ATHENS UNIVERSITY OF ECONOMICS & BUSINESS

M.Sc. in Data Science

ΟΙΚΟΝΟΜΙΚΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΑΘΗΝΩΝ
ATHENS UNIVERSITY
OF ECONOMICS
AND BUSINESS

Text Analytics

Instructor: Ion Androutsopoulos

Assignment 05 - Natural Language Processing with Convolutional Neural Networks

Anagnos Theodoros (p3352323) - Michalopoulos Ioannis (p3352314) - Kafantaris Panagiotis (p3352328) - Vigkos Ioannis (p3352326)

6.6.2024

Contents

1 Exercise 02

In this report, we

The implementation details and code can be accessed through the following Colab notebook link:

Link here

2 Dataset and Preprocessing

2.1 Dataset

We used a Twitter dataset from Kaggle for our sentiment analysis task. The dataset includes tweets labeled with sentiment categories: positive, negative, and neutral. The dataset statistics are as follows:

- Number of records: *4870*
- Number of classes: 3 (Positive, Negative, Neutral)
- Vocabulary size: *6000 terms, vectorized with TF-IDF*

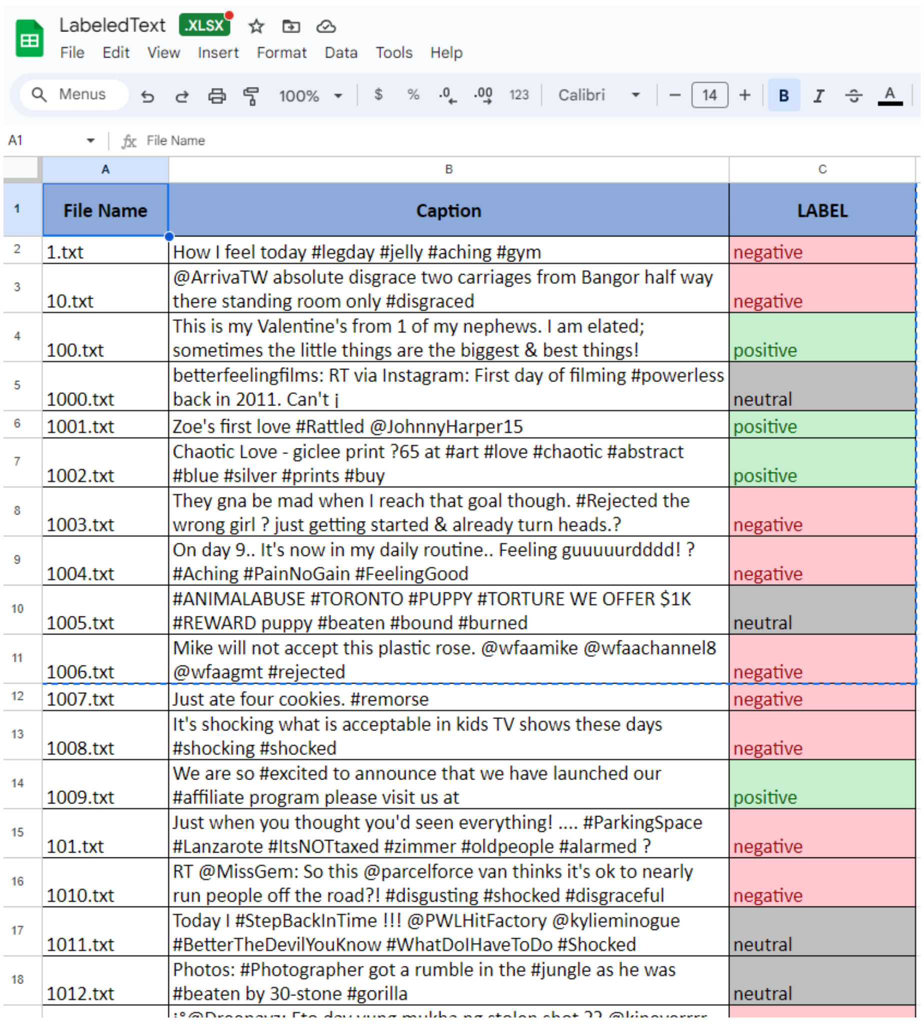
Provenance:

- This dataset enriches a benchmark dataset available at: `https://mcrlab.net/research/mvsa-sentiment-analysis-on-multi-view-social-data/`

Collection Methodology:

- This dataset is enriched by augmenting a dataset of images and captions with sentiment analysis-generated labels. A method was put forth to determine the ideal sentiment using the caption feature to implement six different sentiment analysis techniques. Final classification of each dataset item is based on the percentage of positive, negative, and neutral polarities extracted for each caption.

Sample:


LabeledText .XLSX
File Edit View Insert Format Data Tools Help
Menus 100% $ % .0 .00 123 Calibri 14 B I A
A1 File Name

File Name	Caption	LABEL
1.txt	How I feel today #legday #jelly #aching #gym	negative
10.txt	@ArrivaTW absolute disgrace two carriages from Bangor half way there standing room only #disgraced	negative
100.txt	This is my Valentine's from 1 of my nephews. I am elated; sometimes the little things are the biggest & best things!	positive
1000.txt	betterfeelingfilms: RT via Instagram: First day of filming #powerless back in 2011. Can't ¡	neutral
1001.txt	Zoe's first love #Rattled @JohnnyHarper15	positive
1002.txt	Chaotic Love - giclee print ?65 at #art #love #chaotic #abstract #blue #silver #prints #buy	positive
1003.txt	They gna be mad when I reach that goal though. #Rejected the wrong girl ? just getting started & already turn heads.?	negative
1004.txt	On day 9.. It's now in my daily routine.. Feeling guuuuurdddd! ? #Aching #PainNoGain #FeelingGood	negative
1005.txt	#ANIMALABUSE #TORONTO #PUPPY #TORTURE WE OFFER $1K #REWARD puppy #beaten #bound #burned	neutral
1006.txt	Mike will not accept this plastic rose. @wfaamike @wfaachannel8 @wfaagmt #rejected	negative
1007.txt	Just ate four cookies. #remorse	negative
1008.txt	It's shocking what is acceptable in kids TV shows these days #shocking #shocked	negative
1009.txt	We are so #excited to announce that we have launched our #affiliate program please visit us at	positive
101.txt	Just when you thought you'd seen everything! .... #ParkingSpace #Lanzarote #ItsNOTtaxed #zimmer #oldpeople #alarmed ?	negative
1010.txt	RT @MissGem: So this @parcelforce van thinks it's ok to nearly run people off the road?! #disgusting #shocked #disgraceful	negative
1011.txt	Today I #StepBackInTime !!! @PWLHitFactory @kylieminogue #BetterTheDevilYouKnow #WhatDoIHaveToDo #Shocked	neutral
1012.txt	Photos: #Photographer got a rumble in the #jungle as he was #beaten by 30-stone #gorilla	neutral
	[illegible]	

Figure 1: Dataset sample

2.2 Preprocessing

The preprocessing steps included:

- **Converting text to lowercase:** This step helps in normalizing the text data by removing case sensitivity, ensuring that words like "Happy" and "happy" are treated the same.
- **Removing URLs, HTML tags, punctuation, numbers, and extra spaces:** These elements are generally not useful for sentiment analysis and can be considered as noise. Removing them helps in focusing on the actual textual content.
- **Converting chat words to their full forms:** Chat words (e.g., "idk" to "I don't know") are expanded to their full forms to enhance the understanding of the text.

- **Tokenization and removing stop words:** Tokenization splits the text into individual words, and stop words (common words like "the", "is") are removed to reduce noise and focus on the significant words.
- **Lemmatization using NLTK:** Lemmatization reduces words to their base or root form (e.g., "running" to "run"), which helps in treating different forms of a word as a single item.

3 Baseline Classifiers

3.1 Logistic Regression Classifier

We used TF-IDF (Term Frequency-Inverse Document Frequency) features with unigram and bigram representations to capture both individual words and pairs of consecutive words. TF-IDF helps in identifying the importance of words in a document relative to the entire dataset.

Logistic Regression is a linear model commonly used for binary and multi-class classification tasks. It estimates the probability that a given input belongs to a particular class by applying the logistic function. In our case, we trained a logistic regression classifier on TF-IDF features to serve as a baseline for comparison with more complex models.

3.2 RNN

In addition to the logistic regression model, we also used a Recurrent Neural Network (RNN) model with a bidirectional Long Short-Term Memory (LSTM) layer and self-attention mechanism as another baseline. This model leverages the sequential nature of text data by processing input in both forward and backward directions through the bidirectional LSTM, and the self-attention mechanism helps the model to focus on relevant parts of the input sequence when making predictions. This RNN-based approach allows us to capture more complex patterns and dependencies in the text data.

4 CNN Model

We develop a Convolutional Neural Network (CNN) model with multiple filter sizes (2-gram, 3-gram, and 4-gram) using the Keras Functional API for text classification. Here is a summary of the key components:

4.1 Inputs and Embedding

- The model starts with an input layer for string data.
- This input is passed through a text vectorizer to convert the text into sequences of integers.
- An embedding layer is applied to the input sequences to transform them into dense vectors of fixed size.

4.2 Dropout

- A dropout layer is added to the embeddings to prevent overfitting by randomly setting a fraction of input units to 0 at each update during training.

4.3 Multi-Filter CNNs:

- The model employs convolutional layers with filter sizes of 2, 3, and 4 to capture different n-gram patterns in the text.
- Each convolutional filter is followed by two additional convolutional layers to increase the depth.
- Global Max Pooling is applied after each set of convolutions to downsample the input representations.

4.4 Concatenation and Output:

- The pooled outputs from all filter sizes are concatenated into a single vector.
- A dropout layer is applied to the concatenated vector for further regularization.
- A dense output layer with a softmax activation function is used for multi-class classification.

4.5 Model Compilation and Training:

- The model is compiled with categorical crossentropy loss and the Adam optimizer.
- The model summary is printed to provide an overview of the architecture.
- Training and validation data are prepared by converting the input text data into TensorFlow string tensors.
- An early stopping callback is configured to monitor validation performance and stop training when no improvement is observed.

4.6 Training Data Preparation:

- Training, validation, and test datasets are preprocessed by joining words in each text sample and converting them into numpy arrays of strings.
- These arrays are then converted to TensorFlow tensors for model training.

The overall structure allows the model to capture various patterns in the text through multiple convolutional layers with different filter sizes, aiming to improve the performance on text classification tasks.

5 Hyperparameter Tuning

The optimal hyperparameters for the CNN model have been identified to improve its performance and generalization capability. The number of filters has been set to 192, allowing the model to capture more nuanced features from the input text. A dropout rate of 0.4 is used to prevent overfitting by randomly setting 40% of the input units to zero during training, promoting better generalization. The learning rate is set at 0.001, which ensures stable and efficient convergence, balancing progress towards optimal weights without overshooting. These optimized hyperparameters collectively enhance the model's ability to learn effectively while minimizing overfitting.

6 Results

6.1 Model Performance

- The overall accuracy of the model on the validation set is 66%, which indicates a moderate performance.
- The model performs best on the positive class, with an F1-score of 0.76, while it performs the worst on the neutral class with an F1-score of 0.60.
- Precision and recall values are relatively balanced across the classes, though the neutral class has slightly lower performance.
- Overfitting:
  - The significant discrepancy between the training and validation metrics is a clear sign of overfitting.
  - Techniques such as adding more regularization, using dropout layers, or employing data augmentation could help mitigate overfitting.
  - Early stopping could also be used more effectively to halt training as soon as validation performance starts to degrade.

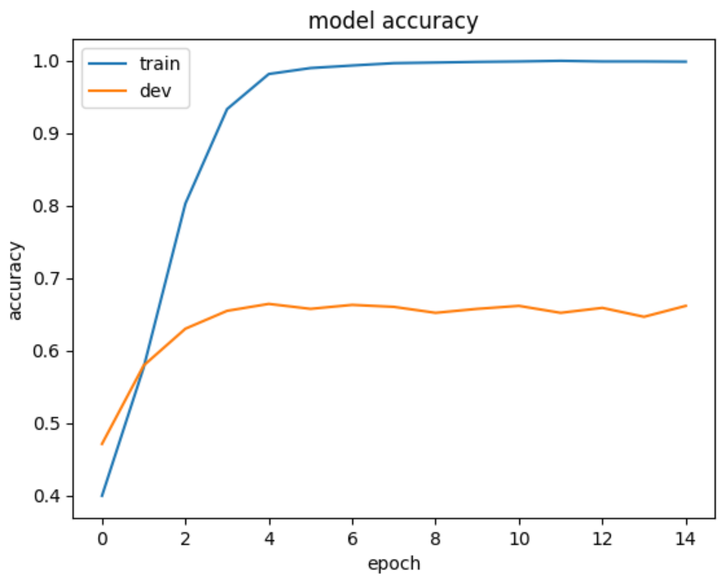


Figure 2: Learning curves showing the model accuracy on the training and development sets as a function of epochs.

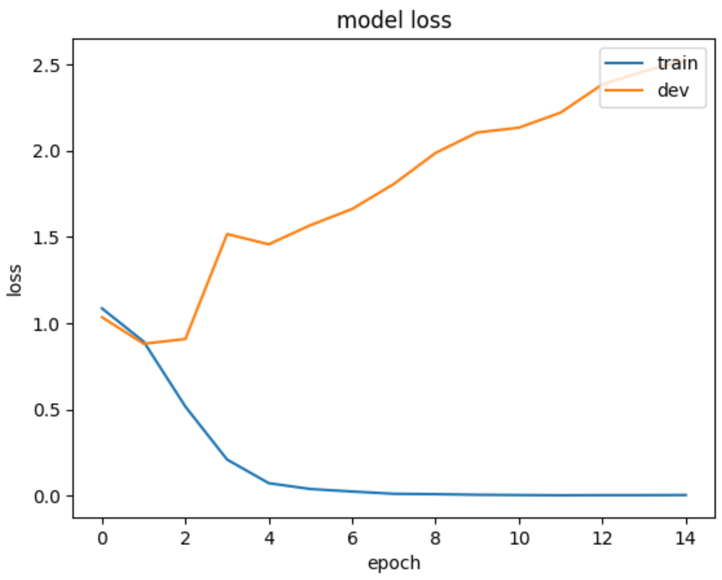


Figure 3: Learning curves showing the model loss on the training and development sets as a function of epochs.

6.2 Metrics for Model Evaluation

6.2.1 Baseline Logistic Regression

Table 1: Performance Metrics for Logistic Regression on the Test Subset

Class	Precision	Recall	F1-Score	Support
Negative	0.70	0.64	0.67	221
Neutral	0.61	0.66	0.63	273
Positive	0.74	0.73	0.73	237
Accuracy			0.67 (731)	
Macro Avg	0.68	0.67	0.68	731
Weighted Avg	0.68	0.67	0.68	731

6.2.2 Baseline RNN Model

Table 2: Performance Metrics for Logistic Regression on the Test Subset

Class	Precision	Recall	F1-Score	Support
Negative	0.74	0.67	0.70	221
Neutral	0.69	0.63	0.66	273
Positive	0.73	0.88	0.80	237
Accuracy			0.72 (731)	
Macro Avg	0.72	0.72	0.72	731
Weighted Avg	0.72	0.72	0.72	731

6.2.3 CNN Model

Table 3: Performance Metrics for the Model on the Test Subset

Class	Precision	Recall	F1-Score	Support
Negative	0.58	0.66	0.62	221
Neutral	0.63	0.57	0.60	273
Positive	0.77	0.76	0.76	237
Accuracy			0.66 (731)	
Macro Avg	0.66	0.66	0.66	731
Weighted Avg	0.66	0.66	0.66	731

7 Conclusion

Comparing the RNN and CNN models, the RNN outperforms the CNN with an overall accuracy of 72% versus 66%. The RNN shows better precision, recall, and F1-scores across all classes, particularly excelling in the positive class with an F1-score of 0.80 compared to the CNN's 0.76. The RNN also demonstrates more balanced performance across all metrics, suggesting it is better at capturing the sequential dependencies in the text data, leading to improved generalization and robustness in sentiment classification.

8 Exercise 03

8.1 Introduction

In this report, we present a solution for POS tagging using a stacked CNN with n-gram filters, residual connections, and a dense layer. This task involves using Keras/TensorFlow to implement the model, tuning the hyperparameters, and comparing the performance with baseline models.

The implementation details and code can be accessed through the following Colab notebook link: Link here

8.2 Dataset and Preprocessing

8.3 Dataset

We used the Universal Dependencies English Web Treebank dataset for our POS tagging task. The dataset includes sentences annotated with POS tags. The statistics are as follows:

- Number of sentences: 8780 (train), 1882 (dev), 1882 (test)
- Vocabulary size: 20202 terms (vectorized with Word2Vec)

Provenance:

- Universal Dependencies: `http://universaldependencies.org/`

8.4 Preprocessing

The preprocessing steps included:

- Converting text to lowercase.
- Removing punctuation, numbers, and extra spaces.
- Creating sliding windows for words.
- Mapping words to pre-trained Word2Vec embeddings.
- Tokenizing windows using the created vocabulary.
- Encoding POS tags using LabelEncoder and OneHotEncoder.

8.5 Baseline Models

8.6 Frequentic Baseline Models

To understand the distribution of tags in the training data, we calculate the most frequent tag. The training data consists of POS-tagged words, which we analyse to determine the most common tag. The baseline model predicts the tag for each word based on its most frequent tag in the training data. If a word is not encountered in the training data, the model assigns the overall most frequent tag.

8.6.1 Results

8.6.2 Most Frequent Tag

The most frequent tag in the training data is: **NOUN**.

8.6.3 Baseline Model Accuracy

The accuracy of the baseline model on the test set is: **0.88**.

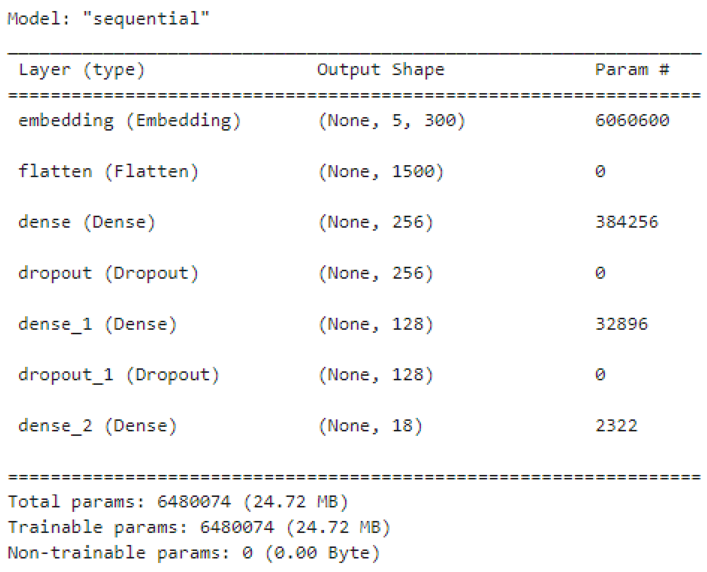
```
              precision    recall  f1-score   support

         ADJ       0.89      0.82      0.85      1947
         ADP       0.86      0.87      0.87      2654
         ADV       0.91      0.75      0.82      1539
         AUX       0.94      0.90      0.92      1951
       CCONJ       0.98      1.00      0.99       946
         DET       0.96      0.97      0.97      2448
        INTJ       0.93      0.81      0.86       125
        NOUN       0.76      0.93      0.83      5159
         NUM       0.94      0.85      0.89       589
        PART       0.71      0.99      0.83       879
        PRON       0.97      0.93      0.95      2741
       PROPN       0.94      0.73      0.82      1834
       PUNCT       0.99      0.99      0.99      3523
       SCONJ       0.67      0.61      0.64       580
         SYM       0.83      0.74      0.78        97
        VERB       0.87      0.80      0.83      3313
           X       0.62      0.28      0.38        47
           _       0.98      0.80      0.88       388

    accuracy                           0.88     30760
   macro avg       0.88      0.82      0.84     30760
weighted avg       0.89      0.88      0.88     30760
```

8.7 MLP Baseline

We implemented an MLP baseline using Keras. The architecture included an embedding layer, two dense layers with ReLU activation, and dropout for regularization. The model was trained using categorical cross-entropy loss and the Adam optimizer.

8.8 MLP Model Summary

The following table shows the architecture of the MLP model. It includes an embedding layer to convert words into dense vectors, followed by flatten, dense, and dropout layers. The model uses a softmax activation in the output layer for multi-class classification.



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 5, 300)            6060600

 flatten (Flatten)           (None, 1500)              0

 dense (Dense)               (None, 256)               384256

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 128)               32896

 dropout_1 (Dropout)         (None, 128)               0

 dense_2 (Dense)             (None, 18)                2322

=================================================================
Total params: 6480074 (24.72 MB)
Trainable params: 6480074 (24.72 MB)
Non-trainable params: 0 (0.00 Byte)
```

Figure 4: MLP Model Summary

Figure 5 and Figure 6 depict the adjusted MLP model's accuracy and loss over 25 epochs for both training and validation datasets. In Figure 5, the training accuracy shows a rapid increase, stabilizing around 0.90, while the validation accuracy closely follows, indicating good generalization. Figure 6 shows the loss curves, where the training loss decreases significantly, leveling off below 0.8, while the validation loss initially decreases and then fluctuates around 0.8. The close alignment of training and validation metrics suggests that the adjustments made to the MLP model have improved its performance, reducing overfitting and enhancing generalization to validation data.

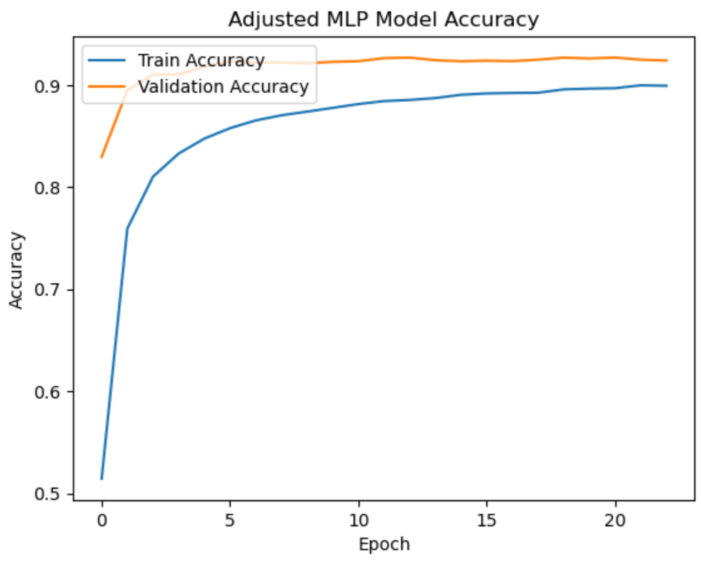


Figure 5: Adjusted MLP Model Accuracy

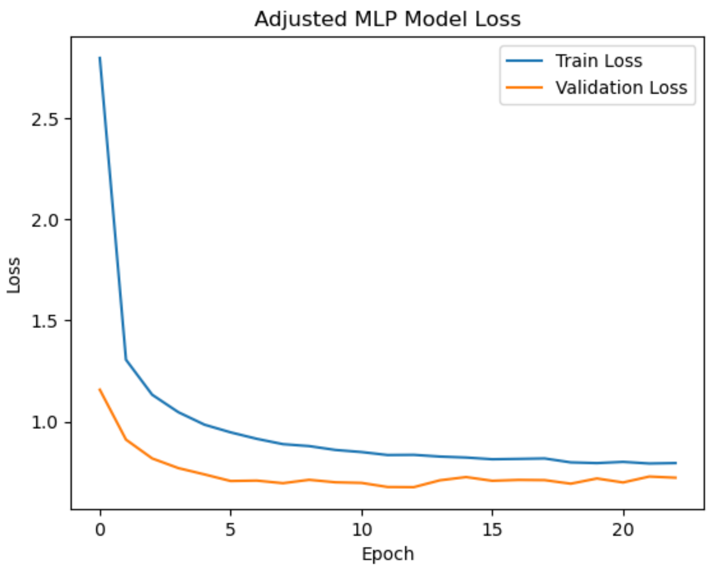


Figure 6: Adjusted MLP Model Loss

Table 4: Classification Report for the Adjusted MLP Model on Test Data

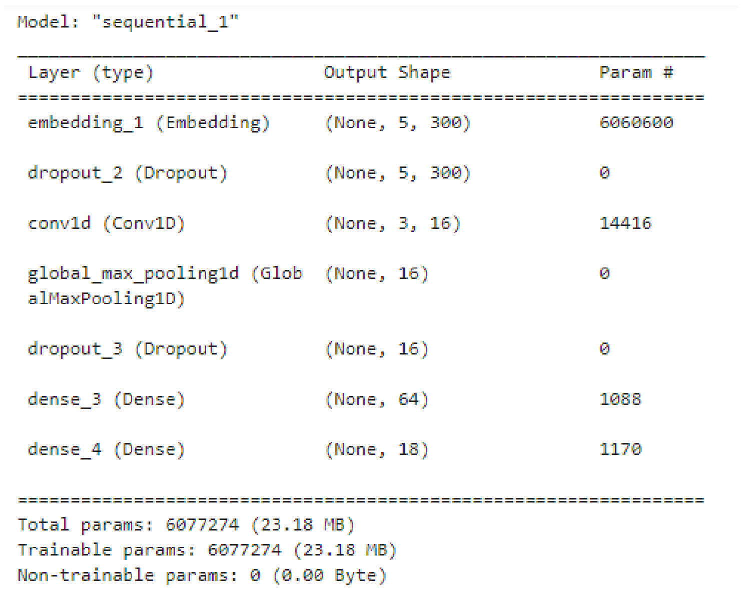
Class	Precision	Recall	F1-Score	Support
ADJ	0.86	0.90	0.88	1947
ADP	0.87	0.98	0.92	2654
ADV	0.93	0.79	0.85	1539
AUX	0.97	0.98	0.98	1951
CCONJ	0.99	0.99	0.99	946
DET	0.98	0.98	0.98	2448
INTJ	0.67	0.06	0.12	125
NOUN	0.89	0.94	0.91	5159
NUM	0.95	0.88	0.91	589
PART	0.96	0.95	0.95	879
PRON	0.98	0.98	0.98	2741
PROPN	0.85	0.84	0.85	1834
PUNCT	0.98	0.99	0.98	3523
SCONJ	0.73	0.50	0.59	580
SYM	0.20	0.01	0.02	97
VERB	0.95	0.91	0.93	3313
X	0.00	0.00	0.00	47
_	0.95	0.93	0.94	388
Accuracy	0.92 (30,760)			
Macro Avg	0.82	0.76	0.77	30,760
Weighted Avg	0.92	0.92	0.92	30,760

8.9 Baseline CNN

We implemented a baseline CNN using Keras. The architecture included an embedding layer, a single convolutional layer with n-gram filters, a global max-pooling layer, and dense layers with dropout for regularization. The model was trained using categorical cross-entropy loss and the Adam optimizer.

8.10 CNN Model Summary

The following table shows the architecture of the baseline CNN model. It includes an embedding layer to convert words into dense vectors, followed by a convolutional layer, global max-pooling, and dropout layers. The model uses a softmax activation in the output layer for multi-class classification.



```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 5, 300)            6060600

 dropout_2 (Dropout)         (None, 5, 300)            0

 conv1d (Conv1D)             (None, 3, 16)             14416

 global_max_pooling1d (Glob  (None, 16)                0
 alMaxPooling1D)

 dropout_3 (Dropout)         (None, 16)                0

 dense_3 (Dense)             (None, 64)                1088

 dense_4 (Dense)             (None, 18)                1170

=================================================================
Total params: 6077274 (23.18 MB)
Trainable params: 6077274 (23.18 MB)
Non-trainable params: 0 (0.00 Byte)
```

Figure 7: Baseline CNN Model Summary

Figure 8 and Figure 9 depict the baseline CNN model's accuracy and loss over 20 epochs for both training and validation datasets. In Figure 8, the training accuracy shows a gradual increase, while the validation accuracy fluctuates and declines after a certain point, indicating overfitting. Figure 9 shows the loss curves, where the training loss decreases steadily, but the validation loss increases after the initial epochs, further indicating overfitting.

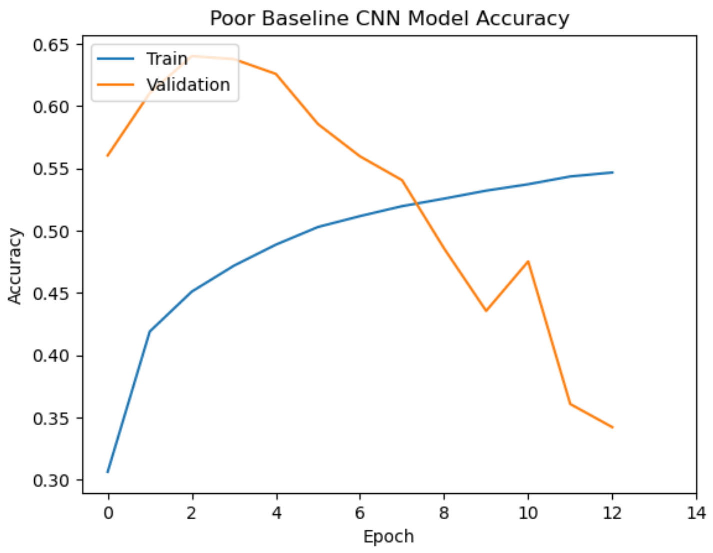


Figure 8: Baseline CNN Model Accuracy

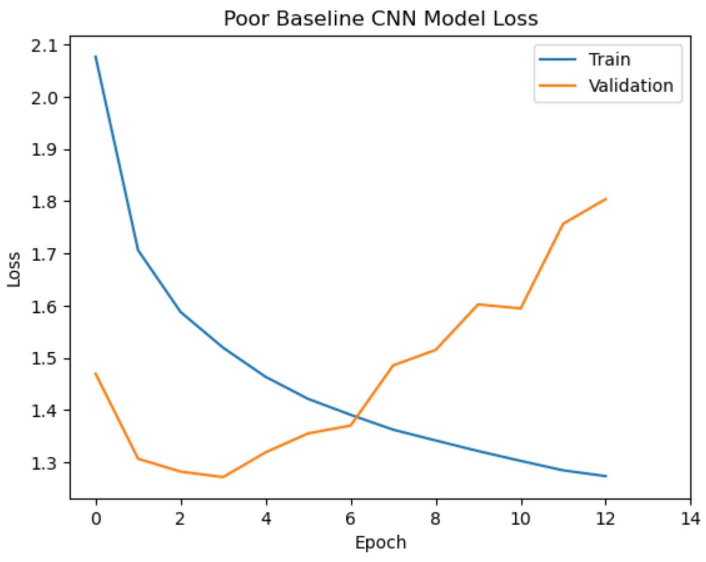


Figure 9: Baseline CNN Model Loss

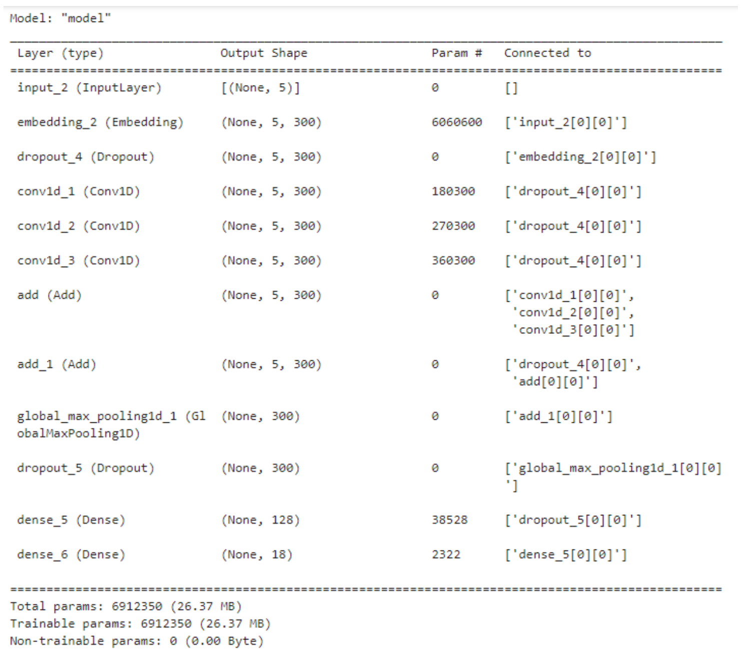
Table 5: Classification Report for the Baseline CNN Model on Test Data

Class	Precision	Recall	F1-Score	Support
ADJ	1.00	0.01	0.01	1947
ADP	0.65	0.87	0.74	2654
ADV	0.60	0.36	0.45	1539
AUX	0.80	0.95	0.87	1951
CCONJ	0.85	0.91	0.88	946
DET	0.88	0.79	0.83	2448
INTJ	0.00	0.00	0.00	125
NOUN	0.43	0.81	0.56	5159
NUM	0.00	0.00	0.00	589
PART	0.85	0.07	0.13	879
PRON	0.65	0.94	0.77	2741
PROPN	0.85	0.44	0.58	1834
PUNCT	0.95	0.67	0.78	3523
SCONJ	1.00	0.01	0.02	580
SYM	0.00	0.00	0.00	97
VERB	0.54	0.59	0.57	3313
X	0.00	0.00	0.00	47
_	1.00	0.01	0.01	388
Accuracy		0.63 (30,760)		
Macro Avg	0.61	0.41	0.40	30,760
Weighted Avg	0.70	0.63	0.59	30,760

8.11 Stacked CNN with n-gram filters (n = 2, 3, 4)

8.12 CNN Model Summary

The following table shows the architecture of the Stacked CNN model. It includes an embedding layer to convert words into dense vectors, followed by convolutional layers with n-gram filters, residual connections, and a dense layer. The model uses a softmax activation in the output layer for multi-class classification.



```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 input_2 (InputLayer)        [(None, 5)]                  0         []

 embedding_2 (Embedding)     (None, 5, 300)               6060600   ['input_2[0][0]']

 dropout_4 (Dropout)         (None, 5, 300)               0         ['embedding_2[0][0]']

 conv1d_1 (Conv1D)           (None, 5, 300)               180300    ['dropout_4[0][0]']

 conv1d_2 (Conv1D)           (None, 5, 300)               270300    ['dropout_4[0][0]']

 conv1d_3 (Conv1D)           (None, 5, 300)               360300    ['dropout_4[0][0]']

 add (Add)                   (None, 5, 300)               0         ['conv1d_1[0][0]',
                                                                     'conv1d_2[0][0]',
                                                                     'conv1d_3[0][0]']

 add_1 (Add)                 (None, 5, 300)               0         ['dropout_4[0][0]',
                                                                     'add[0][0]']

 global_max_pooling1d_1 (Gl  (None, 300)                  0         ['add_1[0][0]']
 obalMaxPooling1D)

 dropout_5 (Dropout)         (None, 300)                  0         ['global_max_pooling1d_1[0][0]
                                                                    ']

 dense_5 (Dense)             (None, 128)                  38528     ['dropout_5[0][0]']

 dense_6 (Dense)             (None, 18)                   2322      ['dense_5[0][0]']

==================================================================================================
Total params: 6912350 (26.37 MB)
Trainable params: 6912350 (26.37 MB)
Non-trainable params: 0 (0.00 Byte)
```

Figure 10: Stacked CNN Model Summary

8.13 Architecture

We implemented a stacked CNN with n-gram filters (n = 2, 3, 4), residual connections, and a dense layer. The architecture consisted of:

- **Embedding layer:** Converts words into 300-dimensional vectors using pre-trained Word2Vec embeddings.
- **Convolutional layers:** Three convolutional layers with n-gram filters (2, 3, 4) and residual connections.
- **Global Max Pooling layer:** Reduces the output of convolutional layers to fixed-size vectors.
- **Fully connected layer:** Dense layer with ReLU activation.
- **Dropout layers:** Prevents overfitting by randomly setting input units to zero.
- **Output layer:** Softmax activation for multi-class classification.

Figures 11 and 12 depict the accuracy and loss of the Stacked CNN model over 20 epochs for both training and validation datasets. The training accuracy rapidly increases and converges around 0.98, while the validation accuracy remains stable around 0.95, indicating good generalization. Figure 12 shows the loss curves, where the training loss decreases steadily and the validation loss remains relatively low and stable, suggesting minimal overfitting.

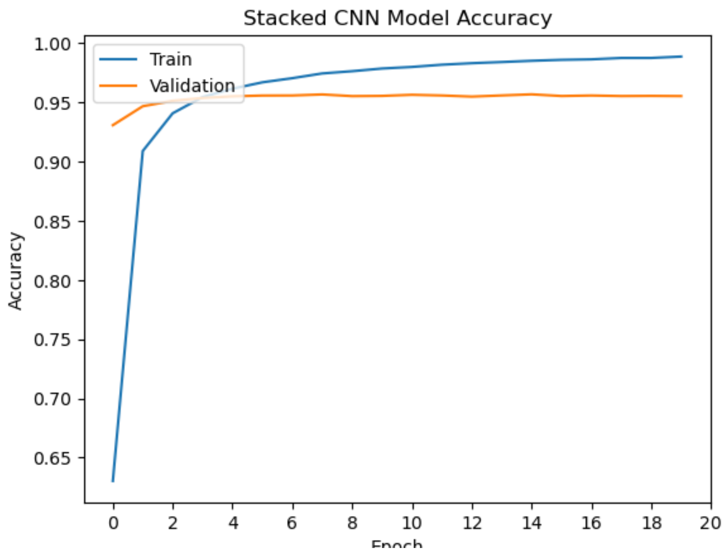


Figure 11: Stacked CNN Model Accuracy

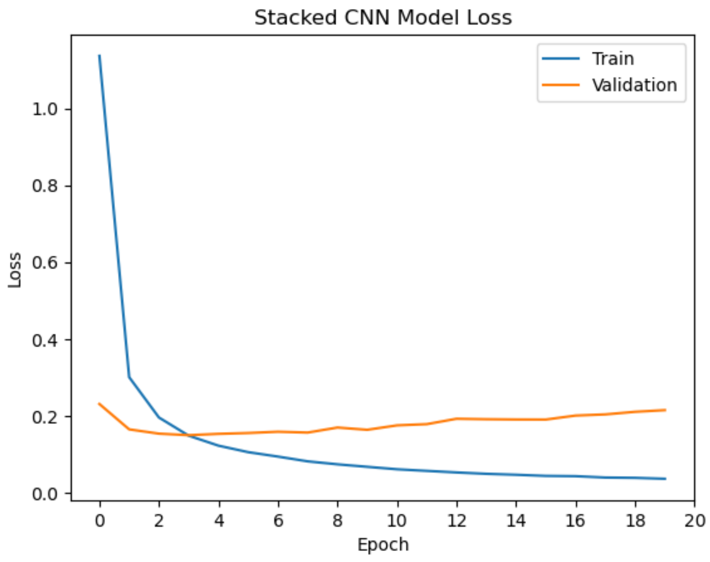


Figure 12: Stacked CNN Model Loss

Table 6: Classification Report for the Stacked CNN Model on Test Data

Class	Precision	Recall	F1-Score	Support
ADJ	0.92	0.91	0.92	1947
ADP	0.94	0.96	0.95	2654
ADV	0.93	0.88	0.91	1539
AUX	0.99	0.99	0.99	1951
CCONJ	0.97	1.00	0.98	946
DET	0.99	0.99	0.99	2448
INTJ	0.86	0.90	0.88	125
NOUN	0.94	0.94	0.94	5159
NUM	0.83	0.98	0.90	589
PART	0.98	0.99	0.98	879
PRON	0.99	0.99	0.99	2741
PROPN	0.90	0.90	0.90	1834
PUNCT	0.99	0.99	0.99	3523
SCONJ	0.87	0.84	0.85	580
SYM	0.79	0.90	0.84	97
VERB	0.95	0.95	0.95	3313
X	0.65	0.36	0.47	47
_	0.99	0.98	0.98	388
Accuracy	0.95 (30,760)			
Macro Avg	0.92	0.91	0.91	30,760
Weighted Avg	0.95	0.95	0.95	30,760

8.14 Training and Hyperparameter Tuning

The model was trained using the Adam optimizer. Early stopping and model checkpoints were used to monitor validation performance and prevent overfitting. Dropout layers with a rate of 0.5 were used to randomly set input units to zero during training.

8.15 Results

8.16 Evaluation Metrics

Table 7: Performance Metrics for Stacked CNN on the Test Subset

Class	Precision	Recall	F1-Score	Support
ADJ	0.95	0.88	0.91	1788
ADP	0.90	0.93	0.91	2034
ADV	0.92	0.87	0.89	1176
AUX	0.98	0.99	0.98	1543
CCONJ	0.88	0.99	0.93	737
DET	0.96	0.98	0.97	1897
INTJ	0.67	0.69	0.68	120
NOUN	0.85	0.92	0.88	4137
NUM	0.76	0.70	0.73	542
PART	0.89	0.94	0.91	649
PRON	0.95	0.97	0.96	2162
PROPN	0.81	0.78	0.80	2077
PUNCT	0.95	0.99	0.97	3096
SCONJ	0.85	0.88	0.86	384
SYM	0.76	0.75	0.75	109
VERB	0.92	0.91	0.91	2606
X	0.00	0.00	0.00	39
_	0.92	0.91	0.91	354
Accuracy	0.92 (25,450)			
Macro Avg	0.84	0.84	0.84	25,450
Weighted Avg	0.92	0.92	0.92	25,450

The Stacked CNN achieved a test accuracy of 0.95, demonstrating significant improvement over the baseline models.

8.17 Hyperparameter Tuning Insights

To optimize the performance of our Stacked CNN model, we employed a hyperparameter tuning strategy using the Keras Tuner library. The tuning process aimed to find the best combination of hyperparameters that maximized the validation categorical accuracy.

8.18 Tuning Methodology

We defined a search space for hyperparameters, including the dropout rates, number of filters, units in the dense layer, and the learning rate. The tuning was performed using the `RandomSearch` tuner, which explored various combinations to identify the optimal settings.

8.19 Tuning Results

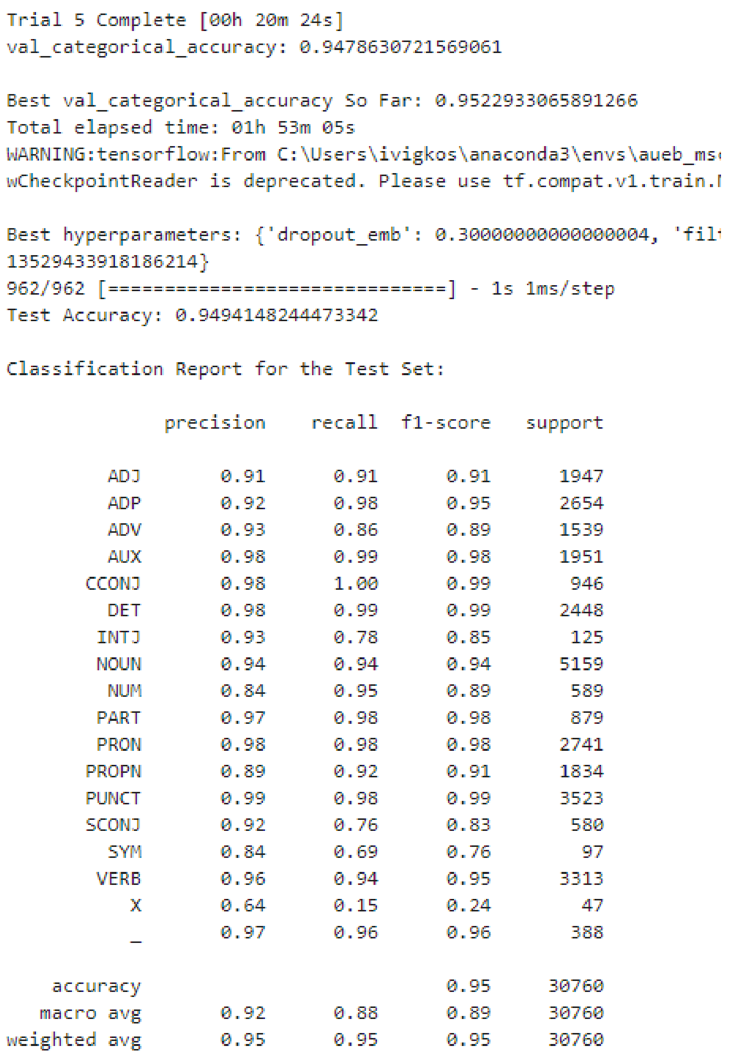
The hyperparameter tuning process was completed in approximately 1 hour and 53 minutes, resulting in the following best hyperparameters:

- **Dropout rate for embedding layer:** 0.3
- **Number of filters:** 96
- **Dropout rate for convolutional layers:** 0.3
- **Dense units:** 64
- **Learning rate:** 0.000135

The tuned Stacked CNN model achieved a test accuracy of 0.95. The classification report for the test set is presented below:

Table 8: Classification Report for the Tuned Stacked CNN Model on Test Data

Class	Precision	Recall	F1-Score	Support
ADJ	0.91	0.91	0.91	1947
ADP	0.92	0.98	0.95	2654
ADV	0.93	0.86	0.89	1539
AUX	0.98	0.99	0.98	1951
CCONJ	0.98	1.00	0.99	946
DET	0.98	0.99	0.99	2448
INTJ	0.93	0.78	0.85	125
NOUN	0.94	0.94	0.94	5159
NUM	0.84	0.95	0.89	589
PART	0.97	0.98	0.98	879
PRON	0.98	0.98	0.98	2741
PROPN	0.89	0.92	0.91	1834
PUNCT	0.99	0.98	0.99	3523
SCONJ	0.92	0.76	0.83	580
SYM	0.84	0.69	0.76	97
VERB	0.96	0.94	0.95	3313
X	0.64	0.15	0.24	47
_	0.97	0.96	0.96	388
Accuracy	0.95 (30,760)			
Macro Avg	0.92	0.88	0.89	30,760
Weighted Avg	0.95	0.95	0.95	30,760



```
Trial 5 Complete [00h 20m 24s]
val_categorical_accuracy: 0.9478630721569061

Best val_categorical_accuracy So Far: 0.9522933065891266
Total elapsed time: 01h 53m 05s
WARNING:tensorflow:From C:\Users\ivigkos\anaconda3\envs\aueb_ms
wCheckpointReader is deprecated. Please use tf.compat.v1.train.

Best hyperparameters: {'dropout_emb': 0.30000000000000004, 'fil
13529433918186214}
962/962 [==============================] - 1s 1ms/step
Test Accuracy: 0.9494148244473342

Classification Report for the Test Set:

              precision    recall  f1-score   support

         ADJ       0.91      0.91      0.91      1947
         ADP       0.92      0.98      0.95      2654
         ADV       0.93      0.86      0.89      1539
         AUX       0.98      0.99      0.98      1951
       CCONJ       0.98      1.00      0.99       946
         DET       0.98      0.99      0.99      2448
        INTJ       0.93      0.78      0.85       125
        NOUN       0.94      0.94      0.94      5159
         NUM       0.84      0.95      0.89       589
        PART       0.97      0.98      0.98       879
        PRON       0.98      0.98      0.98      2741
       PROPN       0.89      0.92      0.91      1834
       PUNCT       0.99      0.98      0.99      3523
       SCONJ       0.92      0.76      0.83       580
         SYM       0.84      0.69      0.76        97
        VERB       0.96      0.94      0.95      3313
           X       0.64      0.15      0.24        47
           _       0.97      0.96      0.96       388

    accuracy                           0.95     30760
   macro avg       0.92      0.88      0.89     30760
weighted avg       0.95      0.95      0.95     30760
```

Figure 13: Classification Report for the Tuned Stacked CNN Model on Test Data

The optimized model demonstrates a significant improvement in performance, particularly in terms of precision, recall, and F1-score for most classes. The high test accuracy and balanced performance across different POS tags indicate the model's robustness and effectiveness in POS tagging tasks.

8.20 Evaluation of the Best CNN Model

The best CNN model, selected through hyperparameter tuning, was evaluated on the test set to determine its performance. The model achieved a high test accuracy of 0.95, indicating robust performance across the different parts of speech (POS) tags.

8.21 Test Set Performance

The classification report for the test set provides a detailed breakdown of the model's precision, recall, and F1-score for each POS tag. The results are as follows:

Table 9: Classification Report for the Best CNN Model on Test Data

Class	Precision	Recall	F1-Score	Support
ADJ	0.91	0.91	0.91	1947
ADP	0.92	0.98	0.95	2654
ADV	0.93	0.86	0.89	1539
AUX	0.98	0.99	0.98	1951
CCONJ	0.98	1.00	0.99	946
DET	0.98	0.99	0.99	2448
INTJ	0.93	0.78	0.85	125
NOUN	0.94	0.94	0.94	5159
NUM	0.84	0.95	0.89	589
PART	0.97	0.98	0.98	879
PRON	0.98	0.98	0.98	2741
PROPN	0.89	0.92	0.91	1834
PUNCT	0.99	0.98	0.99	3523
SCONJ	0.92	0.76	0.83	580
SYM	0.84	0.69	0.76	97
VERB	0.96	0.94	0.95	3313
X	0.64	0.15	0.24	47
_	0.97	0.96	0.96	388
Accuracy	0.95 (30,760)			
Macro Avg	0.92	0.88	0.89	30,760
Weighted Avg	0.95	0.95	0.95	30,760

8.22 Precision-Recall AUC for Each Class

The Precision-Recall AUC provides a comprehensive view of the model's performance for each class, especially in the context of imbalanced data. The following figure illustrates the Precision-Recall curves and AUC scores for each class after running the Stacked CNN model.

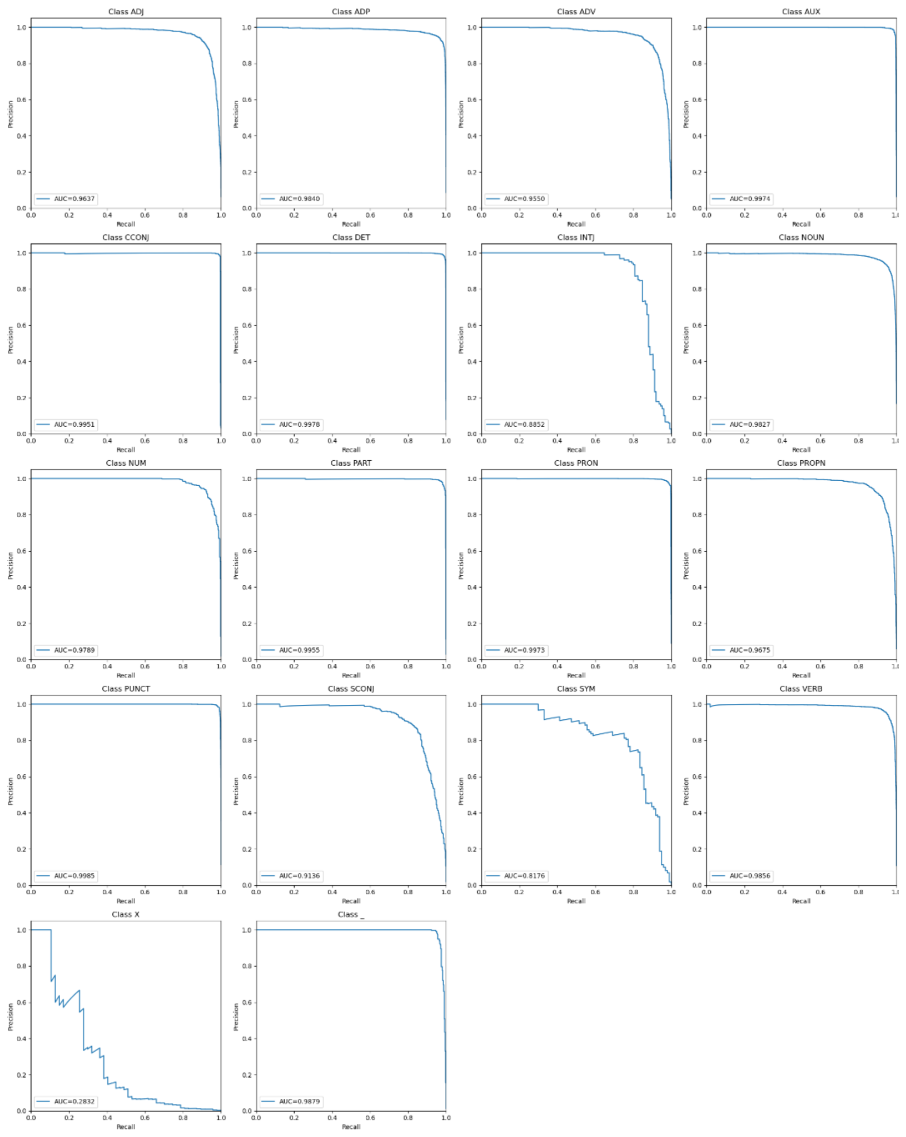


Figure 14: Precision-Recall AUC for Each Class

Table 10: Precision-Recall AUC for Each Class

Class	Precision-Recall AUC
ADJ	0.9637
ADP	0.9840
ADV	0.9550
AUX	0.9974
CCONJ	0.9951
DET	0.9978
INTJ	0.8852
NOUN	0.9827
NUM	0.9789
PART	0.9955
PRON	0.9973
PROPN	0.9675
PUNCT	0.9985
SCONJ	0.9136
SYM	0.8176
VERB	0.9856
X	0.2832
_	0.9879

8.23 Insights from Precision-Recall AUC

The Precision-Recall AUC scores reveal the following insights:

- Classes such as **AUX**, **PRON**, and **VERB** achieved high AUC scores, indicating the model's strong ability to distinguish these classes effectively.
- The **X** class had a notably low AUC score, highlighting the model's difficulty in identifying this rare class, which could be attributed to insufficient representation in the training data.
- Most major classes like **NOUN**, **ADJ**, **ADP**, and **DET** achieved high AUC scores, demonstrating the model's overall strong performance across common POS tags.

8.24 Insights and Discussion

The Stacked CNN demonstrated significant improvement over both baseline models. The MLP achieved notable performance gains; however, the CNN showed further enhancements across most metrics, except for a few classes like **X** and **SYM**. The learning curves indicate that the CNN effectively learned from the training data, although there were signs of slight overfitting in some classes.

8.25 Impact of Data Preprocessing

Extensive preprocessing steps, including text normalization, tokenization, and lemmatization, contributed to the improved performance of the CNN classifier. By removing noise and standardizing the text data, the model could focus on the most relevant features for POS classification.

8.26 Conclusion

The Stacked CNN for POS tagging demonstrated superior performance compared to baseline models, achieving a test accuracy of 0.95. The model effectively learned to classify POS tags, leveraging n-gram filters and residual connections to capture contextual information. Future work could explore more advanced architectures and larger datasets to further enhance performance. Additionally, incorporating character-level embeddings and experimenting with other regularization techniques could yield even better results.