# A Deep Learning Study on Electroencephalography Data

Blake Gella    Howard Zhang    Nathan Pereira    Premkumar Giridhar

## Abstract

*In this paper, we experiment with different techniques for classifying and augmenting data collected from an electroencephalogram (EEG) device. This is extremely noisy, one-dimensional time-series data over 22 different channels, which poses a great challenge for the classification task. We test many neural network architectures such as traditional CNNs, LSTMs, multi-headed self-attention layers, diffusion models, and VAEs to study their impact on the final classification accuracy. We found that each model architecture had its own drawbacks and that the traditional CNN performed the best given the amounts of training data provided, yielding a test accuracy of 70.4%.*

## 1. Introduction

### 1.1. Diffusion Models

A diffusion model works by scheduling noise of varying strengths over a range of time steps [2, 7]. During training, the noise at a randomly selected time step is added to an input, and the model will learn to predict the noise to remove it. During inference, the input can be initialized to noise or a given signal/image. The model will progressively denoise and noise the input across all time steps, resulting in a synthetic output. This process can be used to synthetically augment the training set for the downstream classification task.

We converted the original 2D implementation of the diffusion model into 1D. We also added cross attention blocks to the UNET to make the model know which class to create. We referenced the noising process from the hugging face official implementation [5]. Due to how much harder noise is to distinguish from data in 1D, we passed the EEG data through an FFT before entering the model and an IFFT after leaving the model. When training a model on both synthetic and real data, the synthetic data was split into both the training and validation sets so validation also show improvement and callbacks like early stopping can be used. Although the original DDPM paper used 1000 time steps, we used 50 time steps to reduce computation time, as we had limited compute units. To compensate, we increased the beta noise schedule so each time step denoises strongly. In addition, we also added the ability to initialize the input signal to something other than noise, meaning a signal can be inputted to the denoising process. We chose to use an L2 reconstruction loss instead of the traditional L1 loss, as previous papers have shown the increased variability and therefore expressive power of L2 loss models [6].

### 1.2. Variational Autoencoders

Variational autoencoders use an encoder to project the input data into a lower dimensional latent space, and decodes it back to the original data [4]. It trains on a reconstruction loss and a similarity loss with normal distributions (with KL divergence).

Variational autoencoders are often used for cases where we need to generate new data similar to an input sample that is different in a specific way - for example, adding certain features to an input image that weren't there before. However, we can also sample the latent space directly to generate new data that is similar in features to our training data while being generalizable in a similar manner, possibly to help augment our training data for the classification model.

### 1.3. Long Short Term Memory Networks

LSTMs are a type of RNN used to process time-series data [3]. Unlike RNNs, LSTMs have a more direct link from earlier time steps to the output through the use of "cell states". With EEG being time-series data, it seems only natural to attempt the classification task with an LSTM. Leveraging the "memory" mechanism of LSTMs could help the final layers of the network take into account EEG signals at earlier time steps for classification.

### 1.4. Multi-Headed Self-Attention Blocks

MHA layers were first introduced to deep learning through NLP, where global attention through weighted connections between words aided in many language processing tasks. Other papers have recently realized the added benefit of global attention in the context of other fields, such as image recognition [1]. This global attention could aid in the EEG classification task; allowing earlier time blocks to communicate and be analyzed in comparison to more recent ones could help provide key insights for the model to predict the correct output class.

## 2. Results

All of our tables are given in the Appendix. In table 1, we show the test accuracy of multiple model architectures(CNN, MHA, LSTM) when trained on either one subject or all subjects. We test on each subject individually as well as all together. We see that models trained on all subjects perform much better. We also see that traditional CNNs perform better than LSTMs and MHA. All of these models, except for the optimized CNN, were trained with the following data augmentations: max pooling, average pooling with noise, min pooling with noise, 1.2 scaling, 0.8 scaling, and subsampling. We note that training with these augmentations increased the test accuracy from 54.2% to 69.5% (on CNNs on all test subjects). Moreover, upon optimizing the CNN architecture (discussed in the methods section), we were able to reach a test accuracy of 70.4%.

Table 2 includes test accuracy when training and testing on shorter time windows. We see the highest results on a model with a time window of 768.

Table 3 includes test accuracy when adding synthetic data to the training set. The test accuracies are lower than the base CNN with all real, augmented data. However, we see a slight upward trend when increasing the amount of synthetic data. However, we were unable to generate more than 8192 trials due to limited GPU compute resources on Google Colab.

We also were able to train a variational autoencoder with a reconstruction loss of 30. However, given the time constraints and limited gpu hours, we were unable to test it on a CNN model or use it for data generation.

## 3. Discussion

### 3.1. On CNN

CNNs performed the best out of any model, and we were able to reach 70.4% in an optimized version. They may perform well due to their ability to handle both local features directly and global features indirectly. Since the entire signal is seen at the input of a model and every neuron at the end sees a large portion of input through the receptive fields, the model can make informed decisions with features throughout the trial. Moreover, the CNN does not have as many parameters as other models and does not have many layers to backpropagate through, meaning the model does not have to learn at much (less overfitting) and does not suffer from a vanishing gradient problem.

### 3.2. On Single-Subject Training

When optimizing the classification accuracy by training only on subject 1, we get very poor results. These results are significantly improved by training on the entire dataset across all subjects; as stated before, this is simply due to the much larger amount of data at our disposal. The improved results suggest that the accuracy is heavily dependent on amount of training data. We therefore add further data augmentations beyond max pooling, average pooling, and subsampling.

Augmentation *did* help improve the classification accuracy. With limited training data, the model was prone to overfitting on noise and had to be regularized (or stopped early) to prevent it. However, by creating and including additional training data, we were able to help the model identify actual features (as opposed to noise) more easily. However, our augmentation techniques were limited to very simple techniques such as pooling and sub-sampling; attempts to augment with the diffusion model performed worse overall.

### 3.3. On Shorter Time Windows

We found that the optimal time window was around 768 time steps. This is likely the case since larger time windows of 1000 could result in too much (possibly redundant) information being provided to the model. The inherent noise of the data could then lead to a poor result. On the other hand, shorter time windows obviously have the issue of not including key features of the data that could help in classification. This trend is seen in Fig. 3.

### 3.4. On Diffusion

Although the reconstruction loss in the diffusion model was as low as 0.5, the output of a diffusion process initialized to random noise did not have an EEG structure, resembling noise rather than a proper EEG signal, like in Fig. 1. However, when the model was initialized with the class' average EEG, the shape of the EEG output looked very natural and had variety of shapes, as shown in Fig. 2. This could be attributed to the data being 1D, in which noise is harder to distinguish from the data. Moreover, as the diffusion process' loss is based on predicting the noise, having any noise in the training data makes the model heavily biased towards outputting noisy data, as it learns to create noise.

Initializing on the average of the class for each channel helps alleviate this issue. Using this approach, the model reproduces noise similar to that found in the training data while also promoting a unique shape for each data instance.

Nevertheless, the results from training the CNN on real and synthetic data yields a worse test accuracy than only training on real data. This could be attributed to the limitations of diffusion in 1D noise. It could also be attributed to the inherent noise in the EEG data, which interferes with training. However, we do see a slight upward trend when including more synthetic data. This could be because the variance within the synthetic data improves the model's ability to generalize to real data as more synthetic data is added to the training set. Further work could include generating a larger-scale dataset(with enough compute units). Further work could also include cross attention for each subject, which could improve the shape of synthetic EEGs.

## 3.5. On LSTMs

We found that LSTMs performed poorer than traditional CNN models. A likely reason why is the LSTM placing greater weight on later time steps. Although they are capable of using "cell blocks" to "remember" previous time steps, they favor the features of the final time steps over the previous ones. This makes sense when outputting phrases in natural language, since the upcoming word would likely be most influenced by the word immediately preceding it. However, for EEG data, the class of a particular signal is most likely dependent on important features in the middle, not necessarily the last few steps. Further work could be done to improve upon this architecture for EEG data, such as utilizing LSTMs in a short time window to output features, which are then fed into a CNN for downstream classification.

## 3.6. On Multi-headed Self-attention blocks

Multi-headed self-attention (MHA) blocks are able to manipulate the weight they place on different time steps unlike LSTMs. However, they suffer from the issue of increased global attention. Important features that could lend a hand in classification are typically localized in particular areas in the signal. MHA blocks typically place equal importance on all time steps, meaning important steps are weighted just as much as unimportant steps. We depend on model training to find the correct weights later on. Given, the low training data, it would make more sense

## 3.7. On VAEs

Due to the time constraints and the limited compute hours, we were unable to implement a VAE in a CNN model or use it as a data generation method. However, by achieving a low loss, we were able to show that a model that utilizes a VAE is possible. Future works could discuss if this reduces loss and, thus, could improve CNNs and Diffusion models. Future works could also discuss if inputting from the latent space is better than inputting the original signal for test accuracy.

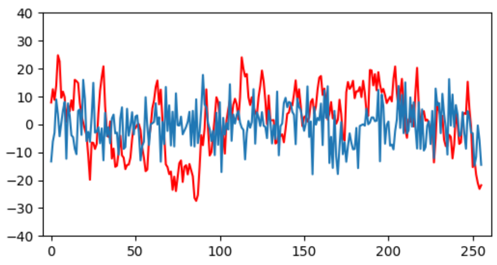

Figure 1. Generated trial with diffusion model with random noise input.

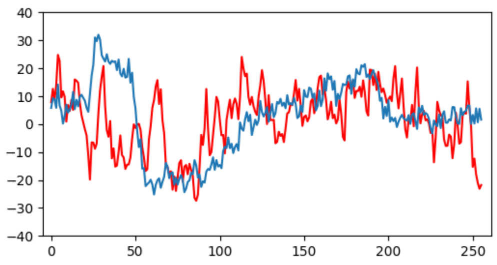

Figure 2. Generated trial with diffusion model with per-channel average as input.

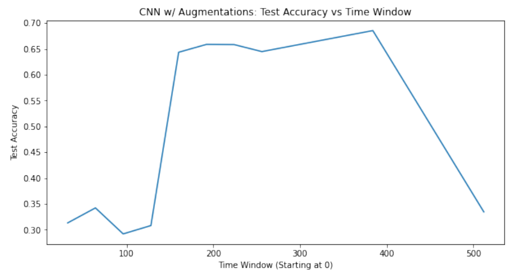

Figure 3. Graph showing test accuracy when training and testing on trials of different time windows.

## References

[1] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 1

[2] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020. 1

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

[4] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. 1

[5] Rogge and Rasul. The annotated diffusion model. https://huggingface.co/blog/annotated-diffusion. Accessed: 2023-03-14. 1

[6] Chitwan Saharia, William Chan, Huiwen Chang, Chris Lee, Jonathan Ho, Tim Salimans, David Fleet, and Mohammad Norouzi. Palette: Image-to-image diffusion models. In *ACM*

*SIGGRAPH 2022 Conference Proceedings*, pages 1–10, 2022. 1

[7] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR, 2015. 1

## 4. Methods

Unless specified otherwise, all models ran with data augmentations, specifically max pooling, average pooling with noise, min pooling with noise, 1.2 scaling, 0.8 scaling, and subsampling. All models were trained using an ADAM optimizer and 1e-3 learning rate. For better performance, gradient accumulation equivalent to the total amount of batches in an epoch was used for each model. Moreover, each model uses gradient clipping of 2. All models also use cross entropy loss. Callbacks of early stopping and learning rate reduction on plateau were used to train models quicker, except on the optimized CNN.

### 4.1. CNN w/o Augmentations

The CNN model without data augmentations has four layers of 1D convolution, swish activation, maxpool of 3, 1D batchnorm, and a dropout of 0.4. This was a model developed at the beginning that we wanted to update with robust activations. Moreover, we found that a dropout of 0.4 was optimal in pytorch. The kernel sizes were the following: 25, 50, 100, 200. We found that increasing or decreasing any of the kernel sizes decreased the test accuracy, so we kept these as kernel sizes.

### 4.2. CNN w/ Augmentations

The CNN model with augmentations was adjusted to work with variable time windows. The architecture follows the previous model. The activation was changed to ELU, as we found it worked better for smaller time windows. We also adjusted the kernel size to progress from 10 to 11 and dropout from 0.5 to 0.4. We did the latter so certain inputs would be not be relied on, but intermediate calculations would still be preserved.

### 4.3. Optimized CNN w/ Augmentations

The optimized CNN model with augmentations is like the base CNN model with augmentations, but the kernel size is 32, 64, 100, 200. We removed the data augmentations of scaling and minpooling, as we determined those yielded high validation accuracy but low test accuracy. We thought that the beginning layers were bottle-necked too much, so we increased the number of filters to allow more information to pass through. We also removed early stopping callbacks and LR reduction on plateau to allow the model to learn more, even if there are temporary increases in validation loss.

### 4.4. CNN+LSTM w/ Augmentations

The CNN+LSTM model follows the CNN model without the final convolution block and with two bidirectional LSTM modules before the FC. The first LSTM has 128 hidden units and the second LSTM has 32 hidden units. We did this to force the model to bottle neck before the output, meaning less parameters had to be learned.

### 4.5. CNN+MHA w/ Augmentations

The CNN+MHA model follows the CNN model with a multihead self-attention module after the convoutional blocks and before the fully connect layer. We chose to use 4 heads, as that would evenly split the 200 embedding dimension.

### 4.6. Diffusion Model

The Diffusion Model is a UNET with passthrough for each platform. There are 4 platforms, each containing two resblocks, one residual and attention block, and one up/down sampling block. The amount of filters of all resblocks started at 192 and would double for each subsequent layer until the bottom of the UNET was reached. At the bottom, there are one resblock, one residual and attention block, and then one resblock. At the end of unet is a projection layer. We implemented time embedding shifting in the resblocks so the time step would have more strength changing the values. We also implemented self attention and cross attention with the embedding of the class. We did this to have more control over what classes to produce and allow features to be referenced between different classes.

The training procedure of the diffusion model follows the original DDPM paper. We use a linear beta scheduler and denoise EEG data at different time steps. To make an inference, we initialize an input to noise or a signal and continually denoise and noise until reaching the last time step. We differed from the DDPM paper by decreasing time steps to 50 and increasing the noise schedule to 0.02-0.2. While this hurt performance, we did this to decrease inference time, which allowed us to generate more samples on the limited GPU hours we had. L1 and L2 losses were also tested here, but we decided with L2 loss due to the Palette paper (mentioned in the main report).

We then generated 8192 samples of EEG data (1024 for each class). We split 5000 for training and 3192 for validation. We trained a base CNN model with augmentations varying percents of the 8192 trials.

### 4.7. VAE

As we did not include this in our results, this section will be kept short. We used 6 1d-convolutions (with kernel size 11) with batch normalizations and HardSwish activations. Then 2 parallel linear layers for getting the latent space mean and variance. We then used 6 1d transpose convolutions with the same activations and normalizations to project it back to input dimensions. We used a L2 reconstruction loss with a KL divergence similarity loss weighted 0.004 with a learning rate of 3e-4 to achieve a loss of 30.

| Architecture and Training Procedure | Test Acc | Sub 1 TA | Sub 2 TA | Sub 3 TA | Sub 4 TA | Sub 5 TA | Sub 6 TA | Sub 7 TA | Sub 8 TA | Sub 9 TA |
|---|---|---|---|---|---|---|---|---|---|---|
| CNN, No Aug | 54.2 | 46.0 | 42.0 | 52.0 | 64.0 | 61.7 | 55.1 | 64.0 | 48.0 | 55.3 |
| CNN, All subs | 69.5 | 62.3 | 55.4 | 72.3 | **71.1** | **80.5** | 63.6 | **74.3** | 62.6 | 76.9 |
| CNN Opt, All subs | **70.4** | **65.0** | **56.0** | **72.5** | 65.0 | 79.3 | 63.8 | 71.5 | **65.5** | **78.8** |
| CNN+LSTM, All subs | 55.6 | 42.0 | 40.3 | 62.0 | 59.7 | 70.5 | 52.2 | 59.1 | 46.6 | 69.0 |
| CNN+MHA, All subs | 62.3 | 56.0 | 49.1 | 63.4 | 58.9 | 75.7 | **66.5** | 65.4 | 59.4 | 67.8 |
| CNN, Sub 1 | 29.7 | **36.3** | 24.6 | 23.7 | **35.1** | 23.4 | 24.5 | 32.3 | 30.3 | 35.0 |
| CNN+LSTM, Sub 1 | 21.7 | 28.0 | 20.0 | 18.0 | 28.0 | 17.0 | 20.4 | 16.0 | 22.0 | 25.5 |
| CNN+MHA, Sub 1 | **33.0** | 24.3 | **29.1** | **34.6** | 34.0 | **24.3** | **34.1** | **32.9** | **43.1** | **41.6** |

Table 1. Test accuracy across various model architectures. Unless specified otherwise, each model was trained on augmented data. The test accuracy for each subject is also included. Best models for the one subject and all subject training procedures are highlighted in bold.

| Time Window | 64 (32) | 128 (64) | 192 (96) | 256 (128) | 320 (160) | 384 (192) | 448 (224) | 512 (256) | 768 (384) | 1000 (500) |
|---|---|---|---|---|---|---|---|---|---|---|
| Test Accuracy | 31.3 | 34.2 | 29.2 | 30.8 | 64.3 | 65.9 | 65.8 | 64.5 | **68.5** | 33.4 |

Table 2. Test accuracy across various time windows. The base CNN model was used. The numbers in parentheses reflect the effective time window, as our data was subsampled at a factor of 0.5. The best test accuracy is highlighted in bold.

| % Synth Used | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
|---|---|---|---|---|---|---|---|---|---|---|
| Test Accuracy | 61.9 | 64.6 | 65.9 | 65.1 | 66.3 | 66.9 | 66.8 | 65.4 | 66.1 | 66.4 |

Table 3. Test accuracy vs. the percent of all synthetic data (out of 8192 trials) used in the train and validation datasets. The CNN model with data augmentations was used. Although the test accuracies are lower than the base CNN, there is a slight upward trend when adding more synthetic data.