Combining CNNs with LSTM and VAEs for the EEG dataset

Feiqian Zhu
UCLA
Fz247@g.ucla.edu

Wenhui Sui
UCLA
berylbir@g.ucla.edu

Lexa Huang
UCLA
lexa.h@g.ucla.edu

Shaodian Wang
UCLA
wangsd@g.ucla.edu

Abstract

Convolutional Neural Networks (CNNs) have been shown to be effective in the general problem of electroencephalogram (EEG) decoding. We attempted to combine CNNs with other deep learning models, Variational Autoencoders (VAEs) and Long Short-Term Memory (LSTM) and investigated their effectiveness on EEG data. Concluding that VAEs and LSTM do not improve the performance of the Base CNN model, we were able to gain valuable insights from our implementation of these models.

1. Introduction

This paper evaluates the performance of Convolutional Neural Networks (CNNs) with the addition of Variational Autoencoder (VAE) and Long Short-Term Memory (LSTM) architecture in classification across all subjects in the EEG dataset. We would like to investigate the performance of a CNN+LSTM architecture in comparison to the baseline deep CNN model presented by Schirrmeister et al. [1]. For VAE, we would like to investigate the use and effectiveness of VAEs to generate EEG data from our training set. Details and hyperparameters for each model are included at the end of the work.

1.1. Base Convolutional Neural Network (Base CNN)

The base CNN that we use in this report is the same as the one proposed by Schirrmeister et al. [1]. Its architecture is shown in Figure 4, consisting of four convolution blocks, where Maxpool, Batch Normalization and Dropout are added after the convolutional layers.

1.2. Convolutional Neural Network with Long Short-Term Memory (CNN+LSTM)

We constructed two models incorporating LSTM in CNN. The first uses the same Base CNN as above, while the second includes modifications to the CNN blocks. The architectures of the two models are shown in Figures 1 and 2, consisting of four convolution blocks, followed by an LSTM block.

For the second model, the CNN blocks are modified: an ELU and then Batch Normalization is followed by each convolution layer, and a Max Pool layer is added thereafter for every block after the first convolution block. A dropout layer is present at the end of the four convolution blocks.

In the LSTM block, we included two bidirectional LSTM blocks, followed by a dropout layer and finally a dense layer with SoftMax activation which will be used for classification.

We added an additional dimension to make our input more aligned with that of an actual image so that CNN portion can perform better. We also performed input trimming after doing a bit of exploratory data analysis, which seemed to indicate that data after the 500th interval is not particularly useful in making predictions.

1.3. Convolutional Neural Network with Variational Autoencoder (CNN+VAE)

This model uses the Base CNN model, with the addition of an L2 regularizer, learning rate decay, and one more convolution layer in the first convolution block, as proposed by Schirrmeister et al. in [1]. The architecture of the VAE is shown in Figure 3 below, where we have an encoder that encodes the input to our latent space size of 64 and a decoder that produces output of the same dimensions as the input.

We also performed some preprocessing on the input data for these experiments based on the discussion. Once again, we added an additional dimension and performed input trimming. Thereafter we performed Maxpooling, averaging with Gaussian noise, and subsampling with Gaussian noise. We then concatenated across samples to augment our dataset.

We trained our VAE separately from our CNN. We assume a normal distribution for our input. Therefore, the loss metric used for VAE is

$$loss = loss^{reconstruction} - k \cdot loss^{KL},$$

where $loss^{reconstruction}$ is the mean square error. The VAE is

first trained for 200 epochs, where our training and validation losses converged to 0.0123 and 0.0127 respectively. Thereafter, we train the CNN with the output values of our VAE.

We believe adding a VAE may help remove features that the model may overtrain on, and make the CNN more generalizable, since the original dataset is noisy.

2. Results

The final test accuracy across all subjects for the Base CNN, CNN+LSTM, CNN+LSTM (modified) and CNN+VAE are shown in Table 1.

The plots for training accuracy and validation loss accuracy for the Base CNN and CNN+VAE are shown in Figures 5-8.

3. Discussion

3.1. Convolutional Neural Network with Long Short-Term Memory (CNN+LSTM)

Base CNN architecture. Adding the LSTM block to the Base CNN proposed by Schirrmeister et al. [1] resulted in low test accuracy. Hence, for CNN+LSTM (modified), we modified the CNN blocks, and were able to achieve higher test accuracy.

Test accuracy. The performance of the CNN+LSTM model is poor, as evidenced by the low test accuracy of 0.6795. This is much lower compared to the train accuracy after 100 epochs, which is 0.9754. Thus, it is possible that the model overfitted on the training data, as the addition of the LSTM block results in increased model complexity. This contrasts with the base CNN model which, despite having a worse train accuracy of 0.9362 after training for 100 epochs, has a better test accuracy of 0.7218. This further suggests overfitting, as apart from the architecture of the CNN components, the only difference is the two extra LSTM blocks.

3.2. Convolutional Neural Network with Variational Autoencoder (CNN+VAE)

Preprocessing. When training the VAE on the dataset, using the same preprocessing method as used for the base model results in exploding gradients after a few epochs. This may be due to a larger initialization of weights in the VAE architecture. Additionally, the activations are not normalized around 0, which causes them to explode over epochs. We were able to solve the problem by adding an extra step in the preprocessing: normalizing the input data to the range [-1,1].

Training accuracy and validation loss trajectory. Comparing Figure 5 and Figure 7, we can observe that the Base CNN model has a much smoother training accuracy and validation loss trajectory than that for CNN+VAE, and the latter has much larger variations between epochs. This might be due to a large learning rate, making it harder for the CNN+VAE model to converge to an optimal solution with the same number of epochs.

Test accuracy. The test accuracy of CNN+VAE is much lower than the base CNN. Although the VAE output represents a less noisy version of the original dataset, it is possible that the VAE output also contains much less information, which may have resulted in the lower test accuracy. It is also possible that our assumptions of a normal distribution for the input data is incorrect, and a more general loss function based on evidence lower bound (ELBO) should be used.

Future work. In the future, we would like to investigate the use of VAE in generating data that can augment the existing EEG dataset, and whether this would lead to improved model performance.

References

[1] Schirrmeister, R.T., Springenberg, J.T., Fiederer, L.D.J., Glasstetter, M., Eggensperger, K., Tangermann, M., Hutter, F., Burgard, W. and Ball, T. (2017), Deep learning with convolutional neural networks for EEG decoding and visualization. Hum. Brain Mapp., 38: 5391-5420. https://doi.org/10.1002/hbm.23730

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 500, 22, 25)	275
conv2d_1 (Conv2D)	(None, 500, 1, 25)	13775
max_pooling2d (MaxPooling2D)	(None, 167, 1, 25)	0
batch_normalization (BatchNormalization)	(None, 167, 1, 25)	100
dropout (Dropout)	(None, 167, 1, 25)	0
conv2d_2 (Conv2D)	(None, 167, 1, 50)	12550
max_pooling2d_1 (MaxPooling2D)	(None, 56, 1, 50)	0
batch_normalization_1 (BatchNormalization)	(None, 56, 1, 50)	200
dropout_1 (Dropout)	(None, 56, 1, 50)	0
conv2d_3 (Conv2D)	(None, 56, 1, 100)	50100
max_pooling2d_2 (MaxPooling2D)	(None, 19, 1, 100)	0
batch_normalization_2 (BatchNormalization)	(None, 19, 1, 100)	400
dropout_2 (Dropout)	(None, 19, 1, 100)	0
conv2d_4 (Conv2D)	(None, 19, 1, 200)	200200
max_pooling2d_3 (MaxPooling2D)	(None, 7, 1, 200)	0
batch_normalization_3 (BatchNormalization)	(None, 7, 1, 200)	800
dropout_3 (Dropout)	(None, 7, 1, 200)	0
time_distributed (TimeDistributed)	(None, 7, 200)	0
bidirectional (Bidirectional)	(None, 7, 256)	337920
bidirectional_1 (Bidirectional)	(None, 64)	74240
dropout_4 (Dropout)	(None, 64)	0
dense (Dense)	(None, 4)	260

Total params: 690,820
Trainable params: 690,070
Non-trainable params: 750

Figure 1: CNN+LSTM Architecture

Layer (type)	Output Shape	Param #
permute_3 (Permute)	(None, 500, 1, 22)	0
conv2d_12 (Conv2D)	(None, 491, 1, 16)	3536
elu_12 (ELU)	(None, 491, 1, 16)	0
batch_normalization_12 (BatchNormalization)	(None, 491, 1, 16)	64
conv2d_13 (Conv2D)	(None, 482, 1, 32)	5152
elu_13 (ELU)	(None, 482, 1, 32)	0
batch_normalization_13 (BatchNormalization)	(None, 482, 1, 32)	128
max_pooling2d_9 (MaxPooling2D)	(None, 241, 1, 32)	0
conv2d_14 (Conv2D)	(None, 232, 1, 64)	20544
elu_14 (ELU)	(None, 232, 1, 64)	0
batch_normalization_14 (BatchNormalization)	(None, 232, 1, 64)	256
max_pooling2d_10 (MaxPooling2D)	(None, 116, 1, 64)	0
conv2d_15 (Conv2D)	(None, 107, 1, 128)	82048
elu_15 (ELU)	(None, 107, 1, 128)	0
batch_normalization_15 (BatchNormalization)	(None, 107, 1, 128)	512
max_pooling2d_11 (MaxPooling2D)	(None, 53, 1, 128)	0
dropout_6 (Dropout)	(None, 53, 1, 128)	0
time_distributed_3 (TimeDistributed)	(None, 53, 128)	0
bidirectional_6 (Bidirectional)	(None, 53, 256)	264192
bidirectional_7 (Bidirectional)	(None, 64)	74240
dropout_7 (Dropout)	(None, 64)	0
dense_3 (Dense)	(None, 4)	260

Total params: 450,932
Trainable params: 450,452
Non-trainable params: 480

Figure 2: CNN+LSTM (modified) Architecture

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	[(None, 1, 22, 250)]	0	[]
sequential_4 (Sequential)	(None, 1, 22, 128)	48640	['input_3[0][0]', 'input_3[0][0]']
dense_10 (Dense)	(None, 1, 22, 64)	8256	['sequential_4[0][0]']
dense_11 (Dense)	(None, 1, 22, 64)	8256	['sequential_4[1][0]']
lambda_1 (Lambda)	(None, 1, 22, 64)	0	['dense_10[0][0]', 'dense_11[0][0]']
sequential_5 (Sequential)	(None, 1, 22, 250)	40570	['lambda_1[0][0]']
concatenate_1 (Concatenate)	(None, 1, 22, 378)	0	['dense_10[0][0]', 'dense_11[0][0]', 'sequential_5[1][0]']

Total params: 105,722
Trainable params: 105,722
Non-trainable params: 0

Figure 3: VAE Architecture

Layer (type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 250, 22, 25)	275
conv2d_6 (Conv2D)	(None, 250, 1, 25)	13775
max_pooling2d_4 (MaxPooling2D)	(None, 84, 1, 25)	0
batch_normalization_4 (BatchNormalization)	(None, 84, 1, 25)	100
dropout_4 (Dropout)	(None, 84, 1, 25)	0
conv2d_7 (Conv2D)	(None, 84, 1, 50)	12550
max_pooling2d_5 (MaxPooling2D)	(None, 28, 1, 50)	0
batch_normalization_5 (BatchNormalization)	(None, 28, 1, 50)	200
dropout_5 (Dropout)	(None, 28, 1, 50)	0
conv2d_8 (Conv2D)	(None, 28, 1, 100)	50100
max_pooling2d_6 (MaxPooling2D)	(None, 10, 1, 100)	0
batch_normalization_6 (BatchNormalization)	(None, 10, 1, 100)	400
dropout_6 (Dropout)	(None, 10, 1, 100)	0
conv2d_9 (Conv2D)	(None, 10, 1, 200)	200200
max_pooling2d_7 (MaxPooling2D)	(None, 4, 1, 200)	0
batch_normalization_7 (BatchNormalization)	(None, 4, 1, 200)	800
dropout_7 (Dropout)	(None, 4, 1, 200)	0
flatten_1 (Flatten)	(None, 800)	0
dense_1 (Dense)	(None, 4)	3204

Total params: 281,604
Trainable params: 280,854
Non-trainable params: 750

Figure 4: Base CNN Architecture

Model	Test Accuracy
Base CNN	0.722
CNN+LSTM	0.619
CNN+LSTM (modified)	0.679
CNN+VAE	0.248

Table 1: Test Accuracy across all categories for each model

Model	No. of Epochs	Optimizer	Learning rate	Loss
Base CNN	200	Adam	5e-4	Cross-Entropy
CNN+LSTM	100	Adam	5e-4	Cross-Entropy
CNN+LSTM (modified)	100	Adam	5e-4	Cross-Entropy
CNN+VAE	200	Adam	-	Reconstruction loss + KL loss

Table 2: Hyperparameters

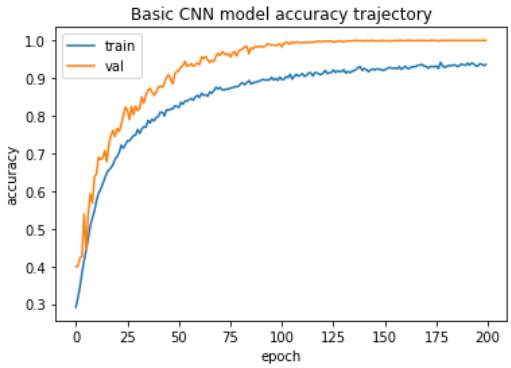

Figure 5: Base CNN model training accuracy

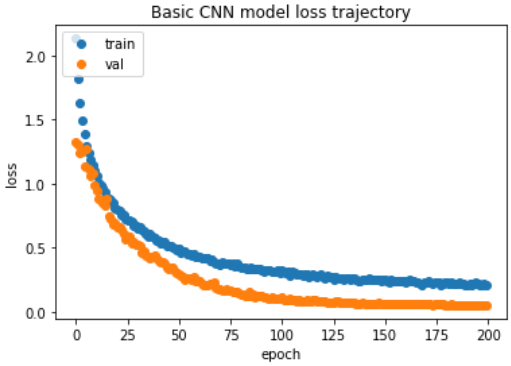

Figure 6: Base CNN model loss

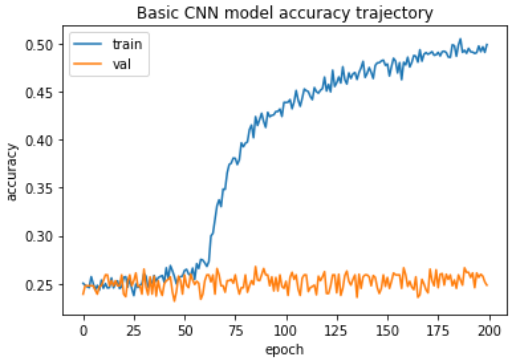

Figure 7: CNN+VAE model training accuracy

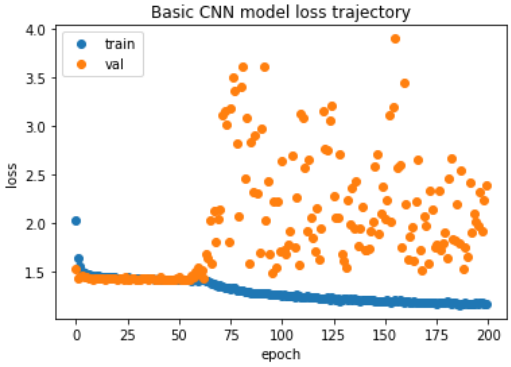

Figure 8: CNN+VAE model loss