VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Natural Language Processing Assignment

Sentiment Analysis

An experiment on CNN and LSTM models

Semester: 232

Advisors: Prof. Quan Thanh Tho
Students: Ta Gia Khang - 2152642

HO CHI MINH CITY, MAY 2024

Contents

1 Motivation

The swift expansion of web-based applications, like social media platforms and blogs, has led to an abundance of comments and reviews on everyday activities. Sentiment analysis, which involves gathering and scrutinizing people's opinions, thoughts, and impressions on a wide range of topics, products, subjects, and services, has become crucial. These opinions offer valuable insights for corporations, governments, and individuals to gather information and make informed decisions. Employing natural language processing and text mining, sentiment analysis discerns and extracts subjective information from textual data. This article provides a comprehensive overview of the methodology involved in this process. It then assesses, contrasts, and explores various approaches to gain a nuanced understanding of their pros and cons. Finally, it delves into the challenges of sentiment analysis, aiming to chart potential future paths.

To summary, sentiment analysis is a multiclass classification task, which is described as below figure:

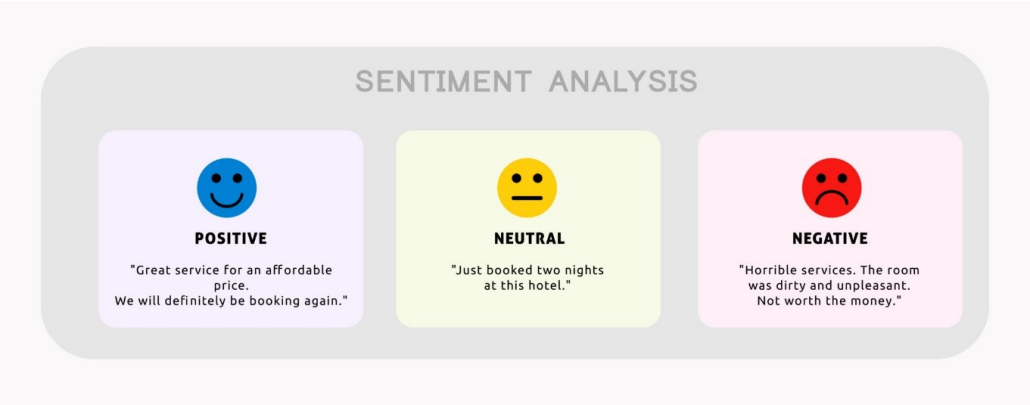

Figure 1: A simple description for sentiment analysis

Our problem can be summarized as:

- **Input:** a sentence (document).
- **Output:** the emotional class of that sentence.

2 Background

2.1 Convolution 1D layer and Convolution Neural Networks

Convolutional Neural Networks (CNNs) are networks initially created for image-related tasks that can learn to capture specific features regardless of locality.

For a more concrete example of that, imagine we use CNNs to distinguish pictures of Cars vs. pictures of Dogs. Since CNNs learn to capture features regardless of where these might be, the CNN will learn that cars have wheels, and every time it sees a wheel, regardless of where it is on the picture, that feature will activate.

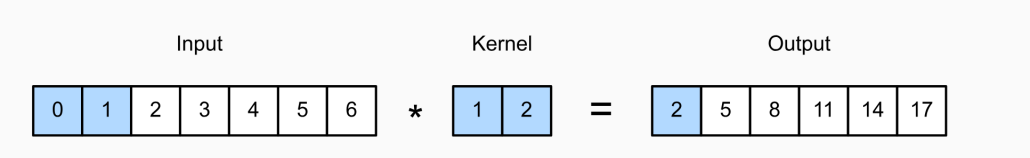

Figure 2: One-dimensional cross-correlation operation. The shaded portions are the first output element as well as the input and kernel tensor elements used for the output computation: 0x1+1x2=2

In our particular case, in the field of Natural Language Processing, a 1D Convolution layer could capture a negative phrase or known as “latent features” such as "don’t like" regardless of where it happens in the tweet. For example:

- I **don’t like** watching those types of films
- That’s the one thing I really **don’t like**.
- I saw the movie, and I **don’t like** how it ended.

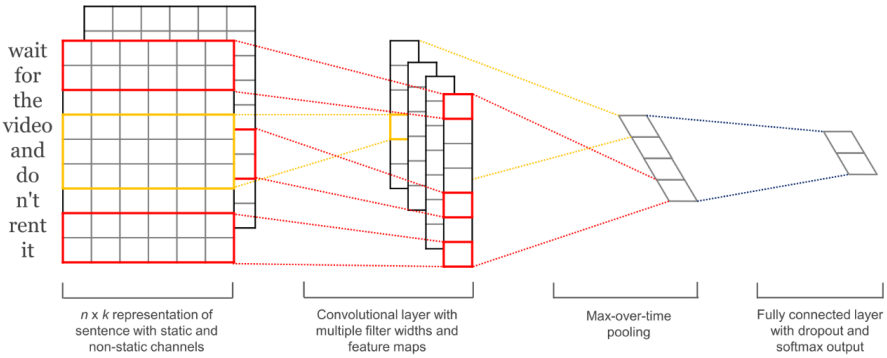

Figure 3: An example of using Conv1D layer in a neural network

In practice, more than one Conv1D filter is used in a layer in order to capture more latent features from our samples.

2.2 LSTM and LSTMs Neural Networks

Long-Short Term Memory (LSTMs) is a type of recurrent neural network that has a memory that "remembers" previous data from the input and makes decisions based on that knowledge. These networks are more directly suited for written data inputs, since each word in a sentence has meaning based on the surrounding words (previous and upcoming words).

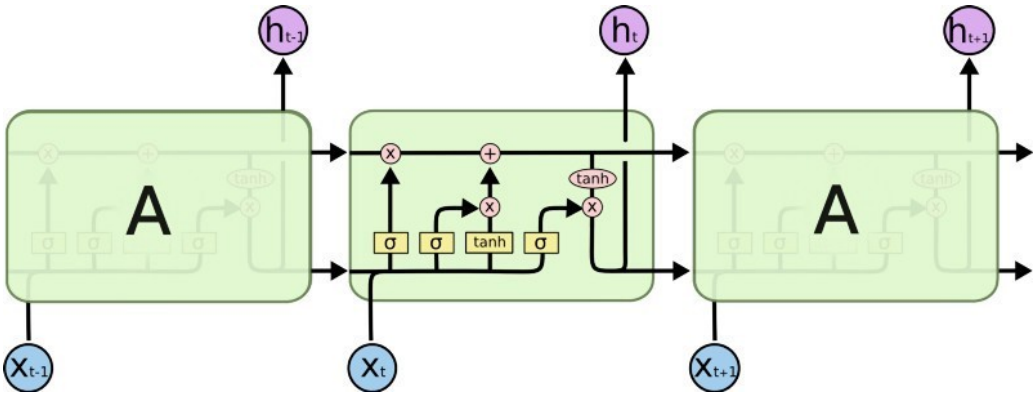

Figure 4: The Long Short-Term Memory (LSTM) cell can process data sequentially and keep its hidden state through time

In our particular case, it is possible that an LSTM could allow us to capture changing sentiment in a sentence. For example, a sentence such as: *"Used to really like Samsung, but now discovered that Samsung's screen quality is deteriorating, so feeling a bit bored."*, has words with conflicting sentiments that would end-up confusing a simple Feed-Forward network. The LSTM, on the other hand, could learn that sentiments expressed towards the end of a sentence mean more than those expressed at the start.

3 Methodologies

3.1 Dataset analysis and Data preprocessing

3.1.1 Dataset

In this article we will use the dataset of the VLSP Contest, which include many comments for technology devices in social network. Each comment is labeled with a emotional class which is -1, 0, 1 for Negative, Neutral and Positive. This is some samples in our dataset:

	Class	Data
0	-1	Mình đã dùng anywhere thế hệ đầu, quả là đầy t...
1	-1	Quan tâm nhất là độ trễ có cao không, dùng thi...
2	-1	dag xài con cùi bắp 98k....pin trâu, mỗi tội đ...
3	-1	logitech chắc hàng phải tiền triệu trở lên dùn...
4	-1	Đang xài con m175 cùi mía , nhà xài nhiều chuộ...

Figure 5: The first 5 rows in our used dataset

In total this dataset contains 5100 labeled comments in training set and 1050 for test set. More specifically, the below table will show the number of samples for each label in our training dataset:

Positive	Neutral	Negative	Total
1700	1700	1700	5100

And this below table show the number of samples for each label in our test set:

Positive	Neutral	Negative	Total
350	350	350	1050

3.1.2 Data preprocessing

To achieve higher accuracy and get more convenient in computation, some preprocessing method has been used as:

- The initial dataset cannot be used immediately because the data distribution is not good (group label together), so we have to do one more step: **Shuffle** the dataset using `Sklearn shuffle function` to create random distribution of data.
- Change label into one-hot: Initially our class is in the set of -1, 0, 1, for computation purpose, we change each value into one-hot vector, which is easier to compute loss later.
- We used PyVi, a crucial package specifically designed for processing Vietnamese text data. The library is equipped with multiple features that render it pertinent to the project at hand. And we chose **ViTokenizer** as our Tokenizer for this experiment. You can take a look at this package **here**.
- For the Word-Embedding step, we used the given vi-model-CBOW with `EMBEDDING DIMENSION = 400`. Word embeddings are essentially vector representations of words that capture their semantic content. This is achieved by placing semantically similar words closer in the vector space, thereby allowing algorithms to understand and leverage these semantic relationships.

To complete the experiments, we used **TensorFlow** framework and its APIs for model implementing purpose and do some experiment with those models.

3.2 Classification Head

For this multiclass classification task, we've experimented with several models. Ultimately, we opted to use the same classification head for all, ensuring a fairer comparison. Our chosen configuration involves one hidden layers, each with 32 units, and we've employed the ReLU activation function. The output layer consists of three units, activated by Softmax. Additionally, to prevent overfitting, we applied L2 regularization with a coefficient of 0.01 to each layer.

3.3 CNN model

In the context of sentiment analysis, the CNN typically starts with an embedding layer where words are transformed into vector representations. We use the embedding layer that have been mentioned in the **3.1**. Following this, multiple convolutional layers and pooling layers are applied. Each convolutional layer uses multiple filters that slide across the input text to compute dot products. This operation enables the model to automatically and adaptively learn spatial hierarchies of features, capturing important local and global semantics of the sentence.

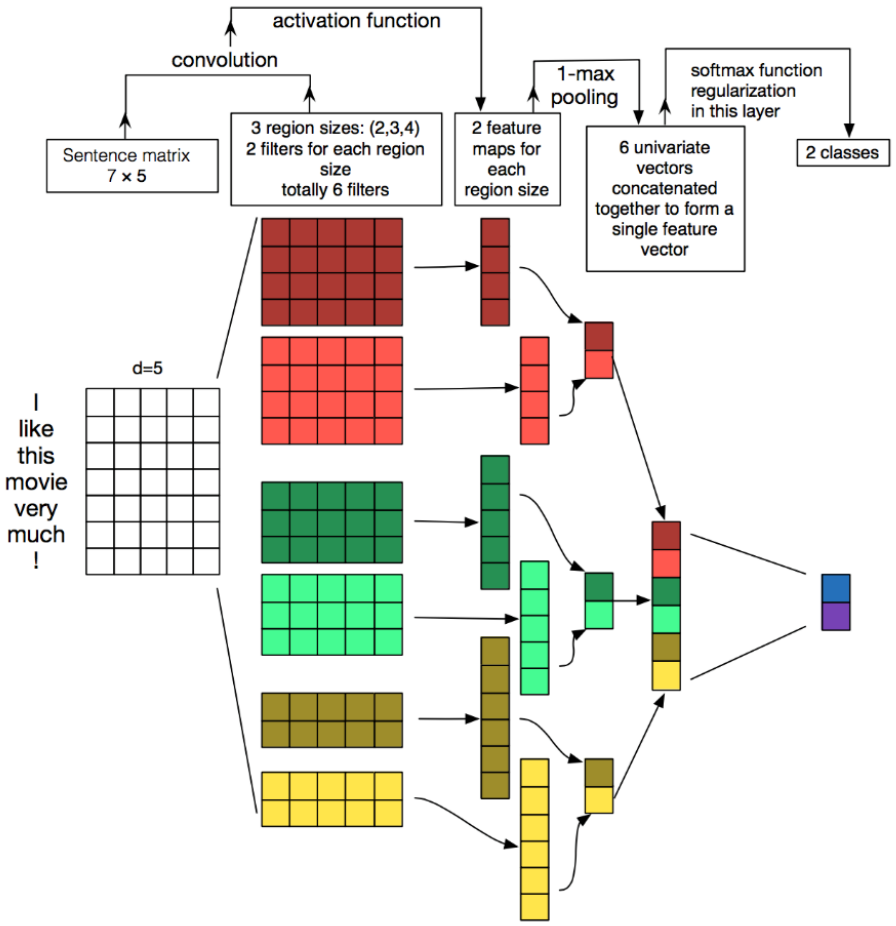

Figure 6: The proposed idea for the CNN-only model

The pooling layers serve to reduce the spatial size of the representation, reducing the computational complexity and helping to prevent overfitting. Max pooling is a commonly used technique, which retains the maximum value from each of the filter's output, providing a fixed-size output regardless of the input length.

Finally, the output from the pooling layers is concatenated and flattened, then fed into the fully connected layers, with the last layer performing the classification. For sentiment analysis, the final layer would typically use **softmax** activation function to output the probabilities of each sentiment class.

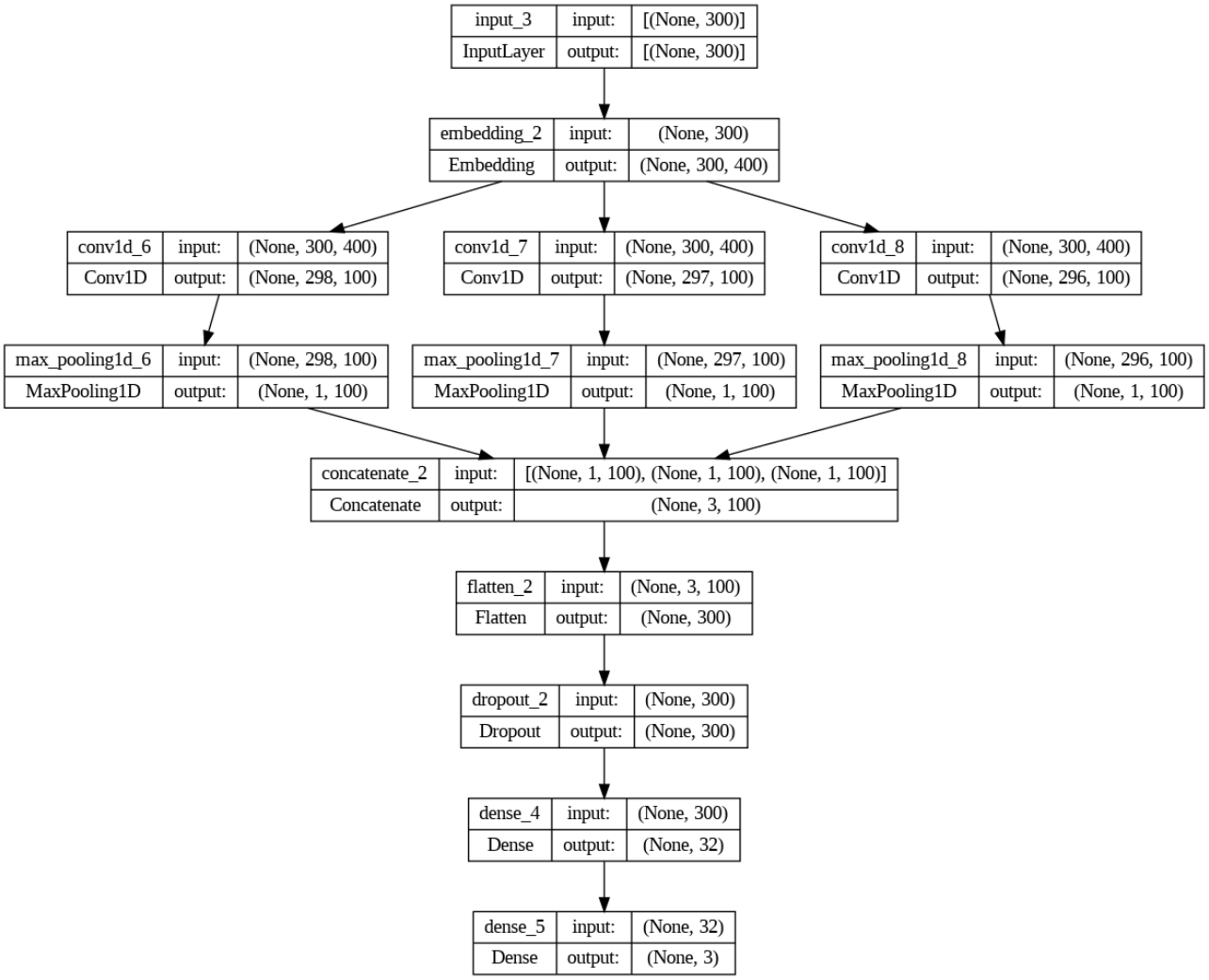

Figure 7: The implemented network in TensorFlow

You can have a look at our implementation via **this link**.

3.4 LSTM model

An LSTM network for sentiment analysis typically starts with an embedding layer where words are converted into vector representations. These vector representations can be learned while training the model, or pre-trained word embeddings can be used.

The LSTM layer then processes these sequences of vectors, maintaining information about the dependencies between the words in the sequence. The LSTM layer consists of multiple memory cells that keep track of dependencies in the input sequence. These memory cells include a forget gate, an input gate, and an output gate. The forget gate decides what information to discard from the cell state, the input gate decides what new information to store in the cell state, and the output gate determines what the next hidden state should be.

The output from the LSTM layer(s) is then typically passed to one or more fully connected layers, with the final layer using a softmax activation function for multi-class sentiment classification.

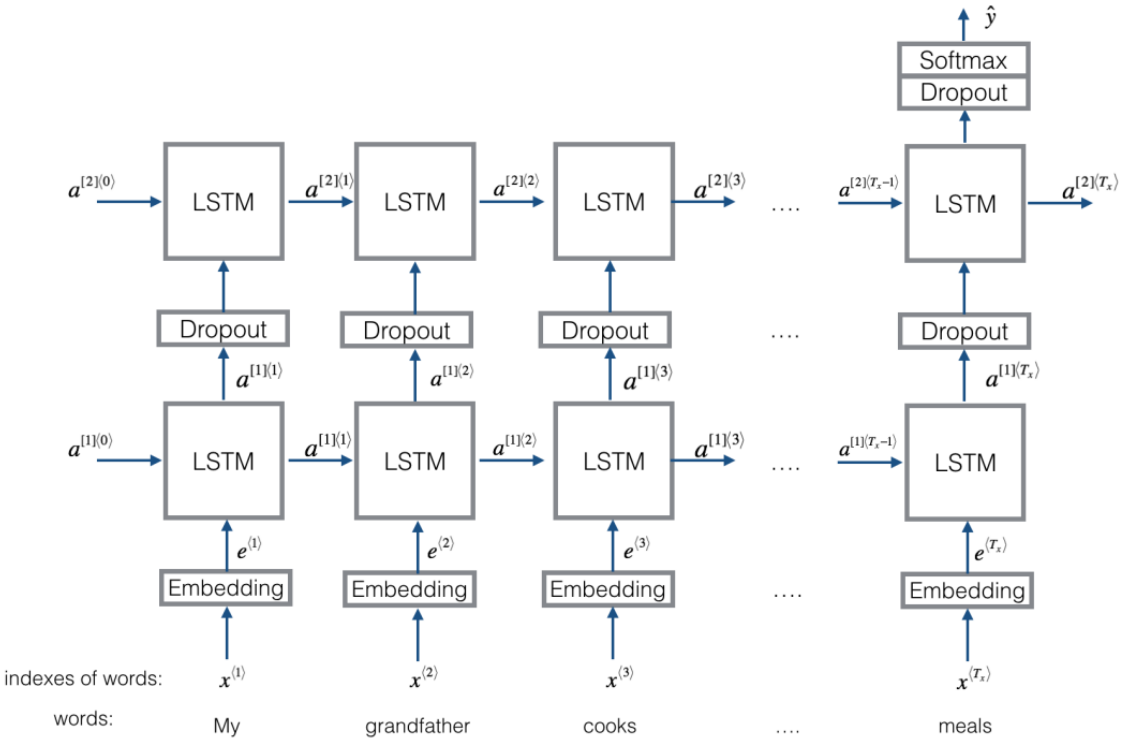

Figure 8: Our proposed model using LSTM

In this article we will use the Bidirectional LSTM, which gives us better accuracy result. Howerver, there are many points we need to note:

- The main advantage of a bidirectional LSTM is that it can learn positional features from both side
- The disadvantage of a bidirectional LSTM is that it needs more parameter and compute longer

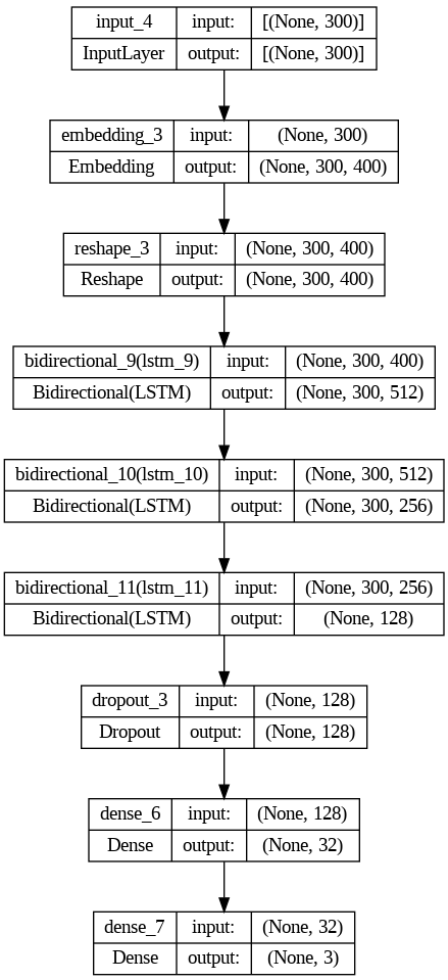

Figure 9: The implemented network in TensorFlow

You can have a look at our implementation via **this link**.

3.5 CNN-LSTM model

The CNN-LSTM model combination consists of an initial convolution layer which will receive word embeddings as input. Its output will then be pooled to a smaller dimension which is then fed into an LSTM layer. The intuition behind this model is that the convolution layer will extract local features and the LSTM layer will then be able to use the ordering of said features to learn about the input's text ordering.

Our first CNN-LSTM model will include an 1D-Convolution layer, then its output will be max-pooled then go into the LSTM layer. You can have a look at our implementation via **this link**.

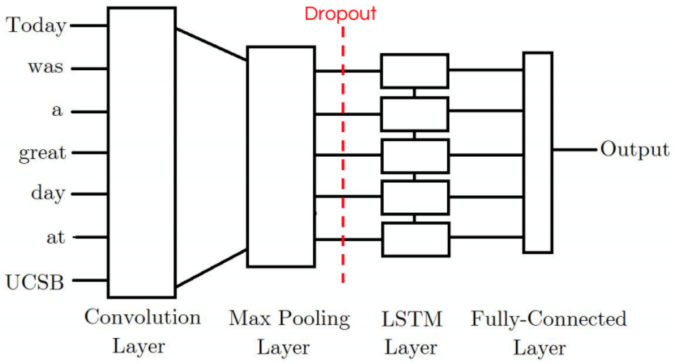

Figure 10: The simple network with CNN and LSTM

Moreover, we try to combined the proposed CNN model in **3.1** with the LSTMs, which resulted in a more complicated model. In addition, with the same training setting, this model shows the better result than the Simple CNN-LSTM.

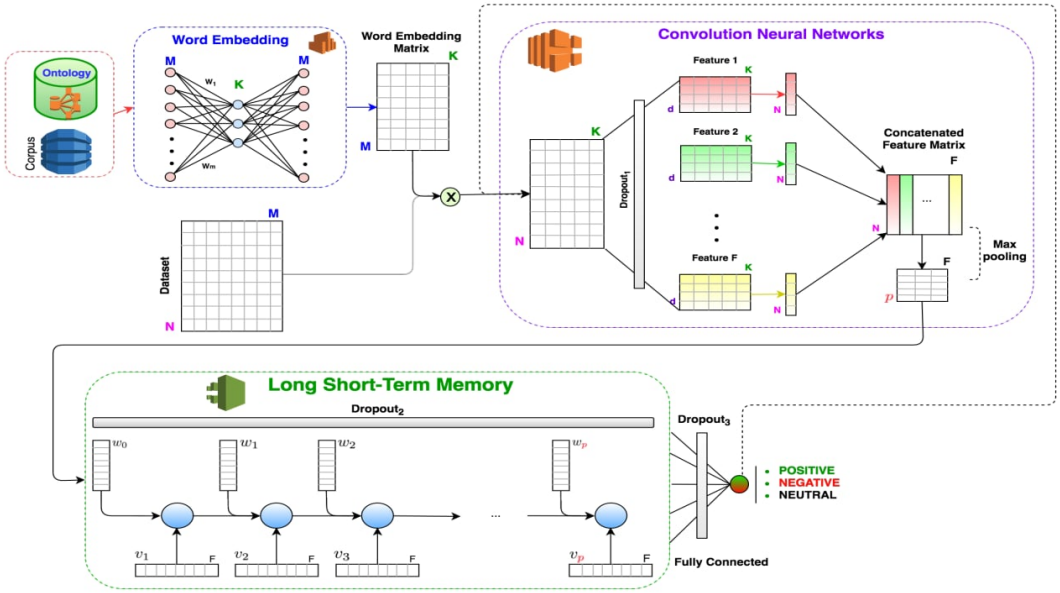

Figure 11: Our proposed architecture for CNN-LSTM with multi window size of Convolution layer

For a more detailed perspective on our implemented model, the following figure will illustrate it.

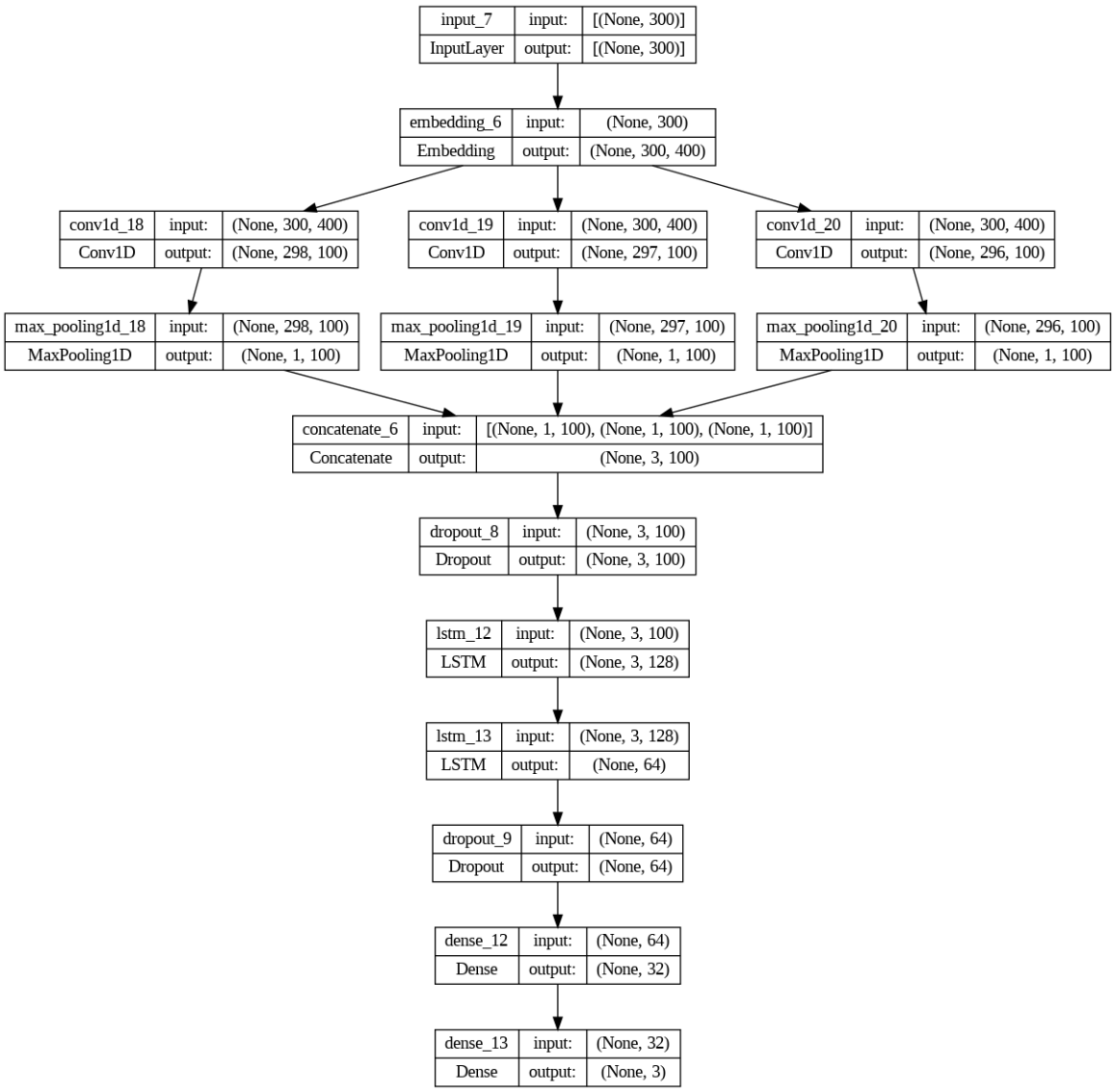

Figure 12: The implemented network in TensorFlow

You can have a look at our proposed CNN-LSTM model implementation via **this link**.

3.6 LSTM- CNN model

During our implementing process, we concluded that it's possible to combined those CNN-only and LSTM-only model in the above part to create a LSTM-CNN model.

Our LSTM-CNN model consists of an initial LSTM layer which will receive word embeddings for each token in the tweet as inputs. The intuition is that its output tokens will store information not only of the initial token, but also any previous tokens; In other words, the LSTM layer is generating a new encoding for the original input. The output of the LSTM layer is then fed into a convolution layer which we expect will extract local features. Finally the convolution layer's output will be pooled to a smaller dimension and ultimately outputted as either a positive or negative label.

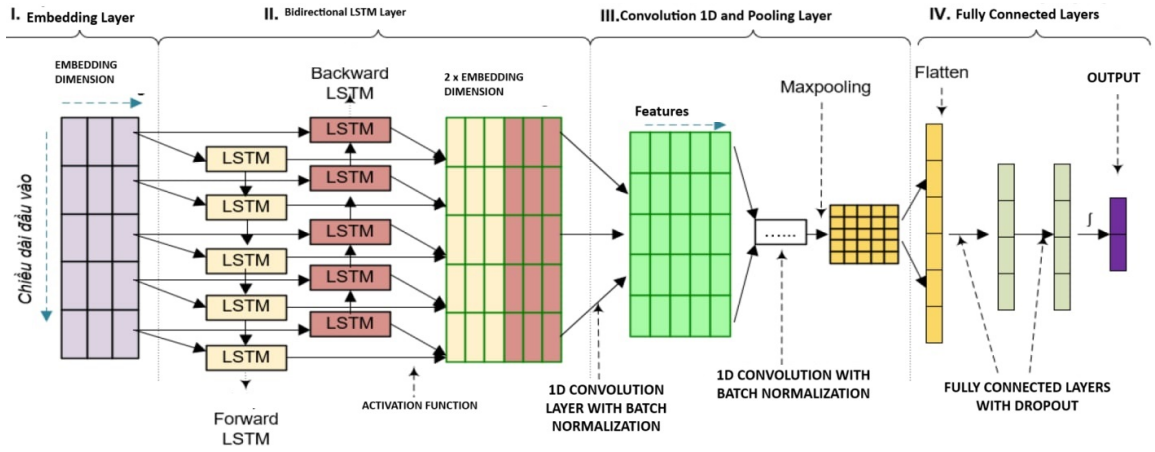

Figure 13: Our idea for LSTM - CNN architecture

For a more detailed perspective on our implemented model, the following figure will illustrate it.

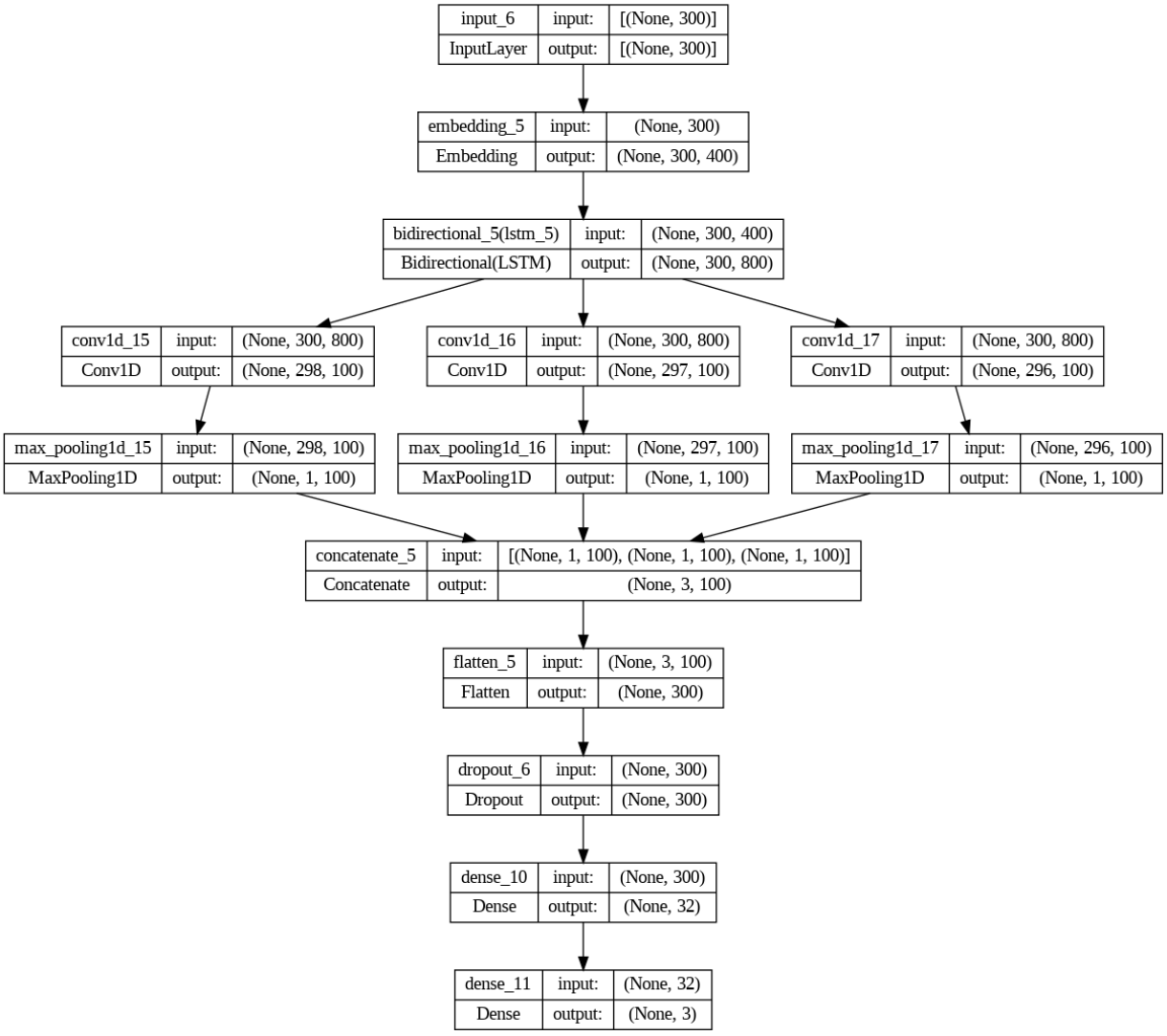

Figure 14: The implemented network in TensorFlow

You can have a look at our proposed LSTM-CNN model implementation via **this link**.

4 Experiment Results

4.1 Environment Setup and Hyper parameters tuning

For the fair comparison between the models, we have try many set of hyper-parameters and choose the most suitable set and used it to all our model. The below table shows our hyperparameter used for the models in this article.

Hyperparameter	Value
Embedding Dimension	400
Epoch	40
Batch Size	256
Filters	100
Kernel Size	3,4,5 for Convolution1D layer
Dropout	0.5

4.2 Evaluation

To evaluate the performance of the models, the chosen evaluation criteria are accuracy and loss. Accuracy is one of the most common evaluation metrics in the field of Natural Language Processing, especially in this project. The formula for calculating classification accuracy reflects the percentage of total words in the text that are correctly classified:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

Loss quantifies the discrepancy between the actual and predicted values produced by the machine learning model. Lower values of the loss function indicate a better fit of the model to the data, implying a more accurate representation of the underlying relationships.

Actual Accuracy: We have trained those above model, evaluating with the validation set and gain the result below:

Model	Accuracy
CNN	68.95
LSTM	65.9
Simple CNN-LSTM	63.52
CNN-LSTM	68.0
LSTM-CNN	68.19

5 Discussion

5.1 Learning rates

Firstly, we notice that CNN and CNN-LSTM model need more epoch to learn and overfit less quickly, as opposed to LSTM and LSTM-CNN models. This below table shows number of epochs and the models corresponding accuracy in our experiment:

Epochs	CNN	LSTM	CNN-LSTM	LSTM-CNN
3	35	59.3	52.1	63.52
20	67.1	65.52	65.24	65.9
40	68.95	65.9	68.0	68.19

This distinction in learning dynamics likely stems from architectural variances. CNNs excel in capturing spatial and temporal patterns but demand more data and training time. LSTMs, on the other hand, quickly grasp long-range dependencies but may overfit faster.

Combination models strive to blend these advantages while showcasing distinct learning patterns. The accuracy of CNN-LSTM reaches a plateau sooner than that of LSTM-CNN, hinting at potential constraints in fully leveraging temporal dependencies.

In the other hand, to handle the overfitting in those models, we used Early Stopping technique with patience = 15.

5.2 Drop rates

We also notice that it is important to add a Dropout layer after any Convolutional layer in both CNN-LSTM and LSTM-CNN model to enhance better result.

Dropout Prob.	**CNN-LSTM**	**LSTM-CNN**
20 percent	58.4%	59.1%
50 percent	68.0%	68.19%
98 percent	51.7%	48.1%

Table 1: *Dropout probability and the corresponding result for each model*

Adding a Dropout layer after Convolutional layers in both CNN-LSTM and LSTM-CNN models is crucial for enhancing performance. Dropout helps prevent overfitting by introducing randomness during training, encouraging the network to learn more robust features that generalize better to unseen data. This simple addition improves model generalization, mitigates overfitting, and enhances overall performance.

6 Conclusion

We have experimented with some models using for text classification, which come from CNNs, LSTMs, and combining CNN and LSTM architectures. For text classification harnesses LSTM's capacity to retain contextual understanding over lengthy texts while leveraging CNN's ability to extract local textual features. We conducted experiments on the given dataset and verified that the models can achieve good accuracy with the validation set. However, empirical evidence indicates that the performance gain from this fusion model is not substantial and may sometimes even underperform compared to a basic LSTM model.

As text length and complexity increase, text classification becomes more challenging. To address this, some modern technique as Attention mechanisms in Deep Learning models can significantly enhance classification efficacy, particularly for extended text sequences.

In terms of future work, we would like to test other types of technique as Attention or Self-attention and see what effects this has on the accuracy of our systems. It would also be interesting to find a better way to deal with misspellings or other irregularities found on Vietnamese Language. We believe this could be achieved by getting a better Vietnamese Word-embedding.

7 Related Works

- Pedro M Sosa, Shayan Sadigh, Twitter Sentiment Analysis with Neural Networks, (2018).
- Deep Sentiment Analysis Using CNN-LSTM Architecture of English and Roman Urdu Text Shared in Social Media by Lal Khan, Ammar Amjad, Kanwar Muhammad Afaq and Hsien-Tsung Chang.

Acknowledgement

This work was an assignment of us in the course of Natural Language Processing in Semester 232, thank you Prof. Tho Quan for giving us some materials to complete the experiment.

- **Video Presentation: here**.
- All of the source code is upload: **here**.

References

[1] Lei Zhang, Shuai Wang, Bing Liu, Deep Learning for Sentiment Analysis: A Survey

[2] Devlin, J., Chang, M., Lee, K., and Toutanova, K. BERT: pre-training of deep bidirectional transformers for language understanding. CoRR abs/1810.04805 (2018).

[3] Kim, Y. Convolutional neural networks for sentence classification. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP) (Doha, Qatar, Oct. 2014), A. Moschitti, B. Pang, and W. Daelemans, Eds., Association for Computational Linguistics, pp. 1746–1751.

[4] Wang, J., Yu, L.-C., Lai, K. R., and Zhang, X. Dimensional sentiment analysis using a regional cnn-lstm model. In Proceedings of the 54th annual meeting of the association for computational linguistics (volume 2: Short papers) (2016), pp. 225–230.

[5] Pedro M Sosa, Shayan Sadigh, Twitter Sentiment Analysis with Neural Networks, (2018).