernel_size=(1, 2)),
            nn.Dropout(p=0.5)
        )
        self.cov4 = nn.Sequential(
            nn.Conv2d(100, 200, kernel_size=(1, 5), stride=(1, 1)),
            nn.BatchNorm2d(200),
            self.get_activation(),
            nn.MaxPool2d(kernel_size=(1, 2)),
            nn.Dropout(p=0.5),
            nn.Flatten()
        )
        self.fc = nn.Sequential(
            nn.Linear(8600, 2, bias=True)
        )

    def get_activation(self):
        if self.activation == "relu":
            return nn.ReLU()
        elif self.activation == "leakyrelu":
            return nn.LeakyReLU()
        elif self.activation == "elu":
            return nn.ELU(alpha=1.0)
```

```python
    def forward(self, x):
        x = self.cov1(x)
        x = self.cov2(x)
        x = self.cov3(x)
        x = self.cov4(x)
        x = self.fc(x)
        return x
```

上圖的程式碼是一樣按照助教給的 pdf 裡面的 DeepConvNet architecture table 實做出來的，我一樣多增加了 get_activation()，來幫助我可以選擇要用哪種 activation function

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

(1) ReLU

$$ReLU(x) = max(0,x)$$

ReLU 的主要優點是計算簡單、速度快，並且可以解決 backpropagation 時的梯度消失問題，但是他有一個缺點就是" dying ReLU problem"，也就是對於負數輸入，其導數為 0，這導致在 backpropagation 時，權

重無法更新。

(2) Leaky ReLU

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

Leaky ReLU 不會讓所有的負值都變為 0，而是給原本是負的值一個很小的正斜率 α，α 小於 1。這樣即使輸入是負的，權重也能獲得一些更新，也解決了 dying ReLU problem。

(3) ELU

$$\text{ELU}(x) = \begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

ELU 也是對 ReLU 的改良，上面的 α 是一個可以自己設定的參數。ELU 優點是在負數輸入時的輸出介於 -α 和 0 之間，且能保持一定的 gradient，解決了 dying ReLU problem。另一個優點是如果是複數輸入的話，其輸出值經過調整 α，可以接近於 0，有助於減少訓練過程中的 gradient 差異。

3. Experimental results

以下是我此實驗的 Hyper Parameters:

Batch size= 64、Learning rate = 0.001、Epochs = 300、Optimizer: Adam、

Loss function: torch.nn.CrossEntropyLoss()

A. The highest testing accuracy

◆ **Screenshot with two models**

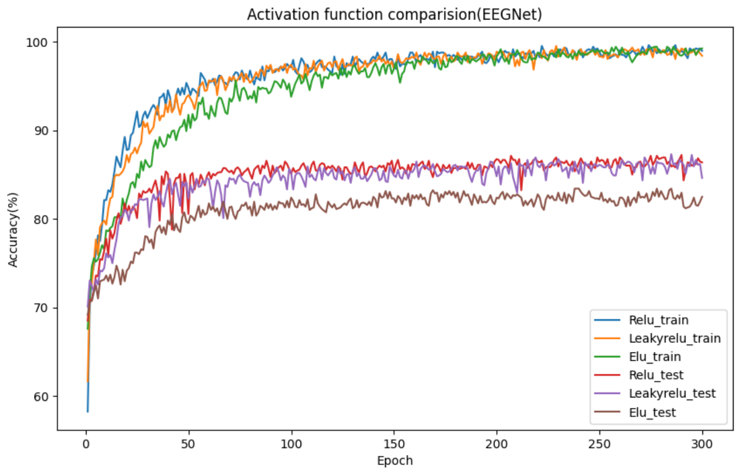
```
[Running] python -u "c:\Users\user\Desktop\Source code\test_best_models.py"
EEGNet activation function: relu, Test Accuracy: 87.22%
EEGNet activation function: leakyrelu, Test Accuracy: 87.31%
EEGNet activation function: elu, Test Accuracy: 83.43%

[Done] exited with code=0 in 4.486 seconds

[Running] python -u "c:\Users\user\Desktop\Source code\test_best_models.py"
DeepConvNet activation function: relu, Test Accuracy: 81.94%
DeepConvNet activation function: leakyrelu, Test Accuracy: 81.85%
DeepConvNet activation function: elu, Test Accuracy: 81.20%

[Done] exited with code=0 in 4.553 seconds
```

	Relu	Leaky Relu	ELU
EEGNet	87.22%	**87.31%**	83.43%
DeepConvNet	81.94%	81.85%	81.20%

從上表可以看出最高的 testing accuracy，是在 EEGNet 用 Leaky Relu 當激活函數

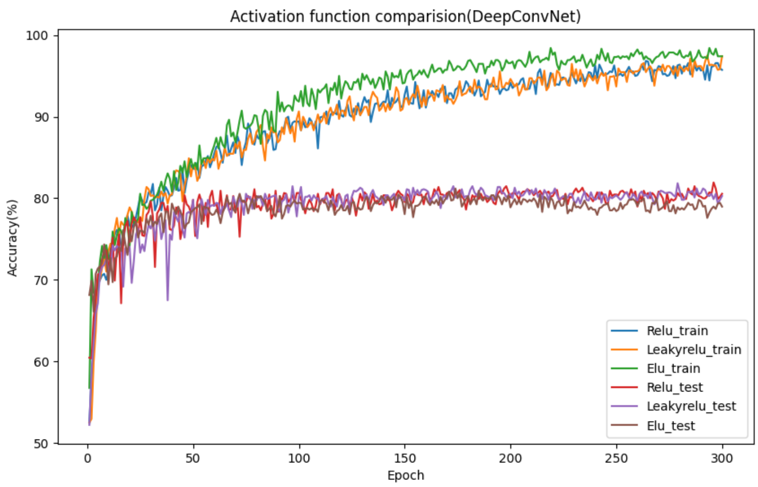
B. Comparison figures

◆ **EEGNet**



上圖是在 EEGNet 用不同 activation function 測試所有 train data 跟 test data 的 accuracy 比較圖

◆ **DeepConvNet**



上圖是在 DeepConvNet 用不同 activation function 測試所有 train data 跟 test data 的 accuracy 比較圖

4. Discussion

A. Anything you want to share

```python
def train_model(model, train_loader, test_loader, loss_function, optimizer, device, epochs):
    epoch_train_accuracy = []
    epoch_test_accuracy = []  # 用於儲存每個激活函數的測試準確度
    best_accuracy = 0
    for epoch in range(epochs):
        model.train()
        running_loss = 0.0
        correct_predictions = 0
        total_samples = 0

        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)

            optimizer.zero_grad()

            outputs = model(inputs)
            _, predicted = torch.max(outputs, 1)
            loss = loss_function(outputs, labels)

            loss.backward()
            optimizer.step()

            running_loss += loss.item()
            correct_predictions += (predicted == labels).sum().item()
            total_samples += labels.size(0)

        epoch_loss = running_loss / len(train_loader)
        epoch_accuracy = (correct_predictions / total_samples) *100
        epoch_train_accuracy.append(epoch_accuracy)

        model.eval()
        with torch.no_grad():
            correct_predictions = 0
            total_samples = 0
            for inputs, labels in test_loader:
                inputs, labels = inputs.to(device), labels.to(device)
                outputs = model(inputs)
                _, predicted = torch.max(outputs, 1)
                correct_predictions += (predicted == labels).sum().item()
                total_samples += labels.size(0)
            test_accuracy = (correct_predictions / total_samples) * 100
        epoch_test_accuracy.append(test_accuracy)
        # Save the model with the highest test accuracy
        if test_accuracy > best_accuracy:
            best_accuracy = test_accuracy
            best_model_wts = copy.deepcopy(model.state_dict())
        print(f"Epoch [{epoch+1}/{epochs}], Loss: {epoch_loss:.4f}, Train Accuracy: {epoch_accuracy:.2f}%, Test Accuracy: {test_accuracy:.2f}%")
    return epoch_train_accuracy, epoch_test_accuracy, best_model_wts
```

上圖是我訓練模型的程式碼

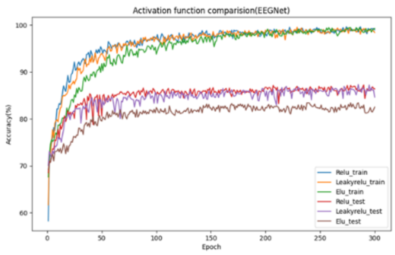
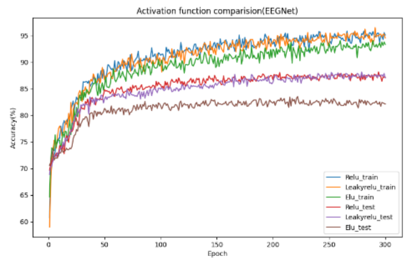
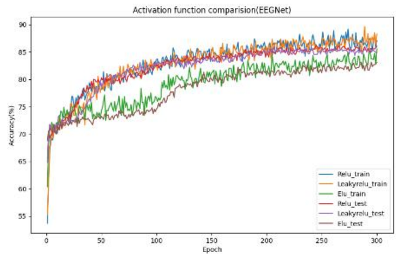
(1) 訓練模型時遇到的問題及解決方法

如果要在每個 epoch 同時進行 train model 跟 evaluate model 的話，model.train()不可以在 for epoch in range(epochs)上面，要在迴圈裡面每個 epoch 開始時的地方加上，如果 model.train()放在 for epoch in range(epochs)上面的話，會導致在第一個 epoch 訓練完模型後，因為 model.eval()一直存在，之後 epoch 訓練都會省略 dropout 層跟 batch normalization，導致模型很快就 overfitting。我一開始就遇到上面的問題。

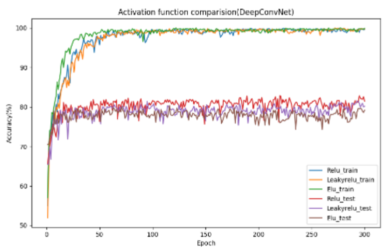
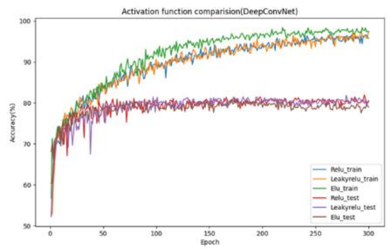
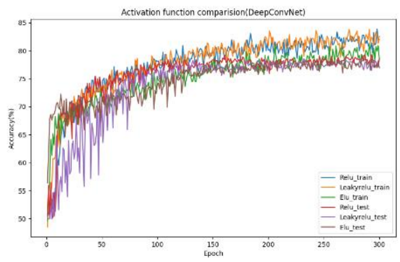
(2) 儲存模型參數時遇到的問題及解決方法

我一開始在 accuracy 最高時用 best_model_wts = model.state_dict()去儲存的當下 epoch 的模型參數，但是發現這個方法，有可能導致模型儲存的參數不是當下 epoch 的模型參數，所以後來改成用 best_model_wts = copy.deepcopy(model.state_dict()) 這個方法來儲存參數，最後經過測試發現這個方法可以把當下模型的參數完整拷貝到 best_model_wts 這個變數裡面，不會像第一個方法可能儲存到的不是當下模型的參數，而是存成最後一個 epoch 的模型參數。

(3) EEGNet 不同 dropout rate 的 accuracy 比較

<table>
<tr><th>dropout rate = 0.25 (default)</th><th>dropout rate = 0.5</th><th>dropout rate = 0.75</th></tr>
<tr><td>EEGNet activation function: relu, Test Accuracy: 87.22%
EEGNet activation function: leakyrelu, Test Accuracy: 87.31%
EEGNet activation function: elu, Test Accuracy: 83.43%</td><td>EEGNet activation function: relu, Test Accuracy: 88.15%
EEGNet activation function: leakyrelu, Test Accuracy: 88.15%
EEGNet activation function: elu, Test Accuracy: 83.52%</td><td>EEGNet activation function: relu, Test Accuracy: 86.20%
EEGNet activation function: leakyrelu, Test Accuracy: 85.83%
EEGNet activation function: elu, Test Accuracy: 83.15%</td></tr>
<tr><td>
</td><td>
</td><td>
</td></tr>
</table>

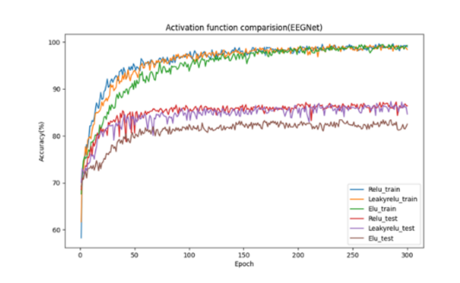
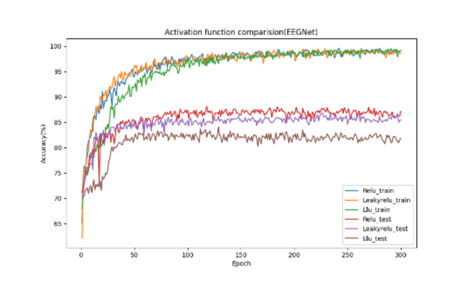
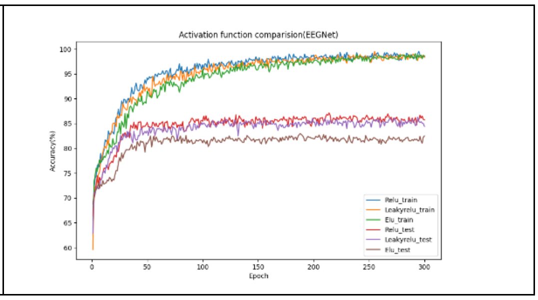
(4) DeepConvNet 不同 dropout rate 的 accuracy 比較

<table>
<tr><th>dropout rate = 0.25</th><th>dropout rate = 0.5 (default)</th><th>dropout rate = 0.75</th></tr>
<tr><td>DeepConvNet activation function: relu, Test Accuracy: 82.87%
DeepConvNet activation function: leakyrelu, Test Accuracy: 81.48%
DeepConvNet activation function: elu, Test Accuracy: 80.37%</td><td>DeepConvNet activation function: relu, Test Accuracy: 81.94%
DeepConvNet activation function: leakyrelu, Test Accuracy: 81.85%
DeepConvNet activation function: elu, Test Accuracy: 81.20%</td><td>DeepConvNet activation function: relu, Test Accuracy: 79.26%
DeepConvNet activation function: leakyrelu, Test Accuracy: 78.52%
DeepConvNet activation function: elu, Test Accuracy: 78.61%</td></tr>
<tr><td>
</td><td>
</td><td>
</td></tr>
</table>

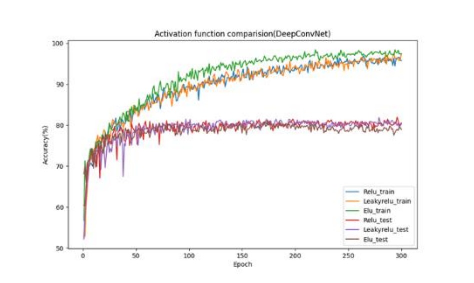
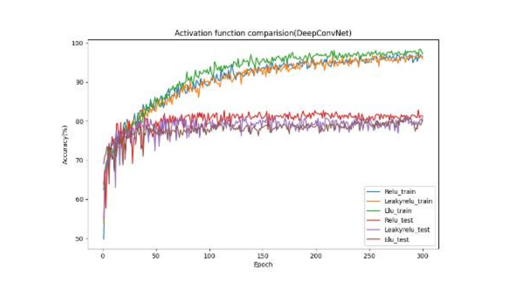
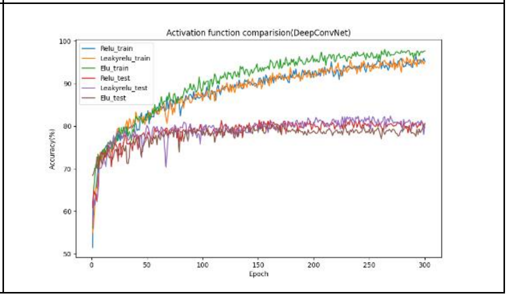
從(3)、(4) dropout rate = 0.75 的 accuracy 比較圖可以看到如果 dropout rate 太大的話大部分的神經元都被隨機地 dropout，將導致模型的有效參數減少，可能導致模型 underfitting，尤其可以看到 DeepConvNet dropout rate = 0.75 的 accuracy 曲線前期震盪很嚴重。另外從(3)、(4) dropout rate = 0.25 的 accuracy 比較圖可以看到，可能因為 DeepConvNet 的 layers 數較多，導致 dropout rate 設比較小的話，training data 的 accuracy 很快就會達到接近 100%，可是 testing data 卻還是維持在 80%左右，如果我 epoch 數再設更多的話，可能會造成 overfitting，至於 EEGNet 可能 layers 數較少，反而比較不容易 overfitting。

(5) EEGNet 不同 Batch size 的 accuracy 比較

<table>
<tr><th>Batch size = 64 (default)</th><th>Batch size = 32</th><th>Batch size = 128</th></tr>
<tr><td>EEGNet activation function: relu, Test Accuracy: 87.22%
EEGNet activation function: leakyrelu, Test Accuracy: 87.31%
EEGNet activation function: elu, Test Accuracy: 83.43%</td><td>EEGNet activation function: relu, Test Accuracy: 88.15%
EEGNet activation function: leakyrelu, Test Accuracy: 86.94%
EEGNet activation function: elu, Test Accuracy: 84.07%</td><td>EEGNet activation function: relu, Test Accuracy: 87.13%
EEGNet activation function: leakyrelu, Test Accuracy: 86.30%
EEGNet activation function: elu, Test Accuracy: 82.96%</td></tr>
</table>

(6) DeepConvNet 不同 Batch size 的 accuracy 比較

Batch size = 64 (default)	Batch size = 32	Batch size = 128
DeepConvNet activation function: relu, Test Accuracy: 81.94% DeepConvNet activation function: leakyrelu, Test Accuracy: 81.85% DeepConvNet activation function: elu, Test Accuracy: 81.20%	DeepConvNet activation function: relu, Test Accuracy: 82.87% DeepConvNet activation function: leakyrelu, Test Accuracy: 81.48% DeepConvNet activation function: elu, Test Accuracy: 80.83%	DeepConvNet activation function: relu, Test Accuracy: 81.57% DeepConvNet activation function: leakyrelu, Test Accuracy: 82.41% DeepConvNet activation function: elu, Test Accuracy: 80.37%
 	 	

較大的 Batch size 在每次 weight update 時使用更多的樣本，可以提供更穩定的 gradient descent 方向，但也可能使模型陷入局部最優解。另一方面，較小的 Batch size 會導致更高的變異性，可能有助於模型跳出局部最優解，但也可能使學習過程更加不穩定。從(5)、(6) 的 accuracy 比較圖可以看出 32-128 的 Batch size 對這個 dataset 來說部會太大也不會太小。

5. Extra

A. Implement any other classification model

我實測 ShallowcovNet，以下是我此實驗的 Hyper Parameters:

Batch size= 64、Learning rate = 0.001 、Epochs = 300、Optimizer: Adam、Loss function: torch.nn.CrossEntropyLoss()

以下是我模型的程式碼

```python
class ShallowcovNet(nn.Module):
    def __init__(self, activation="relu"):
        super(ShallowcovNet, self).__init__()
        self.activation = activation

        self.cov1 = nn.Sequential(
            nn.Conv2d(1, 64, kernel_size=(1, 25)),
            self.get_activation(),
            nn.BatchNorm2d(64, eps=1e-05, momentum=0.1),
            nn.MaxPool2d(kernel_size=(1, 2), stride=(1, 2)),
            nn.Dropout(p=0.5)
            )
        self.cov2 = nn.Sequential(
            nn.Conv2d(64, 128, kernel_size=(2, 25), stride=(1, 1)),
            self.get_activation(),
            nn.BatchNorm2d(128, eps=1e-05, momentum=0.1),
            nn.MaxPool2d(kernel_size=(1, 2), stride=(1, 2)),
            nn.Dropout(p=0.5)
            )
        self.fc = nn.Sequential(
            nn.Linear(21632, 2)
            )

    def get_activation(self):
        if self.activation == "relu":
            return nn.ReLU()
        elif self.activation == "leakyrelu":
            return nn.LeakyReLU()
        elif self.activation == "elu":
            return nn.ELU(alpha=1.0)

    def forward(self, x):
        x = self.cov1(x)
        x = self.cov2(x)
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        return x
```

以下是我三種 activation function 的 testing accuracy

```
ShallowcovNet activation function: relu, Test Accuracy: 81.20%
ShallowcovNet activation function: leakyrelu, Test Accuracy: 81.30%
ShallowcovNet activation function: elu, Test Accuracy: 81.94%
```

以下是我的 activation function Comparison figures