國立陽明交通大學

NATIONAL YANG MING CHIAO TUNG UNIVERSITY

Deep Learning

Lab3: EEG classification

Name: 許子駿

Student ID: 311551166

Institute of Computer Science and Engineering

2023 年 4 月 3 日

Table of Contents

Chapter 1

Introduction

This task is to implement EEGNet and DeepConvNet models with different activation functions, i.e, `ReLU`, `LeakyReLU`, and `ELU`. The goal of this task is to solve EEG classification problem using BCI competition III - IIIb dataset.
BCI Competition III - IIIb dataset includes cued motor imagery with online feedback, which is non-stationary classifier, with 2 classes, left hand and right hand, from 3 subjects, 2 classes and 2 bipolar EEG channels.[1]

[1]Reference: `http://www.bbci.de/competition/iii/desc_IIIb.pdf`

Chapter 2

Experiment setup

Experiment setup is listed in this chapter including 3 parts, dataset, models, and activation functions.

2.1 Dataset

BCI Competition III - IIIb dataset includes cued motor imagery with online feedback, which is non-stationary classifier, with 2 classes, left hand and right hand, from 3 subjects, 2 classes and 2 bipolar EEG channels (See Figure 2.1). The dataset is split into training and testing set, and the input data shape is `[B, 1, 2, 750]`, and output is `[B, 2]`, where `B` represents batch size and the third dimension indicates 2 channels (See Figure 2.2).

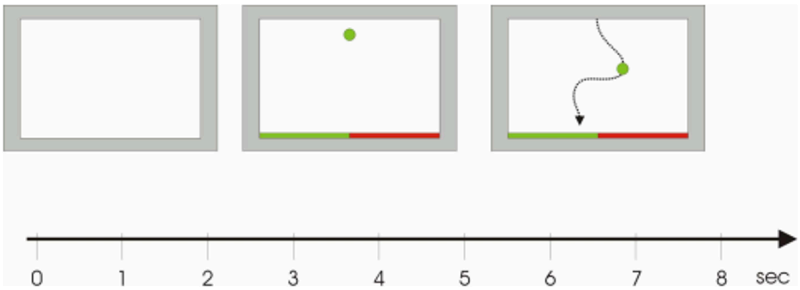

Figure 2.1: BCI Competition III - IIIb dataset.

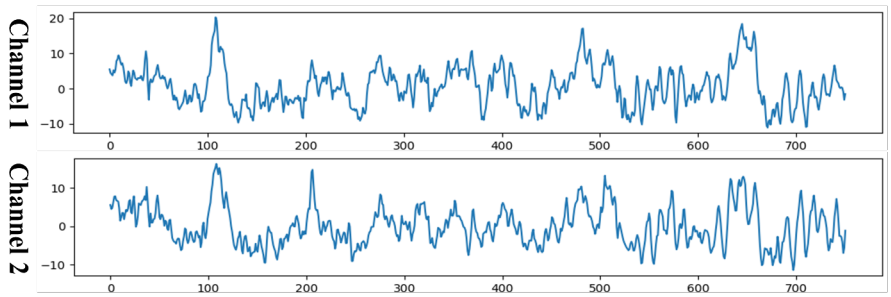

Figure 2.2: 2 channels of BCI Competition III - IIIb dataset.

2.2 Models

Models details are listed in this section including 2 parts, EEGNet and DeepConvNet.

2.2.1 EEGNet

EEGNet is compounded of 3 different 2D convolution blocks, normal, depthwise, and separable 2D convolution blocks, and the last layer of this model is a linear layer (See Figure 2.3). Details are in Listing 2.1.

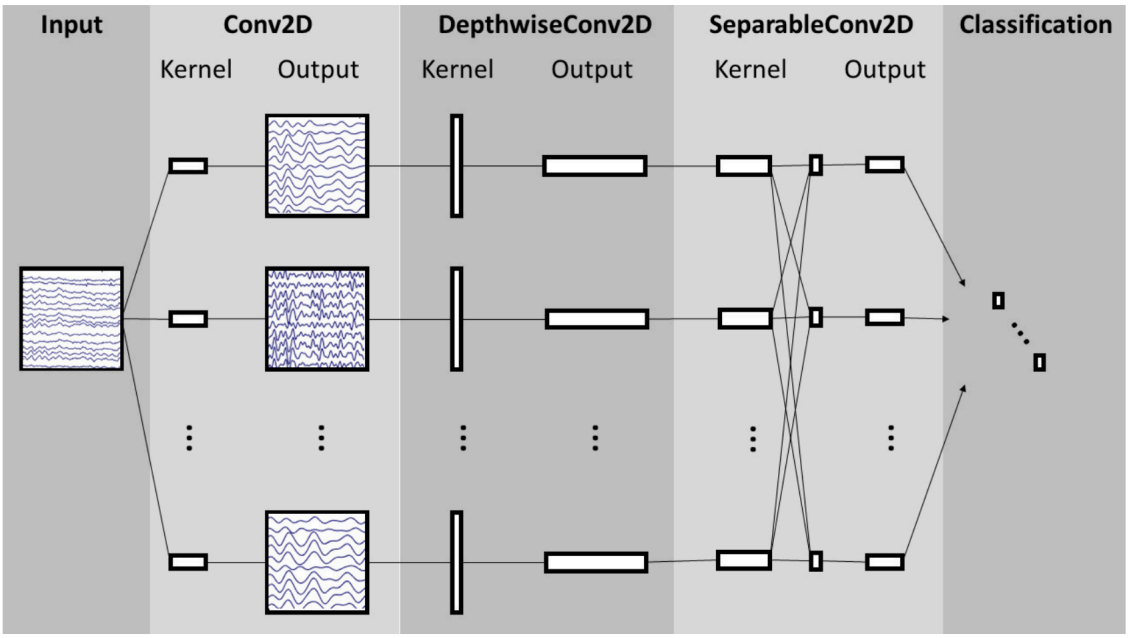

Figure 2.3: Architecture of EEGNet.

```python
self.conv_2d = nn.Sequential(
    nn.Conv2d(
        in_channels=1,
        out_channels=16,
        kernel_size=(1, 51),
        stride=(1, 1),
        padding=(0, 25),
        bias=False
    ),
    nn.BatchNorm2d(16)
)
self.depthwise_conv = nn.Sequential(
    nn.Conv2d(
        in_channels=16,
        out_channels=32,
        kernel_size=(2, 1),
        stride=(1, 1),
        groups=16,
        bias=False
    ),
    nn.BatchNorm2d(32),
    self.act,
    nn.AvgPool2d(kernel_size=(1, 4), stride=(1, 4), padding=0),
    nn.Dropout(p=dropout)
)
self.separable_conv = nn.Sequential(
    nn.Conv2d(
        in_channels=32,
        out_channels=32,
        kernel_size=(1, 15),
        stride=(1, 1),
        padding=(0, 7),
        bias=False
    ),
    nn.BatchNorm2d(32),
    self.act,
    nn.AvgPool2d(kernel_size=(1, 8), stride=(1, 8), padding=0),
    nn.Dropout(p=dropout)
)
self.linear = nn.Sequential(
    nn.Flatten(),
    nn.Linear(in_features=736, out_features=2, bias=True)
)
```

Listing 2.1: Python code of EEGNet.

2.2.2 DeepConvNet

DeepConvNet is compounded of multiple convolution blocks, and the number of channels starts from 1 to 25, then 50, 100, and finally 200. The last layer of this model is also a linear layer (See Figure 2.4). Details are in Listing 2.2.

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 2.4: Architecture of DeepConvNet.

```
self.first_conv = nn.Sequential(
    nn.Conv2d(
        in_channels=1,
        out_channels=25,
        kernel_size=(1, 5),
        bias=False
    ),
    nn.Conv2d(
        in_channels=25,
        out_channels=25,
        kernel_size=(2, 1),
        bias=False
    ),
    nn.BatchNorm2d(25),
    self.act,
    nn.MaxPool2d(kernel_size=(1, 2)),
```

```
    nn.Dropout(p=dropout)
)
layers = [25, 50, 100, 200]
self.convs = nn.ModuleList([
    nn.Sequential(
        nn.Conv2d(
            in_channels=layers[i],
            out_channels=layers[i + 1],
            kernel_size=(1, 5),
            bias=False
        ),
        nn.BatchNorm2d(layers[i + 1]),
        self.act,
        nn.MaxPool2d(kernel_size=(1, 2)),
        nn.Dropout(p=dropout))
    for i in range(len(layers) - 1)
])
convs_output_size = 373
flatten_size = 200 * reduce(lambda x, _: round((x - 4) / 2), layers[: -1],
    convs_output_size) # 8600.
self.linear = nn.Sequential(
    nn.Flatten(),
    nn.Linear(in_features=flatten_size, out_features=2, bias=True)
)
```

Listing 2.2: Python code of DeepConvNet.

2.3 Activation functions

In this task, 3 activation functions, i.e., ReLU, LeakyReLU, and ELU, are used to compare performance of different activation functions.

2.3.1 ReLU

ReLU clips all negative values and returns 0, and positive values remain the same (See Figure 2.5). It's one of the mostly used activation functions, and the gradient computation cost is low during backpropagation step. Disadvantage is positive values may be infinite and neurons which have negative values may not update.

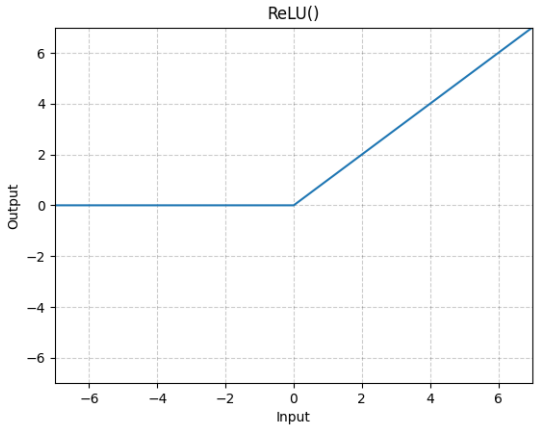

Figure 2.5: Illustration of ReLU activation function.

2.3.2 LeakyReLU

`LeakyReLU` is similar to `ReLU`, but the negative values backpropagate partially (See Figure 2.6).

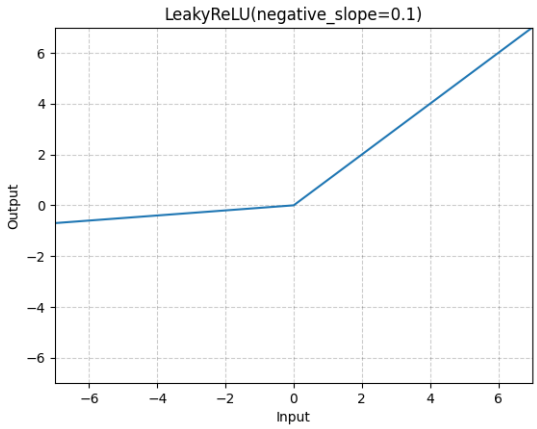

Figure 2.6: Illustration of LeakyReLU activation function.

2.3.3 ELU

`ELU` is similar to `LeakyReLU`, but the negative values returns are smoothed due to the exponential computation (See Figure 2.7).

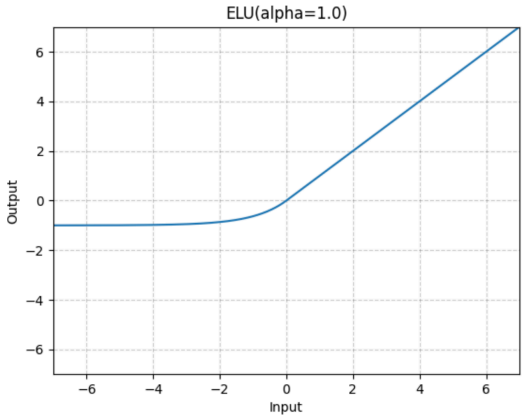

Figure 2.7: Illustration of ELU activation function.

Chapter 3

Experimental results

Results of different batch sizes, learning rates, and optimizers are listed in this chapter. Experiments of this chapter are done with common setup below:

- Seed: 42.
- Number of epochs: 500.

3.1 Best testing accuracy

3.1.1 EEGNet

Best testing accuracy of EEGNet is found with this setup:

- Optimizer: Adam.
- Learning rate: 5×10^{-3}.
- Batch size: 128.
- Dropout: 50%.

From Table 3.1 and Figure 3.1, we can see that training and testing accuracy with `ReLU` and `LeakyReLU` are similar, but accuracy with `ELU` is not good enough.

Accurcy \ Activation	ReLU	LeakyReLU	ELU
Testing	97.04%	**97.41%**	95.93%
Training	**88.52%**	87.87%	83.52%

Table 3.1: Details of EEGNet best-accuracy setup
with different activation functions (**Blue** is the best of each row).

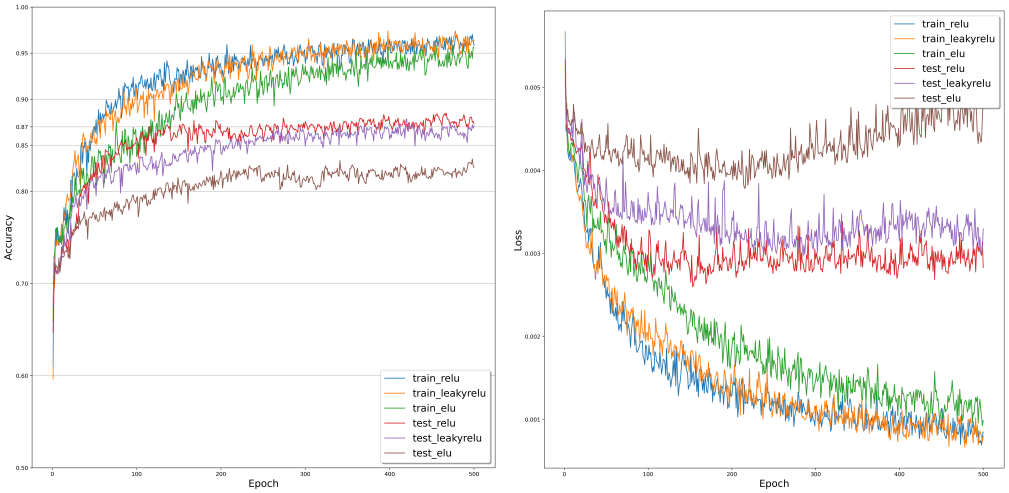

Figure 3.1: Best accuracy of EEGNet and its loss.

3.1.2 DeepConvNet

Best testing accuracy of DeepConvNet is found with this setup:

- Optimizer: Adam.
- Learning rate: 5×10^{-4}.
- Batch size: 64.
- Dropout: 50%.

From Table 3.2 and Figure 3.2, we can see that training and testing accuracy with each of three activation functions are similar.

Accurcy \ Activation	ReLU	LeakyReLU	ELU
Training	95.28%	95.65%	**97.87%**
Testing	**82.78%**	82.50%	81.02%

Table 3.2: Details of DeepConvNet best-accuracy setup with different activation functions (**Blue** is the best of each row).

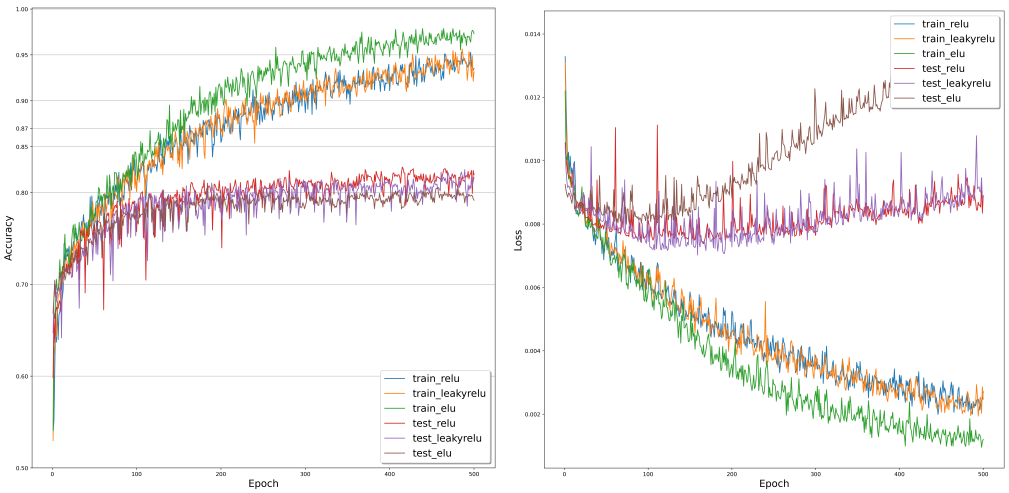

Figure 3.2: Best accuracy of DeepConvNet and its loss.

3.2 Ablation study

3.2.1 EEGNet

From Table 3.3, we can see that, in generally,

- `ReLU` and `LeakyReLU` do better, and `ELU` is always worst.
- Batch size 128 and learning rate 5×10^{-3} are best, larger or smaller is worse.
- Adam is better than SGD with other setup fixed.
- Lower dropout (25%) got higher training accuracy, but a little bit lower for testing. Larger dropout (75%) is worse on both sets.

Details are in Chapter A.0.1.

Accurcy \ Activation	ReLU	LeakyReLU	ELU
Best			
Training	97.04%	**97.41%**	95.93%
Testing	**88.52%**†	87.87%	83.52%
Smaller batch size (64)			
Training	97.41%	**97.50%**	95.65%
Testing	**87.78%**	86.57%	82.50%
Larger batch size (256)			
Training	**97.04%**	96.39%	95.83%
Testing	**87.59%**	86.85%	83.98%†
Smaller learning rate (1×10^{-3})			
Training	94.91%	**96.20%**	93.98%
Testing	86.20%	**88.33%**†	83.89%
Larger learning rate (1×10^{-2})			
Training	**97.31%**	97.04%	95.00%
Testing	**86.39%**	85.28%	83.15%
Different optimizer (SGD)			
Training	**93.24%**	93.15%	91.57%
Testing	85.65%	**86.30%**	81.20%
Smaller dropout (25%)			
Training	**99.91%**†	**99.91%**†	99.72%†
Testing	**87.04%**	86.11%	82.69%
Larger dropout (75%)			
Training	90.28%	**90.37%**	84.72%
Testing	**87.04%**	86.39%	82.13%

Table 3.3: Ablation study of EEGNet.
(**Blue** is the best of each row, † is the best of each activation function).

3.2.2 DeepConvNet

From Table 3.4, we can see that, in generally,

- Accuracy of each activation function is similar.
- Batch size 64 and learning rate 5×10^{-4} are best, larger or smaller is worse.
- Adam is better than SGD with other setup fixed.
- Lower dropout (25%) got higher training accuracy, but a little bit lower for testing. Larger dropout (75%) is worse on both sets.

Details are in Chapter A.0.2.

Activation / Accurcy	ReLU	LeakyReLU	ELU
Best			
Training	95.28%	95.65%	**97.87%**
Testing	**82.78%**†	82.50%†	81.02%
Smaller batch size (32)			
Training	96.02%	97.57%	**98.43%**
Testing	80.65%	80.56%	**81.02%**
Larger batch size (128)			
Training	93.43%	94.17%	**96.94%**
Testing	**81.76%**	81.67%	80.19%
Smaller learning rate (1×10^{-4})			
Training	86.39%	87.13%	**88.70%**
Testing	80.19%	**81.11%**	79.17%
Larger learning rate (1×10^{-3})			
Training	97.22%	97.78%	**98.89%**
Testing	81.57%	**82.04%**	81.39%†
Different optimizer (SGD)			
Training	81.76%	82.69%	**84.26%**
Testing	78.24%	78.06%	**78.61%**
Smaller dropout (25%)			
Training	**100.00%**†	**100.00%**†	**100.00%**†
Testing	**82.13%**	81.48%	80.46%
Larger dropout (75%)			
Training	80.09%	**82.13%**	81.20%
Testing	77.87%	78.33%	**78.98%**

Table 3.4: Ablation study of DeepConvNet.
(**Blue** is the best of each row, † is the best of each activation function).

Chapter 4

Conclusion

After trying so many experiments, I found some patterns for this task:

- `Adam` optimizer is **better** than `SGD` optimizer.
- `ReLU` and `LeakyReLU` are **better** than `ELU` for EEGNet, but similar for DeepConvNet.
- EEGNet has better accuracy than DeepConvNet, though DeepConvNet has larger model size (150577) than EEGNet (17874), and this may be due to the inappropriate model structure, like number of layers, kernel size, and etc.

Appendices

Appendix A

More results

A.0.1 EEGNet

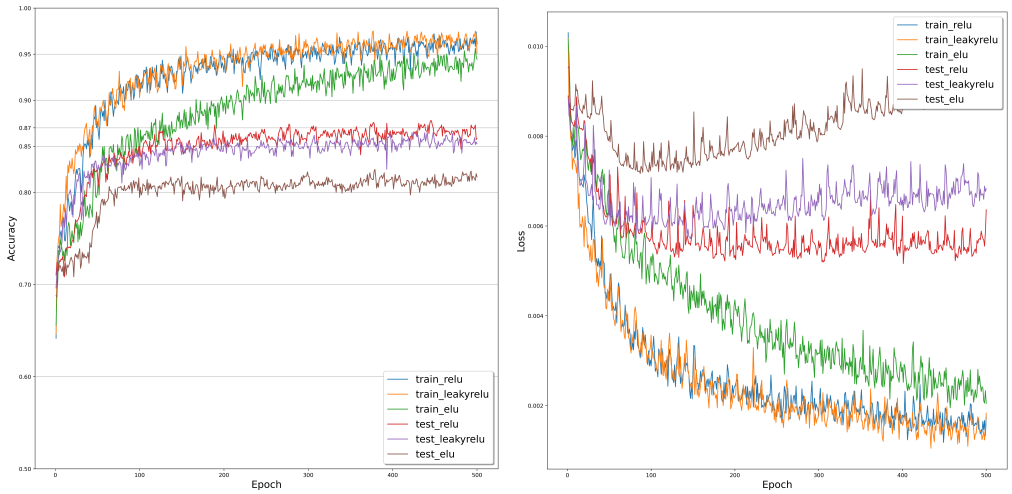

Figure A.1: EEGNet with Adam, 64 batch size, 5×10^{-3} learning rate, and 50% dropout.

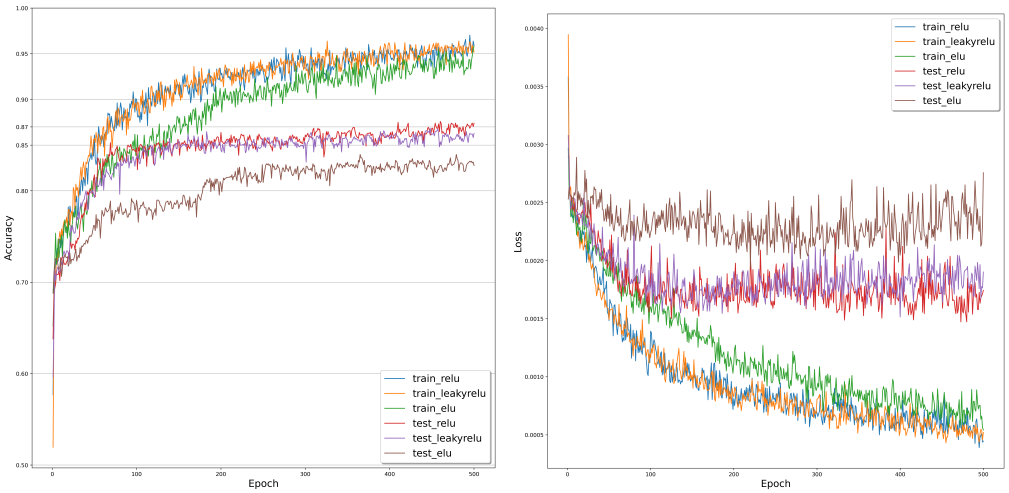

Figure A.2: EEGNet with Adam, 256 batch size, 5×10^{-3} learning rate, and 50% dropout.

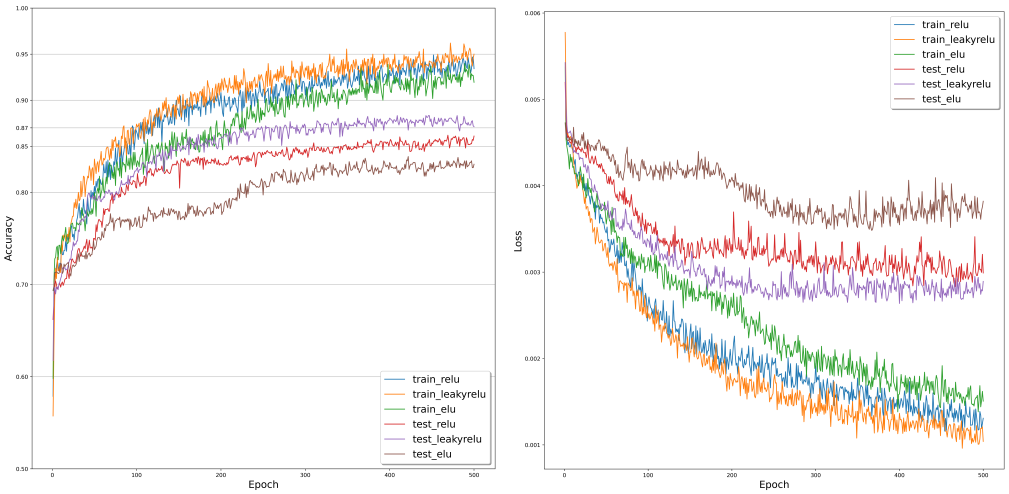

Figure A.3: EEGNet with Adam, 128 batch size, 1×10^{-3} learning rate, and 50% dropout.

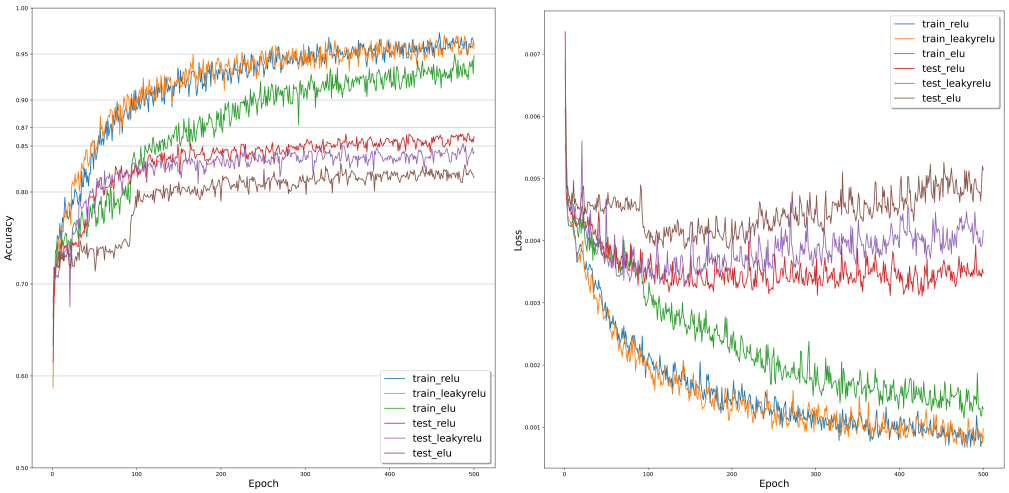

Figure A.4: EEGNet with Adam, 128 batch size, 1×10^{-2} learning rate, and 50% dropout.

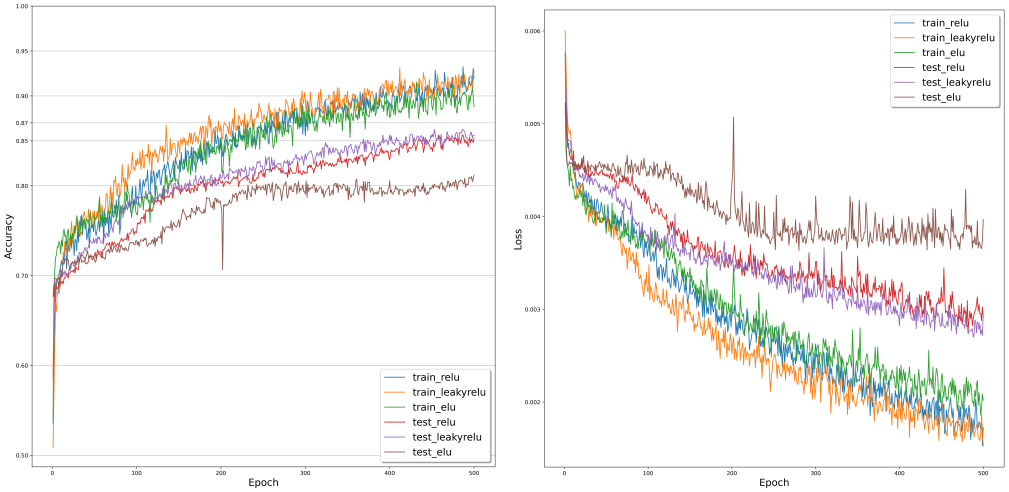

Figure A.5: EEGNet with SGD, 128 batch size, 5×10^{-3} learning rate, and 50% dropout.

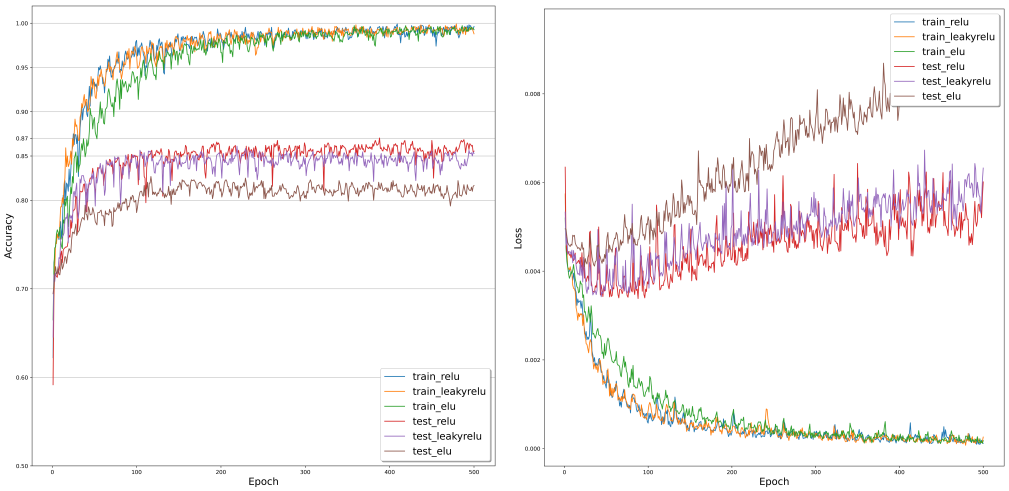

Figure A.6: EEGNet with Adam, 128 batch size, 5×10^{-3} learning rate, and 25% dropout.

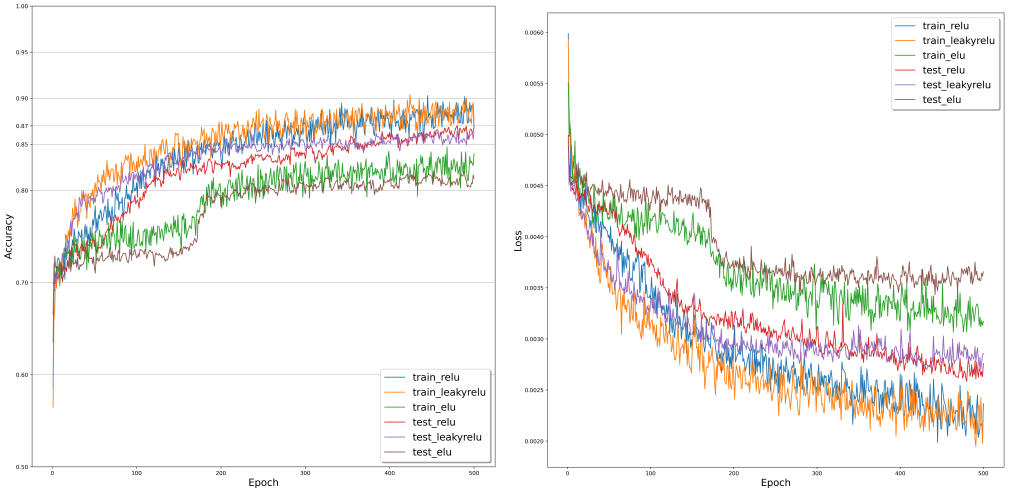

Figure A.7: EEGNet with Adam, 128 batch size, 5×10^{-3} learning rate, and 75% dropout.

A.0.2 DeepConvNet

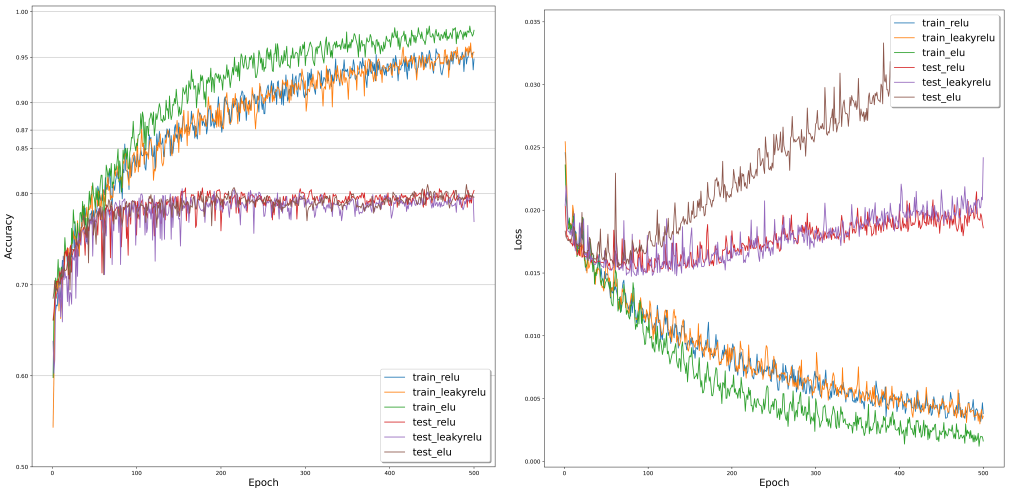

Figure A.8: DeepConvNet with Adam, 32 batch size, 5×10^{-4} learning rate, and 50% dropout.

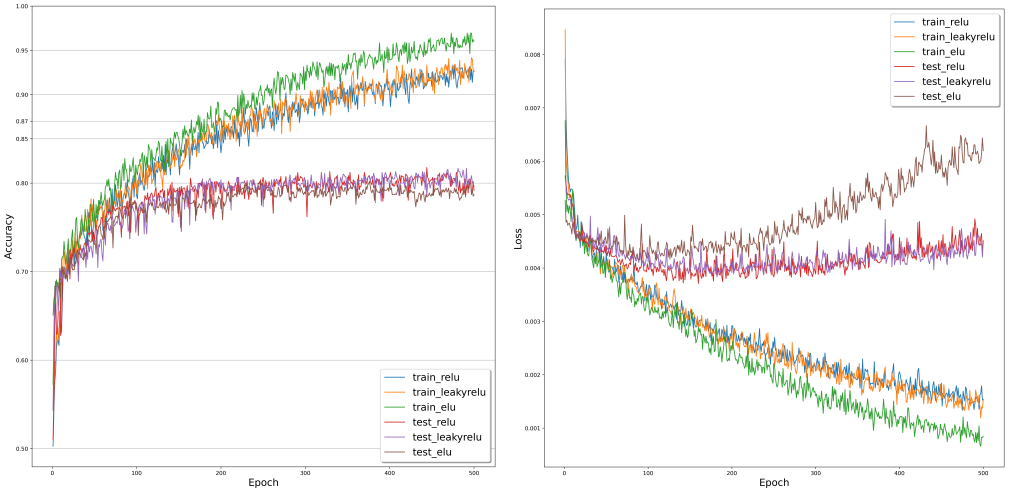

Figure A.9: DeepConvNet with Adam, 128 batch size, 5×10^{-4} learning rate, and 50% dropout.

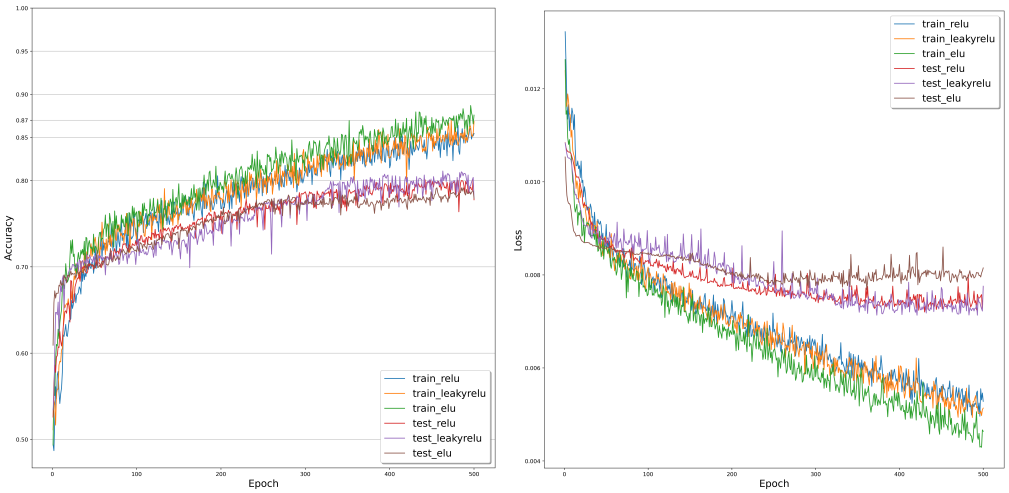

Figure A.10: DeepConvNet with Adam, 64 batch size, 1×10^{-4} learning rate, and 50% dropout.

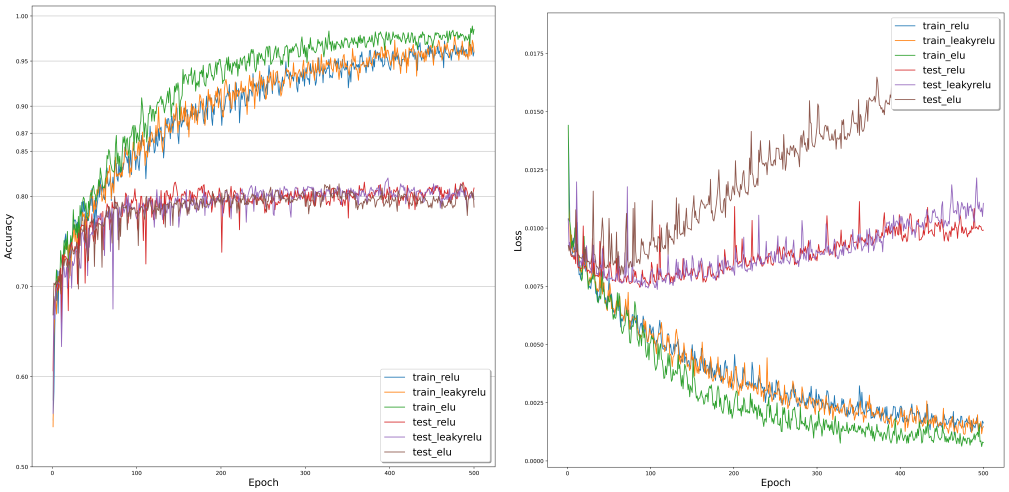

Figure A.11: DeepConvNet with Adam, 64 batch size, 1×10^{-3} learning rate, and 50% dropout.

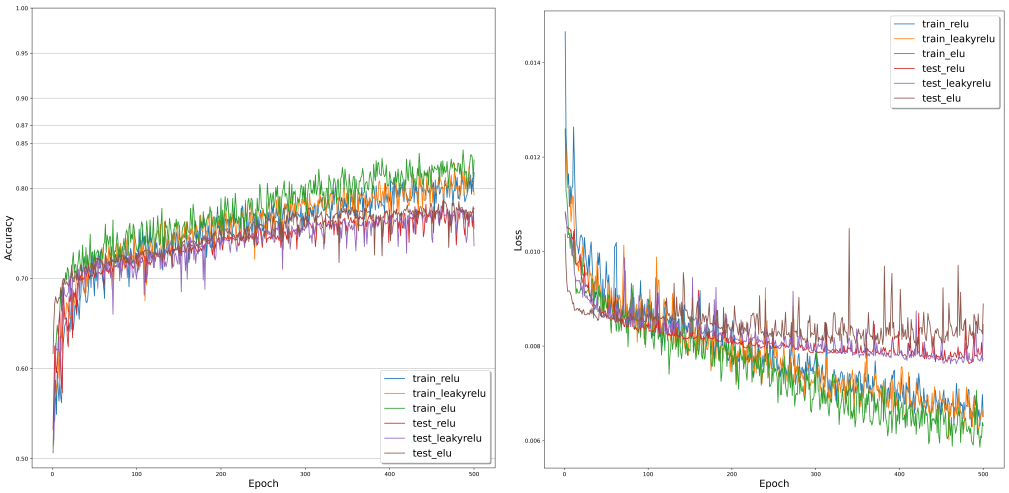

Figure A.12: DeepConvNet with SGD, 64 batch size, 5×10^{-4} learning rate, and 50% dropout.

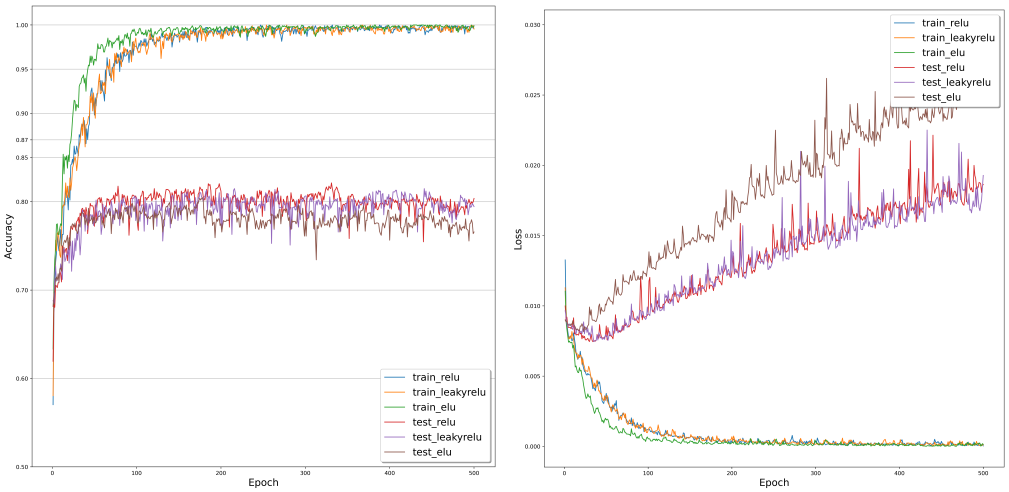

Figure A.13: DeepConvNet with Adam, 64 batch size, 5×10^{-4} learning rate, and 25% dropout.

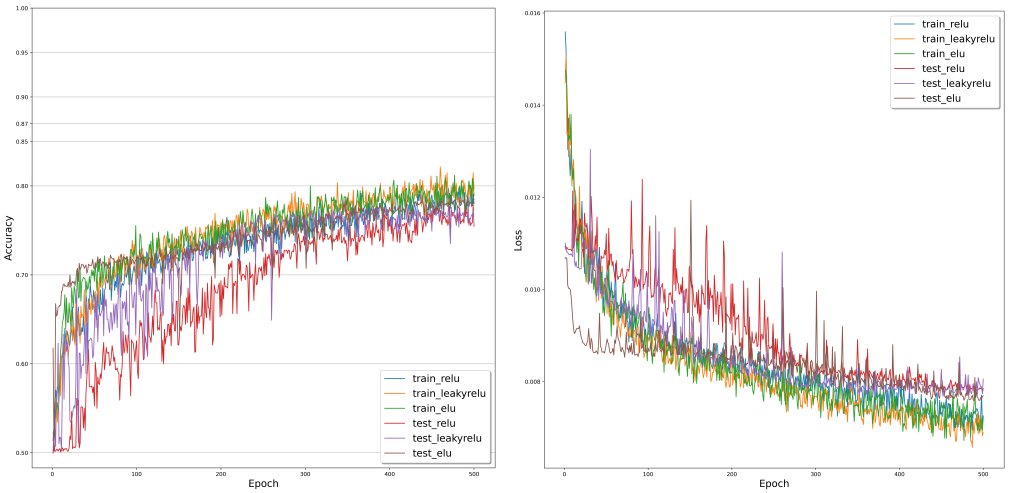

Figure A.14: DeepConvNet with Adam, 64 batch size, 5×10^{-4} learning rate, and 75% dropout.