Sentiment Classification with Neural Networks

Saisrijith Reddy Maramreddy

December 2025

1 Background on Sentiment Analysis and Neural Text Classification

Sentiment analysis aims to interpret opinions and emotional tone in text, a task made difficult on Twitter due to informal and highly variable language. CNNs address these challenges by learning embeddings and hierarchical n-gram features directly from raw text, enabling them to capture sentiment cues such as negation and contrast. Their ability to scale and model local context makes them well suited for large sentiment datasets like Sentiment140.

1.1 Why Model Architecture and Hyperparameters Matter

A CNN's performance is shaped by its architecture—kernel sizes, filter counts, and depth determine how much linguistic structure the model can capture. Training hyperparameters, especially learning rate, strongly influence convergence stability and generalization. By varying these factors, this project evaluates how model capacity and optimization choices affect accuracy and efficiency in large-scale sentiment classification.

2 Data Description and Objective

The Sentiment140 dataset, consisting of 1.6 million tweets labeled as either positive or negative, provides a realistic and challenging benchmark for neural text classification. Tweets contain informal grammar, emojis, abbreviations, and noise—mirroring real-world sentiment expression. To ensure balanced learning, the dataset was equalized across classes and split into 80% training, 10% validation, and 10% testing.

2.1 Objective

This project investigates how architectural complexity and training hyperparameters independently influence CNN performance on the Sentiment140 dataset. In the architecture experiment, we modify the baseline CNN by adding one additional convolutional layer to assess whether increased depth improves accuracy. In a separate hyperparameter study, conducted on a different enhanced CNN variant, we evaluate the effects of learning rate, optimizer choice, kernel size, embedding dimension, and hidden-layer depth on convergence, training time, and overall performance.

3 Model Architectures

3.1 Architecture Experiment

3.1.1 Baseline CNN

The baseline CNN architecture serves as the starting point for the architecture experiment. It consists of:

- an embedding layer that maps each token to a 100-dimensional vector,
- a single 1D convolutional layer with 128 filters and a kernel size of 5 to capture local n-gram features,
- a global max pooling layer that extracts the strongest activation across the sequence,
- a fully connected output layer with sigmoid activation for binary sentiment prediction.

This model provides a lightweight benchmark for evaluating how deeper convolutional structures influence performance.

3.1.2 Modified CNN

The modified architecture expands the model's capacity by deepening the convolutional stack and widening its receptive field. Specifically:

- a second Conv1D layer with 256 filters is added on top of the baseline's 128-filter layer, enabling hierarchical feature extraction—the first layer captures short n-grams, while the second composes them into higher-level sentiment patterns,
- a larger kernel size (7 instead of 5) increases the receptive field, allowing the model to capture longer phrases and broader contextual cues,
- the increased filter count provides richer feature diversity for handling informal sentiment markers common in tweets.

These changes allow the model to detect sentiment spread across multi-word expressions or contrastive statements. However, the added depth and wider filters increase parameter count and training cost, providing a clear opportunity to evaluate whether this additional complexity leads to meaningful performance improvements.

3.2 Hyperparameter Experiment

3.2.1 Baseline CNN

For the hyperparameter experiment, the Baseline CNN again serves as a reference model. It consists of:

- an embedding layer that maps tokens into a 100-dimensional vector space,
- a single Conv1D layer with 128 filters and kernel size 5 to extract short-range patterns,
- a global max pooling layer to retain the most salient activation across the sequence,
- a fully connected sigmoid output layer for binary sentiment prediction.

3.2.2 Modified CNN

For the hyperparameter experiment, the Modified CNN extends the baseline architecture by increasing its depth with two additional convolutional layers. This design provides a more expressive model on which the effects of different training hyperparameters can be meaningfully observed. The architecture consists of:

- the same embedding layer and initial Conv1D layer as the baseline CNN,
- two additional Conv1D layers, creating a three-layer convolutional stack capable of hierarchical feature extraction,
- a global max pooling layer applied after the convolutional stack,
- a fully connected sigmoid output layer for binary sentiment prediction.

This deeper architecture is used as the fixed foundation for all hyperparameter variations, ensuring that differences in performance arise from the training configurations rather than structural changes to the model.

4 Training Setup

To ensure consistency and comparability, both experiments follow structured training procedures. The specific hyperparameter settings for each experiment are listed below.

Architecture Experiment

- Optimizer: SGD (batch size = 32)
- Learning rates tested: {0.1, 0.01, 0.001}
- Training schedule: maximum 20 epochs with early stopping (patience = 3)
- Recorded metrics: training/validation loss, validation error rate, epoch time
- Evaluation metrics: accuracy, false positive rate (FPR), false negative rate (FNR), total error rate

Hyperparameter Experiment

- Learning rates tested: {0.01, 0.1}
- Optimizers: SGD and Adam
- Hidden-layer settings: baseline vs. two added hidden layers (three total)
- Kernel sizes tested: 5 and 3
- Item length: 50 and 250
- Embedding dimensions: 100 and 500
- Training schedule: maximum 20 epochs with early stopping (patience = 3)

Across both experiments, learning rate had a pronounced effect on convergence: a very small rate (0.001) yielded slow but stable learning, a moderate rate (0.01) converged more quickly but showed mild sensitivity to validation fluctuations, and the highest rate (0.1) produced the fastest improvement and generally achieved the strongest validation and test accuracy without instability.

5 Results

5.1 Architecture Experiment: Performance Comparison

Table 1: Accuracy, parameter count, training-time characteristics, and architecture descriptions for both CNN models across learning rates.

Model	Params	Architecture	LR	Train (%)	Val (%)	Test (%)	Epochs	Total Time (s)
Baseline CNN	1,064,357	Embedding layer (100d); Conv1D with 128 filters (kernel size 5)	0.001	79.67	76.89	76.81	18	3548
Modified CNN	1,507,749	Baseline extended with second Conv1D (256 filters) and larger kernel size (7) for hierarchical feature extraction and longer-range context.	0.001	81.68	76.22	76.19	16	3613
Baseline CNN	1,064,357	Embedding layer (100d); Conv1D with 128 filters (kernel size 5)	0.010	80.97	77.55	77.62	9	1656
Modified CNN	1,507,749	Baseline extended with second Conv1D (256 filters) and larger kernel size (7)	0.010	85.22	77.00	77.14	7	1535
Baseline CNN	1,064,357	Embedding layer (100d); Conv1D(128, k=5)	0.100	81.80	78.09	**78.14**	14	2281
Modified CNN	1,507,749	Baseline extended with second Conv1D (256 filters) and larger kernel size (7)	0.100	83.25	77.60	77.59	5	1070

The modified CNN contains roughly 40% more parameters due to the added convolutional layers and expanded filter bank, resulting in higher computational cost per epoch. Interestingly, the deeper model tended to overfit earlier, triggering early stopping sooner and reducing total training time despite slower epochs. Across all learning rates, the simpler baseline CNN achieved higher validation and test accuracy, indicating that the added depth and capacity did not yield measurable performance benefits for this task.

Table 2: Hyperparameter experiment results on the Modified CNN.

Setting	Value	Final Architecture	Params	Train (%)	Val (%)	Test (%)	Epochs	Total Time (s)
Learning Rate	0.01	Embedding(100d); Conv1D×3 (128f, k=5); FC Hidden Layers=1; SGD (LR=0.01)	1,228,453	83.26	78.55	78.69	9	1707
Learning Rate	0.10	Embedding(100d); Conv1D×3 (128f, k=5); FC Hidden Layers=1; SGD (LR=0.10)	1,228,453	83.64	79.05	79.07	9	1873
Optimizer	SGD	Embedding(100d); Conv1D×3 (128f, k=5); FC Hidden Layers=1; SGD (LR=0.01)	1,228,453	83.31	78.23	78.58	10	2185
Optimizer	Adam	Embedding(100d); Conv1D×3 (128f, k=5); FC Hidden Layers=1; Adam (LR=0.01)	1,228,453	49.99	50.09	49.98	8	1923
Hidden Layers (Fully Connected)	1 Layer	Embedding(100d); Conv1D×3 (128f, k=5); FC Hidden Layers=1; SGD (LR=0.01)	1,228,453	83.58	78.52	78.55	10	2134
Hidden Layers (Fully Connected)	3 Layers	Embedding(100d); Conv1D×3 (128f, k=5); FC Hidden Layers=3; SGD (LR=0.01)	1,261,477	83.92	78.62	78.63	10	2560
Input Length	50	Embedding(100d); Conv1D×3 (128f, k=5); FC Hidden Layers=1; SGD (LR=0.01)	1,228,453	83.32	78.70	78.73	9	1848
Input Length	250	Embedding(100d); Conv1D×3 (128f, k=5); FC Hidden Layers=1; SGD (LR=0.01)	1,228,453	81.04	77.36	77.47	9	1848
Kernel Size	3	Embedding(100d); Conv1D×3 (128f, k=3); FC Hidden Layers=1; SGD (LR=0.01)	1,137,317	82.76	79.23	79.12	10	2310
Kernel Size	5	Embedding(100d); Conv1D×3 (128f, k=5); FC Hidden Layers=1; SGD (LR=0.01)	1,228,453	83.62	78.59	78.57	10	2340
Embedding Dim	100	Embedding(100d); Conv1D×3 (128f, k=5); FC Hidden Layers=1; SGD (LR=0.01)	1,228,453	84.01	78.42	78.57	11	2596
Embedding Dim	500	Embedding(500d); Conv1D×3 (128f, k=5); FC Hidden Layers=1; SGD (LR=0.01)	5,484,853	86.27	79.46	**79.54**	8	1896

The numerical results reveal several consistent patterns across learning rates, model sizes, and training configurations. A higher learning rate (0.10) produced the fastest convergence, with both trials stopping at epoch 9 and achieving the strongest validation and test accuracy (79.05% and 79.07%). In contrast, lower rates slowed optimization noticeably, and the smaller value (0.01) reduced test accuracy by roughly 0.4 percentage points despite identical epoch counts. Increasing model size offered little benefit: neither the deeper convolutional model from the architecture experiment nor the expanded classifier head with additional dense layers in the hyperparameter study yielded meaningful improvements. Adding two extra fully connected layers increased per-epoch training time (256s vs. 213s) but improved test accuracy by only 0.08 percentage points (78.63% vs. 78.55%), showing that the additional parameters were not justified by the marginal gain. A similar trend appeared with input length: shorter sequences (50 tokens) outperformed longer ones (250 tokens), improving test accuracy from 77.47% to 78.73%, suggesting that longer inputs introduce noise or padding that slows and destabilizes training. Kernel size also played a clear role—smaller kernels (3 or 5) generalized better, with the smallest kernel achieving 79.12% test accuracy. Embedding dimension was the only modification that consistently improved performance: 500-dimensional embeddings achieved the highest accuracy overall (79.54%) and converged faster (epoch 8), though at a substantial increase in parameter count (5.48M vs. 1.23M).

Optimizer behavior produced the most dramatic contrast: SGD remained stable across all settings, whereas Adam collapsed to chance-level accuracy (50%) because, at the tested learning rates, its adaptive updates caused divergence. A smaller learning rate (e.g., 0.001) would likely provide a fairer comparison and should be explored in future work. Overall, the most efficient balance between accuracy and computational cost was achieved not through added architectural depth but through well-chosen hyperparameters—namely a higher learning rate, smaller kernel size, shorter input length, and larger embedding dimension.

Discussion

Across the architecture experiments, the baseline CNN consistently outperformed the deeper modified model at all learning rates, indicating that the added convolutional capacity did not improve generalization. In the hyperparameter experiments on the modified CNN, performance was driven primarily by learning rate and embedding dimension, with 500-dimensional embeddings achieving the best test accuracy (79.54%). Smaller kernels and shorter input sequences also improved results, while increasing the number of fully connected layers yielded only marginal gains despite higher training cost. Overall, well-chosen hyperparameters contributed more to performance than additional architectural depth for this sentiment classification task.

Appendix

All source code, model training, and analysis notebooks used in this project are available at:
`Sentiment Classification with Neural Networks(Architecture Experiments)`
`Sentiment Classification with Neural Networks(Hyperparameter Experiments)`