Movie Reviews Classification with Convolutional Neural Network (CNNs) and Word Embeddings

Mukhil Muruganantham Prakaash (mmm9280@psu.edu)

Github repository: https://github.com/Mukhil4703/Midterm_project_NLP

Problem Definition

- The goal of this project is to classify movie reviews into two categories positive or negative using Convolutional Neural Networks (CNNs) and word embeddings. Movie reviews are a rich source of information, and classifying their sentiment is a critical task in Natural Language Processing (NLP), with applications in areas such as customer feedback, content analysis, and social media monitoring. In this project, we leverage the IMDB dataset, which consists of 50,0 b00 labeled movie reviews which contain the viewer's sentiment and makes it a ideal dataset, to train a model that accurately predicts the sentiment of unseen reviews.

Dataset Overview and Curation

The IMDB movie review dataset, widely recognized as best movie reviews data with a large collection of 50000 labelled movie reviews as positive and negative. (two categories). This dataset has been pre-processed and organized as following

1. Training Set: 25000 labelled reviews (12500 positive and 12500 negative)
2. Test Set: 25000 labelled reviews (12500 positive and 12500 negative)

Each review is represented as a sequence of integer values corresponding to the words in the vocabulary, allowing for a simplified and standardized input for machine learning models. The dataset contains 10,000 of the most frequently occurring words in the reviews, ensuring that the model focuses on the most relevant and common terms. This is particularly important for reducing noise in the data and improving model performance

So we make steps to ensure that dataset is appropriate for training in CNN model

i. Vocabulary Selection

Only the top 10000 most frequently used words are used in the dataset. This selection helps in reducing the dimensionality of the data and focuses the model on the most impactful words.

ii. Padding and Truncating

All movie reviews are padded and truncated to fixed length of 100 words. This step ensures that all input sequences have consistent lengths which is essential for feeding the

data into the CNN model. Padding also prevents loss of info from shorter reviews, while truncation makes sure that longer reviews don't overwhelm the model.

iii. Labelling

Each is labelled as 1 (positive) or 0 (negative) this binary nature of the labes simplifes the classification and allow to focus on distinguishing between the two possible sentiment categories.

This curation process ensures the data is clean, consistent and ready for training a CNN model. The use of a well defined vocabulary, standardized input length makes it easier for the model to learn meaningful features. Which improves the models ability to classify unseen reviews.

The IMDB dataset is ideal for sentiment analysis tasks due to its size, diversity, and structure. With 50,000 labeled reviews, it offers a rich source of data for training deep learning models. The dataset's binary sentiment labels (positive and negative) make it a straightforward classification task, which is beneficial for evaluating the effectiveness of sentiment analysis models. Additionally, the dataset's usage of frequent words helps to focus on the most impactful features, which aids in improving the model's generalization capabilities

Word Embeddings

To effectively capture the semantic meaning of words, we utilized GloVe (Global Vectors for Word Representation) embeddings with 100-dimensional vectors. These pre-trained embeddings, trained on a large corpus, encode contextual meaning by mapping words with similar usage patterns to close vector spaces. Using pre-trained embeddings allows the model to leverage prior linguistic knowledge, improving performance on sentiment classification.

Each review in the dataset was converted into a sequence of integers, where each integer represents a word in the vocabulary of 10,000 most frequent words. These sequences were then padded or truncated to a fixed length of 100 words, ensuring uniform input size for all samples. The embeddings were initialized with pre-trained GloVe vectors and fine-tuned during training to adapt to the dataset-specific nuances.

Algorithms and Model Architecture

We experimented with three different CNN-based architectures for sentiment classification, progressively refining the model to improve accuracy and generalization.

Model 1:

- Embedding Layer: Pre-trained GloVe embeddings (100d), vocabulary size = 10,000
- Conv1D Layer: 128 filters, kernel size = 5, activation = ReLU

- Global Max-Pooling Layer: Extracts the most important features from convolutional outputs
- Dense Layer: 64 neurons
- Output Layer: Sigmoid activation
- Batch Size: 128
- Learning Rate: 0.001
- Epochs: 5
- Dropout: None

Model 2:

- Embedding Layer: Same as Model 1
- Conv1D Layers:
    - First layer: 256 filters, kernel size = 5, activation = ReLU
    - Second layer: 128 filters, kernel size = 3, activation = ReLU
- Global Max-Pooling Layer
- Dense Layer: 256 neurons
- Output Layer: Sigmoid activation
- Batch Size: 64
- Learning Rate: 0.001
- Epochs: 10
- Dropout: Rate = 0.5 for regularization

Model 3:

- Embedding Layer: Same as previous models
- Conv1D Layer: 128 filters, kernel size = 5, activation = ReLU
- Global Max-Pooling Layer
- Dense Layers: 128 neurons → Dropout (rate = 0.5) → 64 neurons
- Output Layer: Sigmoid activation
- Batch Size: 32

- Learning Rate: 0.0005 (Lowered to stabilize training)
- Epochs: 30
- Dropout: Rate = 0.5 to reduce overfitting

After evaluating different architectures, Model 3 was selected as the final model due to its superior performance in terms of accuracy and generalization on the test set.

Training and Optimization Details

We applied various training strategies and hyperparameter tuning techniques to improve model performance:

- Loss Function: Binary Cross-Entropy, as this is a binary classification problem (positive vs. negative sentiment).
- Optimizer: Adam, chosen for its adaptive learning rate adjustments and efficient convergence.
- Learning Rate Adjustments:
    - Initial Learning Rate = 0.001 (Models 1 & 2)
    - Final Model Learning Rate = 0.0005 (Reduced for stable convergence)
- Batch Size Variations:
    - 128 for Model 1, 64 for Model 2, and 32 for Model 3 (Progressively decreasing to improve learning stability).
- Epochs:
    - 5 (Model 1) → 10 (Model 2) → 30 (Model 3)
    - Early stopping was implemented to prevent overfitting.
- Regularization Techniques:
    - Dropout (rate = 0.5) was introduced in Models 2 & 3 to reduce overfitting.
    - Early Stopping (patience = 3 epochs) was applied to halt training when validation loss stopped improving.
- Data Preprocessing:
    - Tokenized and converted reviews into sequences of integers.
    - Each review was padded/truncated to 100 words.

- Stopwords were not explicitly removed, as CNNs can learn to ignore unimportant words.

The models were trained using Google Colab with GPU acceleration, significantly reducing training time. After training, the final model was evaluated on the test set, and hyperparameters were fine-tuned based on validation performance.

The Results

The models were trained using the IMDB dataset and evaluated using accuracy and loss metrics. Below are the final results for each model:

Model 1 Performance:

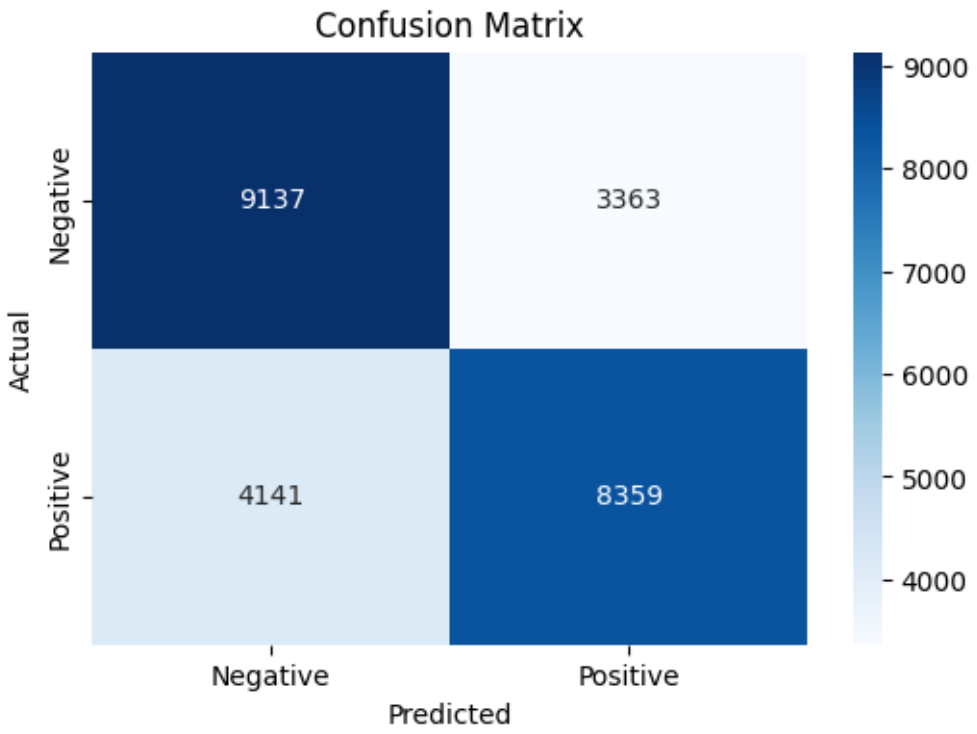
- Final Training Accuracy: 73.52%
- Final Validation Accuracy: 67.63%
- Test Accuracy: 69.98%
- Test Loss: 0.5646

Classification Report:

	precision	recall	f1-score	support
0	0.69	0.73	0.71	12500
1	0.71	0.67	0.69	12500
accuracy			0.70	25000
macro avg	0.70	0.70	0.70	25000
weighted avg	0.70	0.70	0.70	25000

Analysis Model 1 Performance:

- Balanced precision and recall means it performs similarly in both classes
- Overfitting is minimal as validation and test accuracy align closely.



- The model is slightly biased towards classifying reviews as negative as higher FN (4141) than FP (3363)

Model 2 Performance:

- Final Training Accuracy: 95.51%
- Final Validation Accuracy: 66.88%
- Test Accuracy: 66.88%
- Test Loss: 1.1975

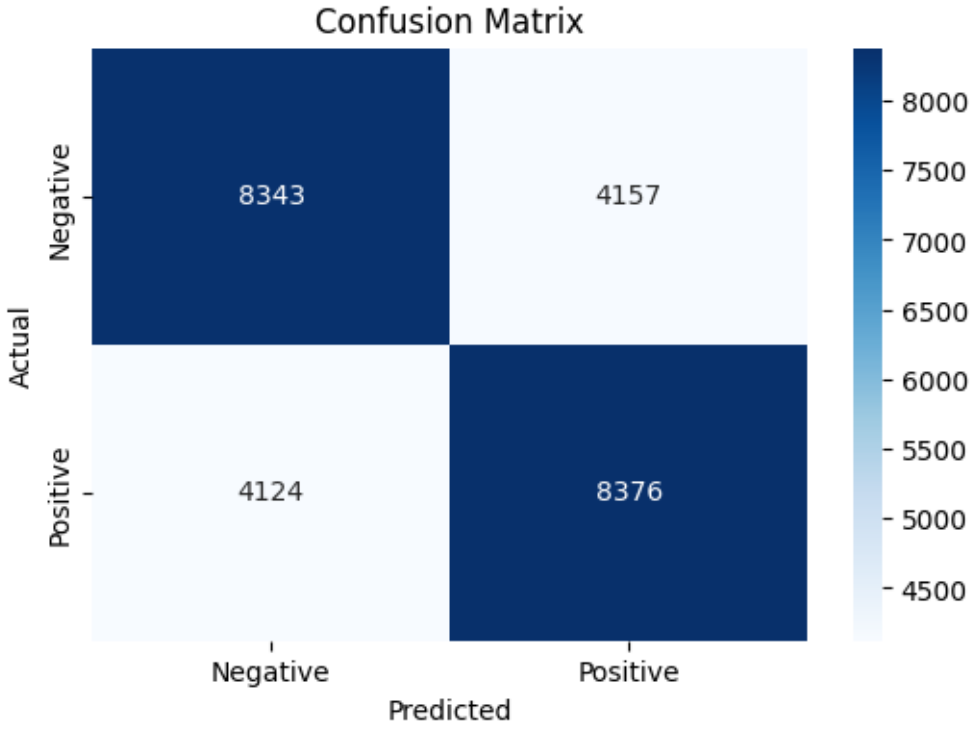
Classification Report:

	precision	recall	f1-score	support
0	0.67	0.67	0.67	12500
1	0.67	0.67	0.67	12500
accuracy			0.67	25000
macro avg	0.67	0.67	0.67	25000
weighted avg	0.67	0.67	0.67	25000

Analysis Model 2 Performance:

- Training accuracy is much higher than test accuracy indicates severe overfitting

- This model fails to generalize well to unseen data despite it having higher complexity (more layers and neurons)



- FP are Higher than model 1 however more negative reviews misclassified
- FN is similar to model 1 but slightly lower

Model 3 (Final Model) Performance:

- Final Training Accuracy: 93.52%
- Final Validation Accuracy: 79.98%
- Test Accuracy: 83.04%
- Test Loss: 0.5128

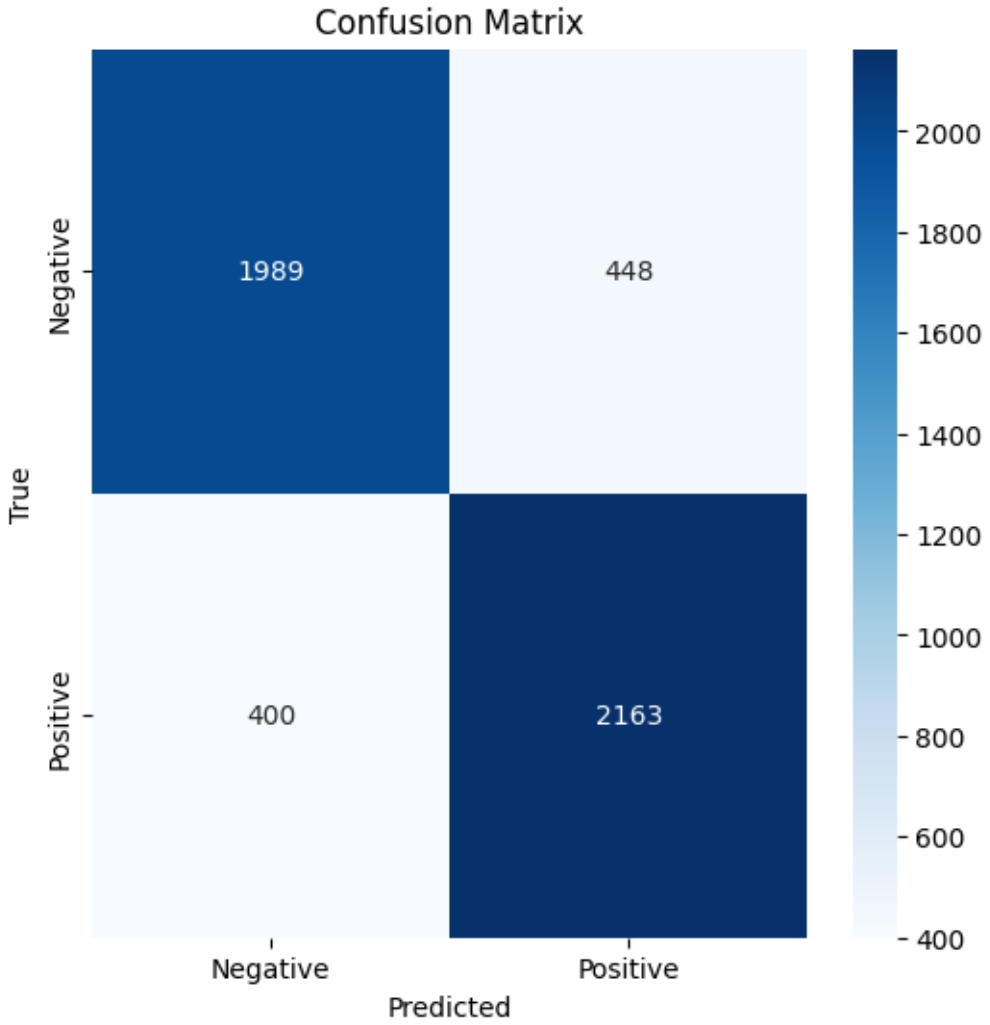
Classification Report:

	precision	recall	f1-score	support
0	0.83	0.82	0.82	2437
1	0.83	0.84	0.84	2563
accuracy			0.83	5000
macro avg	0.83	0.83	0.83	5000
weighted avg	0.83	0.83	0.83	5000

Analysis:

Model 3 Performance:

- This model has the best overall test accuracy of 83.04%
- Lower misclassification rate indicates better generalization, balanced FP and FN values indicate the model is not biased towards any particular class.



- FP and FN are lower than previous models. Indicates this generalizes well and achieves the best trade off between precision and recall among all.

Analysis and Experiments

Hyperparameter Tuning Analysis

- One of the most significant parts of the project was adjusting the hyperparameters, such as the learning rate (LR), batch size, and dropout rate, to understand their effects on both training stability and test performance.
- Initially, I set the learning rate to 0.001, but I quickly ran into issues with the model not converging properly, which led to fluctuations in the validation loss. After tweaking it down to 0.0005, I noticed much smoother convergence and low test loss.
- This taught me that even small changes in learning rate can make a big difference in model performance.

Overfitting and Generalization

- Model 2's performance raised an important concern about overfitting. It achieved an impressive 95.51% on the training set, but the test accuracy was only 66.88%. This made it clear that the model was not generalizing well to unseen data.
- To address this, I incorporated dropout (0.5) for both Model 2 and Model 3, along with batch normalization in Model 3. These changes seemed to have worked, as Model 3 showed much better generalization, with a significant improvement in test accuracy and stability.
- This experiment really emphasized the importance of balancing training accuracy with generalization.

Impact of Batch Size on Stability

- Another crucial experiment involved varying the batch size across different models to see its effect on training performance. Model 1 used a batch size of 128, which provided stability, but it resulted in slightly slower convergence.
- Model 2 used 64, which made training less stable, and this likely contributed to its overfitting issue. Finally, Model 3 used a smaller batch size of 32, which led to more updates per epoch, and ultimately helped improve the test accuracy.
- This showed me that smaller batch sizes can help with generalization by introducing more stochastic updates into the training process.

Confusion Matrix and Error Analysis

- In terms of error analysis, I looked at the false positives (FP) and false negatives (FN) across all models. Model 1 showed a higher number of false negatives, misclassifying positive reviews as negative, while Model 2 showed both false positives and false negatives almost equally, which reflected its generalization issues.
- Model 3, however, had a much more balanced classification, with significantly fewer errors in both categories.

- This analysis was crucial in understanding how well the models performed and which specific areas needed improvement.

Lessons and Experiences Learned

The Importance of Hyperparameter Tuning

- This project reaffirmed how important hyperparameter tuning is. Adjusting factors like learning rate, batch size, and dropout rates had a huge impact on the model's performance, and I learned that even small tweaks could lead to dramatic shifts in how well a model performs.
- It's clear now that understanding these parameters and how they affect the model is essential for building effective models.

Tackling Overfitting

- A major takeaway from this project was how critical it is to address overfitting. In the case of Model 2, the high training accuracy, coupled with the poor test accuracy, highlighted the danger of overfitting.
- By adding dropout and batch normalization to Model 3, I was able to mitigate the overfitting and see a much better balance between training and test performance.
- It reinforced the idea that regularization techniques are a must-have when building robust models.

Evaluating with More Than Just Accuracy

- One of the key lessons was learning that accuracy alone doesn't give the full picture. The project taught me the importance of using other evaluation metrics like precision, recall, and F1-score.
- These helped me understand where the models were going wrong, especially in cases where accuracy might look fine but the model is still making significant errors in classification.

Real-World Applications

- In terms of practical application, this project reinforced the relevance of sentiment analysis, especially in industries like marketing and customer service where understanding customer sentiment can drive decision-making.