COMPARING PERFORMANCE OF RECURRENT NEURAL NETWORK ARCHITECTURES USING DATA ON WILDFIRE PROPAGATION

By

Alec Thomas-Michael Creasy

A thesis submitted in partial fulfillment

of the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

Middle Tennessee State University

May 2025

Thesis Committee:

Dr. Arpan Sainju

Dr. Joshua Phillips

Dr. Suk Seo

ACKNOWLEDGEMENTS

I would like to thank Dr. Phillips for his guidance during the Research Methods Course, as well as Dr. Sainju for his valuable mentorship during my endeavors in learning to conduct research in the field of Computer Science. I also would like to thank the two reviewers for their thoughtful feedback and meticulous attention to detail. Their insights helped to introduce logical and useful additions and modifications to this manuscript, and they have my gratitude for their time and care for this manuscript.

ABSTRACT

Machine learning techniques have been used in recent studies to predict wildfire propagation. In these endeavors, Artificial Neural Networks (ANNs) have been a popular choice to build models to solve this task. Within these models, two subsets of ANNs, Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) are a popular choice for predicting the likelihood of a fire spreading across a region over a specified number of time steps. Most existing research makes use of the Long Short-Term Memory (LSTM) variant of the RNN architecture, a special type of RNN that can retain long-term context amongst many time steps in training. The Gated Recurrent Unit (GRU) RNN is a simplified version of the LSTM architecture that hasn't been explored as much. This study explores the performance differences between the LSTM and GRU architectures on wildfire propagation prediction tasks using quantifiable metrics.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I.

INTRODUCTION

Artificial Neural Networks

In the world today, a certain subset of Artificial Intelligence has been changing how the world operates. The underlying technology behind these tools is a model based on the human brain called an Artificial Neural Network, henceforth known as an ANN. The intuition of these models is that they can detect and learn patterns in that dataset and can then be deployed to be able to generalize based on unseen data. ANNs come in several different types of architectures and can be applied to many use cases, a few of such being Convolutional Neural Networks (CNN) for spatial data and Recurrent Neural Networks (RNN) for temporal and time series data.

CNNs and RNNs for Wildfire Propagation Prediction

CNNs are widely used to detect spatial dependencies in data. This approach can be used in cases such as image classification and image segmentation. Image classification is a task where the model will predict the most likely class label for an image, and image segmentation is where the model will attempt to classify different parts of an image.

RNNs are used for temporal and time series data. Essentially, any data that involves an element of time such as minutes, hours, or days. One such use case of an RNN is for stock price prediction. Given the price of a stock over a 10 day period, what is its likely price on day 11? RNNs come in different types of variations, such as the original Elman RNN, the Long Short-Term Memory (LSTM), and the Gated Recurrent Unit RNN (GRU). The LSTM and GRU were introduced to solve the issue with vanishing gradients with long-term dependencies in traditional RNNs. With a long sequence of data, long-term dependencies can begin to be forgotten or not utilized in the prediction of the next step in the sequence. The LSTM and GRU were introduced to help mitigate this issue.

One such task is combining CNNs and RNNs for wildfire propagation prediction using

satellite data. This task utilizes the CNN element to capture spatial information from a satellite which can then detect fires, foliage, etc. The RNN component can be used for satellite data over time, such as days. When combined with the RNN, a sequence of days can be fed into the model and can be trained to predict where the fire will most likely be burning the next day in the sequence.

One such architecture used is the ConvLSTM model, combining the CNN with the LSTM architecture. While quite a bit of work has been done utilizing the ConvLSTM model, there is virtually no literature on the performance of a CNN combined with a GRU architecture for wildfire propagation. This shows a need for the impact on performance when using a CNN combined with a GRU when trained on wildfire propagation data.

RNN Architectures Used for Wildfire Propagation Prediction

In existing work, most all models in their capacities utilize an approach known as ConvLSTM. It is typically used to detect spatial data whether that be data from a satellite or simulated spatial data over time. With all of this research, however, little to no work has been done utilizing a CNN combined with a GRU. Henceforth, this approach will be referred to as a ConvGRU. This study aims to determine the differences in performance when using a GRU as opposed to an LSTM in the model. The running hypothesis during this experiment is that both models would perform very similarly, with the ConvLSTM having a slight edge due to the more complex nature of its cells.

CHAPTER II.

BACKGROUND

Convolutional Neural Networks

CNNs are able to improve upon the base ANN structure, especially in image and spatial data, by introducing the convolutional layer as shown by O'Shea and Nash in 2015 [6]. The convolutional layer utilizes a grid known as a kernel which then slides across the input extracting features to be used in classification. A pooling layer can be introduced if desired to reduce the size of the input to conserve only the most important features extracted. Some data is lost during this process, but will greatly reduce the size of the input, saving computational power on very large data.

Recurrent Neural Networks

RNNs use the power of recursion to retain context within a network, remembering past information that can be used for predictions of the next step in a sequence. RNNs modify ANNs by connecting the hidden layer recursively across multiple time steps as shown by Salenhinejad et al. in 2018 [3]. The recursive nature allows the network to remember key information throughout a sequence of inputs as opposed to getting one input from one snapshot in time. The standard and original architecture suffers from the vanishing gradient problem, where long-term dependencies begin to be forgotten or not used the further the sequence goes as showcased by Bengio et al. in 1994 [8]. A few different approaches have been used to solve this problem: the LSTM RNN and the GRU RNN.

GRU and LSTM networks

Both the LSTM and GRU utilize additional gates to eliminate the vanishing gradient problem and retain long-term information. The added mechanisms help solve the vanishing gradient problem by introducing new ways to ensure long-term information is continuously fed through the model across all time steps.

The LSTM RNN adds a context vector, and the cell is made up of 3 gates: the input gate,

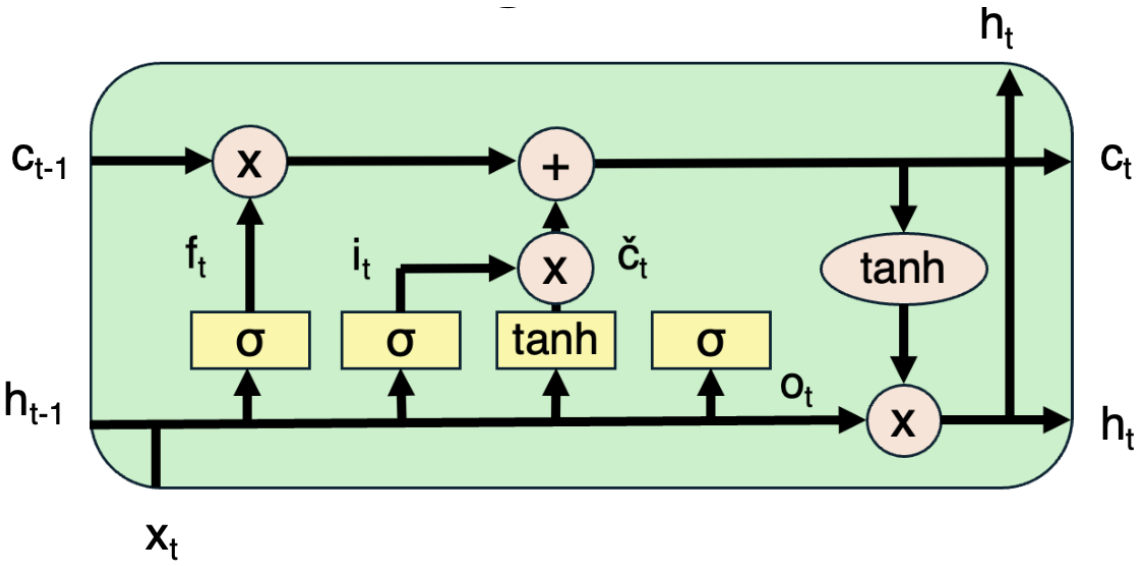

Figure 1: This figure shows the basic structure of an LSTM cell. x_t represents the current input, c_{t-1} represents the context vector at time $t-1$, h_{t-1} represents previous hidden state, f_t represents the forget gate, i_t and $\check{c}_t$ represent the input gate, h_t represents the current new hidden state, and o_t represents the output.

the forget gate, and the output gate. The input gate determines what new information should be added from the current input and previous hidden states. The forget gate determines what information should be kept or discarded from the context. The output gate then determines what should be passed to the context and the hidden state from the current cell [Salehinejad et al., 2018] [3] (See Figure 1).

The GRU RNN is basically a simplified version of the LSTM cell. The GRU, by contrast, has no context vector and utilizes only 2 gates: the reset gate and the update gate. The reset gate determines how much of the hidden state to retain at the current time step. The update gate determines how much of the current input should be added to the hidden state at the current time step. The simplified nature of the GRU introduces less computational overhead, but can suffer in performance in some cases [Salehinejad et al., 2018] [3] (See Figure 2).

Both architectures have been compared to discover what types of advantages each have over the other. Irie et al. in 2016 performed an experiment to see which is better at speech

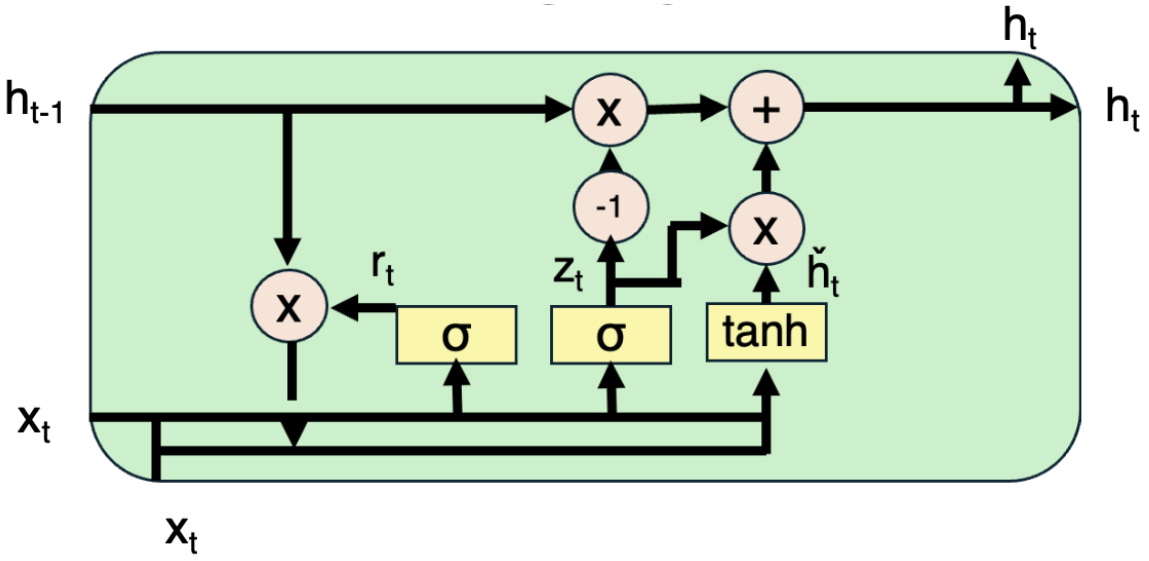

Figure 2: This figure shows the basic structure of a GRU cell. x_t represents the current input, h_{t-1} represents previous hidden state, r_t represents the reset gate, z_t represents the update gate, and h_t represents the current new hidden state.

modeling, and found that the LSTM architecture slightly beats out the GRU architecture [5]. Studies like these do show that the LSTM architecture is typically better for these temporal tasks than GRU architectures. Irie et al. does however that in some aspects, the GRU does have small noteworthy performance bumps over the LSTM architecture [5].

GRU and LSTM Networks in Wildfire Propagation Prediction

In various studies such as Burge et al. 2020 [4], CNNs and LSTMs have been widely used to predict and model wildfire spread patterns. More recently, techniques have been used with ConvLSTM and self-attention mechanisms on satellite data and simulated data as studied in Masrur et al., 2024 and showed just how accurate baseline ConvLSTM models can be with an F1-Score of 91.8% in the satellite data case [1]. There has also been work done using multi-modal data as well, employing CNN and LSTM on data in Russian regions as Shadrin et al. did in 2024 [2]. GRUs have, in some cases, been used for wildfire propagation when using a comparison of performance using only RNN architectures with no CNN

involvement, and can even outperform the LSTM in some edge cases as Perumal and Zyl showcased in 2020 [7].

CHAPTER III.

METHODS

The methodology of this work follows this flow: data collection, data pre-processing, creating and training the ConvLSTM and ConvGRU models, validating the results, and analyzing the results.

Data Collection

The data was collected from NASA's FIRMS database. FIRMS stands for Fire Information for Resource Management System, and provides near real-time satellite updates of fire spread. The data we are using is from the Butte County Camp Fire from November 8, 2018-November 25, 2018. The data provides latitude and longitude boundaries and areas of wildfire spread with confidence intervals. For our model, a confidence of 50% or higher indicates a positive signal for a fire (See Figure 3).

Data Pre-processing

The data is split into grids where each unit is $1km^2$. The model is then fed with a central pixel with a 3x3 grid, the center representing the area that will be predicted with the borders of the grid added for spatial complexity. The data is fed in with batch sizes of 32, and includes the first 5 time steps, where the 6th time step will be the next prediction.

The data will be loaded in with a PyTorch DataModule, which help to organizes and does a lot of boilerplate pre-processing for the data being fed into the model. The DataModule is responsible for setting the data up to be fed into a PyTorch network, which will be explained in the next section. The data is split with an 80/20 split, with 80% of the data being data that will be used in training and 20% of the data being data that will be used for model validation.

Model Structure

The models are built using PyTorch, an open-source machine learning framework based in Python, and PyTorch Lightning which is built on top of PyTorch for a streamlined

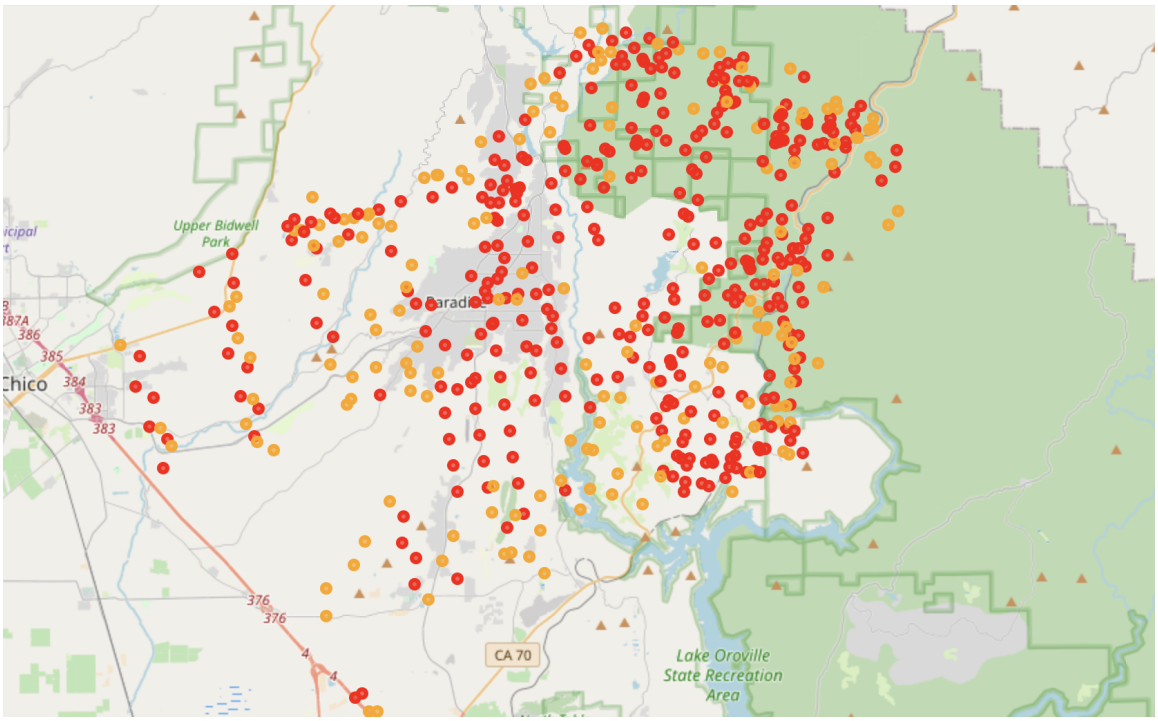

Figure 3: Map view of the data acquired from the NASA FIRMS dataset in Butte, County, CA from the 2018 campfires. Red dots represent a fire detection with a confidence rating of 80% or higher, with orange dots representing a confidence score of less than 80%.

development of machine learning models. Each model follows an incredibly similar structure. The ConvLSTM model is the baseline, and the ConvGRU is essentially only switching out the LSTM cell for a GRU cell. Each network utilizes 2 2D convolutional layers combined with LSTM or GRU, depending on the model, with a kernel size of 3 and a stride of 1 with padding. The first layer outputs 64 channels and the second 128 channels. Each cell in the ConvLSTM and ConvGRU models uses an LSTM and GRU cell for each time step of input, respectively. The final layer is a 3D convolutional layer with a kernel size of 3x3x3, a stride of 1, and outputs 256 channels. This 3D layer is needed to account for the third dimension of time. Each layer uses the Hyperbolic Tangent activation function for the cell input in the LSTM cell and the output in the GRU cell, with all other gates using Sigmoid. Sigmoid is also be used for our final prediction, as it is needed for binary classification (Figures 4 and

5).

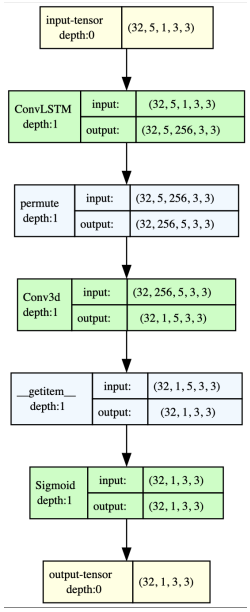

Figure 4: Graphical representation of the ConvLSTM model. Input tensors are passed to the 2 ConvLSTM layers, the data is permuted so that the 3D Convolutional layer can receive the data with the current dimensions, and then the output is passed to the Sigmoid activation function, predicting the next step in the sequence.

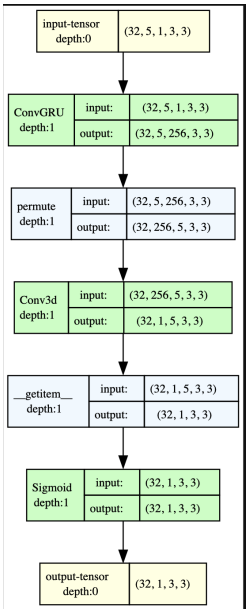

Figure 5: Graphical representation of the ConvGRU model. Input tensors are passed to the 2 ConvGRU layers, the data is permuted so that the 3D Convolutional layer can receive the data with the current dimensions, and then the output is passed to the Sigmoid activation function, predicting the next step in the sequence.

Training the Models

The models both use the Adam optimizer with a learning rate of 0.001, ensuring parameters are not adjusted too much each training epoch. The model records loss for training and validation, as well as F1 scores for training and validation. The models will be trained for 50 epochs. An epoch is simply one pass through the training data, and then a backpropagation algorithm is applied to update the weights. The Binary Cross Entropy Loss function is used to evaluate how much to update the weights between the layers.

$$tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$$

$$sigmoid(x) = 1/(1 + e^{-x})$$

x = current input

$$L_{BCE} = -1/n\sum_{i=1}^{n}(y_i \cdot \log \hat{y}_i + (1-y_i) \cdot log(1-\hat{y}_i))$$

n = number of samples

y_i = correct answer for current sample

$\hat{y}_i$ = prediction for current sample

Validation of Results and Analysis

After each training epoch and after training is complete, the validation portion of the data will have its F1 and loss calculated by running the data through the model. The validation data will not be used for backpropagation updates, and is strictly to see how the model performs at generalization throughout training and at the end of training. The F1 score is a combination of the Precision and Recall methods, and can be used in place of accuracy although it is not a replacement. The F1 score will give us a good indicator as to how well our model is generalizing to unseen data which, in our case, is the validation set. Using the loss and F1-score curves, analysis of the model can be performed.

$$Precision = TP/TP + FP$$

$$Recall = TP/TP + FN$$

$$F1 = (2 \cdot Precision \cdot Recall)/(Precision + Recall)$$

TP = True Positive

FP = False Positive

FN = False Negative

All code pertaining to the project is open-sourced. All documentation needed to setup and run the experiment can be found in the GitHub repo at `https://github.com/Alectmc/WildfireRNNComparison`. The repo contains all of the instructions needed to run the experiment.

CHAPTER IV.

RESULTS

Figures 6 and 7 shows the results of both networks during and after training. The x-axis is the epoch number, with the 50th epoch being where the model stood at the end of training. The y-axis shows the Binary Cross Entropy Loss for the loss curves and the F1 Score for the F1 curves.

The results for the trial were quite interesting. Loss typically goes down as the model continues to be fit during training and this typically will result in an increase in accuracy as the model goes through training. In these results, the validation loss starts rather shaky, but begins to go up around the 15th epoch for both models. When a loss curve begins to go up, this is evidence of a phenomenon known as overfitting. Overfitting occurs when the model starts to learn to memorize the training data but struggles to generalize on new, unseen data. In Masrur et al.'s baseline ConvLSTM model, they do not report results that can evidence overfitting, but do use 1,929 samples with multiple features such as wind components and soil moisture [1]. This study does not use these other features and simply uses the satellite data with less samples, only using 1,268 samples. Given the lower quantity of data and the importance of other features such as weather forecasts, foliage, and terrain data, this is likely a reason the model overfit as opposed to the Masrur et al.'s results.

Interestingly, the F1 for both models jumps dramatically to about 50% and both models linger around this area through the end of training. The ConvGRU actually outperforms

Table 1: Final F1-Scores of validation data after training

Model	**CCE Loss**	**F1-Score**
ConvLSTM	1.151137	50.7519%
ConvGRU	1.146242	50.6319%

the ConvLSTM for most of the training. At the end however, the ConvLSTM does surpass the ConvGRU but both meet around the same score. With how close both scores are, it seems they could keep going back and forth, but it is worth noting ConvGRU does slightly outperform the ConvLSTM for much of training.

Looking at Table 1, however, we can observe that both models finish with incredibly close F1-scores and loss values, with the ConvLSTM model having the slight edge in F1-score with a score of 50.7519% to the ConvGRU's 50.6319%. While the different here is marginal, this does support the idea that the ConvLSTM would have a slightly higher F1-score, but this difference is marginal at best, and isn't enough to make a fully confident conclusion. The reason being if we look at training, the ConvGRU actually beat out the ConvLSTM for a lot of training, so it could just so happen that at the final epoch, the ConvLSTM just barely beat out the ConvGRU. We do notice, however, that the loss is slightly lower in the ConvGRU model. This could mean that the ConvGRU isn't experiencing an overfitting problem quite as bad as the ConvLSTM model, but looking at the curves in Figure 6, it seems this difference is also quite marginal. It is worth noting that the loss for the ConvGRU is lower throughout the 35-50th epochs, however.

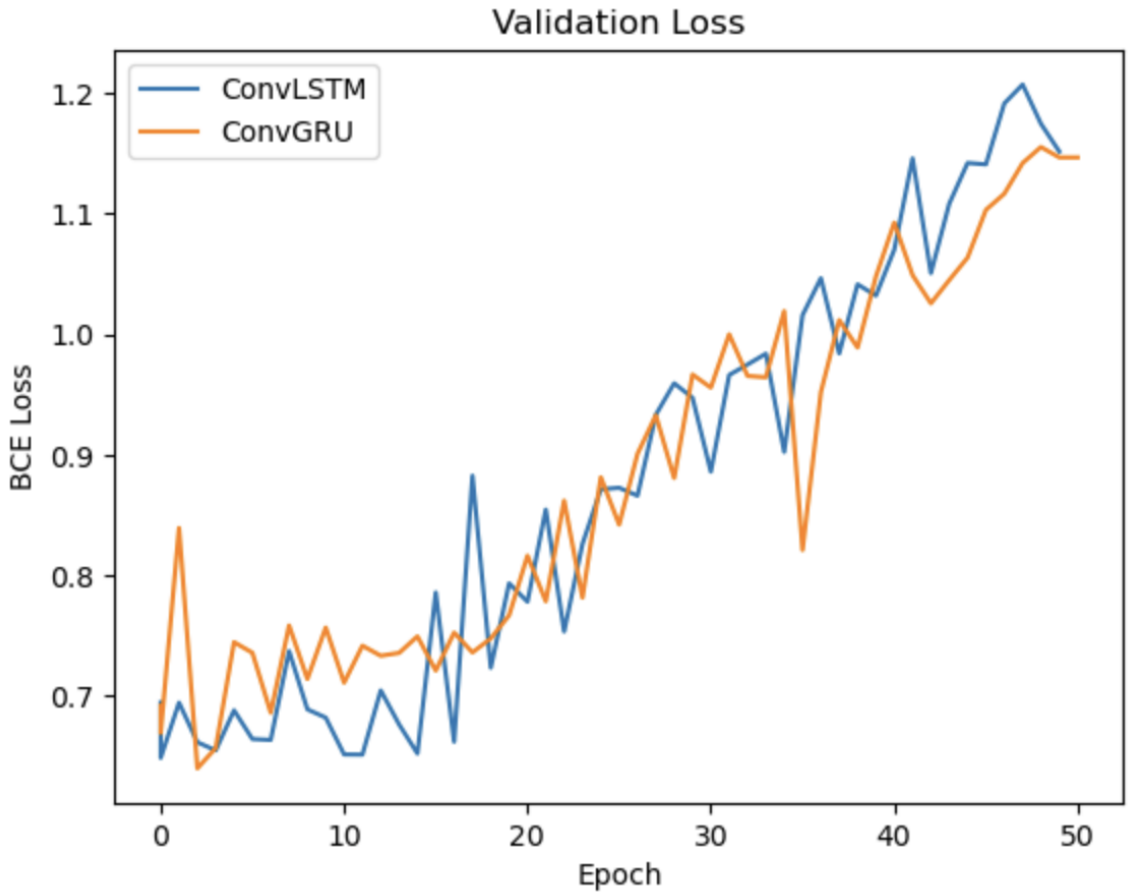

Figure 6: These graphs show the loss curves for the validation data. The blue line represents the ConvLSTM network and the orange line represents the ConvGRU network.

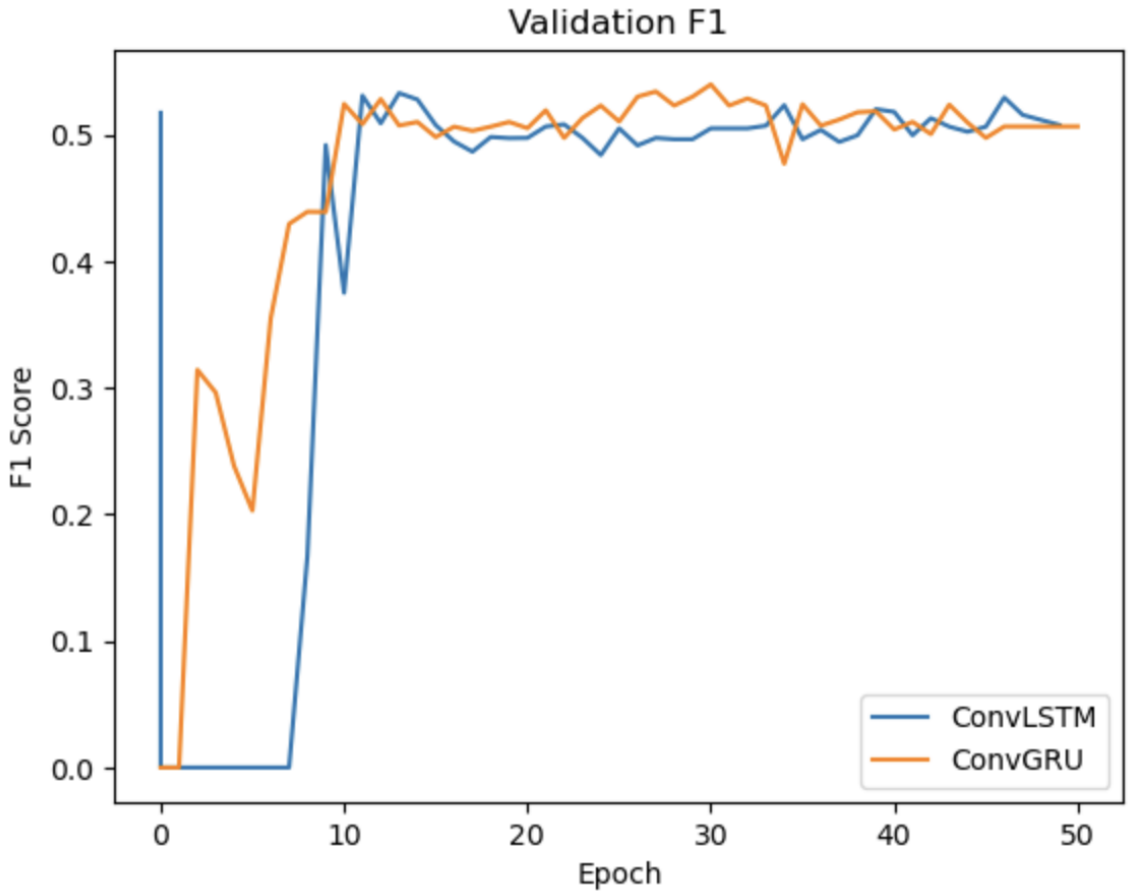

Figure 7: These graphs show the F1 scores for each epoch of training with the validation data. The blue line represents the ConvLSTM network and the orange line represents the ConvGRU network.

CHAPTER V.

DISCUSSION

The results show a clear issue of overfitting within the models, and this is likely due to having a lack of sufficient data. Overfitting can occur because of this, as well as a lack of variety in the data. Adding more data, as well as augmenting the data that is currently being used, could possibly help mitigate or even resolve this phenomenon. Although both models overfit, both overfit in very similar patterns, backing up the original hypothesis that both models will perform very similarly.

The F1 scores show results similar to what was expected. Both models perform very closely to each other. Curiously, the ConvGRU model actually outperforms the ConvLSTM for much of training, but both meet at around the same F1 score at the end of training. While both models do trade places for the lead, the ConvGRU model appears to take a marginal lead around the 20th epoch until the 30th epoch. After this, both models trade the lead in F1 score very often. Both models performing so similarly is expected, but the ConvGRU outperforming the ConvLSTM for a good amount of time was not expected, as the added complexity of the LSTM cell was expected to help give it an edge over the ConvGRU model.

Given more data along with data augmentation, the overfitting issue may be able to be mitigated a bit, and a clearer picture of the models' performances would present itself. However, in this experiment it was quite clear that both models, even with their overfitting issues, did perform very similarly to one another. Even with the loss indicating overfitting, both models actually did show very similar patterns in their loss and F1 curves.

In conclusion, it is very clear that both architectures perform very similarly, with the ConvGRU architecture seeming to perform a little better than expected at some points on wildfire propagation data. With the better performance, however, the performance bump is very small, and does not seem statistically significant. In the future, work could be done with more robust datasets as well as more robust models with more layers and more parameters.

Data augmentation would also allow the models to see more representations of the data, helping to reduce overfitting and give a better picture of how each RNN architecture stacks up against one another with wildfire propagation data. From here, other mechanisms could be employed such as self-attention to see how the models would generalize with even more tools to understand and generalize the data, giving a clearer picture to the advantages of using ConvGRU and ConvLSTM based architectures.

BIBLIOGRAPHY

[1] Arif Masrur, Manzhu Yu, A. T. Capturing and interpreting wildfire spread dynamics: attention-based spatiotemporal models using convlstm networks. *Ecological Informatics*, 82:102760, 2024.

[2] Dmitrii Shadrin, Svetlana Illarionova, F. G. K. E. M. M. I. L. R. B. E. B. Wildfire spreading prediction using multimodal data and deep neural network approach. *Scientific Reports*, 14(1):2606, 2024.

[3] Hojjat Salehinejad, Sharan Sankar, J. B. E. C. S. V. Recent advances in recurrent neural networks. *arXiv*, 2018.

[4] John Burge, Matthew Bonanni, M. I. L. H. Convolutional lstm neural networks for modeling wildland fire dynamics. *arXiv*, 2020.

[5] Kazuki Irie, Zolt´ an T¨ uske, T. A. R. S. u. H. N. Lstm, gru, highway and a bit of attention: An empirical overview for language modeling in speech recognition. *2016 Interspeech*, pages 3519–3523, 2016.

[6] O'Shea, K. and Nash, R. An introduction to convolutional neural networks. *arXiv*, 2015.

[7] Perumal, R. and Zyl, T. L. V. Comparison of recurrent neural network architectures for wildfire spread modelling. *2020 International SAUPEC/RobMech/PRASA Conference*, pages 1–6, 2020.

[8] Y. Bengio, P. Simard, P. F. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.