Deep Learning and Practice Report
Lab4-1: EEG Classification

1 Introduction

Brain **c**omputer **i**nterface (BCI) has provided people a way to communicate with computers using neural signal. This neural signal is generally chosen from various types of **e**lectro**e**ncephalo**g**ram (EEG) signals. In recent years, convolutional neural networks have been widely applied to EEG-based BCIs for feature extraction and classification. In this lab, we are asked to implement two simple models for EEG classification, which are EEGNet and DeepConvNet. The model is trained and evaluated on a BCI competition dataset. We also need to compare the testing results using different activation functions, including ReLU, Leaky ReLU and ELU.

2 Experiment Setups

2.1 The details of your model

2.1.1 EEGNet

Figure 1 illustrates the architecture of EEGNet. Different from traditional CNN, EEGNet divides the step of feature extraction into roughly three parts. The network first uses a normal 2D CNN (second column) to generate several feature maps from the input. Then, a depthwise convolution (third column) is used to learn the frequency-specific spatial filters. It's worthy to note that the convolution is connected to each feature map individually. Afterwards, the separable convolution (fourth column) separately learns a temporal summary for each feature map, a pointwise convolution is then adopted to mix all feature maps together. Finally, the model performs classification based on the output features.

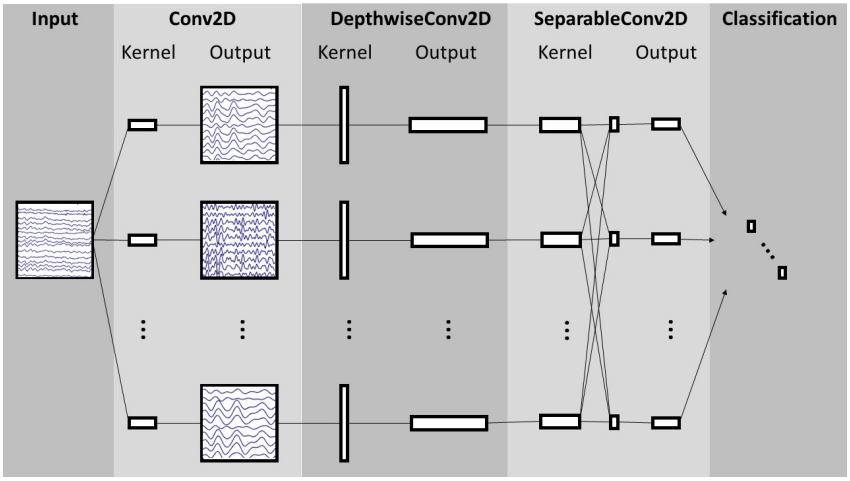

Figure 1: Overall architecture of EEGNet.

Model	Layer	Input Channels	Output Channels	Kernel Size
EEGNet	Conv2D	1	16	$(1, 51)$
	DepthwiseConv2D	16	32	$(2, 1)$
	SeparableConv2D	32	32	$(1, 15)$
DeepConvNet	Conv2D	1	25	$(1, 5)$
	Conv-Block-1	25	25	$(2, 1)$
	Conv-Block-2	25	50	$(1, 5)$
	Conv-Block-3	50	100	$(1, 5)$
	Conv-Block-4	100	200	$(1, 5)$

Table 1: Detailed parameters for implementing EEGNet and DeepConvNet.

2.1.2 DeepConvNet

The table shown in figure 2 describes the detailed architecture of DeepConvNet. I designed the model based on the table, where $C = 2$, $T = 750$ and $N = 2$. The model consists of a normal 2D CNN, four convolution blocks and a fully connected layer for classification. Each convolution block contains a 2D CNN, followed by a batch normalization, an activation function, a 2D max pooling and a dropout layer.

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

Figure 2: Overall architecture of DeepConvNet.

2.2 Explain the activation function (ReLU, Leaky ReLU, ELU)

In this lab, three kinds of activation functions are implemented, including ReLU, Leaky ReLU and ELU. Figure 3 shows the curve of each activation function.

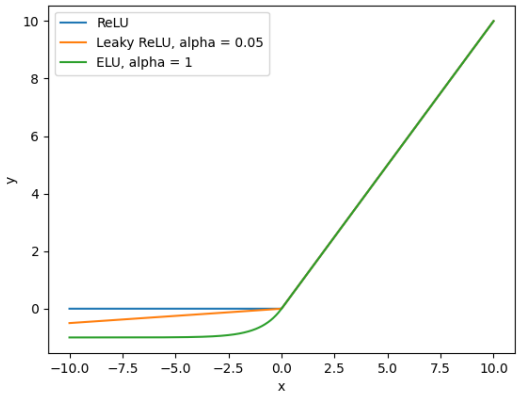

Figure 3: Curves of different activation functions.

2.2.1 ReLU

Rectified **l**inear **u**nits (ReLU) is a simple activation function commonly used in deep learning models. The formula for ReLU function is as follows.

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

2.2.2 Leaky ReLU

Despite the simple calculation of ReLU, there still exists some problems. Since the function contributes zero value for negative inputs ($x < 0$), the gradient will be zero. This makes those neurons going into that state stop responding to the loss function, which is known as the dying ReLU problem. Therefore, Leaky ReLU is proposed to address this issue. It is a variant of ReLU and the formula is as follows.

$$\text{LeakyReLU}(x, \alpha) = \begin{cases} x, & \text{if } x \geq 0 \\ \alpha x, & \text{if } x \leq 0 \end{cases}, \text{ where } 0 \leq \alpha \leq 1.$$

This provides a small negative value for negative inputs ($x < 0$), which can somehow mitigate the problem mentioned above.

2.2.3 ELU

ELU is another variant of ReLU aiming to tackle dying ReLU problem. Its formula is as follows.

$$\text{ELU}(x, \alpha) = \begin{cases} x, & \text{if } x > 0 \\ \alpha e^x - 1, & \text{if } x \leq 0 \end{cases}, \text{ where } 0 \leq \alpha \leq 1.$$

Learning Rate	EEGNet			DeepConvNet		
	ReLU	Leaky Relu	ELU	ReLU	Leaky Relu	ELU
1.0×10^{-2}	86.39%	86.02%	85%	81.48%	82.69%	81.57%
5.0×10^{-3}	84.91%	86.11%	84.81%	81.30%	82.96%	81.11%
2.0×10^{-3}	**87.87%**	86.67%	82.41%	82.41%	80.93%	80.74%
1.0×10^{-3}	**87.87%**	85.09%	85.09%	81.39%	82.78%	81.11%
5.0×10^{-4}	86.39%	86.57%	82.59%	81.76%	82.69%	81.57%
2.0×10^{-4}	**87.87%**	**88.15%**	83.98%	81.76%	81.39%	80.74%
1.0×10^{-4}	**88.15%**	86.76%	84.72%	81.94%	82.22%	80.83%

Table 2: Testing accuracy under different learning rates.

This function also gives negative inputs ($x < 0$) a small negative value. Different from ReLU and Leaky ReLU, ELU has a smooth variation around $x = 0$, making the function differentiable. Also, it provides non-linearity and may enhance the model's ability for classification.

3 Experimental Results

3.1 The highest testing accuracy

3.1.1 Screenshot with two models

Figure 4 shows the screenshots of the training results . The results include training loss, training accuracy and testing accuracy. The best testing accuracy is displayed in the last line.

```
Using leaky_relu as activation function...
Building model...

Building trainer...

Start training...
Epoch:    0, Train Loss: 0.6357, Train Accuracy: 0.6417, Test Accuracy: 0.6704
Epoch:   10, Train Loss: 0.4227, Train Accuracy: 0.8028, Test Accuracy: 0.7370
Epoch:   20, Train Loss: 0.2956, Train Accuracy: 0.8741, Test Accuracy: 0.7963
Epoch:   30, Train Loss: 0.2251, Train Accuracy: 0.9185, Test Accuracy: 0.8380
Epoch:   40, Train Loss: 0.1869, Train Accuracy: 0.9296, Test Accuracy: 0.8500
Epoch:   50, Train Loss: 0.1548, Train Accuracy: 0.9389, Test Accuracy: 0.8472
Epoch:   60, Train Loss: 0.1521, Train Accuracy: 0.9398, Test Accuracy: 0.8648
Epoch:   70, Train Loss: 0.1088, Train Accuracy: 0.9620, Test Accuracy: 0.8500
Epoch:   80, Train Loss: 0.1080, Train Accuracy: 0.9648, Test Accuracy: 0.8574
Epoch:   90, Train Loss: 0.0913, Train Accuracy: 0.9704, Test Accuracy: 0.8639
Epoch:  100, Train Loss: 0.0842, Train Accuracy: 0.9722, Test Accuracy: 0.8546
Epoch:  110, Train Loss: 0.0707, Train Accuracy: 0.9796, Test Accuracy: 0.8583
Epoch:  120, Train Loss: 0.0588, Train Accuracy: 0.9815, Test Accuracy: 0.8620
Epoch:  130, Train Loss: 0.0755, Train Accuracy: 0.9731, Test Accuracy: 0.8648
Epoch:  140, Train Loss: 0.0556, Train Accuracy: 0.9843, Test Accuracy: 0.8685
Epoch:  150, Train Loss: 0.0575, Train Accuracy: 0.9778, Test Accuracy: 0.8630
Epoch:  160, Train Loss: 0.0655, Train Accuracy: 0.9750, Test Accuracy: 0.8713
Epoch:  170, Train Loss: 0.0589, Train Accuracy: 0.9778, Test Accuracy: 0.8574
Epoch:  180, Train Loss: 0.0630, Train Accuracy: 0.9815, Test Accuracy: 0.8648
Epoch:  190, Train Loss: 0.0430, Train Accuracy: 0.9861, Test Accuracy: 0.8722
Epoch:  200, Train Loss: 0.0427, Train Accuracy: 0.9843, Test Accuracy: 0.8546
Epoch:  210, Train Loss: 0.0502, Train Accuracy: 0.9870, Test Accuracy: 0.8713
Epoch:  220, Train Loss: 0.0510, Train Accuracy: 0.9833, Test Accuracy: 0.8778
Epoch:  230, Train Loss: 0.0341, Train Accuracy: 0.9898, Test Accuracy: 0.8676
Epoch:  240, Train Loss: 0.0357, Train Accuracy: 0.9889, Test Accuracy: 0.8639
Epoch:  250, Train Loss: 0.0417, Train Accuracy: 0.9889, Test Accuracy: 0.8546
Epoch:  260, Train Loss: 0.0721, Train Accuracy: 0.9815, Test Accuracy: 0.8583
Epoch:  270, Train Loss: 0.0422, Train Accuracy: 0.9843, Test Accuracy: 0.8657
Epoch:  280, Train Loss: 0.0292, Train Accuracy: 0.9907, Test Accuracy: 0.8667
Epoch:  290, Train Loss: 0.0326, Train Accuracy: 0.9935, Test Accuracy: 0.8574
Best Epoch: 213, Test Accuracy: 0.8815
```

(a) EEGNet

```
Using leaky_relu as activation function...
Building model...

Building trainer...

Start training...
Epoch:    0, Train Loss: 1.0572, Train Accuracy: 0.5167, Test Accuracy: 0.6111
Epoch:   10, Train Loss: 0.5937, Train Accuracy: 0.7102, Test Accuracy: 0.7556
Epoch:   20, Train Loss: 0.5193, Train Accuracy: 0.7583, Test Accuracy: 0.7546
Epoch:   30, Train Loss: 0.4883, Train Accuracy: 0.7796, Test Accuracy: 0.6833
Epoch:   40, Train Loss: 0.3917, Train Accuracy: 0.8148, Test Accuracy: 0.7880
Epoch:   50, Train Loss: 0.3553, Train Accuracy: 0.8472, Test Accuracy: 0.7806
Epoch:   60, Train Loss: 0.3499, Train Accuracy: 0.8481, Test Accuracy: 0.7907
Epoch:   70, Train Loss: 0.3084, Train Accuracy: 0.8685, Test Accuracy: 0.7880
Epoch:   80, Train Loss: 0.2835, Train Accuracy: 0.8787, Test Accuracy: 0.8065
Epoch:   90, Train Loss: 0.2835, Train Accuracy: 0.8833, Test Accuracy: 0.8037
Epoch:  100, Train Loss: 0.2897, Train Accuracy: 0.8806, Test Accuracy: 0.8083
Epoch:  110, Train Loss: 0.2410, Train Accuracy: 0.8944, Test Accuracy: 0.8102
Epoch:  120, Train Loss: 0.2450, Train Accuracy: 0.9046, Test Accuracy: 0.7889
Epoch:  130, Train Loss: 0.2116, Train Accuracy: 0.9074, Test Accuracy: 0.8130
Epoch:  140, Train Loss: 0.2358, Train Accuracy: 0.8954, Test Accuracy: 0.7963
Epoch:  150, Train Loss: 0.1850, Train Accuracy: 0.9213, Test Accuracy: 0.8083
Epoch:  160, Train Loss: 0.1782, Train Accuracy: 0.9333, Test Accuracy: 0.8083
Epoch:  170, Train Loss: 0.1797, Train Accuracy: 0.9315, Test Accuracy: 0.8102
Epoch:  180, Train Loss: 0.1629, Train Accuracy: 0.9287, Test Accuracy: 0.8074
Epoch:  190, Train Loss: 0.1720, Train Accuracy: 0.9222, Test Accuracy: 0.8074
Epoch:  200, Train Loss: 0.1544, Train Accuracy: 0.9407, Test Accuracy: 0.8102
Epoch:  210, Train Loss: 0.1293, Train Accuracy: 0.9491, Test Accuracy: 0.8130
Epoch:  220, Train Loss: 0.1161, Train Accuracy: 0.9537, Test Accuracy: 0.8037
Epoch:  230, Train Loss: 0.1508, Train Accuracy: 0.9315, Test Accuracy: 0.8111
Epoch:  240, Train Loss: 0.1720, Train Accuracy: 0.9315, Test Accuracy: 0.8083
Epoch:  250, Train Loss: 0.1316, Train Accuracy: 0.9537, Test Accuracy: 0.8259
Epoch:  260, Train Loss: 0.1135, Train Accuracy: 0.9528, Test Accuracy: 0.8213
Epoch:  270, Train Loss: 0.1028, Train Accuracy: 0.9574, Test Accuracy: 0.8176
Epoch:  280, Train Loss: 0.1146, Train Accuracy: 0.9593, Test Accuracy: 0.8139
Epoch:  290, Train Loss: 0.1094, Train Accuracy: 0.9611, Test Accuracy: 0.8269
Best Epoch: 294, Test Accuracy: 0.8296
```

(b) DeepConvNet

Figure 4: Screenshot of two models with highest testing accuracy.

3.1.2 Anything you want to present

Table 2 demonstrates the testing accuracy using different learning rates. Each model is trained for 300 epochs in total and the best testing accuracy is reported. The batch size is 64. Those who achieved 87% are marked in bold, and the highest accuracy for both models are highlighted. With ELU as the activation function, both models have the lowest testing accuracy. On the other hand, models with ReLU and Leaky ReLU have similar behavior and they tend to fit the test dataset better. Moreover, it is obvious that EEGNet outperforms DeepConvNet significantly. Therefore, we can infer that depthwise separable convolution can effectively capture more information from the EEG signals compared to normal deep convolutional networks.

3.2 Comparison figures

Figure 5 shows the learning curves for both models using different activation functions. We can see that the testing accuracy of all models stop increasing after training for about 50 epochs while the training accuracy keeps getting higher, indicating the models are overfitting on the training dataset. Also, models trained with ELU seem to perform the worst since the gap between their training and testing accuracy are the greatest among all models.

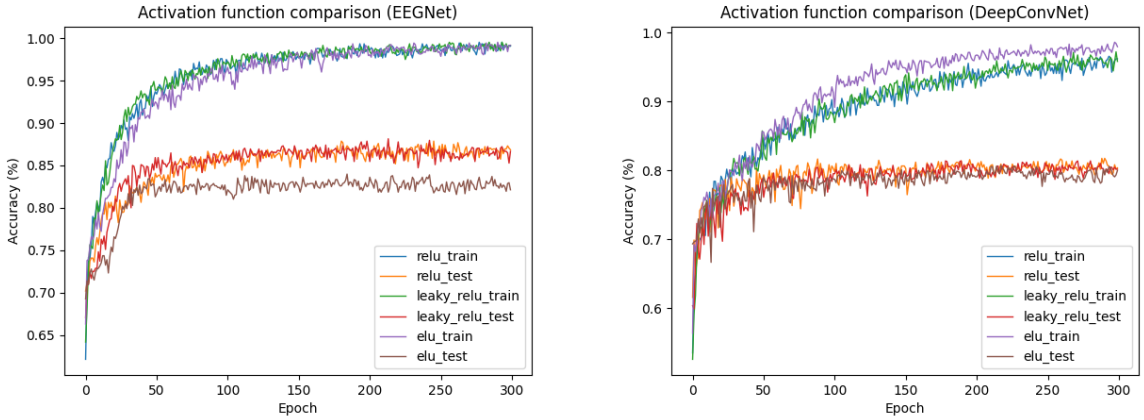

Figure 5: Learning curves for EEGNet and DeepConvNet.

4 Discussion

4.1 Anything you want to share

Batch normalization and dropout are widely used techniques while training deep neural networks. Both techniques aim to train the neural network in a faster and more stable manner. Batch normalization scales the means and variances of each batch of data and can prevent the input from falling into the saturation region of some activation functions (e.g., sigmoid, tanh), resulting in the gradient vanishing problem. By randomly dropping out some neurons during training, dropout not only reduces the training time but also drastically reduce the possibility of the model overfitting to the training dataset.

Batch Norm	Dropout	EEGNet			DeepConvNet		
		ReLU	Leaky Relu	ELU	ReLU	Leaky Relu	ELU
V	V	**86.75%**	86.22%	84%	83.46%	83.46%	82.44%
-	V	78.58%	78.04%	78.01%	82.97%	83.3%	82.11%
V	-	84.58%	83.97%	82.08%	79.3%	77.6%	77.66%

Table 3: Testing accuracy under different settings.

Consequently, I perform some experiments by training the model with and without batch normalization or dropout layer. The experiment results are shown in table 3. Each model is trained for 10 times and I report the average testing accuracy. I indeed observe that training with both batch normalization and dropout produces the best outcome. Specifically, removing batch normalization causes a severe damage to EEGNet, and dropout has a considerable impact on both models, especially on DeepConvNet.