C147/C247 Final Project Write-Up

Neil Kane
UCLA
105306580
neilkane1317@g.ucla.edu

Brendan Rossmango
UCLA
505370692
bmango643@gmail.com

Akshay Gupta
UCLA
305414381
akshaygupta.us@gmail.com

Abstract

We analyzed EEG Data using a baseline CNN model, CNN + LSTM, and VAE. The baseline CNN model and CNN+LSTM model were trained with hyperparameter optimization using the entire dataset as well as only person 1 data. Additionally, the amount of time fed into the models was varied. The data was also augmented using the given data augmentation function and slightly altered to vary how many time samples were provided. The paper also analyzes problems faced with using VAE for data augmentation and generation.

1. Introduction

For this project we went with the default EEG dataset [1] and considered a baseline CNN model in addition to CNN + LSTM and CNN + VAE. For each of these models the work done along with the motivation behind it will be discussed.

1.1. Vanilla CNN

Before considering any of the post-CNN models, the given baseline model with the data augmentation was extended to answer the given problems. Specifically hyperparameter training was performed for each individual combination of type of input data (entire or person 1) along with time (the number of provided time samples was varied) to create unique model weights for using the same high level CNN architecture. This would then serve as a reference point that the other models results can then be compared with.

The specific optimizations were performed by trying different combinations of the initial number of filters (which scaled up by 2 after every CNN layer), kernel size, pool kernel size, pool type, and activation function. This was iteratively done as the many permutations would take far too long and together a total of 6 models were created. The model trained on the entire data set was evaluated on each individual person as well as the entire test set while the person 1 models were only evaluated on person 1 samples from the test set. The results from the best 6 models for each arrangement of input data set and amount of time will be discussed with the entire results being available in ”cnn_results.ipynb”. Furthermore the trained models and model weights will also be provided for each of these arrangements.

1.2. CNN + LSTM

After optimizing the baseline CNN model, we also extended the given CNN+LSTM hybrid and used the same data preprocessing method - trimming, maxpooling, averaging, adding noise, subsampling, and dimension reshaping. We did optimizations over the structure of the CNN+LSTM, number of layers, number of units per convolutional layer and LSTM, regularization factors, learning rates for the optimizers, pool kernel size, and dropout factors. Likewise in the base CNN model, the number of filters was doubled for each CNN layer.

First, we optimized for the first subject, training on just subject 1’s data and then over all the data and testing on subject 1’s data. Then we trained on all data and tested on all data, and finally we trained the model over a function of time: 10 models each with different time periods from 25 to 250. The results are found in ”CNN_LSTM.ipynb”.

1.3. CNN + VAE

In addition to using CNN and LSTM, our study also included an unsupervised learning approach using a generative model: the Variational Autoencoder (VAE), to create artificial EEG data. For this, we created two components: an encoder and a decoder. Both components will use deep neural networks to compress the high-dimensional EEG data into a lower-dimensional space, and then map the transformed data back into a higher-dimensional space. Through this, we aim to have the VAE learn a probabilistic model of the input data distribution and eliminate noise present in the data. By doing this, the artificially created data will be more meaningful to any CNN that is fed this data, allowing for the CNN to make more accurate predictions. In the following section, we will show the results of this VAE approach.

2. Results

2.1. Vanilla CNN

Overall when considering the best CNN model from Table 1 in the Appendix we observe that the best overall accuracy when using the entire dataset with the relevant samples (500 using the provided data augmentation) achieved an accuracy of 70.71%. From the Entire 500 samps confusion matrix we also see that this model was able to perform quite well for label 1 corresponding to cue onset right. Interestingly when reducing the samples to 300 from Table 2, the best accuracy achieved 68.28% which is quite similar but from the confusion matrix we see that it actually performs better for labels 0 (cue onset left) and 3 (cue onset tongue) while performing worse in labels 1 (cue onset right) and 2 (cue onset foot).

Comparing inputs with less time samples we do see the expected trend with accuracy decreasing on all people however comparing results for person 1 we see from Tables 4 and 5 that the accuracy is actually better with 300 samples compared to 500. From those tables and also Table 6 we see that that accuracy among specific people is not uniform with some notable values such as 79.00% for person 8 using 500 samples contrasting with the 62.50% for person 1.

Table 1: CNN, Entire Dataset (500 samps)

Models	Accuracy(%)
(25,20,2,Avg,elu)	59.82
(12,20,2,Avg,elu)	64.79
(12,10,2,Avg,elu)	66.70
(12,10,3,Avg,elu)	70.71

Table 4: CNN, Best Entire on Individuals (500 samps)

Person	Accuracy(%)
1	62.50
2	59.50
3	79.00
4	72.50
5	79.26
6	73.47
7	71.50
8	79.00
9	78.19

Table 7: CNN, Person 1 (500 samps)

Models	Accuracy(%)
(25,20,2,Avg,elu)	51.50
(5,20,2,Avg,elu)	65.50
(5,15,2,Avg,elu)	61.50
(5,15,3,Avg,elu)	70.00

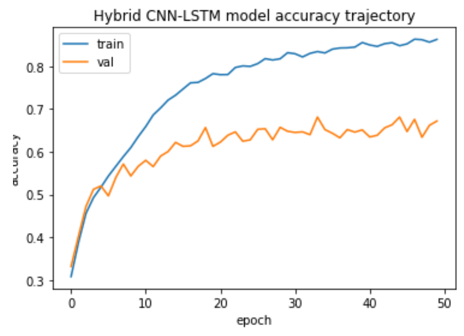


Figure 1. CNN+LSTM, All Data, Accuracy Trajectory

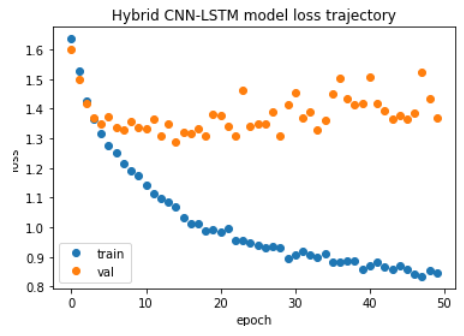


Figure 2. CNN+LSTM, All Data, Loss Trajectory

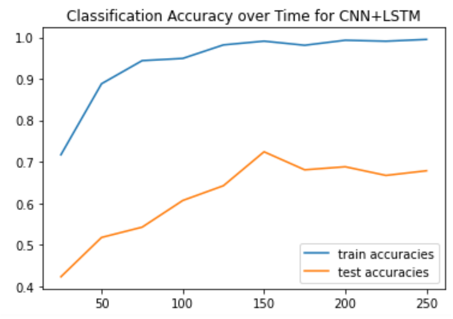


Figure 3. CNN+LSTM, Accuracy Over Time

2.2. CNN + LSTM

The best CNN+LSTM model when using the entire dataset with the trimmed samples (500) achieved an accuracy of 71.39%. When training on just subject 1's data and testing on subject 1, we get a lower classification accuracy of 60%, and when training on the entire dataset and testing on subject 1, the classification accuracy is even lower at 58%. Finally, in our evaluation of the classification accuracy as a function of time, we achieved the highest accuracy for the CNN+LSTM at 72.4% at time period t = 150.

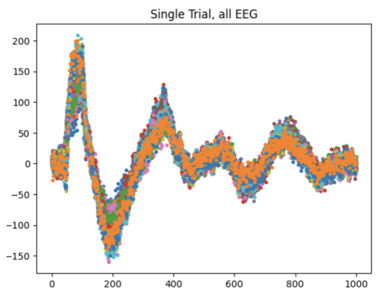


Figure 4. CNN+VAE, All Data, Single Trial. Artificially generated EEG data sample for all 22 channels from the VAE model. The sample resembles an EEG signal, however all 22 channels follow the same distribution which is clear sign of failure by the VAE model.

2.3. CNN + VAE

Unfortunately, we failed to train our VAE to generate strong enough data to warrant feeding it into a CNN. In the figure below, we show a sample artificial EEG signal generated by the model. In this figure, we see what appears to be a valid EEG signal. However, upon further examination we noticed that all 22 of the different probe channels, represented by each of the individual colors follow the same pattern and are essentially the same signal according to the model. This is not the case with real data. Given that the data was clearly incorrect and because of the time constraints present, we were unable to correct this problem.

3. Discussion

3.1. Vanilla CNN

From the initial data augmentation provided we are aware that after 500 time samples much of the EEG data is the same among each of the labels. However the results described earlier do give some insight on predicting the left and tongue directions. We saw that using only 300 time samples lead to better performance for these two labels with degradation in right and foot for an overall decrease in accuracy. This can possibly mean that for predicting left and tongue the first 300 samples provide a greater degree of contrast that can be learned by the CNN however after 300 samples much of the data is the same.

Furthermore from the results we also know that EEG data from person to person is not the same and from Tables 4,5, and 6 we see stark differences in the accuracy. This could be due to variations in how the EEG data was captured and the CNN may have learned its model weights based on the people with more contrasting signals for each of the labels. We see for instance person 5 performed fairly consistently and always achieved greater than 70% accuracy even when only using the first 100 samples.

3.2. CNN + LSTM

A number of fine-tuning methods were used to improve the base CNN+LSTM given to us. We tried many things like adding dropout layers after the CNN layers, kernel L2 regularizers, more CNN layers, more LSTM layers, different learning rates for the Adam optimizer, and more. Adding more CNN or LSTM layers made both the training and test accuracies worse because the model became overly complex. Adding dropout layers and regularizers helped reduce the training accuracy (in some trials, the training accuracy reached 1.0, which means the model was overfitting to the data, so adding regularizers helped). Using different learning rates instead of 0.001 did not help - the learning stopped sooner because either the gradients zigzagged or plateaued. Finally, changing the number of filters for each CNN layer to a power of 2 resulted in better results.

For the first subject's results, the accuracy was worse when we trained on the entire dataset because the model learned patterns from all subjects 0-8 and applied these patterns to the first subject. It performed better when learning patterns only from subject 1. However, the accuracy for subject 1 when training only on subject 1's data is still lower than the accuracy when using the entire dataset because the dataset size for only subject 1 is not large enough, so complex patterns about the data cannot be learned when compared to using the entire dataset.

3.3. CNN + VAE

The problem with our VAE model was that it mapped individual signals onto the same random distribution in the latent space. Research of the problem indicates that the cause of the VAE mapping everything to the same distribution is called posterior collapse: a phenomena where the learned latent space has very little information of the input space causing the decoder to "oversmooth" and generate relatively arbitrary output [2]. One cause named by Takida is an inappropriate choice of posterior variance, which can introduce a sub-optimal local optima leading to posterior collapse.

The posterior collapse resulted in our ability to create a plausible dataset suitable for training our CNN. Due to time constraints, we were unable to fix the problem. Research suggests that this problem may be solved by exploring alternative models like Wasserstein Auto-Encoders (WAE) or Regularized Auto-Encoders with Gradient Penalty (RAE-GP) [2]. These models are deterministic autoencoders by providing a more robust but expensive framework to avoid the loss of information in the latent space, and thus avoid mapping different inputs to the same output [2]. In further experiments, we hope to either better tune our posterior variance or apply a deterministic auto encoder in order to effectively generate artificial EEG data suitable for a CNN.

References

[1] C. Brunner et al. *BCI Competition 2008 – Graz data set A*. URL: https://www.bbci.de/competition/iv/desc_2a.pdf.

[2] Yuhta Takida et al. *Preventing Oversmoothing in VAE via Generalized Variance Parameterization*. URL: https://arxiv.org/abs/2102.08663. (accessed: 03.20.2023).

4. Appendix

As mentioned earlier the Vanilla CNN uses the provided 4-layer architecture with hyperparameter training. The following tables show what parameters were used for the number of filters for the first layer (doubles for each subsequent layer), kernel length, pool size, pool type, and activation.

Table 2: CNN, Entire Dataset (300 samps)

Models	Accuracy(%)
(25,20,2,Avg,elu)	62.81
(12,20,2,Avg,elu)	65.18
(12,10,2,Avg,elu)	67.95
(12,10,3,Avg,elu)	68.28

Table 3: CNN, Entire Dataset (100 samps)

Models	Accuracy(%)
(25,20,2,Avg,elu)	50.85
(12,20,2,Avg,elu)	53.27
(12,10,2,Avg,elu)	52.09
(12,10,3,Avg,elu)	55.02

Table 5: CNN, Best Entire on Individuals (300 samps)

Person	Accuracy(%)
1	65.00
2	60.00
3	72.00
4	62.50
5	76.60
6	75.00
7	74.00
8	78.00
9	77.66

Table 6: CNN, Best Entire on Individuals (100 samps)

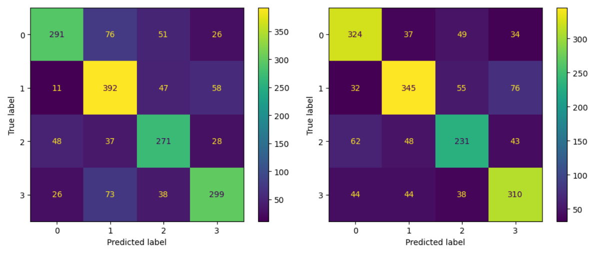


Figure 5. CNN, Entire 500 samps and Entire 300 samps

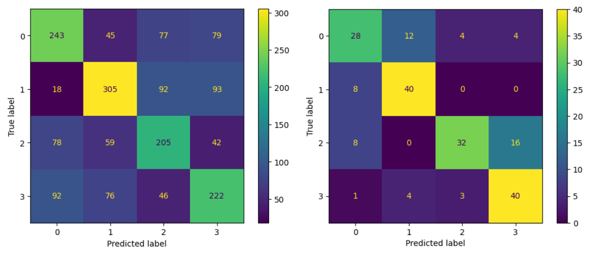


Figure 6. CNN, Entire 100 samps and Person 1 500 samps

Person	Accuracy(%)
1	53.50
2	43.50
3	49.00
4	62.50
5	72.87
6	63.27
7	71.50
8	53.00
9	53.19

Table 8: CNN, Person 1 (300 samps)

Models	Accuracy(%)
(25,20,2,Avg,elu)	46.00
(5,20,2,Avg,elu)	43.00
(5,15,2,Avg,elu)	57.50
(5,15,3,Avg,elu)	64.00

Table 9: CNN, Person 1 (100 samps)

Models	Accuracy(%)
(25,20,2,Avg,elu)	43.50
(5,20,2,Avg,elu)	42.50
(5,15,2,Avg,elu)	43.50
(5,15,3,Avg,elu)	45.50

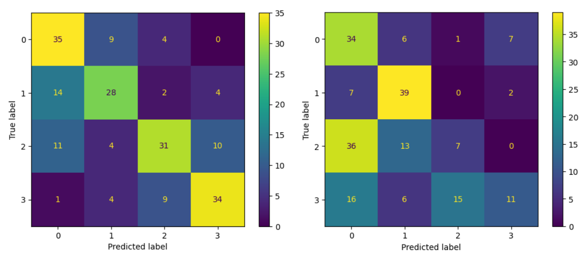


Figure 7. CNN, Person 1 300 samps and Person 1 100 samps

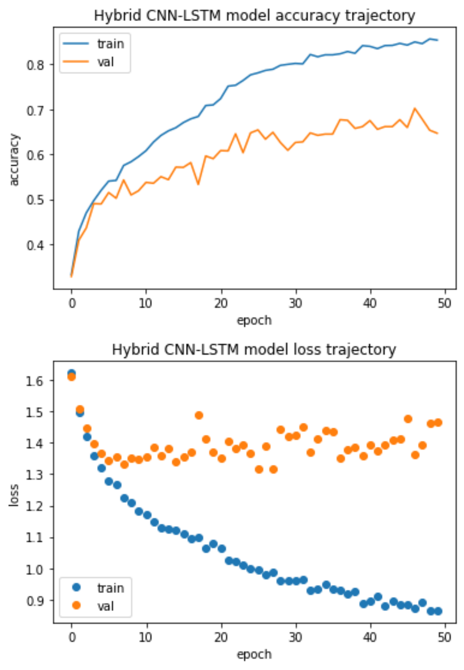


Figure 8. CNN+LSTM, Subject 1 trained on all data, Accuracy and Loss Trajectory

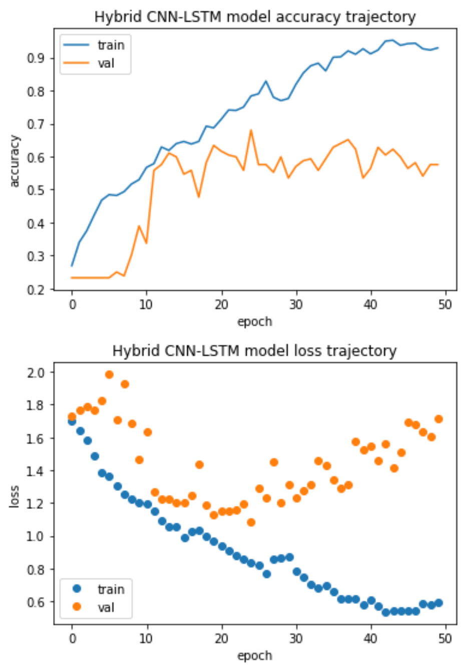


Figure 9. CNN+LSTM, Subject 1 trained on subject 1 data, Accuracy and Loss Trajectory

5. Performance

Performance of each model

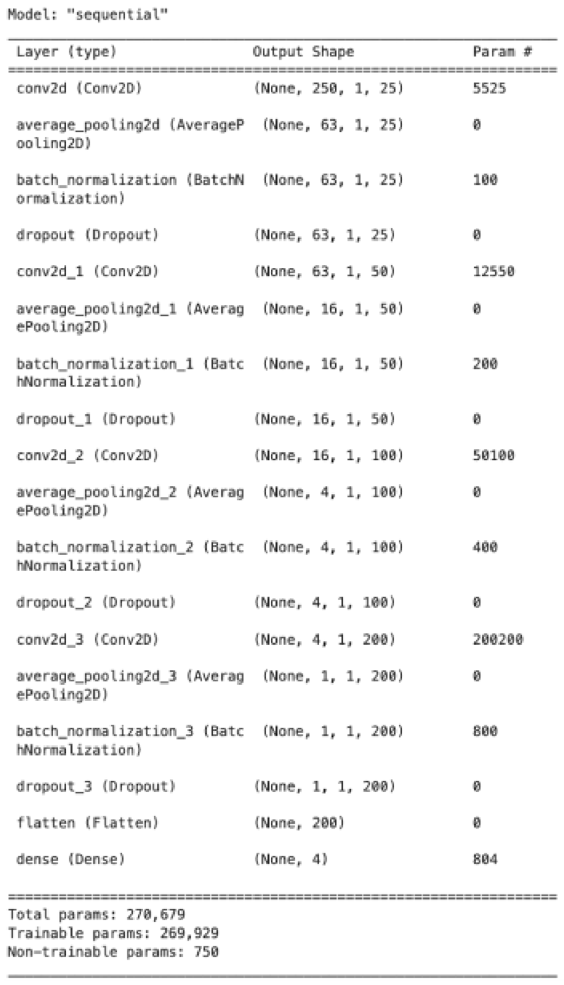
Model	Accuracy(%)
CNN, all data, 500 samples	70.71
CNN, all data, 300 samples	68.28
CNN, all data, 100 samples	53.27
CNN, s1 data, 500 samples	70
CNN, s1 data, 300 samples	64
CNN, s1 data, 100 samples	45.5
CNN+LSTM, all data	71.39
CNN+LSTM, s1 data, train s1 data	60
CNN+LSTM, s1 data, train all data	58
CNN+LSTM, all data over time, best acc	72.4

6. Architectures

The architecture for the CNN is a 4-layer CNN with a fully connected layer with a softmax activation. Each layer has an average pooling of size 4, and each layer is followed by a batch normalization and dropout layer. The number of filters for layer 1 is 25, and each subsequent layer doubles the number of filters, where the 4th layer has 200 filters. There is no regularization done in the layers because batch normalization and dropout is already employed.

The architecture for the CNN+LSTM is a 4-layer CNN followed by a 30-unit LSTM and a fully-connected layer with a softmax activation. A dropout layer, p = 0.5, follows each CNN and LSTM layer, and a batch normalization layer follows each CNN layer. Each CNN layer has an L2 regularization factor of 0.001 a max pool size of 4, and an exponential linear unit activation. The first CNN layer has 32 filters, and each subsequent layer doubles the number of filters, where the 4th layer has 256 filters. The FC layer also has an L2 regularization factor of 0.001.

Both the CNN and CNN+LSTM used the same data augmentation process of trimming, maxpooling, subsampling, averaging with noise, and dimension reshaping. Both used the Adam optimizer and exponential linear unit activations.



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 250, 1, 25)        5525

 average_pooling2d (AverageP  (None, 63, 1, 25)        0
 ooling2D)

 batch_normalization (BatchN  (None, 63, 1, 25)        100
 ormalization)

 dropout (Dropout)           (None, 63, 1, 25)         0

 conv2d_1 (Conv2D)           (None, 63, 1, 50)         12550

 average_pooling2d_1 (Averag  (None, 16, 1, 50)        0
 ePooling2D)

 batch_normalization_1 (Batc  (None, 16, 1, 50)        200
 hNormalization)

 dropout_1 (Dropout)         (None, 16, 1, 50)         0

 conv2d_2 (Conv2D)           (None, 16, 1, 100)        50100

 average_pooling2d_2 (Averag  (None, 4, 1, 100)        0
 ePooling2D)

 batch_normalization_2 (Batc  (None, 4, 1, 100)        400
 hNormalization)

 dropout_2 (Dropout)         (None, 4, 1, 100)         0

 conv2d_3 (Conv2D)           (None, 4, 1, 200)         200200

 average_pooling2d_3 (Averag  (None, 1, 1, 200)        0
 ePooling2D)

 batch_normalization_3 (Batc  (None, 1, 1, 200)        800
 hNormalization)

 dropout_3 (Dropout)         (None, 1, 1, 200)         0

 flatten (Flatten)           (None, 200)               0

 dense (Dense)               (None, 4)                 804

=================================================================
Total params: 270,679
Trainable params: 269,929
Non-trainable params: 750
_________________________________________________________________
```

Figure 10. CNN Architecture

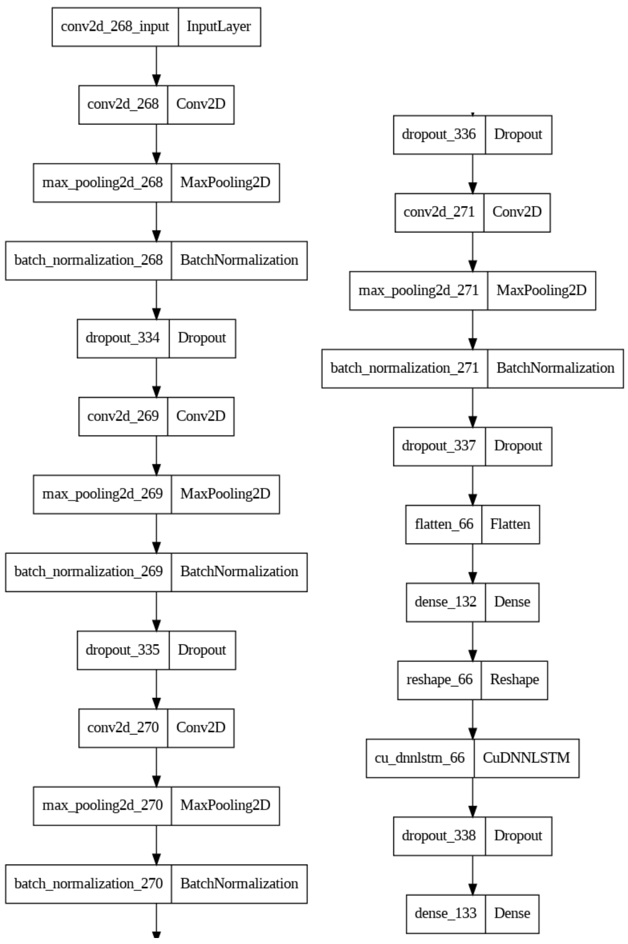

conv2d_268_input | InputLayer
conv2d_268 | Conv2D
max_pooling2d_268 | MaxPooling2D
batch_normalization_268 | BatchNormalization
dropout_334 | Dropout
conv2d_269 | Conv2D
max_pooling2d_269 | MaxPooling2D
batch_normalization_269 | BatchNormalization
dropout_335 | Dropout
conv2d_270 | Conv2D
max_pooling2d_270 | MaxPooling2D
batch_normalization_270 | BatchNormalization
dropout_336 | Dropout
conv2d_271 | Conv2D
max_pooling2d_271 | MaxPooling2D
batch_normalization_271 | BatchNormalization
dropout_337 | Dropout
flatten_66 | Flatten
dense_132 | Dense
reshape_66 | Reshape
cu_dnnlstm_66 | CuDNNLSTM
dropout_338 | Dropout
dense_133 | Dense

Figure 11. CNN+LSTM Architecture

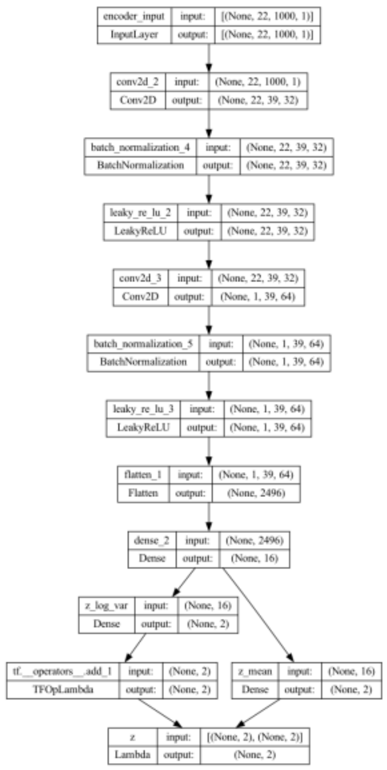

encoder_input | input: | [(None, 22, 1000, 1)]
InputLayer | output: | [(None, 22, 1000, 1)]
conv2d_2 | input: | (None, 22, 1000, 1)
Conv2D | output: | (None, 22, 39, 32)
batch_normalization_4 | input: | (None, 22, 39, 32)
BatchNormalization | output: | (None, 22, 39, 32)
leaky_re_lu_2 | input: | (None, 22, 39, 32)
LeakyReLU | output: | (None, 22, 39, 32)
conv2d_3 | input: | (None, 22, 39, 32)
Conv2D | output: | (None, 1, 39, 64)
batch_normalization_5 | input: | (None, 1, 39, 64)
BatchNormalization | output: | (None, 1, 39, 64)
leaky_re_lu_3 | input: | (None, 1, 39, 64)
LeakyReLU | output: | (None, 1, 39, 64)
flatten_1 | input: | (None, 1, 39, 64)
Flatten | output: | (None, 2496)
dense_2 | input: | (None, 2496)
Dense | output: | (None, 16)
z_log_var | input: | (None, 16)
Dense | output: | (None, 2)
tf.__operators__.add_1 | input: | (None, 2)
TFOpLambda | output: | (None, 2)
z_mean | input: | (None, 16)
Dense | output: | (None, 2)
z | input: | [(None, 2), (None, 2)]
Lambda | output: | (None, 2)

Figure 12. CNN+VAE Encoder Architecture

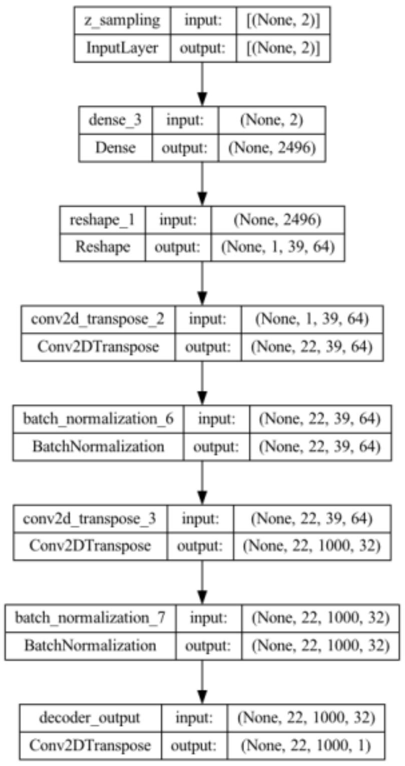

z_sampling | input: | [(None, 2)]
InputLayer | output: | [(None, 2)]
dense_3 | input: | (None, 2)
Dense | output: | (None, 2496)
reshape_1 | input: | (None, 2496)
Reshape | output: | (None, 1, 39, 64)
conv2d_transpose_2 | input: | (None, 1, 39, 64)
Conv2DTranspose | output: | (None, 22, 39, 64)
batch_normalization_6 | input: | (None, 22, 39, 64)
BatchNormalization | output: | (None, 22, 39, 64)
conv2d_transpose_3 | input: | (None, 22, 39, 64)
Conv2DTranspose | output: | (None, 22, 1000, 32)
batch_normalization_7 | input: | (None, 22, 1000, 32)
BatchNormalization | output: | (None, 22, 1000, 32)
decoder_output | input: | (None, 22, 1000, 32)
Conv2DTranspose | output: | (None, 22, 1000, 1)

Figure 13. CNN+VAE Decoder Architecture