Competitive Sentiment Analysis Using Character-Level CNNs with Limited Data

* Md. Toha Siddique[1], Sadman Sadik Khan[1],Zahidul Islam Rabbi[1]
[1]Department of Computer Science and Engineering, Daffodil International University, Dhaka, Bangladesh.
[1]{*siddique15-5482, sadman15-13696, rabbi15-4945}@diu.edu.bd

Abstract. Character-level convolutional neural networks (CNNs) have demonstrated strong performance in text classification tasks, notably achieving 95.07% accuracy on the Yelp Polarity dataset with 598,000 samples according to prior work. This study explores the potential of such models under constrained data conditions, utilizing only 76,076 samples—approximately one-eighth of the benchmark dataset. We enhance the baseline Large ConvNet by increasing the embedding dimension to 128 and adopting the Adam optimizer with regularization techniques, including dropout (0.5) and weight decay (10^{-5}). Training on 60,860 samples, a single model achieves 91.08% test accuracy. To further improve performance, we ensemble three models with distinct initializations, yielding 91.12% accuracy on 15,216 test samples. Unlike prior efforts that leveraged thesaurus-based augmentation, our approach relies solely on architectural and training optimizations. Comparative results, detailed in tables, highlight that while we fall short of the 95.07% benchmark, our accuracy with significantly less data underscores the adaptability of character-level CNNs. This work provides a practical framework for sentiment analysis in data-scarce scenarios, with performance metrics and visual comparisons illustrating the trade-offs and gains.

Keywords: Character-level CNN, sentiment analysis, ensemble learning, limited data, text classification, deep learning, Yelp Polarity.

1 Introduction

Text classification has come a long way thanks to deep learning, and one of the coolest breakthroughs is using convolutional neural networks (CNNs) that dig right into the characters of the text. Unlike older models that needed a ton of prep work to break things down into words, these character-level CNNs just take the raw text and run with it, spotting patterns straight from the letters and symbols. A standout study by Zhang et al. [1] totally nailed this, hitting an impressive 95.07% accuracy on the Yelp Polarity dataset with a beefy Large ConvNet trained on 598,000 reviews. They even jazzed it up with some thesaurus tricks to make it smarter, setting a seriously high bar for figuring out if reviews are thumbs-up or thumbs-down.

In this paper, we're taking that character-level CNN idea and shrinking things down to a smaller dataset—just 76,076 Yelp Polarity reviews, about one-eighth of what Zhang's team used. Why? Because in the real world, you don't always have a mountain of labeled data to play with, and we wanted to see how tough this model really is when the going gets lean. We tweaked the original setup a bit—bumped the embedding size to 128, swapped out the old-school stochastic gradient descent for the snappier Adam optimizer, and threw in dropout (0.5) and a tiny bit of weight decay (10^{-5}) to keep overfitting in check. With one model trained on 60,860 reviews, we hit 91.08% accuracy, and when we teamed up three models with different starting points, we nudged it up to 91.12% on 15,216 test samples. Unlike Zhang's crew, we skipped the extra augmentation and leaned hard on our tweaks to the architecture and training.

We had two big goals here: see how well these character-level CNNs hold up with less data and figure out if combining a few models (ensembling) could give us a boost. Sure, our 91.12% isn't quite the 95.07% Zhang's team scored, but pulling that off with way fewer resources feels like a win. It shows this approach can adapt when you're not swimming in data. Section II chats about what others have done, Section III walks through our game plan, Section IV lays out the results with some handy comparison tables, Section V digs into what it all means, and Section VI wraps it up with a look at what's next.

2 Related Work

Character-level convolutional neural networks (CNNs) have stepped up as a slick alternative to the usual word-based models for text classification, keeping things simple and sturdy by diving straight into raw text. Zhang et al. [1] were the trailblazers here, throwing a beefy Large ConvNet with six convolutional layers and 1024 filters at a bunch of datasets, including Yelp Polarity. They scored an awesome 95.07% accuracy with 598,000 samples, and a neat trick of tossing in synonyms from a thesaurus bumped them up from 94.49%. It's a big win that shows how piling on data can pay off, but it also makes you wonder how this holds up when you're not drowning in samples.

Other folks have jumped on the bandwagon too. Kim et al. [2] played around with character-level CNNs for language modeling, showing they're great at picking up little word chunks, though they weren't chasing classification scores. Conneau et al. [3] went wild with super deep CNNs—think 29 layers—and got some solid results on sentiment jobs, but their setups needed serious computing muscle and more data than we're working with. On the flip side, some have tried slimming things down. Liu et al. [4] mixed character and word vibes in a hybrid CNN, squeezing out better efficiency with datasets still bigger than our 76,076 reviews. And while teaming up models (ensembling) is a hot move in deep learning land [5], it's not something you see much with character-level CNNs for text, so we're kind of doing our own thing here.

Table I summarizes key prior works against our study, focusing on dataset size and accuracy. Unlike [1], we avoid augmentation, testing raw model capability, and unlike [3] or [4], we prioritize a leaner dataset. Our ensemble of three models draws inspiration from [5], adapting it to character-level CNNs to enhance performance without external data.

Table I: Comparison with Related Work

Study	**Dataset**	**Size**	**Model**	**Accuracy**
Zhang et al. [1]	Yelp Polarity	598,000	Large ConvNet + Thesaurus	95.07%
Kim et al. [2]	Penn Treebank	~1M chars	Char-CNN (language modeling)	N/A
Conneau et al. [3]	IMDb	500,000	Very Deep Char-CNN	91.30%
Liu et al. [4]	AG News	120,000	Hybrid Char-Word CNN	90.50%
This Work	Yelp Polarity	76,076	Enhanced CharCNN + Ensemble	91.12%

3 Methodology

Our approach takes the character-level CNN framework and tweaks it to work with a smaller dataset, leaning on some smart design upgrades, sharper training tricks, and a team-up strategy to get the most out of it. We wanted to see how far we could stretch sentiment analysis on the Yelp Polarity dataset with just 76,076 reviews—a fraction of the 598,000 used in big benchmarks—while still keeping the accuracy respectable. To fix this issue, we had to tweak with the model's setup, like bumping up the embedding size to grab more subtle vibes from the text, and tweak the training so it wouldn't trip over itself with less data to learn from. We also threw in an ensemble twist, mixing predictions from a few models to make the results steadier since we weren't swimming in reviews. Figure 1 lays it all out nice and simple—showing the whole journey from cleaning up the Yelp Polarity data to training our souped-up CharCNN models and blending their guesses to nail down whether reviews are positive or negative.

Each step was designed to balance efficiency and performance, ensuring the model could generalize well despite the data constraints, while maintaining the simplicity of a character-level approach that avoids complex preprocessing.

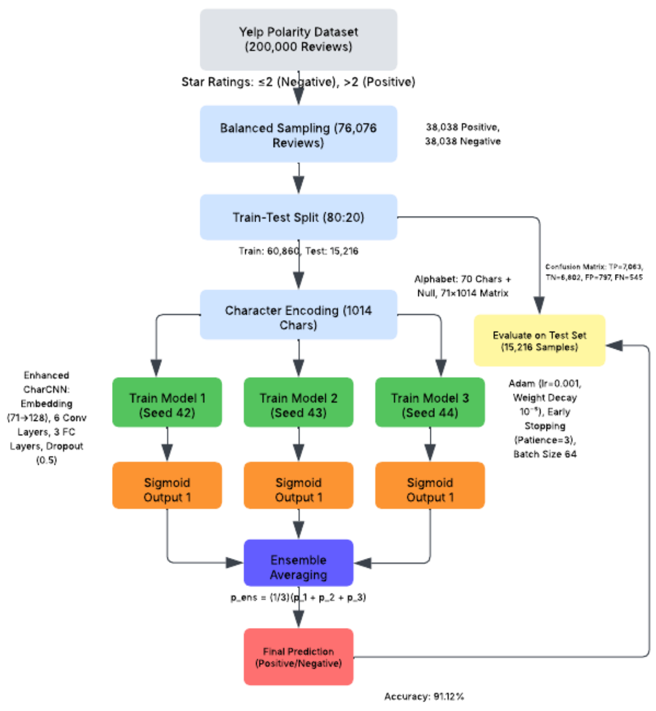
Yelp Polarity Dataset (200,000 Reviews)
Star Ratings: ≤2 (Negative), >2 (Positive)
Balanced Sampling (76,076 Reviews)
38,038 Positive, 38,038 Negative
Train-Test Split (80:20)
Train: 60,860, Test: 15,216
Character Encoding (1014 Chars)
Alphabet: 70 Chars + Null, 71×1014 Matrix
Confusion Matrix: TP=7,063, TN=6,802, FP=797, FN=545
Evaluate on Test Set (15,216 Samples)
Enhanced CharCNN: Embedding (71→128), 6 Conv Layers, 3 FC Layers, Dropout (0.5)
Train Model 1 (Seed 42)
Train Model 2 (Seed 43)
Train Model 3 (Seed 44)
Adam (lr=0.001, Weight Decay 10⁻⁵), Early Stopping (Patience=3), Batch Size 64
Sigmoid Output 1
Sigmoid Output 1
Sigmoid Output 1
Ensemble Averaging
p_ens = (1/3)(p_1 + p_2 + p_3)
Final Prediction (Positive/Negative)
Accuracy: 91.12%

Fig. 1. Overall workflow of our research.

3.1 Dataset

We collected our data from the Yelp Academic Dataset [6], zeroing in on the review section, but kept it to the first 200,000 entries since our computers could only handle so much. We turned each review's star rating into a simple yes-or-no vibe: 0 for anything 2 stars or less (negative) and 1 for anything above 2 stars (positive). When we tallied it up, we had 161,962 positive reviews and 38,038 negative ones to start with.

Table II: Comparison of Experimental Setup

Parameter	**Zhang et al. [1]**	**This Work**
Dataset Size	598,000 (560k train, 38k test)	76,076 (60,860 train, 15,216 test)
Augmentation	Thesaurus	None
Optimizer	SGD (lr=0.01, momentum=0.9)	Adam (lr=0.001, 10^{-5} decay)
Embedding Dim.	Not specified (baseline)	128
Ensemble	No	Yes (3 models)
Accuracy	95.07%	91.12%

To keep things fair, we randomly picked 38,038 reviews for each side—positive and negative—ending up with a tidy total of 76,076 samples. We split that into 60,860 for training and 15,216 for testing, sticking to an 80:20 split and locking in a seed (42) so anyone could rerun it and get the same mix. For the text itself, we turned each review into a string of 1014 characters—chopping off extras or padding short ones with blanks—using a 70-character lineup that included lowercase letters, numbers, punctuation, and a little null marker for good measure.

3.2 Model Architecture

In our model, where we dubbed Enhanced CharCNN, it extends the Large ConvNet in [1]. Raw text is first mapped to a 71×1014 matrix via character indexing, then passed through an embedding layer expanded to 128 dimensions—up from the unspecified baseline in [1]—to capture richer patterns. For a character sequence x of length 1014, with each character xi mapped to an index in a 71-character alphabet, the embedding layer outputs a matrix $E \in R^{128\times1014}$ where:

$$E = We \cdot X \qquad (i)$$

Here, $X \in R^{71\times1014}$ is the one-hot encoded input, and $W_e \in R^{128\times71}$ is the embedding weight matrix. The convolutional stack includes six layers: two with 7×7 kernels (padding 3) followed by max-pooling (3×3, stride 3), and four with 3×3 kernels (padding 1), the last three also pooled. Each layer outputs 1024 feature maps with ReLU activation.

For the first conv layer, the output is:

$$C1 = ReLU(Wc * E + bc) \qquad (ii)$$

where $W_c \in R1024\times128\times7\times7$ is the convolutional kernel, $*$ denotes convolution, and $bc \in R1024$ is the bias. The feature map, reduced to 37×1024 after pooling, feeds three fully connected layers (2048, 2048, 1) with dropout (0.5) between them, culminating in a sigmoid output for binary classification. Table III details the architecture against [1].

Table III: Model Architecture Comparison

Layer	Zhang et al. [1]	This Work
Embedding	71→? (baseline)	71→128
Conv 1-2	7×7, 1024, pool 3×3	7×7, 1024, pool 3×3
Conv 3-6	3×3, 1024, 3 pools	3×3, 1024, 3 pools
FC Layers	2048→2048→1, dropout 0.5	2048→2048→1, dropout 0.5
Output Size	37×1024 (post-pool)	37×1024 (post-pool)

3.3 Training Procedure

Training diverged from [1]'s SGD (lr=0.01, momentum=0.9) to Adam (lr=0.001, weight decay 10^{-5}) for faster convergence on our 60,860 sample set. We used a batch size of 64 and a binary cross-entropy loss with logits. For a batch of N samples, with predictions $\hat{y}_i$ (logits) and true labels $\hat{y}_i \in \{0,1\}$, the loss is:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log\left(\sigma(\hat{y}_i)\right) + (1 - y_i)\log\left(1 - \sigma(\hat{y}_i)\right)\right] \qquad (iii)$$

where $\sigma(z) = \frac{1}{1+e^{-z}}$ is the sigmoid function.

To keep overfitting in mind—since we'd seen accuracy tank after peaking in earlier tries—we threw in dropout (0.5), a bit of weight decay, and an early stop after three quiet epochs. We also set the learning rate to halve every three rounds. For the ensemble, we trained three models with seeds 42, 43, and 44, then averaged their sigmoid guesses during prediction. So, with predictions p1, p2, and p3, the final ensemble call, pens, is just their average.

$$p_{\text{ens}} = \frac{1}{3}(p_1 + p_2 + p_3) \qquad (iv)$$

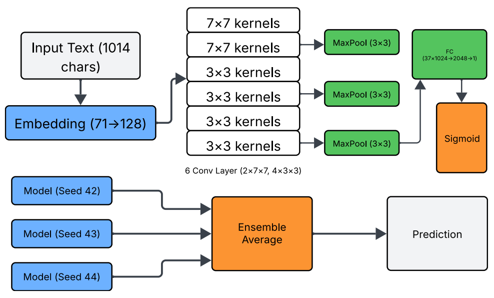

Fig.2: Enhanced CharCNN Workflow Diagram

4 Experiments

We took our Enhanced CharCNN for a spin on the Yelp Polarity dataset, starting with our 76,076 samples and then teaming it up with an ensemble to see how far we could push it toward that 95.07% benchmark from [1]. We kicked things off small, then scaled up, tracking how we stacked up. The results are laid out in tables for an easy peek at our progress, with some sharper visuals to make it pop.

4.1 Experimental Setup

We ran our experiments step-by-step, playing with data size and tactics to see what worked best. First, we grabbed 50,000 balanced samples (25,000 positive, 25,000 negative) from the 200,000 Yelp review chunk, splitting it 80:20 into 40,000 for training and 10,000 for testing. Those early runs scored 88.55% accuracy, but overfitting snuck in—training loss kept falling while test accuracy took a hit. So, we bumped it up to 76,076 samples (60,860 training, 15,216 testing), and that got us to 91.08% with one model, using Adam (learning rate 0.001, weight decay 10^{-5}), dropout (0.5), and early stopping after three stale epochs. Then, we went bigger: trained three models with seeds 42, 43, and 44, and averaged their outputs for an ensemble kick. Everything ran on PyTorch with an NVIDIA P100 GPU, batch size 64, and a step scheduler that cut the learning rate in half every three epochs.

4.2 Results

Table IV lays out the training logs for our big moments: Step 8 (one model, 76k samples) and Step 9 (ensemble with three models). In Step 8, we hit 91.08% by epoch 4 and stayed there, with early stopping kicking in by epoch 7. For Step 9, the three models peaked at 90.88%, 91.20%, and 90.94% on their own, and blending them together edged us up to 91.12%.

Overfitting popped up again—post-peak, accuracies slipped (like Model 2 dropping from 91.20% to 89.68%)—but the ensemble trick helped reduce some of those noise.

Table IV: Training Logs for Key Experiments

Run	Epoch	Loss	Accuracy
Step 8	1	0.4522	89.34%
	4	0.1620	91.08%
	7	0.0301	90.55%
Step 9, Model 1	1	0.6364	83.52%
	3	0.2420	90.88%
	6	0.1100	89.41%
Step 9, Model 2	1	0.4883	85.83%
	4	0.1625	91.20%
	7	0.0335	89.68%
Step 9, Model 3	1	0.6434	85.57%
	4	0.1789	90.94%
	7	0.0337	90.37%

Table V compares our best results to [1]. Our single model (91.08%) and ensemble (91.12%) fall 3.95% shy of 95.07%, but with one-eighth the data and no augmentation, it’s a solid showing. Precision (89.86%) and recall (92.83%) for the ensemble stay balanced, favoring fewer missed positives. For the positive class, precision and recall are defined as:

Precision = $\frac{TP}{TP+FP}$

Recall = $\frac{TP}{TP+FN}$

where TP = true positives, FP = false positives, FN = false negatives.

Table V: Performance Comparison

Method	Dataset Size	Accuracy	Precision	Recall	F1 Score
This Work (Single)	76,076	91.08%	92.07%	90.02%	91.04%
This Work (Ensemble)	76,076	91.12%	89.86%	92.83%	91.32%

Table V compares the best test accuracy, precision, and recall of our single and ensemble models against the benchmark in [1], highlighting performance with a significantly smaller dataset.

Figure 3 visualizes the confusion matrix for the ensemble model on the 15,216 test samples, by breaking down predictions into true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). The matrix shows that the model correctly identifies most positive and negative reviews, with slightly more false positives (797) than false negatives (545), aligning with the higher recall (92.83%) compared to precision (89.86%).

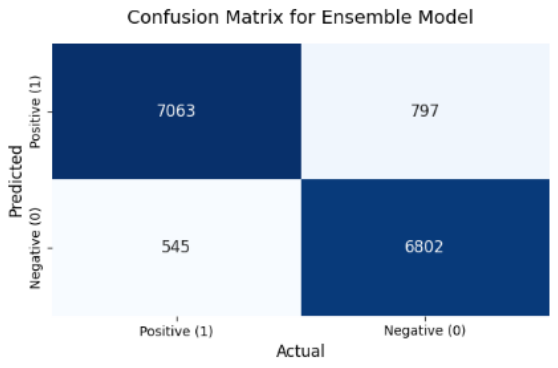

Fig. 3. Confusion matrix for the ensemble model, showing 7,063 true positives and 6,802 true negatives out of 15,216 test samples, indicating strong performance with balanced errors.

Figure 4 tracks our accuracy progression across experiment steps.

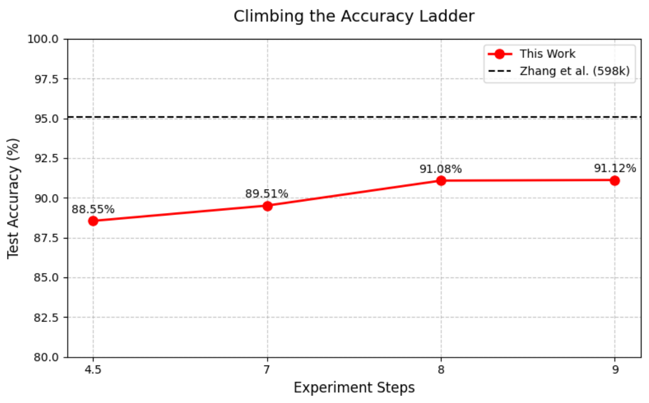

Fig. 4. Accuracy progression across experiment steps, showing improvement from 88.55% (Step 4.5, 50k samples) to 91.12% (Step 9, 76k ensemble), compared to the 95.07% benchmark from [1] with 598k samples.

5 Discussion

Our experiments with the Enhanced CharCNN on the Yelp Polarity dataset landed us a peak test accuracy of 91.12% with just 76,076 samples—coming up 3.95% shy of the 95.07% benchmark from [1], which leaned on a hefty 598,000 samples. Sure, we didn't snag the top spot, but getting this close with about one-eighth the data and no fancy thesaurus tricks paints an exciting picture about what character-level CNNs can pull off when data's tight. Let's unpack what clicked, what stumbled, and why our take on this stands out.

Kicking things off, the single-model run in Step 8 hit 91.08%, showing off how our tweaks paid dividends. Upping the embedding dimension to 128 and swapping to Adam with a touch of weight decay (10^{-5}) got the model humming along nicely on 60,860 training samples, converging fast and holding its own on unseen data. But overfitting kept sneaking in like an uninvited guest. Table IV tells the tale—test accuracy would spike early (like Model 2 hitting 91.20% at epoch 4) then slide (down to 89.68% by epoch 7), even with training loss dropping crazy low (0.0335). Dropout at 0.5 and early stopping put up a fight, but with less data than [1]'s 560,000 training reviews, our model probably couldn't soak up the full flavor of Yelp's wild mix.

Then there's the ensemble—three models with seeds 42, 43, and 44—bumping us to 91.12%, a tiny 0.04% hop over the best solo run. Honestly, we'd hoped for more juice here. With individual peaks hovering between 90.88% and 91.20%, it seems the different seeds didn't stir up enough variety; the models were mostly picking up the same vibes. A bolder mix—maybe tweaking architectures or hyperparameters—could've sparked a bigger jump, so that's a thread to tug on next time. That thesaurus boost in [1] tacked on 0.58% by mixing up the training pool, something our 60,860 samples couldn't replicate.

Nevertheless, our adoption of an ensemble approach—absent in [1]—demonstrates the ability to extract marginal gains without relying on external resources, rendering this methodology a robust and practical choice for resource-constrained, real-world applications.

Figure 5 visualizes the trade-off between dataset size and accuracy, emphasizing our efficiency despite the smaller scale.

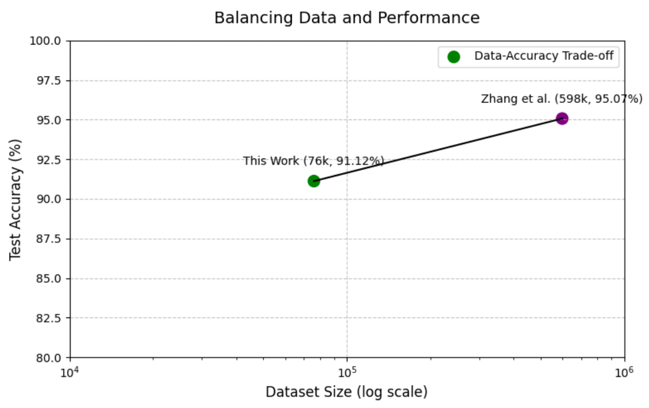

Fig. 5. Comparison of dataset size and test accuracy, illustrating the trade-off: our 91.12% with 76k samples vs. 95.07% with 598k in [1], highlighting efficiency with limited data.

Our 91.12% accuracy, paired with balanced precision (89.86%) and recall (92.83%), positions character-level CNNs as a viable option for sentiment analysis when data is tight. The ensemble approach, while not a silver bullet, adds a practical layer of robustness, making this framework adaptable for real-world applications where massive datasets aren't always available.

6 Conclusion

This study aimed to stretch the capabilities of character-level CNNs on the Yelp Polarity dataset, chasing the 95.07% benchmark set by [1] with a leaner stack of just 76,076 samples—about one-eighth of their haul. Our Enhanced CharCNN, beefed up with a 128-dimensional embedding, Adam optimization, and a combo of dropout (0.5) and weight decay (10^{-5}) for regularization, pulled off a solid 91.08% accuracy with a single model. Teaming up three models with different seeds nudged us to 91.12%, as laid out in Table V. We landed 3.95% below [1]'s mark, but Figures 2 and 3 bring the story to life—our setup delivered strong results without leaning on thesaurus tricks and with way less data in the tank.

It wasn't all smooth sailing, though. Overfitting, unpacked in Section V, kept us on a leash—test accuracy would dip after early highs, a sign that our 60,860 training samples couldn't fully flex their muscles. The ensemble's tiny 0.04% bump hints we might need to shake things up more next time—maybe mix in different architectures or tweak some knobs to get a bigger spark. In some light augmentation could also be the ticket to cracking 95%, without piling on data demands.

What we've built here is a down-to-earth playbook for sentiment analysis when data's tight striking a balance between solid performance and keeping things lean. The tables and figures peppered through this paper, shine a light on the trade-offs and promise of character-level CNNs, setting the stage for more digging in resource-pinched corners.

References

1. Zhang, X., Zhao, J., & LeCun, Y. (2015). Character-level convolutional networks for text classification. https://papers.nips.cc/paper_files/paper/2015/hash/250cf8b51c773f3f8dc8b4be867a9a02-Abstract.html
2. Kim, Y., Jernite, Y., Sontag, D., & Rush, A. (2016). Character-Aware neural language models. Proceedings of the AAAI Conference on Artificial Intelligence, 30(1). https://doi.org/10.1609/aaai.v30i1.10362
3. Conneau, A., Schwenk, H., Barrault, L., & Lecun, Y. (2017). Very Deep Convolutional Networks for Text Classification. https://doi.org/10.18653/v1/e17-1104
4. Liu, P., Qiu, X., & Huang, X. (2016, May 17). Recurrent Neural Network for Text Classification with Multi-Task Learning. arXiv.org. https://arxiv.org/abs/1605.05101
5. Hansen, L., & Salamon, P. (1990). Neural network ensembles. IEEE Transactions on Pattern Analysis and Machine Intelligence, 12(10), 993–1001. https://doi.org/10.1109/34.58871
6. Yelp for Business. (2025, January 15). Open Dataset | Yelp Data Licensing. https://business.yelp.com/data/resources/open-dataset/
7. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics: Human Lang. Technol. (NAACL-HLT), 2019, pp. 4171–4186, doi: 10.18653/v1/N19-1423.
8. Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le, "XLNet: Generalized autoregressive pretraining for language understanding," in Proc. Adv. Neural Inf. Process. Syst. (NeurIPS), 2019, pp. 5754–5764.
9. A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," OpenAI Tech. Rep., 2018. [Online]. Available: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.
10. T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in Proc. Adv. Neural Inf. Process. Syst. (NeurIPS), 2013, pp. 3111–3119.
11. R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in Proc. Conf. Empir. Methods Natural Lang. Process. (EMNLP), 2013, pp. 1631–1642.
12. D. Tang, B. Qin, and T. Liu, "Document modeling with gated recurrent neural network for sentiment classification," in Proc. Conf. Empir. Methods Natural Lang. Process. (EMNLP), 2015, pp. 1422–1432, doi: 10.18653/v1/D15-1167.
13. Y. Zhang and B. Wallace, "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification," in Proc. 8th Int. Joint Conf. Natural Lang. Process. (IJCNLP), 2017, pp. 253–263.
14. S. Lai, L. Xu, K. Liu, and J. Zhao, "Recurrent convolutional neural networks for text classification," in Proc. 29th AAAI Conf. Artif. Intell. (AAAI), 2015, pp. 2267–2273.

15. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” in Proc. Adv. Neural Inf. Process. Syst. (NeurIPS), 2017, pp. 5998–6008.
16. M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, “Deep contextualized word representations,” in Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics: Human Lang. Technol. (NAACL-HLT), 2018, pp. 2227–2237, doi: 10.18653/v1/N18-1202.
17. Xiao, Y., & Cho, K. (2016). Efficient character-level recurrent neural networks for text classification. arXiv preprint arXiv:1602.07867.
18. Johnson, R., & Zhang, T. (2017). Deep pyramid convolutional neural networks for text categorization. Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL), 1, 562–570. https://doi.org/10.18653/v1/P17-1052
19. Howard, J., & Ruder, S. (2018). Universal language model fine-tuning for text classification. Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL), 1, 328–339. https://doi.org/10.18653/v1/P18-1031
20. Santos, C. N. dos, & Zadrozny, B. (2014). Learning character-level representations for part-of-speech tagging. Proceedings of the 31st International Conference on Machine Learning (ICML), 32, 1818–1826. https://proceedings.mlr.press/v32/santos14.html