

Capstone Chronicles

2024 Selections

MS-Applied Artificial Intelligence
University of San Diego



Dear Reader,

It is with great pleasure that we introduce the inaugural edition of *Capstone Chronicles*, a collection of outstanding Capstone projects from the MS in Applied Artificial Intelligence program at the University of San Diego (USD) in 2024. This publication serves as a testament to the dedication, creativity, and analytical expertise of our students as they tackle real-world challenges through AI-driven solutions.

The University of San Diego's innovative online AI master's degree program is committed to training current and future artificial intelligence professionals for the important and fascinating work ahead. The strengths of our program include a significant emphasis on real-world applications, ethics, moral responsibility, and social good in designing AI-enabled systems, and has been developed by AI experts in close collaboration with key industry and government stakeholders to provide in-depth practical and technical training.

Each graduating cohort (Spring, Summer, and Fall) included in this magazine consists of 25-30 students. In the Capstone course, students apply the knowledge and skills acquired in the program to develop AI-enabled systems. Working in teams, they identify a problem or research question, develop a project proposal outlining an approach to solving it, implement their solution, and test or evaluate the result. Students must identify and cleanse a dataset, choose appropriate tools and algorithms, and ensure that at least one neural network or deep learning-based model is developed and trained from scratch. The work must be original, going beyond pre-built model architectures and tutorials to demonstrate a deep understanding of AI techniques.

We hope that *Capstone Chronicles* serves as both an inspiration and a resource for future students, researchers, and practitioners in the field of artificial intelligence. By sharing these exemplary projects, we aim to celebrate the accomplishments of our students and contribute to the broader discourse on applied AI.

We extend our gratitude to the students whose hard work is showcased in these pages, as well as to the faculty and mentors who have guided them throughout their journeys.

Thank you for your interest in *Capstone Chronicles* and the MS-Applied Artificial Intelligence program at the University of San Diego.

The 2024 Capstone Chronicles Editorial Team

Anna Marbut

Ebrahim Tarshizi

Table of Contents

Spring 2024

Smart AI Stock Trading System	5
<i>Nathan Metheny, Javon Kitson, Adam Graves</i>	
Detecting Fake News Using Natural Language Processing	26
<i>Abdul Shariq, Kayla Wright, Lauren Taylor</i>	
Human versus Artificial Intelligence Distinguishment	51
<i>Jeremy Cryer, Jason Raimondi, Shane Schipper</i>	
Virtual Teaching Assistant	74
<i>Joseph Binny, Christopher J. Watson, Viktor Veselov</i>	
Electricity Distribution Topology (Meter to Transformer) Classification	92
<i>Bin Lu, Trevor Mcgirr</i>	

Summer 2024

Object Detection for Unmanned Aerial Vehicles	115
<i>Carson Edmonds, Patricia Enrique, Jeremy Krick</i>	
Monthly Passenger Count Prediction for the San Francisco International Airport	131
<i>Jamileh Jabangiry, Prachi Khanna, Se'Lina Lasher</i>	
Lung Disease Detection Using Convolutional Neural Networks	159
<i>Isaack Karanja, Reed Oken, Alec Anderson</i>	
A.S. Linguist Final Project	181
<i>Shyam Adhikari, Caterina Gallo, Paul Tha</i>	

Fall 2024

Deep Learning Image Captioning	206
<i>Steve Amancha, Rabul Das, Juliet Lawton</i>	
Diabetes Management System: Personalized blood glucose prediction and insulin requirement system with Deep learning	235
<i>Angel Benitez, Dina Shalaby, Gary Takahashi, Eyoha Mengistu</i>	
SepsiTCT: A Stacked Convolutional Transformer Model for Early Sepsis Detection in ICU Patients	259
<i>Tyler Foreman, Ahmed Ahmed, Eric Barnes, Ryan Laxamana</i>	

Spring 2024



Final Capstone Project:
Smart AI Stock Trading System

Group 6: Nathan Metheny, Javon Kitson, Adam Graves

University of San Diego

AAI-590: Capstone Project

Professor: Anna Marbut

April 15, 2024

GitHub: <https://github.com/noface-0/AAI-590-01-Capstone/tree/main>

Presentation: <https://www.youtube.com/watch?v=smgWoTH2GzY>

Introduction

This project introduces an advanced stock trading system utilizing AI-based algorithmic models. Algorithmic trading has significantly gained traction worldwide, with substantial growth noted in the U.S., where the market was valued at USD 14.42 billion in 2023 and is projected to reach USD 23.74 billion in the next five years (Mordor Intelligence, n.d.). The adoption of these systems has increased due to their efficiency, accuracy, and capability to process large volumes of data swiftly, gaining acceptance by regulatory bodies like the SEC and FINRA.

Contemporary algorithmic stock trading systems, such as TradeStation, rely on predictive models to forecast daily stock prices and refine these predictions down to specific moments within the trading day. The core functionality allows traders to execute orders based on specified limit prices, ensuring trades occur within predetermined cost boundaries. However, these systems often struggle to fully grasp and react to the multifaceted and interconnected nature of market dynamics due to their limited perspective, as they typically focus on individual stock patterns without fully considering the broader market's state space, which includes the interplay of various stocks and their collective influence on market behavior.

The aim of our research is to explore the performance of Deep Reinforcement Learning (DRL) within the context of financial markets. The project aims to deploy machine learning methods to construct an autonomous system that executes intelligent trading decisions. This requires the development of a model that can process and interpret the vast state and action spaces of the stock market to perform trades with the objective of optimizing financial returns. For these types of algorithms, a Deep Learning

(DL) model is more accurate than a standard Machine Learning (ML) model and performs well on unstructured data. However, it also requires a massive amount of training data and expensive hardware and software (Jakhar & Kaur, 2020).

The research will examine the configuration, training, and assessment of the system, comparing its performance with traditional trading strategies. A robust dataset from First Rate Data, consisting of 10,120 tickers and their relevant trading values, will be used to build the models. Alongside the technical, the study will investigate the conceptual aspects of reinforcement learning and the applicability to financial markets. This includes an analysis of the difficulties encountered when implementing DRL in a highly instable environment.

The model's reference behavior is designed to balance risk and reward efficiently, guiding the trading algorithm to make decisions that align with the expected risk-adjusted returns. This integration ensures that the system remains robust and responsive, capable of navigating market volatilities while adhering to the risk constraints. Our hypothesis is that this approach can adapt to market dynamics, make intelligent decisions, and produce an optimal portfolio to interact with.

Data Summary

Our dataset consisted of historical stock data from a paid licensed First Rate Data (firstratedata.com) (First Rate Data, 2023) and derived technical indicators for a diverse range of stocks spanning nearly two decades. The dataset included 35 variables, which were a combination of original stock price data and augmented

variables engineered to enhance the predictive capabilities of our Feedforward Neural Network and Deep Reinforcement Learning (DRL) models.

The variables included in the dataset are basic data fields required for stock trading, such as open, high, low, close, and volume, all of which are numeric values associated with a timestamp. In addition, we have augmented variables that were derived from the original stock price data and included numeric fields. These original variables, such as price and volume data, were directly related to our project goal of developing a DRL model for stock trading. They provided the foundation for the model to learn patterns and make trading decisions. The inclusion of augmented variables, like technical indicators, provided further signals of market movements, thereby improving the model's predictive capabilities.

We have another dataset that consists of client input used to build a client account. This data is used to calculate the risk tolerance assigned to the client. The calculations are not AI-related and are based on a combination of age, investing experience, and net worth. The risk tolerance is categorized into three levels, which have an impact on the trading portfolio.

We found significant correlations among the variables, particularly between price-related variables (e.g., open, high, low, close) and volume. Strong correlations were also observed between the original and augmented variables, as the latter were derived from the former. A representation of field correlations can be viewed in the heatmap in the visualization section (Figure 5).

Background Information

Stock trading has been a domain of significant interest for academic researchers, business entrepreneurs, and financial institutions. The goal of maximizing returns while minimizing risk has driven the development of various methods and technologies to predict market movements and make informed trading decisions. Traditionally, stock trading strategies have relied on fundamental analysis, technical analysis, and human expertise. Fundamental analysis is a method of evaluating a company through means of its financials and potential for growth. In contrast, technical analysis is an approach that relies on analyzing past market data to recognize patterns that could suggest future price movements. Human traders use a combination of these approaches, along with their experience and intuition, to make trading decisions.

However, these methods have inherent limitations in their ability to capture the complex dynamics of the stock market and adapt to evolving market conditions. Specifically, these techniques fail to consider the entirety of the state space, where all other stocks and their corresponding patterns should be taken into account. In recent years, significant progress has been made in solving challenging problems across various domains using deep reinforcement learning (DRL) (Henderson et al., 2018). DRL merges deep learning with reinforcement learning and allows an agent to learn optimal actions through continual interactions with a pre-defined, structured environment. In the context of stock trading, the agent (our DRL model) observes the state of the market (e.g., stock prices, technical indicators) and takes actions (e.g., buy, sell, hold) to maximize a reward signal (e.g., portfolio value, profit). The agent learns from its experiences of profit and loss and adjusts its strategy over time to improve its performance while profit is the goal.

Our project focuses on the application of DRL in stock trading, aiming to create an autonomous system that can learn from historical data and adapt to changing market conditions. Numerous academic articles discussing the use of DRL models to automate stock trading activity are available. For example, Yang et al. (2020) propose an ensemble strategy combining Proximal Policy Optimization (PPO), Advantage Actor Critic (A2C), and Deep Deterministic Policy Gradient (DDPG) in their research paper "Deep Reinforcement Learning for Automated Stock Trading: An Ensemble Strategy." This approach integrates the strengths of these three actor-critic-based algorithms, aiming to create a robust system that adapts to various market conditions.

Similar to their approach, we employ a combination of FNN, SAC, and PPO to train the stock trading component of our system. The actor network, implemented in the ActorSAC class, is responsible for selecting actions (trading decisions) based on the current state of the market. It takes the state as input and outputs the mean and log-standard deviation of a Gaussian distribution (Frisch et al., 2016). The action is then sampled from this distribution using reparameterization, allowing for the learning of a stochastic policy. The critic network, implemented in the CriticSAC class, estimates the Q-values of state-action pairs. It takes the state and action as input and outputs two Q-value estimates using separate neural networks. The use of two Q-value estimates helps to stabilize the learning process and mitigate overestimation bias.

In contrast to single-model systems, our approach is advanced and sophisticated, harnessing the collective intelligence of multiple models to enhance decision-making accuracy and adaptability within the dynamic landscape of the stock market. Current popular algorithmic stock trading systems, such as TradeStation, have

been based on the ability to predict the trading price of the stock on a day-to-day basis. As they advanced, they had the ability to go deeper into the prediction at a certain point of time, with the foundation being the ability to trade on the condition of the limit price entered.

Our research demonstrates that DRL provides significant alpha, as the stock trading strategies learned through this approach have resulted in returns that are substantially higher than those of the market average or a relevant benchmark. DRL models have also been successfully applied in the field of robotics. Haarnoja et al. (2018) discuss the valuable properties of the Soft Actor-Critic (SAC) algorithm in their research paper "Soft Actor Critic—Deep Reinforcement Learning with Real-World Robots," where they used models to train a robot to move, a 3-finger dexterous robotic hand to manipulate an object, and a 7-DoF Sawyer robot to stack Lego blocks.

Furthermore, Nan et al. (2021) incorporated additional external factors that are subject to frequent changes and often unable to be inferred solely from historical trends in their research paper "Sentiment and Knowledge-Based Algorithmic Trading with Deep Reinforcement Learning." To address this, they employed Partially Observable Markov Decision Processes (POMDP), taking into account events outside the realm of stock trading, such as the destruction of a trading data center, a scenario that actually occurred on September 11th.

In our architecture, we utilize a Genetic Agent (GA) to select a subset of stocks from a larger pool based on a predefined objective and in line with the strategy based on the client portfolio input. This ensures that the trading is within regulation requirements. The DRL models (SAC and PPO) in our project are responsible for

making trading decisions based on market conditions, and the FNN model is used as the underlying architecture for both the actor and critic networks in the SAC and PPO algorithms to predict future stock prices. This combination is well-suited to build a successful stock trading system.

Experimental Methods

Our research and implementations were built upon the foundation provided by the FinRL library (AI4Finance-Foundation, n.d.). FinRL is an open-source framework that facilitates the application of deep reinforcement learning in quantitative finance. By leveraging the FinRL library, we were able to efficiently implement and experiment with the PPO, while building the custom Genetic Algorithm (GA), Feedforward Network (FNN), Soft Actor-Critic (SAC), and Twin Delayed Deep Deterministic Policy Gradient (TD3) agents to fit into the broader architecture. The FinRL library provided a solid starting point for our research, offering a range of pre-built environments, agents, and evaluation metrics that accelerated our development process and allowed us to focus on the specific adaptations and optimizations required for our ensemble approach.

The project employs an ensemble approach, combining Deep Reinforcement Learning (DRL), a Feedforward Neural Network (FNN), and a Genetic Algorithm (GA) for stock trading, price prediction, and portfolio optimization. The DRL model makes trading decisions based on market conditions, while the FNN model predicts future stock prices, providing additional input to the DRL model. An individual's portfolio questionnaire input creates an additional dataset, from which a unique ID is identified, and a risk factor is calculated.

The risk factor, ranging from 0 to 30, is determined based on the individual's net worth, trading experience, age, and trading goals. The range is split into three categories: minimum drawdown (0-10), maximum return (11-20), and a combination of minimum drawdown and maximum return (21-30).

The Genetic Algorithm (GA) follows a standard evolutionary process encapsulated in the `GeneticAlgorithm` class. This class incorporates portfolio initialization, fitness evaluation of symbols calculating drawdowns, selection of parent orders, crossover to create children orders, mutation based on probabilities, output of trade results, and termination upon finding a satisfactory solution. The `time_interval`, `start_date`, and `end_date` variables align with the inputs used for FNN and DRL training. The GA returns the best individual portfolio found, along with its corresponding returns and drawdown. This constrained portfolio is then used for downstream training and trading.

The Feedforward Neural Network (FNN) model architecture is composed of three main components: an input layer that receives the initial data, multiple hidden layers that process the distribution, and an output layer that produces a final, singular prediction. Notable design choices include an input layer size determined by the number of input data features, a list of possible hidden layer structures [(32, 16), (64, 32), (128, 64), (256, 128)] for flexible hyperparameter selection, an output layer size of 1, dropout regularization with a default of 0.5, batch normalization, and a ReLU activation function.

FNN model training involves downloading historical stock data, dividing it into training and validation sets (typically 80/20), setting up the model architecture, and using Huber Loss (Huber F., 1964) as the loss function. Training utilizes the Adam

optimizer (Diederik P., Kingma, & Ba J. 2017) to adjust weights over 10 epochs with a batch size of 64, evaluating the model's performance on the validation set after each epoch. Optuna (Akiba T., Sano S., Yanase T., Ohta T., & Koyama M. 2019) is used for hyperparameter tuning and architectural adjustments, efficiently searching the hyperparameter space to find the best combination of learning rate and hidden layer sizes that minimize validation loss.

Two popular DRL algorithms, Soft Actor-Critic (SAC) and Proximal Policy Optimization (PPO), were experimented with for stock trading. The DRL agents were trained on the optimized portfolio returned by the GA, focusing their learning on a more promising subset of stocks. The SAC model's architecture uses multi-layer perceptrons (MLPs) with ReLU activation for both actor and critic networks. The actor network's output forms a Gaussian distribution for action decisions, while the critic network provides two Q-value estimates. The SAC algorithm's training process involves the agent interacting with the environment, gathering experiences, and updating the critic and actor networks using the collected experiences. Hyperparameters such as learning rate, discount factor, entropy coefficient, and network architecture were tuned to optimize performance.

PPO, an on-policy DRL algorithm (Schulman J., 2017), also consists of an actor network and a critic network. The actor network selects actions based on the current state, while the critic network estimates the value of each state (Chen & Xiao, 2023). PPO trains by having the agent interact with the environment, gather data, and update the actor and critic networks. The actor network updates aim to maximize future rewards without straying too far from the previous policy, while the critic network updates focus

on reducing the discrepancy between predicted and actual state values.

Hyperparameters such as learning rate, discount factor, batch size, and clip range were adjusted to optimize PPO's performance.

Both SAC and PPO agents were trained using a rollout buffer to store collected experiences, interacting with the stock market environment for a specified number of iterations. The models were optimized by tuning various hyperparameters and experimenting with different reward scaling techniques and exploration strategies. The trained DRL agents were then evaluated on a separate test dataset to assess their ability to generate profitable trading strategies in unseen market conditions, using performance metrics such as cumulative returns and Sharpe ratio to compare the effectiveness of the SAC and PPO algorithms.

Results & Conclusion

The advanced stock trading system we developed, which leverages Deep Reinforcement Learning (DRL), Feedforward Neural Networks (FNN), and Genetic Algorithms (GA), has yielded promising results. By structuring the system into separate processing components and utilizing technical analysis, we designed a multiple model architecture capable of making intelligent trading decisions that balance risk and reward efficiently, based on the investor's profile input. The system's performance metrics suggest that this approach can effectively adapt to market dynamics and generate an optimal portfolio for interaction.

The system calculates a risk factor based on the investor's profile data (Figures 3 and 4), which the GA then uses to generate an appropriate portfolio. The DRL models,

Soft Actor-Critic (SAC) and Proximal Policy Optimization (PPO), were trained using these optimized portfolios, along with the Feedforward Neural Network's (FNN) future price prediction. This strategic combination allowed the system to focus on a subset of stocks, enhancing learning efficiency and trading performance. The SAC model, known for its sample-efficient learning, and the PPO model, recognized for balancing performance and stability, were instrumental in navigating the complex stock market environment. When tested against the validation data, the system generated a 52% return on investment above the initial value (Figure 6).

The assessment of our FNN model's performance demonstrated its ability to generalize effectively to new data, although there is room for improvement. The validation metrics, including Root Mean-Squared Error (RMSE) and R-Squared (R^2), indicate that the model successfully captures trends and patterns in stock price movements. The high performance can be primarily attributed to the immediate look-forward period, as the model only predicts the next timeframe. However, these results also highlight opportunities for further refinement to minimize prediction errors and improve accuracy, suggesting that with additional tuning, the model could achieve even more reliable predictions over longer, more future-oriented timeframes.

To further refine the system, we optimized the models by tuning various hyperparameters, such as the learning rate, discount factor, and network architectures (e.g., number of hidden layers and units). We also experimented with different reward scaling techniques and exploration strategies to improve the agents' performance, finding that a three-level reward scaling would best mitigate the risk factor related to the trade portfolio. The trained DRL agents were then evaluated on a separate test dataset

to assess their ability to generate profitable trading strategies in unseen market conditions, using performance metrics such as cumulative returns and Sharpe ratio to compare the effectiveness of the SAC and PPO algorithms.

When comparing the performance of the SAC and PPO algorithms, both models demonstrated strong trading strategies. The PPO model achieved a 52% return on investment above the initial value when tested against the validation data, whereas the SAC model generated an 89% return on investment.

The PPO algorithm exhibited higher stability and more consistent performance across learning, while the SAC model demonstrated increased speed in learning and adaptation in different environments. The Sharpe ratio was slightly higher for the PPO model, indicating a better balance between returns and risk.

Overall, the SAC and PPO algorithms proved to be highly effective in the advanced stock trading system. The PPO model is preferred for its stability, while the SAC model may be favored in situations requiring more extensive adaptation or when a higher risk appetite is acceptable. The selection between the two algorithms ultimately depends on the specific requirements of the trading system, such as the need for consistent performance or the ability to adapt to dynamic market conditions.

During the exploratory data analysis, we encountered some issues, such as missing data, which was primarily due to stocks being delisted and no longer traded. As DRL models learn from the entirety of the state space, the most practical way of handling missing data was to remove it, which was also the case for stocks listed later than the beginning timestamp. While this data holds training value, future work should focus on research aiming to extract this value. As Woodford M. and Xie Y. (2020)

suggest, "The most reasonable method to resolve is to backfill and forward fill with a monetary price of zero." Additional analysis was performed to check for duplicate values and format errors.

Our current setup operates within a paper trading framework, which closely mimics real-world market conditions but does not fully account for certain factors that can impact trading performance. It is crucial to acknowledge that while our system demonstrated advanced capabilities in navigating the complex stock market environment, the simulated nature of the paper trading environment has its limitations.

Factors such as the cost of executing trades and slippage are not entirely considered in our current setup. These can have a non-marginal impact on the system's overall performance in real-world trading scenarios. The promising results obtained in our paper trading environment may not directly translate to live trading, as the market inefficiencies and additional costs associated with real-world trading can significantly influence the outcome.

Regardless, the performance metrics obtained in our simulated environment serve as a valuable proof-of-concept, demonstrating the potential of integrating DRL, FNN, and GA in developing an advanced stock trading system. However, to gain a more accurate representation of the system's performance, future iterations of the model should carefully evaluate and incorporate these real-world factors.

The insights gained from this study provide a solid foundation for further research and development. By refining the model's architecture, incorporating additional market factors, and testing its performance in more realistic trading scenarios, we can bridge the gap between our paper trading results and real-world applications. This will enable

us to better assess the system's robustness and adaptability to live market conditions and make necessary adjustments to optimize its performance in practical trading environments.

Visualizations

Figure 1: High Level Diagram Flow:

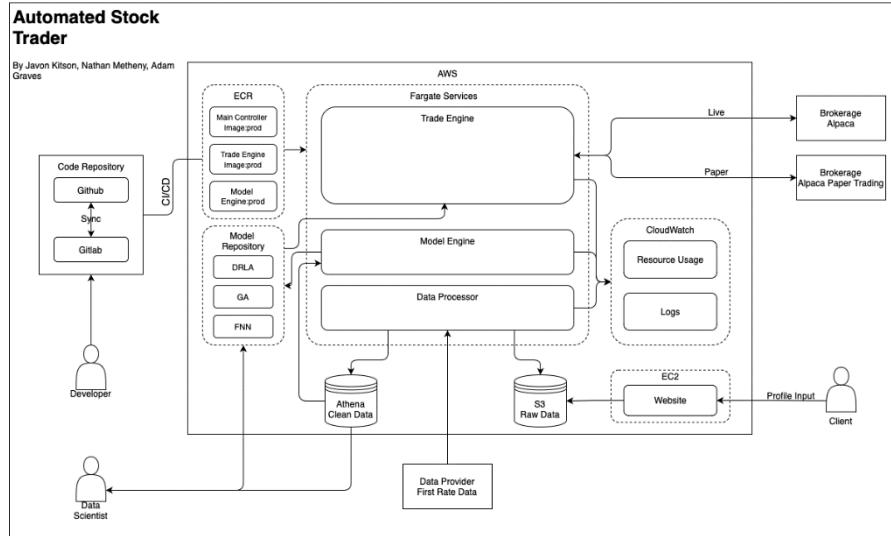


Figure 2: Diagram Of Trading Flow:

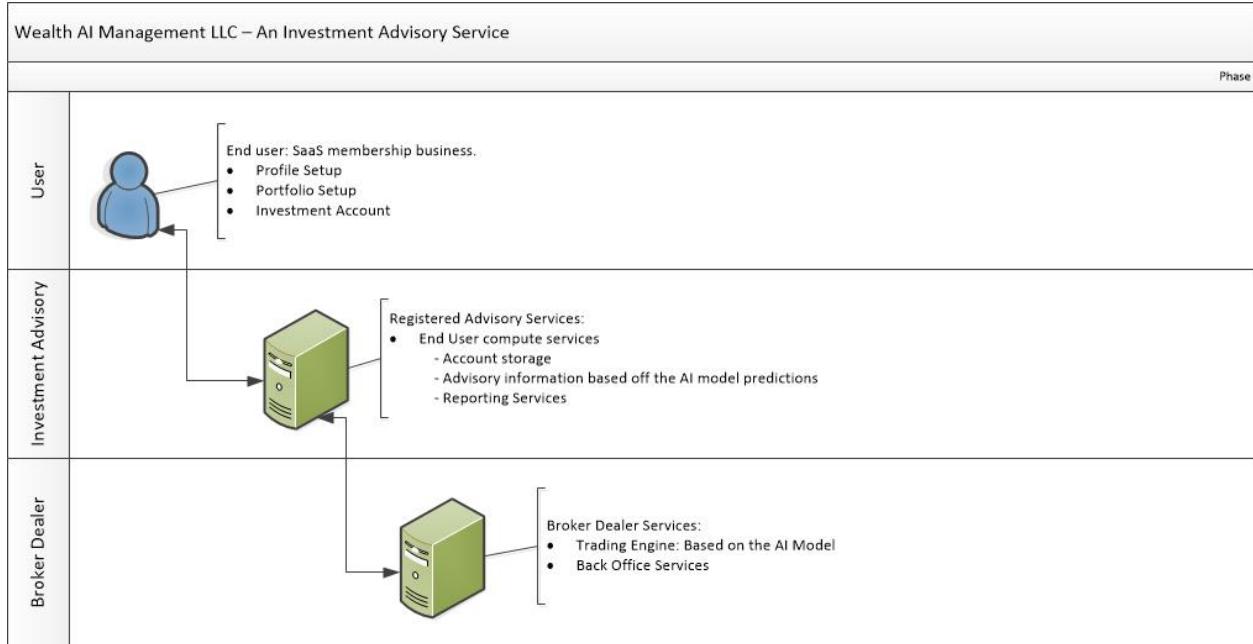


Figure 3: Profile Data



Portfolio Fields

- Name
- Social Security number or taxpayer identification number
- Address
- Telephone number
- E-mail address
- Date of birth
- Driver's license, passport information, or other government-issued identification
- Employment status and occupation
- Whether you are employed by a brokerage firm
- Annual income
- Other investments
- Financial situation and needs
- Tax status
- Investment experience and objectives
- Investment time horizon
- Liquidity needs and tolerance for risk
- Financial and trading record
- Net worth
- Trading experience
- Financial knowledge

Figure 4: Profile Form

The screenshot shows a web browser window with the URL 'localhost'. The main title is 'Submit Form or Upload File'. The page is divided into two sections: 'Form' on the left and 'File Upload' on the right. The 'Form' section contains fields for Name, Address, Telephone Number, ID (SSN or TIN), Email, and a date input field showing '04/14/2024'. It also includes a checkbox for 'Employed by a brokerage firm?' with the option 'Unemployed' selected. Below these are dropdown menus for 'Annual Income', 'Married', 'Trading Experience (0-1)', 'Net Worth', and 'Financial Knowledge (0-1)'. A 'Submit Form' button is at the bottom. The 'File Upload' section has a 'Choose File' button with 'no file selected' and an 'Upload' button.

Figure 5: Heatmap Plot for Data Field Correlation

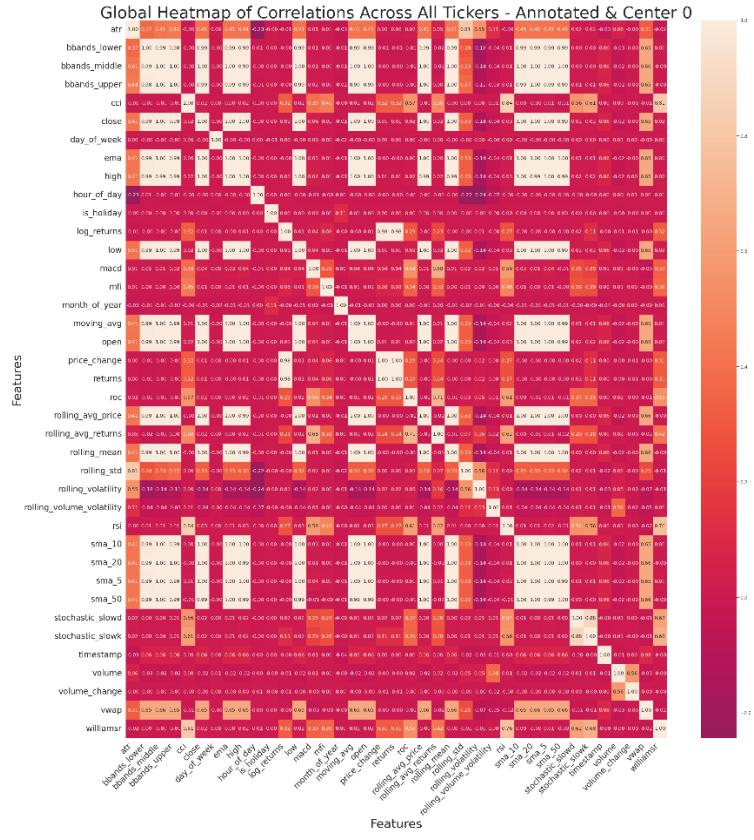


Figure 6: PPO Performance

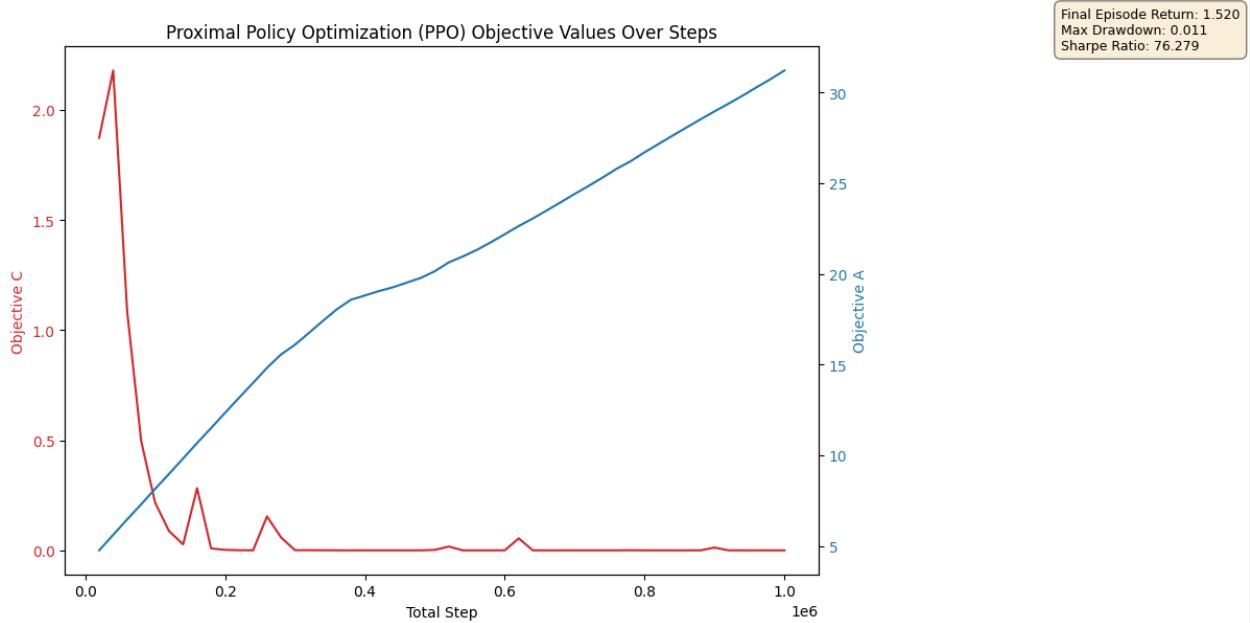
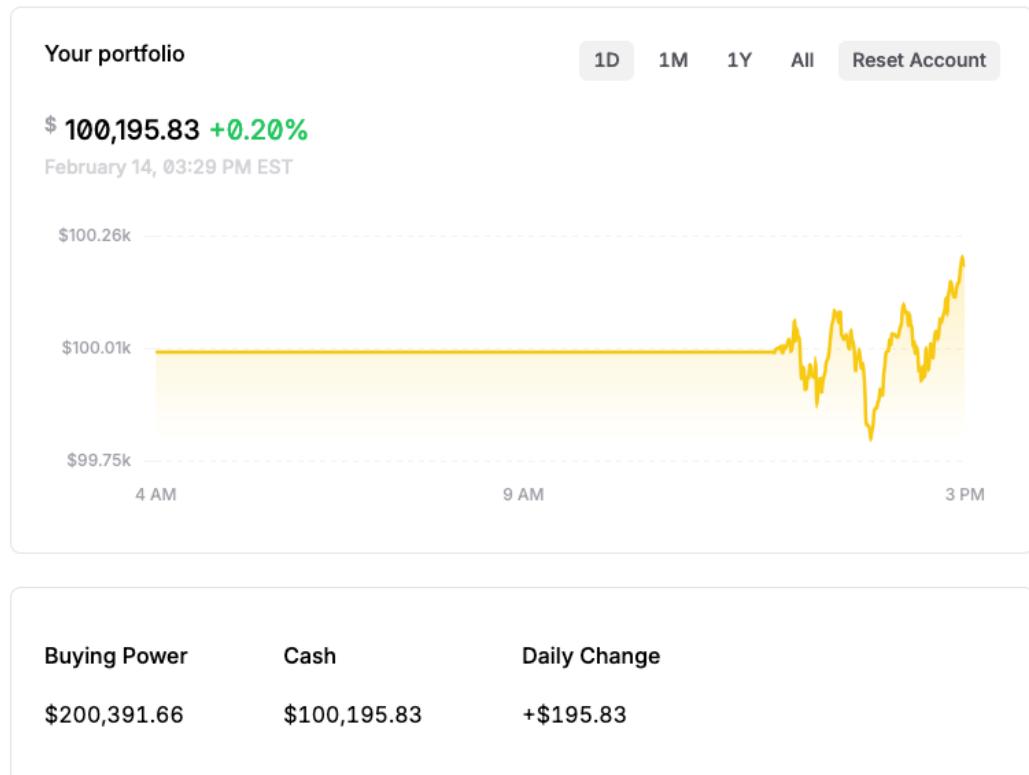


Figure 7: Trading Report per Portfolio



References

1. AI4Finance-Foundation. (n.d.). FinRL: A deep reinforcement learning library for automated stock trading in quantitative finance. GitHub. Retrieved [2024], from <https://github.com/AI4Finance-Foundation/FinRL>
2. Chen, Y., & Xiao, J. (2023). Target search and navigation in heterogeneous robot systems with deep reinforcement learning. arXiv preprint arXiv:2308.00331.
3. First Rate Data. (2023). Historical Stock Data [Data set]. Retrieved from <https://www.firstratedata.com>
4. Haarnoja T., Zhou A., Abbeel P., & Levine S., Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, Taken from:
extension://efaidnbmnnibpcajpcgkclefindmkaj/https://proceedings.mlr.press/v80/haarnoja18b/haarnoja18b.pdf
5. Haarnoja T., Pong V., Hartikainen K., Zhou A., Dalal M., & Levine S.(2018), Actor Critic—Deep Reinforcement Learning with Real-World Robots, Taken from:
<https://bair.berkeley.edu/blog/2018/12/14/sac/>
6. Heeswijk W., PhD, Proximal Policy Optimization (PPO) Explained, November 29, 2022.
7. Lin, C. C., & Marques, J. A. (2023). Stock market prediction using artificial intelligence: A systematic review of systematic reviews.
<https://doi.org/10.2139/ssrn.4341351>

8. Nan, A., Perumal, A., & Zaiane, O. R. (2022). Sentiment and knowledge based algorithmic trading with deep reinforcement learning. Lecture Notes in Computer Science, 167-180. https://doi.org/10.1007/978-3-031-12423-5_13
9. Woodford, M., & Xie, Y. (2020). Fiscal and monetary stabilization policy at the zero lower bound: Consequences of limited foresight.
<https://doi.org/10.3386/w27521>
10. Yang, H., Liu, X., Zhong, S., & Walid, A. (2020). Deep reinforcement learning for automated stock trading. Proceedings of the First ACM International Conference on AI in Finance. <https://doi.org/10.1145/3383455.3422540>
11. Yarats, D., & Kostrikov, I. (2020). Soft Actor-Critic (SAC) implementation in PyTorch. GitHub. https://github.com/denisyarats/pytorch_sac

Detecting Fake News Using Natural Language Processing

Detecting Fake News Using Natural Language Processing

Abdul Shariq, Kayla Wright and Lauren Taylor

University of San Diego

AAI 590: Capstone Project

Professor Anna Marbut

Detecting Fake News Using Natural Language Processing

Introduction

Our project involves the detection of fake news using machine learning and Natural Language Processing techniques while prioritizing explainability for prediction. In recent years, the term "fake news" has gained significant attention, referring to news articles lacking factual basis and often intended to mislead or promote certain agendas (Desai & Oehrli, 2023). These articles may contain outright falsehoods or omit crucial contextual information. The detection of fake news is crucial due to its detrimental impact on society and democratic processes. For instance, during the 2016 United States election, misinformation played a significant role, influencing voters and undermining the democratic process (Guess et al., 2020). Moreover, fake news during the COVID-19 pandemic has disrupted public health responses and posed serious risks to public safety (Nelson et al., 2020).

The data used for production will be a combination of multiple datasets from Kaggle and universities, spanning topics including politics, news, and sports. We decided to use multiple sets to increase the available data and to introduce diversity.

The AI product/model targets various end users, including individuals, media organizations, fact-checking agencies, and social media platforms. Individuals can verify news authenticity before sharing, while media organizations and fact-checkers can utilize it to detect misleading content. Integration into social media platforms would help flag or remove fake news, curbing its spread. Our project aims to develop a system employing natural language processing and machine learning to identify fake news. Using deep learning techniques, we'll classify text as trustworthy or fake, deploying the best-performing model through Gradio for

Detecting Fake News Using Natural Language Processing

user-friendly interaction. Future plans involve expanding to a user interface or mobile app for real-time verification, empowering users to combat misinformation effectively.

Data Summary

To diversify our training data, we aggregated five datasets from Kaggle and the University of Victoria, spanning various domains such as fake news detection, the Syrian war, and the Egyptian Football League. These datasets collectively comprise thousands of text entries, ranging from approximately 7,000 to 20,000 rows each. Our cleaning and preprocessing pipeline involves standard procedures like removing duplicates, null values, and special characters, alongside text normalization techniques such as stemming and lowercase conversion. Standard cleaning steps are applied uniformly, tailored to each dataset's specific characteristics, such as the tweet-based nature of the Egyptian Football League dataset. These steps ensure consistency and quality across all datasets, essential for subsequent analysis and modeling tasks. The resulting dataset, totaling around 94,000 rows and two columns (text and class), exhibits a balanced distribution between fake news (0) and real news (1).

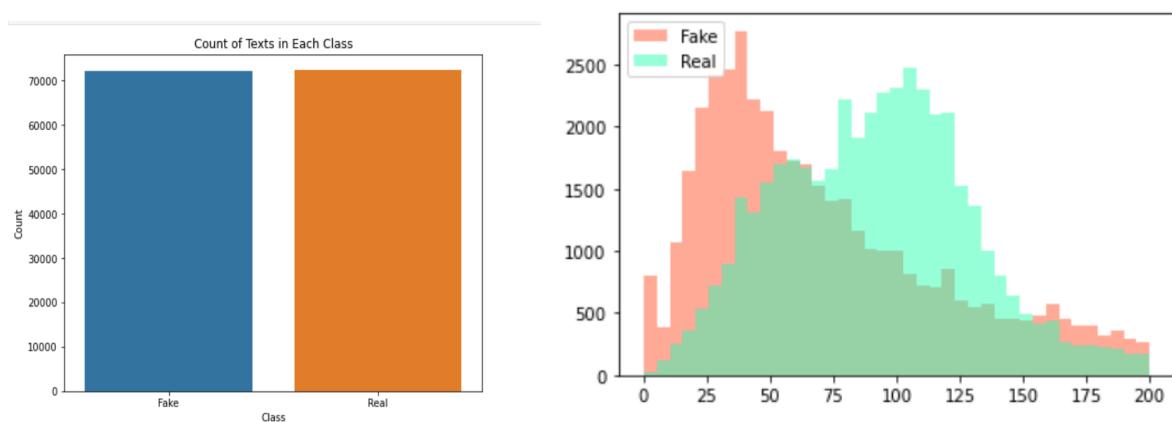


Figure 1: Class balance of labels for the combined dataset and word count in real and fake texts

Detecting Fake News Using Natural Language Processing

During the research process, we discovered that a common feature of fake news is its tendency to use emotional language (Hayes-Bohanan, 2023). To translate this into a predictive feature we employed nltk's SentimentIntensityAnalyzer to perform sentiment analysis. We used the compound score of each piece of text, a float value between -1 and 1 where more negative values mean negative sentiment and positive indicating positive sentiment. When plotting the results by class label we discovered that true text was more likely to use neutral wording while fake news was more likely to use more emotional language. Overall these results support the predictive power of text sentiment for fake news detection.

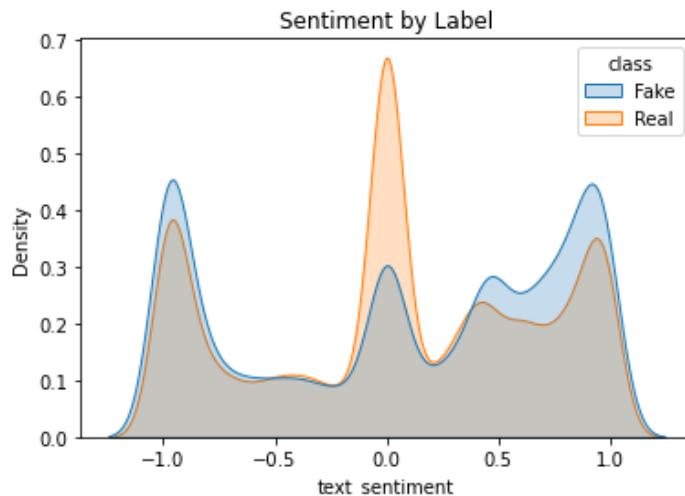


Figure 2: Distribution of sentiment polarity scores in real and fake texts.

Interpreting Fake News Word Cloud

We analyzed the fake and real text by creating Word Clouds displaying common words found in texts. Below are the Word Clouds and our interpretations.

Detecting Fake News Using Natural Language Processing



Figure 3: *Merged Dataset Fake News Wordcloud*

Examination of individual and merged data sets Word Clouds provides the following insight. Firstly, named entities such as "Donald Trump," "Hillary Clinton," "United States," and "White House" are prominent, indicating a focus on people, places, and organizations often exploited to make unverifiable claims. Secondly, the prevalence of emotional language with words like "attack", "didn't", "believe", and "doesn't" suggests an intent to provoke strong reactions in readers. Thirdly, the presence of "pic Twitter" implies dissemination through Twitter, a platform prone to misinformation, urging caution in assessing claims shared there. Lastly, the inclusion of framing indicates manipulation of language to cater to specific audiences, emphasizing the need for critical evaluation and awareness of personal biases when consuming news.

Interpreting True News Word Cloud

Detecting Fake News Using Natural Language Processing

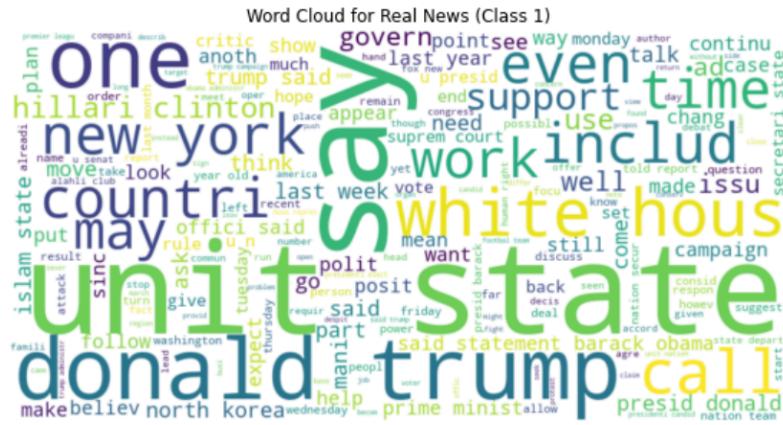


Figure 4: Merged Dataset True News Wordcloud

Analysis of individual and merged data sets Word Clouds provides the following insight.

Firstly, the prevalence of words related to current events such as "Trump," "NRA," "Thursday," "New York," and "election" indicates a focus on recent developments and important issues.

Secondly, the predominantly neutral language observed in the word cloud, featuring words like "meeting", "people", "statement", and "issue" suggests a commitment to factual reporting devoid of emotional language or unsubstantiated claims. Thirdly, the inclusion of sources like "CNN", "BBC", and "AP" signifies a diverse range of credible sources contributing to true news articles, ensuring a well-rounded perspective. Lastly, the presence of factual language with words like "deal," "rule," and "percent" underscores a focus on objective reporting, enhancing the reliability and trustworthiness of true news content.

Background Information

Fake news detection is critical due to its societal impacts, necessitating automated methods amid the vast online information landscape. Natural Language Processing (NLP) techniques, including text classification, sentiment analysis, and topic modeling, have emerged as promising tools for combating misinformation (Waheed et al., 2022). Additionally,

Detecting Fake News Using Natural Language Processing

graph-based models offer effective solutions, as seen in applications targeting celebrity gossip and healthcare misinformation (Chandra et al., 2020). These methods leverage relationships between articles, sources, and entities to uncover patterns of misinformation propagation. Social network analysis (Sivasankari and Vadivu, 2021) is another valuable approach. This type of analysis examines news article dissemination and user interactions on social platforms to identify suspicious patterns, such as those highlighted by Wasim (2020).

The project encompasses three main categories of machine-learning methods. First is traditional algorithms like Logistic Regression, Random Forest, Decision Tree, and Gradient Boosting. These are typically utilized for binary classification tasks (Bharadwaj, Shao, 2019). Long Short-Term Memory (LSTM), a type of Recurrent Neural Network (RNN), is employed for analyzing text data, capable of capturing long-term dependencies (Padalko et al., 2024). Lastly, DistilBERT, a lightweight version of the BERT model (Szczepanski et al., 2021), pre-trained on extensive text data, excels in capturing contextual information and semantic relationships. The model has been proven effective for various NLP tasks including text classification.

Next is LIME (Sangani, 2021), which we will be using for Model Explainability. Local Interpretable Model-Agnostic Explanations (LIME) is a technique used to interpret the predictions of machine learning models. When coupled with LSTM, it provides explanations for the model's decisions, thereby enhancing transparency and trustworthiness.

Our implementation utilizes a Bidirectional Long Short-Term Memory (BiLSTM) network (Padalko et al., 2024), chosen for its effectiveness in various tasks like time-series prediction, natural language processing, and speech recognition. Unlike standard recurrent neural networks (RNNs), LSTMs can look back over 1000 timesteps, thanks to their unique architecture

Detecting Fake News Using Natural Language Processing

featuring forget, input, and output gates with sigmoid activation. Any value that gets multiplied by zero is forgotten while any other values are kept to cascade down the cell and network (Staudemeyer & Morris, 2019).

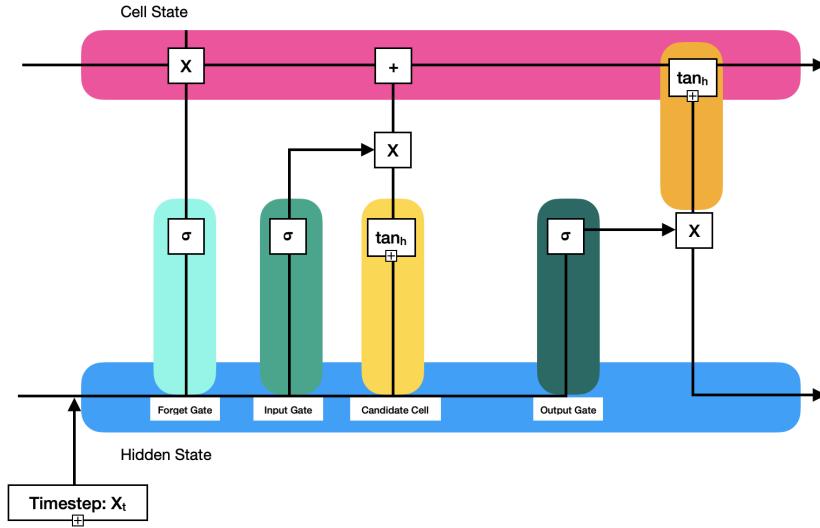


Figure 5: A general layout of an LSTM cell. The inputs are the previous cell state (c_{t-1}), the previous hidden state (h_{t-1}), and the input at timestep t . The cell state acts as a highway that transfers relevant information down the cell. As the highway continues, information is added or removed through the other gates. The hidden state acts as the cell's memory, containing gates that regulate important information.

The forget gate, the first step in an LSTM cell, processes the previous timestep h_{t-1} , and input x_t through a sigmoid function, yielding the forget gate output f_t . The input gate updates the cell state, with the previous hidden state h_{t-1} , and input x_t passes through it. Simultaneously, the same information passes through a tanh function to create candidate values for the cell state. The input gate and candidate values are then pointwise multiplied to update the cell state. The

Detecting Fake News Using Natural Language Processing

previous cell state is updated by pointwise multiplication with the output of the forget gate, added to the output of the input gate and candidate cell's multiplication, followed by a tanh function adjustment. This process yields the final cell state using the equation: (forget gate output f_t) * (previous cell state c_{t-1}) + (input gate output i_t) * (new cell state c_t). Finally, the output gate determines the next hidden state, crucial for prediction, by selectively retaining values from the previous hidden state and input through a sigmoid function. The final products of this LSTM cell are the cell state and the updated hidden state (Phi, 2020).

While initially employing a unidirectional LSTM network, we achieved greater success with a Bidirectional LSTM network. This architecture comprises interconnected LSTM cells that process information both in a forward and backward direction, capturing context from past and future inputs. This is particularly advantageous for NLP tasks. For example, the word “bark” could have different meanings depending on the context, and bidirectional LSTMs have the architecture to find insight from context clues (Zhao, 2023).

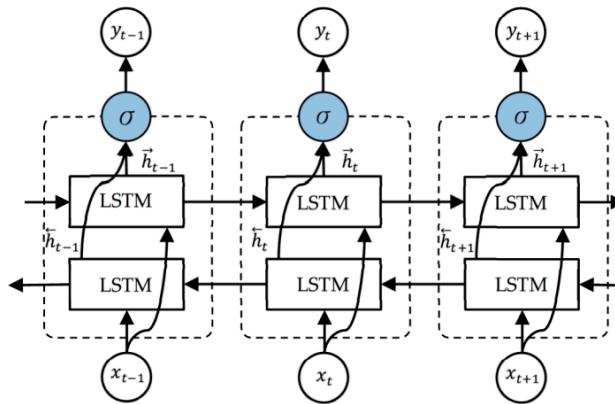


Figure 6: A basic layout of the data flow between bidirectional LSTM cells. A previous input x_{t-1} , current input x_t , and future input x_{t+1} are fed into a backward-facing LSTM network $t-n$ and then fed into a forward-facing LSTM network. Output values are put into a sigmoid function

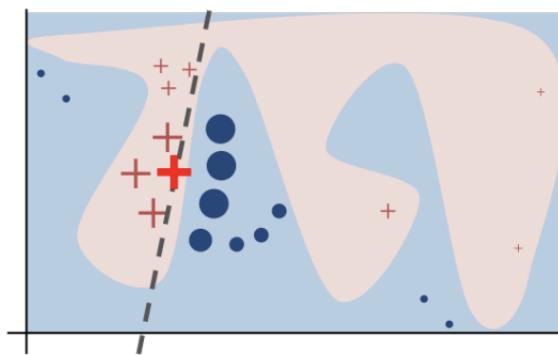
Detecting Fake News Using Natural Language Processing

to place these values into a vector, y , which contains output values of the past, present, and future data (Li et al., 2020).

To enhance the interpretability of our Bidirectional LSTM, we applied LIME (Local Interpretable Model-Agnostic Explanations), aiding in understanding black box model predictions. LIME facilitates users in comprehending why a text may be misleading, fostering learning and awareness of potential red flags. By employing a family of small interpretable linear models, LIME approximates the complex model's output, simplifying explanations while regularizing model complexity (DeepFindr, 2021).

The explanation of LIME is made by the following loss function:

$\xi(x) = \operatorname{argmin}_{g \in G} L(f, g, \pi x) + \Omega(g)$. The variables f are the complex model, g the simple model, and x the input. $L(f, g, \pi x)$ gives an approximation of the complex model, using the simple model in the general area of the input (πx). $\Omega(g)$ Regularizes the complexity of the simple model used to keep the explanation simple. The segment $\operatorname{argmin}_{g \in G}$ minimizes the two loss functions: approximating the complex model in a local area and the complexity measure (DeepFindr, 2021).



Detecting Fake News Using Natural Language Processing

Figure 7: *The pink and blue area represents the complex model's decisions f , which LIME is unaware of. The location of the bold red cross is the requested instance to be explained, the other crosses/circles are instances that use the model. These are then weighed based on how close it is to the requested instance. The dashed line is the explanation determined, but only locally* (Ribeiro, 2016).

Prior Research

Bidirectional LSTM Networks have been effectively utilized for fake news detection. Islam et al. (2022) developed a Bidirectional LSTM model capable of classifying news articles into False, Partially False, and True categories. Bahad et al. (2019) conducted a comparative study, evaluating Bidirectional LSTM against other models for binary fake news classification.

One relevant paper for our project is Hamed et al. (2023), the model incorporates sentiment analysis by analyzing comments, employing Bidirectional LSTM, and utilizing sentiment analysis with the TextBlob library. The model achieved notable performance with 96.89% accuracy and 97.81% F1 score.

The model combined the output of the Bi-LSTM and the sentiment analysis via concatenation. This combined output was then fed through a sigmoid function to obtain the final prediction (Hamed et al., 2023). We hope to draw inspiration from this paper and incorporate the concatenation method into our model.

Methods

Various machine learning models including logistic regression, decision trees, gradient boosting, and random forest were utilized. Techniques like stratified k-fold cross-validation and grid search have been implemented for hyperparameter tuning, particularly with random forest

Detecting Fake News Using Natural Language Processing

classifiers. Deep learning methods such as Long Short-Term Memory (LSTM) networks, featuring single-layer, stacked, and bi-directional configurations, are employed to capture temporal dependencies in textual data, alongside advanced models like DistilBERT, to discern semantic intricacies within textual data.

The first phase of model training involved employing various supervised learning algorithms, including logistic regression, decision trees, gradient boosting, and random forest classifiers. The data was split into training and testing sets using a 75-25 ratio. The accuracy scores for the random forest were the best among all classification models. Cross-validation techniques such as stratified k-fold validation and hyperparameter tuning using grid search were utilized further to enhance model robustness and performance.

In the second phase of training, tokenization and padding of text data were performed. Data was split into training and testing sets using an 80-20 ratio. Three different types of LSTM models were trained: basic LSTM, stacked LSTM, and bidirectional LSTM. Each LSTM model was compiled using the RMSprop optimizer and binary cross-entropy loss function. Confusion matrices and classification reports were generated to evaluate the performance of each model on the testing data, with the bidirectional LSTM model achieving the highest accuracy.

In the third phase, we took advantage of Bi-LSTM's capability to learn from both past and future sequences simultaneously and evaluated multilayer bidirectional LSTM models. The two-layer model emerged as the most favorable option due to its relatively simpler architecture, shorter training duration, and efficient parameter utilization. Despite being simpler, it achieved commendable accuracy and demonstrated efficient resource utilization compared to its counterparts where more layers were involved.

Detecting Fake News Using Natural Language Processing

Further, sentiment analysis results were incorporated as features for classification. Three models were trained with sentiment features: Model 1 was a Bi-LSTM with sentiment score concatenation, Model 2 was a Bi-LSTM with two dense layers, and Model 3 involved a Bi-LSTM with five dense layers. While Model 3 achieved the highest accuracy on the test set, the marginal improvement in accuracy compared to Model 1 suggests that the added complexity is not justified. Further experimentation with LIME was built to assess the explainability of the models.

The Bidirectional LSTM model features two Bidirectional LSTM layers, effectively capturing temporal dependencies within text data while balancing complexity and computational efficiency. ReLU activation functions are employed in dense layers to mitigate the vanishing gradient problem, with sigmoid activation utilized in the output layer for binary classification. Dropout layers are strategically incorporated to prevent overfitting by randomly dropping units during training, with dropout rates adjusted based on model complexity to maintain generalizability. Adding three or four layers to the model brings unnecessary complexity without significant accuracy improvements, resulting in diminishing returns.

The model optimization process involved adjusting hyperparameters and architectural elements to improve performance and efficiency. This optimization included fine-tuning parameters such as learning rate, batch size, dropout rates, and optimizer choice through iterative adjustments based on training and evaluation results. The number of nodes and hidden layers was optimized to balance model complexity and computational efficiency, with multiple Bidirectional LSTM layers being incorporated to capture temporal dependencies. Activation functions, including ReLU, were chosen to introduce non-linearity and mitigate the vanishing

Detecting Fake News Using Natural Language Processing

gradient problem. Dropout regularization was strategically chosen to prevent overfitting.

Adjustments were made to the embedding dimension and vocabulary size to balance the richness of word representations and computational efficiency.

The final step involves the integration of a pre-trained two-layer BI-LSTM deep learning model, with Local Interpretable Model-agnostic Explanations (LIME) for text classification tasks. The model is loaded from a pre-trained h5 file, eliminating the need for training from scratch. Text input undergoes preprocessing, including tokenization, removal of stopwords and punctuation, and stemming, to ensure uniformity in input representation. LIME provides local explanations for model predictions, highlighting the key features contributing to each prediction. The explanations are visualized as HTML content and accompanying images, facilitating human understanding of the model's decision-making process. The Gradio library enables the creation of a user-friendly interface for interactively inputting text and viewing both model predictions and LIME explanations, enhancing interpretability and transparency in the classification process.

Results

Table 1

Results (Weighted Average) From a Variety of Machine/Deep Learning Methods Used to Detect Fake News.

Model	Accuracy	Precision	Recall	F1-Score
Random Forest	0.94	0.94	0.94	0.94
Distill-BERT	0.95	0.95	0.95	0.95
LSTM	0.94	0.94	0.94	0.94
Stacked LSTM	0.94	0.94	0.94	0.94
Bidirectional LSTM	0.94	0.94	0.94	0.94
Two-layer Bi-LSTM (Best Model)	0.94	0.94	0.94	0.94

Detecting Fake News Using Natural Language Processing

Three-layer Bi-LSTM	0.94	0.94	0.94	0.94
Four-layer Bi-LSTM	0.93	0.93	0.93	0.93
Bidirectional LSTM w/ Sentiment Analysis	0.94	0.94	0.94	0.94
Two-layer Bi-LSTM w/ Sentiment Analysis	0.94	0.94	0.94	0.94
Five-layer Bi-LSTM w/ Sentiment Analysis	0.94	0.94	0.94	0.94

The results showcase strong performance across various models, with Random Forest, LSTM, Stacked LSTM, and Bidirectional LSTM achieving an accuracy, precision, recall, and F1-score of 0.94. Overall we observed a ceiling effect, where all models scored high, with little variation between architectures.

Interestingly, the addition of sentiment analysis to Bidirectional LSTM models maintains this high-performance level, even though it did not help to yield better performance. However, as the number of layers increases in the Bidirectional LSTM architecture, a slight decrease in performance is observed, with the four-layer model achieving a slightly lower accuracy, precision, recall, and F1-score of 0.93. Overall, the Bi-LSTM model with two layers is preferred for its simplicity, efficiency, and accuracy.

Despite the complex and noisy nature of the dataset, which required extensive data cleaning the Bi-LSTM model demonstrated robust performance without overfitting. This was enabled by the inherent properties of LSTM (Long Short-Term Memory) networks, which excel at capturing temporal dependencies in sequential data like text, providing resilience against overfitting. The training accuracy/loss curves exhibited consistent improvement over epochs, with decreasing loss and increasing accuracy across different variations, indicating effective learning and convergence of the model. The best model achieved an impressive accuracy of

Detecting Fake News Using Natural Language Processing

94.20% on the test set, further confirming its effectiveness in accurately classifying fake news articles.

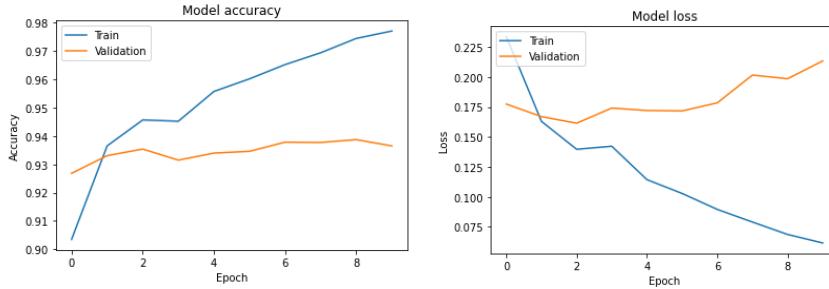


Figure 8: Two-layer Bi-LSTM Accuracy and Loss Curves

One rather unexpected result was with models including sentiment analysis. Based on our preliminary analyses and literature review the sentiment analysis score as a feature was predicted to boost performance. While many factors could cause this discrepancy, we have some educated guesses. First is the aforementioned ceiling effect, observed in the model results, leaving little room for improvement.

Second, is that the tool, SentimentIntensityAnalyzer, may not be well suited for a proportion of our text. Data used for training included both social media posts and news articles. Social media posts tend to be less structured and shorter than articles. The pre-trained nltk sentiment analyzer is documented to be less effective with longer and more structured texts (White, 2020). In Hamed et al.(2023) TextBlob was used for sentiment analysis. This library is typically used for more formal language analysis (White, 2020). However, Hamed et al.(2023) were able to use TextBlob on social media-like data effectively. TextBlob could potentially be a fit for our mixed dataset in the future.

Our team hypothesized that certain words, phrases, and sentences would help a model identify if news was fake or misleading. To try and understand our model's high performance,

Detecting Fake News Using Natural Language Processing

we utilized LIME for some explanations. Since the dataset is quite diverse, there was found to be certain language used in the majority of fake news articles that cued our model into predicting quite accurately. For example, one common theme in fake news that our team noticed was the lack of a professional tone, such as referring to Hillary Clinton as just “Hillary”. The word “Hillary” is flagged as fake, while “Clinton” is not because it is perceived to be more professional when referring to a person previously mentioned by their last name. Our team also noticed that words getting repeated frequently were also flagged as fake. In one example, shown below “FBI” is repeated 10 times in one paragraph.

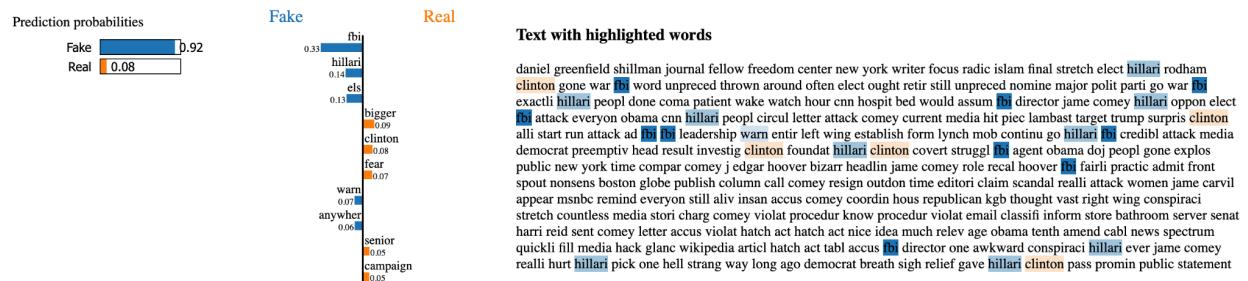


Figure 9: The LIME explanation from our two-layer bidirectional LSTM when a fake news article is passed through.

Since our project’s goal was to help users identify fake news and educate them on how to identify fake news in the future, performance is quite important. Achieving high accuracy and precision was important to our team. Accuracy is a measure of overall correctness in our model while precision is a measure of positive predictive values. It is more harmful to the user if a fake news article is labeled as correct (false positive) because misleading information can be spread throughout a community. There is often a tradeoff between recall and precision, which leads to

Detecting Fake News Using Natural Language Processing

our team focusing more heavily on architectures that value a high precision on the positive, real news value.

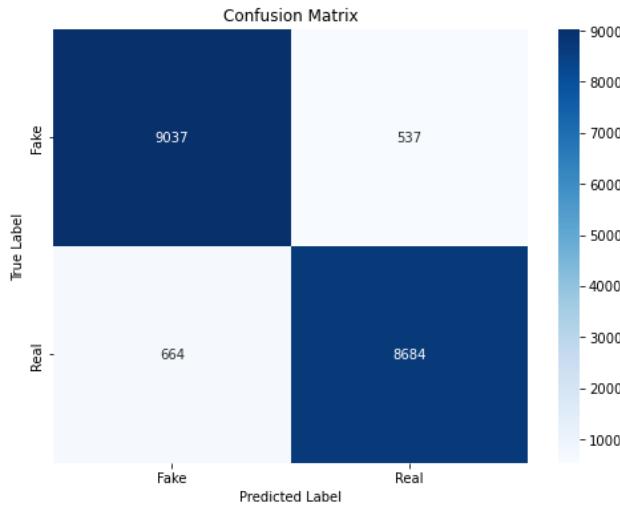


Figure 10: Model Confusion Matrix

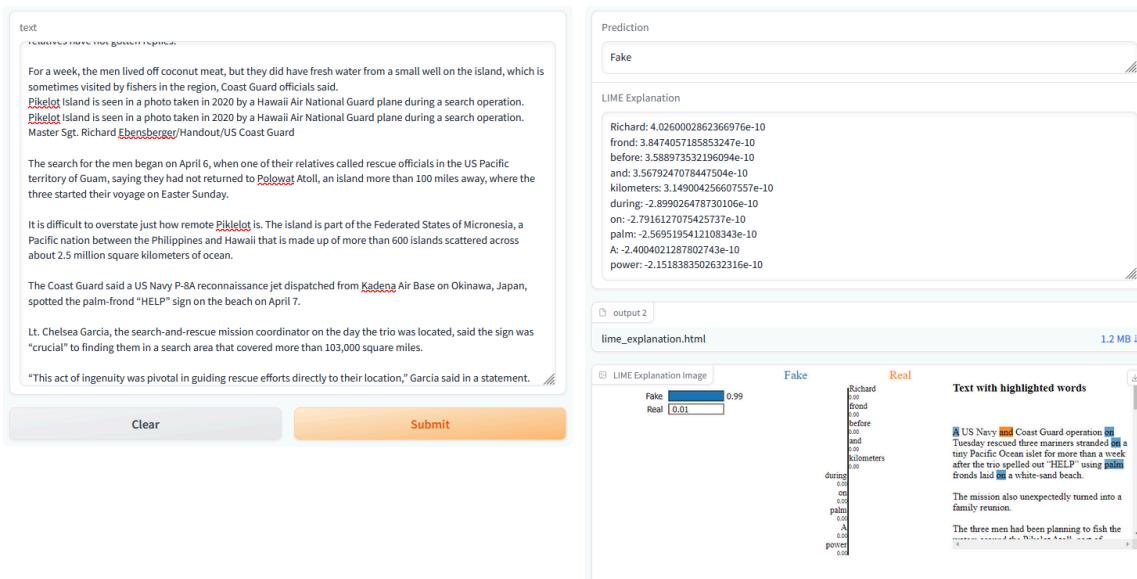


Figure 11: Gradio UI working with Model and predicting fake news with explanation.

Detecting Fake News Using Natural Language Processing

The figure above depicts a webpage constructed using Gradio, facilitating the input of news articles. Upon clicking the submit button, a pre-trained model is invoked to predict whether the news is fake or real. In addition to the prediction, the webpage displays explanations highlighting keywords, distinguishing between fake and true news. These explanations are presented both visually in an image format and in downloadable HTML format, enhancing the user's understanding of the prediction process.

Conclusion

Future Directions

In the future, we aim to incorporate multimodal (text and images) prediction into our system, a secondary goal that couldn't be explored in the current project scope. One prominent multimodal dataset, Fakeddit, features Reddit-like posts with text and image pairs, comments, and metadata, boasting over a million samples (Nakamura et al., 2020). While the dataset's size offers learning advantages, it also poses challenges due to significant time and computational requirements, potentially requiring sampling for balance. For image data, a Convolutional Neural Network (CNN) architecture is preferable, such as custom models or pre-trained options like ResNet or VGG-16, which would complement the LSTM model in tandem.

Model Productionization

In real-world implementation, user interaction with the model is pivotal. Hosting the model on an online interface or mobile app, either standalone or integrated into social media platforms, is proposed for single-instance prediction with minimal latency. Given the evolving nature of information, regular data updates and model retraining are imperative, facilitated by a Continuous Integration/Continuous Deployment (CI/CD) pipeline. Monitors will be employed to

Detecting Fake News Using Natural Language Processing

detect significant performance drift. Licensing the model with a Responsible AI license ensures ethical use, with a code of ethics guiding restrictions to balance innovation and ethical considerations (Ding, 2022).

Detecting Fake News Using Natural Language Processing

References

- Ahmed H, Traore I, Saad S. (2017). "Detection of Online Fake News Using N-Gram Analysis and Machine Learning Techniques. In: Traore I., Woungang I., Awad A. (eds) Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments. ISDDC 2017. Lecture Notes in Computer Science, vol 10618. Springer, Cham (pp. 127- 138).
- Alhasan, Mohammad (2020) Dataset: Fake News Dataset [Kaggle]
<https://www.kaggle.com/datasets/mohamadalhasan/a-fake-news-dataset-around-the-syria-n-war>
- Ahmed, Wasim, et al. (2020, May) Article: COVID-19 and the 5G Conspiracy Theory: Social Network Analysis of Twitter Data
<https://www.jmir.org/2020/5/e19458>
- Bahad, P., Saxena, P., & Kamal, R. (2019). Fake News Detection using Bi-directional LSTM-Recurrent Neural Network. *Procedia Computer Science*, 165, 74–82.
<https://doi.org/10.1016/j.procs.2020.01.072>
- Bhatia, Ruchi (2020), Dataset: Source-based Fake News Classification [Kaggle]
<https://www.kaggle.com/datasets/shawkyelgendi/fake-news-football?select=real.csv>
- Bharadwaj, Pranav, and Shao, Zongru (June, 2019) Article: Fake news detection with semantic feature and text mining.
https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3425828
- Chandra, Shantanu et al., (2020, Nov 23) Article: Graph-based Modeling of Online Communities for Fake News Detection
<https://arxiv.org/abs/2008.06274>

Detecting Fake News Using Natural Language Processing

Desai, S., & Oehrli, J. A. (2023, June 6). Research Guides: “Fake News,” Lies and Propaganda:

How to Sort Fact from Fiction: What is “Fake News”? Umich.edu.

<https://guides.lib.umich.edu/fakenews>

Ding, J. (2022). *TTW: Machine Learning Model Licenses*. HackMD.

<https://hackmd.io/@jending12/HyvMU8sJo>

EL-GENDY, Shawky [2023, March], Dataset: Fake News Football [Kaggle]

<https://www.kaggle.com/datasets/shawkyelgendy/fake-news-football?select=real.csv>

Guess, A. M., Nyhan, B., & Reifler, J. (2020). Exposure to untrustworthy websites in the 2016 US election. *Nature Human Behaviour*, 4(5).

<https://doi.org/10.1038/s41562-020-0833-x>

Hamed, S. Kh., Ab Aziz, M. J. A., & Yaakub, M. R. (2023). Fake News Detection Model on Social Media by Leveraging Sentiment Analysis of News Content and Emotion Analysis of Users’ Comments. *Sensors*, 23(4), 1748.

<https://doi.org/10.3390/s23041748>

Hamed, Suhaib Kh. Et al. (2023, February 04) Article: Fake News Detection Model on Social Media by Leveraging Sentiment Analysis of News Content and Emotion Analysis of Users’ Comments

<https://www.mdpi.com/1424-8220/23/4/1748>

Islam, T., MD Alamin Hosen, Mony, A., MD Touhid Hasan, Jahan, I., & Kundu, A. (2022). A Proposed Bi-LSTM Method to Fake News Detection. *2022 International Conference for Advancement in Technology (ICONAT)*.

<https://doi.org/10.1109/iconat53423.2022.9725937>

Detecting Fake News Using Natural Language Processing

Mahi, Sadman Sakib (2023, June) Dataset: Fake News Detection Dataset with Pre-trained Model

[Kaggle]

<https://www.kaggle.com/datasets/sadmansakibmahi/fake-news-detection-dataset-with-pre-trained-model/data>

Nelson, T., Kagan, N., Critchlow, C., Hillard, A., & Hsu, A. (2020). The Danger of Misinformation in the COVID-19 Crisis. *Missouri Medicine*, 117(6), 510–512.

<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7721433/>

Padalko, Halyna, et al. (2024, Jan 08) Article: A novel approach to fake news classification using LSTM-based deep learning model.

<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10800750/>

Phi, M. (2020, June 28). *Illustrated guide to LSTM's and GRU's: A step-by-step explanation*. Medium. <https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21>

DeepFindr. (2021, February 25). *Explainable AI explained! | #3 LIME* [Video]. Youtube.
<https://www.youtube.com/watch?v=d6j6bofhj2M>

Li, Y.-H., Harfiya, L. N., Purwandari, K., & Lin, Y.-D. (2020). Real-time cuffless continuous blood pressure estimation using Deep Learning Model. *Sensors*, 20(19), 5606.

<https://doi.org/10.3390/s20195606>

Nakamura, K., Levy, S., & William Yang Wang. (2020). Fakeddit: A New Multimodal Benchmark Dataset for Fine-grained Fake News Detection. Proceedings of the 12th Conference on Language Resources and Evaluation (LREC 2020), 6149–6157.

Detecting Fake News Using Natural Language Processing

Raza S, Ding C. (2022, Jan 30) Research Paper: Fake news detection based on news content and social contexts: a transformer-based approach. *Int J Data Sci Anal.* 2022;13(4):335-362.
doi: 10.1007/s41060-021-00302-z. Epub 2022 Jan 30. PMID: 35128038; PMCID: PMC8800852.

<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8800852/>

Ribeiro, M., Singh, S., & Guestrin, C. (2016). “Why should I Trust You?”: Explaining the predictions of any classifier. *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations.*

<https://doi.org/10.18653/v1/n16-3020>

Sangani, Raj (2021, Nov 11) Article: Interpreting and LSTM through LIME

<https://towardsdatascience.com/interpreting-an-lstm-through-lime-e294e6ed3a03>

Sivasankari, S, and Vadivu, G. (2021, July 21) Article: Tracing the fake news propagation path using social network analysis.

<https://link.springer.com/article/10.1007/s00500-021-06043-2>

Szczepanski, Mateusz, et al. (2021, Dec 08) Article: New explainability method for BERT-based model in fake news detection.

<https://www.nature.com/articles/s41598-021-03100-6>

Staudemeyer, R. C., & Morris, E. R. (2019). – *Understanding LSTM – a Tutorial into Long Short-Term Memory Recurrent Neural Networks.*

University of Victoria, Dataset: The ISOT Fake News Dataset

Detecting Fake News Using Natural Language Processing

<https://onlineacademiccommunity.uvic.ca/isot/2022/11/27/fake-news-detection-datasets/>

White, B. (2020). *Sentiment Analysis: VADER or TextBlob?* Medium.

<https://towardsdatascience.com/sentiment-analysis-vader-or-textblob>

Yuan, L.; Jiang, H.; Shen, H.; Shi, L.; Cheng, N. (2023, September 04) Research Paper:

Sustainable Development of Information Dissemination: A Review of Current Fake News Detection Research and Practice.

<https://www.mdpi.com/2079-8954/11/9/458>

Waheed, Samer Abdulateef et al., (2022, Feb 22) Article: Topic Modeling and Sentiment Analysis of Online Education in the COVID-19 Era Using Social Networks Based Datasets

<https://www.mdpi.com/2079-9292/11/5/715>

Zhao, Y. (2023, December 19). *RNN, LSTM, and Bidirectional LSTM: Complete guide: Dagshub.* DagsHub Blog. <https://dagshub.com/blog/rnn-lstm-bidirectional-lstm/>

Human versus Artificial Intelligence Distinguishment

Group 2

Jeremy Cryer, Jason Raimondi, and Shane Schipper

Shiley-Marcos School of Engineering, University of San Diego

AAI 590 Capstone Project

Professor Anna Marbut

4/15/2024

Introduction

The objective of this project is to determine the origin of text, enabling the capability to distinguish the creation of the content from either human or artificial intelligence (AI). As AI continues to improve rapidly, it is becoming increasingly difficult to differentiate content between being generated by humans or AI. This is a positive feature for many AI applications; however, this also introduces risk for others. Consider scenarios such as fake text or images displayed in news feeds or AI-generated essays submitted by students for coursework. This can result in the spread of misinformation, issues with academic integrity, as well as hurting academic performance.

Essentially anyone with an internet-connected device (e.g., computer, mobile device) can be the end user of the team's AI application, providing this capability to predict content origin. In the couple examples discussed above, anyone could use this application to check text for their origin if they came across something in their newsfeed that they are questioning. In the education example, teachers can make use of this application to check submitted essays and other papers to help uphold academic integrity. Academic integrity has become a large concern due to the capability of AI that even universities, such as the University of Kentucky, have sent out notices to their students to help control this. As stated from a post from the school, it states, "We are committed to upholding academic integrity and ensuring fairness. As part of that process, we want to highlight resources available to support proctoring and plagiarism detection. Earlier this month, Turnitin – a vendor we utilize – began providing an artificial intelligence (AI) detector, integrated with Canvas, which offers an estimate of how many sentences in a written submission may have been generated by artificial intelligence. According to the software developers, educators can use this estimate, as they do with the plagiarism detector output, to determine if

further review, inquiry or discussion with the student is needed” (Stokes, 2023). This post came from the university's administration alerting everyone that AI use is a concern for academic integrity, and they are using AI content detectors, such as ones being sought out in this project, to help address this problem.

This project uses data from multiple datasets, all openly available on Kaggle¹. First, there is an AI versus human text dataset, which contains nearly 500,000 AI and human-generated essays, gathered from multiple sources which is from Gerami, S. (2024). For additional data, this project also uses a dataset for AI-generated versus student-generated text. This is a smaller dataset, with 1,103 samples provided by Dongre, P. (2023). However, it provides some additional samples with added variety. This dataset is also more informal, so it is possible that grammar could prove to be an important factor. These datasets are anticipated to be sufficient for development of a minimum viable solution to the problem.

The primary goal of this project is performing binary classification and providing the result (i.e., generated by human or AI) to the AI application users. To make the application results more appealing and trustworthy, this team also plans to provide the probability (i.e., confidence) of its predictions. As a secondary project goal, the team plans to develop an interactive web application, providing users with the ability to enter or paste in copied text for analysis and obtain prediction results on-demand.

Dataset Summary

For this section, the data selection and preprocessing performed will be discussed. This section will go over the data selection, details, preprocessing, and common relationships noticed as the Exploratory Data Analysis (EDA) and feature engineering processes are performed. Data is the cornerstone of any predictive model, and, if not properly selected and understood, the

¹ <https://www.Kaggle.com>

model's performance is likely to be inadequate. The quality of the data used to train and test on directly affects the model's performance. Even the best-performing algorithms can prove to be useless when paired with poor data quality.

Dataset Selection

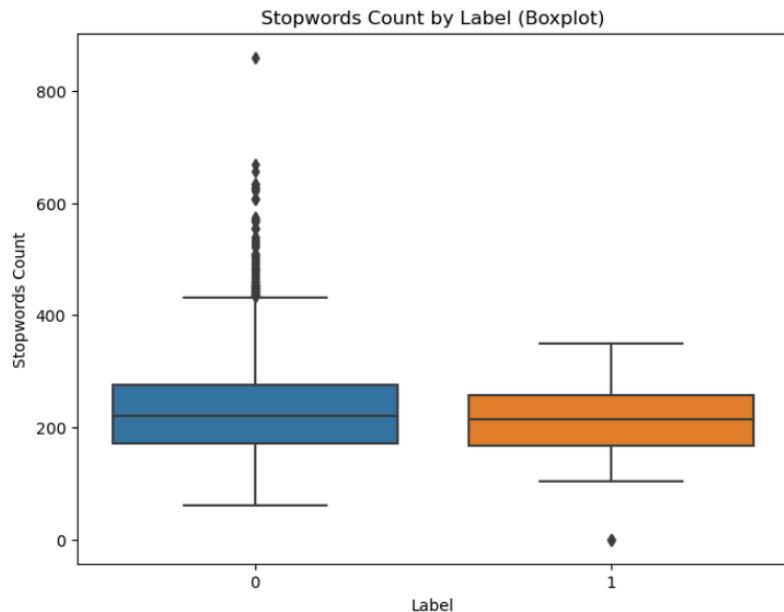
The main dataset selected was a very simple yet vast data set. It was found on Kaggle, provided by Gerami, and contains data for text created by both AI and human sources. The dataset contains nearly 500,000 samples of text labeled as either AI or human-generated. No other features came with the raw dataset. A secondary dataset was also used for small batch testing, as it consisted of less formal text and was at more of a high school student writing level. This dataset has the same features as the main dataset, but it is much smaller, with only slightly over 1,000 samples. This dataset was also found on Kaggle, provided by Dongre. This dataset is to represent an edge case, where the grammar from a high school level text is drastically different from expected AI-generated text and will help the team's understanding if the model is performing well on “easy” predictions. This dataset is only being used as an additional gauge of initial performance but will not be used for evaluation of the team's developed models.

EDA, Preprocessing, and Feature Engineering

When investigating the main dataset, it was determined that data cleaning and preprocessing needs were minimal. From a data incompleteness standpoint, only one data point was missing, so the sample was simply discarded from the dataset. Tokenization is a common feature engineering step for any Natural Language Processing (NLP) problem. As stated on a stackacademic blog entry, “Tokenization is very important as it is a base step of feature engineering, and it determines how our model will interpret the data. Thus, it is very important for an NLP Engineer to tokenize the data appropriately in order to avoid confusion” (Patel,

2023). Patel (2023) also mentions other commonly-used steps during the tokenization process, such as considerations to dropping all upper case, punctuation, and stop words. Normally, this is a key preprocessing step for any NLP analysis. However, in this article, it states, “Deleting stop words in the pre-processing stage impacted the classification performance negatively since the selection of stop words play a crucial role in differentiating human and AI” (Islam et al, 2023). Originally, it was contemplated to perform these additional steps, as most NLP models benefit from this, but the logic behind this and the additional article’s insights suggested against doing so. In response to this information, the team performed EDA to understand the difference and potential impact of stop words between the two classes. To accomplish this analysis, a new feature was engineered to list the stop word count for each data sample. According to Figure 1 and Table 1, where label 0 is human-generated and 1 is AI-generated, there is a vast difference in stop words between the two text generation sources.

Figure 1



Note: Stop Words Count by Label

Table 1

label	count	mean	std	min	25%	50%	75%	max
0	3964.0	231.314834	86.460627	60.0	172.0	221.0	276.0	860.0
1	1036.0	213.020270	53.243843	0.0	167.0	213.5	258.0	350.0

Note: Table of Statistics for Stop Words by Label.

This indicates that the extra granularity provided by inclusion of stop words could prove to have predictive power for the team's models. This discovery was one of the strongest relationships identified during EDA that helps support limiting certain preprocessing steps to suit this specific problem. Due to this realization, it was decided to keep stop words during tokenization, such as punctuation and capitalization. Additional features, such as word count, perplexity, sentiment polarity, etc., were also engineered for further consideration, as limited features came with the chosen dataset. A list containing engineered features is displayed in Figure 2.

Figure 2

	Feature
0	orig_str
1	word_count
2	label
3	avg_sentence_length
4	perplexity
5	sentiment_polarity
6	stopwords_count
7	punctuation_count
8	flesch_reading_ease
9	flesch_kincaid_grade

Note: List Containing Engineered Features

Background Information

For this problem, the end goal is to create an application that can successfully and accurately differentiate between text generated by either human or AI. There have been several models and applications created to this day that can accomplish this with varied approaches.

After researching this problem and existing solutions, successful outcomes were more prominent with traditional machine learning algorithms such as Extreme Gradient Boosting (XGBoost), as well as with deep learning transformer-based model architectures such as DistilBERT (Sanh et al., 2020). As discussed in this section, the architecture approaches that were attempted to solve this problem will be discussed at a high level.

Traditional Machine Learning Approaches

Traditional machine learning approaches to this problem have largely focused on algorithms such as random forests, Support Vector Machines (SVMs), and XGBoost. Models based on these algorithms are typically trained on datasets composed of meta-text features, as the raw text sequence is incompatible. Generally, with a robust feature set, these various types of models perform well but may be lacking as compared to more complex model architectures, such as transformers.

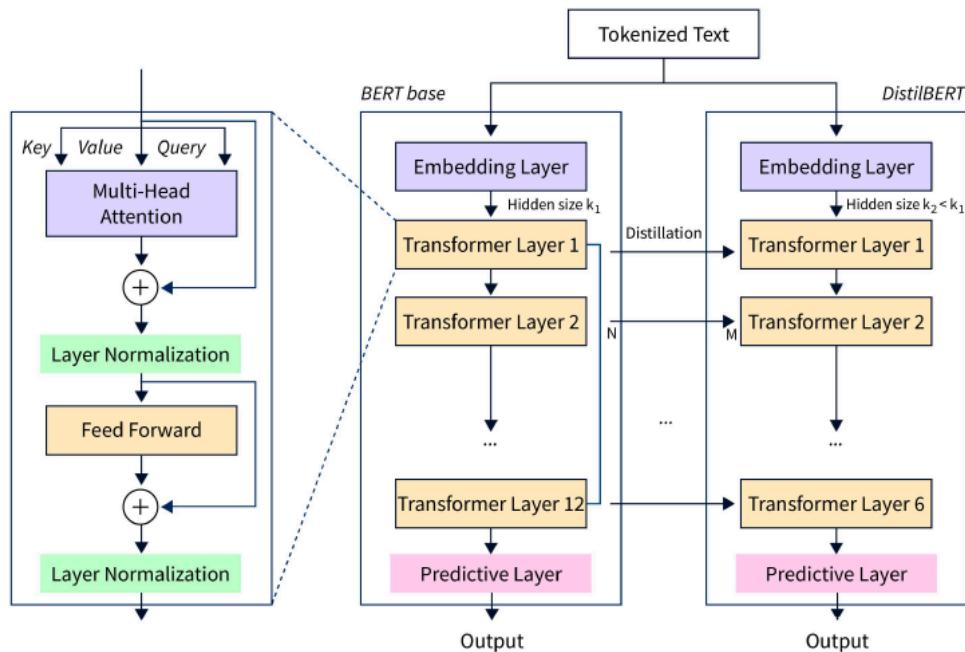
Previous works, such as from Lorenz Mindner (2023), have focused largely on features, such as perplexity, to help perform classification. This feature refers to the predictability of words by a language model based on the previous token. Per Lorenz Mindner (2023), additional features such as word frequency counts, stop word and punctuation counts, and sentiment polarity have also been shown to be effective for this task. A combination of these features should form a robust feature set for baselining traditional machine learning methods for comparison to transformer-based approaches, which will be discussed below.

DistilBERT

DistilBERT is one example of a model architecture that has been proven to work for this application. It was found that Kumar et al. (2024) approached this problem by using a popular LLM, DistilBERT, which is a “distilled”, or, more lightweight version of BERT (Bidirectional

Encoder Representations from Transformers) and trained on the same corpus of text. As the names imply, both variants are transformer-based models containing an encoder but no decoder. The DistilBERT architecture can be seen in Figure 3.

Figure 3



Note: Architecture of DistilBERT (Islam, 2023)

As can be seen when referencing the figure, the architecture takes tokenized text as input and uses an embedding layer, six transformer layers (versus twelve in BERT base), and a predictive layer that generates output. Each transformer layer contains a multi-head attention mechanism, layer normalization, a feed forward neural network, and another layer for normalization.

In this research project, experiments performed by Kumar et al. (2024) concluded that DistilBERT achieved high performance in the task of detecting AI-generated text, around 94

percent overall accuracy. Combined with the efficiency of this distilled version of BERT, it provides a promising solution to this problem.

Experimental Methods

For this problem, the team chose to experiment with three different models, a pre-trained DistilBERT transformer model, a custom PyTorch transformer model, and a model based on traditional machine learning algorithms. Previously mentioned, in the methodology background section, all three models and their conceptual background was discussed. Due to limitations about compute resources available for a large dataset (i.e., nearly 500,000 samples), all three models were presented with a similar challenge. This constraint influenced some of the choices made in how the models were trained, evaluated, and optimized, which is discussed in more detail for each specific model.

Pre-trained DistilBERT

For the pre-trained DistilBERT model, the pre-trained “cased” variant was selected to take into account capitalized words and letters which were deemed likely useful for the prediction being made. This preloaded architecture uses a vocabulary size of 28,996 tokens, 512 maximum position embeddings or sequence length, six transformer layers, 12 attention heads for each attention layer, 3,072 hidden dimensions of the feedforward layer, 20 percent dropout rate for the sequence classifier, and a Gaussian Error Linear Unit (gelu) activation function.

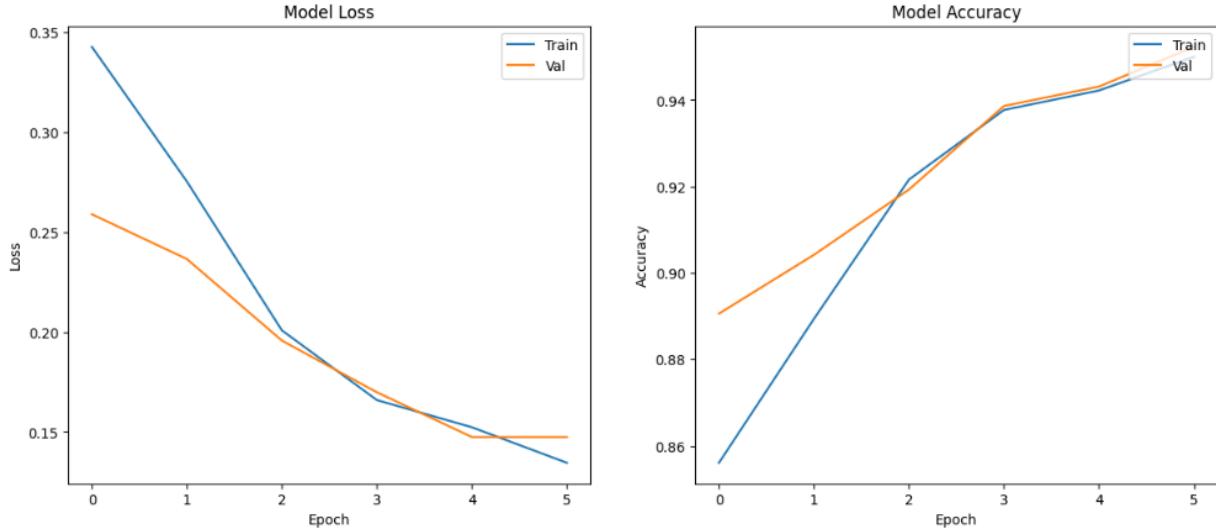
For this and the additional models that will be mentioned, data was split in the same manner. First, either the entire or subset of the dataset is imported. Afterwards, the majority class (i.e., Human - Class 0) is down sampled until it balances the dataset with an equal number of samples as AI - Class 1. The resulting balanced dataset is then split into 90 percent training and 10 percent testing, with an additional split of the training dataset into 80 percent training and 20

percent validation. For each of these splits, stratified sampling and shuffling was used to maintain a class balance and properly mix the samples. Torch datasets were then created while tokenizing and padding the sequences, followed by creation of data loaders to batch the datasets. A batch size of 16 was used, as larger batch sizes (e.g., 32 and 64) were prone to GPU memory issues. A learning rate of 0.001 was configured and Binary Cross Entropy (BCE) loss was used as the loss function during training, as the problem at hand is a binary classification task with only two classes. A training loop with binary accuracy was created, which allowed each training epoch to display loss and accuracy for both training and validation datasets. This same training loop was also used for the custom model.

After experimenting with this pretrained model using different data sizes and carefully reviewing preparation items, the team was not able to obtain good results. Accuracy for both the train and validation datasets was around 50 percent each, which is the same as random chance for these binary predictions. To optimize, the team modified the preloaded model's configuration parameters. First, the vocabulary size was increased from 28,996 to 200,000. The custom transformer's vocabulary was around this size and had good performance, so it was decided to optimize the pretrained model in some aspects to be more like the custom model. It is important to note that vocabulary sizes are not generally this large, however, this custom vocabulary was because the team used a different tokenizer for the custom transformer than the standard tokenizer associated with DistilBERT models. This tokenizer was a function based on words instead of sub-words traditionally configured for a smaller size limit. Next, the following parameters were reduced: hidden dimensions from 3,072 to 1,024, transformer layers from six to two, and sequence classifier dropout rate from 0.2 to 0.05.

Following optimization, the training results were more promising. The training run was configured for 10 epochs but stopped after six due to a timeout on the Google Colab session. However, it was still sufficiently trained at that point, with the train accuracy at 0.9500 and the validation accuracy at 0.9523. The model learning curves can be seen in Figure 4.

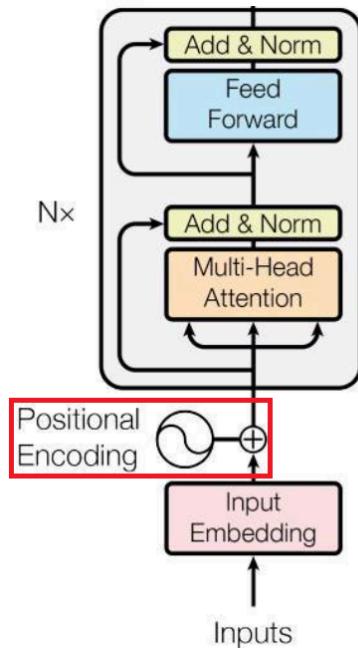
Figure 4



Note: Training - DistilBERT Model Loss and Model Accuracy

Custom Transformer

Due to the research and conclusions with DistilBERT, it was determined to not only experiment using a pre-trained DistilBERT model but to also build a custom transformer model from scratch using the PyTorch framework provided by Paszke et al. (2019). In this case, the framework provides flexibility to create similar layers as the DistilBERT architecture uses. To reduce the complexity level for this model, pre-built transformer layers were used, which already contain the additional components as seen in Figure 3. It was also beneficial to use some modified portions of code from the PyTorch transformer tutorial, provided by “Language modeling” (n.d.), in order to incorporate positional encoding (See Figure 5) which was needed to provide the model with information about the sequences of text.

Figure 5

Note: Positional Encoding in Transformer Architecture (“Language modeling,” n.d.)

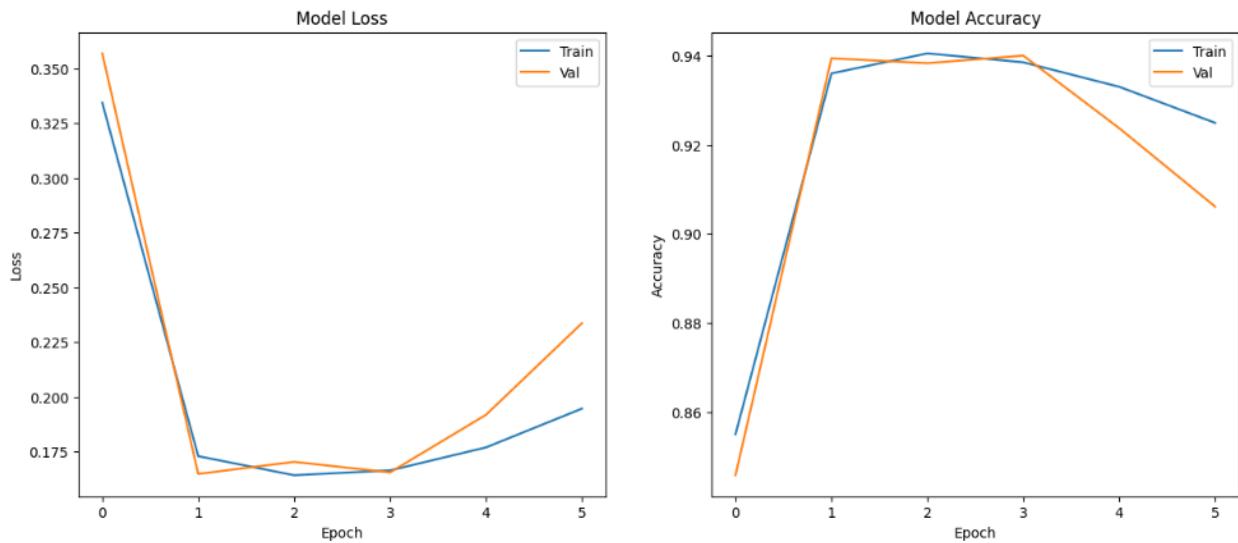
This architecture enabled more customization and flexibility to the model as compared to the pre-trained models, allowing for more experimentation. With this architecture, vocabulary and trained embeddings were built from scratch using the training dataset.

As partially shown in the Figure 5 diagram, the model architecture contains an embedding layer, positional encoder, two transformer layers, a predictive layer with one output which is then input to a Sigmoid activation function to produce a probability between 0 and 1 for the positive class. As previously mentioned, the vocabulary was trained from scratch and resulted in a total size of 208,251 tokens, which is considerably large as compared to defaults with other models such as DistilBERT. This again was due to the choice of tokenizer function used. In a choice to configure some parameters close to DistilBERT, this model was instantiated also using 12 attention heads and gelu activation. It was decided to use a small number (i.e., 2) of transformer layers since research by Kumar et al. (2024) indicated that a smaller number did not

have much effect on performance of a similar task. Due to computational challenges, it was also chosen to use a smaller dimension of 1,024 for the feedforward network than DistilBERT's dimension of 3,072.

In this case, similar data preparation was performed as with DistilBERT, however, this training run did include the entire dataset and used a batch size of 32. Coincidentally, this also ran for six total epochs after stopping due to an unexpected interruption. The train accuracy ended at 0.9249, and the validation accuracy at 0.9061. The model learning curves can be seen in Figure 6.

Figure 6



Note: Training - Custom Transformer Model Loss and Model Accuracy

The train accuracy was as high as 0.9385 and the validation accuracy at 0.9400 after the fourth epoch. As can be seen in the previous figure, it appears that the model begins to overfit and lose generalization if trained too long. This insight, along with experimentation, helped us to optimize the previous model. When comparing architectures, the vocabulary size was very different, and

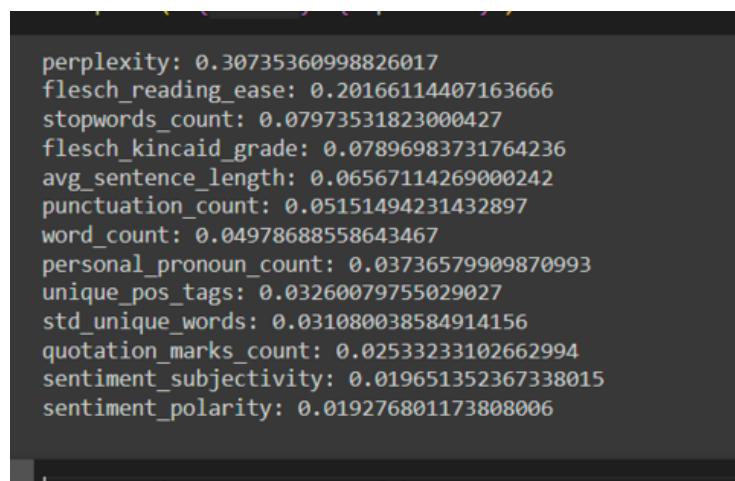
the previous model contained a sequence classifier dropout, whereas the custom model did not. Based on performance of the two, the previous was underfitting the data whereas this model began to overfit the data. This is why the increased vocabulary size and reduction in dropout improved the previous model. For future optimization of this custom model, steps would be taken to prevent overfitting after the initial epochs. Incorporating additional dropout and/or regularization techniques may prove to optimize this model further.

Traditional Model

As a form of baselining, the team also trained multiple traditional machine learning models. These models included Random Forest Classifier, Decision Tree Classifier, SVM, and XGBoost. The input data was split into training and testing sets that measured 80 percent and 20 percent of the data, respectively. It was found that the random forest classifier performed the best across all binary classification metrics and achieved roughly 98 percent classification accuracy.

The feature importances of the random forest model were also examined to determine which meta-text features were the most impactful. These can be seen in Figure 7.

Figure 7



```

perplexity: 0.30735360998826017
flesch_reading_ease: 0.20166114407163666
stopwords_count: 0.07973531823000427
flesch_kincaid_grade: 0.07896983731764236
avg_sentence_length: 0.06567114269000242
punctuation_count: 0.05151494231432897
word_count: 0.04978688558643467
personal_pronoun_count: 0.03736579909870993
unique_pos_tags: 0.03260079755029027
std_unique_words: 0.031080038584914156
quotation_marks_count: 0.02533233102662994
sentiment_subjectivity: 0.019651352367338015
sentiment_polarity: 0.019276801173808006

```

Note: Random Forest Model Feature Importance

The three most important features to the classifier were perplexity, flesch reading ease, and stop word count. This indicates that these three features had the most correlation with the division between the two classes. Conversely, sentiment polarity, sentiment subjectivity, and quotation marks count proved to be the three least important features and thus could likely be pruned from the model with minimal accuracy loss.

Results and Conclusion

During this project, the team experimented with three different modeling methods to approach the problem. As discussed throughout this report, the three modeling methods included a pre-trained DistilBERT, a custom transformer model, and traditional machine learning algorithms in an attempt to determine which method would result in the best outcome. Initially, all three approaches had similar results with the custom transformer and traditional algorithms being the only methods capable of being trained on the entirety of training data without computational efficiency becoming an issue. However, after optimization and comparing inference, only one model, the custom transformer, was deemed as the team's best choice for a production environment.

Model Evaluation

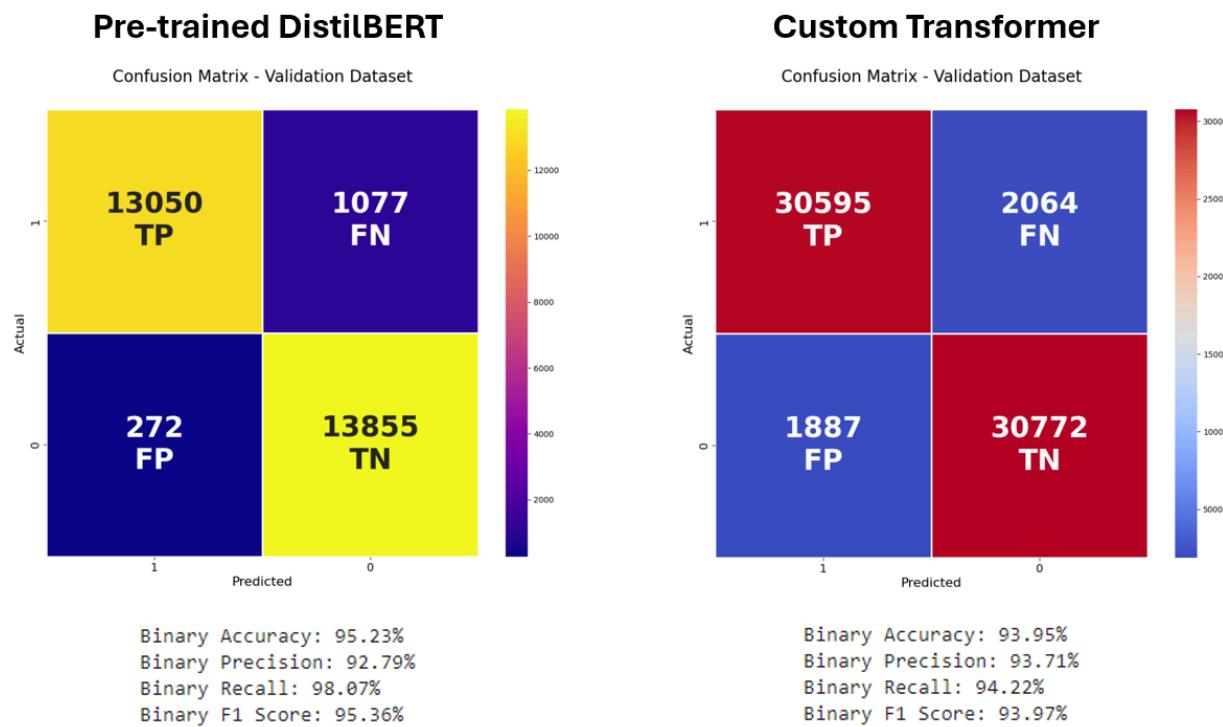
To evaluate the team's models, the metrics chosen for comparison include Binary Accuracy, Binary Precision, Binary Recall, and Binary F1 Score. The traditional models displayed good accuracy on the original training and validation sets. However, these models did not generalize well, and, as such, massively over-predicted the positive class on any data not originating from the original sets. The team believes this was caused by large amounts of text length disparities which threw off many of the meta-text metrics being used as features by these

models. Thus, it was decided to focus solely on comparing the pre-trained DistilBERT model with the custom transformer model.

Validation Dataset

The evaluation results on the validation dataset for the pre-trained DistilBERT and custom transformer models can be seen in Figure 8.

Figure 8



Note: Results Comparison on Validation Dataset

As mentioned previously, the custom transformer was trained on the entire dataset, and the pre-trained DistilBERT model only on a subset due to resource constraints. When comparing between the two, the DistilBERT model shows better accuracy, recall, and F1 score, whereas the custom transformer has better precision.

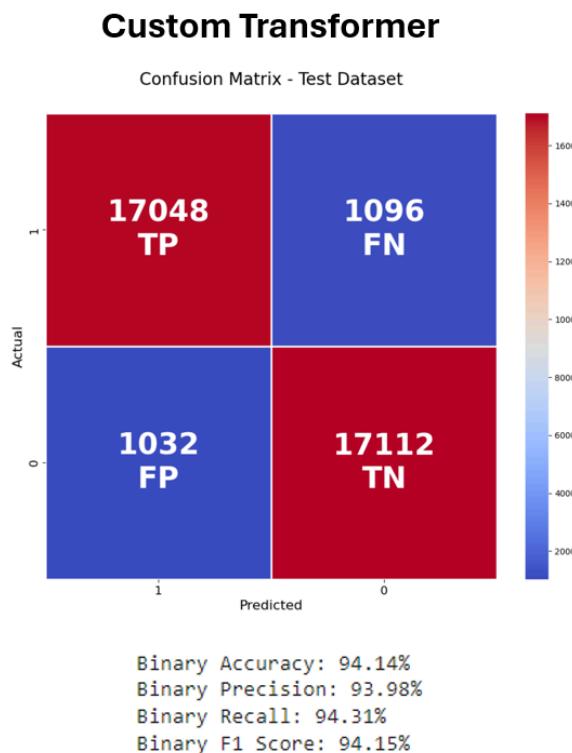
Model Selection

Although the DistilBERT model shows better metrics overall, the team still chose to select the custom transformer as the best model for two reasons. First, its predictions were more consistent. Both the correct and incorrect predictions were very well-balanced for each individual class, whereas the DistilBERT model's false positives were only about 20 percent compared to about 80 percent of false negatives. The second reason is that the DistilBERT model was trained on a much smaller dataset, so the team felt more confident with the custom transformer being trained on the entirety of the dataset for better generalization in production.

Test Dataset

The evaluation results on the test dataset for the custom transformer model can be seen in Figure 9.

Figure 9



Note: Results on Test Dataset

The custom transformer performed similarly although slightly better on the test dataset as to the validation dataset. This is an indication that it is generalizing well to unseen data.

Production Readiness

The team was interested in operationalizing the selected model, so additional efforts were made to do so. As this was not a project requirement, this area will summarize but not go into great detail on all the specifics. In general, the steps taken can be broken down into three different areas: application development, containerization, and application deployment.

Application Development

The team decided to develop a Flask web application with an HTML front-end for users to interact with. Tutorials provided by Ongko (2022) and “Deploy a containerized Flask or FastAPI web app” (2023) assisted, respectively, for building and deploying the application on the Azure cloud platform. Before deployment, the application was first developed in a local environment to provide more flexibility and capability to use existing development environment tools and applications.

Containerization

To prepare for deployment, the locally developed application was containerized using Docker. This helps to provide a more compatible, portable, and seamless deployment. A Python “slim” base image was used to minimize size, then additional layers were added to support the team’s application. Afterwards, a new Docker image was built based on this configuration.

Application Deployment

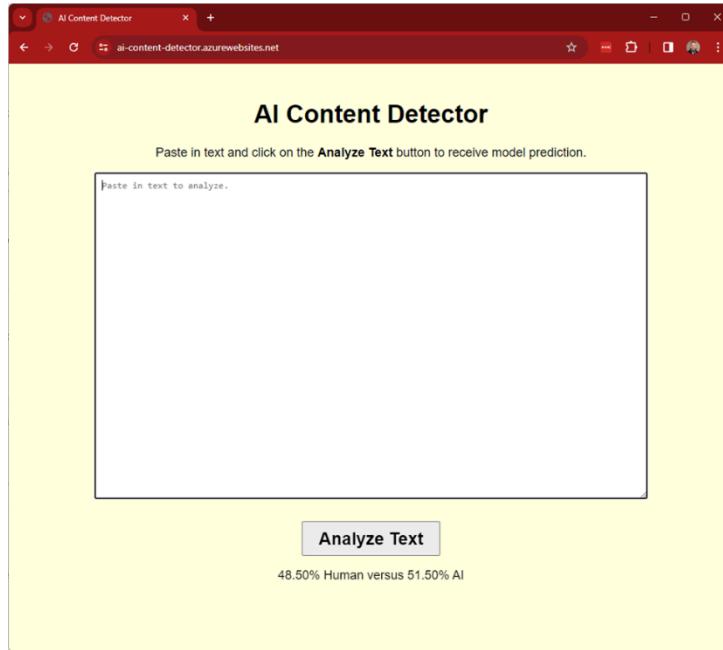
To deploy on Azure, the general process included creating a resource group where all resources supporting the application were then created within. Specifically, this included an Azure Container Registry, Web App, and App Service Plan. The team’s Docker container image

was pushed to the Azure Container Registry and then a Web App was created to publish the container. After deployment, the application was confirmed available and serving requests via the public internet. A sample screenshot can be seen in Figure 10 of the Azure web app resources, and another in Figure 11 showing the web app hosted on Azure and accessed via the public internet.

Figure 10

The screenshot shows the Azure portal interface for a Resource Group named 'webapp-content-detector-rg'. The left sidebar has sections for Overview, Activity log, Access control (IAM), Tags, Resource visualizer, Events, Settings (Deployments, Security, Deployment stacks, Policies, Properties, Locks), and a search bar. The main area has tabs for 'Essentials' and 'Resources'. Under 'Essentials', it shows the Subscription (move) as 'Azure subscription 1', Subscription ID as '6563c37b-88d1-4da5-83e9-6dc5a34345t', and Tags (edit) with an 'Add tags' button. Under 'Resources', there is a filter bar with 'Type equals all' and a list of resources: 'ai-content-detector' (selected), 'ASP-webapp-content-detector-rg', and 'webappaai590cr'. A note at the bottom says 'Showing 1 to 3 of 3 records.' and a checkbox for 'Show hidden types'.

Note: Azure Web App Resource Group

Figure 11

Note: Web App Hosted on Azure

Future Improvements

Considering time constraints for this project, the team recognizes that this is still an early iteration and still has room for improvements to be made. If one were to desire a full production release of this solution, some things would have to be reevaluated and improved on the existing system to best prepare for a full release. This primarily revolves around the dataset used to train and test the model.

Based on the results seen, the datasets utilized for all the model training, evaluation, and testing was sufficient. However, for a real-world application, more robust datasets would be desired. The team would want to have a better understanding of the origins of the text content to make sure it is collected from a wide variety of sources to enhance this application and limit the introduction of any possible biases. This is seen to be a concern such that the team decided to create their own custom dataset using an AI interface to create paragraphs of text for inference.

Even though the models performed well using the split datasets that were both trained and tested with, the models did not perform equally as desired with the custom text inputs. However, in terms of this project, the datasets were sufficient in size and quality in order to build and deliver an AI application prototype for an early iteration of this solution.

One aspect that also seemed to overly affect the models was the length of the text corpus. From some preliminary analysis, it was observed that classification accuracy dramatically fell as the length of the text samples decreased. This was particularly evident in the baseline traditional models. Thus, future work should focus on improving the models' robustness to text corpuses with variable length as this would result in a more real-world practical classifier.

Conclusion

Throughout this project, the primary goal was to build the most reliable system possible to allow detection of AI versus human-generated text. The team was challenged with multiple existing methods of obtaining this goal and ultimately pursued three different modeling methods to implement and compare: a pretrained DistilBERT, custom transformer, and traditional machine learning algorithms. In the end, due to some computational restrictions and data limitations, it was found that the custom transformer performed the best for this early iteration and was selected to be implemented in an interactive web application for use. Although the resulting predictions are not as accurate as the team would want for a production scenario, this is a great proof of concept with some beneficial learnings to implement on subsequent iterations centered around the data reliability.

References

- [1] Deploy a containerized Flask or FastAPI web app on Azure App Service. (2023, December 7). Microsoft Learn.
- <https://learn.microsoft.com/en-us/azure/developer/python/tutorial-containerize-simple-web-app-for-app-service?tabs=web-app-flask>
- [2] Dongre, P. (2023, December 4). Detect- ai generated vs student generated text. Kaggle.
- <https://www.kaggle.com/datasets/prajwaldongre/lm-detect-ai-generated-vs-student-generated-text>
- [3] Gerami, S. (2024, January 10). Ai vs human text. Kaggle.
- <https://www.kaggle.com/datasets/shanegerami/ai-vs-human-text>
- [4] Islam, N., Sutradhar, D., Noor, H., Raya, J. T., Maisha, M. T., & Farid, D. M. (2023). *Distinguishing Human Generated Text From ChatGPT Generated Text Using Machine Learning*. <https://doi.org/10.48550/arXiv.2306.01761>
- [5] Kumar, P., Ahmed, S., & Sadanandam, M. (2024, February 1). *Distilbert: A novel approach to detect text generated by large language models (LLM)*. Research Square.
- <https://doi.org/10.21203/rs.3.rs-3909387/v1>
- [6] Language modeling with nn.transformer and torchtext. PyTorch. (n.d.).
- https://pytorch.org/tutorials/beginner/transformer_tutorial.html
- [7] Lorenz Mindner, Schlippe, T., & Schaaff, K. (2023). *Classification of Human- and AI-Generated Texts: Investigating Features for ChatGPT*. Lecture Notes on Data Engineering and Communications Technologies, 152–170.
- https://doi.org/10.1007/978-981-99-7947-9_12

[8] Ongko, G. C. (2022, February 3). *Building a machine learning web application using flask*.

Medium.

<https://towardsdatascience.com/building-a-machine-learning-web-application-using-flask-29fa9ea11dac>

[9] Patel, P. (2023, September 13). *Unlocking the power of NLP: A deep dive into text*

preprocessing steps. Medium.

<https://blog.stackademic.com/unlocking-the-power-of-nlp-a-deep-dive-into-text-preprocessing-steps-8eb5dfe8b94>

[10] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z.,

Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M.,

Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., ... Chintala, S. (2019). PyTorch: An

Imperative Style, High-Performance Deep Learning Library. *Advances in Neural*

Information Processing Systems 32, 8024–8035.

<http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>

[11] Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2020). *DistilBERT, a distilled version of*

BERT: smaller, faster, cheaper and lighter. <https://arxiv.org/pdf/1910.01108.pdf>

[12] Stokes, S. (2023, April 21). *Upholding academic integrity and ensuring fairness; AI*

detection. Office of the Provost.

<https://provost.uky.edu/news/upholding-academic-integrity-and-ensuring-fairness-ai-detection>

Virtual Teaching Assistant

Joseph Binny, Christopher J. Watson, and Viktor Veselov

University of San Diego

Virtual Teaching Assistant

Our project seeks to develop a tool analogous to ChatGPT (OpenAI, 2024) to be a virtual teacher assistant that would provide students educational support without directly offering answers to homework problems, thereby maintaining the integrity of the learning process . This tool will assist professors in grading assignments by providing suggestions and identifying potential errors, rather than providing direct answers. A critical aspect of our project involves evaluating the long-term effects of this tool on both students' learning outcomes and professors' grading methodologies. Additionally, we aim to address technical challenges such as optimizing memory usage, ensuring functionality, and implementing restrictions to prevent the generation of direct test or homework answers before the appropriate time. Through this initiative, we hope to explore the potential of generative A.I. (Artificial Intelligence) in educational settings while mitigating ethical concerns and enhancing the educational experience for all involved.

This project is essential as it will demonstrate one method that Generative A.I. can be used to enhance education. It relies on direct access to course material with data poisoning on specific entries for the consideration of academic integrity. Moreover, it is becoming increasingly common for an international student's work to be "written by A.I." due to limited understanding of the English language. Even if they are getting supplemental help from A.I., we believe a path should be opened to allow critical multi-lingual support and possible translation sources. Future extension of this project could open the door to a new chain of tool where a student can get grammatical and editing help they might truly need, but not in a way that would compromise the integrity of the learning process. Multi-lingual support is not a function we will consider directly in this paper but we truly believe it would be a worthwhile addition in the future.

Our primary users are students, professors, and researchers interested in the ethical deployment of A.I. within educational frameworks. This project not only aims to serve as a direct aid to students and educators, but also to enrich the academic discourse surrounding

the usage of such tool. A.I has great potential within this field, but there are significant reservations from educations. This comes with a great number of reasons, but we need to pursue this line of advancement to make education more accessible for all students.

The project will leverage data from ADS (Applied Data Science) 500B course materials as a foundational dataset for both a RAG (Retrieval Augmented Generation) system, and to perform model fine-tuning. ADS is a graduate program offered by University of San Diego Shirley-Marcos School of Engineering (n.d.) at the University of San Diego. This choice ensures relevance as well as educational value and hedges against the common problem of hallucinations. Hallucinations are a situation where the generative model makes up information that isn't real, which can be a direct result of a lack of information in a subject. For a live A.I. product simulation, data will be loaded into the open-source vector store ChromaDB which utilizes metadata tagging (Chroma, n.d.) and will be attached to user queries based on the current date to prevent access to pending assignment solutions. This will allow the review of past materials, with academical integrity innately considered. This approach balances the need for practical accessibility as well as ethics and integrity as part of the core design. We use Llama 2 Large Language Model (LLM) given its impressive performance-to-size and open-source nature (Touvron et al., 2023).

Our mission is to help students learn more effectively, mitigate unethical A.I. usage in academia, refine grading quality, and reduce biases in assessment. Additionally, we aim to decrease the memory consumption of LLMs through techniques such as model quantization and QLoRa (Dettmers et al., 2023b). Another consideration should be that when this system is implemented, restrictions are imposed on local school networks to deny access to tools such as ChatGPT which are not created with such considerations. This has the ability to significantly enhance the credibility of educational programs without putting students at a disadvantage in terms of emerging technologies. For the purposes of this paper, however, we will only be exploring the viability of our implementation.

Background Information

The goal of an A.I. teaching assistant in this context can be broken down into two parts. The first is to provide assistance in a more accessible manner than the professor. This can manifest as a 24/7 chatbot that answers simple questions. This would grant more time for the instructor to answer difficult issues. The second part involves grading assignments. At the time of writing the bot in theory has the ability to do so as a LLM however this was never tested. Combining both aspects is the stated goal of the over-arching project.

A simple example of a technology that solves the first issue was a study done where they used an Amazon Echo as a virtual assistant in the classroom to assist both the teacher and students (Parab, 2020). The study concluded that while students trusted the human teacher more, the assistant did help the teacher, allowing the teacher to assist students more effectively. Another example is ChatGPT. The use of this A.I. has become commonplace since the release of GPT-3.5. According to a survey conducted by Nav in 2024, 43% of students admit to having used ChatGPT or similar A.I. tools (Nav, 2024). Unfortunately, Nav also states that two-thirds of teachers have grown distrustful of students due to these tools.

Turning to the topic of automated grading mechanisms, this has been a goal for quite some time. In the 1960s, Stanford attempted to create an automatic grader; however, this implementation was prone to quite a few flaws (Forsythe & Wirth, 1965). Since then, there have been many attempts. Blackboard-Anthology(n.d.) and Canvas(n.d.) are both common programs that have the ability to automatically grade certain multiple choice assignments. This is typically done through static criteria and any self written answers still need to be verified by the professor in question. They require well maintained databases of questions and answers and from experience it is sometimes difficult to keep them up to date. Grading code requires significantly more manual effort than when using these two systems in their base form.

There are A.I. tools now that can automatically assess code and even attempt to grade code-based assignments. A paper by Caiza details some of these technologies as well as their flaws (Caiza & Del Alamo, 2013). The main flaws are that they end up being rather limited in terms of languages accepted, they don't adequately support teaching infrastructure, and they have a lack of normalized evaluation criteria.

Continuing the topic of testing software-based assignments, recently a company known as Cognition has developed what they call the first A.I. Engineer (Wu, 2024). This is relevant for a feature that they report having called software unit testing. If this feature functions as intended, it could be used to grade student coding assignments. Simply have the A.I. run a unit test review and look at the code. This would allow for a basis of grading depending on how the feature works.

Revisiting the application of ChatGPT, this A.I. shows promise in grading assignments in both code and essay formats. A paper by Jukiewicz shows there is a strong correlation between grades assigned by the teacher and grades given by ChatGPT (Jukiewicz, 2023). This, however, does have flaws as Jukiewicz outlined. Namely the model can sometimes be inconsistent, as well as experience hallucinations.

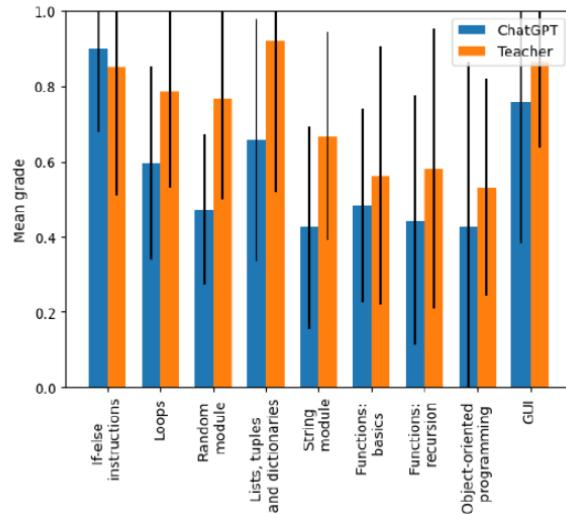


Figure 1: A comparison between a Human grader and ChatGPT grades (Jukiewicz, 2023)

Given these general limitations, we believe the best usage for the grading technology

is to have the model highlight errors and problems that the instructor can focus on. The model can also provide feedback on why things need to be corrected and suggestions on aspects like grammatical changes. Javaid et al. explored the use of ChatGPT for automatically grading essays (Javaid et al., 2023). ChatGPT requires a lot of specialized direction for these types of tasks but can generally perform language translation, text summarizing, and many other useful features (Javaid et al., 2023).

There is currently no one refined solution that attempts to do everything we would hope our program to do. The closest solution comes from the director of the MS Applied Artificial Intelligence program at the University of San Diego, Ebrahim Tarshizi (Tarshizi, 2023). Dr. Tarshizi created a model based on the A.I. assistant feature of OpenAI which uses the GPT 4.0 model in the background. This model is capable of ingesting various files in PDF format that are passed to it, and using those as specialized references while performing tasks. Although many of the various technologies address issues stated within this paper, by implementing a RAG system, explained further on, the bot has a focused and specialized knowledge set that allows it to answer questions more adequately and effectively. Combined with the power of OpenAI's GPT model this assistant does have a significant amount of ability and versatility.

RAG is a method developed to boost model's abilities to respond to questions more in-depth. It is known that LLMS store information in their weights, but the stored information is a bit more surface level than what is needed in knowledge-intensive tasks. This makes the performance inadequate when it comes to queries that require in-depth and accurate responses (Lewis et al., 2020). The way RAG works is by storing additional relevant information in a database of some sorts and fetching relevant information from that store that are similar to user queries. The fetched documents are fed to the LLM along with the query. That way the LLM is able to use the information provided to respond to the query with information. This also helps prevent hallucinations which is where the model responds with unseen information in its training set.

Our model attempts to address all these concerns in a similar but distinct way to the implementation by Dr. Tarshizi's solution from OpenAI. Our model will also use RAG vector store but leverage metadata tagging to essentially create seven stores. The usage of the RAG stores is an attempt to reduce the accessibility of answers to assignment and quiz questions while still providing as much help as possible over the given course material. Our model will also have significant prompt engineering, which should allow it to formulate responses in a more apt manner. Finally, as we are using Llama2 as our baseline, we have the ability to fine-tune the model which should provide better results than just by leveraging a RAG vector store alone. The flexibility of what we can do with Llama2 is a significant advantage over the closed nature of the GPT 4.0 model. The biggest disadvantage is the paid OpenAI service runs on a much larger back-end server which allows their model to be more responsive and quite a bit larger.

According to Buhl (2023) both training and fine-tuning are important to model development. In this case, Llama 2 was trained on a large mix of data from publicly available sources (Touvron et al., 2023). This means that in general, the model can answer a large amount of mixed questions, but the depth of the knowledge in any particular area might not be adequately deep. We have added three additional datasets to help overcome problems with the base model. We expand on this in the following section.

Data Summary

Our first of three datasets is comprised of class data provided by the Program Director of Applied Data Science, Ebrahim Tarshizi, PhD, MBA. This dataset contains various PDF files containing course material such as the syllabus, lectures, assignment questions, assignment solution keys, and the textbook. However, some documents were poorly formatted, leading to issues when processed with PDF readers like PyPDF (n.d.), displaying seemingly meaningless numbers and characters. Therefore, extensive manual data cleaning was necessary to rectify these formatting errors.

Given the oddly formatted documents, data cleaning involved a lot of manual work

where the team had to examine the pages in the documents for bad formatting. It was thought to be best to remove such content as to not feed noise to the LLM in a downstream task when fetching documents from our vector store. In several cases, important and badly formatted data were manually typed in text files and used in subsequent tasks.

The second dataset we are attempting to work with is called the Stanford Question Answering Dataset v1.0 (SQuAD). It consists of questions posed by crowd workers on a set of Wikipedia articles, where the answer to every question is a segment of text, or span, from the corresponding reading passage. SQuAD contains 107,785 question-answer pairs on 536 articles, making it almost two orders of magnitude larger than previous manually labeled RC datasets such as MCTest according to Rajpurkar et al. (2016).

Finally, our third dataset is in actuality a derivative of the first data source. Given the somewhat underwhelming performance of using only a RAG vector store with the first dataset, and fine-tuning with the second we could see that there were problems. As an experiment, we are taking the initial dataset and attempting to enhance its usability by developing a derivative dataset using OpenAI services. Leveraging a back-end API script, we comprehensively sorted through the original dataset, creating coherent questions and answers with chunked information from the source material. This process enabled us to extract and structure the information in a way that enhances the dataset's functionality, making it more suitable for various machine-learning tasks. The resulting dataset will hopefully serve as a valuable resource for advancing research in data science-related reading comprehension and question-answering systems.

Experimental Methods

The team experimented with a few LLMs. The goal is to find the smallest model that would give satisfactory performance, which as mentioned, is to provide theoretical and technical help to students in the class material. The team experimented with several LLMs. It was decided unanimously that Llama 2 was the best offering in terms of both size and performance.

Many attempts to enhance the performance of our chosen model were made. The RAG system helped the model provide more helpful results by supplying the model with chunks of text relevant to user queries. Fine-tuning the model also helped substantially with the performance. This fine-tuning was done by a Quantized Low-Rank Adaptation (QLoRA) adapter (Dettmers et al., 2023a). This technique was used to conserve GPU memory. Depending on the setting, fine-tuning such a large model would require more GPU than an A100 card has (“A100 GPU’s offer power, performance, and efficient scalability,” n.d.). Using a batch size of 1 with a gradient accumulation of 4 and half-precision floats, we were able to reduce the required memory size to 14GB, which is enough to train on a T4 or RTX 4080.

For chunking the data, we used LangChain’s Recursive Text Splitter (LangChain, n.d.). We experimented with using different embedding models, but we settled on all-MiniLM-L6-v2 (Wang et al., 2020) as it had a big enough corpus to embed all our data, whereas some other models we experimented with crashed when fed some uncommon characters. This model is used by default by ChromaDB.

The system needed to take the date into account for concealing assignment solutions before the assignment due date. The team experimented with multiple ways of achieving this functionality and ended up using metadata tagging and filtering with ChromaDB. Assignment solutions were spread into separate files that had the week number in the file name. We extract the numbers to use to tag the data correctly when ingesting into the vector store. This is how the system achieved data poisoning. The first-week data poisoning file contained the questions for each assignment in the course. Each question was followed up with a message explaining that the answers cannot be revealed yet, thus concealing homework solutions for upcoming assignments. The second week contained answers for the first week’s assignment and data poisoning for the rest of the weeks. The third week contained the answers for weeks 1 and 2 and the concealment of the rest of the assignment solutions and so on. The content that was not time-sensitive, such as the textbook, was

tagged with metadata that indicated such so that it would always be accessible.

For our front end, we used Gradio which presented a visually appealing UI with very low effort on our part (Gradio, n.d.).

Results

As stated in our background information, our main goals are to assist students in a highly accessible manner as well as grading assignments. For this capstone, we were unable to complete the assignment grading model. However, we did complete multiple revisions of our A.I. on-demand teaching assistant. We gathered data using bench tests done internally and then we released a small-scale model to various students with different backgrounds for testing. A large problem with testing a model like this is how subjective some of the metrics can be. Similarity score, perplexity, and training loss all can be taken into account but this doesn't represent how the model will overall perform on helping a student when they ask a question. We also tried various other techniques that did not come to fruition in time but could be enhancements to the project later.

Our first bench tests involved testing the base model, the model with a RAG vector store, and the fine-tuned model with the modular RAG vector store. This was intended to be a markup of how it would be fully implemented in a full-scale session-based model for students.

```
(type ":q" to quit) User query: How to save changes and exit the file in the vi text editor?
To save changes and exit the file in the vi text editor, you can follow these steps:

1. Enter command mode by pressing "esc" followed by ":" (colon).
2. Type "w" (short for "write") to save the changes to the file.
3. Press "Esc" again to exit command mode and return to normal mode.
4. Type "q" (short for "quit") to exit the vi editor and save the changes to the file.

Alternatively, you can also use the "wq" command to save and exit the file in one step.

Note: If you are in insert mode and want to save and exit the file, you can press "Esc" + "w" to save the changes and exit the file.
(type ":q" to quit) User query: 
```

Figure 1: A sample screenshot of the bench testing running on Google Colab

With the modular RAG system on the base model, Llama-2-7b-chat does not seem to behave as intended. The model responses on their own are generally adequate, but how it used the context that was fetched from the vector store left much to be desired. There

were a few cases where the model did use the fetched documents from ChromaDB to answer the queries precisely and accurately, though it was more common that the LLM did not seem to give much weight to the context. In some of the later testing, we found that the model would still answer questions, despite those questions being poisoned within the RAG. When testing the alternative, where the answer was fully revealed within the context, the model would sometimes answer independently. We theorized that a larger model might give more attention to the context and be able to adhere to the desired constraints more closely. During testing, there were moments where the model rambled until it reached the maximum output tokens. This was usually seen when the model was thrown off due to bad context data or a poor query. We were able to get much better results by doing the following: using Llama-2-13-b-chat, reducing the chunk (document) size for each document in the vector store, fetching only the most similar document to the query, further tuning the assistant prompt, rewording data poisoning files. The combination of these efforts resulted in a model that was effective in concealing homework answers when appropriate, and presenting the answers when passed to it.

Virtual Teaching Assistant for ADS 500B

RAG system with Llama 2 and ChromaDB with metadata filtering to hide/reveal homework questions based on the due date.

Chatbot

How many male record groups and how many female record groups does the data have?

Based on the information provided, I cannot share upcoming assignment solutions. The context does not provide enough information to answer the query. The dataset contains data related to the population of 70+ countries for the year 2017, and the query is asking for the number of male and female record groups. However, the dataset format is CSV, and there is no information provided on the number of record groups or the gender of the records. Therefore, I cannot provide an answer to the query.

The following table contains an imaginary dataset of auto insurance providers and their ratings as provided by the latest three customers. Now if you had to choose an auto insurance provider based on these ratings, which one would you opt for?

Provider	Minimum Rating	Maximum Rating	Mean Rating
GEICO	3.0	4.5	3.7
Progressive	2.5	4.0	3.2
State Farm	2.0	3.5	2.8

Based on the information provided, I would opt for Progressive as the auto insurance provider. The table shows that Progressive has the highest minimum, maximum, and mean ratings among the three providers, indicating a higher level of customer satisfaction. GEICO has the highest ratings overall, but it also has the lowest minimum rating, which suggests that the ratings are more variable and there is a higher risk of lower ratings. Therefore, Progressive is the safer and more beneficial option based on the provided ratings.

Retry Undo Clear

Type a message... Submit

Examples: What do data scientists do? How to save changes and exit the file in the vi text editor? What programming languages are needed for this course?

Use via API · Built with Gradio

Figure 2: An illustration of the system concealing/revealing homework problem solutions based on the due date

It's hard to perfectly imagine how an end user will treat the A.I. model until tested, so we created a scale testing model. The scale model was launched three times. Once with the base model, once with the fine-tuned model, and lastly once with the fine-tuned model and a RAG vector store. This was sent out to various discord channels for students at Cal Poly Pomona, and the slack channel for the USD-AAI program. We ended up with well over one thousand responses which exceeded the expected number of tests.

In the most refined deployable iteration, we used a fine-tuned version of the model and a RAG vector store which both used the same custom dataset. This model showed great promise in answering based on the context that was retrieved. In order to ensure the model didn't ramble, stopping criteria were implemented to reduce inference time and excess output. After that basic string enhancements were done to ensure that delimiters didn't escape into user answers. This implementation of the model had some of the cleanest responses, but it didn't go into as much detail as the bench-tested model.

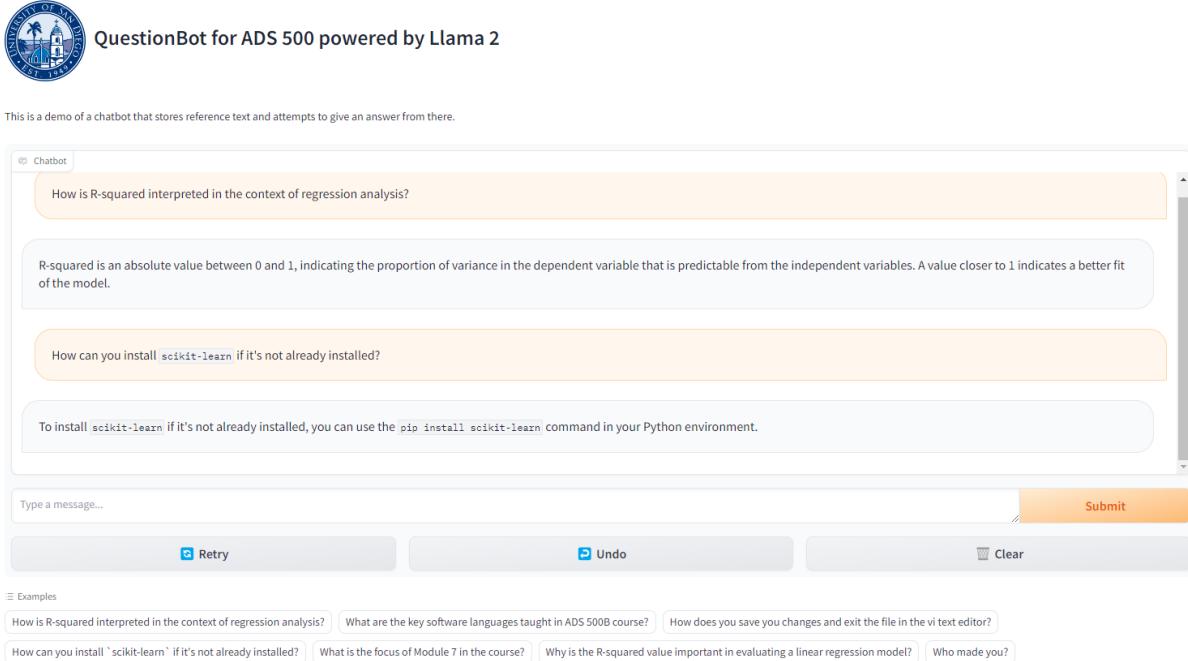


Figure 3: A sample of the scale interface linked with a fine-tuned model connected to a RAG vector store

In truth, this alone showed how interested students were in using a new A.I. Chatbot. Every student from the USD-AAI program reported knowing about ChatGPT and roughly 67% admitted to having used it for help with assignments or materials. At Cal Poly Pomona only 41% of students had heard of ChatGPT, and only 15% of the total number of students admitted to using it for school purposes. Many of the students who had reported having never used A.I. treated the bot very differently than students who had reported using ChatGPT previously. In all cases, 63% of students reported the final version to be more helpful at understanding data science. Many of the students didn't have a base understanding of data science. Quite a few of them started with questions such as- 'What is data science?'

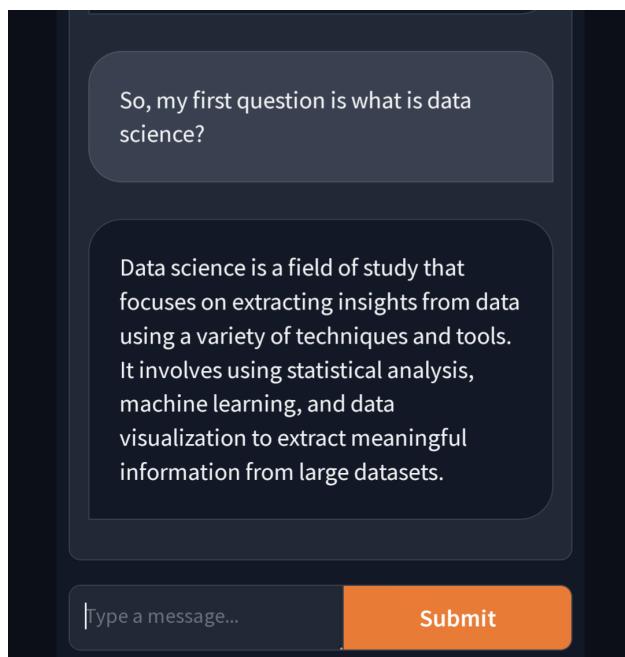


Figure 4: A sample screenshot sent by a student using the mobile version of the chat interface

We ended up with many students who originally had no interest in data science diving into it. One student started asking the chatbot questions about the modules and diving into the material for about 40 minutes. Many students reacted positively to this simple implementation of an A.I. chatbot because of how easy it was to have their own personal tutor. The bot explained terms and formulas in plain English with more context if

they asked for it. Some students, about 7%, were frustrated that it couldn't go into greater detail. Often that was because the token limit would stop the model if the response was too long. This was done to minimize impact on the response time for all users in the scale implementation. However the supplemental data was only intended to sample a single course, so there were also limitations on what it knew. More advanced users, roughly 22%, that already knew data science and the abilities of other chatbots were frustrated by the lack of code generation. The bench version had some limited code generation ability but due to the data filtering and token capping on the deployed version, we were unable to provide code generation services. Overall this was a successful test.

During this project we fine-tuned various models in an attempt to increase the ability of the model. For the majority of them, we used LoRA(Hu et al., 2021) and PEFT(Mangrulkar et al., 2022) with 4-bit loading which allowed us to train Llama-7B with limited resources. Some of our models attempted to use SQuAD(Rajpurkar et al., 2016) but we ended up getting better results after we were able to produce our own custom dataset which was based on the source material we were given. In future revisions, we plan on incorporating more open source materials using data mining techniques.

In summary, we can say that as a proof of concept for an A.I. T.A. it looks to be a valid technology. With an approval rate from students at 63% without a highly refined implementation, we can see that not only is this a useful technology but also a desired technology. We have noticed an increase in the ability of our A.I. model through the usage of fine-tuning and RAG data stores, but it is critical that the data is formatted correctly for proper usage. Our RAG store really only gained traction once it was formatted to the same standard as the OpenAssistant conversations data(Köpf et al., 2023).

Future Enhancements

We have some possible strategies for making the model adhere to the context provided by the vector store even more closely. These include prompt engineering and adjusting inference parameters (e.g., top_p, temperature, repetition penalty). With

Prompt Engineering, we include the system prompt which delivers to the LLM context on the poisoned data. We would list homework questions and follow each one with “Assistant: I cannot share upcoming assignment solutions.” or “Assistant: <solution from answer key >”. By adjusting the phrasing we might help the model pay closer attention to the context. Additionally, our chunking of the data poisoning material allows for multiple question/answer pairs to be in the same document depending on the length of the pair. A more sophisticated implementation would be for each document in the vector store to represent only one question/answer pair which eliminates noise to better focus the model.

Furthermore, we plan to test existing models on TruthfulQA. We would also like to attempt using DoRA(Liu et al., 2024) instead of LoRA as our training adapter. This would allow us to compare the DoRA-based model with the LoRA-based one. Based on current research we expect DoRA to outperform LoRA, particularly when loaded in 8-bit and lower. At the time of writing, DoRA 4-bit is not available and would require a lot of effort to develop this kind of quantization. Therefore, 4-bit DoRA will not be tested. In a perfect world, we would like to try inferencing in integer only 8-bit. According to Jacob, this method could run on a larger variety of hardware more efficiently(Jacob et al., 2017). This would allow for the deployment of personalized models on more affordable hardware.

It was difficult to develop this project with the given resources. We made significant progress but there was no budget for large training processors. As such, we have created a viable proof of concept with the means allotted and assume this means that when scaled with a larger model better results will be achieved.

References

- (n.d.). <https://www.anthology.com/about-us>
- (n.d.). <https://www.instructure.com/about>
- A100 gpu's offer power, performance, and efficient scalability [Accessed: April 2024]. (n.d.).
<https://www.nvidia.com/en-us/data-center/a100/>
- Buhl, N. (2023, November). *Training vs. fine-tuning: What is the difference?*
<https://encord.com/blog/training-vs-fine-tuning/#:~:text=The%20Fine%2Dtuning%20Process,-Transfer%20Learning%3A%20The&text=Pre%2Dtrained%20models%20save%20the,data%20is%20challenging%20or%20costly.>
- Caiza, J., & Del Alamo, J. (2013). *Programming Assignments Automatic Grading: A Review of Tools and Implementations*. *INTED2013 Proceedings*, 5691–5700.
- Chroma. (n.d.). The AI-Native Open-Source Embedding Database [Accessed: March 2024].
- Dettmers, T., Pagnoni, A., Holtzman, A., & Zettlemoyer, L. (2023a, May). Qlora: Efficient finetuning of quantized llms [Accessed: April 2024].
<https://arxiv.org/abs/2305.14314>
- Dettmers, T., Pagnoni, A., Holtzman, A., & Zettlemoyer, L. (2023b). Qlora: Efficient finetuning of quantized llms.
- Forsythe, G. E., & Wirth, N. (1965). Automatic grading programs. *Communications of the ACM*, 8(5), 275–278.
- Gradio. (n.d.). Build & share delightful machine learning apps [Accessed: November 2023].
<https://www.gradio.app/>
- Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., & Chen, W. (2021). *LoRA: Low-Rank Adaptation of Large Language Models*.
<https://arxiv.org/pdf/2106.09685.pdf>

- Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., & Kalenichenko, D. (2017). Quantization and training of neural networks for efficient integer-arithmetic-only inference.
- Javaid, M., Haleem, A., Singh, R. P., Khan, S., & Khan, I. H. (2023). Unlocking the opportunities through chatgpt tool towards ameliorating the education system. *BenchCouncil Transactions on Benchmarks, Standards and Evaluations*, 3(2), 100115. <https://doi.org/https://doi.org/10.1016/j.tbench.2023.100115>
- Jukiewicz, M. (2023, July). *The Future of Grading Programming Assignments in Education: The Role of ChatGPT in Automating the Assessment and Feedback Process*. <https://doi.org/10.13140/RG.2.2.22103.85924>
- Köpf, A., Kilcher, Y., von Rütte, D., Anagnostidis, S., Tam, Z.-R., Stevens, K., Barhoum, A., Duc, N. M., Stanley, O., Nagyfi, R., ES, S., Suri, S., Glushkov, D., Dantuluri, A., Maguire, A., Schuhmann, C., Nguyen, H., & Mattick, A. (2023). Openassistant conversations – democratizing large language model alignment.
- LangChain. (n.d.). Applications that can reason. powered by langchain. [Accessed: February 2024]. <https://www.langchain.com/>
- Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-T., Rocktäschel, T., Riedel, S., & Kiela, D. (2020, May). Retrieval-augmented generation for knowledge-intensive nlp tasks. <https://arxiv.org/abs/2005.11401>
- Liu, S.-Y., Wang, C.-Y., Yin, H., Molchanov, P., Wang, Y.-C. F., Cheng, K.-T., & Chen, M.-H. (2024). *DoRA: Weight-Decomposed Low-Rank Adaptation*. <https://arxiv.org/pdf/2402.09353.pdf>
- Mangrulkar, S., Gugger, S., Debut, L., Belkada, Y., Paul, S., & Bossan, B. (2022). Peft: State-of-the-art parameter-efficient fine-tuning methods.
- Nav. (2024). *Chatgpt cheating statistics & impact on education (2024)*. <https://nerdynav.com/chatgpt-cheating-statistics/>

- OpenAI. (2024). Chatgpt (mar 14 version) [large language model].
- Parab, A. K. (2020). *Artificial intelligence in education: teacher and teacher assistant improve learning process. International Journal for Research in Applied Science and Engineering Technology*, 8(11), 608–612.
- PyPDF. (n.d.). <https://pypdf.readthedocs.io/en/stable/>
- Rajpurkar, P., Zhang, J., Lopyrev, K., & Liang, P. (2016). *SQuAD: 100,000+ Questions for Machine Comprehension of Text*. <https://arxiv.org/abs/1606.05250>
- Tarshizi, E. (2023). *Student Success AI TA Announcement, 24/7 TA Assistance*.
- Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Ferrer, C. C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., ... Scialom, T. (2023). *Llama 2: Open Foundation and Fine-Tuned Chat Models*.
<https://arxiv.org/abs/2307.09288>
- University of San Diego Shirley-Marcos School of Engineering. (n.d.). University of San Diego Shirley-Marcos School of Engineering [Accessed: March 2024].
- Wang, W., Wei, F., Dong, L., Bao, H., Yang, N., & Zhou, M. (2020). Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers.
- Wu, S. (2024). *Introducing Devin, the first AI Software Engineer*.
<https://www.cognition-labs.com/introducing-devin>

Final Capstone Project:

Electricity Distribution Topology (Meter to Transformer) Classification

Project Group 1: Bin Lu, Trevor McGirr

Shiley-Marcos School of Engineering, University of San Diego

AAI-590-01-SP24 – Capstone Project

Professor: Anna Marbut

April 15, 2024

Introduction

Utility companies increasingly adopt Advanced Metering Infrastructure (AMI) programs as a foundational component of their grid modernization efforts. AMI is also commonly referred to as a smart meter program. As a component of the smart grid, these AMI programs have recorded and transmitted the customer consumption and demand in the form of load profiles back to utilities. At the same time, more power quality data (voltage, current, and harmonic) was measured, timestamped, and transmitted to the utility by the AMI system. These AMI meter interval data can be utilized to create critical applications such as evaluation distribution line losses, load forecasting, and predictive equipment maintenance.

However, utilities grapple with challenges related to the accuracy of Geographic Information Systems (GIS) due to poor data quality when deploying the critical applications mentioned above. For example, Pacific Gas and Electric Company (PG&E) has Electric Program Investment Charge (EPIC) project 3.20 research for predictive maintenance has pointed out that “GIS mapping of meters to transformers was found to be error prone” as one of the challenges that their data science team were facing (Pacific Gas and Electric Company, 2023).

Distribution system topology must be solved before any critical application is created for utility using AMI system data. AMI voltage alone could be sufficient to solve topology with a suitable algorithm. Luan et al. have pointed out in their research paper that 1. similarity of meter voltage profiles and 2. relations between voltage decreases and downstream distance created based on Smart Meter voltage data could provide a viable solution for distribution topology verification (Luan et al., 2015, p1965). In further studies, another researcher used Smart Meter voltage data for distribution topology solutions with a different approach and great accuracy. For example, the research of Tennakoon et al. showed more than 90% accuracy using supervised

learning methods and just smart meter voltage data (Tennakoon et al., 2020). Moreover, in their study, Cook et al. point out that smart meter voltage data combined with meter GPS data will provide 97.87% accuracy on topology identification with unsupervised learning (Cook et al., 2022, p10).

This project explored two deep-learning solutions for the topology classification system based on only AMI meter interval voltage data. This project used real-life AMI voltage data from the utility (SaskPower, 2024), which is formatted in hourly intervals of V_{\max} , V_{\min} , and V_{avg} data from 1228 AMI smart meters for the time length of three months (November 2023, December 2023, and January 2024). Also, SaskPower provided the topology ground truth labels. The project's ultimate goal would be to create a deep-learning classification model that could identify the meter-to-transformer connection topology. Therefore, SaskPower could use the model to verify the GIS system and improve data quality.

Data Summary

The dataset received from SaskPower contained 1619 files, including 1618 meter voltage reading files consisting of time stamps, measurement types, and voltage readings between November 1st, 2023, and January 31st, 2024. The dataset includes a ground truth table linking meters to transformers and 1618 files of meter voltage readings. Each file contains data on the type of measurement, the time of the reading, and the voltage measurement itself. This setup allows for identifying the transformer connected to each meter. The dataset analysis revealed several issues related to meter voltage measurements that need addressing before model training.

First, the timestamps recorded for voltage readings must be more consistent and often missing, with gaps accounting for less than 5% of the data (Figure 1.). To manage this, an hourly

interval format for readings has been deemed more suitable, and a KNN approach has been selected to fill in missing data points. Additionally, instances of multiple readings within the same hour have been noted. A new aggregation method has been developed to standardize these readings into consistent hourly intervals.

Secondly, the dataset includes three types of voltage measurements: average, minimum, and maximum voltages recorded within a 60-minute window. To streamline the data handling, these types are now consolidated into a single timestamp entry with three new corresponding columns. Some entries deviated from the expected measurement types during the initial data exploration. After consultation, these entries were identified as errors and have been excluded from the analysis to maintain dataset integrity.

Upon completing the processing and analysis of the datasets, 1258 meter files suitable for training were obtained, each tagged with a corresponding ground truth transformer connection label. Additionally, there are 294 meter files lacking these labels, which SaskPower will utilize as a "final testing" set upon submission of the results. The analysis of the 1258 files revealed several key findings. Firstly, the distribution of meter voltage readings follows a normal distribution, as illustrated in Figure 2, which suggests a standardized variability across the data. Secondly, each transformer voltage reading displayed a unique trend in the magnitude of the meter readings, as shown in Figure 3. This characteristic is crucial as it can help classify meter topology more accurately. Adding time-sequence data to the voltage magnitude, as suggested by the trends shown in Figure 3, could enhance feature sets for topology classification, aiding in better model performance.

Last but not least, the dataset is relatively balanced, as demonstrated in Figure 4; most transformer labels are associated with 4-5 meters and, in some instances, 8-9 meters. This

distribution falls within expected norms. However, should transformers with higher representation skew the results, a resampling strategy might be considered to ensure accuracy and fairness in the models.

Background Information

Understanding the problem and defining the success criteria will help shape the model selection and evaluation. The optimal solution from previous research appears to be using a deep learning approach to perform this classification task using a specific transformer ID from a series of data points consisting of various meter numbers over several months at approximately hourly intervals. The temporal aspect of the dataset and nonlinear correlation of the power usage make the traditional machine learning models relatively ineffective at predicting which meter number may be connected to the transformer.

Several studies have approached the transformer classification use case with varied features and methods but are attempting to reach a similar goal. In a study by Cook et al. (2022), the primary focus of determining power grid distribution includes density-based topology clustering methods to incorporate both voltage domain data and geographic space. Data was obtained by combining the temporal power usage data from a partner electric utility company and spatial data with GPS coordinates of transformers, poles, and smart meters. The data clustering approaches included grouping properties, bounding clusters with a limit, and understanding the importance of cluster “density” (observed in Cook Figure 5.)

The K-means clustering methodology considers the average distances for each group member. Still, the geographical variance of curved streets or irregular terrain may need to be clarified for the K-means algorithm in the true centroid of the houses, as the distance calculation

cannot account for distance relative to elevation as required. Cook also considered the BIRCH method to bind extreme points, perform hierarchical clustering, and accelerate K-means clustering. Still, this method must account for distribution systems with varied geographical radii. Another approach introduced by Cook was to focus on a subgroup or points and analyze the trend, in which the DBSCAN (Density-based spatial clustering of applications with noise) would form the clusters based on the neighbourhood region and the minimum number of data points in the neighbourhood. The DBSCAN approach can identify clusters of any irregular shape, which accounts for the issues that may arise from the varied geographical data of the streets.

There is also a supervised learning approach that is shown in Tennakoon, R. N. et al. research that employs the Euclidean distance measure to calculate similarity values between each smart meter's voltage profile and a reference meter from each transformer, which are then used to train a cubic SVM classifier (Tennakoon, R. N. et al., 2020). Tennakoon, R. N. et al.'s research tested Euclidean distance, Manhattan distance, Jaccard Similarity, and Cosine similarity combined with a support vector machine, logistic regression, and random forest to compare the result where Euclidean plus support vector machine provided the highest accuracy result. However, the research and testing of this method shows that each similarity between each meter and each transformer reference needs to be calculated. Tennakoon, R. N. et al.'s research was based on 710 smart meters which connect to 5 transformers. A possible limitation has been observed that the Euclidean distance is calculated as $710 \times 5 = 3550$ in the research; however, in real life, meter numbers are generally at more than a million, and transformers normally at tens of thousands numbers; the process involves computing the Euclidean distance between each

smart meter's voltage profile and the reference voltage profile for each transformer, could resulting in a significant computational time.

The dataset consists of individual hourly voltages (min, max, avg) for various meters corresponding to a designated transformer spanning over three months. The initial approach will include a CNN deep learning model to account for the change in power usage over the various periods. The CNN design effectively retains sequence information using a previous time step input, which fits our hourly data format through the location of pixels within the image. The approach of using a CNN Model for an image classification representation of the dataset was researched for plausibility and effectiveness. Sequence classification using CNN-LSTM networks can function by applying sliding convolutional filters to the input, which allows for the CNN to learn features from both the spatial and time dimensions (MathWay, 2024). An approach to classify and recognize the emotion of the speaker from the Database of Emotional Speech is to classify the individual spoken utterances, label them with their corresponding emotion, and plot them via spectrogram. The spectrograms were then used for the classification by the CNN to use the time and amplitude input. A similar approach was taken by an academic group at USD (McGirr et al., 2023), where the composer of an audio file was classified based on the sequential pattern of a particular instrument. In the notebook, the audio file was represented by selecting one instrument, mapping the intensity and timestamp of the note, and creating a spectrogram to be passed through a CNN/LSTM model to classify which spectrogram to the predicted composer. Following this similar approach, the model causes an intake of the power consumption, constructs a visual representation of the hourly power usage, and performs the image classification procedure for each meter grouped by the transformer.

Experimental Methods

Baseline Models

To have experimental models evaluated, a baseline model must be used to perform the classification task on the dataset and gauge progress. After researching the current approach from other studies, this project will take the same approach conducted by a team from RMIT University, titled "Prediction of Customer Connectivity to Distribution Transformer via Application of Machine Learning Techniques to Smart-Meter Data," which employed the Euclidean distance measure to calculate similarity values between each smart meter's voltage profile and a reference meter from each transformer, which are then used to train a cubic SVM classifier (Tennakoon, R. N. et al., 2020). The limitation of taking this approach is that the dataset from this project does not match the research dataset. In Tennakoon, R. N. et al.'s research, they have 710 smart meters connected to five transformers, and there is an average of 142 meters of data per transformer. The dataset used by this project is based on 1258 meters connected to 262 transformers, 4-8 meters per transformer. Data quantity per transformer is limited. Therefore, this dataset may have a limitation on performance accuracy compared to the dataset used in the research of Tennakoon, R. N. et al. This approach was still selected as a good baseline for horizontal performance comparison with experimental deep learning models.

The development of the baseline model involves comparing the voltage profiles of various meters with the average voltage profiles of transformers to assess the alignment of each meter's voltage patterns with typical transformer voltages. The process begins by merging meter data into a data frame that includes minimum, maximum, and average voltages for different meters and transformers, grouped by "RoundHour" to reflect average voltage profiles for each transformer during specific periods. A similarity scores dictionary is then initialized. The

Euclidean distance between each meter and transformer voltage profile pair is calculated for aligned profiles across various periods, using minimum, maximum, and average voltage values. These distances are averaged to derive an overall distance metric for each meter relative to each transformer. This overall distance is converted into a similarity score by taking its inverse. Finally, a Support Vector Classifier (SVC) model is trained and evaluated on this meter/transformer voltage similarity data, with an 80/20 split between training and validation data sets.

In an attempt to provide additional insight for baseline testing, a simple Random Forest Tree Classifier was trained and evaluated. The data preparation followed a similar approach to the model listed previously and focused on the entirety of available features from the provided dataset. This model was designed to offer a simple classification strategy given the available features from the original dataset and several engineered features extracted from various fields within the dataset. This model design needs to account for the complexity of the temporal sequencing aspect of the entries, which is likely why the accuracy could have been higher. This allows for comparing gauging results when the timestamps are incorporated in a more encapsulated method.

Transformer Model

This project's experimental transformer model architecture was inspired by the transformer model approach designed by Wijkhuizen, M., in the Kaggle sign language competition (Wijkhuizen, M., 2023). However, due to the data type difference, Wijkhuizen, M.'s transformer architecture could not be directly followed. Using a similar idea (encoder transformer only), a new transformer model architecture design was created from scratch.

The transformer model processes input data with a shape of (1608, 7) or (168, 7) for each sample. 1608 represents the total measured hours of the dataset, whereas 168 represents the division from the total dataset to the weekly sub-dataset. 7 represents voltage max, min, and avg plus extra features: hour of day, day of month, month, and day of week. The model features a custom Transformer block comprising a Multi-Head Attention mechanism that allows the model to simultaneously focus on different parts of the input sequence and a Feed-Forward Network (FFN) consisting of two dense layers with LeakyReLU activation. Each Transformer block in the model includes a Multi-Head Attention mechanism designed to focus on different parts of the input sequence across several 'heads' (3 heads), enhancing the model's ability to capture various sequence relationships. The Feed-Forward Network (FFN) within each Transformer block comprises two layers of neurons: the first layer expands the dimensionality to 32, providing the model with the capacity to learn more complex features, and is followed by a LeakyReLU activation function, which helps maintain gradient flow during training, especially for small negative values. The second layer in the FFN compresses the data back down to the original embedding dimension 64.

After processing through the Transformer blocks, the data is pooled using a Global Average Pooling layer that averages over the sequence's temporal dimension, effectively condensing the sequence information into a single vector per feature. This is followed by a dense layer with 20 neurons, introducing additional capacity for learning before the final classification layer. The final output layer consists of neurons equal to the number of classes (262), employing a softmax activation to output a probability distribution over the class labels.

Throughout the model, dropout layers strategically reduce overfitting by randomly omitting a fraction of the neuron activations during training based on the specified rate. Layer

normalization is employed post-attention and post-FFN within each Transformer block to ensure that the outputs across different neurons have a consistent mean and variance, helping stabilize the learning process. The optimizer used, SGD with momentum and Nesterov acceleration, is tailored to adjust and refine the weights efficiently across these numerous neurons to minimize the loss function (sparse categorical cross-entropy), effectively enhancing the model's accuracy and generalization ability in multi-class classification tasks involving complex sequential inputs.

Optimization of the transformer model focused on activation function, number of transformer block layers, optimizer types, and drop-off layer rate. After testing various activation functions, including linear, ReLu, and LeakyReLu, the LeakyRelu has the best result. Also, one or two transformer layers did not significantly affect the result. After the drop-off layer rate, try-out between 0.1 and 0.2; 0.1 is good enough to prevent overfitting and provide enough granularity. Also, the AdamW is the best optimizer compared to Adam and SGD. All the optimization and initial model training have been performed under 1000 epochs with training and test split at 85/15 ratio.

CNN Model (2nd Experimental Model)

The second deep learning approach for this project included the representation of the dataset in a varied format to fit the required format for the CNN model design. The same dataset was used, but instead, the hourly timestamps of each entry were used, and then the same min, max, average, and meter number was used to classify the transformer ID. The dataset was augmented to pad the grouping of meter number entries to achieve a square image of 22x22, corresponding to approximately three months and twenty-four entries per day for each meter number, with some data points filtered or padded as needed. The next step to prep the dataset for the model was to create images by using the voltage min, max, and average as R, G, and B

channels, grouping by their corresponding transformer ID, and using the Standard Scaler to fit the 255 feature range of the images. With the images in their respective transformer directories, a CNN model was constructed to pass in the images to classify the transformer ID based on the sequential pixels of the meter number image. The CNN model consisted of an Input layer, with a convolutional layer of 32 filters of a 3x3 size and using the ReLU activation function. The input shape is the side of the image length (or 22 for the previously generated meter number images) and a coloured three channel for R, G, and B values. This layer was intended to capture the more basic features in the images and spread the colours seen in the varied power usage per hour. The Max Pooling layer helped to reduce the spatial dimension from the previous layer by half. The additional set of Conv2D and MaxPooling2D layers were intended to capture the additional features within the images. The flattening layer flattened the 3D output of the previous layers into a 1D array for the remaining dense layer. The first dense layer serves as a classification layer using the features extracted by the previous Conv2D and Pooling layers. In contrast, the second dense layer is the output layer that matches the neurons to the number of classes (or transformer IDs) in the dataset. The flattening layer flattened the 3D output of the previous layers into a 1D array for the remaining dense layer. The model was compiled using the ‘Adam’ optimizer, a ‘categorical cross entropy’ loss function, and focused on "Accuracy" for the evaluation metric during the model performance.

The CNN model was trained for a set of 300 epochs with a batch size of 32, and the steps per epoch were dynamically calculated based on the sample used divided by the batch size for completeness of training. An additional model was constructed with a dropout layer added, but similar results were achieved. Additional considerations on this approach include altering the model design and dataset to add an LSTM design with the existing CNN model. This approach

will group the timestamps by day for the pixels (24 pixels for each day), and then the LSTM design will be able to perform a sequence on the daily/weekly sequence. This approach will accomplish a similar temporal approach. However, the model will handle the sequencing through the LSTM design instead of the temporal aspect being handled as a pure image classification design.

Results & Conclusion

Baseline Model

This project research began by establishing the baseline model, a crucial step in the project. Following Tennakoon, R.N. et al.'s methodology (Tennakoon, R.N. et al., 2020), Euclidean distance was calculated to measure similarity values between each smart meter's voltage profile and a reference meter from each transformer. Then, a support vector machine (SVC) was used for transformer prediction based on the smart meter similarity scores. After fine-tuning the SVM model at 0.65, the SVC model achieved final accuracy in weighted precision at 0.68 and recall at 0.65 on 262 transformer classifications. Comparatively, Tennakoon, R.N. et al.'s research, using the same method, achieved an accuracy of over 0.90 with 710 smart meters connected to five (5) transformers. This highlights the significance of the research, as it demonstrates that the data set used for this project, where 1258 meters were connected to 262 transformers, may not be able to reach high accuracy in the final result. However, the correlation between the meter voltage measurement and the transformer connection is still suitable for predicting the distribution topology connections.

Transformer Model

During the transformer model experiments, different data formats were used to test the possibility of increasing the data quality; for example, one of the tests was a separate three-month data set every week that retained the week of the month and month of the year in the feature beside voltage measurements. The final transformer model training result was training accuracy at 0.60, validation at 0.40, and test accuracy at 0.375. The training accuracy and loss curve are shown in Figure 6.

Model training accuracy continues to improve with the loss getting lower, which shows the transformer model is learning. Before 200 epochs, the validation accuracy closely follows the training accuracy. However, after 200 epochs, the validation accuracy and loss have plateaued, indicating the model is starting to overfit after the 200 epochs of training. This is after the drop-off layer and other regularization methods have been added to the model. This indicated that the dataset would require more data from different meters in the same transformer to improve the granularity and model accuracy.

This project has also explored the synthetic sample data argumentation on the dataset. By applying the Synthetic Minority Oversampling Technique (SMOTE) on the training dataset, the training and validation accuracy has been boosted up to 0.58, and the risk of overfitting has been reduced, as shown in Figure 7. This could also mean that if more new meter data are collected, the model accuracy could increase significantly.

CNN Model

The approach for the CNN model design followed a stricter preparation than standard image classification tasks, as the sequence of the pixels in the image needed to remain in the generated location and not shifted, flipped or rotated to preserve the temporal characteristics. With the limited dataset, the model was prone to overfitting, and the initial training runs

displayed training accuracy topping out at 80%. In contrast, the validation accuracy maxed out at 45%, which can be observed in Figure 8. Including an additional dropout layer helped alleviate the overfitting of training data. However, the model was still prone to overfitting. After approximately 200 epochs, the model begins to overfit more heavily, as observed in the Transformer model training process. The CNN model design would likely improve with increased training data instead of reducing the model's complexity and requiring additional dropout layers and regularization.

Conclusion and Recommendations

Based on the baseline model testing result, the traditional machine learning method under current data conditions achieves 0.65 accuracy compared to over 0.90 accuracy in the original research. This indicates that more meters of data representation will be needed for higher accuracy. The test showing the potential of the deep learning transformer model that is based on timestamp voltage data will only learn from the data; however, based on the nature of deep learning, it usually requires more data than the traditional machine learning data method, the potential of the deep learning model like transformer could benefit the electricity distribution topology classification task. The traditional machine learning method requires a long computation time on the similarity score; the deep learning method would be a better solution than the traditional machine learning method.

One possible accuracy improvement on the deep learning model is mainly from more AMI meters deployed in the field, and more data needs to be collected from the meter. Another could be testing a pre-trained deep learning transformer model to perform transfer learning, where the base model will be trained and have an initial weight assigned to the pre-trained model. The additional data would enhance the model performance without overfitting through

increased model complexity, precisely without needing to exhaustively augment the existing dataset or apply the dropout and normalization to the degree to which the current approaches are taken. Each model design and approach provides a viable method. Still, additional data either confirm the efficacy of the architecture or highlight the flaws in the approach if the model does not improve with the larger dataset. To productionize the approach, the continual intake of meter data could help to retrain the model with more extensive and current usage data. This constant influx of data into the training dataset would enhance the demand for deep-learning approaches.

Visualizations

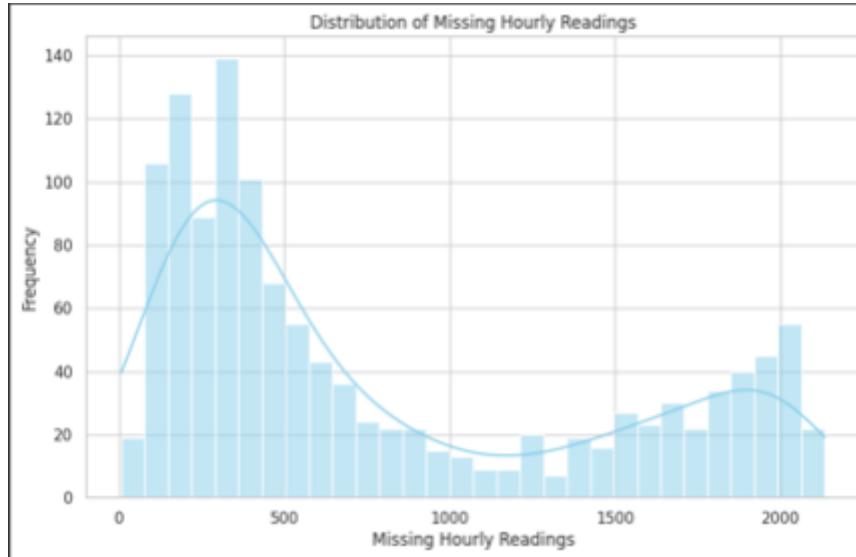


Figure 1. Dataset Missing Hourly Readings

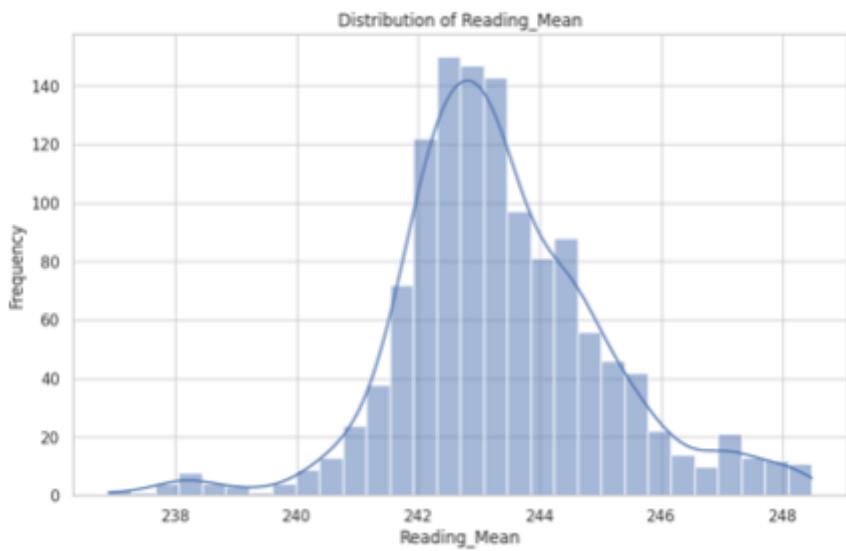


Figure 2. Data Distribution Plot

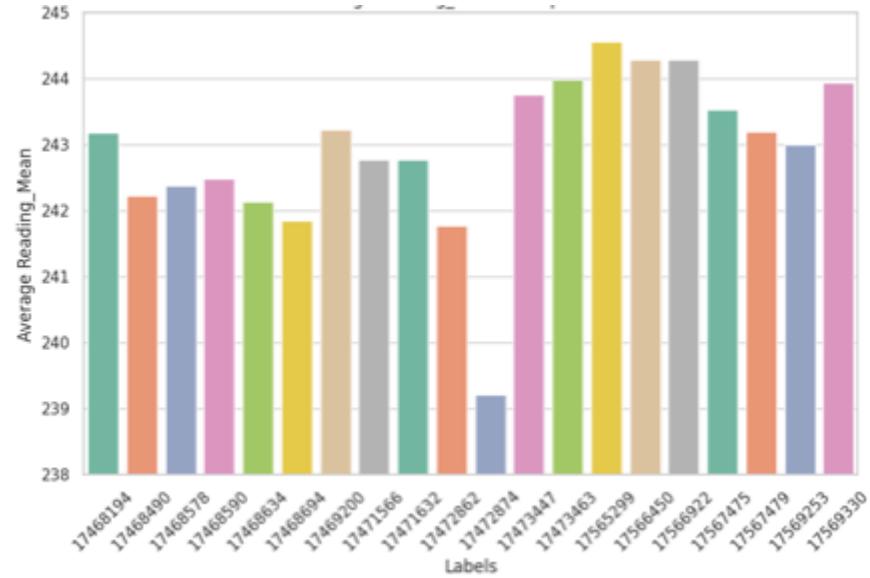


Figure 3. Top 20 Transformer Voltage Reading Mean

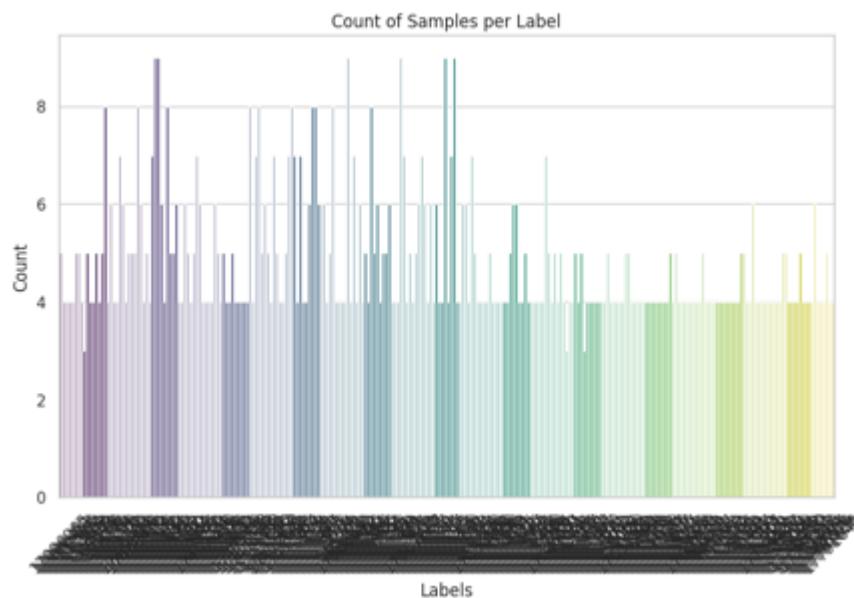


Figure 4. Dataset Classification Balance Plot

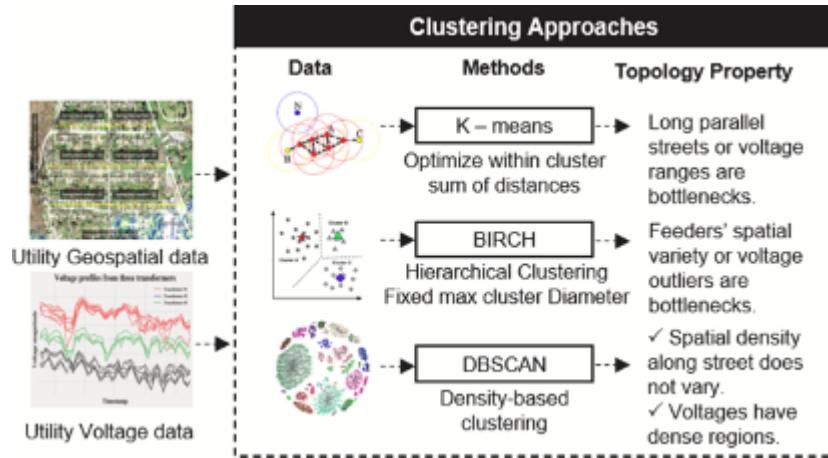


Figure 5. Clustering Solution Cook, E. et al. (2022)

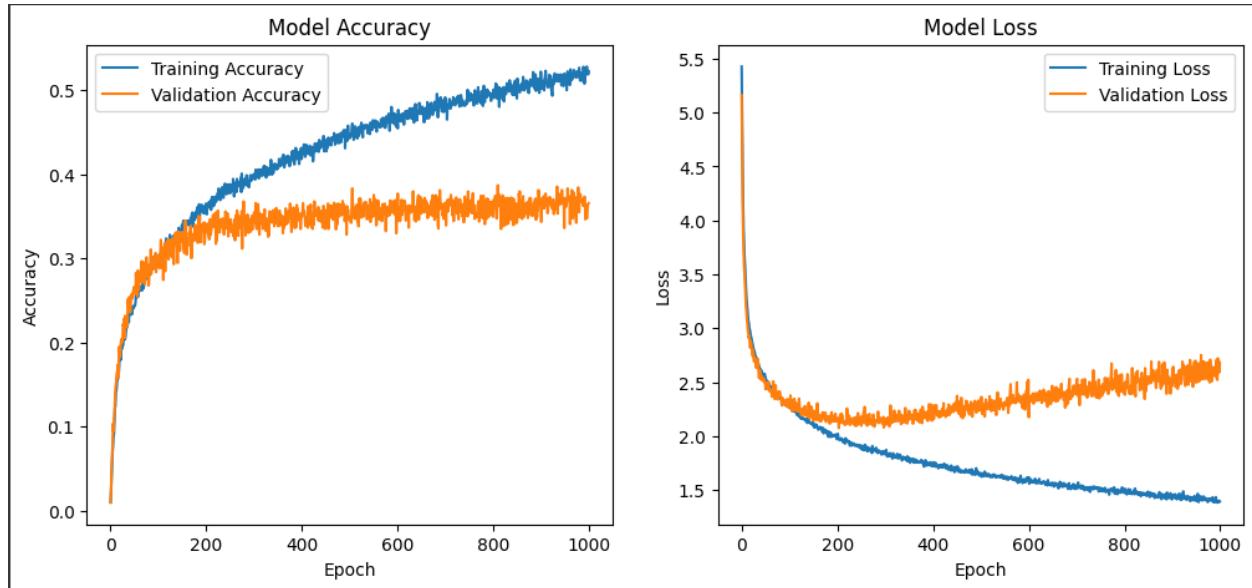


Figure 6. Transform Model Training Curve with Original Dataset

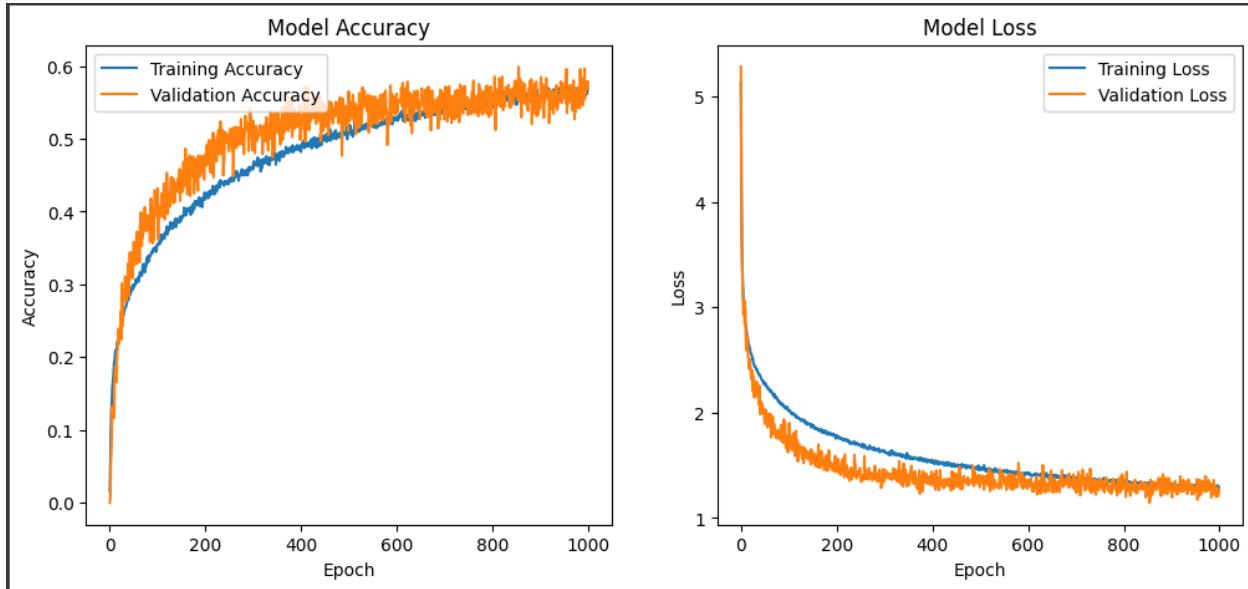


Figure 7. Transformer Model Training Curve with SMOTE Oversample

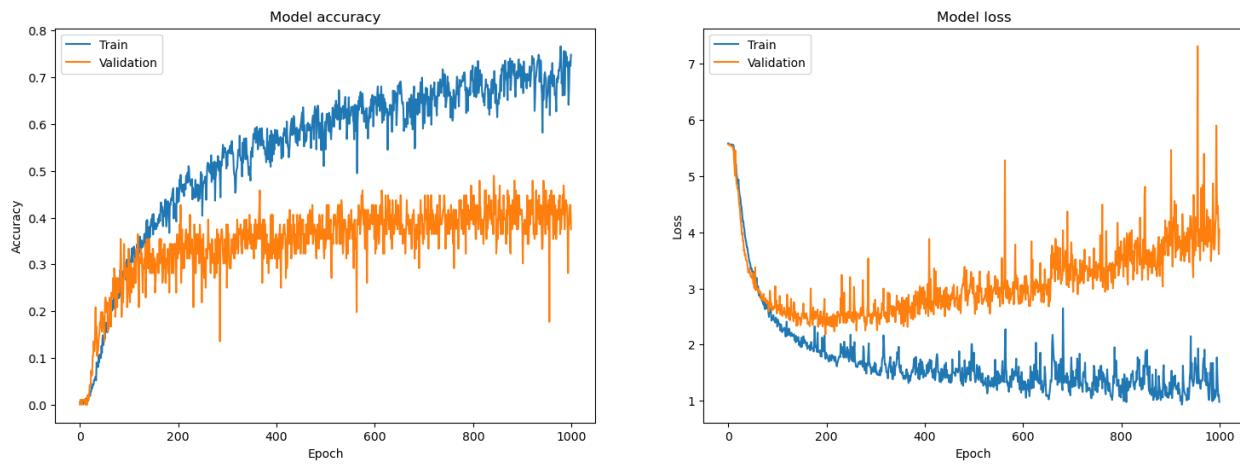


Figure 8. CNN Model Training Evaluation and Loss

Reference

- Cook, E., Saleem, B. M., Weng, Y., Abate S., Kelly-Pitou, K., & Grainger, B. (2022). Density-based clustering algorithm for associating transformers with smart meter via GPS-AMI data. *International Journal of Electrical Power and Energy Systems*, 142 (2022) 108291.
- EPIC Final Report: Data Analytics for Predictive Maintenance. (2023). Pacific Gas and Electric Company. https://www.pge.com/pge_global/common/pdfs/about-pge/environment/what-we-are-doing/electric-program-investment-charge/EPIC-3.20-Final-Report.pdf
- Luan, W., Peng, J., Maras, M., Lo, J., & Harapnuk, B. (2015). Smart Meter Data Analytics for Distribution Network Connectivity Verification. *IEEE Transactions on Smart Grid*, VOL 6, NO. 4, July 2015, 1964-1971.
- MathWorks. (2024). Sequence Classification Using CNN-LSTM Network. MathWorks. <https://www.mathworks.com/help/deeplearning/ug/sequence-classification-using-cnn-lstm-network.html>
- McGirr, T., Saha, O. (2023) Music Classification using Deep Learning. University of San Diego <https://github.com/tmcgirr/AI-511-Music>
- SaskPower (2024). [AMI Meter Voltage Dataset with Transformer Label] [Unpublished raw data]. Saskatchewan Power Corporation.
- Tennakoon, R. N., Kulathilaka, A. T., Giri, R., Khafaf, N. A., Meegahapola, L., & Jalili, M. (2020, November 29 – December 3). Prediction of Customer Connectivity to Distribution Transformer via Application of Machine Learning Techniques to Smart-Meter Data [Conference presentation]. Australasian University Power Engineering Conference, AUPEC 2020, Hobart, TAS, Australia.

Wijkhuizen, M. (2023, April 04). GISLR TF Data Processing & Transformer Training. Kaggle.

<https://www.kaggle.com/code/markwijkhuizen/gislr-tf-data-processing-transformer-training>

Summer 2024



Object Detection for Unmanned Aerial Vehicles

Carson Edmonds

Patricia Enrique

Jeremy Krick

Department of Engineering, University of San Diego

AAI-590: Capstone Project

Professor Marbut

August 12, 2024

[GitHub Link](#)

Introduction

The main challenge facing unmanned aerial vehicles (UAVs) is maintaining a safe distance from obstacles, a task known as sense and avoid (SAA). Despite careful route planning and generally sparse airspace, autonomous drones may still unexpectedly encounter airborne or static obstacles in their path. The autonomous SAA system for UAVs is responsible for handling situational awareness, decision-making, and control of the aircraft to execute evasive maneuvers effectively. To address this issue various sensing options are available on UAVs, such as radar, LIDAR, passive electro-optical sensors, and passive acoustic sensors. However, using visual cameras for the SAA task is appealing due to their lightweight and cost-effectiveness (Amazon Prime Air, 2021).

For this analysis, a SAA solution is developed by applying computer vision techniques to monocular video and images produced by a singular vision camera. The designed model is intended for use on a UAV, to assess data from the cameras in real-time. For static objects, the task is confined to the detection and location of the object. For airborne objects, trajectory planning is necessary to determine if route modifications need to be made. This task can be completed through the analysis of the object's motion over time by detecting and tracking the object across successive video frames.

The data used to develop the necessary model is generated by two aircraft equipped with high resolution cameras and made available by Amazon Web Services (AWS) (Amazon, 2021). The dataset consists of 4,943 flight sequences of around 120 seconds each and over 5.9 million images. About half of the images contain airborne objects and are labeled accordingly with an average of 1.3 labels per image. The remaining images contain no airborne objects and have no corresponding label.

The goal of this analysis is to develop a model that can detect airborne objects within a distance that allows for corrective maneuvering if necessary and track airborne objects to allow for future motion prediction.

Dataset Summary

The dataset used for this analysis consists of video frames, formatted to a standard 2448-pixel width by 2048-pixel height and encoded as 8-bit grayscale images. Bounding boxes were manually labeled for all visible airborne objects in the images and are classified as helicopters, airplanes, birds, airborne, drones, and flocks as shown in Figure 1. The airborne label contains objects that do not fall into the other categories such as hot air balloons and ultra lights. The object labels are text datatypes and any additional information pertaining to the images such as size and distance to the object are stored as float datatypes. Figure 2 illustrates the distribution of objects in the dataset. It is noted that the distribution of objects per image is skewed to favor those with only one labeled object as shown in Figure 3. Although this finding is not a critical deterrent from the goal of the analysis, it highlights the limitations of applying the results to an autonomous drone operating in a more congested airspace.

Figure 1

Labeled image used for the training dataset

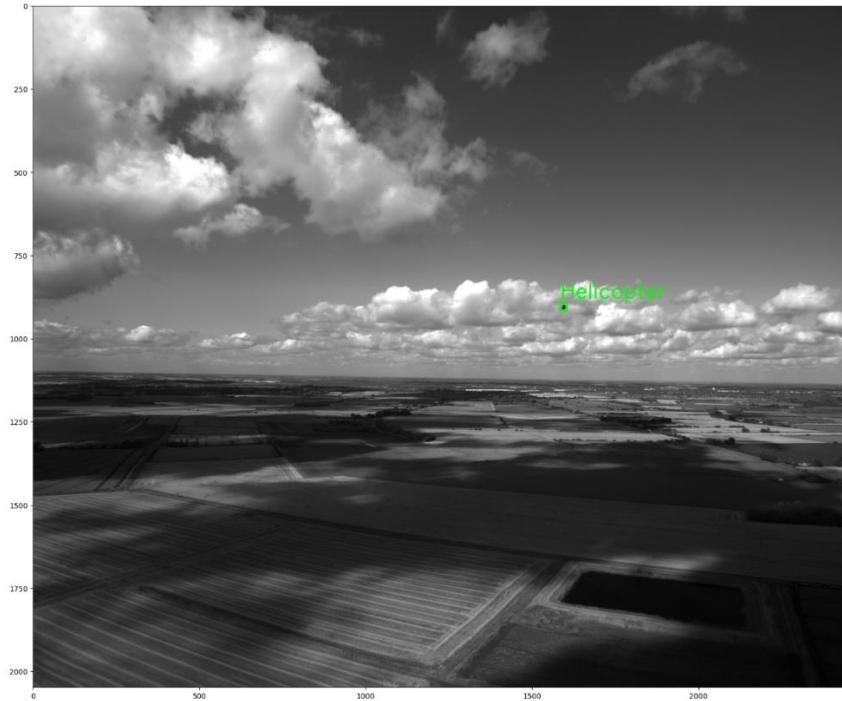
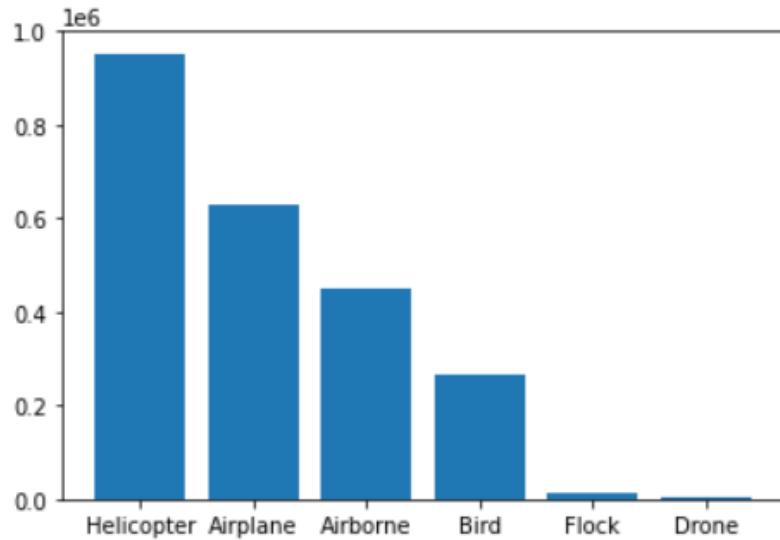
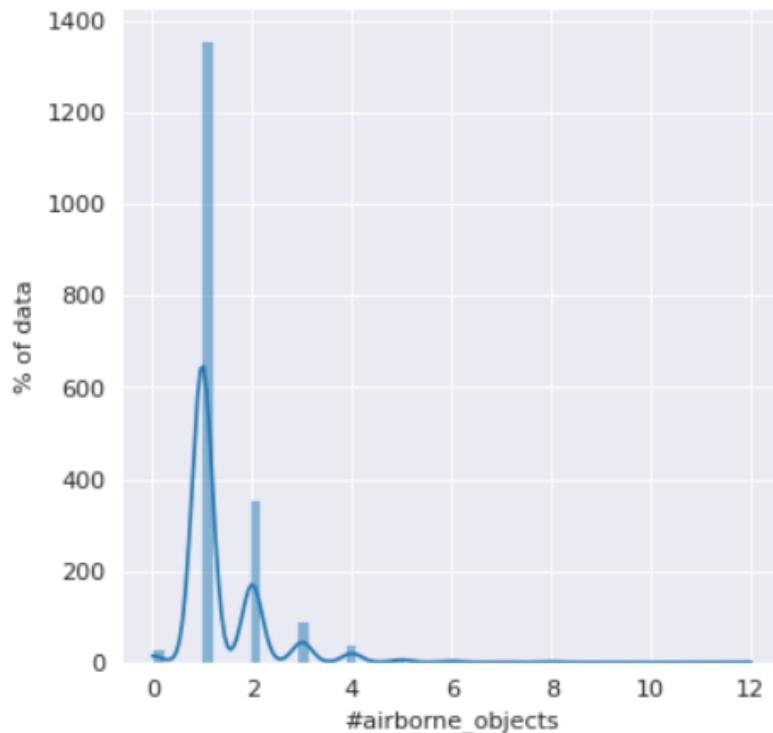


Figure 2

Object distribution in the dataset

**Figure 3**

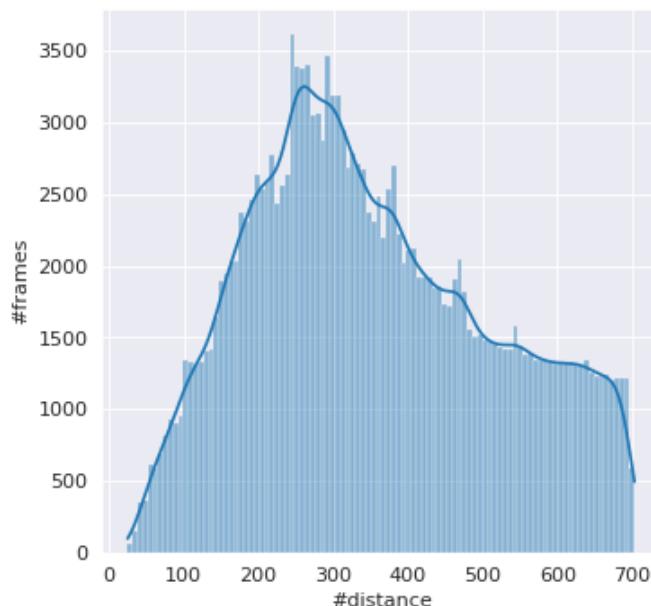
Objects per image distribution



An issue that arises with handling this dataset is the size and resources needed to load the complete data. The original dataset is up to 13TB and for the purpose of this analysis 500GB (about 167,000 images across 859 flights) are loaded and stored to accommodate computing and timing constraints. The dataset was assembled and made available for the purpose of computer vision applications and is thus preemptively cleaned and standardized. As such, there are no duplicated data points and minimal missing values. Due to the nature in which the data was collected, both planned and unplanned objects crossed paths with the UAV. Only objects which were planned have an associated variable storing the distance between the UAV and the detected object, all unplanned objects contain a missing value for this feature. To prevent the loss of data, this missing value is filled with the mean distance value. The distribution of the distance between the UAV and the detected object can be seen in Figure 4. This figure illustrates a skew in the detection of objects, where a majority of the objects detected are between 200m and 300m. This indicates that the performance of the model may be dependent on the distance the UAV is from the object being identified.

Figure 4

Distribution of the distance between that UAV and the detected object



Background Information

Currently, there are a few approaches implemented to achieve SAA in UAVs without the use of artificial intelligence methods. The most notable are documented by NASA and the MIT Lincoln Laboratory (Balachandran, et al., 2018). A distributed software architecture was developed by NASA, known as the Independent Configurable Architecture for Reliable Operations of Unmanned Systems (ICAROUS) with the purpose of enabling safe UAV operations. This method uses a publisher-subscriber middleware to interface between the autopilot system and the algorithms for path planning, traffic avoidance, geofence handling, and decision making. The SAA portion of ICAROUS is handled by the Detect and Avoid Alerting Logic for Unmanned Systems (DAIDALUS) which is a software library available under NASA's Open-Source Agreement. At the core of this concept is a mathematical definition describing when two aircraft are well clear of each other based on distance and time variables, and predefined threshold values. When these thresholds are violated, DAIDALUS provides guidance on maneuvers and ranges to maintain or reestablish the well clear definition.

Alternatively, the Ground-Based Sense-and-Avoid (GBSAA) system was developed by the Lincoln Laboratory in partnership with the U.S. Army Program Manager Unmanned Aircraft Systems to specifically comply with the Federal Aviation Regulations (FARs) requirements to remain well clear of other aircraft (MIT Lincoln Library, n.d.). The GBSAA uses 3D radars as well as Federal Aviation Administration (FAA) radars to generate an image of the airspace and locate any aircraft that may pose a collision threat. The radar data is processed to analyze aircraft locations and estimate the risk posed by other aircraft to the UAV. If risks are detected, the GBSAA issues warnings and provides maneuver guidance to the GBSAA or UAV operator. This method provides the benefit of being an open architecture, allowing for integration with other sensors and the use of radar eliminates the need for additional equipment on the UAV making it a suitable option for UAVs of all sizes.

Alternatively, computer vision methods can be used as a solution to the SAA problem outlined.

A convolutional neural network (CNN) is a computer vision method most commonly applied to the analysis of images and is inspired by the visual process used by humans. Using this method a model can be developed to determine whether helicopters, airplanes, birds, airborne, drones, or flocks are present in the images it is fed from the UAV.

The Vision Transformer (ViT) pretrained model is pretrained on the ImageNet and ImageNet-21k datasets which are large-scale object detection, segmentation, and captioning datasets designed for research in a wide variety of object categories and is considered a benchmark for computer vision models (Dosovitskiy, et al., 2021). The dataset consists of over 200,000 categories including people, cars, animals, and food each with several hundred images. The ViT model was introduced in 2021 as a more computational efficient and accurate model than the CNN. The transformer is designed for computer vision by breaking down an input image into a series of patches which are flattened into vectors and mapped into a smaller dimension. Transfer learning allows the knowledge gained from the pretrained ViT model to be applied to the SAA task without the need to train a new model from scratch. This process further reduces the computational efforts and increases the accuracy of the performance.

A paper published in the 2023 IEEE International Conference on Sensors, Electronics and Computer Engineering compared the performance of ViT and CNN based models on a classification task using a UAV dataset composed of 1359 images (Zhang, 2023). The CNN based models used were a traditional CNN with 14 layers, ResNet50, and VGG16 which were evaluated against the ViT-b16 pretrained model. The study concluded that the ViT model outperformed the CNN based models in the classification and object detection tasks. However, the training process was more elaborate for the ViT model and played a critical role in the overall performance. It should be noted that the dataset used contained one object per image and the results achieved may not be generalized to a task with multi object detection and classification.

An alternative pretrained model for object detection and image segmentation is the You Only Look Once (YOLO) model, pretrained using the Common Objects in Context (COCO) dataset (Jocher et al., 2023). The dataset consists of 80 object categories including people, cars, animals, and sport equipment (Jocher & Laughing, 2023). The latest version of this model is YOLOv9 which was released in February 2024 with the intention of outperforming convolutional-based and transformer-based models in accuracy and speed. Similar to the ViT model, transfer learning allows the YOLO model to be applied to the SAA task with less computational resources and a higher performance when compared to a traditional CNN model.

The performance of the YOLOv5 model on a dataset generated by a UAV is studied in the paper *Innovation in Livestock Surveillance: Applying the YOLO Algorithm to UAV Imagery and Videography* (Kurniadi, et al., 2023). The dataset used to train the model consisted of 3131 images containing cows and 836 not containing cows. For this classification task the model achieved the best results when the UAV was five meters away from the object and not in motion with an accuracy of 75%. The poorest performance occurred when the UAV was ten meters away from the object and in motion with an accuracy of 0. For the purposes of this study the restrictions on the UAV distance and motion were not limiting factors; however, for other applications these restrictions should be taken into consideration.

Experimental Methods

For this analysis a traditional CNN model and a ViT pretrained model are developed to address the SAA task presented.

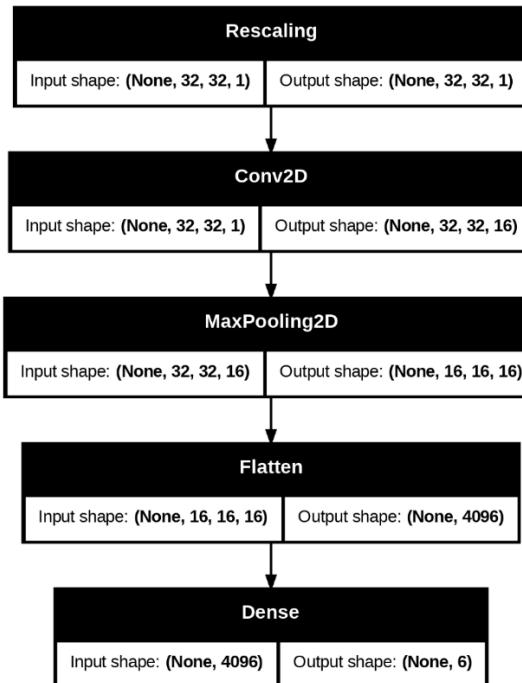
CNN

The architecture of the generated CNN model consists of three layers known as the convolution layer, the pooling layer, and the fully connected layer. The purpose of the convolution layer is to obtain significant features from the input by using filter parameters which are learned throughout the training

process. The pooling layer is used to reduce the computational complexity and the number of parameters by reducing the spatial size of the input. The final fully connected layers flatten the data into a one-dimension vector and process the data for classification (Goodfellow et al., 2016). An outline of the simple model architecture can be seen in Figure 5.

Figure 5

CNN architecture



For the model training process, the data is split into training, validation, and testing datasets in a 70:20:10 ratio respectively. The training and validation datasets are used during the training process, and the testing dataset is reserved for the final evaluation of the model. After the images are pre-processed, they are used as an input into the CNN model built using TensorFlow Keras (“Convolutional Neural Network (CNN)”, n.d.). The model is evaluated on accuracy, precision, and recall with additional emphasis on precision to prevent false alarms which may trigger unnecessary UAV maneuvers. Due to the multi classification nature of the problem, a categorical cross-entropy loss function defined by TensorFlow is implemented (“tf.keras.losses.CategoricalCrossentropy”, n.d.).

To optimize the model developed, various aspects are modified and tested. The size and the number of layers in the model architecture are both modified. A simple model is preferred over a more complex one due to computational efforts and as a result a model with a few larger layers is tested before a model with multiple hidden layers. During the training process the number of epochs and the batch size are adjusted depending on the results seen. If the model appears to be overfitting and learning patterns specific to the training dataset, the number of epochs is reduced. Conversely, if the model appears to not be learning the relationship between the images and the labels, the number of epochs is increased.

ViT

The ViT pretrained model is designed for video classification, detection, and segmentation and uses an image as its input type (“Getting started with transforms v2”, n.d.). Figure 6 illustrates the bounding boxes generated by the model for the intended purpose of object detection and it can be seen that the model is not trained to detect objects in images taken by a UAV for the SAA task. The expected result is the object detection and bounding box from Figure 1 however, the model is incorrectly detecting clouds as one of the objects.

Figure 6

ViT object detection and segmentation



To correct the behavior seen, the pretrained model is fine-tuned with the UAV training set to learn the relationships between the images and the labels. During this process the model is further optimized by selecting hyperparameter values that achieve the best performance. These parameters include batch size, epochs, learning rate, and an optimizer and are selected via a trial-and-error method.

Results/Conclusion

After the model development and optimization processes are complete, it is determined that the CNN model with a filter size of 16, a 3x3 kernel, and 100 epochs achieves the best performance. The model performance metrics are summarized in Table 1. From the metrics obtained, the model is not properly learning to identify and classify objects in the images from the UAV. The low precision and high recall indicate that the imbalanced dataset is drastically affecting the model results. The accuracy of the model is illustrated in Figure 7 and shows that the accuracy is increasing as expected with the training data; however, the validation accuracy is not improving as rapidly as the model learns. Figure 7 further illustrates the poor model performance through the loss curve. The increase in validation loss indicates that the model is overfitting to the training dataset and will not be able to generalize to new data.

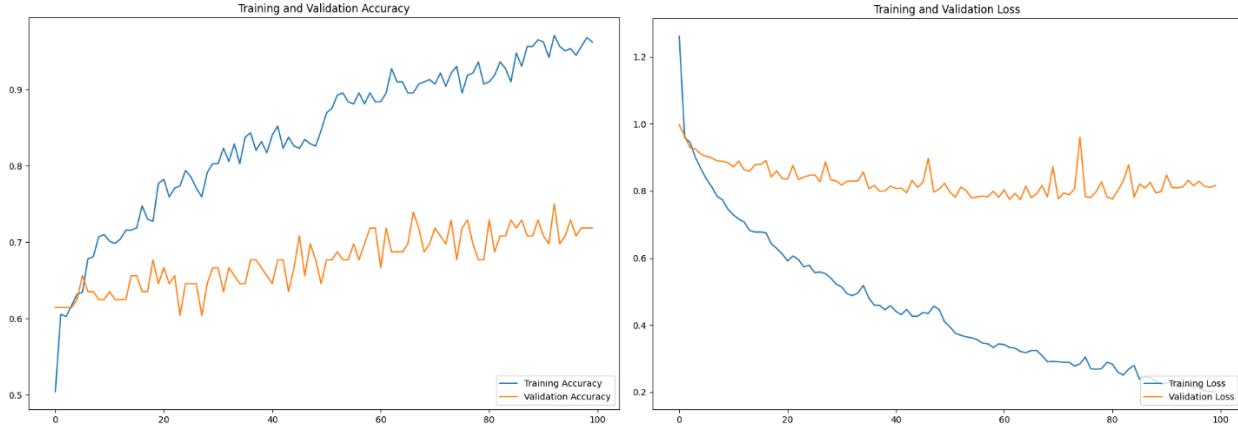
Table 1

CNN model performance metrics

Evaluation Dataset	Accuracy	Precision	Recall
Validation	0.719	0.465	0.896
Testing	0.745	0.432	0.872

Figure 7

CNN model accuracy (left) and loss (right)



The training process for the ViT model reveals that the best classification performance is obtained with an epoch and batch size of 3, a learning rate of 0.0002, and the Adam optimizer. Table 2 summarizes the final training performance of the model after optimization. It can be seen that neither the training loss nor the testing accuracy change significantly during the model training indicating that the model is not learning the desired relationships.

Table 2

ViT model training performance

Epoch	Training loss	Testing Accuracy
1	0.642	0.67
2	0.572	0.67
3	0.639	0.67

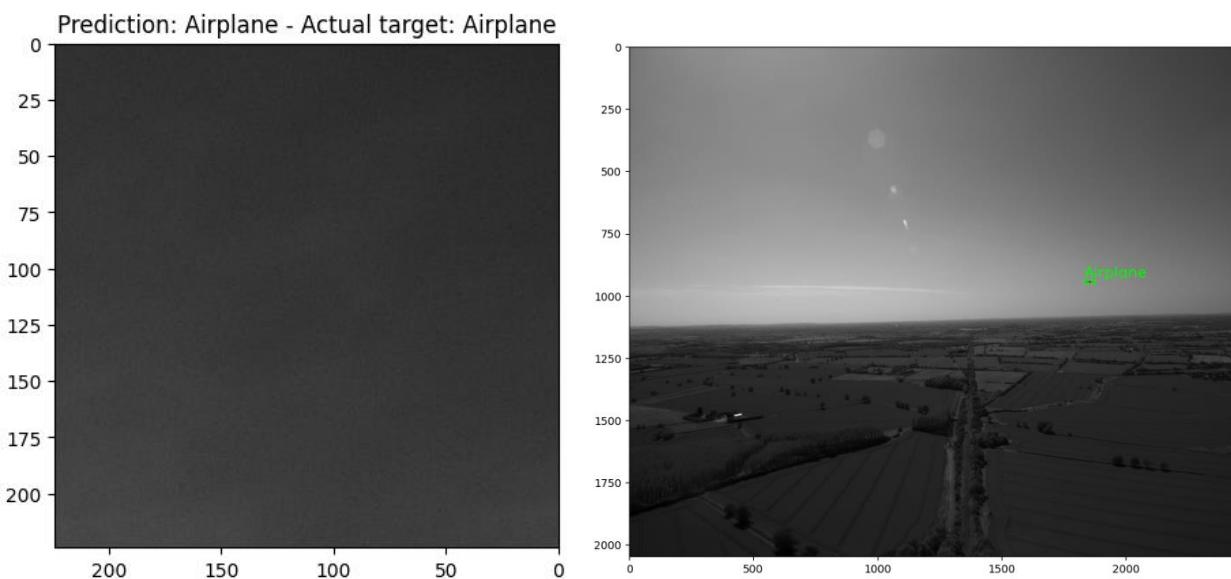
Despite the poor loss and accuracy results, the ViT model improves in the object detection results once it is fine-tuned using the training dataset when compared to the initial model. Figure 8 shows the expected bounding box for an input image on the left and the model results after optimization on the right. It can be seen that the model learned during the training set and is able to successfully locate the object in the image. Implementing both the classification and object detection aspects of the fine-tuned model results in the predictions seen in Figure 9.

Figure 8

Optimized ViT model bounding boxes

**Figure 9**

ViT model classification and object detection after fine-tuning



The results and conclusions drawn from this analysis are limited by the computational resources available. The use of a larger dataset for training the CNN model for the SAA task presented is expected to yield more promising results as the issue of overfitting may be mitigated. A larger training dataset would also benefit the ViT model by allowing more objects to be successfully detected in the images. Additionally, due to the environment in which the model is expected to be deployed, further exploration is required to reduce the latency of the model outputs. Once the model metrics are optimized, the integration of the computer vision model with the UAV can be established. A secondary interface needs to be developed to translate the results produced by the model into an action which the UAV can take.

Works Cited

Amazon (2021). *Airborne Object Tracking Dataset*. (Version 1.0)[Data set]. AWS.

<https://registry.opendata.aws/airborne-object-tracking/>

Amazon Prime Air (2021). *Airborne Object Tracking Challenge*. Alcrowd.

<https://www.aicrowd.com/challenges/airborne-object-tracking-challenge>

Balachandran, S., Muñoz, C., Consiglio, M., Feliu, M., & Patel, A. (2018). Independent Configurable Architecture for Reliable Operation of Unmanned Systems with Distributed Onboard Services. *2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC)*.

<http://dx.doi.org/10.1109/DASC.2018.8569752>

Convolutional Neural Network (CNN) (n.d.). TensorFlow. Retrieved July 29, 2024, from

<https://www.tensorflow.org/tutorials/images/cnn>

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR 2021*.

<https://arxiv.org/abs/2010.11929>

Getting started with transforms v2. (n.d.). PyTorch. Retrieved July 28, 2024, from

https://pytorch.org/vision/main/auto_examples/transforms/plot_transforms_getting_started.html#sphx-glr-auto-examples-transforms-plot-transforms-getting-started-py

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. *MIT Press*.

Jocher, G. & Laughing, Q. (2023, November 22). *COCO Dataset*. Ultralytics. Retrieved December 4, 2023, from <https://docs.ultralytics.com/datasets/detect/coco/>

- Jocher, G., Laughing, Q. & Exel, A. (2023, November 22). *Object Detection*. Ultralytics. Retrieved December 4, 2023, from <https://docs.ultralytics.com/tasks/detect/>
- Kurniadi, F. A., Setianingsih, C. & Syaputra, R. E. (2023). Innovation in Livestock Surveillance: Applying the YOLO Algorithm to UAV Imagery and Videography. *2023 IEEE 9th International Conference on Smart Instrumentation, Measurement and Applications (ICSIMA)*, 246-251. <https://doi.org.sandiego.idm.oclc.org/10.1109/ICSIMA59853.2023.10373473>
- MIT Lincoln Library (n.d.). *Ground-Based Sense-And-Avoid System*. Lincoln Library Massachusetts Institute of Technology. <https://www.ll.mit.edu/r-d/projects/ground-based-sense-and-avoid-system>
- tf.keras.losses.CategoricalCrossentropy* (n.d.). TensorFlow. Retrieved July 29, 2024, from https://www.tensorflow.org/api_docs/python/tf/keras/losses/CategoricalCrossentropy
- Vision Transformer (ViT)* (n.d.). Hugging Face. Retrieved July 22, 2024, from https://huggingface.co/docs/transformers/en/model_doc/vit
- Vuong Tuan Khanh (2021). *Airborne Object Detection and Tracking*. Github. https://github.com/VuongTuanKhanh/Airborne-Object-Detection-and-Tracking/blob/main/notebooks/Dataset_EDA_and_Download.ipynb
- Zhang, J. (2023). Towards a High-Performance Object Detector: Insights from Drone Detection Using ViT and CNN-based Deep Learning Models. *2023 IEEE International Conference on Sensors, Electronics and Computer Engineering (ICSECE)*. <https://doi.org.sandiego.idm.oclc.org/10.1109/ICSECE58870.2023.10263514>

Final Capstone Project:

Monthly Passenger Count Prediction for the San Francisco International Airport

Team 1: Jamileh Jahangiry, Prachi Khanna and Se'Lina Lasher

Shiley-Marcos School of Engineering, University of San Diego

AAI 590: Capstone Project

Anna Marbut M.S.

August 12, 2024

Table of Contents

<u>Table of Contents.....</u>	1
<u>Introduction.....</u>	2
<u>Data Summary.....</u>	2
<u>Background on Chosen Experimental Methods.....</u>	4
<u>Experimental Methods and Trainings.....</u>	8
<u>Results and Conclusions.....</u>	13
<u>References.....</u>	16
<u>Visualizations.....</u>	19

Introduction

In this project, we aim to predict the monthly total passenger count at San Francisco International Airport (SFO) using historical data from 2005 to 2018. Accurate passenger forecasting is crucial for airport management, as it enables better resource allocation, staff scheduling, and operational planning (Kamath, 2024). It also helps airlines optimize flight schedules and manage capacity, ultimately enhancing the passenger experience (Kamath, 2024).

Our dataset, sourced from Kaggle's [SF Air Traffic Passenger and Landings Statistics](#), is provided by the city of San Francisco. This comprehensive dataset includes detailed records of passenger counts and landings, which we use to develop our predictive models.

The project has a dual focus: not only do we strive to build an accurate predictive model, but we also seek to evaluate whether advanced deep learning techniques, including pre-trained models, are necessary to achieve satisfactory results. With the rise of complex neural network architectures, there's growing interest in leveraging these models for time series forecasting. Pretrained models, in particular, offer a starting point for developing sophisticated predictions without extensive training from scratch.

By comparing deep learning models, including a pre-trained model, with more traditional forecasting approaches, we aim to determine if the added complexity of these techniques is justified by a significant improvement in predictive accuracy. This insight will help inform future decisions for data science projects in similar contexts, ensuring that the most efficient and effective methods are employed.

Data Summary

Our dataset, sourced from Kaggle's SF Air Traffic Passenger and Landings Statistics, provides detailed monthly records of air traffic data at San Francisco International Airport. The

air-traffic-passenger-statistics.csv file, which we selected for our analysis, contains the following features from **Table 1**:

Variable	Data Type	Definition
Activity Period	Integer	The year and month at which passenger activity took place.
Operating Airline	String	The Airline name for the operator of aircraft with passenger activity.
Operating Airline IATA Code	String	International Air Transport Association (IATA) two-letter designation for the airline.
Published Airline	String	Airline name that issues the ticket and books revenue for passenger activity.
Published Airline IATA Code	String	International Air Transport Association (IATA) two-letter designation for the airline.
GEO Summary	String	Location within the US (“domestic”) or outside the US (“international”) without stops.
GEO Region	String	A more detailed breakdown of the GEO Summary field to designate the region in the world.
Activity Type Code	String	Boarding a flight (“enplanements”), getting off a flight (“deplanements”), and transiting to another location (“in-transit”).
Price Category Code	String	Indicates whether the published airline is a low-cost carrier or not.
Terminal	String	The airport terminal designations at SFO where passenger activity took place.

Table 1. Definitions and table provided by the City of San Francisco in the Kaggle

Dataset (City of San Francisco, n.d.).

Before proceeding to data aggregation, we addressed the issue of missing data. Two variables, Operating Airline IATA Code and Published Airline IATA Code, each had 63 missing values out of 18,885 entries. After reviewing the data, we found that the missing values were linked to companies like Swissport USA, Pacific Aviation, Trego Dugan Aviation, Servisair, and Boeing Company. These companies are involved in aviation services, including ground handling and manufacturing, rather than airline operations (*Trego Dugan Aviation*, n.d.). As a result, they

do not have IATA codes, which are used exclusively for airline identification (*IATA Codes*, n.d.). To handle this, we filled the missing IATA code values with "N/A" to indicate that these entities do not operate as traditional airlines.

To prepare the data for analysis, we aggregated the features by month, which involved performing counts for categorical variables like GEO Summary and Operating Airline to ensure that the data reflected a monthly perspective. The Activity Period was converted into a Datetime stamp to accurately represent the time series data. Following this aggregation, our dataset was reduced from 18,885 entries to 156. **Figure 1** illustrates what the monthly data looks like over the entire dataset.

Additionally, a new feature named Season was engineered and categorically encoded. This feature was added based on the evidence from the seasonal decomposition graph (**Figure 2**), which highlighted both seasonal and trend components in the data. The graph underscored the importance of capturing seasonal patterns, prompting us to include the Season feature to represent different times of the year. By focusing on aggregating data monthly and incorporating this new feature, we were able to better capture overall trends, seasonal variations, and other factors influencing passenger counts at SFO, providing a comprehensive view necessary for accurate forecasting across our various prediction methods.

Background on Chosen Experimental Methods

To investigate the necessity of deep learning models for accurately predicting monthly passenger counts, we explored both traditional and advanced approaches. Linear Regression was selected as one of the traditional methods to serve as a baseline model.

Linear Regression is a statistical technique that models the relationship between a dependent variable and one or more independent variables by fitting a linear equation to the

observed data (Kibet, 2023). In the context of time series forecasting, this method can incorporate lagged values of the time series as features to account for temporal dependencies (Gomedé, 2024). This approach allows the model to leverage past observations to predict future values, providing a simple yet informative baseline for comparison.

While Linear Regression may not capture complex patterns and nuances in the data as effectively as more sophisticated models, it is valuable for understanding basic trends and relationships. For instance, it can reveal whether there is a general upward or downward trend in passenger counts over time. The model's simplicity also makes it easier to interpret and implement, and it serves as a useful starting point for evaluating the added value of more complex models. Proper data preprocessing, such as detecting and handling outliers and missing values, is critical to ensure the model's accuracy and reliability.

In addition to our application, Linear Regression has been successfully applied in various time series forecasting scenarios, such as predicting stock prices, weather conditions, and economic indicators (Thilakarathne, 2020). These examples demonstrate its versatility and utility as a foundational forecasting tool.

We utilized the statistical method SARIMA (Seasonal AutoRegressive Integrated Moving Average) for our project on predicting monthly passenger counts at San Francisco International Airport (SFO). SARIMA is an extension of the ARIMA model that incorporates seasonal differencing to handle periodic fluctuations, making it well-suited for time series datasets with regular seasonal variations (Artley, 2022). In this context, SARIMA was particularly appropriate due to the clear annual seasonal trends identified in the SFO data (Figure 2). These trends, influenced by holidays, vacation periods, and annual events, significantly impact passenger traffic patterns (Bouwer et al., 2024).

The SARIMA model captures these seasonal patterns by including terms for seasonal autoregression, differencing, and moving averages (Artley, 2022). This approach allows the model to represent the increased passenger traffic during peak travel times, such as summer vacations and winter holidays, as well as the decreased volumes during off-peak periods (Bouwer et al., 2024). By accounting for these fluctuations, SARIMA can provide more accurate forecasts that reflect the cyclical nature of the data.

To illustrate SARIMA's effectiveness in capturing seasonal variations, consider its application in forecasting cooling degree-days (CDD). In this study, SARIMA was employed to predict CDD, a measure that quantifies the demand for cooling based on temperature variations (Bilgili, 2023). The study showed how SARIMA effectively managed seasonal fluctuations and trends in CDD data, demonstrating its ability to model complex seasonal patterns. This application is relevant to our work because it parallels the challenge of forecasting seasonal patterns in air traffic data at SFO.

In addition to using SARIMA for handling seasonal variations, we also incorporated the Prophet model developed by Facebook. Prophet is designed to manage time series data with strong seasonal effects and missing data points, making it a valuable tool for our analysis (Khare, 2023). Unlike deep learning models that require extensive computational resources and training time, Prophet offers a more resource-efficient approach by leveraging its pre-trained capabilities and fine-tuning options.

Prophet works by decomposing time series data into trend, seasonality, and holiday components. This decomposition allows Prophet to handle irregularities in the data, such as missing values and outliers, with robust forecasting performance (Khare, 2023). For our dataset, which exhibits clear seasonal trends and occasional outliers, Prophet's flexibility and

interpretability make it particularly suitable. It allows for the easy incorporation of seasonal patterns and holiday effects, which are crucial for accurate passenger count predictions.

To illustrate Prophet's effectiveness, consider its application in forecasting retail sales. In a study by Taylor and Letham (2018), Prophet was used to forecast sales data with strong seasonal effects and irregular patterns. The model successfully captured seasonal fluctuations and provided accurate forecasts despite the presence of missing data and outliers. This example demonstrates how Prophet can manage complex time series data, similar to how it will be applied to our passenger count data at SFO. By incorporating Prophet, we enhance our forecasting capability, ensuring reliable predictions even with data irregularities and seasonal trends.

In our analysis, we first utilize Linear Regression, SARIMA, and Prophet to establish a baseline for forecasting performance. These models help us understand the effectiveness of traditional and statistical approaches in capturing the key patterns in our data.

Following this, we will evaluate whether advanced deep learning models—specifically Long Short-Term Memory (LSTM) networks and Transformers—can offer further improvements. The goal is to determine if these models provide significant benefits over the baseline models and if their complexity is justified.

LSTM networks are a type of Recurrent Neural Network (RNN) designed to address the vanishing gradient problem that can occur in traditional RNNs (Or, 2020). They use memory cells and gating mechanisms to retain and update information over long sequences, which is crucial for capturing temporal dependencies and trends (Or, 2020). In forecasting, LSTMs can learn from sequences of historical data to predict future values, making them useful for identifying long-term patterns and trends in time series data. For instance, a study by Fischer and

Krauss (2018) demonstrated the application of LSTMs to stock market forecasting. The researchers found that LSTMs could effectively capture complex temporal patterns and dependencies in financial data, which is similar to the patterns we might encounter in passenger counts at SFO. This ability to model long-term dependencies makes LSTMs a promising choice for improving forecast accuracy in our context.

Transformers, on the other hand, utilize self-attention mechanisms to process entire sequences of data simultaneously. This allows them to capture complex relationships and long-range dependencies more effectively than traditional RNNs, which process data sequentially (Lanza, 2023). By incorporating positional encoding, Transformers maintain the order of elements in the sequence, which is essential for time series data. An example of their application is seen in a study by Zhang et al. (2022), where Transformers were used for electricity demand forecasting. The study showed that Transformers could manage intricate patterns and seasonal effects in time series data, highlighting their potential for handling the seasonal and trend components in our passenger count data.

By integrating these advanced models, we aim to assess whether LSTMs and Transformers can provide significant enhancements over Linear Regression, SARIMA, and Prophet. The goal is to determine if the added complexity and resource demands of these models are justified for improving our forecasts of monthly passenger counts.

Experimental Methods and Trainings

The first model implemented to compare against the performance of our two deep learning models was a Linear Regression model. We explored several variations, including a simple Linear Regression with past target data only and another that incorporated correlated variables, such as the frequency of operating airlines within a month. Certain airlines

consistently held a higher percentage of the total passenger count (**Figure 3**). However, for conciseness, we will focus only on the best-performing version among the various models and methods attempted.

For the Linear Regression model, the best-performing version utilized not only past values of the target variable but also incorporated seasonal information. The dataset, consisting of 156 entries created during the EDA and Feature Engineering stage, was processed using a sequence function. This function used the data from the last six months (entries) to create the input features (X), which included the previous six months of passenger counts and the associated seasons for those months. The target variable (y) was the passenger count for the following month. This X and y were then split into training, validation, and test sets, with 90 entries for training, 30 for validation, and 30 for testing.

To train the Linear Regression model, we used the **LinearRegression** class from the **sklearn.linear_model** library. The model was fitted to the training data by calling the **fit** method, which calculated the optimal coefficients for the linear equation by minimizing the mean squared error between the predicted and actual passenger counts after training, the model's performance was evaluated on the validation and test sets. The metrics used for evaluation were MSE and the coefficient of determination (R^2), providing measures of the model's accuracy and goodness-of-fit, respectively.

Following the implementation of Linear Regression, we employed the SARIMA model for forecasting. SARIMA is well-suited for time series data with strong seasonal patterns, such as our dataset, which exhibits clear annual trends.

The SARIMA model was implemented using the **SARIMAX** class from the **statsmodels.tsa.statespace.sarimax module**. We configured the model with an *order* of (1, 1, 1)

and a *seasonal_order* of (1, 1, 1, 12). The order parameter includes one autoregressive term, one differencing term for stationarity, and one moving average term. The *seasonal_order* parameter captures the seasonal components with a periodicity of 12 months, corresponding to the annual cycle we've seen in **Figure 2**.

For evaluation, the SARIMA model was trained on the training set and tested on the same set of unseen data used for Linear Regression, which consisted of the last 30 months of data. This consistency in test data across models ensured a fair comparison of performance metrics, such as MSE and R².

SARIMA used only the passenger count data as its input, focusing on modeling the time series itself. By capturing both short-term fluctuations and long-term seasonal effects, SARIMA provided accurate forecasts of monthly passenger counts at SFO, leveraging its ability to handle complex seasonal patterns effectively.

Following the evaluation of SARIMA, which effectively captured seasonal patterns and trends in the passenger count data, we turned to Prophet to further explore its capabilities in time series forecasting. Prophet, developed by Facebook, is designed to handle time series data with strong seasonal effects and missing data points (Seasonality, Holiday Effects, And Regressors | Prophet, n.d.).

Prophet was implemented using the prophet library, which requires a specific DataFrame format. We first created a DataFrame with columns ds for dates and y for the target variable, which in this case is the monthly passenger count. After preparing this DataFrame, we split the data into training and testing sets. For consistency, the test set consisted of the last 30 months of data, the same period used in previous models.

Typically, Prophet utilizes the entire dataset for training, but in our case, we chose to reserve the last 30 months for testing to evaluate how Prophet performs against unseen data. This approach allowed us to assess Prophet's forecasting accuracy and compare it with the results from SARIMA and Linear Regression.

Following the implementation of Prophet, which effectively decomposed the time series into trend and seasonal components, we explored the potential of deep learning models to further enhance forecasting accuracy. The first deep learning model we decided to use was the Transformer.

For the Transformer model, we adapted the approach from Jeff Heaton's implementation, "Transformer-Based Time Series with PyTorch (10.3)." Transformers are particularly adept at handling sequential data due to their self-attention mechanisms, which can capture long-range dependencies. However, they do not inherently recognize the sequence of the data. To address this, Heaton's model includes a positional encoding mechanism that uses sinusoidal functions to encode positional information, allowing the model to account for the order of inputs.

The model architecture was constructed with several layers, including an encoder that maps the input data to a higher-dimensional space, a positional encoding component that adds positional information, and a transformer encoder to process the input sequence. The final layer is a decoder that maps the transformed data back to the output space, predicting the next value in the sequence.

The data preparation for the transformer involved segmenting the time series into sequences of six months, with the corresponding passenger count for the following month as the target. This structured the problem as a sequence-to-value prediction task. The dataset was split into training, validation, and test sets, consisting of 90, 30, and 30 sequences, respectively. The

test set, comprising the last 30 months, was used consistently across all models for evaluation to ensure comparable performance metrics.

The model training involved using a MSE loss function, the Adam optimizer, and a learning rate scheduler to adjust the learning rate based on validation loss improvements. The training process included early stopping to prevent overfitting, stopping the training when the validation loss did not improve for five consecutive epochs. Despite experimenting with modifications, such as removing dropout layers to potentially enhance performance, the original configuration provided the best results. The model's effectiveness was measured against the same unseen test data used for all other models, evaluating its ability to generalize and accurately forecast future passenger counts based on past trends.

After assessing the performance of the Transformer model, we turned our attention to the LSTM network, our final deep learning model. Given its strength in capturing long-term dependencies in sequential data, we aimed to determine if the LSTM could provide additional insights or improvements over the previously tested models, including Linear Regression, SARIMA, Prophet, and Transformers.

To explore the effectiveness of LSTM networks, we applied a model specifically designed to capture temporal dependencies in sequential data. Our dataset preparation for LSTM was consistent with previous models: the data was split into training, validation, and test sets with 90, 30, and 30 sequences, respectively, using the past 6 months of passenger counts to predict the subsequent month's count. The test set, consisting of the last 30 months of data, remained consistent across all models for comparative evaluation.

The LSTM model underwent several optimization trials to fine-tune its architecture and hyperparameters. We experimented with different configurations, including varying the number

of LSTM layers, adjusting the learning rate, and altering the dropout settings. Specifically, we tested models with higher learning rates than the baseline, removed dropout layers, and combinations of these adjustments. The best-performing configuration featured an LSTM model with the original learning rate of 0.01 and no dropout layers. We hypothesize that the relatively small size of the training dataset, comprising only 90 entries, made dropout less effective in preventing overfitting, as it can potentially hinder learning when the data size is limited.

The final model architecture consisted of two LSTM layers, with the first layer having 5 units and the second layer 3 units. These layers were followed by a Dense layer with a single output unit to predict the next value in the sequence. The model was compiled using the Adam optimizer with the original learning rate of 0.01 and theMSE) loss function. To prevent overfitting, we employed early stopping during training, monitoring the validation loss and stopping the training process when it did not improve for three consecutive epochs. This strategy allowed us to determine the optimal number of epochs dynamically, ensuring that the model did not overfit the training data.

The LSTM model's performance was evaluated on the same unseen test data as the previous models. This consistent approach ensured that we could fairly compare the predictive accuracy and generalization capabilities of the LSTM model against Linear Regression, SARIMA, Prophet, and Transformer models.

Results and Conclusions

In evaluating the performance of various models for forecasting monthly passenger counts at SFO, we identified Linear Regression with seasonal components as the top performer. As shown in **Figure 4**, this model achieved the lowest MSE and highest R² on the test data, indicating its robust performance in predicting future passenger counts.

Linear Regression with Seasonal Components demonstrated strong performance across all metrics, with the lowest MSE and highest R^2 among the models. The model's effective capture of seasonal patterns contributed to its high accuracy. **Figure 5** illustrates the actual versus predicted values for this model, highlighting its precision in predicting the unseen test data.

Prophet, a pretrained model designed to handle time series data with strong seasonal effects and missing values, performed well but did not surpass Linear Regression. With an R^2 of 0.881 on the test data, Prophet effectively managed the seasonal patterns and outliers but was less accurate compared to Linear Regression. **Figure 6** shows the predictions made by Prophet alongside the actual values, providing insight into its forecasting capabilities.

SARIMA, showed moderate performance. It achieved an R^2 of 0.813 on the test set, indicating that while it captured the underlying trend and seasonality, it was less effective compared to the top models. **Figure 7** depicts the SARIMA model's predictions, highlighting its alignment with actual values but also showing areas where it diverges.

The Transformer model, adapted from Jeff Heaton's approach, was the least effective. Despite leveraging positional encoding to handle sequential data, it performed poorly with an R^2 of -1.288 on the test data. **Figure 8** illustrates the Transformer's predictions, revealing significant deviations from actual values and highlighting its struggles with the small dataset and univariate input.

LSTM, our final deep learning model, demonstrated limited effectiveness with an R^2 of 0.401. This model showed some potential but did not outperform the simpler models. **Figure 9** presents the LSTM's predictions, indicating that despite various optimizations, the model did not achieve high accuracy.

The analysis of these models suggests that while deep learning approaches like Transformer and LSTM offer sophisticated methods for time series forecasting, they may not always outperform simpler models like Linear Regression with seasonal data included, particularly when the dataset is small and univariate. Deep learning models often require larger datasets and more diverse features to uncover complex patterns effectively.

In future work, expanding the dataset, exploring additional features, and experimenting with more advanced architectures could improve the performance of deep learning models. Additionally, incorporating multiple types of data and refining the training process may yield better results.

References

- Artley, B. (2022, April 26). *Time Series Forecasting with ARIMA , SARIMA and SARIMAX*. Towards Data Science. Retrieved July 29, 2024, from
<https://towardsdatascience.com/time-series-forecasting-with-arima-sarima-and-sarimax-e61099e78f6>
- Bilgili, M. (2023, July 31). Time series forecasting on cooling degree-days (CDD) using SARIMA model. *Natural Hazards*, 118, 2569–2592.
<https://doi.org/10.1007/s11069-023-06109-4>
- Bouwer, J., Hausmann, L., Lind, N., Verstreken, C., & Xanthopoulos, S. (2024, January 8). *How airlines can handle seasonal travel demand*. McKinsey. Retrieved July 29, 2024, from
<https://www.mckinsey.com/industries/travel-logistics-and-infrastructure/our-insights/how-airlines-can-handle-busier-summers-and-comparatively-quiet-winters>
- City of San Francisco. (n.d.). *San Francisco International Airport (SFO) - Air Traffic Passenger and Landings Statistics*, Version 68. Retrieved August 11, 2024, from
<https://www.kaggle.com/datasets/san-francisco/sf-air-traffic-passenger-and-landings-statistics>
- Dolphin, R. (2020, October 21). *LSTM Networks | A Detailed Explanation*. Towards Data Science. Retrieved July 29, 2024, from
<https://towardsdatascience.com/lstm-networks-a-detailed-explanation-8fae6aefc7f9>
- Fischer, T., & Krauss, C. (2018, October 16). Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270(2), 654-669. <https://doi.org/10.1016/j.ejor.2017.11.054>

Gomede, E. (2024, March 19). *Analyzing the Impact of Lagged Features in Time Series Forecasting: A Linear Regression Approach*. Cubed. Retrieved July 29, 2024, from <https://blog.cubed.run/analyzing-the-impact-of-lagged-features-in-time-series-forecasting-a-linear-regression-approach-730aaa99dfd6>

Grammarly. (2024). Grammarly. <https://www.grammarly.com>

Heaton, J. (2023, November 1). *Transformer-Based Time Series with PyTorch (10.3)*. YouTube.

Retrieved July 31, 2024, from https://www.youtube.com/watch?v=NGzQpphf_Vc

IATA Codes. (n.d.). IATA. Retrieved August 9, 2024, from

<https://www.iata.org/en/services/codes/>

Kamath, P. (2024, February 6). *Future of Air Travel Forecast for Aviation Business Travel*.

ITILITE's. Retrieved July 29, 2024, from <https://www.itilite.com/blog/air-travel-forecast/>

Khare, P. (2023, May 13). *Understanding FB Prophet: A Time Series Forecasting Algorithm*.

Medium. Retrieved July 29, 2024, from

<https://medium.com/illumination/understanding-fb-prophet-a-time-series-forecasting-algorithm-c998bc52ca10>

Kibet, M. (2023, April 21). *A Guide to Regression Analysis with Time Series Data*. InfluxData.

Retrieved July 29, 2024, from

<https://www.influxdata.com/blog/guide-regression-analysis-time-series-data/>

Lanza, E. (2023, August 2). *How to Apply Transformers to Time Series Models | by Intel | Intel Tech*. Medium. Retrieved July 29, 2024, from

<https://medium.com/intel-tech/how-to-apply-transformers-to-time-series-models-spacetimeformer-e452f2825d2e>

Or, B. (2020, October 10). *The Exploding and Vanishing Gradients Problem in Time Series*.

Medium. Retrieved July 29, 2024, from

<https://medium.com/metaor-artificial-intelligence/the-exploding-and-vanishing-gradients-problem-in-time-series-6b87d558d22>

Seasonality, Holiday Effects, And Regressors | Prophet. (n.d.). Meta Open Source. Retrieved July 30, 2024, from

https://facebook.github.io/prophet/docs/seasonality,_holiday_effects,_and_regressors.html

Taylor, S. J., & Letham, B. (2018). Forecasting at Scale. *THE AMERICAN STATISTICIAN*, 72(1), 37-45.

Thilakarathne, A. L. (2020, October 9). *Creating a Model for Weather Forecasting Using Linear Regression*. Medium. Retrieved July 29, 2024, from

<https://medium.com/swlh/creating-a-model-for-weather-forecasting-using-linear-regression-b18c1590e8d7>

Trego Dugan Aviation. (n.d.). Trego Dugan Aviation - Home. Retrieved August 9, 2024, from
<https://www.trego-dugan.com/>

Zhang, G., Wei, C., Jing, C., & Wang, Y. (2022, August 18). Short-Term Electrical Load Forecasting Based on Time Augmented Transformer. *International Journal of Computational Intelligence Systems*, 15(67).

<https://link.springer.com/article/10.1007/s44196-022-00128-y>

Visualizations

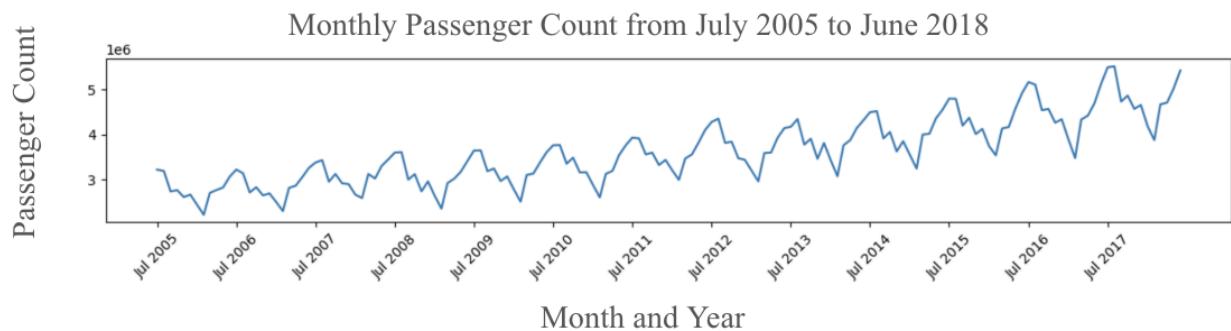


Figure 1. Monthly passenger count at San Francisco International Airport (SFO) from July 2005 to June 2018, showing a steady upward trend with seasonal fluctuations.

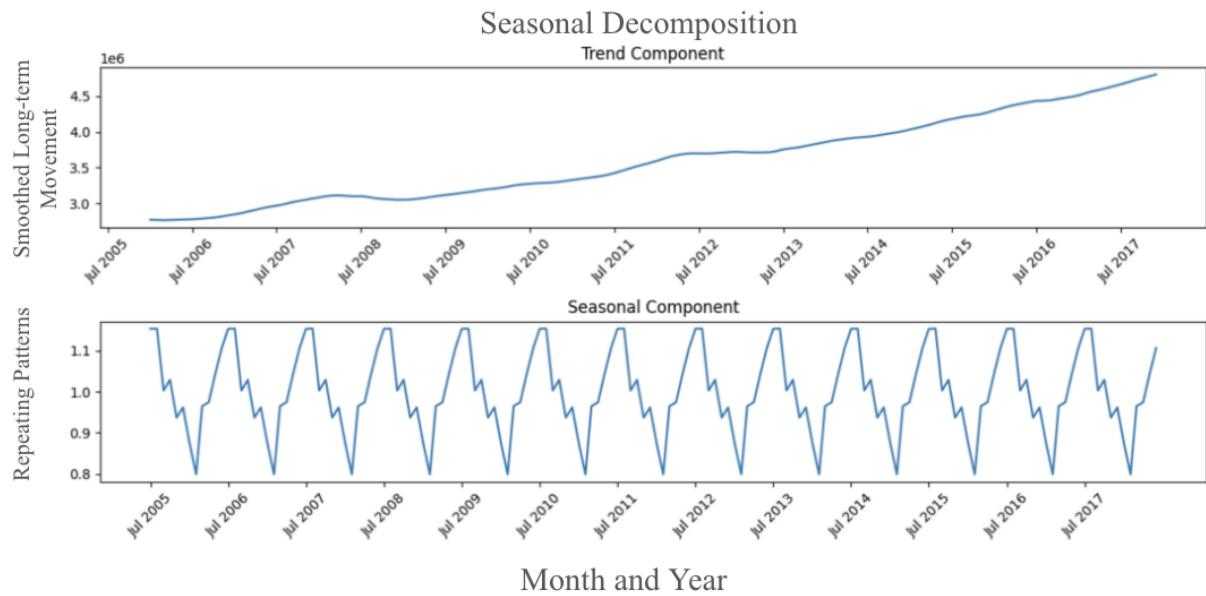


Figure 2. Seasonal decomposition of the monthly passenger count at San Francisco International Airport (SFO) from July 2005 to June 2018. The figure shows the trend component (top), indicating a steady long-term increase, and the seasonal component (bottom), illustrating regular annual fluctuations.

Operating Airline	Mean no. of passengers per year	Share [%]
United Airlines	15571737	46.7
SkyWest Airlines	3184454	9.5
American Airlines	3115884	9.3
Virgin America	2703137	8.1
Delta Air Lines	2486702	7.5
Southwest Airlines	2386783	7.2
Alaska Airlines	1252873	3.8
US Airways	1201186	3.6
JetBlue Airways	814195	2.4
Air Canada	628920	1.9

Figure 3. Average annual passenger count and market share by operating airline at San Francisco International Airport (SFO), with United Airlines holding the largest share at 46.7%.

Model	Train MSE	Train R2	Val MSE	Val R2	Test MSE	Test R2
Linear Regression with Seasons	4494523002.920854	0.9781967200015596	14528335118.989582	0.9370542605637647	26787243321.336082	0.985546032292227
Transformer w/ Dropout	011914720582.4544	-2.0008010841300746	242026394195.458	-0.05005090119803172	583151843623.658	-1.288160925793866
LSTM No Dropout	180726272506.15775	0.6168920614623117	101700336301.60755	0.5553683549112176	152617401295.39673	0.4811625524906367
Prophet	7487072625.830776	0.9794189735016396	None	None	30320490017.289713	0.881829000024614
SARIMA	1483900448568.06885	0.5920921333617261	None	None	47632509654.153484	0.813100449495719

Figure 4. Performance comparison of the best-performing hyperparameters for various models in predicting monthly passenger counts at San Francisco International Airport (SFO). Metrics include Mean Squared Error (MSE) and R-squared (R^2) for training, validation, and test datasets.

Linear Regression w/ Seasons Actual vs. Predicted

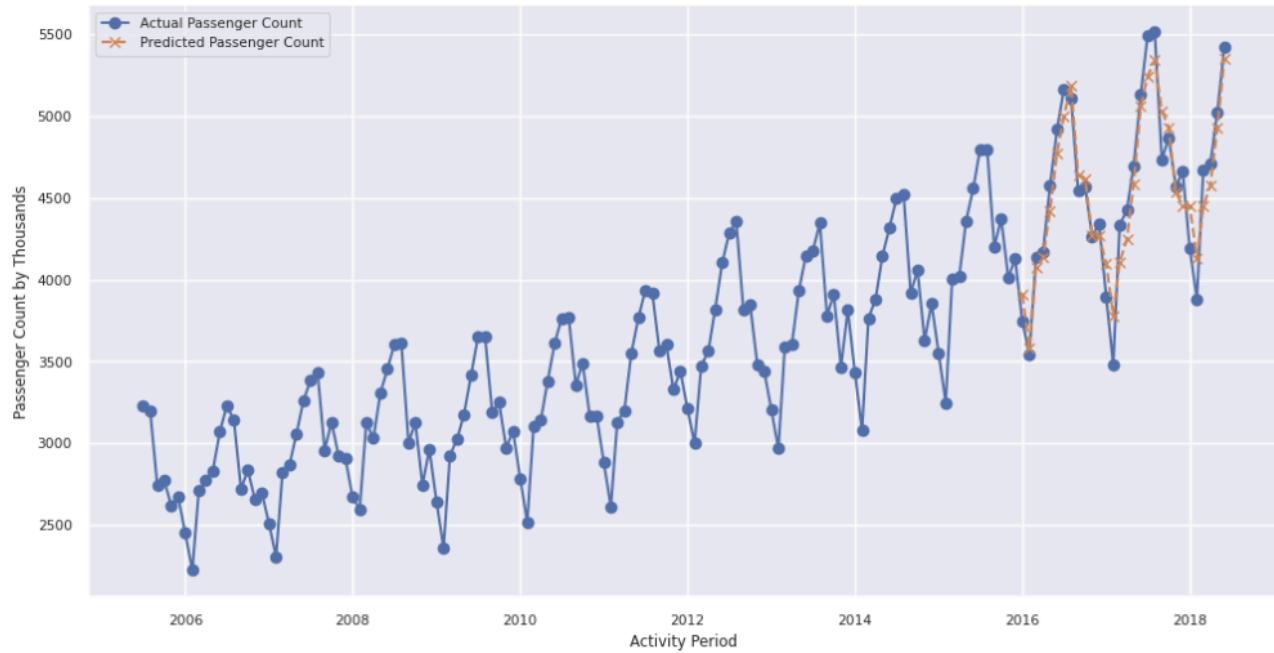


Figure 5. Comparison of Actual vs. Predicted Passenger Count at SFO Using Linear Regression Model Incorporating Seasonal Features. The model predicts passenger count one month ahead, showing close alignment between actual and predicted values over time with an ability to account for peak periods.

Transformer Actual vs. Predicted

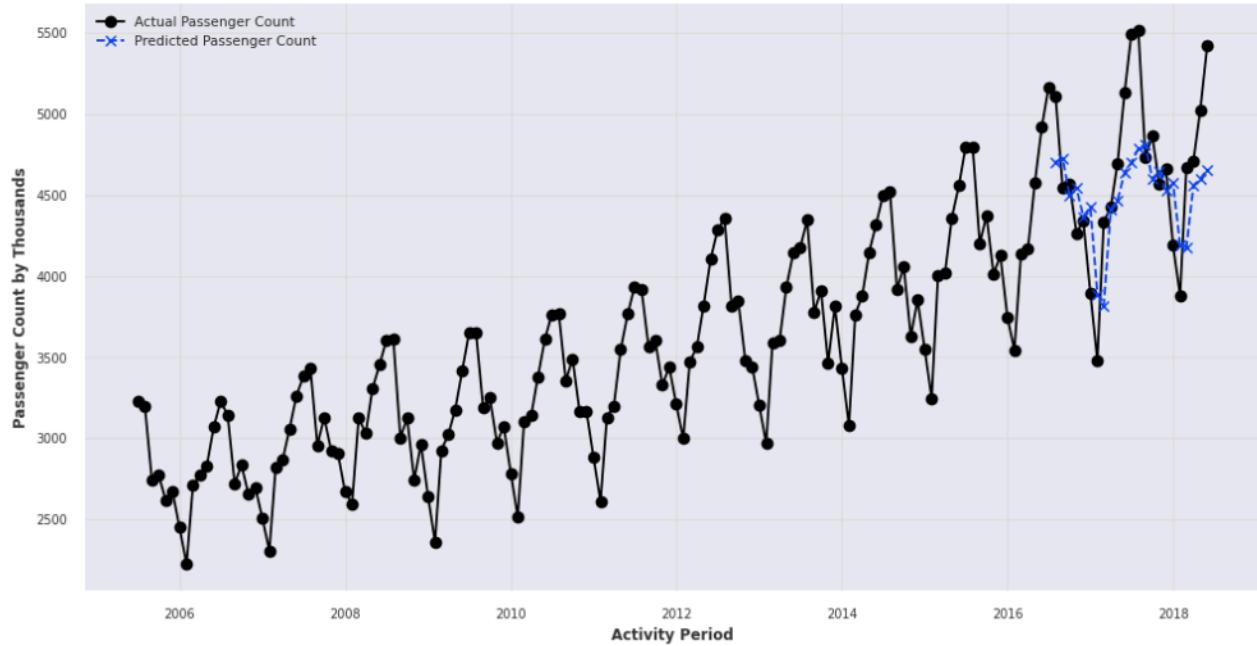


Figure 6. Comparison of Actual vs. Predicted Passenger Count at SFO Using Transformer Model with Dropout Layers on Unseen Data. The model's predictions (in blue) poorly match the actual passenger counts, with significant deviations at peak periods, indicating challenges in capturing seasonal trends and overall patterns.

LSTM w/ no Dropouts Actual vs. Predicted

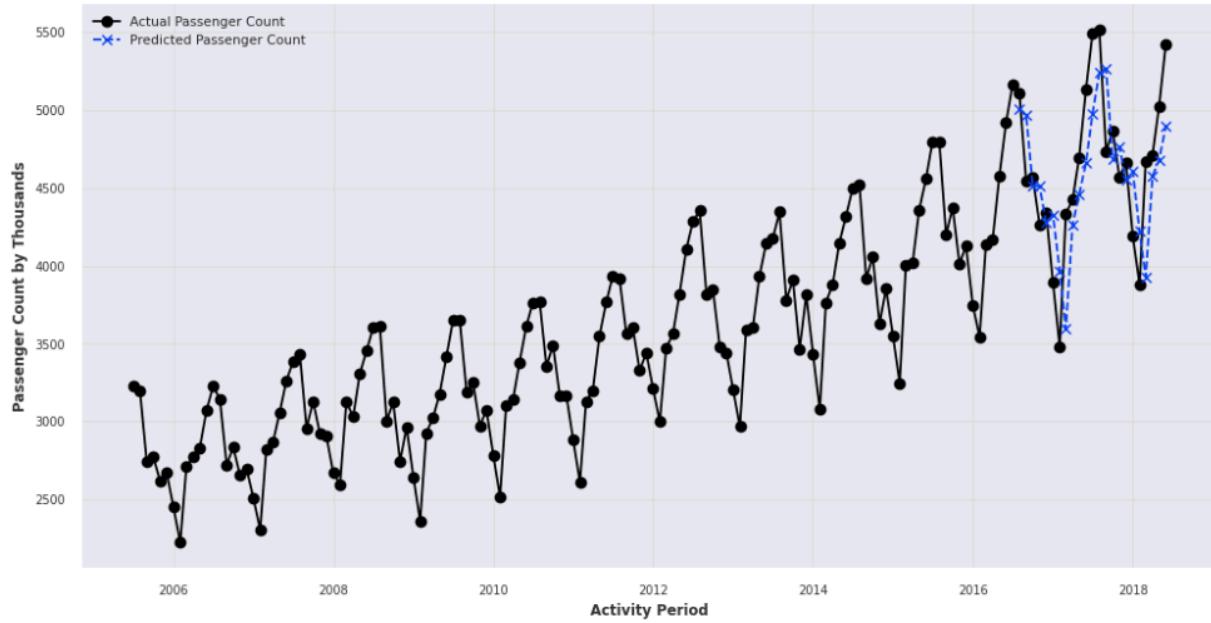


Figure 7. Comparison of Actual vs. Predicted Passenger Count at SFO Using LSTM Model

without Dropout Layers on Unseen Data. The predictions (in blue) show a limited alignment with actual passenger counts, particularly failing to accurately capture peak values, indicating challenges in the model's ability to generalize and predict future trends accurately.

Prophet Actual vs. Predicted

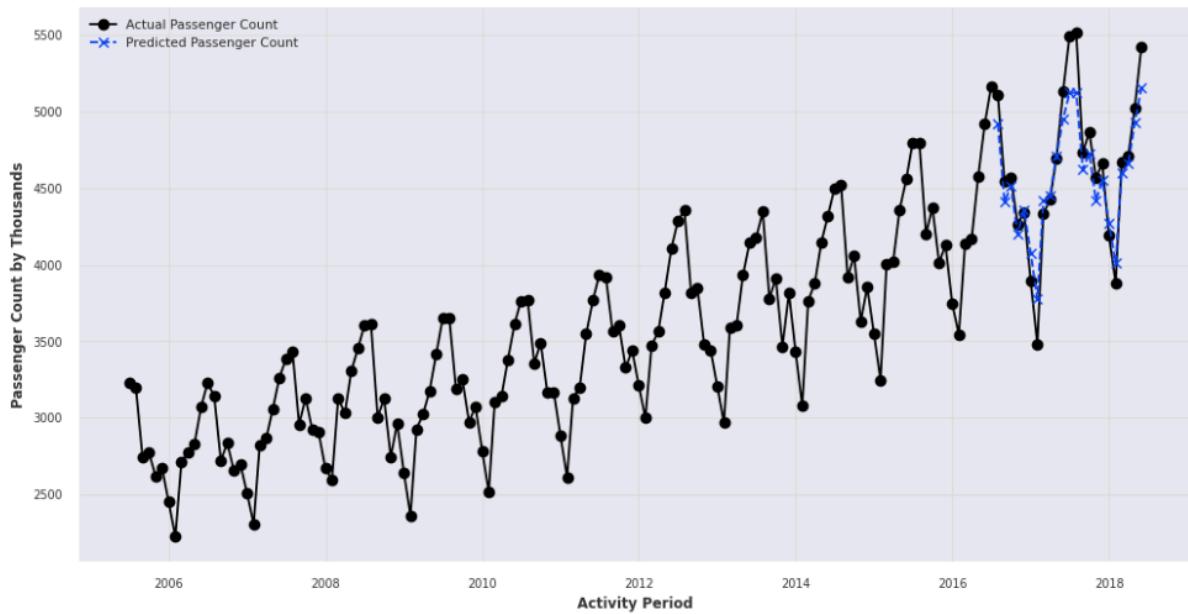


Figure 8. Comparison of Actual vs. Predicted Passenger Count at SFO Using the Pre-Trained Prophet Model on Unseen Data. The predictions (in blue) show some alignment with actual passenger counts, but particularly failing to accurately capture peak values, indicating challenges in the model's ability to generalize and predict future trends accurately.

SARIMA Actual vs. Predicted

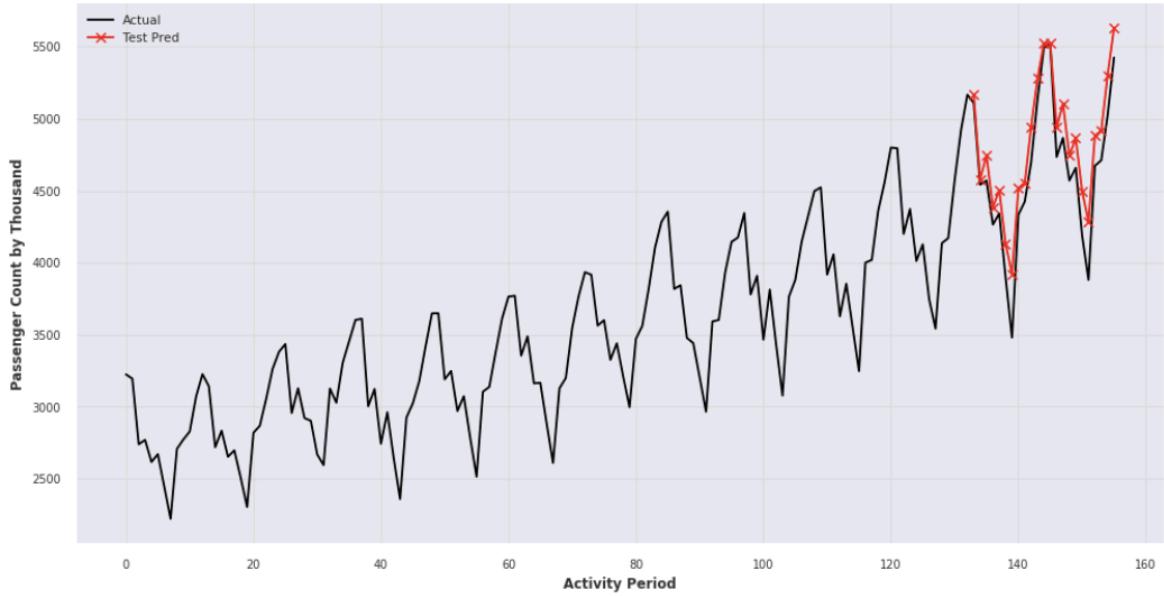


Figure 9. Comparison of Actual vs. Predicted Passenger Count at SFO Using the SARIMA Method on Unseen Data. The predictions (in red) show few alignment with actual passenger counts, particularly failing to accurately capture peak values as it tends to overestimate.

Lung Disease Detection Using Convolutional Neural Networks

Group 5: Isaack Karanja, Reed Oken, Alec Anderson

University of San Diego

AAI-590 Capstone Project

August 12, 2024

Introduction

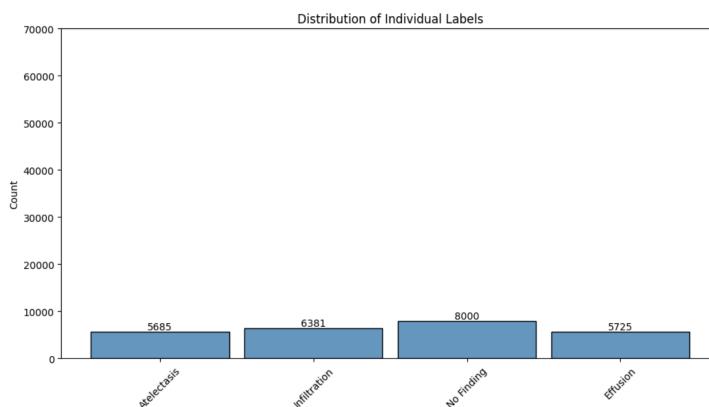
Neural networks excel at classification tasks due to their ability to learn and understand complex patterns. This capability can be applied to many fields, including medical imaging, where notably Convolutional Neural Networks (CNNs) have shown promise in detecting diseases from X-ray images (Varshni et al., 2019). This research paper aims to leverage CNNs in an attempt to accurately classify chest X-rays to multiple lung diseases. Three conditions are studied in this report, including Effusion, Atelectasis, and Infiltration, with “No Finding” included for a baseline comparison. By including “No Finding,” we establish a control group that represents healthy individuals, which improves our approach to distinguish between diseased and healthy cases. We will compare the results of several models including CNNs built from scratch, a CNN designed using a fine-tuned approach, and a pre-trained model.

According to the Pan American Health Organization, chronic obstructive pulmonary disease (COPD) and lower respiratory infections are the fourth and fifth leading causes of death in North and South America (PAHO, 2019). Our goal is to study the early detection of acute respiratory conditions in an effort to curb the prevalence of chronic diseases that can lead to patient mortality. While the three conditions discussed in this paper are not chronic respiratory diseases themselves, they can contribute to or result from chronic respiratory diseases if not properly managed. By classifying these conditions through neural networks, the goal is for healthcare providers involved in lung disease diagnosing to work more effectively and reduce the global burden of respiratory illnesses. This paper serves as an outline for how this could be developed in a real-world setting on X-ray data. For this project, we utilized the National Institutes of

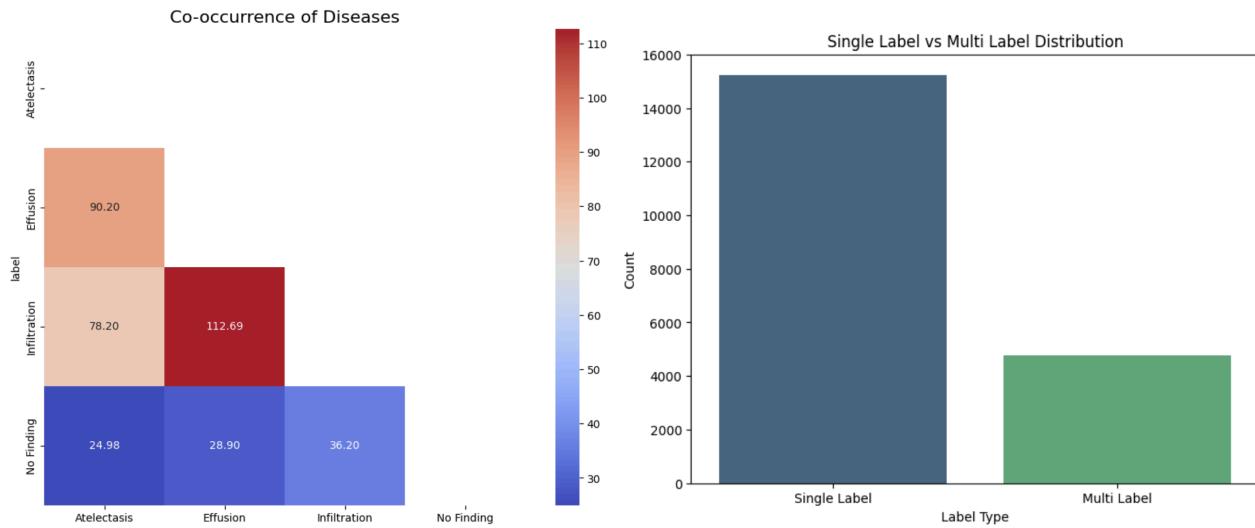
Health (NIH) Chest X-ray Dataset, which includes 112,120 X-ray images that are 1024 x 1024 in pixel size with disease labels from 30,805 unique patients. It should be noted that these X-ray images were annotated using Natural Language Processing with an estimated 90% accuracy (National Institutes of Health, n.d.).

Dataset Summary

The NIH dataset contains images of 14 diseases with varying distributions. This diversity posed a challenge for building a deep learning model, as imbalanced data can lead to biased predictions and poor performance on underrepresented classes. To mitigate these imbalances, we selected well-represented diseases and combined them with 'No Finding.' From the 112,120 images, we created a sample of 8,000 'No Finding' images and 4,000 images for each disease, plus additional multi-label occurrences. This approach aligned with the dataset's original distribution by including a higher 'No Finding' count while maintaining sufficient diseased cases for training. Our primary objective was to balance the data sufficiency, include multi-label observations, and factor in computational constraints. Overall, this sample represents 23% of the entire dataset and reduced the data by roughly one-fourth the original size.

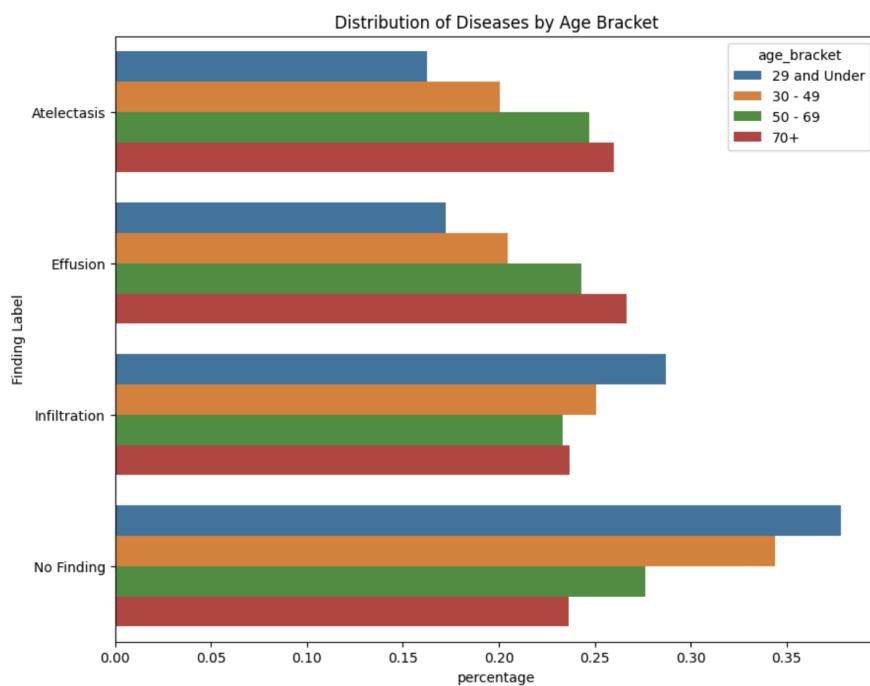


Approaching this project as a multi-label classification problem is inherently more challenging than single-label classification. In single-label classification, each instance is associated with only one label, whereas in multi-label classification, an observation can belong to multiple classes simultaneously (Bogatinovski et. al., 2021). For example, we noted a higher number of Infiltration with Effusion cases, and a higher number of Atelectasis with Effusion cases, compared to other co-occurrences. The distribution of these co-occurrences and the number of single label versus multiple labels are shown in the following figures:



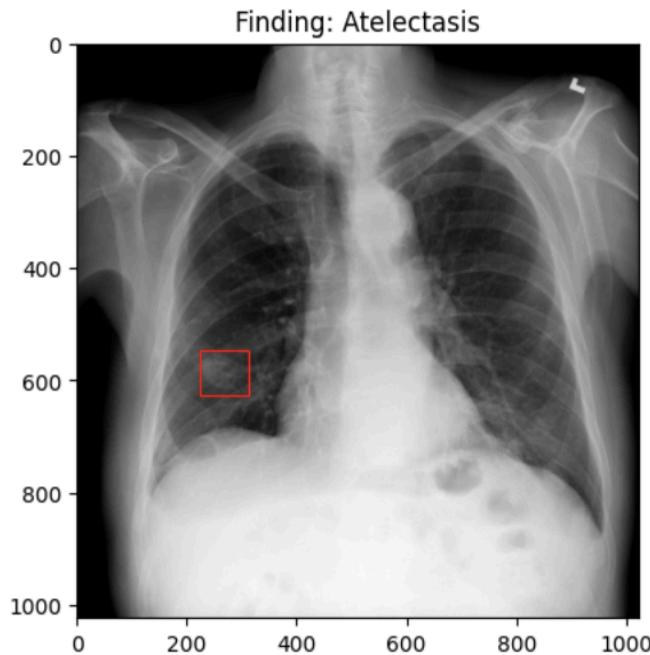
During the initial data preprocessing phase, we transformed the raw data through encoding categorical variables and label binarization for multi-label targets. Several variables were included in this dataset, notably the Image Index, Finding Label, Image Width and Height, Image Pixel Spacing, and Image Path. Additional variables such as patient age and gender were also visualized for their distributions. We observed a nearly equal distribution of male and female patients, which ensures our models are sufficiently exposed to X-ray images from both male and female patients during training,

preventing bias and simplifying the model. This reduced the overall complexity by not having to adjust our model to factor imbalances in gender. Similarly, all age groups in the dataset had a prevalence of Atelectasis, Effusion, and Infiltration. The consistency of diseases in both younger and older patients confirmed they were suitable choices for our models. These findings allowed us to narrow the scope of our research and ensure the models are focused primarily on the visual characteristics of our X-rays.



The dataset included a total of 457 X-ray images that include bounding boxes. Bounding boxes are critical in the context of medical image analysis as they delineate the regions of interest (ROIs) within an X-ray, highlighting areas where anomalies, such as signs of disease, are present (Gunnell, 2024). These annotated regions are essential for training and focusing on the precise areas that are clinically relevant, improving the model's accuracy in detecting and classifying diseases. However, the limited number of X-rays with bounding boxes presents a significant limitation to our project. This restricts

the amount of learning that can be applied to the exact localization of the diseases, potentially affecting the model's ability to generalize and accurately detect conditions in new, unannotated X-rays. Below is a sample image from our dataset, illustrating a bounding box highlighting an area of Atelectasis:



Minimal data augmentation techniques were applied to preserve the underlying characteristics of the chest X-ray images while still enhancing the model's generalization capability. The augmentation includes subtle transformations such as random rotation, contrast adjustment, translation, and zoom, each with a very low intensity. This careful approach ensures that the essential diagnostic features of the X-ray images remain intact, while also providing the model with a slightly varied dataset to improve its robustness against small changes in the input images.

Background Information

Studying Effusion, Atelectasis, and Infiltration poses a substantial challenge in medical diagnoses due to their overlapping symptoms and complex manifestations in chest X-rays. Effusion, the abnormal accumulation of fluid in the pleural space, can compromise respiratory function and is often a sign of underlying systemic diseases such as heart failure (Krishna, Antoine, & Rudrappa, 2024). Atelectasis, the collapse or incomplete expansion of lung tissue, can lead to severe respiratory distress if not promptly addressed (Johns Hopkins Medicine, n.d.). Infiltration, characterized by the presence of substances like fluid or cells within the lung parenchyma, is frequently associated with infections like pneumonia (Core Concepts in Clinical Infectious Diseases (CCCID), 2016). By selecting these diseases, we aimed to leverage CNNs and transfer learning in order to identify these conditions individually while also recognizing their co-occurrence.

Transfer learning is a technique widely used in medical imaging that improves performance on models with limited data. Salehi et. al. (2023) address the challenges of small datasets and computational limitations in *A Study of CNN and Transfer Learning in Medical Imaging*. By reusing models that are pre-trained on large datasets, such as ImageNet or our NIH Chest X-Ray dataset, transfer learning can reduce the need for extensive labeled data. This can improve accuracy, shorten training time and manage class imbalances (Salehi et. al., 2023). However despite these advantages, transfer learning may still transfer biases from pre-trained models, and requires significant computational resources for fine-tuning.

CNNs are exceptionally well-suited to this medical imaging project due to their inherent strengths in feature extraction, spatial awareness, and efficient handling of visual data. As demonstrated in work by Krizhevsky et al. (2012), CNNs excel at automatically learning and extracting intricate features from images, crucial for identifying subtle patterns and anomalies in medical images like X-rays. Their hierarchical structure enables the capture of both low-level features (e.g., edges, textures) and high-level features (e.g., organ shapes, disease-specific patterns), making them highly effective for detecting and classifying diseases within the NIH Chest X-ray Dataset (White et al., 2023). Furthermore, CNNs preserve the spatial relationships between pixels, a critical aspect of medical image analysis where the location and arrangement of features are essential for accurate diagnosis. For instance, in chest X-rays, the spatial context of the heart, lungs, and blood vessels is crucial for identifying abnormalities (Krizhevsky et al., 2012). CNNs retain this context throughout the analysis, leading to more accurate and reliable results.

Several studies have demonstrated the effectiveness of building Convolutional Neural Networks (CNNs) from scratch using large medical datasets. A notable study by Dahmane et. al. (2019) involved a hybrid approach combining a CNN built from scratch with a pre-trained VGG19 model to detect pneumonia in chest X-ray images. Their custom-designed CNN architecture, when combined with the VGG19 model for feature extraction, achieved over 99% accuracy on the Guangzhou Women and Children's Medical Center dataset. This hybrid approach, which leverages both the strengths of pre-trained models and custom-designed CNNs, is something we explored in our research to further optimize our model's performance.

In the context of neural architecture search (NAS) and hyperparameter optimization, utilizing a combination of Keras Tuner and random search is an effective strategy for refining convolutional neural networks (Cetiner & Metlek, 2023). Keras Tuner automates the exploration of neural network architectures and hyperparameters, not only streamlining the model development process but also leading to quicker optimization and performance improvements. By leveraging random search, this approach provides a more efficient alternative to exhaustive methods like grid search, as it samples randomly within a predefined hyperparameter space and enables the discovery of near-optimal configurations with fewer trials (Cetiner & Metlek, 2023). This is particularly advantageous when dealing with complex tasks such as multi-label classification, where the search space can be vast and computationally demanding.

Experimental Methods

Three methods were explored in our classification project, including an iterative CNN built from scratch, a CNN optimized through Neural Architecture Search (NAS) using Keras Tuner, and a transfer learning approach using the VGG16 model. We will evaluate the effectiveness of these models in diagnosing different lung conditions and to understand their strengths and limitations.

Custom CNN Models:

The initial custom CNN was designed with an architecture consisting of three convolutional layers followed by max-pooling operations. A flatten layer was then used to transform multi-dimensional feature maps into a one-dimensional vector, followed by

two dense layers. ReLU activation functions were employed in all layers except the final one, which used a sigmoid activation function to output probabilities for each of the four diagnostic classes: ‘No Finding,’ ‘Infiltration,’ ‘Effusion,’ and ‘Atelectasis.’ The Adam optimizer was chosen for its strong performance across deep learning tasks, and the dataset was split into training, validation, and test sets with an 80%-10%-10% ratio. The model was trained with a batch size of 18 due to computational constraints.

During the development process, several iterations of this custom CNN were explored. The initial model, which did not include any regularization, was followed by versions incorporating dropout and varying degrees of L2 regularization to address overfitting. We then experimented with increasing model complexity by adding more layers and units and later chose a best-performing model for further evaluation. This approach allowed us to refine the model architecture and identify the most effective regularization techniques.

Neural Architecture Search (NAS) with Keras Tuner:

To test additional CNN architectures, we employed NAS using Keras Tuner (Keras, n.d.). This tool allowed us to efficiently explore various hyperparameters and architectural choices. The search space included the number of convolutional and dense layers, units in each layer, activation functions, dropout rates, and learning rates. The NAS-optimized CNN was trained using a binary cross-entropy loss function, with validation accuracy as the primary evaluation metric. The training process involved monitoring validation loss for early stopping to prevent overfitting, with the Keras

'ModelCheckpoint' tool utilized to save the best-performing models based on validation loss.

Transfer Learning with VGG16:

In addition to the previous CNNs, we explored transfer learning using the pre-trained VGG16 model (Simonyan & Zisserman 2015). The VGG16 model, with its weights pre-trained on the ImageNet dataset, was used as a feature extractor. The original classification layers of VGG16 were removed, and a custom classifier head was added, consisting of a flatten layer, a dropout layer, two dense layers with ReLU activation, and a final sigmoid output layer for multi-label classification. This approach leveraged the feature representations learned by VGG16 on a massive dataset, which was particularly useful given the smaller size of our chest X-ray sample.

All models were trained using the binary cross-entropy loss function with a threshold of 0.3, which is a common loss function for multi-label classification problems. This lower threshold was chosen to improve recall, which is crucial in medical diagnoses where identifying all possible cases of a condition may be more critical than avoiding false positives (Folio3AI, n.d.). Accuracy, F1 and AUC-ROC scores were used as evaluation metrics, with AUC-ROC scores included to understand model performance across different labels. The training procedure involved monitoring validation loss for early stopping, and experiments with larger image sizes (e.g., 512 x 512 and 300 x 300) showed potential for improved performance, though they required significantly more memory. While grayscale conversion was also explored, it did not

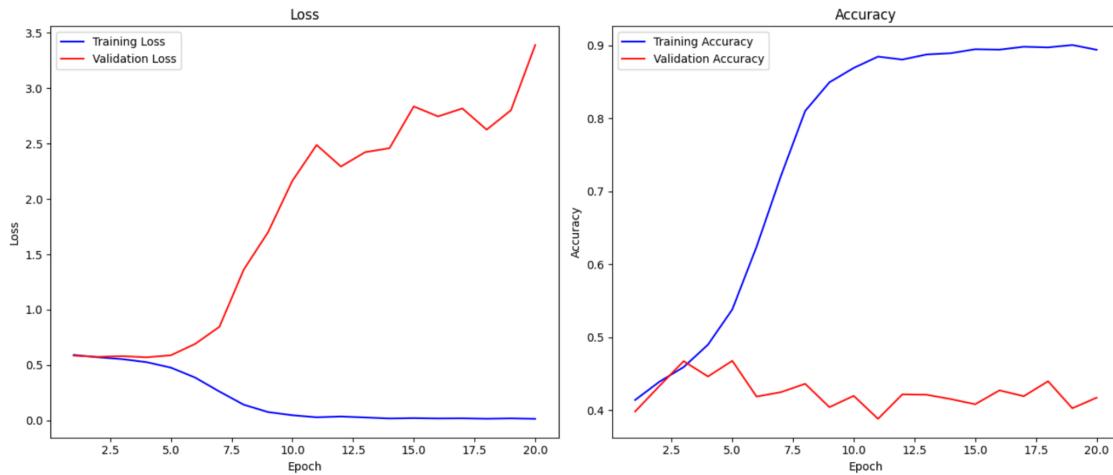
yield noticeable benefits compared to using RGB images, leading to the decision to use RGB images with a resolution of 256x256 for the final models.

In summary, each model variant offered insights into balancing complexity, regularization, and data processing in the context of multi-label classification of lung diseases. The combination of custom-built CNNs, NAS-optimized models, and transfer learning approaches provided a comprehensive evaluation of different strategies for this classification task.

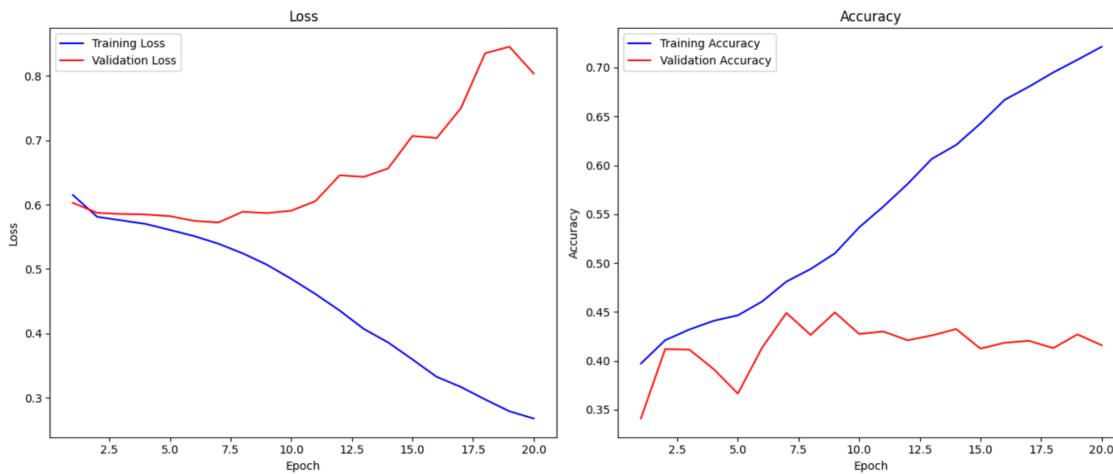
Results

Our initial modeling of the chest X-ray classification task utilized the basic Convolutional Neural Network (CNN) without any regularization techniques. This model achieved noticeable improvements in training accuracy and reductions in loss over successive epochs. However, the validation accuracy did not mirror this trend and the validation loss began to increase after a few epochs, signaling overfitting. This discrepancy between training and validation metrics was visualized in the training and validation accuracy/loss curves, where the training loss consistently decreased while the validation loss initially decreased but then began to rise. The training accuracy improved steadily, whereas validation accuracy exhibited very minimal improvements,

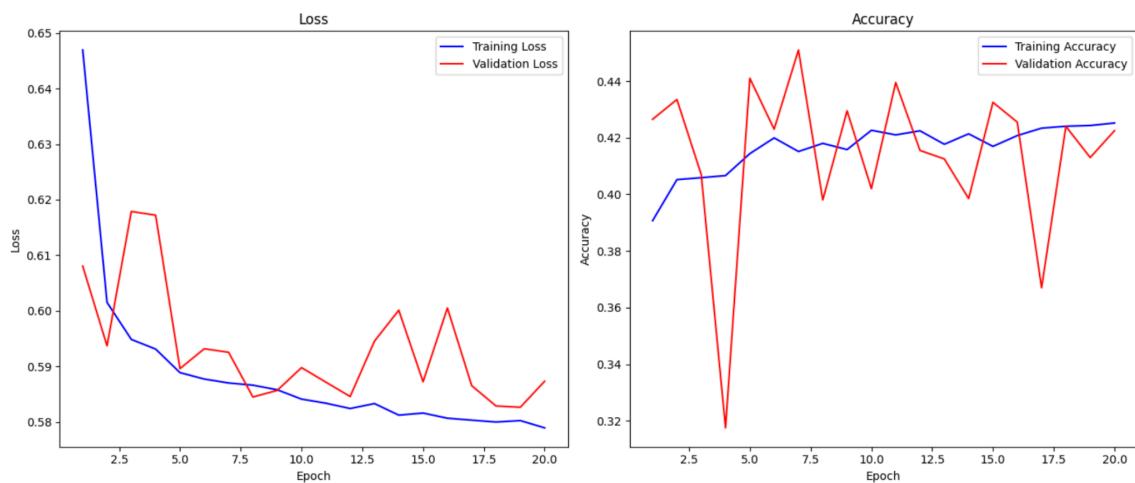
confirming the problem of overfitting that we needed to address.



In response to the overfitting observed in the initial model, dropout regularization was introduced. The inclusion of dropout regularization resulted in a slightly more stable model with minor improvements in validation accuracy. However, overfitting was still evident after several epochs, as indicated by the figure below. While training loss continued to decrease and training accuracy increased, validation loss did not show a corresponding decline, highlighting that overfitting is a persistent challenge in our analysis.

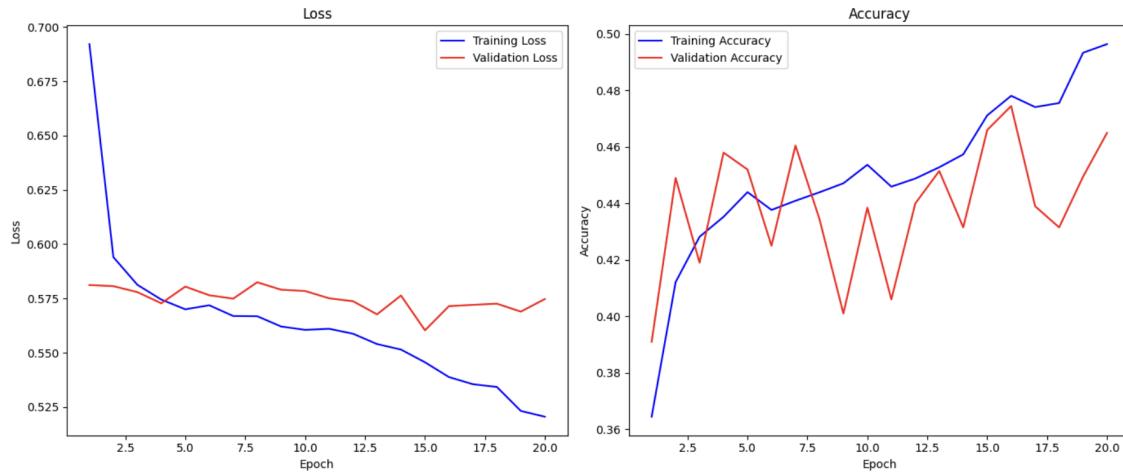


After trying different techniques to improve our model, we used L2 regularization to reduce overfitting. The validation loss and accuracy both plateaued and it was evident the model was overly constrained. We then tried using a weaker form of L2 regularization, shown below. This approach resulted in a slight improvement. The validation loss decreased and was improving at generalizing to new data, and the accuracy stayed within a range that indicated better performance than previous models. However, these changes were not enough to significantly improve the model's overall performance. This indicated that we needed to try other methods in addition to dropout and regularization.



We then introduced additional layers to increase model complexity and applied dropout regularization. These changes aimed to improve generalization by forcing the model to develop more robust features, thus increasing performance with reduced overfitting. The updated model showed improved validation accuracy for nearly all epochs and reached a test accuracy of 45.3%, and a test loss of 0.56. Overall, our iterative approach with CNNs built from scratch was promising, however further

refinements techniques using model optimization tools were needed.



Keras Tuner was then used in combination with RandomSearch to optimize the structure and hyperparameters of a CNN model. The model-building function adjusted key hyperparameters such as the number of units in the convolutional layers, the type of activation functions, the dropout rates, and the learning rate. Our objective was to minimize validation loss with early stopping, and model checkpointing to prevent overfitting and to save the best-performing model, respectively. The search was conducted over 15 trials, each comprising a maximum of 10 epochs.

The optimal hyperparameters determined by Keras Tuner were as follows:

- 64 units in the first convolutional layer
- 3 convolutional layers
- 128 units in the dense layer
- A learning rate of 0.001

Using these optimal hyperparameters, the model was trained and evaluated. The model achieved a test accuracy of 47.65% and a test loss of 0.552. Additionally, the

AUC-ROC scores showed decent discriminatory power, particularly for ‘Infiltration’. The macro and micro average AUC-ROC scores were 0.7227 and 0.7357, respectively.

The pretrained VGG16 model was then employed to enhance classification performance. Known for its robustness in image classification tasks, VGG16 was adapted to accept grayscale input by adding a layer to convert single-channel images to three channels. This model served as a feature extractor with additional layers appended for final classification. Training over 20 epochs with early stopping and checkpointing ensured optimal performance, resulting in a test accuracy of 49.75% and a test loss of 0.52. The classification report showed improvements in precision and F1 scores. AUC-ROC scores were higher across all classes compared to the Keras Tuner optimized model.

Lastly, we leveraged the strengths of our custom-built CNN, the CNN optimized using Keras Tuner, and the VGG16 models by employing a weighted average ensemble approach. This method combined predictions from the best-performing models, resulting in a slight improvement in overall performance metrics. The ensemble achieved an average AUC-ROC of 0.76, matching the performance of the VGG16 model, and demonstrated a balanced improvement across precision, recall, and F1 scores. Notably, the ensemble approach yielded the highest F1 score and AUC-ROC, particularly for ‘Effusion.’

Precision	No Finding	Infiltration	Effusion	Atelectasis	Average Precision Score
Custom CNN	0.44	0.55	0.42	0.51	0.48
Keras Tuner	0.42	0.53	0.39	0.51	0.46
VGG16	0.47	0.6	0.41	0.57	0.51
Weighted Ensemble	0.46	0.6	0.42	0.56	0.51

Recall	No Finding	Infiltration	Effusion	Atelectasis	Average Recall Score
Custom CNN	0.64	0.65	0.56	0.87	0.68
Keras Tuner	0.72	0.74	0.73	0.89	0.77
VGG16	0.71	0.83	0.69	0.81	0.76
Weighted Ensemble	0.71	0.83	0.7	0.84	0.77

F1 Score	No Finding	Infiltration	Effusion	Atelectasis	Average F1 Score
Custom CNN	0.52	0.60	0.48	0.64	0.56
Keras Tuner	0.53	0.62	0.51	0.65	0.58
VGG16	0.56	0.70	0.52	0.67	0.61
Weighted Ensemble	0.56	0.70	0.53	0.67	0.62

AUC-ROC	No Finding	Infiltration	Effusion	Atelectasis	Average AUC-ROC Score
Custom CNN	0.72	0.80	0.62	0.73	0.72
Keras Tuner	0.72	0.82	0.62	0.73	0.72
VGG16	0.76	0.87	0.65	0.77	0.76
Weighted Ensemble	0.76	0.87	0.65	0.77	0.76

Conclusion

Model Performance and Problem Application

While the results in this research show promise, they also indicate that further improvements are necessary before the model can be reliably applied in clinical settings for chest X-ray classification. The current performance level suggests that our model could potentially assist radiologists by providing an initial screening or a second opinion, but it is not yet sufficiently accurate for autonomous diagnosis. Deep learning models for medical imaging need to achieve high accuracy, and our dataset only consists of frontal radiographs and as Rajpurkar et al. (2017) noted, up to 15% of accurate diagnosis

require lateral view which limits this model's reliability for practical clinical application.

Our model's performance, while encouraging, underscores the need for further refinement to meet stringent clinical requirements. The AUC-ROC scores, particularly for 'Infiltration', demonstrate the model's ability to discriminate between classes. However, the overall accuracy indicates that misclassifications are still frequent. In a clinical context, this could lead to both false positives and false negatives, which could have significant implications for patient care if not properly managed.

Unexpected Results

The most notable unexpected result was the significant performance boost achieved by both the ensemble model and VGG16 pretrained model, compared to individual models. This finding aligns with recent research by Zhang et al. (2020), which suggests that ensemble methods can capture complementary features from different architectures, leading to more robust predictions. This unexpected synergy between the VGG16, custom-CNN, and Keras Tuner models highlights the potential of combining transfer learning (through VGG16) with custom-optimized architectures. It suggests that future work in medical image classification could benefit from exploring diverse ensemble combinations rather than focusing solely on optimizing individual models.

Future Work and Deployment

Further work could include a multi-faceted approach encompassing model optimization, data enhancement, and deployment. For model optimization, we aim to explore more advanced CNN architectures such as ResNet, which have demonstrated

superior performance in image classification tasks. We also plan to implement attention mechanisms to help the model focus on the most relevant parts of the chest X-rays, potentially improving both accuracy and interpretability (Guan et al., 2018). Given the multi-label nature of our dataset, investigating multi-task learning approaches for simultaneous detection of multiple conditions could significantly enhance overall performance (Ruder, 2017). In terms of data enhancement, we intend to acquire a more diverse dataset with bounding boxes to improve the model's generalization capabilities. We will also enhance our data augmentation techniques, incorporating methods like color jittering and noise injection to increase model robustness (Shorten & Khoshgoftaar, 2019). Additionally, we plan to incorporate clinical metadata to supplement image data, providing additional context that could enhance classification accuracy (Baltruschat et al., 2019).

Regarding deployment, we will develop a user-friendly interface for radiologists to easily input X-ray images and receive model predictions. This will be supported by robust data pipelines for real-time processing of X-ray images in a clinical setting. To ensure compliance with healthcare data regulations (e.g., HIPAA), we will implement appropriate data security and privacy measures. Extensive clinical validation studies will be conducted to assess the model's performance in real-world scenarios. Post-deployment would involve establishing a monitoring system for continuous evaluation of model performance and implementing a feedback mechanism for radiologists to report and help correct misclassifications, enabling ongoing model improvement. Developing clear guidelines for integrating the model's predictions into clinical workflows would be required, emphasizing its role as a supportive tool rather

than a replacement for expert judgment. These steps align with best practices for deploying machine learning models in healthcare settings, as outlined by He et al. (2019). We aim to develop a more accurate, robust, and clinically applicable model for chest X-ray interpretation support.

Works Cited

National Institutes of Health Chest X-ray Dataset. (n.d.). *NIH Chest X-rays*. Kaggle. Retrieved from <https://www.kaggle.com/datasets/nih-chest-xrays/data>

Simonyan, K., & Zisserman, A. (2015). *Very deep convolutional networks for large-scale image recognition*. In International Conference on Learning Representations (ICLR) <http://arxiv.org/abs/1409.1556v6>

Keras, (n.d.) *KerasTuner*. Retrieved from https://keras.io/keras_tuner/

Varshni, D., Nijhawan, R., Thakral, K., Mittal, A., & Agarwal, L. (2019). *Pneumonia detection using CNN based feature extraction*. In IEEE Xplore. <https://doi.org/10.1109/ACCESS.2019.2962842>

Baltruschat, I. M., Nickisch, H., Grass, M., Knopp, T., & Saalbach, A. (2019). *Comparison of deep learning approaches for multi-label chest X-ray classification*. Scientific Reports, 9(1), 6381. <https://doi.org/10.1038/s41598-019-42294-8>

Guan, Q., Huang, Y., Zhong, Z., Zheng, Z., Zheng, L., & Yang, Y. (2018). *Diagnose like a radiologist: Attention guided convolutional neural network for thorax disease classification*. arXiv preprint <https://doi.org/10.48550/arXiv.1801.09927>

Cetiner, H., & Metlek, S. (2023). CNNTuner: *Image Classification with A Novel CNN Model Optimized Hyperparameters*. Bitlis Eren Üniversitesi Fen Bilimleri Dergisi, 12(3), 746-763. <https://doi.org/10.17798/bitlifsen.1294417>

Johns Hopkins Medicine. (n.d.). *Atelectasis*. <https://www.hopkinsmedicine.org/health/conditions-and-diseases/atelectasis>

Krishna, R., Antoine, M. H., & Rudrappa, M. (2024). *Pleural effusion*. In *StatPearls*. StatPearls Publishing. <https://www.ncbi.nlm.nih.gov/books/NBK459262/>

Core Concepts in Clinical Infectious Diseases (CCCID). (2016). *Lung infiltrate*. ScienceDirect. <https://www.sciencedirect.com/topics/nursing-and-health-professions/lung-infiltrate>

He, J., Baxter, S. L., Xu, J., Xu, J., Zhou, X., & Zhang, K. (2019). *The practical implementation of artificial intelligence technologies in medicine*. Nature Medicine, 25(1), 30-36. <https://doi.org/10.1038/s41591-018-0307-0>

Gunnell, M. (2024, April 6). *Bounding box*. Techopedia. <https://www.techopedia.com/definition/bounding-box>

Rajpurkar, P., Irvin, J., Zhu, K., Yang, B., Mehta, H., Duan, T., ... & Ng, A. Y. (2017). *Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning*. arXiv <https://doi.org/10.48550/arXiv.1711.05225>

Bogatinovski, J., Todorovski, L., Džeroski, S., & Kocev, D. (2021). *Comprehensive comparative study of multi-label classification methods*. IEEE.

Ruder, S. (2017). *An overview of multi-task learning in deep neural networks*. arXiv <https://doi.org/10.48550/arXiv.1706.05098>

Shorten, C., & Khoshgoftaar, T. M. (2019). *A survey on image data augmentation for deep learning*. Journal of Big Data, 6(1), 60. <https://doi.org/10.1186/s40537-019-0197-0>

Zhang, Y., Yang, Q., & Zhang, Z. (2020). *An overview of multi-task learning in deep neural networks*. arXiv <https://doi.org/10.48550/arXiv.1706.05098>

Salehi, A. W., Khan, S., Gupta, G., Alabdullah, B. I., Almjally, A., Alsolai, H., Siddiqui, T., & Mellit, A. (2023). *A study of CNN and transfer learning in medical imaging: Advantages, challenges, future scope*. Sustainability, 15(7), 5930. <https://doi.org/10.3390/su15075930>

Folio3AI. (n.d.). *Explaining precision and recall in machine learning*. Folio3AI. <https://www.folio3.ai/blog/precision-and-recall-in-machine-learning/>

Hou, G., Qin, J., Xiang, X., Tan, Y., & Xiong, N. N. (2021). *AF-Net: A medical image segmentation network based on attention mechanism and feature fusion*. Computers, Materials & Continua, 69(2), 1878-1891. <https://doi.org/10.32604/cmc.2021.017481>

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). *ImageNet classification with deep convolutional neural networks*. Communications of the ACM, 60(6), 84-90. <https://doi.org/10.1145/3065386>

White, C., Safari, M., Sukthanker, R., Ru, B., Elsken, T., Zela, A., Dey, D., & Hutter, F. (2023). *Neural architecture search: Insights from 1000 papers*. arXiv:2301.08727 [cs.LG]. <https://doi.org/10.48550/arXiv.2301.08727>

Dahmane, O., Khelifi, M., Beladgham, M., & Kadri, I. (2021). *Pneumonia detection based on transfer learning and a combination of VGG19 and a CNN built from scratch*. Indonesian Journal of Electrical Engineering and Computer Science, 24(3), 1469-1480. <https://doi.org/10.11591/ijeecs.v24.i3.pp1469-1480>

A.S.Linguist Final Project

Shyam Adhikari, Caterina Gallo, Paul Thai

Shiley-Marcos School of Engineering, University of San Diego

AAI-590 - Capstone Project

Professor Anna Marbut

August 12, 2024

Githib link:

<https://github.com/Lyfae/AAI-590-Capstone.git>

Introduction

This project aims at developing an application able to elaborate live sign language questions and return both textual and sign language answers on general topics, thereby offering communication support to deaf and dumb individuals.

It has been estimated that about 72 million people worldwide rely on sign language, which represents the most common choice of communication among those suffering from hearing and/or speaking impairments (Vaidhya and Deiva Preetha, 2022). However, there are more than 300 different sign languages in the world (Sign Language, n.d.) and each of them is known and used by small groups of people only. This can seriously limit the possibilities and opportunities of people suffering from this kind of disability, who usually need to be accompanied by family members or educators able to understand their needs and translate them to the external world. Thus, the ultimate goal of our project is to develop a technology responding to the communication needs of deaf and dumb people who know the American Sign Language (ASL), which is the dominant sign language of deaf communities in North America (American Sign Language, 2021). The application we propose could in fact be beneficial for ASL users, to both communicate with people who do not know their language and have simple dialogues with a chatbot.

For this project, we developed two different machine learning models: a convolutional neural network (CNN) serving as sign language interpreter and a chatbot model to hold on textual conversations. Thus, we adopted two different datasets. The first one is an ASL alphabet dataset from Kaggle (ASL Alphabet, 2021) and was used to train the CNN model. The second dataset, which is also from Kaggle, contains conversations for building a chatbot (3K

Conversations Dataset for ChatBot, n.d.) and was employed to fine-tune a Flan-T5-Base chatbot model (google/flan-t5-base, n.d.).

Putting everything together, we created an application capable of interacting with the two models. A webcam grabs the sign-language gestures from the user, which are then passed to the CNN model to interpret the different inputs. These are combined together to form a string of text feeding into the chatbot model, which in turn returns the response back for the user to see. As an additional option, our application also has the ability to display the gestures from the chatbot's responses as a succession of ASL images. In shorter terms, this project can be thought of as an interactive chatbot and a two way interpreter, from sign language to text and vice versa.

Data Summary

ASL Alphabet Dataset

The ASL alphabet dataset used to train the CNN model consists of 29 unique folders containing images, each averaging about 12-13 KB in size. Of these, 26 folders correspond to the letters of the US alphabet and the remaining three represent "space", "delete" and "nothing." Figure 1 shows one example of image for each of the 29 possible classes in the training set. The dataset appears balanced, with 3000 images per label and no missing data issues, that suggests no major data collection biases.

The extracted variables from the ASL alphabet dataset, mean pixel intensity and standard deviation, directly relate to the project goal of image classification for ASL signs. Mean pixel intensity provides a measure of the overall brightness of an image, while standard deviation measures the variability within the pixel values. As shown in Figure 2, the correlation analysis from the ASL alphabet dataset revealed a weak positive correlation between mean pixel intensity and standard deviation. This relationship tells us that, while there is some dependency, the

variables are distinct enough to contribute independently to the predictive model, reducing multicollinearity concerns in model training.

Figure 1

Examples of images in the training set of the ASL alphabet dataset for all the 29 possible classes

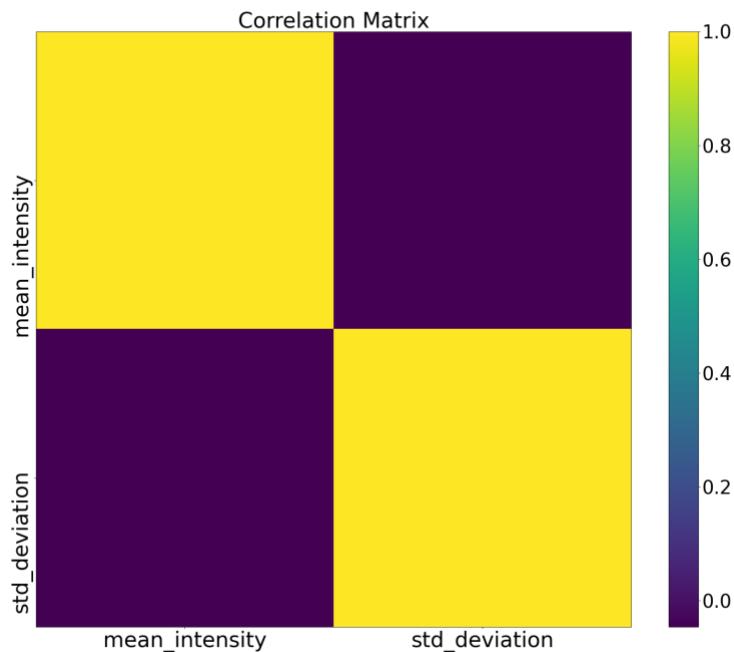


Note. The “del” label in the second panel of the last row stands for “delete”.

In order to address potential overfitting and improve model generalization, feature scaling and data augmentation were applied before model training. Training ASL images were rescaled (by dividing pixel values by 255) and converted to grayscale, and data augmentation techniques, such as rotation, horizontal and vertical shifting, shearing, zooming and horizontal flipping, were employed.

Figure 2

Correlation between mean pixel intensity and standard deviation for the images of the ASL alphabet dataset



Conversations Dataset

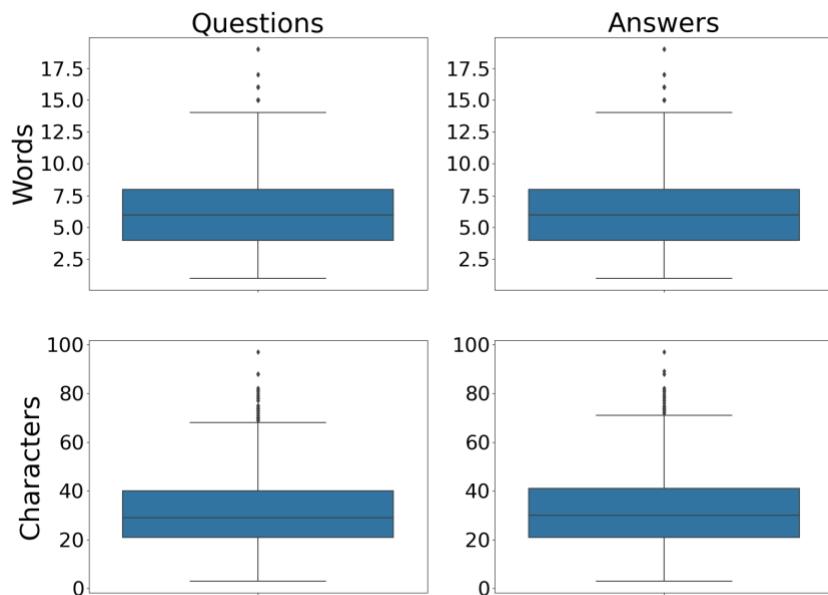
The conversations dataset used to fine-tune the pre-trained chatbot model is stored as a CSV file with two columns: one for questions and the other for answers, both containing data of the string datatype. It includes 3725 questions and answers and no missing values.

As illustrated in the boxplots of Figure 3, there are 6.35 ± 2.80 / 6.52 ± 2.91 words and 31.26 ± 13.95 / 32.21 ± 14.54 characters in questions / answers. The number and type of the top 20 most common words in questions and answers are also shown in the bar plots of Figure 4. Since we did not remove stopwords from the list of questions and answers, it is reasonable to find that

pronouns, auxiliaries, verbs and articles are the most frequent words in the conversations datasets.

Figure 3

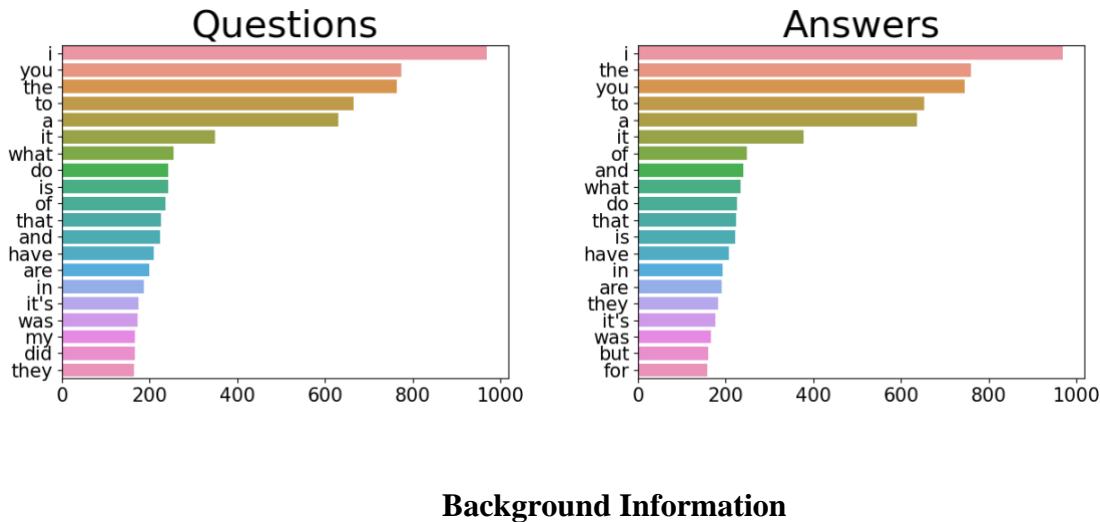
Boxplots of the number of words and characters in the questions and answers of the conversations dataset



During text preprocessing, we applied some modifications to both questions and answers. We first removed contractions, by modifying expressions like “i’ve” with “I have” or “i’m” with “I am”, eliminated punctuation and finally applied lowercasing. Regarding tokenization, which is a common preprocessing step while building a natural language model, it was implemented through the tokenizer of the pre-trained model we chose for the project, that is Flan-T5-Base.

Figure 4

Top 20 most common words in the questions and answers of the conversations dataset



Background Information

Other groups working in both academic research (Berkley School of Information, n.d.; Huang et al., 2021; Parsdani et al., 2018; Voinea, 2017) and commercial projects (Aquino, 2023; Lenovo StoryHub, 2023) already developed an AI application similar to ours. The main differences among the different projects mainly depend on (i) the type of gesture recognition the system is able to perform, (ii) the format of the outputs returned from the chatbot, (iii) the kind of topics the chatbot can hold a conversation on, and (iv) the machine learning methods adopted to develop the sign language interpreter and the chatbot.

Concerning gesture recognition, it can be static (Aquino, 2023) or dynamic (Lenovo StoryHub, 2023). While in static gesture recognition hand gestures are given as a sequence of images each showing a specific sign, in dynamic gesture recognition hand gestures, and in some cases facial expressions, have to be captured and recognized from a live video (Huang et al., 2021). Also, the signals returned from the chatbot can be just texts (Voinea, 207) or both audio

and text outputs (Pardasani et al., 2018), and the type of conversations one can have with the chatbot varies with the project scope, spanning from providing psychotherapy and psychological counseling (Huang et al., 2021) to assisting users in retail environments (Berkley School of Information). In terms of machine learning methods, different approaches can be adopted to build well functioning sign language interpreters and chatbots, based on the characteristics of inputs and outputs, the size of the datasets used to build the models, and other constraints like time and machine capacity.

The project we present leverages supervised learning with neural networks, similar to other notable projects, including a MIDS Capstone project from UC Berkeley (American Sign Language Bot, n.d.) and a project developed by GitHub user Sophgdnfor (Sophgdn, n.d.). The MIDS Capstone project from UC Berkeley focused on assisting ASL speakers in retail environments using real-time video intelligence to interpret ASL. The team provided an overview of data processing, model implementation and error analysis in their invention. SignBot, an initial prototype developed by Sophgdn, uses LUIS, a Microsoft's machine learning-based service, to translate natural language speech into Australian Sign Language. The bot operates by translating messages into videos that depict how they would be communicated in sign language. The developers use LUIS to simplify complex sentences, making them compatible with the Australian Sign Language vocabulary. According to their README file, the application records a live feed of the user communicating via sign language, processes this feed using LUIS, and presents interpretation options for the user to choose from.

Our project incorporates similar approaches to both mentioned projects but with some notable differences. Like the UC Berkeley project, we used a live feed in our application, but while their primary goal was to focus on ASL in retail, our vision was to create a more

generalized model that can be utilized in various conversational contexts beyond retail. Unlike SignBot, which translates the video feed into potential comments or sentences for the user to select, our project takes the sentences and feeds them into a natural language model to generate textual responses, then converted into ASL images for users to visualize.

Machine Learning Methods

Choosing the right model architectures for both the sign language interpreter and the chatbot to integrate in our final application was a crucial step during project development. The reasons why we selected a CNN model for the sign language interpreter and fine-tuned a Flan-T5-Base model for the chatbot are provided in the following subsections.

Sign Language Interpreter

The CNN model seemed to us an appropriate choice for our sign language interpreter due to its exceptional ability to handle image data, making it well-suited for the ASL alphabet dataset. CNNs excel at hierarchical feature extraction through the use of convolutional and pooling layers, allowing the model to learn intricate patterns and spatial dependencies in the ASL images effectively. This hierarchical learning enables the model to identify essential features, such as edges, shapes and textures, which are crucial for accurate sign recognition. Another advantage of CNNs is the use of weight sharing in convolutional layers, which significantly reduces the number of parameters compared to traditional fully connected neural networks. This reduction in parameters makes CNNs computationally efficient.

Common drawbacks of CNNs, like its tendency to overfit, can usually be mitigated by implementing data augmentation techniques, such as random rotation, shifting, shearing, zooming, and flipping, in the ImageDataGenerator for the training set. Additionally, to enhance

model performance and address the typical need of CNN models for vast amounts of labeled data, the introduction of the rescale parameter (which ensures the normalization of pixel values) and the use of a substantial dataset of images for training are recommendable strategies to take into account during model development.

Based on all the above and given the huge number of images in the ASL alphabet dataset, the implementation of a CNN model was judged by us as an optimal solution for our sign language interpreter.

Chatbot

Regarding the chatbot, we thought of transfer learning as our best option. It leverages the knowledge of a previously trained model to properly perform a new task, while requiring a reduced amount of new data, computational power and time compared to a completely new model built from scratch. In addition, given a new dataset, the fine-tuning procedure enables pre-trained models to expand their original tasks and functions by adjusting a portion of their weights.

When building complex models like chatbots, possible pre-trained models could be Flan-T5-Base (google/flan-t5-base, n.d.), Gemma-2 (google/gemma-2-9b, n.d.), GPT-2 (openai-community/gpt, n.d.) and Llama 2 (Meta, n.d.). They are large language models, trained on huge textual datasets and recommended to be applied in a variety of fields, from healthcare to finance, for translation, text classification, question answering, document summarization and more (What is a large language model (LLM)?, n.d.). The success of these models was already demonstrated by other authors working on conversational models. For example, Rajani (n.d.) and Bhandare (n.d.) created very well performing chatbots by respectively fine-tuning the T5 (T5, n.d.) and Flan-T5-Base pre-trained models on the same conversations dataset selected for this project.

Flan-T5-Base, published by Google researchers in 2022, is an improved version of T5 and able to perform 1000 additional tasks with the same number of parameters (google/flan-t5-base, n.d.). It is an encoder-decoder model, suitable for applications like chat/dialogue summarization (Keita, 2023), which is in line with our project goal to provide users with quick answers to general questions through a chatbot.

Thus, considering the numerous advantages of using a pre-trained model, the good results obtained by Bhandare (n.d.) with the Flan-T5-Base model and our conversations dataset, and its ease of implementation, we decided to fine-tune the Flan-T5-Base model to create our chatbot.

Detailed Experimental Methods

CNN Model

The CNN model consists of one input layer, four convolutional layers and three dense layers, with the addition of one output layer. The number of filters in the convolutional layers increases from 16 close to the input layer to 128 close to the first dense layer, while the number of units reduces from 512 in the first dense layer to 128 in the last one before the output layer. Thus, we imposed an ascending number of filters and a descending number of units moving forward in the convolutional and dense layers, respectively. While the ascending number of filters in the convolutional layers allows the model to progressively capture more complex patterns starting from raw pixel data, the descending number of units in the dense layers leads to a gradual removal of the least important image features before reaching the last dense layer. For both convolutional layers and dense layers a “relu” activation function was adopted, while for the output layer a “softmax” function was selected. The “relu” activation function is widely used in CNNs as it introduces non linearity, thereby allowing the model to learn complex relationships

from the training data. The “softmax” activation function is a typical choice in the last dense layer when solving a multiclass classification problem.

To prevent overfitting and improve model generalization to unseen data, different regularization techniques were employed. There is a max pooling layer of size 2x2 after each of the first three convolutional layers, a global average pooling layer and a batch normalization layer before the first dense layer, and a dropout layer with a dropout rate of 50% after each of the first two dense layers. The pooling layers reduce the size of the images by selecting maximum or average pixel values from groups of pixels depending on the type of pooling (max or average), the batch normalization layer modifies the distribution of the previous layer’s output to improve the efficiency of the model training, and the dropout layers randomly drop 50% of their nodes to avoid an overdependence of the model on training data.

We used 70%, 20% and 10% of the data for the training, validation and testing sets, respectively. The model was compiled with the Adam optimizer and a learning rate of 1e-3, which dynamically reduces during training up to 1e-6 in order to minimize the validation loss. Also, we imposed the categorical cross-entropy loss function as the function to be minimized by the optimizer (considering the multiclass nature of the problem), and the accuracy as the metric to monitor model performance during training. The model was trained for a total of 50 epochs and with a batch size of 32, and the model with the maximum accuracy was saved.

The CNN model architecture described above was progressively optimized after several iterations based on the final model performance. We did not implement an automatic procedure to fine-tune our hyperparameters but manually modified the number of convolutional and dense layers, as well as the number of filters/units in convolutional/dense layers, in order to obtain a desirable final model performance.

Fine-Tuned Flan-T5-Base Chatbot Model

The Flan-T5-Base model can be used out of the box without fine-tuning but we chose to fine-tune it with our conversations dataset by referring to the guide by Bhandare (n.d.), thereby obtaining a model that adequately responds to our project needs.

Instead of following the traditional full fine-tuning approach, we prepared the model for k-bit training using PEFT (Parameter-Efficient Fine-Tuning), a technique applicable to large language models (Bhandare, n.d.). It allows one to modify just a small group of model parameters instead of the entire model, with numerous advantages in terms of computational cost, final model size and inference velocity. Among the different PEFT techniques, we applied LoRA (Low-Rank Adaptation), which accelerates fine-tuning of large models while conserving memory (Bhandare, n.d.). Given the model's size, these steps were proved to be crucial to prevent system crashes. Thus, after creating the PEFT configuration, we wrapped the model in PEFT and ran the trainer. We finally saved the tuned model, reloaded it into our environment, and created a pipeline using the Hugging Face Transformers library to facilitate integration into our chatbot system.

We used 80% and 10% of questions/answers for training and validation, respectively. We imposed a learning rate of 1e-3, a total number of epochs of 5, and an initial batch size of 8, which automatically lowers in case of out-of-memory issues. Each epoch was splitted into different steps and the loss was the metric we imposed to check model performance during training.

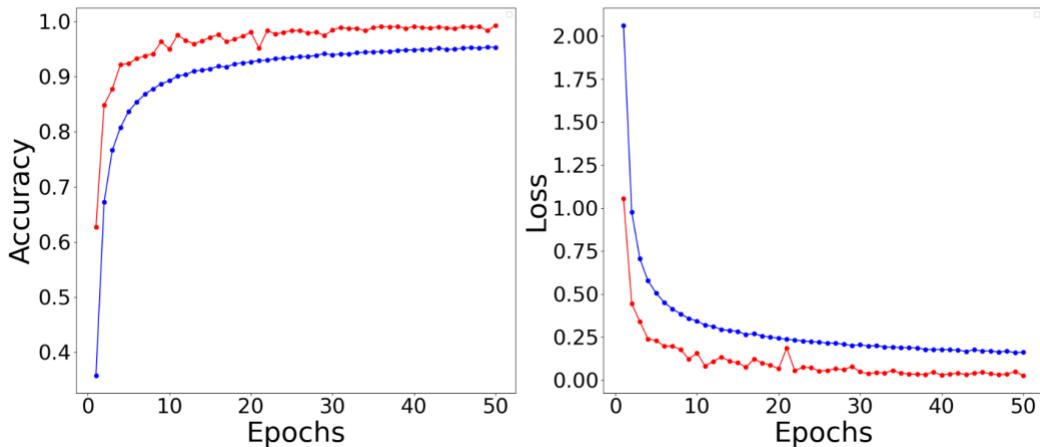
Results

CNN Model

As demonstrated by the learning curves in Figure 5, the CNN model was proved to be effective in achieving its goal. Training and validation accuracy/loss progressively increases/decreases with the number of epochs until reaching a plateau. This means that the model does not overfit or underfit: it properly learned from the training data but it is also able to generalize well on validation data (Bnomial, n.d.).

Figure 5

Representation of training (blue) and validation (red) accuracy (left panel) and loss (right panel) over the total number of epochs for the CNN model

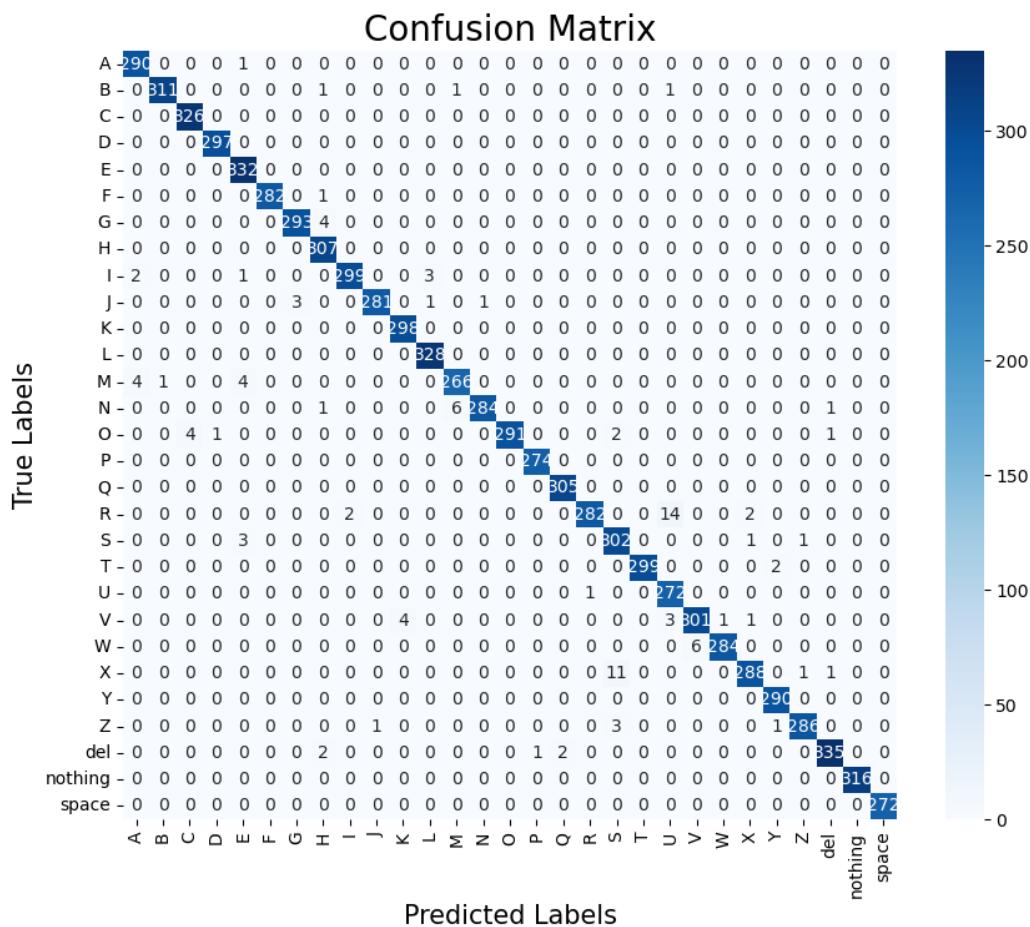


After training the CNN model, we verified its performance on the testing set, which includes 8700 different sign language images. We observed the model capability to predict the letters corresponding to the test images. Thus, we obtained that the macro/weighted averages of the test precision, recall and f1-score are equal to 0.99 for all the classes and the model accuracy is 0.99 as well. The confusion matrix calculated on the testing set is also given in Figure 6. The

good performance of the CNN model on the testing set is the evidence that the developed CNN model can be successfully adopted to translate sign language images into text.

Figure 6

Confusion matrix of the CNN model calculated on the testing set



For our specific application, the optimal performance of the CNN model is in fact pivotal.

Imagine one wants to use our CNN model just to translate a simple sentence from sign language to text. If the model was not always able to properly classify most of the letters, it could lead to a different or nonsense sentence, that would have detrimental consequences on the subsequent

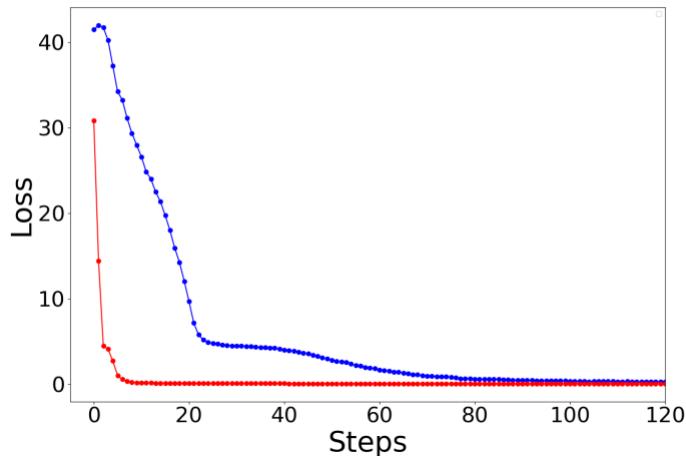
chatbot answers and make the entire project impossible to be carried out. Thus, considering our project goal, the CNN model performance we obtained is absolutely desirable and does not suggest the necessity to implement further model modifications.

Fine-Tuned Flan-T5-Base Chatbot Model

The learning task was properly learned by the chatbot model, as resulting from the behavior of the training and validation loss, which both converge to zero after about 90 steps (Figure 7).

Figure 7

Representation of training (blue) and validation (red) loss for the Flan-T5-Base chatbot model



The performance of the chatbot model was qualitatively evaluated through an initial list of 20 questions on different topics, which should be further expanded in the future for a more rigorous evaluation. The answers provided by the model to each questionnaire are reported in Table 1. As one can see from the latter, model answers are relevant and consistent with corresponding questions, although they are not always right and could be more precise. For

example, “Chris Martin” is not a “British actor” but a “British singer and songwriter” (see question 17), and answering that next Easter will be celebrated “in the spring” is correct but quite vague (see question 18). Also, punctuation is often absent and letter cases can be wrong depending on the question. For instance, city names can be provided in either uppercase, like “Rome” (question 8), or lowercase, like “san francisco” (question 7), that is grammatically incorrect. Another issue with our natural language processing model is its difficulty in distinguishing between the questions “how” and “what.” For instance, when asked, “How are you doing?” the model responds with “I am doing a project.” Although this response is grammatically correct, it is contextually inappropriate. However, the model accurately responds to questions like “What is the color of the sky?” or “What is the capital of Italy?” within the correct context.

Based on the above, we can affirm that our chatbot model is often able to provide pertinent textual answers to the received textual questions and, thus, represent an interesting first solution to the problem we are trying to solve. We cannot say that the imprecisions resulting from our initial evaluation of the chatbot performance are totally unexpected, as chatbots often exhibit similar results. Nonetheless, it is our thought that the proposed chatbot model should be reevaluated on a bigger set of questions and improved, as specified in the “Further Improvements” section.

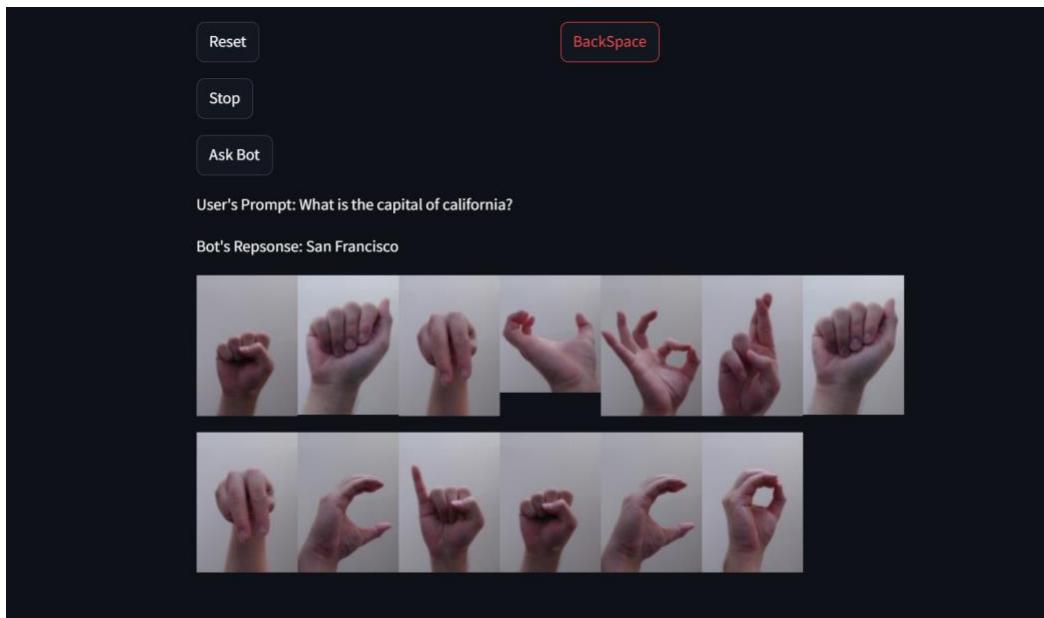
Final Application

After training and completing our two models, we developed an application to facilitate user interaction. Using Streamlit, an open-source Python framework for building interactive data apps, we equipped our application with several key features. First, it can capture an image from a live video feed by utilizing a webcam as a data source. Users can click a button to capture a

frame, which is then processed by our CNN model to predict the associated ASL sign. Next, we included buttons to manipulate the string generated by the CNN model's predictions. These buttons allow users to reset the string, remove the last character, or add a question mark. Finally, our application enables users to chat with the bot. Once the user is satisfied with the constructed string, they can click the 'Ask Bot' button to input the string into our natural language model, which generates a response. Alongside displaying the text generated by the model, the application also offers a feature that visualizes the text using images (Figure 8).

Figure 8

Simulating a real-life avatar using images



Further Improvements

Given the chance to continue working on this project, we will probably work on the chatbot model. As highlighted above, the CNN model already reached an optimal performance and, at the moment, does not seem to require additional improvements. The chatbot model,

instead, still shows a few inaccuracies. Midway through our project, we in fact realized that our conversational dataset was too limited in scope, which restricted the variety of answers our model could generate. A larger dataset would enable us not only to accommodate a broader range of prompts but also to better evaluate the different responses produced by the model. Another possibility for a better chatbot model could be employing a different pre-trained model. In this work, we decided to opt for the Flan-T5-base model, by exploiting the “Parameter Efficient Fine-Tuning” (PEFT) approach. In the future, we could compare the performance of our current chatbot model with that obtained with other pre-trained fine-tuned models. In this way, it could be possible to identify benefits and limitations of each solution in order to choose the best one.

Another improvement involves making the application less hands-on and speeding up its response time. Currently, the application requires the user to click buttons to capture a specific frame, delete the last character in a string, reset the string, add a question mark if necessary, or interact with the bot. Our future goal is to enable the application to recognize other signals that can perform these tasks automatically, reducing the need for direct user interaction with the screen. Our graphical user interface (GUI) currently experiences some lag. Due to the demanding nature of capturing frames, modifying the string, and querying the bot, the Streamlit API struggles to handle all these tasks smoothly. A possible solution could be to create a local GUI using PyQt or Tkinter and deploy it on a separate server, which would help reduce latency between actions and improve the overall user experience.

Finally, the creation of a virtual avatar to translate the textual answers from the chatbot model into a live sign language feedback could be an additional improvement area. For our application, we took the answer given by the chatbot, converted it into a list, and did some fancy

operation in the background to spell out the answer using ASL images (Figure 7). This was our initial attempt to simulate a virtual avatar interaction between the user and the bot. Hopefully, in the future, we will be able to implement fluid animations and various customizations to bring the avatar to life, making users feel like they are interacting with real people.

Table 1

Examples of conversations with the Flan-T5-Base chatbot model. Questions were asked by us while answers were provided by the model

ID	Question Text	Answer Text
1	Hello, how are you?	I'm fine.
2	What is your name?	samuel samuel
3	Are you happy today?	I am happy
4	What is your favorite color?	blue
5	What is your dream?	i want to be a good man
6	Do you think it will rain in the afternoon?	It will be cloudy.
7	What is the capital of California?	san francisco
8	What is the capital of Italy?	Rome
9	When does summer begin?	summer
10	What is the best drink for a hangover?	a stout
11	What is the probability that two apples will fall at the same time?	1 / 2
12	Do you know the definition of artificial intelligence?	No
13	Could you please recommend an American dish?	a fried chicken sandwich
14	Where would you go on vacation this summer?	sweden
15	What is Buddhism?	Buddhism is a religion

16	Could you suggest a person's name with C?	charlie cleveland
17	Who is Chris Martin?	british actor
18	When will next Easter be celebrated?	in the spring
19	What is rosemary?	rosemary is a genus of plants
20	What would you do in case of dehydration?	drink water

References

- Aquino, S. (2023, May 30). *Inside Google's Effort To Use AI To Make ASL Accessible To All*. Forbes. <https://www.forbes.com/sites/stevenaquino/2023/05/30/inside-googles-effort-to-use-ai-to-make-asl-accessible-to-all/>
- *American Sign Language*. (2021, October 29). National Institute on Deafness and Other Communication Disorders. Retrieved August 6, 2024, from <https://www.nidcd.nih.gov/healthamerican-sign-language>
- *American Sign Language Bot*. (n.d.). UC Berkeley School of Information. Retrieved August 6, 2024, from <https://www.ischool.berkeley.edu/projects/2020american-sign-language-bot>
- *ASL Alphabet*. (n.d.). Kaggle. Retrieved July 15, 2024, from <https://www.kaggle.com/datasets/grassknotted/asl-alphabet>
- Bhandare, A. (n.d.). *Fine-tune Flan-T5-base for chat with PEFT/LoRA!* Kaggle. <https://www.kaggle.com/code/ajinkyabhandare2002/fine-tune-flan-t5-base-for-chat-with-peft-lora>
- Bnomial. (n.d.). *Overfitting and Underfitting with Learning Curves*. <https://articles.bnomial.com/overfitting-underfitting-learning->

[curves#:~:text=Learning%20curves%20are%20a%20great, on%20training%20and%20testing%20z](#)

- *google/flan-t5-base.* (n.d.). Hugging Face. Retrieved August 7, 2024, from
<https://huggingface.co/google/flan-t5-base>
- *google/gemma-2-9b.* (n.d.). Hugging Face. Retrieved August 7, 2024, from
<https://huggingface.co/google/gemma-2-9b>
- Huang, X., Wu, B., and Kameda, H. (2021). Development of a Sign Language Dialogue System for a Healing Dialogue Robot. *2021 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, AB, Canada (pp. 867-872).
<https://ieeexplore.ieee.org/document/9730480>
- Keita, Z. (2023, November). *FLAN-T5 Tutorial: Guide and Fine-Tuning*. Datacamp.
<https://www.datacamp.com/tutorial/flan-t5-tutorial>
- Kumar, V. (2020, October 25). *Hands-On Guide To Sign Language Classification Using CNN*. AIM. <https://analyticsindiamag.com/ai-mysteries/hands-on-guide-to-sign-language-classification-using-cnn/>
- Lenovo StoryHub. (2023, October 24). *Lenovo's AI-powered sign language translation solution empowers signers in Brazil*. Lenovo.
<https://news.lenovo.com/ai-powered-sign-language-translation-solution-hearing-brazil/>
- Meta. (n.d.). *Llama 2: open source, free for research and commercial use*.
<https://llama.meta.com/llama2/>

- *openai-community/gpt2*. (n.d.). Hugging Face. Retrieved August 7, 2024, from <https://huggingface.co/openai-community/gpt2>
- Pardasani, A., Sharma, A.K., Banerjee, S., Garg, V., and Roy, D.S. (2018). Enhancing the Ability to Communicate by Synthesizing American Sign Language using Image Recognition in A Chatbot for Differently Abled. *2018 7th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), Noida, India* (pp. 529-532). <https://doi: 10.1109/ICRITO.2018.8748590>
- Rajani, K. (n.d.). *Fine-Tune T5 for Conversational Model!* Kaggle. <https://www.kaggle.com/code/kreeshrajani/fine-tune-t5-for-conversational-model>
- *Sign Language*. (n.d.). National Geographic. Retrieved August 5, 2024, from <https://education.nationalgeographic.org/resource/sign-language/>
- *Sign Language Processing*. (n.d.). Research Sign. Retrieved August 12, 2024 from <https://research.sign.mt/>
- Sophgdn. (n.d.). *GitHub - sophgdn/SignBot: A chatbot that uses LUIS to process natural language and turn it into Australian Sign Language*. GitHub. <https://github.com/sophgdn/SignBot>
- S, P. (2022, July 1). *How to Approach CNN Architecture from Scratch?* Analytics Vidhya. <https://www.analyticsvidhya.com/blog/2022/05/how-to-approach-cnn-architecture-from-scratch/>
- Thakur, A. (2021, December 10). *Translate American Sign Language Using CNN - Ayush Thakur - Medium*. Medium. <https://mein2work.medium.com/using-cnn-to-translate-american-sign-language-10d6443f7cd>

- *T5.* (n.d.). Hugging Face. Retrieved August 7, 2024, from
https://huggingface.co/docs/transformers/en/model_doc/t5
- Vaidhya, G.K., & Deiva Preetha, C.A.S. (2022). A Comprehensive Study on Sign Language Recognition for Deaf and Dumb people. *Journal of Trends in Computer Science and Smart Technology*, 4(3), 163-175. doi: 10.36548/jtcsst.2022.3.005
- Voinea, T. (2017, November 14). *SignKit-Learn: Using Machine Learning to converse with a bot in American Sign Language*. Medium.
<https://medium.com/@TeoVoinea/signkit-learn-using-machine-learning-to-converse-with-a-bot-in-american-sign-language-e122ae8da1b8>
- *What is a large language model (LLM)?* (n.d.). Elastic. Retrieved August 9, 2024, from
<https://www.elastic.co/what-is/large-language-models>
- *3K Conversations Dataset for ChatBot.* (n.d.). Kaggle. Retrieved July 15, 2024, from
<https://www.kaggle.com/datasets/kreeshrajani/3k-conversations-dataset-for-chatbot>

Fall 2024



Deep Learning Image Captioning

Steve Amancha, Rahul Das, Juliet Lawton

University of San Diego

AAI-590: Capstone Project

Professor Anna Marbut

12/9/24

Introduction

The goal of this project is to build an image captioning model that can accurately generate text descriptions of images. Image captioning combines Natural Language Processing and Computer Vision methods to learn semantic relationships between images and language to produce captions for images. Image captioning has many different applications ranging from image search tools to accessibility services for people who are blind or visually impaired. It can even be used to assist in medical diagnosis by generating medical reports from diagnostic imaging (Beddiar & Oussalah, 2023).

The dataset we will use in this project is Flickr30k, introduced in the 2014 paper *From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions* (Young et al.). Flickr30k is an open-source dataset containing thousands of images and reference captions curated for computer vision problems such as image captioning and image retrieval. Each image in the dataset has five associated captions describing the image, obtained by crowdsourcing.

Many different state-of-the-art solutions for using machine learning models to caption images exist already; however, they tend to be very large, making them expensive to train and deploy. As a secondary goal of this project, we hope to build an image captioning model that has reasonable performance, but with a much smaller size than contemporary models. We foresee the end-users of this model being developers who want to integrate an image captioning model into a system that has strict memory or latency constraints, such as a wearable device for people with low or no vision that describes their surroundings to them.

Data Summary

The copy of the Flickr30k dataset used in this project was retrieved from Hugging Face (nlphuji, 2023). There are 31,014 images in the dataset with 5 text captions each for a total of 155,070 unique pairs. The images span a wide range of subjects including people, animals, and inanimate objects. Figure 1 shows an example of an image from the dataset and its captions.

Figure 1

Sample from the Flickr30k Dataset



A man and his daughter are posing for a picture in a brightly colored field of yellow flowers.

A man and a young child are in a meadow filled with yellow flowers.

A dad and his son, both in green, smile in a field of flowers.

A father and son are in a field of yellow flowers.

A man in a collar shirt is holding a babies hand.

Exploratory Data Analysis

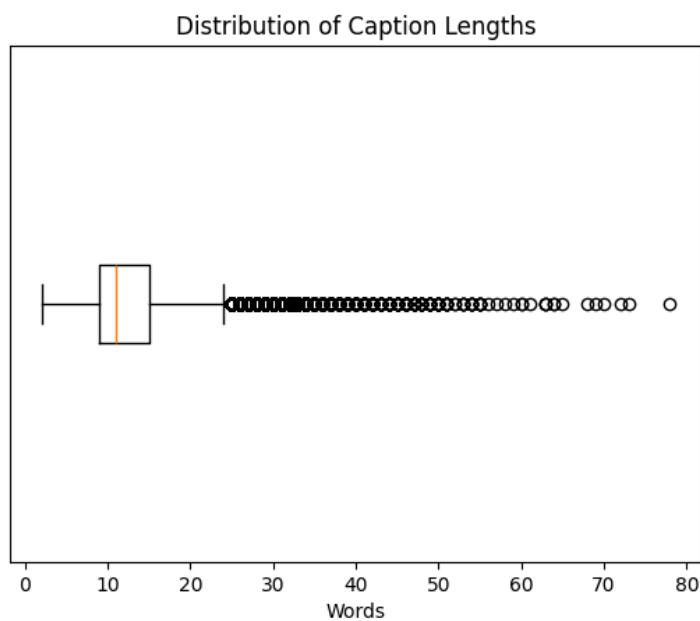
To familiarize ourselves with the data, we used random sampling to examine small samples of the images and their captions. Additionally, we defined a search function to search the images by keywords in their captions. We found that there is a high amount of variability in the images, both in terms of content and attributes of the images such as their dimensions, orientation, and zoom level. Similarly, there is also a high amount of variability in the captions both in length and quality. One of the challenges posed by this dataset is the ambiguity and variation in quality of the image captions. Take for instance the captions for the image in Figure 1 above. Two of the captions identify the child in the image as a girl, two of the captions don't reference the gender of the child, and the other caption identifies the child as a boy. There are

also differences in the level of detail of each caption and the focus of the caption (e.g., only one caption mentions the color of the clothing, one caption does not mention the flowers).

The captions are about 12 words in length on average, but range from as short as 2 words to as long as 78 words. Figure 2 shows this distribution as a box plot. 95% of the captions were 22 words or less, with a substantial number of outliers much larger than this. This is significant because for a model that processes text sequences, the sequences need to be uniform in length.

Figure 2

Distribution of Image Caption Lengths



To get a better idea of what subjects appear most commonly in the images, we used a word cloud to visualize the text corpus (excluding stop words such as “a” or “and”). The size of the words indicates how frequently they appear in the corpus. In the word cloud shown in Figure 3 on the next page, “man” and “woman” are by far the most prominent words, suggesting that most of the images in the dataset are of people.

Figure 3

Word Cloud for Flickr30k Captions



We further explored the text by checking for non-alphabetic characters and examining what context they appear in, and how frequently they appear. It is a common text cleaning step to remove non-alphabetic characters from the text because they can create noisy input that hurts model performance, but in some cases these characters contain important information. For example, captions describing an image of a clock would most likely write the time in numeric form (e.g., 2:20 PM) instead of text form (e.g., two twenty PM). Removing these characters would make such a caption incoherent, which would in turn make the captions generated by the model incoherent.

Most of the non-alphabetic characters we identified in the captions occurred very infrequently. Numeric characters appeared the most frequently, but were still uncommon overall (1776 instances out of 155,070 total captions). This infrequency of appearance made it easy to analyze the usage of these characters in the captions. Some of the special characters, such as “@” and “=” only appeared as typos, whereas other characters such as “#” or “&” were always meaningful. The most ambiguity presented itself with question marks, which sometimes were used literally (e.g. “A man walking reads a wall asking ‘Where are you?’”), and sometimes used

by the annotator to express doubt (e.g., “A crowd admiring modern art?”). We used this information to inform our text cleaning strategy.

Data Cleaning

Common text preprocessing steps like stop word removal and stemming will cause a text generation model to produce incoherent text, so for this reason, we only applied minimal text cleaning operations to the captions. This included converting the text to lowercase, removing punctuation, removing unuseful special characters, removing excess whitespace, and replacing characters that easily translated to a word, such as “%” (percent) or “&” (and), with that word.

After cleaning the captions, we applied tokenization and encoding to convert the captions into a numerical representation. We used either the tokenizer from Tensorflow’s Keras API or the Text Vectorization layer from the same API, which can be used as a tokenizer when adapted to the vocabulary for the data (Abadi et al., 2015). Both tokenization methods split each caption into substrings that are usually at the word level.

The text preprocessing steps were applied to the entire dataset, however, this was not feasible for preprocessing the images. The biggest challenge presented by this dataset is how large it is. Preprocessing all the images at once would require loading all of them into memory, which can quickly exceed available memory. To get around this issue, we designed a data generator that dynamically loads the images and their captions in batches, only loading one batch into a memory at a time. This greatly reduces memory usage. Most of the candidate models we built use pre-trained models to extract features from the images, so we pre-computed the image features for each model and served these pre-computed image features from the data generators for those models instead of the full images, further reducing memory usage and training time. Each of these pre-trained models has its own preprocessing method that scales and normalizes

the images, but for the model that did not use pre-computed image features, we resized the images to be 64x64 pixels and normalized the pixel values to the scale (0, 1).

Initially, we experimented with randomly selecting one caption for each image to reduce the memory and time needed to train the models but after the improvements described above, we were able to train all of the models on all of the image-caption pairs for a total of 155,070 training examples.

Background Information

Image captioning is a popular deep learning problem and has been studied extensively. One of the most common kinds of architecture for this problem is the encoder-decoder pattern (*Papers with Code - Image Captioning*, n.d.). At a high level, this pattern can be broken into two general parts:

- (1) An input image and its caption are each encoded into dense feature representations.
- (2) The output of the previous step is decoded into an output sequence of words and combined phrases to produce an image caption.

The encoder can be any kind of neural network that is well-suited to image data, and the decoder can be any kind of neural network well-suited to sequential data. One common method is to use a CNN (Convolutional Neural Network) for the encoder for the images and an LSTM (Long Short Term Memory) as an encoder for the captions, or as the decoder. In this architecture, the feature maps extracted by the CNN can be used as input to the LSTM decoder along with the captions as one input sequence. In generating image captions, image information is included in the initial state of an LSTM (Hossain et al., 2018). Alternatively, the LSTM can be used as part of the encoder for the caption independently of the CNN, and then the combined representations can be decoded through a feed forward network. The next words are generated based on the

current time step and the previous state of the input. This process continues until it gets the end token of the sentence.

As an alternative to using a CNN for the image encoder, a vision transformer (ViT) can be used instead. Vision transformers are a vision-only variation of traditional transformer architecture that are designed to process images as sequential patches that can be treated as a sequence of tokens (Dosovitskiy et al., 2021). Vision transformers are much less compute resource intensive than CNNs, but require a substantial amount of training data to reach equivalent performance.

Another approach is to use a transformer for the decoder (*Vision Encoder Decoder Models*, n.d.). In this approach, a transformer or pre-trained language model like BERT or GPT-2 is used to decode the input sequences into captions. One popular pre-trained model that follows this pattern is ViT GPT2, which uses a vision transformer as the encoder and GPT-2 as the decoder (nlpconnect, 2023).

A unique aspect of training machine learning models to generate text is that during training, the model is shown the correct captions for each image both as input and the ground truth to compare predictions against. Including the reference captions for an image as input to the decoder is part of a technique known as teacher forcing (Wong, 2019). Teacher forcing uses the correct output as input to the decoder part of the model at each sequential step instead of using the model predictions for the previous step. This technique makes training the model faster and more stable by preventing the accumulation of errors that can happen when the model makes very bad predictions in the early stages of training.

Experimental Methods

Training Protocol

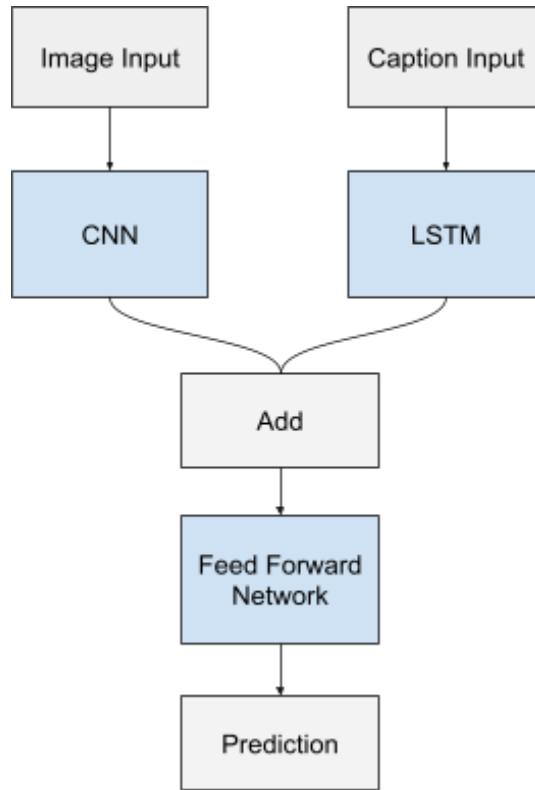
We built and tested six models, four of which were CNN-LSTM models and two of which were visual attention models. To train the models, the data was broken into training, testing, and validation sets. The Flickr30k dataset comes with predefined training splits used for benchmarking, so these splits were used instead of using sampling. The training set has about 94% of the data (29,000 samples) and the testing and validation set each have about 3% of the data (1,000 samples each). All of the models were trained using cross-entropy loss, with accuracy logged as an additional metric. Categorical cross-entropy is typically used to evaluate multi-class classification models, but it is well suited for this task because the models output a single word at each time step, which can be interpreted as a multi-class classification problem where the possible class labels are the tokenizer vocabulary. The models were trained for 5-15 epochs, with callbacks used to decrease the learning rate and stop training the model if performance on the validation set stopped improving. Another callback was used to display a caption prediction for a test image after every epoch. Each model used a similar training protocol, but was optimized differently based on the particulars of the model.

CNN-LSTM Architecture

Four of the candidate models are variations of CNN-LSTM architecture. Each model has an encoder, broken into two parts – an image encoder and a caption (or language) encoder – and a decoder. Figure 5 shows a high level diagram for this architecture.

Figure 5

CNN-LSTM Architecture Diagram



In the image encoder, a CNN is used to extract features from the images. In each convolutional layer of the CNN, the model learns to capture progressively abstract features. For example, earlier layers of the CNN might learn to identify corners or edges, while deeper layers of the CNN may learn to identify more intricate patterns like shapes or textures. These features are then compressed into a dense representation.

In the language encoder, a similar process is carried out for the captions. The captions are broken up into independent segments, embedded to capture their semantic context, and the LSTM learns temporal relationships from the segments. For example, the caption “A man is wearing a hat” would become multiple training examples: (a, man), (a man, is), (a man is,

wearing) and so on. Figure 5 shows a full example of this process. Breaking the captions up like this allows the encoder to capture both smaller dependencies between individual words as well as longer dependencies that capture the context of the full caption. Similar to the image encoder, the features extracted by the LSTM are compressed into a dense representation.

Figure 6

Example of Training Data Structure for CNN-LSTM Models

“Simple caption example.”

[START] → simple

[START] simple → caption

[START] simple caption → example

[START] simple caption example → [END]

The image representation and the caption representation are then combined into a single representation and then fed as input to the decoder, which is a feed forward network. The decoder predicts the next word in a caption, one word at a time. During inference, captions are built incrementally starting with the start sequence token.

The following table shows the variations in the architecture of each of the candidate CNN-LSTM models. All of the pre-trained CNNs used on this project use weights from training on ImageNet.

Table 1*Candidate Models*

Model	Image Encoder	Caption Encoder	Decoder
A	Custom CNN	LSTM	Feed Forward Network
B	MobileNetV3	LSTM	Feed Forward Network
C	VGG16	LTSM	Feed Forward Network
D	ResNet50	LSTM	Feed Forward Network

Custom CNN-LSTM

For this model, a simple CNN was built from scratch for the image encoder instead of using a pre-trained CNN to extract image features as a preprocessing step. The CNN has 3 convolutional layers each with a stride of 2 and 16, 32, and 64 filters, respectively. The number of filters doubles every layer in order to be able to capture more from the deeper layers of the network, and a stride of 2 was chosen because the input images for this model are relatively small (64x64) to stay within memory constraints.

This model stacks two LSTMs in the language encoder, each with 256 units. Dropout layers are used in the language encoder and the decoder to reduce overfitting, however, increasing the dropout rate above 0.2 hurt the performance of the model without improving overfitting. The model was trained for 10 epochs with a learning rate of 1e-3, ending on epoch 8 after early stopping with a reduced learning rate of 1e-5.

MobileNetV3-LSTM

This model is very similar to the Custom CNN-LSTM model, but uses image features extracted from MobileNetV3-Small (Howard et al., 2019) as input to the image encoder.

MobileNetV3 is part of a family of lightweight CNN models that are optimized to run on a wide range of hardware, including phone CPUs, making them very efficient and small compared to other pre-trained CNNs. The feature vectors extracted with MobileNetV3-Small have a shape of (576,) which is relatively small. The MobileNetV3-LSTM model was trained for 15 epochs with a learning rate of 1e-3, ending on epoch 11 after early stopping with a reduced learning rate of 1e-7.

VGG16-LTSM

This model uses VGG16 (Simonyan & Zisserman, 2015) as the image feature extractor. VGG16 is an object detection and classification algorithm which can classify 1000 images of 1000 different categories with 92.7% accuracy. VGG16 is a popular image classification model due to its unique architecture of a deep neural network with relatively small filters (3x3). The feature vectors extracted with VGG16 have a shape of (512,) making it the smallest pre-trained CNN tested. A single LSTM is used in this model to encode the caption, and a single dense layer is used in the hidden layers of the decoder. This model was trained for 5 epochs and a learning rate of 1e-3.

ResNet50-LTSM

The final variation of the CNN-LSTM architecture uses ResNet50 (He et al., 2015). The ResNet CNN family uses residual learning, in which layers of the network learn from residuals (the difference between correct output and input) between layers instead of the full transformation, to build substantially deeper networks than other CNN architectures. The feature vectors for ResNet50 have the shape (2048,), making this the largest pre-trained CNN used.

This model originally shared a very similar architecture to the other CNN-LSTM models discussed, but was modified to concatenate the encoded image and embedded caption into one

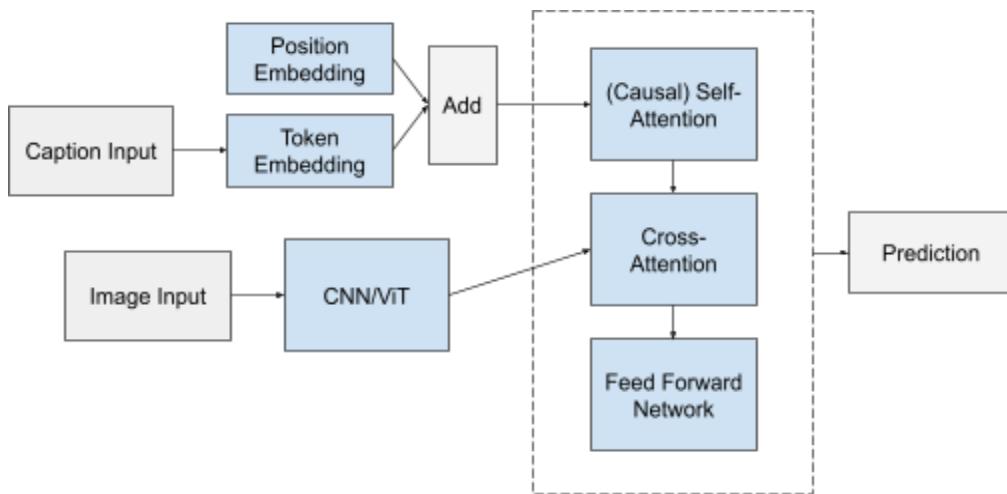
sequence and run this sequence through the LSTM instead of feeding only the caption sequences to the LSTM. This was done to see if better captions could be generated by including the image features as context for the LSTM, however, this approach was not effective and the model loss did not decrease past one epoch.

Visual Attention Models

The second architecture explored was a visual attention architecture in which a multi-layer transformer block is used as the decoder. A high-level visualization of this architecture is presented in the diagram below. The dashed lines encapsulate the transformer block, which can be stacked.

Figure 7

Visual Attention Architecture Diagram



Similar to the CNN-LSTM architecture, the visual attention architecture uses a pre-trained model, either a CNN or a vision transformer, to extract features from the images. For the captions, a sequential embedding is created. Because transformers process sequences in parallel, by default, transformers are permutation invariant (Vaswani et al., 2023). This means

that they do not distinguish between different orderings of a sequence, e.g., “The box is on the chair” and “The chair is on the box” would be treated as the same sequence and have the same output.

A notion of order can be added to the sequence by creating a positional embedding that encodes the position of each token in a sequence. There are two primary kinds of positional embeddings: fixed and learned (Zakka, 2023). Fixed positional embeddings encode position using a fixed function, typically a sinusoidal function. Learned positional embeddings use a trainable embedding layer to learn positional encodings of a specific size. Fixed positional embeddings are not updated during training, making them useful for handling sequences that may be much longer than what was seen during training. For our use case this is not an issue since the model is generating captions with a maximum length that is smaller than the training sequences, so we chose to use a learned positional embedding layer. This positional embedding is added to the token embedding to augment the semantic information captured by the token embedding with sequential information.

Each block of the decoder consists of a self-attention layer, a cross-attention layer, and a feed forward network. Although not pictured for simplicity in the diagram in Figure 7, there are also residual connections after the self-attention and cross-attention layers as well as normalization layers after each of the three main components. For each block, the sequential embedding is the input to the self-attention layer. In a self-attention layer, the input is both the query and the key (and value), which has the practical effect of allowing tokens in the sequence to “attend” to other tokens in the same sequence and update the sequential embedding with richer contextual information about relationships and dependencies between different parts of the

sequence. As mentioned earlier, transformers “see” the entire sequence at once, so causal masking is applied to prevent tokens from attending to tokens later in the sequence.

The updated sequential embedding and the image features are then fed to the cross-attention layer, where the sequential embedding is the query and the image features are the key. This allows tokens in the sequential embedding to attend to the image, enriching the sequential embedding again, this time with contextual information about the relationships between parts of the image and tokens in the sequence. The output of the cross-attention layer then passes through a feed forward network to further refine the sequential embedding. At this point, the updated embedding is either fed to the next block of the decoder as input, or is fed through the output layer to make a prediction.

DenseNet121 Visual Attention

The first variation of the visual attention architecture uses DenseNet121(Huang et al., 2018) as the image feature extractor. DenseNet is a family of CNNs pre-trained on ImageNet. The DenseNet models differ from other pre-trained CNNs by using dense convolutional layer blocks, in which there are direct connections between all the layers in a block instead of using feed-forward connections. This has the benefit of improving feature propagation and mitigating the vanishing gradient problem.

This model uses two decoder layers and was trained with a dropout rate of 0.5. Like the CNN-LSTM models, it was trained with early stopping and reduce learning rate on plateau callbacks, as well as a callback that generates a prediction from the model after every epoch. It was trained for 15 epochs and a learning rate of 1e-4, and trained for 10 epochs before stopping.

ViT Visual Attention

The second variation used a vision transformer instead of a CNN as the image feature extractor. The specific vision transformer used was the google/vit-base-patch16-224-in21k model available on Hugging Face (Wightman, 2019). This model also uses two decoder layers, but was trained with a dropout rate of 0.3. It was trained for 15 epochs and a learning rate of 1e-4, and trained for 11 epochs before stopping.

Inference

Image captioning is an autoregressive task, meaning that during inference, the captions are generated sequentially with each word predicted based on the words predicted in prior steps. To predict a caption for a new image, the model is called with the image and the start sequence token as input. The model predicts a word, and that word is added to the input sequence and the model is called again. This continues until either the model predicts the end sequence token, indicating the end of the caption, or the caption reaches a specified maximum length.

The approach described above is a greedy search, which is fast to compute but can sometimes produce suboptimal results because it only considers the top word at each time step. An alternative approach for generating captions is to use a beam search. With beam search using k beams, multiple candidate sequences are maintained and at each time step, the candidate sequences are sorted by their joint probabilities and only the top k sequences are kept. Once all of the candidate sequences are complete, the sequence (caption) with the highest joint probability is chosen.

A benefit of the visual attention approach is that the attention scores from the cross-attention layers in the decoder can be mapped onto an input image to visualize which part

of an image the model attended to when predicting the next token in a sequence, which adds a measure of interpretability and explainability that the CNN-LSTM models do not have.

Figure 8

Sample Attention Maps for each Visual Attention Model

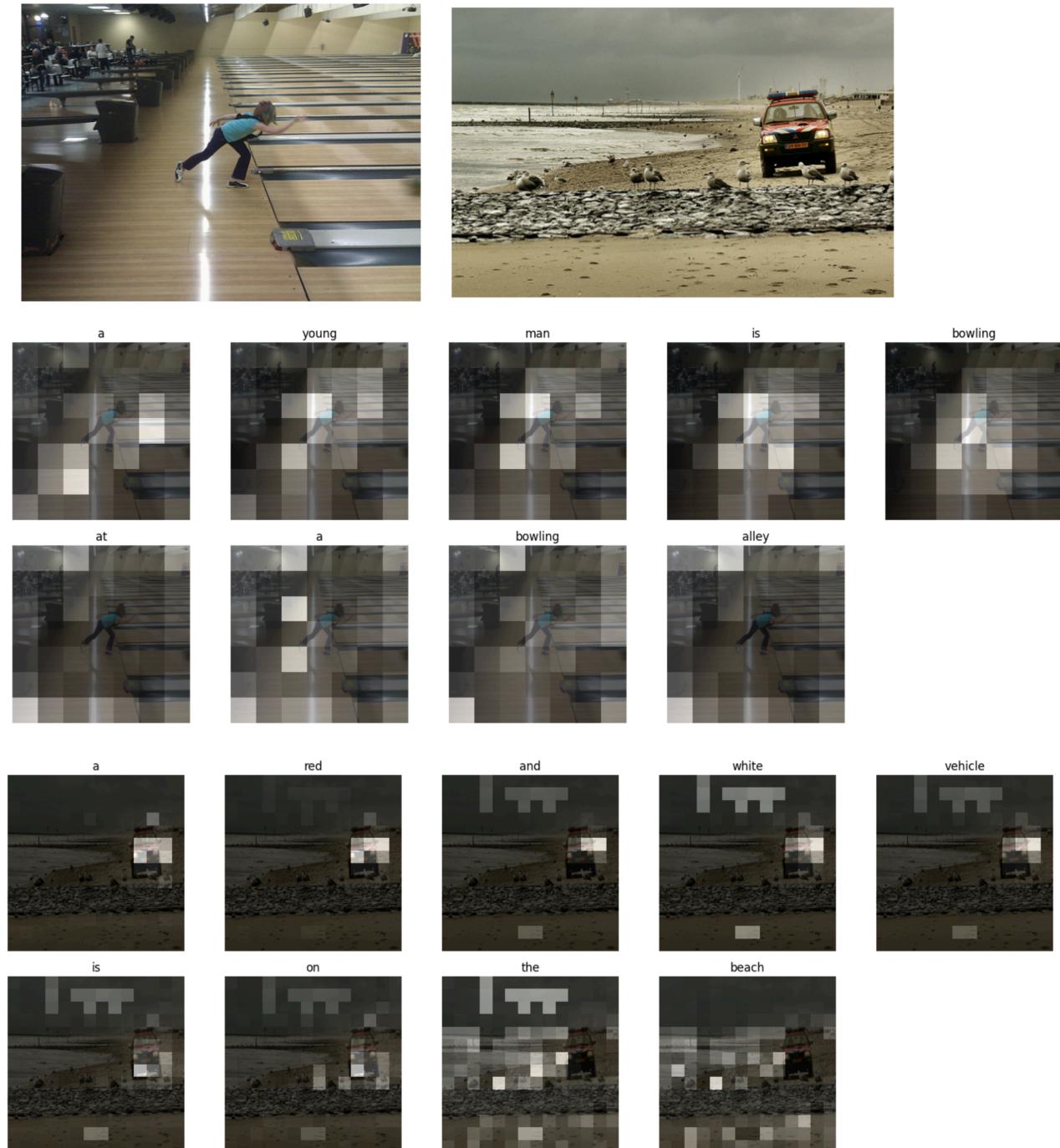


Figure 8 shows an example of attention maps for a single caption generated by each model. The attention maps are overlaid as heatmaps on the resized input images, where lighter cells have higher average attention scores. The first set of attention maps are from the DenseNet visual attention model, and the second set are from the ViT visual attention model. The difference in detail between the two sets of attention maps is due to the different image feature map sizes for each model (7x7 for DenseNet, 14x14 for ViT).

For the image on the top left, the predicted caption was “a young man is bowling at a bowling alley.” In the maps associated with the words describing the person in the image (young man is bowling), the cells in the center of the image where the person is located have higher scores. Conversely, the words describing the setting (at a bowling alley) have higher attention scores for the cells that border the image. In the second set of attention maps for the caption “a red and white vehicle is on the beach,” there is also a clear distinction between attention to the foreground (a red and white vehicle) and the background (on the beach).

Results and Conclusion

The visual attention model architecture showed substantial improvement in performance over the CNN-LSTM model architecture, with the visual attention model using ViT achieving the highest performance of all the candidate models. Most of the models were able to generate coherent captions for images with varying accuracy and quality, with the exception of the Custom CNN-LSTM and ResNet50-LSTM models. The CNN-LSTM models generally produce accurate captions for easily identifiable images, but the visual attention model variations showed increased capacity for accurately captioning less interpretable images with higher ambiguity, with room for improvement.

Evaluation

During training, validation loss was monitored to measure the improvement in the model across epochs. The validation loss was also used to lower the learning rate when model improvement plateaued and end the training early if the model began to overfit or stop improving. Accuracy was also logged as a secondary metric, however, is not a very reliable metric for image captioning because it measures how many of the tokens predicted by the model exactly match the ground truth tokens. This is not particularly useful for overall evaluation because a generated caption could be a correct description of an image but still have low accuracy if it uses different words than the reference captions it is evaluated against.

Figure 9

Sample Caption Generated by Visual Attention Model (DenseNet)



Actual captions:

young dirt bikers try to get a dirt bike up a sandy hill
a young man pushes his motocross bike up a dirt hill
a fallen dirt biker is aided by another
the dirt bike rider is deep in the sand
two boys lifting a dirt bike

Predicted caption:

a dirt biker in a red and white uniform is riding a dirt bike

For example, in the figure on the previous page none of the reference captions describe the uniform the dirt bikers are wearing, while the generated caption is focused on their uniforms. This would result in a lower accuracy score, even though the generated caption is correct. That being said, it is simple and efficient to compute, so it can be used as a proxy for how well the model is learning from the data during training. The loss function, cross-entropy loss, is less susceptible to this issue because it evaluates the predicted probabilities for all tokens and not exact matches.

Although dropout layers and other regularization techniques were used in the models, they did show signs of slight to moderate overfitting during training, depending on the model. This was especially prominent for the visual attention model variations. Since adding regularization to the models did not make much of a difference in the overfitting behavior, this could indicate that the learning capacity of the models is greater than the available amount of data.

For the final evaluation of the models, we computed BLEU and METEOR scores for predictions generated by each model on the test set. BLEU (Bilingual Evaluation Understudy) is a commonly used set of metrics for natural language processing tasks that evaluate the quality of generated text by computing the proportion of n-gram overlap (Doshi, 2021). BLEU-1, for instance, compares unigram overlap between a predicted caption and a set of references while BLEU-4 compares 4-gram overlap. BLEU is a popular choice of metric because it is easy to compute and interpret, but it is limited by not being able to capture semantic similarity. Similar to accuracy, BLEU scores would penalize the caption in Figure 9 for using words that do not appear in the reference captions, even though they correctly describe the image.

METEOR (Metric for Evaluation of Translation with Explicit ORdering) also measures the alignment of generated text against of references, but is a more sophisticated metric then BLEU because it treats semantically similar words as the same and uses weighted averages of various evaluation metrics such as precision and recall to measure both accuracy and fluency (Avinash, 2024). This makes it a more expressive measure, but more expensive to compute. The table below shows the scores for each model (ResNet50-LSTM was not evaluated).

Table 2

Evaluation Scores for Each Model, Normalized to 0-100 Scale

Model	BLEU-1	BLEU-2	BLEU-3	BLEU-4	METEOR
Visual Attention (ViT) (B)	65	47	37	23	42
Visual Attention (ViT) (G)	65	46	36	22	41
Visual Attention (DenseNet) (B)	59	41	32	19	38
Visual Attention (DenseNet) (G)	61	42	32	19	38
MobileNetV3-LSTM (G)	57	39	30	17	35
VGG16-LSTM (G)	36	22	17	9	30
Custom CNN-LSTM (G)	46	27	20	11	26

Note. (G) indicates greedy search. (B) indicates beam search.

The METEOR scores for the visual attention models are very good relative to the size and complexity of the Flickr30k dataset, however, the BLEU-4 scores range from passable for the visual attention models to not very good for the CNN-LSTM models. The fact that the METEOR scores are consistently higher than the BLEU-4 scores likely indicates that the captions are accurate, but tend to use different words than the reference captions. The top three models registered on Papers with Code evaluated against the Flickr30k test set have the BLEU-4

scores 15.7, 21.3, and, 30.1 and the METEOR scores 15.3, 20, 23 (Karpathy & Fei-Fei, 2015) (Cornia et al., 2018) (Zhou et al., 2019). Unfortunately, it is not possible to directly compare our scores to these because BLEU and similar metrics can vary greatly depending on implementation details such as choice of tokenizer and normalization techniques (Papineni et al., 2002), and this information is not available in the papers for these models.

Since we can't directly compare our models to other published models using objective metrics, as a subjective metric we ran a small set of images from the test set through both our DenseNet121 Visual Attention model and ViT GPT2, which is a popular pretrained model for image captioning (nlpconnect, 2023). Figure 10 shows a sample of the images tested and the predicted captions from each model.

Figure 10

Predicted Caption Comparison: DenseNet121 Visual Attention vs. ViT GPT2



Ours: two ballet dancers perform on stage

ViT GPT2: a man and a woman are dancing on stage



Ours: a bulldozer is driving down a dirt road

ViT GPT2: a yellow and black truck is on a dirt road



Ours: a woman with a white shirt is standing in front of a crowd

ViT GPT2: a woman with long hair and a red tie

From the small set of images we tested, it seems that the performance of the DenseNet121 Visual Attention model is comparable to results produced by ViT GPT2, but ViT GPT2 has an advantage in being able to generate more diverse descriptions than our model. Since the secondary goal of this project was to see if we could build an image captioning model with a much smaller size than contemporary models but with comparable performance, we also compared the size of the models (measured by file size of the model). The DenseNet 121 Visual Attention model is 97 MB, making it less than 1/10th of the size of ViT GPT2, which is 982 MB.

Conclusion and Next Steps

Looking to the future, there are many interesting avenues of investigation for improving and expanding this project. To start, we could add a temperature value to the model prediction logic to make the caption generation non-deterministic. Using our current methods, the model will always predict either the token with the highest probability (for greedy search) or the set of tokens with the highest joint probabilities (for beam search), but introducing temperature would allow for adjusting the predicted probabilities to pick tokens that are slightly less probable. This could improve the diversity of predicted captions and make them more human-like.

If feasible, it would be beneficial to train the same models on a bigger dataset, such as MS COCO. Typically, generative machine learning models are trained on enormous amounts of data, and the fact that all of the models are overfitting slightly-to-moderately suggests that these models have more learning capacity than the available data. Additionally, when testing the web app that we built as a demo for one of the visual attention models, we observed that the model tends to produce poorer quality captions for pictures captured with smartphones or images that would be uncommon in a collection of personal photography, such as wildlife scenes. We suspect this is because the dataset, curated in 2014, primarily consists of images from Flickr, which at the

time likely hosted primarily personal photography shot with traditional cameras. MS COCO is periodically updated, so training the models with this data could rectify this problem.

Another approach that could boost the performance of the models is augmenting the images to increase variation in the training data. One way to do this is by adding random transformations to the images before pre-computing the image features, such as flipping the images vertically or rotating them slightly. Another technique would be to add Gaussian noise to the images. The increased variation in the images from this process could help the models generalize to unseen images and avoid overfitting during training.

Finally, we could try other architectures not covered in this project, such as the architecture described in the 2016 paper *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention* (Xu et al.). This paper describes an alternative way of implementing visual attention from what was done in this project, by integrating an attention mechanism into the time step loop of an LSTM layer.

References

- Beddiar, R., & Oussalah, M. (2023). Chapter 12 - Explainability in medical image captioning. In J. Benois-Pineau, R. Bourqui, D. Petkovic, & G. Quénot (Eds.), *Explainable Deep Learning AI* (pp. 239–261). Academic Press.
- <https://doi.org/10.1016/B978-0-32-396098-4.00018-1>
- Young, P., Lai, A., Hodosh, M., & Hockenmaier, J. (2014). From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions. *Transactions of the Association for Computational Linguistics*, 2, 67–78.
- https://doi.org/10.1162/tacl_a_00166
- nlphuji. (2023). *Flickr30k* [Data set]. <https://huggingface.co/datasets/nlphuji/flickr30k>
- Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., ... Zheng, X. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems* [Software]. Available from <https://www.tensorflow.org/>
- Papers with Code - Image Captioning*. (n.d.). Retrieved December 7, 2024, from <https://paperswithcode.com/task/image-captioning>
- Hossain, M. Z., Sohel, F., Shiratuddin, M. F., & Laga, H. (2019). A comprehensive survey of deep learning for image captioning. *ACM Computing Surveys (CsUR)*, 51(6), 1-36.
- Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2021). *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale* (No. arXiv:2010.11929). arXiv. <https://doi.org/10.48550/arXiv.2010.11929>

- Vision Encoder Decoder Models.* (n.d.). Hugging Face. Retrieved December 7, 2024, from
https://huggingface.co/docs/transformers/v4.47.0/en/model_doc/vision-encoder-decoder
- nlpconnect. (2023). *nlpconnect/vit-gpt2-image-captioning* · Hugging Face.
<https://huggingface.co/nlpconnect/vit-gpt2-image-captioning>
- Wong, W. (2019, October 15). *What is Teacher Forcing?* Medium.
<https://towardsdatascience.com/what-is-teacher-forcing-3da6217fed1c>
- Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q. V., & Adam, H. (2019). *Searching for MobileNetV3* (No. arXiv:1905.02244). arXiv. <https://doi.org/10.48550/arXiv.1905.02244>
- Simonyan, K., & Zisserman, A. (2015). *Very Deep Convolutional Networks for Large-Scale Image Recognition* (No. arXiv:1409.1556). arXiv.
<https://doi.org/10.48550/arXiv.1409.1556>
- He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Deep Residual Learning for Image Recognition* (No. arXiv:1512.03385). arXiv. <https://doi.org/10.48550/arXiv.1512.03385>
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2023). *Attention Is All You Need* (No. arXiv:1706.03762). arXiv.
<https://doi.org/10.48550/arXiv.1706.03762>
- Zakka, C. (2023). *Positional Embeddings - The Large Language Model Playbook*.
<https://cyrilzakka.github.io/llm-playbook/pos-embed.html>
- Huang, G., Liu, Z., Maaten, L. van der, & Weinberger, K. Q. (2018). *Densely Connected Convolutional Networks* (No. arXiv:1608.06993). arXiv.
<https://doi.org/10.48550/arXiv.1608.06993>

Wightman, R. (2019). *PyTorch image models* [GitHub repository]. GitHub.

<https://doi.org/10.5281/zenodo.4414861>

Doshi, K. (2021, May 11). *Foundations of NLP Explained — Bleu Score and WER Metrics*.

Medium.

<https://towardsdatascience.com-foundations-of-nlp-explained-bleu-score-and-wer-metrics-1a5ba06d812b>

Avinash. (2024, August 7). LLM evaluation metrics — BLEU, ROGUE and METEOR

explained. Medium.

<https://avinashselvam.medium.com/llm-evaluation-metrics-bleu-rogue-and-meteor-explained-a5d2b129e87f>

Cornia, M., Baraldi, L., Serra, G., & Cucchiara, R. (2018). *Paying More Attention to Saliency:*

Image Captioning with Saliency and Context Attention (No. arXiv:1706.08474; Version 4). arXiv. <https://doi.org/10.48550/arXiv.1706.08474>

Karpathy, A., & Fei-Fei, L. (2015). *Deep Visual-Semantic Alignments for Generating Image Descriptions* (No. arXiv:1412.2306; Version 2). arXiv.

<https://doi.org/10.48550/arXiv.1412.2306>

Zhou, L., Palangi, H., Zhang, L., Hu, H., Corso, J. J., & Gao, J. (2019). *Unified Vision-Language Pre-Training for Image Captioning and VQA* (No. arXiv:1909.11059; Version 3). arXiv.

<https://doi.org/10.48550/arXiv.1909.11059>

Papineni, K., Roukos, S., Ward, T., & Zhu, W.-J. (2002). BLEU: A method for automatic evaluation of machine translation. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, 311–318.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhutdinov, R., Zemel, R., & Bengio, Y. (2016). *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention* (No. arXiv:1502.03044). arXiv. <https://doi.org/10.48550/arXiv.1502.03044>



Final Capstone Project:

**Diabetes Management System: Personalized blood glucose prediction and insulin
requirement system with Deep learning**

Group 9: Angel Benitez, Dina Shalaby, Gary Takahashi, Eyoha Mengistu

University of San Diego

AAI-590: Capstone Project

Anna Marbut

Monday, December 09, 2024

Introduction

Type 1 diabetes mellitus (T1D) is a global healthcare challenge, affecting 9.5% of the population worldwide (Mobasseri, 2020) and 0.4% in the United States (Fang, 2024). Management requires precise insulin dosing to avoid complications such as hypoglycemia, which impairs cognitive function, and hyperglycemia, which can result in diabetic ketoacidosis (Lizzo, 2023). While CGM and insulin pumps have improved glycemic control, predicting and preventing dangerous glucose fluctuations remains a critical unmet need. Long-term control, assessed through HbA1c levels, reduces risks of complications such as retinopathy and neuropathy, underscoring the importance of accurate glucose and A1c predictions (WHO, 2011).

Predictive systems for glycemic control address critical challenges in T1D management, reducing healthcare costs and improving quality of life. By integrating glucose forecasting with dietary insights, this project supports personalized medicine and empowers patients to make data-driven decisions about their health.

The proposed system in this paper benefits T1D patients using CGM and insulin pumps and individuals with brittle or refractory type 2 diabetes (Freckman, 2020). Healthcare providers and researchers can also leverage the system to refine treatment strategies.

This project leverages three complementary datasets: the DiaTrend Dataset providing CGM and insulin pump data from 54 participants (Eyth, 2023), the Tidepool Dataset offering three months of comprehensive CGM and insulin data with demographic metadata (Tidepool, 2024), and the Nutrition 5k Dataset containing 3492 meal photographs with nutritional information for

glycemic load prediction (Glycemic, 2024), and carbohydrate content estimation supporting insulin dosing in T1D patients (Deeb, 2017).

The project aims to develop a multi-component system that will do the following:

- a. Predict Blood Glucose levels; utilizing LSTM/GRU models to forecast glucose levels and recommend insulin doses based on patient-specific data.
- b. Predict HbA1c levels; analyzes glucose trends to estimate long-term control.
- c. Meal Analysis System; predicts carbohydrate content and glycemic load from meal photographs using CNNs and Vision Transformers.

Together, these components provide a holistic toolkit for improving glycemic control, increasing HbA1c target outcomes, and ultimately to reduce complications associated with diabetes.

Dataset Summary

This project leverages three datasets to address the comprehensive management of diabetes:

1. **DiaTrend Dataset:** Dartmouth Health developed longitudinal data from continuous glucose monitors (CGMs) and insulin pumps for 54 participants (Eyth, 2023). This dataset offers insights into real-world diabetes management by providing detailed variables such as blood glucose (BG) levels (mg/dl), insulin doses, and carbohydrate intake. Additional features include patient-specific parameters, such as insulin-to-carbohydrate ratios and HbA1c values, critical for long-term glucose control predictions.
2. **Tidepool Dataset:** The Tidepool proprietary dataset is part of the Big Data Project and consists of three months of data from CGM and insulin pump devices (Tidepool, 2024). It includes variables like glucose values, basal and bolus insulin rates, patient

demographics, and device usage metadata. This dataset is instrumental for developing models to predict real-time glycemic events and insulin dosing requirements.

3. **Nutrition 5k Dataset:** Provided by Google Research, this dataset comprises food images and associated nutritional labels for over 5,000 meals (Thames, 2021). Key data features include carbohydrate, protein, fat content, and total calories. This dataset enables carbohydrate content estimation, essential for meal-time insulin bolusing and glycemic load prediction from meal photographs, offering key insights into dietary impacts on glucose levels.

Together, these datasets address unique and complementary management domains, enabling the development of an integrated system for diabetes management.

While exploring our chosen datasets we discovered some data issues and had to implement some preprocessing methodologies to prepare our data for training.

The dataset had missing metadata fields, and device settings were excluded when 100% null or irrelevant to the predictive task. The Nutrition 5k dataset contained some metadata rows without corresponding images, and these were removed to ensure the integrity of the image-label pairs. We also found irregularities and noise in the DiaTrend dataset. Large outliers in glucose levels were handled by capping the values at the 1st and 99th percentiles to reduce noise without discarding valuable data. The Tidepool dataset contained formatting inconsistencies in the recorded time values, requiring standardization to a common datetime format. The Nutrition 5k dataset contained some ingredient labels that were essentially invisible, such as salt, olive oil, vinegar, etc. which the camera would not reliably detect. These items often dominated in the validation set, exacerbating class imbalance, which adversely affected model inference. These items, deemed “irrelevant” were excluded from the training dataset.

The origins of these data issues stem from device limitations, user errors, and the variability of day-to-day living conditions in real-world data collection. For instance, the intermittent nature of Tidepool's physical activity labels reflects practical challenges in consistently tracking exercise data.

Certain intervariable relationships were evident, such as the correlation in the DiaTrend dataset between glucose levels and insulin bolus doses, confirming the expected impact of pharmacologic interventions. There were also correlations between demographic variables and glucose trends, which although weak, suggest that there may be additional benefit to personalized management.

When looking at the Tidepool dataset, there also were strong correlations between basal insulin rates and bolus, indicative of active glucose level management. A weaker than anticipated correlation between glucose levels and physical activity labels might indicate issues with data acquisition quality. Lastly the Nutrition 5k dataset showed high correlation between carbohydrate content and total calories ($r = 0.87$) reinforcing the central role of carbs in glycemic load predictions.

Background Information

Managing diabetes effectively has been a focus of both academic research and commercial innovation. Numerous efforts have explored predictive models and automated systems for blood glucose management, meal analysis, and long-term glycemic control.

Looking at existing academic research, we see there are studies employing LSTM networks and hybrid architectures to demonstrate their ability to predict blood glucose trends effectively. These models use CGM data and insulin dosing history, addressing challenges such as data irregularities and variability in glucose responses (Zhang et al., 2024).

Fine-tuning insulin dosing to align with carbohydrate intake could significantly enhance glycemic control and improve quality of life. Current guidelines recommend that prandial insulin algorithms be tailored to the carbohydrate content of meals (Tascini, 2018). By leveraging food image-based carbohydrate content prediction, our approach seeks to validate this theory and offers a promising avenue for advancing T1D management.

Some researchers have also explored CNNs and Vision Transformers (ViTs) for food image analysis, achieving high accuracy in predicting macronutrient content and glycemic load (Hui, 2024). Ando et al, (2019) created DepthCalorieCam, a mobile application to estimate caloric content of various foods with limited success. Lastly, Google's work on Nutrition5k and studies integrating CGM data with demographic information (Nguyen, 2021) illustrate the feasibility of comprehensive machine learning solutions for diabetes care.

Transformer-based models have emerged as powerful tools for predicting Hemoglobin A1C (HbA1c), offering interpretability and scalability in analyzing long-term glucose trends (Acuna

et al., 2023). These methodologies are particularly effective in real-world applications, enabling automated meal assessments.

Zhang et al. (2024) employed multi-stage LSTM networks for real-time glucose trend prediction, achieving significant accuracy improvements over traditional statistical models.

Exploring the commercial efforts of some of these studies, we see companies like Omnipod and Tandem have integrated CGMs with insulin pumps to provide ML-triggered real-time alerts for glycemic events (Ciprich, 2024). Medtronic had used the Sugar.IQ system which predicted glucose levels in 60 minutes, but discontinued this in 2022 in favor of 5 minute monitoring with basal insulin rate adjustments (Medtronic, 2024). With regards to mobile applications, apps such as mySugr and Carb Manager assist in tracking glucose levels and dietary intake but lack the predictive capabilities and automation proposed in this project. There are also Food Recognition Systems similar to Google's work on the Nutrition5k dataset demonstrating the potential of deep learning for dietary analysis, laying the groundwork for innovative food tracking applications (Thames et al., 2021).

Despite advancements, existing solutions often lack integration, requiring manual inputs or focusing on specific aspects of diabetes management. This project aims to address these gaps by combining predictive glucose modeling, long-term HbA1c estimation, and automated meal analysis in a single system.

To address these integration challenges, our project employs a carefully selected combination of advanced machine learning models. At the core of our glucose prediction system are Long Short-Term Memory (LSTM) networks, specialized RNNs designed to excel at capturing

temporal dependencies in sequential data. Their sophisticated gating mechanisms effectively address common challenges like vanishing gradients, making them particularly suitable for predicting both blood glucose trends and HbA1c levels from continuous glucose monitoring data.

For the analysis of meal-related data, we implemented two complementary vision-based approaches. Convolutional Neural Networks (CNNs), specifically MobileNetV3Large and YOLOv8 architectures, extract hierarchical spatial features from food images with high accuracy. These models excel at identifying food items and estimating carbohydrate content. Additionally, Vision Transformers (ViTs) provide a different perspective by dividing images into patches and applying self-attention mechanisms, enabling effective analysis of both global and local features for predicting glycemic load and macronutrient content.

The suitability of these methods is demonstrated through their specific strengths. LSTMs effectively model both short-term fluctuations and long-term trends in glucose levels, with their sequential processing capabilities naturally aligning with the temporal nature of CGM data. The vision-based models complement this by excelling at food image analysis, with ViTs providing global context while CNNs capture detailed local features, ensuring comprehensive nutritional predictions

These well known approaches and existing projects validate the methodologies chosen for this study and provide benchmarks for model performance and system design.

Experimental Methods

This project employs three core machine learning models, each tailored to address specific tasks in diabetes management:

The LSTM Model for Blood Glucose Prediction features two sequential LSTM layers, each with 128 hidden units, followed by a dense layer with 64 nodes with ReLU activation. A final dense layer outputs the predicted blood glucose value as a continuous variable. Dropout layers placed after each LSTM layer prevent overfitting. The input features include normalized blood glucose values, rolling averages, temporal features like time of day, and patient demographics. The architecture captures short-term glucose fluctuations and long-term trends, addressing the temporal nature of CGM data. The output presents continuous glucose value predictions, optimized for accuracy in both normal ranges and hypo-/hyperglycemic events.

For the meal analysis component, Vision Transformer (ViT) models and the CNN-based YOLOv8 models were used. These include convolutional layers for feature extraction, followed by dense layers for regression tasks. The YOLOv8 architecture is pre-trained on the COCO8 dataset, with additional dense layers fine-tuned for carbohydrate prediction. The ViT divides food images into patches and applies self-attention mechanisms to extract global and local features. Dense layers are added for regression to predict macronutrients and glycemic load. For the multilabel classification approach, we used MobileNetV3Large, a reduced-resource modern CNN with high performance, also well-suited for image recognition tasks. It also comes pre-trained (on ImageNet), rendering it suitable for transfer learning.

These models require input food images to be resized to 224×224 pixels. Data augmentation, such as rotations and brightness adjustments, was implemented to increase model robustness. The model was used for inference to predict carbohydrates, proteins, fats, and glycemic load. Our other LSTM Model for HbA1c prediction was a 50-unit bidirectional LSTM that processed glucose trends both forward and backward. This was followed by a dropout layer with a rate of 0.3, another 30-unit LSTM layer, and a dense layer with 25 nodes using ReLU activation. The final output layer predicts HbA1c levels as a continuous variable. Its input features include time-series glucose data, rolling statistics, wavelet coefficients, and time spent in different glycemic ranges. This model outputs predicted HbA1c values, representing long-term glucose control. Each model was trained using structured procedures to ensure robust learning and generalization.

All datasets were split into training (80%) and validation (20%) subsets. For CNNs and ViTs, the Nutrition5k dataset was split based on dish IDs to prevent data leakage. LSTM models used time based splitting to preserve temporal order. Models were trained for 10–80 epochs depending on convergence behavior, with early stopping implemented to prevent overfitting. Batch sizes ranged from 16 (for ViTs) to 64 (for LSTMs), balancing computational efficiency and gradient stability.

The selection of loss functions was tailored to each model's specific task and requirements. For our LSTM models, we implemented Mean Squared Error (MSE) for regression tasks, as it's quadratic nature heavily penalizes larger errors, which is particularly crucial for medical applications where accuracy is paramount. The CNN and ViT models used Mean Absolute Error (MAE) for predicting carbohydrate content, ensuring interpretability of predictions by directly

measuring deviations from true values. The MobileNetV3 implementation leveraged transfer learning, with the freezing layer empirically set. Labels were binarized for multilabel classification and selected binary cross entropy as the loss function, to effectively handle multiple class assignments.

For hyperparameter tuning we used grid search across our LSTM models. Similar grid search approaches for our CNN and ViT Models focused on batch sizes, learning rates, and a number of dense layer configurations.

To prevent overfitting, we implemented various regularization techniques across our models. Dropout layers were incorporated into all models and in addition L2 regularization was applied to dense layers in the ViT and YOLOv8 models. Although we explored batch normalization for LSTM models, we ultimately excluded it due to minimal performance improvements. The MobileNetV3Large's trainable dense layers utilized a single dropout layer combined with L2 kernel regularizers. Our learning rate optimization incorporated step decay schedules, reducing rates by a factor of 0.1 when validation loss plateaued for three epochs, enabling more precise model fine-tuning in later training stages. Feature engineering played a crucial role, with LSTM models benefiting from wavelet coefficients and cyclic temporal encodings, while our CNN and ViT models leveraged data augmentation techniques to improve generalization across diverse food images.

Results and Discussion

Tidepool Dataset

In capturing results for this dataset we started with a baseline simple linear regression model then compared it to our trained LSTM model. The LSTM model demonstrated significant

improvements in predicting glucose levels when compared to the baseline linear regression model. The Mean Squared Error (MSE) was reduced from 0.7131 with the linear regression model to 0.0642 with the LSTM model, illustrating the latter's superior ability to minimize prediction errors, particularly with respect to outlier glucose readings, which are crucial for managing hypo- and hyperglycemic events (Perlmutter et al., 2008). The reduction in MSE indicates that the LSTM model was more effective at predicting glucose values accurately, which is critical in preventing dangerous health episodes.

The LSTM model also achieved an R^2 score of 0.9930, compared to 0.9224 for the linear regression model, indicating a much better fit to the data and a higher capability for explaining the variance in glucose levels. This enhanced explanatory power highlights the LSTM's strength in capturing the temporal and non-linear dependencies present in the data. The Mean Absolute Error (MAE) was recorded at 0.1254, and the Mean Absolute Percentage Error (MAPE) was 0.0179, indicating that the model's predictions, on average, were within 1.79% of actual values, making it a reliable tool for managing diabetes.

The residuals, representing the difference between actual and predicted glucose values, were centered around zero, indicating low bias and validating that the model was not overfitting. The residual distribution in Figure 1 showed a roughly symmetric pattern, further supporting the model's robustness and consistency across different glucose levels. This insight is crucial for real-world deployment, where the ability to generalize across various patient profiles is paramount.

Figure 2 presents the comparison between actual and predicted glucose values for the LSTM model. The majority of the points lie close to the line of equality, suggesting that the LSTM

model delivers strong predictive performance, especially for outlier glucose readings, which is essential for effective management of hypo- and hyperglycemic conditions.

Building on these results, future iterations of the system may explore the incorporation of advanced models such as the Informer ProbSparse self-attention mechanism (Zhou et al., 2021) or ensemble methods to further enhance predictive accuracy and robustness.

The LSTM model's ability to predict significant glucose variations accurately forms a key component of a holistic diabetes management system. By integrating this model with other components, such as long-term HbA1c prediction and dietary analysis through food image recognition, the system provides a comprehensive toolkit for managing diabetes. The glucose prediction model contributes to real-time monitoring, the HbA1c model offers insights into long-term trends, and food image analysis aids in dietary management, all of which collectively empower patients to take proactive control of their health.

Nutrition5k Dataset

The Nutrition5k dataset was analyzed using two different approaches:

In our first method, an end-to-end approach directly inferring the carbohydrate content from food images using YOLOv8 and Vision Transformer (ViT) models. YOLOv8 demonstrated strong performance, achieving a validation MAE of 0.07 after optimization and a 50% improvement in validation loss through hyperparameter tuning. The predictions for carbohydrate content were highly accurate, with an average deviation of just $\pm 5\text{g}$. The ViT Model performed slightly less effectively than YOLOv8, with higher validation loss and slower convergence. Figure 4 provides a detailed analysis of the model's prediction performance.

Optimization efforts for the YOLOv8 focused on tuning batch sizes, learning rates, and dense layer configurations, yielding optimal parameters of a batch size of 16, a learning rate of 0.0001, and dense layer units of (512, 256, 128). The adjustments made resulted in closer alignment between predicted and actual carbohydrate values.

The analysis encountered several challenges that affected model performance. One significant limitation in the Nutrition5k dataset was the lack of diversity, as it was based on items common in a California diet. Expansion of the dataset to include items from other cuisines would improve generalizability. Data handling issues included unequal label lengths, missing images, and unnormalized numerical variables. These issues were effectively mitigated by linking total dish nutritional values as key labels, filtering the dataset to include only complete entries, and applying a standard scaler to normalize numerical variables. These preprocessing steps were essential to ensure data integrity and improve the reliability of the model.

Another challenge was volatility of the validation curves. The validation jumps could be mitigated by alternative learning rate schedules and enhanced data augmentation techniques to stabilize training and improve model convergence. Addressing these challenges would enhance this model as a practical and scalable tool to manage insulin dosing precision.

The second method framed this problem as multilabeled classification, where each dish could contain multiple different ingredients. A dictionary was created to associate dish images with the metadata of ingredients of each dish, while target labels were binarized. The MobileNetV3Large CNN was trained to recognize the ingredients in each image, and then predict on a validation

dataset. The nutritional information of each of the predicted ingredients was obtained from a separate dictionary, and the glycemic index, and glycemic load were calculated and displayed.

Results:

- Standard accuracy was high (0.974) due to high true-negative prediction, which was likely an outcome of class imbalance. The more strict subset accuracy (where even one ingredient mismatch flags a prediction error) was 0.302. The more forgiving Hamming score, which simply performs XOR on y_{true} and y_{pred} was 0.987, and the area under the precision-recall curve was 0.72.
- A custom F1 score was created which was essentially the same as a min-weighted F1-score, however it was calculated at the end of each epoch, and counted only relevant classes, to avoid penalty where there was no ingredient in the predicted or actual label class. The final validation F1-score was 0.6409.

Challenges:

- Class imbalance was significant, rendering the standard accuracy metric as misleading.
- The quality of the images was deliberately inferior. As the authors described: The images were intended to mimic the lighting seen in campus cafeterias, and intentionally included dishes that were ambiguous and sometimes partially occluded by other ingredients.
- Invisible Ingredients: Certain ingredients, such as salt, vegetable oils, vinegar, etc. cannot be detected in images, thereby rendering their presence irrelevant. It was necessary to review the ingredients list and decide which items were not likely to be detectable by the

camera, and would only further class imbalance and complicate the analysis. These were excluded from the metadata analysis, and required an extra processing step.

Future directions and suggestions for improvement

- Training on better quality images.
- Explore other larger vision architectures for possible improved performance
- Consideration of experimental and proposed techniques for dealing with class imbalance, such as MLSMOTE (Charte, 2015).

DiaTrend Dataset

The DiaTrend dataset was pivotal in predicting HbA1c levels, with the LSTM model achieving impressive metrics: a Mean Squared Error (MSE) of 0.0623, a Mean Absolute Error (MAE) of 0.1731, and an R² score of 0.9332. Scatter plots comparing predicted and actual HbA1c values indicated close alignment, though a slight increase in variability was observed for higher HbA1c levels (Figure 3). Feature importance analysis using SHAP highlighted key contributors, such as normalized glucose levels, rolling statistics (mean and standard deviation), and wavelet coefficients, underscoring the model's ability to capture long-term glucose trends effectively.

The model demonstrated strong reliability in HbA1c predictions across a range of values, proving its potential for integration into real-time diabetes management systems. Its strengths included a high degree of generalizability across subjects and strong explanatory power, as reflected by the high R² score. However, some limitations were evident. Predictions for higher HbA1c values showed greater deviations, likely due to underrepresentation in the training data.

Additionally, class imbalance, with sparse data for certain HbA1c ranges, affected the model's ability to predict extreme cases accurately.

Future improvements will focus on addressing these challenges by leveraging techniques like SMOTE to handle data imbalance and exploring additional features, such as lifestyle factors, to enhance predictions. Hybrid architectures, including transformers, will also be considered to improve performance, particularly for outlier cases.

Across the three datasets analyzed, advanced machine learning methods proved highly effective in addressing critical aspects of diabetes management. The Tidepool dataset supported accurate short-term glucose predictions with LSTM models, while the Nutrition5k dataset enabled carbohydrate estimation using YOLOv8 and glycemic load predictions via MobileNetV3Large, assisting in meal-time insulin optimization. The DiaTrend dataset provided reliable long-term glucose trend analysis for HbA1c prediction.

Although challenges such as missing data, dataset limitations, and variability in extreme cases remain, these models have shown significant potential for real-world application. Future work will emphasize refining the models, expanding the datasets, and integrating these components into a comprehensive diabetes management system to further enhance patient care.

Building on our system and the model performance, we've identified a few key areas for future improvement. Dataset expansion presents a significant opportunity for improvement, particularly through incorporating more diverse data such as additional cuisine types in Nutrition 5k and

extended monitoring periods for both Tidepool and DiaTrend datasets, while addressing class imbalance through synthetic data generation techniques like SMOTE or GANs.

Our model refinement strategy would explore advanced architectures, including Informer ProbSparse self-attention mechanism (Zhou et al., 2021) and hybrid transformer-LSTM models for improved temporal predictions, alongside the implementation of ensemble methods to leverage strengths across different model types.

We envision developing a unified pipeline to synchronize insights across all datasets, creating seamless interactions between dietary, glucose, and long-term control components. Additionally, incorporating domain-specific constraints such as safety thresholds for insulin dosing, would ensure reliable and clinically acceptable outputs in the real world.

The proposed system has the potential to revolutionize diabetes management by providing a personalized, data-driven solution. The Practical applications span multiple areas, empowering patients with actionable insights into glucose trends, dietary impacts, and long-term glycemic control. This would also assist healthcare providers in designing tailored treatment plans and potentially reducing healthcare costs by improving glycemic control and minimizing adverse events.

To transition from research to real-world deployment, we've identified several critical implementation steps. Regulatory compliance will require adherence to healthcare standards like HIPAA and GDPR for data privacy, alongside FDA approval validation for clinical use. The system deployment phase will focus on developing user-friendly interfaces for both patients and clinicians, while optimizing computational requirements for real-time predictions on edge devices such as smartphones and insulin pumps.

The final phase will involve comprehensive pilot studies and user feedback collection to evaluate the system's real-world performance and refine its features based on end-user experiences.

By addressing these steps, the system can transition from an academic prototype to a scalable, impactful solution, improving quality of life for individuals with diabetes worldwide. This systematic approach to deployment ensures both technical excellence and practical utility in real-world healthcare settings.

Visualizations

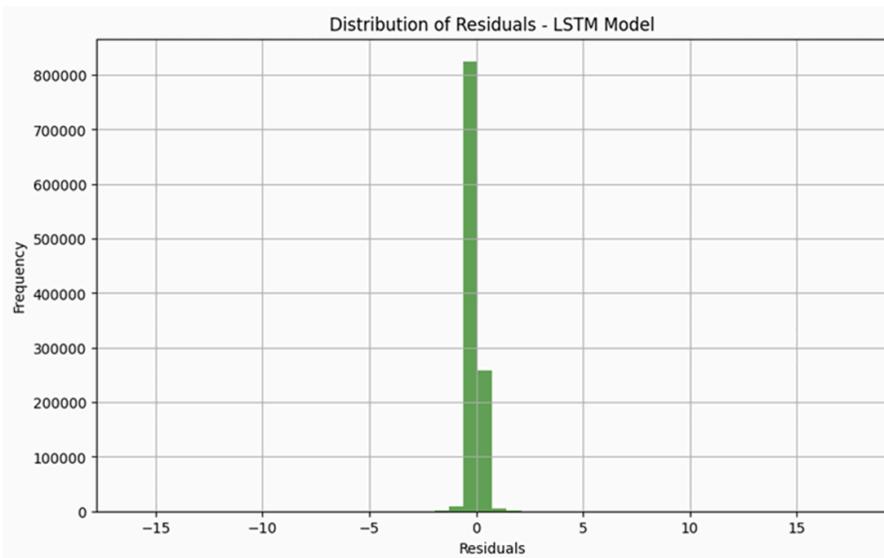


Figure 1: A graph showing the distribution residuals between actual and predicted glucose values. Centering around zero, with a sharp peak, indicating low bias in the model's predictions. This roughly symmetric distribution validates that the model is not overfitting and is performing well across different glucose levels.

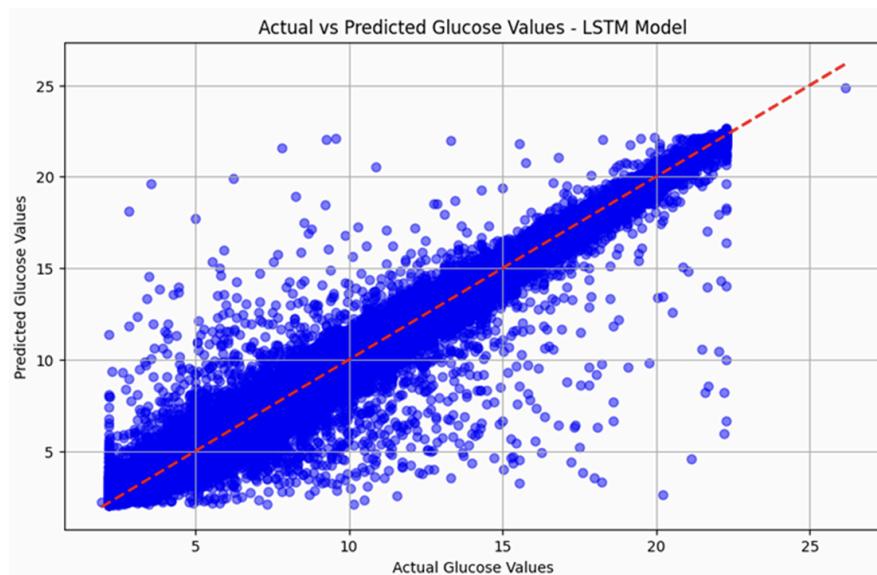


Figure 2: A graph showing the Actual vs. Predicted Glucose Values for the LSTM Model. Most of the data points lie close to the line of equality, suggesting good predictive performance. This plot supports our claim of improved model accuracy over the baseline, especially for outlier glucose readings, which is crucial for managing hypo- and hyperglycemic conditions.

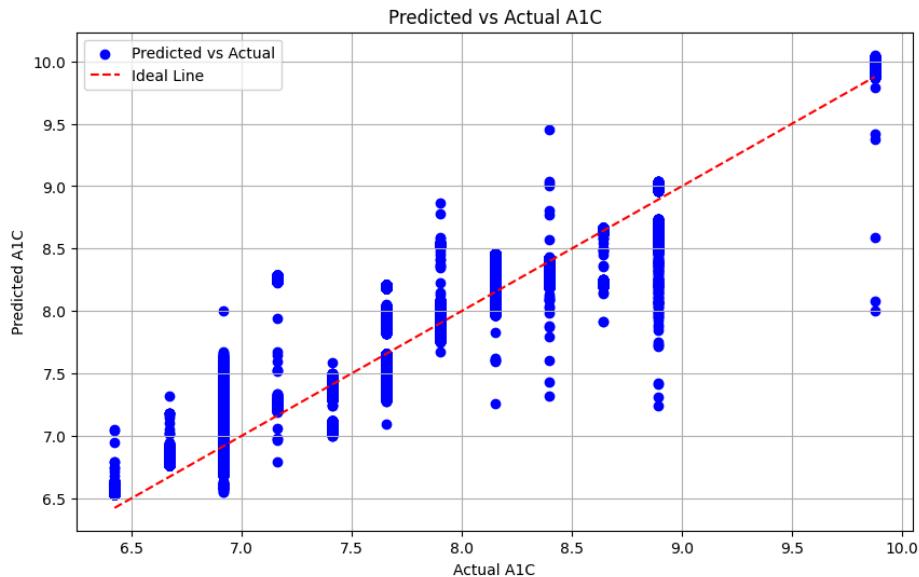


Figure 3: This scatter plot displays the relationship between actual and predicted HbA1c values for the LSTM model. The red dashed line represents the ideal line of equality, where predictions match the actual values. The concentration of points near the line indicates strong predictive accuracy, while slight deviations, particularly at higher HbA1c ranges, suggest minor variability in the model's performance for extreme cases. This visualization underscores the model's capability in accurately predicting HbA1c trends, contributing to improved long-term glycemic control.

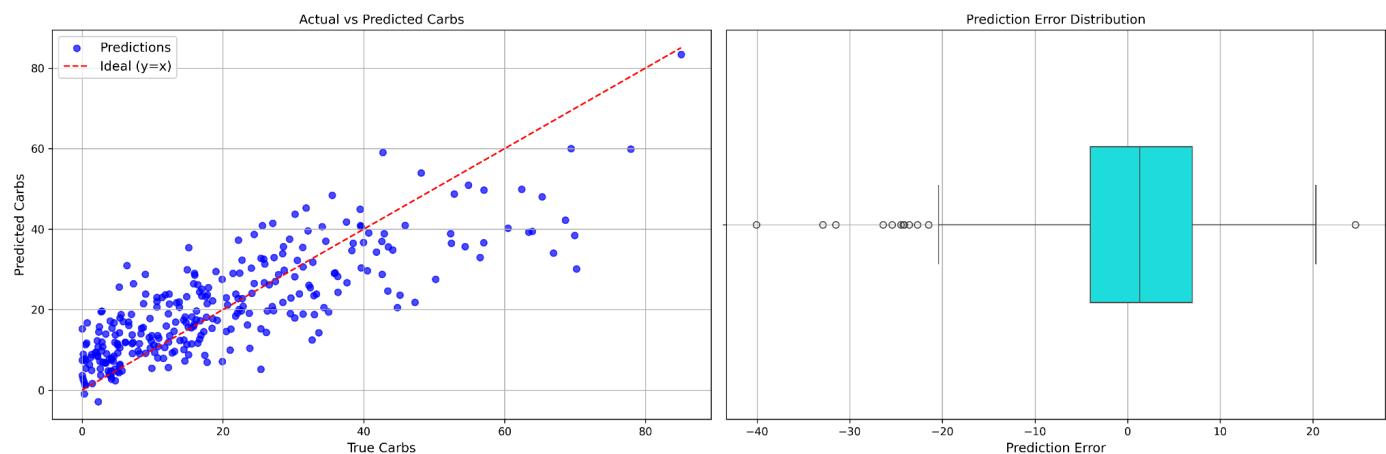


Figure 4: The scatter plot shows the relationship between actual and predicted carbohydrate values, with the red dashed line ($y=x$) representing ideal predictions. While the majority of points are clustered near the ideal line, there is some noticeable spread, particularly at higher true values, indicating that the model's predictions deviate more as carbohydrate levels increase. This suggests some heteroscedasticity. The box plot of prediction errors further reveals that the errors are generally centered around zero, indicating minimal systematic bias. However, the presence of a few outliers on both the positive and

negative sides highlight instances where the model struggled to predict accurately. The interquartile range (IQR) shows that the middle 50% of the errors are relatively small, but the overall error spread suggests room for improvement in reducing variance and addressing edge cases. This can be achieved by adding more data samples at the low and high ends of the carb food dish distribution.

References

- Acuna, E., Aparicio, R., & Palominpo, V. (2023). Analyzing the Performance of Transformers for the Prediction of the Blood Glucose Level Considering Imputation and Smoothing. *Big Data Cogn Comput*, 7(1), 41. DOI: 10.3390/bdcc7010041
- Ando, Y., Ege, T., Cho, J., Yanai, K. (2019). DepthCalorieCam: A mobile application for volume-based food calorie estimation using depth cameras. In Proceedings of the 5th International Workshop on Multimedia Assisted Dietary Management, pp. 76–81
doi:10.1145/3347448.3357172
- Charte, F., Rivera, A.J., del Jesus, M.J., Herrera, F. (2015). MLSMOTE: Approaching imbalanced multilabel learning through synthetic instance generation. *Knowledge-Based Systems* 89:385-397. doi:10.1016/j.knosys.2015.07.019
- Ciprich, A. (2024). A Comprehensive Guide to Choosing an Automated Insulin Delivery System: 2024 Update.
<https://www.t1dnutritionist.com/post/a-comprehensive-guide-to-choosing-an-automated-insulin-delivery-system-2024-update>
- Deeb A, Al Hajeri A, Alhmoudi I, Nagelkerke N. Accurate Carbohydrate Counting Is and Important Determinant of Postprandial Glycemia in Children and Adolescents With Type 1 Diabetes on Insulin Pump Therapy. *Journal of Diabetes Science and Technology*.
doi:[10.1177/1932296816679850](https://doi.org/10.1177/1932296816679850)
- Eyth, E., Naik, R. (2023). In: StatPearls [Internet]. Treasure Island (FL): StatPearls Publishing; 2024 Jan-. Available from: <https://www.ncbi.nlm.nih.gov/books/NBK549816/>
- Fang, M., Wang, D., Selvin, E. (2024). Prevalence of Type 1 Diabetes Among US Children and Adults by Age, Sex, Race, and Ethnicity. *JAMA*; 331(16):1411–1413.
doi:10.1001/jama.2024.2103
- Freckmann, G., Buck, S., Waldenmaier, D., Kulzer, B., Schnell, O., Gelchsheimer, U., Ziegler, R., Heinemann, L. (2020). Insulin Pump Therapy for Patients With Type 2 Diabetes Mellitus: Evidence, Current Barriers, and New Technologies. *J Diabetes Sci Technol*. 2021

Jul;15(4):901-915. doi: 10.1177/1932296820928100. Epub 2020 Jun 1. Erratum in: J Diabetes Sci Technol. 2021 Jun 16:19322968211027984. doi: 10.1177/19322968211027984.

Glycemic Load. (2024). Glycemic Load Explained: Definition, Formula, Benefits, and Examples. Retrieved from <https://glycemic-index.net/glycemic-load/>

Hui, W., Haixia,T., Ronghui, J., Liyan, M., Ling, Y., Jingyao, C., Feng, L. (2024). Nutritional composition analysis in food images: an innovative Swin Transformer approach. Frontiers in Nutrition 11. doi: 10.3389/fnut.2024.1454466

<https://www.frontiersin.org/journals/nutrition/articles/10.3389/fnut.2024.1454466>

Lizzo, J.M., Goyal, A., & Gupta, V. (2023). Adult Diabetic Ketoacidosis. In StatPearls [Internet]. Treasure Island (FL): StatPearls Publishing. Retrieved from <https://www.ncbi.nlm.nih.gov/books/NBK560723/>

Medtronic. (2024). Sugar.IQ discontinuation.

<https://www.medtronicdiabetes.com/products/sugar.iq-diabetes-assistant>

Mobasseri, M., Shirmohammadi, M., Amiri, T., Vahed, N., Hosseini Fard, H., Ghojazadeh, M.. Prevalence and incidence of type 1 diabetes in the world: a systematic review and meta-analysis. (2020). Health Promot Perspect; 10(2):98-115. doi: 10.34172/hpp.2020.18. Erratum in: Health Promot Perspect. 2024 Jul 29;14(2):202-205. doi:10.34172/hpp.43143.

Nguyen, T., Wang, X., & Zhao, Y. (2021). Multi-Modal Predictive Models of Diabetes Progression: A Comprehensive Approach Using CGM and Wearable Data. arXiv. Retrieved from <https://arxiv.org/abs/1907.12175>

Perlmutter, L. C., Flanagan, B. P., Shah, P. H., & Singh, S. P. (2008). Glycemic control and hypoglycemia: is the loser the winner?. Diabetes care, 31(10), 2072–2076.
<https://doi.org/10.2337/dc08-1441>

Tascini, G., Berioli, M. G., Cerquiglini, L., Santi, E., Mancini, G., Rogari, F., Toni, G., & Esposito, S. (2018). Carbohydrate Counting in Children and Adolescents with Type 1 Diabetes. *Nutrients*, 10(1), 109. <https://doi.org/10.3390/nu10010109>

Thames, Q., Karpur, A., Norris, W., Xia, F., Panait, L., Weyand, T., & Sim, J. (2021). Nutrition5k: Towards automatic nutritional understanding of generic food. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 8903-8911).
<https://arxiv.org/abs/2103.03375>

Tidepool. (2024). <https://www.tidepool.org/>

Turpin, C., Catan, A., Guerin-Dubourg, A., Debussche, X., Bravo, S., Álvarez, E., Van Den Elsen, J., Meihac, O., Rondeau, P., & Bourdon, E. (2020). Enhanced Oxidative Stress and Damage in Glycated Erythrocytes. *PLOS ONE*, 15(7), e0235335. DOI: 10.1371/journal.pone.0235335

WHO. (2011). Use of Glycated Haemoglobin (HbA1c) in the Diagnosis of Diabetes Mellitus: Abbreviated Report of a WHO Consultation. Geneva: World Health Organization. Retrieved from <https://www.ncbi.nlm.nih.gov/books/NBK304271/>

Zhang, Z., Imtiaz, H., & Lo, B. (2024). Long-Term HbA1c Prediction Using Multi-Stage CGM Data Analysis. *SLAS Technology*. Retrieved from <https://slas-technology.org/article/S2472-6303%2824%2900103-1/fulltext>

Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H., & Zhang, W. (2021, May). Informer: Beyond efficient transformer for long sequence time-series forecasting. In Proceedings of the AAAI conference on artificial intelligence (Vol. 35, No. 12, pp. 11106-11115). <https://ojs.aaai.org/index.php/AAAI/article/view/17325>

**SepsiSCT: A Stacked Convolutional Transformer Model
for Early Sepsis Detection in ICU Patients**

Group 3: Tyler Foreman, Ahmed Ahmed, Eric Barnes, Ryan Laxamana
University of San Diego
AAI-590: Capstone Project
Professor Anna Marbut
Monday, December 9, 2024

Introduction

This project addresses the critical challenge of predicting sepsis onset in ICU patients by applying machine learning techniques to clinical data that is commonly available in the ICU. Sepsis, a severe inflammatory response to infection, can progress quickly and is a leading cause of mortality in intensive care settings (Singer et al., 2016). Early detection of sepsis is essential, as timely medical intervention can drastically reduce mortality rates and improve patient outcomes (Fleischmann et al., 2016). Our goal is to develop a predictive model capable of accurately identifying the onset of sepsis at least six hours before clinical symptoms are observed, giving healthcare providers essential lead time for intervention.

The model would be deployed as part of an early warning system for healthcare professionals in ICU environments, where rapid response to early indicators of sepsis can prevent the progression to septic shock and other severe complications. The dataset selected for this project is the Prediction of Sepsis dataset from the PhysioNet Computing in Cardiology Challenge 2019, which includes extensive time-series physiological data, such as vital signs, lab results, and demographic information critical to sepsis prediction (Reyna et al., 2019). In a live system, the model would process real-time data from ICU patient monitoring devices, allowing for continuous sepsis risk assessment and timely alerts for clinicians.

The ultimate goal of this project was to develop a reliable, high-performing tool for early sepsis detection. By developing this predictive model, we aim to contribute to healthcare AI by offering a practical tool for ICU settings and advancing the understanding of Machine Learning (ML) and Deep Learning (DL) techniques for critical care applications. This project's findings could help improve the use of predictive modeling in healthcare, especially for conditions

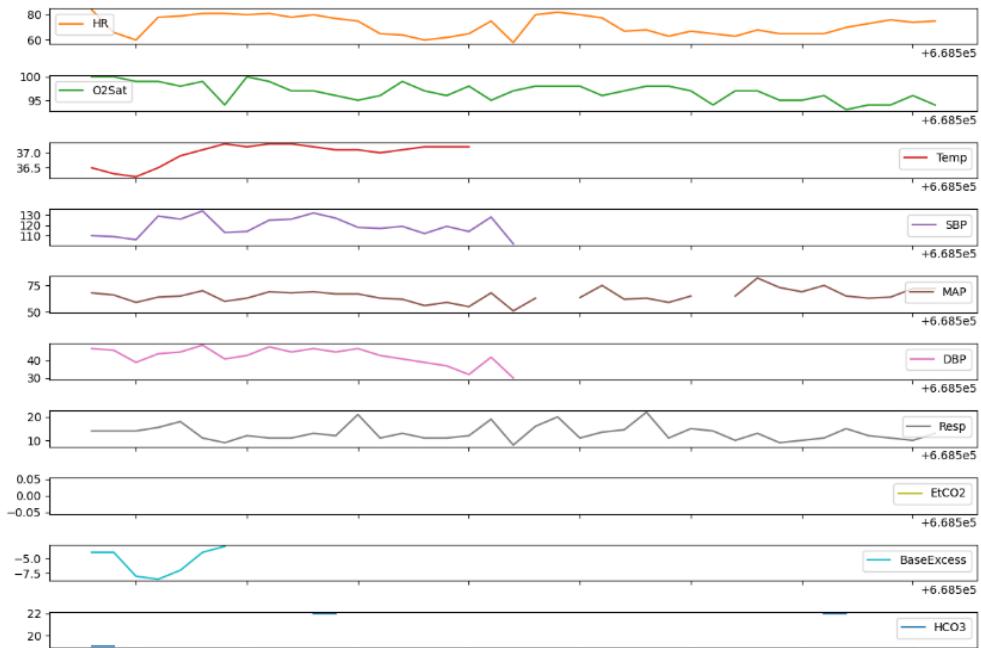
requiring rapid response, and influence future research on sepsis and other time-sensitive illnesses.

Data Summary

The dataset consisted of time-series samples of clinical and laboratory readings grouped to a unique patient identifier. There were 1,552,210 total samples related to 40,336 unique patients (Reyna et al., 2019). Each sample provided 40 explanatory variables, with 8 continuous (float) variables related to clinical readings of the patient's vital signs at the time step, 26 continuous (float) variables corresponding to readings obtained from laboratory samples and 6 variables that capture the patient's demographic characteristics, including age and gender. Each sample was captured at an hourly time step and included a target binary variable that indicates whether the patient developed sepsis at the time step or not.

Figure 1

Sample patient time series readings of vital signs and some laboratory variables



Upon initial interrogation, it was clear that the dataset had a high prevalence of missing data at given timesteps. This created some initial concern about the quality of the dataset overall. However, further inspection revealed that the sparse readings were largely isolated to the 26 variables corresponding to the laboratory samples. This sparsity is to be expected, as these readings are generally taken far less frequently in an ICU setting than the vital signs, which are streaming in automatically from monitoring devices. While less prevalent, the missing data did occur in the vital sign readings as well, meaning that all continuous variables would require some level of imputation in order to be useful for downstream modeling tasks. This imputation was conducted to capture a “last reading” of the variable and carry it forward using a forward fill technique. In addition, patients that had greater than 30% of their vital sign variable data missing were removed.

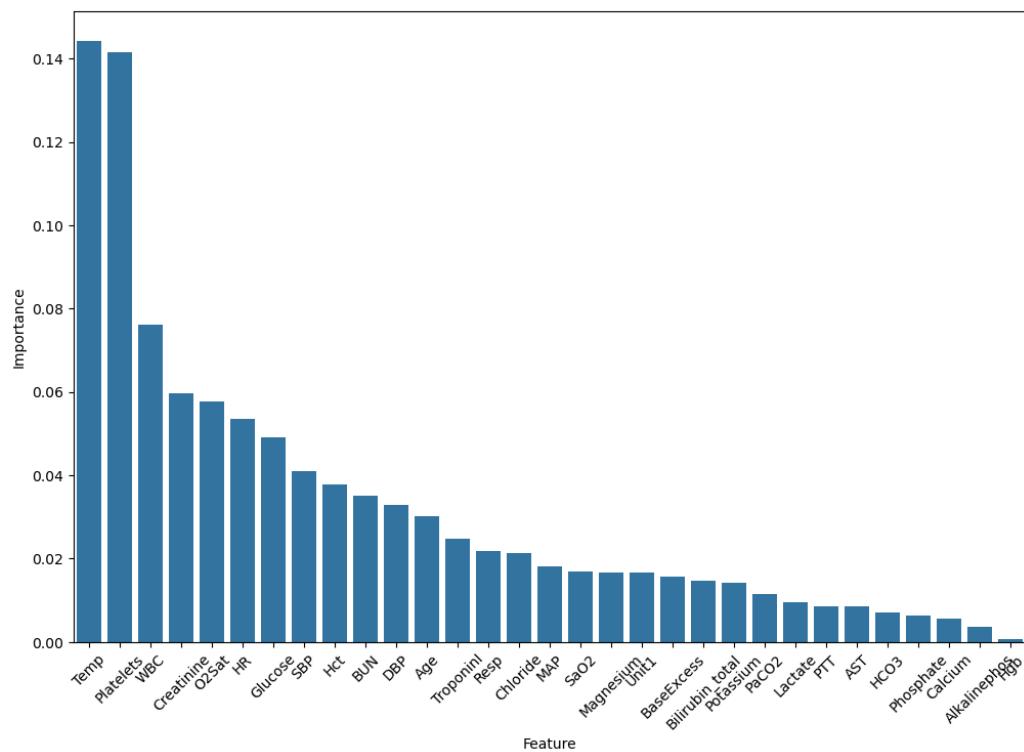
Further analysis was conducted to analyze the correlation characteristics between the input variables and the target in order to support feature selection. This produced evidence of moderate correlation between some variables but did not produce any definitive results aside from some cross correlation between variables that may be redundant, such as blood pressure readings (redundant variables were removed from the dataset). As a result, additional techniques were employed to determine the variables that were most relevant to predicting sepsis outcome.

The chosen approach was a measurement of feature importance, which was obtained by training a Decision Tree Classifier (DTC) on a summarized version of the dataset. The dataset was rolled up to the patient level and the DTC was trained to classify the sepsis outcome of the patient. This enabled the generation of a feature importance report, displayed in Figure 2, which provided a ranking of features deemed important by the model for determining the sepsis

outcome and those that were not. This information was used to conduct feature selection, reducing the total variables from 40 to 21.

Figure 2

Feature importance ranking used as input for feature selection



Another observation made during the interrogation of the data was that the length of stay (number of time steps) for each patient varied across a fairly large range of 8 hours minimum to 336 hours maximum with an average stay of approximately 38 hours. As mentioned previously, the ability to predict sepsis at least six hours in advance of diagnosis has been shown to improve patient outcomes (Fleischmann et al., 2016). In support of this objective, the patients would need to be filtered to those with a sufficient number of time steps to support a lookback window of at least 48 hours (a hyperparameter tuned through experimentation) and a prediction horizon of 6 hours, so 54 hours total. The dataset was filtered to remove patients that did not meet this criteria. This resulted in a smaller dataset, comprising a total of 3,882 patients.

The final issue observed in the dataset was a significant class imbalance in the target variable (sepsis outcome). This is to be expected, as most patients in the ICU do not actually develop sepsis. However, this imbalance would need to be handled to ensure that our model does not become biased towards predicting negative sepsis outcomes. The team decided that rather than perform this balancing on the dataset itself (ie: via over- or under-sampling), the model would be weighted to handle the imbalance instead.

With the dataset cleaned, feature-selected and filtered to patients with suitable data density and time series lengths, two additional preparation tasks were performed. First, the continuous features were scaled using a standard scaler and our single categorical feature (sex) was one-hot encoded. Second, derived features were created for each of the continuous lab variables to capture the “lag” in the last time step where the value was updated. The goal of these features is to help the model place the appropriate weight on the value of these readings as they progress through time without regular updates.

Finally, a novel feature, the Deterioration Index (DI), was engineered to encode changes in overall patient health over time. The DI was a calculated score designed to monitor changes in a patient’s vital signs and lab values, enabling the detection of rapid trends or deviations that may signal clinical deterioration, such as the onset of sepsis. Effectively, the DI encodes medical knowledge that a human doctor or nurse may apply in a clinical setting in order to detect degradation in a patient’s overall condition, potentially leading to sepsis. The feature also incorporates a rate of change from the previous time step, providing time-based trend information as to whether a patient is deteriorating or improving. In the case of the improving condition, where a patient moves from a moderate or severe condition back to normal, the score for the feature at that time step is -1, which reduces the overall index. A higher DI value

indicates a higher rate of overall deterioration in the patient's condition. Example score calculations for a subset of the features are illustrated in Table 1.

Table 1

Feature level scoring examples for rollup into the Deterioration Index

Feature	Normal (0 pts)	Moderate Condition (+1 pts)	Severe Condition (+2 pts)
Heart Rate	70-100 bpm	100-120 bpm (mild tachycardia) -or- 60-70 bpm (mild bradycardia)	> 120 bpm (severe tachycardia) -or- < 60 bpm (severe bradycardia)
Temperature	36-37.9°C	38–39°C (mild fever) -or- 35–35.9°C	> 39°C (severe fever) -or- < 35°C (hypothermia)
Platelets	$\geq 100 \times 10^3 \mu\text{L}$	$50–100 \times 10^3/\mu\text{L}$	$< 50 \times 10^3/\mu\text{L}$

Each score was then weighted for clinical importance using the feature importance derived from the DTC and summed into a composite index as seen in Equation (1).

$$\text{Deterioration Index} = \sum (\text{Score}_i \times \text{Weight}_i) \quad (1)$$

This calculation ensures that features with higher clinical importance, such as temperature, platelets, and WBC, exert a greater influence on the index. Additionally, the scoring system accounts for improvements back to normal ranges, providing a dynamic and clinically meaningful measure of patient stability or decline.

Background Information

Machine learning (ML) and deep learning (DL) models have emerged as powerful tools for the early detection and management of sepsis. These models leverage large datasets, often derived from electronic health records (EHRs), to identify patterns that may not be apparent to clinicians. By integrating patient demographics, vital signs, laboratory results, and other clinical parameters, ML and DL algorithms have shown some success in the ability to predict the onset of sepsis, enabling timely interventions that are critical for improving outcomes (Reyna et al., 2020).

The COMPOSER model is an example of a real-time deep learning system that uses a feed-forward neural network to predict sepsis within four hours of onset (Boussina et al., 2024). It integrates multimodal data from EHRs, including vital signs, lab results, and patient demographics, and employs conformal prediction to minimize false positives, ensuring high-confidence alerts. TREWS, another widely used ML-based model, demonstrated reductions in mortality and organ failure through provider-validated sepsis alerts (Adams et al., 2022). Similarly, Sepsis Watch employs machine learning to provide early warnings but has yet to report direct patient-centered outcome improvements (Sendak et al., 2020). While these models show promise, they often face challenges such as high false positive rates, implementation complexity, and poor generalizability across diverse healthcare settings (Wong et al., 2021).

Ongoing research in the adjacent field of brain-computer interfaces has yielded impressive results in interpreting multiple time-series signals from the brain and conducting classification tasks based on the interpretation of the composite of these signals (Zhao et al., 2024). These efforts have explored the efficacy of convolutional neural networks (CNN) and attention-based Transformer models to conduct this task. CNNs demonstrated success in

effectively extracting features from multiple time series that lead to improved classification results. The Transformer self-attention mechanism has demonstrated a unique ability to evaluate global dependencies over long sequences and to focus on the most significant elements of the signal while ignoring others. Other work explored how these architectures could be combined to realize the strengths of both architectures (Zhao et al., 2024).

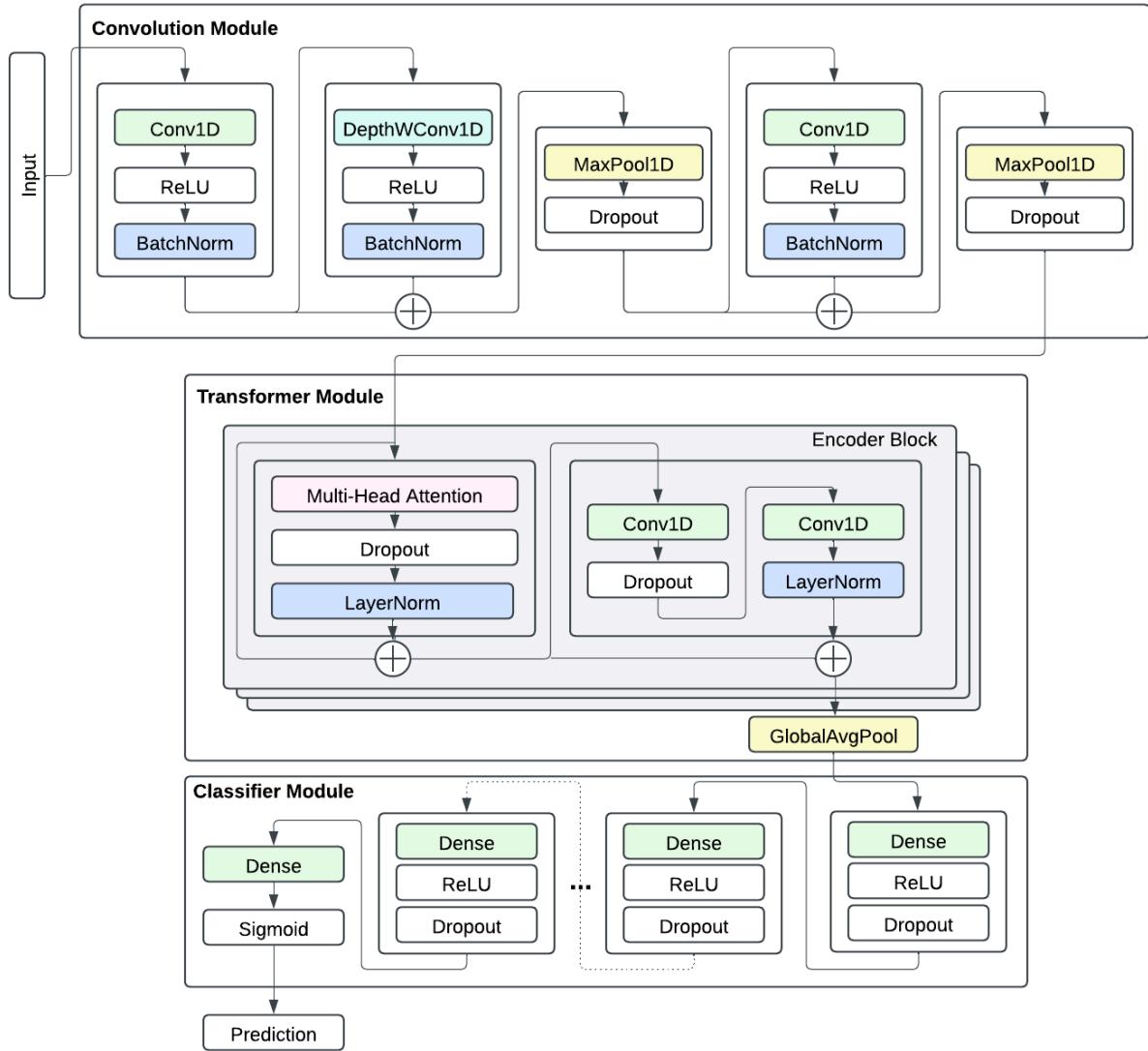
While not directly related to sepsis detection, the task of performing classification on multiple time-series brain signals shares many similar characteristics to the challenge of classifying sepsis from multiple streams of vital sign and lab readings, including the need to deal with multiple streaming features and being robust to missing data readings of some features (Zhao et al., 2024).

Experimental Methods

Inspired by this work, this project focused on the development of a novel model architecture that combined the feature extraction capabilities of CNNs with the long context understanding of the attention-based transformer. Through experimentation with separate CNN and Transformer models, an optimal base architecture for each was developed and combined into a stacked hybrid model, resulting in the Stacked Convolutional Transformer (SCT) Model architecture. The high level framework of the SCT is illustrated in Figure 3 below.

Figure 3

High level architecture of the Stacked Convolutional Transformer model



The SCT is composed of three distinct modules: the Convolutional Module, the Transformer Module and the Classifier Module. Together, the modules perform end-to-end sepsis classification of a patient time-series data stream. Within the Convolutional Module, a total of three one-dimensional convolution operations are performed with two max pooling

layers, with the first stages performing convolution along the spatial axis and the latter stages performing convolution along the time axis. Since overfitting was an observed issue in research, several regularization techniques are employed, including batch normalization, dropout and skip-layer connections. The feature maps output from the Convolutional Module feed into the Transformer Module, where it is passed through multiple encoder blocks, each with a multi-head attention mechanism and feed-forward network (Vaswani et al., 2017). Additional regularization techniques are employed, including layer normalization and skip connections. Finally, the output from the transformer is fed to the Classifier Module, where it runs through multiple fully-connected layers before reaching the output layer, which is a single-neuron fully connected layer with a sigmoid activation function to perform binary classification.

In order to prepare for experimentation, the team first sought to identify appropriate metrics that would provide an effective, comparable measurement of model performance. Recall and precision were identified as important metrics for the task of binary classification. While binary accuracy was also measured, it was not used as a primary performance metric due to it being an unreliable indicator of overall performance when dealing with a highly imbalanced dataset (Ghanem et al., 2023). The business understanding of this task also led the team to slightly optimize for recall over precision. In this case, strong recall performance translates into a lower rate of positive sepsis cases being missed by the model. This is critical in a clinical setting, as a missed positive sepsis case could lead to a negative outcome for a patient. On the other hand, strong precision performance indicates a low occurrence of false positives - that is, patients that do not develop sepsis being flagged as a sepsis risk. The team determined that some level of false positives were acceptable, though it was still important to minimize as much as possible. Overall, model performance was measured using both metrics with slightly more

importance placed on recall. To facilitate measuring the balance between the two, an additional metric, Area Under the Receiver Operating Characteristic Curve (AUC-ROC) was employed.

This was used alongside precision and recall to evaluate overall performance.

The business objective of this project was to accurately predict sepsis risk in patients at least 6 hours in advance of sepsis onset (the prediction horizon). As a time-series classification task, another important aspect to the solution is the lookback window. This is effectively the amount of data (number of time steps) that must be provided to the model in order for it to make reliable predictions. Ideally, a lower amount of data would be necessary. The team experimented with several combinations of lookback window size as well as the prediction horizon. This included a lookback window of 24 hours with 3 and 6 hour prediction horizons, as well as a look back window of 48 hours with 3 and 6 hour prediction horizons. Performance metrics were compared between each configuration to determine the appropriate combination. In the end, the 48/6 combination was selected due to more reliable performance. This was split into train, test and validation datasets with an 80/10/10 split ratio. Class weights were generated from the training dataset to be passed to the model during training. This enabled the model to account for the class imbalance in the dataset that was described previously.

Using this dataset, the team experimented with baseline Linear Regression (LR), Long Short Term Memory (LSTM), CNN and Transformer-based models separately to gauge standalone performance of each architecture. While the LR and LSTM experiments did not yield positive results, the promising performance of the CNN and Transformer models corroborated with the team's research that led to the development of the SCT architecture. However, additional work was necessary to optimize the SCT configuration against the dataset and task.

To support the optimization work, the SCT was implemented as a configurable hyper-model, with a suite of parameters that could be specified to determine the model's characteristics. This included parameters to define configurations such as the number and size of the attention heads in each transformer encoder, the number of encoders to create and the depth and width of the fully connected network. The hyper-model also supported configurable hyperparameters, such as learning rate.

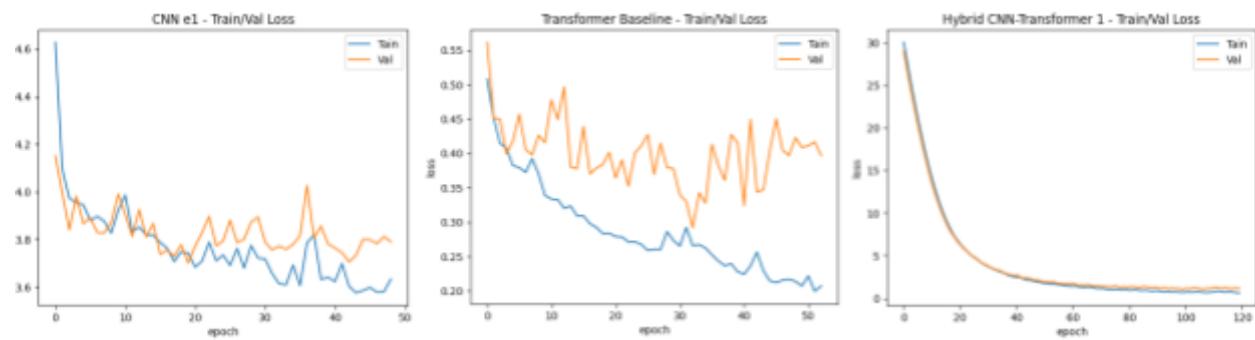
An optimization job was executed using the Keras Random Search Tuner (Keras Team, n.d.-c). For each configurable parameter, an acceptable range of values was provided to the tuner which then performed 25 trials, each consisting of randomly generated values for each parameter, within this defined range. Within each trial, a model was compiled with the generated parameters and trained on the dataset. The tuner monitored the progress of the training job, looking for an optimal value of validation loss across all trials. The model configuration that achieved the lowest validation loss was selected as the best model and used for final performance validation against the hold out test dataset.

Results

Through experimentation with varying depths and configurations, the best of the standalone CNN and Transformer-based models were able to achieve good performance in both precision and recall metrics. However, both model types tended to overfit on the training dataset and did not converge smoothly, as illustrated in the example training/validation loss curves shown in the left two graphs in Figure 1 below. The condition did not improve much with the addition of regularization techniques, including dropout, L1 regularization and batch normalization. The move to the SCT architecture effectively eliminated the overfitting problem, as illustrated in the far right graph in Figure 1.

Figure 4

Training and validation loss curves for optimized CNN, Transformer and SCT models



The development of the Deterioration Index demonstrated success in improving model performance. The results of this were most evident in the 16% gain in precision performance observed in baseline models trained on a dataset that incorporated the DI compared to the same model trained on data without the DI. This helped reduce the rate of false positives, an important component of the project's performance objective.

Overall, the SCT model posted consistently strong performance metrics, even before the tuning and optimization routines outlined above were performed. Critically, the model struck a good balance between slightly optimizing for recall, while maintaining good precision performance. Once optimized through a tuning process, the SCT model posted the strongest performance metrics compared to all previous models and demonstrated the same strong generalizing characteristics observed during experimentation. This is most evident in the consistently high AUC-ROC scores of 0.93 on the validation dataset and 0.92 on test dataset. The final performance results of the tuned SCT model and top CNN and Transformer models are outlined in Table 1 below.

Table 2

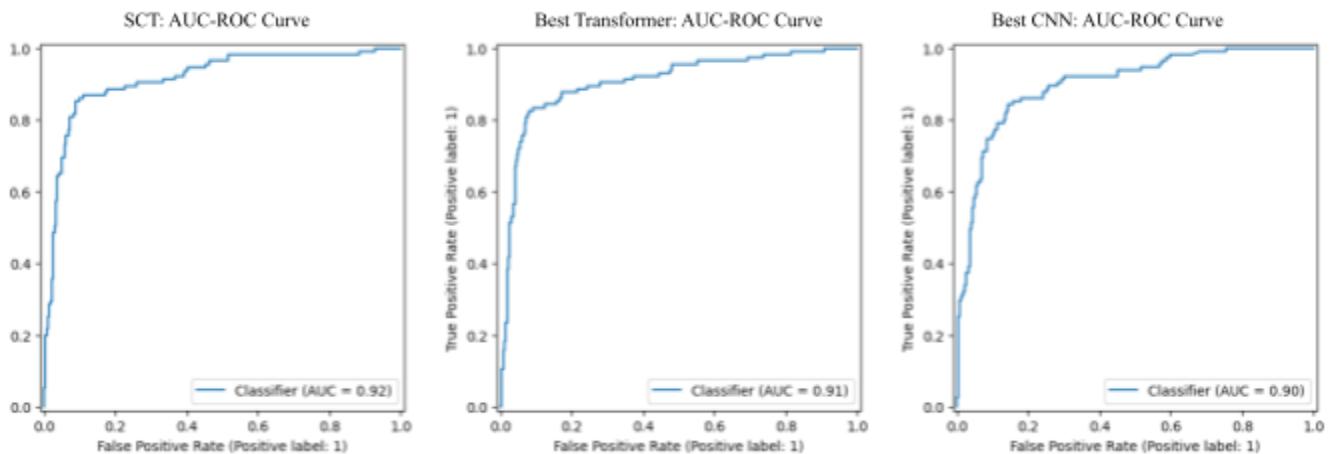
Performance metrics of the top three model architectures vs. holdout test data

Model	AUC-ROC	Recall	Precision	Binary Acc.
Tuned SCT Model	0.92	0.84	0.80	0.89
Best Transformer	0.91	0.83	0.79	0.88
Best CNN	0.90	0.76	0.76	0.86

The performance metrics demonstrate that the SCT model was able to predict positive sepsis outcomes in patients with a high degree of reliability 6 hours in advance of sepsis onset. It also demonstrated that the model generalized to unseen data well. Further, the SCT demonstrated the ability to optimize for minimizing false negatives (ie: not missing positive sepsis outcomes) while maintaining a reasonable balance with a low number of false positives, as shown in Figure 3 below. These characteristics fulfilled the objectives of this project and the performance of the SCT model exceeded initial expectations.

Figure 5

AUC-ROC Curves for the three top performing models



Evaluation of sample predictions of the model also revealed interesting insights as to how the DI may support the model in building short and long term context of overall patient condition. Examples are illustrated in Figure 4 below. On the left, the DI of two sample patients are plotted together with the SCT's predicted sepsis outcome. The SCT predicted a positive sepsis outcome for the patient with the red line in this graph and a negative outcome for the patient with the green line. Both predictions were correct. This visualization illustrates how the DI may help the SCT build long term context of the patient's condition over time. The red line shows a larger index value trending upwards, while the green line shows a lower overall index value trending downwards. On the right is an example patient where the SCT correctly predicted a negative sepsis outcome, even though the overall trend seemed to indicate deterioration. The graph indicates that the patient seems to make a drastic recovery during the last hour of the series. This may provide some evidence to the DI capturing rapid changes and helping the model weigh both short and long term context. In this case, it helped the model potentially avoid making a false positive prediction. This supports the overall improvement in false positive performance observed during baseline testing.

Figure 6

Plots of Deterioration Index (DI) vs. Model Predictions. Left: A patient where SCT correctly predicted positive sepsis (red) and one where SCT correctly predicted negative sepsis (green).

Right: A patient where SCT correctly predicted a negative outcome even though the trend pointed to deterioration.



Conclusion

This project demonstrated promising results while underscoring the many challenges that are faced when attempting to accurately predict sepsis onset in a clinical setting. These challenges included the high occurrence of missing data, sparsity in certain feature readings with respect to time (lab samples), significant class imbalance and the need to build up a relatively long context window of readings for a patient before the predictions became reliable.

The development of the Deterioration Index feature proved helpful in overcoming some of the challenges. Since this feature looked at changes in critical attributes at the current time step as compared to the previous time step, a lag was naturally incorporated, which appeared helpful in smoothing over missing data as well as the sparse lab readings.

When combined with the novel SCT architecture, the system was able to strike a good balance between optimizing for a low rate of false negatives and maintaining a relatively low occurrence of false positives. The system also generalized well to unseen data. By managing the overfitting problem and minimizing the rate of false positives, the system was able to overcome two of the most common challenges faced by previous works in this area (Wong et al., 2021).

Overall, these results were encouraging and set a solid foundation for additional work to further improve sepsis predictability in clinical settings. One particular area that warrants deeper exploration is to shorten the context window required by the model to make reliable predictions. This project experimented with both 24 and 48 hour windows, eventually landing on 48 hours due to the difficulty in obtaining reasonable performance with only 24 hours of context. This could likely be significantly improved through optimizations to the data preparation pipeline. Specifically, applying additional filtering of patients with 24 hours (or less) of context based on data quality and completeness would be a recommended area of focus, as this cleaner, more consistent data may enable the model to perform well even with the shorter context. Additionally, the team conducted some preliminary investigation into a second novel feature that would capture cross-correlation between vital signs and lab readings. This work was not completed and would be another area for further research.

References

Fleischmann, C., Scherag, A., Adhikari, N. K., et al. (2016). *Assessment of Global Incidence and Mortality of Hospital-treated Sepsis*. American Journal of Respiratory and Critical Care Medicine, 193(3), 259-272.

Johnson, A. E., Pollard, T. J., Mark, R. G., et al. (2020). *MIMIC-IV (Version 1.0)*. PhysioNet. <https://doi.org/10.13026/a3wn-hq05>

Reyna, M. A., Josef, C., Jeter, R., Shashikumar, S. P., Westover, M. B., Nemati, S., Clifford, G. D. (2019). *Early Prediction of Sepsis from Clinical Data: The PhysioNet/Computing in Cardiology Challenge 2019*. Critical Care Medicine, 47(11), 1794–1802.

Singer, M., Deutschman, C. S., Seymour, C. W., et al. (2016). *The Third International Consensus Definitions for Sepsis and Septic Shock (Sepsis-3)*. JAMA, 315(8), 801-810.

Boussina, A., Shashikumar, S. P., Malhotra, A., Owens, R. L., El-Kareh, R., Longhurst, C. A., Quintero, K., Donahue, A., Chan, T. C., Nemati, S., & Wardi, G. (2024). Impact of a deep learning sepsis prediction model on quality of care and survival. *Npj Digital Medicine*, 7(1). <https://doi.org/10.1038/s41746-023-00986-6>

Sendak, M. P., Ratliff, W., Sarro, D., Alderton, E., Futoma, J., Gao, M., ... Balu, S. (2020). Real-world integration of a sepsis deep learning technology into routine clinical care: Implementation study. JMIR Medical Informatics, 8(7), e15182. <https://doi.org/10.2196/15182>

Wong, A., Otles, E., Donnelly, J. P., Krumm, A., McCullough, J., DeTroyer-Cooley, O., ...

Koppel, R. (2021). External validation of a widely implemented proprietary sepsis prediction model in hospitalized patients. *JAMA Internal Medicine*, 181(8), 1065–1070.

<https://doi.org/10.1001/jamainternmed.2021.2626>

Jones, A. E., Shapiro, N. I., & Trzeciak, S. (2007). Lactate clearance vs central venous oxygen saturation as goals of early sepsis therapy: A randomized clinical trial. *JAMA*, 297(7), 739-746. <https://doi.org/10.1001/jama.297.7.739>

Seymour, C. W., Liu, V. X., Iwashyna, T. J., Brunkhorst, F. M., Rea, T. D., Scherag, A., ... & Angus, D. C. (2016). Assessment of clinical criteria for sepsis: For the third international consensus definitions for sepsis and septic shock (Sepsis-3). *JAMA*, 315(8), 762-774. <https://doi.org/10.1001/jama.2016.0288>

Pinsky, M. R. (2007). Functional hemodynamic monitoring. *Intensive Care Medicine*, 28(3), 386-396. <https://doi.org/10.1007/s00134-007-0532-6>

Johnson, A. E. W., Pollard, T. J., Shen, L., Lehman, L. W. H., Feng, M., Ghassemi, M., ... & Mark, R. G. (2020). MIMIC-III, a freely accessible critical care database. *Scientific Data*, 3(1), 160035. <https://doi.org/10.1038/sdata.2016.35>

Zhao, W., Jiang, X., Zhang, B., Xiao, S., & Weng, S. (2024). CTNet: a convolutional transformer network for EEG-based motor imagery classification. *Scientific Reports*, 14(1).

<https://doi.org/10.1038/s41598-024-71118-7>

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017, June 12). *Attention is all you need*. arXiv.org.

<https://arxiv.org/abs/1706.03762>

Ghanem, M., Ghaith, A. K., El-Hajj, V. G., Bhandarkar, A., De Giorgio, A., Elmi-Terander, A., & Bydon, M. (2023). Limitations in Evaluating machine learning models for imbalanced binary outcome classification in Spine Surgery: A Systematic review. *Brain Sciences*, 13(12), 1723.

<https://doi.org/10.3390/brainsci13121723>

Keras Team. (n.d.-c). *Keras documentation: KerasTuner API documentation*.

https://keras.io/keras_tuner/api/