Final Project Report (FPR)

Final Project Report (FPR)

Student Number: 20067078 Student Name: Edwin Kpakpo Allotey-Acquaye

Course: MSc. Data Science and Analytics (Advanced Research) 7COM1039-0109-2022 - Advanced Computer Science Masters Project

Supervised by: Mohammadhassan Tayaraninajaran

Report Type: Final Project Report

Sentiment Analysis using BERT-DenseNet on Textual Images

August 2023

MSc Final Project Declaration

This report is submitted in partial fulfillment of the requirement for the degree of Master of Science in MSc. Data Science and Analytics at the University of Hertfordshire (UH).

I, Edwin Kpakpo Allotey-Acquaye confirm that the work presented in this thesis is my own.

No part of this thesis was previously presented for another degree at this or any other institution.

Information that has been derived from other sources has been indicated in this thesis.

Acknowledgment

I would like to express a heartfelt gratitude to Edwin Allotey Acquaye, my father, Esther Allotey Acquaye, my mother, and my sisters, Florence, and Eileen Allotey-Acquaye for always believing in me and offering the purest form of love and support throughout the course of this degree.

A big thank you to my brother, Boma Thankgod-Ibulubo, for always availing yourself to help both academically and socially. I appreciate you.

I greatly appreciate the trust and belief in me from My supervisor, Mohammadhassan Tayaraninajaran who contributed to my research by encouraging me to embark on this research.

Finally, I would like to thank my friend, Harpreet Singh for his profound support during this research. Allowing me to bounce ideas off his highly intellectual mind.

Table of Contents

Abstract

Sentiment analysis is a natural language processing technique that involves determining the sentiment or emotional tone expressed in text. It aims to classify text as positive, negative, neutral, or sometimes more nuanced sentiments. Sentiment analysis has widespread applications, including social media monitoring, customer feedback analysis, and market sentiment assessment. Techniques range from rule-based systems to advanced machine learning models like neural networks and transformers. Preprocessing, feature extraction, and labeled training data play key roles. Sentiment analysis helps businesses and researchers understand public sentiment, make informed decisions, and tailor communication strategies based on the emotional content of text data. In this report, we have implemented a Novel technique called BERT-DeseNet which is an extension of an unconventional natural language technique method known as Text image classification. Experiments were conducted on the IMDB movie review dataset to determine the accuracy of this architecture. Additional experiments were conducted to determine whether sentence length influenced the accuracy of the classifications. This research contained promising results to pave the way for Densenet architecture to be used for NLP tasks.

Chapter 1: Introduction

In recent times, the sharing of information has never been easier. The vast use of the internet and portable devices has given users the opportunity to share a huge amount of information easily and quickly so a vast number of platforms. This has also given the opportunity for sentiment analysis to show its power as this is a means of examining text. Sentiment analysis is the use of natural language processing, Machine learning, and computational linguistics techniques to identify, extract, and analyze emotions, attitudes, and opinions in expressed or spoken language (Mejova, 2009). Sentiment analysis has a wide range of applications. With the widespread use and development of the internet, the development of internet technologies has seen the explosive use of many platforms for the use of business, news, social interactions and films, stock market prediction, depression diagnosis, and real-time election analysis. Getting information from data simply involves users checking reviews on films as a way of guiding their choices on things they want to buy or what they want to watch. Users might just have to take advantage of their access to the internet to acquire large amounts of relevant information at their fingertips as well as share their opinions, feelings, and attitudes on the internet. The ability to share opinions and reviews was greatly influenced by the success of services, goods, events, organizations, etc. Comments or evaluations are collected by these individual platforms for the analysis of user satisfaction and sentiments. The kind of data collected matches the organizational requirement. For instance, electronic business platforms collect reviews from users to improve products and services.

The analysis of individual comments and reviews by humans is impossible due to the amount of data that has to be analyzed. Thus, the use of a sentiment analysis system is needed for the classification of user feedback as positive or negative (Salur and Aydin, 2020). The increase in demand in many fields has rendered sentiment very useful as it involves the development of systems that analyze and summarize the opinions of users from an unstructured review document. The goal of sentiment analysis is to determine the overall sentiment of a piece of text, such as a review, tweet, or news article, and classify it as positive, negative, or neutral. Sentiment analysis at a general level can be classified into lexicon-based and machine learning-based (Prakash et al., 2020). The machine learning approach uses learning algorithms and classifier models that are trained and tested on a known dataset whereas the lexicon-based approach uses the calculation of sentiment polarity from the dictionaries of annotated words with sentiment scores.

Traditionally in neural networks, for sentiment analysis classification, text data is tokenized into smaller units (words or subwords), and then converted into numerical representations using word embeddings like Word2Vec or GloVe. These embeddings capture semantic meaning and context. Machine learning models, like recurrent neural networks (RNNs), more recently, attention-based transformers, learn from these embeddings to classify sentiment. Evaluation metrics such as precision, recall, and F1-score measure model performance

Deep learning revolutionized sentiment analysis by automatically learning intricate patterns in text data. Convolutional Neural Networks (CNNs) are key players, leveraging filter layers to capture spatial features like n-grams and sentiment indicators. CNNs excel in feature extraction, identifying relevant word combinations for sentiment classification. By hierarchically analyzing text at various scales, they grasp both local nuances and broader context. These models require minimal manual feature engineering and adapt well to diverse

data. In essence, CNNs' ability to extract spatial features from text significantly boosted sentiment analysis accuracy and efficiency, making them pivotal in the field's success.

Recently, CNNs have been adopted to aid in sentiment analysis classification tasks. These have however been limited to one-dimensional convolutions. (Singh *et al.*, 2022a) explored the use of 2D CNN Resnet architecture for sentiment analysis classification tasks. This served as the inspiration for the research in this paper. The research in this paper focuses on using a 2D CNN Densenet architecture for sentiment analysis classification. In this process, word embedding matrices are transformed into greyscale images with a BLSTM cell (Singh *et al.*, 2022a). The generated greyscale textual images were then classified using 2D Resnet architecture, conventionally used for computer vision tasks.

1.1 Background

In the era of extensive online communication, sentiment analysis emerges as a vital tool for comprehending the sentiments hidden within vast volumes of text data. This subfield of natural language processing (NLP) aims to automatically categorize text as positive, negative, or neutral, holding great significance across domains. While traditional approaches relied on rule-based methods, the rise of deep learning, notably Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), revolutionized sentiment analysis. These architectures excel in capturing intricate linguistic nuances and patterns, surpassing earlier methods in accuracy and efficiency.

Our sentiment analysis classification report delves into the evaluation of these advanced models. By incorporating a state-of-the-art attention mechanism like BERT (Bidirectional Encoder Representation for transformers) in textual image classification, we comprehensively assess the models' performance. The DenseNet architecture is chosen to play a part in this process. However, Along the line, other DenseNet architectures will be considered and tested to identify the best-performing architecture for the sentiment analysis task being proposed.

This report not only highlights the success of deep learning techniques, particularly CNNs, in enhancing sentiment analysis accuracy but also unravels their limitations. As we navigate through this analysis, we aim to shed light on how these models have reshaped sentiment analysis, paving the way for deeper insights into public opinion, customer satisfaction, and other critical areas where understanding sentiment is paramount.

1.1.2 Hypothesis

- BERT-Densenet architecture cannot classify data for sentiment analysis.
- Sentence length does not impact accuracy.

1.1.3 Research Questions

The use of BLSTM on textual data for sentiment analysis has revolutionized the way analysis is conducted using text (Singh *et al.*, 2022b). This process has proved a useful way of analyzing text but has also opened the doors for a more accurate of conducting this task.

For this study we have posed the following research questions:

- Can BERT-DenseNet architecture classify textual data for sentiment analysis?

- Does sentence length impact the accuracy of classification?

- Does dataset size impact accuracy?

'Can a BERT-DenseNet model produce accurate sentiment analysis results when used on textual images?'. This research aims to dive deeper into the quest to find answers and solutions to the problem of making this process better and more accurate.

1.2 Aims and Objectives

Sentiment analysis is being appreciated and introduced to our everyday lives more and more through systems that are used daily. It is only fair for these systems to perform better at their tasks. This research was conducted with the following objectives in mind:

- To determine whether a BLSTM-Densenet model is more accurate than existing models.
- To determine the best-performing DenseNet architecture that would achieve higher accuracy when paired with the BERT model for sentiment analysis.
- To determine whether dataset size has an effect on accuracy.

1.3 Scope of Research

This research covers the use of BLSTM models and DenseNet architectures. The addition of other DenseNet architectures was introduced into this research to determine the architecture with the best performance for sentiment analysis.

1.4 Justification of Research

Using DenseNet for sentiment analysis offers feature reuse, gradient flow stability, and deep representations. Its architecture captures linguistic nuances efficiently, benefiting nuanced sentiment understanding. Regularization and parameter efficiency aid in working with limited data. Transfer learning from image domains and ensemble approaches further enhance results. Experimentation remains essential to find the best architecture for specific sentiment analysis tasks.

1.5 Potential Ethical, Legal, Professional, and Social Issues Consideration

Sentiment analysis tasks present the opportunity for businesses and organizations to gain valuable insights, make data-driven decisions, enhance customer experiences, and respond more effectively to public sentiment. To ensure that these tasks are carried out responsibly, it is necessary to address the legal, ethical, professional, and social issues that might be faced with the sentiment analysis tasks.

1.5.1 Legal Considerations:

- Data Privacy and Protection**:** When undertaking sentiment analysis tasks, it is important for individuals and organizations to ensure that the data used is collected and processed in accordance with data protection laws, such as the General Data Protection Regulation

(GDPR) in the European Union or the California Consumer Privacy Act (CCPA) in the United States. Appropriate consent must also be obtained from data subjects if required.

- Data Ownership and Consent**:** The ownership rights of the data being used should be clearly defined for the data being used for sentiment analysis. It must be ensured that professionals have the legal right to use the data and consent from the data owners should be considered when necessary.

- Use of Third-Party Data**:** When using third-party data sources for sentiment analysis, it must be ensured that the right to access and use the data for the specific purpose of the task is obtained while respecting the terms of use or any licensing agreements associated with the data.

- Intellectual Property: Copyright and intellectual property rights when using textual data for sentiment analysis should be respected. It is the responsibility of individuals or professionals to ensure that the legal right to use and analyze the text data is obtained before use for specific tasks.

1.5.2 Ethical Considerations:

- Bias and Fairness: It is extremely important to identify biases in the sentiment analysis model and data. Biased models can perpetuate unfair stereotypes and influence decisions in ways that are discriminatory. Training data and model predictions should be regularly evaluated to prevent bias and encourage fairness for the specified tasks.

- Avoid Manipulation: It is extremely important to be cautious of using sentiment analysis to manipulate public opinion or artificially influence sentiment. The misuse of sentiment analysis for political, commercial, or malicious purposes can lead to misinformation. The tasks should be carried out responsibly and ethically, avoiding manipulation or misinterpretation while being transparent about the use of the results of the tasks.

- Long-Term Impact: The long-term impact of sentiment analysis results on individuals and society should be thoroughly considered. Misclassifications or misinterpretations of the tasks can have far-reaching consequences. The outputs of these tasks should be monitored over time to ensure their accuracy and relevance, while continuously evaluating the potential societal impact of the results.

- Accountability: The limitations of sentiment analysis should be clearly stated. The capabilities of the tools used should not be overstated. Responsibilities for the errors that may arise should be taken up by professionals or individuals involved in the sentiment analysis tasks. Mechanisms should be established for users to report concerns and receive support.

1.5.3 Professional Considerations:

- Human Validation: Despite automation, the validation of the results of sentiment analysis tasks by humans is crucial, especially for critical decisions. This helps to identify the nuances that may have been overlooked by automated systems. Human validation or verification should be employed for critical decisions and sensitive contexts while developing mechanisms to escalate cases that the models cannot confidently classify.

- Handling Neutral Sentiments: Proper considerations of how to handle neutral or mixed sentiment cases should be made as these may be challenging to classify accurately. It is crucial to determine the threshold neutrality. A threshold for classifying sentiment as neutral or mixed should be determined based on the use case of the application.

- Documentation: The process of sentiment analysis should be thoroughly documented, including the data sources, preprocessing steps, model configurations, and evaluation metrics. Professionals should maintain detailed documentation of the entire process for transparency and reproducibility.

- Cultural Sensitivity: Professionals should be mindful of cultural differences in sentiment expression. Different communities may express sentiment in diverse ways. This is very important to take into consideration during the sentiment analysis task. The use of diverse and representative datasets that include sentiments expressed in various cultural contexts should be considered while allowing the possibility of accepting feedback from users and stakeholders to ensure cultural variations.

- Competence: It should be ensured that the necessary skills and expertise are possessed by professionals involved in the sentiment analysis task in order to carry out the task accurately and responsibly. The right man should be used for the right job.

1.5.4 Social Considerations:

- Public Awareness: Public awareness should be increased about sentiment analysis to encourage informed discussions about its benefits and challenges. The benefits, limitations, and potential risks of sentiment analysis should be promoted through public awareness campaigns.

- Regulation and Guidelines: Compliance with relevant regulations, ethical guidelines, and industry best practices when conducting sentiment analysis should be taken seriously, especially when working with sensitive data. It is crucial to adhere to relevant regulations and industry ethical guidelines when collecting and analyzing sentiment data.

- Educational Use: Users should be educated on how sentiment analysis works and its limitations to prevent undue reliance on its outcomes. Clear guidelines should be provided on how to interpret sentiment analysis results.

- Emotional Well-Being: Sentiment analysis might detect negative emotions or mental health concerns. It should be considered to offer appropriate resources or support to users who express distressing sentiments. Appropriate resources should be provided to reach out when negative sentiments indicative if distress is detected. Assumptions about an individual's well-being based solely on sentiment scores should be avoided.

1.6 Organization of Subsequent Sections of this Research Report

In the following chapter, the literature relevant to this research is conducted. The models, architectures, as well as performance of the models, are evaluated. The following chapters discuss the experimentation process and the fine tunes and tweaks that were made to these models to achieve optimum performance. The final chapter holds information on the experimental results, challenges, recommendations, shortcomings, and conclusions of this research while talking about the ways in which this research could be conducted in the future to achieve better and more accurate results.

CHAPTER 2: Literature Review

2.1 Overview

(Kim et al., 2004) describe sentiment as 'the affective part of opinions'. This can be related to the emotion, behind what words mean. The understanding of sentiment is ideal for the proper understanding of communication between humans. In the current age and time, as communication has reached an ultimately high level, it has called for a deeper understanding of the meanings of communication between different parties. The quest for this understanding has brought about the need for sentiment analysis.

The literature review section of the sentiment analysis classification task encompasses a comprehensive journey through the evolution of sentiment analysis methodologies. Initially, sentiment analysis relied on lexical analysis (Ramchand et al., 2002) and rule-based systems (Yao et al., 2019), which although insightful, struggled with context complexities. The paradigm shifted with the rise of machine learning techniques, like SVMs (Ahmad et al., 2017) and Naive Bayes (Dey *et al.*, 2016), yielding improved results. However, the real breakthrough emerged with the advent of deep learning, notably CNNs (Singh *et al.*, 2022a) and RNNs (Tang et al., 2015). CNNs excelled in capturing local features through filter layers, while RNNs adeptly dealt with sequential data and temporal dependencies. Pre-trained models like BERT (Hoang et al., 2019) and GPT (Barbon et al., 2022) introduced transfer learning to sentiment analysis, demonstrating impressive generalization capabilities. Yet, challenges like sarcasm detection and cultural nuances persist. This literature review contextualizes the study within this continuum of advancements, offering insights into various methodologies, strengths, and ongoing research directions. Its aim is to contribute to the enhancement of sentiment analysis techniques, underscoring the transformative impact of deep learning and pre-trained models.

2.2 Domain knowledge

Domain knowledge could be defined as a helpful tool for limiting or pruning the search space for a discovery process, which improves system efficiency (Anand et al., 1995). The knowledge of the right tools and processes that need to be used to determine the best possible output from a model is extremely important in the quest for discovering what works for a particular experiment. With the classification of sentiment analysis, there are several options to be considered for the optimum performance of the models to find the best-performing model to be used.

2.3 Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNNs) are a class of artificial neural networks designed for processing sequences of data. Unlike traditional feedforward neural networks, where data flows in one direction from input to output, RNNs have connections that loop back on themselves, allowing them to maintain an internal state that captures information about previous elements in the sequence (Medsker et al., 2001). In (Singh *et al.*, 2022b)'s research, the combination of an RNN and CNN was used to perform sentiment analysis on images produced from textual data contained in tweets. The RNN was used to extract time dependency from the input sequence. Though effective, (Li et al., 2016) discovered that the RNN language model was unable to cover the time order structure of the entire text and handle long-term dependencies. RNNs are particularly suited for tasks involving sequential data, such as time series analysis, natural language processing, speech recognition, etc. The key feature of RNNs is their ability to maintain a hidden state that can capture dependencies and patterns across different time steps or positions in a sequence.

2.4 Convolutional Neural Networks (CNNs)

CNNs are a specialized breed of neural networks specifically designed for processing grid-like data (Yamashita *et al.*, 2018), most notably images. At their core, CNNs wield convolutional layers that scan input data with learnable filters, capturing intricate patterns and features that form the building blocks of visual information. (Ouyang *et al.*, 2015) adopted the use of CNNs to extract local features from the data. This feature extraction enables models to recognize sentiment-indicative phrases, words, and structures, enhancing the accuracy and performance of sentiment analysis classifications. Coupled with pooling layers that strategically downsample data, CNNs showcase their adeptness at preserving vital features while reducing computational load. The implications of CNNs extend across various domains, finding their zenith in computer vision tasks. From image classification, where they ascertain the content of images, to object detection, where they not only identify but also pinpoint objects' locations, CNNs have revolutionized our capacity to process visual data. Their hierarchical architecture enables the networks to grasp the nuances of visual information, progressively learning from basic to sophisticated features.

2.4.1 2D CNNs

Convolutional Neural Networks (CNNs) shine in image analysis, with 2D CNNs excelling in spatial data. Using convolutional and pooling layers, they detect patterns and features, transcending pixels. Their translational invariance enables recognizing features regardless of location. (Wang et al., 2022) used a multi-layer 2D CNN for extraction and mining of fine-grained information from texts to gain deeper insights and understanding of context. This vastly improved the capability of the classification model being used. (Singh *et al.*, 2022b) also used a 2-dimension CNN model to extract information from textual images as this was proposed to be highly effective for feature extraction from the images after they were generated by BLSTM and reshaped into greyscale images.

2.5 DenseNet

DenseNet is a neural network architecture designed to address the challenges of vanishing gradients and feature reuse in deep networks (Yang *et al.*, 2018). It introduces the concept of

"dense connectivity," where each layer receives inputs not just from the previous layer, but also from all preceding layers (Nugroho et al., 2020). This promotes extensive feature reuse, enhancing gradient flow and information propagation. DenseNet is built around dense blocks, comprising multiple layers grouped together. In a dense block, the output of each layer is concatenated with the outputs of all previous layers, creating a dense connection. This approach enhances feature extraction and allows the network to learn both local and global patterns effectively (Takahashi et al., 2021). Transition layers with pooling and dimensionality reduction are added between dense blocks to manage network complexity. The global average pooling and a classification layer complete the architecture. DenseNet's compactness, efficient memory usage, and ability to learn rich features make it popular for various computer vision tasks, contributing to improved model performance with fewer parameters.

2.6 Transformers

A transformer model is a neural network that discovers context and subsequently meaning by tracing relationships in sequential data, such as the words in this phrase. (Merritt, 2022). Transformers are particularly known for their effectiveness in capturing long-range dependencies in sequences and their parallelizable architecture, which makes them highly efficient for training on modern hardware (Borzunov *et al.*, 2022). (Cheng *et al.*, 2021) explored the use of multimodal Sparse Phased Transformer (SPT) to generate sparse attention matrices and to compress long sequences to a shorter sequence of hidden states. This model paired with parameter sharing was able to capture multimodal interactions with a reduction in model size and improved sample efficiency. Their research backed the use of Transformers for sentiment analysis tasks as it deemed transformers highly efficient in capturing relationships in sequence data. (Shangipour et al., 2020)'s research was performed using BERT, a transformer-based model to determine whether a sentence passed was sarcastic or not. This research agrees with the fact that transformers are highly powerful models that efficiently capture complex patterns and relationships in data and are highly adaptable to NLP tasks.
(Kalyan, et al., 2021) investigated the use of BERT, a transformer-based model for pretraining. Transformers offer advantages in NLP pretraining by capturing context, handling dependencies, and enabling transfer learning. They reduce feature engineering, support multilingual tasks, and excel in text generation, achieving state-of-the-art performance across various tasks. This versatility, scalability, and availability of pre-trained models make them a highly efficient choice in NLP research and applications.

Transformers provide a solution to the transfer learning problem faced by traditional models (Li *et al.*, 2021). Transformers address transfer learning in NLP through a dual-phase process. Initially, they pre-train on extensive text data, learning linguistic structures and context through masked or autoregressive language modeling. This produces a language model with comprehensive language understanding. Then, in the fine-tuning phase, this pre-trained model is adapted to specific tasks using smaller labeled datasets.

2.7 Bidirectional Attention Networks (BANs)

In natural language processing (NLP), the pursuit of comprehending the intricate relationships between words in a sentence has been a relentless endeavor. This pursuit has led to the emergence of innovative neural network architectures, with Bi-directional Attention Networks (BANs) standing as a prime example of a transformative approach that leverages bidirectional

interactions to capture both local and global contextual nuances in text (Aich *et al.*, 2021). For sentiment analysis tasks, position information is highly important as it helps to understand context. (Gu *et al.*, 2018) proposed a bidirectional attention network based on a bidirectional GRU which uses a bidirectional attention mechanism to concentrate on the positional information of aspect terms as well as the relationship between aspect terms and sentences. By synergizing these attention mechanisms, BANs cultivate a holistic understanding of textual relationships that stretch across both time and space. The combination of local and global interactions triggers a profound comprehension, bringing to light not only the individual contributions of each word but also the collective symphony they orchestrate.

However, BANs' strength doesn't end at their bidirectional and multifaceted attention mechanisms. They also exhibit keen responsiveness to the tasks at hand, particularly in scenarios like question answering. Here, the model dynamically adjusts its focus by aligning the question with the context, ensuring that the most relevant information is extracted. (Chen et al., 2019) proposed a Bidirectional Attentive Memory Network for question answering. This two-way flow model allows for the exchange of interactions between the questions and the knowledge bases with a Bidirectional Attentive Memory network called BAMnet. The use of BAN in this model allowed the model to perform significantly better than existing information-retrieval models. This allows for performing better NLP tasks.

2.8 Sentence Length

Sentence length casts a profound influence on the task of text classification. This dynamic relationship between the length of sentences and the effectiveness of classification models unveils a multifaceted landscape that navigates various dimensions of NLP tasks.
Consider the context-driven aspect; longer sentences embody a richer tapestry of information. The depth of detail embedded within extensive sentences can enhance the subtle understanding of text, bolstering the model's ability to grasp semantic complexities. Yet, this wealth of context comes at a computational price. Longer sentences inflate the dimensionality of inputs, imposing demands on computational resources that could hinder efficiency. This holds particularly true for models like Convolutional Neural Networks (CNNs) and Transformers, where extensive computations are intrinsic. (Adi *et al.*, 2017) proposed that sentence length is one of the core elements of any sequence. A longer sentence length ultimately gives a better context and a deeper narrative, they require models that can manage their complexity. On the other hand, shorter sentences might sacrifice holistic context, which could lead to misinterpretation or inadequate feature extraction (Bodapati, et al., 2019). Models like Transformers leverage attention mechanisms to decipher relationships within sentences. Longer sentences require broader attention spans, potentially escalating computational demand. Conversely, shorter sentences might encounter challenges in generalizing effectively due to their limited context. Furthermore, the impact of sentence length is nuanced by the nature of the task. Sentiment analysis thrives in the presence of extended context, as it uncovers subtle emotional differences.

(Singh *et al.*, 2022a)'s research was conducted to perform sentiment analysis on textual images with a dataset containing a max sentence length of 34 words. This achieved a huge feat in the field of NLP as it performed the task accurately with optimum detail and achieved high levels of performance. (Søgaard *et al.*, 2014)'s research hypothesized that sentence length is a very important metric when performing NLP tasks with his experimentation and research and drew

the conclusion that the length of the sentence is involved in the overall accuracy of models when performing NLP tasks.

2.9 BERT (Bidirectional Encoder Representations from Transformers)

BERT (Bidirectional Encoder Representations from Transformers) is a groundbreaking model in Natural Language Processing (NLP) that has redefined language understanding. It employs a transformer architecture to capture contextual information from both the left and right sides of a word, revolutionizing tasks like sentiment analysis, question answering, and text classification. Through unsupervised pretraining on massive text corpora, BERT learns rich contextualized word embeddings. Fine-tuning specific tasks tailors BERT's knowledge for diverse applications, showcasing its remarkable transfer learning capabilities. BERT's architecture and pretraining method have inspired subsequent models like RoBERTa, XLNet, and more. This paradigm shift in NLP revolves around contextual embeddings, enabling machines to comprehend subtle language intricacies, ultimately advancing the field's capabilities and applications. BERT has emerged as a trailblazing model that has revolutionized the way machines understand language. At its core, BERT functions by employing a sophisticated transformer architecture, enabling it to grasp the nuances of context in language, thereby opening the door to advanced NLP applications. At the heart of BERT's functioning lie its inputs, which are sequences of tokens extracted from sentences. Employing tokenization, these sentences are dissected into smaller linguistic units, supplemented with special tokens like [CLS] (Pogiatzis, 2019) and [SEP] (Lai *et al.*, 2020), marking the start and end of sentences, respectively. This input format allows BERT to navigate language with a granular focus, accommodating intricate contextual relationships. BERT's architecture is a pivotal aspect of its prowess. It incorporates layers of self-attention mechanisms and feed-forward neural networks, capturing both the left and right context of each token. This bidirectional understanding is a remarkable departure from previous models that only considered one direction of context. Such architecture grants BERT an unparalleled capability to discern the intricate web of meaning woven into words through their interaction with neighboring words.
The genius of BERT's training mechanism lies in its combination of masked language modeling (Bitton *et al.*, 2021) and next-sentence prediction. During pretraining, BERT learns to predict masked-out words within sentences and whether pairs of sentences naturally follow each other. This process instills BERT with a profound comprehension of the contextual dynamics that shape language. The outcome of BERT's pretraining is a set of contextualized word embeddings that encode the rich semantic information encapsulated in language. These embeddings become the foundation for BERT's versatility. By appending task-specific layers, BERT can be fine-tuned for various downstream applications like sentiment analysis, question answering, and more. Its capacity to generate contextually nuanced embeddings enables it to outperform previous models, as it effectively comprehends the multifaceted nature of language.

2.10 BERT Embeddings

In natural language processing (NLP), the emergence of BERT (Bidirectional Encoder Representations from Transformers) embeddings has ignited a paradigm shift. These contextual word embeddings, derived from the BERT model, redefine how words are represented within sentences. Unlike traditional embeddings that offer static representations, BERT embeddings imbue words with contextual sensitivity, capturing the intricate dance of language's contextual

nuances. (Alsentzer *et al.*, 2019)'s research praises BERT embeddings. The research pushes for the use of BERT embeddings for clinical NLP researchers. (Lin *et al.*, 2022) discovered that when a BertGCN is optimized with an auxiliary classifier that operates on Bert embeddings, this achieves better performances and faster convergence.

BERT's ingenuity lies in its bidirectional training strategy within a transformer architecture (Khadhraoui *et al.*, 2022). Trained on extensive text corpora, BERT learns to predict words by considering both preceding and succeeding words, embracing the holistic essence of language. This profound understanding enables BERT to create embeddings that reflect not only the word itself but also its contextual import. The result is a set of embeddings rich in semantic depth (Yenicelik et al., 2020). (Wang et al., 2021)'s research investigated the use of BERT embeddings for the use of better skip-gram embeddings. It was concluded that BERT, while serving as context embeddings was able to give a better balance between syntactic and surface features and was able to select useful context more efficiently. They concluded that BERT achieved a very strong context representation and incorporated considerable information from extensive pre-training, in addition to syntactic and surface properties.

BERT embeddings navigate word polysemy (Garí et al., 2021)—where words have multiple meanings—along with capturing the nuanced influences of neighboring words. Their flexibility extends to fine-tuning for specific NLP tasks, rendering them adaptable to a spectrum of applications, from sentiment analysis to question answering (Alberti et al., 2019). Here BERT performs extremely well when used for jointly predicting short and long answers instead of using a pipeline approach.

BERT embeddings represent a leap forward in contextualized language representation, powering modern NLP with unmatched precision. Their ability to embody context, decipher subtle nuances, and adapt to diverse tasks speaks to the potency of transformer-based models. Engineered to harmonize seamlessly with BERT-based models, the tokenizer imparts a contextual understanding that transcends traditional tokenization methods. Employing a "word piece" approach, the tokenizer dissects words into subword tokens, enabling it to capture intricate linguistic nuances and handle morphologically complex languages adeptly (Xu *et al.*, 2019). This proficiency extends to domain-specific or infrequent words, ensuring a comprehensive vocabulary coverage. Notably, the tokenizer adeptly manages lengthier texts, adeptly truncating or segmenting them to align with BERT's token limitations while maintaining context coherence. By introducing special tokens and segment embeddings (Liu *et al.*, 2022), the tokenizer fosters an ideal input structure for BERT models, refining their ability to decipher and classify text accurately.

2.11 BERT Embeddings vs. other embeddings

The power of word embeddings cannot be understated. These representations serve as the bedrock for understanding textual data and extracting meaningful insights (Bahgat et al., 2020). Among the pantheon of embeddings, BERT embeddings, Word2Vec, GloVe, and their contemporaries stand as transformative innovations, each leaving its indelible mark on the landscape of language understanding. At the forefront of this spectrum, BERT embeddings shine with their contextual sensitivity—a hallmark that fundamentally alters how words are perceived within sentences. Unlike Word2Vec and GloVe, which provide static representations unaffected by context, BERT embeddings possess an intricate understanding of the surrounding words. This contextual awareness enables BERT to capture the intricate dance of polysemy and

nuances, facilitating the extraction of profound meaning from text. A defining feature of BERT embeddings is their bidirectional training strategy—a departure from the context-independent training of Word2Vec and GloVe. By scrutinizing both the words preceding and succeeding a target word, BERT achieves a holistic comprehension of language. This dynamic understanding translates into embeddings that encapsulate not only the word's essence but also its contextual significance. Furthermore, the adaptability of BERT embeddings is underscored by their fine-tuning capability (Sun *et al.*, 2019). This feature enables these embeddings to be tailored for specific NLP tasks, empowering them to resonate with domain-specific nuances and deliver tailored solutions. In comparison, Word2Vec and GloVe embeddings offer their unique strengths. Word2Vec's static representations are adept at capturing semantic relationships (El Boukkouri *et al.*, 2019), while GloVe excels at encoding global word associations through corpus-wide co-occurrence statistics (Pennington et al., 2014). The choice of embedding hinges on the intricacies of the task at hand. BERT embeddings are unparalleled in their mastery of context, making them ideal for sentiment analysis, question-answering, and language generation (Rönnqvist *et al.*, 2019). However, for tasks emphasizing global semantic relationships or computational efficiency, Word2Vec and GloVe remain steadfast allies (Bhoir et al., 2017). BERT. embeddings, standing as champions of context, redefine how words are perceived within sentences, opening avenues for deeper insights. In recent times, BERT is seen to have performed exceptionally well as compared to other options. For this reason, BERT was the ideal choice for this research.

Based on the above literature we have decided to use attention mechanism to generate the textual images and then these images will be classified using DenseNet architecture. In addition to that the earlier BLSTM-ResNet model did not explore the sentence length impact on the model performance so, in this report work is also conducted on the sentence length impact on network performance.

2.12 BLSTM-ResNet

(Singh *et al.*, 2022a) proposes that BLSTM can be used to form textual images by processing the input text in a sequential manner, one word or character at a time. It utilizes two sets of hidden states, one that processes the input text in a forward direction and another that processes the input text in a backward direction. For this research, BERT is incorporated to form textual images as it performed better in recent times.
BLSTM uses word embeddings to generate textual images while capturing temporal dependencies. The Embedding layer inputs word embeddings into the network. The BLSTM (Bidirectional Long Short Term Memory) block, based on RNN, treats text data as a time series. It captures dependencies between words by considering previous and following words. This Bidirectional RNN creates a two-dimensional matrix of dimensions TxH, where T is time steps and H is BLSTM's hidden units. The resulting matrix is reshaped to TxHx1 via a reshape layer, resembling an image with Height, Width, and Channel dimensions, where a channel of one denotes a greyscale or monochromatic image. In Harpreet's paper, the evaluation metric used is accuracy as the model dataset used is balanced.

Chapter 3: Design Methodology

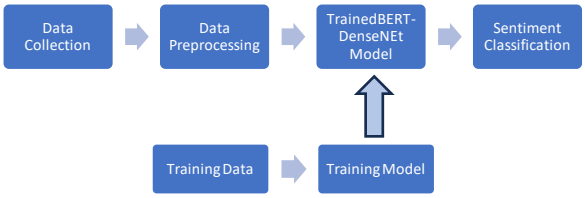

Fig 1: Model Overview

The proposed model for this research is divided into two parts. BERT would be used to generate textual images from word embeddings while capturing temporal dependencies and the second, a DenseNet-inspired 2D-CNN to explore feature space. After, an Average Pooling Layer, a Flatten Layer, and an Output Layer. The notion of using a deep learning model to classify textual images is first explored in (Singh *et al.*, 2022a) where they have implemented the BLSTM-ResNet model

3.1 BERT-DenseNet model

BERT employs a bidirectional architecture to capture both past and future dependencies in text, a significant difference from traditional sequential models. It achieves this by leveraging a Transformer model, which processes the entire sequence of words simultaneously. Through a self-attention mechanism, BERT assigns varying weights to words based on their relationships with others, enabling it to comprehend word dependencies regardless of their position. BERT generates contextualized embeddings. As it tokenizes text into subword units and processes them, each subword token is represented by a high-dimensional vector, encapsulating its meaning relative to the entire sentence. This process involves multiple layers of self-attention and neural networks. Each row in the matrix corresponds to the embedding of a specific subword token. To provide positional context, BERT adds positional encodings to its embeddings. This addresses the fact that BERT processes text in parallel rather than sequentially. These encodings supply information about the position of each token in the sequence, enabling BERT to differentiate between tokens with similar spelling but distinct positions. The end of the sequence produces a two-dimensional matrix that assumes dimensions of LxH, where L denotes the sequence length (number of subword tokens), and H signifies the hidden size of the BERT model. This matrix is then reshaped with the help of a reshape layer to produce a matrix with dimensions LxHx1 which is the equivalent of a greyscale image where L is the height, H is the width and 1 is the channel.

3.2 Google Colab

The experimentation and code involved in this project were all run and generated on Google Colab. Google Colab, or Collaboratory, is a cloud-based platform by Google for collaborative Python coding in Jupyter Notebook format. It offers free access to GPU-equipped virtual machines, ideal for data analysis and machine learning. Users can work together in real time, harness GPU acceleration for computationally intensive tasks, and integrate notebooks with Google Drive. Pre-installed libraries, Markdown support, and code snippets simplify tasks.

Colab's cloud-based nature eliminates local installation requirements, making it popular for research, learning, and prototyping.

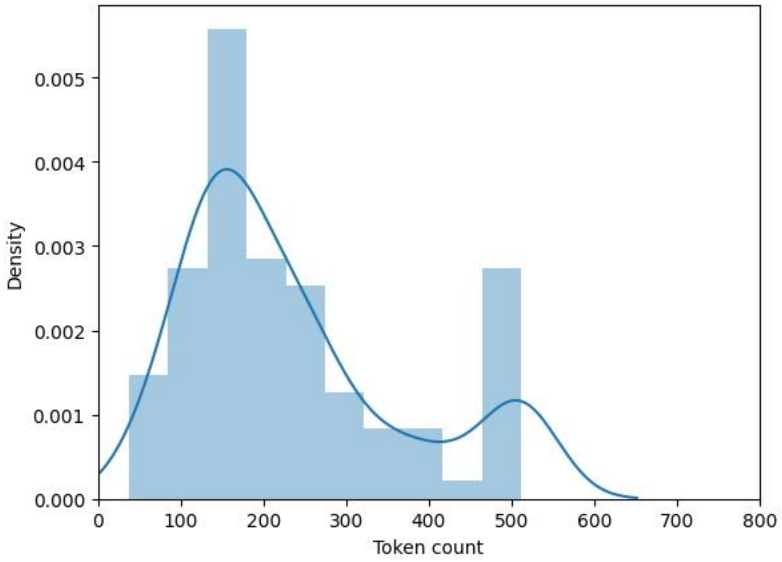

Fig. 2: IMDB movie dataset Token Density

Fig. 2 shows the distribution of the token count of the IMDB movie review dataset chosen for this research. From the image, we can see that the dataset contains sentences with large token values ending with a value of 512.

3.3 DATASET

For evaluation and training, several datasets were considered. However, the choice of the dataset to be used for this research was the IMDB movie review dataset (*IMDB Dataset of 50K Movie Reviews*, 2011). This dataset is a binary sentiment analysis dataset made up of 50,000 reviews from IMDb that have been classified as positive or negative. The dataset includes an equal number of positive and negative reviews. It's sizable, ideal for training models, and provides binary sentiment classification tasks. With varied text lengths and languages, it serves as a benchmark for sentiment analysis, aiding in developing algorithms for sentiment classification within the natural language processing domain.

This research is conducted on movie reviews from the IMDB movie review dataset which contained sentences with a max token of 512 words for the purpose of determining the effect sentence length has on the accuracy of NLP tasks. This is a huge leap as the models introduced in this research attempt to capture the sentiment of these sentences with iterations in the number of tokens of the sentences, i.e., 32, 64, 128, 256, and 512. This allows the models to gain more context into the true meaning of the reviews of this dataset and to further understand the context of these words as well as their effect on accurate classifications.

3.4 Data Processing

The dataset contains movie reviews that were typed by people. Once the human factor was involved, it was assumed to contain some amount of noise or errors related to text input which

needed to be cleaned before being fed to the model. The worst was assumed of the dataset to anticipate errors that may or may not have been contained in the dataset. For filtering the textual data, Python possesses a regular expression package where string characters are mentioned to be removed from these sentences. The characters mentioned in this expression package are characters that go along with focused characters. These are removed without affecting the sentiment of the sentence. The list of words often removed from textual data for sentiment analysis is as follows:

URLs – They mostly start with ‘www’, ‘http://’, ‘https’. These are references or addresses used to locate resources on the internet. These characters do not affect the sentiment of the sentences and are therefore removed in the preprocessing step.

Hashtags - Hashtags are words or phrases preceded by the "#" symbol, often used on social media platforms to categorize, and organize content. They allow users to label their posts with relevant keywords or topics, making it easier for others to discover and engage with similar content.

3.5 BERT

The text data is tokenized into subwords to handle vocabulary and context effectively. The BERT embeddings are obtained by passing the tokenized sequences through the model. The sentiment analysis classifier is constructed on top of the BERT embeddings, incorporating task-specific layers for classification. The model is trained using an optimizer and loss function, while hyperparameter tuning is conducted. Evaluation is performed on a test set, showcasing the model's accuracy, precision, recall, and F1-score.

3.6 Feature Extraction

The BERT DenseNet model merges BERT embeddings and DenseNet's structure for feature extraction. BERT captures context and DenseNet hierarchically extracts features. Combined features encompass global context and local patterns, used by a classification head for sentiment analysis. Through training, the model adjusts weights to align features with sentiment labels. During evaluation, the learned features enable sentiment prediction. The combination of BERT and DenseNet enhances sentiment analysis by efficiently capturing subtle and contextual features from text, leading to improved accuracy.

3.7 BERT Embeddings

In this research, BERT embeddings would play a crucial role as they are pretrained contextual word representations that capture the context and meanings of the text. They enhance performance by leveraging contextual understanding. Their output of contextualized word embeddings captures the semantic and contextual information of each token based on the surrounding words within the sentence producing a matrix of embeddings where each row corresponds to a token and each column corresponds to a dimension in the embedded space. In neural networks, they operate as the gateway, shaping raw text into a format that algorithms can process. As neural networks train, word embeddings evolve.

3.8 BERT Pretraining

Pretraining with BERT (Bidirectional Encoder Representations from Transformers) involves training a language model on a massive corpus of text to learn contextualized word representations. BERT employs a transformer architecture, which captures bidirectional relationships between words, enabling it to understand the nuances of language. In this research, pretraining on BERT would allow the model to predict missing words in sentences and learn comprehensive language representations that encompass a wide range of linguistic patterns and contextual information which would later be used for fine-tuning for the sentiment analysis classification task. The largest sentence length of 512(Fig.2) tokens was used for pretraining to train the model to maximum capacity. Additionally, the model was trained on the entire dataset of 50,000 rows where 2% of the dataset was used as a test set.

3.9 DENSENET

DenseNet, a potent architecture initially developed for computer vision tasks, has found a dynamic application in Natural Language Processing (NLP) classification tasks. DenseNet, known for its dense connectivity and feature reuse, serves as an innovative approach to enhance the understanding of language in the context of classification. An embedding layer is added to the DenseNet model using pre-trained BERT embeddings. Dense and transition blocks are constructed, enhancing feature extraction. The model gains the ability to leverage both the deep features learned by DenseNet and the contextualized embeddings provided by BERT, resulting in a robust sentiment analysis classifier with a strong understanding of semantic context.

3.10 2D CNN Densenet Architecture

The fusion of 2D Convolutional Neural Networks (CNNs) and DenseNet architecture presents a formidable approach that revolutionizes image understanding. This combined model would combine the strengths of both models to yield a better performance for the sentiment analysis task by promoting feature reuse and combining multi-level features to ensure parameter efficiency while the 2D CNN captures local patterns and hierarchical features. The combination of these models allows for a potent model for classification. This research incorporated the smallest Densenet architecture, Densenet-121 with a tweak in the number of layers in each dense block due to limited computational resources. The layers were also reduced to increase the training speed and time.

3.11 Model Training

The model was trained with a learning rate of 2e-5. A learning rate of 2e-5 involves optimizing the model's performance using a specific rate of parameter updates. A learning rate of 2e-5 corresponds to a small value, which often facilitates gradual convergence and prevents overshooting during training. It allows the model to make subtle adjustments to its weights, enhancing its ability to fit the data accurately. However, a lower learning rate might extend training time. During training, the model's weights are adjusted iteratively based on the gradients of the loss function with respect to the parameters. A small learning rate like 2e-5

implies cautious weight updates, striking a balance between steady progress and avoiding large changes that might lead to divergence. This learning rate was chosen as it works to the advantage of fine-tuning BERT as this is a pre-trained model, which ensures that prior knowledge is retained when adapting to task-specific variations. Training the model using a learning rate of 2e-5 involves careful weight adjustments that strike a balance between convergence speed and stability. This is important for fine-tuning as the task demands a coalition of prior knowledge while adapting to new data and tasks.

3.12 Model Validation

Model validation for an NLP task is a crucial step to assess the performance and generalization of a trained model. This research incorporates validation of 1% of the entire dataset. It involved using a validation dataset that the model hadn't seen during training. The model's predictions on the validation data are compared with the actual labels to calculate evaluation metrics like accuracy, precision, recall, F1-score, etc. This process helped to determine how well the model was likely to perform on unseen data.

3.13 Experiment

Two experiments were designed for this project.

3.13.1 Experiment 1

The first is to determine whether a 2D CNN Densenet model would be able to classify sentiment based on the IMDB movie review dataset accurately. This experiment leverages the power of both architectures to classify sentiment based on the text in the movie review dataset. Here, the architecture takes as input a sequence of word embeddings representing the words in a sentence. Each word embedding is a vector that captures the meaning of the word within the context of the sentence. Each word or character in the text is mapped to its corresponding embedding vector. If using word embeddings, these vectors are stacked to create an embedding matrix. Textual images are then formed by converting sequential word embeddings into two-dimensional visualizations. Pre-trained word embeddings are aligned in an embedding matrix, reshaped to the 2-dimension greyscale image, and normalized. This reshaped matrix is then transformed into grayscale images. The 2D CNN processes the text data by treating it as a 2D matrix with one dimension representing the words and the other dimension representing their embedding dimensions. Average pooling is then applied to reduce the dimensionality while retaining crucial information. After, flattening and dense layers follow to process the features for final classification. This research uses the smallest architecture of Densenet; Densenet-121 architecture (*DenseNet PyTorch class*, 2019). This variation of Densenet architecture was further modified to have 1 layer in each of the 4 dense blocks instead of [6, 12, 24, 16]. The dense layers receive feature maps from previous layers in a dense block. They combine these feature maps through channel-wise concatenation, facilitating comprehensive information sharing. This dense connectivity empowers each layer to capture diverse patterns and abstractions, enhancing sentiment understanding by considering varied linguistic cues. The dense layers effectively reuse features from all preceding layers, promoting fine-grained detail detection. The transition layers help to control feature map growth, while global average pooling aggregates spatial information. The transition layers in the Densenet architecture then reduce feature map dimensions through convolution and pooling. They balance feature

extraction and computational efficiency, aiding in capturing local and global linguistic patterns essential for sentiment understanding in text data. The global average pooling combines feature map information by computing their average values, condensing their essence. This aids generalization. Subsequently, fully connected layers process these aggregated features, iteratively transforming them through weighted connections to yield sentiment predictions. The final fully connected layer produces a probability distribution over sentiment classes. Activation functions like softmax convert scores to probabilities. This process encapsulates the model's ability to comprehend and classify sentiment in text data, rendering DenseNet a powerful tool for sentiment analysis tasks. The activation functions, ReLU introduces complexity capture, and loss functions quantify prediction accuracy. Training is then performed on 90% of the data fed into the architecture. The model is then trained using backpropagation and gradient descent algorithms. Learning rate and dropout rates are tuned to optimize performance while updating weights. The fine-tuning refines hyperparameters for optimal parameter values, enhancing sentiment prediction performance.

3.13.2 Experiment 2

This experiment is conducted to determine whether sentence length influences the accuracy of the 2D CNN sentiment analysis classification. The IMDB movie review dataset contains 50,000 rows of data with a maximum token surpassing 512. The experiment was designed to run multiple iterations with different sentence lengths i.e., 32, 64, 128, 256, 512 on different datasets sized i.e., 500, 1000, 2000, 4000, and 8000. A sentence length of 32 is started with the dataset of size 500, working its way through the data sizes mentioned and ending at 512. This process is repeated for sentence lengths of 64, 128, 256, and 512 The following metrics are then recorded for analysis; to accurately determine if there is a correlation between them and ultimately investigate if the dataset size influences the accuracy of the classification:

Precision: Precision in a sentiment analysis classification report quantifies how accurate the model is in identifying positive sentiment instances, thus providing insight into the model's performance concerning false positives.
Precision = True Positives / (True Positives + False Positives)

Recall: Recall indicates the model's effectiveness in identifying actual positive sentiments within the dataset. A high recall score implies that the model is capturing a significant portion of the positive sentiments, making it suitable for tasks where correctly identifying positive sentiments is crucial, such as customer feedback analysis or medical diagnosis.
Recall = True Positives / (True Positives + False Negatives)

F1-Score: F1-score in a classification report for sentiment analysis offers an aggregated assessment of a model's precision and recall, providing a holistic view of its performance across different sentiment classes. The F1-score is particularly useful when dealing with imbalanced classes or when false positives and false negatives have different implications. It provides a comprehensive evaluation of how well the model performs across all sentiment categories. A higher F1-score indicates a better balance between precision and recall, suggesting the model has a good trade-off between identifying positives and avoiding false positives and false negatives.
F1-score = 2 * (Precision * Recall) / (Precision + Recall)

Confusion Matrix: The confusion matrix plays a crucial role after a sentiment analysis classification report by providing a detailed view of a model's performance. In sentiment analysis, where classifying sentiments like positive, negative, and neutral is essential, the confusion matrix visually breaks down the actual and predicted class labels. It helps diagnose errors, highlights misclassifications, and is particularly valuable for imbalanced sentiment distributions. By calculating metrics like accuracy, precision, recall, and F1-score per sentiment class, it offers insights into the model's performance across different sentiments. This analysis aids in identifying areas of improvement, refining the model's predictions, and making informed decisions. The confusion matrix serves as a vital tool for delving beyond general metrics, enabling a deeper understanding of the model's sentiment classification capabilities, and assisting in achieving more accurate and effective sentiment predictions.

3.14 Code Implementation

```
class SentimentClassifier(nn.Module):

  def __init__(self, n_classes):
    super(SentimentClassifier, self).__init__()
    self.bert = BertModel.from_pretrained(PRE_TRAINED_MODEL_NAME, return_dict=False)
    self.drop = nn.Dropout(p=0.3)
    self.last = nn.Linear(self.bert.config.hidden_size, n_classes)
    self.activ = torch.nn.Sigmoid()

  def forward(self, input_ids, attention_mask):
    _, pooled_output = self.bert(
      input_ids=input_ids,
      attention_mask=attention_mask
    )
    output = self.drop(pooled_output)
    output = self.last(output)

    return self.activ(output)
```

Fig. 3: The sentiment classifier is a class created to generate contextualized BERT embeddings.

```
def densenet121(*, weights = None, progress: bool = True, **kwargs: Any) -> DenseNet:
    r"""Densenet-121 model from
    `Densely Connected Convolutional Networks <https://arxiv.org/abs/1608.06993>`_.

    Args:
        weights (:class:`~torchvision.models.DenseNet121_Weights`, optional): The
            pretrained weights to use. See
            :class:`~torchvision.models.DenseNet121_Weights` below for
            more details, and possible values. By default, no pre-trained
            weights are used.
        progress (bool, optional): If True, displays a progress bar of the download to stderr. Default is True.
        **kwargs: parameters passed to the ``torchvision.models.densenet.DenseNet``
            base class. Please refer to the `source code
            <https://github.com/pytorch/vision/blob/main/torchvision/models/densenet.py>`_
            for more details about this class.

    .. autoclass:: torchvision.models.DenseNet121_Weights
        :members:
    """
    weights = None

    return _densenet(32, (1, 1, 1, 1), 64, weights, progress, **kwargs)
```

Fig. 4: DenseNet Incorporation and layer count

From the DenseNet repository, the DenseNet architecture is obtained and modified to suit the purpose of the experimentation. A custom DenseNet architecture with one layer in each of the dense blocks is used which ends up with a depth of 4.

```
model = CustomDenseNet()
model = model.to(device)
print(model)
optimizer = AdamW(model.parameters(), lr=learning_rate, correct_bias=False)
total_steps = len(train_DataLoader) * EPOCHS

scheduler = get_linear_schedule_with_warmup(
  optimizer,
  num_warmup_steps=0,
  num_training_steps=total_steps
)

loss_fn = nn.BCELoss().to(device)
```

Fig.5: Custom DenseNet Model

```
[ ]  #Dataloader Class
     def data_loader(df, tokenizer, sen_length, batch):
       ds=imdb_sentiment(
           reviews=df.text.to_numpy(),
           targets=df.polarity.to_numpy(dtype=int),
           tokenizer=tokenizer,
           sen_length=sen_length
       )

       return DataLoader(
           ds,
           batch_size=batch_size,
           num_workers=0
       ) #num_workers 2 is efficient read https://deeplizard.com/learn/video/kWVgvsejXsE


     tokenizer=BertTokenizerFast.from_pretrained(Pre_trained_model)

     # Load datasets
     train_DataLoader=data_loader(df_train,tokenizer,sen_length,batch_size)
     test_DataLoader=data_loader(df_test,tokenizer,sen_length,batch_size)
     valid_DataLoader=data_loader(df_valid,tokenizer,sen_length,batch_size)
```

Fig.6: Data loader Class

The data loader class is used to efficiently load and prepare the data for training, validation, and testing during the training process.

Chapter 4: Results and Analysis

4.1 BERT-Densenet Experiment

This experiment is conducted to determine whether a BLSTM-Densenet architecture can accurately classify textual images for sentiment analysis tasks. Here, the entire dataset is first pre-trained with a BERT model, where its tokenizer splits text into subword tokens, while padding and truncation ensure fixed-length input. Special tokens like [CLS] and [SEP] are inserted, and for sentence pairs, segment IDs indicate sentence boundaries. Converting tokens to numerical IDs facilitates input for BERT. The Input is then formatted with token IDs, segment IDs, and attention masks. After pretraining, the 2D CNN Densenet architecture is introduced. The 2D CNN DenseNet architecture transforms text data into a 2D matrix of word embeddings. Convolutional layers analyze local patterns, while dense blocks extract intricate features. The transition layers control dimensions, and global average pooling summarizes information. The fully connected layer maps pooled features to sentiment classes. Finally, the training is done on the IMDB movie review data and fine-tuning to optimize the model.

Fig 7: BERT Model as a feature extractor

Fig 8: 2D CNN Densenet architecture

4.2 Sentence Length vs. Dataset Size Experiment

After each iteration, the classification report, confusion matrix, a plot of training and validation accuracy against epoch and the values of train loss and accuracy, and validation loss and accuracy were generated for a better understanding of the experimentation and results.

Sentence Length = 32/ Dataset = 500 samples

```
Epoch 1/5
----------
Train loss 0.6574387543731266 accuracy 0.6644444444444444
Val   loss 0.5502636098861694 accuracy 0.72

saving best model
Epoch 2/5
----------
Train loss 0.6064783249961005 accuracy 0.7222222222222222
Val   loss 0.497866632938385 accuracy 0.8

saving best model
Epoch 3/5
----------
Train loss 0.5858456671237946 accuracy 0.7288888888888889
Val   loss 0.47523239493370056 accuracy 0.8

Epoch 4/5
----------
Train loss 0.5757213032245636 accuracy 0.7266666666666667
Val   loss 0.46564739525318144 accuracy 0.78

Epoch 5/5
----------
Train loss 0.570954022804896 accuracy 0.7311111111111112
Val   loss 0.463268107175827 accuracy 0.78

CPU times: user 11min 55s, sys: 2.2 s, total: 11min 57s
Wall time: 2min 59s
```

Fig 9: 32-500 training and validation metrics

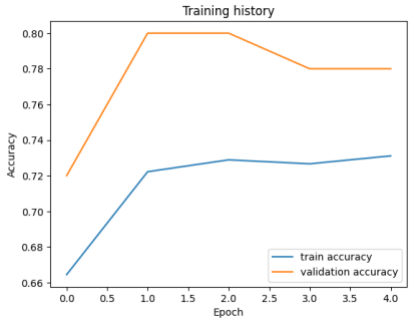

Fig 10: 32-500 training history

```
sentence length 32
dataset size 500
batch size 2
              precision    recall  f1-score   support

    negative     0.7810    0.6737    0.7234       524
    positive     0.6880    0.7920    0.7363       476

    accuracy                         0.7300      1000
   macro avg     0.7345    0.7328    0.7298      1000
weighted avg     0.7367    0.7300    0.7295      1000
```

Fig 11: 32-500 classification report

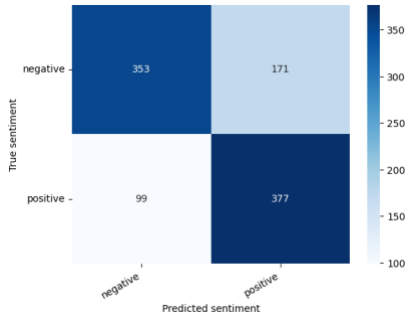

Fig 12: 32-500 Confusion matrix

Sentence Length = 64/ Dataset = 500 samples

```
Epoch 1/5
----------
Train loss 0.6062369508213467 accuracy 0.66
Val   loss 0.39970826923847197 accuracy 0.9

saving best model
Epoch 2/5
----------
Train loss 0.5460694583919313 accuracy 0.7311111111111112
Val   loss 0.3596052569150925 accuracy 0.86

Epoch 3/5
----------
Train loss 0.5347010666131973 accuracy 0.7355555555555555
Val   loss 0.3463650065660477 accuracy 0.86

Epoch 4/5
----------
Train loss 0.52909451160166 accuracy 0.7377777777777778
Val   loss 0.3408583465218544 accuracy 0.86

Epoch 5/5
----------
Train loss 0.5262118045488994 accuracy 0.7355555555555555
Val   loss 0.3394504591822624 accuracy 0.86

CPU times: user 17min 37s, sys: 2.1 s, total: 17min 39s
Wall time: 4min 24s
```

Fig 13: 64-500 training and validation metrics

Fig 14: 64-500 training history

```
sentence length 64
dataset size 500
batch size 2
              precision    recall  f1-score   support

    negative     0.8595    0.8053    0.8315       524
    positive     0.7996    0.8550    0.8264       476

    accuracy                         0.8290      1000
   macro avg     0.8295    0.8302    0.8290      1000
weighted avg     0.8310    0.8290    0.8291      1000
```

Fig 15: 64-500 classification report

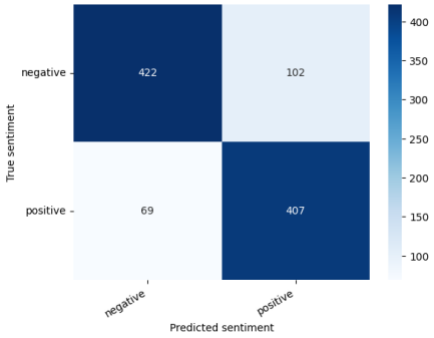

Fig 16: 64-500 confusion matrix

Sentence Length = 128/ Dataset = 500 samples

```
Epoch 1/5
----------
Train loss 0.5670286788543065 accuracy 0.7311111111111112
Val   loss 0.26233308225870133 accuracy 0.96

saving best model
Epoch 2/5
----------
Train loss 0.5019845043288337 accuracy 0.7466666666666667
Val   loss 0.22309510827064513 accuracy 0.96

Epoch 3/5
----------
Train loss 0.4862364865011639 accuracy 0.7377777777777778
Val   loss 0.20845186203718186 accuracy 0.96

Epoch 4/5
----------
Train loss 0.47775393956237366 accuracy 0.7422222222222222
Val   loss 0.20166052997112274 accuracy 0.96

Epoch 5/5
----------
Train loss 0.47361733661757577 accuracy 0.7444444444444445
Val   loss 0.1993175759911537 accuracy 0.96

CPU times: user 29min 21s, sys: 2.18 s, total: 29min 23s
Wall time: 7min 20s
```

Fig 17: 128-500 training and validation metrics

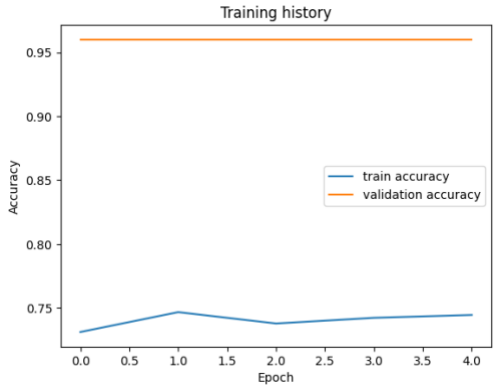

Fig 18: 128-500 training history

```
sentence length 128
dataset size 500
batch size 2
              precision    recall  f1-score   support

    negative     0.9475    0.8263    0.8828       524
    positive     0.8324    0.9496    0.8871       476

    accuracy                         0.8850      1000
   macro avg     0.8899    0.8880    0.8850      1000
weighted avg     0.8927    0.8850    0.8849      1000
```

Fig 19: 128-500 classification report

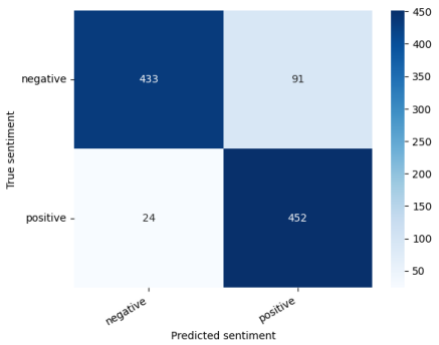

Fig 20: 128-500 confusion matrix

Sentence Length = 256/ Dataset = 500 samples

```
Epoch 1/5
----------
Train loss 0.5417372461160024 accuracy 0.7511111111111111
Val   loss 0.2220994460582733 accuracy 0.98

saving best model
Epoch 2/5
----------
Train loss 0.47788192053635914 accuracy 0.7711111111111111
Val   loss 0.1814376574754715 accuracy 0.98

Epoch 3/5
----------
Train loss 0.46111897110939026 accuracy 0.7911111111111111
Val   loss 0.16547098278999328 accuracy 0.98

Epoch 4/5
----------
Train loss 0.451991608010398 accuracy 0.8
Val   loss 0.15888919055461884 accuracy 0.98

Epoch 5/5
----------
Train loss 0.4475310967365901 accuracy 0.8088888888888889
Val   loss 0.15745322823524474 accuracy 0.98

CPU times: user 51min 51s, sys: 38.5 s, total: 52min 30s
Wall time: 13min 6s
```

Fig 21: 256-500 training and validation metrics

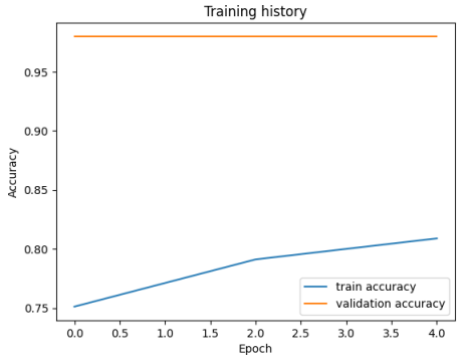

Fig 22: 256-500 training history

```
sentence length 256
dataset size 500
batch size 2
              precision    recall  f1-score   support

    negative     0.9124    0.9542    0.9328       524
    positive     0.9469    0.8992    0.9224       476

    accuracy                         0.9280      1000
   macro avg     0.9297    0.9267    0.9276      1000
weighted avg     0.9288    0.9280    0.9279      1000
```

Fig 23: 256-500 classification report

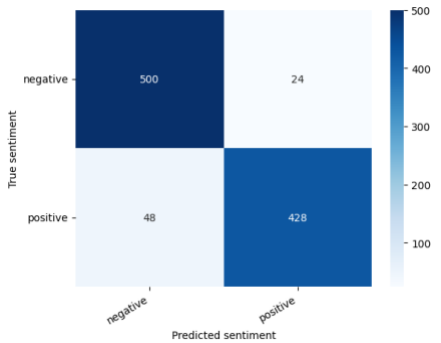

Fig 24: 256-500 confusion matrix

Sentence Length = 512/ Dataset = 500 samples

```
Epoch 1/5
---------
Train loss 0.5269410770469242 accuracy 0.7711111111111111
Val   loss 0.206554916203022 accuracy 0.96

saving best model
Epoch 2/5
---------
Train loss 0.45463992847336665 accuracy 0.8
Val   loss 0.16679459303617478 accuracy 0.96

Epoch 3/5
---------
Train loss 0.4354250852929221 accuracy 0.8266666666666667
Val   loss 0.15276477098464966 accuracy 0.98

saving best model
Epoch 4/5
---------
Train loss 0.42508581976095833 accuracy 0.8377777777777777
Val   loss 0.1467528772354126 accuracy 0.98

Epoch 5/5
---------
Train loss 0.4201779606607225 accuracy 0.8466666666666667
Val   loss 0.1457569819688797 accuracy 0.98

CPU times: user 1h 35min 44s, sys: 4.21 s, total: 1h 35min 48s
Wall time: 23min 56s
```

Fig 25: 512-500 training and validation metrics

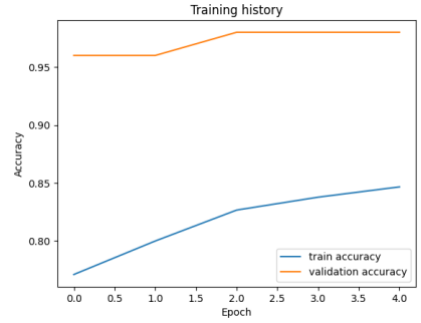

Fig 26: 512-500 training history

```
sentence length 512
dataset size 500
batch size 2
              precision    recall  f1-score   support

    negative     0.9086    0.9676    0.9372       524
    positive     0.9615    0.8929    0.9259       476

    accuracy                         0.9320      1000
   macro avg     0.9351    0.9302    0.9315      1000
weighted avg     0.9338    0.9320    0.9318      1000
```

Fig 27: 512-500 classification report

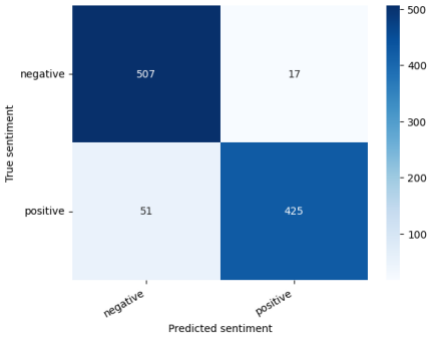

Fig 28: 512-500 confusion matrix

Sentence Length = 32/ Dataset = 1000 samples

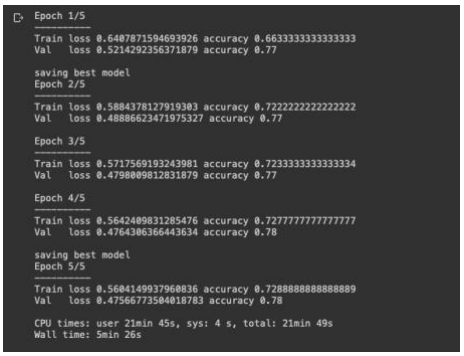

Fig 29: 512-500 training and validation metrics

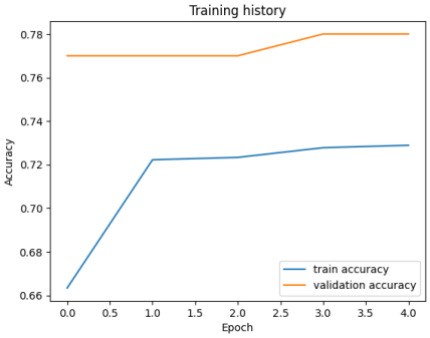

Fig 30: 512-500 training history

	precision	recall	f1-score	support
negative	0.6938	0.8607	0.7683	524
positive	0.7914	0.5819	0.6707	476
accuracy			0.7280	1000
macro avg	0.7426	0.7213	0.7195	1000
weighted avg	0.7403	0.7280	0.7219	1000

Fig 31: 512-500 classification report

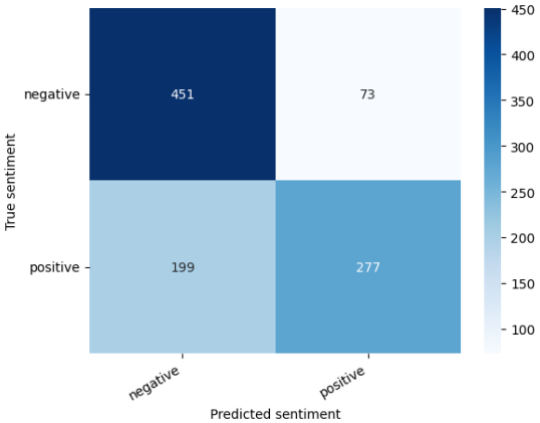

Fig 32: 512-500 confusion matrix

Sentence Length = 32/ Dataset = 1000 samples

```
Epoch 1/5
----------
Train loss 0.5904496614469422 accuracy 0.6822222222222222
Val   loss 0.3767224852740765 accuracy 0.85

saving best model
Epoch 2/5
----------
Train loss 0.5399149613910251 accuracy 0.74
Val   loss 0.35360432393848895 accuracy 0.85

Epoch 3/5
----------
Train loss 0.5306670825680097 accuracy 0.75
Val   loss 0.3470396790653467 accuracy 0.84

Epoch 4/5
----------
Train loss 0.5252911653452449 accuracy 0.7533333333333333
Val   loss 0.3442673798650503 accuracy 0.84

Epoch 5/5
----------
Train loss 0.5224020755290986 accuracy 0.7533333333333333
Val   loss 0.3432885591685772 accuracy 0.84

CPU times: user 36min 55s, sys: 4.26 s, total: 36min 59s
Wall time: 9min 14s
```

Fig 33: 32-1000 training and validation metrics

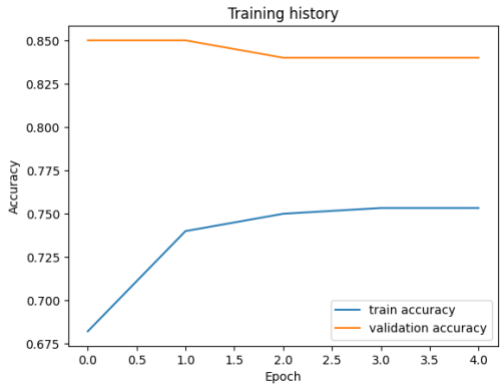

Fig 34: 32-1000 training history

	precision	recall	f1-score	support
negative	0.9154	0.6813	0.7812	524
positive	0.7262	0.9307	0.8158	476
accuracy			0.8000	1000
macro avg	0.8208	0.8060	0.7985	1000
weighted avg	0.8253	0.8000	0.7977	1000

Fig 35: 32-1000 classification report

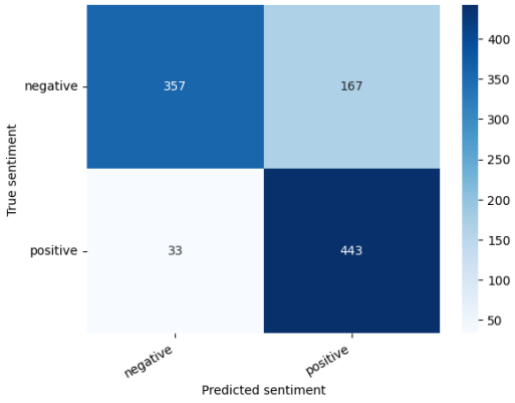

Fig 36: 32-1000 confusion matrix

Sentence length = 64/Dataset =1000 samples

```
Epoch 1/5
----------
Train loss 0.5904496614469422 accuracy 0.6822222222222222
Val   loss 0.3767224852740765 accuracy 0.85

saving best model
Epoch 2/5
----------
Train loss 0.5399149613910251 accuracy 0.74
Val   loss 0.35360432393848895 accuracy 0.85

Epoch 3/5
----------
Train loss 0.5306670825680097 accuracy 0.75
Val   loss 0.3470396790653467 accuracy 0.84

Epoch 4/5
----------
Train loss 0.5252911653452449 accuracy 0.7533333333333333
Val   loss 0.3442673798650503 accuracy 0.84

Epoch 5/5
----------
Train loss 0.5224020755290986 accuracy 0.7533333333333333
Val   loss 0.3432885591685772 accuracy 0.84

CPU times: user 36min 55s, sys: 4.26 s, total: 36min 59s
Wall time: 9min 14s
```

Fig 37: 64-1000 training and validation metrics

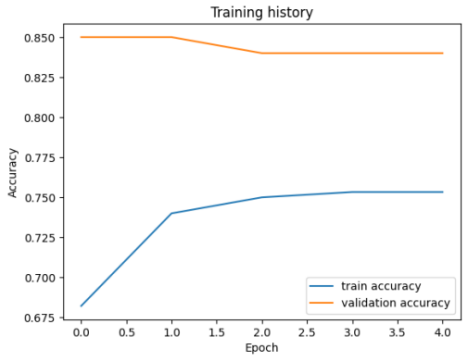

Fig 38: 64-1000 training history

```
sentence length 64
dataset size 2000
batch size 2
              precision    recall  f1-score   support

    negative     0.8769    0.7748    0.8227       524
    positive     0.7803    0.8803    0.8272       476

    accuracy                         0.8250      1000
   macro avg     0.8286    0.8275    0.8250      1000
weighted avg     0.8309    0.8250    0.8249      1000
```

Fig 39: 64-1000 classification report

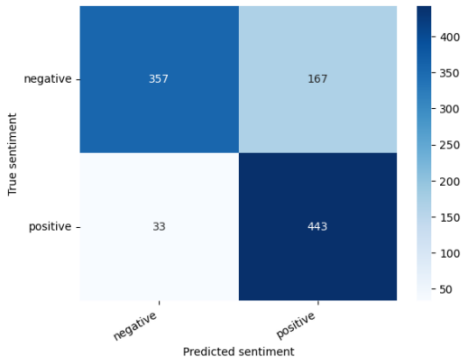

Fig 40: 64-1000 confusion matrix

Sentence Length = 128/ Dataset = 1000 samples

```
Epoch 1/5
----------
Train loss 0.5257509000433815 accuracy 0.7533333333333333
Val   loss 0.2143597562611103 accuracy 0.95

saving best model
Epoch 2/5
----------
Train loss 0.47938611974318823 accuracy 0.7655555555555555
Val   loss 0.19266624227166176 accuracy 0.95

Epoch 3/5
----------
Train loss 0.46688437978426617 accuracy 0.7822222222222223
Val   loss 0.1843022883683443 accuracy 0.95

Epoch 4/5
----------
Train loss 0.45952276786168417 accuracy 0.79
Val   loss 0.1805921034514904 accuracy 0.95

Epoch 5/5
----------
Train loss 0.45552166253328324 accuracy 0.7955555555555556
Val   loss 0.17998140953481198 accuracy 0.95

CPU times: user 56min 53s, sys: 4.29 s, total: 56min 57s
Wall time: 14min 13s
```

Fig 41: 128-1000 training and validation metrics

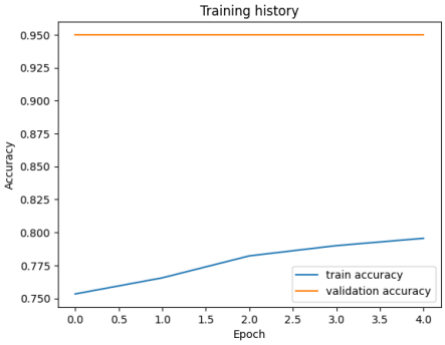

Fig 42: 128-1000 training history

```
sentence length 128
dataset size 1000
batch size 2
              precision    recall  f1-score   support

    negative     0.9407    0.8473    0.8916       524
    positive     0.8485    0.9412    0.8924       476

    accuracy                         0.8920      1000
   macro avg     0.8946    0.8943    0.8920      1000
weighted avg     0.8968    0.8920    0.8920      1000
```

Fig 43: 128-1000 classification report

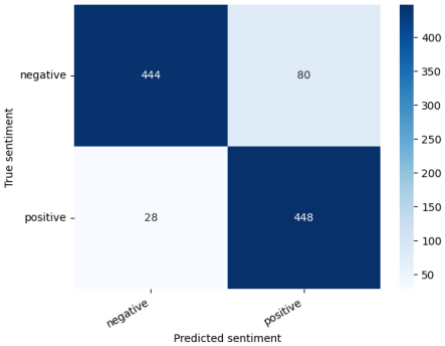

Fig 44: 128-1000 confusion matrix

Sentence Length = 256/ Dataset = 1000 samples

```
Epoch 1/5
----------
Train loss 0.5061307091183133 accuracy 0.7622222222222222
Val   loss 0.1711149789392948 accuracy 0.98

saving best model
Epoch 2/5
----------
Train loss 0.4501260522670216 accuracy 0.79
Val   loss 0.1436767064034939 accuracy 0.98

Epoch 3/5
----------
Train loss 0.4304373171594408 accuracy 0.8122222222222222
Val   loss 0.13285815834999085 accuracy 0.98

Epoch 4/5
----------
Train loss 0.418738003273805 accuracy 0.82
Val   loss 0.12865713238716125 accuracy 0.98

Epoch 5/5
----------
Train loss 0.41292506982882815 accuracy 0.8322222222222222
Val   loss 0.12720931679010392 accuracy 0.98

CPU times: user 1h 45min 29s, sys: 8.26 s, total: 1h 45min 38s
Wall time: 26min 22s
```

Fig 45: 256-1000 training and validation metrics

Fig 46: 256-1000 training history

```
sentence length 256
dataset size 1000
batch size 2
              precision    recall  f1-score   support

    negative     0.9582    0.8760    0.9153       524
    positive     0.8752    0.9580    0.9147       476

    accuracy                         0.9150      1000
   macro avg     0.9167    0.9170    0.9150      1000
weighted avg     0.9187    0.9150    0.9150      1000
```

Fig 47: 256-1000 classification report

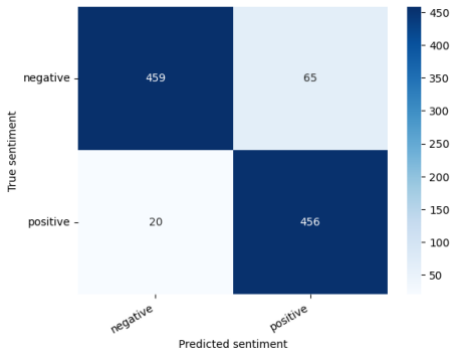

Fig 48: 256-1000 confusion matrix

Sentence Length = 512/ Dataset = 1000 samples

```
Epoch 1/5
----------
Train loss 0.47640518956714206 accuracy 0.7877777777777778
Val   loss 0.12197064824402332 accuracy 1.0

saving best model
Epoch 2/5
----------
Train loss 0.41239823957284294 accuracy 0.8177777777777778
Val   loss 0.09770825527608394 accuracy 1.0

Epoch 3/5
----------
Train loss 0.3880535748766528 accuracy 0.8466666666666667
Val   loss 0.08954287867993116 accuracy 1.0

Epoch 4/5
----------
Train loss 0.3743481410874261 accuracy 0.86
Val   loss 0.08668891083449125 accuracy 1.0

Epoch 5/5
----------
Train loss 0.36772656329804 accuracy 0.8688888888888889
Val   loss 0.08603928573429584 accuracy 1.0

CPU times: user 3h 12min 35s, sys: 8.26 s, total: 3h 12min 43s
Wall time: 48min 10s
```

Fig 49: 512-1000 training and validation metrics

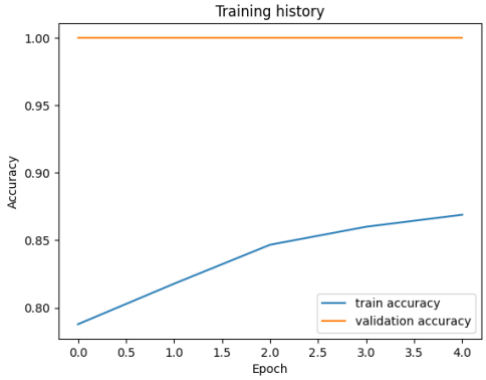

Fig 50: 512-1000 training history

```
sentence length 512
dataset size 1000
batch size 2
              precision    recall  f1-score   support

    negative     0.9387    0.9351    0.9369       524
    positive     0.9289    0.9328    0.9308       476

    accuracy                         0.9340      1000
   macro avg     0.9338    0.9339    0.9339      1000
weighted avg     0.9340    0.9340    0.9340      1000
```

Fig 51: 512-1000 classification report

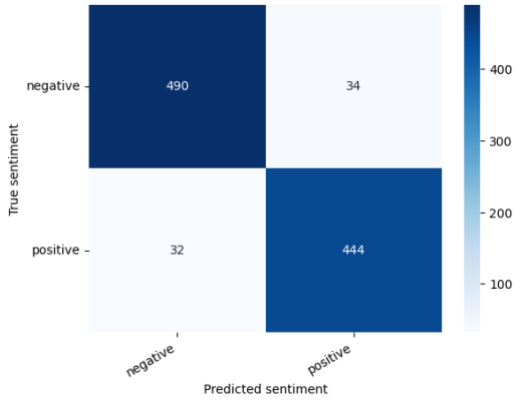

Fig 52: 512-1000 confusion matrix

Sentence Length = 32/ Dataset = 2000 samples

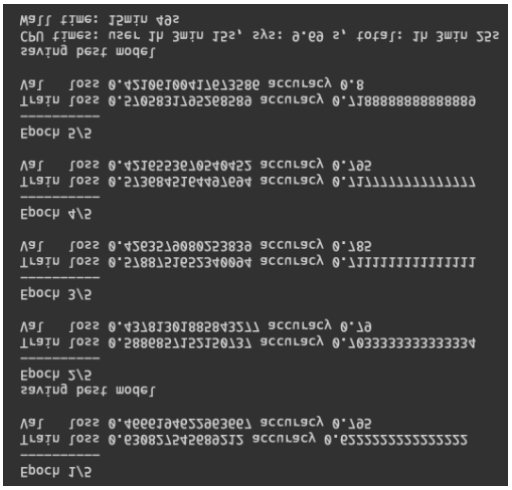

Fig 53: 32-2000 training and validation metrics

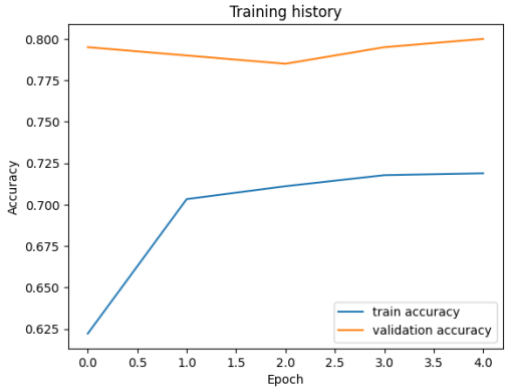

Fig 54: 32-2000 training history

```
sentence length 32
dataset size 2000
batch size 2
              precision    recall  f1-score   support

    negative     0.7594    0.7290    0.7439       524
    positive     0.7143    0.7458    0.7297       476

    accuracy                         0.7370      1000
   macro avg     0.7369    0.7374    0.7368      1000
weighted avg     0.7379    0.7370    0.7371      1000
```

Fig 55: 32-2000 classification report

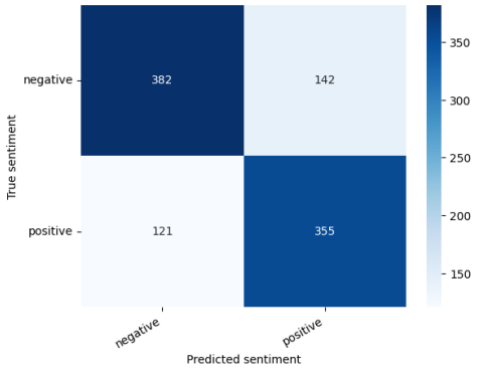

Fig 56: 32-2000 confusion matrix

Sentence Length = 64/ Dataset = 2000 samples

```
Epoch 1/5
----------
Train loss 0.5648233515686459 accuracy 0.6972222222222222
Val   loss 0.31282605867832897 accuracy 0.87

saving best model
Epoch 2/5
----------
Train loss 0.532903691712353 accuracy 0.725
Val   loss 0.29229115396738053 accuracy 0.87

Epoch 3/5
----------
Train loss 0.5222991772161589 accuracy 0.7361111111111112
Val   loss 0.2856088300421834 accuracy 0.875

saving best model
Epoch 4/5
----------
Train loss 0.5153007516430483 accuracy 0.7433333333333333
Val   loss 0.2841300847195089 accuracy 0.875

Epoch 5/5
----------
Train loss 0.5112894162287315 accuracy 0.7461111111111111
Val   loss 0.28581669913604857 accuracy 0.865

CPU times: user 1h 19min 50s, sys: 23.6 s, total: 1h 20min 13s
Wall time: 20min 1s
```

Fig 57: 64-2000 training and validation metrics

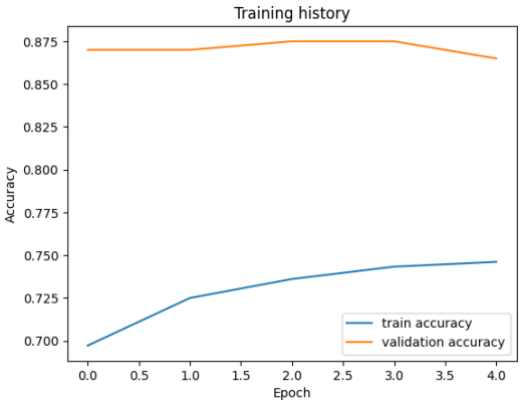

Fig 58: 64-2000 training history

```
sentence length 64
dataset size 2000
batch size 2
              precision    recall  f1-score   support

    negative     0.8769    0.7748    0.8227       524
    positive     0.7803    0.8803    0.8272       476

    accuracy                         0.8250      1000
   macro avg     0.8286    0.8275    0.8250      1000
weighted avg     0.8309    0.8250    0.8249      1000
```

Fig 59: 64-2000 classification report

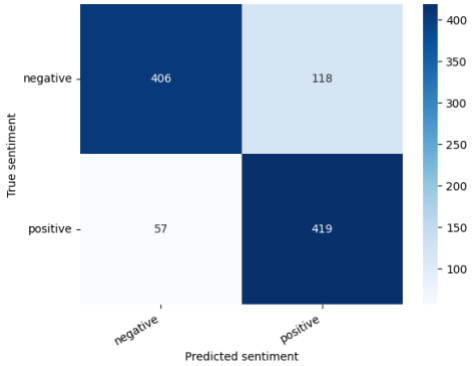

Fig 60: 64-2000 confusion matrix

Sentence Length = 128/ Dataset = 2000 samples

```
Epoch 1/5
----------
Train loss 0.5196425241563055 accuracy 0.7444444444444445
Val   loss 0.22279854044318198 accuracy 0.94

saving best model
Epoch 2/5
----------
Train loss 0.4812623718049791 accuracy 0.7738888888888888
Val   loss 0.21022095710039138 accuracy 0.93

Epoch 3/5
----------
Train loss 0.46288151719503934 accuracy 0.7883333333333333
Val   loss 0.20453081011772156 accuracy 0.93

Epoch 4/5
----------
Train loss 0.4504931193341811 accuracy 0.8027777777777778
Val   loss 0.2031116347014904 accuracy 0.925

Epoch 5/5
----------
Train loss 0.44412410260902513 accuracy 0.8105555555555556
Val   loss 0.20675463732331992 accuracy 0.92

CPU times: user 1h 33min 16s, sys: 45 s, total: 1h 34min 1s
Wall time: 23min 28s
```

Fig 61: 128-2000 training and validation metrics

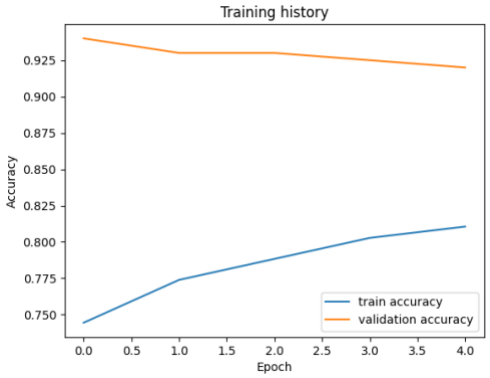

Fig 62: 128-2000 training history

```
sentence length 128
dataset size 2000
batch size 2
              precision    recall  f1-score   support

    negative     0.9424    0.8435    0.8902       524
    positive     0.8456    0.9433    0.8918       476

    accuracy                         0.8910      1000
   macro avg     0.8940    0.8934    0.8910      1000
weighted avg     0.8963    0.8910    0.8910      1000
```

Fig 63: 128-2000 classification report

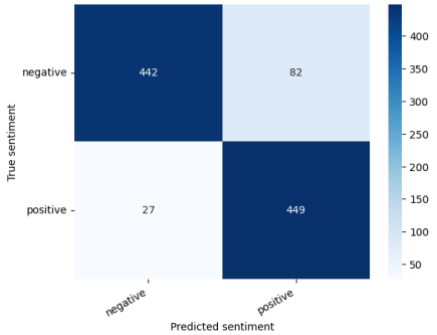

Fig 64: 128-2000 confusion matrix

Sentence Length = 256/ Dataset = 2000 samples

```
Epoch 1/5
----------
Train loss 0.47986166490448845 accuracy 0.7605555555555555
Val   loss 0.1412930953130126 accuracy 0.97

saving best model
Epoch 2/5
----------
Train loss 0.422324273660779 accuracy 0.8383333333333334
Val   loss 0.12492258077487349 accuracy 0.97

Epoch 3/5
----------
Train loss 0.39232734779516854 accuracy 0.8816666666666667
Val   loss 0.1200011870265007 accuracy 0.97

Epoch 4/5
----------
Train loss 0.3748645886116558 accuracy 0.9022222222222223
Val   loss 0.11720681304112077 accuracy 0.97

Epoch 5/5
----------
Train loss 0.36672236867249014 accuracy 0.9072222222222223
Val   loss 0.11724435674026608 accuracy 0.975

saving best model
CPU times: user 3h 29min 20s, sys: 10.2 s, total: 3h 29min 30s
Wall time: 52min 18s
```

Fig 65: 256-2000 training and validation metrics

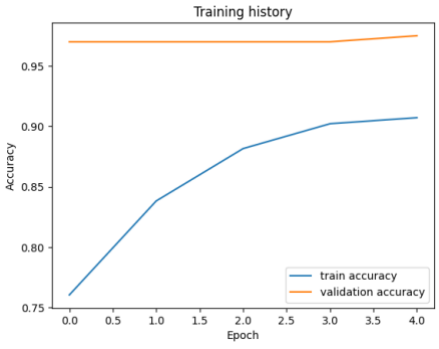

Fig 66: 256-2000 training history

```
sentence length 256
dataset size 2000
batch size 2
              precision    recall  f1-score   support

    negative     0.9266    0.9389    0.9327       524
    positive     0.9318    0.9181    0.9249       476

    accuracy                         0.9290      1000
   macro avg     0.9292    0.9285    0.9288      1000
weighted avg     0.9290    0.9290    0.9290      1000
```

Fig 67: 256-2000 classification report

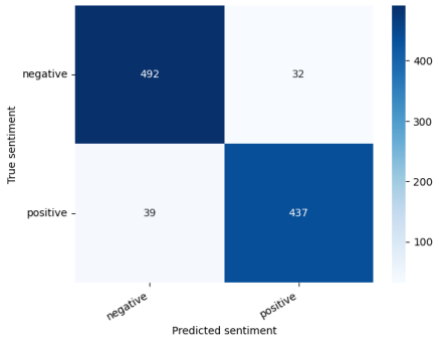

Fig 68: 256-2000 confusion matrix

Sentence Length = 512/ Dataset = 2000 samples

```
Epoch 1/5
----------
Train loss 0.48887826414571867 accuracy 0.7616666666666667
Val   loss 0.17817273784428836 accuracy 0.955

saving best model
Epoch 2/5
----------
Train loss 0.41937919947836133 accuracy 0.8433333333333334
Val   loss 0.1566531678289175 accuracy 0.96

saving best model
Epoch 3/5
----------
Train loss 0.3827933216840029 accuracy 0.8988888888888888
Val   loss 0.1494039484485984 accuracy 0.96

Epoch 4/5
----------
Train loss 0.36123500334719816 accuracy 0.9216666666666666
Val   loss 0.1454890549555421 accuracy 0.97

saving best model
Epoch 5/5
----------
Train loss 0.3509547857277923 accuracy 0.9305555555555556
Val   loss 0.14441654775291682 accuracy 0.985

saving best model
CPU times: user 6h 48min 56s, sys: 15.7 s, total: 6h 49min 12s
Wall time: 1h 42min 16s
```

Fig 69: 512-2000 training and validation metrics

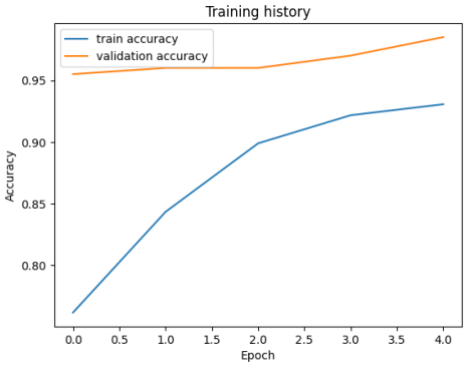

Fig 70: 512-2000 training history

```
sentence length 512
dataset size 2000
batch size 2
              precision    recall  f1-score   support

    negative     0.8873    0.9771    0.9301       524
    positive     0.9716    0.8634    0.9143       476

    accuracy                         0.9230      1000
   macro avg     0.9295    0.9203    0.9222      1000
weighted avg     0.9275    0.9230    0.9226      1000
```

Fig 71: 512-2000 classification report

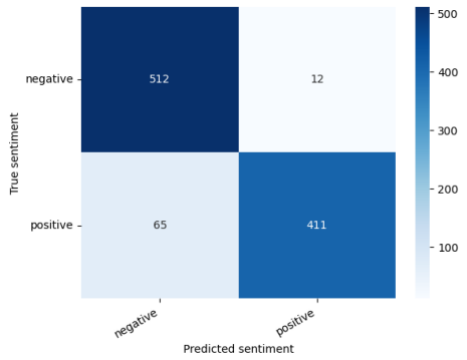

Fig 72: 512-2000 confusion matrix

Sentence Length = 32/ Dataset = 4000 samples

```
Epoch 1/5
----------
Train loss 0.5645951602542609 accuracy 0.7538888888888889
Val   loss 0.49950712493487764 accuracy 0.755

saving best model
Epoch 2/5
----------
Train loss 0.4723196045348519 accuracy 0.8005555555555556
Val   loss 0.47621970091547283 accuracy 0.755

Epoch 3/5
----------
Train loss 0.45275620828595076 accuracy 0.8041666666666667
Val   loss 0.4692482650279999 accuracy 0.7525

Epoch 4/5
----------
Train loss 0.4448610690602085 accuracy 0.8072222222222222
Val   loss 0.4666781170027597 accuracy 0.765

saving best model
Epoch 5/5
----------
Train loss 0.4415764975966069 accuracy 0.8105555555555556
Val   loss 0.46598621351378305 accuracy 0.7725

saving best model
CPU times: user 1h 22min 53s, sys: 50.9 s, total: 1h 23min 44s
Wall time: 20min 54s
```

Fig 73: 32-4000 training and validation metrics

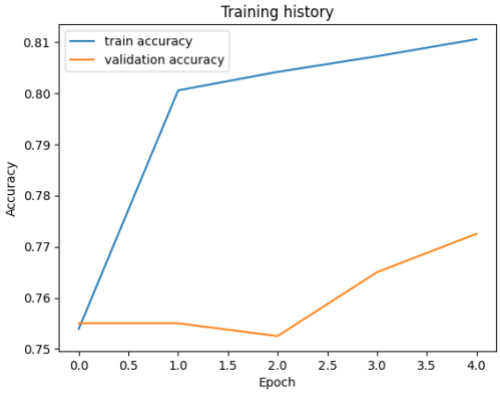

Fig 74: 32-4000 training history

```
sentence length 32
dataset size 4000
batch size 2
              precision    recall  f1-score   support

    negative     0.7842    0.7214    0.7515       524
    positive     0.7181    0.7815    0.7485       476

    accuracy                         0.7500      1000
   macro avg     0.7512    0.7514    0.7500      1000
weighted avg     0.7528    0.7500    0.7501      1000
```

Fig 75: 32-4000 classification report

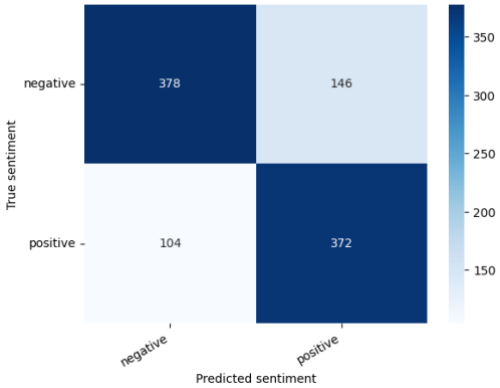

Fig 76: 32-4000 confusion matrix
Sentence Length = 64/ Dataset = 4000 samples

```
Epoch 1/5
----------
Train loss 0.5389346867276911 accuracy 0.7211111111111111
Val   loss 0.41047788092068266 accuracy 0.8675

saving best model
Epoch 2/5
----------
Train loss 0.4051118466937751 accuracy 0.8497222222222223
Val   loss 0.3565034398010799 accuracy 0.865

Epoch 3/5
----------
Train loss 0.38118619720141095 accuracy 0.8516666666666667
Val   loss 0.34151630316461834 accuracy 0.8725

saving best model
Epoch 4/5
----------
Train loss 0.37174064286968167 accuracy 0.8558333333333333
Val   loss 0.336045686687742 accuracy 0.8625

Epoch 5/5
----------
Train loss 0.3679034966125823 accuracy 0.8575
Val   loss 0.335048000727381 accuracy 0.8625

CPU times: user 2h 21min 21s, sys: 5min 30s, total: 2h 26min 51s
Wall time: 36min 46s
```

Fig 77: 64-4000 training and validation metrics

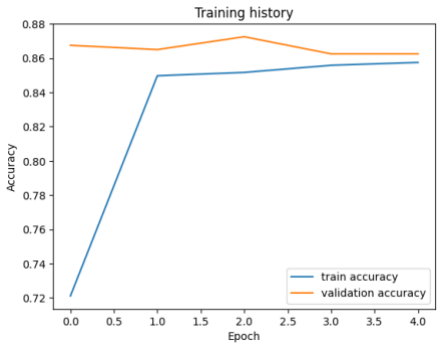

Fig 78: 64-4000 training history

```
sentence length 64
dataset size 4000
batch size 2
              precision    recall  f1-score   support

    negative     0.8404    0.8645    0.8523       524
    positive     0.8460    0.8193    0.8324       476

    accuracy                         0.8430      1000
   macro avg     0.8432    0.8419    0.8424      1000
weighted avg     0.8431    0.8430    0.8429      1000
```

Fig 79: 64-4000 classification report

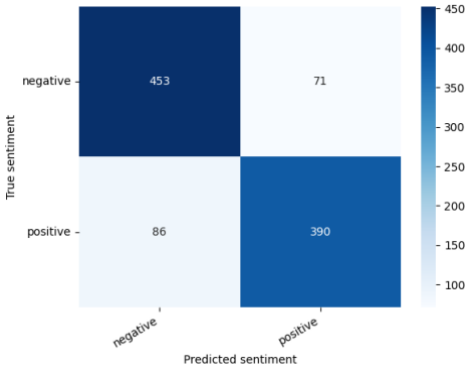

Fig 80: 64-4000 confusion matrix

Sentence Length = 128/ Dataset = 4000 samples

```
Epoch 1/5
----------
Train loss 0.44063154019807516 accuracy 0.8705555555555555
Val   loss 0.2845231166907719 accuracy 0.925

saving best model
Epoch 2/5
----------
Train loss 0.2957884078485924 accuracy 0.9236111111111112
Val   loss 0.2564712017774582 accuracy 0.93

saving best model
Epoch 3/5
----------
Train loss 0.2719670413878926 accuracy 0.9244444444444444
Val   loss 0.24129892034190042 accuracy 0.9225

Epoch 4/5
----------
Train loss 0.2623266415637836 accuracy 0.9258333333333333
Val   loss 0.2349854005234582 accuracy 0.9225

Epoch 5/5
----------
Train loss 0.2584356388501954 accuracy 0.9266666666666666
Val   loss 0.23301678683076585 accuracy 0.9225

CPU times: user 3h 21min 23s, sys: 23min 22s, total: 3h 44min 46s
Wall time: 38min 39s
```

Fig 81: 128-4000 training and validation metrics

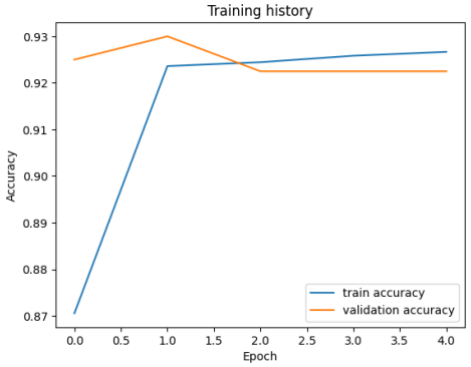

Fig 82: 128-4000 training history

```
sentence length 128
dataset size 4000
batch size 2
              precision    recall  f1-score   support

    negative     0.9286    0.8683    0.8974       524
    positive     0.8647    0.9265    0.8945       476

    accuracy                         0.8960      1000
   macro avg     0.8966    0.8974    0.8960      1000
weighted avg     0.8982    0.8960    0.8960      1000
```

Fig 83: 128-4000 classification report

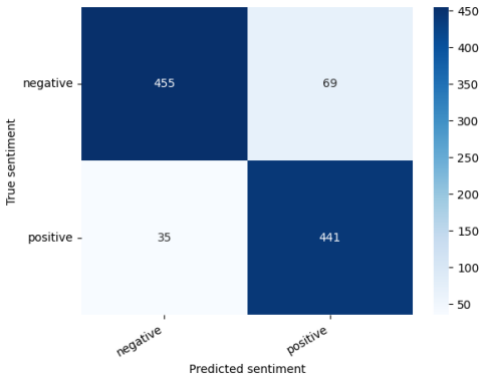

Fig 84: 128-4000 confusion matrix

Sentence Length = 256/ Dataset = 4000 samples

```
Epoch 1/5
----------
Train loss 0.4451091875109756 accuracy 0.9216666666666666
Val   loss 0.27023565982069286 accuracy 0.9725

saving best model
Epoch 2/5
----------
Train loss 0.27337138145639184 accuracy 0.9638888888888889
Val   loss 0.22820921455110824 accuracy 0.98

saving best model
Epoch 3/5
----------
Train loss 0.24003286999568604 accuracy 0.9658333333333333
Val   loss 0.20756878597395761 accuracy 0.985

saving best model
Epoch 4/5
----------
Train loss 0.2260801002644656 accuracy 0.9661111111111111
Val   loss 0.1979065431015832 accuracy 0.9825

Epoch 5/5
----------
Train loss 0.22042483222066311 accuracy 0.9666666666666667
Val   loss 0.19507742140974318 accuracy 0.9825

CPU times: user 5h 24min 22s, sys: 44min 3s, total: 6h 8min 25s
Wall time: 1h 4min 21s
```

Fig 85: 256-4000 training and validation metrics

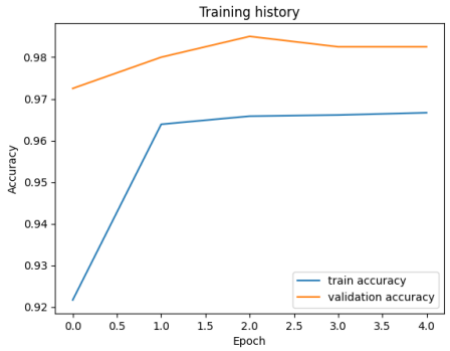

Fig 86: 256-4000 training history

```
sentence length 256
dataset size 4000
batch size 2
              precision    recall  f1-score   support

    negative     0.9264    0.9370    0.9317       524
    positive     0.9298    0.9181    0.9239       476

    accuracy                         0.9280      1000
   macro avg     0.9281    0.9275    0.9278      1000
weighted avg     0.9280    0.9280    0.9280      1000
```

Fig 87: 256-4000 classification report

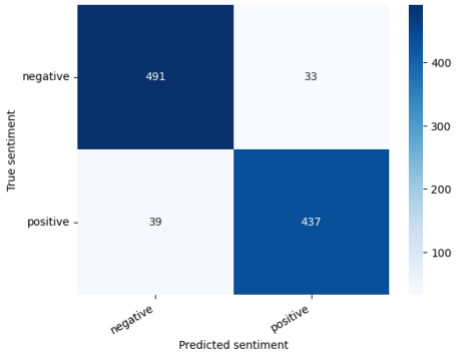

Fig 88: 256-4000 confusion matrix

Sentence Length = 512/ Dataset = 4000 samples

```
Epoch 1/5
----------
Train loss 0.414493461449941 accuracy 0.9258333333333333
Val   loss 0.2468435743025371 accuracy 0.985

saving best model
Epoch 2/5
----------
Train loss 0.23255349068265213 accuracy 0.9836111111111111
Val   loss 0.19508724127496993 accuracy 0.9875

saving best model
Epoch 3/5
----------
Train loss 0.19847465476445986 accuracy 0.9858333333333333
Val   loss 0.17558326039995467 accuracy 0.99

saving best model
Epoch 4/5
----------
Train loss 0.18492930369418964 accuracy 0.9863888888888889
Val   loss 0.16751107573509216 accuracy 0.99

Epoch 5/5
----------
Train loss 0.1795052297805485 accuracy 0.9875
Val   loss 0.1656320286648614 accuracy 0.99

CPU times: user 10h 22min 36s, sys: 1h 36min 9s, total: 11h 58min 46s
Wall time: 3h 4min 24s
```

Fig 89: 512-4000 training and validation metrics

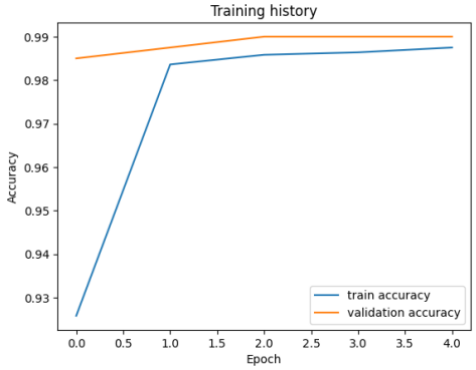

Fig 90: 512-4000 training history

```
sentence length 512
dataset size 4000
batch size 2
              precision    recall  f1-score   support

    negative     0.9380    0.9523    0.9451       524
    positive     0.9466    0.9307    0.9386       476

    accuracy                         0.9420      1000
   macro avg     0.9423    0.9415    0.9418      1000
weighted avg     0.9421    0.9420    0.9420      1000
```

Fig 91: 512-4000 classification report

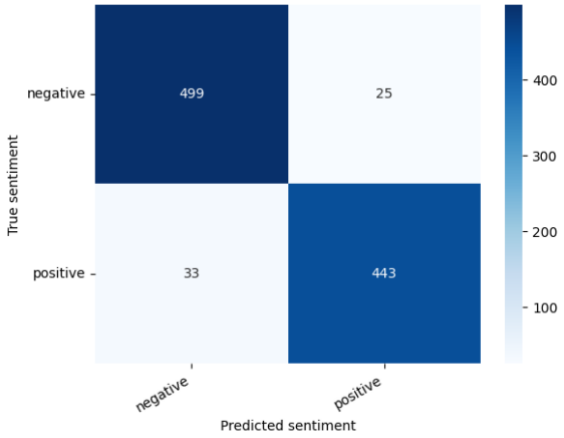

Fig 92: 512-4000 confusion matrix

Sentence Length = 32/ Dataset = 8000 samples

```
Epoch 1/5
----------
Train loss 0.5301053903799141 accuracy 0.7490277777777777
Val   loss 0.4543740015763503 accuracy 0.8

saving best model
Epoch 2/5
----------
Train loss 0.4492638596391256 accuracy 0.7877777777777778
Val   loss 0.4417905257298396 accuracy 0.79625

Epoch 3/5
----------
Train loss 0.4391687515035l4 accuracy 0.7901388888888888
Val   loss 0.43885968990692723 accuracy 0.8

Epoch 4/5
----------
Train loss 0.4351286616473071 accuracy 0.7918055555555555
Val   loss 0.43654064260996306 accuracy 0.8

Epoch 5/5
----------
Train loss 0.4333132186294657 accuracy 0.7931944444444444
Val   loss 0.43605093314097476 accuracy 0.7975

CPU times: user 2h 38min 59s, sys: 7min 51s, total: 2h 46min 51s
Wall time: 41min 39s
```

Fig 93: 32-8000 training and validation metrics

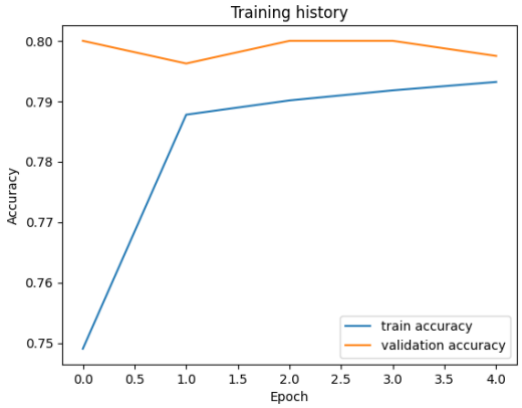

Fig 94: 32-8000 training history

```
sentence length 32
dataset size 8000
batch size 2
              precision    recall  f1-score   support

    negative     0.7954    0.7195    0.7555       524
    positive     0.7205    0.7962    0.7565       476

    accuracy                         0.7560      1000
   macro avg     0.7579    0.7578    0.7560      1000
weighted avg     0.7597    0.7560    0.7560      1000
```

Fig 95: 32-8000 classification report

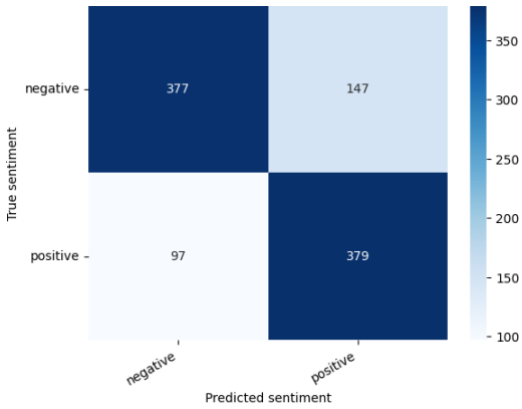

Fig 96: 32-8000 confusion matrix

Sentence Length = 64/ Dataset = 8000 samples

```
Epoch 1/5
----------
Train loss 0.4463147154424043 accuracy 0.8438888888888889
Val   loss 0.35506051549544704 accuracy 0.88375

saving best model
Epoch 2/5
----------
Train loss 0.3494618796141802 accuracy 0.8636111111111111
Val   loss 0.33132764582450575 accuracy 0.8825

Epoch 3/5
----------
Train loss 0.3352575585641692 accuracy 0.8652777777777778
Val   loss 0.32308655748000514 accuracy 0.88125

Epoch 4/5
----------
Train loss 0.3295449379798585 accuracy 0.8666666666666667
Val   loss 0.3196130876357739 accuracy 0.88125

Epoch 5/5
----------
Train loss 0.3271477855412306 accuracy 0.8668055555555556
Val   loss 0.31872570514678955 accuracy 0.88125

CPU times: user 4h 30min 53s, sys: 14min 21s, total: 4h 45min 14s
Wall time: 1h 11min 29s
```

Fig 97: 64-8000 training and validation metrics

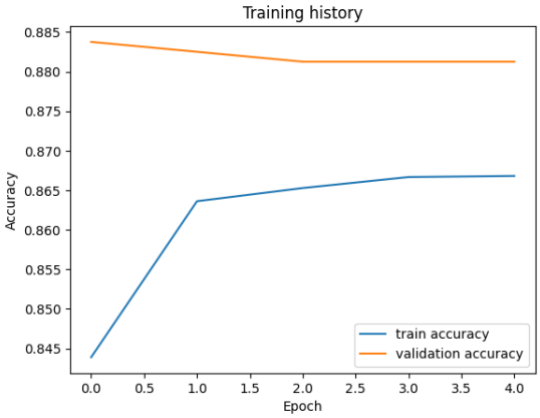

Fig 98: 64-8000 training history

```
sentence length 64
dataset size 8000
batch size 2
              precision    recall  f1-score   support

    negative     0.8711    0.7996    0.8338       524
    positive     0.7977    0.8697    0.8322       476

    accuracy                         0.8330      1000
   macro avg     0.8344    0.8347    0.8330      1000
weighted avg     0.8362    0.8330    0.8330      1000
```

Fig 99: 64-8000 classification report

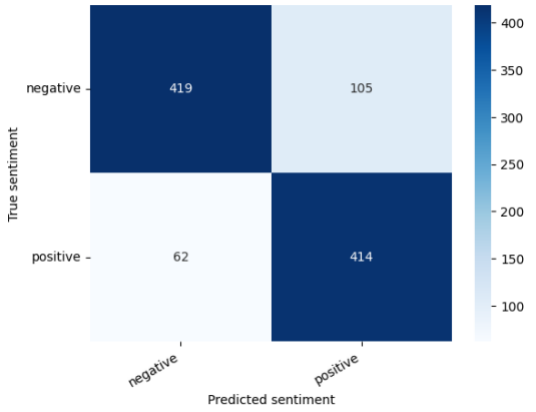

Fig 100: 64-8000 confusion matrix
Sentence Length = 128/ Dataset = 8000 samples

```
Epoch 1/5
----------
Train loss 0.3808023485198485 accuracy 0.8747222222222222
Val   loss 0.27109406888484955 accuracy 0.93875

saving best model
Epoch 2/5
----------
Train loss 0.2637892362818254 accuracy 0.9273611111111111
Val   loss 0.2423832084123905 accuracy 0.94

saving best model
Epoch 3/5
----------
Train loss 0.24620209937602017 accuracy 0.9276388888888889
Val   loss 0.23207519146112296 accuracy 0.9425

saving best model
Epoch 4/5
----------
Train loss 0.23871453159150824 accuracy 0.9279166666666666
Val   loss 0.22796944013008705 accuracy 0.9425

Epoch 5/5
----------
Train loss 0.23557712928911226 accuracy 0.9279166666666666
Val   loss 0.22702812231504 accuracy 0.94

CPU times: user 7h 39min 18s, sys: 40min 11s, total: 8h 19min 30s
Wall time: 2h 5min 46s
```

Fig 101: 128-8000 training and validation metrics

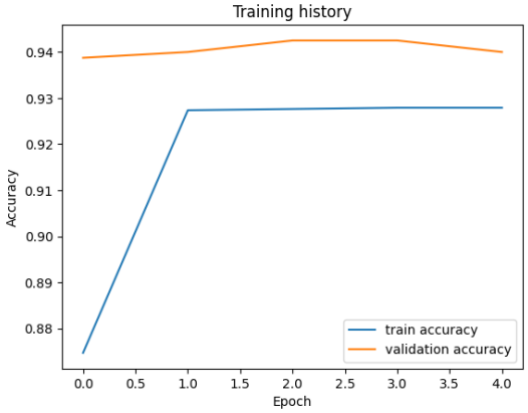

Fig 102: 128-8000 training history

```
sentence length 128
dataset size 8000
batch size 2
              precision    recall  f1-score   support

    negative     0.9274    0.8779    0.9020       524
    positive     0.8730    0.9244    0.8980       476

    accuracy                         0.9000      1000
   macro avg     0.9002    0.9011    0.9000      1000
weighted avg     0.9015    0.9000    0.9001      1000
```

Fig 103: 128-8000 classification report

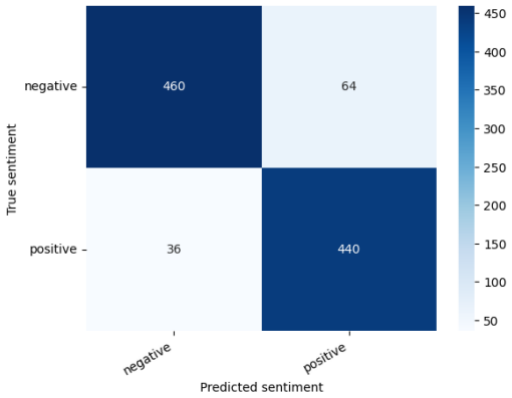

Fig 104: 128-8000 confusion matrix
Sentence Length = 256/ Dataset = 8000 samples

```
Epoch 1/5
----------
Train loss 0.31520527535307724 accuracy 0.9526388888888889
Val   loss 0.1978627500625757 accuracy 0.9675

saving best model
Epoch 2/5
----------
Train loss 0.19317521510925967 accuracy 0.9686111111111111
Val   loss 0.1692023787360925 accuracy 0.97125

saving best model
Epoch 3/5
----------
Train loss 0.17375373306263864 accuracy 0.97
Val   loss 0.15929837582203057 accuracy 0.97

Epoch 4/5
----------
Train loss 0.1652054404262948 accuracy 0.9718055555555556
Val   loss 0.1551820463859118 accuracy 0.9725

saving best model
Epoch 5/5
----------
Train loss 0.16148622999940299 accuracy 0.9718055555555556
Val   loss 0.15395834812751183 accuracy 0.97

CPU times: user 10h 51min 35s, sys: 1h 36min 29s, total: 12h 28min 4s
Wall time: 3h 10min 12s
```

Fig 105: 256-8000 training and validation metrics

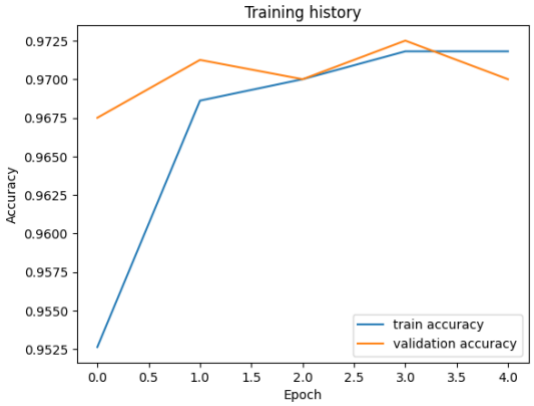

Fig 106: 256-8000 training history

```
sentence length 256
dataset size 8000
batch size 2
              precision    recall  f1-score   support

    negative     0.9442    0.9046    0.9240       524
    positive     0.8996    0.9412    0.9199       476

    accuracy                         0.9220      1000
   macro avg     0.9219    0.9229    0.9219      1000
weighted avg     0.9230    0.9220    0.9220      1000
```

Fig 107: 256-8000 classification report

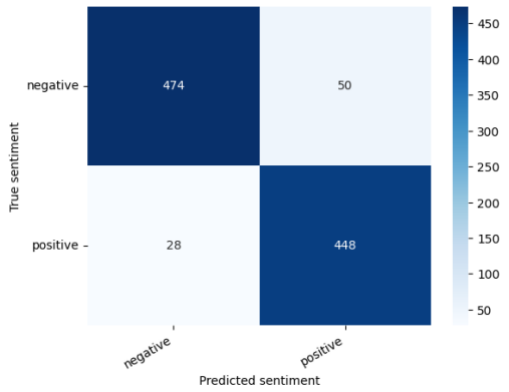

Fig 108: 256-8000 confusion matrix
Sentence Length = 512/ Dataset = 8000 samples

```
Epoch 1/5
----------
Train loss 0.30285046928751785 accuracy 0.9591666666666666
Val   loss 0.18182240426540375 accuracy 0.985

saving best model
Epoch 2/5
----------
Train loss 0.17332432036642478 accuracy 0.9876388888888888
Val   loss 0.1463826562349613 accuracy 0.9875

saving best model
Epoch 3/5
----------
Train loss 0.1506282716198305 accuracy 0.9877777777777778
Val   loss 0.13390946502868945 accuracy 0.98625

Epoch 4/5
----------
Train loss 0.14083838423268985 accuracy 0.9875
Val   loss 0.13012812229303214 accuracy 0.98625

Epoch 5/5
----------
Train loss 0.13645979893946014 accuracy 0.9883333333333333
Val   loss 0.12888404383109167 accuracy 0.9875
```

Fig 109: 512-8000 training and validation metrics

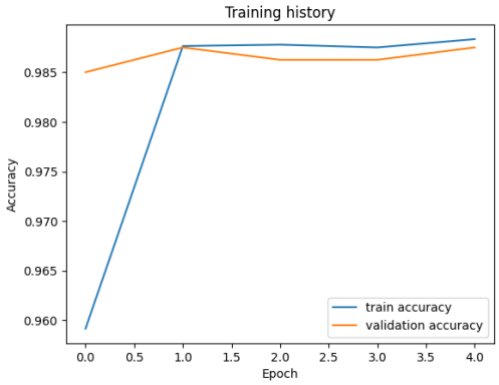

Fig 110: 512-8000 training history

```
sentence length 512
dataset size 8000
batch size 2
              precision    recall  f1-score   support

    negative     0.9485    0.9141    0.9310       524
    positive     0.9091    0.9454    0.9269       476

    accuracy                         0.9290      1000
   macro avg     0.9288    0.9298    0.9289      1000
weighted avg     0.9297    0.9290    0.9290      1000
```

Fig 111: 512-8000 classification report

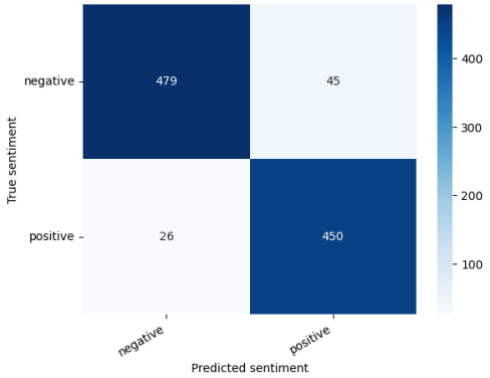

Fig 112: 512-8000 confusion matrix

The images below are a visual representation of the metrics contained in the classification report, grouped by dataset size. From these visualizations, a pattern can be established. As the

sentence length increases, the values of accuracy, precision, recall, and f1-score increase. This tells us that when the sentence length in the dataset increases, the accuracy of the Densenet model for classification increases also. In addition, the dataset size was increased for the experiments after each iteration to show that as the dataset size increases, the accuracy of the architecture also increases. Further buttressing the point that the accuracy of the architecture increases with an increase in dataset size as well.

4.3 Classification Report Visualizations

Fig 113: Classification report visualization for dataset size 500

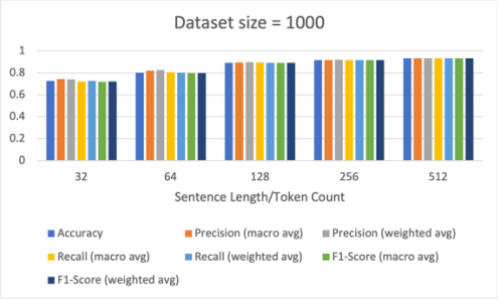

Fig 114: Classification report visualization for dataset size 1000

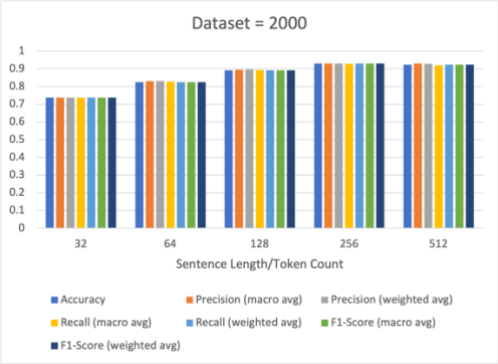

Fig 115: Classification report visualization for dataset size 2000

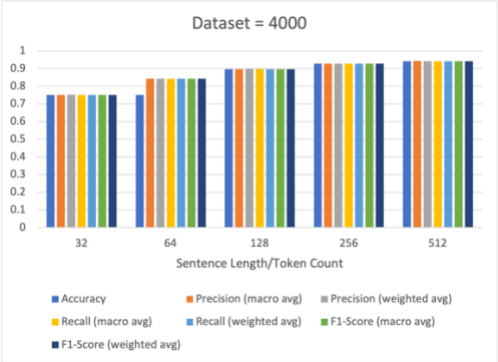

Fig 116: Classification report visualization for dataset size 4000

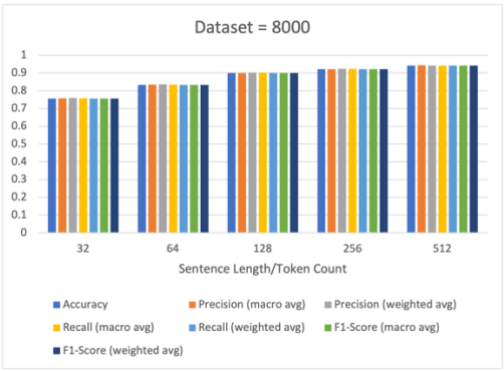

Fig 117: Classification report visualization for dataset size 8000

The images above (Fig. 113 – Fig. 117) showcase the degree of accuracy a 2D CNN-Densenet architecture possesses. These results not only show how well this combination of a 2D CNN model and a Densenet architecture perform but further show the impact sentence length has on the accuracy of the sentiment analysis task by the model.

4.4 Dataset size vs. accuracy for varying sentence lengths

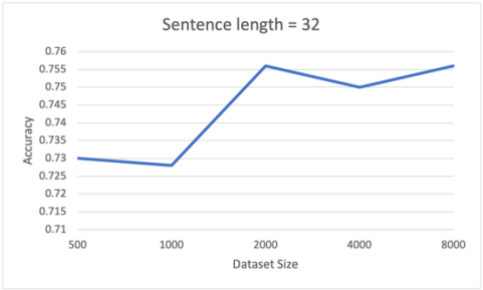

Fig. 118: Accuracy vs. Dataset Size for Sentence Length 32

Fig. 119: Accuracy vs. Dataset Size for Sentence Length 64

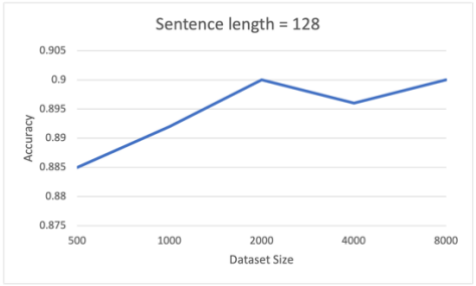

Fig. 120: Accuracy vs. Dataset Size for Sentence Length 128

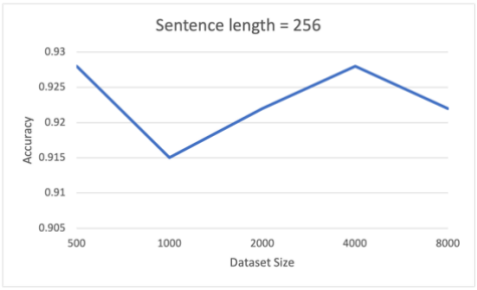

Fig. 121: Accuracy vs. Dataset Size for Sentence Length 256

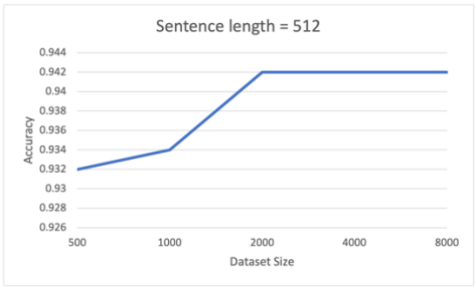

Fig. 122: Accuracy vs. Dataset Size for Sentence Length 512

The images above (Fig. 118 – Fig. 122) reflect the effect dataset size has on the accuracy of Sentiment analysis classification. In these Images, it is evident that as the dataset size increases, the accuracy of the classification also increases. We observe that the model accuracy dips with some of the samples. However, it is noticed that, for each dataset size and sentence length, the model accuracy increases as the dataset increases. This is to say that dataset size does indeed have an effect on sentence length.

Chapter 5: Conclusion, Challenges, Observations, Recommendations and Future Work

5.1 Conclusion

This research had the aim of determining whether a 2D CNN Densenet architecture could be used for sentiment analysis classifications, while also investigating whether the length of the sentences has an impact on the accuracy of the classifications. To the best of my knowledge and research, this is an area that has not been ventured as Densenet is most practical for image classifications. Experiments were conducted and the results were analyzed in Chapter 4. The results obtained were positive and pointed in the direction of a successful classification, while also backing the point that sentence length influences the accuracy of classifications. The literature review in combination with the experiments and results were able to answer and validate the proposed hypotheses.

The hypotheses and research questions posed were:
Null hypothesis 1: Densenet architecture cannot classify data for sentiment analysis.
Null hypothesis 2: Sentence length does not impact accuracy.

To properly direct the research, the following research questions were posed:

Research question 1: Can DenseNet architecture classify textual data for sentiment analysis?
Research question 2: Does sentence length impact the accuracy of classification?
Research question 3: Does dataset size impact accuracy?

Considering the results of the experiments, the following can be said:

The use of the two-dimensional CNN Densenet architecture performed very well for the sentiment analysis classifications. I accurately classified the data and proved that Densenet could be used for sentiment analysis tasks.

The higher the sentence length, the more accurate the classification.

5.2. Challenges and Observations

Sentiment analysis and NLP classification tasks often require a high level of computing power to run through the dataset for classifications. The major challenge faced during this research was the unavailability of enough resources to further explore the dataset. The CPU and GPU capabilities of my computer were a problem as they were unable to run the code locally. Google Colab was relied on to conduct these experiments as it provided me with enough computing power to run and execute code for the experimentation process.

5.3 Recommendations

This research encourages more time and investment into DenseNet as an alternative to existing architecture for NLP tasks.

5.4 Future Works

Future works of this research will be aimed at incorporating Densenet-121 architecture as well as the other Densenet architectures to determine the best-performing Densenet model for NLP classification tasks.
In the future, this model will be tested on an imbalanced dataset to investigate whether the accuracy of the model increases as well. Also, the reason why the accuracy for varying dataset sizes dropped for the various sentence lengths will be explored in future works conducted on this research.

References

Adi, Y. *et al.* (2017) 'Analysis of sentence embedding models using prediction tasks in natural language processing', *IBM Journal of Research and Development*, 61(4/5), p. 3:1-3:9. Available at: https://doi.org/10.1147/JRD.2017.2702858.

Ahmad, M., Aftab, S. and Ali, I. (2017) 'Sentiment Analysis of Tweets using SVM', *International Journal of Computer Applications*, 177, pp. 975–8887. Available at: https://doi.org/10.5120/ijca2017915758.

Aich, S. *et al.* (2021) 'Bidirectional Attention Network for Monocular Depth Estimation', in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11746–11752. Available at: https://doi.org/10.1109/ICRA48506.2021.9560885.

Alberti, C., Lee, K. and Collins, M. (2019) 'A BERT Baseline for the Natural Questions'. arXiv. Available at: http://arxiv.org/abs/1901.08634 (Accessed: 26 August 2023).

Alsentzer, E. *et al.* (2019) 'Publicly Available Clinical BERT Embeddings'. arXiv. Available at: http://arxiv.org/abs/1904.03323 (Accessed: 26 August 2023).

Anand, S.S., Bell, D.A. and Hughes, J.G. (1995) 'The role of domain knowledge in data mining', in *Proceedings of the fourth international conference on Information and knowledge management - CIKM '95*. *the fourth international conference*, Baltimore, Maryland, United States: ACM Press, pp. 37–43. Available at: https://doi.org/10.1145/221270.221321.

Bahgat, M., Wilson, S. and Magdy, W. (2020) 'Towards Using Word Embedding Vector Space for Better Cohort Analysis', *Proceedings of the International AAAI Conference on Web and Social Media*, 14, pp. 919–923. Available at: https://doi.org/10.1609/icwsm.v14i1.7358.

Barbon, R.S. and Akabane, A.T. (2022) *Sensors | Free Full-Text | Towards Transfer Learning Techniques—BERT, DistilBERT, BERTimbau, and DistilBERTimbau for Automatic Text Classification from Different Languages: A Case Study*. Available at: https://www.mdpi.com/1424-8220/22/21/8184 (Accessed: 25 August 2023).

Bhoir, S., Ghorpade, T. and Mane, V. (2017) 'Comparative analysis of different word embedding models', in *2017 International Conference on Advances in Computing, Communication and Control (ICAC3)*. *2017 International Conference on Advances in Computing, Communication and Control (ICAC3)*, pp. 1–4. Available at: https://doi.org/10.1109/ICAC3.2017.8318770.

Bitton, Y. *et al.* (2021) 'Data Efficient Masked Language Modeling for Vision and Language'. arXiv. Available at: https://doi.org/10.48550/arXiv.2109.02040.

Bodapati, J., Veeranjaneyulu, N. and Shaik, S. (2019) 'Sentiment Analysis from Movie Reviews Using LSTMs', *Ingénierie des systèmes d information*, 24(1), pp. 125–129. Available at: https://doi.org/10.18280/isi.240119.

Borzunov, A. *et al.* (2022) 'Training Transformers Together', in *Proceedings of the NeurIPS 2021 Competitions and Demonstrations Track*. *NeurIPS 2021 Competitions and Demonstrations Track*, PMLR, pp. 335–342. Available at: https://proceedings.mlr.press/v176/borzunov22a.html (Accessed: 20 August 2023).

Chen, Y., Wu, L. and Zaki, M.J. (2019) 'Bidirectional Attentive Memory Networks for Question Answering over Knowledge Bases'. arXiv. Available at: https://doi.org/10.48550/arXiv.1903.02188.

Cheng, J. *et al.* (2021) 'Multimodal Phased Transformer for Sentiment Analysis', in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. *EMNLP 2021*, Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, pp. 2447–2458. Available at: https://doi.org/10.18653/v1/2021.emnlp-main.189.

DenseNet PyTorch class (2019). Available at: https://www.pytorch.org (Accessed: 21 August 2023).

Dey, L. *et al.* (2016) 'Sentiment Analysis of Review Datasets Using Naive Bayes and K-NN Classifier', *International Journal of Information Engineering and Electronic Business*, 8(4), pp. 54–62. Available at: https://doi.org/10.5815/ijieeb.2016.04.07.

El Boukkouri, H. *et al.* (2019) 'Embedding Strategies for Specialized Domains: Application to Clinical Entity Recognition', in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*. Florence, Italy: Association for Computational Linguistics, pp. 295–301. Available at: https://doi.org/10.18653/v1/P19-2041.

Garí Soler, A. and Apidianaki, M. (2021) 'Let's Play Mono-Poly: BERT Can Reveal Words' Polysemy Level and Partitionability into Senses', *Transactions of the Association for Computational Linguistics*, 9, pp. 825–844. Available at: https://doi.org/10.1162/tacl_a_00400.

Gu, S. *et al.* (2018) 'A Position-aware Bidirectional Attention Network for Aspect-level Sentiment Analysis', in *Proceedings of the 27th International Conference on Computational Linguistics*. *COLING 2018*, Santa Fe, New Mexico, USA: Association for Computational Linguistics, pp. 774–784. Available at: https://aclanthology.org/C18-1066 (Accessed: 25 August 2023).

Hoang, M., Bihorac, O.A. and Rouces, J. (2019) 'Aspect-Based Sentiment Analysis using BERT', in *Proceedings of the 22nd Nordic Conference on Computational Linguistics*. *NoDaLiDa 2019*, Turku, Finland: Linköping University Electronic Press, pp. 187–196. Available at: https://aclanthology.org/W19-6120 (Accessed: 25 August 2023).

IMDB Dataset of 50K Movie Reviews (2011). Available at: https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews (Accessed: 23 August 2023).

Jing, L.-P., Huang, H.-K. and Shi, H.-B. (2002) 'Improved feature selection approach TFIDF in text mining', in *Proceedings. International Conference on Machine Learning and Cybernetics*.

Proceedings. International Conference on Machine Learning and Cybernetics, pp. 944–946 vol.2. Available at: https://doi.org/10.1109/ICMLC.2002.1174522.

Kalyan, K.S., Rajasekharan, A. and Sangeetha, S. (2021) 'AMMUS : A Survey of Transformer-based Pretrained Models in Natural Language Processing'. arXiv. Available at: https://doi.org/10.48550/arXiv.2108.05542.

Khadhraoui, M. *et al.* (2022) 'Survey of BERT-Base Models for Scientific Text Classification: COVID-19 Case Study', *Applied Sciences*, 12(6), p. 2891. Available at: https://doi.org/10.3390/app12062891.

Kim, S.-M. and Hovy, E. (2004) 'Determining the Sentiment of Opinions', in *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*. *COLING 2004*, Geneva, Switzerland: COLING, pp. 1367–1373. Available at: https://aclanthology.org/C04-1200 (Accessed: 14 August 2023).

Lai, T.M. *et al.* (2020) 'A Simple But Effective Bert Model for Dialog State Tracking on Resource-Limited Systems', in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8034–8038. Available at: https://doi.org/10.1109/ICASSP40776.2020.9053975.

Li, D. and Qian, J. (2016) 'Text sentiment analysis based on long short-term memory', in *2016 First IEEE International Conference on Computer Communication and the Internet (ICCCI)*. *2016 First IEEE International Conference on Computer Communication and the Internet (ICCCI)*, pp. 471–475. Available at: https://doi.org/10.1109/CCI.2016.7778967.

Li, Y. *et al.* (2021) 'Benchmarking Detection Transfer Learning with Vision Transformers'. arXiv. Available at: https://doi.org/10.48550/arXiv.2111.11429.

Lin, Y. *et al.* (2022) 'BertGCN: Transductive Text Classification by Combining GCN and BERT'. arXiv. Available at: http://arxiv.org/abs/2105.05727 (Accessed: 26 August 2023).

Liu, Q. *et al.* (2022) 'Open-World Semantic Segmentation via Contrasting and Clustering Vision-Language Embedding', in S. Avidan et al. (eds) *Computer Vision – ECCV 2022*. Cham: Springer Nature Switzerland (Lecture Notes in Computer Science), pp. 275–292. Available at: https://doi.org/10.1007/978-3-031-20044-1_16.

Medsker, L.R. and Jain, L.C. (2001) *CRC Press LLC*. The CRC Press International Series on Computational Intelligence. Available at: https://d1wqtxts1xzle7.cloudfront.net/31279335/___Recurrent_Neural_Networks_Design_And_Applicatio%28BookFi.org%29-libre.pdf?1390940527=&response-content-disposition=inline%3B+filename%3DEffects_of_Liquid_and_Encapsulated_Lacti.pdf&Expires=1692534650&Signature=N5NEaJhFPPT-USkfKiYfbNyt25bH8iFYdarpOpylXW0W1wEMx0uN4RJ0Bxh6VqXgoP6ra3~myWbDRiamrLhvhBta-fexOKANff-AXKlfqcP1BLngwyTCbX1va~8wjIjsOrfBeFJnRT1pMBC2Z14tw5oXZlQDTeeb5ahFyUi5w07jOFeQaQFtcO9hN-g~Q7d1K1SQWOVr1xE2qCdSnNQS0tD1kQYd008otM-

CFDXTDwDCumAbE1bvjl4xdsJiWzNeqzQhX4x8X0nib9hKW47ulwjlr1oY01RKNGi75MQKeb354
2y93gRsV87xcETHTGqFxsClat7T0WU3l0Vt9DzS0w__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA.

Mejova, Y. (2009) 'Sentiment Analysis: An Overview'.

Merritt, R. (2022) *What Is a Transformer Model?*, *NVIDIA Blog*. Available at: https://blogs.nvidia.com/blog/2022/03/25/what-is-a-transformer-model/ (Accessed: 20 August 2023).

Nugroho, A. and Suhartanto, H. (2020) 'Hyper-Parameter Tuning based on Random Search for DenseNet Optimization', in *2020 7th International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE)*. *2020 7th International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE)*, pp. 96–99. Available at: https://doi.org/10.1109/ICITACEE50144.2020.9239164.

Ouyang, X. *et al.* (2015) 'Sentiment Analysis Using Convolutional Neural Network', in *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*. *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*, pp. 2359–2364. Available at: https://doi.org/10.1109/CIT/IUCC/DASC/PICOM.2015.349.

Pennington, J., Socher, R. and Manning, C. (2014) 'GloVe: Global Vectors for Word Representation', in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. *EMNLP 2014*, Doha, Qatar: Association for Computational Linguistics, pp. 1532–1543. Available at: https://doi.org/10.3115/v1/D14-1162.

Pogiatzis, A. (2019) *NLP: Contextualized word embeddings from BERT*, *Medium*. Available at: https://towardsdatascience.com/nlp-extract-contextualized-word-embeddings-from-bert-keras-tf-67ef29f60a7b (Accessed: 23 August 2023).

Prakash, T.N. and Aloysius, A. (2020) 'A Comparative study of Lexicon based and Machine learning based Classifications in Sentiment analysis', *International Journal of Data Mining Techniques and Applications*, 9(2). Available at: https://doi.org/10.20894/IJDMTA.102.009.002.001.

Ramchand, G. and Svenonius, P. (2002) 'The Lexical Syntax and Lexical Semantics of the Verb-Particle Construction'. Available at: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=26160e796a58259191c31c06978a0d2493865829 (Accessed: 25 August 2023).

Rönnqvist, S. *et al.* (2019) 'Is Multilingual BERT Fluent in Language Generation?' arXiv. Available at: https://doi.org/10.48550/arXiv.1910.03806.

Salur, M.U. and Aydin, I. (2020) 'A Novel Hybrid Deep Learning Model for Sentiment Classification', *IEEE Access*, 8, pp. 58080–58093. Available at: https://doi.org/10.1109/ACCESS.2020.2982538.

Shangipour ataei, T., Javdan, S. and Minaei-Bidgoli, B. (2020) 'Applying Transformers and Aspect-based Sentiment Analysis approaches on Sarcasm Detection', in *Proceedings of the Second Workshop on Figurative Language Processing*. *Fig-Lang 2020*, Online: Association for Computational Linguistics, pp. 67–71. Available at: https://doi.org/10.18653/v1/2020.figlang-1.9.

Singh, H. *et al.* (2022a) 'Sentiment Analysis using BLSTM-ResNet on Textual Images', in *2022 International Joint Conference on Neural Networks (IJCNN)*. *2022 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. Available at: https://doi.org/10.1109/IJCNN55064.2022.9892883.

Singh, H. *et al.* (2022b) 'Sentiment Analysis using BLSTM-ResNet on Textual Images', in *2022 International Joint Conference on Neural Networks (IJCNN)*. *2022 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. Available at: https://doi.org/10.1109/IJCNN55064.2022.9892883.

Søgaard, A. *et al.* (2014) 'What's in a p-value in NLP?', in *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*. *CoNLL 2014*, Ann Arbor, Michigan: Association for Computational Linguistics, pp. 1–10. Available at: https://doi.org/10.3115/v1/W14-1601.

Sun, C. *et al.* (2019) 'How to Fine-Tune BERT for Text Classification?', in M. Sun et al. (eds) *Chinese Computational Linguistics*. Cham: Springer International Publishing (Lecture Notes in Computer Science), pp. 194–206. Available at: https://doi.org/10.1007/978-3-030-32381-3_16.

Takahashi, N. and Mitsufuji, Y. (2021) 'Densely Connected Multi-Dilated Convolutional Networks for Dense Prediction Tasks', in. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 993–1002. Available at: https://openaccess.thecvf.com/content/CVPR2021/html/Takahashi_Densely_Connected_Multi-Dilated_Convolutional_Networks_for_Dense_Prediction_Tasks_CVPR_2021_paper.html (Accessed: 26 August 2023).

Tang, D., Qin, B. and Liu, T. (2015) 'Document Modeling with Gated Recurrent Neural Network for Sentiment Classification', in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. *EMNLP 2015*, Lisbon, Portugal: Association for Computational Linguistics, pp. 1422–1432. Available at: https://doi.org/10.18653/v1/D15-1167.

Wang, L., Meng, Z. and Yang, L. (2022) 'A multi-layer two-dimensional convolutional neural network for sentiment analysis', *International Journal of Bio-Inspired Computation*, 19(2), pp. 97–107. Available at: https://doi.org/10.1504/IJBIC.2022.121236.

Wang, Y., Cui, L. and Zhang, Y. (2021) 'Improving Skip-Gram Embeddings Using BERT', *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 29, pp. 1318–1328. Available at: https://doi.org/10.1109/TASLP.2021.3065201.

Xu, B. *et al.* (2019) 'An Adaptive Wordpiece Language Model for Learning Chinese Word Embeddings', in *2019 IEEE 15th International Conference on Automation Science and*

Engineering (CASE). *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, pp. 812–817. Available at: https://doi.org/10.1109/COASE.2019.8843151.

Yamashita, R. *et al.* (2018) 'Convolutional neural networks: an overview and application in radiology', *Insights into Imaging*, 9(4), pp. 611–629. Available at: https://doi.org/10.1007/s13244-018-0639-9.

Yang, G. *et al.* (2018) 'Dual-Channel Densenet for Hyperspectral Image Classification', in *IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium*. *IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium*, pp. 2595–2598. Available at: https://doi.org/10.1109/IGARSS.2018.8517520.

Yao, L., Mao, C. and Luo, Y. (2019) 'Clinical text classification with rule-based features and knowledge-guided convolutional neural networks', *BMC Medical Informatics and Decision Making*, 19(3), p. 71. Available at: https://doi.org/10.1186/s12911-019-0781-4.

Yenicelik, D., Schmidt, F. and Kilcher, Y. (2020) 'How does BERT capture semantics? A closer look at polysemous words', in *Proceedings of the Third BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*. *BlackboxNLP 2020*, Online: Association for Computational Linguistics, pp. 156–162. Available at: https://doi.org/10.18653/v1/2020.blackboxnlp-1.15.

Appendix A: BERT-DenseNet code

Analysis metrics code

```
n = 8000 #number of samples MAx-> 49000
random_state = 42
EPOCHS = 5
sen_length = 512 #maximun length of tokens in a document (chosen based
on distribution)

Pre_trained_model='bert-base-uncased'
batch_size=2

PATH = "SentimentBert-WithoutStopWordsUncasedRemoved@urlsDigits.pt"
#saving best model with parameters

learning_rate = 2e-5
momentum = 0.5
log_interval = 10
```

Sentiment Classifier Code

```
class SentimentClassifier(nn.Module):

  def __init__(self, n_classes):
    super(SentimentClassifier, self).__init__()
    self.bert = BertModel.from_pretrained(PRE_TRAINED_MODEL_NAME,
return_dict=False)
    self.drop = nn.Dropout(p=0.3)
    self.last = nn.Linear(self.bert.config.hidden_size, n_classes)
    self.activ = torch.nn.Sigmoid()

  def forward(self, input_ids, attention_mask):
    _, pooled_output = self.bert(
      input_ids=input_ids,
      attention_mask=attention_mask
    )
    output = self.drop(pooled_output)
    output = self.last(output)

    return self.activ(output)
```

Densenet architecture for feature extraction code

```
import re
from collections import OrderedDict
from functools import partial
from typing import Any, List, Optional, Tuple
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.utils.checkpoint as cp
from torch import Tensor

__all__ = [
    "DenseNet",
    "DenseNet121_Weights",
    "DenseNet161_Weights",
    "DenseNet169_Weights",
    "DenseNet201_Weights",
    "densenet121",
    "densenet161",
    "densenet169",
    "densenet201",
]

class _DenseLayer(nn.Module):
    def __init__(
        self, num_input_features: int, growth_rate: int, bn_size:
int, drop_rate: float, memory_efficient: bool = False
    ) -> None:
        super().__init__()
        self.norm1 = nn.BatchNorm2d(num_input_features)
        self.relu1 = nn.ReLU(inplace=True)
        self.conv1 = nn.Conv2d(num_input_features, bn_size *
growth_rate, kernel_size=1, stride=1, bias=False)

        self.norm2 = nn.BatchNorm2d(bn_size * growth_rate)
        self.relu2 = nn.ReLU(inplace=True)
        self.conv2 = nn.Conv2d(bn_size * growth_rate, growth_rate,
kernel_size=3, stride=1, padding=1, bias=False)

        self.drop_rate = float(drop_rate)
        self.memory_efficient = memory_efficient

    def bn_function(self, inputs: List[Tensor]) -> Tensor:
        concated_features = torch.cat(inputs, 1)
        bottleneck_output =
self.conv1(self.relu1(self.norm1(concated_features)))  # noqa: T484
        return bottleneck_output

    # todo: rewrite when torchscript supports any
    def any_requires_grad(self, input: List[Tensor]) -> bool:
        for tensor in input:
            if tensor.requires_grad:
                return True
        return False
```

Transition block code

```
class _Transition(nn.Sequential):
    def __init__(self, num_input_features: int, num_output_features:
int) -> None:
        super().__init__()
        self.norm = nn.BatchNorm2d(num_input_features)
        self.relu = nn.ReLU(inplace=True)
        self.conv = nn.Conv2d(num_input_features, num_output_features,
kernel_size=1, stride=1, bias=False)
        self.pool = nn.AvgPool2d(kernel_size=2, stride=2)
```

Model training code

```
def train_epoch(
  model,
  data_loader,
  loss_fn,
  optimizer,
  device,
  scheduler,
  n_examples
):
  model = model.train()

  losses = []
  correct_predictions = 0

  for d in data_loader:
    contextualised_embedding = d["contextualised_embedding"].to(device)
    targets = d["target"].to(device)

    outputs = model(
      contextualised_embedding
    )
    preds = torch.round(outputs.view(-1))

    loss = F.binary_cross_entropy(outputs.view(-1), targets.float())
    #print(outputs,preds)

    correct_predictions += torch.sum(preds == targets)
    losses.append(loss.item())

    loss.backward()
    nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
    optimizer.step()
    scheduler.step()
    optimizer.zero_grad()

  return correct_predictions.double() / n_examples, np.mean(losses)
```

Model Evaluation Code

```
def eval_model(model, data_loader, loss_fn, device, n_examples):
    model = model.eval()

    losses = []
    correct_predictions = 0

    with torch.no_grad():
        for d in data_loader:
            contextualised_embedding =
d["contextualised_embedding"].to(device)
            targets = d["target"].to(device)

            outputs = model(contextualised_embedding)
            preds = torch.round(outputs.view(-1))

            loss = F.binary_cross_entropy(outputs.view(-1),
targets.float())

            correct_predictions += torch.sum(preds == targets)
            losses.append(loss.item())

    return correct_predictions.double() / n_examples, np.mean(losses)
```

Training Loop code

```
%%time

 history = defaultdict(list)
 best_accuracy = 0

 for epoch in range(EPOCHS):

     print(f'Epoch {epoch + 1}/{EPOCHS}')
     print('-' * 10)

     train_acc, train_loss = train_epoch(
     model,
     train_DataLoader,
     loss_fn,
     optimizer,
     device,
     scheduler,
     len(df_train)
     )

     print(f'Train loss {train_loss} accuracy {train_acc}')

     val_acc, val_loss = eval_model(
     model,
     valid_DataLoader,
     loss_fn,
     device,
     len(df_valid)
     )

     print(f'Val   loss {val_loss} accuracy {val_acc}')
     print()

     history['train_acc'].append(train_acc.cpu())
     history['train_loss'].append(train_loss)
     history['val_acc'].append(val_acc.cpu())
     history['val_loss'].append(val_loss)

     if val_acc > best_accuracy:
         best_accuracy = val_acc
         print('saving best model')
         torch.save({
                 'epoch': EPOCHS,
                 'model_state_dict': model.state_dict(),
                 'optimizer_state_dict': optimizer.state_dict(),
                 'loss': loss_fn,
                 }, SavePath)
```

Training History diagram code

```
plt.plot(history['train_acc'], label='train accuracy')
plt.plot(history['val_acc'], label='validation accuracy')

plt.title('Training history')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()
```

Loading best model code

```
# load best model
Load_model = densenet121()
Load_optimizer = AdamW(Load_model.parameters(), lr=learning_rate,
correct_bias=False)

checkpoint = torch.load(SavePath)
Load_model.load_state_dict(checkpoint['model_state_dict'])
Load_optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
Load_epoch = checkpoint['epoch']
Load_loss = checkpoint['loss']

Load_model.eval()
```

Model prediction code

```
def get_predictions(model, data_loader):
  model = model.eval()

  review_texts = []
  predictions = []
  prediction_probs = []
  real_values = []

  with torch.no_grad():
    for d in data_loader:

      texts = d["review_text"]
      contextualised_embedding =
d["contextualised_embedding"].to(device)
      targets = d["target"].to(device)

      outputs = model(contextualised_embedding)

      preds = torch.round(outputs.view(-1))
      loss = F.binary_cross_entropy(outputs.view(-1),
targets.float())

      probs = F.softmax(outputs, dim=1)

      review_texts.extend(texts)
      predictions.extend(preds)
      prediction_probs.extend(probs)
      real_values.extend(targets)

  predictions = torch.stack(predictions).cpu()
  prediction_probs = torch.stack(prediction_probs).cpu()
  real_values = torch.stack(real_values).cpu()
  return review_texts, predictions, prediction_probs,
real_values
```

Classification Report code

```
print("sentence length", sen_length)
print("dataset size", n)
print("batch size", batch_size)
print(classification_report(y_test, y_pred,
target_names=class_names,digits=4))
```

Confusion matrix generation Code

```
def show_confusion_matrix(confusion_matrix):
  hmap = sns.heatmap(confusion_matrix, annot=True, fmt="d",
cmap="Blues")
  hmap.yaxis.set_ticklabels(hmap.yaxis.get_ticklabels(), rotation=0,
ha='right')
  hmap.xaxis.set_ticklabels(hmap.xaxis.get_ticklabels(), rotation=30,
ha='right')
  plt.ylabel('True sentiment')
  plt.xlabel('Predicted sentiment');

cm = confusion_matrix(y_test, y_pred)
df_cm = pd.DataFrame(cm, index=class_names, columns=class_names)
show_confusion_matrix(df_cm)
```

Appendix B: Classification report tables grouped by dataset size.

	Accuracy	Precision (macro avg)	Precision (weighted avg)	Recall (macro avg)	Recall (weighted avg)	F1-Score (macro avg)	F1-Score (weighted avg)
32	0.73	0.7345	0.7367	0.7328	0.73	0.7298	0.7295
64	0.829	0.8295	0.831	0.8302	0.829	0.829	0.8291
128	0.885	0.8899	0.8927	0.888	0.885	0.885	0.8849
256	0.928	0.9297	0.9288	0.9267	0.928	0.9276	0.9279
512	0.932	0.9351	0.9338	0.9302	0.932	0.9315	0.9318

Table 1: Classification report for dataset = 500

	Accuracy	Precision (macro avg)	Precision (weighted avg)	Recall (macro avg)	Recall (weighted avg)	F1-Score (macro avg)	F1-Score (weighted avg)
32	0.728	0.7426	0.7403	0.7213	0.728	0.7195	0.7219
64	0.8	0.8208	0.8253	0.806	0.8	0.7985	0.7977
128	0.892	0.8946	0.8968	0.8943	0.892	0.892	0.892
256	0.915	0.9167	0.9187	0.917	0.915	0.915	0.915
512	0.934	0.9338	0.934	0.9339	0.934	0.9339	0.934

Table 2: Classification report for dataset = 1000

	Accuracy	Precision (macro avg)	Precision (weighted avg)	Recall (macro avg)	Recall (weighted avg)	F1-Score (macro avg)	F1-Score (weighted avg)
32	0.737	0.7579	0.7597	0.7578	0.756	0.756	0.756
64	0.833	0.8344	0.8362	0.8347	0.833	0.833	0.833
128	0.9	0.9002	0.9015	0.9011	0.9	0.9	0.9001
256	0.922	0.9219	0.923	0.9229	0.922	0.9219	0.922
512	0.942	0.9423	0.9421	0.9415	0.942	0.9418	0.942

Table 3: Classification report for dataset = 2000

	Accuracy	Precision (macro avg)	Precision (weighted avg)	Recall (macro avg)	Recall (weighted avg)	F1-Score (macro avg)	F1-Score (weighted avg)
32	0.75	0.7512	0.7528	0.7514	0.75	0.75	0.7501
64	0.7501	0.8432	0.8431	0.8419	0.843	0.8424	0.8429
128	0.896	0.8966	0.8982	0.8974	0.896	0.896	0.896
256	0.928	0.9281	0.928	0.9275	0.928	0.9278	0.928
512	0.942	0.9423	0.9421	0.9415	0.942	0.9418	0.942

Table 4: Classification report for dataset = 4000

	Accuracy	Precision (macro avg)	Precision (weighted avg)	Recall (macro avg)	Recall (weighted avg)	F1-Score (macro avg)	F1-Score (weighted avg)
32	0.756	0.7579	0.7597	0.7578	0.756	0.756	0.756
64	0.833	0.8344	0.8362	0.8347	0.833	0.833	0.833
128	0.9	0.9002	0.9015	0.9011	0.9	0.9	0.9001
256	0.922	0.9219	0.923	0.9229	0.922	0.9219	0.922
512	0.942	0.9423	0.9421	0.9415	0.942	0.9418	0.942

Table 5: Classification report for dataset = 8000

Appendix C: Dataset size vs. Accuracy tables by sentence length

Dataset Size	Accuracy
500	0.73
1000	0.728
2000	0.737
4000	0.75
8000	0.756

Table 6: Dataset size vs. accuracy of sentence length = 32

Dataset Size	Accuracy
500	0.833
1000	0.8
2000	0.833
4000	0.7501
8000	0.833

Table 7: Dataset size vs. accuracy of sentence length = 64

Dataset Size	Accuracy
500	0.885
1000	0.892
2000	0.9
4000	0.896
8000	0.942

Table 8: Dataset size vs. accuracy of sentence length = 128

Dataset Size	Accuracy
500	0.928
1000	0.915
2000	0.922
4000	0.928
8000	0.922

Table 9: Dataset size vs. accuracy of sentence length = 256

Dataset Size	Accuracy
500	0.932
1000	0.934
2000	0.942
4000	0.942
8000	0.942

Table 10: Dataset size vs. accuracy of sentence length = 512