Tribhuvan University

Institute of Science and Technology

Seminar Report

On

"Comparative Analysis of CNN Variants for Fake News Classification"

Submitted to

Central Department of Computer Science and Information Technology

Tribhuvan University, Kirtipur

Kathmandu, Nepal

Submitted by

Santosh Neupane

Roll no. 04/2079

In partial fulfillment of the requirement for Masters Degree in Information technology (MIT), 2nd semester

Tribhuvan University

Institute of Science and Technology

Supervisor's Recommendation

I hereby recommend that this seminar report, prepared under the supervision of Jagdish Bhatta entitled **'Comparative Analysis of CNN Variants for Fake News Classification'** be accepted as a fulfillment of a partial requirement of the degree of Masters in Information Technology.

.....

Asst. Prof. Jagdish Bhatta

(Supervisor)

Central Department of Computer Science and Information Technology

Letter of Approval

This is to certify that the seminar report prepared by Mr. Santosh Neupane entitled **"Comparative Analysis of CNN Variants for Fake News Classification"** in partial fulfillment of the requirements for the degree of Masters in Information Technology has been well studied. In our opinion, it is satisfactory in the scope and quality of a project for the required degree.

Evaluation Committee

.....
Asst. Prof. Sarbin Sayami
(H. O. D)
Central Department of Computer Science and Information Technology

.....
Asst. Prof. Jagdish Bhatta
(Supervisor)
Central Department of Computer Science and Information Technology

.....
(Internal)

Acknowledgement

I would like to express my sincere gratitude to **Asst. Prof. Mr. Jagdish Bhatta,** for his valuable guidance in carrying out this work under his supervision. I am grateful for his excellent guidance, trust and corrections to my seminar work.

With immense pleasure, I would like to thank **Asst. Prof. Mr. Sarbin Sayami,** Head of Central Department of Computer Science and Information Technology, Tribhuvan University for his encouragement in the completion of the seminar.

At last, but not least, with immense pleasure, I submit my deepest gratitude to the Central Department of Computer Science and Information Technology, Tribhuvan University, and all the faculty members of CDCSIT for providing the platform to explore the knowledge of interest.

Santosh Neupane (04/079)

Abstract

The explosion of social media allowed individuals to spread information without cost, with little investigation and fewer filters than before. This amplified the problem of fake news, which has become a major concern nowadays due to the negative impact it brings to the communities. To tackle the rise and spread of fake news, automatic detection techniques have been researched building on artificial intelligence and machine learning. The recent achievements of deep learning techniques in complex natural language processing tasks, make them a promising solution for fake news detection too. This work researches a hybrid deep learning model that combines convolutional and recurrent neural networks and attention weights for fake news classification. The baseline CNN model and CNN-LSTM showed similar precision, recall, and F1-score metrics with 96% on all metrics, while CNN-Attention has a slight increase in precision, recall, and F1-score with 97% on all metrics. CNN model and CNN-LSTM showed an accuracy of about 96% and CNN-Attention showed an accuracy of 97.14%. The confusion matrix showed CNN-Attention has the highest true positives while baseline CNN has the highest true negative predictions.

Keywords: CNN, CNN-LSTM, CNN-Attention, fake news detection

Contents

Acknowledgement i

Abstract ii

List of Abbreviations v

List of Figures vi

List of Tables vii

1 Introduction **1**

1.1 Introduction 1

1.2 Problem Statement 2

1.3 Objective 2

2 Previous Works, Discussions, and Findings **3**

2.1 Background Study 3

2.1.1 CNN 3

2.1.2 CNN-LSTM 4

2.1.3 CNN-Attention 5

2.2 Literature Review 5

2.3 Methodology 7

2.3.1 Dataset Description 8

2.3.2 Data Cleaning and Preprocessing 8

2.3.3 Tokenization and Sequencing 8

2.3.4 Padding and Embedding 9

2.3.5 Train-Test split 9

2.3.6 CNN and It's Variants 9

2.3.7 Evaluation 15

2.4 Implementation 16

2.4.1 Implementation Tools 16

2.4.2 Implementation Details 16

2.5 Findings and Result 22

2.5.1 Confusion Matrix 22

2.5.2 Accuracy Parameters 24
2.5.3 Overfitting and Generalization 25
2.5.4 Result Analysis 26

3 Conclusion **27**

3.1 Conclusion 27
3.2 Future Recommendation 27
References 28

List of Abbreviations

CNN Convolutional Neural Network

GloVe Global Vectors for Word Representation

IDE Integrated Development Environment

LSTM Long Short-Term Memory

ML Machine Learning

NLP Natural Language Processing

NLTK Natural Language Toolkit

RNN Recurrent Neural Network

List of Figures

2.1 CNN architecture 3
2.2 CNN-LSTM 4
2.3 CNN model workflow 7
2.4 Dataset structure 8
2.5 Convolutional Neural Network (CNN) model architecture 11
2.6 CNN-LSTM model architecture 12
2.7 CNN-Attention model architecture 14
2.8 Python code for data cleaning and lemmatization 17
2.9 Python code for sequencing and padding text 18
2.10 Python code for generating embedding vector with GloVe 18
2.11 CNN implementation 19
2.12 CNN-LSTM implementation 20
2.13 Attention implementation 21
2.14 CNN-Attention implementation 21
2.15 Code for confusion matrix 22
2.16 Confusion matrix of baseline CNN 23
2.17 Confusion matrix of CNN-LSTM 23
2.18 Confusion matrix of CNN-Attention 24
2.19 Training and validation accuracy over epochs for CNN, CNN-LSTM, and CNN-Attention 25
2.20 Training and validation loss over epochs for CNN, CNN-LSTM, and CNN-Attention 25

List of Tables

2.1 Accuracy parameters of CNN variants 24

Chapter 1

Introduction

1.1 Introduction

Fake news refers to misinformation or false information presented as news with the intent to deceive. In the digital age, online platforms have enabled journalists and non-journalists to reach a massive audience, disseminating information including articles, claims, statements, posts, and other types of content related to public figures. This democratization of information sharing raises significant social concerns regarding the authenticity and intentions behind the news. Fake news can have severe real-world consequences, such as inciting violence, manipulating stock markets, and exacerbating political issues.

Detecting fake news is a critical application of Natural Language Processing (NLP). Automated fake news detection is the task of assessing the truthfulness of claims in news [1]. To prevent the spread of fake news, it is essential to adopt effective strategies. Deep learning techniques, in particular, show great promise in fake news detection. This paper explores using Convolutional Neural Network (CNN) and its variants to classify news articles. The primary objective of fake news detection is to accurately identify and distinguish fake news from genuine news.

CNNs are widely used as they succeed in many text classification tasks. It is used for extracting features with a variety of metadata. CNN-LSTM is a variant that uses both CNN and Long Short-Term Memory (LSTM) model, where CNN is used for feature extraction and LSTM for sequence modeling. Another powerful variant involves the integration of attention mechanisms with CNNs. The attention mechanism helps the model focus on the most crucial parts of the text, thereby improving the overall performance in text classification tasks. By selectively emphasizing significant features, the model can better understand the context and nuances within the text, leading to more accurate fake news detection.

This research aims to compare various CNN-based models and analyze their performance in the task of fake news classification. We delve into the specifics of each variant, evaluating their strengths and limitations. By conducting a comprehensive analysis, we aim to identify the most effective model for detecting fake news, contributing to the broader effort of combating misinformation.

1.2 Problem Statement

Spreading misinformation through fake news presents a challenge for human society. It causes negative impacts in social, political, and economic sectors and is a major threat to democracy. Early detection of fake news possibly at its source can help to prevent damages that can be caused by it. In the age of social media, it's more difficult to tackle the spread and impact of fake news. An efficient algorithm needs to be explored so that it becomes difficult to spread fake news. Identifying the vocabulary that is used to mislead readers is the essential task of identifying fake news. A difficult challenge is classifying news through word-level context. As a result, the purpose of this research is to use hidden patterns in news text to detect fake news.

1.3 Objective

The objective of this seminar is

- To compare the efficiency and performance of three CNN architectures: Baseline CNN, CNN-LSTM, and CNN-Attention in the context of fake news detection, utilizing a comprehensive dataset.

Chapter 2

Previous Works, Discussions, and Findings

2.1 Background Study

2.1.1 CNN

A CNN is a specialized artificial neural network designed primarily for processing structured grid-like data, such as images. Unlike traditional neural networks, CNNs are specifically tailored to capture spatial hierarchies and patterns in data through a series of convolutional layers, pooling layers, and fully connected layers.

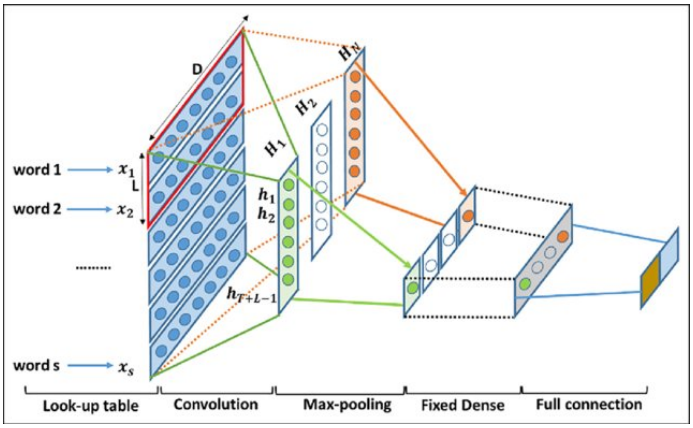

Figure 2.1: CNN architecture

1. **Convolutional layer:** The convolutional layer applies convolutional operations to sequential data, such as sentences or documents. The input is typically a matrix where each row represents a word encoded as a vector, often using word embeddings like Word2Vec, Global Vectors for Word Representation (GloVe). The convolutional layer uses filters (kernels) of various widths that slide over the input text to capture local patterns and n-grams.

2. **Pooling layer:** Pooling layers, such as max pooling, are often used to downsample the feature maps, reducing their dimensionality while retaining the most critical features. For text, max pooling is commonly used to capture the most important feature (e.g., the highest value) from each feature map, regardless of its position.

3. **Fully connected layers:** These layers, typically found at the end of the network, connect every neuron in one layer to every neuron in the next layer. They are responsible for combining the extracted features to make the final classification or prediction.

2.1.2 CNN-LSTM

Combining CNN with LSTM networks is a powerful technique for handling data with spatial and temporal dependencies. The CNN-LSTM architecture leverages the strengths of both CNNs and LSTMs to handle complex data with spatial and temporal characteristics. The output of the CNN layer (i.e. the feature maps) is the input for the Recurrent Neural Network (RNN) layer of LSTM units/cells that follow. The RNN layer uses the local features extracted by CNN and learns the long-term dependencies of the local features of news articles to classify them as fake or real. The proposed model is depicted in fig 2.2.

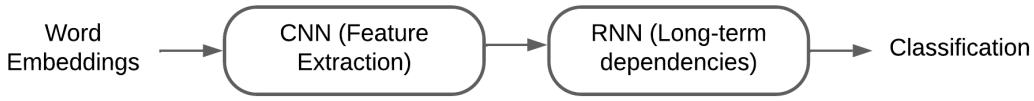

Figure 2.2: CNN-LSTM

The LSTM consists of the following control gates :

1. **Input gate:** The input gate determines which values from the input should be used to modify the memory. It utilizes a sigmoid function to decide which values to let through (ranging from 0 to 1) and a tanh function to assign weights to the values, indicating their level of importance.

2. **Forget gate:** The forget gate determines what information should be discarded from the memory cell. It also employs a sigmoid function, which looks at the previous state and current input, and outputs values between 0 (omit this information) and 1 (keep this information) for each element in the memory cell state.

3. **Output gate:** The output gate combines the input and memory of the LSTM block to determine the output. It uses a sigmoid function to decide which values to pass through (ranging from 0 to 1) and a tanh function assigns weights to the values,

indicating their level of importance. The weighted values are then multiplied with the output of the sigmoid function to generate the final output.

2.1.3 CNN-Attention

Combining CNN with attention mechanisms is a powerful approach to enhance the ability of CNNs to focus on the most relevant parts of the input data. Attention mechanisms enable models to focus on the most relevant parts of the input data, dynamically weighing different parts based on their importance.

1. **Attention layer:** The attention mechanism computes attention scores for the feature maps generated by the CNN. These scores indicate the relevance of each feature in the context of the classification task. The attention mechanism enables the model to focus on the most important parts of the text, enhancing its ability to capture global context and long-range dependencies. The attention scores are used to compute a weighted sum of the feature maps, producing a context vector that represents the most relevant features of the text.

2.2 Literature Review

Fake news detection has become an essential area of research due to the rapid spread of misinformation through social media and online news platforms. CNN, originally designed for image processing, has shown significant promise in text classification tasks, including fake news detection. This literature review explores key developments, methodologies, and findings in using CNNs for detecting fake news.

Yoon Kim's seminal paper applied CNNs to sentence classification tasks. The model used static and dynamic word embeddings, and multiple filter sizes to capture various n-gram features. The study showed that CNNs could capture important local features and patterns in text data, which are crucial for distinguishing between fake and real news [2]. Wang extended the application of CNNs to fake news detection by training a CNN model on the LIAR dataset. This study highlighted CNN's ability to learn intricate patterns in the text that are indicative of false or fake news, such as sensational language and specific word usage [3].

The study [4] proposed using CNNs directly on character-level text, bypassing the need for word embeddings. The model showed that character-level representations could be effective for text classification, especially in cases where word-level information might be sparse or noisy. Conneau and colleagues proposed using very deep CNNs with up to 29 convolutional layers. They demonstrated that deeper networks, along with residual connections, could capture more complex features and improve text classification accuracy [5].

Combining CNNs with RNNs leverages the strengths of both architectures: CNNs for local feature extraction and RNNs for capturing sequential dependencies. Lai et al. introduced a Recurrent Convolutional Neural Network (RCNN) for text classification, which improved performance by combining convolutional and recurrent layers [6]. The research by Nasir et al. shows the use of a hybrid CNN-RNN model on two fake-news datasets (ISO and FA-KES) with better detection results than other non-hybrid baseline models [7].

Attention mechanisms have been integrated with CNNs to enhance their ability to focus on relevant parts of the text. Yang et al. proposed a Hierarchical Attention Network (HAN) that applied attention at both word and sentence levels [8]. Attention mechanisms help in identifying key phrases and sentences that are indicative of fake news.

2.3 Methodology

The image below shows the general workflow of the CNN model for fake news classification. The process involves several steps, which are described below:

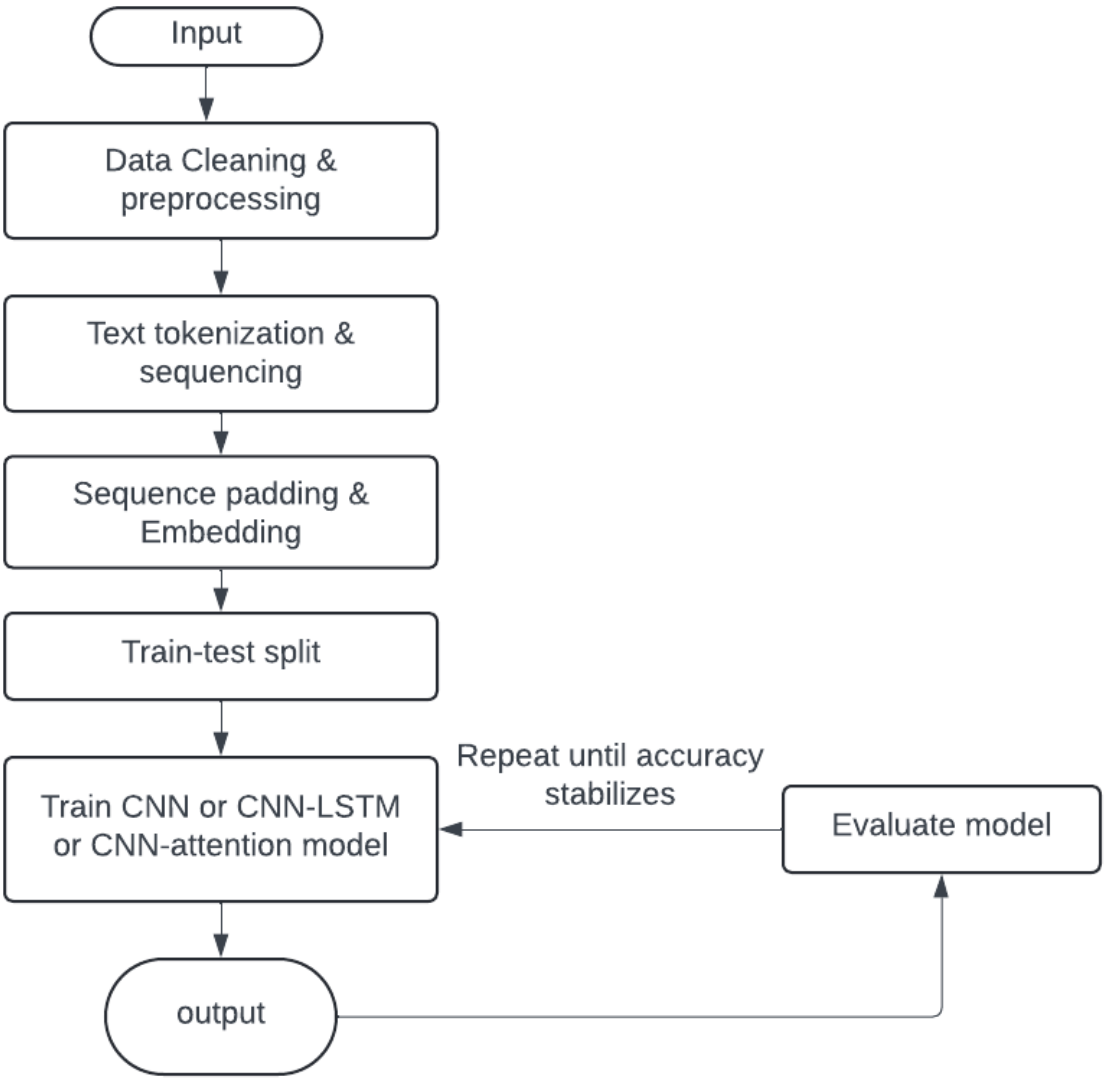

Figure 2.3: CNN model workflow

2.3.1 Dataset Description

Input Data for this model is taken from Kaggle [9], an online repository known for its diverse collection of datasets and Machine Learning (ML) competitions. The dataset consists of 20,761 entries, with 10,387 labeled as true news and the remaining 10,374 as fake news. In this research, the text field and label field of the data are used for training the model. The label has 1 for fake news and 0 for true news.

id	text	label
0	House Dem Aide: We Didn't Even See Comey's Let...	1
1	Ever get the feeling your life circles the rou...	0
2	Why the Truth Might Get You Fired October 29, ...	1
3	Videos 15 Civilians Killed In Single US Airstr...	1
4	Print \nAn Iranian woman has been sentenced to...	1

Figure 2.4: Dataset structure

2.3.2 Data Cleaning and Preprocessing

Data cleaning and preprocessing are done to reduce noise or irrelevant information like stopwords. It helps to standardize text by converting it to lowercase, which improves the performance of the model. The input data is passed to a function that converts text to lowercase, removes stopwords, and lemmatizes each word. Lemmatization is the process of reducing words to their base or root form, which helps in standardizing words for tasks like text analysis and NLP.

2.3.3 Tokenization and Sequencing

To transform the text in each entry before feeding it to the CNN model, the first step is to tokenize words. Tokenization is the process of breaking down text into smaller units called tokens. These tokens can be words, subwords, or characters. The goal of tokenization is to convert raw text into a format that can be numerically processed by a neural network.

Once tokenized, the tokens need to be converted into sequences of numerical values that the CNN can process.

2.3.4 Padding and Embedding

Padding is done to ensure all sequences have the same length. This is often done by padding shorter sequences with zeros or truncating longer sequences. Embedding in the context of NLP refers to the representation of words (or tokens) as dense vectors of real numbers. These vectors capture semantic meanings and relationships between words, enabling the use of words in machine learning models. Embeddings transform words into a numerical form that preserves syntactic and semantic properties, making them suitable for neural networks and other machine learning algorithms. A pretrained GloVe embedding is used. It is an unsupervised learning algorithm for obtaining vector representations for words. Unlike other word embeddings that are based solely on local context, GloVe uses global statistical information from a corpus to produce dense word vectors.

2.3.5 Train-Test split

The dataset is divided into training and testing subsets. The training subset is used to train the model while the testing subset is used to evaluate the trained model using metrics like accuracy score, precision, recall, etc. 70 percent of the dataset is used for training while 30 percent is used for testing. The number of epochs is adjusted during training to assess whether the model is overfitting or underfitting.

2.3.6 CNN and It's Variants

For binary classification problems such as distinguishing between Fake and True news, the mechanisms through which baseline CNN, CNN-LSTM, and CNN-Attention models process and classify news text data are elaborated upon here. The focus is on their capability to classify text sequences as either fake or true.

CNN model

CNNs are effective at recognizing local patterns in text, such as specific phrases or word combinations, which can be indicative of fake news. It can learn discriminative features from text and can analyze the sentiment of text by detecting overly sensational or biased

language often found in fake news.

The CNN model uses two convolution layers each followed by a dropout and max pooling layer. Dropout is used to avoid overfitting. CNN captures the spatial pattern on text sequences which are then passed to fully connected layers to perform classification as shown in figure 2.5.

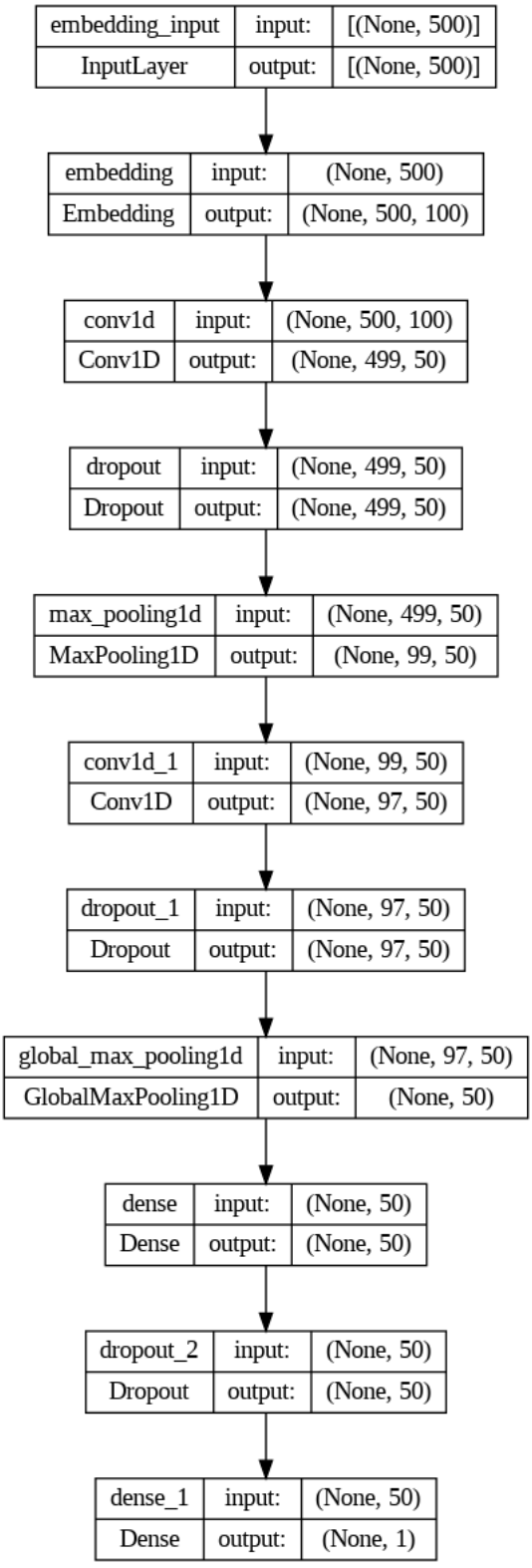

Figure 2.5: CNN model architecture

CNN-LSTM model

It is a hybrid model that uses CNN and LSTM layers. The pattern detected by CNN is passed to RNN/LSTM layer that learns the long-term dependencies of input text.

LSTMs are designed to capture long-range dependencies in sequences, making them well-suited for understanding context and nuances in news articles. It helps to model the relationship between words and sentences over the entire article making them suitable to use for fake news classification.

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 500, 100)	1000000
conv1d_2 (Conv1D)	(None, 499, 50)	10050
dropout_3 (Dropout)	(None, 499, 50)	0
max_pooling1d_1 (MaxPooling1D)	(None, 99, 50)	0
conv1d_3 (Conv1D)	(None, 97, 50)	7550
dropout_4 (Dropout)	(None, 97, 50)	0
max_pooling1d_2 (MaxPooling1D)	(None, 19, 50)	0
lstm (LSTM)	(None, 50)	20200
dense_2 (Dense)	(None, 50)	2550
dropout_5 (Dropout)	(None, 50)	0
dense_3 (Dense)	(None, 1)	51

Total params: 1040401 (3.97 MB)
Trainable params: 40401 (157.82 KB)
Non-trainable params: 1000000 (3.81 MB)

Figure 2.6: CNN-LSTM model architecture

CNN-Attention model

It is also a hybrid model which computes attention scores for the data generated by the CNN layer. This hybrid approach leverages the local feature extraction capabilities of CNNs and the context-awareness of attention mechanisms, making it highly effective for tasks like fake news detection.

The attention mechanism calculates attention weights that signify the importance of each word (or feature) in the context of the entire article. It provides contextual information to each feature or word, which is crucial for classifying news as fake or true. The context vector is the weighted sum of the input vectors, with weights determined by the attention mechanism. The attention mechanism itself involves computing intermediate scores using a learned weight matrix W and bias vector b, applying the tanh activation function, and normalizing these scores with the softmax function to obtain the attention weights.

$$\text{context} = \sum_i (\text{softmax}(\tanh(W \cdot x_i + b)) \cdot x_i)$$

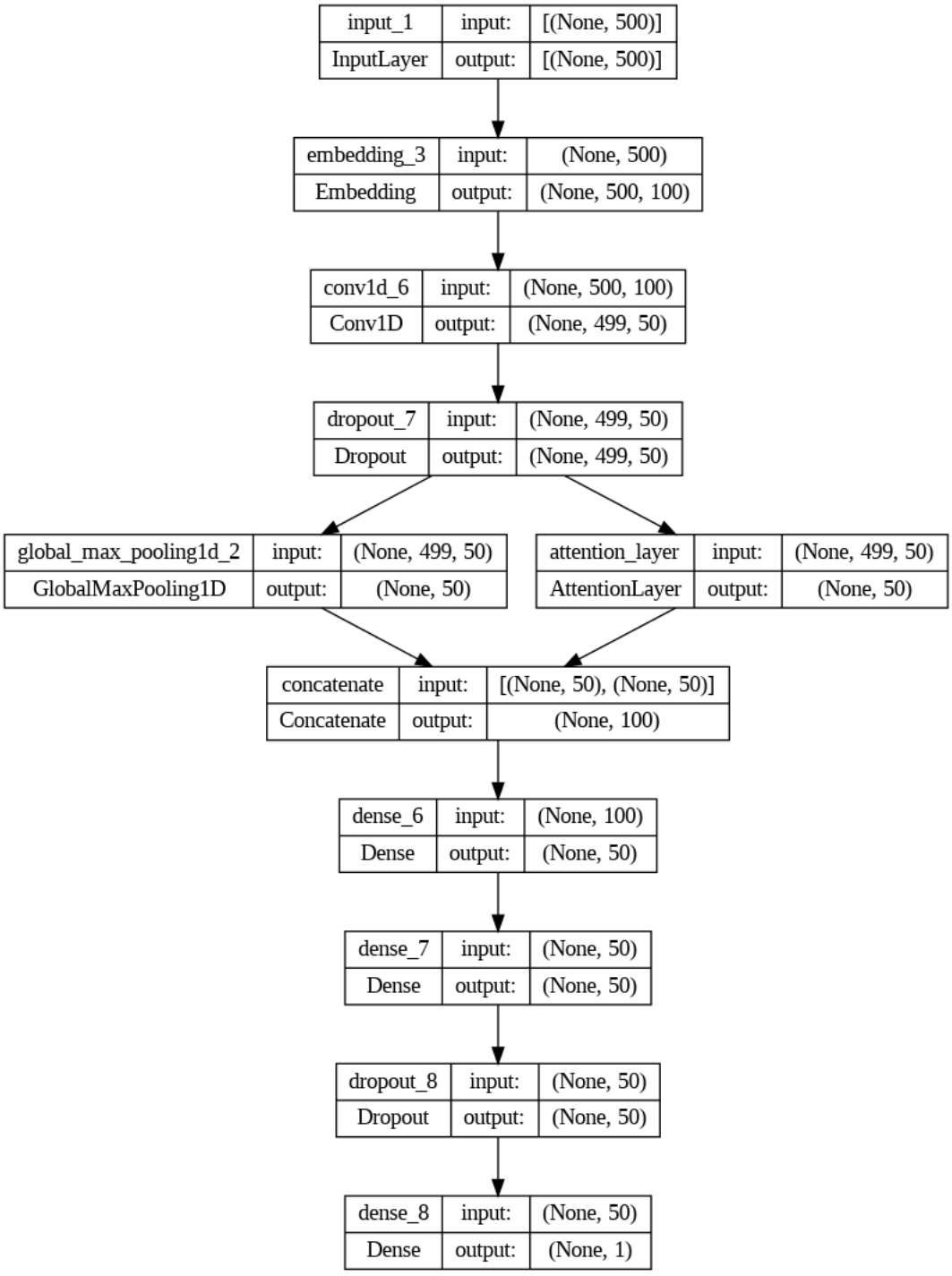

Figure 2.7: CNN-Attention model architecture

2.3.7 Evaluation

To determine the consistency and accuracy of a classification model, typical assessment metrics are determined according to,

Accuracy: Accuracy is the ratio of correctly predicted instances to the total instances. It gives an overall measure of how often the model is correct.

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Number of Instances}}$$

Precision: Precision (also called Positive Predictive Value) is the ratio of correctly predicted positive observations to total predicted positives. It tells you how many of the instances predicted as positive are positive.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

Recall: Recall (also called Sensitivity or True Positive Rate) is the ratio of correctly predicted positive observations to all observations in the actual class. It tells you how many of the actual positive instances were captured by the model.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

F1 score: The harmonic mean of recall and precision is the F1 Score. As a result, this score takes both false positives and false negatives into account. F1 score is more useful than accuracy when the class distribution is unbalanced.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

2.4 Implementation

2.4.1 Implementation Tools

Python is used as the programming language to code the program. Google Colab is used as an Integrated Development Environment (IDE). Similarly, different libraries are also used, such as:

Tensorflow: TensorFlow is used to build and train the CNN-based Fake News classifier.

Keras: Keras is used to simplify the process of building and training the CNN models.

NLTK: Natural Language Toolkit (NLTK) is a powerful library in Python for working with human language data. It provides easy-to-use interfaces and text-processing libraries for classification, tokenization, stemming, tagging, parsing, and more.

NumPy: NumPy is used to store and manipulate training and testing data.

Pandas: Pandas is used to load and clean training and testing data.

Matplotlib: Matplotlib is used to visualize the results of the Fake News classifier.

2.4.2 Implementation Details

Data Cleaning and Preprocessing

The train and test data are passed through a function that removes stopwords, and lowercase, and then lemmatizes each word for effective learning of the model. The NLTK library is used to perform data cleaning and preprocessing.

Text Sequencing and Padding

The text data were transformed into sequences of words using a tokenizer, and each sequence was padded with zeros to have a fixed length of 500 words. This step ensured that all sequences had consistent dimensions for efficient processing. The sequence length of 500 words is chosen based on the experiment on the model's performance.

```
# Initialize the lemmatizer
lemmatizer = WordNetLemmatizer()

# Function to get the part of speech tag for lemmatization
def get_wordnet_pos(word):
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ,
                "N": wordnet.NOUN,
                "V": wordnet.VERB,
                "R": wordnet.ADV}
    return tag_dict.get(tag, wordnet.NOUN)

def lemmatize(sentence):
  sentence = sentence.lower()

  # Tokenize the sentence
  words = word_tokenize(sentence)

  # Lemmatize each word
  lemmatized_words = [lemmatizer.lemmatize(word, get_wordnet_pos(word)) \
                      for word in words \
                      if not word in stopwords.words('english')]

  # Join the lemmatized words into a sentence
  lemmatized_sentence = ' '.join(lemmatized_words)

  return lemmatized_sentence
```

Figure 2.8: Python code for data cleaning and lemmatization

Embedding

The text sequences are converted to dense 100-dimensional vectors of real numbers using GloVe embedding as shown in figure 2.10. The embeddings are trained on very large text corpora which ensures embeddings generalize well to new unseen text. The pre-trained weights also help the model to train faster without compromising performance.

```
# Text preprocessing
max_words = 10000
max_sequence_len = 500

# Model parameters
embedding_dim = 100

tokenizer = Tokenizer(num_words=max_words, oov_token='<OOV>')
tokenizer.fit_on_texts(X_train)

X_train_sequences = tokenizer.texts_to_sequences(X_train)
X_test_sequences = tokenizer.texts_to_sequences(X_test)

X_train_padded = pad_sequences(X_train_sequences, maxlen=max_sequence_len, padding='post')
X_test_padded = pad_sequences(X_test_sequences, maxlen=max_sequence_len, padding='post')
```

Figure 2.9: Python code for sequencing and padding text

```
embeddings_index = dict()
f = open('glove.6B.100d.txt')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

embedding_matrix = np.zeros((max_words, 100))
for word, index in tokenizer.word_index.items():
    if index > max_words - 1:
        break
    else:
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[index] = embedding_vector
```

Figure 2.10: Python code for generating embedding vector with GloVe

CNN Architecture

Baseline CNN

The CNN model consists of an Embedding layer, and two convolutional layers followed by a pooling layer. The sequence of words is initially transformed into 100-dimensional vectors through an embedding layer, enhancing the model's ability to process text input effectively. Dropout is used to combat overfitting in training data, finally, the model is passed to a dense layer. The convolution layer applies the filter and the Pooling layer down-samples feature maps retaining the most crucial features only. The model concludes with a softmax-activated dense layer, enabling binary classification.

```
# Build the model
model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=max_sequence_len,\
                    weights=[embedding_matrix], trainable=False))
model.add(Conv1D(50, 2, activation='relu'))
model.add(Dropout(0.2))
model.add(MaxPooling1D(5))
model.add(Conv1D(50, 3, activation='relu'))
model.add(Dropout(0.3))
model.add(GlobalMaxPooling1D())
model.add(Dense(50, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy'])

# Train the model
history = model.fit(X_train_padded, y_train, epochs=12, batch_size=64,
                    validation_split=0.2, shuffle=True)

# Evaluate the model
loss, accuracy = model.evaluate(X_test_padded, y_test)

print(f'Test Accuracy: {accuracy}')
```

Figure 2.11: CNN implementation

CNN-LSTM

The CNN-LSTM model uses CNN layers followed by LSTM layers. The CNN layer is similar to classic CNN architecture, LSTM layers consist of 50 units. The LSTM uses the tanh activation function and sigmoid for recurrent activation.

```
# Build the model
model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=max_sequence_len,
                    weights=[embedding_matrix], trainable=False))
model.add(Conv1D(50, 2, activation='relu'))
model.add(Dropout(0.2))
model.add(MaxPooling1D(5))
model.add(Conv1D(50, 3, activation='relu'))
model.add(Dropout(0.3))
model.add(MaxPooling1D(5))
model.add(LSTM(50))
model.add(Dense(50, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Train the model
history  = model.fit(X_train_padded, y_train, epochs=12,batch_size=64,
                     validation_split=0.2, shuffle=True)

# Evaluate the model
loss, accuracy = model.evaluate(X_test_padded, y_test)

print(f'Test Accuracy: {accuracy}')
```

Figure 2.12: CNN-LSTM implementation

CNN-Attention

The CNN-Attention model uses a CNN layer followed by an Attention layer. The Attention layer uses the output from the preceding CNN layer to compute attention weights and produce a context vector. The code to calculate attention weights is shown in figure 2.13.

```python
# Custom attention layer
class AttentionLayer(Layer):
    def __init__(self, **kwargs):
        super(AttentionLayer, self).__init__(**kwargs)

    def build(self, input_shape):
        self.W = self.add_weight(name='attention_weight', \
                                 shape=(input_shape[-1], input_shape[-1]), \
                                 initializer='random_normal', trainable=True)

        self.b = self.add_weight(name='attention_bias', \
                                 shape=(input_shape[-1],), initializer='zeros',\
                                 trainable=True)

        super(AttentionLayer, self).build(input_shape)

    def call(self, x):
        e = K.tanh(K.dot(x, self.W) + self.b)
        a = K.softmax(e, axis=1)
        output = x * a
        return K.sum(output, axis=1)
```

Figure 2.13: Attention implementation

```python
inputs = Input(shape=(max_sequence_len,))
embedding = Embedding(max_words, embedding_dim, input_length=max_sequence_len,
                      weights=[embedding_matrix], trainable=False)(inputs)
conv1 = Conv1D(50, 2, activation='relu')(embedding)
dropout1 = Dropout(0.2)(conv1)
pool1 = GlobalMaxPooling1D()(dropout1)
attention = AttentionLayer()(dropout1)
concat = Concatenate()([pool1, attention])
dense1 = Dense(50, activation='relu')(concat)
dense2 = Dense(50, activation='relu')(dense1)
dropout = Dropout(0.5)(dense2)
outputs = Dense(1, activation='sigmoid')(dropout)

model = Model(inputs=inputs, outputs=outputs)

# Compile the model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Train the model
history  = model.fit(X_train_padded, y_train, epochs=12,batch_size=64,
                     validation_split=0.2,
                     shuffle=True)

# Evaluate the model
loss, accuracy = model.evaluate(X_test_padded, y_test)
```

Figure 2.14: CNN-Attention implementation

Model Optimization and Training

For model optimization, the Adam optimizer and binary cross-entropy loss function were utilized. The evaluation metrics included accuracy, precision, and recall. The models were trained for 12 epochs.

Model Evaluation

The models' performance is assessed using a test dataset by generating categorical predictions through the 'predict' method. These predictions are compared to actual labels to form a confusion matrix, from which accuracy, precision, and recall are calculated to evaluate the models' effectiveness in classifying true or fake news.

```
y_pred_prob = model.predict(X_test_padded)
y_pred_classes = (y_pred_prob > 0.5).astype("int32")

# Create the confusion matrix
cm = confusion_matrix(y_test, y_pred_classes)

# Plot the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap=plt.cm.Blues)
plt.show()
```

Figure 2.15: Code for confusion matrix

2.5 Findings and Result

2.5.1 Confusion Matrix

The confusion matrix displays the performance evaluation of a classification model on a test dataset. It consists of actual classes and predicted classes.

Confusion matrix of Baseline CNN

True Positives (TP): There are 2951 true positives, correctly predicting 'Fake'.

False Negatives (FN): There are 161 false negatives, incorrectly predicting 'Fake'.

False Positives (FP): There are 95 false positives, incorrectly predicting 'True'.

True Negatives (TN): There are 2999 true negatives, correctly predicting ‘True’.

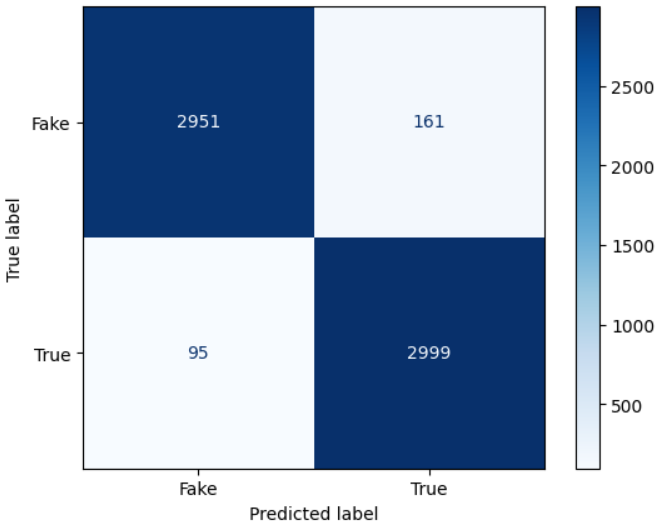

Figure 2.16: Confusion matrix of baseline CNN

Confusion matrix of CNN-LSTM

True Positives (TP): There are 3028 true positives, correctly predicting ‘Fake’.

False Negatives (FN): There are 84 false negatives, incorrectly predicting ‘Fake’.

False Positives (FP): There are 143 false positives, incorrectly predicting ’True’.

True Negatives (TN): There are 2951 true negatives, correctly predicting ‘True’.

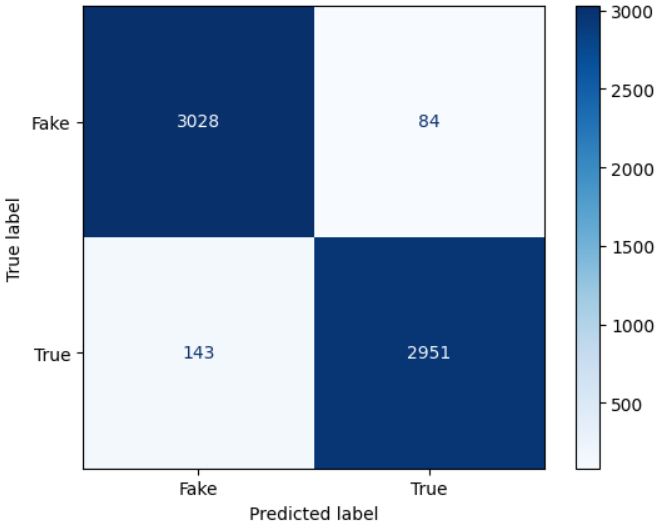

Figure 2.17: Confusion matrix of CNN-LSTM

Confusion matrix of CNN-Attention

True Positives (TP): There are 3039 true positives, correctly predicting 'Fake'.

False Negatives (FN): There are 73 false negatives, incorrectly predicting 'Fake'.

False Positives (FP): There are 104 false positives, incorrectly predicting 'True'.

True Negatives (TN): There are 2990 true negatives, correctly predicting 'True'.

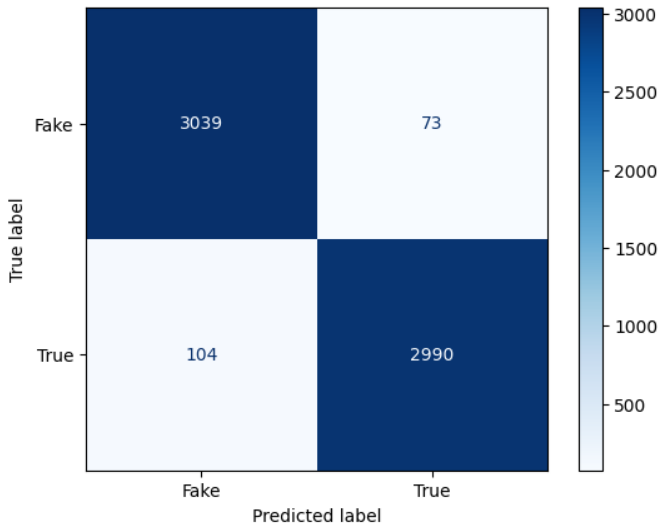

Figure 2.18: Confusion matrix of CNN-Attention

2.5.2 Accuracy Parameters

The accuracy parameters using CNN variants are evaluated on the test dataset. The observations are listed below:

Table 2.1: Accuracy parameters of CNN variants

Model	CNN	CNN-LSTM	CNN-Attention
Accuracy	95.87%	96.34%	97.14%
Precision	96%	96%	97%
Recall	96%	96%	97%
F1 score	96%	96%	97%

Comparing CNN and its variants, there is stable precision and recall while precision, recall, and accuracy slightly increase for the CNN-Attention variant. Based on accuracy parameters CNN-Attention has performed better.

2.5.3 Overfitting and Generalization

The training and validation loss and accuracy plots demonstrated a smooth convergence throughout the training process, indicating that the model effectively learned from the data without signs of overfitting or underfitting.

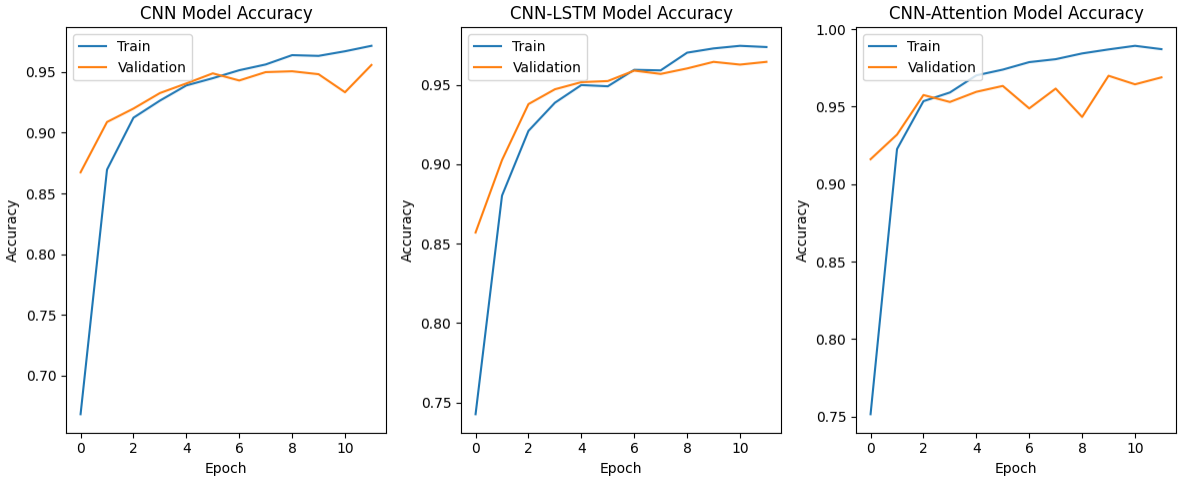

Figure 2.19: Training and validation accuracy over epochs for CNN, CNN-LSTM, and CNN-Attention

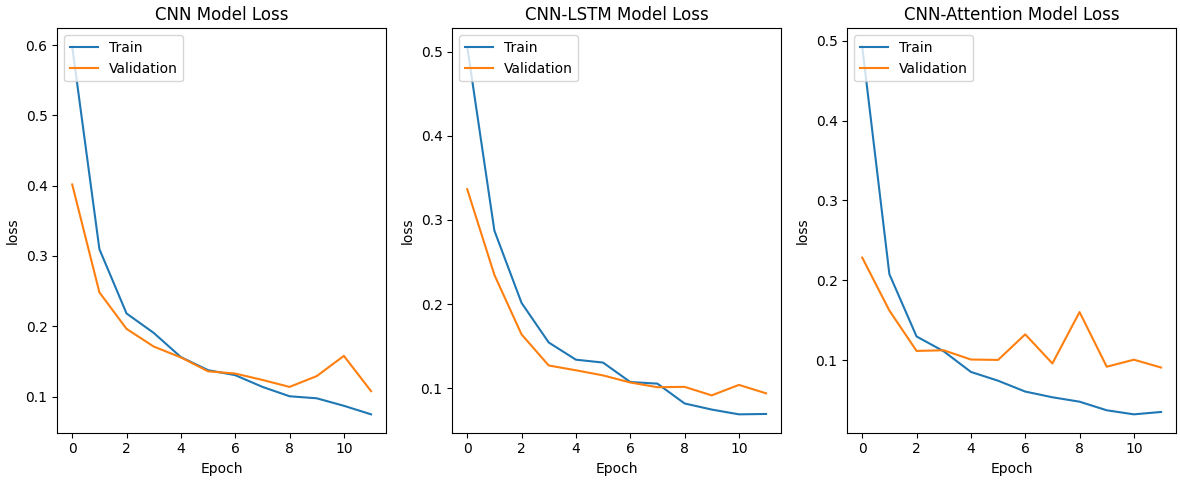

Figure 2.20: Training and validation loss over epochs for CNN, CNN-LSTM, and CNN-Attention

2.5.4 Result Analysis

The accuracy, precision, and recall observed on various CNN models suggest that the local patterns detected by baseline CNN like specific words or phrase combinations are enough for fake news classification. The long-range dependencies by the LSTM layer did not increase accuracy by a significant amount compared to the CNN-Attention model, which suggests long-range dependencies are not critical or enough for the task on the current dataset. Better performance of CNN-Attention, +0.8% accuracy than CNN-LSTM and +1.3% accuracy than baseline CNN suggest, focusing on informative parts of the text by dynamically weighing using attention mechanism can enhance model performance. It highlights the fact that adding contextual information to words using an attention mechanism is indeed useful.

The training and validation accuracy and loss over epoch are similar on all models suggesting, that all models are converging to a similar level of performance. Models are learning effectively from the data and are not significantly overfitting or underfitting. The models converges at 12^{th} epoch.

Chapter 3

Conclusion

3.1 Conclusion

The study compared the performance of three CNN-based models - baseline CNN, CNN-LSTM, and CNN-Attention - for Fake News classification. Evaluation metrics such as accuracy, precision, recall, and F1 score were used.

The CNN-Attention model emerged as the top performer, exhibiting the highest accuracy, precision, and recall. Despite a slightly higher count of false positives compared to baseline CNN, it demonstrated superior overall performance in accurately classifying 'Fake' and 'True' news.

In conclusion, the CNN-Attention model stands as the optimal choice for Fake News classification tasks, offering superior accuracy, precision, and recall with 97.14% accuracy and 97% precision and recall on the test dataset.

3.2 Future Recommendation

Exploring more recent and advanced CNN architectures or hybrid models that combine CNN with other neural network structures (e.g., Transformers) could potentially enhance classification accuracy further. Integrating multi-modal data (e.g. text, images, videos) could improve the model's ability to understand and detect fake news. The use of cross-lingual datasets could prevent models from biasing toward specific languages or regions.

Furthermore, developing and training a model on Nepali news texts could improve the detection of fake news in the Nepali language, catering to the local population's needs and enhancing the overall reliability of the model in different linguistic scenarios. This would involve creating or curating a large, labeled dataset of Nepali news articles, both real and fake.

References

[1] Ray Oshikawa, Jing Qian, and William Yang Wang. A survey on natural language processing for fake news detection, 2020.

[2] Yoon Kim. Convolutional neural networks for sentence classification, 2014.

[3] William Yang Wang. "liar, liar pants on fire": A new benchmark dataset for fake news detection. In Regina Barzilay and Min-Yen Kan, editors, *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 422–426, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[4] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification, 2016.

[5] Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann Lecun. Very deep convolutional networks for text classification, 2017.

[6] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification. In *AAAI Conference on Artificial Intelligence*, 2015.

[7] Jamal Nasir, Osama Khan, and Iraklis Varlamis. Fake news detection: A hybrid cnn-rnn based deep learning approach. *International Journal of Information Management Data Insights*, 1:100007, 04 2021.

[8] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In Kevin Knight, Ani Nenkova, and Owen Rambow, editors, *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, San Diego, California, June 2016. Association for Computational Linguistics.

[9] William Lifferth. Fake news | kaggle, 2018. `https://kaggle.com/competitions/fake-news`.