5-27-2019 11:00 AM

# Incorporating Figure Captions and Descriptive Text into Mesh Term Indexing: A Deep Learning Approach

Xindi Wang, *The University of Western Ontario*

Supervisor: Mercer, Robert E., *The University of Western Ontario*

A thesis submitted in partial fulfillment of the requirements for the Master of Science degree in Computer Science

# Abstract

The exponential increase of available documents online makes document classification an important application in natural language processing. The goal of text classification is to automatically assign categories to documents. Traditional text classifiers depend on features, such as, vocabulary and user-specified information which mainly relies on prior knowledge. In contrast, deep learning automatically learns effective features from data instead of adopting human-designed features. In this thesis, we specifically focus on biomedical document classification. Beyond text information from abstract and title, we also consider image and table captions, as well as paragraphs associated with images and tables, which we demonstrate to be an important feature source to our method.

**Keywords:** text classification, MeSH term indexing, deep learning, convolutional neural network, recurrent neural network, attention model

# Summary for Lay Audience

Text classification, especially document classification, is a process of assigning categorical labels to each document. In recent years, the number of digital documents, for instance scientific publications, continues to increase exponentially. The huge amount of open data on the internet is critical for research, and it is important to index them in a proper way. Manual indexing is inefficient and costs a lot of time and money, so automatic document indexing is a burgeoning field of research.

Deep learning has become an essential part of artificial intelligence due to the improvements in parallel computing and supporting hardware. Deep learning has performed exceptionally in many domains, such as computer vision and natural language processing. Compared to traditional machine learning algorithms, deep learning is more flexible and requires less domain knowledge when dealing with the tasks. It can automatically learn effective features from data instead of adopting human-designed features.

In this thesis, we deal with large scale document classification in an automatic way using deep learning approaches, and we specifically focus on biomedical document classification. Beyond text information from the abstract and title, we also consider image and table captions, as well as paragraphs associated with images and tables, which we demonstrate to be an important feature source for our method.

# Acknowledgements

Throughout the writing of this dissertation, I have received a great deal of support and assistance. I would first like to thank my supervisor, Dr. Robert Mercer, for his expertise, ideas, feedback, time, support and encouragement.

I would like to acknowledge Dr. Hongjun Wang, and Dr. Qiang Wu, at Shandong University, China. I want to thank them for offering me access to their high-quality computational devices, which was a great help for my research.

I would like to thank my parents for their support and encouragement throughout my life and made it possible for me to complete this degree. Finally, my friends and lab-mates, who were great support in offering me valuable suggestions.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

In this chapter, we provide a background of text classification, discuss the importance of the research question and our contribution toward solving it, and also provide the thesis layout.

## 1.1 Background

In the internet age, the number of electronic documents has grown exponentially. Manual classification of documents cannot keep up with the high demand of document processing. Thus, document classification and information retrieval has attracted special attention. They are important problems in the fields of natural language processing and artificial intelligence and solutions are urgently required.

Text classification is a task that automatically assigns categorical labels to documents. It is important in the area of document management, web searching, and document filtering. Text classification applications make our lives easier and more convenient. For instance, Google News customizes users' pages, and it filters news results based on browsing history. Email clients, such as Microsoft Outlook, filter spam effectively to protect the users. Browsers block inappropriate or offensive contents to ensure that users are under a safety web environment.

In recent years, digital libraries have become global information and knowledge networks, improving the quality of life for people as well as providing numerous research opportunities for scholars. The number of scientific publications continues to increase exponentially. Hence, digital libraries are a significant component in research. Scientific digital libraries, such as those of publishers (e.g., Springer's SpringerLink and Elsevier's ScienceDirect), open access repositories (e.g., arXiv.org), and scientific societies (e.g., IEEE Explore and ACM Digital Library), add new publications continuously [39]. Digital libraries usually offer various services, such as, metadata searching, resource discovering, and reference services. Content-based services of digital libraries depend greatly on the system upon which it is built, an example being

the semantic information of the corresponding domain, such as the ACM Computing Classification System for Computing and Medical Subject Headings (MeSH) [39].

The huge amount of open data in digital libraries is critical for research, and it is important to index them in a proper way. Processing documents associated with searchable tags enhances the functionality and organizational productivity of digital libraries. Manual indexing is inefficient, using a lot of time and money. Automatic document indexing is a burgeoning field of research.

In this thesis, we deal with automatic biomedical document indexing. We will give a detailed description of our research question in Section 1.2. We primarily apply deep neural network models to seek solutions to our research question since these models have shown excellent performance in various related areas, such as text classification, machine translation, and question-answer systems. In Chapter 2, we will review current research in text classification using deep neural networks.

## 1.2 Research Question

In this thesis, we primarily deal with text classification, in particular, multi-label classification of biomedical documents.

### 1.2.1 Text Classification

Text classification is a process that assigns labels or tags to text according to its contents. It can be done manually or automatically. Most text classification tasks were done by human annotators, prior to the information age. A human annotator reads and interprets the content of the text and then classifies it into certain categories. Traditional text classification is time consuming and expensive, especially when dealing with large number of documents. Currently, there is a trend to support text classification through automatic tools as it does the same job as human annotators, but accomplishes it in more accurate, and efficient ways.

Automatic text classification is an important application and research topic in natural language processing because of increasing numbers of online documents. Automatic text classification has been used in many areas. Some of the examples are listed below:

**Knowledge organization** includes document indexing and classification in libraries, databases, etc. It organizes documents in order to improve the efficiency of user queries, such as the library taxonomy system, which, for example, reduces readers traversing libraries.

**Information filtering** is a process of removing redundant or irrelevant information from the

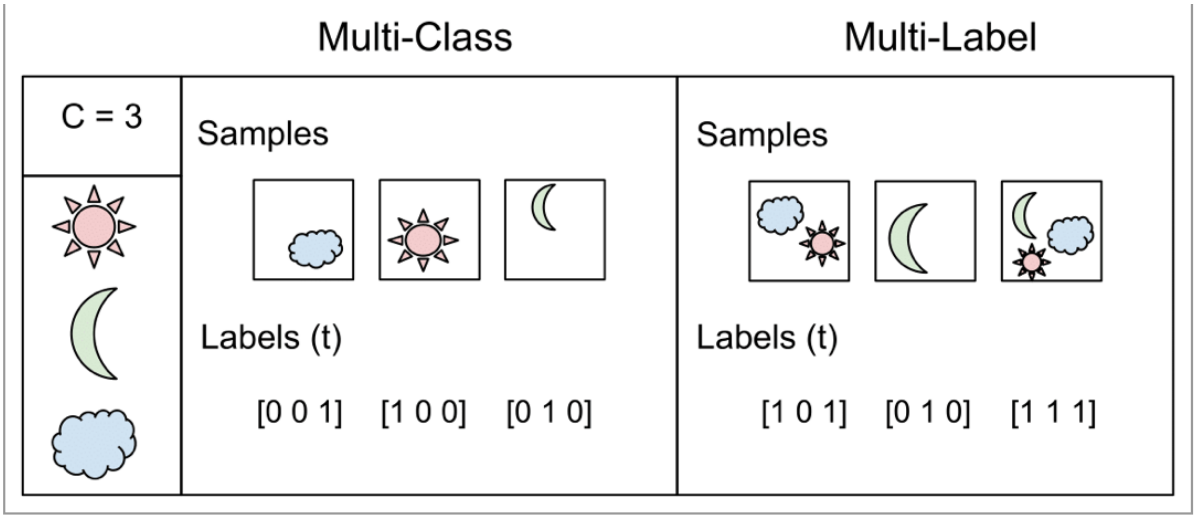


Figure 1.1: Multi-class classification VS Multi-label classification (taken from the source: https://gombru.github.io/2018/05/23/cross_entropy_loss/)

information flow. Since people can obtain information from various sources, the requirement of getting more relevant information keeps increasing. Artificial information filtering technology is in urgent demand to meet users' requirements in text classification and filtering; it constantly introduces users to valuable information only.

**Topic detection and tracking** is a process of clustering topics and tracing the appearance of those topics; the method helps users deal with information overload.

Text classification saves time and money in general, leading to its continued and enthusiastic usage in both business and research.

There are three different types of classification: binary classification, multi-class classification and multi-label classification. Binary classification classifies elements into one of two categories; multi-class classification classifies elements into one of at least three classes; multi-label classification classifies elements into a set with at least two target labels. Figure 1.1 shows the difference between multi-class classification and multi-label classification. In our thesis, we are dealing with multi-label classification in biomedical documents. We will define multi-label classification in the next subsection.

### 1.2.2 Multi-label Classification

Multi-label classification is a variant of the classification problem. It classifies a document into a set of target labels; each is mutually exclusive. We define the problem as follows: suppose we have the training data which is given as $n$ document-label pairs $\{x_i, y_i\}_{i=1}^{n}$, where $x_i \in X \in \mathbb{R}^D$

and $y_i \in \{0,1\}^L$, $D$ is the number of document features and $L$ is the number of total labels, so each document $x_i$ is associated with a set of relevant labels, denoted by label vector $y_i$. The goal of multi-label classification is to find the most relevant subset of labels from the space of categories for each document, which is $X \rightarrow \{0,1\}^L$.

### 1.2.3 Medical Subject Headings Indexing

MEDLINE[1] and PubMed[2] are databases that can access publications of life sciences and biomedical topics. They are maintained by the United States National Library of Medicine (NLM). The MEDLINE database includes bibliographic information for articles in various disciplines of the life sciences and biomedicine, such as medicine, health care, biology, biochemistry and molecular evolution. The database contains more than 25 million records in over 5,200 worldwide journals. More than 800,000 citations were added to MEDLINE in 2017, which is more than 2,000 updates daily. PubMed is a web server that can freely access the MEDLINE database of references and abstracts. Some PubMed records have full text articles available on PubMed Central[3]. Journals in MEDLINE are indexed according to Medical Subject Heading (MeSH) terms, which is the NLM's controlled vocabulary thesaurus. MeSH is a hierarchically-organized terminology indexing system that categories biomedical documents in NLM databases. The 2018 version of MeSH contains 28,939 headings. Among these MeSH terms, there are 29 check tags that are a special group of MeSH terms describing subjects of research. NLM's MeSH Indexing has been produced by human annotators at significant economic cost. It is important to explore the automatic indexing method to reduce expenditure and improve efficiency.

In Chapter 2, we will introduce current approaches in text classification as well as in MeSH indexing, and we will present different classification techniques performing MeSH indexing in Chapter 4.

## 1.3 Contributions

The contributions of our thesis are as follows:

- We explore the use of multi-channel deep learning architectures toward solving the automatic MeSH indexing tasks.

[1] https://www.nlm.nih.gov/bsd/medline.html
[2] https://www.nlm.nih.gov/bsd/pubmed.html
[3] https://en.wikipedia.org/wiki/PubMed_Central

- We provide experimental results that show that including figure and table information in addition to the typical title and abstract information improves the performance of automatic MeSH indexing.
- We provide a labeled full-text (title, abstract, figure and table captions, as well as paragraphs related to figures and tables) biomedical document dataset for the research community.

## 1.4 Structure of this document

The aim of this thesis is to discover and imporve automatic MeSH indexing. This chapter has given the background and has introduced the research problems in this thesis. Chapter 2 discusses traditional machine learning approaches in text classification, explores the deep learning approaches, and examines current attempts in MeSH indexing. Chapter 3 provides the necessary theoretical background required to understand the content of the thesis. Chapter 4 explains multichannel deep learning approaches in automatic MeSH indexing and shows the improved performance when adding full text information. Finally, Chapter 5 concludes the thesis, and outlines potential future work.

# Chapter 2

# Related Work

The goal of text classification is to automatically assign labels to certain documents. Traditional statistics-based methods depend on features, and prior knowledge of the designers. In contrast, deep neural networks learn effective features from data automatically.

In this chapter, we will briefly review statistic-based machine learning techniques that are applied on text classification, describe the various neural networks approaches, and explain current works regarding MeSH indexing.

## 2.1 Traditional Machine Learning Approaches in Text Classification

Text classification is an application of supervised machine learning task; it trains a classifier using a labeled dataset that contains text documents and their labels. Traditional text classification usually involves four phases: text preprocessing, feature extraction, training classifier, and classification model. The first step of text preprocessing is tokenization that chops the input document into small pieces, namely tokens, such as words, phrases and symbols. The text documents are then processed to minimal meaningful units. The next step is normalization that converts a list of tokens into a uniform sequences, such as converting text into lowercase, and converting numbers into words. After performing text preprocessing, feature extraction works on the pre-processed data; it selects a subset of terms describing the data before applying classifiers. In traditional text classification, features are reliant on the prior knowledge of the designer. Engineers develop strategies are to select simple and sufficient features. The final step in text classification is to train a classifier and get the classification result using features acquired from previous steps. There are many classification models available, such as logistic regression, decision tree, rule based classification, support vector machine, k-nearest neigh-

bour, and naive Bayesian [40]. We will discuss different classification methods that are widely used in text classification in the next chapter.

## 2.2 Deep Learning Approaches in Text Classification

Deep learning has become an essential part of artificial intelligence due to improvements in parallel computing and supporting hardware. Currently, complex and deep neural networks are feasible and are used now in industries. Deep learning has performed exceptionally in many domains, such as computer vision and natural language processing. Compare to machine learning algorithms, deep learning is more flexible and requires less domain knowledge when dealing with the tasks. The section analyzes various text classification techniques based on deep neural networks.

### 2.2.1 Convolutional Neural Networks in Text Classifcation

Convolutional neural networks (CNN) is a type of feed-forward artificial neural networks. CNN has been proven successfully in the areas of classification and image recognition. There are three main layers in CNN architecture: convolutional layer, pooling layer, and fully-connected layer. Convolutional layer is used to extract various input features. It applies the convolution operation on input neurons; output neurons connect to local regions of input neurons. After the convolutional layer, feature maps are generated. The pooling layer reduces dimensionality of feature maps by taking the maximum value, average or sum. A fully connected layer turns the local feature into global, and computes a class score. Chapter 3 will have a detailed review of CNN. In this section, we outline seven CNN related structures in text classification.

The most popular model that CNN has been applied in language processing especially in text classification is TextCNN (proposed by Kim in 2014) [19], shown in Figure 2.1. In the paper, the authors proposed a shallow neural net with one convolutional layer followed by a max-pooling layer; final classification layer is a fully connected layer with dropout and softmax outputs. TextCNN first embedded natural words to word vectors, which used pre-trained embeddings on 100 billion words of Google News [30]. Then, convolutional filters operated on embedded word vectors with multiple filter sizes, and feature maps were generated subsequently. A max-pooling had been applied on those feature maps, which returned the most important feature in each filter. The pooling process extracted one feature from each filter and passed them to the fully connected softmax layer which outputted the probability distribution over classes. In the experimental results, TextCNN has shown excellent performance in multi-

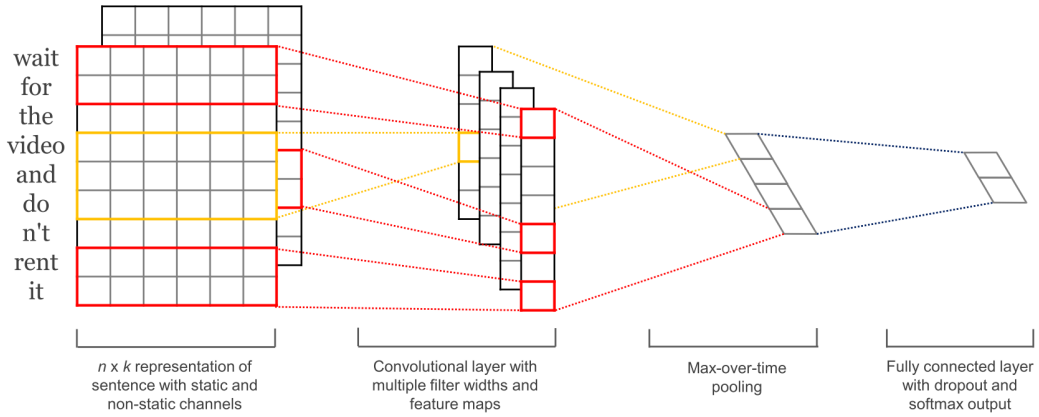


Figure 2.1: Model Architecture of TextCNN (taken from the source: Convolutional Neural Networks for Sentence Classification, Kim 2014 [19])

class classification.

Similar to TextCNN, Kalchbrenner et al. [18] proposed a dynamic convolutional neural network (DCNN) in 2014. The DCNN model contains an embedding layer, convolutional layer, folding layer and max-pooling layer, illustrated in Figure 2.2. Comparing to the aforementioned TextCNN model, DCNN uses convolutional neural networks with dynamic k-max pooling and operates wide convolution on the embedding matrix in the convolutional layer. Dynamic k-max pooling allows the network to capture $k$ most important features and reserves the relative positions among theses features; it is defined as:

$$k_t = max(k_{top}, \lceil \frac{L-l}{L} s \rceil)$$

where $l$ is the number of current convolutional layer, $L$ is the total number of convolutional layers in the network, $s$ is the sentence length, and $k_{top}$ is the fixed parameter for the topmost convolutional layer. Wide convolutions ensure that all weights in the filter reach every word in the sentence. Figure 2.3 shows the difference between narrow convolutions and wide ones. The DCNN model achieves high performance in sentiment classification, which is considered as a strong baseline in text classification.

In the paper Johnson et al. published in 2015 [15], the authors proposed two types of CNN structure to solve text categorization tasks: seq-CNN and bow-CNN. These apply CNN directly on high dimensional vectors, i.e., one-hot encodings instead of using low dimension word embeddings. The seq-CNN straightforward adapts CNN from image to text, and the bow-CNN employs bag-of-word conversion in the convolution layer.

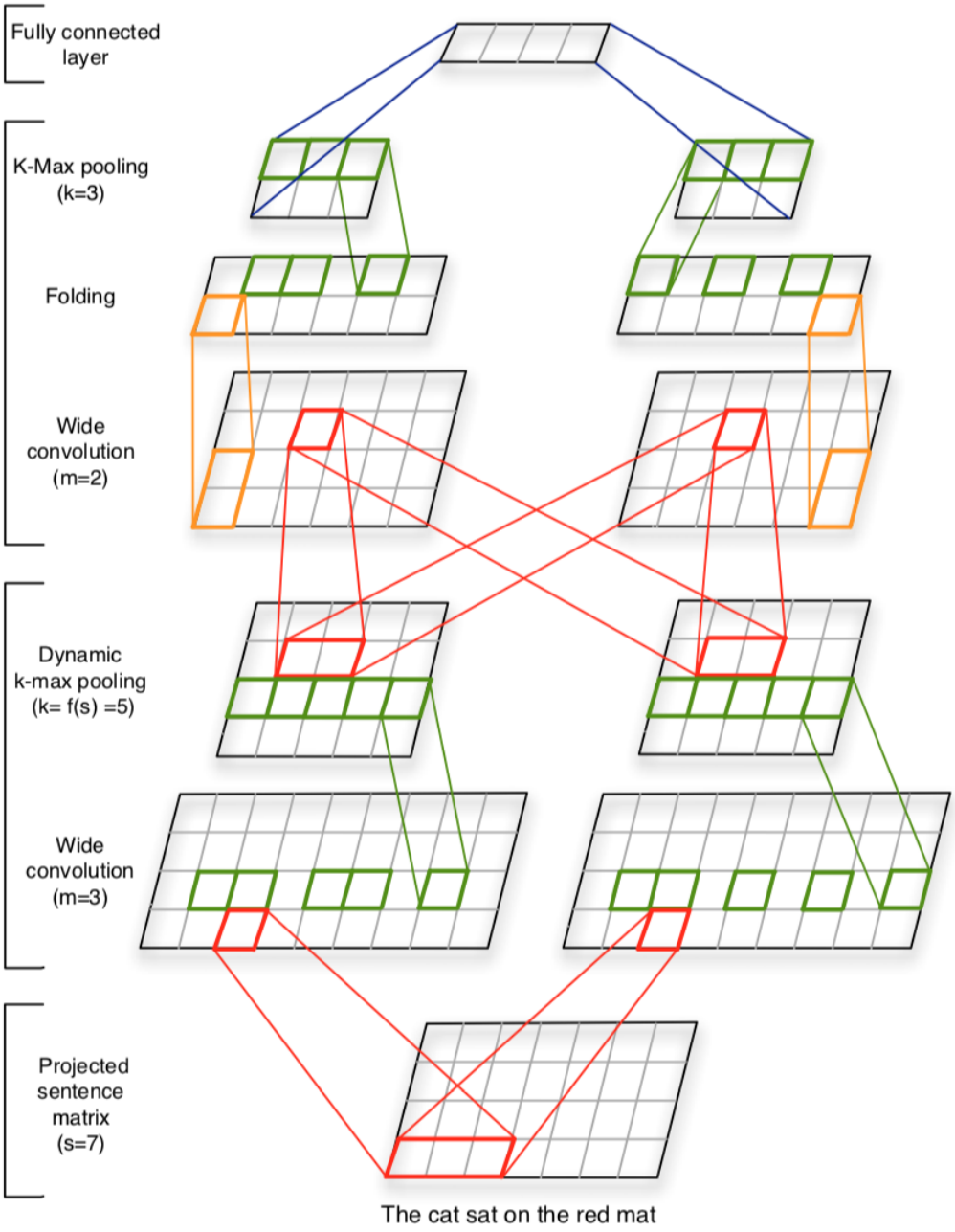


Figure 2.2: Model Architecture of DCNN for a Sentence with Seven Words (taken from the source: A Convolutional Neural Network for Modelling Sentences, Kalchbrenner et al. [18])

In the seq-CNN model, the embedding layer uses one-hot-encoding directly to represent the input document. Suppose the input document $D = (w_1, w_2, \cdots, w_n)$ with vocabulary size $V$, then each word can be represented as a V-dimensional one-hot vector. Convolution filters with region size $p$ are employed on the one-hot represented document matrix, so the size of

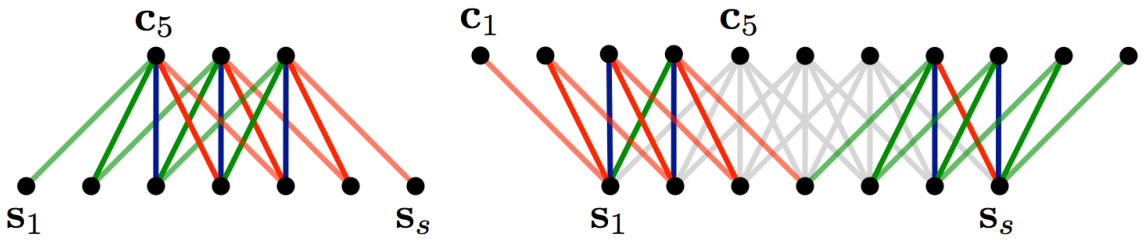


Figure 2.3: Two Types of Convolution with Filter Size 5, Narrow Convolution on the Left-hand-side and Wide on the Right (taken from the source: A Convolutional Neural Network for Modelling Sentences, Kalchbrenner et al. [18])

the region vector, namely the convolution filter, is $p|V|$. The convolutional layer learns features from high dimensional text regions and returns low dimensional feature vectors.

Seq-CNN has potential problems if the dataset has a large vocabulary size and the region size is large. Since high dimensional region vectors lead to a large number of weigh vectors, the model needs to learn more parameters. An alternative bow-CNN reduces the dimensionality of region vectors. Similar to seq-CNN, bow-CNN also uses one-hot encodings as word embedding vectors. The only difference is that bow-CNN introduces bag-of-word conversion in the convolutional layer on region vectors; thus, the dimension of region vector becomes V-dimensional rather than $p|V|$ in the seq-CNN model. The region vector is a V-dimensional binary vector where the *ith* position is '1' if and only if the word in the vocabulary appears in the text region. The seq-CNN and bow-CNN models use dynamic k-max pooling similar to DCNN [18]; the pooling takes $k$ largest values from the feature map and $k$ depends on the length of the document. Dropout layer and $L2$-norms are applied on the penultimate layer to improve performance. The last layer is the classification layer, which outputs the probability distribution over classes. The architectures of seq-CNN and bow-CNN can have two or more convolutional layers in parallel as well. Johnson's paper demonstrates an alternative model that applies CNN directly on high-dimensional one-hot vector of words in the text regions. It learns word embeddings straight from small text regions, and adds bag-of-word conversion to the convolutional layer in order to reduce the number of parameters and simplify the complexity of CNN model.

Semi-supervised CNN [16] is built on top of seq-CNN and bow-CNN [15]. It uses a semi-supervised CNN framework to learn word embedding from a small text region on an unlabeled dataset.The learned word embeddings are fed into the neural network as an additional input to one-hot vectors, which is different to seq-CNN and bow-CNN. Semi-supervised CNN trains on unlabeled data and it learns two-view embeddings from the text region. Unlabeled data can

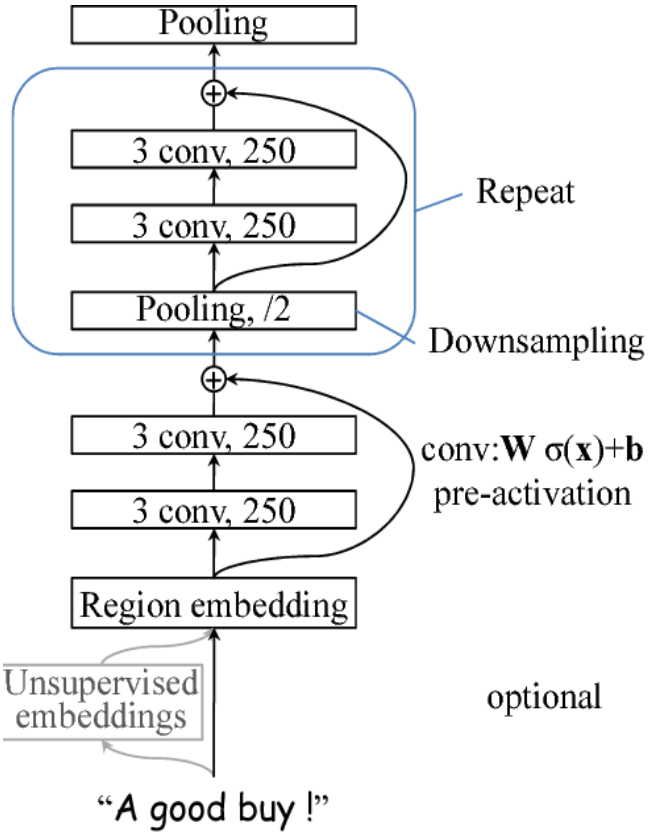


Figure 2.4: Model Architecture of Deep pyramid CNN (taken from the source: Deep Pyramid Convolutional Neural Networks for Text Categorization, Johnson et al. [17])

be taken from additional large corpus or chosen from the same classification data source. The model structure of learning two-view embedding is similar to bow-CNN. Two-view embedding concatenated with one-hot embedding as a new input, and is fed into the convolution layer in bow-CNN. Semi-supervised CNN is a new model that learns classification features from word embedding trained in both unlabeled data and labeled data.

Deep pyramid CNN (DPCNN), proposed by Johnson et al. in 2017 [17], is the first attempt in applying deep neural networks in text classification at the word level. It is built on semi-supervised CNN [16] and has a more complex architecture (Figure 2.4). DPCNN applies region embedding to represent the input document, which is similar to semi-supervised CNN [16]. Then, two convolutional blocks are applied on the embedding matrix to extract features. Convolutional blocks contain two convolutional layers and a shortcut that avoids gradient explosion and vanishing. A pooling layer is added after each convolution block, and it applies max-pooling across feature maps. The final pooling layer aggregates the input document into one most important feature, and passes it to the fully connected layer to compute the probability distribution over classes.

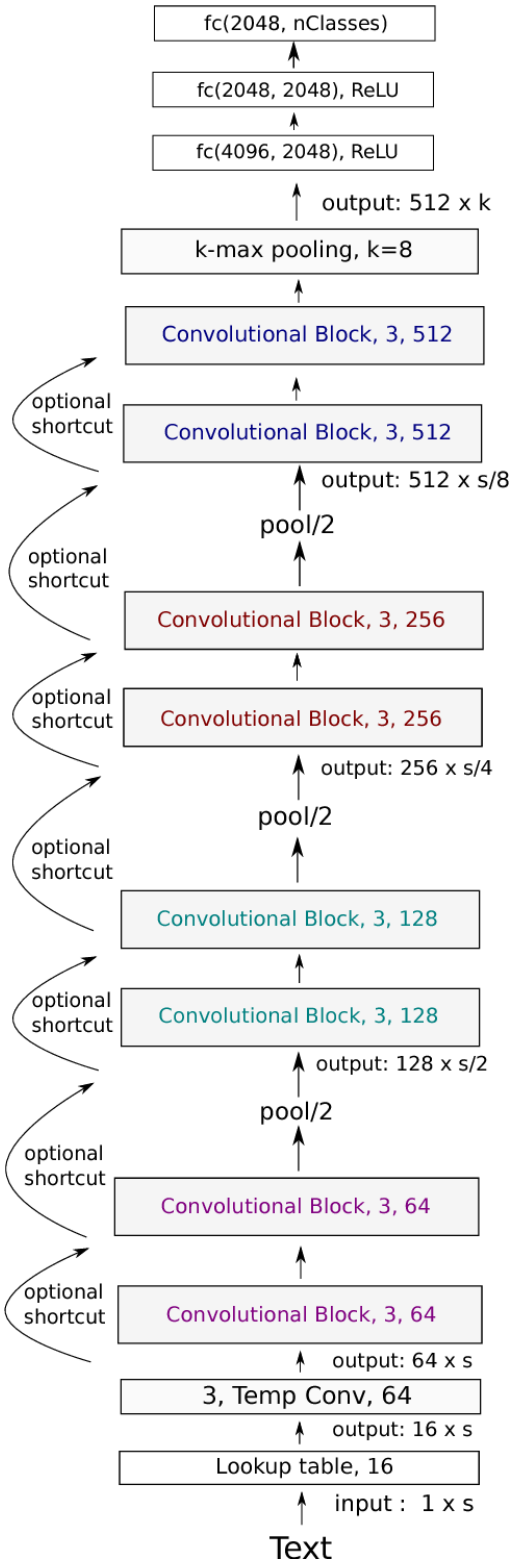


Figure 2.5: Model Architecture of Very Deep CNN (taken from the source: Very Deep Convolutional Networks for Text Classification, Schwenk et al. [36])

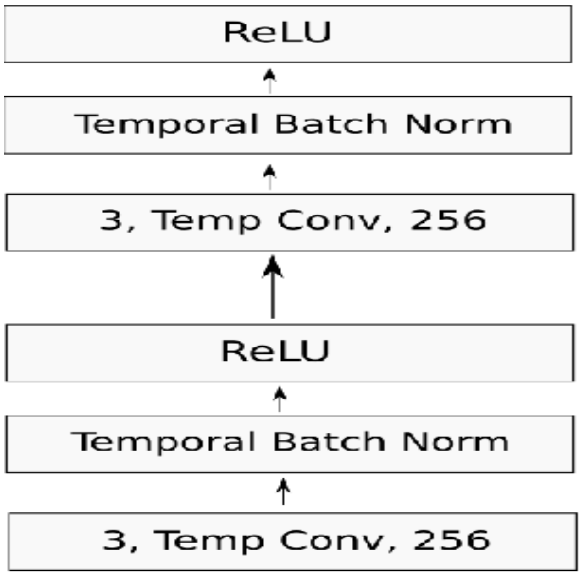


Figure 2.6: Example of A Convolutional Block in Very Deep CNN Model (taken from the source: Very Deep Convolutional Networks for Text Classification, Schwenl et al. [36])

Very deep CNN (VDCNN) model, proposed by Schwenl et al. in 2017 [36], is the first attempt in applying deep CNN at character level in natural language processing. The overall VDCNN model is shown in Figure 2.5. First, the model projects a sentence to a tensor of size $(f_0, s)$, where $s$ is a fixed length of characters in a sentence and $f_0$ is the dimension of the lookup table for characters. Then, a convolution layer is applied on the input tensor, followed by three pooling operations. Before each pooling layer, two convolutional blocks are deployed. Each convolutional block contains a sequence of two convolutional layers, each followed by a batch normalization layer with ReLu activation, shown in Figure 2.6. Two more convolutional blocks are applied on the output of the last pooling operation, followed by a k-max pooling layer. Then, the network extracts $k$ most important features, and preserves their position relationships. Lastly, three fully connected layers with ReLu activation output classification results. VDCNN achieves excellent results in large-scale datasets, which shows that increasing the depth of model will improve performance.

XML-CNN proposed by Liu et al. in 2017 [25], is the first attempt that applies CNN on extreme large multi-label text classification. Similar to TextCNN [18], XML-CNN is also built on dense word vectors and learns features of documents through convolutional layers. The two main variations that differ from the original TextCNN are convolutional filters and pooling methods. The structure is illustrated in Figure 2.7. XML-CNN inputs a document embedding matrix with dimension $n \times k$, where $n$ is the length of the padded document and $k$

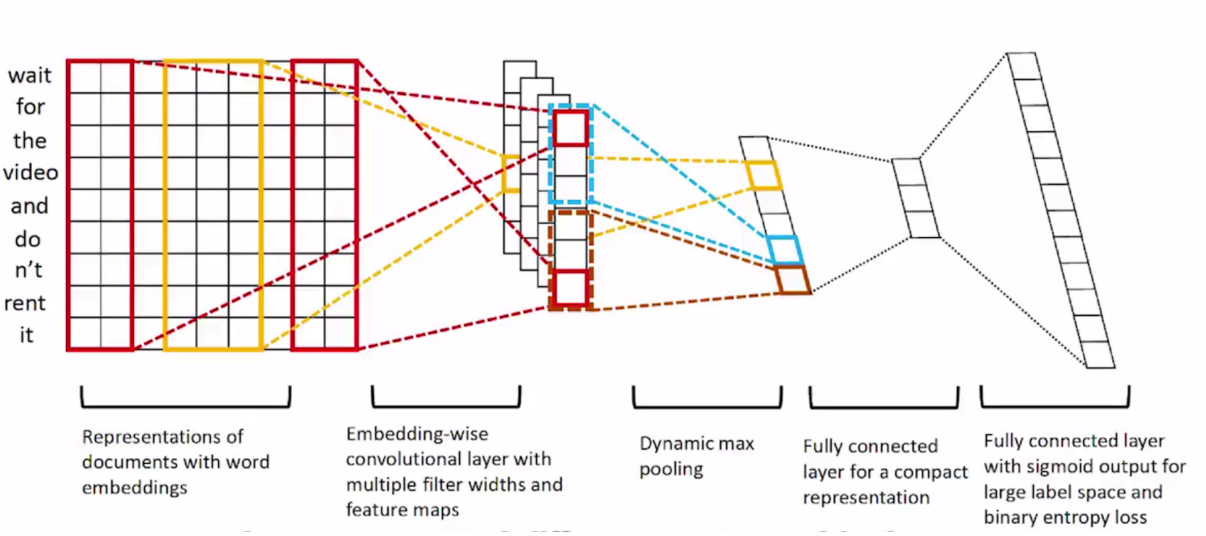


Figure 2.7: Model Architecture of XML-CNN (taken from the source: Deep Learning for Extreme Multi-label Text Classification, Liu et al. [25])

is word embedding dimensionality. Convolutional filters with multiple filter sizes are applied on the input matrix, and feature maps are returned afterwards. Compared to TextCNN, XML-CNN's holistic filters work with the entirety of words at every position and capture global features of the document. In the max-pooling layer, XML-CNN divides feature maps into $k$ chunks, and takes the maximum value inside each chunk to form outputs. In between the max-pooling output layer and final classification layer, XML-CNN adds a hidden bottleneck layer in order to reduce the number of parameters. Lastly, dropout and binary cross-entropy loss over sigmoid are applied on the final layer. In practice, XML-CNN has demonstrated an excellent performance in extreme multi-label classification.

### 2.2.2 Recurrent Neural Networks in Text Classification

Recurrent neural networks (RNN) are popular artificial neural networks used when dealing with sequential data, such as text and speech. Unlike CNN, which has relatively independent inputs and outputs, the outputs of RNN are dependent on its inputs and memories from previous time steps. RNN is expanded in time steps, and it can be unfolded as multiple copies of RNN cells, shown in Figure 2.8. At time step $t$, $x_i$ is the input, $s_i$ is the hidden state or "memory" that have been gathered so far, and $o_i$ is the output. Information is passed from one RNN cell to its successor, and the previous information is connected to the current states according to "memory". A detailed description of RNN is provided in Chapter 3.

In 2016, Zhou et al. [47] proposed an attention-based bidirectional long short-term mem-

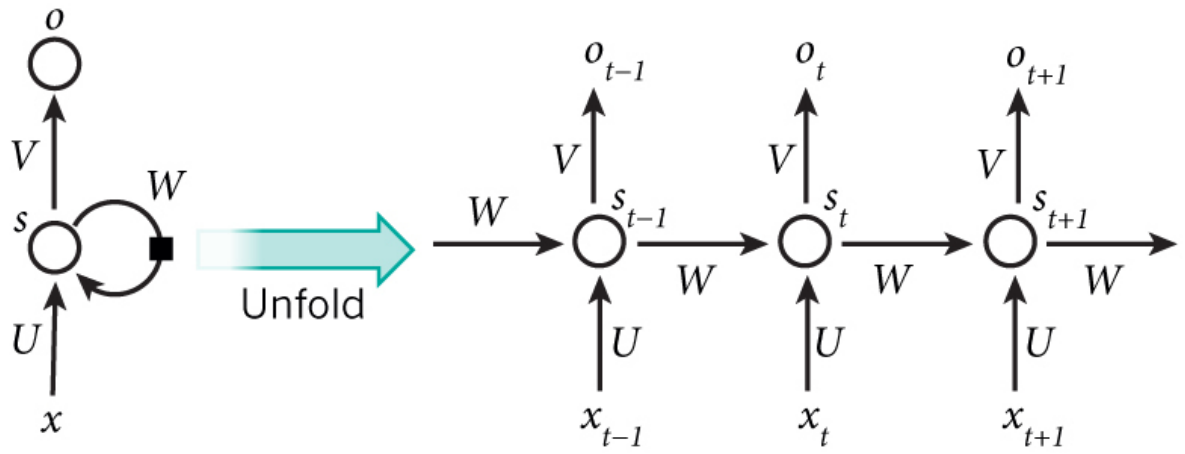


Figure 2.8: A Recurrent Neural Network (taken from the source: http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/)

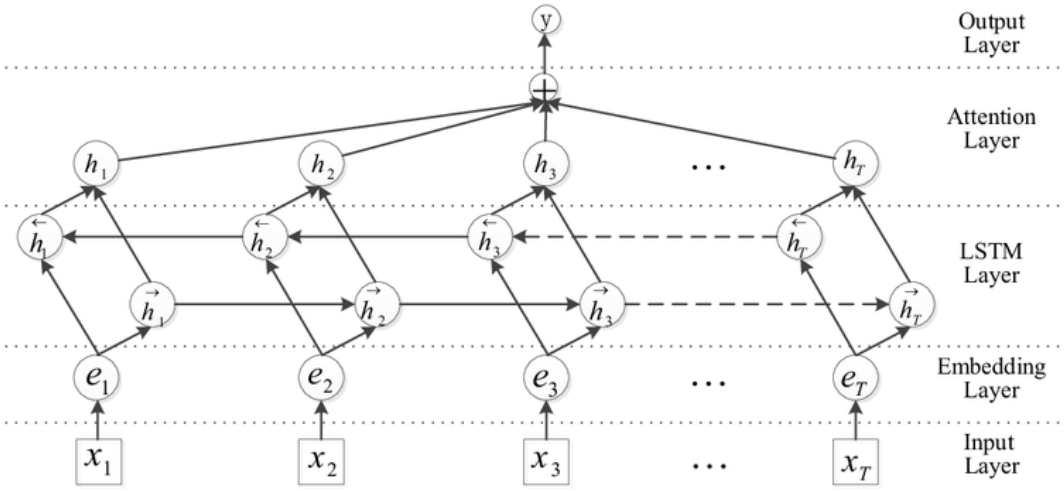


Figure 2.9: Model Architecture of Attention-based Long Short-term Memory Networks (taken from the source: Attention-Based Bidirectional Long Short-Term Memory Networks for Relation Classification, Zhou et al. [47])

ory network (biLSTM) in solving relation classification tasks at the word level. The proposed model captures important features using attention-based biLSTM rather than deriving features from lexical resources in current state-of-the-art systems. The model is formed by five layers, namely input layer, embedding layer, LSTM layer, attention layer and output layer. The model's structure is described in Figure 2.9. The input layer takes the original sentence and

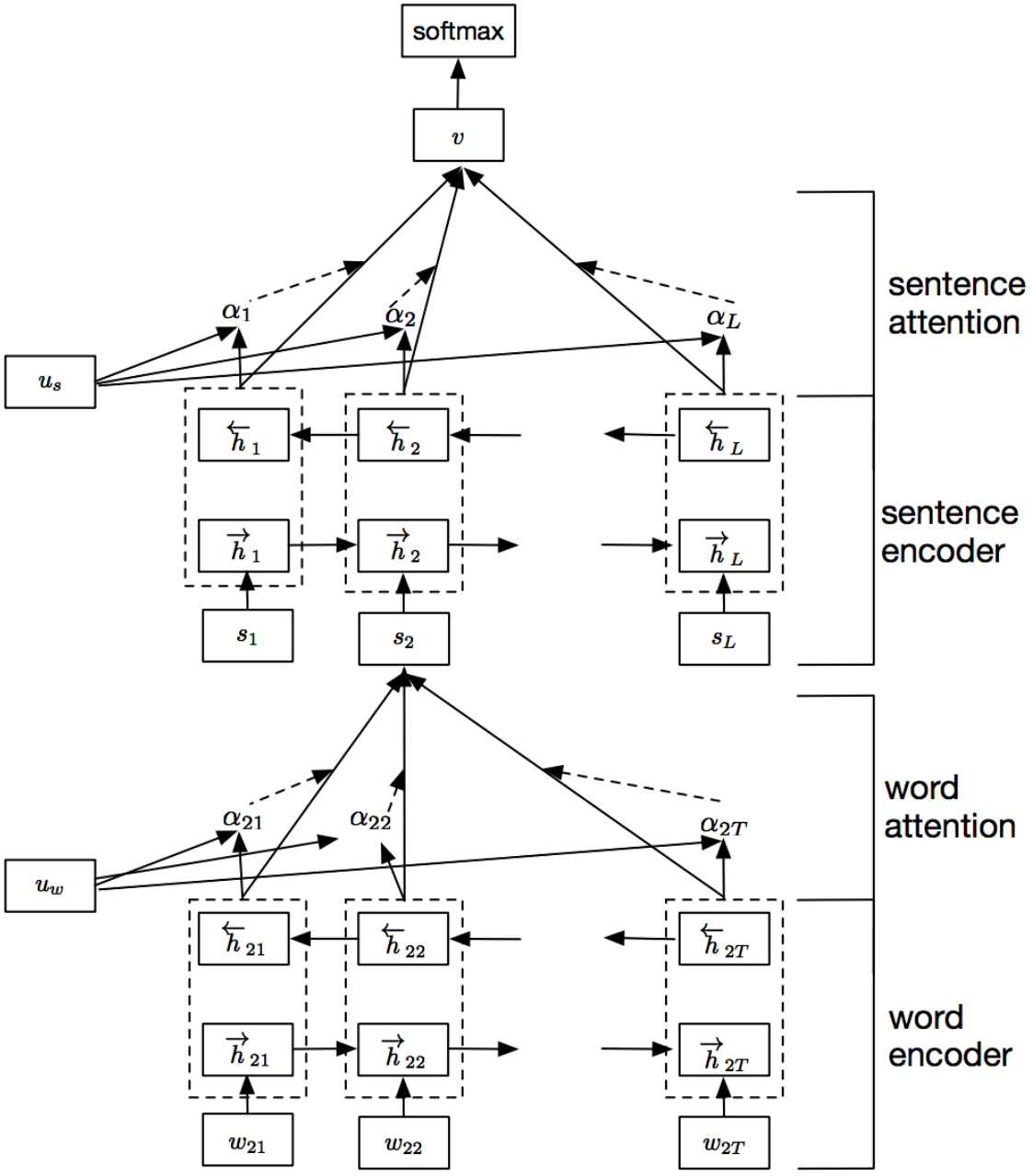


Figure 2.10: Model Architecture of Hierarchical Attention Network (taken from the source: Hierarchical Attention Networks for Document Classification, Yang et al. [44])

the embedding layer projects every word in the sentence to a low dimension vectors. The following LSTM layer captures strong features from embedded word vectors. Then the attention layer produces a weight matrix that connects the word level features extracted from the LSTM layer to a sentence level feature vector. Lastly, a softmax-based output layer takes the sentence

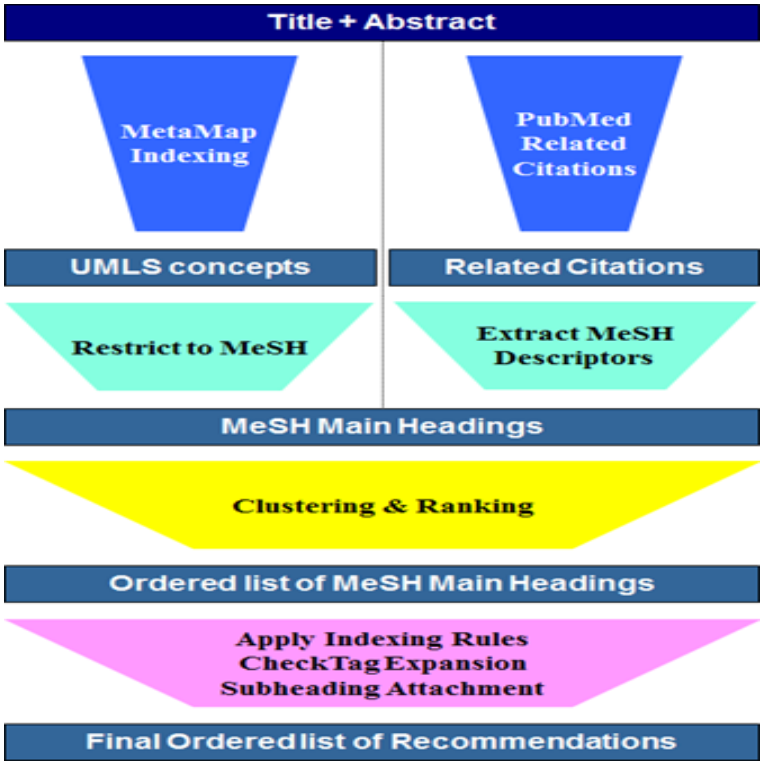


Figure 2.11: Current MTI Processing Flow (taken from the source: https://ii.nlm.nih.gov/MTI/ )

level feature vector as input, and outputs probability distributions for each relation class.

Hierarchical Attention Network (HAN) proposed by Yang et al. [44], is the first model that brings attention mechanisms into text classification. Its "context vector" finds the importance of word and sentences in documents. The architecture is illustrated in Figure 2.10. The model takes pre-trained word embedding, and applies bi-directional GRU to the embedding matrix. Outputs from the hidden layer interact with context vectors, and generate weights for each word. A sentence is represented by the weighted word. The model applies the same process on sentences to get the presentation of input document. Lastly, a softmax layer is used to generate the probability distribution over classes.

## 2.3 Related Work in MeSH Indexing

Due to the growth of documents in MEDLINE, and the increasing number of MeSH terms every year, automatic MeSH indexing is a difficult challenge. The Medical Text Indexer (MTI) [3] produced by the U.S. National Library of Medicine (NLM), is the first program that automatically produces MeSH indexing recommendations. Given an article in MEDLINE with

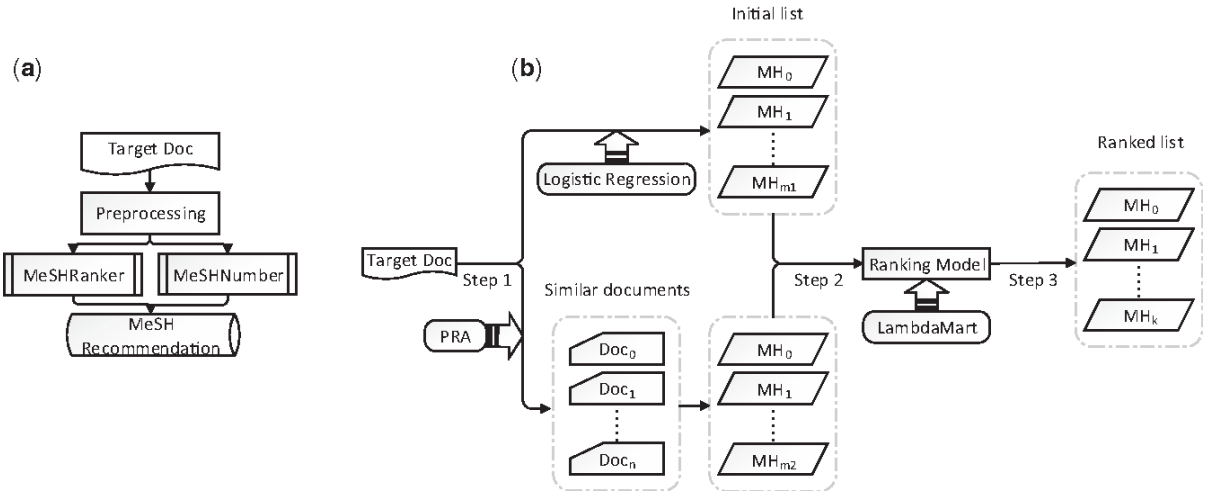


Figure 2.12: A Work Flow of MeSHRanker (a) and MeSHLabeler (b) (taken from the source: MeSHLabeler: improving the accuracy of large-scale MeSH indexing by integrating diverse evidence [26])

title and abstract, MTI will provide a ranked list of MeSH terms. The initial system of MTI was developed in 2002, and has been continuously improved over the years. The current MTI processing flow is shown in Figure 2.11. There are two main components in MTI, namely MetaMap [2], and the PubMed Related Citations (PRC) [24]. MetaMap analyzes documents and annotates them using Unified Medical Language System (UMLS)[1]. Restrict-To-Mesh [20] maps from UMLS to MeSH terms. The PRC algorithm [2] with k-nearest neighbours (k-NN) uses the document similarity to find MeSH terms. MTI is an important tool in MeSH indexing and indexers can use MTI suggestions for documents they are annotating.

BioASQ [3], a European Union-funded project, has organized challenges on automatic MeSH indexing since 2013. Participants are required to annotate unlabelled PubMed citations with abstracts and titles using their models before these articles were indexed by human annotators. The winning system in 2013, for example, used the MetaLabeler algorithm [37] to learn two models, one for ranking, and the other for predicting the number of related labels. MeSHLabeler won the first place in the 2014, which includes two components: MeSHRanker and MeSHNumber. MeSHRanker returns a ranked list of candidate MeSH terms. MeSHNumber predicts the number of output MeSH terms [26]. A workflow of MeSHLabeler is visualized in Figure 2.12. DeepMeSH won the best system in 2017. It incorporates deep semantic information into MeSHLabeler, using a dense semantic representation for documents, namely document to vectors (D2V). Additionally, DeepMeSH has another classifier to find the number

[1]https://www.nlm.nih.gov/research/umls/
[2]https://ii.nlm.nih.gov/MTI/Details/related.shtml
[3]http://bioasq.org

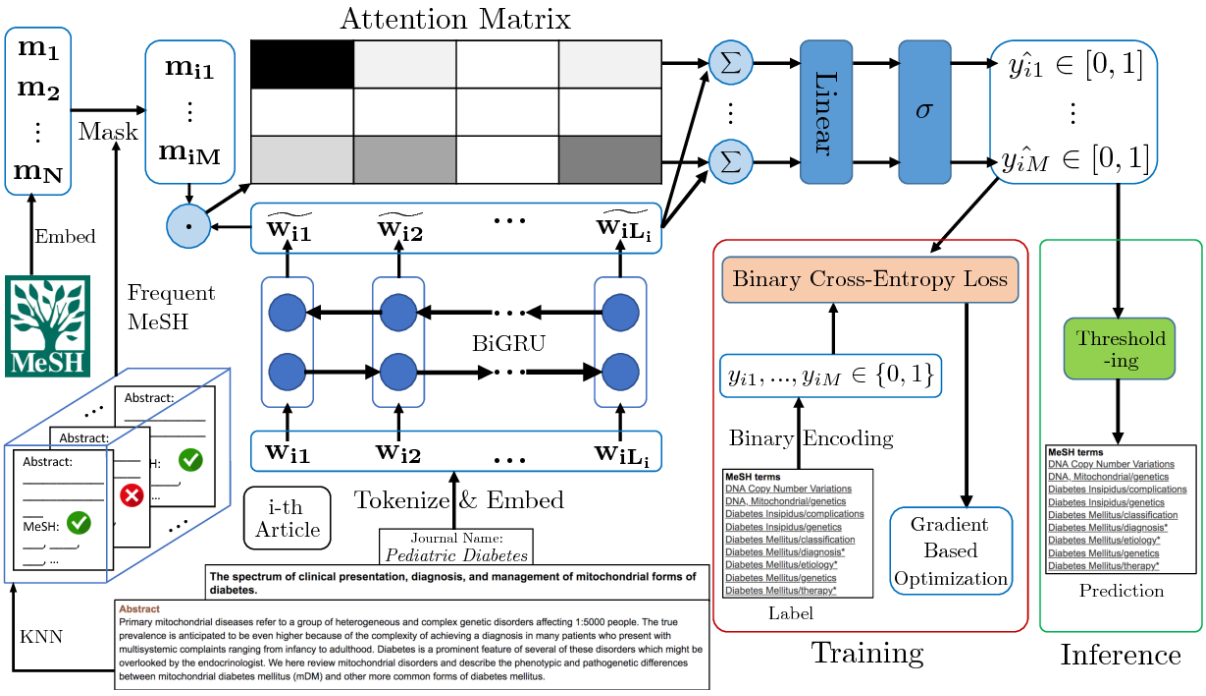


Figure 2.13: Model Architecture of AttentionMeSH (taken from the source: AttentionMeSH: Simple, Effective and Interpretable Automatic MeSH, [10])

of MeSH terms returned. AttentionMeSH, also proposed in 2017 [10], uses a bi-direction recurrent gated unit (BiGRU) to capture contextual features, and attention mechanisms to select MeSH terms from the candidate list. The model architecture is illustrated in Figure 2.13.

Rios et al. in 2015 [34] used CNN to classify 29 most frequent MeSH terms on a small dataset with 9,000 citations. Gargiulo et al. in 2018 [7] applied deep CNN on 1,115,090 papers with abstracts and titles for each article. Besides deep learning approaches, machine learning algorithms also have been explored in the hopes of solving MeSH indexing tasks. A few examples are, Naïve Bayes (NB), support vector machines (SVM), linear regression, and AdaBoost [13] [14].

# Chapter 3

# Theoretical Framework

In this chapter, we explain theoretical concepts in different classification methods and model evaluation techniques which have been applied in the work described. We first introduce word2vec embeddings as the most common text representation in natural language processing. Next, we briefly describe classifiers in traditional machine learning and then move to classification methods in deep neural networks. Lastly, we show different evaluation metrics applied in the presented work.

## 3.1 Text Representations

Word embedding is mapping words or phrases from the vocabulary to vectors of real numbers. Tomas Mikolov [29, 30] proposed word2vec embeddings which have become one of the most important components in natural language processing. Word2vec represents words as vectors, these word vectors retain meanings and relationships from the original words in the vector spaces, i.e. $V(king) - V(man) + V(woman) = V(queen)$, where $V(x)$ represents word vector for word $x$, as shown in Figure 3.1.

The architecture of word2vec is a two layer neural network where input is a context and produces a set of representing vectors. Word2vec utilizes two model architectures to construct the embedding: continuous bag-of-words(CBOW) and continuous skip-gram. The CBOW model takes the context words as input and predicts the middle word. For instance, given a text sequence $w_{t-2}, w_{t-1}, w_t, w_{t+1}, w_{t+2}$, the CBOW model interests in the probability of generating $w$ given context words $w_{t-2}$, $w_{t-1}$, $w_{t+1}$ and $w_{t+2}$, as shown in Figure 3.2, left-side. Conversely, in skip-gram architecture, the model predicts the surrounding context words given a single word as shown in Figure 3.2, right-side. We use domain specific pre-trained word2vec embeddings in our described work and more details will be provided in the next chapter.

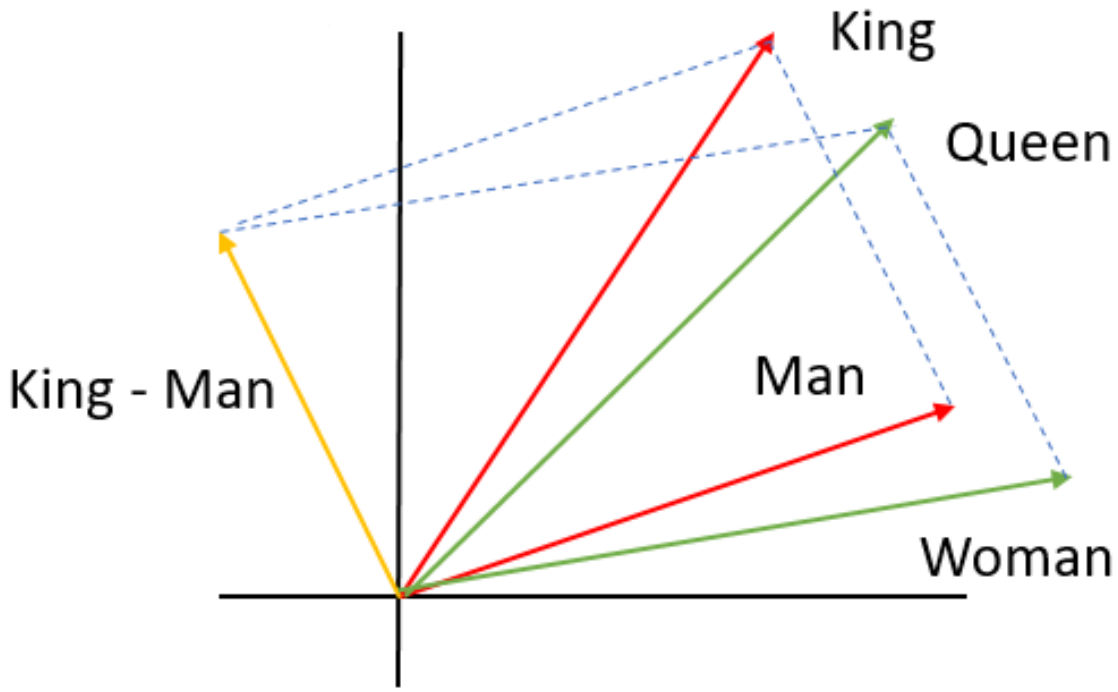


Figure 3.1: Vector Difference between Words (taken from the source: https://blogs.mathworks.com/loren/2017/09/21/math-with-words-word-embeddings-with-matlab-and-text-analytics-toolbox/)

## 3.2 Machine Learning Classifiers

Classification is a method where we group data into a given number of categories. It is a supervised learning problem which can be performed on both structured and unstructured data. The goal of classification is to accurately predict the class to new given data. In this section, we briefly discuss classification algorithms that are commonly used in machine learning.

**Logistic Regression** is a powerful algorithm for binary classification. It arises from the desire to model the posterior probabilities of classes via linear functions in features. [8]. We suppose two class labels are 1 and 0 in binary classification. For every input sample $x$, the probability of output given $x$ is defined:

$$P(Y = 1 \mid x, w) = \frac{e^{w^T x+b}}{1 + e^{w^T x+b}} = \frac{1}{1 + e^{-(w^T x+b)}}$$

$$P(Y = 0 \mid x, w) = 1 - \frac{e^{w^T x+b}}{1 + e^{w^T x+b}} = \frac{1}{1 + e^{w^T x+b}}$$

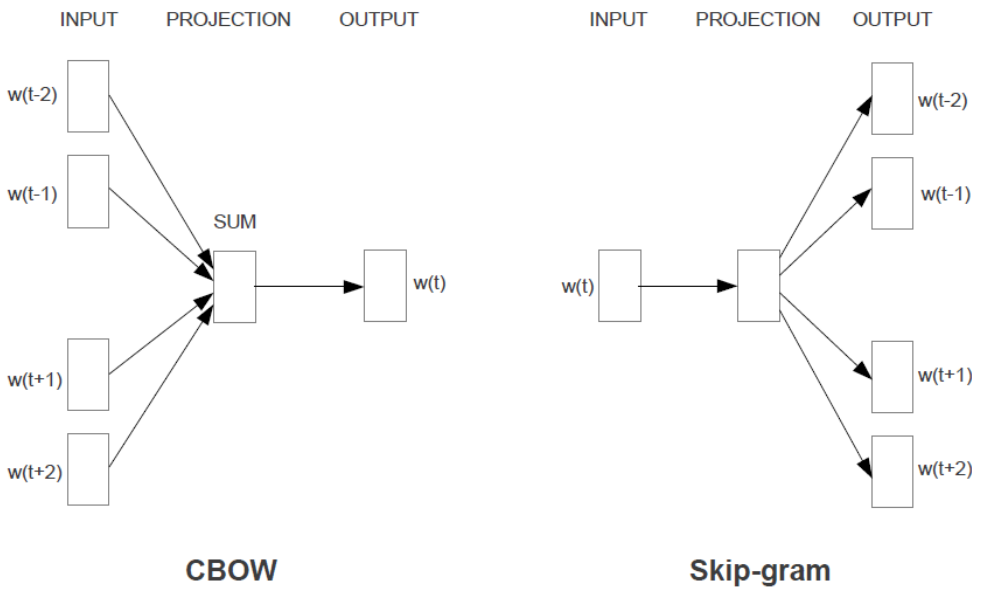


Figure 3.2: Architecture of Word2vec (taken from the source: Efficient Estimation of Word Representations in Vector Space Mikolov et al. 2013 [29])

where $x \in R^n$, $Y \in 0, 1$, $w \in R^n$ and $b \in R$. $w$ is weights and $b$ is bias, and they are hyper-parameters learned from training. The loss function is defined as:

$$J(w) = -\frac{1}{m}\sum_{i=1}^{n} y_i \log(h_w(x_i) + (1 - y_i)\log(1 - h_w(x_i))$$

where

$$h_w(x) = \frac{1}{1 + e^{-w^T x + b}}$$

$w$ is updated using gradient descent:

$$w_{j+1} \leftarrow w_i - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_w(x_i - y_i)x_i^j)$$

**Support Vector Machine** is a discriminative classifier, and it outputs an optimal hyper-plane which separates new examples by using labeled training data. It has been often used in solving non-linear classification and regression tasks.

**Decision Tree** is a simple non-parametric supervised classification algorithm. It predicts target classes by learning pre-defined rules inferred from the given data.

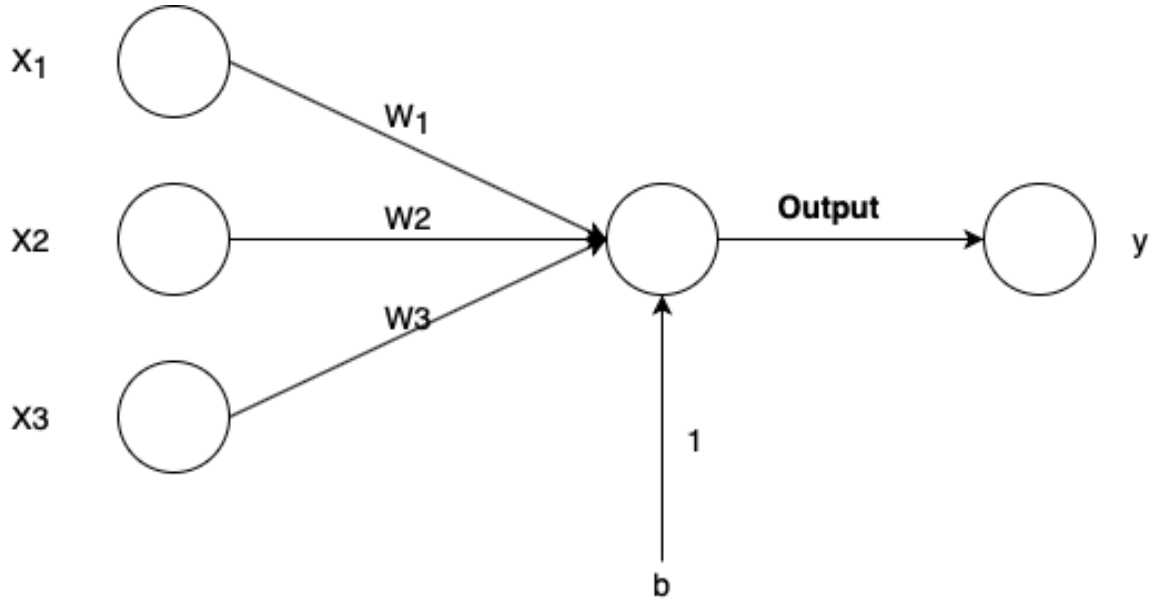


Figure 3.3: Perceptron

## 3.3 Artificial Neural Networks

Artificial neural networks (ANN) are composed of neurons, which are inspired by biological neural networks, an important constitution of human brains. ANN is not an algorithm but a system which learns a task by using existing data. In this section, different neural network architectures used in this thesis are discussed.

### 3.3.1 Feed-forward Neural Networks

Feed-forward neural networks, or multilayer perceptrons (MLPs) is type of artificial neural networks that aims to learn a function that maps $y = f(x, \theta)$ through learning $\theta$.

Neural networks emerged from the concept of perceptron.The perceptron is a machine learning algorithm, which takes inputs $x_1, x_2, ...., x_n$ and return a single binary output $y$. Figure 3.3 shows an example of a perceptron.

In this example, the perceptron has three inputs: $x_1, x_2$, and $x_3$; each input $x_i$ has a weight determined by the importances of corresponding input. The output $h_{w,b}$ is defined as:

$$h_{w,b}(x) = f(W^T x) = f(\sum_{i=1}^{m} w_i x_i + b)$$

where $x_i \in R^d$, $W = w_1, w_2, ..., w_m$, $b$ is the bias term and $f$ is an activation function. Non-linear functions, such as sigmoid function, hyperbolic tangent function (*tanh*), and Rectified Linear

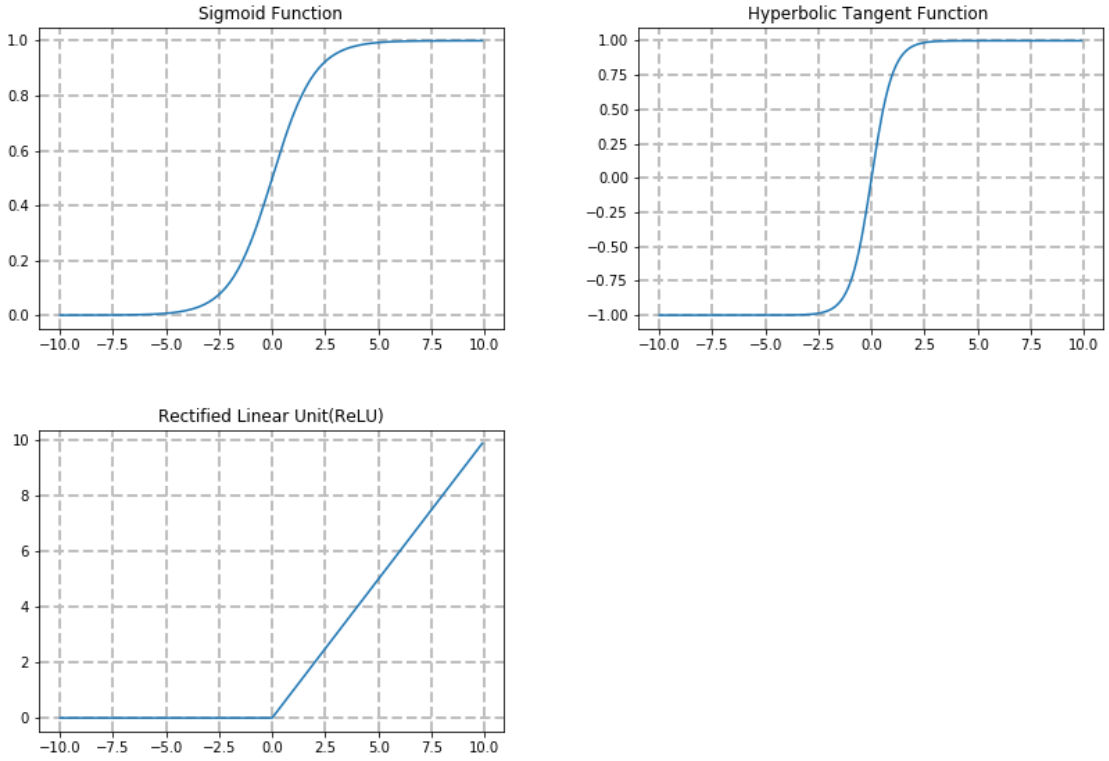


Figure 3.4: Activation Functions

Units (ReLU), are commonly used as activation functions in neural networks. Their curves are shown in Figure 3.4.

A neural network consists of multiple layers of perceptrons, and neurons (each neuron is a preceptron) are connected in a feed-forward way. Figure 3.5 is an example of a multi-layer neural network. The first layer in the network is called the input layer which is constituted by input neurons. The last layer is called the output layer, and the layers in the middle are called hidden layers.

### 3.3.2 Convolutional Neural Networks

Convolutional neural networks (CNNs), a variant of feed-forward neural networks, were proposed by Yann LeCun in 1998 [23], and they are inspired by multilayer perceptrons in essence. CNNs have been widely used in recognition and classification tasks, for instance: facial recognition, handwriting recognition and documents classification. The success of CNNs is in adopting local connectivity and weight sharing strategies to exploit local correlation. CNNs has three types of layers: convolutional layer, pooling layer and fully connected layer. We suppose inputs

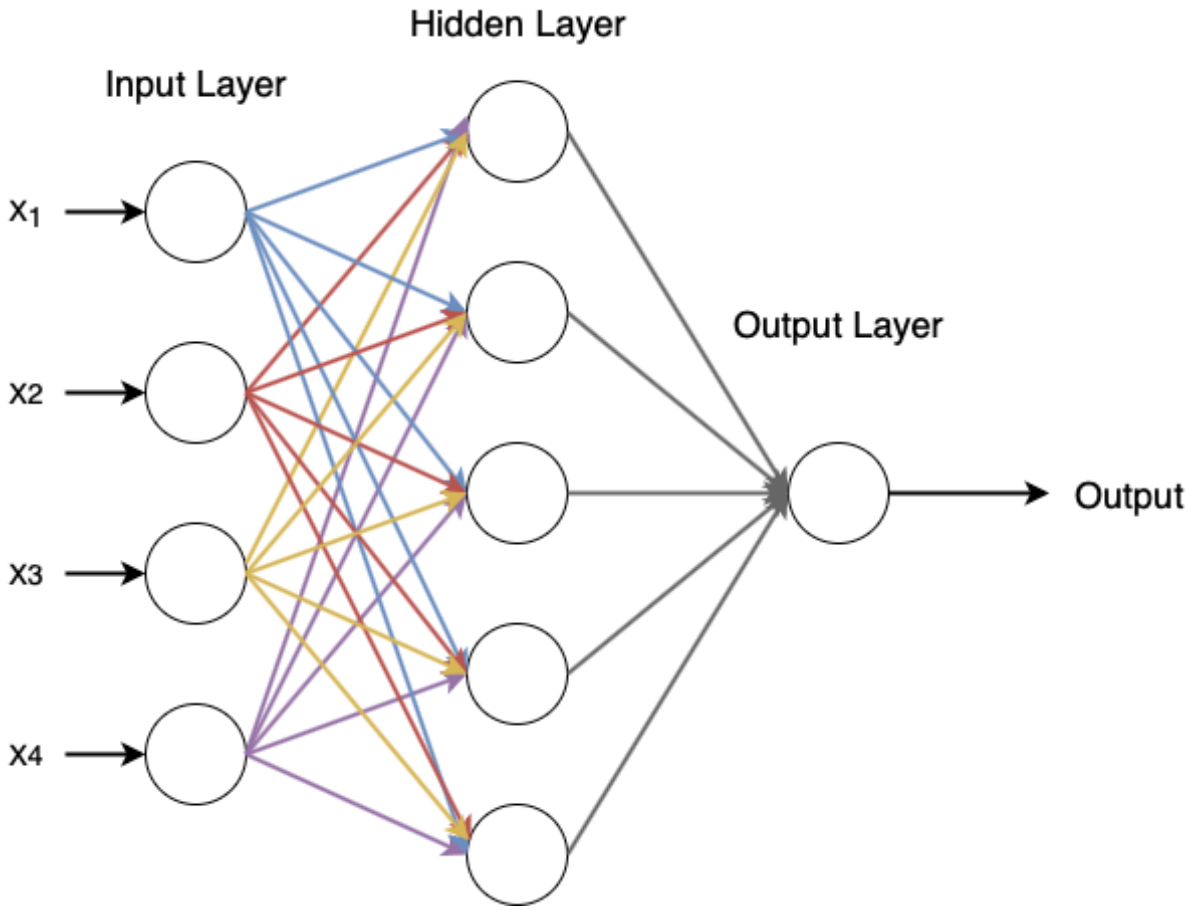


Figure 3.5: Multilayer Neural Network

vectors for CNN described below are two-dimensional.

#### 3.3.2.1 Convolutional Layer

Convolutional Layer is the core block in convolutional neural networks. CNN extract features from input vectors by applying convolution operation[1] on subregions of the input vectors repeatedly, as shown in Figure 3.6. The output generated by the convolutional layer is called ***feature map***. Convolution preserves the local connectivity in the subregions of the input vectors. Mapping from a single two-dimensional input vector to the $k - th$ feature map $h^k$ using a convolutional filter (shown in Figure 3.6) can be defined as follows:

$$h_{ij}^k = ReLu((W^k \times x)_{ij} + b^k$$

[1]https://en.wikipedia.org/wiki/Convolution

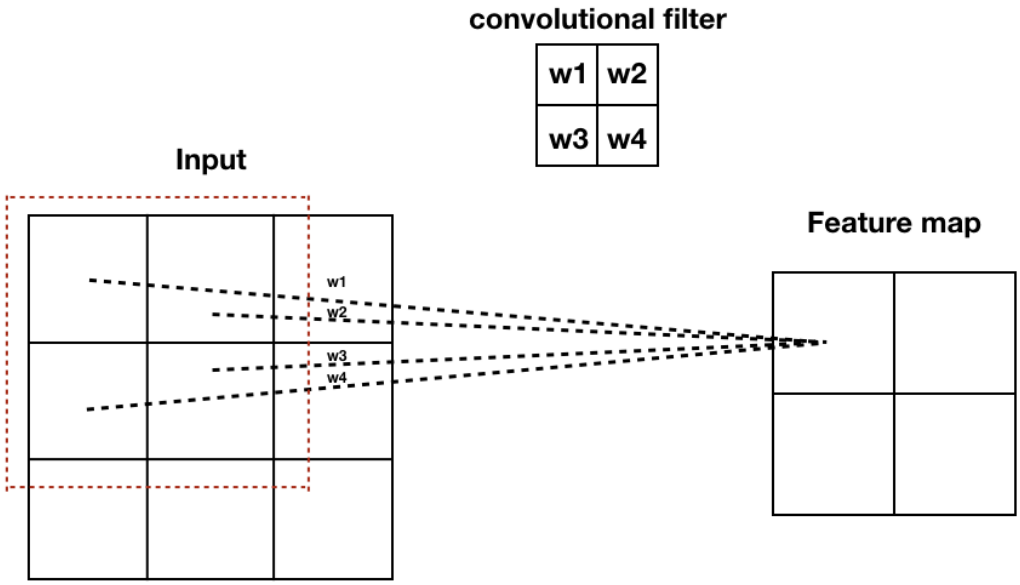


Figure 3.6: Visualization of a $2 \times 2$ filter convolving around a $3 \times 3$ input and producing a $2 \times 2$ feature map.

where $W^k$ is the weight and $b^k$ is the bias factor of the convolutional filter. $i$ and $j$ are the column and row indices of a neuron in the input vector. *ReLu* is short for Rectified Linear Unit, which is a non-linear activation function used to bound the result of convolution operation to a certain range.

#### 3.3.2.2 Maxpooling Layer

Pooling layer is usually added in the convolutional structure. It is used to reduce the dimension size and number of parameters in order to control overfitting[2]. Max pooling is the most common pooling operation used in CNN. Similar to the convolutional filter, max pooling operates in the feature map obtained from the convolutional layer and picks the maximum value over each filter, which creates a downscaled feature map. An example of $2 \times 2$ max pooling filter with stride 1 is shown in Figure 3.7.

[2]https://en.wikipedia.org/wiki/Overfitting

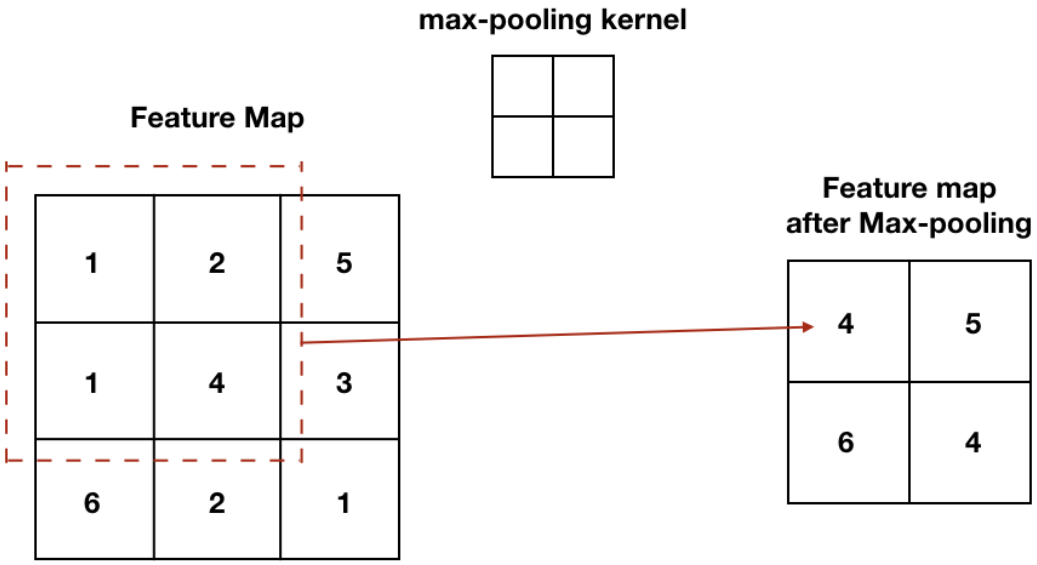


Figure 3.7: Visualization of a $2 \times 2$ max pooling filter operating on a $3 \times 3$ feature map with stride 1.

#### 3.3.2.3 Fully Connected Layer

Fully connected layer connects every neuron in the previous layer to every neuron in the next layer using activation functions. Dropout[3] is usually placed on the fully connected layer in order to prevent overfitting in the network.

#### 3.3.2.4 Classification Process

CNN models take pre-trained word vectors as input and pass them through convolution layers with different filters size. Next, the pooling layer downscales and flattens the feature map obtained from the convolution. Afterwards, extracted features are passes into the fully connected layer, and finally, probabilistic values are returned for final classification. Figure 3.8 describes a complete flow for the CNN classification process.

### 3.3.3 Recurrent Neural Networks

Recurrent neural networks(RNN) are classes of neural networks which perform recursive operation on neurons in the direction of sequential evolution. RNN is used to work on sequential

[3]https://en.wikipedia.org/wiki/Dropout_(neural_{networks})

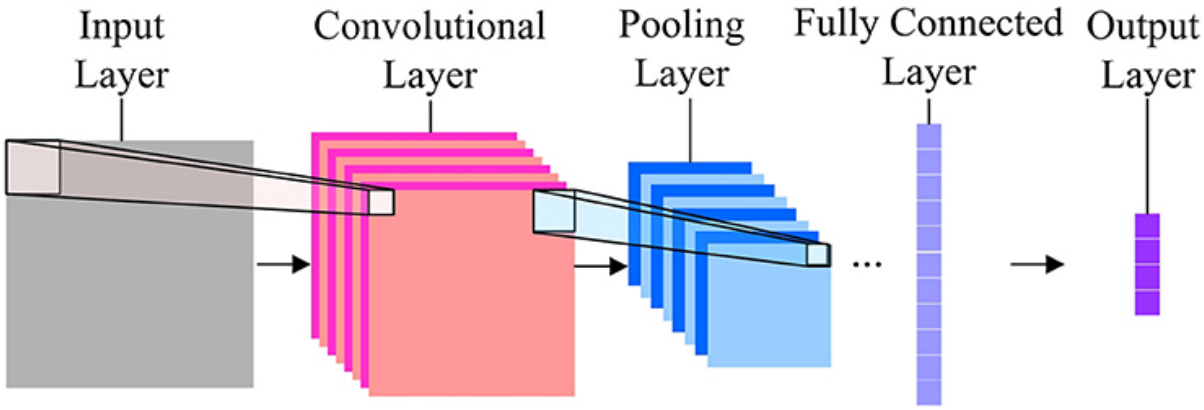


Figure 3.8: Convolutional Neural Network (taken from the source: https://www.frontiersin.org/articles/10.3389/fpsyg.2017.01745/full)

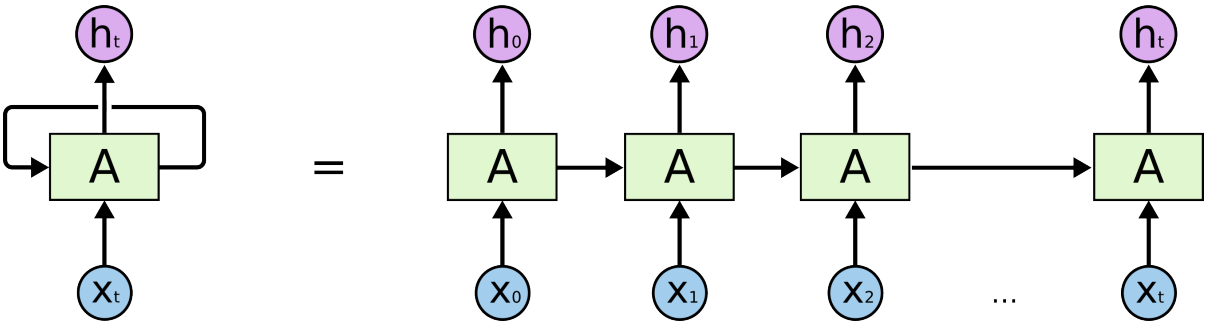


Figure 3.9: An Unrolled Recurrent Neural Network (taken from the source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

data, such as text and speech.

In Figure 3.9, A is a chunk of a neural network, it takes an input $x_t$ and gives an output $h_t$. Information can be passed from one RNN cell to its successor. It can be unfolded as multiple copies of RNN cells, each one passes information to its next one. Recurrent nets perform well in operating sequences of vectors, and it can use sequential information. We can think about RNNs have memory that can gather information from what has been calculated so far. They can connect previous information to the current task, such as using previous words to predict the next word in a sentence. The RNN calculates the hidden state $h_t$ and output $y_t$ at time step $t$ as following:

$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h)$$

$$y_t = \sigma_y(W_y h_t + b_y)$$

where $W, U$ are the weights matrices, $b$ is the bias parameter, $x_t$ is the input vector at time step

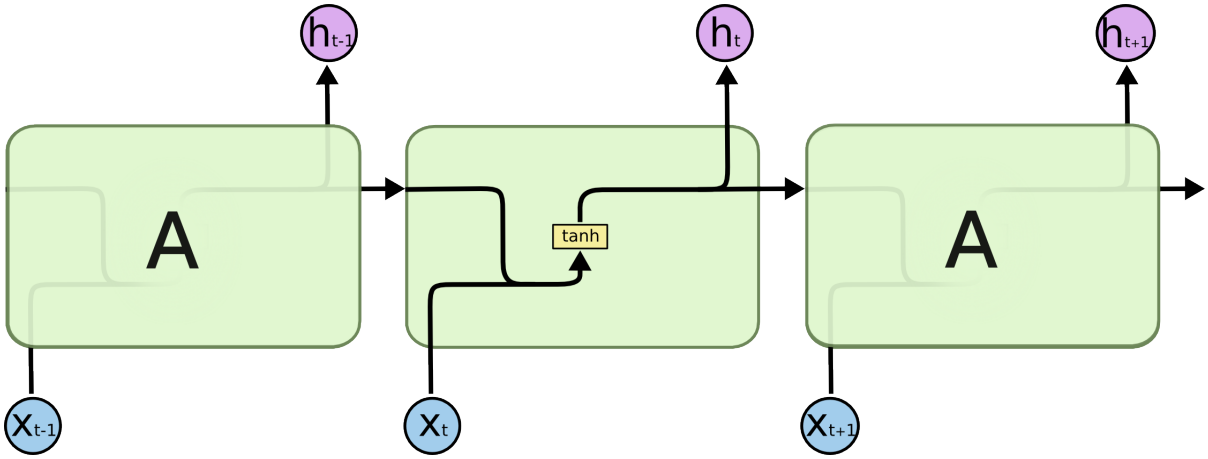


Figure 3.10: An Internal Structure of Recurrent Neural Networks (taken from the source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

$t$, and $\sigma_h$, $\sigma_y$ are the activation functions at the hidden layer and output layer, respectively. An internal structure of RNN unit is described in Figure 3.10.

#### 3.3.3.1 Long Short Term Memory Network

Long short term memory network (LSTM) proposed by Hochreiter and Schmidhuber [9] is a special kind of RNN, which can capture long-term dependencies of sequential data. Hochreiter et al introduced "Gate" and "Memory block" into the RNN model in order to avoid the long-term dependency problem. LSTM uses the same chain-like structure as the standard RNN, but it includes a memory block in every LSTM unit. Every memory block in LSTM unit has an input gate, an output gate, a forget gate and a cell state. These gates can learn and control which data in the input sequence is important to keep or throw away. The cell state contains the historical information, which remembers the input information. The LSTM calculates the hidden state $h_t$ as following:

$$i_t = \sigma_{gate}(W_i x_t + U_i h_{t-1} + b_i)$$

$$f_t = \sigma_{gate}(W_f x_t + U_f h_{t-1} + b_f)$$

$$o_t = \sigma_{gate}(W_o x_t + U_o h_{t-1} + b_o)$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c)$$

$$h_t = o_t \circ \sigma_h(c_t)$$

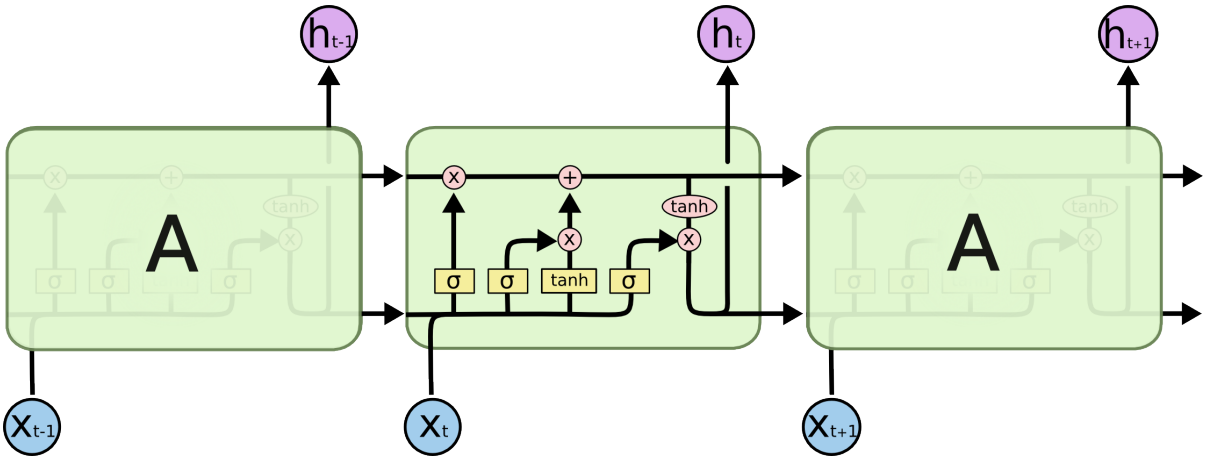


Figure 3.11: An Internal Structure of LSTM (taken from the source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

where $i_t, f_t, o_t, c_t, h_t$ stand for input gate, forget gate, output gate, cell state and hidden state, respectively. $x_t$ is the input vector to the LSTM unit, $W_\lambda$, $U_\lambda$ and $b_\lambda$ are the weights bias parameters for coresponding gate $\lambda$ and state $\lambda$. $\sigma_{gate}, \sigma_c, \sigma_h$ stand for activation functions in gate, cell state and hidden state. Operation $\circ$ denotes Hadamard product, which is an element-wise product over matrices. The internal structure of LSTM unit is shown in Figure 3.11.

Bi-directional LSTM is a variation of the standard LSTM. It connects two hidden units from two directions, namely forward and backward, i.e. $h_t = [\overrightarrow{h_t}, \overleftarrow{h_t}]$, where $\overleftarrow{h_t}$ is generated from right-to-left and $\overrightarrow{h_t}$ is generated form left-to-right. An example of biLSTM is shown in Figure 3.11.

## 3.4 Attention Mechanism

Attention mechanism was first proposed in the field of visual imaging in the 1990s and has become popular when the google mind team published "Recurrent Models of Visual Attention" in 2014 [31] which applied the attention mechanism in image classification. In the same year, Bahdanau et al. [4] proposed an RNN Encoder-Decoder framework in machine translation which was the first attempt to adopt attention model in natural language processing. Afterwards, attention mechanism has been widely used in various NLP tasks based on neural networks, such as RNN and CNN.

Attention mechanism is inspired by human perception. People don't generally process a scene in whole at one. Instead, they often focus on specific parts based on their needs and

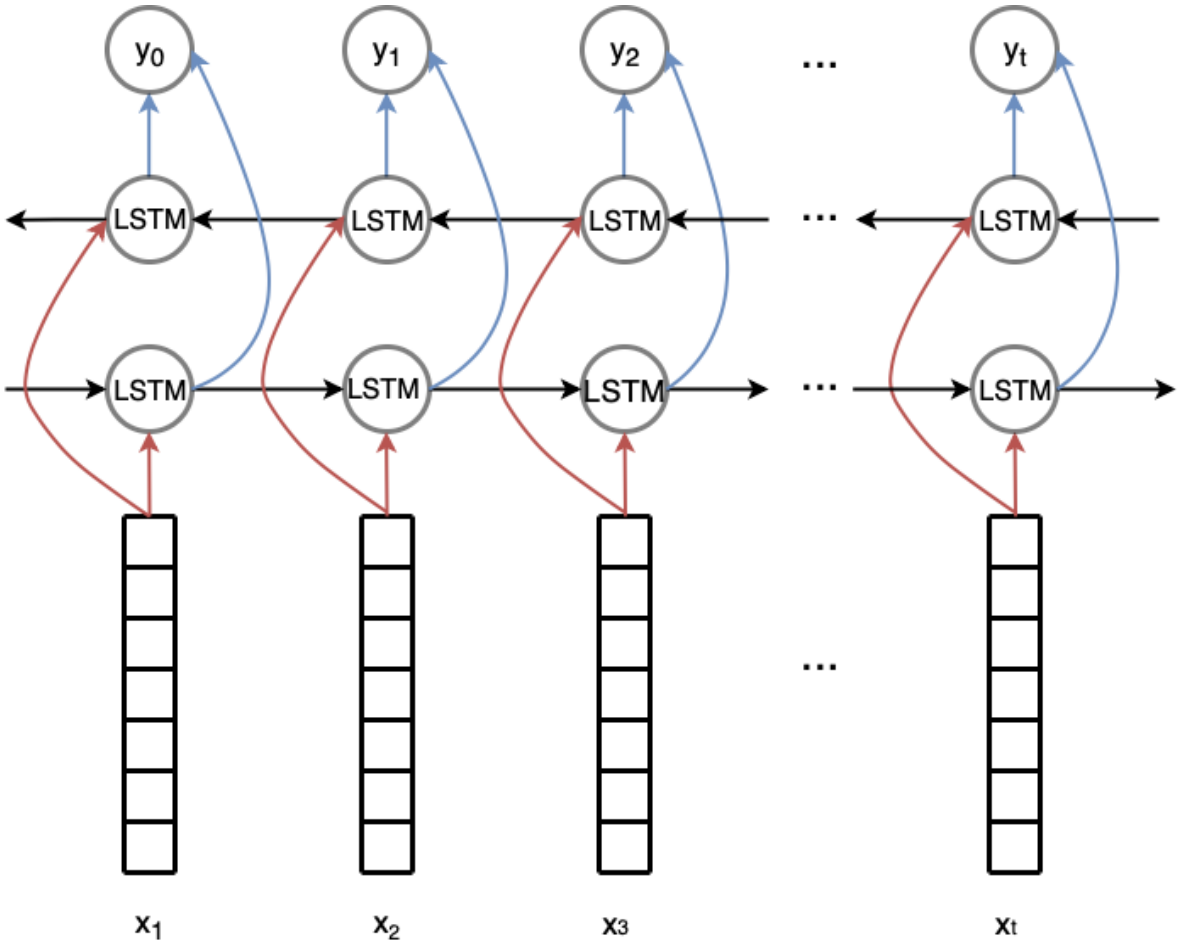


Figure 3.12: Bi-directional Long Short Term Memory Networks (biLSTM)

combine similar scenes to build internal relations [33]. An attention mechanism allows neural networks to focus on the relevant portion of the features more than the irrelevant parts. In this section, we discuss basic attention mechanisms and self-attention which are used in this thesis.

An attention mechanism is a process of computing a context vector which is a weighted average over all hidden states. We define attention mechanism as a function that maps a query and a set of key-value pairs to an output, which is similar to [41]. The query vector is the previous decoder state, key and value vectors are the encoder hidden states. The output is a weighted average, where weights are calculated by a compatibility function between keys and values. Given a query $q$ with values $v_1, v_2, ..., v_n$, and keys $k_1, k_2, ..., k_n$ we can compute the output at the attention layer $z$ [1]:

$$z = \sum_{j=1}^{n} \alpha_j(v_j)$$

$$\alpha_j = \frac{expf(k_j, q)}{\sum_{i=1}^{n} expf(k_i, q)}$$

where $\alpha_j$ is the normalized coefficient, $f(k_i, q)$ is the compatibility score between $k_i$ and $q$.

The compatibility function [41] is defined as:

$$f(k, q) = \frac{(k)(q)^T}{\sqrt{d_k}}$$

where $d_k$ is the dimension of keys.

### 3.4.1 Self-attention Mechanisms

The Google machine translation team published "Attention is All you Need" [41] in 2017, which made self-attention a hot research topic in all kinds of NLP tasks. Self-attention is an attention mechanism that calculates a representation of every position in the input sequence itself. It assigns the weights of importance to each word in the sequence. Query, key and value have been created for each position $x_i$ in the sequence, and then attention mechanism has been applied at $x_i$. At last input sequence $X = (x_1, x_2, ..., x_n)$ is transformed to another sequence of equal length $Y = (y_1, y_2, ..., y_n)$, where $x_i, y_i \in \mathbb{R}^d$. The computation of the attention function is on a set of queries, keys and values, which can be packed together into metrics respectively Q, K, and V. The output matrix is computed as:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

## 3.5 Model Evaluation Techniques

There is no generally accepted standard for the evaluation of multi-label classifications. Evaluation metrics adopted from multi-class classification and binary classification are used to measure multi-label classification in an efficient way. In this section, we present three groups of measures suggested by Tsoumakas et al. [38] and Kosmopoulos et al. [21], namely bipartition-based, ranking-based and hierarchy evaluation. In this section, we describe some of the evaluation metrics used in our thesis to compute the quality of the classification models.

To set the stage to discuss the three metrics, we define a test set of $N$ document-label pairs $\{x_i, y_i\}_{i=1}^{N}$ taken from the dataset, where $x_i$ is the document text and $y_i \in \{0, 1\}^L$. The vector $y_i$ denotes the set of true labels (i.e., MeSH terms) for each document $i$ (0 meaning the label is not in the set, 1 meaning it is in the set), $N$ denotes the number of test examples, and $L$ is the total number of labels. Given a document $x_i$, the set of labels predicted by the classifiers is

denoted as $\{\hat{y}_i\}_{i=1}^{N}$, where $\hat{y}_i \in \{0,1\}^L$, and the ranking indexes of predicted labels among the top $k$ is denoted as $r_k(\hat{y})$, where $\hat{y} = \{\hat{y}_i\}_{i=1}^{N}$.

### 3.5.1 Bipartition-base Evaluation

Bipartition evaluation is divided further into example-based and label-based. Example-based measurements calculate hamming loss, precision, recall and F-score (in our evaluation) the top5, top10, and top15 ranked labels over all of the documents of the test set. Label-based evaluation are calculated based on each label in the label set, for instance, macro-average precision, micro-average precision, macro-average F-score, and micro-average F-score.

#### 3.5.1.1 Example-based Evaluation

Hamming loss is used to evaluate the frequency that a true label is misclassified, namely the label that belongs to the document is not predicted or the label that does not belongs to the document is predicted. It is the fraction of the number of misclassified labels to the total number of labels. The smaller the hamming loss, the better the model performs. Hamming loss is defined as follows:

$$Hamming\ Loss = \frac{1}{N \cdot L} \sum_{i=1}^{N} \sum_{j=1}^{L} (y_i \oplus \hat{y}_i)$$

where $N$ is the number of test examples, $L$ is the number of total labels, and $\oplus$ denotes exlusive-or (same as $XOR$ in logic operation). $(y_i \oplus \hat{y}_i) = 1$ if $y_i \neq \hat{y}_i$, and 0 otherwise.

Example-based precision (EBP) calculates the percentage of relevant labels returned for each test document. It is represented by the ratio of positive predictions that are correctly classified, and it is defined as follows:

$$EBP = \frac{1}{N} \sum_{i=1}^{N} \frac{|y_i \cap \hat{y}_i|}{|\hat{y}_i|}$$

Example-based recall (EBR) is the ratio of actual positive cases. It shows the fraction of relevant labels that have been successfully retrieved. The formula is:

$$EBR = \frac{1}{N} \sum_{i=1}^{N} \frac{|y_i \cap \hat{y}_i|}{|y_i|}$$

Example-based F-score (EBF) is the harmonic mean of precision and recall, and is defined

as follows:

$$EBF = \frac{1}{N}\sum_{i=1}^{N}\frac{2\times|y_i\cap\hat{y}_i|}{|y_i|+|\hat{y}_i|}$$

#### 3.5.1.2 Label-based Evaluation

In order to evaluate averaging performance across labels, two averaging operations are considered, i.e. macro-average and micro-average. Macro-average methods compute global means per label and they consider all labels as equally important. Micro-average methods aggregate scores of all labels to compute average metrics and they weigh more to high frequency labels [43].

We denote $TP_j$, $FP_j$ and $FN_j$ as true positives, false positives, and false negatives respectively for each label $l_j$ in the set of total labels $L$.

$$\text{Macro-average Precision} = \frac{1}{L}\sum_{j=1}^{L}\frac{TP_j}{TP_j+FP_j}$$

$$\text{Micro-average Precision} = \frac{\sum_{j=1}^{L}TP_j}{\sum_{j=1}^{L}TP_j+\sum_{j=1}^{L}FP_j}$$

$$\text{Macro-average Recall} = \frac{1}{L}\sum_{j=1}^{L}\frac{TP_j}{TP_j+FN_j}$$

$$\text{Micro-average Recall} = \frac{\sum_{j=1}^{L}TP_j}{\sum_{j=1}^{L}TP_j+\sum_{j=1}^{L}FN_j}$$

$$\text{Macro-average F-score} = 2\times\frac{\text{Macro-average Recall}\times\text{Macro-average Precision}}{\text{Macro-average Recall}+\text{Macro-average Precision}}$$

$$\text{Micro-average F-score} = 2\times\frac{\text{Micro-average Recall}\times\text{Micro-average Precision}}{\text{Micro-average Recall}+\text{Micro-average Precision}}$$

### 3.5.2 Ranking-based Evaluation

Ranking-based evaluation ranks the predicted labels and aims to rank the relevant labels higher than the irrelevant ones. It is a robust evaluation against outliers in the returned predicted set. In this thesis, we use precision at k and normalized discounted cumulative gain to evaluate the performance of classification models.

Precision at k (p@k) evaluates retrieved labels at a given cutoff rank, considering the topmost relevant labels returned by the classification method. P@k is therefore a localized ranking method that evaluating the quality of ranking based on the top $k$ most important retrieved labels

[28]. It is defined as follows:

$$\text{p@k} = \frac{1}{k} \sum_{l \in r_k(\hat{y})} y_l$$

Discounted cumulative gain (DCG) is used to measure the effectiveness of ranking. The higher the ranking quality, the higher the DCG score. It has the assumption that highly relevant documents are more useful to the user and should have a higher score in ranking; documents with low relevance should be ranked lower conversely. The formula of DCG accumulated at ranking position $k$ is defined as:

$$\text{DCG@k} = \sum_{l \in r_k(\hat{y})} \frac{y_l}{log\,(l+1)}$$

Ranking results vary in size when having queries, so DCG value should be normalized across different queries, which is achieved with normalized discounted cumulative gain (nDCG) [12]. The nDCG is obtained by dividing DCG with Ideal DCG (IDCG), which is the maximum possible DCG at position $k$.

$$\text{IDCG} = \sum_{l=1}^{min(k,||y||_0)} \frac{1}{log\,(l+1)}$$

$$\text{nDCG@k} = \frac{\text{DCG@k}}{\text{IDCG}}$$

### 3.5.3 Hierarchical Evaluation

Hierarchical evaluation is used to measure hierarchical classification that classify elements into a hierarchy of classes. It measures the performance based on the golden truth and predicted labels as well as their ancestors and descendants. We defined the golden truth $Y_{aug}$ and predicted labels $\hat{Y}_{aug}$ of hierarchical evaluation as sets that are augmented with ancestors and descendants of the true and predicted classes within distance $N$, where $N \in \{1, \infty\}$.

$$Y_{aug} = Y \cup Y_{distance\_1} \cup ... \cup Y_{distance\_N}$$

$$\hat{Y}_{aug} = \hat{Y} \cup \hat{Y}_{distance\_1} \cup ... \cup \hat{Y}_{distance\_N}$$

Hierarchical precision (HP) measures the percentage of relevant labels returned, including their ancestors and descendants, and hierarchical recall (HR) shows the ratio of positive cases that are correctly classified. HP and HR are defined as follows:

$$HP = \frac{1}{N} \sum_{i=1}^{N} \frac{|\hat{Y}_{aug} \cap Y_{aug}|}{|\hat{Y}_{aug}|}$$

$$HR = \frac{1}{N} \sum_{i=1}^{N} \frac{|\hat{Y}_{aug} \cap Y_{aug}|}{|Y_{aug}|}$$

# Chapter 4

# Automatic Medical Subject Heading Indexing

In this chapter, we have two aims: 1) to design and compare four architecturally different deep learning models that can automatically assign medical subject headings (MeSH terms) to given biomedical documents; and 2) to compare the models when trained with different data sources. We first define our problem below, and then discuss the models and data which are used in this thesis. Finally, we show detailed experiments that have been performed to achieve these goals.

## 4.1 Problem Statement

Automatic document classification is defined as assigning categories to documents based on their contents. Machine learning, statistical methods and deep neural networks are commonly used to construct classifiers. In this thesis, we have implemented four deep learning models for automatic medical subject heading indexing. We have also curated four datasets based on two sets of documents. For each set of documents, the datasets are comprised of title and abstract text, and text comprising these two sources together with figure and table caption text and paragraphs that mention and discuss the figures and tables.

We wish to determine which of the four deep learning models performs best on the MeSH indexing task and we want to observe which data source produces the best trained model. The four models are described in Section 4.2 and the datasets used to train and evaluate the models are described in Section 4.3.1. We now turn to a discussion of the MeSH indexing task.

Medical subject headings (MeSH)[1] comprise a vocabulary that has been used to uniformly and consistently index biomedical literature. MeSH terms are arranged in a hierarchical struc-

[1] https://www.nlm.nih.gov/mesh/meshhome.html

ture and are updated annually. MeSH terms are distinctive features of MEDLINE[2], which are great tools for indexers and searchers. Indexers from the National Library of Medicine (NLM) use MeSH terms to classify documents based on the contents of journal articles in the MEDLINE database. Searchers and researchers use MeSH terms to assist subject searching in MEDLINE, PubMed[3] and other databases.

Currently, MeSH term indexing is performed by a large number of human annotators, who review full text documents and assign suitable MeSH terms to each article. Human annotation is time consuming and costly. Research shows that the average cost of annotation one document is around $9.40 [32], which is a huge cost for indexing a large number of documents. Meanwhile, a large number of documents are uploaded to MEDLINE and PubMed databases every day (approximately 2,000 on average, but as many as 4,000, on a daily basis)[4]. It is challenging to annotate all new coming documents in a relatively short time. Therefore, a system that can index a large numbers of biomedical articles is highly desired.

Multi-label classification studies the problem where each document is associated with a set of labels [46]. In the MeSH indexing problem, each MeSH term can be treated as a class label and each biomedical article can have multiple MeSH terms. We regard automatic MeSH term indexing as an extreme large-scale multi-label classification problem.

The learning framework is defined as follows. Suppose $\mathcal{X}$ represents a set of input documents and $\mathcal{Y}$ is the set of corresponding MeSH terms for each input document. Then $\mathcal{X} \in \mathbf{R}^D$ denotes the D-dimensional input space, and $\mathcal{Y} = \{y_1, y_2, \ldots, y_L\}$ denotes the label space with $L$ possible class labels. Multi-label classification studies the learning function $f : \mathcal{X} \to 2^{\mathcal{Y}}$ using the training set $\mathcal{D} = (x_i, Y_i)$. Each instance $(x_i, Y_i)$, $x_{i_j} \in \mathcal{X}$ is a D-dimensional feature vector $(x_{i_1}, x_{i_2}, \ldots, x_{i_n})$, where $j \in \{1, 2, \ldots, J\}$, $J$ is the number of words in document $x_i$, and $Y_i \subseteq \mathcal{Y}$ is the set of labels associated with instance $x_i$. The objective of multi-label classification is to predict the proper label set $Y_k$ for any unseen instance $x_k$ [46].

Two sources of challenge should be considered when solving automatic MeSH indexing tasks [45]. First, the number of MeSH terms is large and they have widely varying occurrence frequencies. There are around 30,000 MeSH terms in total and they are updated annually. The frequency of each MeSH term appearing in document labels is quite biased. For instance, of the 30,000 MeSH terms, the most frequent term "Humans", appears in 8,152,852 citations; and "Pandanaceae", on the other hand, only appears in 31 documents [45]. Second, the number of MeSH terms assigned to each document varies. Some documents have more than 30 MeSH terms, while some have fewer than 5. In this thesis, we use the 2018 version of MeSH which

[2]https://www.nlm.nih.gov/bsd/medline.html
[3]https://www.nlm.nih.gov/bsd/pubmed.html
[4]https://www.nlm.nih.gov/bsd/medline.html

contains 28,939 headings in total.

In the following section, we have implemented four novel deep learning architectures to approach the MeSH indexing challenge.

## 4.2 Classifiers

In this section, we describe four novel deep learning approaches to automatically assign proper MeSH terms to given documents. To make use of multimodal features, our models have two input channels:

- Channel 1: word embeddings from the abstract and title
- Channel 2: word embeddings from figure and table captions, and corresponding paragraphs that mention the figures and tables

The word embedding matrix for each text segment is $e \in \mathbb{R}^{d \times n}$, where $n$ is the number of words in the text segment and $d$ is the dimension of the word embeddings. Therefore, for each document, we have two embedding matrices for different aspects of the text, namely $e_{AT}$ and $e_{CP}$, where $e_{AT}$ denotes word embedding matrix from abstract and title, and $e_{CP}$ is from captions and paragraphs.

### 4.2.1 Multichannel TextCNN

We implemented a TextCNN [19] but with multichannel inputs. The model structure is shown in Figure 4.1 (a system generated model summary is presented in Appendix A). For each channel, the architecture is similar to the work presented by Kim [19]. The model learns feature representations by passing documents into different convolutional filters with various sizes. Suppose we have the d-dimensional word embedding vectors $w^i \in \mathbb{R}^d$, where $i$ corresponds to the $i$-th word in the document. The entire input document in each channel can be represented as $e_{1:n} = [e_1, e_2, ..., e_n] \in \mathbb{R}^{d \times n}$, where $n$ is the length of the document. Embedded documents are input and passed to the convolutional layer. In the convolutional layer, we have three convolutional filters of size 3, 4, and 5, with 128 filters each, so the convolutional windows are $m \times d$, where $m \in \{3, 4, 5\}$. After the convolutional operation, we obtain 128 feature maps from each filter size. The feature maps are passed to the pooling layer, which takes the maximum value for each associated feature map. After pooling, the feature maps for each channel are concatenated to form a single feature vector. This feature vector is then passed to a fully connected bottleneck layer with 512 hidden units, and then followed by a sigmoid classifier to return probability values over the 28,939 MeSH terms.

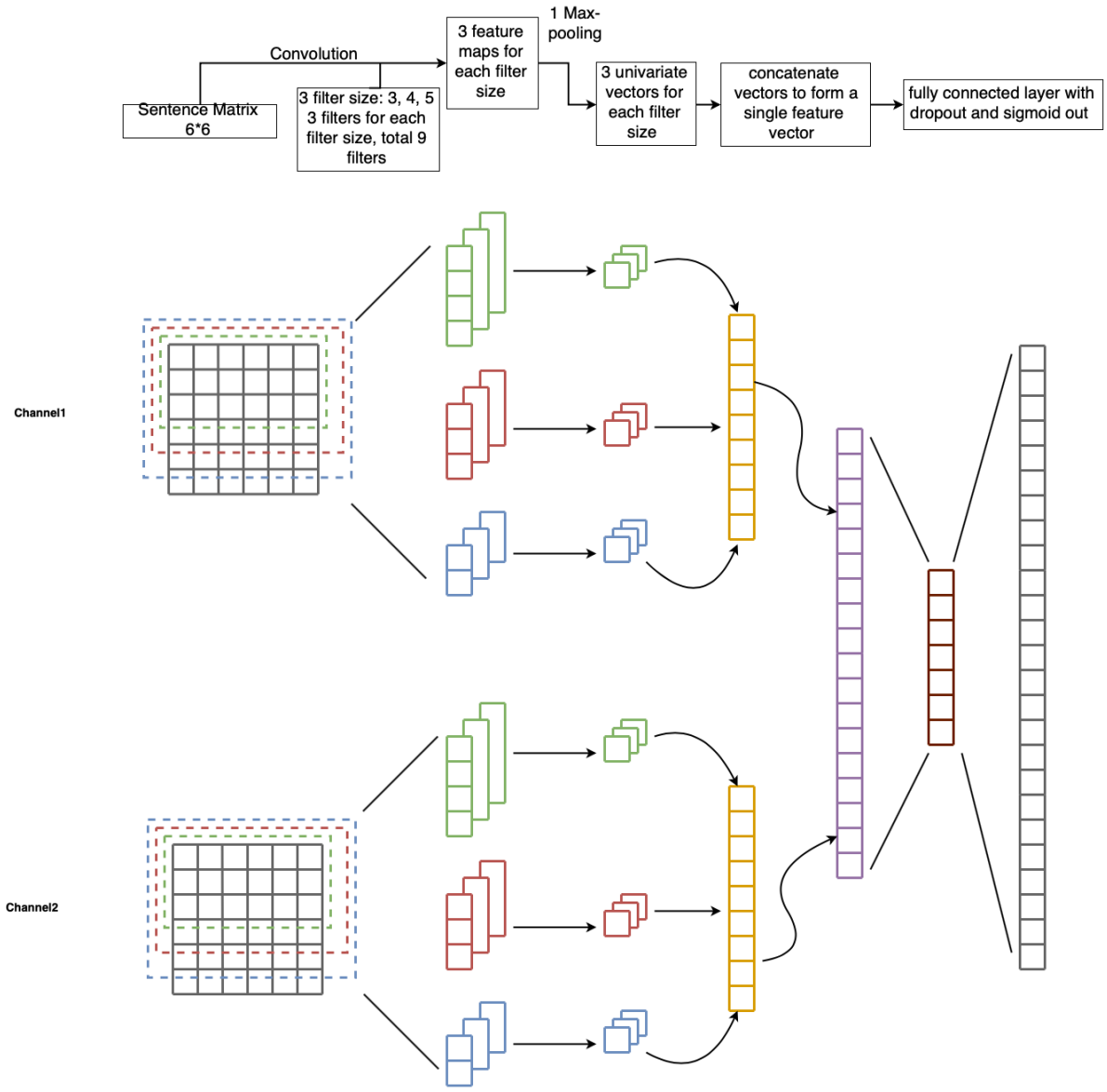


Figure 4.1: Multichannel TextCNN Architecture

### 4.2.2 Multichannel XMLCNN

In this model, we implemented an XMLCNN but with multichannel inputs. Figure 4.2 shows the model structure (a system generated model summary is presented in Appendix B). Similar to the work presented by Liu et al. [25], our model learns feature vectors by inputting an embedded document matrix into various convolutional filters. Especially, we use a vertical filter and adopt a k-chunk max-pooling in this model. We input the embedded document $e_{1:n} = [e_1, e_2, \ldots, e_n] \in \mathbb{R}^{d \times n}$, where $n$ is the length of the document and $d$ is the word embedding dimensionality. The convolutional layer has three kernels of size 2, 4 and 8, with 128 filters

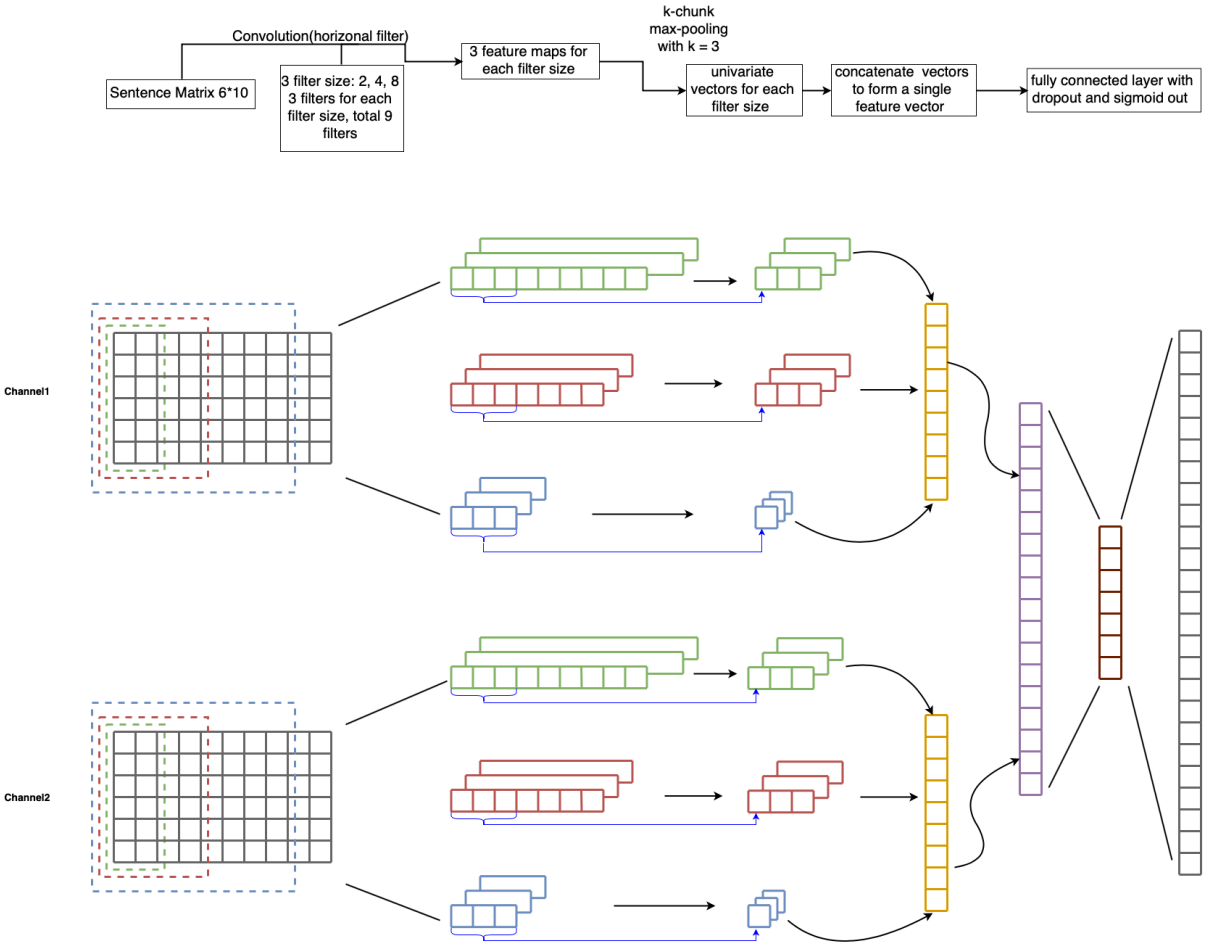


Figure 4.2: Multichannel XMLCNN Architecture

each. The convolutional window sizes are $n \times m$, where $m \in \{2, 4, 8\}$. This is followed by a k-chunk max pooling layer of window size $k \times 1$. Instead of taking the maximum value in each feature map, $p$ features are generated through k-chunk pooling to enrich the captured information. For an $l$-dimensional feature map obtained from the convolutional layer $f = [f_1, f_2, \ldots, f_l]$, we divided it into $p$ chunks, where $p = \frac{l}{k}$. Each chunk is pooled to a single feature by picking the largest value within the chunk. After the pooling layer, feature vectors for each channel are obtained and then concatenated to form a single feature vector. Finally, these features are passed through a layer with sigmoid activations to return values to be interpreted as probabilities for the 28,939 MeSH terms.

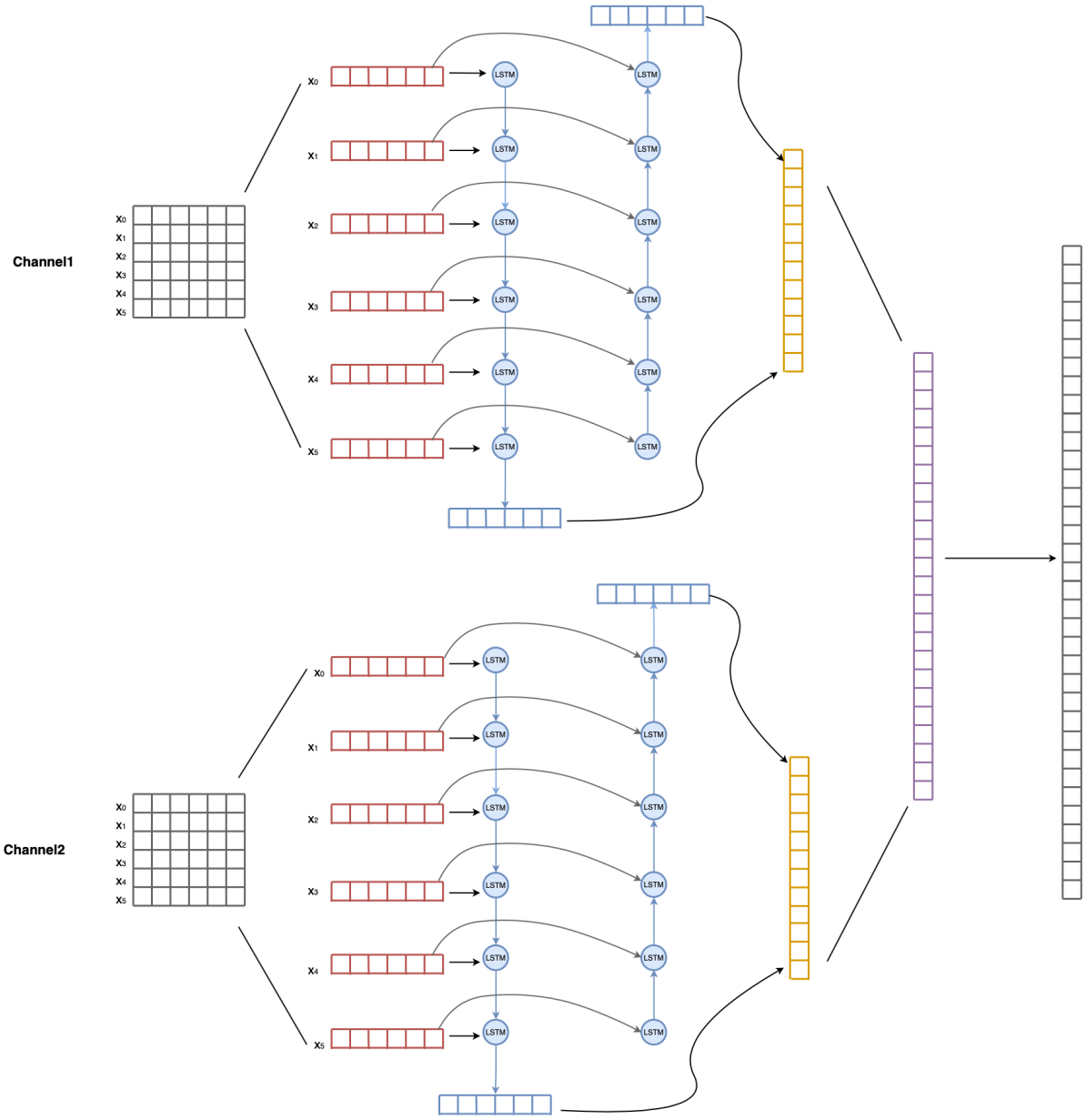


Figure 4.3: Multichannel biLSTM Architecture

### 4.2.3 Multichannel biLSTM

In this subsection, we describe our implementation of a bi-directional LSTM (biLSTM) [35] with multichannel inputs. We chose this model because biLSTMs are the preferred model for capturing contextual information of long texts [22]. Figure 4.3 shows the model structure (a system generated model summary is presented in Appendix C). For each channel, we used biLSTM as an encoder to learn hidden states from both the forward and backward directions. A biLSTM processes each word in the document by using their word embedding vectors in

both the forward and backward directions in order to calculate the hidden states $h$ at time step $t$ (i.e., the representation of the sentence up to and including the word at location $t$). At each time step $t$, we concatenate the final hidden state as follows:

$$h_t = \overrightarrow{h_t} \oplus \overleftarrow{h_t}$$

where $\oplus$ indicates concatenation, $\overrightarrow{h_t}$ represents the hidden state at $t$ from the forward direction and $\overleftarrow{h_t}$ is the hidden state at $t$ from the backward direction. In the forward direction, $\overrightarrow{h_t}$ is generated from $\overrightarrow{h_t} = LSTM(x_t, \overrightarrow{h_{t-1}})$. Similarly, the backward direction is calculated from $\overleftarrow{h_t} = LSTM(x_t, \overleftarrow{h_{t-1}})$. $LSTM$ represents the operations inside each LSTM cell and for detailed calculations the reader is referred to Section 3.3.3 in the previous chapter. The outputs obtained from biLSTM in both channels are concatenated to form a single feature vector, and then it is passed to a sigmoid function to return probabilities over the 28,939 MeSH terms.

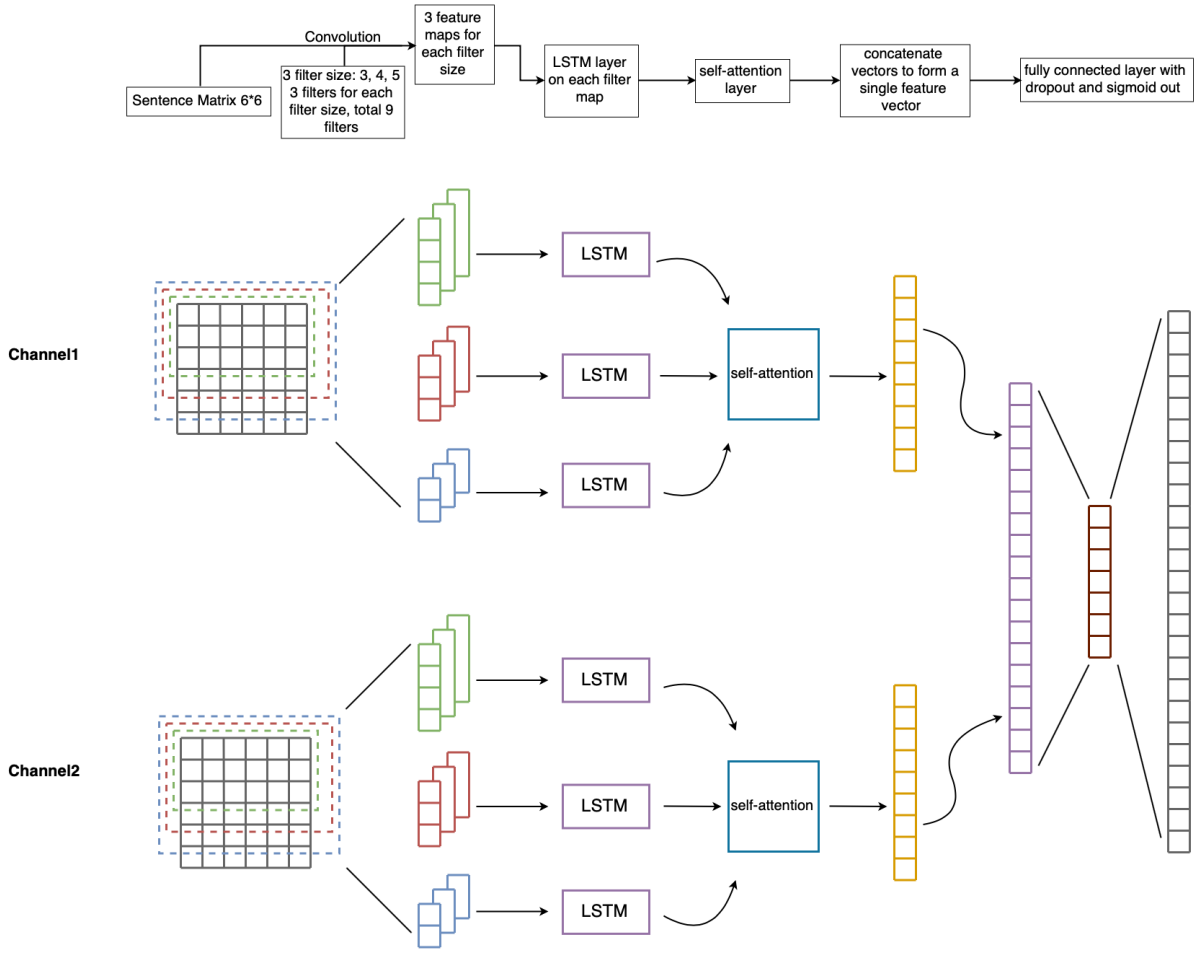


Figure 4.4: Multichannel convLSTM Architecture

### 4.2.4 Multichannel Attention Based convLSTM

In this model we use a stacked CNN and LSTM for feature extraction. We chose this model because it combines the attributes of the CNN and LSTM models. The model structure is shown in Figure 4.4 (a system generated model summary is presented in Appendix D). The convolutional layer is similar to the multichannel TextCNN model which processes the embedded document matrix to convolutional filters of size 3, 4, and 5, with 128 filters each. Then instead of the max-pooling layer, the feature map is processed with an LSTM. The outputs are then passed to a self-attention layer to obtain an attention representation of the vectors. At last, the attention vectors from both channels are concatenated and passed to a sigmoid function to return probabilities over the 28,939 MeSH terms.

## 4.3 Setup

In this section, dataset that are used in this thesis, and data pre-processing steps are first discussed, Second, we explain how we generate word embedding. Finally, we show the detailed experiment setup and model hyper-parameters that are used in this thesis.

### 4.3.1 Datasets

Most existing approaches in automatic MeSH indexing are performed on datasets with abstracts and titles only. In this thesis, we created a full text dataset which contains table and figure captions as well as associated paragraphs, as we believe figures and tables might provide important MeSH features for classification. The two datasets we used are described as follows:

- **2015 Subject Extraction Test Collection (SETC2015)**: SETC2015 contains 14828 PMC full text articles used in the "Extracting Characteristics of the Study Subjects from Full-Text Articles" [6]. We used this dataset to create the following two datasets:
  - **Abstract and Title (Small)**: labelled documents from SETC2015 which contains abstract and title only
  - **Full Text (Small)**: labelled documents from SETC2015 which contains abstract, title, figure and table captions, and associated paragraphs (we defined "associated paragraphs" as paragraphs that have key words "Figure" and "Table")
- **PMC Full Text Collection**[5] **(PMC Collection)**: We download 257,590 PMC full text documents in the XML format, and used this dataset to create the following two datasets:

[5] https://www.ncbi.nlm.nih.gov/pmc/tools/ftp/

| Datasets | $D$ | $N_{train}$ | $N_{test}$ | $F$ | $L$ | $\bar{L}$ | $\tilde{L}$ |
|---|---|---|---|---|---|---|---|
| AT (S) | 14828 | 13346 | 1482 | 63004 | 14365 | 13.15 | 13.5 |
| Full (S) | 14828 | 13346 | 1482 | 148330 | 14365 | 13.15 | 13.5 |
| AT (L) | 257590 | 231831 | 25759 | 188693 | 22881 | 13.34 | 150 |
| Full (L) | 257590 | 231831 | 25759 | 669999 | 22881 | 13.34 | 150 |

Table 4.1: Statistics of the Datasets. $D$ is the total number of documents; $N_{train}$is the number of training documents; $N_{test}$ is the number of test documents; $F$ represents the total number of unique tokens in all documents; $L$ is total number of class labels; $\bar{L}$ is total number of labels per document; $\tilde{L}$ is total number of documents per label

- **Abstract and Title (Large)**: labelled documents from PMC Collection which contains the abstract and title only
- **Full Text (Large)**: labelled documents from PMC Collection which contains abstract, title , figure and table captions, and associated paragraphs

Table 4.1 provides statistical information for the described datasets. We considered our dataset with labels, which covered 28,939 MeSH terms in total. MeSH terms are organized in a "tree" hierarchical structure with 16 main branches shown in Figure 4.5. Each branch has many levels of sub-branches, and each MeSH term has a position in the hierarchy, an example is shown in Figure 4.6. We exploited MeSH in hierarchical structure and split them into 5 levels. The number of MeSH terms in the first, the second, the third, the fourth and the fifth level, are: 16, 120, 1903, 6,808, and 11,127, respectively.

### 4.3.2 Data Pre-processing

The full-text source files from PMC are in XML format. We retrieved article information (include PMID, abstract, title, captions and paragraphs) from the downloaded XML files and scripted MeSH terms for each article on PubMed by using PMID which is a unique identifier number used in PubMed for each article. An example [27] of our extracted data is shown below.

- **PMID**: 19333414
- **Title**: Investigation on the protective effect of $\alpha$-mannan against the DNA damage induced by aflatoxin $B_1$ in mouse hepatocytes
- **Abstract**: Aflatoxin B(1) is a contaminant of agricultural and dairy products that can be related to mutagenic and carcinogenic effects. In this report we explore the capacity of $\alpha$-mannan (Man) to reduce the DNA damage induced by AFB (1) in mouse hepatocytes.

A. Anatomy
B. Organisms
C. Diseases
D. Chemicals and Drugs
E. Analytical, Diagnostic and Therapeutic Techniques and Equipment
F. Psychiatry and Psychology
G. Phenomena and Processes
H. Disciplines and Occupations
I. Anthropology, Education, Sociology and Social Phenomena
J. Technology, Industry, Agriculture
K. Humanities
L. Information Science
M. Named Groups
N. Health Care
V. Publication Characteristics
Z. Geographicals

Figure 4.5: Main Branches in the MeSH Hierarchy

For this purpose we applied the comet assay to groups of animals which were first administered Man (100, 400 and 700 mg/kg, respectively) and 20 min later 1.0 mg/kg of AFB (1). Liver cells were obtained at 4, 10, and 16 h after the chemical administration and examined. The results showed no protection of the damage induced by AFB(1) with the low dose of the polysaccharide, but they did reveal antigenotoxic activity exerted by the two high doses. In addition, we induced a co-crystallization between both compounds, determined their fusion points and analyzed the molecules by UV spectroscopy. The obtained data suggested the formation of a supramolecular complex between AFB (1) and Man.

- **Captions**:
  - Antigenotoxic effect of $\alpha$-mannan (Man) against the DNA damage induced by aflatoxin B1 (AFB1) in mouse hepatocytes.
  - Results are the mean ± SD of 5 mice per group (100 nuclei per doses) statistically significant difference with respect to the value of the control groups and, with

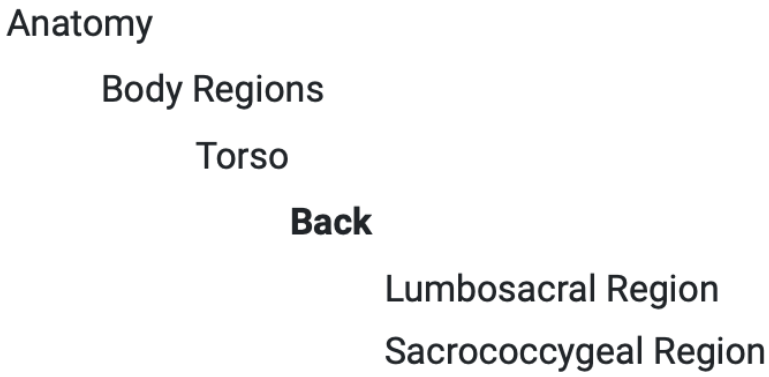


Figure 4.6: An Example of MeSH Hierarchy

respect to the value obtained in mice treated with AFB1 only. ANOVA and Student-Newman Keuls tests, $p \leq 0.05$.

- UV spectrum of the crystals formed by aflatoxin B1 (AFB1) plus Man (a), and the corresponding to AFB1 (b). The $\lambda$ values in the ordinate of (a) vary from 0.06 to 3.50, in (b) the values are from 0.06 to 1.00. The maximum peaks in (a) (224, 266, and 363 nm) show a correspondence with the detected in (b) (222, 264, and 358 nm).
- UV spectrum of the crystals formed by Man. The $\lambda$ values in the ordinate vary from 0.005 to 0.230. No peaks were found in the spectrum.

- **Paragraphs**:

  - Figure 1 shows the comet measurements obtained in our assay. To summarize, at the fourth hour of the schedule we found no significant DNA damage induced by the tested chemicals: mice treated with the control agents had a mean T/N index of 1.1, and animals treated with AFB1 as well as those administered with 100 mg/kg of Man plus the mutagen had a slight comet increase. At 10 h we found similar behaviour regarding the control and the Man treated animals, although in mice administered only AFB1 we determined a T/N index increase of about four times as much. With respect to the groups treated with the combination of chemicals, no protection was observed when the low dose of Man was applied. However, a clear antigenotoxic effect was found with the two high doses; particularly with 700 mg/kg of Man, the prevention of DNA damage was about 50%. Then, at 16 h, the genotoxicity of AFB1 and the protection exerted by Man continued although to a lesser extent. The higher level of DNA damage at 10 h with respect to that found

in the other evaluated times corresponds to the schedule where maximum breakage is present in our model, and it agrees with results reported for an acute comet assay with a single administration; moreover, because the assay also detects the repair process [17] which, in our case, can begin after 10 h of exposure to the chemicals.

– Figure 2a shows the UV spectrum of the crystals obtained. The presence of AFB1 in the mixture is clear, as indicated by its characteristic maximum peaks at 223, 261, and 364 nm, which were very similar to those detected in the spectrum of the mutagen in independent form (Figure 2b). Moreover, the spectrum obtained with the crystals differs sharply from that obtained with the polysaccharide alone, which showed no peaks in the range from 220 to 400 nm (Figure 3).

- **MeSH Terms**: Administration, Oral | Aflatoxins | Animals | DNA Damage | Hepatocytes | Mannans | Mice | Mutagens

In pre-processing, we first did word level tokenization on our input documents, which splits a document into a list of individual words. Then we further prepared our data by using following process:

- Set all characters to lowercase
- Convert numbers to "NUM"
- Convert percentage sign "%" to "PERCENTAGE"
- Convert chemical notations(i.e. $H_2O$) to "CHEM"
- Remove punctuation
- Convert relation symbols, namely "=", "<", ">", "≤", "≥", to "EQUAL", "LESS", "GREATER", "LessAndEqual", "GreaterAndEqual"

A comparison of input text and processed output in shown below:

- **Input Text**: Results are the mean ± SD of 5 mice per group (100 nuclei per doses) statistically significant difference with respect to the value of the control groups and, with respect to the value obtained in mice treated with AFB1 only. ANOVA and Student-Newman Keuls tests, $p \leq 0.05$.
- **Processed Output**: results are the mean sd of NUM mice per group NUM nuclei per doses statistically significant difference with respect to the value of the control groups and with respect to the value obtained in mice treated with CHEM only anova and student newman keuls tests p LessAndEqual NUM

After the above process, we utilized Keras [5] Tokenizer API to vectorize our data into a sequence of integers, and each integer represents the index of a token in the dictionary generated by dataset. An example outputs by Keras Tokenizer is shown below:

- **Pre-processed Data**: investigation on the protective effect of alpha mannan against the dna damage induced by aflatoxin CHEM CHEM in mouse hepatocytes
- **Tokenizer Output**: 2297, 22, 2, 1892, 102, 3, 1045, 11201, 304, 2, 119, 800, 112, 14, 18349, 31, 31, 6, 299, 3001

### 4.3.3 Generate Word Embeddings

We used pre-trained BioASQ word embedding vectors [11] to represent each token in the dictionary. Theses pre-trained word vectors are trained on 10,876,004 biomedical English abstracts from PubMed, and contains vectors of 1,701,632 distinct words. The dimensionality of the word vectors is 200.

### 4.3.4 Experiment Setup and Model Hyperparameters

In order to support our hypothesis that adding more information, specifically figure and table information, improves the performance of biomedical document classification, we perform our experiment in two parts: datasets (Abstract and Title (Small) and Abstract and Title (Large)) with abstract and title only have been fitted to single channel models, and datasets (Full Text (Small) and Full Text (Large)) with captions and paragraphs added have been fitted into multi-channel models.

The training of our proposed methods mentioned above is performed using binary cross-entropy as loss function on sigmoid classifier. We use sigmoid function to return the probability score of each class. The sigmoid function is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

And binary cross-entropy is formulated as:

$$H(q) = -\frac{1}{L}\sum_{i=1}^{L} y_i \cdot log(\sigma(y_i)) + (1 - y_i) \cdot log(1 - \sigma(y_i))$$

where $\sigma$ is sigmoid function, L is the total number of labels, $y_i$ is the original label of document $i$, and $\sigma(y_i)$ is predicted probability of label $y$ for document $i$. The sigmoid binary cross-entropy optimizes a label one-versus-all loss based on the max-entropy. Hyper-parameters are

chosen based on grid search and reference to previous work. In TextCNN model, we used filter windows of 3, 4, 5 with 128 filters each, dropout rate of 0.5. For XMLCNN model, we used filter windows of 2, 4, and 8 with 128 filters each, dropout rate of 0.5, k was 10 in k-chunk max-pooling, and the number of hidden unit was 512 at the bottleneck layer. In biLSTM model, we used 200 hidden units in biLSTM and followed by one fully connected layer. In convLSTM model, we used filter windows of 3, 4, and 5 with 128 filters each, dropout rate of 0.5, and 128 hidden units in LSTM, 256 hidden unit at fully connect layer. We used batch size 10 and 64 for Full Text (Large) and other datasets (Abstract and Title (Small), Full Text (Small), Abstract and Title (Large)), and learning of 0.001 for each model. We ran 20 epochs on TextCNN and XMLCNN model, and ran 5 epochs on biLSTM and convLSTM due to the running time. For each dataset, we used 90% of the data as training set and 10% to test the performance of the model. We randomly reserved 20% of the training data as the validation set, and remaining 80% is used for training the model. All experiments are performed on the Nvidia GeForce 1080Ti GPU. Models for Full Text(Large) is performed on 2 GPUs and other datasets are performed on a single GPU.

## 4.4 Experiments

In this section, we first provide in the next subsection the evaluation metrics that are used in our assessments, and then discuss the experimental results in the subsections following that.

### 4.4.1 Evaluation Metrics

In automatic MeSH indexing, even if the label space is very large, only relatively few MeSH terms match each document. To evaluate the performance of the models we discussed in Section 4.2, we used the evaluation metrics mentioned in Section 3.5, namely hamming loss, example-based, label-based and hierarchical based precision, recall, and F-score, precision at top $k$ ($p@k$), and normalized discounted cumulative gain at top $k$ ($nDCG@k$). For ranking-based evaluation, we used $k \in \{1, 3, 5, 10, 15\}$ for $p@k$, and $k \in \{1, 3, 5\}$ for $nDCG@k$. For example-based, label-based and hierarchical-based evaluation, we returned the top $k$ predicted labels and did the calculation accordingly, where $k \in \{5, 10, 15\}$. The example-based, ranking-based, and hierarchical evaluation metrics are calculated for each document and an average score over all documents in the test set is returned. Likewise, the label-based evaluation is calculated for each label and an average score over all labels in the test set is returned.

### 4.4.2 Results

We first conducted our experiment on datasets with title and abstract only, and then we further conducted the experiment over datasets that include captions and paragraphs as well. As will be pointed out and discussed below, the (multichannel) TextCNN model outperformed the other models. We wanted to investigate the importance of having a multichannel rather than single channel model. So, with the TextCNN model, we also performed an extra experiment with datasets that contain the full text information: we passed word embeddings for titles, abstracts, captions and paragraphs to a single channel TextCNN. Four datasets have been used in the experiments. The small dataset is SETC2015, mentioned in Section 4.3.1, which derives AbstractAndTitle (Small) and FullText (Small). AbstractAndTitle (Small) represents SETC2015 based on title and abstract only, and FullText (Small) also includes figure and table related captions as well as paragraphs. The large dataset is the PMC collection, discussed in Section 4.3.1, which also derives two datasets: AbstractAndTitle (Large) and FullText (Large). AbstractAndTitle (Large) contains the abstracts and titles only, and FullText (Large) adds the captions and paragraphs. The running time for the LSTM based classifiers, i.e., biLSTM and convLSTM, are nearly 17 days for one epoch on the FullText (Large) dataset. Due to this computation time, the two LSTM based models are not tested on the FullText (Large) dataset. Since LSTM based models target on sequence patterns they cannot be parallelized; therefore, acceleration of LSTM models has limitations so they suffer from high cost on large datasets.

The performance of precision@$k$ ($p$@$k$) and $nDCG$@$k$ among various models on all four datasets are shown in Tables 4.2 and 4.3. Each row in the table compares all methods on a specific dataset, where the best result over all models is printed as bold. Underlined items are the best result when comparing the AbstractAndTitle and the FullText datasets for each evaluation metric for each model for the small and large datasets

Comparisons are also made between the AbstractAndTitle (Small) and FullText (Small) datasets and theAbstractAndTitle (Large) and FullText (Large) datasets for each model. The best performance in each evaluation metric for each of these comparisons is underlined. The performance of $nDCG$@$k$ has the same trend as $p$@$k$, and thus $nDCG$@10 and $nDCG$@15 were omitted in Table 4.3.

Investigating Tables 4.2 and 4.3 led to the following:

- Comparisons across classifiers led to the following conclusions:
  - The TextCNN model performed the best among the four models.
  - The models that include convolution (TextCNN, XMLCNN, and convLSTM) perform better than the solely LSTM based model (biLSTM) suggesting that MeSH term indexing is more word-based than sentence-based.

| | | Methods | | | |
|---|---|---|---|---|---|
| Datasets | Metrics | TextCNN | XMLCNN | biLSTM | convLSTM |
| AbstractAndTitle (Small) | $p@1$ | **0.76197** | 0.66217 | 0.69589 | 0.67701 |
| | $p@3$ | **0.58283** | 0.50124 | 0.45493 | 0.47516 |
| | $p@5$ | **0.47633** | 0.40984 | 0.38732 | 0.39056 |
| | $p@10$ | **0.39641** | 0.30883 | 0.30329 | 0.30642 |
| | $p@15$ | **0.37281** | 0.27728 | 0.26323 | 0.27283 |
| FullText (Small) | $p@1$ | **0.80512** | 0.67296 | 0.66689 | 0.65678 |
| | $p@3$ | **0.62980** | 0.51090 | 0.46168 | 0.45448 |
| | $p@5$ | **0.52057** | 0.40850 | 0.38233 | 0.37559 |
| | $p@10$ | **0.41958** | 0.31431 | 0.30469 | 0.29200 |
| | $p@15$ | **0.39587** | 0.28940 | 0.25829 | 0.26118 |
| AbstractAndTitle (Large) | $p@1$ | **0.87600** | 0.81032 | 0.72476 | 0.71905 |
| | $p@3$ | **0.70951** | 0.63501 | 0.51066 | 0.50545 |
| | $p@5$ | **0.60532** | 0.51975 | 0.41922 | 0.41641 |
| | $p@10$ | **0.51000** | 0.42509 | 0.32699 | 0.32480 |
| | $p@15$ | **0.47127** | 0.39069 | 0.28286 | 0.28083 |
| FullText (Large) | $p@1$ | **0.87907** | 0.81609 | - | - |
| | $p@3$ | **0.72139** | 0.63781 | - | - |
| | $p@5$ | **0.61479** | 0.52601 | - | - |
| | $p@10$ | **0.51793** | 0.43378 | - | - |
| | $p@15$ | **0.48009** | 0.39977 | - | - |

Table 4.2: Evaluation of the Four Models: Results of $p@k$. Boldface indicates the best $p@k$ results for each $k$ across all models. Underlined items are the best result when comparing the AbstractAndTitle and the FullText datasets for each evaluation metric for each model for the small and large datasets

  - The models that include convolution show consistent (or almost consistent) improvements when adding more information. The LSTM based model is less consistent.

- Comparisons within the same dataset but with different contents led to the following conclusions:

  - Multi-channel models perform better with the CNN based models, which indicates captions and paragraphs provide useful information for the CNN classifiers.
  - In the best performing model, TextCNN, when comparing the results, using data that comes from abstracts and titles only with the full text data, the Small dataset shows the greater improvement, approximately 2-5 percentage points for each $p@k$ and each $nDCG@k$ value. The improvement for the Large dataset is typically closer

| | | Methods | | | |
|---|---|---|---|---|---|
| Datasets | Metrics | TextCNN | XMLCNN | biLSTM | convLSTM |
| AbstractAndTitle (Small) | *nDCG*@1 | **0.76196** | 0.66217 | 0.69588 | 0.67700 |
| | *nDCG*@3 | **0.62306** | 0.53812 | 0.50582 | 0.51683 |
| | *nDCG*@5 | **0.53918** | 0.46519 | 0.44659 | 0.44854 |
| FullText (Small) | *nDCG*@1 | **0.80512** | 0.67296 | 0.66689 | 0.65677 |
| | *nDCG*@3 | **0.66982** | 0.54769 | 0.50255 | 0.49689 |
| | *nDCG*@5 | **0.58409** | 0.46716 | 0.43766 | 0.43217 |
| AbstractAndTitle (Large) | *nDCG*@1 | **0.87600** | 0.81031 | 0.72476 | 0.71904 |
| | *nDCG*@3 | **0.74737** | 0.67468 | 0.55394 | 0.54950 |
| | *nDCG*@5 | **0.66640** | 0.58510 | 0.48040 | 0.47756 |
| FullText (Large) | *nDCG*@1 | **0.87907** | 0.81609 | - | - |
| | *nDCG*@3 | **0.75737** | 0.67778 | - | - |
| | *nDCG*@5 | **0.67521** | 0.58053 | - | - |

Table 4.3: Evaluation of the Four Models: Results of *nDCG@k*. Boldface indicates the best *nDCG@k* results for each $k$ across all models. Underlined items are the best result when comparing the AbstractAndTitle and the FullText datasets for each evaluation metric for each model for the small and large datasets

to 1 percentage point. It should be noted that the Large dataset has a somewhat higher, thus more difficult to improve upon, abstract and title baseline for each metric (10 percentage points or more than the Small dataset). This could be that the Small and Large datasets were generated from documents with different attributes. We have not investigated this possibility.

- Comparisons over different sizes of the dataset led to the following conclusion:
  - Comparing AbstractAndTitle (Large) to AbstractAndTitle (Small) and FullText (Large) to FullText (Small) shows an approximately 7-13 percentage point improvement for each *p@k* and *nDCG@k*. This result indicates that more training examples simply gives better models, so the extra information provided by the new data sources does not have as significant an effect as the increase in the number of training examples.

Table 4.4 to Table 4.7 report the performance of flat and hierarchical evaluations of all presented models on all datasets giving a further assessment of introducing the extra information sources. We selected to return top $k$ labels from the label sets, where $k \in \{5, 10, 15\}$. We evaluated our models on example-based, label-based and hierarchical evaluations. *EBP*, *EBR* and *EBF* calculate models' performance over each document; *MaP*, *MaF*, *MiP*, and *MiF* evaluate models over all labels; HP and HR measures model performance based on gold-standard and

| | | Metrics | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Datasets | top_*k*_selected | *EBP* | *EBR* | *EBF* | *MiP* | *MaP* | *MiF* | *MaF* | $HP_1$ | $HR_1$ | $HP_2$ | $HR_2$ |
| AbstractAndTitle (Small) | @5 | 0.47684 | 0.18676 | 0.26091 | 0.49753 | 0.49941 | 0.45628 | 0.47804 | 0.52049 | 0.13520 | 0.60490 | 0.14347 |
| | @10 | 0.34863 | 025301 | 0.28808 | 0.47330 | 0.49814 | 0.45588 | 0.47810 | 0.38057 | 0.20811 | 0.45634 | 0.22511 |
| | @15 | 0.30089 | 0.27835 | 0.28789 | 0.45798 | 0.49706 | 0.45283 | 0.47807 | 0.32516 | 0.24080 | 0.39877 | 0.26243 |
| FullText (Small) | @5 | 0.52113 | 0.20028 | 0.28156 | 0.50257 | 0.49832 | 0.45933 | 0.47790 | 0.53849 | 0.16136 | 0.60841 | 0.17620 |
| | @10 | 0.37723 | 0.27009 | 0.30927 | 0.47832 | 0.49539 | 0.45976 | 0.47740 | 0.38576 | 0.23705 | 0.44186 | 0.26472 |
| | @15 | 0.32464 | 0.29827 | 0.30945 | 0.46255 | 0.49373 | 0.45700 | 0.47721 | 0.32605 | 0.26791 | 0.38026 | 0.30378 |
| AbstractAndTitle (Large) | @5 | 0.60625 | 0.23896 | 0.33234 | 0.57454 | 0.50219 | 0.36413 | 0.39817 | 0.63210 | 0.19579 | 0.68524 | 0.19668 |
| | @10 | 0.46160 | 0.33370 | 0.38038 | 0.47892 | 0.49987 | 0.40743 | 0.40113 | 0.47824 | 0.30389 | 0.53204 | 0.31513 |
| | @15 | 0.40430 | 0.37220 | 0.38565 | 0.43378 | 0.49733 | 0.41410 | 0.40280 | 0.41333 | 0.35164 | 0.46788 | 0.37134 |
| FullText (Large) | @5 | 0.61569 | 0.24341 | 0.33833 | 0.58120 | 0.50253 | 0.36939 | 0.39963 | 0.64027 | 0.19934 | 0.70217 | 0.19967 |
| | @10 | 0.46767 | 0.33916 | 0.38614 | 0.48437 | 0.49992 | 0.41285 | 0.40316 | 0.49180 | 0.30675 | 0.55795 | 0.31847 |
| | @15 | 0.41073 | 0.37868 | 0.39212 | 0.44003 | 0.49679 | 0.42042 | 0.40511 | 0.42621 | 0.35704 | 0.49112 | 0.37776 |

Table 4.4: Flat and Hierarchical Measures for TextCNN in Different Datasets. top_*k*_selected indicates the top *k* labels returned by the classifier; EBP indicates example-based precision; EBR indicates example-based recall; EBF indicates example-based F-score; MiP indicates micro-average precision; MaP indicates macro-average precision; MiF indicates micro-average F-score; MaF indicates macro-average F-score; HP indicates hierarchical precision; HR indicates hierarchical recall; *m* denotes the distance from the original label to its ancestors and descendants in $HP_m$ and $HR_m$

| | | Metrics | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Datasets | top_*k*_selected | *EBP* | *EBR* | *EBF* | *MiP* | *MaP* | *MiF* | *MaF* | $HP_1$ | $HR_1$ | $HP_2$ | $HR_2$ |
| AbstractAndTitle (Small) | @5 | 0.41042 | 0.15681 | 0.22097 | 0.48996 | 0.49996 | 0.44842 | 0.47792 | 0.42348 | 0.09548 | 0.47456 | 0.08615 |
| | @10 | 0.27218 | 0.19448 | 0.22300 | 0.45770 | 0.49972 | 0.44030 | 0.47793 | 0.30462 | 0.12003 | 0.35582 | 0.12076 |
| | @15 | 0.22643 | 0.20905 | 0.21638 | 0.43861 | 0.49947 | 0.43369 | 0.47793 | 0.23001 | 0.13844 | 0.28807 | 0.14397 |
| FullText (Small) | @5 | 0.40910 | 0.15367 | 0.21746 | 0.48985 | 0.49994 | 0.44834 | 0.47777 | 0.43655 | 0.09237 | 0.48340 | 0.08753 |
| | @10 | 0.27185 | 0.19175 | 0.22088 | 0.45834 | 0.49966 | 0.44065 | 0.47777 | 0.29177 | 0.12545 | 0.33252 | 0.14206 |
| | @15 | 0.22679 | 0.20725 | 0.21546 | 0.44001 | 0.49953 | 0.43465 | 0.47784 | 0.22611 | 0.14883 | 0.28784 | 0.1678 |
| AbstractAndTitle (Large) | @5 | 0.52054 | 0.20150 | 0.28186 | 0.51552 | 0.50020 | 0.32636 | 0.39558 | 0.53244 | 0.14821 | 0.59463 | 0.15162 |
| | @10 | 0.37936 | 0.27130 | 0.31055 | 0.41214 | 0.49778 | 0.35022 | 0.39638 | 0.38140 | 0.22483 | 0.43733 | 0.23671 |
| | @15 | 0.32734 | 0.30006 | 0.31144 | 0.36843 | 0.49513 | 0.35132 | 0.39668 | 0.32258 | 0.26073 | 0.37816 | 0.27932 |
| FullText (Large) | @5 | 0.52686 | 0.19920 | 0.27910 | 0.51285 | 0.50030 | 0.32472 | 0.39620 | 0.53730 | 0.14162 | 0.60720 | 0.13990 |
| | @10 | 0.37621 | 0.26844 | 0.30760 | 0.40985 | 0.49795 | 0.34835 | 0.39709 | 0.38466 | 0.21823 | 0.45025 | 0.22582 |
| | @15 | 0.32430 | 0.29719 | 0.30852 | 0.36646 | 0.49528 | 0.34953 | 0.39743 | 0.32487 | 0.25346 | 0.38872 | 0.26796 |

Table 4.5: Flat and Hierarchical Measures for XMLCNN in Different Datasets. top_*k*_selected indicates the top *k* labels returned by the classifier; EBP indicates example-based precision; EBR indicates example-based recall; EBF indicates example-based F-score; MiP indicates micro-average precision; MaP indicates macro-average precision; MiF indicates micro-average F-score; MaF indicates macro-average F-score; HP indicates hierarchical precision; HR indicates hierarchical recall; *m* denotes the distance from the original label to its ancestors and descendants in $HP_m$ and $HR_m$

predicted labels as well as their ancestors and descendants. CNN based models, i.e., TextCNN and XMLCNN, show support on our initial hypothesis, but RNN based models, i.e., biLSTM and convLSTM, do not. We are more interested in the results reported from TextCNN model, as it is the best performing model among all four presented models. We have showed flat and hierarchical evaluations over all presented models, but will only discuss the TextCNN model below. The exploration led to the following conclusions:

- CNN based models show that adding captions and paragraphs indeed provide valuable information in automatic MeSH indexing.

| | | Metrics | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Datasets | top_*k*_selected | *EBP* | *EBR* | *EBF* | *MiP* | *MaP* | *MiF* | *MaF* | $HP_1$ | $HR_1$ | $HP_2$ | $HR_2$ |
| AbstractAndTitle (Small) | @5 | 0.38788 | 0.14921 | 0.20993 | 0.48738 | 0.49998 | 0.44572 | 0.47764 | 0.35962 | 0.09429 | 0.36151 | 0.08436 |
| | @10 | 0.26689 | 0.19073 | 0.21875 | 0.45670 | 0.49992 | 0.43905 | 0.47771 | 0.28087 | 0.11067 | 0.29518 | 0.09449 |
| | @15 | 0.21827 | 0.20191 | 0.20880 | 0.43633 | 0.49985 | 0.43128 | 0.47775 | 0.19504 | 0.12627 | 0.23569 | 0.10919 |
| FullText (Small) | @5 | 0.38285 | 0.14556 | 0.20517 | 0.48689 | 0.49998 | 0.44536 | 0.47777 | 0.35835 | 0.09413 | 0.36919 | 0.08683 |
| | @10 | 0.26183 | 0.18495 | 0.21290 | 0.45644 | 0.49992 | 0.43852 | 0.47784 | 0.28152 | 0.11140 | 0.30056 | 0.09846 |
| | @15 | 0.21145 | 0.19398 | 0.20134 | 0.43574 | 0.49983 | 0.43024 | 0.47786 | 0.21827 | 0.11794 | 0.23837 | 0.10857 |
| AbstractAndTitle (Large) | @5 | 0.41991 | 0.16242 | 0.22740 | 0.44581 | 0.49999 | 0.28252 | 0.39523 | 0.39331 | 0.10021 | 0.40592 | 0.09209 |
| | @10 | 0.29005 | 0.20643 | 0.23683 | 0.33833 | 0.49993 | 0.28777 | 0.39534 | 0.31171 | 0.11911 | 0.33533 | 0.10522 |
| | @15 | 0.23814 | 0.21874 | 0.22685 | 0.28924 | 0.49985 | 0.27601 | 0.39541 | 0.20636 | 0.13980 | 0.23879 | 0.14646 |

Table 4.6: Flat and Hierarchical Measures for biLSTM in Different Datasets. top_*k*_selected indicates the top *k* labels returned by the classifier; EBP indicates example-based precision; EBR indicates example-based recall; EBF indicates example-based F-score; MiP indicates micro-average precision; MaP indicates macro-average precision; MiF indicates micro-average F-score; MaF indicates macro-average F-score; HP indicates hierarchical precision; HR indicates hierarchical recall; *m* denotes the distance from the original label to its ancestors and descendants in $HP_m$ and $HR_m$

| | | Metrics | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Datasets | top_*k*_selected | *EBP* | *EBR* | *EBF* | *MiP* | *MaP* | *MiF* | *MaF* | $HP_1$ | $HR_1$ | $HP_2$ | $HR_2$ |
| AbstractAndTitle (Small) | @5 | 0.39159 | 0.14984 | 0.21059 | 0.48786 | 0.49998 | 0.44589 | 0.47766 | 0.36694 | 0.09566 | 0.37573 | 0.08773 |
| | @10 | 0.27147 | 0.19219 | 0.22099 | 0.45767 | 0.49992 | 0.43965 | 0.47773 | 0.28829 | 0.11256 | 0.30727 | 0.09861 |
| | @15 | 0.22207 | 0.20334 | 0.21128 | 0.43741 | 0.49984 | 0.43197 | 0.47776 | 0.20923 | 0.12827 | 0.24484 | 0.14678 |
| FullText (Small) | @5 | 0.37571 | 0.14577 | 0.20418 | 0.48606 | 0.49998 | 0.44507 | 0.47789 | 0.35179 | 0.09227 | 0.36856 | 0.08728 |
| | @10 | 0.25765 | 0.18424 | 0.21109 | 0.45512 | 0.49992 | 0.43785 | 0.47795 | 0.27781 | 0.10824 | 0.30539 | 0.09829 |
| | @15 | 0.21133 | 0.19477 | 0.20180 | 0.43529 | 0.49984 | 0.43022 | 0.47798 | 0.19906 | 0.12125 | 0.24007 | 0.10991 |
| AbstractAndTitle (Large) | @5 | 0.41717 | 0.16159 | 0.22615 | 0.44401 | 0.49999 | 0.28125 | 0.39538 | 0.39200 | 0.09997 | 0.40737 | 0.09193 |
| | @10 | 0.28814 | 0.20544 | 0.23553 | 0.33661 | 0.49993 | 0.28625 | 0.39548 | 0.31169 | 0.11899 | 0.33776 | 0.10530 |
| | @15 | 0.23716 | 0.21800 | 0.22602 | 0.28818 | 0.49985 | 0.27489 | 0.39555 | 0.21815 | 0.13815 | 0.25314 | 0.14638 |

Table 4.7: Flat and Hierarchical Measures for convLSTM in Different Datasets. top_*k*_selected indicates the top *k* labels returned by the classifier; EBP indicates example-based precision; EBR indicates example-based recall; EBF indicates example-based F-score; MiP indicates micro-average precision; MaP indicates macro-average precision; MiF indicates micro-average F-score; MaF indicates macro-average F-score; HP indicates hierarchical precision; HR indicates hierarchical recall; *m* denotes the distance from the original label to its ancestors and descendants in $HP_m$ and $HR_m$

- When comparing AbstractAndTitle to FullText Multichannel in the Small and Large datasets, we see an approximate .5-5 percentage point improvement in all of the measures except *MaP*. Most importantly, there is improvement in precision without a decrease in recall. The obtained results further suggest that our hypothesis that adding captions and paragraphs indeed provides valuable information in automatic MeSH indexing.

- Comparing *EBP*, which is the same as $HP_0$, with the *HP* values, an approximate 1-5 percentage point improvement in all cases at $HP_1$ and an approximate 6-13 percentage point improvement in all cases at $HP_2$ can be seen. These observations indicate that some of the predicted MeSH terms are not exactly the same as the gold-standard labels, but the model has suggested MeSH terms that are in the correct branch of the MeSH term hierarchy. With this latter observation we have investigated how the predicted results

correspond to the gold-standard results.

An in-depth analysis of the hierarchical evaluation on the AbstractAndTitle (Large) and FullText (Large) datasets are reported in Table 4.8. We have computed the average number of gold-standard MeSH term labels in common with the predicted labels including $m$ levels up and $n$ levels down over all documents, where $m \in \{0, 1, 2\}$ and $n \in \{0, 1, 2\}$. Each row in the table compares model performance at a certain MeSH hierarchy, where $C_m$ indicates the predicted label augmented with children with distance $m$, and $P_n$ is predicted label augmented with parents with distance $n$. As an example: $C_0\_P_1$ on AT (L) with the top 5 predicted labels indicates that if the predicted labels are augmented with their parents with distance *1*, the number of common labels between true labels and predicted ones will increase on average by *0.0256* over all documents in the test set. For each top_$k$_predicted labels returned by the TextCNN model, comparisons within the same dataset but expanded augmentations show that the number of common MeSH terms between the gold-standard and predicted ones increase in all cases (except two instances of an increased window size for the multichannel TextCNN, the single channel TextCNN when augmenting only with parent labels, and the AT (L) dataset for top_5_predicted). Looking only at each column, this can be more than a ten-fold increase. Comparing AT with Full multichannel TextCNN the increase is approximately three times when adding captions and related texts. This observation gives us confidence in concluding that the multichannel TextCNN model gives MeSH terms that are in the correct branch of the MeSH hierarchy and adding figure captions and related texts does provide valuable improvement in automatic MeSH indexing.

The $p@k$ and $nDCG@k$ performance of the single channel TextCNN and the multichannel TextCNN on all four datasets is summarized in Table 4.9. Each row in the table compares all datasets on a specific metric, where the best score for each metric (the Small and Large datasets being observed separately) is in boldface. In addition to noting the best scores, it should be observed that the single channel TextCNN on the Full (Large) dataset performs worse than TextCNN on the AbstractAndTitle (Large) dataset. These results clearly indicate

| | top_5_predicted | | top_10_predicted | | top_15_predicted | |
|---|---|---|---|---|---|---|
| | AbstractAndTitle (Large) | FullText (Large) | AbstractAndTitle (Large) | FullText (Large) | AbstractAndTitle (Large) | FullText (Large) |
| $C_0\_P_1$ | 0.0256 | 0.0748 | 0.0236 | 0.1107 | 0.0290 | 0.1424 |
| $C_1\_P_0$ | 0.1119 | 0.4551 | 0.2420 | 0.8545 | 0.2791 | 1.0379 |
| $C_0\_P_2$ | 0.2387 | 0.4904 | 0.2933 | 0.6983 | 0.3405 | 0.8269 |
| $C_2\_P_0$ | 0.1591 | 0.6528 | 0.3202 | 1.1166 | 0.3681 | 1.3542 |
| $C_2\_P_1$ | 0.1847 | 0.7276 | 0.3438 | 1.2267 | 0.3971 | 1.4954 |
| $C_1\_P_2$ | 0.3506 | 0.9455 | 0.5354 | 1.5528 | 0.6196 | 1.8649 |

Table 4.8: Hierarchical Analysis of TextCNN Results. top_$k$_selected indicates the top $k$ labels returned by the classifier; $C_m\_P_n$ indicates that the predicted labels are compared to the gold-standard labels augmented with their children with distance $m$ and ancestors with distance $n$

| Metrics | | Datasets and Models | | | | | |
|---|---|---|---|---|---|---|---|
| | | AbstractAndTitle (S) Single_Channel | Full (S) Single_Channel | Full (S) Multichannel | AbstractAndTitle (L) Single_Channel | Full (L) Single_Channel | Full (L) Multichannel |
| *p@k* | *p*@1 | 0.76197 | 0.78220 | **0.80512** | 0.87600 | 0.72305 | **0.87907** |
| | *p*@3 | 0.58283 | 0.59699 | **0.62980** | 0.70951 | 0.51016 | **0.72139** |
| | *p*@5 | 0.47633 | 0.48901 | **0.52057** | 0.60532 | 0.41908 | **0.61479** |
| | *p*@10 | 0.39641 | 0.38815 | **0.41958** | 0.51000 | 0.32631 | **0.51793** |
| | *p*@15 | 0.37281 | 0.35910 | **0.39587** | 0.47127 | 0.28318 | **0.48009** |
| *nDCG@k* | *nDCG*@1 | 0.76197 | 0.78219 | **0.80512** | 0.87600 | 0.72305 | **0.87907** |
| | *nDCG*@3 | 0.62306 | 0.63744 | **0.66982** | 0.74737 | 0.55327 | **0.75737** |
| | *nDCG*@5 | 0.53918 | 0.55227 | **0.58409** | 0.66640 | 0.48009 | **0.67521** |

Table 4.9: Results for TextCNN in *p@k* and *nDCG@k*. Boldface indicates the best result for each metric on the Small (S) and Large (L) datasets.

that when dealing with the same dataset, multi-channel TextCNN performs the best, which indicates that integrating captions and paragraphs through a separate channel indeed helps to improve the performance of classification. Also, the multi-channel TextCNN outperforms the single channel TextCNN, suggesting that the multi-channel TextCNN architecture has an advantage over the single channel TextCNN architecture. The reason for this could be that the single channel model misses some important features in the captions and related paragraphs in the convolutional and pooling layers. To be more explicit, the single channel model may be extracting insignificant features in the convolutional layer from which the max-pooling layer can take only one value in each filter.

The final analysis of the TextCNN model is to compare its performance with the performance reported in previous research. Normally, a quantitative comparison requires a number of features of the comparison to be the same: similar task, and similar amount and type of training data. Typically, this is made possible because the research community has reached a consensus on a set of training data and the task to be performed. Because of the immaturity of this research area, i.e., the comparison of automatic MeSH term indexing to human indexing, a standard dataset has not yet been established, so a quantitive comparison to the existing work is impracticable at this time. In addition the work that has been presented here focusses on training neural models using a novel dataset that contains information in addition to the abstract and title information used in previous work. So instead of a quantitative comparison, we have done a qualitative comparison with a Deep convolution model proposed by Gargiulo et al. [7]. They have used the titles and abstracts of 11,150,090 articles to train their classifiers. We did a qualitative comparison using *EBP*, *EBR* and *EBF*. Results are shown in Table 4.10.

The 11,150,090 papers that the Deep convolution model trains on are almost 40 times more documents compared to our Large dataset. This model outperforms our multichannel model on precision but has a low recall, which means that their model has a high false negative rate. Our TextCNN model improves both the precision and recall when moving from single channel to multichannel and has a better F-score compared to the Deep convolution model.

| Models | *EBP* | *EBR* | *EBF* |
|---|---|---|---|
| TextCNN (abstract and title only) | 0.4043 | 0.3722 | 0.3856 |
| Multichannel TextCNN | 0.4107 | 0.3787 | 0.3921 |
| Deep convolution model [7] | 0.7211 | 0.1312 | 0.2220 |

Table 4.10: Comparison with Existing Models. *EBP* indicates example-based precision; *EBR* indicates example-based recall; *EBF* indicates example-based F-score

| Models | *MiF* | *EBP* | *EBR* | *EBF* | *MaP* | *MaR* | *MaF* | *MiP* | *MiR* |
|---|---|---|---|---|---|---|---|---|---|
| TextCNN | 0.3909 | 0.4459 | 0.2163 | 0.2797 | 0.4994 | - | 0.4377 | 0.4902 | - |
| Multichannel TextCNN | 0.3927 | 0.4508 | 0.2183 | 0.2825 | 0.4995 | - | 0.4382 | 0.4925 | - |
| ceb1 | 0.6561 | 0.6900 | 0.6371 | 0.6449 | 0.6237 | 0.5146 | 0.5090 | 0.6838 | 0.6306 |

Table 4.11: Test on BioASQ Test Cases. $p@k$ indicates precision at $k$; *MiF* indicates micro-average F-score; *EBP* indicates example-based precision; *EBR* indicates example-based recall; *EBF* indicates example-based F-score; *MaP* indicates macro-average precision; *MaR* indicates macro-average recall; *MaF* indicates macro-average F-score; *MiP* indicates micro-average precision; *MiR* indicates micro-average recall

We also tested our model on a set of BioASQ challenge test documents, dry run 2 in 2018, in particular. This dataset is comprised of 9715 documents. Each document has a PMID, and abstract and title. In order to obtain the MeSH terms for these documents, we scraped the PubMed site using the 9715 PMIDs. We compared our best model, TextCNN, with one of the participating models, ceb1. Results of ceb1's performance are provided on BioASQ's webpage[6]. Table 4.11 provides a summary of the *MiF*, *EBP*, *EBR*, *EBF*, *MaP*, *MaR*, *MaF*, *MiP*, and *MiR* results on this set of test cases. This comparison needs to take into account some important factors. Firstly, ceb1 is trained on 13,486,072 documents which is about 40 times more articles compared to our training set. Having more data to train on is an important factor in the quality of the trained model. Secondly, we are using a model trained on a possibly different genre of biomedical articles. We did not compare the 9715 articles with our training dataset because this is beyond the scope of this thesis. It is assumed that the test documents for the BioASQ dry run 2 are from a similar genre as some of the documents used for training by ceb1. Thirdly, it is difficult to compare model performance when the models are trained on different datasets since the quality of the transfer of knowledge is difficult to quantify. When comparing TextCNN and Multichannel TextCNN, we see a slight improvement in all measures, which is what is expected given the results discussed earlier in this thesis.

In all comparisons it is important to report important aspects of the model testing. In this last comparison, our test model was the multichannel TextCNN. The only aspect that was different from the normal multichannel TextCNN model testing was that we supplied the

[6]http://participants-area.bioasq.org/results/7a/

abstract and title word embeddings to both channels of the multichannel TextCNN model, since the BioASQ test cases only have abstract and title provided and the multichannel TextCNN model requires input to both channels. For purposes of understanding the ceb1 model, we unfortunately have no knowledge of the design of ceb1 (ceb1 is the only model with results presented on the dry run 2 test set).

MeSHProbeNet [42], the winner of the BioASQ challenge for the MeSH indexing task in 2018, is a self-attentive deep neural network. The model uses a bidirectional RNN followed by self-attentive MeSH probes to extract useful features, and finally a multi-view neural classifier generates a set of relevant MeSH terms. The self-attentive layer weights the input sequence by assigning attention scores and interprets word importance. MeSHprobeNet and Multichannel TextCNN have the same intuition for solving the MeSH indexing problem by using word level features, but extract word importance using different approaches.

# Chapter 5

# Conclusions

In this chapter, we conclude our work presented in this thesis by summarizing our work and discussing our future plans.

## 5.1 Conclusions

In this thesis we set out to explore the use of deep learning and two sources of training information for the MeSH indexing task. For the first item we investigated multichannel variants of four deep learning models. For the second, we generated four datasets for the training and testing of these models from two available datasets, one small and one much larger. For each of these two dataset sizes we curated two datasets, one containing the titles and abstracts of the documents found in the dataset, and one containing the titles, abstracts, figure and table captions, and paragraphs related to those figures and tables. This exploration led to the following contributions:

- We have implemented four multichannel deep neural network models to capture important features from the given documents. Two of them are CNN-based and two are LSTM-based. The multichannel TextCNN model handily outperforms the other three models on all performance measures. Looking only at TextCNN, it showed consistently improved performance for all performance measures on both the small and large datasets when provided with the complete information source. The CNN models converged in a few days of training for both the small and large datasets with the restricted and complete information sources. The LSTM-based models took significantly more time to converge, as was expected. We abandoned the training of the LSTM-based models on the large dataset with complete information because it was taking days of computing to complete only one epoch of the training.

That the TextCNN model showed the best performance was somewhat unexpected, as the best results for many NLP tasks are observed when LSTM-based models are used. The performance of TextCNN may suggest that the MeSH indexing task is more word-based than sentence-based.

The implementation code is publicly available.

- Notably, our experiments explored the use of training information obtained from different parts of the document. The results for TextCNN, the best performing model, indicate that adding more information sources indeed improves performance. This observation supports the initial hypothesis that figure and table captions as well as the associated paragraphs can provide valuable information for automatic MeSH indexing (at least for our multichannel TextCNN model).

- Our curated biomedical document datasets are available to the research community. To the best of our knowledge, the dataset comprising 257,590 full-text documents is the largest biomedical dataset curated for the MeSH indexing task, up to now.

We curated our dataset from PubMed Central, which is a database that contains a large number of free full text biomedical documents. Our dataset might be biased as some biomedical articles in certain fields never release free version.

## 5.2 Future Work

In this thesis, we mainly focused on finding a classifier to capture important features in the document. We manually set the number of MeSH terms returned from the model, i.e., in our thesis, we asked our model to return the top $k$ predicted MeSH terms, where $k \in \{5, 10, 15\}$. In the future, we plan to improve our model by implementing a ranking system module which can be added right after the classifier. The ranking module can automatically suggest the number of labels returned for each document, which could help the indexing system to return a more accurate set of MeSH terms.

We also intend to extend our experiments on different optimizers, learning rates and classifiers in order to improve the performance of our models.

This work is an attempt to explore classifiers that can help to reduce the workload for human annotators, not to replace their work. With this aim in mind, we also want to develop a tool which could help human annotators locate the places in the document that has text important for determining MeSH terms in order to improve the efficiency of MeSH indexing.

Additionally, we intend to work with biomedical researchers to see whether the TextCNN model can provide better MeSH terms for their research papers then are provided by NLM human annotators.

# Bibliography


[1] Artaches Ambartsoumian and Fred Popowich. Self-attention: A better building block for sentiment analysis neural network classifiers. In *WASSA@EMNLP*, 2018.

[2] Alan R Aronson and François-Michel Lang. An overview of MetaMap: historical perspective and recent advances. *Journal of the American Medical Informatics Association*, 17(3):229–236, 05 2010.

[3] Alan R. Aronson, James G. Mork, Clifford W. Gay, Susanne M. Humphrey, and Willie J. Rogers. The nlm indexing initiative's medical text indexer. *Studies in health technology and informatics*, 107 Pt 1:268–72, 2004.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[5] François Chollet et al. Keras. `https://keras.io`, 2015.

[6] Dina Demner-Fushman and James G. Mork. Extracting characteristics of the study subjects from full-text articles. *AMIA ... Annual Symposium proceedings. AMIA Symposium*, 2015:484–91, 2015.

[7] Francesco Gargiulo, Stefano Silvestri, and Mario Ciampi. Deep convolution neural network for extreme multi-label text classification. In *Proceedings of the 11th International Joint Conference on Biomedical Engineering Systems and Technologies - Volume 5: AI4Health,*, pages 641–650. INSTICC, SciTePress, 2018.

[8] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction*. Springer, 2 edition, 2009.

[9] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[10] Indexer, Qiao Jin, Bhuwan Dhingra, and William W. Cohen. Attentionmesh : Simple , effective and interpretable automatic mesh. 2018.

[11] Ion Androutsopoulos Ioannis Pavlopoulos, Aris Kosmopoulos. Continuous space word vectors obtained by applying word2vec to abstracts of biomedical articles.

[12] Kalervo Järvelin and Jaana Kekäläinen. Ir evaluation methods for retrieving highly relevant documents. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 41–48. ACM, 2000.

[13] Antonio Jimeno-Yepes, James G. Mork, Dina Demner-Fushman, and Alan R. Aronson. A one-size-fits-all indexing method does not exist: Automatic selection based on meta-learning. *JCSE*, 6:151–160, 2012.

[14] Antonio Jimeno-Yepes, James G. Mork, Dina Demner-Fushman, and Alan R. Aronson. Comparison and combination of several mesh indexing approaches. *AMIA ... Annual Symposium proceedings. AMIA Symposium*, 2013:709–18, 2013.

[15] Rie Johnson and Tong Zhang. Effective use of word order for text categorization with convolutional neural networks. In *HLT-NAACL*, 2015.

[16] Rie Johnson and Tong Zhang. Semi-supervised convolutional neural networks for text categorization via region embedding. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 919–927. Curran Associates, Inc., 2015.

[17] Rie Johnson and Tong Zhang. Deep pyramid convolutional neural networks for text categorization. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 562–570. Association for Computational Linguistics, 2017.

[18] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. In *ACL*, 2014.

[19] Yoon Kim. Convolutional neural networks for sentence classification. In *EMNLP*, 2014.

[20] Olivier Bodenreider KinWah Fung. Utilizing the umls for semantic mapping between terminologies. *AMIA ... Annual Symposium proceedings. AMIA Symposium*, 2007.

[21] Aris Kosmopoulos, Ioannis Partalas, Eric Gaussier, Georgios Paliouras, and Ion Androutsopoulos. Evaluation measures for hierarchical classification: a unified view and novel approaches. *Data Mining and Knowledge Discovery*, 29(3):820–865, May 2015.

[22] Siwei Lai, Liheng Xu, Kang Liu, and Jian Zhao. Recurrent convolutional neural networks for text classification. In *AAAI*, 2015.

[23] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[24] Jimmy Lin and W. John Wilbur. Pubmed related articles: a probabilistic topic-based model for content similarity. *BMC Bioinformatics*, 8(1):423, Oct 2007.

[25] Jingzhou Liu, Wei-Cheng Chang, Yuexin Wu, and Yiming Yang. Deep learning for extreme multi-label text classification. In *SIGIR*, 2017.

[26] Ke Liu, Shengwen Peng, Junqiu Wu, ChengXiang Zhai, Hiroshi Mamitsuka, and Shanfeng Zhu. Meshlabeler: improving the accuracy of large-scale mesh indexing by integrating diverse evidence. *Bioinformatics*, 31 12:i339–47, 2015.

[27] Eduardo Osiris Madrigal-Santillán, José Antonio Morales-González, Manuel Sánchez-Gutiérrez, Alicia Reyes-Arellano, and Eduardo Madrigal-Bujaidar. Investigation on the protective effect of Îś-mannan against the dna damage induced by aflatoxin b1 in mouse hepatocytes. *International Journal of Molecular Sciences*, 10:395 – 406, 2009.

[28] Brian McFee and Gert R Lanckriet. Metric learning to rank. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 775–782, 2010.

[29] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[30] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.

[31] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *Advances in neural information processing systems*, pages 2204–2212, 2014.

[32] James G. Mork, Antonio Jimeno-Yepes, and Alan R. Aronson. The nlm medical text indexer system for indexing biomedical literature. In *BioASQ@CLEF*, 2013.

[33] Ronald A Rensink. The dynamic representation of scenes. *Visual cognition*, 7(1-3):17–42, 2000.

[34] Anthony Rios and Ramakanth Kavuluru. Convolutional neural networks for biomedical text classification: Application in indexing biomedical articles. In *Proceedings of the 6th ACM Conference on Bioinformatics, Computational Biology and Health Informatics*, BCB '15, pages 258–267, New York, NY, USA, 2015. ACM.

[35] Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45:2673–2681, November 1997.

[36] Holger Schwenk, Loïc Barrault, Alexis Conneau, and Yann LeCun. Very deep convolutional networks for text classification. In *EACL*, 2017.

[37] Lei Tang, Suju Rajan, and Vijay K. Narayanan. Large scale multi-label classification via metalabeler. In *WWW*, 2009.

[38] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis P. Vlahavas. Mining multi-label data. In *Data Mining and Knowledge Discovery Handbook*, 2010.

[39] Grigorios Tsoumakas, Manos Laliotis, Nikos Markantonatos, and Ioannis Vlahavas. Large-scale semantic indexing of biomedical publications at bioasq.

[40] Krina Vasa. Classification through statistical and machine learning methods : A survey. In *International Journal of Engineering Development and Research*, 2016.

[41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[42] Guangxu Xun, Kishlay Jha, Ye Yuan, Yaqing Wang, and Aidong Zhang. Meshprobenet: A self-attentive probe net for mesh indexing. *Bioinformatics*, 2019.

[43] Yiming Yang. An evaluation of statistical approaches to text categorization. *Information Retrieval*, 1(1):69–90, Apr 1999.

[44] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alexander J. Smola, and Eduard H. Hovy. Hierarchical attention networks for document classification. In *HLT-NAACL*, 2016.

[45] Chengxiang Zhai, Hiroshi Mamitsuka, Junqiu Wu, Ke Liu, Shanfeng Zhu, and Shengwen Peng. MeSHLabeler: improving the accuracy of large-scale MeSH indexing by integrating diverse evidence. *Bioinformatics*, 31(12):i339–i347, 06 2015.

[46] M. Zhang and Z. Zhou. A review on multi-label learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 26(8):1819–1837, Aug 2014.

[47] Peng Zhou, Wei Shi, Jun Tian, Zhenyu Qi, Bingchen Li, Hongwei Hao, and Bo Xu. Attention-based bidirectional long short-term memory networks for relation classification. In *ACL*, 2016.

# Appendix A

# Summaries of the Models for TextCNN

In the following, we provide system generated model summaries for TextCNN described in Chapter 4. The four summaries are TextCNN models training on AbstractAndTitle (Small), FullText (Small), AbstractAndTitle (Large) and FullText (Large) respectively.

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_1 (InputLayer) | (None, 694) | 0 | |
| embedding_1 (Embedding) | (None, 694, 200) | 12601000 | input_1[0][0] |
| conv1d_1 (Conv1D) | (None, 692, 128) | 76928 | embedding_1[0][0] |
| conv1d_2 (Conv1D) | (None, 691, 128) | 102528 | embedding_1[0][0] |
| conv1d_3 (Conv1D) | (None, 690, 128) | 128128 | embedding_1[0][0] |
| batch_normalization_1 (BatchNor | (None, 692, 128) | 512 | conv1d_1[0][0] |
| batch_normalization_2 (BatchNor | (None, 691, 128) | 512 | conv1d_2[0][0] |
| batch_normalization_3 (BatchNor | (None, 690, 128) | 512 | conv1d_3[0][0] |
| max_pooling1d_1 (MaxPooling1D) | (None, 1, 128) | 0 | batch_normalization_1[0][0] |
| max_pooling1d_2 (MaxPooling1D) | (None, 1, 128) | 0 | batch_normalization_2[0][0] |
| max_pooling1d_3 (MaxPooling1D) | (None, 1, 128) | 0 | batch_normalization_3[0][0] |
| concatenate_1 (Concatenate) | (None, 3, 128) | 0 | max_pooling1d_1[0][0]<br>max_pooling1d_2[0][0]<br>max_pooling1d_3[0][0] |
| flatten_1 (Flatten) | (None, 384) | 0 | concatenate_1[0][0] |
| dropout_1 (Dropout) | (None, 384) | 0 | flatten_1[0][0] |
| dense_1 (Dense) | (None, 28940) | 11141900 | dropout_1[0][0] |

Total params: 24,052,020
Trainable params: 11,450,252
Non-trainable params: 12,601,768

Figure A.1: System Generated Summary for TextCNN Model on AbstractAndTitle (Small) Dataset

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_1 (InputLayer) | (None, 742) | 0 | |
| input_2 (InputLayer) | (None, 12393) | 0 | |
| embedding_1 (Embedding) | (None, 742, 200) | 29666200 | input_1[0][0] |
| embedding_2 (Embedding) | (None, 12393, 200) | 29666200 | input_2[0][0] |
| conv1d_1 (Conv1D) | (None, 740, 128) | 76928 | embedding_1[0][0] |
| conv1d_2 (Conv1D) | (None, 739, 128) | 102528 | embedding_1[0][0] |
| conv1d_3 (Conv1D) | (None, 738, 128) | 128128 | embedding_1[0][0] |
| conv1d_4 (Conv1D) | (None, 12391, 128) | 76928 | embedding_2[0][0] |
| conv1d_5 (Conv1D) | (None, 12390, 128) | 102528 | embedding_2[0][0] |
| conv1d_6 (Conv1D) | (None, 12389, 128) | 128128 | embedding_2[0][0] |
| batch_normalization_1 (BatchNor | (None, 740, 128) | 512 | conv1d_1[0][0] |
| batch_normalization_2 (BatchNor | (None, 739, 128) | 512 | conv1d_2[0][0] |
| batch_normalization_3 (BatchNor | (None, 738, 128) | 512 | conv1d_3[0][0] |
| batch_normalization_4 (BatchNor | (None, 12391, 128) | 512 | conv1d_4[0][0] |
| batch_normalization_5 (BatchNor | (None, 12390, 128) | 512 | conv1d_5[0][0] |
| batch_normalization_6 (BatchNor | (None, 12389, 128) | 512 | conv1d_6[0][0] |
| max_pooling1d_1 (MaxPooling1D) | (None, 1, 128) | 0 | batch_normalization_1[0][0] |
| max_pooling1d_2 (MaxPooling1D) | (None, 1, 128) | 0 | batch_normalization_2[0][0] |
| max_pooling1d_3 (MaxPooling1D) | (None, 1, 128) | 0 | batch_normalization_3[0][0] |
| max_pooling1d_4 (MaxPooling1D) | (None, 1, 128) | 0 | batch_normalization_4[0][0] |
| max_pooling1d_5 (MaxPooling1D) | (None, 1, 128) | 0 | batch_normalization_5[0][0] |
| max_pooling1d_6 (MaxPooling1D) | (None, 1, 128) | 0 | batch_normalization_6[0][0] |
| concatenate_1 (Concatenate) | (None, 3, 128) | 0 | max_pooling1d_1[0][0]<br>max_pooling1d_2[0][0]<br>max_pooling1d_3[0][0] |
| concatenate_2 (Concatenate) | (None, 3, 128) | 0 | max_pooling1d_4[0][0]<br>max_pooling1d_5[0][0]<br>max_pooling1d_6[0][0] |
| concatenate_3 (Concatenate) | (None, 6, 128) | 0 | concatenate_1[0][0]<br>concatenate_2[0][0] |
| flatten_1 (Flatten) | (None, 768) | 0 | concatenate_3[0][0] |
| dropout_1 (Dropout) | (None, 768) | 0 | flatten_1[0][0] |
| dense_1 (Dense) | (None, 28940) | 22254860 | dropout_1[0][0] |

Total params: 82,205,500
Trainable params: 22,871,564
Non-trainable params: 59,333,936

Figure A.2: System Generated Summary for TextCNN Model on FullText (Small) Dataset

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_1 (InputLayer) | (None, 1461) | 0 | |
| embedding_1 (Embedding) | (None, 1461, 200) | 37738800 | input_1[0][0] |
| conv1d_1 (Conv1D) | (None, 1459, 128) | 76928 | embedding_1[0][0] |
| conv1d_2 (Conv1D) | (None, 1458, 128) | 102528 | embedding_1[0][0] |
| conv1d_3 (Conv1D) | (None, 1457, 128) | 128128 | embedding_1[0][0] |
| batch_normalization_1 (BatchNor | (None, 1459, 128) | 512 | conv1d_1[0][0] |
| batch_normalization_2 (BatchNor | (None, 1458, 128) | 512 | conv1d_2[0][0] |
| batch_normalization_3 (BatchNor | (None, 1457, 128) | 512 | conv1d_3[0][0] |
| max_pooling1d_1 (MaxPooling1D) | (None, 1, 128) | 0 | batch_normalization_1[0][0] |
| max_pooling1d_2 (MaxPooling1D) | (None, 1, 128) | 0 | batch_normalization_2[0][0] |
| max_pooling1d_3 (MaxPooling1D) | (None, 1, 128) | 0 | batch_normalization_3[0][0] |
| concatenate_1 (Concatenate) | (None, 3, 128) | 0 | max_pooling1d_1[0][0]<br>max_pooling1d_2[0][0]<br>max_pooling1d_3[0][0] |
| flatten_1 (Flatten) | (None, 384) | 0 | concatenate_1[0][0] |
| dropout_1 (Dropout) | (None, 384) | 0 | flatten_1[0][0] |
| dense_1 (Dense) | (None, 28940) | 11141900 | dropout_1[0][0] |

Total params: 49,189,820
Trainable params: 11,450,252
Non-trainable params: 37,739,568

Figure A.3: System Generated Summary for TextCNN Model on AbstractAndTitle (Large) Dataset

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_1 (InputLayer) | (None, 1542) | 0 | |
| input_2 (InputLayer) | (None, 57340) | 0 | |
| embedding_1 (Embedding) | (None, 1542, 200) | 134000000 | input_1[0][0] |
| embedding_2 (Embedding) | (None, 57340, 200) | 134000000 | input_2[0][0] |
| conv1d_1 (Conv1D) | (None, 1540, 128) | 76928 | embedding_1[0][0] |
| conv1d_2 (Conv1D) | (None, 1539, 128) | 102528 | embedding_1[0][0] |
| conv1d_3 (Conv1D) | (None, 1538, 128) | 128128 | embedding_1[0][0] |
| conv1d_4 (Conv1D) | (None, 57338, 128) | 76928 | embedding_2[0][0] |
| conv1d_5 (Conv1D) | (None, 57337, 128) | 102528 | embedding_2[0][0] |
| conv1d_6 (Conv1D) | (None, 57336, 128) | 128128 | embedding_2[0][0] |
| batch_normalization_1 (BatchNor | (None, 1540, 128) | 512 | conv1d_1[0][0] |
| batch_normalization_2 (BatchNor | (None, 1539, 128) | 512 | conv1d_2[0][0] |
| batch_normalization_3 (BatchNor | (None, 1538, 128) | 512 | conv1d_3[0][0] |
| batch_normalization_4 (BatchNor | (None, 57338, 128) | 512 | conv1d_4[0][0] |
| batch_normalization_5 (BatchNor | (None, 57337, 128) | 512 | conv1d_5[0][0] |
| batch_normalization_6 (BatchNor | (None, 57336, 128) | 512 | conv1d_6[0][0] |
| max_pooling1d_1 (MaxPooling1D) | (None, 1, 128) | 0 | batch_normalization_1[0][0] |
| max_pooling1d_2 (MaxPooling1D) | (None, 1, 128) | 0 | batch_normalization_2[0][0] |
| max_pooling1d_3 (MaxPooling1D) | (None, 1, 128) | 0 | batch_normalization_3[0][0] |
| max_pooling1d_4 (MaxPooling1D) | (None, 1, 128) | 0 | batch_normalization_4[0][0] |
| max_pooling1d_5 (MaxPooling1D) | (None, 1, 128) | 0 | batch_normalization_5[0][0] |
| max_pooling1d_6 (MaxPooling1D) | (None, 1, 128) | 0 | batch_normalization_6[0][0] |
| concatenate_1 (Concatenate) | (None, 3, 128) | 0 | max_pooling1d_1[0][0]<br>max_pooling1d_2[0][0]<br>max_pooling1d_3[0][0] |
| concatenate_2 (Concatenate) | (None, 3, 128) | 0 | max_pooling1d_4[0][0]<br>max_pooling1d_5[0][0]<br>max_pooling1d_6[0][0] |
| concatenate_3 (Concatenate) | (None, 6, 128) | 0 | concatenate_1[0][0]<br>concatenate_2[0][0] |
| flatten_1 (Flatten) | (None, 768) | 0 | concatenate_3[0][0] |
| dropout_1 (Dropout) | (None, 768) | 0 | flatten_1[0][0] |
| dense_1 (Dense) | (None, 28940) | 22254860 | dropout_1[0][0] |

Total params: 290,873,100
Trainable params: 22,871,564
Non-trainable params: 268,001,536

Figure A.4: System Generated Summary for TextCNN Model on FullText (Large) Dataset

# Appendix B

# Summaries of the Models for XML-CNN

In the following, we provide system generated model summaries for XML-CNN described in Chapter 4. The four summaries are XML-CNN models training on AbstractAndTitle (Small), FullText (Small), AbstractAndTitle (Large) and FullText (Large) respectively.

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_1 (InputLayer) | (None, 742) | 0 | |
| embedding_1 (Embedding) | (None, 742, 200) | 12601000 | input_1[0][0] |
| reshape_1 (Reshape) | (None, 742, 200, 1) | 0 | embedding_1[0][0] |
| conv2d_1 (Conv2D) | (None, 1, 199, 128) | 190080 | reshape_1[0][0] |
| conv2d_2 (Conv2D) | (None, 1, 197, 128) | 380032 | reshape_1[0][0] |
| conv2d_3 (Conv2D) | (None, 1, 193, 128) | 759936 | reshape_1[0][0] |
| batch_normalization_1 (BatchNor | (None, 1, 199, 128) | 512 | conv2d_1[0][0] |
| batch_normalization_2 (BatchNor | (None, 1, 197, 128) | 512 | conv2d_2[0][0] |
| batch_normalization_3 (BatchNor | (None, 1, 193, 128) | 512 | conv2d_3[0][0] |
| max_pooling2d_1 (MaxPooling2D) | (None, 1, 19, 128) | 0 | batch_normalization_1[0][0] |
| max_pooling2d_2 (MaxPooling2D) | (None, 1, 19, 128) | 0 | batch_normalization_2[0][0] |
| max_pooling2d_3 (MaxPooling2D) | (None, 1, 19, 128) | 0 | batch_normalization_3[0][0] |
| concatenate_1 (Concatenate) | (None, 1, 57, 128) | 0 | max_pooling2d_1[0][0]<br>max_pooling2d_2[0][0]<br>max_pooling2d_3[0][0] |
| flatten_1 (Flatten) | (None, 7296) | 0 | concatenate_1[0][0] |
| dense_1 (Dense) | (None, 512) | 3736064 | flatten_1[0][0] |
| dropout_1 (Dropout) | (None, 512) | 0 | dense_1[0][0] |
| dense_2 (Dense) | (None, 28940) | 14846220 | dropout_1[0][0] |

Total params: 32,514,868
Trainable params: 19,913,100
Non-trainable params: 12,601,768

Figure B.1: System Generated Summary for XML-CNN Model on AbstractAndTitle (Small) Dataset

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_1 (InputLayer) | (None, 742) | 0 | |
| input_2 (InputLayer) | (None, 14840) | 0 | |
| embedding_1 (Embedding) | (None, 742, 200) | 29666200 | input_1[0][0] |
| embedding_2 (Embedding) | (None, 14840, 200) | 29666200 | input_2[0][0] |
| reshape_1 (Reshape) | (None, 742, 200, 1) | 0 | embedding_1[0][0] |
| reshape_2 (Reshape) | (None, 14840, 200, 1 | 0 | embedding_2[0][0] |
| conv2d_1 (Conv2D) | (None, 1, 199, 128) | 190080 | reshape_1[0][0] |
| conv2d_2 (Conv2D) | (None, 1, 197, 128) | 380032 | reshape_1[0][0] |
| conv2d_3 (Conv2D) | (None, 1, 193, 128) | 759936 | reshape_1[0][0] |
| conv2d_4 (Conv2D) | (None, 1, 199, 128) | 3799168 | reshape_2[0][0] |
| conv2d_5 (Conv2D) | (None, 1, 197, 128) | 7598208 | reshape_2[0][0] |
| conv2d_6 (Conv2D) | (None, 1, 193, 128) | 15196288 | reshape_2[0][0] |
| batch_normalization_1 (BatchNor | (None, 1, 199, 128) | 512 | conv2d_1[0][0] |
| batch_normalization_2 (BatchNor | (None, 1, 197, 128) | 512 | conv2d_2[0][0] |
| batch_normalization_3 (BatchNor | (None, 1, 193, 128) | 512 | conv2d_3[0][0] |
| batch_normalization_4 (BatchNor | (None, 1, 199, 128) | 512 | conv2d_4[0][0] |
| batch_normalization_5 (BatchNor | (None, 1, 197, 128) | 512 | conv2d_5[0][0] |
| batch_normalization_6 (BatchNor | (None, 1, 193, 128) | 512 | conv2d_6[0][0] |
| max_pooling2d_1 (MaxPooling2D) | (None, 1, 19, 128) | 0 | batch_normalization_1[0][0] |
| max_pooling2d_2 (MaxPooling2D) | (None, 1, 19, 128) | 0 | batch_normalization_2[0][0] |
| max_pooling2d_3 (MaxPooling2D) | (None, 1, 19, 128) | 0 | batch_normalization_3[0][0] |
| max_pooling2d_4 (MaxPooling2D) | (None, 1, 19, 128) | 0 | batch_normalization_4[0][0] |
| max_pooling2d_5 (MaxPooling2D) | (None, 1, 19, 128) | 0 | batch_normalization_5[0][0] |
| max_pooling2d_6 (MaxPooling2D) | (None, 1, 19, 128) | 0 | batch_normalization_6[0][0] |
| concatenate_1 (Concatenate) | (None, 1, 57, 128) | 0 | max_pooling2d_1[0][0]<br>max_pooling2d_2[0][0]<br>max_pooling2d_3[0][0] |
| concatenate_2 (Concatenate) | (None, 1, 57, 128) | 0 | max_pooling2d_4[0][0]<br>max_pooling2d_5[0][0]<br>max_pooling2d_6[0][0] |
| concatenate_3 (Concatenate) | (None, 1, 114, 128) | 0 | concatenate_1[0][0]<br>concatenate_2[0][0] |
| flatten_1 (Flatten) | (None, 14592) | 0 | concatenate_3[0][0] |
| dense_1 (Dense) | (None, 512) | 7471616 | flatten_1[0][0] |
| dropout_1 (Dropout) | (None, 512) | 0 | dense_1[0][0] |
| dense_2 (Dense) | (None, 28940) | 14846220 | dropout_1[0][0] |

Total params: 109,577,020
Trainable params: 50,243,084
Non-trainable params: 59,333,936

Figure B.2: System Generated Summary for XML-CNN Model on FullText (Small) Dataset

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_1 (InputLayer) | (None, 1542) | 0 | |
| embedding_1 (Embedding) | (None, 1542, 200) | 37738800 | input_1[0][0] |
| reshape_1 (Reshape) | (None, 1542, 200, 1) | 0 | embedding_1[0][0] |
| conv2d_1 (Conv2D) | (None, 1, 199, 128) | 394880 | reshape_1[0][0] |
| conv2d_2 (Conv2D) | (None, 1, 197, 128) | 789632 | reshape_1[0][0] |
| conv2d_3 (Conv2D) | (None, 1, 193, 128) | 1579136 | reshape_1[0][0] |
| batch_normalization_1 (BatchNor | (None, 1, 199, 128) | 512 | conv2d_1[0][0] |
| batch_normalization_2 (BatchNor | (None, 1, 197, 128) | 512 | conv2d_2[0][0] |
| batch_normalization_3 (BatchNor | (None, 1, 193, 128) | 512 | conv2d_3[0][0] |
| max_pooling2d_1 (MaxPooling2D) | (None, 1, 19, 128) | 0 | batch_normalization_1[0][0] |
| max_pooling2d_2 (MaxPooling2D) | (None, 1, 19, 128) | 0 | batch_normalization_2[0][0] |
| max_pooling2d_3 (MaxPooling2D) | (None, 1, 19, 128) | 0 | batch_normalization_3[0][0] |
| concatenate_1 (Concatenate) | (None, 1, 57, 128) | 0 | max_pooling2d_1[0][0]<br>max_pooling2d_2[0][0]<br>max_pooling2d_3[0][0] |
| flatten_1 (Flatten) | (None, 7296) | 0 | concatenate_1[0][0] |
| dense_1 (Dense) | (None, 512) | 3736064 | flatten_1[0][0] |
| dropout_1 (Dropout) | (None, 512) | 0 | dense_1[0][0] |
| dense_2 (Dense) | (None, 28940) | 14846220 | dropout_1[0][0] |

Total params: 59,086,268
Trainable params: 21,346,700
Non-trainable params: 37,739,568

Figure B.3: System Generated Summary for XML-CNN Model on AbstractAndTitle (Large) Dataset

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_1 (InputLayer) | (None, 1542) | 0 | |
| input_2 (InputLayer) | (None, 57340) | 0 | |
| embedding_1 (Embedding) | (None, 1542, 200) | 134000000 | input_1[0][0] |
| embedding_2 (Embedding) | (None, 57340, 200) | 134000000 | input_2[0][0] |
| reshape_1 (Reshape) | (None, 1542, 200, 1) | 0 | embedding_1[0][0] |
| reshape_2 (Reshape) | (None, 57340, 200, 1 | 0 | embedding_2[0][0] |
| conv2d_1 (Conv2D) | (None, 1, 199, 128) | 394880 | reshape_1[0][0] |
| conv2d_2 (Conv2D) | (None, 1, 197, 128) | 789632 | reshape_1[0][0] |
| conv2d_3 (Conv2D) | (None, 1, 193, 128) | 1579136 | reshape_1[0][0] |
| conv2d_4 (Conv2D) | (None, 1, 199, 128) | 14679168 | reshape_2[0][0] |
| conv2d_5 (Conv2D) | (None, 1, 197, 128) | 29358208 | reshape_2[0][0] |
| conv2d_6 (Conv2D) | (None, 1, 193, 128) | 58716288 | reshape_2[0][0] |
| batch_normalization_1 (BatchNor | (None, 1, 199, 128) | 512 | conv2d_1[0][0] |
| batch_normalization_2 (BatchNor | (None, 1, 197, 128) | 512 | conv2d_2[0][0] |
| batch_normalization_3 (BatchNor | (None, 1, 193, 128) | 512 | conv2d_3[0][0] |
| batch_normalization_4 (BatchNor | (None, 1, 199, 128) | 512 | conv2d_4[0][0] |
| batch_normalization_5 (BatchNor | (None, 1, 197, 128) | 512 | conv2d_5[0][0] |
| batch_normalization_6 (BatchNor | (None, 1, 193, 128) | 512 | conv2d_6[0][0] |
| max_pooling2d_1 (MaxPooling2D) | (None, 1, 19, 128) | 0 | batch_normalization_1[0][0] |
| max_pooling2d_2 (MaxPooling2D) | (None, 1, 19, 128) | 0 | batch_normalization_2[0][0] |
| max_pooling2d_3 (MaxPooling2D) | (None, 1, 19, 128) | 0 | batch_normalization_3[0][0] |
| max_pooling2d_4 (MaxPooling2D) | (None, 1, 19, 128) | 0 | batch_normalization_4[0][0] |
| max_pooling2d_5 (MaxPooling2D) | (None, 1, 19, 128) | 0 | batch_normalization_5[0][0] |
| max_pooling2d_6 (MaxPooling2D) | (None, 1, 19, 128) | 0 | batch_normalization_6[0][0] |
| concatenate_1 (Concatenate) | (None, 1, 57, 128) | 0 | max_pooling2d_1[0][0]<br>max_pooling2d_2[0][0]<br>max_pooling2d_3[0][0] |
| concatenate_2 (Concatenate) | (None, 1, 57, 128) | 0 | max_pooling2d_4[0][0]<br>max_pooling2d_5[0][0]<br>max_pooling2d_6[0][0] |
| concatenate_3 (Concatenate) | (None, 1, 114, 128) | 0 | concatenate_1[0][0]<br>concatenate_2[0][0] |
| flatten_1 (Flatten) | (None, 14592) | 0 | concatenate_3[0][0] |
| dense_1 (Dense) | (None, 512) | 7471616 | flatten_1[0][0] |
| dropout_1 (Dropout) | (None, 512) | 0 | dense_1[0][0] |
| dense_2 (Dense) | (None, 28940) | 14846220 | dropout_1[0][0] |

Total params: 395,838,220
Trainable params: 127,836,684
Non-trainable params: 268,001,536

Figure B.4: System Generated Summary for XML-CNN Model on FullText (Large) Dataset

# Appendix C

# Summaries of the Models for biLSTM

In the following, we provide system generated model summaries for biLSTM described in Chapter 4. The four summaries are TextCNN models training on AbstractAndTitle (Small), FullText (Small), AbstractAndTitle (Large) and FullText (Large) respectively.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 694)               0
_________________________________________________________________
embedding_1 (Embedding)      (None, 694, 200)          12601000
_________________________________________________________________
bidirectional_1 (Bidirection (None, 400)               641600
_________________________________________________________________
dense_1 (Dense)              (None, 28940)             11604940
=================================================================
Total params: 24,847,540
Trainable params: 12,246,540
Non-trainable params: 12,601,000
_________________________________________________________________
```

Figure C.1: System Generated Summary for biLSTM Model on AbstractAndTitle (Small) Dataset

```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, 742)          0
__________________________________________________________________________________________________
input_2 (InputLayer)            (None, 12393)        0
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 742, 200)     29666200    input_1[0][0]
__________________________________________________________________________________________________
embedding_2 (Embedding)         (None, 12393, 200)   29666200    input_2[0][0]
__________________________________________________________________________________________________
bidirectional_1 (Bidirectional) (None, 400)          641600      embedding_1[0][0]
__________________________________________________________________________________________________
bidirectional_2 (Bidirectional) (None, 400)          641600      embedding_2[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 800)          0           bidirectional_1[0][0]
                                                                 bidirectional_2[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 28940)        23180940    concatenate_1[0][0]
==================================================================================================
Total params: 83,796,540
Trainable params: 24,464,140
Non-trainable params: 59,332,400
__________________________________________________________________________________________________
```

Figure C.2: System Generated Summary for biLSTM Model on FullText (Small) Dataset

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 1461)              0
_________________________________________________________________
embedding_1 (Embedding)      (None, 1461, 200)         37738800
_________________________________________________________________
bidirectional_1 (Bidirection (None, 400)               641600
_________________________________________________________________
dense_1 (Dense)              (None, 28940)             11604940
=================================================================
Total params: 49,985,340
Trainable params: 12,246,540
Non-trainable params: 37,738,800
_________________________________________________________________
```

Figure C.3: System Generated Summary for biLSTM Model on AbstractAndTitle (Large) Dataset

# Appendix D

# Summaries of the Models for convLSTM

In the following, we provide system generated model summaries for convLSTM described in Chapter 4. The four summaries are TextCNN models training on AbstractAndTitle (Small), FullText (Small), AbstractAndTitle (Large) and FullText (Large) respectively.

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_1 (InputLayer) | (None, 742) | 0 | |
| embedding_1 (Embedding) | (None, 742, 200) | 12601000 | input_1[0][0] |
| conv1d_1 (Conv1D) | (None, 740, 128) | 76928 | embedding_1[0][0] |
| conv1d_2 (Conv1D) | (None, 739, 128) | 102528 | embedding_1[0][0] |
| conv1d_3 (Conv1D) | (None, 738, 128) | 128128 | embedding_1[0][0] |
| lstm_1 (LSTM) | (None, 128) | 131584 | conv1d_1[0][0] |
| lstm_2 (LSTM) | (None, 128) | 131584 | conv1d_2[0][0] |
| lstm_3 (LSTM) | (None, 128) | 131584 | conv1d_3[0][0] |
| lambda_1 (Lambda) | (None, 3, 128) | 0 | lstm_1[0][0]<br>lstm_2[0][0]<br>lstm_3[0][0] |
| lambda_2 (Lambda) | (None, 3, 128) | 0 | lambda_1[0][0] |
| flatten_1 (Flatten) | (None, 384) | 0 | lambda_2[0][0] |
| dense_1 (Dense) | (None, 256) | 98560 | flatten_1[0][0] |
| dropout_1 (Dropout) | (None, 256) | 0 | dense_1[0][0] |
| dense_2 (Dense) | (None, 28940) | 7437580 | dropout_1[0][0] |

Total params: 20,839,476
Trainable params: 8,238,476
Non-trainable params: 12,601,000

Figure D.1: System Generated Summary for convLSTM Model on AbstractAndTitle (Small) Dataset

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_1 (InputLayer) | (None, 742) | 0 | |
| input_2 (InputLayer) | (None, 12393) | 0 | |
| embedding_1 (Embedding) | (None, 742, 200) | 29666200 | input_1[0][0] |
| embedding_2 (Embedding) | (None, 12393, 200) | 29666200 | input_2[0][0] |
| conv1d_1 (Conv1D) | (None, 740, 128) | 76928 | embedding_1[0][0] |
| conv1d_2 (Conv1D) | (None, 739, 128) | 102528 | embedding_1[0][0] |
| conv1d_3 (Conv1D) | (None, 738, 128) | 128128 | embedding_1[0][0] |
| conv1d_4 (Conv1D) | (None, 12391, 128) | 76928 | embedding_2[0][0] |
| conv1d_5 (Conv1D) | (None, 12390, 128) | 102528 | embedding_2[0][0] |
| conv1d_6 (Conv1D) | (None, 12389, 128) | 128128 | embedding_2[0][0] |
| lstm_1 (LSTM) | (None, 128) | 131584 | conv1d_1[0][0] |
| lstm_2 (LSTM) | (None, 128) | 131584 | conv1d_2[0][0] |
| lstm_3 (LSTM) | (None, 128) | 131584 | conv1d_3[0][0] |
| lstm_4 (LSTM) | (None, 128) | 131584 | conv1d_4[0][0] |
| lstm_5 (LSTM) | (None, 128) | 131584 | conv1d_5[0][0] |
| lstm_6 (LSTM) | (None, 128) | 131584 | conv1d_6[0][0] |
| lambda_1 (Lambda) | (None, 3, 128) | 0 | lstm_1[0][0]<br>lstm_2[0][0]<br>lstm_3[0][0] |
| lambda_3 (Lambda) | (None, 3, 128) | 0 | lstm_4[0][0]<br>lstm_5[0][0]<br>lstm_6[0][0] |
| lambda_2 (Lambda) | (None, 3, 128) | 0 | lambda_1[0][0] |
| lambda_4 (Lambda) | (None, 3, 128) | 0 | lambda_3[0][0] |
| flatten_1 (Flatten) | (None, 384) | 0 | lambda_2[0][0] |
| flatten_2 (Flatten) | (None, 384) | 0 | lambda_4[0][0] |
| concatenate_1 (Concatenate) | (None, 768) | 0 | flatten_1[0][0]<br>flatten_2[0][0] |
| dense_1 (Dense) | (None, 256) | 196864 | concatenate_1[0][0] |
| dropout_1 (Dropout) | (None, 256) | 0 | dense_1[0][0] |
| dense_2 (Dense) | (None, 28940) | 7437580 | dropout_1[0][0] |

Total params: 68,371,516
Trainable params: 9,039,116
Non-trainable params: 59,332,400

Figure D.2: System Generated Summary for convLSTM Model on FullText (Small) Dataset

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_1 (InputLayer) | (None, 1542) | 0 | |
| embedding_1 (Embedding) | (None, 1542, 200) | 37738800 | input_1[0][0] |
| conv1d_1 (Conv1D) | (None, 1540, 128) | 76928 | embedding_1[0][0] |
| conv1d_2 (Conv1D) | (None, 1539, 128) | 102528 | embedding_1[0][0] |
| conv1d_3 (Conv1D) | (None, 1538, 128) | 128128 | embedding_1[0][0] |
| lstm_1 (LSTM) | (None, 128) | 131584 | conv1d_1[0][0] |
| lstm_2 (LSTM) | (None, 128) | 131584 | conv1d_2[0][0] |
| lstm_3 (LSTM) | (None, 128) | 131584 | conv1d_3[0][0] |
| lambda_1 (Lambda) | (None, 3, 128) | 0 | lstm_1[0][0]<br>lstm_2[0][0]<br>lstm_3[0][0] |
| lambda_2 (Lambda) | (None, 3, 128) | 0 | lambda_1[0][0] |
| flatten_1 (Flatten) | (None, 384) | 0 | lambda_2[0][0] |
| dense_1 (Dense) | (None, 256) | 98560 | flatten_1[0][0] |
| dropout_1 (Dropout) | (None, 256) | 0 | dense_1[0][0] |
| dense_2 (Dense) | (None, 28940) | 7437580 | dropout_1[0][0] |

Total params: 45,977,276
Trainable params: 8,238,476
Non-trainable params: 37,738,800

Figure D.3: System Generated Summary for convLSTM Model on AbstractAndTitle (Large) Dataset

# Curriculum Vitae

| | |
|---|---|
| **Name:** | Xindi Wang |
| **Post-Secondary Education and Degrees:** | The University of British Columbia<br>Vancouver, BC<br>2012-2015 B.Sc.<br><br>The University of Western Ontario<br>London, ON<br>2017-2019 M.Sc. |
| **Related Work Experience:** | Teaching Assistant and Research Assistant<br>The University of Western Ontario<br>2017 - 2019 |

**Publications:**

X. Wang, & R. Mercer. Incorporating Figure Captions and Descriptive Text in MeSH Term Indexing. Proceedings of the BioNLP 2019 workshop, Association for Computational Linguistics.