A Deeper Look at Computational Sarcasm

Devin Pelser

215023955

A thesis submitted to
the University of KwaZulu-Natal,
College of Agriculture, Engineering and Science,
in fulfilment of the requirements for the degree of

Master of Computer Science

Supervisor: Prof. Hugh Murrell

Mathematics, Statistics and Computer Science

University of KwaZulu-Natal

November 2019

I, Devin Pelser , declare that:

(i) The research reported in this thesis, except where otherwise indicated, is my original research.

(ii) This thesis has not been submitted for any degree or examination at any other university.

(iii) This thesis does not contain other persons' data, pictures, graphs or other information, unless specifically acknowledged as being sourced from other persons.

(iv) This thesis does not contain other persons' writing, unless specifically acknowledged as being sourced from other researchers. Where other written sources have been quoted, then:

  a) their words have been re-written but the general information attributed to them has been referenced:

  b) where their exact words have been used, their writing has been placed inside quotation marks, and referenced.

(v) This thesis does not contain text, graphics or tables copied and pasted from the Internet, unless specifically acknowledged, and the source being detailed in the dissertation/thesis and in the References sections.

Candidate: Devin Pelser

Signature: [illegible]

As the candidate's supervisor I agree to the submission of this thesis for examination.

Supervisor: Prof. Hugh Murrell

Signature: H. C. Murrell

20th November 2019

ABSTRACT

A Deeper Look at Computational Sarcasm

Devin Pelser
Mathematics, Statistics and Computer Science
Master of Computer Science

Recent work in automated sarcasm detection has placed a heavy focus on context and meta-data. Whilst certain utterances indeed require background knowledge and commonsense reasoning, previous works have only explored shallow models for capturing the lexical, syntactic and semantic cues present within a text. In this work, we evaluate the efficacy of three deep convolutional archetypes for sarcasm classification. In particular, we propose a 58-layer network, DweNet, implemented with dense connectivity to model the isolated utterance and extract richer features therein. We compare our approach against recent state-of-the-art architectures which integrate extrinsic information, and demonstrate competitive results whilst modelling only the content of the text. Specifically, DweNet outperforms a network which, in addition to implementing local-context, performs user profiling to capture the stylometric traits of the author, as well as modelling the entire discourse of the forum, on a large-scale social media dataset. An analysis of the dependency of prior feature maps in generating the final feature map is provided to illustrate the key role of low-level features in our model's ability to rival context-based networks. A case study is presented, supporting that our approach accurately classifies additional uses of clear sarcasm, which a standard CNN misclassified. Social media has been the target domain in the majority of published research, however the datasets are often plagued with informal language and spelling errors. Hence, we provide benchmarks on a new *formally* written dataset which is free of these concerns, for future work in content-based modelling. A comparison of FastText and GloVe embeddings is presented, to establish the suitability of each for sarcasm detection, as well as the effect of non-static representations. The effect of local context is investigated to determine whether DweNet is capable of utilising contextual information, provided in the form of parent comments, to obtain increased performance on a social media dataset. Finally, the influence of the data preprocessing techniques is studied, focusing on the mapping of infrequently used words to an unknown token, to demonstrate that these uncommon words are often not needed to confidently discern the presence of sarcasm within a given phrase.

Keywords: Sarcasm, NLP, CNN, ResNet, DenseNet, DweNet

PUBLICATIONS ARISING

[1] Devin J. Pelser and Hugh Murrell, *Deep and Dense Sarcasm Detection*, https://arxiv.org/abs/1911.07474.

ACKNOWLEDGMENTS

I would first like to thank my supervisor Prof. Hugh Murrell at the University of KwaZulu-Natal. Prof. Murrell was always available whenever I needed assistance in my research, happily responding to numerous emails sent in a brief span of time. He consistently allowed the research to remain my own, but kept me on the right path when I needed it. His patience and guidance is greatly appreciated.

I would also like to express my profound gratitude to my girlfriend Caitie Leggat for her unfailing support and continuous encouragement throughout this year, especially during the darker times. Caitie was always willing to listen about my research and help me in whatever manner she could, and for that I am gratefully indebted to her. This accomplishment would not have been possible without her.

Thank you to my parents for always believing in me and motivating me towards success. The work ethic you both instilled in me was invaluable in the writing of this thesis. Finally, thank you to Caitie, Hennie and Jayne for proofreading the work and providing useful insight and corrections.

Contents

List of Figures

Chapter 1

Introduction

Sarcasm is a complex linguistic phenomenon in which the intended meaning of an utterance is not the same as its literal meaning [1]. A sarcastic phrase will often have positive wording, but intends to convey a negative opinion. Consider the sentence "*Great! I love waking up sick!*". We are easily able to recognise the sarcasm due to the presence of strong polarity shifts and common-sense; no one *loves* waking up sick. The literal meaning of the sentence has been discounted, with the speaker expecting the listener to understand the implied intent.

Due to this perceived polarity shift, as well the figurative and nuanced nature of sarcasm, the accurate detection of sarcastic utterances has been a challenging task within opinion mining and sentiment analysis [2]. With the rise of social media use, and the need for understanding comments therein, affective computing has gained an increase in popularity. However, existing systems for summarising reviews, or monitoring brand sentiment, often fail to detect the implicit meaning behind sarcastic remarks [3] and consequently mislead the analysis of sarcastic phrases, with interpretations usually being taken as literal.

This presents a clear issue for a company wishing to evaluate their current public standing. Their social media account may be plagued with insincere comments like "*Such a great company, you guys realllly know how to make your customers happy!*". From a purely semantic perspective, the general opinion contained within the comment is positive. Consequently, the analysis of these comments, from a semantic viewpoint, would give the company a false feeling of confidence that their customers are happy with their service. Whilst a human is able to discern the negative pretense of these sorts of comments, it is not as easily identified within the context of machine learning due to the overall positive sentiment formed by the words 'great' and 'happy'.

The problem becomes increasingly pronounced with the use of more complex sarcasm, such as a comment which cannot be identified based solely on the lexical and pragmatic cues within.

Consider the following, "*The Sharks played so well last night!*". This sentence could be sincere or sarcastic, depending on how the sharks *actually* performed. If they won, the comment is likely genuine, however, if they lost by a large margin, it is reasonable to assume the comment was sarcastic. These forms of remarks require real-world knowledge to confidently classify and as a result, the need for extrinsic information has been identified in past literature [4]. Additionally, comments on social media are often in response to a previous comment, further demonstrating the need for contextual information. Another key issue when dealing with social media commentary, is the commonality of spelling errors and colloquialisms. These have been observed to diminish the reliance of purely grammatical hints [5], with prior works having found difficulty in accurately classifying these informally written comments [6].

In recent times, the incorporation of contextual information and meta-data has become a dominant facet of sarcasm detection. Prior works have seen great success in generating detailed user profiles, prior to categorising the user's comment [4] [7]. This is done by modelling a user's sarcastic tendencies towards certain topics as well as their personality traits. Further, online communities tend to surround a central belief or opinion [8], which could be political, or something as mundane as "dogs are better than cats". As a result, an arbitrary positive comment about a cat, on a forum specifically discussing why dogs are better than cats, would be more prone to sarcasm than an identical comment posted on a general pet forum.

This recent dependency on external information presents two issues for deploying real-world systems. Privacy is the first concern, as governments and businesses are increasingly collecting, analysing and selling the detailed information obtained from social media for marketing purposes [9]. However, we have seen an increased social awareness of these privacy infringements and a strong push has been made to place a user's data back in their own hands [10] [11] [12]. Baruh et al. [13] provide an analysis of 166 studies over 34 countries in relation to online privacy, finding that users concerned with privacy were reluctant to use online services which required them to share information. Consequently, the meta-data needed to generate these user profiles may not always be available due to a user's privacy settings [14], or the user may be new to the forum and as such, not much prior information will exist. Further, this information is dependent on the domain, and prevents these networks from generalising outside of social media ecosystems. Another issue is the potential unreliability and the large overhead that is introduced when placing the focus on generating detailed profiles of users or forums. A person's opinion is subject to change and their user profile may no longer truly represent who they are. For example, a user could have had a bad experience with some company, and after complaining a few times online, the problem may have been resolved to their satisfaction. However, given their prior posting history, any positive statement directed at the business would essentially be viewed as sarcastic. This effectively transforms the task of sarcasm detection into something more specific: how does *this* person *usually* feel about *this* topic.

The key aim of this work is to explore whether additional local cues could be extracted from an isolated utterance. In particular, we propose a deep neural network implemented with dense connectivity to build rich feature maps purely from the linguistic structure of the text. We do not seek to outperform the state-of-the-art, but rather, to determine if our model is able to rival recent approaches, all of which make use of context and meta-data. Our approach uses multiple convolution layers - with direct connections between each - to facilitate the use of both low-level, simple features and complex hierarchical ones, in order to develop a deeper understanding of the sarcastic patterns within a given text. We hypothesise that these diverse feature-maps would give further insight into the sarcastic intent, or lack thereof, within a sentence.

1.1 Problem Statement

Sarcasm, a complex linguistic phenomenon commonly found in online communities, is problematic within the field of sentiment analysis. Existing systems have been found to struggle to accurately predict the sentiment of a sarcastic remark due to positive words being used to convey a negative attitude. With the growing popularity of social media, sarcasm has presented itself as an issue within opinion mining on these domains, such as a brand wishing to monitor their reputation on Twitter. An analysis of a brand's Twitter commentary may potentially lead to inaccurate conclusions as these sarcastic phrases are taken literally. Sarcasm detection models have been extensively studied, however, in recent times a heavy focus has been placed on using context and meta-data. Whilst certain sentences do require background knowledge to classify, no research exists into whether a model using only the isolated utterance could rival a model incorporating context. This would provide an indication whether additional information is present in the standalone text. Deep learning based models have seen great success across many domains, including Natural Language Processing (NLP). Within the field of computer vision, ResNet and DenseNet led to state-of-the-art results on numerous baselines through the use of shortcut connections, which facilitated an increased flow of information within the networks. ResNet-type models have also been found to be beneficial for many NLP tasks. However, no research exists into very deep models implemented with residual or dense-like connections for sarcasm detection. These diverse feature maps - arising from the dense connections - as well as the complex hierarchical features extracted by deep networks, may allow for additional intrinsic cues to be captured within an utterance.

1.2 Aims and Objectives

This study aims to evaluate the efficacy of three deep learning architectures for modelling an isolated utterance in order to classify it as sarcastic or not. Our study was motivated by the increasing dependency on user profiling and context within recent sarcasm detection systems, as well as the lack of content-based systems in recent literature. This prompted the question as to whether or not additional local information is available in the text, or if we indeed need to rely on user embeddings and context to obtain state-of-the-art results. The objectives of this work are as follows:

1. To implement and evaluate three deep learning architectures, a CNN, residual network and densely connected network, for sarcasm classification.
2. To determine whether the residual and dense archetypes are able to extract richer information from the isolated utterance.
3. To provide benchmarks and analysis on a formal dataset, which is free from informal language and does not depend on local context.
4. To determine whether a network, which models only the content of the text, is able to compete with recent approaches which incorporate context and meta-data.

1.3 Research Contributions

The key contributions of this work is a deep and densely connected network which models only the content of the text, but is able to rival the results of recent approaches which integrate context and meta-data. Further, the effect of different word embeddings on the performance of sarcasm classification is investigated and presented. A comparison is provided, contrasting the performance of our three networks against those proposed in prior literature, on a large-scale social media dataset. A formally written English dataset, free from the issues which commonly plague social media datasets, is benchmarked and analysed. The use of deep networks - which use residual and dense-like connectivity - presented in this study are an original work and to the best of our knowledge have not been applied to the task of sarcasm detection in past literature. The key contributions are as follows:

1. The design and implementation of a deep, densely connected network, which is able to compete with recent state-of-the-art approaches, whilst making no use of context nor meta-data.
2. The performance comparison of three deep architectures on a large-scale social media dataset against prior state-of-the-art methods.
3. Benchmarking results on a new, formally written English dataset for sarcasm detection.
4. The analysis of the effect of different word embeddings in relation to accurate sarcasm detection.

1.4 Research Scope and Limitations

This study focuses only on the use of convolution-based architectures and does not consider other archetypes which have seen great success in past works. Transfer learning is not considered despite having shown progression across numerous NLP domains. Our networks make no use of the context or meta-data available within the main benchmark dataset, focusing entirely on modelling the isolated utterance. The datasets used in this study range from 17,074 to 267,615 entries, which is sufficient for deep learning tasks [15]. Prior studies on the main comparison dataset present their results as a percent out of 100, consequently a few obtain the same result, but we have no means with which to properly determine which model was the best across that score. Further, the studies do not indicate how they have rounded their results. With this in mind, we round our results down when presenting the final scores to prevent any potential bias or unfair comparisons which may arrise from rounding up.

1.5 Thesis Structure

The remaining chapters in this thesis are structured in the following manner:

- **Chapter 2: Background and Related Work**

  In this chapter, a literature review of the prior published work surrounding sarcasm classification is conducted as well as an overview of the required background knowledge for the work herein.

- **Chapter 3: Method**

  This chapter provides a detailed description of the studied datasets, the data preprocessing techniques used, the proposed models and an overview of the various experiments performed.

- **Chapter 4: Discussion and Results**

  The results and analysis of findings for each investigation are presented within this chapter. A comparison of the obtained results, with that of prior published work, is also conferred.

- **Chapter 5: Conclusion**

  A summary of the findings and conclusions reached, as well as proposed future work is presented.

- **Appendix A: Reader Guide**

  In the appendix we detail how to obtain the code and data used in this research, as well as instructions for replicating the results of our experiments.

Chapter 2

Background and Related Work

In this chapter, we present a literature review of prior published work in computational sarcasm, followed by an overview of the background knowledge required for the work within. Initially, the literature review introduces the first works in sarcasm detection and expands on the various methods used and observable trends thereafter. The following section, Artificial Neural Networks, presents a brief outline of the general concepts of a neural network, prior to detailing the architectures investigated in Chapter 3 and Chapter 4.

2.1 Literature Review

Whilst computational sarcasm is still a relatively new field [6] - receiving an increased focus from researchers in recent times due to the massive growth of social media and the need for sentiment analysis therein - various approaches have been presented and examined. Tepperman et al. [16] studied detection within speech through the use of prosodic, spectral (average pitch of utterance, duration of utterance etc.) and contextual cues (gender, laughter etc.). They observed that these spectral and contextual features could be used to discern sarcasm as accurately as a human annotator would, however, prosody alone was found to be insufficient. Carvalho et al. [17] analysed comments on a Portuguese news site and found certain linguistic features were indicative of irony; such as emoticons, excessive punctuation and quotation marks. Further, gestural cues were explored, identifying onomatopoeic expressions for laughter assisted detection. Finally, they demonstrated cues based on deeper linguistic information were relatively inefficient, suggesting the need to explore additional oral cues.

This statistical approach based on surface-level features has been further studied. González-Ibàñez et al. [18] explored unigrams, dictionary-based lexical features and pragmatic features (punctuation, emojis etc.) in relation to machine learning. Further, they provide an analysis of the comparison between sarcastic utterances and nonsarcastic utterances which expressed both positive and negative sentiment. Tsur et al. [19] presented a novel semi-supervised algorithm to detect sarcasm in product reviews and found certain words such as "yay!" or "great!" to be recurring aspects of sarcasm. However, they noted that a combination of more subtle features proved better suited for identifying various facets of sarcasm. Liebrecht et al. [20] extended the idea of n-grams as features through the use of unigrams, bigrams and trigrams, together with hyperboles - signaled by intensifiers and exclamations - to classify Dutch tweets. They further hypothesise that explicit markers of sarcasm, such as a hashtag, are the digital extralinguistic equivalent of the nonverbal cues employed in real life to convey sarcasm. Kreuz and Caucci [21] considered the effect of lexical indicators, such as punctuation, interjections (e.g., "gee" or "gosh") and hyperboles, on sarcasm detection. Further, they studied the detection of sarcasm by human readers by compiling a list of published works which contained the phrase ‘said sarcastically’, however ‘sarcastically’ was removed. They found that these statements were still identified as more sarcastic than other phrases, hypothesising that sarcasm is more formulaic than previously thought.

The lack of accurately labeled datasets [18] has led to social media being used as a primary domain. Large self-annotated corpora have been built using the Twitter API to scrape comments containing tokens implying sarcastic intent (e.g. #sarcasm). Reyes et al. [22] studied the efficacy of recognising ironic tweets at a linguistic level, and provide insights into the issues facing tasks such as sarcasm detection. This approach was followed by Ptácek et al. [23] with a feature set consisting of skipgrams and character n-grams to classify Czech and English tweets. Investigations showed that their language-independent approach was better suited compared to state-of-the-art methods on English datasets. They further demonstrated the difficulties arising from pre-processing techniques on a morphologically rich language like Czech.

Given that a sarcastic utterance often implies the opposite to what is said [24], sentiment and semantic incongruity as features has also been investigated. Riloff et al. [25] studied the presence of positive sentiment co-occurring with negative situational phrases, such as the word ‘love’ followed by an objectionable activity or state, such as ‘taking exams’. They proposed a novel bootstrapping algorithm which automatically learnt a list of positive sentiment phrases and negative situational phrases which occured within tweets. Demonstrating that the identification of these contrasting contexts resulted in improved recall for sarcasm classification. Buschmeier et al. [26] evaluated different supervised classifiers using logistic regression, with a focus on the polarity between written words and the star rating on Amazon reviews. Khattri et al. [27] proposed a novel approach comprised of two parts: a contrast-based predictor to capture sentiment contrasts between a tweet and its responses, and a historical tweet-based predictor to identify a contrast in the sentiment towards some entity in the target tweet and the author’s prior expressed sentiment towards the target.

Word embeddings have seen extensive use for sarcasm classification, with Joshi et al. [28] providing an investigation into augmenting the feature sets of four prior works ([18], [20], [26], [29]) with four differing embeddings. Discordance between word embeddings led to increased performance across numerous datasets, however, it was found embeddings alone were insufficient. An analysis of the commonly misclassified sentences showed these traditional word embeddings were unable to distinguish multiple meanings of a word. Further, the embeddings lack contextual information, causing sentences such as 'Arsenal played so well last night!' to be difficult to classify without background knowledge: if Arsenal did indeed play well, then this is not sarcastic, however if they lost, the sentence is likely sarcastic. Finally, the presence of metaphors also led to classification error, with the authors hypothesising this was likely due to the manner in which figurative language compares concepts which are not directly related.

Many papers have attempted to provide more semantically-aware word representations. Poria et al. [6] demonstrated that embeddings may be better aligned with sarcasm through updating the representations as their model trained. They observed increased accuracy from these non-static embeddings, arguing that the commonality of informal language within social media prevented static embeddings from obtaining suitable representations for the out-of-vocabulary words. Tang et al. [30] demonstrated that traditional embeddings effectively model the syntactic context of words but discard affective information. This results in a clear issue for sentiment analysis from the presence of opposing polarities amongst spatially near words. In an attempt to solve this issue, they proposed sentiment specific word embeddings (SSWE) through the use of three neural networks which incorporated sentiment polarity in their respective loss functions. Their proposed embeddings show improved results in the classification of sarcastic tweets over other traditional embeddings.

Emojis - Unicode graphical symbols - are vastly used on social media in combination with plain text to visually complement or enforce the underlying meaning of the message [31]. These emojis add sentiment and emotion, facilitating more expressive language over text [32]. Following this, Eisner et al. [33] proposed learning emoji embeddings to capture the meaning of emojis as indicated by their Unicode description. They demonstrated that their emoji embeddings, emoji2vec, outperformed word2vec on sentiment analysis tasks, whilst needing fewer training examples. Agrawal and An [34] proposed affective word embeddings for sarcasm (AWES). Investigating both a sentiment and emotion aware model, various affective lexicons were leveraged to compute scores for words and sentences. A Bi-directional LSTM was used to predict the affective label of each word in a sentence, updating the word embedding in the process. A comparison of their embeddings to traditional ones, showed that the sentiment aware model was better suited for short text, whilst the emotion aware model was better suited for long text. Consequently, resulting in both models obtaining better results compared to state-of-the-art baselines. Zhang et al. [35] used a bi-directional gated recurrent network to capture the syntactic and semantic features appearing locally within the text. Further, a pooling neural network captured salient words from a user's tweet history in order to model the user's sarcastic tendencies. They also showed that utilising separate word embeddings, one for the target tweets and another for the historical tweets, led to increased performance.

The need for context was investigated by Wallace et al. [36], observing that identifying irony exclusively from the local text is not always possible. Through empirical evidence they demonstrated that humans often require context to confidently discern ironic intent. Further, they illustrated that machine learning approaches often misclassify the same sentences which the human annotators required additional context to classify. In more recent works, the inclusion of extrinsic information has been increasingly used, with Bamman and Smith [37] making extensive use of contextual information regarding the author and audience. They showed that the inclusion of extra-linguistic knowledge - provided by context - resulted in increased accuracy compared to purely linguistic approaches. Ghaeini et al. [14] reported competitive results by incorporating conversational context together with Bi-LSTMs and attention mechanisms to model the utterance. A manually annotated Reddit dataset was analysed by Wallace et al. [38], proposing features based on forum type (e.g., conservative or liberal), sentiment, named entities and interactions between these. Further reinforcing the argument that the context in which comments are embedded, as well as real world facts, are required for accurate sarcasm detection.

Automatic feature extraction through convolutions within NLP domains was first proposed by Collobert and Weston [39], demonstrating its effectiveness across multiple tasks. Amir et al. [4] extended this to sarcasm to mitigate the effort of handcrafting features; implementing a CNN to learn user embeddings based on an author's post history. These embeddings were used in conjunction with lexical cues to obtain increased performance over prior state-of-the-art approaches. Further deep learning methods have been reported with Poria et al. [6] obtaining state-of-the-art results with an ensemble of a SVM preceded by four CNNs, three of which were pre-trained to extract emotion, sentiment and personality respectively. Ghosh and Veale [40] combined a CNN, with an LSTM and DNN to obtain improved performance against a recursive SVM which employed constituency parse trees fed and labeled with syntactic and semantic information. Hazarika et al. [7] extensively explored meta-data through profiling discussion forums based on all previously written comments, as well as capturing both stylometric and personality traits of the author through user embeddings.

Within the field of computer vision, the application of residual learning [41] resulted in extremely deep networks which led to state-of-the-art results on multiple image recognition tasks. This notion was further enhanced through dense connectivity by Huang et al. [42], showing that the dense variants outperformed their residual counterparts. From 19-layer networks [43] through to 152-layers [41], the use of these very deep models have been observed to demonstrate a significant improvement within the domain of computer vision [44]. However, little research exists into the effectiveness of these deep networks within NLP domains. Zhang et al. [45] went to a depth of 6 layers in their study, reportedly the deepest network for NLP at the time [46]. Schwenk et al. [46] proposed the first use of deep residual archetypes for NLP, reaching a depth of 49 layers with a model similar to the original ResNet. The model was evaluated on the datasets presented in [45], reporting similar or better results from the increased depth. Wang et al. [47] is the only prior paper to implement a densely connected model for NLP tasks. Their model utilised densely connected CNNs together with a multi-scale attention mechanism to obtain state-of-the-art results on five different NLP datasets.

In this dissertation, deep densely connected networks for sarcasm classification are studied; noting that prior approaches make use of shallower models. Previous results indicate the necessity of contextual information and the insufficiency of purely textual cues. However, no contextual information is used in this work, with our primary research goal being to ascertain whether our deeper network is able to extract richer sarcastic patterns intrinsic to the text.

2.2 Artificial Neural Networks

Artificial Neural Networks refer to the general field of computational techniques inspired by the processing of information within the brain through neuron clusters [48] [49]. Given some input, a neural network will act on this to produce some output. This output can range from a continuous probability distribution of some quantity, such as the blood pressure of a human, to a categorization of some property, such as whether an image is of a cat or dog [50]. Effectively, a neural network is a simplistic mechanical biomimicry of neurons, which acts as a non-linear function approximator [51].

Artificial neurons can be connected to one another similarily to a brain's axons. The output of some neuron, or its *activation*, is dependent on a linear combination of the neighboring neurons' activations [52]. The connections between neurons can be strong or weak and by altering these magnitudes the neural network can learn from some input stimuli. Neural networks are often built from stacking multiple neuron layers, and, in general, the activation of some layer forms the input to the following layer [53]. Nonlinearities can be introduced through *activation functions*, which simply determine the degree to which a neuron responds to some input. A basic artificial neuron is seen below in Fig. 2.1.

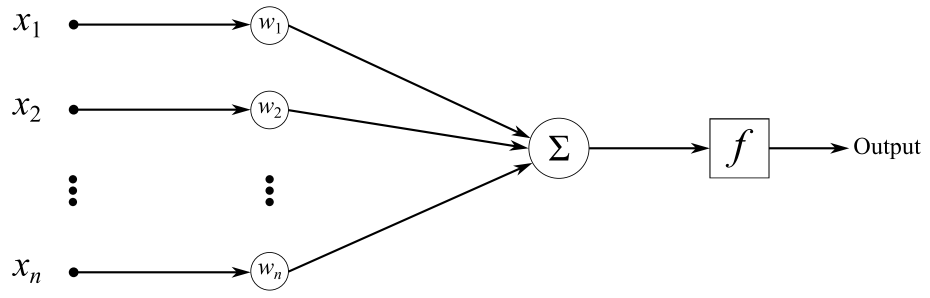


Figure 2.1 A basic artificial neuron.

The inputs, $x_1, x_2, \ldots, x_n$ can be seen as stimuli to the dendrites of a biological neuron. Each input has a corresponding weight, $w_1, w_2, \ldots, w_n$, which is effectively the degree to which a biological neuron would respond to the stimulus. Continuing with the biological perspective, these signals are passed to the cell body, represented by the center circle and square. Here, two operations, Σ and f, are performed to determine how the neuron will respond to the stimulus, resulting in an output.

We now describe a general sense of the mathematical model behind a neural network, with certain details and more specific models expanded on in the following sections. The input signal to the i^{th} layer is a vector $\mathbf{x}^{(i)}$, where the j^{th} neuron in the i^{th} layer is denoted as $x_j^{(i)}$. The activation of a particular neuron is determined by two separate functions. First, a weighted linear sum, $\mathbf{z}^{(i)}$, of the n neurons from the previous layer is taken, where the weights, $w_{kj}^{(i)}$, represent the connection strength of the axon between the k^{th} neuron in the $(i-1)^{th}$ layer and the j^{th} neuron in the i^{th} layer. That is,

$$z_j^{(i)} = w_{1j}^{(i)} x_1^{(i)} + w_{2j}^{(i)} x_2^{(i)} + \ldots + w_{nj}^{(i)} x_n^{(i)}. \tag{2.1}$$

Secondly, an activation function f operates on each element of $\mathbf{z}^{(i)}$ to produce the output. Many choices of activation functions exist, depending on the task at hand, a few of which are discussed in Section 2.2.3. Hence, the activation for the j^{th} neuron in the i^{th} layer is:

$$x_j^{(i+1)} = f(z_j^{(i)}) = f(w_{1j}^{(i)} x_1^{(i)} + w_{2j}^{(i)} x_2^{(i)} + \ldots + w_{nj}^{(i)} x_n^{(i)}). \tag{2.2}$$

That is, given two layers with n and m neurons respectively, the objective of the connections between these layers is to take an input vector $\mathbf{x}^{(i)} \in \mathbb{R}^n$ and produce an output vector $\mathbf{z}^{(i+1)} \in \mathbb{R}^m$. This can be seen as the multiplication of $\mathbf{x}^{(i)}$ with a weight matrix $\mathbf{W}^{(i)} \in \mathbb{R}^{(m \times n)}$. That is:

$$\mathbf{z}^{(i)} = \mathbf{W}^{(i)} \mathbf{x}^{(i)}. \tag{2.3}$$

where the weight matrix has form:

$$\mathbf{W}^{(i)} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ w_{m1} & \cdots & \cdots & w_{mn} \end{pmatrix}$$

and contains the connection strengths for all individual axons. Finally, by combining equations 2.1, 2.2 and 2.3, the activation of the i^{th} layer is:

$$\mathbf{x}^{(i+1)} = f(\mathbf{z}^{(i)}) = f(\mathbf{W}^{(i)}\mathbf{x}^{(i)}). \tag{2.4}$$

This activation $\mathbf{x}^{(i+1)}$ can then be used as the input for the $(i+1)^{th}$ layer and the process may repeat itself until the network produces the final output. Effectively, information travels from input neurons to output neurons, passing through many layers composed of linear and non-linear transformations [15]. Through different connections and activation functions, a neural network can essentially approximate a complex function, such as whether a sentence is sarcastic or not. We now discuss certain concepts, such as the training and evaluation of neural networks, as well as more advanced architectures, in the following sections.

2.2.1 Training

The weights within an artificial neural network need to be adjusted from their initial values in order to fit the data. This is accomplished by training the network, specifically, we use *supervised learning* [15] to train our models for the task of binary sarcasm classification.

The objective of supervised learning is to learn a particular set of weights which fit the model to a given set of input and output data. In our work, the datasets are split into two subsets: a *training set* and a *testing set*. The training set is used to optimize the network, that is, the data within the training set is the data which must be fit by updating the weights. The testing set is the data the trained model is tested on, providing a measure for how well the network performs on data it has not seen before. To ensure an accurate estimation of the model's performance on general data, the test set should not be used to optimize the network in any way.

The datasets used should be of high quality. The definition of quality data, as provided by Roebuck [54], is data which is a 'correct representation of the real-world construct to which the data refers'. That is, the data should not contain a high degree of noise or incorrect labels. As we will see, the primary dataset used within this work is indeed of high quality. Further, Goodfellow et al. [15] propose a general guideline of 5000 entries within a dataset to obtain acceptable results with deep learning methods. Each of the datasets used within this work meet this demand by a considerable margin.

Cost Function

In order to optimize a network, we require an expression representing the current error in terms of the weights commonly called the *cost function*, $J(W)$. The closer the predicted outputs are to their true labels, the lower the error or loss [55]. For a binary classification problem, given some input **x**, the model must return the probability of the input belonging to the two respective classes. That is, in the context of this work, an output of $[0.2, 0.8]$ indicates a prediction that the input is 20% nonsarcastic and 80% sarcastic. The *binary cross entropy* loss function is ideal for binary classification. It utilises the negative log-likelihood, seen in Eqn. 2.5 to penalise the model for unconfident predictions.

$$L(p(y_i)) = -\log(p(y_i)) \tag{2.5}$$

where $L(p(y_i))$ is the error of the prediction $p(y_i)$ for a target label y_i. For example, if the target output is sarcastic, a predicted probability of 90% that the input is sarcastic would return a low loss, whilst a predicted probability of 20% would return a very high loss. The sum of these losses over each class, for each datapoint, is the overall loss of the network, that is, the cost function seen in Eqn. 2.6.

$$J(W) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i)) \tag{2.6}$$

where y_i is the target label and $p(y_i)$ is the predicted probability of that sentence belonging to the target class for all N datapoints. Hence, the goal during training is to find weights which minimise (W).

Effectively, the lower the loss of a network, the greater its *confidence* of the classification. Given a datapoint belonging to the second class, an output of [0.49, 0.51] would indicate the model classifies the sentence correctly. However, looking at the probability distributions, we can clearly see the network was not confident in the classification, predicting it had only a 51% chance of belonging to the target class. The loss incurred during training is the *training loss*, whilst the *testing loss* is the loss over the test set.

Optimisation

Given the size of some neural networks, the minimisation of the cost function, $J(W)$, is often too complex for analytical means. Hence, numerical methods are used to minimise the loss with respect to the weights, i.e to find a minimum within the error surface [52].

The standard technique used is called *Gradient Descent*, which updates weights iteratively by taking steps against the current error gradient, $\nabla J(W)$. The size of this step is the loss gradient times the *learning rate*, α. However, with the growth of neural networks and deep learning, gradient descent has been found to be problematic - often requiring a long period of time to converge [56]. Ruder [57] demonstrated the tendency for gradient descent to converge on local minima instead of the desired global minimum. He further highlighted the issues surrounding the choice of α, with small values leading to slower training times, whilst large values hinder convergence. Dauphin et al. [58] argued that the proximity to stationary points - within the error space - led to a slow progression of the search. Furthermore, they showed the prevalency of these saddle points in higher dimensional problems.

Numerous approaches have been proposed to overcome these challenges, with the *Adaptive Moment Estimation* algorithm (Adam) proposed by Kingma and Ba [59] being a popular choice. The algorithm sets a step size which is invariant to the error gradient, and incorporates the momentum from the previous iteration within the current iteration. Consequently, the algorithm does not slow down when near stationary points, and the use of the previous step's momentum ensures local minima are skipped. Furthermore, Adam implements an adaptive learning rate for different weights, effectively allowing sparse features to have greater impact on the search [52]. Adam is chosen as the optimisation technique throughout the work herein.

2.2.2 Evaluation Metrics

To evaluate the performance of our binary classifier, a few different metrics are used. Suppose we had some dataset with binary labels: 0 or 1. The classification of such a dataset has four potential outcomes: *true positive*, *true negative*, *false positive* and *false negative*. These outcomes, as defined by Fawcett [60], are detailed in the list below:

- True positive (TP): correct positive prediction.
- False positive (FP): incorrect positive prediction.
- True negative (TN): correct negative prediction.
- False negative (FN): incorrect negative prediction.

A 'positive' prediction usually indicates the relevant category being studied. That is, in the context of our work, a positive refers to some text which belongs to the sarcastic class. The following subsections use the above definitions to define their respective evaluation method. Initially, we present three statistical measures and then define two common metrics used for sarcasm classification: accuracy and F1-score.

Precision, Recall and Specificity

Below, we present the statistical measures *precision* (P), *recall* (R) and *specificity* (S), as defined by Powers [61].

Precision is the positive predictive value, it is calculated as the number of correct positive predictions divided by the total number of positive predictions, seen in Eqn. 2.7 below.

$$\mathrm{P} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}. \tag{2.7}$$

Recall, also referred to as *sensitivity*, is the true positive rate. It is calculated as the number of correct positive predictions divided by the total number of positives, seen in Eqn 2.8 below.

$$\mathrm{R} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} = \frac{\mathrm{TP}}{\text{Total Positives}}. \tag{2.8}$$

Finally, specificity is the true negative rate. It is calculated as the number of correct negative predictions divided by the total number of negatives, seen in Eqn 2.9 below.

$$\mathrm{S} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}} = \frac{\mathrm{TN}}{\text{Total Negatives}}. \tag{2.9}$$

Accuracy

Accuracy is simply a measure of the absolute correctness of the network. It is measured as the proportion of correctly classified data, divided by the total amount of data, seen in Eqn 2.10 below.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FN} + \text{FP}} = \frac{\text{TP} + \text{TN}}{\text{Total Positives} + \text{Total Negatives}} \tag{2.10}$$

This metric does not take confidence into account and is only concerned with the final classifications of the network. Consequently, a model may exhibit a high testing loss, but still obtain an excellent accuracy. This would arise from the network being uncertain during classification, that is, for some datapoint, the model outputs a probability tensor of [0.45, 0.55]. The network would classify this datapoint as belonging to the second class, given it had the higher probability. However, the network was not very confident in this classification, returning a 45% probability that it belongs to the first class and a 55% probability of belonging to the second. Based on the task at hand, this may or may not be of concern. Since we are only interested in whether the model accurately classified the text, confidence is mostly inconsequential. However, for example, within the domain of medical diagnostics, one would want a confident network, as the potential consequences of a misclassification are more severe than simply being incorrect.

Further, accuracy may not always be the best metric for evaluating performance, especially on imbalanced datasets [62]. Consider the following imbalanced dataset in which 90% of the data belongs to the first class, whilst 10% belongs to the second. Using accuracy as a measure of performance, the network would be able to obtain 90% by simply guessing that every datapoint belongs to the first class. By favouring the majority class, the model will appear to perform well based on accuracy, but in reality it actually performs poorly. This is problematic and thus better metrics exist for these circumstances, such as the F1-score presented next.

F1-score

F1-score is a commonly used metric for evaluating performance on imbalanced data. The measure takes into account the disproportionality between the two classes using the statistical measures precision and recall [61]. Effectively, the F1-score is the harmonic mean of precision and recall, shown below in Eqn 2.11.

$$F_1 = \left(\frac{2}{\text{recall}^{-1} + \text{precision}^{-1}} \right) = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}. \tag{2.11}$$

Finally, substituting the above formulae for precision and recall yields:

$$F_1 = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}. \tag{2.12}$$

This metric penalises a tendency to overly classify datapoints towards the majority class, which is clearly preferable on imbalanced datasets.

2.2.3 Activations

Sigmoid

The sigmoid function is an S-shaped curve, seen in Fig. 2.2, which squashes the input signal between 0 and 1. It has form:

$$f(\mathbf{x}) = \frac{1}{1+e^{-\mathbf{x}}}. \tag{2.13}$$

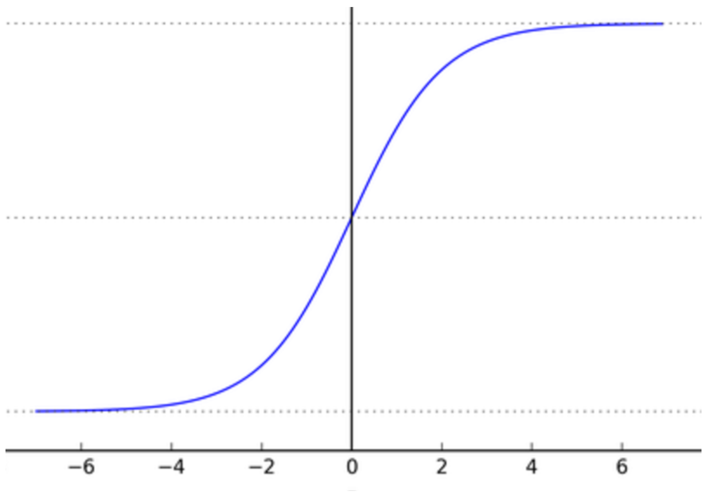


Figure 2.2 The Sigmoid curve, taken from [63].

Initially many models utilised the sigmoid as an activation function, however it has fallen out of popularity in recent times due to the following reasons:

- The vanishing gradient problem

- The output is not zero centered consequently making optimisation difficult
- Gradients may become saturated or killed
- Slow convergence

ReLU

The Rectified Linear Unit [64], or ReLU, casts negative values in the input signal to 0, seen in Fig. 2.3. Due to its linearity, the slope does not saturate when our input grows large, consequently resulting in quicker convergence compared to the Tanh function. Further, it is also free of the vanishing gradient problem [65]. The deactivation of negative values introduces sparsity within the network, this in turn results in a model less prone to overfitting, as well as faster performance due to the decreased number of computations [66].

$$f(\mathbf{x}) = max(0, \mathbf{x}). \tag{2.14}$$

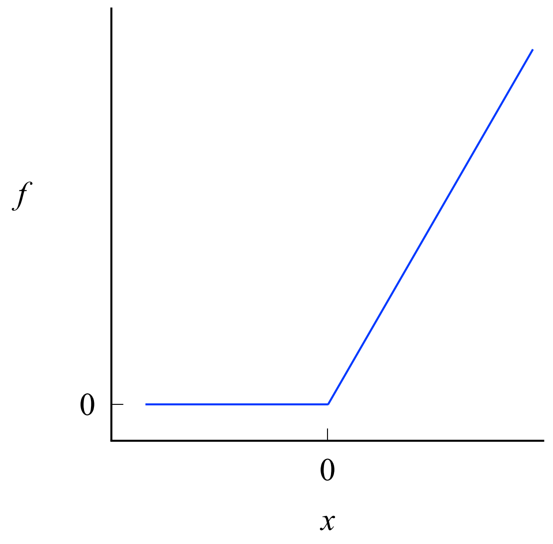


Figure 2.3 The ReLU function, adapted from [67].

In recent years, the ReLU function has become the standard in most deep learning models [68]. However, ReLU should only be implemented within the hidden layers of a network, as it is not suited for classification. Consequently, the output layers are Softmax functions (for classification into two or more classes) or a simple linear layer (for regression problems). Further, the presence of fragile gradients during the model's training may result in dead neurons, that is, a weight update which causes the neuron to never activate on any datapoints again. Consequently, a large portion of the network could end up doing nothing. This issue is more pronounced when the learning rate is too high or by the presence of a large negative bias [69].

In the event that the dying neuron problem cannot be mitigated with a low learning rate, the Leaky ReLU was proposed [69]. Rather than simply casting negative values to zero, a small slope is introduced in order to keep weight updates alive. That is, for a negative input, the output will have form:

$$f(x) = 0.01x \quad \text{when } x < 0. \tag{2.15}$$

Softmax

Within a neural network, the softmax function [70] maps the non-normalised output to a probability distribution over the predicted classes. That is, the function takes an input vector of N values and normalises it into a probability distribution of N probabilities. The standard form of the softmax function is seen in Eqn. 2.16 below:

$$\sigma(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}} \quad \text{for} \quad i = 1, \ldots, N \tag{2.16}$$

where $\mathbf{x} = (x_1, \ldots, x_N) \in \mathbb{R}^N$. The resulting output will have elements in the interval $(0, 1)$ and add up to 1, effectively allowing us to interpret it as probabilities for each class. Consequently, this function is used at the head of a network in conjunction with cross entropy loss for classification tasks [15].

2.2.4 Regularisation

A deep network has increased computational capacity compared to a shallower equivalent. However - due to the additional parameters - the model has a greater tendency to overfit [71]. A complex network may capture extra noise present in the data, consequently resulting in a network which fits the training set well, but generalises poorly. This can be identified by a very low training loss but high testing loss. We can consider this to be the model ‘memorising’ the data, instead of capturing the general trends present within. Regularisation is a form of regression, constraining the coefficient estimates towards zero; effectively discouraging the model from learning an overly complex function, primarily to prevent overfitting [72]. There are many forms of regularisation, in particular weight decay and dropout, which are expanded on below.

Weight Decay

Weight decay is a form of regularisation introduced by Krogh and Hertz [73] which penalises the learning of large weights. When weight updates are made, a small constant, wd, is defined, usually $wd = 0.1$. This constant is multiplied by the original weight, and used as the penalty term. That is, the current weight times wd is subtracted from the original weight during an update. This effectively scales the weights and encourages the network to learn values which are smaller in magnitude.

Weight decay has been observed to not only improve generalisation, but also training accuracy [74]. Loshchilow and Hutter [75] demonstrated that when Adam is used as the optimiser, weight decay led to a far better degree of generalisation as opposed to adding an L_2 regulariser. Further, Zhang et al. [76] identified the mechanism by which weight decay provides a regularising effect within a network using Adam and batch normalisation, arguing that the scaling of weights led to an increase in the *effective* learning rate of the model. This consequently results in an increased regularisation effect applied to gradient noise [77].

Dropout

Dropout, as proposed by Hinton et al. [78], is another regularisation technique to prevent overfitting by randomly omitting a proportion of feature detectors during a particular training case. Essentially, dropout functions by 'deactivating' a random subset of activations within each layer, with p indicating the probability of some output being deactivated. For example, a probability of $p = 0.5$ corresponds to a 50% chance that each activation will be switched off.

A mask of 0 and 1 is created during the forward pass and applied to the layer's output, activations which are masked with a 0 are switched off, whilst those masked with a 1 are unchanged. Consequently, during backpropagation only the active neurons are considered for the calculation of error gradients. This effectively causes the network to learn the same concept with different combinations of neurons each time. This can be viewed as using a 'thinner' equivalent of the model during training time. Noting that during testing, dropout is deactivated and the complete network is used. This has been shown to increase a deep model's testing accuracy and decrease its tendency to overfit [79]. We see an example of dropout applied to a simple fully connected network in Fig. 2.4 below.

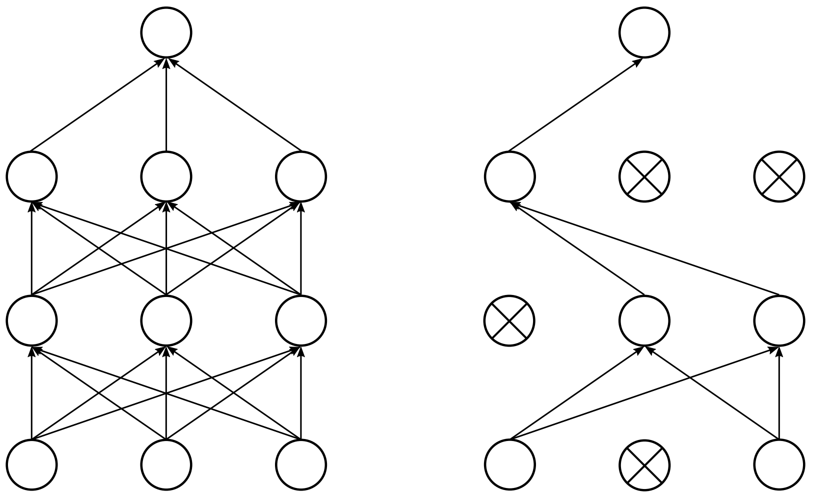


Figure 2.4 A 4 layer fully connected network before (left) and after (right) dropout is applied, adapted from [79].

2.2.5 Word Embeddings

In the 1990s, language representation models were dominated by discrete vector-space representations which depended on large and often sparse matrices [80]. In particular, Bag-of-Words (BOW) representations and feature extraction methods such as Tf-idf [81] and count vectorizers were popular. In 2003, the substitution of sparse n-gram models with continuous, dense representations of words was proposed, leading to the first notion of *word embeddings* [82], also named *word representations* or *word vectors*. Simply, this is the transformation of words in the vocabulary to vectors of continuous real numbers, e.g. *good* $\rightarrow$ (…,0.29, …,0.46, …). This is facilitated by a mathematical embedding from a high-dimensional sparse vector space (such as a one-hot encoding of sentences) to a denser, lower-dimensional space [83]. Using feed-forward or more advanced network architectures the co-occurrence of words and other linguistic regularities and patterns may be encoded into the vectors [80]. An example of which being contextually similar words are mapped closer in the embedding space than contextually dissimilar words.

One of the first popular models was proposed by Collobert and Weston [39], utilising a convolutional neural network to generate the word vectors. Applying these word representations to various NLP tasks, such as named entity recognition and semantic role-labeling, showed significant improvements without the need for hand-crafted features. Since then, word embeddings have been used in a vast variety of NLP tasks and have shown competitive performance across many domains [84] [85] [86] [87].

Many methods of generating word embeddings have been proposed, however a select handful have seen extensive use, such as Word2vec, GloVe and FastText. These are outlined below.

Word2vec

Mikolov et al [88] proposed two methods, Continuous Bag-of-Words (CBOW) and Skip-Grams (SG), for generating word embeddings through the use of a shallow network architecture. These ideas were expanded in [89] via the incorporation of negative sampling and sub-sampling of frequent words, leading to the release of Google's vastly popular *word2vec*[1] model. The CBOW network's goal is to predict a word given a context window, that is, given "The man is _ a car", the model will predict "driving". The SG version aims to do the opposite, given some word, it will predict the context window [80]. Statistically, the CBOW model is able to smoothen over a large amount of distributional data by treating the entire context as a single observation, making it effective for short sentences and large datasets. However, the SG model is better suited for longer sentences with smaller datasets as it treats each context-target pair as a new observation [83].

GloVe

Global Vector [90], or *GloVe*[2], is another frequently used word embedding model. The network is built around the important idea that one can derive semantic relationships between words from a co-occurrence matrix [91]. Consequently, it is trained on the non-zero entries of a global word co-occurrence matrix. This effectively allows GloVe to incorporate global corpus statistics, whilst also preserving the underlying linear substructures present in word2vec type models [83]. Research has shown GloVe scales well on larger text corpora and outperformed state-of-the-art methods in word analogy, word similarity and named entity recognition tasks [90]. Furthermore, Çano and Morisio [80] demonstrated GloVe embeddings were better for sentiment analysis compared to word2vec.

FastText

Bojanowski et al. [92] expanded on word2vec by treating each word as a bag of character n-grams. That is, the embedding for some word is represented as a sum of its character-level embeddings. This resulted in better representations for rare and out-of-corpus words, as an appropriate embedding could be assigned to an unknown word through the use of its character n-grams. This is in contrast to other approaches which cannot effectively handle unknown words. This was further

[1]https://code.google.com/archive/p/Word2Vec/

[2]http://nlp.stanford.edu/projects/glove/

enhanced by Joulin et al. [93] by deriving a text embedding, for some input text, from the average of its word embeddings. A prediction of the sentence label is then made using this latent text embedding. These two works formed the base of the *FastText*[3] embeddings [94].

2.2.6 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a form of artificial networks which capture hierarchical features from the input data to produce a global feature vector through the use of convolutions [95]. Unlike fully connected layers, neurons within a convolution-layer are connected to a subset of the following convolution-layer's neurons. This idea followed the discovery of the locally-sensitive, orientation-selective neurons within the mammalian visual system [96]. CNNs have since attained state-of-the-art results on many image processing tasks [97] [44] [98] [99], however their efficacy across numerous NLP domains has also been established [100] [101] [102]. An outline of convolutions and feature extraction is presented next, followed by the structural considerations required to implement a CNN for text.

Convolutions

Given some 2-dimensional data, a convolution is performed by sliding a *filter*, or *kernel*, over the input. The filter is simply a small matrix of weights which performs elementwise multiplication to produce an output matrix, called a *feature map*. The weights within a filter are learnt during training to capture specific characteristics of the input. Effectively, the filter represents the local connectivity between neurons and can be viewed as the receptive field of a biological neuron [103]. A simple convolution is illustrated below in Fig. 2.5.

Multiple feature maps can be produced through the use of many filters. These output feature maps form the output *channels* within a CNN, whilst the input matrices form the input channels. An intuitive explanation for the channels is the RGB values for some image: the first channel corresponds to red, the second to green and the third to blue. As a result, an image with a single input channel would be grayscale. The idea is to extract different features with each channel, and then combine these to form more complex features within the next layer. Consequently, sequential convolutions result in increasingly hierarchical features being captured.

[3] https://fasttext.cc/docs/en/english-vectors.html

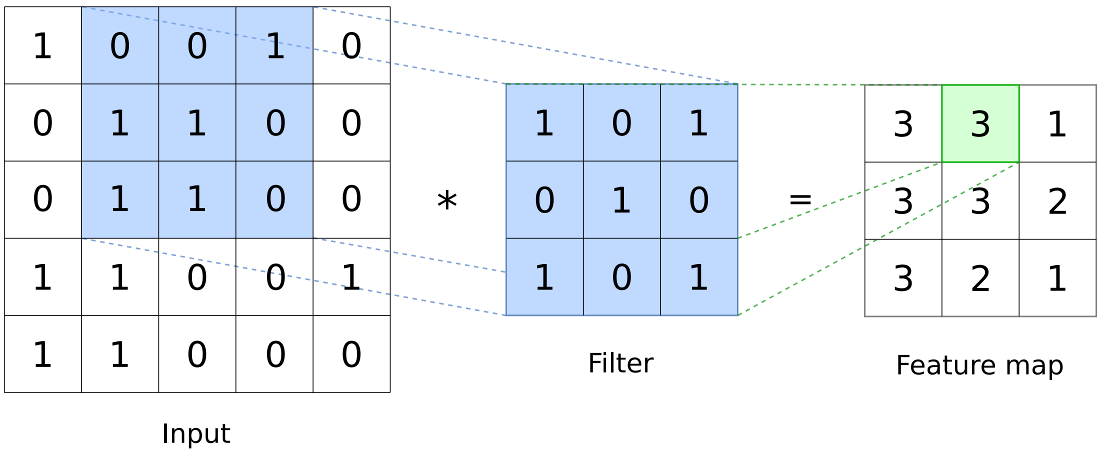


Figure 2.5 A 3x3 filter convolving over a 5x5 input to produce a 3x3 feature map.

Given this, the activation of the j^{th} neuron in the i^{th} layer will have form:

$$x_j^{(i+1)} = f\Big(\sum_{l \in C_j} x_l^{(i)} * k_{lj}^{(i)}\Big) \tag{2.17}$$

where C_j represents the input channels and $k_{lj}^{(i)}$ is the filter in the i^{th} layer representing the weights between the preceeding l^{th} neuron and the j^{th} neuron [104].

Goodfellow et al. [15] identified three concepts which facilitated the performance of convolutions: *sparse interactions*, *parameter sharing* and *equivariant representations*. Given the locally connected architecture of a CNN, the neuron interaction is said to be sparse. Decreasing the filter size minimises this interaction and allows for key features to be detected with fewer parameters, resulting in a decreased memory requirement and fewer operations performed in computing the output. Parameter sharing arises from the convolution operator. As the filter slides along the input, parameters are shared amongst all input elements. Whilst the model runtime is unaffected [52], the storage requirements are greatly reduced. Parameter sharing causes the layers in a CNN to inherit the property of equivariance, that is, changing the input results in the same perturbance of the output.

Pooling Layers / Feature Extraction

After a convolution is performed, the resulting output channels are often passed to a pooling layer to extract the features captured within. Simply, the aim of a pooling layer is to represent some feature vector as a smaller vector containing the most important features. This is often done through the use of max pooling or average pooling.

Max pooling takes the highest value of an input vector and returns it, whilst k-max pooling returns the max value for every k values. This is effective when dealing with images, as the highest value could be seen as the predominant pixel color, and taking the max would give a good indication of the colour of the input pixels [105].

Average pooling, as the name implies, returns the average over the input vector. Similarly to k-max pooling, the input can be divided and averages obtained over each division. Max pooling consequently discards some information by only considering the largest values, whilst average pooling provides a more general extraction mechanism [106].

Pooling layers are usually used after repeated convolutions, in order to combine the channels into a single feature vector. However, they can also be present between layers to facilitate the downsampling of input dimensions [15], expanded on in Section 3.2. Furthermore, pooling layers can be likened to convolutions, where the filter simply returns the max or average value as it slides over the input.

Architecture for Text

The input to a CNN for text classification is usually a tokenized sentence. Utilising word embeddings, a sentence matrix is formed, containing the vector representations of each token in the input [107]. Given an embedding of dimension d and a sentence W of length s, a sentence matrix of dimension $s \times d$ is created. Padding is used to ensure each sentence in the corpus has equal length, which effectively allows us to treat the input sentence as an image with 1 channel.

A convolution is then performed over this matrix, which is believed to capture various n-grams from the input through differing sized filters [108]. That is, given our input sentence $W = w_1, \ldots, w_s$, each token is embedded as a d-dimensional vector, resulting in the sentence matrix $\mathbf{W} =< \mathbf{w}_1, \ldots, \mathbf{w}_s > \in \mathbb{R}^{n \times d}$. Here, we have used the same notation for the word embedding vectors as was used for the weights in the previous section. This is by design as later on we will attempt to learn better representations as training takes place in the same manner in which we learn better weights.

A filter of size $k \times d$ is applied over matrix $\mathbf{W}$, with dot products taken to obtain information about the various k-grams encountered by the filter [100]. This is explained in more detail in Section 3.2.1. The resulting feature maps are fed to a pooling layer, with outputs passed to a non-linearity, usually ReLU. Finally, a fully connected layer, with a softmax activation, is implemented at the head of the model to output the classification result. Many filter sizes can be used in parallel by concatenating the results before passing them to the fully connected layers.

2.2.7 Residual Neural Networks

The hierarchical feature detection of CNNs is a result of stacking convolution-layers, with detected features becoming increasingly complex as the input progresses through the network. However, simply increasing network depth has been observed to cause the infamous vanishing gradient problem [109], which can be described as the shrinking of gradients towards zero resulting from repeated applications of the chain rule. Further, deeper networks are inherently more complex, causing an increased tendency to overfit [71]. He et al. [41] demonstrated a 56-layer CNN had a higher training and testing error than that of a 20-layer CNN on image classification tasks, providing empirical evidence that, after a certain degree, deeper networks result in model degradation. A framework, based on the mathematical concept of residuals, was proposed to investigate and potentially provide a solution to this problem, detailed below. These networks were called *residual neural networks*, or *ResNets*.

Residual Learning

Consider the underlying mapping $H(\mathbf{x})$ to be fit by a subset of the network (that is a few stacked layers), with $\mathbf{x}$ denoting the input to this stack. One might hypothesise that multiple non-linear layers can asymptotically approximate complicated functions (however, this is still an open question [110]). Assuming this, a residual function, $H(\mathbf{x}) - \mathbf{x}$, can be similarily approximated. Hence, instead of expecting the stacked layers to approximate $H(\mathbf{x})$, we let the layers approximate a residual $F(\mathbf{x}) := H(\mathbf{x}) - \mathbf{x}$. Consequently, the original function becomes $F(\mathbf{x}) + \mathbf{x}$. Given the above hypothesis, both forms should be capable of effectively approximating the desired function, however, the ease of learning may differ. The authors were motivated by the counter-intuitive result of the deeper network degrading the performance, hypothesising that the networks had difficulty approximating identity mappings over multiple non-linear layers. This reformulation allows the residuals to be driven towards zero, $F(\mathbf{x}) = 0$, in the case that an identity mapping is optimal.

Shortcut Connections

The network's building blocks adopted residual learning to every two (basic block) or three (bottleneck block) convolution-layers. Here a convolution-layer consists of a [Conv, BatchNorm, ReLU] sequence. Formally, the building block is considered to be:

$$\mathbf{y} = F(\mathbf{x}, \{W_i\}) + \mathbf{x} \tag{2.18}$$

where $\mathbf{x}$ and $\mathbf{y}$ are the input and output of the considered layers and the function $F(\mathbf{x}, \{W_i\})$ represents the residual mapping to be learned. The shortcut connection performs the operation $F + \mathbf{x}$ by element-wise addition on the feature maps, channel by channel, with the final non-linearity taking place after the identity is added. Given this, a residual block with 2 stacked layers will learn a residual mapping $F = W_2(W_1\mathbf{x})$. We note that the shortcut connections do not introduce additional parameters, nor increased computational complexity. Clearly, the dimensions of $\mathbf{x}$ and F must be equal in Eqn. (2.18), however this is not always the case. To ensure matching dimensions, down-sampling is performed by means of some linear projection W_s (typically a 1×1 convolution with stride 2). That is:

$$\mathbf{y} = F(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}. \tag{2.19}$$

Architecture

The basic block consists of two 3×3 convolution-layers, as described above. The bottleneck block was designed to decrease time complexity when training very deep models. Hence, it consists of three convolution-layers with filters of size 1×1, 3×3 and 1×1, respectively. The two 1×1 convolutions decrease and increase the dimension by a factor of 4, allowing the expensive 3×3 convolution to be performed quicker. This is seen in Fig. 2.6.

The input initially goes through a 7×7 convolution with stride 2, followed by a maxpooling layer with stride 2, before entering the residual-layers, described below.

The residual component of the network is built by 4 residual blocks, and follows 2 design rules:

- The channel count remains constant in each residual block.
- If the input dimension is halved, the number of channels is doubled, effectively preserving the time complexity per block.

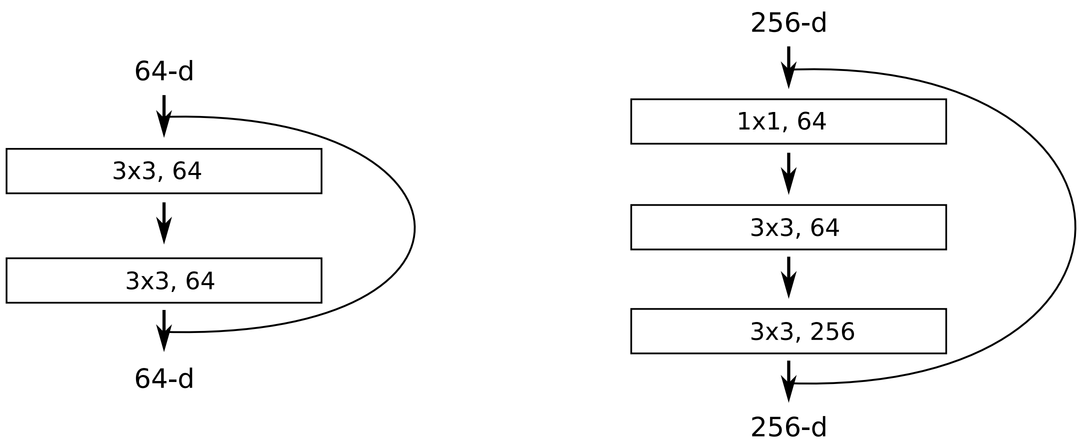


Figure 2.6 The residual building blocks: **Left:** Basic. **Right:** Bottleneck (adapted from [41]).

Following these rules, we can build networks of increasing depth by simply adding more layers to the 4 residual blocks (res-blocks). The initial convolution-layer within the second, third and fourth res-block performs downsampling by adopting a stride of 2. Consequently, shortcut connections between res-blocks will have form Eqn. (2.19), whilst shortcut connections within a res-block will have form Eqn. (2.18). A global average pooling layer [111] followed by a 1000-way fully connected layer with softmax activation completes the network.

2.2.8 Densely Connected Convolutional Neural Networks

Within typical CNN and residual architectures, the final classification is made using the most complex features. This introduces a potential issue when going very deep: information regarding the input or gradient may be lost once it has passed through the entire network. The concept of providing paths between early layers and later layers has been implemented in a variety of papers. Above we saw shortcut connections within ResNets [41], effectively sharing the layer's input with the final activation function. Other networks utilising this include: Highway Networks [112] and FractalNets [113]. Working with the intent to maximise the flow of information between layers, Huang et al. [42] introduced the concept of *densely connected convolutional neural networks*, or *DenseNets*, expanded on below.

Dense Connectivity

In contrast to residual blocks, where the identity is added to the output, DenseNets introduce direct connections from a layer to all subsequent layers within the dense block. Specifically, features are

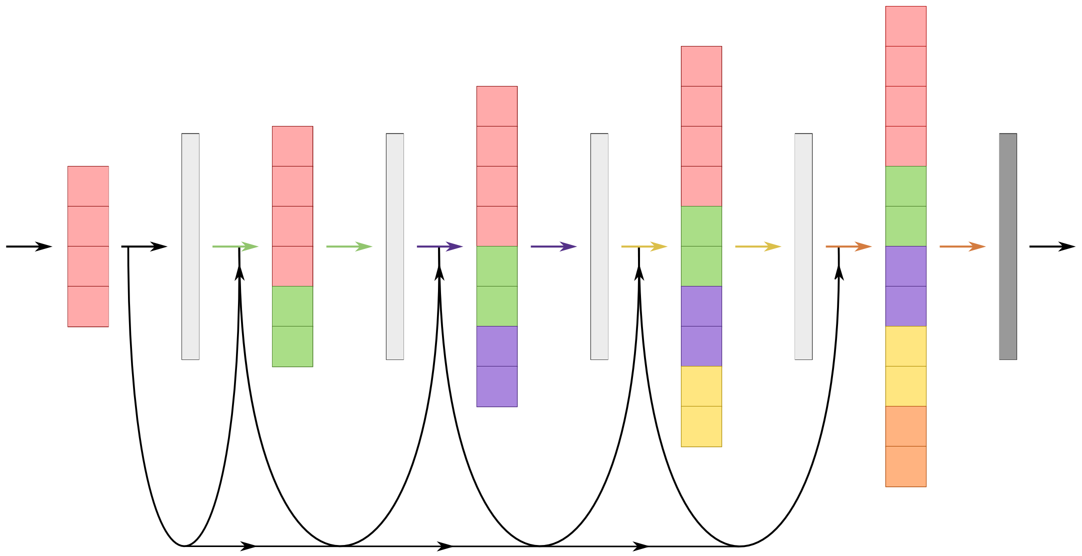


Figure 2.7 A 4 layer dense block.

combined using concatenation, effectively allowing each layer to receive a "collective knowledge" of all prior layers. Formally, the i^{th} layer of a dense block can be viewed as:

$$\mathbf{y}_i = F_i([\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{i-1}]), \tag{2.20}$$

where $\mathbf{y}_i$ is the layer's output, $F_i(\cdot)$ is the composite function of the three consecutive operations [Conv, BatchNorm, ReLU], $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{i-1}$ are the feature maps corresponding to the preceeding $i-1$ layers and $[\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{i-1}]$ refers to their sequential concatenation. A 5-layer dense block can be seen in Fig. 2.7.

The four red blocks represent the input channels passed to the first layer, indicated by the light gray rectangle. This layer outputs 2 green channels which are concatenated to the input before being passed to the next layer. This process repeats until the concatenated channels are passed to a transition layer, represented by the dark gray rectangle.

The implementation of these direct connections facilitated the production of rich and diverse feature maps, by combining low-features extracted during the initial convolutions, as well as complex high-level features found deeper in the network. The authors demonstrated the dependency of a feature map, with all those prior to it, providing empirical evidence that the final classification made use of both low and high-level features.

Architecture

A fundamental facet of convolutional networks is downsampling to change the feature map size. This would clearly cause issues with the concatenation operation in Eqn. (2.20). To accommodate this, similarly to a ResNet, the network is split into multiple dense blocks separated by a transition layer. The transition layer facilitates downsampling through a 1×1 convolution followed by 2×2 average pooling.

Within a denseblock, each layer produces GR feature maps which are concatenated to the input of the following layer. This causes an increase in the number of feature maps as one progresses through the block. GR is defined as the *growth rate* of the network and essentially regulates how much new information each layer adds to the global state.

Bottleneck layers were also investigated. The input dimension is first reduced by a 1×1 convolution before performing the 3×3 convolution. Unlike a residual model, the dimension is not restored to its initial state but instead the latter 1×1 convolution outputs $4GR$ feature maps. A compression factor, θ, is defined to compact the model. If a dense block produces m feature maps, the transition layer generates θm maps, where $0 < \theta \leq 1$. Hence, a value of $\theta = \frac{1}{2}$ would cause the transition layer to halve the channel count. This prevents the model from growing excessively large with time when using large growth rates or deep depths.

In their experiments, bottlenecking and compression proved beneficial in decreasing the error rate across several benchmark datasets. Further, it was shown DenseNets outperform their ResNet counterparts on all benchmarks.

Chapter 3

Method

In this chapter the models investigated, datasets used and experiments performed are detailed. We begin by outlining the datasets and provide details regarding their testing and training sizes. Next, we describe the preprocessing methods used to transform the data into a suitable format for use within our networks.

Three architectures are investigated within, with a description of each presented. These proposed archetypes have many structural considerations during implementation, such as the number of hidden layers. With this in mind, we propose experiments into structural alterations of each network.

From the results of the above structural investigations, three final models are proposed, two of which are treated as baselines, and the third as our proposed approach. The model chosen as the proposed approach is based on its performance during the preliminary experiments. An outline of the benchmarking procedure follows, detailing the hyperparameters used, the metrics for measuring performance and how results are obtained.

Finally, exploratory investigations are presented, which are concerned with non-structural aspects of the networks, such as the suitability of the preprocessing techniques used, as well as how local context impacts testing performance.

3.1 Datasets

Three datasets are used to compare and evaluate the various models described herein. The sentences in the first dataset are limited to 64 tokens in length, whilst the latter two datasets are limited to a length of 128 tokens. Sentences exceeding these lengths are removed from the dataset. This is done to better facilitate padding, which is needed for batched convolutions.

The first dataset considered is the Kaggle *Newsheadlines dataset for Sarcasm Detection* [114], hereby referred to as the *Headlines* dataset. Many prior papers have focused on social media datasets (Twitter being the predominant source), which are noisy and *informal* in structure. Informal in the context of this work is a dataset which makes use of colloquialisms, contains spelling errors and excessive punctuation. To avoid these issues, this dataset was constructed by collecting headlines from two news websites: TheOnion[1] and HuffPost[2]. The former is well-known for producing satirical versions of current events and comprises the sarcastic entries in the dataset whilst the latter is a legitimate news site with their headlines used as the nonsarcastic entries. This dataset is used as the main corpus throughout this work for the following reasons:

- The headlines are written formally, that is they are free of spelling mistakes and informal language use.
- TheOnion solely publishes satirical news, hence we are able to obtain high quality labels with considerably less noise than social media datasets.
- Social media posts are often made in response to another post, whilst news headlines are self-contained. This allows us to investigate whether the networks are able to model the sarcastic characteristics without local context being of concern.

The dataset is fairly balanced, consisting of a total of 26709 entries in which 11724 are sarcastic and 14985 are nonsarcastic. The data is randomly split into a training and testing set, with the testing set consisting of 20% of the data. The final training and testing splits are seen below in table 3.1, noting that the ratio between sarcastic and nonsarcastic entries is 1:~1.3 in both sets.

	Nonsarcastic	Sarcastic
Training set	11988	9379
Testing set	2997	2345

Table 3.1 Training and testing splits for the Headlines dataset.

The second dataset considered is the main balanced Reddit dataset from the paper *A Large Self-Annotated Corpus for Sarcasm* - SARC - presented by Khodak et al. [115], hereby referred to as

[1] https://www.theonion.com/
[2] https://www.huffpost.com/

the *Reddit Main* dataset. We note that the SARC V2.0[3] option is used throughout this dissertation. The authors generated the data by using the Reddit API to scrape comments containing the tag '\s' for the sarcastic entries, whilst comments lacking the tag were taken as non-sarcastic. The tag, '\s', is commonly used by Redditors to indicate their comment is in jest, making it a reliable indicator of sarcasm. The use of informal language and spelling errors is frequent in this dataset, as well as the potential for noisy labels: a user could have made a sarcastic comment, but forgot to use the tag, effectively causing the sarcastic comment to be labeled as nonsarcastic.

In their investigations, they found that humans outperformed their various proposed models, arguing that contextual information is required to detect sarcasm in certain topics. The dataset includes the parent comment (i.e. what the user was replying to), as well as additional child comments for each entry. Furthermore, it contains details regarding which subreddit the post occured on, the number of upvotes it received, and other metadata. We make use of only the original child comment throughout our work, that is, no context or metadata is incorporated into our models. This is done to determine whether our network - which models only the isolated utterance - is able to rival recent approaches which make extensive use of the context and metadata.

The dataset is considerably larger than the Headlines dataset, consisting of a total of 267615 comments in which 137233 are sarcastic and 130382 are nonsarcastic. The authors provide the training and testing splits, with details thereof contained in Table 3.2 below, noting that the sarcastic to non-sarcastic ratio is 1:~0.95 for both sets.

	Nonsarcastic	Sarcastic
Training set	104209	109713
Testing set	26173	27520

Table 3.2 Training and testing splits for the Reddit Main dataset.

The final dataset used is the political Reddit dataset from SARC V2.0, generated in the same manner as the above, however only the political subreddit is scraped. This dataset is hereby referred to as the *Reddit Pol* dataset. It is comprised of 17074 comments, equally split into 8537 sarcastic and nonsarcastic entries. Table 3.3 details the training and testing splits, noting that the ratio between sarcastic and nonsarcastic entries is exactly 1:1.

[3]http://nlp.cs.princeton.edu/SARC/2.0/

	Nonsarcastic	Sarcastic
Training set	6834	6834
Testing set	1703	1703

Table 3.3 Training and testing splits for the Reddit Pol dataset.

Finally, a summary of the three respective datasets is presented in Table 3.4 below.

	Training		Testing	
	nonsarc	*sarc*	*nonsarc*	*sarc*
Headlines	11988	9379	2997	2345
Reddit Main	104209	109713	26173	27520
Reddit Pol	6834	6834	1703	1703

Table 3.4 Summary of the Headlines and Reddit datasets.

3.1.1 Data Preprocessing

Data preprocessing is defined as any form of preprocessing performed on raw data to prepare it for another procedure. Preprocessing transforms the data into a format which will be more readily processed by the following procedure. In the context of this work, data preprocessing occurs in two main stages, namely: *Tokenisation* and *Numericalisation*.

Tokenisation

Tokenisation is defined as the splitting of a sentence into individual tokens which, when placed together, form the original sentence. Splitting occurs over blank spaces, punctuation and word contractions (e.g., don't = ["do" "n't"]). The default tokeniser used throughout this work is the spaCy tokeniser[4]. Consider the sentence 'It's so cold today, yesterday was warmer!'. After tokenisation, the sentence can be represented as a list of its tokens ["It" " 's" "so" "cold" "today" "," "yesterday" "was" "warmer" "!"].

We note that during tokenisation we do not strip punctuation as many prior papers have done. The

[4]https://spacy.io/api/tokenizer/

reasoning behind this decision is that a degree of information is potentially lost when punctuation is discarded. Consider Table 3.5 below, containing three sentences with and without punctuation.

With Punctuation	Without Punctuation
We are ready to eat, Grandad.	We are ready to eat Grandad
You look so. . . pretty.	You look so pretty
We had such a great day, yesterday.	We had such a great day yesterday

Table 3.5 Texts with and without punctuation.

The first sentence implies that food is ready to eat and Grandad is being informed of this. However, after removing the punctuation, the new sentence implies that they are ready to literally eat Grandad. The removal of this single comma, changes a rather ordinary sentence into something somewhat sinister. In the second entry, the pause before the word ‘pretty’ suggests insincerity. However, after removing the punctuation, it appears to be a regular compliment. Finally, the third sentence implies that today was not so great but without punctuation it appears to just be a statement on what a good day yesterday was. From these three examples we can readily see the importance of punctuation in conveying intent. For this reason we retain all punctuation in the original text and treat them no different from regular words.

Special tokens are provided by the FastAI[5] framework to handle certain particularities such as, a word being written entirely in ALL CAPS or the repetition of certain words or characters. In prior papers, the norm has been to lowercase all text and remove any repeated occurrences, but this too, leads to a loss of information and potential ambiguity in the conveyed message. Consider Table 3.6 below with the original sentences and their processed counterparts.

Original	Processed
You are SUCH a bitch.	You are such a bitch.
You are sooooo smart.	You are so smart.
I love love love love you.	I love you.

Table 3.6 Original texts compared to their processed counterparts.

The first sentence clearly implies aggression, however, after lowercasing ‘SUCH’, the sentence could be read in a friendly non-condescending manner, depending on the relationship between the speaker and listener. In the second entry, the sarcastic undertone is very clear, the speaker implies

[5] https://www.fast.ai/

that the listener is not smart. However, after repetition is removed, the processed text appears to be complimentary. Finally, the third sentence appears childish and insincere, whilst the processed counterpart seems formal and sincere. This again shows us the potential loss in the intended meaning of an utterance caused by certain processing techniques. The special tokens provided to handle these nuances are listed in Table 3.7 below.

Token	Description
xxunk	For unknown tokens, i.e words not present in the vocabulary.
xxpad	For padding sentences to a set length.
xxbos	Indicates the beginning of a sentence.
xxfld	Indicates separate fields in text when using data from multiple columns in a database.
xxmaj	Indicates the next word begins with a capital in the original text.
xxup	Indicates the next word is written in all caps in the original text.
xxrep	Indicates the next character is repeated *n* times in the original text.
xxwrep	Indicates the next word is repeated *n* times in the original text.

Table 3.7 Special tokens used during the tokenisation.

Pre-rule	Description
fix_html	Replaces HTML characters/norms.
replace_rep	Whenever a character is repeated more than 3 times, it is replaced by ‘xxrep n char’, where n is the number of occurrences.
replace_wrep	Whenever a word is repeated more than 3 times, it is replaced by ‘xxwrep n word’, where n is the number of occurrences.
spec_add_spaces	Add blank spaces around the ‘/’ and ‘#’ tokens present in the original text.
rm_useless_spaces	Removes multiple consecutive spaces in the text.
Post-rule	Description
replace_all_caps	Replace tokens in ALL CAPS in text with lowercase version and add ‘xxup’ before.
deal_caps	Replaces a token beginning with a capital with lowercase version and add ‘xxmaj’ before.
pad_to	Pads the input with ‘xxpad’ to a set length.

Table 3.8 Pre and post-processing rules used during tokenisation.

The rules provided by the FastAI framework, which are used for text processing, are split into preprocessing and post-processing rules. The pre-rules are applied prior to tokenisation whilst post-rules are applied after. An outline of each rule and its function is seen in Table 3.8.

We note that the 'pad_to' rule is used to pad the Headlines data to lengths of 64, whilst the Reddit Main and Reddit Pol data is padded to a length of 128. This ensures that each sentence in the datasets have the same length, allowing for batched convolutions to be used.

Numericalisation

Numericalisation is the process of converting a set of tokens to unique IDs, effectively allowing them to pass through embedding layers and be assigned representations within the embedding space. FastAI's Vocab class is used for this purpose, storing the vocabulary encountered within the corpus, as well as the correspondence between IDs and tokens. The Vocab class has a threshold on the number of times a word must appear within the corpus to be placed in the vocabulary. The default threshold is 3, that is, a token must occur at least 3 times to be considered in the vocabulary. Words which appear less than 3 times are replaced with the 'unknown' token explained above. An investigation surrounding this threshold is outlined in Section 3.4.2. Consider the tokenised sentence ["we" "are" "going" "on" "holiday"]. After numericalisation, the sentence could be represented as a list of integers, e.g., [2, 3, 57, 23, 19].

Once a text has been numericalised, passing it to an embedding layer is simple. A weight matrix - which can be viewed as a lookup table - is created, containing the embedding for each word in the vocabulary. We can then pass the ID of a token to the embedding layer, and it will return the vector representation of the token. We note that the embedding of the padding token, "xxpad", is simply a list of zeros, and is kept constant throughout the model's training. This is done to prevent the padding from influencing classification.

3.2 Architectures

To evaluate the efficacy of deep learning networks in sarcasm detection, three different architectures were explored: a CNN, ResNet and DenseNet. Investigations into structural alterations of these networks are performed to determine the final configuration of each model. These experiments include adding hidden layers to the head of the network, activation functions and model depth. All models utilise a kernel size of $k = 3$ within their respective blocks. Essentially, the networks are left to learn hierarchical combinations of these 3-grams which are indicative of sarcastic intent.

3.2.1 Architecture Descriptions

Below we introduce the general models for each of the three proposed architectures. These are used to investigate various structural considerations to best capture the concept of sarcasm. We begin with the CNN, then the ResNet and finally the DenseNet. All 3-dimensional example tensors mentioned below will have form:

$$\big[\text{Channel size, Signal size, Embedding size}\big]. \tag{3.1}$$

Similarily 2-dimensional tensors will have form:

$$\big[\text{Channel size, Signal size}\big]. \tag{3.2}$$

The proposed models all begin with the same two starting layers: an embedding and an initial convolution. We remind ourselves a convolution layer consists of a [Conv, ReLU, BatchNorm] sequence, shown in Fig. 3.2. The embedding layer accepts the input text and returns the vector representation for each respective token in the sentence. The initial convolution layer follows, accepting the embedding tensor and outputting n initial channels, with a default value of $n = 16$ used during experimentation.

The 2D-convolution is performed in a 1-dimensional manner as such: the signal dimension is padded once on either side and a kernel of size $k = (3, ED)$ slides over the 3-gram embeddings of the text, where ED is the dimension of the embedding space. This effectively covers the entire embedding and the kernel consequently only moves in a single direction, reducing the embedding dimension to a size of 1. Hence, whilst this convolution has a 2-dimensional kernel, it can be viewed as a typical 1D-convolution.

Figure 3.1 depicts a simplified version of our 2D kernel performing a 1D convolution. Each word is represented by a 5D vector, the kernel, illustrated as the opaque blue rectangle, spans the entire embedding dimension but only covers the three words ‘is’, ‘a’ and ‘complex’. Hence, as it convolves over the input, it simply slides down to the next word and will cover ‘a’, ‘complex’ and ‘linguistic’, effectively behaving in a 1-dimensional manner. The resulting feature map from the first convolution is represented by the blue square containing the number 3.

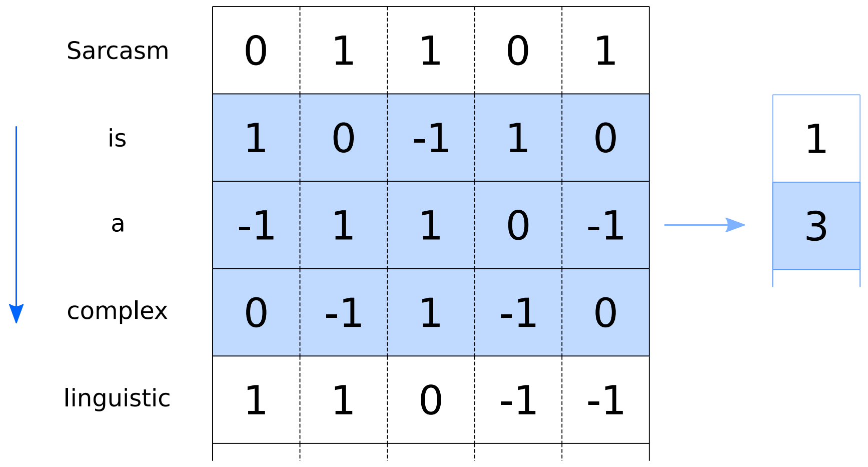


Figure 3.1 Performing a 1D convolution using a 2D kernel.

Consider the following equation, for calculating the output signal, L_{out}, for an input signal, L_{in}, after a convolution:

$$L_{out} = \left\lfloor \frac{L_{in} + 2P - K}{S} + 1 \right\rfloor \tag{3.3}$$

where P is the padding count, K is the kernel size and S is the stride. We do not wish to alter the signal dimension during the initial convolution, that is, we require $L_{in} = L_{out}$. For this to be true, using our default kernel size of $K = 3$ and a stride $S = 1$, we would require a padding count $P = 1$. Using these values we indeed do not change the input signal, shown below:

$$\begin{aligned} L_{out} &= \left\lfloor \frac{L_{in} + 2 \times 1 - 3}{1} + 1 \right\rfloor \\ &= \lfloor L_{in} + 2 - 3 + 1 \rfloor \\ &= \lfloor L_{in} \rfloor \\ &= L_{in}. \end{aligned} \tag{3.4}$$

This is the reason we pad the input signal once on each side.

For example, given word embeddings of dimension 50 and a sentence length of size, $s = 64$, we would receive a $[1, 64, 50]$ tensor from the embedding layer. This is passed to the initial convolution resulting in a $[16, 64, 1]$ tensor, which is then squeezed to remove the singular embedding dimension before being passed to the initial BatchNorm and ReLU layers. The tensor is then passed through the corresponding blocks for its respective archetype. The default pooling operation for feature extraction following the final block is average pooling, however other variants are explored, details of which are contained in Section 3.2.2.

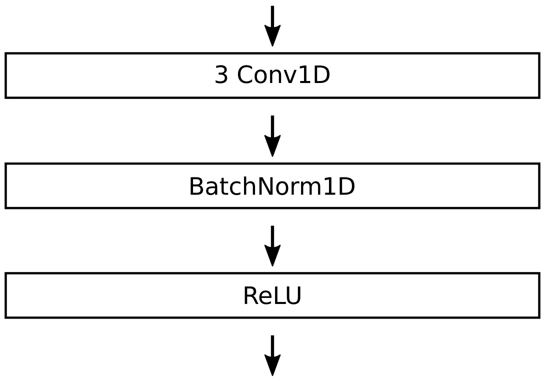


Figure 3.2 A convolution layer with a kernel size of 3.

CNN

The CNN is built from 4 convolution blocks consisting of 2 convolution layers per block. The intial convolution layer within each block doubles the channel count, whilst the second layer performs downsampling through a stride of 2, halving the signal dimension. A typical conv-block is pictured in Fig. 3.3 where ‘Conv-Layer(I, O)’ is a convolution layer with I input channels and O output channels.

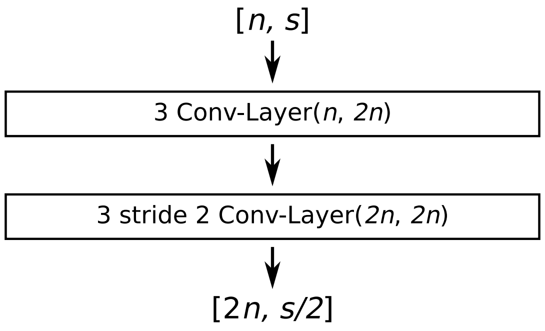


Figure 3.3 A convolution block consisting of 2 conv-layers which double the channel count and halve the signal respectively.

The final block is followed by a pooling layer, extracting the most important features identified within the text. The output tensor of this pooling layer is then passed to a linear layer with an output size of 2, representing the two possible outcomes: sarcastic or nonsarcastic. Finally, a Softmax layer determines the classification of the input text.

In Table 3.9 below, we see an example input tensor change as it moves through the CNN network, $n = 16$ initial channels and embeddings of dimension 50 are used.

Layer	Input	Output
Embedding	[1, 64]	[1, 64, 50]
Initial Conv	[1, 64, 50]	[16, 64]
Conv-Block$_1$	[16, 64]	[32, 32]
Conv-Block$_2$	[32, 32]	[64, 16]
Conv-Block$_3$	[64, 16]	[128, 8]
Conv-Block$_4$	[128, 8]	[256, 4]
Pooling Layer	[256, 4]	[256, 1]
Fully Connected	[256]	[2]

Table 3.9 An example CNN pipeline.

ResNet

Following the designs of He et al. [41], our ResNet is constructed with 4 residual blocks, or res-blocks. A considerably deeper network is able to be built through the stacking of residual layers (res-layers) within each res-block. A res-layer is defined as two conv-layers with a shortcut connection between the inputs and outputs of each, depicted in Fig. 3.4 below. The shortcut connection combines the input channels with the output channels through elementwise addition. A varying number of layers can be present within each res-block, seen in Fig. 3.5. We define the number of layers within each respective block as the *block configuration*. For example, a block configuration of (1, 2, 3, 4) indicates the first block has 1 layer, the second has 2 layers and so forth. Different block configurations result in varying depths which are investigated in Section 3.2.2. The channel count remains constant throughout each res-block. Transition layers exist between each block to facilitate an increase in channel count, as well as downsampling the signal dimension. The default ResNet consists of a single res-layer within each block.

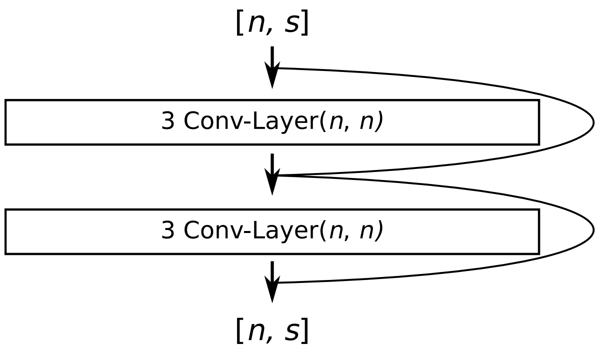


Figure 3.4 A residual layer.

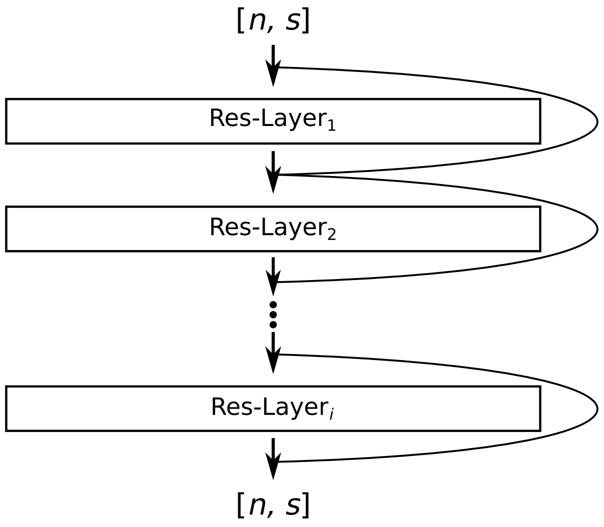


Figure 3.5 A general residual block.

The transition layer consists of a convolution layer with a kernel size $k = 3$ which outputs double the input channels and has a stride of 2 to halve the signal dimension. A skip connection is present over the transition layer, however, due to the aforementioned downsampling, the input requires resizing as it moves through the shortcut, represented by the dashed arrow in Fig. 3.6 below. This is facilitated by a $k = 1$ convolution with stride 2.

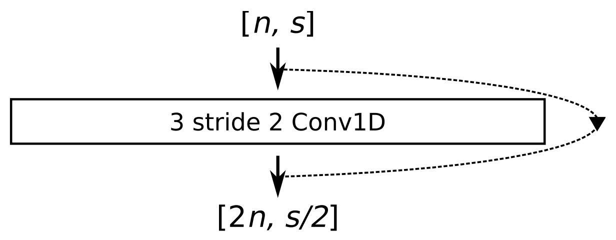


Figure 3.6 A residual transition layer.

Finally, a pooling layer follows the fourth res-block before passing the resultant tensor to a fully connected layer with an output of size 2, a softmax layer is again used for the final classification result.

In Table 3.10 below we see the dimension change as an example tensor moves through a ResNet with a block configuration of (2, 2, 2, 2) and 50-dimensional embeddings.

Layer	Input	Output
Embedding	[1, 64]	[1, 64, 50]
Initial Conv	[1, 64, 50]	[16, 64]
Res-Block$_1$		
Res-Layer × *2*	[16, 64]	[16, 64]
Transition$_1$	[16, 64]	[32, 32]
Res-Block$_2$		
Res-Layer × *2*	[32, 32]	[32, 32]
Transition$_2$	[32, 32]	[64, 16]
Res-Block 3$_3$		
Res-Layer × *2*	[64, 16]	[64, 16]
Transition$_3$	[64, 16]	[128, 8]
Res-Block$_4$		
Res-Layer × *2*	[128, 8]	[128, 8]
Pooling Layer	[128, 8]	[128, 1]
Fully Connected	[128]	[2]

Table 3.10 An example ResNet pipeline.

DenseNet

The proposed DenseNet is structurally similar to the above ResNet. It is composed of 4 dense-blocks constructed by stacking a varying number of dense-layers within. A dense-layer is defined as a single convolution layer with a modified skip connection. The key difference being that the input and output channels are concatenated together, as described in Section 2.2.8. Consequently, the channel count grows by a factor of GR after each dense-layer, defined as the *growth rate*. We note that within each dense-block the signal dimension remains unchanged. The default values used during experimentation are $n = 16$ initial features, $GR = 4$ and a block configuration of (2, 2, 2, 2).

A general dense-block is shown in Fig. 3.7 receiving an input tensor $[n, s]$ of n channels and signal size s. The initial dense-layer has n input channels and produces GR outputs. The input to a dense-layer is concatenated to all sequential dense-layers outputs within the block, represented by the curved black arrows, with the red arrow representing the concatenation of all prior dense-layer outputs. The final output tensor is $[n + (i \times GR), s]$, i.e. the number of output channels for a block, is the input channel count added to the output channels of each dense-layer.

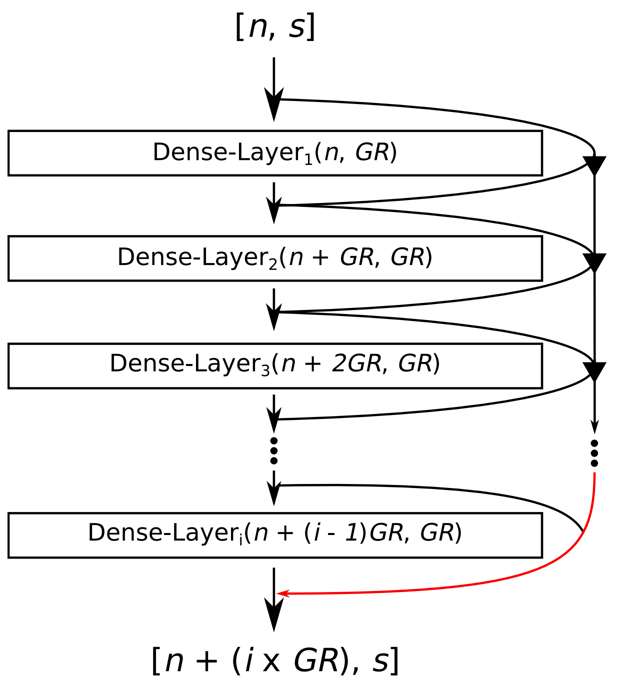

[n, s]
Dense-Layer$_1$(n, GR)
Dense-Layer$_2$(n + GR, GR)
Dense-Layer$_3$(n + 2GR, GR)
Dense-Layer$_i$(n + (i - 1)GR, GR)
[n + (i x GR), s]

Figure 3.7 A general dense block.

A transition layer, seen in Fig. 3.8, is implemented between each dense-block, responsible for halving the signal dimension and channel count. This downsampling is facilitated by a $k = 1$

conv-layer followed by a $k = 2$ average pooling layer. The convolution halves the input channels, preventing the network from growing excessively large with time, whilst the average pooling halves the signal dimension.

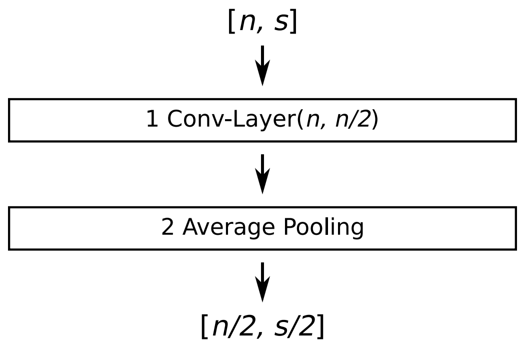


Figure 3.8 A dense transition layer.

The DenseNet ends identically to the above two networks: the final dense-block passes the resultant tensor to an average pooling layer before finally moving it through a linear layer with softmax determining the final classification.

Layer	Input	Output
Embedding	[1, 64]	[1, 64, 50]
Initial Conv	[1, 64, 50]	[16, 64]
Dense-Block$_1$		
Dense-Layer $\times$ *4*	[16 + $(i-1)\times$ 4, 64]	[4, 64]
Transition$_1$	[32, 64]	[16, 32]
Dense-Block$_2$		
Dense-Layer $\times$ *4*	[16 + $(i-1)\times$ 4, 32]	[4, 32]
Transition$_2$	[32, 32]	[16, 16]
Dense-Block$_3$		
Dense-Layer $\times$ *4*	[16 + $(i-1)\times$ 4, 16]	[4, 16]
Transition$_3$	[32, 16]	[16, 8]
Dense-Block$_4$		
Dense-Layer $\times$ *4*	[16 + $(i-1)\times$ 4, 8]	[4, 8]
Pooling Layer	[32, 8]	[32, 1]
Fully Connected	[32]	[2]

Table 3.11 Example DenseNet pipeline.

Table 3.11 details an example input as it moves through a DenseNet with the following configuration: n = 16 initial features, GR = 4 and a block configuration of (4, 4, 4, 4). The variable i refers to the i-th dense block in each dense layer.

3.2.2 Architecture Experimentation

Investigations into various structural changes of each archetype are carried out to determine the final network configurations. These experiments are listed below.

- Number of hidden fully connected layers.
- Activation function: ReLU vs Leaky ReLU.
- Pooling function: average, max, concatenation.
- Network depth.
- Growth rate (DenseNet).

The Headlines dataset is used throughout the above investigations with a constant learning rate of 0.003 and batch size of 64 to allow for fair comparison. The experiments are run 10 times each, with results presented as an average thereof. Ten epochs are used (unless otherwise stated), which is sufficient for the network to begin overfitting on most experiments - identified by an increase in testing loss and decrease in training loss. The models used are as described above, barring the changes resulting from the current experiment. That is, if 2 hidden layers were found to be optimal for a network, during the activation function experiment, the default of 0 hidden layers will be used. This is done to analyse the sensitivity of each network to the above changes in isolation. Finally, the best performing alterations are combined to form the final models for benchmarking.

Hidden Layers

Each base model consists of a single linear layer prior to the softmax output. Networks implementing convolutions exhibit sparse connections, that is, a neuron in a prior layer is only connected to a portion of neurons in the following layer. Additional linear layers at the head of the network allow the model to benefit from full connectivity as well as these sparse connections. This can be viewed as the linear layers receiving the most important features extracted by the pooling function and learning how to best categorise them.

The experiment is performed using a range of 0 - 5 hidden layers. When there is at least 1 hidden layer, the initial layer increases the input by a factor of 2^m, where m is the number of hidden layers, whilst sequential layers decrease it by a factor of 2. The final layer has an output size of 2, as previously discussed.

Activation Function

As described in Section 2.2.3 ReLU functions may result in dead neurons, however the addition of a small negative slope is not always beneficial. This investigation is carried out to determine which is better suited for the task of sarcasm classification. The investigated values for the negative slope are 0.01, 0.02 and 0.03.

Pooling Function

The pooling function is responsible for extracting the most important features generated by the convolutions. Max pooling is commonly used for image classification tasks, however simply taking the largest value could lead to an unnecessary loss of information. Due to this, we hypothesise that average pooling is better suited within the networks. We further investigate concatenating the output of max pooling with that of average pooling, defined as *concatenation pooling*.

Network Depth

A network may be deepened by increasing the number of layers present within its blocks. This experiment studies the effect of increasing depth in relation to sarcasm identification. Adding extra layers has been extensively researched for the CNN and results in the notorious vanishing gradient problem, as well as accuracy saturation. However, we still investigate an increase in depth for the CNN, solely for comparative purposes. Within the ResNet the channel count and signal size remain constant throughout each layer. Hence, additional layers grant the network further feature maps at the current level in the hierarchy. A similar understanding can be applied to the DenseNet, however, increasing the number of layers causes the overall channel output of the block to grow. A noteworthy concern of simply increasing depth is the network's tendency to overfit: a larger model is more prone to overfit on data in comparison to a structurally similar yet smaller model [116]. However, this issue can potentially be mitigated by applying a higher degree of regularisation to the network.

Four depths are initially explored for each architecture: 8, 16, 28 and 48 layers, these proposed depths follow the research of deep networks on various NLP domains by Schwenk et al. [46]. We do not follow their naming convention and only count the conv-layers present within each archetype's respective blocks when determining the layer count. This is done to keep the names of each archetypal variant consistent with one another. The investigated CNN, ResNet and DenseNet configurations are detailed in Table 3.12, Table 3.13 and Table 3.14 respectively. Noting each has $n = 16$ initial features and a default growth rate of $GR = 4$ is used for the dense variants. Finally, the matrix:

$$\begin{bmatrix} k, C_{out} \end{bmatrix} \times L.$$

seen in the tables below, describes the layer configuration present within the respective architectures blocks. That is, the layer has a filter size of k, outputs C_{out} channels and is stacked L times. Note that the CNN and ResNet variants will have two rows within the matrix, as detailed in their descriptions above.

	Output size	8-Layer	16-Layer	28-Layer	48-Layer
Initial Conv	s	$3 \times ED$ Conv, 16			
Conv-Block$_1$	$s/2$	$\begin{bmatrix} 3, 32 \\ 3, 32 \end{bmatrix} \times 1$	$\begin{bmatrix} 3, 32 \\ 3, 32 \end{bmatrix} \times 2$	$\begin{bmatrix} 3, 32 \\ 3, 32 \end{bmatrix} \times 2$	$\begin{bmatrix} 3, 32 \\ 3, 32 \end{bmatrix} \times 3$
Conv-Block$_2$	$s/4$	$\begin{bmatrix} 3, 64 \\ 3, 64 \end{bmatrix} \times 1$	$\begin{bmatrix} 3, 64 \\ 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3, 64 \\ 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3, 64 \\ 3, 64 \end{bmatrix} \times 5$
Conv-Block$_3$	$s/8$	$\begin{bmatrix} 3, 128 \\ 3, 128 \end{bmatrix} \times 1$	$\begin{bmatrix} 3, 128 \\ 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3, 128 \\ 3, 128 \end{bmatrix} \times 5$	$\begin{bmatrix} 3, 128 \\ 3, 128 \end{bmatrix} \times 8$
Conv-Block$_4$	$s/16$	$\begin{bmatrix} 3, 256 \\ 3, 256 \end{bmatrix} \times 1$	$\begin{bmatrix} 3, 256 \\ 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3, 256 \\ 3, 256 \end{bmatrix} \times 5$	$\begin{bmatrix} 3, 256 \\ 3, 256 \end{bmatrix} \times 8$

Table 3.12 CNN configurations for the depth investigation.

	Output size	8-Layer	16-Layer	28-Layer	48-Layer
Initial Conv	s	$3 \times ED$ Conv, 16			
Res-Block$_1$	s	$\begin{bmatrix}3, 16\\3, 16\end{bmatrix} \times 1$	$\begin{bmatrix}3, 16\\3, 16\end{bmatrix} \times 2$	$\begin{bmatrix}3, 16\\3, 16\end{bmatrix} \times 2$	$\begin{bmatrix}3, 16\\3, 16\end{bmatrix} \times 3$
Transition$_1$	$s/2$	3, stride 2 Conv1D			
Res-Block$_2$	$s/2$	$\begin{bmatrix}3, 32\\3, 32\end{bmatrix} \times 1$	$\begin{bmatrix}3, 32\\3, 32\end{bmatrix} \times 2$	$\begin{bmatrix}3, 32\\3, 32\end{bmatrix} \times 2$	$\begin{bmatrix}3, 32\\3, 32\end{bmatrix} \times 5$
Transition$_2$	$s/4$	3, stride 2 Conv1D			
Res-Block$_3$	$s/4$	$\begin{bmatrix}3, 64\\3, 64\end{bmatrix} \times 1$	$\begin{bmatrix}3, 64\\3, 64\end{bmatrix} \times 2$	$\begin{bmatrix}3, 64\\3, 64\end{bmatrix} \times 5$	$\begin{bmatrix}3, 64\\3, 64\end{bmatrix} \times 8$
Transition$_3$	$s/8$	3, stride 2 Conv1D			
Res-Block$_4$	$s/8$	$\begin{bmatrix}3, 128\\3, 128\end{bmatrix} \times 1$	$\begin{bmatrix}3, 128\\3, 128\end{bmatrix} \times 2$	$\begin{bmatrix}3, 128\\3, 128\end{bmatrix} \times 5$	$\begin{bmatrix}3, 128\\3, 128\end{bmatrix} \times 8$

Table 3.13 ResNet configurations for the depth investigation.

	Output size	8-Layer*	16-Layer	28-Layer	48-Layer
Initial Conv	s	$3 \times ED$ Conv, 16			
Dense-Block$_1$	s	$[3, 4] \times 2$	$[3, 4] \times 4$	$[3, 4] \times 4$	$[3, 4] \times 6$
Transition$_1$	$s/2$	2 Average Pooling			
Dense-Block$_2$	$s/2$	$[3, 4] \times 2$	$[3, 4] \times 4$	$[3, 4] \times 4$	$[3, 4] \times 10$
Transition$_2$	$s/4$	2 Average Pooling			
Dense-Block$_3$	$s/4$	$[3, 4] \times 2$	$[3, 4] \times 4$	$[3, 4] \times 10$	$[3, 4] \times 16$
Transition$_3$	$s/8$	2 Average Pooling			
Dense-Block$_4$	$s/8$	$[3, 4] \times 2$	$[3, 4] \times 4$	$[3, 4] \times 10$	$[3, 4] \times 16$

Table 3.14 DenseNet configurations for the depth investigation.

*Due to the transition layer halving the channel count, an 8-layer DenseNet with the default configuration will become narrower as it gets deeper. We however ignore this and obtain the results for comparative purposes.

Growth Rate

Each dense-layer is responsible for producing *GR* channels within its dense-block. Whilst a higher growth rate potentially gives the model more computational power, the network size grows considerably. Increasing the growth rate can be viewed as widening the model, as opposed to deepening it. Five 8-layer DenseNets are implemented, with growth rates of 2, 4, 8, 16 and 32 investigated. Each model begins with $n = 16$ initial channels, and since each variant has identical block configuration, the models only vary in width. The networks with larger growth rates output additional feature maps at each level of the hierarchy, effectively providing the network with extra information to model. Table 3.15 details the model variants used to investigate the effect of width.

	Output size	8-Layer$_2$	8-Layer$_4$	8-Layer$_8$	8-Layer$_{16}$	8-Layer$_{32}$
Initial Conv	s	$3 \times ED$ Conv, 16				
Dense-Block$_1$	s	$[3, 2] \times 2$	$[3, 4] \times 2$	$[3, 8] \times 2$	$[3, 16] \times 2$	$[3, 32] \times 2$
Transition$_1$	$s/2$	2 Average Pooling				
Dense-Block$_2$	$s/2$	$[3, 2] \times 2$	$[3, 4] \times 2$	$[3, 8] \times 2$	$[3, 16] \times 2$	$[3, 32] \times 2$
Transition$_2$	$s/4$	2 Average Pooling				
Dense-Block$_3$	$s/4$	$[3, 2] \times 2$	$[3, 4] \times 2$	$[3, 8] \times 2$	$[3, 16] \times 2$	$[3, 32] \times 2$
Transition$_3$	$s/8$	2 Average Pooling				
Dense-Block$_4$	$s/8$	$[3, 2] \times 2$	$[3, 4] \times 2$	$[3, 8] \times 2$	$[3, 16] \times 2$	$[3, 32] \times 2$

Table 3.15 DenseNet configurations for the growth rate investigation.

3.3 Proposed Models

Three final models are implemented based on the results found from the investigations detailed in Section 3.2.2. The parameters yielding the best results for each experiment - as measured by testing accuracy - are combined to form the final configuration for each archetype. It is noted that this may not be the ideal method of combining the various parameters as each experiment was performed on identical base models. This ignores any potential interaction between each structural change and could result in sub-optimal configurations. Better combinations based on the results may be obtained using optimisation techniques, however this is beyond the scope of this work. Using these combinations, a final CNN and ResNet are built as baselines with which to compare the proposed final DenseNet (named 'DweNet') against. The configurations and diagrams of each network are presented below in the following order: CNN, ResNet, DweNet.

3.3.1 Baselines

Baseline CNN

An 8-layer CNN with a block configuration of (1, 1, 1, 1) is implemented as the baseline. Average pooling is used following the conv-blocks to capture important features. Three hidden layers are present at the head of the network, with Leaky ReLUs inbetween. Leaky ReLUs are further used within the conv-layers, all having a negative gradient of 0.02. A detailed overview of the proposed baseline is depicted in Fig. 3.9 below.

The input to the embedding consists of a single channel of s tokens ($s = 64$ for Headlines and $s = 128$ for Reddit Main/Pol). The embedding layer generates ED-dimensional embeddings for each token. The initial convolution then convolves over the 3-gram embeddings to produce 16 channels. The conv-blocks follow and are labelled as such: 'Conv-Block$_n$, O', where n is the n^{th} conv-block outputting O channels. Each conv-block is responsible for doubling the channel count whilst halving the signal dimension. Hence, at the n^{th} conv-block, the resultant output will have 16×2^n channels and signal size $\frac{s}{2^n}$. The pooling layer averages the signal dimension through a kernel of size $s/16$, to produce $m = 256$ features. This is passed to the fully connected layers, represented by 'FC(I, O)', where I and O are the input and output sizes respectively.

As mentioned in the Network Depth outline, the CNN was deepened solely for comparative purposes - since a large degree of research already exists surrounding this topic. Due to this, we implement an 8-layer variant as the baseline, despite the 16-layer yielding better performance.

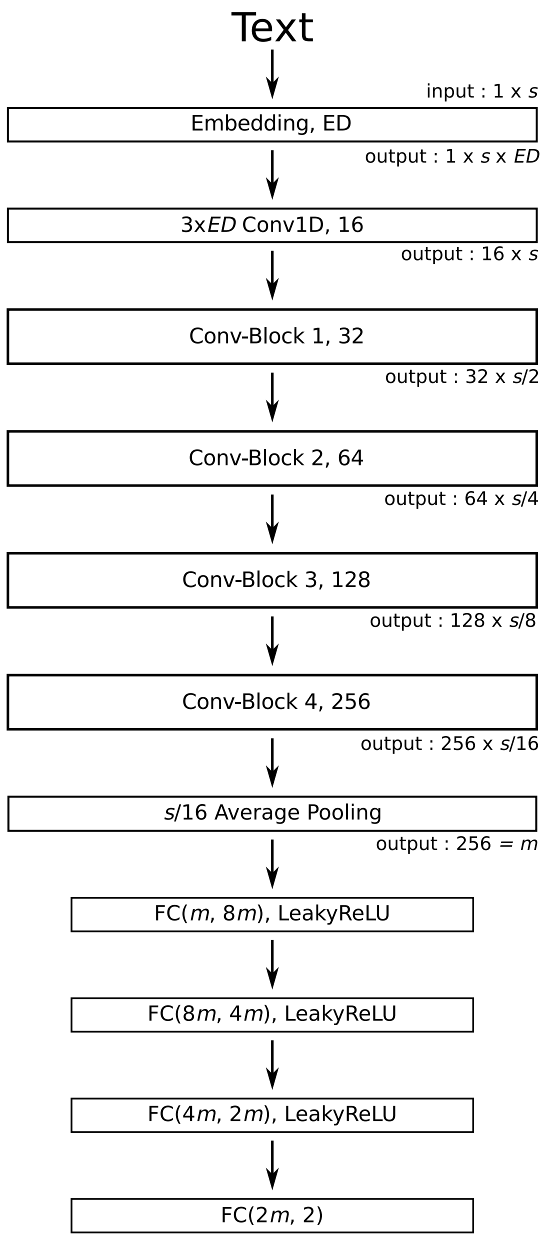


Figure 3.9 Overview of the baseline CNN.

Baseline ResNet

A 28-layer residual network with a block configuration of (2, 2, 5, 5) is implemented as the baseline. Similarly to the CNN, average pooling follows the res-blocks to facilitate feature extraction. A single hidden layer is present at the head of the model, with a standard ReLU activation following. ReLU activations are futher used within each res-layer. The proposed structure of this baseline is depicted in Fig. 3.10 below.

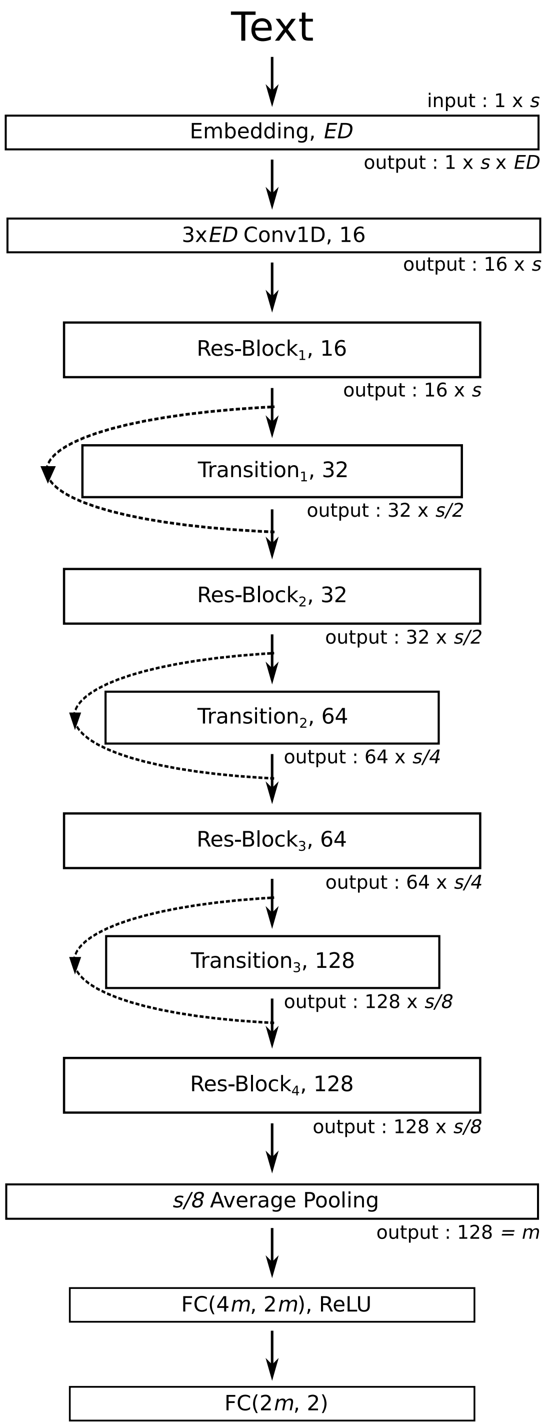


Figure 3.10 Overview of the baseline ResNet.

The intial two layers behave as discussed above within the CNN. Following the initial convolution, the resulting 16 output channels are passed as the input to the first res-block. Noting that every convolution within the blocks have a kernel of size 3, as previously mentioned. A transition layer follows each res-block halving the input signal and doubling the channel count. Thus, at the i^{th} res-block, the resulting output tensor will have size $[16 \times 2^{i-1}, s/2^{i-1}]$. The naming of each layer is as above: 'Res-Block$_i$, O' and 'Transition$_i$, O' refers to the i^{th} res-block or transition layer, which outputs O channels. Finally, the dotted arrows indicate the skip connection acting over the transition layers.

3.3.2 DweNet

The proposed dense model consists of 58 layers resulting from a block configuration of (6, 12, 24, 16), together with a growth rate of 32. Whilst this block configuration was not investigated formally in the depth experiment, it yielded the top results upon further analysis of combining depth and width. Two hidden layers are implemented at the head of the network, with Leaky ReLUs inbetween - each with a negative gradient of 0.02. Identical Leaky ReLUs are further used as the activations within each dense-layer. A combination of average and max pooling (concatenation pooling) is used for feature extraction, concatenating the outputs of each for input to the hidden layers. Given the depth of the model, additional regularisation is introduced through dropout at a rate of $p = 0.2$ applied over each dense-block output. The model arising from this composition is named *DweNet* and is used as the primary model throughout the remainder of this work. The complete structure of DweNet is presented in Fig. 3.11 below.

Again the initial two layers behave similarly as described above, the only difference being $n = 64$ channels are output by the first convolution. This initial channel count is chosen to keep the output of all dense-blocks and transition layers as a power of 2, a design rule adopted from the original DenseNet paper. Given that each dense-layer produces $GR = 32$ channels towards the final output, a dense-block receiving n input channels, will produce $n + 32 \times L$ channels, where L is the number of layers within the block. A transition layer halves the channel count and signal dimension, hence, an input tensor, $[n, s]$, will yield an output tensor, $[n/2, s/2]$. In the diagram, 'Transition$_i$, $O/2$' indicates the i^{th} transition layer, outputting $O/2$ channels.

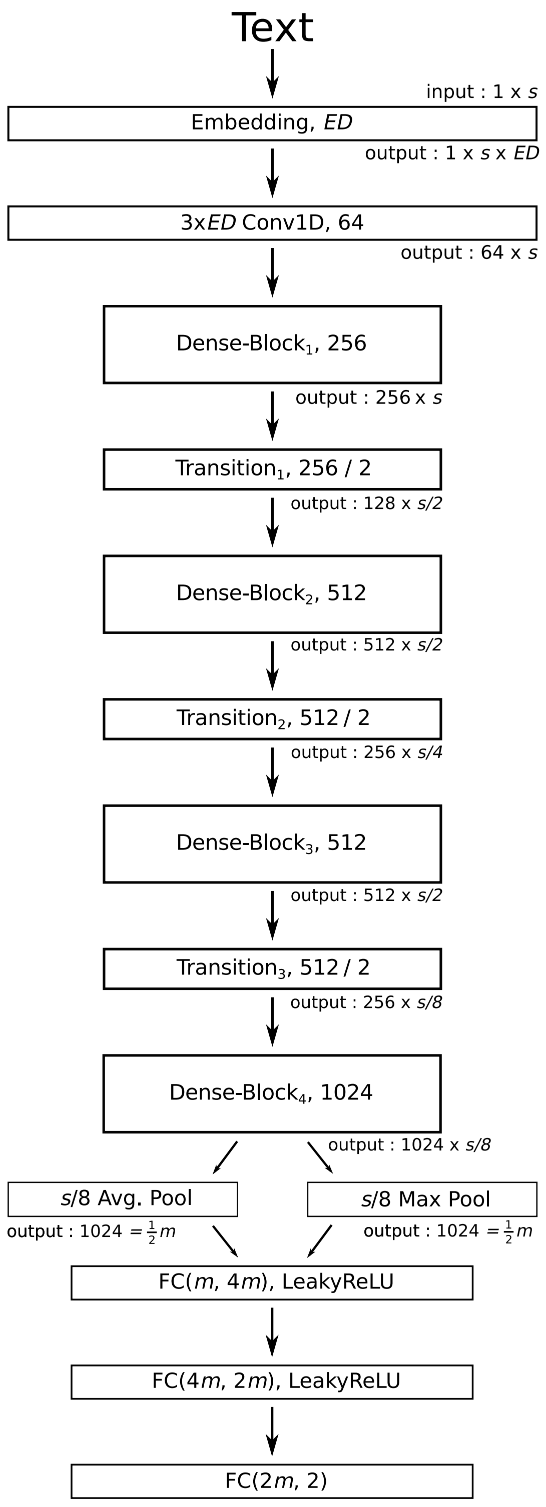


Figure 3.11 Overview of our proposed model DweNet.

3.4 Model Benchmarking and Further Investigation

The conditions used to benchmark our models on the datasets outlined in Section 3.1 are initially described. Then, further exploratory investigations into embeddings, local context and our data preprocessing techniques are introduced.

3.4.1 Benchmarking

The three models proposed in Section 3.3 are evaluated on the Headlines, Reddit Main and Reddit Pol datasets to compare their efficacy in sarcasm classification. Prior results on each dataset are obtained from previous related works to compare the performance of our proposed network, DweNet, against state-of-the-art approaches. Accuracy, as well as F1-score, are used as metrics to evaluate the performance of each model. A learning rate of 0.003, together with the one cycle learning policy introduced by Smith [117] is adopted. Weight decay at a rate of 0.1 applied to all models for regularisation. Weights are initialised using the method outlined by He et al. [118] and are updated through minimisation of the log-loss with the Adam optimiser [59] having momentum range between 0.8 and 0.7. The embedding layer is initialised using 50-dimensional non-static GloVe embeddings, that is, they are updated as training progresses. Words which are not present in the GloVe vocabulary are assigned random representations within the embedding space. This is done by sampling from a normal distribution with a standard deviation of 0.6. A batch size of 64 is used, with each network training for 10 epochs, which is sufficient for overfitting to occur. Further, after each epoch, the networks score on the testing set is recorded, with results presented as an average of the top testing accuracy across 10 runs. The best performing network will then be used for further exploratory investigations, detailed in Section 3.4.2 below.

Based on the results of the preliminary architectural experimentation, we hypothesise DweNet will yield the top results on all datasets compared to our CNN and ResNet baselines.

3.4.2 Exploratory Investigations

DweNet was found to be the best performing model based on the benchmarking results detailed in Section 4.2. Hence, it is used to perform exploratory studies into the following concepts in relation to sarcasm classification:

- Embeddings.
- The effect of local context.

- The efficacy of data pre-processing.

These investigations are performed to identify any weaknesses within the model, or whether improvements may be made. Further, we wish to evaluate whether the potential loss of data resulting from our preprocessing technique is detrimental to the performance of DweNet.

Embeddings

GloVe embeddings of dimension 50 are used as the default weight matrix initialisation given their promising performance in past works. However, in more recent approaches, Facebook's FastText embeddings have shown competitive performance across numerous NLP tasks. This investigation compares the efficacy of the GloVe representations compared to 3 FastText variations, details of which are listed below:

- GloVe_{6B}: 400 thousand word vectors of dimension 50 trained on Wikipedia 2014 and Gigaword 5 (6 billion tokens).
- $\text{FastText}_{\text{1M}}$: 1 million word vectors of dimension 300 trained on Wikipedia 2017, UMBC webbase corpus and statmt.org news dataset (16 billion tokens).
- $\text{FastText}_{\text{1M}-subword}$: 1 million word vectors of dimension 300 trained on Wikipedia 2017, UMBC webbase corpus and statmt.org news dataset (16 billion tokens), incorporating subword information.
- $\text{FastText}_{\text{2M}}$: 2 million word vectors of dimension 300 trained on Common Crawl (600 billion tokens).

Further, given sarcasm's nuanced nature, traditional word embeddings may not be best suited as contextually similar words are mapped to the same neighbourhood within the embedding space. This results in the representations for words such as 'happy' and 'sad' being similar. It is clear, however, that semantically, these words are polar opposites. This presents a potential problem for sarcasm classification due to the strong dependency on what is implied, not what is said. Furthermore, the embeddings are trained on common English datasets, which will rarely contain sarcastic phrases, consequently resulting in vectors well suited for typical English usage, but not necessarily sarcasm. Hence, to better align the embeddings with the semantics of sarcasm, non-static embeddings are used by default. That is, word vectors which are updated through backpropagation as the model trains. In order to demonstrate the suboptimality of standard embeddings, the testing accuracy, recall, specificity and precision yielded on the Headlines dataset of a static and non-static model are compared for each of the above embeddings.

The effect of local context

The previous work herein has focused on modelling the isolated utterance, ignoring any metadata or contextual information provided by the datasets. As previously discussed, comments on social media, in particular Reddit, are often made in response to another post. We refer to this other post as the local context for the original comment. This investigation analyses the effect of local context, provided in the form of parent comments from the Reddit Main dataset. The parent comment is concatenated to the child (original) comment, separated by the 'field' token described in Table 3.7. We investigate concatenating the parent comment after the child comment, as well as before. Further, since each child comment has a corresponding parent comment, the new dataset is double in size. This consequently introduces additional words into the vocabulary, effectively providing the network with extra information to model the utterance with.

The effect of data preprocessing

To evaluate the effectiveness of the preprocessing techniques detailed in Section 3.1.1, the results of DweNet receiving the default preprocessed data and data which simply undergoes basic tokenisation and numericalisation are analysed on the Headlines dataset. In the context of this experiment, basic tokenisation refers to simply splitting the data over white space and punctuation, without making use of the special tokens and rules provided by the FastAI framework, listed in Table 3.7 and Table 3.8. We note that the padding token 'xxpad' is retained due to its requirement for batched convolutions. The 'xxunk' token - used to replace tokens occurring less than 3 times within the corpus - may result in a large loss of information, as many words appear only once or twice within the dataset. To determine whether the usage of this 'unknown' token negatively impacts the performance of the network, a word occurrence threshold of 1 and 2 is investigated. A value of 1 would indicate every token encountered is placed into the vocabulary.

Chapter 4

Discussion and Results

Within this chapter, the results yielded from the architectural investigations are presented and discussed. Model benchmarking is performed on the three proposed networks and an analysis of results compared to prior approaches is conferred. Finally, the exploratory studies are conducted on our top performing model, DweNet, with results and observations examined.

4.1 Architecture Evaluation

The Headlines dataset is used to compare the results of the various structural changes detailed in Section 3.2.2. The findings are presented in the following order: hidden layers, activation function, pooling layer, network depth and finally growth rate. Each investigation ends with a tabled summary detailing the greatest testing accuracy obtained by each alteration. A configuration summary precedes the depth experiment, containing a brief summary of the first three experiments, as well as a table presenting the final combinations for each archetype. These changes are then implemented within the models used for the depth experiment.

4.1.1 Hidden Layers

Each architecture was implemented with zero to five hidden layers and trained on the Headlines dataset. Testing accuracies obtained after each epoch were recorded, the results of which are depicted in Fig. 4.1, Fig. 4.2 and Fig. 4.3 below.

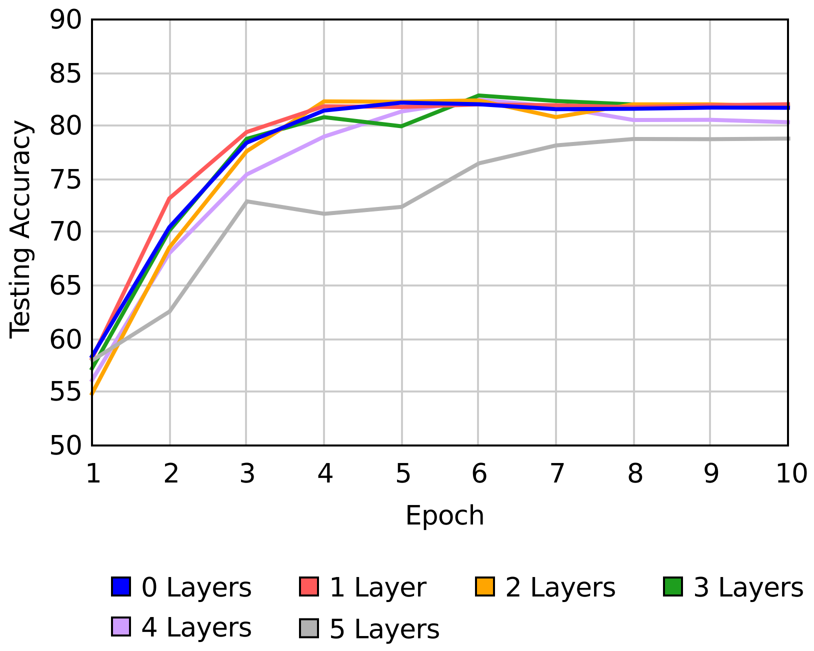


Figure 4.1 CNN testing accuracy vs the number of hidden layers.

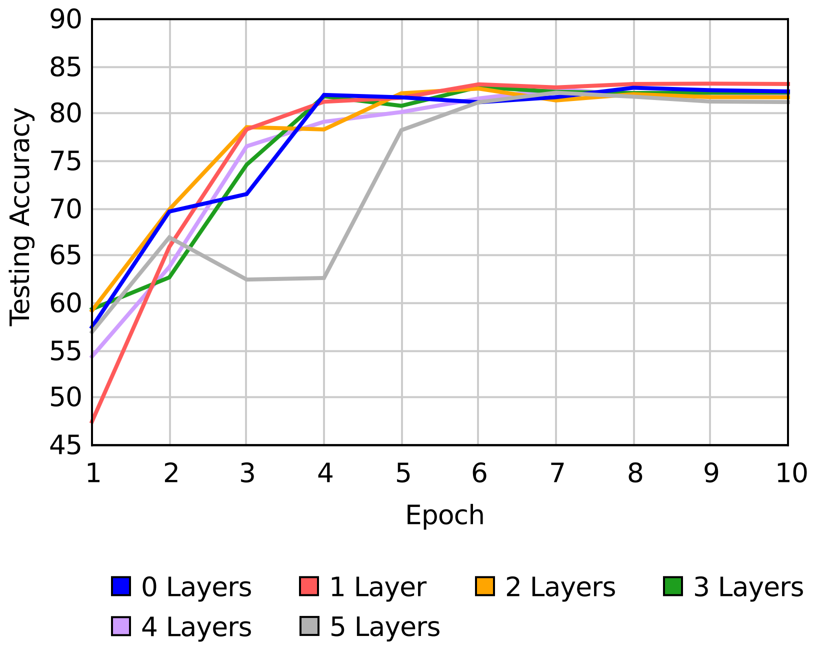


Figure 4.2 ResNet testing accuracy vs the number of hidden layers.

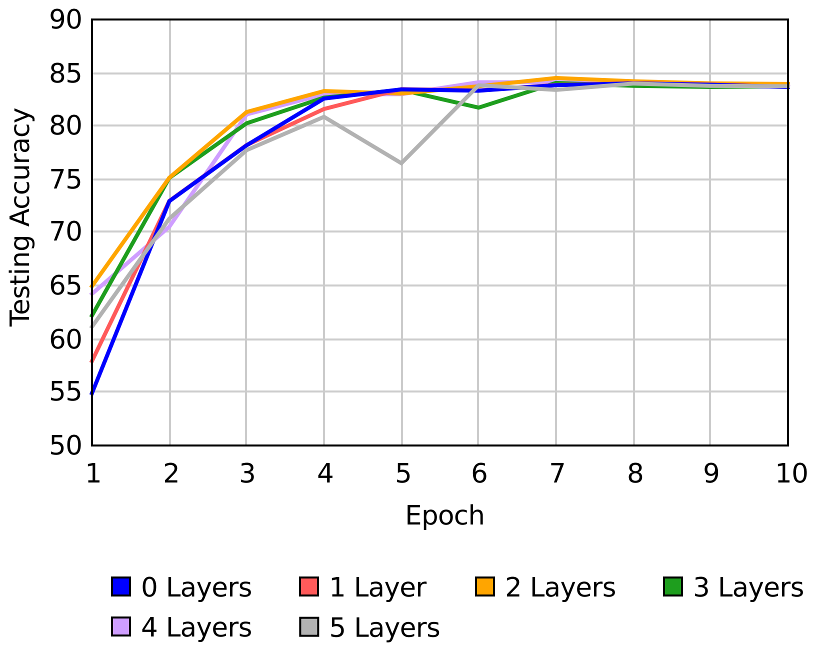


Figure 4.3 DenseNet testing accuracy vs the number of hidden layers.

For each archetype, a general increase in testing accuracy is observed when increasing the number of hidden layers. The increase however, was not linear, with 1 to 3 layers yielding the top results. The training and testing losses after 10 epochs saw marginal changes, suggesting additional hidden layers are beneficial and the models do not exhibit sensitivity to the change - excluding the CNN which experienced a significant drop in performance when implemented with 5 hidden layers.

The top result obtained by each model variant is detailed in Table 4.1, with bold entries indicating the layer count yielding the highest testing accuracy per architecture. Thus we implement the final CNN, ResNet and DenseNet models with 3, 1 and 2 hidden layers respectively.

	Accuracy					
	0 Layers	*1 Layer*	*2 Layers*	*3 Layers*	*4 Layers*	*5 Layers*
CNN	82.19	82.02	82.40	**82.87**	82.47	78.96
ResNet	82.50	**83.19**	82.66	82.83	82.60	82.50
DenseNet	83.99	84.01	**84.51**	84.04	84.11	84.07

Table 4.1 Summary of hidden layer investigation.

4.1.2 Activation Function

Each architecture was implemented with ReLU activations, as well as Leaky ReLU activations with negative gradients ranging from 0.01 to 0.03, and trained on the Headlines dataset. Test accuracies of each variant were recorded for 10 epochs, the results of which are plotted in Fig. 4.4, Fig. 4.5 and Fig. 4.6 below.

The CNN and DenseNet models benefited from a negative gradient within the ReLU activation, the ResNet however, experienced a decrease in accuracy resulting from the leaky activations. Further, each architecture saw a decrease in training and testing loss when using the leaky activations, suggesting the leaky variant is less prone to overfitting compared to a standard ReLU.

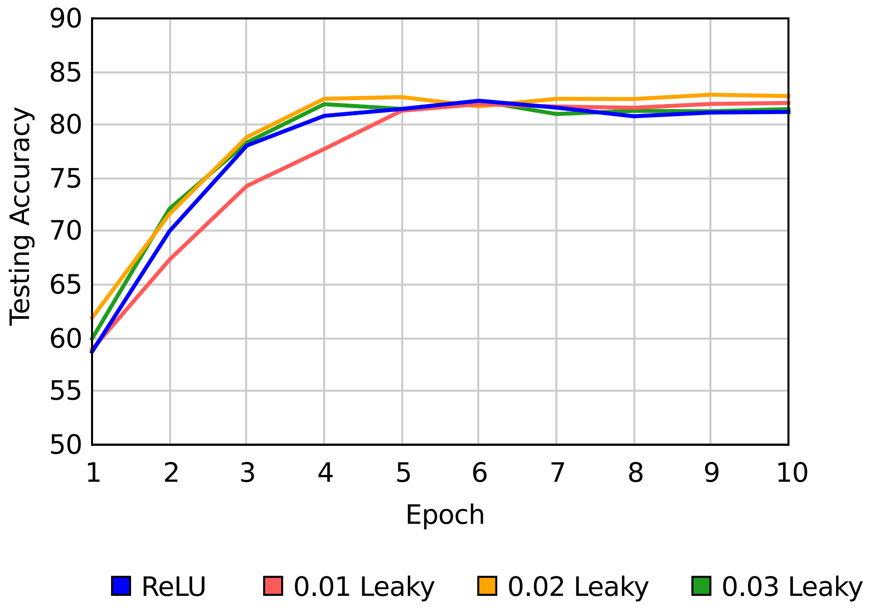


Figure 4.4 CNN testing accuracy vs different activation functions.

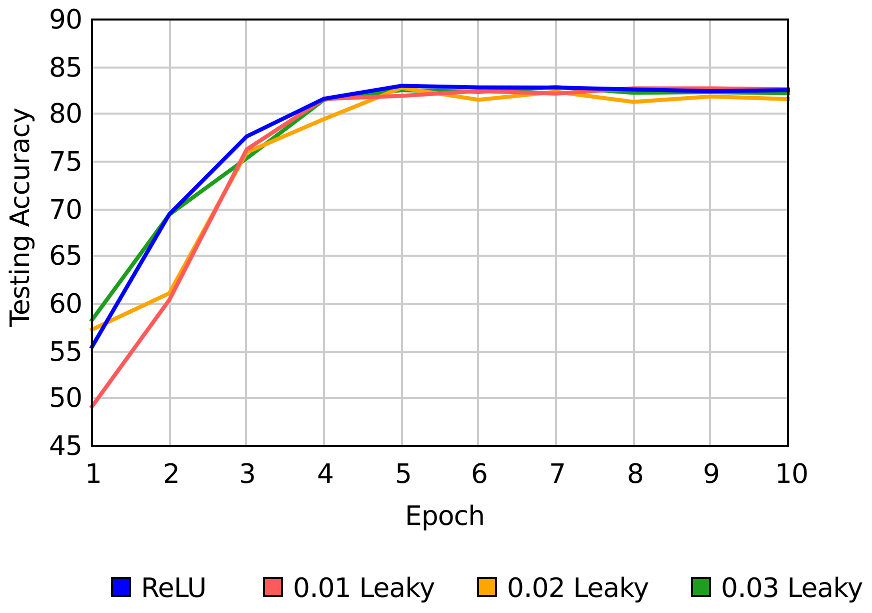


Figure 4.5 ResNet testing accuracy vs different activation functions.

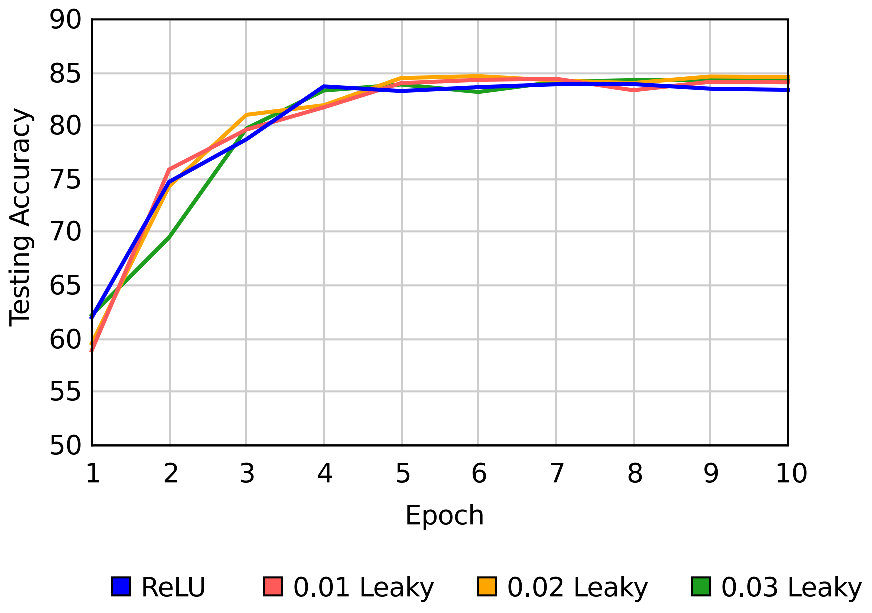


Figure 4.6 DenseNet testing accuracy vs different activation functions.

The highest testing accuracies obtained for each network are detailed in Table 4.2, with bold entries indicating the activation resulting in the top performance per architecture. Both the CNN and DenseNet obtained their top results with a negative slope of 0.02, hence the final models will be implemented with this value, whilst the final ResNet will use standard ReLU activations. Again it is noted that each architecture was relatively insensitive to the investigated changes.

	Accuracy			
	ReLU	*0.01 Leaky*	*0.02 Leaky*	*0.03 Leaky*
CNN	82.27	82.07	**82.84**	82.25
ResNet	**83.00**	82.74	82.79	82.87
DenseNet	83.93	84.43	**84.68**	84.34

Table 4.2 Summary of activation function investigation.

4.1.3 Pooling Function

Each architecture was implemented with average pooling, max pooling and concatenation pooling, to evaluate which method is best suited for feature extraction at the head of the network. Testing accuracies yielded across 10 epochs on the Headlines dataset were recorded and plotted in Fig 4.7, Fig 4.8 and Fig 4.9 below.

The architectures all exhibited low sensitivity to the change of pooling function, with testing accuracies being very similar across all three functions, seen in the charts below. Max pooling performed the poorest across all three architectures, suggesting the loss of information from simply taking the max values is effecting classification. An interesting observation is concatenation pooling resulted in decreased testing accuracies on both the CNN and ResNet, compared to the average pooling function. It was expected that concatenation pooling would lead to an increase in performance, this however, was only the case within the DenseNet.

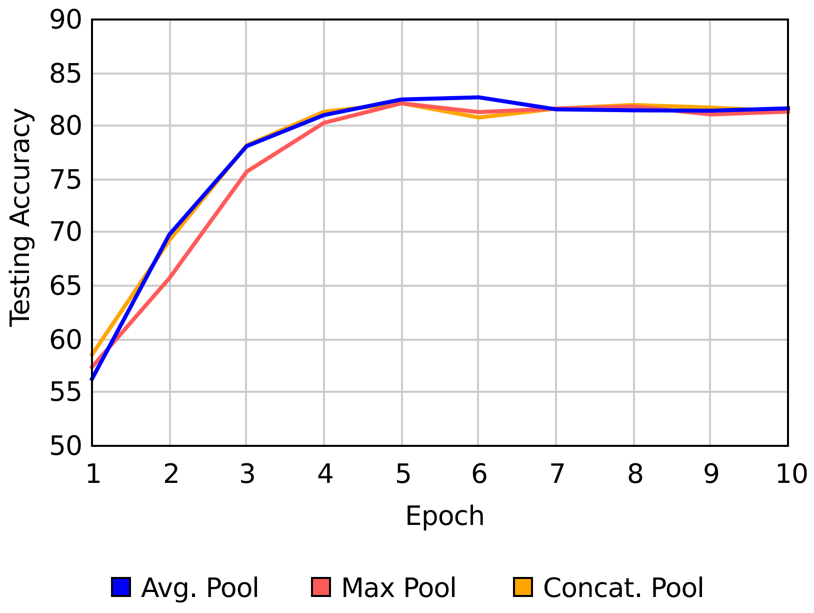


Figure 4.7 CNN testing accuracy vs pooling function.

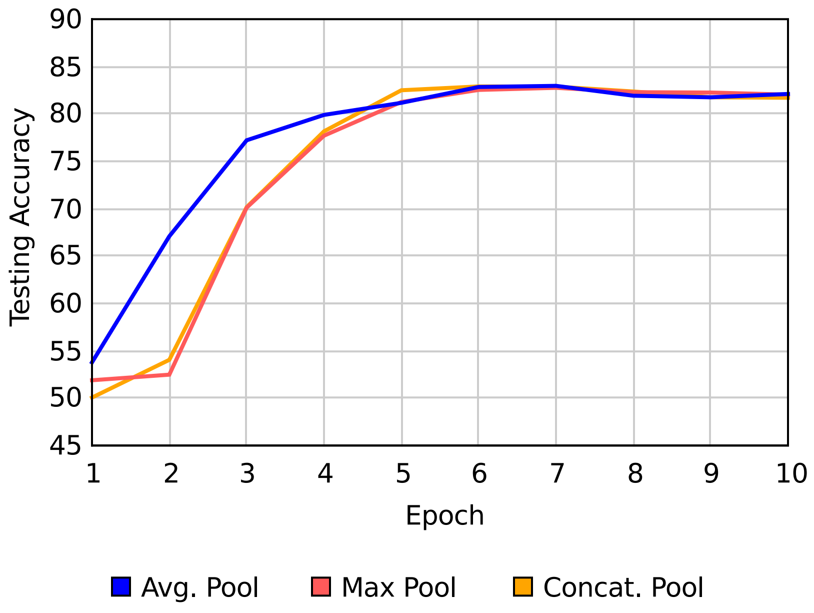


Figure 4.8 ResNet testing accuracy vs pooling function.

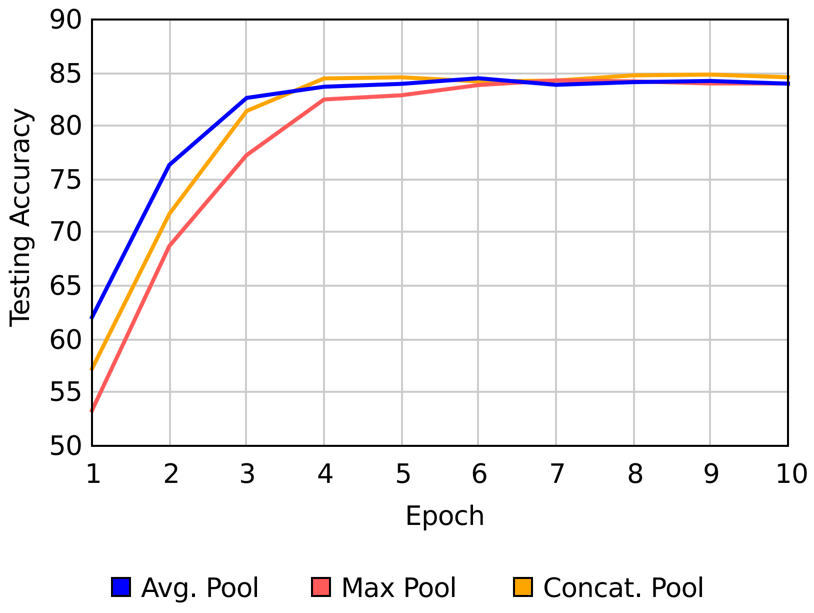


Figure 4.9 DenseNet testing accuracy vs pooling function.

The top testing accuracies obtained by each network variant are recorded in Table 4.3 below, with bold entries indicating the function leading to the best accuracy across each archetype. The CNN and ResNet performed best using average pooling, whilst the DenseNet benefited from concatenation pooling. Hence, we implement the final models accordingly.

	Accuracy		
	Avg. Pooling	*Max Pooling*	*Concat. Pooling*
CNN	**82.70**	82.14	82.15
ResNet	**82.96**	82.75	82.90
DenseNet	84.49	84.29	**84.83**

Table 4.3 Summary of pooling function investigation.

4.1.4 Configuration Summary

The architectures were insensitive to the structural changes resulting from the above experiments, with fragility only noted within the CNN during a 5 hidden layer implementation. The structural variations yielding the best results, as measured by testing accuracy, are combined to implement the final models. The final configurations used are seen in Table. 4.4, these changes are implemented within each archetype prior to the depth investigation below.

	Hidden Layers	Act. Function	Pooling
CNN	3	0.02 Leaky	Average
ResNet	1	ReLU	Average
DenseNet	2	0.02 Leaky	Concatenation

Table 4.4 Best performing model configurations.

4.1.5 Network Depth

Each proposed architecture is implemented with the configurations detailed in Table 4.4 prior to this investigation. The results obtained on the Headlines dataset for varying depths are presented and analysed in the following order: CNN, ResNet, DenseNet. Further experimentation is performed on extremely deep networks based on the observation that depth led to an increase in performance. Finally, a summary of the experiment is presented, providing an analysis into the efficiency of parameter usage within each model.

CNN

The Headlines dataset was run 10 times on each depth CNN configured with identical hyperparameters and embeddings, with results presented as an average thereof. The top testing accuracy, lowest testing and training loss for each depth, are detailed in Table 4.5, with the best result recorded in bold.

	Testing Acc.	Testing Loss	Training Loss
8-Layer	83.88	39.16	1.40
16-Layer	**84.14**	37.91	4.66
28-Layer	83.90	38.11	16.80
48-Layer	59.73	68.78	66.00
48-Layer$_{50}$*	84.13	53.63	0.77

Table 4.5 CNN results: highest testing accuracy and lowest testing/training loss, by depth across 10 epochs.
*Results over 50 epochs.

Doubling the layer count from 8 to 16 resulted in a marginal increase of 0.26% from 83.88% to 84.14%. A further addition of layers from 16 to 28 yielded a score of 83.90%, a decrease of 0.24% vs the 16-layer but still a fraction above the 8 layer by 0.02%. This suggests an increasing depth is of little benefit to CNN models, which is expected based on the results of prior research [41]. A comparison of the testing accuracy over 10 epochs for each depth is seen in Fig. 4.10.

An interesting note is during the first few epochs the deeper models exhibit considerably lower accuracy, indicating that whilst this initial increase in depth results in a gain in accuracy, the deeper models require additional epochs before converging. This is most noticeable within the 48-layer CNN, which exhibited a score of 59.73% after 10 epochs. However, it still had a very high training loss suggesting it should be trained for additional epochs. Given this observation, the 48-layer CNN was rerun for 50 epochs to determine whether it would converge to a similar accuracy as the shallower variants, or whether complete degradation had indeed occured.

Figure 4.11 shows the testing accuracy of the 48-layer CNN required 32 epochs to break the 80% mark, with its highest result of 84.13% obtained at the 49th epoch. This may suggest that even further epochs would result in better performance, however the training loss of the model at the last epoch is 0.77, indicating it has already overfit considerably. The 48-layer outperformed both the 8 and 28-layer variants, and obtained a near identical result as the 16-layer. This however, does not indicate depth has benefited the model due to the pronounced increase of 15.72 in the lowest obtained testing loss, compared to the 16-layer. This indicates that although the model obtains a similar classification accuracy, it is not as confident as the shallower network. Further, the observable fluctuations of ~8% in testing accuracy occuring between the 30th and 50th epoch suggest a degree of fragility resulting from this depth.

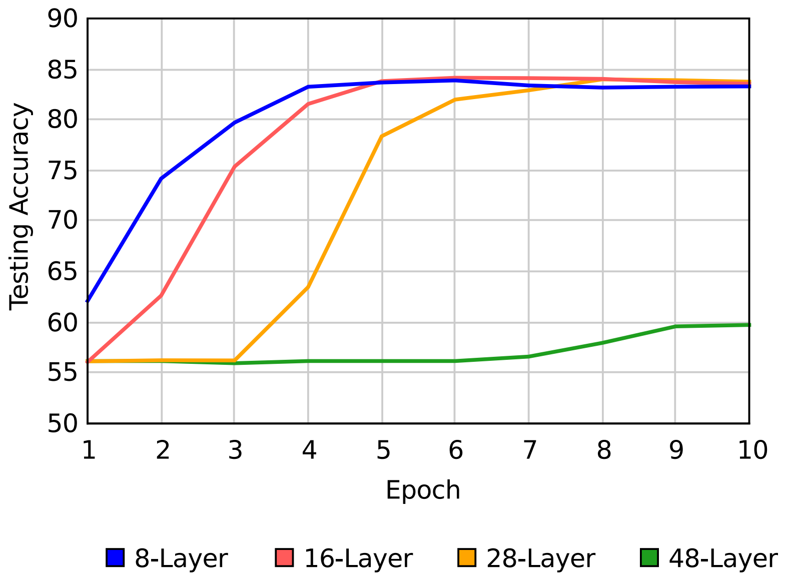


Figure 4.10 CNN testing accuracy vs epoch, by layer count.

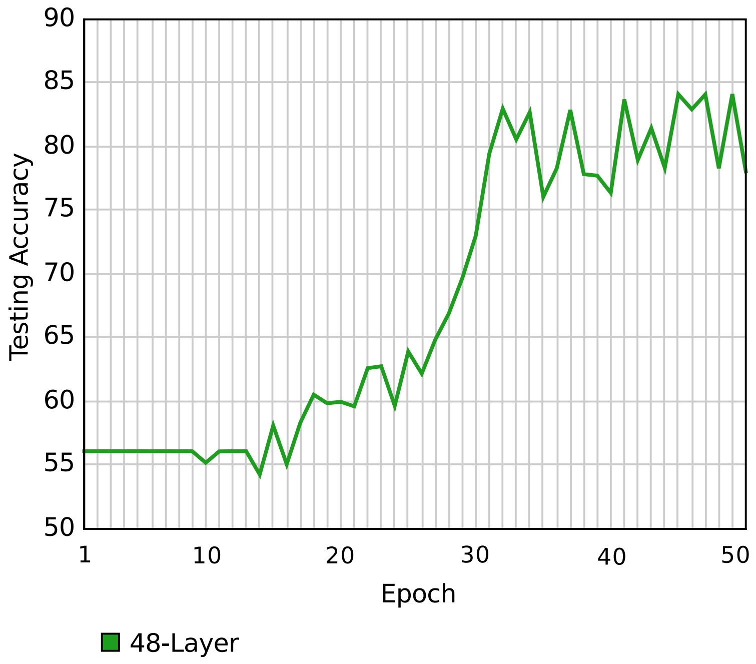


Figure 4.11 48-layer CNN testing accuracy vs Epoch, across 50 Epochs.

Whilst deepening the network from 8 layers results in a minuscule performance boost, it is noted that each depth exhibits similar confidence at its best epoch. The testing losses, plotted below in Fig. 4.12, show that the 8, 16 and 28-layer CNNs have near identical testing loss at their best epochs (4, 6, 7 respectively). Furthermore, the shallower networks (excluding the 48-layer) begin to overfit at a faster rate compared to a deeper counterpart. This is reinforced upon comparison of the training loss for each depth: the 8-layer yielding an incredibly low loss of 1.4, as opposed to the 28-layer with a more reasonable loss of 16.8. However, these deeper models simply need additional epochs to exhibit similar training losses, shown within the 48-layer which obtained a loss of 0.77 after 50 epochs.

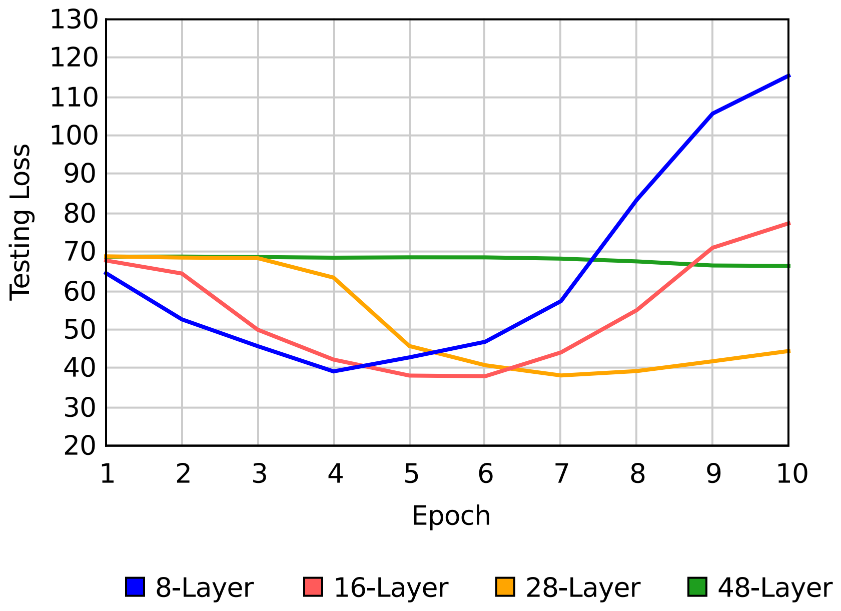


Figure 4.12 CNN testing loss vs epoch, by layer count.

ResNet

Each depth ResNet was configured and trained identically, with results presented as the average across 10 runs on the Headlines dataset. Table 4.6 details the top testing accuracy and lowest testing and training loss obtained by each depth, with the highest result recorded in bold.

	Testing Acc.	Testing Loss	Training Loss
8-Layer	84.21	37.69	2.59
16-Layer	84.90	37.14	8.11
28-Layer	**85.40**	35.86	20.78
48-Layer	80.80	47.91	29.27
48-Layer$_{20}$*	84.80	42.1	6.54

Table 4.6 ResNet results: highest validation accuracy and lowest validation/training loss, by depth across 10 epochs.
*Results over 20 epochs.

Residual learning proved to be beneficial in reinforcing the deeper models, with the 8, 16 and 28-layer variants yielding testing accuracies of 84.21%, 84.90% and 85.40% respectively. This corresponds to an increase of 0.69% and 0.5% as depth increased. The 48-layer network no longer suffers severe degradation within the initial 10 epochs, however it does not provide an increase in accuracy over the shallower models. Consequently exhibiting a 4.6% decrease in accuracy when compared to the 28-layer variant across the initial 10 epochs. This however, decreases to a difference of 0.6% after 20 epochs. It is noted that the deeper networks began with a ~5% lower testing accuracy, also observed within the CNN variants above. This may be attributed to the deeper networks learning additional features during the initial epoch, some of which may be noise. Since the model has not been sufficiently optimised for these features, it is possible that an increased use of this noise is then used to classify the text. Once the deeper models have been further trained, the extracted features are better weighted to fit the task of sarcasm classification. A comparison of the obtained results is seen in Fig. 4.13 below.

Similarily to the 48-layer CNN, the 48-layer ResNet had a higher training loss compared to the shallower variants. Hence, we rerun the experiment for 20 epochs to investigate whether it would converge to a similar accuracy as its shallower counterparts. As seen in Fig. 4.14 below, the 48-layer ResNet slowly gained in accuracy across the last 10 epochs, yielding its best result of 84.80% at epoch 16. The residual networks are suitable for deep depths, but require additional epochs to become accurate. Whilst this is beneficial for improving accuracy, it consequently results in models which require longer training times. Furthermore, the 48-layer variant did not obtain the top result, with the 16 and 28-layer models outperforming it by 0.1% and 0.6% respectively. This suggests that further layers may result in accuracy saturation, which we investigate further below.

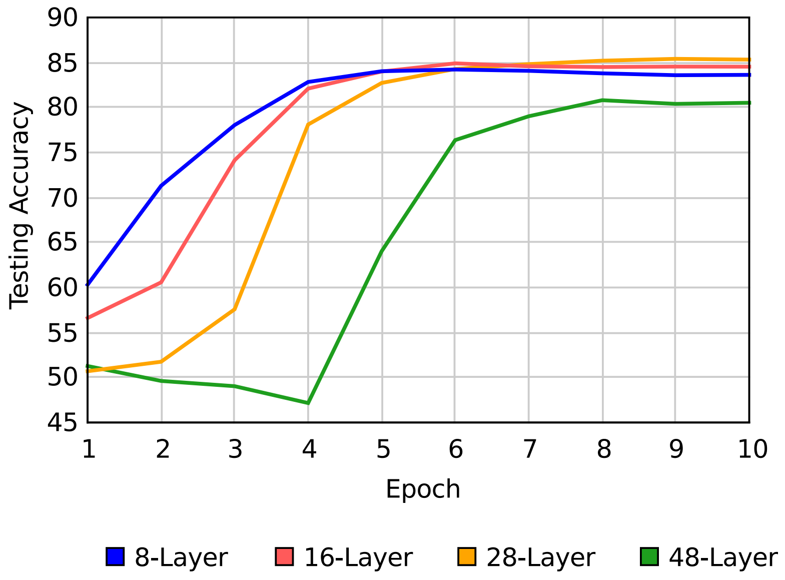


Figure 4.13 ResNet testing accuracy vs epoch, by layer count.

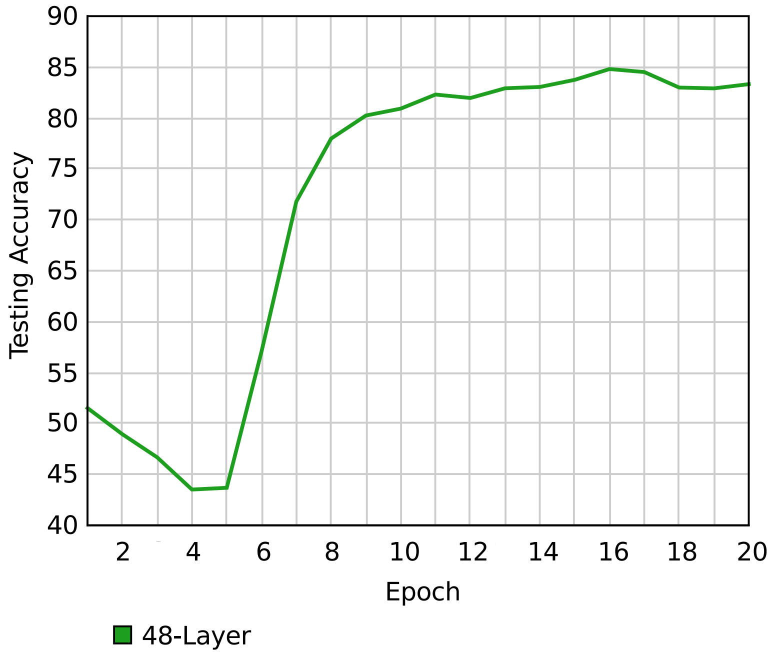


Figure 4.14 48-layer ResNet testing accuracy vs Epoch, across 20 Epochs

Each residual network outperformed its corresponding CNN counterpart, with the 8-layer ResNet obtaining a 0.33% increase in accuracy over the 8-layer CNN. The 16 and 28-layer ResNet variants yielded an increase of 0.76% and 1.5% compared to their equivalent CNNs. This is most notable in comparison of the 48-layer networks over 10 epochs, with an increase in testing accuracy of 21.07% being observed. These results demonstrate that the skip connections occurring between each res-block facilitate an increase in depth, as expected. This is further reinforced by the observed increase in accuracy between the 16 and 28-layer residual variants, whereas the CNN displayed sensitivity to these deeper depths.

An additional benefit of residual learning is identified when comparing the epoch vs current accuracy of the ResNet and CNN: less training is necessary for a deeper residual network to obtain a similar accuracy as a shallower counterpart. The 16-layer ResNet required 3 epochs before obtaining a similar testing accuracy as the 8-layer, whilst the 28-layer required 4. This is in contrast to the 16-layer CNN, which required 4 epochs to obtain an accuracy close to the 8-layer, and the 28-layer which required 6. This observation is most pronounced within the 48-layer models, in which the CNN needed 32 epochs to obtain a score above 80%, whereas the residual variant needed only 8.

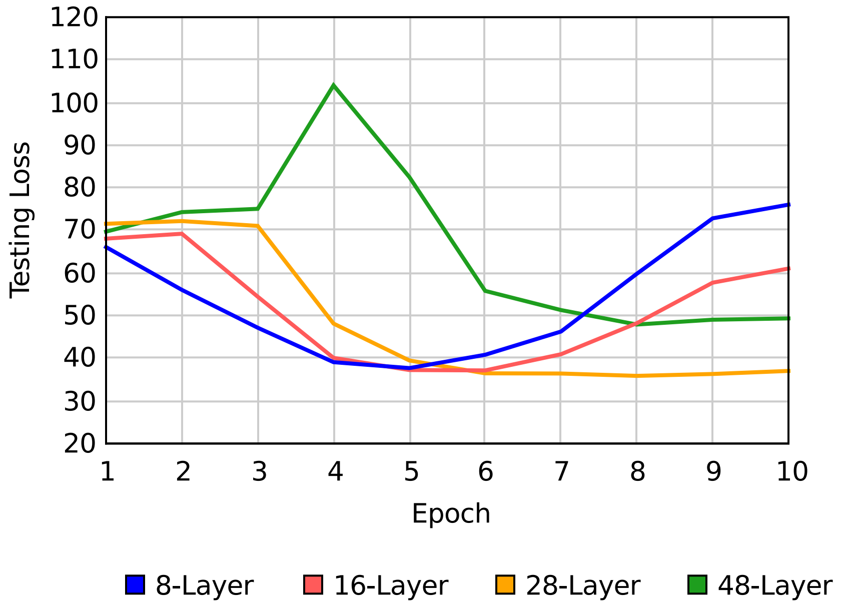


Figure 4.15 ResNet testing loss vs epoch, by layer count.

A comparison of the testing loss of each depth ResNet is shown in Fig. 4.15. The notion of an increased rate of learning identified within the ResNet is further reinforced upon comparison of its training loss with that of the CNN. The minimum loss of the CNN variants occur at distinctively different epochs, causing the plots to overlap once overfitting begins. The ResNet minima are obtained closer together with a less definitive overlap. Noting the testing loss for each depth ResNet is slightly less than its CNN counterpart. Analysis of the training loss of the ResNet and CNN variants indicates the ResNet is less prone to overfitting. With the 8, 16 and 28-layer CNN having a 1.19, 3.45 and 3.98 lower training loss respectively, illustrating a greater tendency to overfit compared to their residual counterparts.

DenseNet

The varying depth DenseNets were trained and tested on the Headlines dataset 10, with results presented as an average over 10 runs. The top testing accuracy and lowest testing and training loss are detailed in Table 4.7 with the bold entry indicating the best result.

	Testing Acc.	Testing Loss	Training Loss
8-Layer	85.55	35.77	6.75
16-Layer	86.66	33.81	7.18
28-Layer	87.29	34.72	7.53
48-Layer	**87.63**	34.05	7.61

Table 4.7 DenseNet results: highest testing accuracy and lowest testing/training loss, by depth.

The DenseNet architecture experienced better performance with an increase in depth, similarily to the ResNet. However, the degree of this improvement is more pronounced within the DenseNet, with the 8-layer resulting in an 85.55% testing accuracy. The 16, 28 and 48-layer models obtained testing accuracies of 86.66%, 87.29% and 87.63% respectively. Resulting in an overall increase of 2.23% compared to the best performing ResNet. Figure 4.16 below, depicts a comparison of the results yielded by each depth over 10 epochs.

Each depth variant outperformed its respective ResNet counterpart, with the 8-layer DenseNet obtaining an increase of 1.33% over the 8-layer ResNet. The 16, 28 and 48-layer DenseNets achieved an increase of 2.13%, 1.89% and 6.83% over their respective ResNet variants. It is noted that the 48-layer DenseNet does not experience the considerable drop in initial performance identified within the ResNet and CNN. Further, the 48-layer dense model obtains the top score across all investigated networks thus far. This continual increase in performance with further depth is due to the dense connectivity within the architecture. The features found early on in the network are reused and not lost due to continual convolutions, allowing the network to form a more diverse feature map with which to make the final classification.

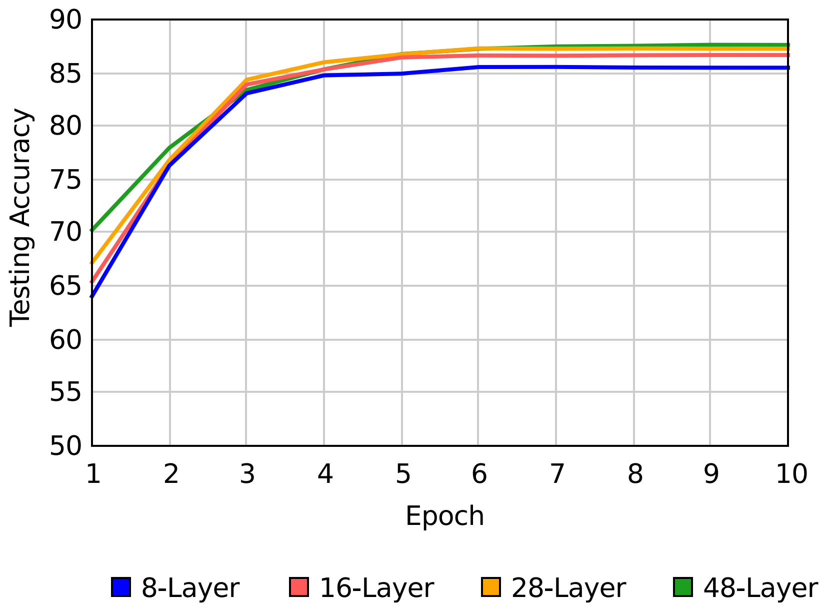


Figure 4.16 DenseNet testing accuracy vs epoch, by layer count.

Furthermore, the variation in the number of epochs required for model convergence is not present within the DenseNet architecture. Each network obtained similar results across each epoch, with the deeper models demonstrating a prominent increase in accuracy over the first 2 epochs. The validation loss between the models has a smaller deviation compared to the prior results, as seen in Fig. 4.17. Noting that whilst the 48-layer network yields the highest accuracy, the lowest testing loss is obtained by the 16-layer network. This suggests that whilst depth increases the dense models ability to detect sarcasm, it does not necessarily increase the confidence with which it makes the classification. The final training loss for each depth is evenly distributed between 6.7-7.6, showing less tendancy to overfit compared to the CNN, but slightly more than the ResNet. A rerun of the 48-layer DenseNet with additional epochs was not required, with the model obtaining its minimum testing loss at the fifth epoch. This effectively demonstrates the capability of these dense models to rapidly converge compared to the other architectures.

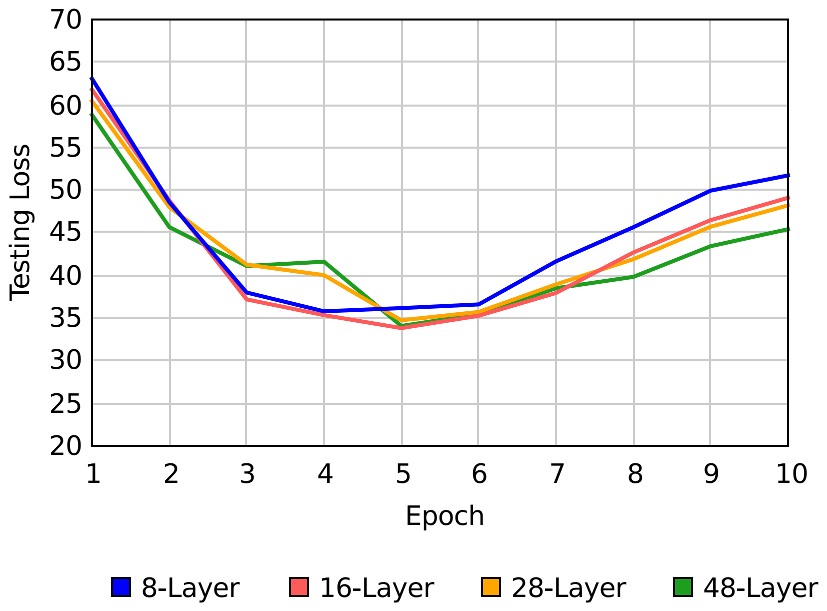


Figure 4.17 DenseNet testing loss vs epoch, by layer count.

Extreme Depths

Prompted by the consistent increase in performance with additional depth identified within the DenseNet, two further models were implemented: a 150-layer ResNet with block configuration (5, 12, 54, 5) and a 130-layer DenseNet with block configuration (6, 12, 64, 48), equivalent to the deepest depths explored by the original papers. These extremely deep models were trained and tested in the same manner as the prior networks. However, 20 epochs were used to ensure the models began to overfit before training stopped, with the training and testing losses tracked to ensure this was the case. The choice of such a substantial increase in layer count is due to the diminished gain in accuracy as depth increased, identified in the DenseNet results. We hypothesise that a further increase would result in negligible benefit. Hence, this extreme depth is chosen to investigate whether the dense model will encounter a decrease in accuracy amongst the initial epochs, as noticed in the ResNet, and to examine whether the overall decrease in accuracy identified in the 48-layer ResNet will be further exacerbated by this depth. Averaged results are recorded in Table 4.8, with the testing accuracy obtained over 20 epochs plotted in Fig. 4.18 below.

	Testing Acc.	Testing Loss
150-Layer ResNet	84.08	43.29
130-Layer DenseNet	87.02	32.16

Table 4.8 Very deep DenseNet and ResNet results: highest testing accuracy and lowest testing loss, across 20 epochs.

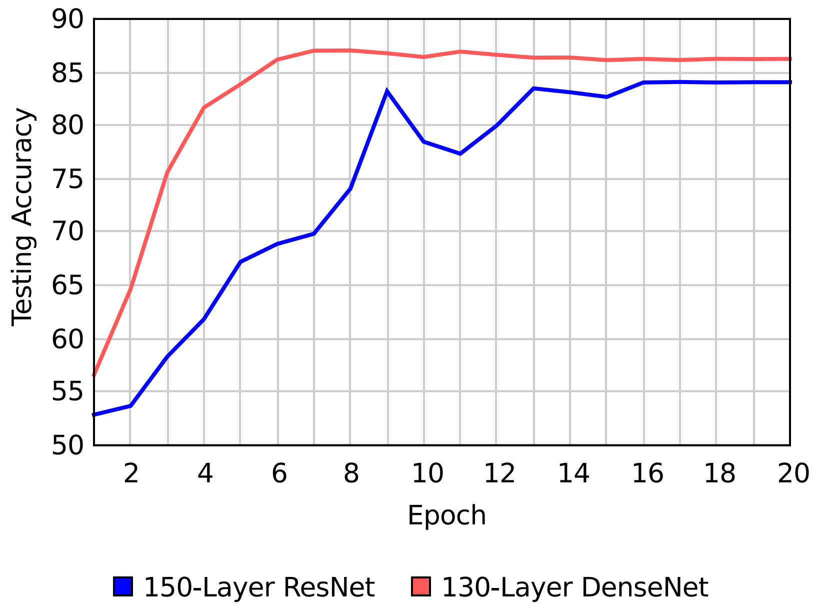


Figure 4.18 150-Layer ResNet and 130-Layer DenseNet testing accuracy vs epoch.

The 130-layer dense model clearly outperforms the 150-layer ResNet, consistent with the above shallower variants. Despite the considerable increase in layers, the DenseNet obtains an accuracy of 87.02% within the first 10 epochs, suggesting that the additional epochs required for convergence within deeper ResNets is not of concern for the dense archetypes. This is noteworthy as it displays the efficiency with which dense connectivity facilitates vast depths. However, the 48-layer DenseNet outperformed the 130-layer variant by 0.61%, whilst the 150-layer ResNet resulted in a decrease of 0.72% compared to its 48-layer equivalent, supporting the hypothesis that depth will only benefit the networks up to a certain point. Further, the accuracy curve yielded by the DenseNet is smooth compared to the residual model, which exhibits fluctuations from epochs 9 to 16. This suggests the residual archetypes are more sensitive to a depth of this degree. Furthermore, the DenseNet obtains a minimum testing loss of 32.16 whilst the ResNet only obtains 43.29, implying the ResNet is considerably less confident when classifying the text. Analysis of the testing losses plotted in Fig. 4.19 strengthens this thought as we observe the ResNet fluctuate across the epochs whilst the DenseNet curve is smooth.

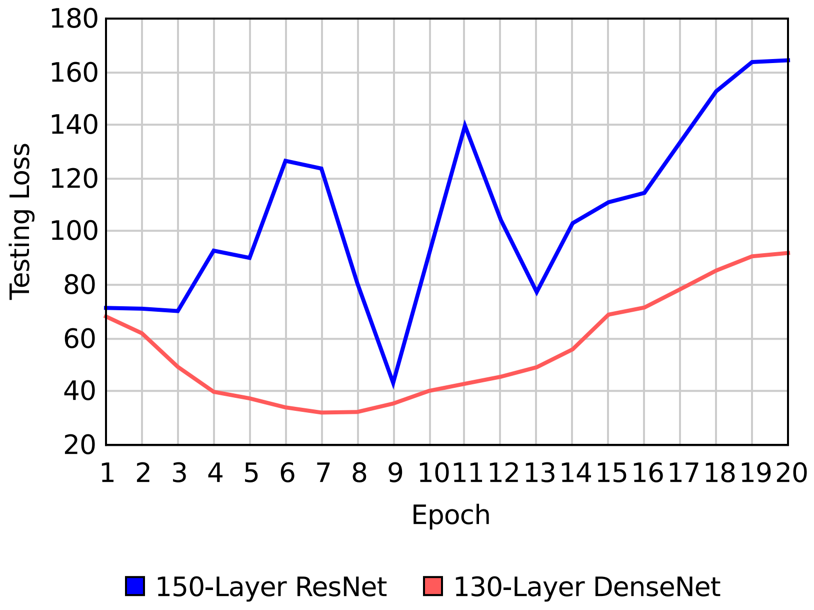


Figure 4.19 150-Layer ResNet and 130-Layer DenseNet testing loss vs epoch.

Summary and Discussion

The CNN did not experience much benefit from an increase in depth, with the 16-layer obtaining a small gain over the 8-layer. The 48-layer CNN required substantially more epochs to converge compared to the ResNet and DenseNet versions. Their respective block structure not only facilitates deeper depths, but also an increase in testing accuracy. This was expected based on prior research into deep residual networks for NLP [46]. The greatest improvement came from the DenseNet, which consistently led to increased accuracy the deeper the network went, yielding the top score of 87.62% with the 48-layer model. Further it is seen that the degree to which depth is assisting the network decreases each time, suggesting that depth will only help to a certain degree within the DenseNet. This was confirmed upon investigation of a 130-layer implementation, which still performed well, but exhibited a small loss in accuracy compared to the 48-layer model. The DenseNet results suggest that the low-level features found earlier on in the network are beneficial towards the final classification. Within the CNN and ResNet these simpler features are lost to the hierarchy - through continual convolutions - as the model grows deeper, which may be the cause of the decreased performance. We now provide analysis of the parameter count of each model, with the testing accuracy vs parameter count plotted in Fig. 4.20.

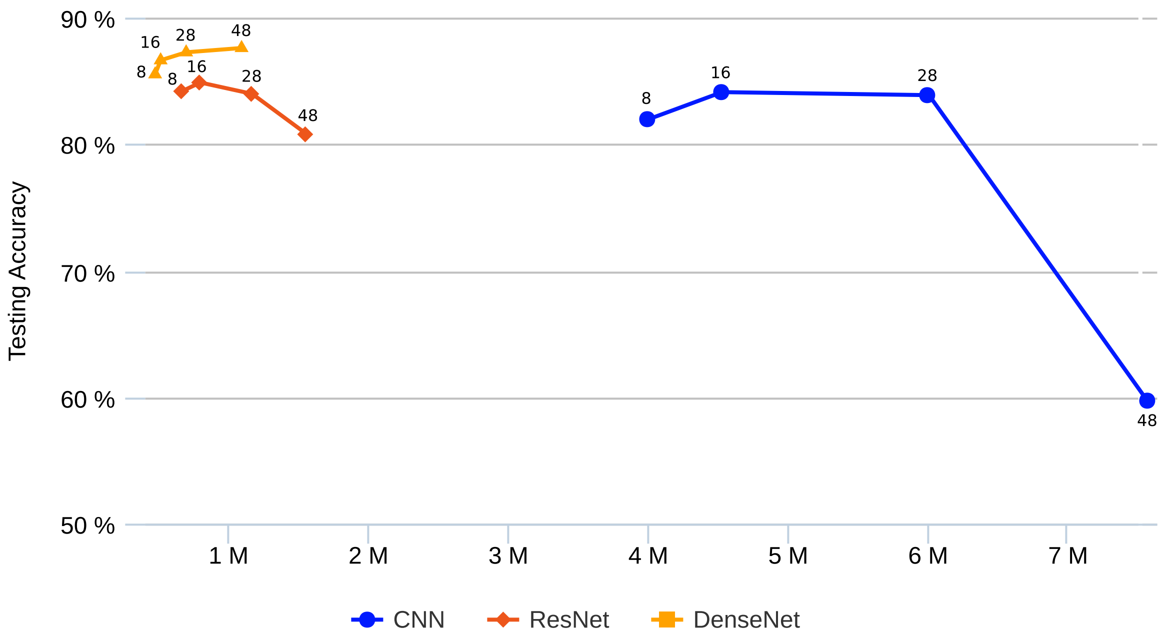


Figure 4.20 Parameter count vs testing accuracy for each of the models. Parameters are represented in millions (M).

The CNN has significantly more parameters compared to the ResNet and DenseNet architectures, with the 48-layer CNN having a colossal 7.56 million, whilst the 48-layer ResNet and DenseNet contain 1.58 million and 1.09 million parameters respectively. Further, the 8-layer CNN has roughly four times the parameter count of the 16 layer ResNet, but resulted in a decrease of 1.02% in accuracy. This supports the hypothesis that depth will be an effective means of increasing performance: fewer parameters are used, but a higher testing accuracy is obtained. Furthermore, the increased performance of the ResNet and DenseNet models compared to CNNs of equal depth suggests that the feature reuse mechanisms within these archetypes are playing a key role in aiding the network. A detailed chart of the parameter count for the ResNet and DenseNet variants is depicted in Fig. 4.21 to better analyse the efficacy of dense connectivity.

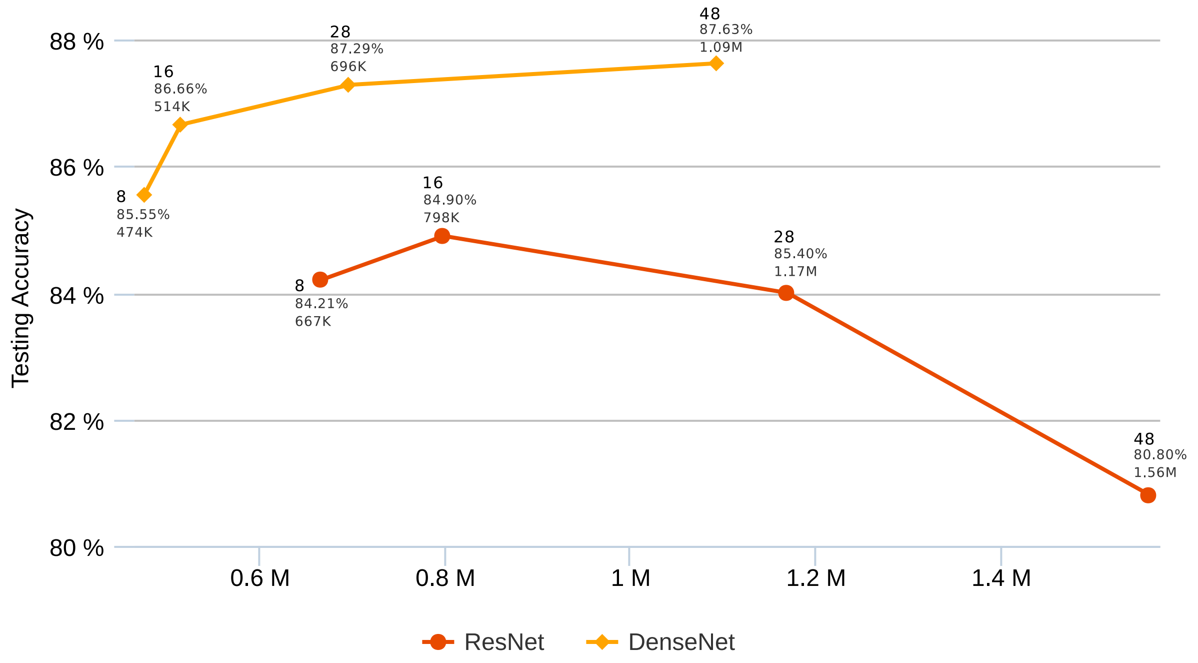


Figure 4.21 Parameter count vs testing accuracy for the ResNet and DenseNet variants. Parameters are represented in millions (M).

The dense models obtain better accuracy with fewer parameters compared to the ResNets. A 474 thousand parameter 8-layer DenseNet outperformed the 798 thousand 8-layer ResNet by 1.34%. This trend is consistent across each depth with the residual networks having roughly ~50% additional parameters, demonstrating inefficient parameter usage within the ResNet. This is most noteworthy when contrasting the 48-layer DenseNet with the 28-layer ResNet: the dense model outperforms the ResNet by 2.23% whilst having 80 thousand fewer parameters but almost twice the depth. Furthermore, considerably fewer parameters are required for an increase in depth within the DenseNet, with only 40 thousand parameters required to transform the 8-layer into a 16-layer, whilst the ResNet demands an additional 131 thousand. These results display the advantage of dense connections within the network, and the decreased parameter count supports the hypothesis that these elementary features extracted during the initial convolutions are indeed valuable for sarcasm classification.

Whilst feature reuse is beneficial, as seen in comparing the CNN to the ResNet, it is the dense network's ability to maintain low-level features throughout the network which ultimately gives it an edge over the residual model. Through skip connections implemented with concatenation, the model is able to build a diverse feature map containing less abstracted representations of the initial features. The clearer retention of these less complex features, when combined with the higher level abstract features, provide the dense networks with richer information with which to model the text, thereby facilitating more efficient and accurate learning.

4.1.6 Growth Rate

A larger growth rate consistently increased testing accuracy within the model, most noticeably from *GR*=2 to *GR*=16 which leads to an increase of accuracy from 85.11% to 86.81%. A small increase of 0.16% is experienced between *GR*=16 and *GR*=32 but the effect appears to taper off at this point, seen in Fig. 4.22 below. A potential explanation for this could be that the model is not finding additional information from the extra feature maps at the current level in the hierarchy. The testing accuracies are near identical during the first three epochs, however, the values begin to diverge from epoch 4 until 6 where convergence occurs.

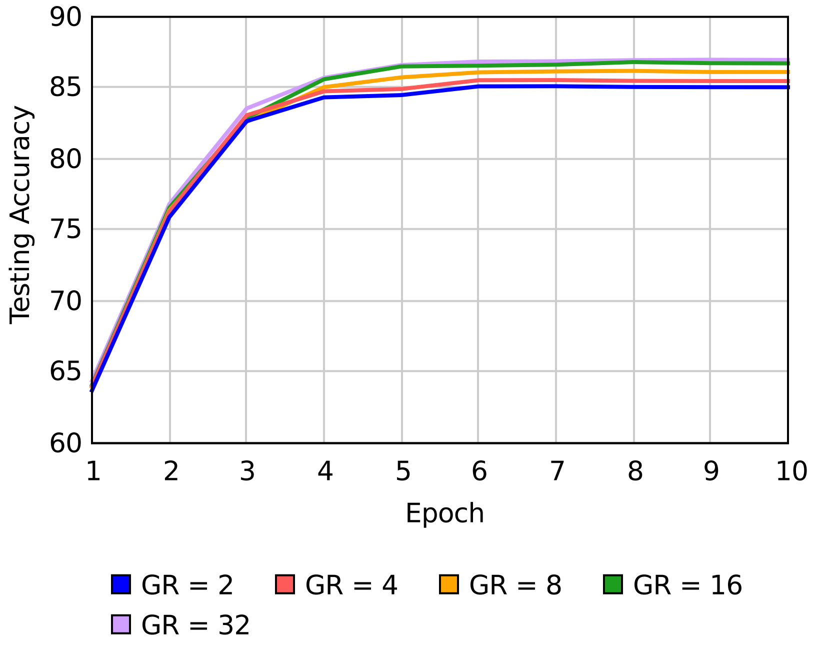


Figure 4.22 Growth rate vs DenseNet testing accuracy.

Furthermore, upon analysis of each variants training loss, the wider networks are not exposed to an increased likelihood of overfitting. The initial loss varies between each configuration, however each converges within a small range of one another post epoch 5, shown in Fig. 4.23 below. This is favourable as whilst widening the network is beneficial, the risk of overfitting is not increased to a degree which would require further regularisation. This demonstrates an increase in width is a viable option to improve convolution based archetypes in sarcasm classification.

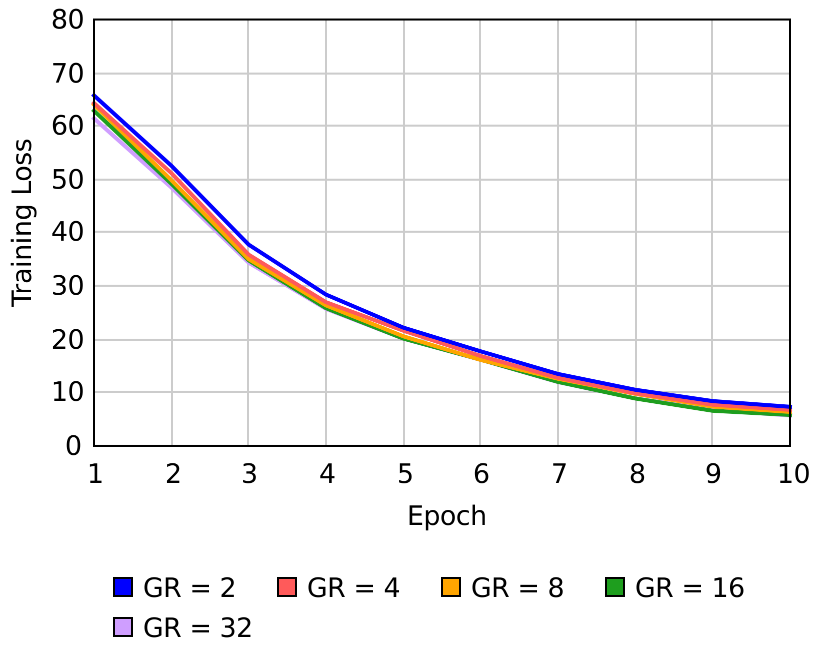


Figure 4.23 Growth rate vs DenseNet training loss.

4.2 Benchmark Comparisons

The models outlined in Section 3.3 are trained and tested on the datasets detailed in Section 3.1 for 10 epochs. This is repeated 10 times, with results presented as an average over the runs. DweNet is the proposed approach of this work, with the introduced ResNet and CNN treated as baselines for comparison. The results obtained by prior papers on the Reddit datasets are presented to provide further comparison of our approach against recent state-of-the-art models. We further demonstrate through empirical means, that our model - DweNet - which uses only the intrinsic properties of the text, is able to rival networks which make extensive use of context and meta-data. We hypothesise that the diverse feature maps resulting from the combination of low-level and complex hierarchical features plays a key role in DweNets ability to compete with these networks. To test this, a study on the dependency on prior feature maps in producing the final feature map within the model is conferred. Finally, a case study on sentences which DweNet accurately classifies - but a standard CNN misclassified - is presented.

4.2.1 Prior Approaches

Prior state-of-the-art approaches for the Reddit Main and Reddit Pol datasets are detailed below.

- **Bag-of-words**: An SVM network which receives the word-counts of the text as a vector of the size V, where V is the vocabulary size.
- **CNN**: A simple CNN with 3 different filter sizes to extract n-gram features, as detailed by Hazarika et al. [7].
- **CNN-SVM**: An ensemble of 4 CNNs in which 3 are pretrained to extract sentiment, emotion and personality features from the given comment proposed by Poria et al. [6]. The outputs are concatenated and passed to an SVM for the final classification.
- **CUE-CNN**: Proposed by Amir et al. [4], user embeddings are modelled to obtain stylometric features which are then combined with a CNN.
- **CASCADE**: Stylometric and personality features of the authors are modelled and fused using canonical correlation analysis to obtain comprehensive user embeddings. Further, for each forum, all previous comments are combined to obtain discourse representations surrounding the topics and patterns therein. This approach was proposed by Hazarika et al. [7] and holds the record results on both datasets.
- **CASCADE$_{\text{NP}}$**: The above CASCADE model but no personality features are used.
- **AMR**: An RNN-based model presented by Ghaeini et al. [14] incorporating BiLSTMs to model input comments and responses thereof. Attention mechanisms, projection and re-reading are also used to provide deeper representations.

4.2.2 Results

Here, we present the results of our model, DweNet, compared to our proposed baselines and the recent approaches outlined above. Table 4.9 details the results obtained on the Headlines, Reddit Main and Reddit Pol datasets. To the best of our knowledge, no prior results have been published for the Headlines dataset, hence we compare DweNet against our proposed baselines alone. Entries recorded in bold represent the results of DweNet, whilst blue entries indicate results which are rivaled by DweNet. Red entries correspond to results which were greater than DweNet's.

Dataset	Model	Acc.	F1
Headlines	CNN_{base}	0.83	0.83
	$\text{ResNet}_{\text{base}}$	0.85	0.85
	DweNet	**0.88**	**0.88**
Reddit Main	CNN_{base}	0.67	0.67
	$\text{ResNet}_{\text{base}}$	0.68	0.68
	DweNet	**0.69**	**0.69**
	Bag-of-words	0.63	0.64
	CNN	0.65	0.66
	CNN-SVM	0.68	0.68
	CUE-CNN	0.70	0.69
	CASCADE	0.77	0.77
	$\text{CASCADE}_{\text{NP}}$	0.68	0.66
	AMR	0.68	0.70
Reddit Pol	CNN_{base}	0.62	0.62
	$\text{ResNet}_{\text{base}}$	0.63	0.63
	DweNet	**0.69**	**0.69**
	Bag-of-words	0.59	0.60
	CNN	0.62	0.63
	CNN-SVM	0.65	0.67
	CUE-CNN	0.69	0.70
	CASCADE	0.74	0.75
	$\text{CASCADE}_{\text{NP}}$	0.68	0.70

Table 4.9 Results of DweNet compared to our proposed baselines, and prior approaches, on the Headlines and Reddit datasets.

DweNet obtained the top result of 88% in accuracy and F1 score on the Headlines dataset, yielding an increase of 5% and 4% over the baseline CNN and ResNet respectively. However, on the Reddit Main dataset, the performance's were similar, with DweNet obtaining 69% on both measures, whilst the CNN and ResNet scored 67% and 68% respectively. The smaller increase in performance experienced by DweNet on the Reddit Main dataset is likely due to the commonality of informal language within the datapoints, which has been shown to decrease the reliability of grammatical cues [5]. Consequently, DweNet is able to extract less additional information from the utterance than on a formal dataset such as Headlines, resulting in similar performance across each model. Finally, DweNet again experienced a significant increase in accuracy compared to the two baselines on the Reddit Pol dataset. Yielding an accuracy and F1 score of 69% whilst the baseline CNN and ResNet obtained 62% and 63% respectively. Whilst Reddit Pol also contains informal language, it is focused on political commentary, and consequently contains less noisy data compared to Reddit Main, which is likely the reason for the substantial performance increase over the other baselines.

Comparing the results against those obtained by recent approaches, we can see that DweNet is able to rival each network except the full CASCADE implementation. This is interesting considering DweNet models only the content of the utterance, whilst the other networks integrate extrinsic information to varying degrees. Table 4.10 details the context used by each network, noting that DweNet rivaled all but the final network in the table. From this, and the results above, we can see DweNet was able to outperform the $\text{CASCADE}_{\text{NP}}$ model, which implemented context - through the parent comments - performed user profiling to capture stylometric features about the authors, as well as modelled the entire discourse of the forum. DweNet also outperformed the CNN-SVM model, which was pretrained to extract emotion, sentiment and personality features from the text, on both Reddit datasets.

The overall top score for Reddit Main was 77% and obtained by CASCADE, a substantial 8% higher than DweNet's results. CASCADE also yielded the top score on Reddit Pol with 74% and 75% accuracy and F1 score respectively. This corresponds to an increase of 5% and 6% on accuracy and F1 compared to DweNet. These significantly higher results demonstrate the necessity of contextual information to obtain the current state-of-the-art. However, due to the promising performance of DweNet compared to the other networks, we can conclude that additional information is indeed available within the isolated utterance. Consequently, the integration of dense connectivity with the contextual information used in the above networks, would likely lead to even better results. However, we leave this for future work, and only explore the effect of local context in Section 4.3.2 in a trivial manner.

	Context	U.E.	Personality	Discourse
DweNet	-	-	-	-
CNN-SVM	-	-	✓	-
CUE-CNN	-	✓	-	-
AMR	✓	-	-	-
$\text{CASCADE}_{\text{NP}}$	✓	✓	-	✓
CASCADE	✓	✓	✓	✓

Table 4.10 A comparison of the extrinsic information used by the above models. U.E. refers to the learning of user embeddings for user profiling within the network.

4.2.3 Low-level Feature Use

A similar approach as Huang et al. [42] is taken to support the hypothesis that low-level features - in combination with abstract hierarchical features - play a key role in increasing sarcasm detection. A 16-Layer DenseNet with block configuration (4, 4, 4, 4) and a growth rate of 4 is trained on the Headlines dataset. The $L1$ norm of the weights connecting the preceding layers to the final layer within each dense block is taken in order to determine the dependency of the final feature map on prior feature maps. A detailed heatmap is shown in Fig. 4.24 to visualise the intensity of this dependency.

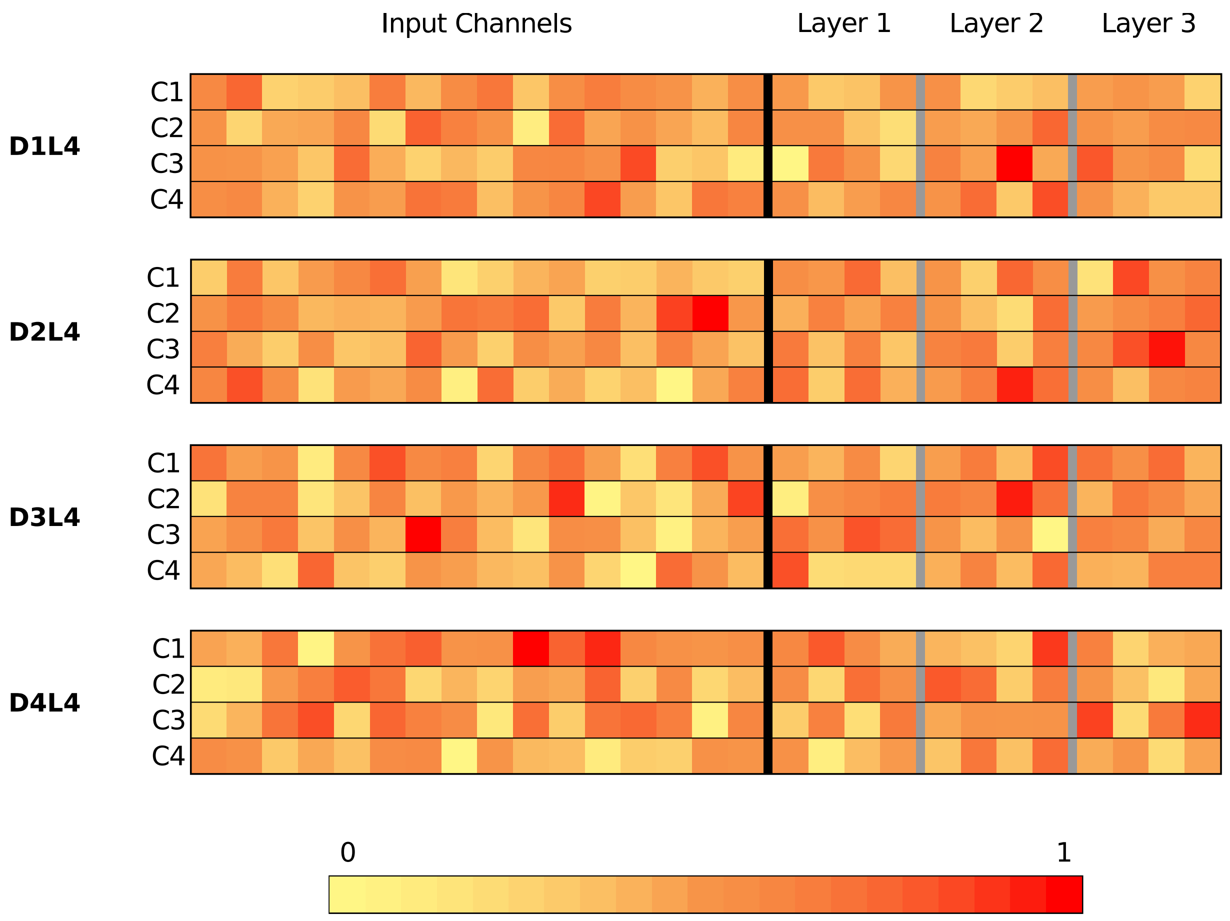


Figure 4.24 The average absolute filter weights of the convolution layers connected to the final layer of each dense block (D1L4 to D4L4), channel by channel (C1 to C4). The colour encodes the normalised magnitude of these weights. The input feature maps are placed behind the vertical black bar, representing low-level features found earlier on in the network. Layer 1, 2 and 3 - separated by the grey rectangles - represent the initial three layers in the block.

From this, we can see that the network assigns relatively strong importance to the input planes of each block, illustrating low-level features are indeed being used to form the final feature maps within each block. Furthermore, the weights connecting the preceding 3 layers also exhibit high values, demonstrating these higher-level features also influence classification. These strong dependencies are identified by the red blocks within the heatmap, whilst orange blocks indicate features of average importance. The yellow blocks display features with little precedence and can be seen scattered throughout the heatmap; hinting that the features extracted by these layers are not significant, or may be contained within another feature map.

4.2.4 Case Study

The above analysis suggests that dense connectivity indeed enables our model to rival related works which make extensive use of contextual information and meta-data, such as an author's post history and trends within a specific subforum. The baseline CNN was trained on the headlines dataset with incorrect classifications recorded. Similarly, we captured all errors made by DweNet and the set difference was taken, that is, headlines which our model correctly identified, but the CNN did not. Analysis of these misclassifications was performed to determine whether our model was able to correctly classify additional headlines in which the sarcasm - or lack thereof - is clear based solely on the utterance. Below, we present a couple of these cases, paired with their actual label - sarcastic or nonsarcastic.

- *Efforts of world's 16 billion chickens still not adding up to much.* - sarcastic
- *CEO unveils bold new plan to undo damage from last year's bold new plan.* - sarcastic
- *Like boxes of shit in your house? Get a cat.* - sarcastic
- *United airlines temporarily suspends cargo travel for pets.* - nonsarcastic
- *There have been more mass shootings this year than there have been days.* - nonsarcastic

The initial three headlines in the above list are all clearly sarcastic, with no background knowledge required to classify it as such. The latter two are seemingly normal news headlines with no sarcastic cues present. All five were incorrectly classified by the CNN, but identified correctly by our approach. This demonstrates that additional cues are available in the isolated text of the utterance than a standard CNN is able to extract. We do however note that our approach fails to classify text which require a clear understanding of the topic or background, such as:

Cops cleared on corruption charges after implicating decorated police dog.

This statement satirises cases where law enforcement were not held accountable for corruption, and further requires an understanding that a dog cannot be culpable of such an offence.

4.3 Exploratory Investigations

DweNet is used to perform exploratory investigations into non-structural aspects of the network, details of which are presented in Section 3.4.2. The obtained results and analysis of any findings therein are provided in the following order: Embeddings, effect of local context, efficacy of data preprocessing.

4.3.1 Embeddings

The embedding layer of DweNet was initialised using the four investigated word representations: GloVe$_{6B}$, FastText$_{1M}$, FastText$_{1M\text{-sub}}$ and FastText$_{2M}$. The model was trained for 10 epochs, which was sufficient for overfitting to occur. The accuracy, recall, specificity and precision obtained on the test set was recorded after each epoch. This was repeated 10 times with the results recorded as an average across all runs. Further, the models were trained and tested using static embedding layers for each word representation, with results calculated as above. The final results obtained are presented in Table 4.11 below.

					Headlines				
	Dim.	$\|V\|$	$\|V_{train}\|$	Emb.	*Acc.*	*R*	*S*	*P*	*Word Coverage*
GloVe$_{6B}$	50	400k	6B	*Non-static*	88.87	87.53	89.91	87.16	99.51
				Static	85.24	85.68	84.88	81.60	
FastText$_{1M}$	300	1M	16B	*Non-static*	86.65	85.50	87.55	84.31	97.93
				Static	83.63	85.08	82.49	79.17	
FastText$_{1M\text{-sub}}$	300	1M	16B	*Non-static*	86.32	84.73	87.57	84.21	97.93
				Static	83.29	85.09	81.89	78.61	
FastText$_{2M}$	300	2M	60B	*Non-static*	86.37	85.64	86.94	83.69	99.24
				Static	83.63	85.30	82.23	79.07	

Table 4.11 DweNet results on the Headlines dataset using different embeddings.

The non-static GloVe embeddings outperformed all non-static FastText variations across each metric, obtaining a 2.22%, 2.03%, 2.36% and 2.85% increase in accuracy, recall, specificity and precision, respectively, against the $\text{FastText}_{\text{1M}}$ model. The higher degree of word coverage provided by the GloVe embeddings may account for this increase in performance, having an additional 1.58% coverage compared to the $\text{FastText}_{\text{1M}}$ embeddings. $\text{FastText}_{\text{2M}}$ has a similar word coverage to GloVe, differing by only 0.27%, however yields similar or worse performance than $\text{FastText}_{\text{1M}}$, suggesting that word coverage is not solely responsible for the better results. Further, all FastText variants obtain similar results across each metric, which may indicate that the initial training of the embeddings is influencing performance.

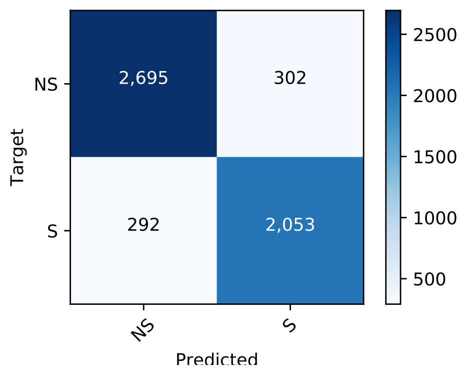
The FastText embeddings are trained on significantly larger English datasets. $\text{FastText}_{\text{1M}}$ and $\text{FastText}_{\text{2M}}$ were trained on 16 billion and 60 billion tokens respectively, compared to GloVe's 6 billion. The datasets used for training will rarely include sarcastic phrases, FastText's larger training size would thus lead to a better understanding of the general lexicalities of the English language. However, traditional English usage is vastly different to sarcasm, suggesting the embeddings which are trained on larger datasets may be more ill-aligned for the semantics of sarcasm. This, in conjuction with FastText's considerably larger dimension, causes the natural structure of English to be ingrained within the FastText embeddings, whereas the GloVe model may update its embeddings to facilitate sarcasm classification more easily. Consequently, during training, the GloVe model is able to better update the embeddings for sarcasm classification. This is reinforced by comparing the increase in recall obtained in the GloVe model, compared to all FastText variants when using a non-static embedding layer.

The non-static GloVe model experiences an increase in recall of 1.85% compared to its static counterpart. An increase to this degree however is not seen within the FastText networks, with the largest increase being 0.42% and the subword model suffering a decrease of 0.36%. This indicates that when the networks are allowed to update the pre-trained embeddings, the FastText models battle to better align the word vectors for sarcasm identification. However, in comparing the specificity of each, we see the non-static variant leads to significant increase across all four embeddings. This consequently results in a substantial increase in precision observed within the non-static models. Hence, whilst allowing the FastText models to learn better embeddings does not improve sarcasm identification, it does benefit non-sarcastic sentence detection. This further suggests the FastText embeddings may have the natural usage of English too well reinforced within its representations.

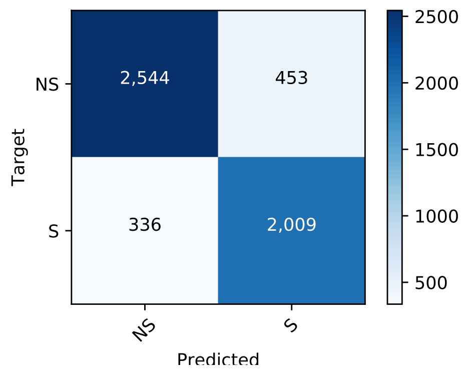
This is visualised in the confusion matrices depicted in Fig. 4.25, Fig. 4.26, Fig. 4.27 and Fig. 4.28 below, the left matrix representing the non-static results and the right matrix the static results. A blue heat map is applied to visualise the difference between the number of correct and incorrect classifications - true positive, false negative, etc. - within each matrix. The non-static GloVe model classifies an additional 44 sarcastic and 151 non-sarcastic sentences correctly compared to the static variant. Whilst the non-static $\text{FastText}_{\text{1M}}$ model correctly classified an additional 152 non-sarcastic sentences, but only 10 sarcastic sentences. The non-static $\text{FastText}_{\text{1M-sub}}$ model incorrectly classifies an extra 8 sarcastic sentences, but sees an increase of 170 non-sarcastic sentences correctly identified. The poorer performance of the subword embedding variant is possibly caused by the high word coverage of the embedding, identifying 97.93% of the words in its vocabulary. The subword embedding is better suited for handling unknown words, that is, words which are not present in the embedding vocabulary. Thus, given the high word coverage, little benefit is seen from better placement of unknown words within the embedding space. Finally, the non-static FastText$_{2M}$ model saw an increase of 138 correctly classified non-sarcastic sentences, but saw a negligible increase of 8 additional sarcastic phrases correctly identified, similar to the FastText$_{1M}$ network.

Furthermore, the GloVe vocabulary is less than half the size of the FastText$_{1M}$ vocabulary, yet yielded the highest word coverage. This suggests GloVe has a better selection of words typically found in sarcastic phrases contained within its vocabulary. Finally, the dimension of GloVe vectors is a sixth of the size of FastTexts, allowing for quicker training and more efficient parameter usage within the embedding layer. This, in conjunction with the above analysis, provides empirical evidence that GloVe word vectors are best suited for sarcasm classification amongst the investigated embeddings, on a formal English dataset.

The observed increase in specificity across all four embeddings is interesting, as during the training of each model, the ability to accurately identify non-sarcastic sentences increases. GloVe alone was able to yield a noteworthy increase in recall through updating the representations. A potential hypothesis for this would be the models are able to identify words which are highly indicative of non-sarcastic sentiment easier than words implying sarcastic intent. An analysis of the embedding space as the model learns may be studied to provide further insight into why a prominent increase in non-sarcastic detection is witnessed, but a significantly smaller increase in sarcasm identification is observed. This, however, is left to future work.

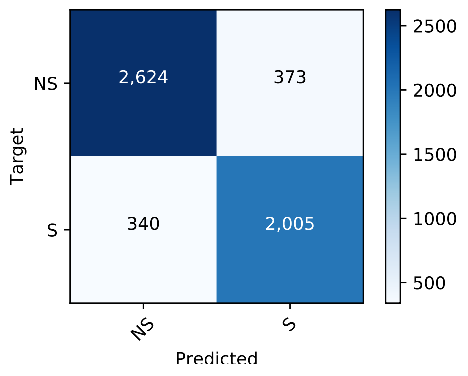


(a) Non-static

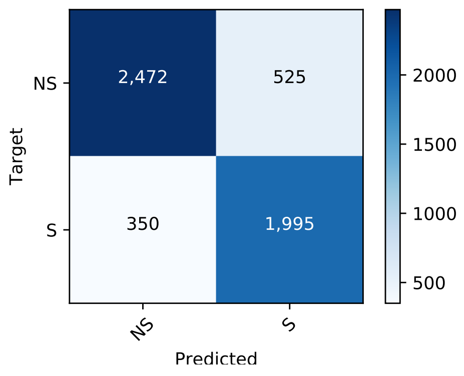


(b) Static

Figure 4.25 Confusion matrices for DweNet implemented with GloVe$_{6B}$ embeddings.

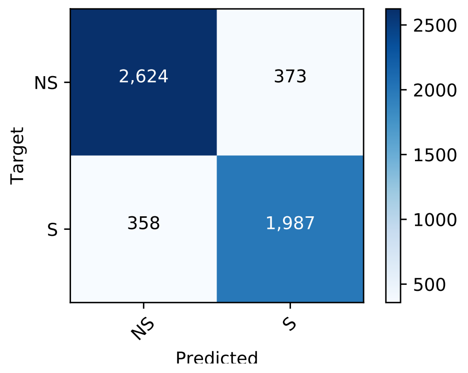


(a) Non-static

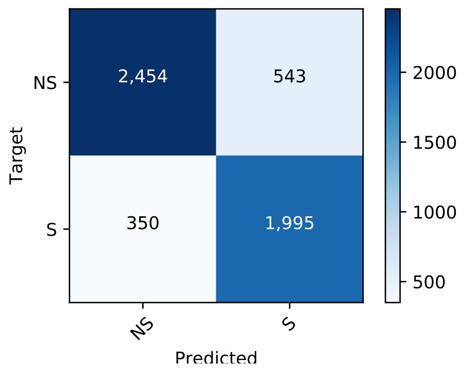


(b) Static

Figure 4.26 Confusion matrices for DweNet implemented with FastText$_{1M}$ embeddings.

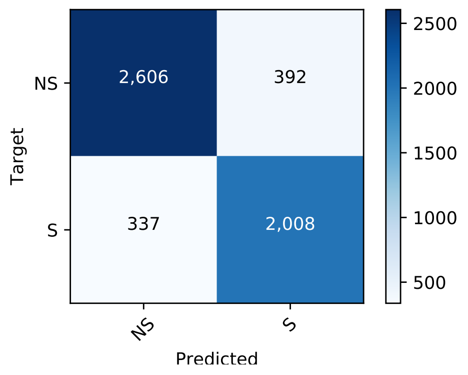


(a) Non-static

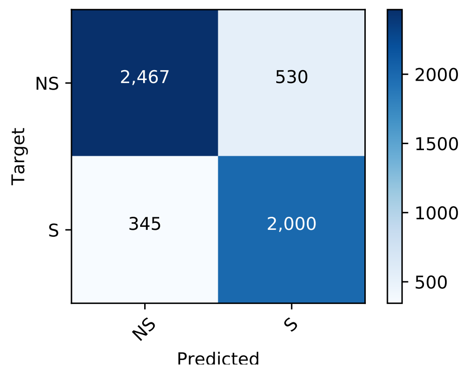


(b) Static

Figure 4.27 Confusion matrices for DweNet implemented with FastText$_{1M\text{-}sub}$ embeddings.



(a) Non-static



(b) Static

Figure 4.28 Confusion matrices for DweNet implemented with FastText$_{2M}$ embeddings.

4.3.2 Effect of Local Context

The Reddit Main dataset was used for this investigation, with DweNet being tested on three variations: the standard Reddit Main detailed in Section 3.1, Reddit Main$_{\text{context}}$ which concatenates the parent comment to the end of the child and Reddit Main$_{\text{context swap}}$ which concatenates the parent comment to the beginning of the child comment. As previously discussed the standard dataset, Reddit Main, makes use of only the child comments for training and testing. Hence, the addition of the parent comment - that is, what the original comment was in response to - can be viewed as adding local context to the model. Table 4.12 details a few examples from the Reddit Main dataset in which the sarcastic intent is only clear once local context has been provided.

C/P	Comment
C	What face swap?
P	My personal favorite face swap.
C	Sounds like a fun guy to be around.
P	I love getting lectured.
C	Probably why they seem to have a firm grasp on how the world works
P	Many of Trump’s Supporters Never Left Their Hometowns
C	Stranger Things, Black Mirror
P	Give me Netflix shows to watch No stranger things, 13 reasons why, AHS, or Black Mirror

Table 4.12 A few samples taken from the Reddit Main dataset in which the sarcastic intent of the child comment (C) is only clear when the parent comment (P) is provided.

The first entry is simply a question, however the addition of the parent comment, '*My personal favorite face swap.*', makes it clear. The user is mocking the other user, implying that they cannot see a face swap had been done. The second entry is about someone being fun to be around, a fairly positive statement, only once we are aware that this person lectures others often can we confidently say the original comment is sarcastic. The third comment could be in regard to a group that indeed does have a firm grasp on how the world works. However, with the parent comment telling us that this group have never left their hometowns, they obviously do not have a firm grasp on the world, which solidifies the sarcastic intent. Finally, the last entry is simply two popular TV shows, which obviously cannot be seen as sarcastic or not. However, looking at the parent comment, the user specifically asked for TV shows that do not include the two mentioned in the child comment. Consequently we can see the child comment was made in jest and is clearly sarcastic.

Whilst the above sentences are readily understood with local context, this is not true for all sentences within the dataset, such as:

> P: *Ted Bundy's signature carved into the defendant table at the Old Orange County Courthouse - Florida*
>
> C: *Great, let's preserve and worship it.*

Even with the addition of the parent comment, the original comment '*Great, let's preserve and worship it.*' cannot be understood as sarcastic or not without knowledge of whom Ted Bundy was. If Ted Bundy was someone famous for positive reasons, this comment would appear sincere. However, we know Ted Bundy was a serial killer and so the response is clearly sarcastic. Why would one want to preserve - let alone worship - the signature of a notoriously evil man? These forms of sentences require further background knowledge to accurately classify and cannot be confidently classified by our approach.

Due to the presence of these sentences which can be understood from local context alone, we hypothesise that the addition of local context - in the form of the parent comment - would lead to an increase in testing accuracy. This however, was incorrect, as seen from the results detailed in Table 4.13.

	Accuracy	$\lvert V \rvert$	*Word Coverage*
Reddit Main	69.18	26291	93.75
Reddit Main$_{\text{context}}$	68.94	43754	89.22
Reddit Main$_{\text{context swap}}$	69.02		

Table 4.13 Accuracy and vocab details for the Reddit Main dataset, with and without local context.

The model receiving the standard Reddit Main dataset yielded the highest testing accuracy of 69.18%, whilst the local context models Reddit Main$_{\text{context}}$ and Reddit Main$_{\text{context swap}}$ obtained accuracies of 68.94% and 69.02% respectively. The 'context swap' model outperforming the 'context' variant is expected: providing the parent comment *after* the child comment is not very logical - as humans clearly read the parent comment and then the child comment. However, the standard model resulting in the top performance was not expected, suggesting that DweNet is not well suited for handling local context. This loss in accuracy with the addition of the parent comment is in contrast to prior works in which local context was found to be beneficial to classification. This is likely due to the trivial manner in which local context was provided to the model, we simply concatenated the two comments and used the 'field' token inbetween. This is in contrast to related approaches which explore more sophisticated methods of handling local context.

Further, the inclusion of the parent comments within the dataset results in a significantly larger vocabulary of 43754 tokens, compared to 26291 tokens contained only in the child comments. Consequently, the word coverage provided by GloVe decreases, which leads to more tokens being assigned random representations within the embedding space. This lower coverage and increased use of randomly initialised embeddings may also be contributing to the weaker performance of the local-context models.

Hence, whilst DweNet is able to compete against networks incorporating context, it does not see the same benefits from such an inclusion. Better techniques exist for implementing local-context within a model, and should lead to better testing accuracy compared to the standard model. We however leave the investigation of such methods to future work.

4.3.3 Effect of Data Preprocessing

In order to evaluate the efficacy of the preprocessing techniques used, the Headlines dataset was preprocessed using four different variations. Table 4.14 below details the results obtained by DweNet using each preprocessing variant, as well as information regarding the resulting vocabulary formed from the preprocessing.

	Headlines			
Preprocessing	$\|V\|$	*OOV*	*Accuracy*	*Word Coverage*
Basic, Min_freq = 1	23439	926	87.95	96.05
Standard, Min_freq = 1			88.09	
Standard, Min_freq = 2	12646	111	88.36	99.12
Standard, Min_freq = 3 *	9121	44	88.87	99.51

Table 4.14 DweNet results using different preprocessing methods on the Headlines dataset.
* The default preprocessing method.

$|V|$ refers to the number of tokens contained within the vocabulary after preprocessing, whilst *OOV* indicates the out-of-vocabulary token count, that is the number of tokens which are not found within the pretrained GloVe embedding. The 'Basic' preprocessing method is simply splitting the words on spaces and punctuation, making no use of the special tokens and rules provided by the FastAI framework, detailed in Table 3.7 and Table 3.8. Whilst the 'Standard' method is the default technique outlined in Chapter 3.1.1 and makes use of these rules and special tokens. The word occurrence threshold, 'Min_freq', indicates the number of times a word must appear within the corpus to be included in the vocabulary. Noting that 'Standard, Min_freq = 3' is the default preprocessing method outlined in Section 3.1.1.

From the first two rows, we can see that the rules and special tokens (excluding the 'xxpad' token and 'xxunk' token) provide a marginal benefit to the network, resulting in an increase of 0.14% in testing accuracy. This negligible gain suggests these rules are only useful in classifying a small number of sentences. A more pronounced gain may be experienced from a different dataset, as the Headlines dataset doesn't make use of the 'replace_rep' and 'replace_wrep' rules often, as it is unlikely to see such repetition in formally written news headlines.

An interesting observation is the gain in testing accuracy as the occurrence threshold is increased. The total number of words in the corpus is 23439, after limiting it to words which occur at least 3 times, the vocabulary size shrinks considerably to 9121, an overall decrease of ~61%. However, this apparent loss of information leads to better classification performance, suggesting these infrequently occurring words are not beneficial towards sarcasm identification. In an attempt to understand why, all words which occurred only once were captured and analysed. Table 4.15 details some of the trends observed across these single-occurrence words.

Locations	Hashtags	Careers	Religions
Haiti	#explainthe90sin4words	Beekeeper	Baha'i
Jamaica	#myroommateisweird	Gynecologist	Evangelicalism
Riyadh	#trickortreatin100years	Neurosurgeon	Satanists
Seychelles	#womenboycotttwitter	Zoologists	Shamanism
Warsaw	#xmasgiftsfromtrump	Waitresses	Zionism
Animals	Medical Disorders	Food	Censoring*
flea	Anorexia	Cheetos	b*tch
ladybug	Bipolar	Dorito	bullsh*t
porcupine	Bulimia	gravy	f**ckup
racehorse	Diphtheria	McMuffins	p***y
squirrels	Endometriosis	marshmallows	sh*t

Table 4.15 Categories of words occurring only once within the Headlines dataset.
* Recorded exactly as they appeared within the corpus.

The infrequently occurring tokens were found to contain many specific words, a majority of which were proper nouns. These words may be the target of sarcasm, but alone do not necessarily have a sarcastic polarity, such as the words occurring under the 'Location', 'Careers', 'Religion', 'Animals', 'Medical Disorders' and 'Food' categories. Whilst most of these tokens will occur within the embedding dictionary, their distinct neutrality - in the context of sarcasm - and infrequent use make them of little importance for identification. Consider the phrase: 'Oh WOW, I realllly love *X*.', this is clearly sarcastic, and we can recognise it as such without knowing the intended target *X* - be it a flea, McMuffins or a city like Warsaw. Hence, the replacement of such words with the 'unknown' token should have little effect on sarcasm identification. We do note that these words may arise in sentences which are not easily identified based on the other tokens in the sentence, such as: 'The weather is great in the Seychelles right now.', which clearly requires context (is the weather *actually* great?) to confidently classify. These context dependent sentences are already difficult to discern, and since we purposively avoid context in this work, the model will battle with these utterances regardless of whether it knows the target, or it is simply 'unknown'.

The hashtags can clearly be seen as an issue. These are obviously OOV words with a precise connotation and attempting to learn a meaningful embedding would be difficult given its single use. Prior works have shown success in splitting a hashtag into its individual words for better understanding. Such a technique may suffer from discarding these single occurrence hashtags, as it could be more meaningfully modelled after splitting. However, since our preprocessing technique does not split hashtags, it is more practical to simply discard the infrequent ones, and rather allow the network to model the general polarity of the 'unknown' hashtags.

The censoring of vulgar language also presents potential issues. Consider the token ‘b*tch’, due to the asterisk it is an OOV word, whilst the intended word ‘bitch’ occurs in the embedding dictionary. This results in words unnecessarily being assigned random representations within the embedding space. Further, there is no rule on the number of asterisks used for replacing characters, as seen in ‘f**ckup’, two asterisks are used to replace a single character ‘u’. This results in many different variations of the same word. Consequently, we can see it is more feasible to simply replace these words with the ‘unknown’ token, rather than attempting to cater for such nuances.

Other categories were observed, such as ‘Names’, ‘Drugs’, ‘Web domains’, ‘Anatomy’ and ‘Numbers’, however the general analysis remains unchanged. These tokens are unique and don’t provide much insight into the sarcastic intent of the utterance without context and ultimately, as seen in the results above, the network performs better when replacing them with the ‘unknown’ token.

Another potential hypothesis for the increase in testing accuracy is the decrease in OOV words caused by a higher threshold. This results in only 44 OOV words when using the default minimum frequency of 3. Consequently, 882 fewer words are assigned random representations within the embedding space when compared to a threshold of 1. This decreased dependency on random assignment may be impacting the accuracy, however, analysis of such is left to future works.

From the above results, we can see that the preprocessing techniques used within this work are not detrimental to the model’s performance, but instead result in a small gain in testing accuracy. Furthermore, analysis was provided to try and understand the role of infrequently occurring words towards accurate sarcasm classification. Whilst it is not entirely clear why the model performs better when replacing these infrequent words, the goal was to establish whether the use of this ‘unknown’ token resulted in testing degradation.

Chapter 5

Conclusion

In this work deep convolutional networks were presented for the task of sarcasm classification. Three archetypes were investigated: a standard CNN, a ResNet and a DenseNet. Preliminary investigations were performed surrounding structural alterations (such as the pooling function, or depth) of each model to determine the configuration used in the final models for benchmarking. The observation from prior literature that deeper networks perform better on numerous NLP domain was extended to the task of sarcasm classification. The efficacy of each final model in accurate sarcasm detection on three English datasets were compared.

The first dataset, Headlines, was composed of newsheadlines taken from a satirical and a genuine news agency. The Headlines corpus was presented and benchmarked as a formal English dataset for future work in content-based detection. The other two datasets were taken from Reddit comments, with the Reddit Main set consisting of all subreddits, whilst the Reddit Pol dataset consisted of comments posted to the political subreddit. The performance of our models on both Reddit datasets were compared to recent approaches. The literature review demonstrated the increasing focus placed on extrinsic information, and the belief that content-modelling was insufficient for sarcasm detection on social media domains. The comparison of results showed that our dense model, DweNet, was capable of rivalling all but one network from recent literature. DweNet obtained 69% accuracy on the Reddit datasets, outperforming a variant of CASCADE which implemented context, user embeddings as well as discourse features. This is noteworthy since DweNet made no use of this context nor meta-data, suggesting that it is able to extract additional local information from the stand-alone utterance than previous studies have thought possible.

A case study was presented, identifying clear uses of sarcasm which the CNN misclassified but DweNet correctly classified. This supports the statement that DweNet was able to capture additional features intrinsic to the text. Hypothesising that the incorporation of low-level features in producing the final feature map played a key role in this, a study into the dependency between the final feature map and early feature maps was conferred. A visualisation of this dependency was presented to illustrate the influence of early features on the final feature map.

FastText's word embeddings were compared to GloVE on the Headlines dataset. This was performed in both a static and non-static manner to analyse the efficacy of each embedding for sarcasm detection. The results indicated GloVE was best suited for sarcasm classification, with the FastText embeddings resulting in weaker performance. The recall, precision and specificity of each model was detailed, with the results suggesting GloVE was the only embedding able to substantially increase sarcasm detection (measured by recall) through updating its representations. However, each embedding saw an increase in correctly classifying non-sarcastic sentences when using non-static representations. A comparison of the training sets and dimensions of GloVE and FastText showed GloVE had significantly less training and only a sixth of the dimension compared to FastText. Consequently, we hypothesised that FastText had the natural structure of English well encoded within its embedding, and battled to better align these with the semantics of sarcasm during training.

The effect of local context was also investigated on DweNet, with the parent comments from the Reddit datasets providing the context. It was found that DweNet was not suited for handling contextual information, and as a result, experienced weaker performance with its inclusion. This is contradictory to the literature which observed context always led to increased performance. However, context was included in a trivial manner and DweNet was clearly unable to take advantage of the additional information provided in such a way. This is unlike networks in recent literature which used more advanced methods of integrating context, often modelling it separate to the original comment entirely.

Finally, the efficacy of the data preprocessing methods were investigated, with results suggesting the default method was best suited. This was interesting as the default method discarded a considerable amount of information through the use of the unknown token. However, an analysis of the words which were cast to the unknown token showed that they were often specific words like proper nouns, or would present themselves as clear issues if they were not mapped to unknown, such as hashtags and censored vulgarities. The numerous rules used in the default preprocessing technique resulted in a marginal increase in performance, suggesting they are only beneficial in classifying a small subset of comments.

In conclusion, this study has shown that a content-based, densely connected convolutional network can compete with more complex models, which incorporate extrinsic information, on a social media dataset for sarcasm classification. This demonstrates that additional local cues are indeed present within an isolated utterance than previous approaches have identified.

In Appendix A we outline how the experiments reported on may be reproduced by the reader via the code and data made available on GitHub, as well as a demonstration on NextJournal.

5.1 Future Work

The following areas were identified for further investigation:

1. Incorporating extrinsic information, in a sufficient manner, with dense connectivity, to implement a model which is able to capture rich features from the content of the text, as well as handle nuanced uses of sarcasm which require background knowledge to confidently discern.
2. Studying the GloVE and FastText embeddings using statistical procedures such as PCA, t-SNE and UMAP, to better determine the extent to which the natural encoding of English within the representations affects sarcasm classification.
3. An analysis of the words which are randomly assigned representations within the embedding space, and the degree to which the random assignment affects sarcasm detection, for the different word embeddings.
4. The use of hyperparameter optimisation techniques to generate an optimal configuration for each of the final models.
5. Implementing transfer learning in conjunction with dense-connectivity. Trial experimentation with such a model yielded an accuracy of 92% on the Headlines dataset which is promising for extension work.
6. Locating suitable datasets to enable new applications such as racism and sexism detection.

Bibliography

[1] Jihen Karoui, Benamara Farah, Véronique Moriceau, Viviana Patti, Cristina Bosco, and Nathalie Aussenac-Gilles. Exploring the impact of pragmatic phenomena on irony detection in tweets: A multilingual corpus study. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 262–272, 2017.

[2] Bo Pang and Lillian Lee. Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval*, 2:1–135, 2007.

[3] Dmitry Davidov, Oren Tsur, and Ari Rappoport. Semi-supervised recognition of sarcastic sentences in twitter and amazon. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*, CoNLL '10, pages 107–116, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[4] Silvio Amir, Byron C. Wallace, Hao Lyu, Paula Carvalho, and Mário J. Silva. Modelling context with user embeddings for sarcasm detection in social media. In *CoNLL*, 2016.

[5] Ranjan Satapathy, Claudia Guerreiro, Iti Chaturvedi, and Erik Cambria. Phonetic-based microtext normalization for twitter sentiment analysis. *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 407–413, 2017.

[6] Soujanya Poria, Erik Cambria, Devamanyu Hazarika, and Prateek Vij. A Deeper Look into Sarcastic Tweets Using Deep Convolutional Neural Networks. *arXiv e-prints*, page arXiv:1610.08815, Oct 2016.

[7] Devamanyu Hazarika, Soujanya Poria, Sruthi Gorantla, Erik Cambria, Roger Zimmermann, and Rada Mihalcea. CASCADE: Contextual sarcasm detection in online discussion forums. pages 1837–1848, August 2018.

[8] Arthur Edwards. (how) do participants in online discussion forums create "echo chambers"?: The inclusion and exclusion of dissenting voices in an online forum about climate change. *Journal of Argumentation in Context*, 01 2013.

[9] Micah Altman, Alexandra Wood, David R O'Brien, and Urs Gasser. Practical approaches to big data privacy over time. *International Data Privacy Law*, 8(1):29–51, 03 2018.

[10] Sophie C. Boerman, Sanne Kruikemeier, and Frederik Zuiderveen Borgesius. Exploring motivations for online privacy protection behavior: Insights from panel data:. 2018.

[11] Åđadiye Deniz. Is somebody spying on us?: Social media users' privacy awareness. 2020.

[12] Emmanuel Ayaburi and Daniel N. Treku. Effect of penitence on social media trust and privacy concerns: The case of facebook. 2020.

[13] Lemi Baruh, Ekin Secinti, and Zeynep Cemalcilar. Online privacy concerns and privacy management: A meta-analytical review. 2017.

[14] Reza Ghaeini, Xiaoli Z. Fern, and Prasad Tadepalli. Attentional multi-reading sarcasm detection. *ArXiv*, abs/1809.03051, 2018.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[16] Joseph Tepperman, David Traum, and Shrikanth Narayanan. "Yeah Right": Sarcasm Recognition for Spoken Dialogue Systems. In *Interspeech 2006*, Pittsburgh, PA, September 2006.

[17] Paula Carvalho, Luís Sarmento, Mário J. Silva, and Eugénio de Oliveira. Clues for detecting irony in user-generated contents: Oh...!! it's "so easy" ;-). In *Proceedings of the 1st International CIKM Workshop on Topic-sentiment Analysis for Mass Opinion*, TSA '09, pages 53–56, New York, NY, USA, 2009. ACM.

[18] Roberto González-Ibáñez, Smaranda Muresan, and Nina Wacholder. Identifying sarcasm in twitter: A closer look. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers - Volume 2*, HLT '11, pages 581–586, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[19] Oren Tsur, Dmitry Davidov, and Ari Rappoport. Icwsm - a great catchy name: Semi-supervised recognition of sarcastic sentences in online product reviews. In *ICWSM*, 2010.

[20] Christine Liebrecht, Florian Kunneman, and Antal van den Bosch. The perfect solution for detecting sarcasm in tweets #not. In *Proceedings of the 4th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, pages 29–37, Atlanta, Georgia, June 2013. Association for Computational Linguistics.

[21] Roger J. Kreuz and Gina M. Caucci. Lexical influences on the perception of sarcasm. In *Proceedings of the Workshop on Computational Approaches to Figurative Language*, FigLanguages '07, pages 1–4, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics.

[22] Antonio Reyes, Paolo Rosso, and Tony Veale. A multidimensional approach for detecting irony in twitter. *Lang. Resour. Eval.*, 47(1):239–268, March 2013.

[23] Tomáš Ptáček, Ivan Habernal, and Jun Hong. Sarcasm detection on Czech and English twitter. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 213–223, Dublin, Ireland, August 2014. Dublin City University and Association for Computational Linguistics.

[24] Aditya Joshi, Pushpak Bhattacharyya, and Mark James Carman. Automatic Sarcasm Detection: A Survey. *arXiv e-prints*, page arXiv:1602.03426, Feb 2016.

[25] Ellen Riloff, Ashequl Qadir, Prafulla Surve, Lalindra De Silva, Nathan Gilbert, and Ruihong Huang. Sarcasm as contrast between a positive sentiment and negative situation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 704–714, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.

[26] Konstantin Buschmeier, Philipp Cimiano, and Roman Klinger. An impact analysis of features in a classification approach to irony detection in product reviews. In *Proceedings of the 5th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, pages 42–49, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

[27] Anupam Khattri, Aditya Joshi, Pushpak Bhattacharyya, and Mark James Carman. Your sentiment precedes you: Using an author's historical tweets to predict sarcasm. In *WASSA@EMNLP*, 2015.

[28] Aditya Joshi, Vaibhav Tripathi, Kevin Patel, Pushpak Bhattacharyya, and Mark Carman. Are word embedding-based features useful for sarcasm detection? In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1006–1011, Austin, Texas, November 2016. Association for Computational Linguistics.

[29] Aditya Joshi, Vinita Sharma, and Pushpak Bhattacharyya. Harnessing context incongruity for sarcasm detection. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 757–762, Beijing, China, July 2015. Association for Computational Linguistics.

[30] Duyu Tang, Furu Wei, Nan Yang, Ming Zhou, Ting Liu, and Bing Qin. Learning sentiment-specific word embedding for twitter sentiment classification. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1555–1565, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

[31] Francesco Barbieri, Miguel Ballesteros, and Horacio Saggion. Are emojis predictable? In *EACL*, 2017.

[32] Petra Kralj Novak, Jasmina Smailovic, Borut Sluban, and Igor Mozetic. Sentiment of emojis. In *PloS one*, 2015.

[33] Ben Eisner, Tim Rocktäschel, Isabelle Augenstein, Matko Bošnjak, and Sebastian Riedel. emoji2vec: Learning Emoji Representations from their Description. *arXiv e-prints*, page arXiv:1609.08359, Sep 2016.

[34] Ameeta Agrawal and Aijun An. Affective representations for sarcasm detection. In *The 41st International ACM SIGIR Conference on Research &#38; Development in Information Retrieval*, SIGIR '18, pages 1029–1032, New York, NY, USA, 2018. ACM.

[35] Meishan Zhang, Yue Zhang, and Guohong Fu. Tweet sarcasm detection using deep neural network. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 2449–2460, Osaka, Japan, December 2016. The COLING 2016 Organizing Committee.

[36] Byron C. Wallace, Do Kook Choe, Laura Kertz, and Eugene Charniak. Humans require context to infer ironic intent (so computers probably do, too). In *ACL*, 2014.

[37] David Bamman and Noah A. Smith. Contextualized sarcasm detection on twitter. In *ICWSM*, 2015.

[38] Byron C. Wallace, Do Kook Choe, and Eugene Charniak. Sparse, contextually informed models for irony detection: Exploiting user communities, entities and sentiment. In *ACL*, 2015.

[39] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 160–167, New York, NY, USA, 2008. ACM.

[40] Aniruddha Ghosh and Tony Veale. Fracking sarcasm using neural network. In *WASSA@NAACL-HLT*, 2016.

[41] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv e-prints*, page arXiv:1512.03385, Dec 2015.

[42] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2016.

[43] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. 2014. cite arxiv:1409.1556.

[44] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. pages 1097–1105, 2012.

[45] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level Convolutional Networks for Text Classification. *arXiv e-prints*, page arXiv:1509.01626, Sep 2015.

[46] Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann Lecun. Very deep convolutional networks for text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1107–1116, Valencia, Spain, April 2017. Association for Computational Linguistics.

[47] Shiyao Wang, Minlie Huang, and Zhidong Deng. Densely connected cnn with multi-scale feature attention for text classification. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, pages 4468–4474. AAAI Press, 2018.

[48] Warren Sturgis McCulloch and Walter F. Pitts. A logical calculus of the ideas immanent in nervous activity. 1990.

[49] Frank F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.

[50] John S. Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing*, pages 227–236. Springer Berlin Heidelberg, 1990.

[51] Les Toop, William Thorpe, and R Fright. Cough sound analysis: A new tool for the diagnosis of asthma? *Family practice*, 6:83–5, 07 1989.

[52] Oscar Bark, Andreas Grigoriadis, Jan Pettersson, Victor Risne, Adéle Siitova, and Henry Yang. A deep learning approach for identifying sarcasm in text. 2017.

[53] David Kriesel. *A Brief Introduction to Neural Networks*. 2007.

[54] K. Roebuck. *Data Quality: High-Impact Strategies - What You Need to Know: Definitions, Adoptions, Impact, Benefits, Maturity, Vendors*. Lightning Source, 2011.

[55] Charles Elkan. The foundations of cost-sensitive learning. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'01, pages 973–978, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[56] David White and Panos A. Ligomenides. Gannet: A genetic algorithm for optimizing topology and weights in neural network design. In *IWANN*, 1993.

[57] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv e-prints*, page arXiv:1609.04747, Sep 2016.

[58] Yann N. Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 2933–2941, Cambridge, MA, USA, 2014. MIT Press.

[59] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[60] Tom Fawcett. An introduction to roc analysis. *Pattern Recogn. Lett.*, 27(8):861–874, June 2006.

[61] David Powers and Ailab. Evaluation: From precision, recall and f-measure to roc, informedness, markedness and correlation. *J. Mach. Learn. Technol*, 2:2229–3981, 01 2011.

[62] Charles Parker. On measuring the performance of binary classifiers. *Knowledge and Information Systems*, 35, 04 2012.

[63] Antonio Gulli. *Deep Learning with Keras: Implementing deep learning models and neural networks with the power of Python*. Packt Publishing, apr 2017.

[64] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA, 2010. Omnipress.

[65] David Sussillo and L. F. Abbott. Random Walk Initialization for Training Very Deep Feedforward Networks. *arXiv e-prints*, page arXiv:1412.6558, Dec 2014.

[66] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *AISTATS*, 2011.

[67] Hugh Murrell and Nando de Freitas. *Deep learning notes using julia with flux*. 2019.

[68] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

[69] Andrew L. Maas. Rectifier nonlinearities improve neural network acoustic models. 2013.

[70] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[71] Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1058–III–1066. JMLR.org, 2013.

[72] A. Dorugade and Dattatraya Kashid. Alternative method for choosing ridge parameter for regression. *Applied Mathematical Sciences (Ruse)*, 4, 01 2010.

[73] Anders Krogh and John A. Hertz. A simple weight decay can improve generalization. In *Proceedings of the 4th International Conference on Neural Information Processing Systems*, NIPS'91, pages 950–957, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

[74] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017.

[75] Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. *arXiv e-prints*, page arXiv:1711.05101, Nov 2017.

[76] Guodong Zhang, Chaoqi Wang, Bowen Xu, and Roger Grosse. Three Mechanisms of Weight Decay Regularization. *arXiv e-prints*, page arXiv:1810.12281, Oct 2018.

[77] Arvind Neelakantan, Luke Vilnis, Quoc V. Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. Adding Gradient Noise Improves Learning for Very Deep Networks. *arXiv e-prints*, page arXiv:1511.06807, Nov 2015.

[78] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv e-prints*, page arXiv:1207.0580, Jul 2012.

[79] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.

[80] Erion Çano and Maurizio Morisio. Word Embeddings for Sentiment Analysis: A Comprehensive Empirical Survey. *arXiv e-prints*, page arXiv:1902.00753, Feb 2019.

[81] Stephen Robertson. Understanding inverse document frequency: on theoretical arguments for idf. *Journal of documentation*, 60(5):503–520, 2004.

[82] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, March 2003.

[83] Lei Zhang, Shuai Wang, and Bing Liu. Deep learning for sentiment analysis: A survey. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*, 8, 2018.

[84] Rami Al-Rfou, Bryan Perozzi, and Steven Skiena. Polyglot: Distributed Word Representations for Multilingual NLP. *arXiv e-prints*, page arXiv:1307.1662, Jul 2013.

[85] Paramveer S. Dhillon, Dean Foster, and Lyle Ungar. Multi-view learning of word embeddings via cca. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, pages 199–207, USA, 2011. Curran Associates Inc.

[86] Tom Kenter and Maarten de Rijke. Short text similarity with word embeddings. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, pages 1411–1420, New York, NY, USA, 2015. ACM.

[87] Matt J. Kusner, Yu Sun, Nicholas I. Kolkin, and Kilian Q. Weinberger. From word embeddings to document distances. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 957–966. JMLR.org, 2015.

[88] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *arXiv e-prints*, page arXiv:1301.3781, Jan 2013.

[89] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. *arXiv e-prints*, page arXiv:1310.4546, Oct 2013.

[90] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *EMNLP*, 2014.

[91] Franco Lancia. Word co-occurrence and similarity in meaning. 01 2007.

[92] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching Word Vectors with Subword Information. *arXiv e-prints*, page arXiv:1607.04606, Jul 2016.

[93] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of Tricks for Efficient Text Classification. *arXiv e-prints*, page arXiv:1607.01759, Jul 2016.

[94] Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. Advances in pre-training distributed word representations. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 2018. European Language Resources Association (ELRA).

[95] Yann LeCun, LÃľon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998.

[96] David H. Hubel and Torsten Nils Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160:106–54, 1962.

[97] S. Lawrence, C.L. Giles, Ah Chung Tsoi, and A.D. Back. Face recognition: a convolutional neural-network approach. *Neural Networks, IEEE Transactions on*, 8(1):98–113, January 1997.

[98] Dan C. Cireşan, Ueli Meier, Jonathan Masci, Luca M. Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, IJCAI'11, pages 1237–1242. AAAI Press, 2011.

[99] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. volume 38, pages 295–307, Washington, DC, USA, February 2016. IEEE Computer Society.

[100] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 655–665, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

[101] Cícero dos Santos and Maíra Gatti. Deep convolutional neural networks for sentiment analysis of short texts. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 69–78, Dublin, Ireland, August 2014. Dublin City University and Association for Computational Linguistics.

[102] Ossama Abdel-Hamid, Abdel-Rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. Convolutional neural networks for speech recognition. *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, 22(10):1533–1545, October 2014.

[103] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard Zemel. Understanding the Effective Receptive Field in Deep Convolutional Neural Networks. *arXiv e-prints*, page arXiv:1701.04128, Jan 2017.

[104] Jake V. Bouvrie. Notes on convolutional neural networks. 2006.

[105] Zhou and Chellappa. Computation of optical flow using a neural network. pages 71–78 vol.2, July 1988.

[106] Sparsh Mittal. A survey of fpga-based accelerators for convolutional neural networks. *Neural Computing and Applications*, 09 2018.

[107]

[108] Alon Jacovi, Oren Sar Shalom, and Yoav Goldberg. Understanding convolutional neural networks for text classification. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 56–65, Brussels, Belgium, November 2018. Association for Computational Linguistics.

[109] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 6(2):107–116, April 1998.

[110] Guido Montúfar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the Number of Linear Regions of Deep Neural Networks. *arXiv e-prints*, page arXiv:1402.1869, Feb 2014.

[111] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *CoRR*, abs/1312.4400, 2013.

[112] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015.

[113] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *CoRR*, abs/1605.07648, 2017.

[114] Rishabh Misra. News headlines dataset for sarcasm detection, 06 2018.

[115] Mikhail Khodak, Nikunj Saunshi, and Kiran Vodrahalli. A Large Self-Annotated Corpus for Sarcasm. *arXiv e-prints*, page arXiv:1704.05579, Apr 2017.

[116] Michael Cogswell, Faruk Ahmed, Ross B. Girshick, C. Lawrence Zitnick, and Dhruv Batra. Reducing overfitting in deep networks by decorrelating representations. *CoRR*, abs/1511.06068, 2015.

[117] Leslie N. Smith. Cyclical learning rates for training neural networks. *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472, 2015.

[118] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.

Appendix A

Model Code and Demonstration

This appendix contains detailed information for obtaining the model source code, as well as the jupyter notebooks to run the models. All code, results and notebooks can be obtained from our GitHub repository: https://github.com/Dev-data-fun/D-D-Sarcasm. For readers who wish to see a demonstration of the benchmarks and do not have the required hardware, an online demonstration is provided through the NextJournal[1] platform: https://nextjournal.com/djpelser/Benchmark_demo. Readers must make an account on NextJournal and 'Remix' the notebook to run it.

A.1 Setup

Local Setup

This work was done on an Ubuntu 18.04.03 system using a GeForce RTX2080 GPU for processing.

The following package versions were used: Python V3.7.1, Pytorch V1.0.1, FastAI V1.0.50, Cuda V10.1.105 and CuDNN V7.5.0.

Project Structure

The complete project is 324 MB. Below is the folder tree for the project:

[1] https://nextjournal.com/

```
Sarcasm
├─ Datasets
│  ├─ Headlines
│  │  └─ Embeddings
│  ├─ Reddit Main
│  │  ├─ Embeddings
│  │  └─ Parent
│  │     └─ Embeddings
│  └─ Reddit Pol
│     └─ Embeddings
├─ Models
│  ├─ CNN_baseline
│  ├─ DweNet
│  └─ ResNet_baseline
└─ Results
   ├─ Benchmarks
   │  ├─ Case Study
   │  ├─ Heatmap
   │  └─ Model Benchmarks
   │     ├─ Baseline CNN
   │     ├─ Baseline ResNet
   │     └─ DweNet
   ├─ Data Preprocessing Investigation
   ├─ Depth Investigation
   ├─ Embedding Investigation
   ├─ Growth Investigation
   └─ Local Context Investigation
```

A.2 Data, Models and Results

Datasets

The three datasets are located under 'Sarcasm/Datasets/', they are:

1. Headlines
2. Reddit Main
3. Reddit Pol

Within each subfolder is an 'Embeddings' folder containing the weight matrix corresponding to the word embeddings for each respective dataset. The 4 available word embeddings are:

1. 50D GloVe embeddings
2. 300D FastText1M embeddings
3. 300D FastText1M subword embeddings
4. 300D FastText2M embeddings

Models

The three models implemented in our work are located under 'Sarcasm/Models/', they are:

1. CNN baseline
2. ResNet baseline
3. DweNet

Each folder contains the source code of the final models, as well as a jupyter notebooks (.pynb files) for loading, training and testing the models. Performance is presented in accuracy and F1 score.

Results

The results of each experiment, as well as data used to analyse findings are contained within 'Sarcasm/Results/'. The experiments performed were:

1. Benchmarking - Performance of each model, on each dataset

2. Data Preprocessing - Effect of different data preprocessing
3. Depth Investigation - Effect of differing depths
4. Embeddings Investigation - Effect of different embeddings, using static and non-static representations
5. Growth Investigation - Effect of the growth rate in DweNet
6. Local Context Investigation - Effect of local context

A.3 Running the Models

Instructions for running the benchmarks and exploratory investigations in this thesis are listed below.

Benchmarks

To run the benchmark, open the jupyter notebook corresponding to the model you wish to run. 7 variables must be set:

1. PADDING - 64 for Headlines dataset, 128 for Reddit Main and Reddit Pol
2. DATASET_PATH - Path to folder containing data: '.../Datasets/Headlines/' for Headlines folder.
3. DATASET - Dataset file name: 'Headlines.csv' for Headlines folder.
4. COL - Column to load data from csv: 'headline' for Headlines, 'comment' for Reddit Main.Pol
5. WEIGHTS - Path to word embeddings:
   '.../Datasets/Headlines/Embeddings/glove/Weights_glove_headlines.pkl'
   for GloVE embeddings for Headlines data
6. EMBED_DIM - Dimension of word embedding: 50 for GloVe, 300 for FastText
7. STATIC - Static or Non-static embeddings: Non-static leads to better performance

Run the code cells in the notebook to run the dataset with the chosen embedding. Results will be displayed in terms of accuracy and F1 score.

Data Preprocessing

To compare benchmark techniques use the embeddings located under 'Sarcasm/Results/Data Preprocessing Investigation/'. The following 2 options are available:

1. Min 1 - Headlines word embedding for a word threshold of 1
2. Min 2 - Headlines word embedding for a word threshold of 2

Note: Only the Headlines dataset is available for this investigation. To use the default preprocessing simply load the standard Headlines embedding.

The following change will need to be made to the jupyter notebook:

1. Edit code cell 5 as such:

```
processor = [TokenizeProcessor(tokenizer=tokenizer),
             NumericalizeProcessor(min_freq=X)]
```

   where X is the Min frequency embedding chosen.

To evalute the performance of the pre and postprocessing rules, make the following change to the jupyter notebook:

1. Edit code cell 5 as such:

```
tokenizer = Tokenizer(SpacyTokenizer, 'en', pre_rules=[],
                      post_rules=[])
```

Embedding Investigation

To compare the different word embeddings available, select the jupyter notebook corresponding to the model you wish to use. The 3 FastText embeddings are available under the Embeddings folder for each dataset.

Make sure to set the EMBED_DIM variable to 50 for GloVe, and 300 for FastText.

To use compare static vs non-static representations, set the STATIC variable accordingly.

Local Context Investigation

To evaluate the effect of local context - provided in the form of parent comments on the Reddit Main dataset - set WEIGHTS to the Parent word embedding for the Reddit Main dataset located under: 'Sarcasm/Datasets/Reddit Main/Parent/Embeddings/glove/'. Then make the following changes to the jupyter notebook:

1. Edit code cell 5 as such:

```
processor = [TokenizeProcessor(tokenizer=tokenizer,
        mark_fields=True), NumericalizeProcessor()]
```

2. Edit code cell 6 as such:

```
data = (TextList.from_csv(DATASET_PATH, DATASET,
        cols=[COL, 'parent'], processor=processor))
```

This will load the parent comment and concatenate it to the child comment, separated by the 'xxfld' token.

Note: Only the Reddit Main dataset is available for this investigation.

A.4 Potential Issues

If the model returns the following error:

```
RuntimeError: cuda runtime error (59)
```

It is likely caused by words in the vocabulary that are not present in the embedding (this can result from using a newer version of fastai). To fix this, save the vocab of the data once the data has been instantiated as such:

```
pickle.dump(data.vocab.itos, open(‘path/to/file/filename.pkl', 'wb))
```

Then download the desired embedding from its respective site, and build the weight matrix using the following:

```
weights_matrix = np.zeros((len(itos), EMBED_DIM))

def gather(emb_dict):

  global weights_matrix

  for i, word in tqdm(enumerate(itos)):

    if i==1:
      continue        # We skip the <pad> token, leaving it to be zeros

    try:
      weights_matrix[i] = list(emb_dict[word])

    except KeyError:
      weights_matrix[i] = np.random.normal(scale=0.6, size=(EMBED_DIM,))
```

Then pass the embedding file to the ’gather’ function and save the resulting ’weights_matrix’ object.