# Enhancing CNNs and LSTMs with Attention for Time Series Classification

Alexander Swartz

May 13th, 2025

Link to code: `https://github.com/AnderSwartz/NNDL-Final-Project/tree/main`

## 1 Introduction

Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs) are fundamental building blocks for ML models of time series datasets. When comparing different models for time series data, the UCR time series archive is often used to provide a variety of time series datasets for comparison. Therefore, to compare the utility of RNNs vs CNNs for time series classification, a separate RNN and CNN can be trained and compared on each of the UCR datasets. While the UCR archive is frequently used to evaluate time series classification models, most experiments emphasize traditional ML techniques that rely on feature-based approaches, rather than deep learning [1,2]. The experiments that do utilize deep learning techniques focus on advanced architectures such as Multi-scale CNN and Time Le-Net, as opposed to basic architecture of "vanilla" CNNs or LSTMs [2]. Therefore, the current experiment elucidates the differences between these two basic architectures for time series classification tasks. Additionally, each of these architectures are augmented with attention mechanisms to determine how they affect each basic architecture. This experiment also explores whether it is appropriate to add attention mechanisms before or after the typical feature extraction done with LSTMs or CNNs. Lastly, a similar analysis across the UCR datasets will consider if attention mechanisms were especially beneficial for certain dataset types.

## 2 Related Work

RNNs maintain a hidden state that evolves over time, which makes them well-suited for time series data with temporal dependence between time steps. In "Deep learning for time series classification: a review", Fawaz et al [3] explain that apart from time series forecasting, RNNs are rarely applied for time series classification. This is due to the vanishing gradient problem, which quickly

becomes problematic with long sequences. RNNs are also predominantly used for forecasting because they are designed to predict an output at each time step. Though LSTMs attempt to mitigate this [4], their struggle to capture long-term dependencies is partially what gave rise to attention. For this reason, I expect the LSTM to perform worse on datasets with longer sequence lengths but significantly benefit from attention mechanisms.

Conversely, CNNs are well-suited for time series data because of their ability to detect local patterns and exhibit translation-invariant pattern recognition. "The result of a convolution (one filter) on an input time series X can be considered as another univariate time series C that underwent a filtering process. Thus, applying several filters on a time series will result in a multivariate time series whose dimensions are equal to the number of filters used" [3]. Additionally, the shared weights of each filter ensure the features can be detected in any translation. Though these qualities make CNNs a good candidate for time series classification, their receptive field – usually small in comparison to the overall time series – limits their ability to capture long-term dependencies. Consequently, I expect CNNs to also benefit from attention mechanisms.

In their "Encoder" model, Fawaz et al. added attention to the feature maps of a CNN [3]. They feed the feature maps to a modified Bahdanau et al. attention mechanism [5]. Half of the filters are used as attention weights, while the other half are used as the attention data. A softmax is computed over the attention weights, which are multiplied by the attention data. Since the feature maps contain information about localized features, applying attention within each feature map allows information from different time steps to influence each other. Similar to sentiment classification, time series classification with attention typically only includes an encoder [3]. Similar to a vanilla transformer, models only using masked self-attention have been successful with some time series tasks [6].

Alternatively, the attention model for time series classification introduced in "Simply Attend and Diagnose" consists of multi-head attention and feed-forward layers, taking inspiration from the vanilla transformer. Instead of relying on earlier convolutional layers to create features that are attended to, this model applies attention directly to the input, suggesting that attention mechanisms can be useful both before and after convolutional representations.

For applying attention mechanisms to LSTMs, Karim et al. add self-attention to the hidden unit states at each time step [6]. Unlike a typical LSTM that just uses the hidden unit state of the final LSTM layer at the final time step, this allows the hidden unit activations earlier in the sequence to influence predictions. Attention is used to produce a context vector that weights the hidden unit representations at different time steps. The learned parameters are the attention weight matrices. They found that this attention layer did not improve performance, but aided in interpretability by enabling the visualization of attention maps.

# 3 Experiments

## 3.1 Vanilla CNN and LSTM

Before adding any attention mechanism, two basic CNN and LSTM models are trained and evaluated on the UCR dataset. The following algorithm box illustrates the basic architecture of the models:

**Algorithm 1** Vanilla CNN Training

**for** Convolutional Blocks **do**
  Apply 1D convolution with fixed number of filters, variable kernel size
  Apply ReLU activation
  Apply 1D max pooling
**end for**
Flatten all feature maps to 1D
Apply final MLP layer
Softmax over classes

**Algorithm 2** Vanilla LSTM Training

Apply LSTM blocks
Flatten over hidden units at all time steps
Apply final MLP Layer
Softmax over classes

The biggest challenge – both with the vanilla CNN vs LSTM and with attention mechanisms – is creating a fair comparison between models. For example, a CNN with 3 convolutional blocks and a large, fully-connected layer should not be compared to an LSTM with a single LSTM layer and a small, fully-connected layer. Therefore, both basic models have 2 layers followed by a fully-connected layer with the same number of units, 32. Although the number of filters in a CNN is more important to the overall capacity of the model than the number of hidden units in an LSTM, it is unclear how to quantify this difference. Consequently, 32 was chosen for both. The length of the 1D kernel in the convolutional layers was chosen uniquely for each dataset according to the sequence length. The first convolutional layer had a kernel size of sequence length / 5, and the second sequence length / 10. This ensures an opportunity to learn both longer and shorter features.

The training loop was the same for every model. All 117 datasets have already been standardized and split into training and test sets [1]. To manage class imbalance, random minority oversampling was applied to the training set. Though LSTMs can more natively handle variable-length sequences, the CNN would a require modification such as using global average pooling instead of a "flatten" layer following the convolutional layers. Since this would break the symmetry and reduce the fairness of comparing these models, the 5 datasets with variable lengths were discarded, leaving 117. For scoring metrics, balanced accuracy is recommended for comparison over all the datasets due to the class

imbalance [1]. Additionally, macro F1 score was used to provide an alternative metric that incorporates precision and recall.

Over all 177 datasets, the vanilla CNN and LSTM performed extremely similarly. Their mean test accuracies were .673 and .685, while their average F1 scores were .645 and .666. Mean test accuracies were also grouped by sequence length, dataset size (number of training samples), and number of classes. The results show CNNs and LSTMs performing similarly across all these groups.

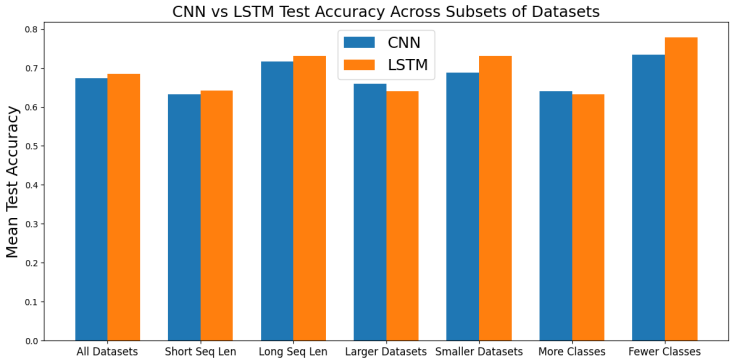


Figure 1: Comparison of Mean Test Accuracy of Vanilla CNN and LSTM model on over all UCR univariate datasets. Datasets were also split using median sequence length, number of samples (Larger vs Smaller Datasets), and number of classes to illustrate how each model performed on that subsets. Results show CNN and LSTM performed extremely similarly on all subsets of the datasets.

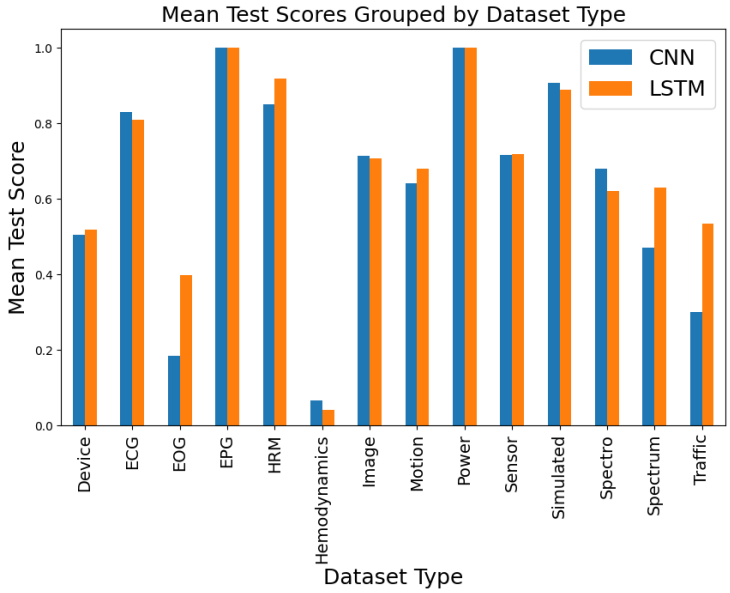


Figure 2: Comparison of Mean Test Accuracy of Vanilla CNN and LSTM model according to the type of dataset. Dataset types are categorized by the domain they were recorded from.

Results were also grouped by Dataset Type. The LSTM shows significant improvement over the CNN in EOG datasets (.398 vs .185) and Traffic datasets (.535 vs .300). Only 2 datasets are included in each of these categories, suggesting these differences could be from the high variance.

## 3.2 Adding Attention Mechanism

For the CNN, 2 new models were created that add self-attention either before or after the convolutional layers. 2 analogous models were also created that add attention to the LSTM. The implementation used Keras MultiHeadAttention layers, which implement the scaled dot product attention from "Attention is all you Need" [7]. Similar to that transformer, this attention layer is followed by layer normalization and a residual connection.

While the CNN did not need any modifications besides the addition of the attention layers, the LSTM required returning the hidden states from all time steps of the last LSTM layer for when attention was added after the LSTM layers, allowing attention to apply across the time domain. This modification increased the number of trainable parameters of the LSTM model with Attention after the LSTM layers. To ensure this increase in model capacity was not confounding the attention, the vanilla LSTM was also made to return hidden states from all time steps. Finally, flattening all hidden states is necessary before the final feed-forward layer, similar after the convolutional layers in the CNNs.

**Algorithm 3** Attention Before Convolutional Layers

Project time series to Q,K,V using MLP
Dot Production Attention (self-attention)
Layer normalization
Residual connection to input time series
Vanilla CNN architecture

**Algorithm 4** Attention After Convolutional Layers

Vanilla CNN architecture
Project **feature maps** to Q,K,V using MLP
Dot Production Attention (self-attention)
Layer normalization
Residual connection to feature maps
Apply final MLP layer
Softmax over classes

**Algorithm 5** Attention Before LSTM Layers

Project time series to Q,K,V using MLP
Dot Production Attention (self-attention)
Layer normalization
Residual connection to input time series
Vanilla LSTM architecture

**Algorithm 6** Attention After LSTM Layers

Vanilla LSTM architecture
Project **LSTM hidden units across all time steps** to Q,K,V using MLP
Dot Product Attention (self-attention)
Layer normalization
Residual connection to LSTM hidden units across all time steps
Apply final MLP layer
Softmax over classes

| | CNN | CNN Attention Before | CNN Attention After | LSTM | LSTM Attention Before | LSTM Attention After |
|---|---|---|---|---|---|---|
| Params for Models on 150 Sequence Length Dataset | 54,370 | 55,269 | 71,234 | 166,370 | 167,269 | 183,234 |
| Params for Models on 2000 Sequence Length Dataset | 534,242 | 534,242 | 551,106 | 2,060,770 | 2,061,669 | 2,077,634 |

Figure 3: Trainable parameters for each model with different sequence lengths

The total trainable parameters are shown in Figure 3 for two datasets with different sequence lengths. The different CNNs all have relatively similar numbers of parameters, especially for longer sequence lengths. This pattern is also demonstrated in the LSTMs. This helps mitigate potential differences in model performance due to a model having significantly more parameters and a strictly greater capacity.

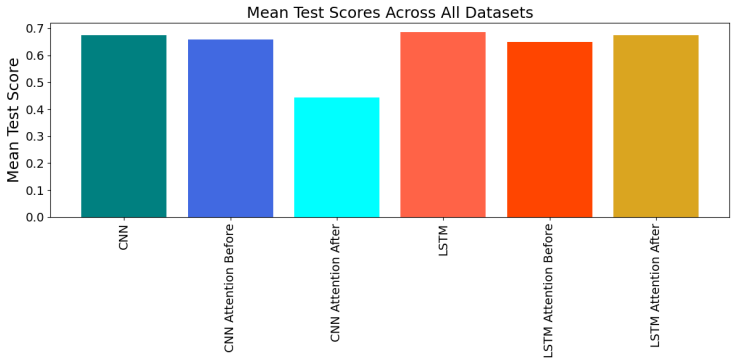


Figure 4: Mean Test accuracies across all 6 models.

The attention mechanisms either had little effect or worsened the mean test accuracy across all datasets. The CNN with Attention before the convolutional layers lowered the accuracy from .674 to .659, while adding attention after the convolutional layers lowered it to .444. The LSTM with Attention before the LSTM layers lowered the accuracy from .685 to .650, while adding attention after the LSTM layers lowered it to .656.

Several experimental errors could have affected these results. Firstly, a lack of positional encoding meant that the attention layers, particularly those at the beginning of models, did not explicitly receive temporal information. Though sinusoidal positional encoding was added to the models and trained for 5 datasets, not enough computational resources and time were available to fully evaluate its utility. Future experiments could evaluate time-series specific encoding techniques such as Time-aware Positional Encoding (TUPE) that have proven effectiveness [8]. These constraints also impacted the feasibility of hyperparameter tuning, which could have reduced the impact of overfitting by allowing for simpler models for datasets with fewer samples and less complexity. The relatively few samples in each UCR dataset enabled quick training, but also diminished the ability for deep learning methods to leverage their ability to learn complex features at multiple levels of abstraction.

The lack of improvement from the attention mechanisms could also be related to the disproportionately larger number of trainable parameters that belonged to the final fully connected layer. Though this is normal for CNNs, the addition of the hidden unit outputs for all time steps of the LSTM drastically increased the number of parameters in the LSTM models as a result of the large final fully connected layer. Global average pooling was briefly evaluated as an alternative to the "flatten" layer. Both removed the temporal dimension of the feature maps and LSTM hidden units before the final fully connected layer, but global average pooling more effectively reduces the total units connected to the final layer. Brief experiments showed significant reductions in test accuracy, suggesting an alternative besides global average pooling or flattening could be effective. Dropout was also introduced within the attention layers and following the final fully-connected layer to prevent overfitting, but did not have significant impact.

# 4 References


[1] Dau, H., Keogh, E., Kamgar, K., Yeh, C.C., Zhu, Y., Gharghabi, S., Ratanamahatana, C., Hu, B., others (2019). The UCR Time Series Classification Archive. arXiv preprint arXiv:1810.07758. `https://www.cs.ucr.edu/~eamonn/time_series_data_2018/`

[2] Bagnall, A., Lines, J., Bostrom, A., Large, J., Keogh, E. (2017). The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. Data mining and knowledge discovery, 31, 606–660.

[3] Fawaz, H., Forestier, G., Weber, J., Idoumghar, L., Muller, P. (2018). Deep learning for time series classification: a review. arXiv e-prints, art. arXiv preprint arXiv:1809.04356.

[4] Mohammadi Foumani, N., Miller, L., Tan, C., Webb, G., Forestier, G., Salehi, M. (2024). Deep learning for time series classification and extrinsic regression: A current survey. ACM Computing Surveys, 56(9), 1–45.

[4] Karim, F., Majumdar, S., Darabi, H., Chen, S. (2017). LSTM fully convolutional networks for time series classification. IEEE access, 6, 1662–1669.

[5] Song, H., Rajan, D., Thiagarajan, J., Spanias, A. (2018). Attend and diagnose: Clinical time series analysis using attention models. In Proceedings of the AAAI conference on artificial intelligence. [6] Bahdanau, D., Cho, K., Bengio, Y. (2015). Neural Machine Translation by Jointly Learning to Align and Translate. In 3rd International Conference on Learning Representations (ICLR).

[7] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, ., Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.

[8] Irani, H., Metsis, V. (2025). Positional Encoding in Transformer-Based Time Series Models: A Survey. arXiv preprint arXiv:2502.12370.