Deep Learning Models for the Classification of EEG Data

Jim Solomon
UID: 005371849
ECE C247 Student
jimsolomon@g.ucla.edu

Zilin Zeng
UID: 006073844
ECE C247 Student
zzeng12@g.ucla.edu

Zihan Wang
UID: 306077614
ECE C247 Student
zhwang2000@g.ucla.edu

Yujie Wang
UID: 406077623
ECE C247 Student
yujie2000@g.ucla.edu

Abstract

This report focuses on utilizing Electroencephalogram signals (EEGs) collected from 9 patients to classify four distinct motor imagery tasks. We explore various deep learning architectures, documenting their impact on this classification task. We create four experiments comparing different models, data augmentation techniques and training strategies. We evaluate these models on individual subjects and across all subjects. Through all our tests, we find that the EEG Net architecture with cosine annealing and label smoothing reaches our highest test accuracy of 75 percent when training and testing across all subjects.

1. Introduction

EEGs are signals obtained by recording brain functions through the scalp [1]. Decoding these signals can enable researchers to predict patients' actions from their brains alone.

Currently, Deep learning is widely used to decode EEG data. Convolutional Neural Network (CNN) [2] based architectures like EEGNet have shown good performance in classifying EEG signals for various tasks [3]. Recurrent Neural Network (RNN) based architectures have also been used for the classification of EEG data.

In this report, we compare the effectiveness of CNN and RNN based architectures in processing EEG data for classification. Our aims are to answer the following questions:

- In optimizing subject 1 classification accuracy, Does it help to train across all subjects?
- How well do our models perform when classifying across all subjects? Were there any interesting trends?
- How does our model perform over time? Does it increase as you have data over longer periods of time? How much time is required to get a reasonable classification accuracy?

To answer these questions, we formulated four experiments comparing various deep learning architectures, data augmentation techniques and training techniques. From these experiments, we find that the EEG Net architecture with cosine annealing and label smoothing performs best when classifying across all subjects. We also gained additional insights to answer the questions above.

2. Experiment Setup

In this section, we describe our experiments and model architectures.

2.1. Experiment 1

In this experiment, we train and test three different architectures on our EEG data. We test these architectures across all subjects. For this experiment, we set out to answer the question: what baseline architecture leads to the best classification accuracy across all subjects. To do this, we implement the CNN architecture described in figure 4, the LSTM architecture described in figure 5 [4] and then combine these two to create the CNN + LSTM architecture described in figure 6. To compare these architectures, we fine tune them, finding the best hyper parameters for each model. Then, we trained and tested them on our EEG data across all subjects.

2.2. Experiment 2

From Experiment 1, we find that CNNs produce the best test accuracies. Given this we compare our CNN architecture from Experiment 1 with the EEGNet architecture de-

scribed in Lawhern's paper [3]. We compare and test for the following:

- Accuracy when trained and tested across all subjects.
- Accuracy when trained across all subjects and tested on subject 1.
- Accuracy when trained and tested on subject 1 alone.

Doing this allows us to answer the first and second question posed in the introduction.

2.3. Experiment 3

In this experiment, we try finding the best data processing and training technique that improves the accuracy of EEGNet across all subjects and on subject 1. The data processing and training techniques we compared are: label smoothing, z score normalization, spectrograms, speech processing techniques and cosine annealing. We find that training our EEGNet with cosine annealing and label smoothing produces the best results.

2.4. Experiment 4

This experiment was created to tackle the third question. For this experiment, we vary our time sequence data from 100 to 1000 in intervals of 100. We evaluated the performance of the EEGNet model across these time steps and found that once we have sequence times greater than 300, our model begins to perform well. Any sequence time below 300 leads to poor test accuracy.

3. Results

In this section, we discuss the results of our experiments

3.1. Experiment 1

For experiment 1, we found that CNNs performed best across all subjects. From table 3, we see that LSTM has the worst performance of our models. We also see that the CNN has slightly better accuracy than the CNN + LSTM. This means that CNN architectures perform better on our EEG dataset than LSTM and CNN+LSTM.

3.2. Experiment 2

For experiment 2, we set out to compare the accuracy of CNN and EEGNet when: (1) trained and tested across all subjects, (2) trained across all subjects and tested on subject 1 and (3) trained and tested on subject 1 alone. From table 1, we find that EEGNet performs best on (1) and (2).

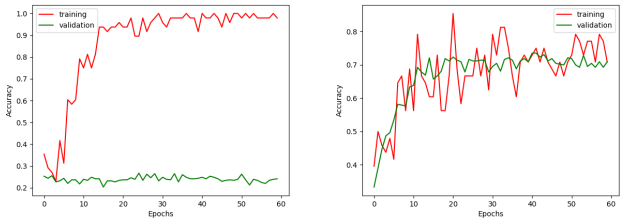
3.3. Experiment 3

For experiment 3, we see that cosine annealing has the best test accuracy when compared to other data augmentation and training strategies. In table 2, EGGNet + Cosine Annealing had an accuracy of approximately 75 %. This is better than the others. EEGNet + label smoothing also had good performance.

3.4. Experiment 4

For experiment 4, as we vary the sequence time length from 100 to 1000, our best model: EEGNet, performs well starting at a time sequence of 300. The time sequence that led to the best model performance was 900.

4. Discussion



(a) LSTM (b) CNN

Figure 1. Training and validation accuracy for two of our models

4.1. Experiment 1

From table 3, we find that CNNs perform best and LSTMs perform worst. This is because during the training process, LSTMs have a very high tendency to overfit our data. From figure 1, we see that the training accuracy for LSTM increases very fast, while the validation accuracy stays at 0.25. This shows that LSTMs tends to overfit the data easily. The CNN architectures doesn't have this risk of overfitting. The training and validation accuracy increase consistently. This is why it performs best.

4.2. Experiment 2

For experiment 2, we trained our EEGNet and CNN model on subject 1 and on all subjects. Then we tested on subject 1 and on all subjects. We found that when testing on subject 1, it was better to train on all subject and than train on subject 1 alone. From table 1, we see that EEG Net and CNN had better accuracy on subject 1 when trained across all subjects than when train on subject 1. This is because by including more subjects, we have a larger and more diverse dataset. This means our model is able to learn better statistics that can be generalized well.

In experiment 2, we also compare the performance of EEGNet and CNN when trained and tested across all subjects. In this test, we found that EEGNet performs better than our standard CNN. This result can be seen in table 3. EEGNet performs better than CNN because it utilizes temporal convolutions learn frequency filters, and depthwise

convolutions to learn frequency-specific spatial. These allow our model to generalize well to EEG specific data [3].

An interesting trend from experiment 1 and 2 was that models that included LSTM performed worse than models that didn't. This is counter intuitive since EEG signal are sequential signals and LSTM should be adept at dealing with sequential data. The reason LSTM failed could be because our dataset is small and noisy. This made it difficult for the LSTM to learn any sequential information. CNNs performed better because through the process of convolution, we filter our data and do feature extractions. These make it easier for CNNs to generalize better.

4.3. Experiment 3

For experiment 3, we tested various data augmentation and training techniques. We tested label smoothing, z score normalization, spectrograms, speech processing techniques and cosine annealing. The results can be found in table 2.

We utilized a Short Time Fourier Transform (STFT) to extract the frequency components of each channel. The STFT returns a spectrogram signal, which introduces an extra dimension of features. We believed doing this would yield better results on the EEG data. From table 2, we see that the STFT achieves 51% accuracy. This was very bad. This could be because our EEG signals are very noisy, leading to poor performance.

We also tried out some feature extraction techniques, derived from audio processing. Since EGG signals and audio signals share sequential properties, we believed performing audio processing techniques could yield better results. Viewing the Linear Predictive Coding analysis serves as a powerful tool for estimating the filter function (i.e. vocal tract from speech signals). The cepstral coefficient is another handy tool that compactly represents audio data with coefficients relevant with signal energy and frequency amplitudes. Performing these data augmentations led to poor performance on our EEG data. It seems our augmented features lacked semantic meaning. This led to us achieving less than 30% accuracy. The results are in table 2.

We implemented cosine annealing, an ensemble method. It imitates the process of training multiple models and averaging the results across those models. From our experiment, annealing tends to find better local minimums after loss curve plateaus. Annealing gives us a 3% boost in accuracy compared to the baseline model. This makes it our most effective regularization technique.

Label smoothing is a regularization technique that improves model generalizability by preventing the model from being overconfident about prediction. The model is encouraged to learn more robust features thus generalizing better during inference. With this, we achieved a little extra performance.

We performed z-normalization across the twenty-two channels of the EEG signal so that the signals across channels are of similar distributions (zero-mean, unit-variance). However, the z-normalization failed to improve our model. This might be because our dataset is small. This means the statistics used to normalize the data might have been heavily biased by noise, leading to poor performance.

4.4. Experiment 4

From figure 2, we found that our EEGNet starts to perform well on our data once we have a sequence time of 300. This means we can expect great results when we have a sequence time of greater than 300. From the plot, we can see that test accuracy improves over time and then begins to plateau after 300. This means that the minimum time required to get great classification accuracy is 300, and we don't need longer sequence times to classify well. If we want best performance with our EEG model, the best sequence time is 900. This resulted in the highest accuracy on EEGNet.

The reason we get these results could be because EEGNet is able to successfully learn information from the noiser part of the EEG data. This means by having longer sequence, the model is able to extra information that generalize better.

5. Conclusion

Through all the experiments, we arrived at the best model for classification tasks on the EEG dataset. **The best model was an EEGNet trained with consine annealing and augmented with label smoothing. This model led to an accuracy of 75%**. The training and validation accuracy plot of our best model can be found in figure 3.

References

[1] Maham Saeidi, Waldemar Karwowski, Farzad V Farahani, Krzysztof Fiok, Redha Taiar, Peter A Hancock, and Awad Al-Juaid. Neural decoding of eeg signals with machine learning: A systematic review. *Brain Sciences*, 11(11):1525, 2021. 1

[2] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 1

[3] Vernon J Lawhern, Amelia J Solon, Nicholas R Waytowich, Stephen M Gordon, Chou P Hung, and Brent J Lance. Eegnet: a compact convolutional neural network for eeg-based brain–computer interfaces. *Journal of neural engineering*, 15(5):056013, 2018. 1, 2, 3

[4] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 1

[5] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders, 2021. 4

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. 4

Model	Training Dataset	Test Accuracy Subject 1	Test Accuracy All Subjects
EGG	All	0.5	0.72
	Sub 1	0.42	N/A
CNN	All	0.46	0.68
	Sub 1	0.44	N/A

Table 1. Experiment 2: This table displays the models, respective hyperparameters, and the testing accuracy for subject 1 and all subjects for both EEG and CNN models. The hyperparameters for EEG is batch size of 64 and a number of epochs set to 50, whereas for CNN, the batch size is 64 and the number of epochs is 60.

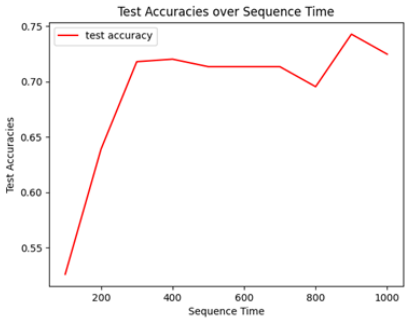


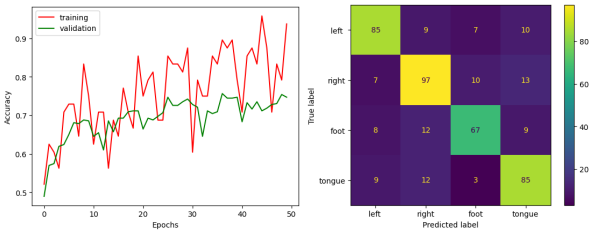
Figure 2. The accuracy versus different time truncation

Model	Strategies	Param Dataset	Test Accuracy	Test Time (sec)
CNN (Baseline accuracy: 0.68)	STFT	Batch size=64 Num epochs=50 All Subjects	0.51	0.44
EEG (Baseline Accuracy: 0.72)	LPC + Cepstral coefficients	Batch size=64 Num epochs=50 All Subjects	0.26	1.52
EEG	**Cosine Annealing**	Warm restart after every 10 epochs Batch size=64 Num epochs=50 All Subjects	**0.75**	0.21
EEG	Label Smoothing	Batch size=64 Num epochs=50 Subjects 1	0.72	0.20
EEG	Z-score Normalization	Batch size=64 Num epochs=50 Subject 1	0.68	0.24

Table 2. Experiment 3: This table compares different data augment strategies, and corresponding test accuracy and test times.

Model	Param	Overall Test Accuracy
CNN	Learning rate=5e-4 Batch size=64 Num epochs=60	0.69
LSTM	Learning rate=5e-4 Hidden size=128 Num layers= 3 Batch size=64 Num epochs=60	0.30
CNN+LSTM	Learning rate=5e-4 Batch size=64 Num epochs=60	0.65
EGG Net	Learning rate=1e-3 Batch size=64 Num epochs=50	0.72
CNN+ STFT	Learning rate=5e-4 Batch size=64 Num epochs=50	0.51
CNN+ LPC+ Cepstrum	Learning rate=5e-4 Batch size=64 Num epochs=50	0.27
EEG Net+ Cosine Annealing	Learning rate=1e-2 Batch size=64 Num epochs=50 Warm restart after every 10 epochs	**0.75**
EEG Net+ Label Smoothing	Learning rate=1e-3 Batch size=64 Num epochs=50 Label_smoothing=0.1	**0.72**
EEG Net+ Z score normalization	Learning rate=1e-3 Batch size=64 Num epochs=50 Label_smoothing=0.1	0.68
ResNet [5]	Learning rate=5e-4 Batch size=64	0.56
MLP+Autoencoder [6]	Learning rate=5e-4 Batch size=64	0.36

Table 3. This table summaries various models, as well as the employed data augmentation techniques, their corresponding hyperparameters and overall test accuracy across all subjects.



(a) (b)

Figure 3. (a). The training and validation accuracy for our best model. (b). The corresponding confusion matrix

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
CNN                                      [64, 4]                   --
├─ModuleList: 1-1                        --                        --
│    └─Conv2d: 2-1                       [64, 25, 400, 1]          13,775
│    └─ELU: 2-2                          [64, 25, 400, 1]          --
│    └─MaxPool2d: 2-3                    [64, 25, 134, 1]          --
│    └─BatchNorm2d: 2-4                  [64, 25, 134, 1]          50
│    └─Dropout: 2-5                      [64, 25, 134, 1]          --
│    └─Conv2d: 2-6                       [64, 50, 134, 1]          31,300
│    └─ELU: 2-7                          [64, 50, 134, 1]          --
│    └─MaxPool2d: 2-8                    [64, 50, 45, 1]           --
│    └─BatchNorm2d: 2-9                  [64, 50, 45, 1]           100
│    └─Dropout: 2-10                     [64, 50, 45, 1]           --
│    └─Conv2d: 2-11                      [64, 100, 45, 1]          125,100
│    └─ELU: 2-12                         [64, 100, 45, 1]          --
│    └─MaxPool2d: 2-13                   [64, 100, 15, 1]          --
│    └─BatchNorm2d: 2-14                 [64, 100, 15, 1]          200
│    └─Dropout: 2-15                     [64, 100, 15, 1]          --
│    └─Conv2d: 2-16                      [64, 200, 15, 1]          500,200
│    └─ELU: 2-17                         [64, 200, 15, 1]          --
│    └─MaxPool2d: 2-18                   [64, 200, 5, 1]           --
│    └─BatchNorm2d: 2-19                 [64, 200, 5, 1]           400
│    └─Dropout: 2-20                     [64, 200, 5, 1]           --
│    └─Flatten: 2-21                     [64, 1000]                --
│    └─Linear: 2-22                      [64, 40]                  40,040
│    └─Linear: 2-23                      [64, 10]                  410
│    └─Linear: 2-24                      [64, 4]                   44
==========================================================================================
```

Figure 4. CNN Architecture

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
LSTM                                     [64, 4]                   --
├─LSTM: 1-1                              [64, 400, 128]            342,016
├─Linear: 1-2                            [64, 4]                   516
==========================================================================================
```

Figure 5. LSTM Architecture

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
CLSTM                                    [64, 4]                   --
├─ModuleList: 1-1                        --                        --
│    └─Conv2d: 2-1                       [64, 25, 400, 1]          13,775
│    └─ELU: 2-2                          [64, 25, 400, 1]          --
│    └─MaxPool2d: 2-3                    [64, 25, 134, 1]          --
│    └─BatchNorm2d: 2-4                  [64, 25, 134, 1]          50
│    └─Dropout: 2-5                      [64, 25, 134, 1]          --
│    └─Conv2d: 2-6                       [64, 50, 134, 1]          31,300
│    └─ELU: 2-7                          [64, 50, 134, 1]          --
│    └─MaxPool2d: 2-8                    [64, 50, 45, 1]           --
│    └─BatchNorm2d: 2-9                  [64, 50, 45, 1]           100
│    └─Dropout: 2-10                     [64, 50, 45, 1]           --
│    └─Conv2d: 2-11                      [64, 100, 45, 1]          125,100
│    └─ELU: 2-12                         [64, 100, 45, 1]          --
│    └─MaxPool2d: 2-13                   [64, 100, 15, 1]          --
│    └─BatchNorm2d: 2-14                 [64, 100, 15, 1]          200
│    └─Dropout: 2-15                     [64, 100, 15, 1]          --
│    └─Conv2d: 2-16                      [64, 200, 15, 1]          500,200
│    └─ELU: 2-17                         [64, 200, 15, 1]          --
│    └─MaxPool2d: 2-18                   [64, 200, 5, 1]           --
│    └─BatchNorm2d: 2-19                 [64, 200, 5, 1]           400
│    └─Dropout: 2-20                     [64, 200, 5, 1]           --
│    └─Flatten: 2-21                     [64, 1000]                --
│    └─Linear: 2-22                      [64, 40]                  40,040
│    └─LSTM: 2-23                        [64, 300]                 1,132,800
│    └─Linear: 2-24                      [64, 4]                   1,204
==========================================================================================
```

Figure 6. CNN + LSTM Architecture

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
EEGNet                                   [64, 4]                   --
├─ModuleList: 1-1                        --                        --
│    └─Conv2d: 2-1                       [64, 8, 22, 400]          1,000
│    └─BatchNorm2d: 2-2                  [64, 8, 22, 400]          16
│    └─ConstrainedConv2d: 2-3            [64, 16, 1, 400]          352
│    └─BatchNorm2d: 2-4                  [64, 16, 1, 400]          32
│    └─ELU: 2-5                          [64, 16, 1, 400]          --
│    └─AvgPool2d: 2-6                    [64, 16, 1, 100]          --
│    └─Dropout: 2-7                      [64, 16, 1, 100]          --
│    └─Conv2d: 2-8                       [64, 16, 1, 100]          1,008
│    └─Conv2d: 2-9                       [64, 16, 1, 100]          272
│    └─BatchNorm2d: 2-10                 [64, 16, 1, 100]          32
│    └─ELU: 2-11                         [64, 16, 1, 100]          --
│    └─AvgPool2d: 2-12                   [64, 16, 1, 12]           --
│    └─Dropout: 2-13                     [64, 16, 1, 12]           --
│    └─Flatten: 2-14                     [64, 192]                 --
│    └─Linear: 2-15                      [64, 100]                 19,300
│    └─Linear: 2-16                      [64, 10]                  1,010
│    └─Linear: 2-17                      [64, 4]                   44
==========================================================================================
```

Figure 7. EEG Architecture

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
ResNet                                   [64, 4]                   --
├─Sequential: 1-1                        [64, 64, 200, 1]          --
│    └─Conv2d: 2-1                       [64, 64, 200, 1]          69,056
│    └─BatchNorm2d: 2-2                  [64, 64, 200, 1]          128
│    └─ELU: 2-3                          [64, 64, 200, 1]          --
├─MaxPool2d: 1-2                         [64, 64, 100, 1]          --
├─Sequential: 1-3                        [64, 64, 100, 1]          --
│    └─ResidualBlock: 2-4                [64, 64, 100, 1]          --
│    │    └─Sequential: 3-1              [64, 64, 100, 1]          37,056
│    │    └─Sequential: 3-2              [64, 64, 100, 1]          37,056
│    │    └─ELU: 3-3                     [64, 64, 100, 1]          --
│    └─ResidualBlock: 2-5                [64, 64, 100, 1]          --
│    │    └─Sequential: 3-4              [64, 64, 100, 1]          37,056
│    │    └─Sequential: 3-5              [64, 64, 100, 1]          37,056
│    │    └─ELU: 3-6                     [64, 64, 100, 1]          --
├─Sequential: 1-4                        [64, 128, 50, 1]          --
│    └─ResidualBlock: 2-6                [64, 128, 50, 1]          --
│    │    └─Sequential: 3-7              [64, 128, 50, 1]          74,112
│    │    └─Sequential: 3-8              [64, 128, 50, 1]          147,840
│    │    └─Sequential: 3-9              [64, 128, 50, 1]          8,576
│    │    └─ELU: 3-10                    [64, 128, 50, 1]          --
│    └─ResidualBlock: 2-7                [64, 128, 50, 1]          --
│    │    └─Sequential: 3-11             [64, 128, 50, 1]          147,840
│    │    └─Sequential: 3-12             [64, 128, 50, 1]          147,840
│    │    └─ELU: 3-13                    [64, 128, 50, 1]          --
│    └─ResidualBlock: 2-8                [64, 128, 50, 1]          --
│    │    └─Sequential: 3-14             [64, 128, 50, 1]          147,840
│    │    └─Sequential: 3-15             [64, 128, 50, 1]          147,840
│    │    └─ELU: 3-16                    [64, 128, 50, 1]          --
│    └─ResidualBlock: 2-9                [64, 128, 50, 1]          --
│    │    └─Sequential: 3-17             [64, 128, 50, 1]          147,840
│    │    └─Sequential: 3-18             [64, 128, 50, 1]          147,840
│    │    └─ELU: 3-19                    [64, 128, 50, 1]          --
├─Sequential: 1-5                        [64, 256, 25, 1]          --
│    └─ResidualBlock: 2-10               [64, 256, 25, 1]          --
│    │    └─Sequential: 3-20             [64, 256, 25, 1]          295,680
│    │    └─Sequential: 3-21             [64, 256, 25, 1]          590,592
│    │    └─Sequential: 3-22             [64, 256, 25, 1]          33,536
│    │    └─ELU: 3-23                    [64, 256, 25, 1]          --
│    └─ResidualBlock: 2-11               [64, 256, 25, 1]          --
│    │    └─Sequential: 3-24             [64, 256, 25, 1]          590,592
│    │    └─Sequential: 3-25             [64, 256, 25, 1]          590,592
│    │    └─ELU: 3-26                    [64, 256, 25, 1]          --
│    └─ResidualBlock: 2-12               [64, 256, 25, 1]          --
│    │    └─Sequential: 3-27             [64, 256, 25, 1]          590,592
│    │    └─Sequential: 3-28             [64, 256, 25, 1]          590,592
│    │    └─ELU: 3-29                    [64, 256, 25, 1]          --
│    └─ResidualBlock: 2-13               [64, 256, 25, 1]          --
│    │    └─Sequential: 3-30             [64, 256, 25, 1]          590,592
│    │    └─Sequential: 3-31             [64, 256, 25, 1]          590,592
│    │    └─ELU: 3-32                    [64, 256, 25, 1]          --
├─Sequential: 1-6                        [64, 512, 13, 1]          --
│    └─ResidualBlock: 2-14               [64, 512, 13, 1]          --
│    │    └─Sequential: 3-33             [64, 512, 13, 1]          1,181,184
│    │    └─Sequential: 3-34             [64, 512, 13, 1]          2,360,832
│    │    └─Sequential: 3-35             [64, 512, 13, 1]          132,608
│    │    └─ELU: 3-36                    [64, 512, 13, 1]          --
│    └─ResidualBlock: 2-15               [64, 512, 13, 1]          --
│    │    └─Sequential: 3-37             [64, 512, 13, 1]          2,360,832
│    │    └─Sequential: 3-38             [64, 512, 13, 1]          2,360,832
│    │    └─ELU: 3-39                    [64, 512, 13, 1]          --
├─AdaptiveAvgPool2d: 1-7                 [64, 512, 1, 1]           --
├─Linear: 1-8                            [64, 4]                   2,052
==========================================================================================
```

Figure 8. Resnet Architecture

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
CAE                                      [6768, 22, 400, 1]        --
├─Sequential: 1-1                        [6768, 8, 100, 1]         --
│    └─Conv2d: 2-1                       [6768, 32, 400, 1]        6,368
│    └─ELU: 2-2                          [6768, 32, 400, 1]        --
│    └─MaxPool2d: 2-3                    [6768, 32, 200, 1]        --
│    └─Conv2d: 2-4                       [6768, 8, 200, 1]         2,312
│    └─ELU: 2-5                          [6768, 8, 200, 1]         --
│    └─MaxPool2d: 2-6                    [6768, 8, 100, 1]         --
├─Sequential: 1-2                        [6768, 22, 400, 4]        --
│    └─ConvTranspose2d: 2-7              [6768, 32, 200, 2]        2,336
│    └─ELU: 2-8                          [6768, 32, 200, 2]        --
│    └─ConvTranspose2d: 2-9              [6768, 22, 400, 4]        6,358
│    └─Sigmoid: 2-10                     [6768, 22, 400, 4]        --
==========================================================================================
```

Figure 9. AutoEncoder Architecture

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
MLP                                      [64, 4]                   --
├─Sequential: 1-1                        [64, 4]                   --
│    └─Flatten: 2-1                      [64, 800]                 --
│    └─Linear: 2-2                       [64, 400]                 320,400
│    └─ReLU: 2-3                         [64, 400]                 --
│    └─Linear: 2-4                       [64, 200]                 80,200
│    └─ReLU: 2-5                         [64, 200]                 --
│    └─Linear: 2-6                       [64, 100]                 20,100
│    └─ReLU: 2-7                         [64, 100]                 --
│    └─Linear: 2-8                       [64, 4]                   404
==========================================================================================
```

Figure 10. MLP Architecture