The Search for Life: Exoplanet Detection with Deep Learning

Natasha Scannell
University of Wisconsin-Milwaukee

THE SEARCH FOR LIFE: EXOPLANET DETECTION WITH DEEP LEARNING

by

Natasha Scannell

A Thesis Submitted in

Partial Fulfillment of the

Requirements for the Degree of

Master of Science

in Computer Science

at

The University of Wisconsin-Milwaukee

May 2021

ABSTRACT

THE SEARCH FOR LIFE: EXOPLANET DETECTION WITH DEEP LEARNING

by

Natasha Scannell

The University of Wisconsin-Milwaukee, 2021
Under the Supervision of Professor Susan McRoy

The discovery of new exoplanets, planets outside of our solar system, is essential for increasing our understanding of the universe. Exoplanets capable of harboring life are particularly of interest. Over 600 GB of data was collected by the Kepler Space Telescope, and about 30 GB is being collected each day by the Transiting Exoplanet Survey Satellite since its launch in 2018. Traditional methods of experts examining this data manually are no longer tractable; automation is necessary to accomplish the task of vetting all of this data to identify planet candidates from astrophysical false positives.

Previous state-of-the-art models, Astronet and Exonet, use deep convolutional neural networks (CNNs) with over 8.8 million parameters. In this paper, I experiment with the application of recurrent networks, attentional models, and scaling down Astronet. I have developed a CNN model with 8x fewer trainable parameters than Astronet with the same accuracy and improved precision. I also provide a CNN-LSTM model with 59x fewer parameters just 1% behind Astronet in accuracy that, with further tuning, may also be a competitive model for particularly resource-constrained uses.

All code for this research is available on GitHub.[1]

[1]https://github.com/nmscannell/exoplanet-detection

To

Lucas

for helpful advice and encouragement

and

Mom

for undying support

TABLE OF CONTENTS

List of Figures **vii**

List of Tables **x**

1 Introduction **1**

1.1 Exoplanets 1

1.1.1 The Search For Life 2

1.1.2 Exoplanet Surveys 3

1.1.3 Detection Methods 3

1.2 Neural Networks 5

1.2.1 Convolutional Neural Networks 6

1.2.2 Long Short Term Memory Networks 8

1.2.3 Attention and Transformers 11

1.3 Astronomical Data Analysis 12

2 Related Work **13**

2.1 Deep Learning on Kepler Data 13

2.1.1 Astronet 13

2.1.2 Exonet 16

2.1.3 Astronet-K2 18

2.1.4 Application of LSTM 18

2.2 Deep Learning on Other Exoplanet Surveys 19

2.2.1 Simulated Transiting Exoplanet Survey Satellite Data 19

2.3 Neural Network Scaling 20

2.3.1 EfficientNet 20

3 Methods **21**

3.1 Dataset 21

3.1.1 Data Augmentation 21

3.2 Models 22

3.2.1 Activation Functions 22

3.2.2 Training 24

3.2.3 Regularization and Dropouts 26

3.2.4 Long Short Term Memory Network 26

3.2.5 Convolutional Long Short Term Memory Network 27

3.2.6 Scaled Down Convolutional Neural Network 28

3.2.7 Attention 28

3.3 Hyperparameter Optimization 32

4 Results **35**

4.1 Metrics 35

4.2 LSTM 36

4.3 CNN-LSTM 39

4.4 CNN 42

4.5 Models with Attention 46

4.5.1 Transformer 46

4.5.2 Other Models with Attention 49

4.6 Comparison of Best Performers to Astronet 50

5 Conclusion and Future Work **53**

LIST OF FIGURES

1-1 Example densely-connected feed-forward network with N inputs, L hidden layers with varying number of units/neurons, and a single output. 6

1-2 Convolutional Layer From Stanford CS230 Course 7

1-3 Max Pooling From Stanford CS231n Course 7

1-4 Rolled Simple RNN unit[2] 8

1-5 Unrolled Simple RNN Unit [2] 9

1-6 Unrolled LSTM Unit [2] 9

2-1 Global and local views of three observations in the dataset.[40] 14

2-2 Astronet architecture, where each layer is labeled as conv<kernel size>-<num feature maps>, maxpool<size>-<stride>, and FC-<num units>[40] 16

2-3 Examples of light curves with centroid curves for a transiting exoplanet and an eclipsing binary star[6] 17

3-1 Example LSTM architecture. Brackets show tested number of neurons for each LSTM/fully connected layer or range of values tested for regularization. 27

3-2 Example CNN-LSTM architecture. Brackets show tested number of neurons for each Conv/LSTM/fully connected layer or range of values tested for regularization. 29

3-3 Example CNN architecture. Brackets show tested number of neurons for each Conv/fully connected layer or range of values tested for regularization. 30

3-4 Example Transformer architecture. Brackets in the MultiHeadAttention layer represent the number of heads and the key dimension. Other brackets show the tested number of neurons for each fully connected layer or range of values tested for regularization. 32

4-1 Accuracy and loss of LSTM$_2$ 36

4-2 Confusion Matrix for LSTM$_2$ 37

4-3 Accuracy and loss of LSTM$_6$ 38

4-4 Confusion Matrix for LSTM$_6$ 38

4-5 Accuracy and loss of CNN-LSTM$_1$ 39

4-6 Confusion Matrix for CNN-LSTM$_1$ 40

4-7 Accuracy and loss of CNN-LSTM$_1$ with L_2 Regularization 41

4-8 Confusion Matrix for CNN-LSTM$_1$ with L_2 Regularization 41

4-9 Accuracy and loss of CNN-LSTM$_3$ 42

4-10 Confusion Matrix for CNN-LSTM$_3$ 42

4-11 Accuracy and loss of CNN$_1$ 43

4-12 Confusion Matrix for CNN$_1$ 43

4-13 Accuracy and loss of CNN$_{12}$ 44

4-14 Confusion Matrix for CNN$_{12}$ 45

4-15 Accuracy and loss of CNN$_{13}$ 45

4-16 Confusion Matrix for CNN$_{13}$ 46

4-17 Accuracy and loss of ATTN$_1$ 47

4-18 Confusion Matrix for ATTN$_1$ 47

4-19 Accuracy and loss of ATTN2$_1$ 48

4-20 Confusion Matrix for ATTN2$_1$ 48

4-21 Accuracy and loss of CNN-LSTM-ATTN2$_3$ 49

4-22 Confusion Matrix for CNN-LSTM-ATTN$_1$ 50

4-23 Accuracy and loss of Astronet 51

4-24 Confusion Matrix for Astronet 51

4-25 Metrics of Top 3 Performers vs Astronet 52

LIST OF TABLES

4.1 Reported Astronet vs Keras Implementation 51

4.2 Best Performers vs Astronet 52

Chapter 1

Introduction

1.1 Exoplanets

For decades astronomers have been searching for exoplanets–planets outside of our solar system. The first confirmed discoveries were made in the early 1990s. Over the next decade, about 50 more exoplanets were discovered. With the introduction of specialized telescopes such as the Kepler Space Telescope and the improvement of data analysis techniques, this number has exploded to thousands of confirmed exoplanets and thousands of more candidates awaiting confirmation. Most of these confirmed planets orbit other stars in systems similar to our solar system; however, some planets are rogue, meaning that they orbit the galactic center rather than a host star. The confirmed planets come in a range of sizes and compositions: 1473 are similar to Neptune and Uranus, 1359 are gas giants similar to Jupiter, 1340 are "Super Earths"–larger than Earth, but smaller than Neptune, these planets are made of rock, gas, or a combination, 163 are terrestrial–similar to Mercury, Venus, Earth, and Mars, these are rocky with an iron-rich core, and the remaining 6 confirmed exoplanets are of an unknown type[1].

[1]https://exoplanets.nasa.gov/discovery/discoveries-dashboard

1.1.1 The Search For Life

For centuries humanity has sought answers to philosophical questions such as: "Where do we come from?" and "Why are we here?" As our knowledge of the universe has expanded, these questions have evolved: "Are there more beings like us in the universe, or are we alone?" and "Are we special?" The primary motivation behind the search for exoplanets is to find planets capable of supporting life. We seek to learn more about our place in the universe and increase our understanding of life. While there are candidates within our solar system, these planets and moons may only harbor simple life forms or may have had lifeforms in the past.

Our knowledge of the necessary building blocks of life is limited to only our understanding of Earth and what conditions allow the sustainability of life forms on our planet. There may be life forms in the universe that require different living conditions. Until there is evidence of what those conditions are, astronomers and astrobiologists are searching for Earth-like planets in extrasolar systems. Primarily, we know that water is essential for life. Thus, planets capable of hosting life must contain liquid water, existing in a range from their star where it is not too hot nor too cold. This habitable zone is called the "Goldilocks zone". Thus far, about 50 confirmed exoplanets exist in the Goldilocks zones of their systems. Exoplanets that may contain life are studied primarily by transmission spectroscopy–the analysis of the star's light through the planet's atmosphere to identify elements present in the atmosphere[2]. If there is evidence of water in the atmosphere or other elements we know are necessary for life, or if there is evidence of smog in the atmosphere, there may be strong evidence for life.

[2]https://exoplanets.nasa.gov/search-for-life/why-we-search

1.1.2 Exoplanet Surveys

Kepler Missions

In 2009 the Kepler Space Telescope was launched with the primary goal of finding Earth-sized planets in the Goldilocks' zone–the distance range from their stars capable of maintaining liquid water. Kepler recorded luminous flux measurements for over 150,000 stars. The telescope's field-of-view was about 0.25 percent of the sky and consisted of primarily very distant stars, hundreds to thousands of light-years away[11]. After encountering technical problems with the telescope, the mission ended in 2013. Despite the lack of control of Kepler, a second mission, K2, lasted from 2014 to 2018. Over 2600 exoplanets have been discovered and confirmed from Kepler data, with thousands more awaiting confirmation. So much data was produced that researchers are still analyzing it and finding new exoplanet candidates[4].

Transiting Exoplanet Survey Satellite (TESS)

TESS was launched in 2018 to continue Kepler's missions. However, TESS was designed to survey the entire visible sky from its orbit–nearly 85%–with a shallower view of stars less than 200 light-years away[3]. The advantage to focusing on closer stars is that ground-based telescopes can do follow-up observations on exoplanet candidates within that range and study their properties.

1.1.3 Detection Methods

Several methods are used to detect and confirm the existence of exoplanets. In this section, we will detail a few of the most common or successful methods.

Radial Velocity

Planets orbit stars and other astronomical bodies due to a gravitational force exerted by the orbited body on the planet. In return, planets also exert a gravitational force on the star they orbit, which causes the star to shift around or "wobble". If the planet or star is massive enough, this movement causes a change in the recorded light instruments receive. Due to the phenomena of doppler shifting, the light emitted will be stretched, or redshifted, as the star moves away from the telescope or squeezed and blueshifted as the star moves towards the telescope. Detecting these periods of a shift in the light emitted from the star is called radial velocity and was one of the first successful methods for detecting exoplanets. This is also one of the most productive methods, having detected 824 confirmed planets, and is typically used to confirm exoplanet candidates that have been discovered with other detection methods[5].

Direct Imaging

As the term implies, direct imaging is a detection method that involves looking for planets transiting, or passing between Earth and its star, in images. Producing photos of transiting exoplanets is difficult since stars are typically much larger than their orbiters and millions of times brighter. Optical telescopes capture large amounts of light from the star that encapsulates the planet. To counter this effect, light blockers are typically added to an instrument. Coronographs are typically used in ground-based telescopes and filter light from stars before the light reaches the instrument's detector. Star shades block light before entering the telescope and are typically separate spacecraft used for space telescopes. As telescopes become more sophisticated, they may be able to photograph atmospheric patterns, oceans, and landmasses on exoplanets. Thus far, 51 confirmed exoplanets have been discovered through direct imaging[5].

Gravitational Microlensing

In Einstein's view of gravity, objects warp the fabric of space; the more massive an object, the more effect it has in warping the space around it. Due to the phenomena, the light from a star can be bent by gravity as a planet transits the star. In data, this appears like the star gets brighter for about a month, then suddenly fades. These events cannot be predicted and are sometimes caused by rogue planets. 106 confirmed exoplanets have been discovered due to gravitational microlensing[5].

Transit Photometry

As a planet transits its star it blocks some of the star's light, preventing it from reaching Earth. This causes a periodic dimming in the star's light measurements, or light curves, over time. The recorded measurements are luminous flux, the measurement of the power of light received by a telescope in Lumens. The periodic dimming in the light curve is referred to as a Threshold Crossing Event (TCE). The depth of the TCE is correlated to the size of the planet–the deeper the curve, the larger the planet, whilst the length of the TCE is correlated to the distance of the planet from the sun–the further the planet, the longer it takes to transit its star, hence a longer dip in the light curve. Large amounts of light curve data have been collected by the Kepler Space Telescope and Transiting Exoplanet Survey Satellite (TESS). Thus far 3306 confirmed exoplanets have been discovered with transit photometry. This is by far the most productive method for discovering exoplanets. My dataset is comprised of transit photometry data from Kepler[5].

1.2 Neural Networks

Neural networks consist of layers of units called neurons that perform a series of operations on input data, learning features in the data that are used to determine an output.

The goal of a neural network may be to learn to classify data, make predictions, translate text, or many other things. Traditional feed-forward networks consist of neurons that receive input from every neuron in the previous layer and pass on their output to every neuron in the next layer, as shown in Figure 1-1. Each neuron calculates the weighted sum of its inputs and passes that sum into an activation function to calculate its output. The weights are trainable parameters for each input to the neuron; due to the dense connectivity of these networks, a deep network has many parameters to learn. Training one of these networks can be computationally expensive and doesn't scale well to larger inputs. These types of networks treat input features independently and are thus incapable of learning spatial or temporal relationships. Kepler light curves contain spatial features and are sequential. I explore different types of networks capable of identifying these features in my research.

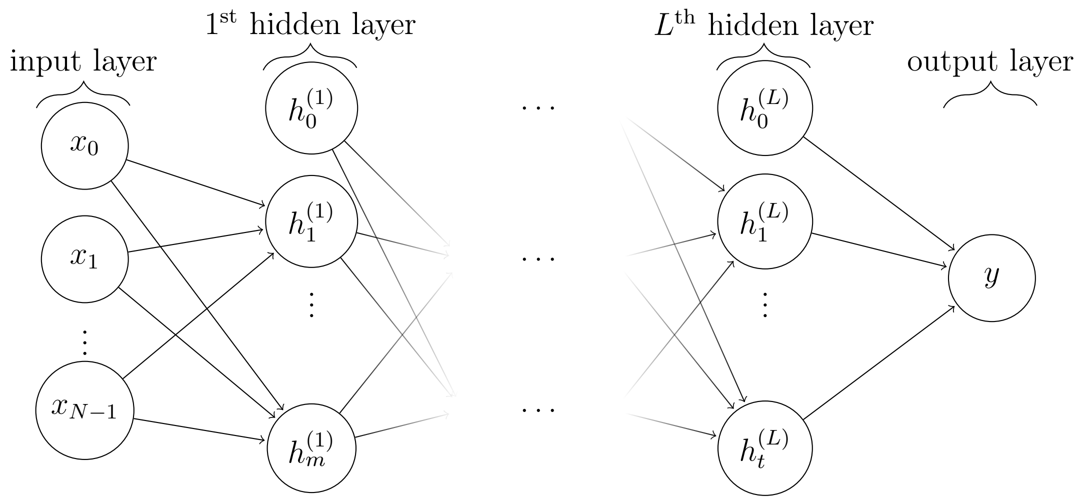


Figure 1-1: Example densely-connected feed-forward network with N inputs, L hidden layers with varying number of units/neurons, and a single output.

1.2.1 Convolutional Neural Networks

Convolutional neural networks (CNNs) are designed to learn spatial features by developing a hierarchy of features that are built up in complexity. CNNs are built from two primary types of layers: convolutional and pooling. Convolutional layers, as shown in

Figure 1-2, consist of a series of filters that convolve across the input, each learning a different shared feature throughout the data[1].

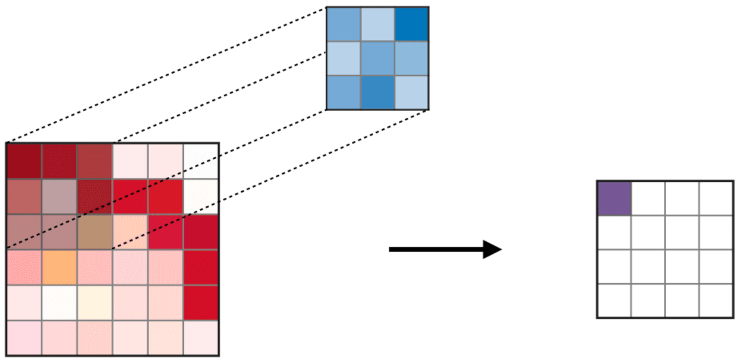


Figure 1-2: Convolutional Layer From Stanford CS230 Course

When the input data is an image, these filters may learn horizontal or vertical lines or edges, then, later on, learn more complex features like an eye. These filters have trainable parameters for a small window that are shared by the entire image, resulting in sparse connectivity in the network and far fewer trainable parameters than a traditional feed-forward network. The result of a convolutional layer is a feature map. The primary goal of a pooling layer is to downsample the input. Pooling layers, as shown in Figure 1-3, consist of a small window that moves along and aggregates the input, finding either the mean or max value in the window and uses that value to represent the window in the output.

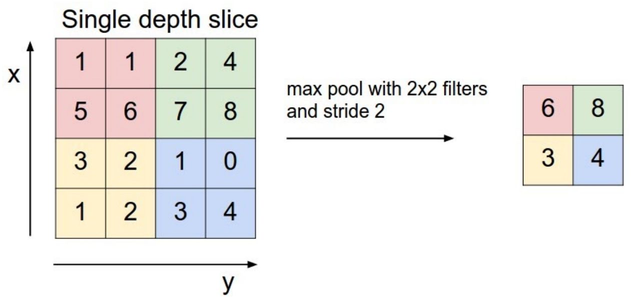


Figure 1-3: Max Pooling From Stanford CS231n Course

Pooling layers simply downsample input, so they have no trainable parameters. This operation passes on a manageable input to the next layer and helps prevent overfitting, which occurs when a network memorizes features in the training data and doesn't generalize well to unseen data. These types of layers alternate in some pattern in a CNN and are typically followed by one or more densely connected layers that ultimately output a desired value or values, such as a classification.

1.2.2 Long Short Term Memory Networks

Recurrent neural networks (RNNs) were developed to handle sequential data, such as text, video, or other time-series data. Besides getting input and producing output like neurons in a traditional network, neurons in an RNN also have connections back to themselves, as depicted in the following figure.

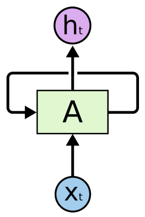


Figure 1-4: Rolled Simple RNN unit[2]

This connection allows the network to learn temporal relationships. Each unit gets the input at time t and the output, or hidden state, of the previous time step, $t-1$, and produces the current hidden state at time t by finding the dot product of the two inputs and passing this through an activation function, typically hyperbolic tangent. The following figure shows what these units look like when they are "unrolled" to depict the time steps.

While these units are capable of learning temporal dependencies, they are too simple to learn over many time steps. RNNs suffer from vanishing gradients, which occurs when weight updates during backpropagation become insignificant, hindering learning. Long Short Term Memory (LSTM) units were developed to counteract this restriction and allow networks to learn over longer sequences.

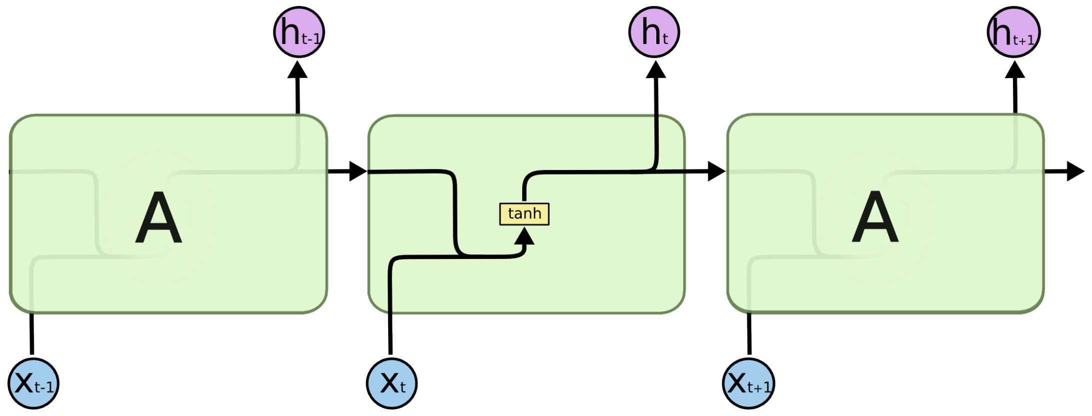


Figure 1-5: Unrolled Simple RNN Unit [2]

LSTM units have a persistent memory-like structure called the cell state. This contains information that can persist through a sequence of time steps, occasionally updated with new information. The LSTM cell contains four "gates" that determine updates to the cell state and calculate the hidden state at the current time step. A depiction of this unit, unrolled, is below.

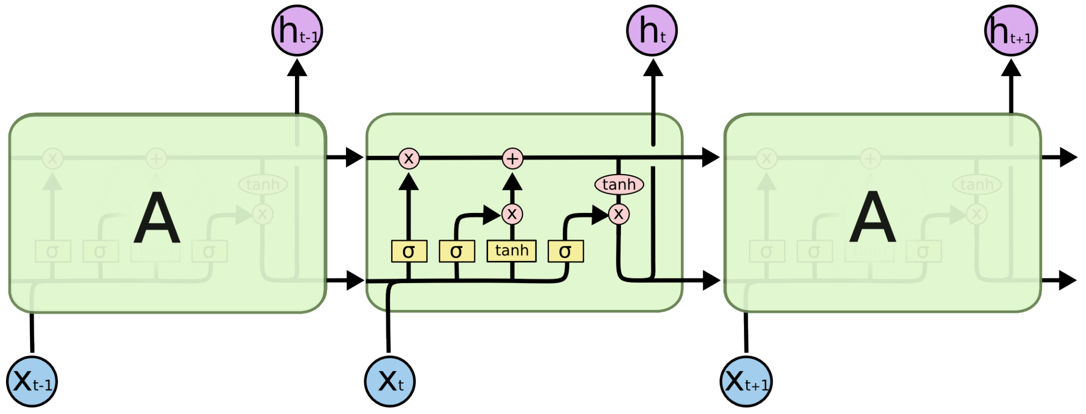


Figure 1-6: Unrolled LSTM Unit [2]

The forget gate determines how much of the information in the previous cell state should persist based on new information from the current time step. The equation for the forget gate is defined as

$$\Gamma_f^t = \sigma(W_f x^t + R_f h^{t-1} + b_f) \tag{1.1}$$

where we find the weighted sum of the current input and the previous hidden states, each having its own set of trainable parameters. This is then passed into the Sigmoid activation function, producing a value in the range [0,1]. The closer it is to 0, the more it forgets, and the closer it is to 1, the more it will remember and the less significant the new input is.

The input gate calculates the information that may be stored at the current time step in the cell state. This is calculated as:

$$\Gamma_i^t = tanh(W_i x^t + R_i h^{t-1} + b_i) \tag{1.2}$$

The update gate determines what, from the input gate, will actually contribute to the updated cell state:

$$\Gamma_u^t = \sigma(W_u x^t + R_u h^{t-1} + b_u) \tag{1.3}$$

The cell state is updated based on the values from the forget gate, input gate, update gate, and the previous cell state:

$$c^t = \Gamma_f^t * c^{t-1} + \Gamma_u^t * \Gamma_i^t \tag{1.4}$$

Finally, the hidden state at the current time step is calculated by the output gate:

$$\Gamma_o^t = \sigma(W_o x^t + R_o h^{t-1} + b_o) \tag{1.5}$$

For a while, LSTM networks have been state-of-the-art for sequential data in tasks such as sentiment analysis and translation tasks. As depicted in the calculations that must be done at each time step and the number of trainable parameters per unit, LSTM networks can be computationally expensive. As a result, newer networks have been developed that can avoid recurrence through the use of attentional mechanisms.

1.2.3 Attention and Transformers

Attentional mechanisms capable of learning what to "focus on" in input, much like the attention that humans and animals have, were developed and used as interfaces in LSTMs[12] and CNNs[46]. Attention involves mapping a query and set of key-value pairs to an output, by calculating a similarity or alignment between the query and keys (or inputs and previous hidden states/outputs in a recurrent layer). The output of the alignment function is put through a softmax function to obtain attention scores. Finally, the result is obtained by computing the dot product between the attention scores and the value vector. In most applications, the key and value vectors are the same. There are various implementations of attention with different forms of computing the alignment of queries and keys[8, 33, 44]. Attention, as defined in [33], is used in some network architectures in this research and is defined as

$$A(Q, K, V) = softmax(QK^T)V \tag{1.6}$$

In 2017 Google Brain introduced Transformers–networks that were developed solely with attentional mechanisms for translation problems, such as translating Engish text to French[44]. Networks for these problems typically involve an encoder that translates the symbolic input to continuous values and a decoder that translates those values back into a symbolic representation. The encoders in Transformers contain stacks of multi-head self-attentional layers and fully connected layers. Scaled dot-product attention is used in the attentional layers that calculate an output as:

$$output = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{1.7}$$

where Q is the query vector, K is the key vector, V is the value vector, and d_k is the dimensionality of the keys/queries. For self-attention, Q, K, and V are the same. The multi-head attention is a linear projection of these vectors in which the attention function is calculated for each in parallel and concatenated to get the final set of values. This allows the network to attend to information from different subspaces from different positions. The decoder is similar in structure to the encoder, but there is additional attention that ensures predictions are only based on previous positions, not future time steps. Vaswani, et al. showed that Transformers can learn long-term dependencies far better than RNNs because the path length for signals to propagate during training for Transformers is only $O(1)$ while the path length is $O(n)$, where n is the length of a sequence, for RNNs. Transformers do not have to perform backpropagation through time to update weights. Vaswani, et al. also showed that their Transformer outperformed state-of-the-art models for translation problems with less computation cost.

1.3 Astronomical Data Analysis

In the early days of the Kepler program, astronomers manually removed false positives from the Kepler pipeline and continued to search for exoplanets in the data manually. As the amount of data increased, data was released to the public, and "citizen science" initiatives were started, where average people interested in astronomy could get involved in manually searching data for exoplanets[18]. Eventually, automation was added to the process of culling the Kepler pipeline with the introduction of Robovetter, a decision tree designed to remove false-positive TCEs[40].

Soon researchers started applying classical machine learning techniques such as random forests and clustering to classify planet candidates. In this paper, I consider using deep learning approaches for exoplanet detection in light curve data from Kepler, namely LSTMs, CNNs, and attentional interfaces. In chapter 2 I detail other approaches to the problem using deep learning. I discuss the dataset, data processing, model architectures, and experiments in Chapter 3. Chapter 4 contains the experiment results. Finally, I provide analysis and future directions in Chapter 5.

Chapter 2

Related Work

2.1 Deep Learning on Kepler Data

2.1.1 Astronet

C Shallue and A Vanderburg[40] experimented with applying linear logistic regression, fully connected network, and convolutional neural network models to search for exoplanets in a dataset they curated from the Kepler Space Telescope. Initially, observations were pulled from the NASA Exoplanet Archive[1], including each star's Kepler ID, number of Threshold Crossing Events (TCEs), the TCE period, TCE duration, and the label associated with the event. The dataset includes 3600 planet candidates (PC), 9596 astrophysical false positives (AFP) such as eclipsing binary stars, and 2541 non-transiting phenomena (NTP). These labels were converted to 1 for "planet" (PC) and 0 for "non-planet" (AFP/NTP). Presearch data conditioning (PDC) light curves for the observations collected were downloaded from the Mikulski Archive for Space Telescopes[2]. PDC curves have instrumental noise removed from the raw curves, as well as any interference from stars near the target[17].

[1]https://exoplanetarchive.ipac.caltech.edu/
[2]https://archive.stsci.edu/

Each light curve was normalized by fitting a spline to the curve and dividing the curve by this spline to remove non-transit variability in the normal brightness of the star[40]. During this process, outliers were removed and the transits were preserved by removing and re-introducing the transits after normalization. Each observation contains two light curves, a "global" view of the entire light curve and a "local" view that is 10% of the length of the global view. Each of these views has multiple periods of the same TCE layered–a process called "phase folding" often used in astronomy to remove noise and intensify periodic events in the signal. Examples of the two views for different observations can be seen in Figure 2-1. The first sample is a strong planet candidate, the second is a planet candidate with a long period (entire transit in 1 bin in global view), and the third is an eclipsing binary star, which looks just like a planet candidate in the local view.

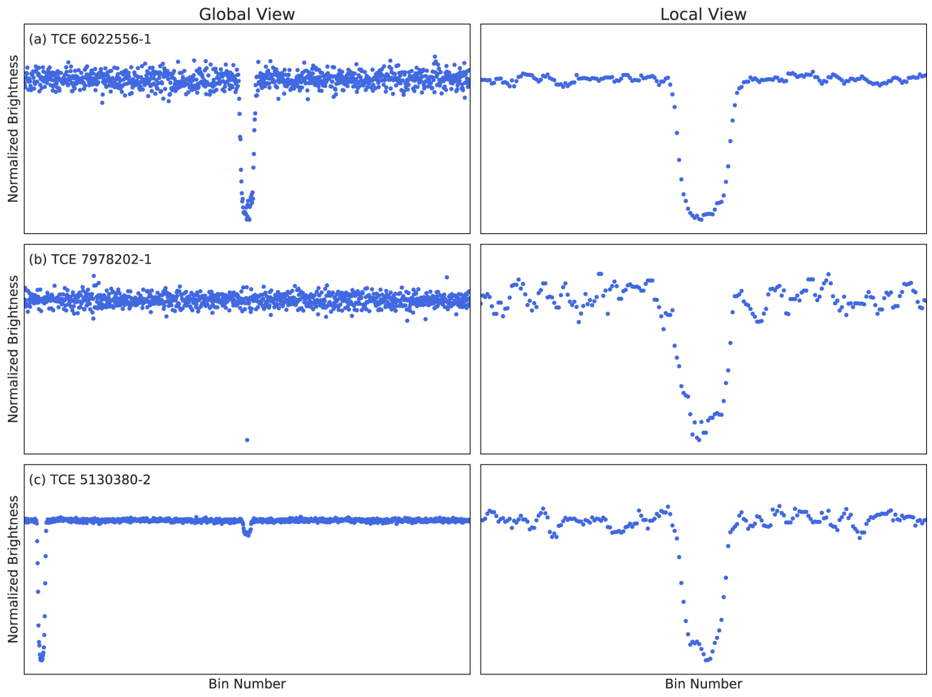


Figure 2-1: Global and local views of three observations in the dataset.[40]

Shallue and Vanderburg augmented the training set during training by randomly applying horizontal flips to half of the input. For each model, they experimented with different architectures for these inputs: one for just the local view, one for just the global view, and one that would work with both the local and global views. By far their best performer was the convolutional network that worked with both the local and global views.

Developed in TensorFlow, Astronet contained multiple one-dimensional convolutional layers and max-pooling layers before a few fully connected layers. The best performing version of Astronet was the architecture that took both the local and global views as input, which were passed through separate convolutional columns before being connected and passed together through the dense layers. Hyperparameters were tuned automatically. Figure 2-2 illustrates the final architecture and depicts the kernel size and number of feature maps for each convolutional layer, the pool size and stride for the max-pooling layers, and the number of units for each fully connected layer.

Their paper states that the ReLU activation function was used for every hidden layer except the final layer, which used sigmoid to return the probability that the observation was a planet. ReLU is a simple piecewise function that returns the input if the input is positive; otherwise, it returns zero. The sigmoid function returns a value in the interval (0, 1) and is typically used for binary classification.

The Adam optimizer was used for training Astronet. Adam is a popular gradient descent learning algorithm because it combines aspects of two very successful algorithms–AdaGrad and RMSProp. Typical stochastic gradient descent uses a single non-adjusting learning rate for all weights. Like AdaGrad, Adam uses learning rates specific for each parameter. These learning rates adapt over time using a moving average of the gradient and squared gradients, expanding on the adaptive behavior of RMSProp[29].

Shallue and Vanderburg achieved a precision of 0.90, recall of 0.95, an accuracy of 0.96, and AUC of 0.988. I compare my results to the results of Astronet in Chapter 4.

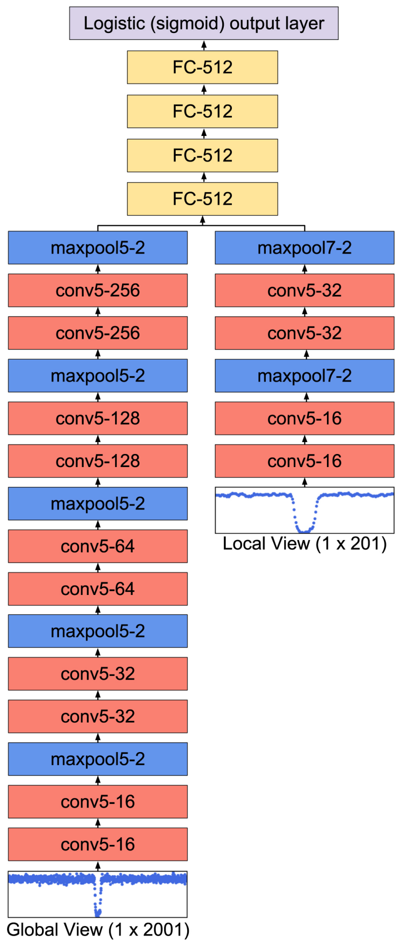


Figure 2-2: Astronet architecture, where each layer is labeled as conv<kernel size>-<num feature maps>, maxpool<size>-<stride>, and FC-<num units>[40]

2.1.2 Exonet

Frontier Development Lab (FDL)[3] is an eight-week research workshop that seeks to apply artificial intelligence to varying areas of space science where each team participating in FDL is made up of machine learning experts and space science researchers. FDL is a partnership between the National Aeronautics and Space Administration, European Space Agency, and commercial partners such as Google and NVIDIA. Ansdell, et al.[6] developed Exonet in FDL, which makes improvements to Astronet. Their dataset was collected and processed similarly to the dataset used by Shallue and Vanderburg. In addition to the global and local views of the light curves, Ansdell, et al. produced

[3]https://frontierdevelopmentlab.org

centroid curves for each sample, which is a sequence of the pixel position of the center of light, to help distinguish eclipsing binary stars, a typical astrophysical false positive, from transiting exoplanets. Figure 2-3 depicts a couple of example centroid curves with their associated light curves for a sample containing a transiting exoplanet and one sample containing an eclipsing binary star. As one star eclipses the other in a binary system, the centroid shifts. These centroids are produced for each of the local and global views for each sample and are provided with the associated view as a second channel of the input to the augmented Astronet. During training, the training samples were augmented in the same fashion as done by Astronet with the addition of random Gaussian noise to each curve.

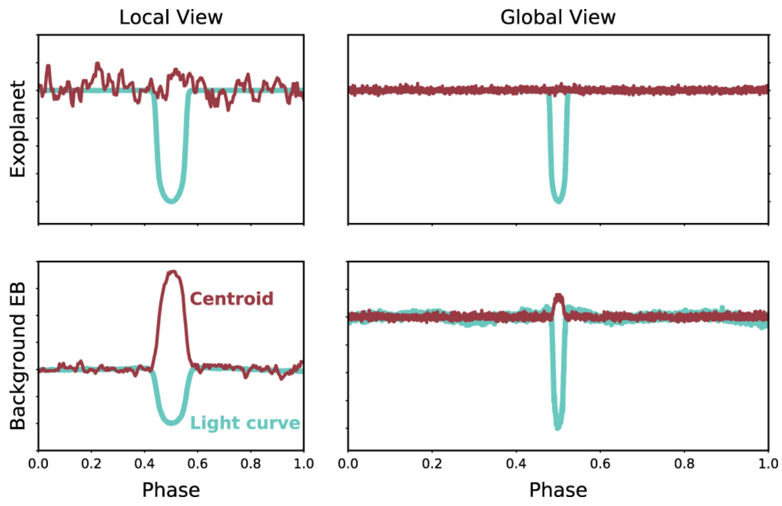


Figure 2-3: Examples of light curves with centroid curves for a transiting exoplanet and an eclipsing binary star[6]

In addition to producing these centroids, Ansdell, et al. collected stellar parameters for each observation, including information such as the star's effective temperature, surface gravity, radius, and mass. These parameters were normalized and concatenated to the outputs of the convolutional columns of Astronet.

Ansdell, et al. did not make any changes to the architecture of Astronet–all layers and hyperparameters for each layer remained the same. They also used the same batch size, number of epochs, and initial learning rate for training as Shallue and Vanderburg used. By simply adding the additional inputs and data augmentation steps, Exonet's accuracy reached 0.975 and precision reached 0.98.

2.1.3 Astronet-K2

Anne Datillo, et al. built on Astronet to develop a network capable of detecting exoplanets in data from the follow-up mission to the Kepler mission, K2, which also used the Kepler Space Telescope[14]. In their work, they produced the input to their model similarly to [40]. In addition to using the local and global views of each light curve, Datillo, et al. included scalar features, such as the planet to star radius ratio and the depth of the dimming in the light curve. These values were concatenated to the output of the two convolutional columns before being fed to the fully connected layers in the network. The network architecture was very similar to [40] except that the max-pooling layers in the convolutional column for the local view had 7x1 sized kernels. During training, they also used random horizontal flips to augment the data and increase the training set size. Astronet-K2 reached an accuracy of 0.978 and an AUC of 0.988.

2.1.4 Application of LSTM

Trisha Hinners, et al. attempted to use an LSTM network for varying classification and regression tasks on Kepler light curves[25]. They did not fold the light curves as Shallue and Vanderburg did; instead, data was simply downsampled by using every 10th data point in the light curve. By not folding the data, their curves may have had weaker signals. Folding the light curve results in stronger signals and less noise due to constructive and destructive interference when the signals are summed. The results after downsampling were then normalized and 2.5σ clipped to remove outliers. Their input for the LSTM consisted of 48.5k samples of 7k data points. LSTMs are limited to training over sequences of no longer than a few hundred in length; sequence inputs of 7k in length will lead to exploding gradients and unreliable learning. The LSTM implemented by Hinners, et al. consisted of 2 LSTM layers with 16 nodes each and a final dense layer with softmax for classification. This network is quite small, with only about 2000 trainable parameters. Their classification task was different from that of Astronet and Exonet–instead of identifying whether a TCE in a light curve belonged to a planet or not, they attempted to determine how many planets existed in the curve or the number of Kepler Objects of Interest (KOIs) or TCEs were present in the light curve. This is a more complicated task than Astronet or Exonet, which just focus on vetting planet candidates from false positives. However, there were clearly issues with their input and network architecture that led to their low accuracy of 0.52. Ultimately, Hinners, et al. abandoned the LSTM network to focus on feature engineering techniques. I haven't found other significant uses of LSTMs or other recurrent networks for the task of classifying Kepler light curves.

2.2 Deep Learning on Other Exoplanet Surveys

2.2.1 Simulated Transiting Exoplanet Survey Satellite Data

H. P. Osborne, et al. applied a slightly augmented version of Exonet to light curve data that was simulated before the launch of the Transiting Exoplanet Survey Satellite (TESS) for the same task of determining whether a TCE was caused by a planet candidate or false positive[36]. Astronet and Exonet each used pre-search data conditioning (PDC) curves from Kepler data, which are light curves that only have instrumental errors removed. Osborne, et al. used this simulated curve, but also the data validation (DV) curve, which has non-planetary signals detrended from the curves that contain known candidates. These two curves were processed similarly to the data used for Exonet with folded local and global views. In addition to these two curves, Osborne, et al. used the centroid curves and scalar transit and stellar parameters that were similarly used in Exonet.

2.3 Neural Network Scaling

2.3.1 EfficientNet

Researchers at Google studied the process of scaling CNNs[43]. Typically, CNNs are scaled by increasing the number of layers (depth) or increasing the number of filters at each layer (width). Less commonly, CNNs are scaled by image resolution. Usually, one or two of these are increased to increase accuracy, but there is a dropoff or plateau in performance as networks get too deep or wide. Tan and Le[43] discovered that it is necessary to scale all three aspects at once to develop efficient, high-performing networks. Their EfficientNet models are capable of performing as well as the top state-of-the-art models for the ImageNet classification task, with far smaller networks. The task of classifying light curves is not nearly as complicated as the ImageNet task, but it may be possible to develop a CNN that is significantly smaller than Astronet but achieves the same metric scores.

Chapter 3

Methods

3.1 Dataset

Shallue and Vanderburg open-sourced the dataset they curated for Astronet[1], which I used for training and testing my models. The process for building their dataset was described in section 2.1.1. The dataset containing 3600 exoplanet candidates and 12,137 non-exoplanet samples was pre-split at random, 80% into training, 10% into validation, and 10% into test, and stored as TFRecord–TensorFlow Record–files with the two views, additional data from the NASA Exoplanet Archive, and the labels for each observation. The validation set was used for hyperparameter tuning, while the test set was used exclusively for testing the performance of each model on unseen data. For training the models, a TensorFlow dataset was created by batching the observations for each set with the local and global views as input and the label as output for each observation.

3.1.1 Data Augmentation

Data augmentation is introduced to the training set either before or during training to increase the size of the training set and produce a model that does not overfit, meaning

[1]https://github.com/google-research/exoplanet-ml/tree/master/exoplanet-ml/astronet

it generalizes well to unseen data. Vanderburg and Shallue[40], as well as Ansdell, et al.[6], randomly flipped local and global views horizontally while training Astronet and Exonet, respectively. Another technique used by Ansdell, et al. was the addition of white Gaussian noise to some of the samples at random. In my research, I also employed these techniques. Additionally, I augmented some samples by "rolling" them by choosing a small number of time steps to shift the data forwards or backward, carrying over the data back to the other side. These three techniques are safe because they do not alter the transits and transits are symmetrical; thus, no data risks changing classes after these three techniques are applied. Other data augmentation techniques that are common for time series data, such as window cropping, window/dynamic time warping, permutations, scaling, or slope-trending were avoided as I thought they would alter the light curve data too much and risk changing the class of the data.

3.2 Models

The following sections describe the neural network architectures, which were all implemented in Keras[2].

3.2.1 Activation Functions

Activation functions are used at each layer of a neural network to transform the weighted sum of the input. There are several activation functions, each having an impact on the performance of the model. Activation functions must be differentiable for learning to occur during backpropagation, which updates weights based on the gradient of the error at each layer. They also must be nonlinear. The three most common activation functions are hyperbolic tangent (tanh), rectified linear unit (ReLU), and logistic (sigmoid). Tanh (equation 3.1) is typically used in the hidden layers of recurrent networks, as described

[2]keras.io

in Chapter 1.

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.1}$$

Tanh has a major drawback: limited sensitivity except around 0, which causes large or small values to saturate the function at 1 and -1, respectively. This leads to a vanishing gradient problem, where error gradients become too small to make significant weight updates, resulting in a lack of learning. The same issue occurs with sigmoid activations (equation 3.2) in hidden layers because the function is saturated near 0 and 1.

$$S(x) = \frac{1}{1 + e^{-x}} \tag{3.2}$$

Sigmoid activations are still used for gates in LSTMs (such as to determine how much of the previous cell state to keep) but are primarily used as an output activation for binary classification.

There are several variants of ReLU[21], but the basic ReLU activation function is defined as

$$f(x) = max(0, x) \tag{3.3}$$

One clear advantage to this function is that it is much simpler in terms of computation than either the tanh or sigmoid functions. In addition, it generally solves the problem of vanishing gradients. ReLU activations are generally considered to be the best choice for densely connected and convolutional layers and made major improvements in state-of-the-art models in areas such as ImageNet classification[31]. For the architectures in this research, all densely connected and convolutional layers use the ReLU activation, except for the final output layer. The output layer uses sigmoid to determine the probability that the given sample light curve contains a transiting exoplanet.

3.2.2 Training

Loss Function

Loss functions are required to measure the performance of a model and provide information for the backpropagation algorithm to update weights in each layer. I used the typical loss function for binary classification, binary cross-entropy, which is calculated as

$$L = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i) \tag{3.4}$$

Optimizer

The primary goal of the optimizer is to minimize the loss function. The gradient descent algorithm does this by calculating the gradient of the loss function with respect to weights at each layer, scales this by a parameter called the learning rate, then updates weights with the negative scaled gradient. Gradients tell the algorithm in which direction to update parameters, while the learning rate tells it how much of an update to make. A learning rate that is too large may cause the algorithm to overshoot the minimum and bounce between slopes in the cost function; on the other hand, if the learning rate is too small, there may be no significant weight updates or training may take a long time. Traditionally, the gradient is calculated for the entire training set, which is precise but often intractable for large datasets. Due to many samples in a dataset being correlated, batch gradient descent can have a lot of redundant calculations. On the other hand, batch training guarantees converging to a minimum (global or local) in the loss surface.

To combat memory issues and computation cost, gradient descent is typically done in small batches. Mini-batch gradient descent approximates the gradient after a small batch of samples and uses this approximation for weight updates. The extreme version, stochastic gradient descent, performs weight updates after every sample. This is less likely to get stuck in an undesirable local minimum but can have a lot of variance in updates leading to bouncing around and never converging to a minimum. Mini-batch gradient descent is more likely to converge and is typically used in state-of-the-art models. The optimum batch size varies depending on the application, dataset, and hyperparameters, but is typically in the range $[32, 256]$. Some research suggests that smaller batches outperform larger batches[34].

There are several optimizers, some with adaptive per-parameter learning rates such as Adam[29], RMSProp[3], and Adagrad[16], and some with non-adaptive learning rates, such as stochastic gradient descent with various types of momentum. The ideal optimizer can vary depending on the problem and the dataset. Adam has separate learning rates for each trainable parameter, which allows the optimizer to finely tune weight updates. Updates are made using an exponentially decaying average of past gradients and past squared gradients. Using these, momentum can be built up, so Adam will move faster if it is moving in the same direction as in the past. Adam is a modern optimizer choice that was used for Astronet and Exonet, so the models in this research are all trained with Adam.

Learning Rate

The learning rate, as stated previously, scales the weight updates during training. An initial learning rate must be chosen for Adam; this was chosen experimentally from a search space of $[0.0001 - 0.006]$. In addition to tuning the initial learning rate, the ReduceLROnPlateau callback was used during training, which reduces the learning rate by a given factor when a given metric doesn't improve for a given number of training epochs. The callback monitored the accuracy of the validation set and halved the learning rate if the accuracy didn't improve for 10 epochs.

[3]http://www.cs.toronto.edu/ tijmen/csc321/slides/lecture_slides_lec6.pdf

3.2.3 Regularization and Dropouts

Regularization is introduced to neural networks to penalize complex models and reduce overfitting. Instead of simply trying to minimize a loss function, an optimizer will try to also minimize a regularization term, which represents complexity. One type of regularization is L_2 regularization, which is the sum of the squares of all of the weights. Weights with high absolute values will be the primary contributors to the model's complexity. Introducing L_2 regularization to the LSTM layers of the LSTM and CNN-LSTM networks successfully decreased overfitting and increased performance. Using L_2 regularization in dense layers in any of the networks was usually unhelpful and decreased performance.

Dropouts are another regularization tool to combat overfitting[41]. Given a probability, p, a dropout layer will "drop" or ignore some connections from one layer to the next layer, adding a sparsity to the next layer's input. At each iteration, the "dropped" units are randomly sampled so the network is essentially training with slightly different networks each time. Some experiments used dropouts with varying values for p including 0.0, 0.2, and 0.4 in LSTM layers or the dense layers at the end of each of the networks.

3.2.4 Long Short Term Memory Network

The first model architecture I built included separate LSTM layers for the local and global light curves. I experimented with different values for the number of hidden units (64, 128, 256), as well as the amount of L_2 regularization that was applied to weights in these layers (0.0001-0.001). The outputs of the LSTM layers were concatenated, then passed through a variable number of dense layers. I experimented with the number of dense layers (2 or 3) and the number of neurons in each layer (16, 32, 64, 128), which stayed the same or decreased as depth increased. Dropouts (0-0.4) were used after the initial dense layer to prevent overfitting. Figure 3-1 depicts the general architecture.

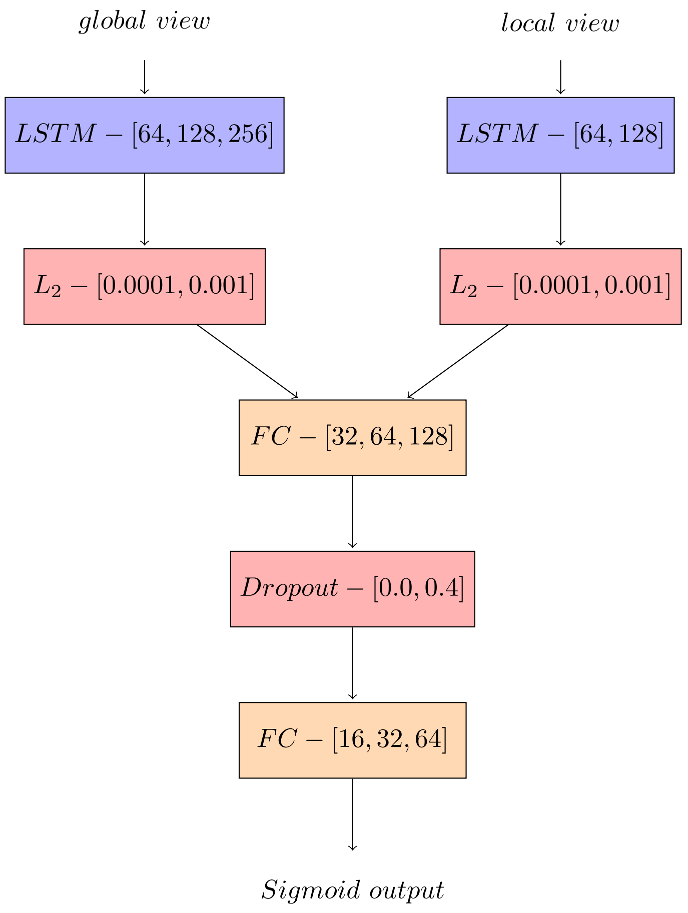


Figure 3-1: Example LSTM architecture. Brackets show tested number of neurons for each LSTM/fully connected layer or range of values tested for regularization.

3.2.5 Convolutional Long Short Term Memory Network

With a relatively low performance from the basic LSTM, I decided to experiment with a convolutional recurrent network, which introduced separate convolutional columns before LSTM layers for each input, as can be seen in Figure 3-2. CNN-LSTMs have been used for various problems involving sequential data, including text classification, sentiment analysis, and translation. The convolutional blocks in the models followed a pattern of two Conv1D convolutional layers followed by one MaxPooling1D layer. Experiments on this architecture included the number of filters (16, 32, 64, 128) and the filter size (3, 5, or 7) in each convolutional layer and the number of convolutional blocks for each column. The number of filters in the convolutional layers doubled in the following block. The convolutional layers used "same" padding so downsampling only happened during pooling, and the pooling layers had varying window sizes (2, 3, 5, or 7) and used a stride of 2. The output of these convolutional columns was passed into an LSTM layer for each column, then concatenated and passed through dense layers. Experiments also were done on the number of neurons in each LSTM layer and fully connected layer, as well as on the L_2 and dropout regularization, as in the LSTM models.

3.2.6 Scaled Down Convolutional Neural Network

Besides working with different types of models from Astronet, I was interested in attempting to scale down Astronet to develop a CNN with comparable performance but far fewer trained parameters, thus reducing computation cost. This model is similar to Astronet in Figure 2-2 but contains fewer convolutional blocks for each input (3 for the global view, 2 for the local view) and narrower and fewer (2 instead of 4) dense layers. The convolutional blocks in the global column followed a pattern of 16-32-64 filters and the local column had 16-32 filters. Filter size for each block was experimented with, as well as the number of dense layers (2, 3, or 4), number of neurons in each dense layer (16, 32, 64, 128), and the amount of dropout regularization (0-0.4). A depiction of the scaled-down CNN can be seen in Figure 3-3.

3.2.7 Attention

Attention was used in a couple of different ways. In the LSTM architecture, an attention layer was added between the LSTM and dense layers. This allows the LSTM network to analyze the output of the LSTM and identify what parts are most crucial for making a classification. The Keras Attention layer calculates the output as

$$A(Q, K, V) = softmax(QK^T)V \tag{3.5}$$

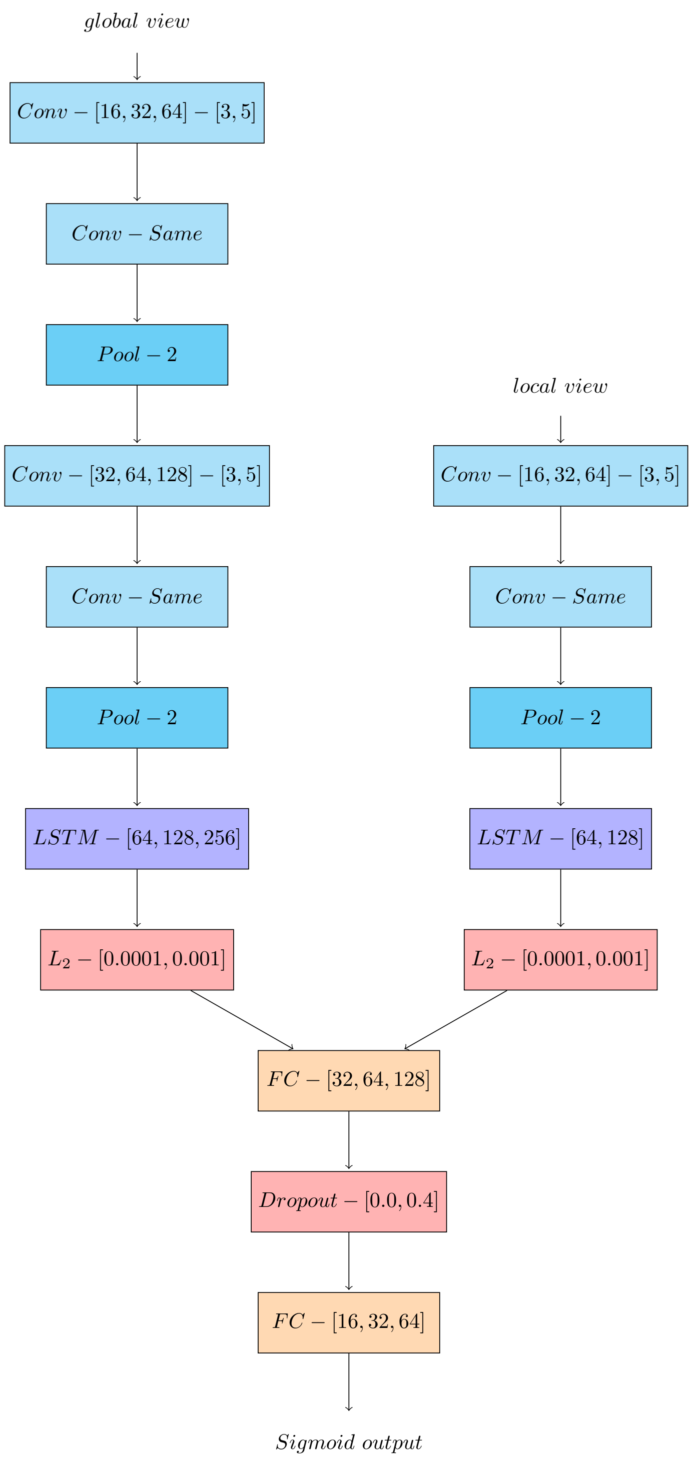


Figure 3-2: Example CNN-LSTM architecture. Brackets show tested number of neurons for each Conv/LSTM/fully connected layer or range of values tested for regularization.

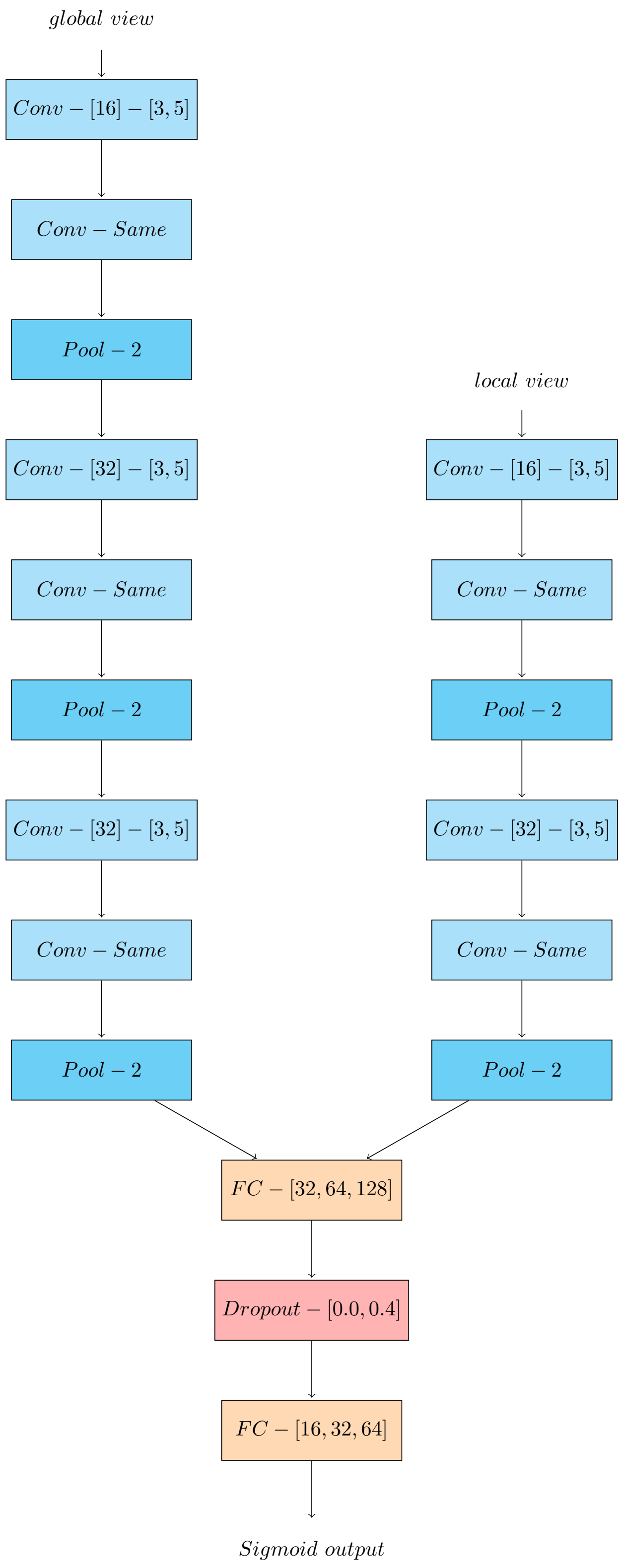


Figure 3-3: Example CNN architecture. Brackets show tested number of neurons for each Conv/fully connected layer or range of values tested for regularization.

where Q refers to the query vector, K is the key vector, and V is the value vector. Typically, the key and value vectors are the same. In one version of the attentional LSTM, there were separate Attention layers for each input after their respective LSTM layers, which used self-attention (key, value, and query vectors were all the output of the LSTM). This model was called LSTM-ATTN$_N$, where N refers to the iteration. Another version, LSTM-ATTN2$_N$, had one Attention layer (not two) that used the output of the local view LSTM as the query vector and the output of the global view LSTM as the key/value vector. The output of this Attention layer was passed into the following dense layers. The Attention layer has no trained parameters and no arguments, so experiments consisted of the same variable values described in the prior LSTM section.

In the CNN-LSTM architecture, attention was used in 3 configurations. In CNN-LSTM-ATTN$_N$, similarly to LSTM-ATTN$_N$, an Attention layer was added for each input after the respective LSTM layer. In CNN-LSTM-ATTN2$_N$, similarly to LSTM-ATTN2$_N$, a single Attention layer was used after the LSTM layers that used the output from both LSTM layers. CNN-LSTM-ATTN3$_N$ used Attention layers between the final convolutional block and the LSTM layer.

The final architecture that used attention used transformers implemented with Keras' MultiHeadAttention layer. ATTN$_N$ used a single MultiHeadAttention layer with the local view as the query vector and the global view as the key/value vector. The output of this layer was flattened then put through dense layers. Experiments were conducted on the amount of L_2 regularization applied to the transformer (0.0001-0.001), the number of heads (2-6) and the key dimension in the transformer, the number of dense layers (2 or 3), the number of neurons in each dense layer (16, 32, 64, 128), and the amount of dropout regularization used (0-0.4). ATTN2$_N$ used separate transformer layers with self-attention for each of the inputs. A depiction of this architecture can be seen in Figure 3-4. The output of these layers was flattened and concatenated before dense layers. Hyperparameters for this model were the same as for ATTN$_N$, but with separate variables for the number of heads and the key dimension in the separate transformer layers.

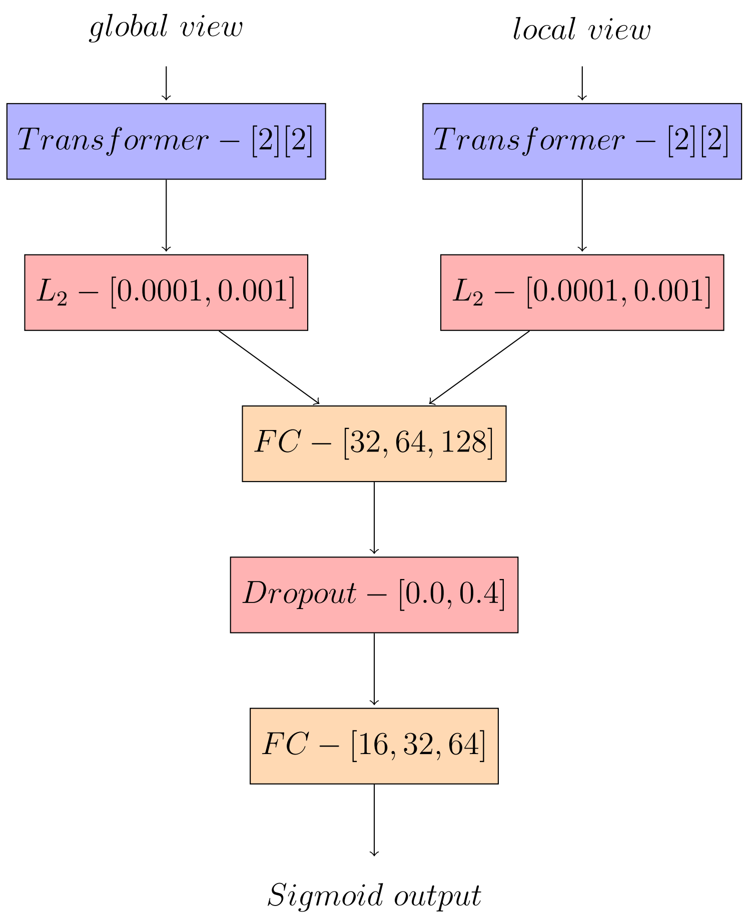


Figure 3-4: Example Transformer architecture. Brackets in the MultiHeadAttention layer represent the number of heads and the key dimension. Other brackets show the tested number of neurons for each fully connected layer or range of values tested for regularization.

3.3 Hyperparameter Optimization

Hyperparameter optimization for neural networks is an iterative cycle that involves choosing a set of hyperparameters for the various layers such as the number of convolutional filters or filter size and for the network such as the learning rate or batch size for training, followed by training the model and evaluating the performance on a test set, then restarting. There are several automated approaches to this task, including the use of genetic algorithms, grid search, random search, and Bayesian model-based optimization. Grid search and random search are each uninformed algorithms that waste time testing poor-performing choices. In hyperparameter experiments, I use Bayesian optimization[30]. Bayesian optimization makes informed choices on what values to choose from the search space for each hyperparameter based on the previous performance of each value. The goal is to minimize or maximize the output of an objective function, which in the case of neural networks involves training a model, evaluating the performance on a test set, then returning a desired metric to optimize. This process begins by building a surrogate probability model of the aforementioned objective function, using it to estimate what hyperparameters should be chosen next. The hyperparameters with the best performance on the surrogate model are chosen and used in the objective function. When the results are calculated, the surrogate model is updated. This process is repeated for a specified number of trials.

I used the hyperopt library for Python[10] to conduct hyperparameter optimization. For each hyperparameter that I wanted to run experiments on for finer tuning, I defined a search space based on initial manual choices and performance. The entire search space for the optimization is defined by a Python dictionary; an example search space for LSTM optimization is below.

```
space = {
    'batch_size': 128,
    'epochs': 100,
    'lr': hp.uniform('lr', 0.0001, 0.001),
    'loc_n1': hp.choice('loc1', [64, 128, 256]),
    'loc_n2': 0,
    'glob_n1': hp.choice('glob1', [64, 128, 256]),
    'glob_n2': 0,
    'n1': hp.choice('n1', [64, 128]),
    'n2': hp.choice('n2', [32, 64]),
    'n3': hp.choice('n3', [16, 32]),
    'n_layers': 1,
    'drop1': hp.uniform('drop1', 0, 0.4),
    'drop2': hp.uniform('drop2', 0, 0.4)
}
```

lr refers to the initial learning rate for training, which is chosen uniformly from the range $[0.0001, 0.001]$. loc_n1 refers to the number of neurons in the local view LSTM layer. The `hp.choice` function chooses a value for the next experiment from the given list of options. Because n_layers is set to 1, no search space was defined for a second LSTM layer. $glob\_n1$ refers to the number of neurons in the global view LSTM layer. $n1$, $n2$, and $n3$ were the neurons for the dense layers after the LSTM layers. $drop1$ and $drop2$ refer to the amount of dropout applied after the first and second dense layers, respectively. I ran 50 or 75 trials for each model to minimize the loss on the validation set.

Chapter 4

Results

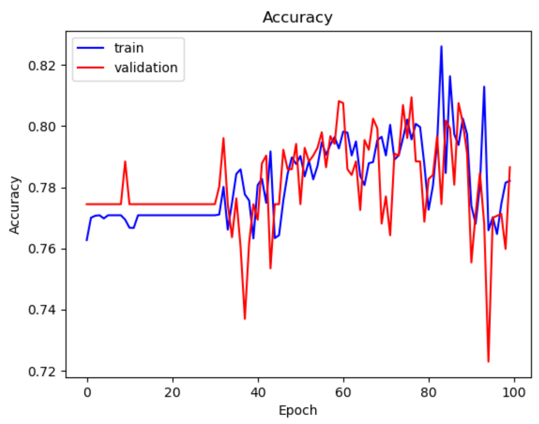
4.1 Metrics

Four metrics were used to measure the performance of the developed models: accuracy, precision, recall, and AUC. Accuracy represents what percentage of samples were correctly classified and is calculated by $\frac{TP+TN}{TP+TN+FP+FN}$, where TP refers to the number of true positives, or samples classified as having exoplanet transits that do, indeed, have exoplanet transits, FP refers to the number of false positives or samples classified as having exoplanet transits that do not have exoplanet transits, TN is the number of true negatives, and FN is the number of false negatives. Precision refers to the percentage of the samples classified as containing exoplanet transits that were correct and is calculated by $\frac{TP}{TP+FP}$. Recall is the number of actual positives that were correctly classified, calculated as $\frac{TP}{TP+FN}$. Finally, AUC refers to the area under the receiver operating characteristic (ROC) curve. The ROC graph depicts the performance of a model at different thresholds used to determine which class a sample belongs to based on a classification output; the performance is the true positive rate, $\frac{TP}{TP+FN}$, vs the false positive rate, $\frac{FP}{FP+TN}$. The default classification threshold is typically 0.5, which means that any output below 0.5 means that the model classifies the sample as a negative sample, and anything above 0.5

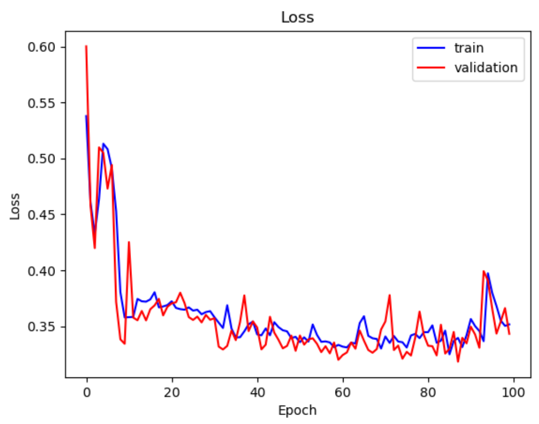
is classified as positive. Increasing the threshold leads to more items labeled as false, which leads to an increase in precision and a decrease in recall; decreasing the threshold has the opposite effect. The optimal threshold is problem-dependent.

4.2 LSTM

Many different LSTM models were experimented with and most had accuracies below 0.80. One of the best performing single layer LSTMs, $LSTM_2$, achieved an accuracy of 0.81, precision of 0.56, and recall of 0.88 on the test set. This model contained separate LSTM layers for the local and global views with 128 units for each layer. The outputs of the LSTM layers were concatenated, then put through a dense layer with 64 neurons and a dropout of 0.2, followed by a dense layer with 16 neurons and a dropout of 0.2, and a final layer with 1 neuron to calculate the result. Models with fewer or more units in the LSTM layer underfit or overfit, respectively. Graphs of the accuracy and loss on the training and validation sets during training can be seen in Figure 4-1.



(a) Accuracy of $LSTM_2$



(b) Loss of $LSTM_2$

Figure 4-1: Accuracy and loss of $LSTM_2$

As can be seen on the left, the accuracy of the model stayed about the same for the first 30 epochs of training. The accuracy increased for another 40 or so epochs before decreasing back to where it originated. The loss graph on the right shows the training and validation losses decreased rapidly at first, then didn't have significant changes after about 10 epochs. The losses slowly declined, then crept back up.

Figure 4-2 is the confusion matrix of this model, depicting how the actual labels of the test set line up with the predicted labels of the model. LSTM_2 correctly identified 317 out of 360, or 88%, exoplanets and 965 out of 1214, or 79%, non-exoplanets. Overall, this is accuracy of 81.4% on the test set. The high number of false positives indicates low precision on the test set.

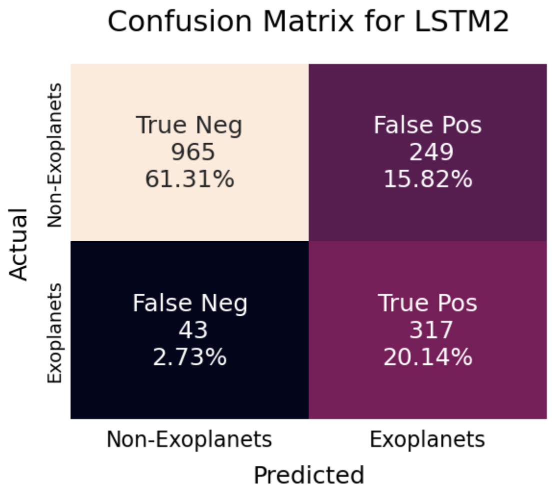
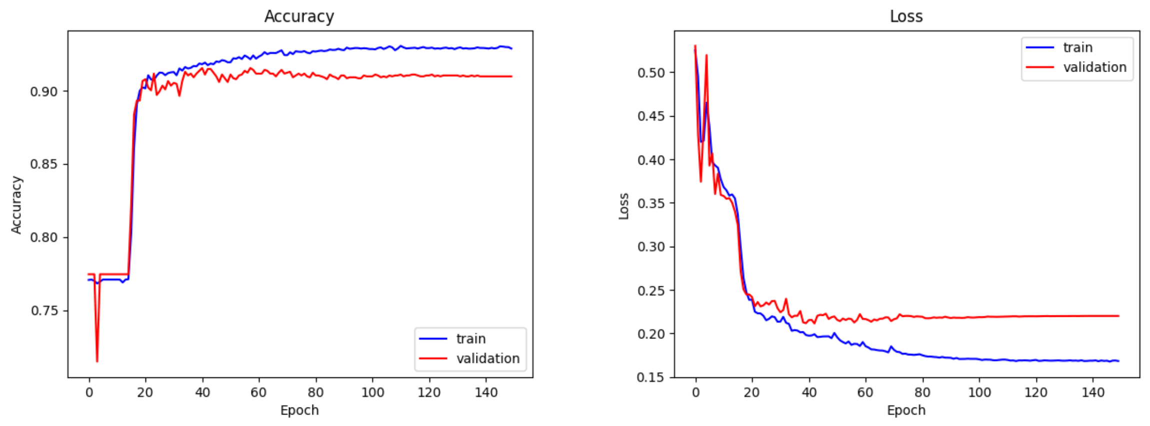


Figure 4-2: Confusion Matrix for LSTM_2

By far the best performing LSTM model, LSTM_6, had two LSTM layers for the global view, the first contained 128 units and the second contained 64 units, and one LSTM layer with 64 units for the local view. All three LSTM layers had L_2 regularization for the weights with a value of 0.0001. The output of the final LSTM layers was concatenated, put through a dense layer with 64 neurons and 0.2 dropouts, a dense layer with 32 neurons, and finally the output layer. This model was trained for 150 epochs with an initial learning rate of 0.006.

Figure 4-3 shows the accuracy and loss for the training and validation sets during the span of training. As can be seen in the loss plot, LSTM_6 was overfitting; increasing the

L_2 regularization may help to hinder this.



(a) Accuracy of LSTM_6 (b) Loss of LSTM_6

Figure 4-3: Accuracy and loss of LSTM_6

Figure 4-4 shows the confusion matrix for LSTM_6. This model had slightly lower performance on identifying exoplanets, correctly identifying 314 of the 360 samples containing exoplanets; however, the model had much higher performance on identifying the samples that were false-positives and correctly identified 1130 of the 1214 samples. This resulted in an overall accuracy of 0.92, precision of 0.79, recall of 0.87, and AUC of 0.97 on the test set. This higher performing model gives promise for using LSTM networks for exoplanet detection.

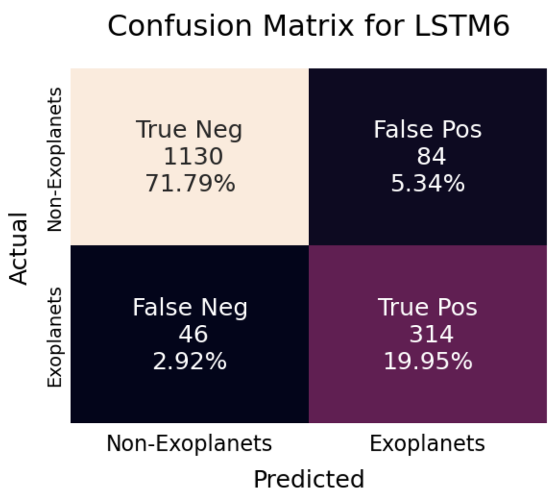
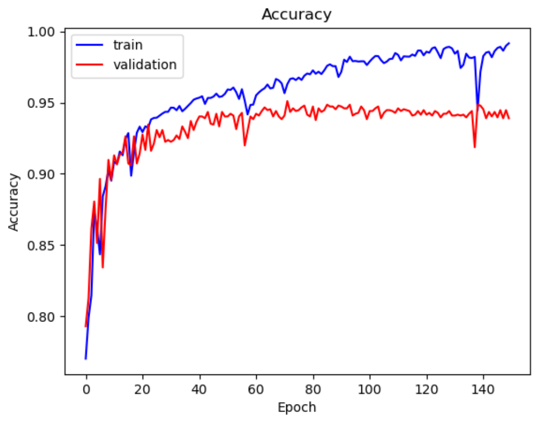


Figure 4-4: Confusion Matrix for LSTM_6

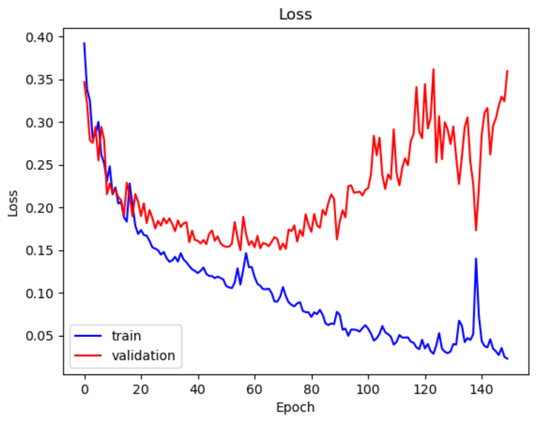
4.3 CNN-LSTM

Because early LSTM models had a difficult time achieving over 80% accuracy on the test set, CNN-LSTM models were explored. CNN-LSTM$_1$ had a column for the local view that contained two Conv1D layers with 32 filters of size 3, a max-pooling layer with windows of size 2, and an LSTM layer with 128 units. The global view column contained two Conv1D layers with 32 filters of size 5, a max-pooling layer with windows of size 2, two Conv1D layers with 64 filters of size 5, another max-pooling layer, and an LSTM layer with 256 units. The output of these columns was concatenated, then put through a dense layer with 64 neurons, a dense layer with 32 neurons, and finally the output layer. This model was trained for 150 epochs with a batch size of 128 and an initial learning rate of 0.003.

Figure 4-5 shows the loss and accuracy during training for the train and validation sets. The model performed very well on the training set and initially on the validation set, but the validation set accuracy slightly declined over time. It is evident in the second plot that the model was severely overfitting. The validation loss initially followed the training loss, but sharply increased for the last half of training. Clearly, this model needed to become simpler or be regularized.



(a) Accuracy of CNN-LSTM$_1$



(b) Loss of CNN-LSTM$_1$

Figure 4-5: Accuracy and loss of CNN-LSTM$_1$

Figure 4-6 shows the confusion matrix for CNN-LSTM$_1$. It correctly identified 326 of 360 samples with exoplanet transits, an increase from the previous LSTM models. CNN-LSTM$_1$ correctly classified 1177 of 1214 samples without exoplanets. The model's overall accuracy is 0.95, precision 0.90, recall 0.91, and AUC 0.98. This is a major improvement over the models without convolutional columns.

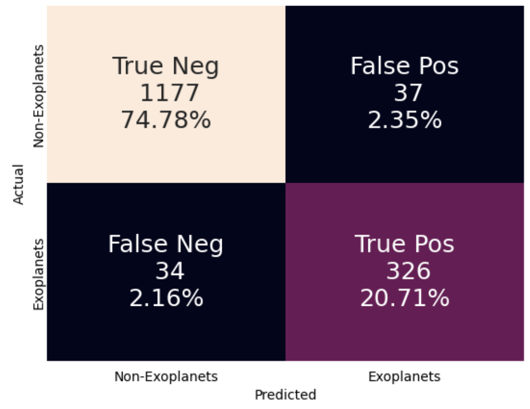
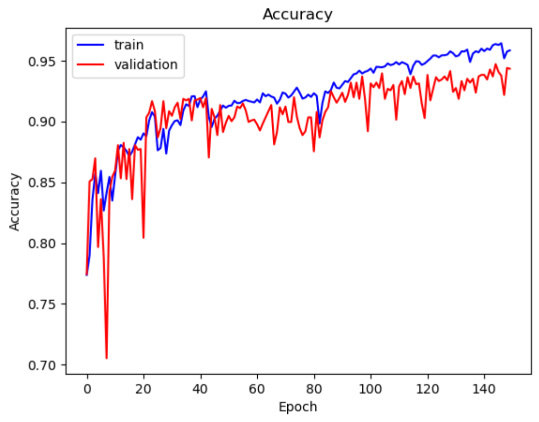
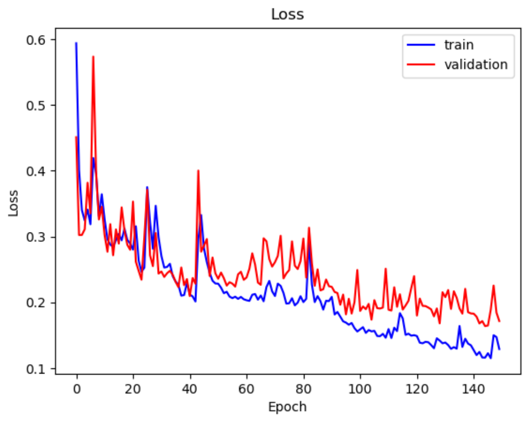


Figure 4-6: Confusion Matrix for CNN-LSTM$_1$

After adding L_2 regularization with a default parameter value of 0.001 to the LSTM layers, CNN-LSTM$_1$ had a decrease in precision but an increase in recall. Accuracy and AUC remained the same. As can be seen in Figure 4-7, the overfitting problem was generally taken care of. However, the model did have a decrease in accuracy and an increase in loss on the train set. The L_2 regularization seems to affect learning and stunt improvement. Future applications of regularization decreased the parameter to values closer to 0.0001, rather than used the default. Figure 4-8 shows that the model performed better on detecting exoplanets, but classified more negative samples as positive than before.



(a) Accuracy of CNN-LSTM$_1$ with L_2 Regularization to LSTM Kernels



(b) Loss of CNN-LSTM$_1$ with L_2 Regularization to LSTM Kernels

Figure 4-7: Accuracy and loss of CNN-LSTM$_1$ with L_2 Regularization

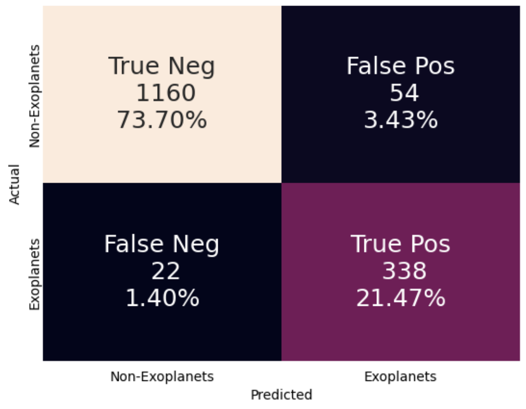
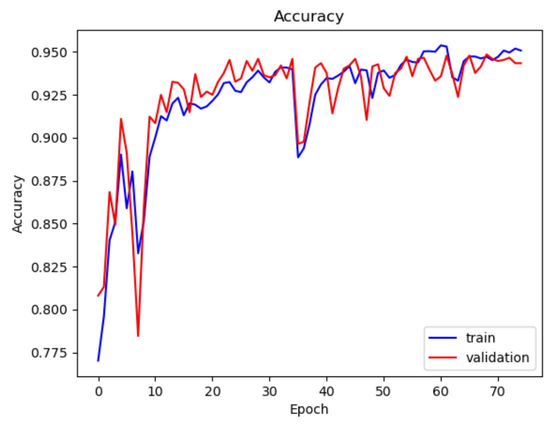
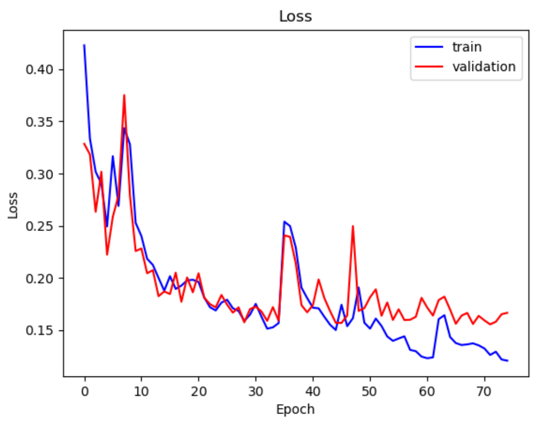


Figure 4-8: Confusion Matrix for CNN-LSTM$_1$ with L_2 Regularization

CNN-LSTM$_3$ had almost the same architecture as CNN-LSTM$_1$, but had fewer units in the LSTM layer of the global input (128 instead of 256) and introduced a dropout of 0.2 after the first dense layer. This network did not have L_2 regularization. CNN-LSTM$_3$ had about the same performance as CNN-LSTM$_1$ with L_2 regularization, which suggests that either simplifying the model or L_2 regularization provide a similar boost to performance and decrease in overfitting. Figures 4-9 and 4-10 provide the details of the accuracy, loss, and classification of CNN-LSTM$_3$. Generally, simplifying LSTM layers led to increased accuracy and precision, but decreased recall. If dense layers were too complex, there was a decrease in precision, but higher recall. Also, these example models show that weight regularization is generally unnecessary; simplifying a model leads to better results and training convergence.



(a) Accuracy of CNN-LSTM$_3$



(b) Loss of CNN-LSTM$_3$

Figure 4-9: Accuracy and loss of CNN-LSTM$_3$

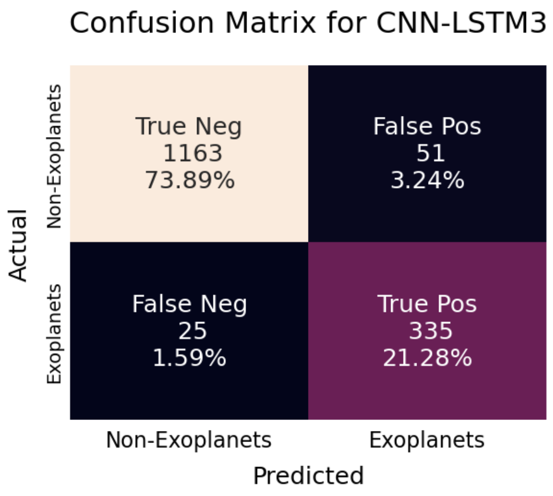
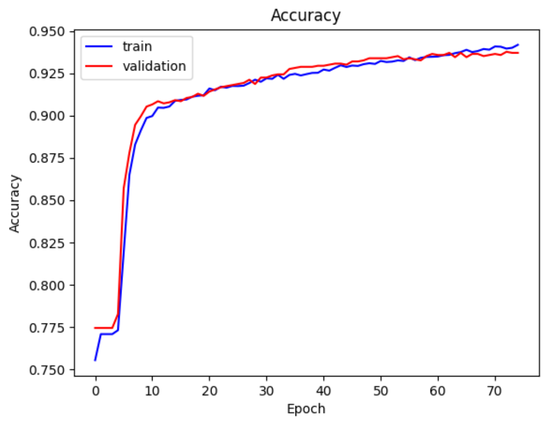


Figure 4-10: Confusion Matrix for CNN-LSTM$_3$

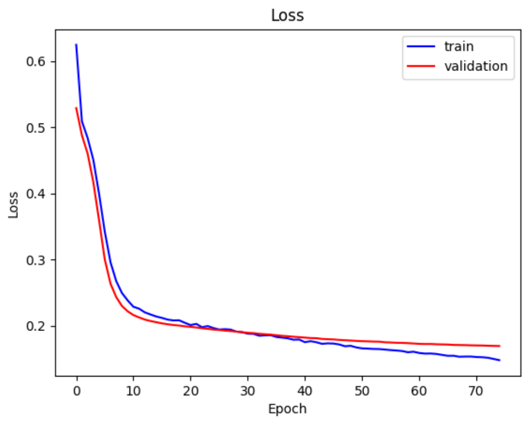
4.4 CNN

In addition to testing other model architectures for exoplanet detection, I was interested in experimenting with a smaller CNN and determining if Astronet's size was necessary.

CNN$_1$ is the first CNN that was tested. The local view was fed into two sequential Conv1D layers with 16 filters of size 5, a max-pooling layer with window size 5, two sequential Conv1D layers with 32 filters of size 5, and finally another max-pooling layer with the same size window. The global view was the same with one additional convolutional block containing two Conv1D layers with 64 filters of size 5. The outputs of these convolutional columns were passed into a dense layer with 64 neurons and a dropout of 0.4, a dense layer with 32 neurons, and finally an output layer. CNN$_1$ was trained for 75 epochs with a batch size of 128 and an initial learning rate of 0.006.



(a) Accuracy of CNN$_1$



(b) Loss of CNN$_1$

Figure 4-11: Accuracy and loss of CNN$_1$

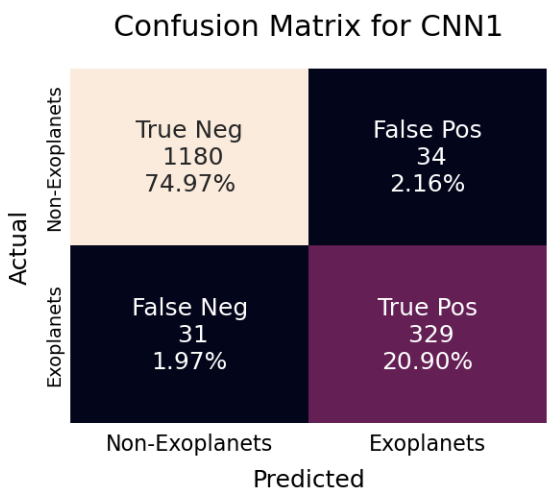
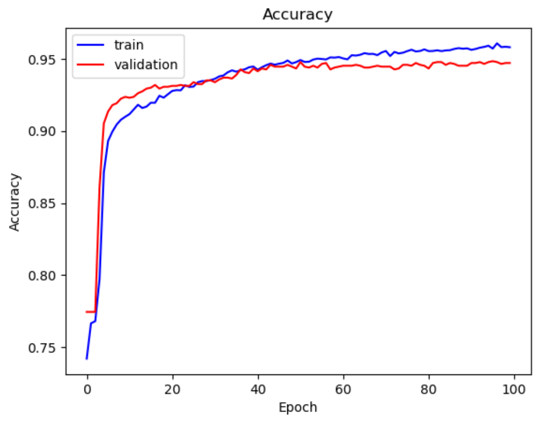


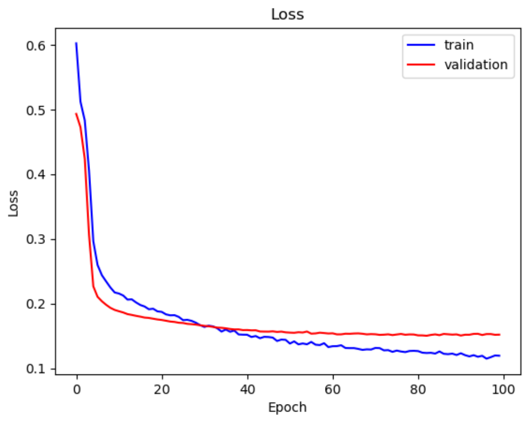
Figure 4-12: Confusion Matrix for CNN$_1$

Ansdell, et al. remarked that Astronet suffered from overfitting[6], which is also shown in Figure 4-23 of the accuracy and loss of training my implementation of Astronet in Keras. Figure 4-11 shows that this scaled-down CNN does not suffer from much overfitting; in fact, the training and validation accuracy and loss are very close for the entire length of training. Figure 4-12 shows that CNN_1 correctly identified 329 of 360 samples with exoplanets and 1180 of 1214 of the negative class. The overall accuracy of the model was 0.96, which rivals Astronet. The precision, 0.91, is slightly better than Astronet's 0.90, but the recall of 0.91 is slightly lower than Astronet's 0.95. Both models had an AUC score of 0.99. The performance of this model is promising, as CNN_1 has 1 million trainable parameters compared to Astronet's 8.8 million and CNN_1 has competitive performance.

CNN_{12} was very similar to CNN_1, except that there were additional dense layers. This model had the same convolutional columns. The outputs of these were concatenated and put through a series of two dense layers with 64 neurons and 0.2 dropout and one layer with 32 neurons and 0.2 dropout before the output layer. Figure 4-13 depicts the accuracy and loss plots for the training and validation sets. Figure 4-14 shows the confusion matrix for this network. Accuracy, precision, and AUC remained the same, but recall increased to 0.93.



(a) Accuracy of CNN_{12}



(b) Loss of CNN_{12}

Figure 4-13: Accuracy and loss of CNN_{12}

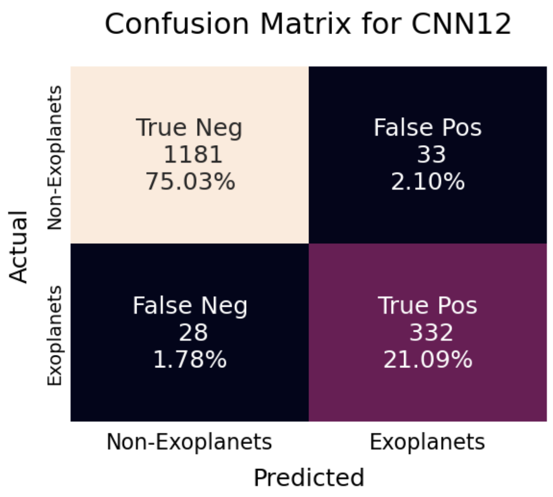
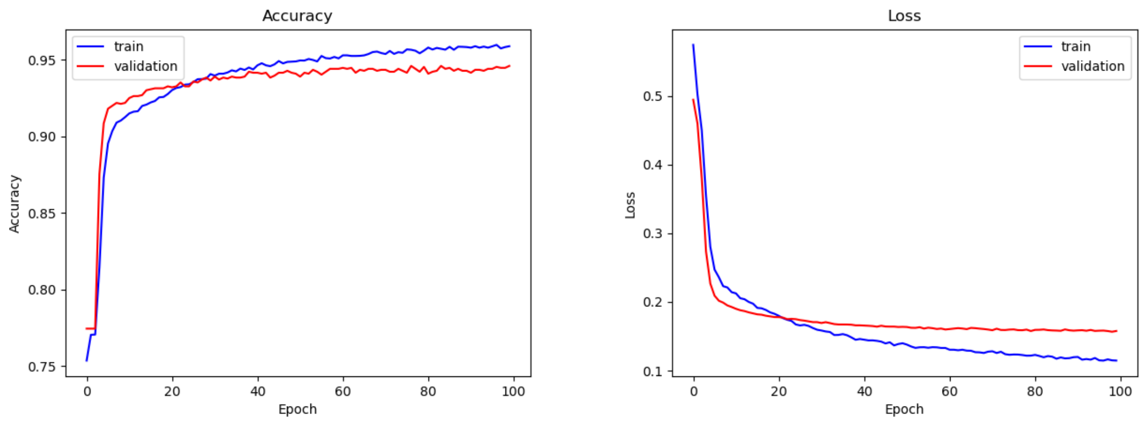


Figure 4-14: Confusion Matrix for CNN_{12}

CNN_{13} is almost the same as CNN_{12}, except that the final dense layer has 64 neurons instead of 32. Figure 4-15 depicts the accuracy and loss plots for training CNN_{13} and Figure 4-16 shows the confusion matrix on the test set. Accuracy, precision, and AUC remained the same, but recall increased to 0.93. Thus far, this is the best performing scaled down CNN.



(a) Accuracy of CNN_{13} (b) Loss of CNN_{13}

Figure 4-15: Accuracy and loss of CNN_{13}

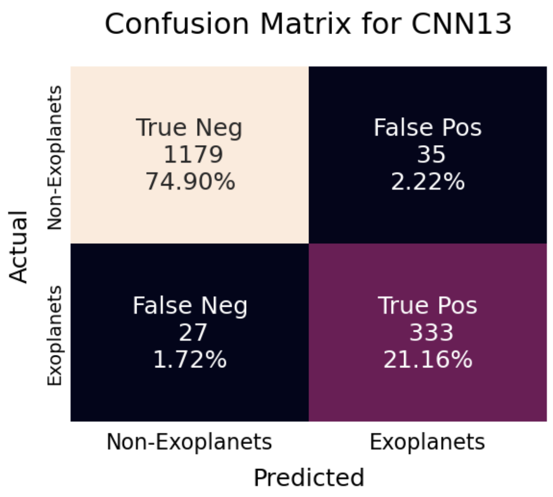


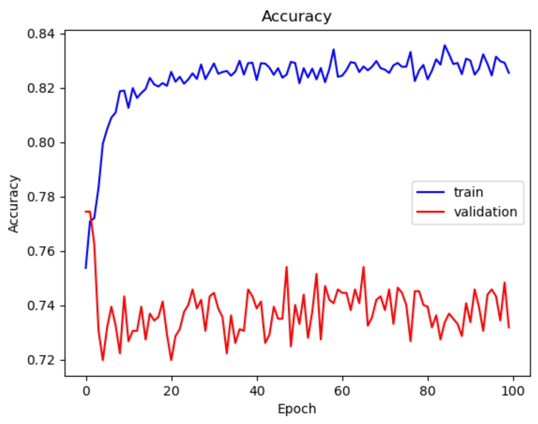
Figure 4-16: Confusion Matrix for CNN_{13}

Zero-padding input for all models always increased all metrics. Additionally, increasing the size of convolutional filters or pooling windows from 5 to 7 decreased precision, but increased recall. As noted for CNN-LSTMs, increasing the number or size of dense layers increased recall, but decreased precision.

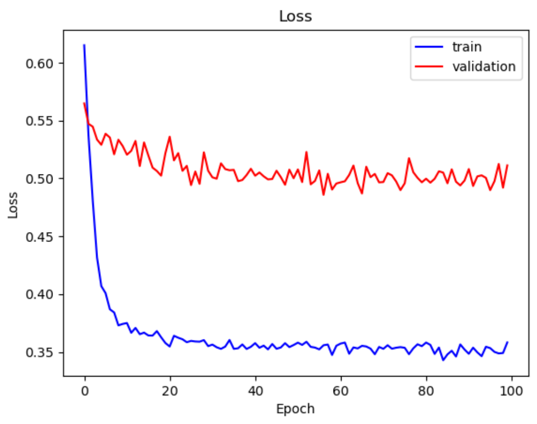
4.5 Models with Attention

4.5.1 Transformer

ATTN_1 was the first tested Transformer model, which has just a single MultiHeadAttention layer with 2 heads and L_2 regularization of e-3. There were no following dense layers. Deeper models were unable to train due to resource constrains. Figure 4-17 shows the accuracy and loss plots; it is clear that this model is extremely overfit. As shown in the confusion matrix in Figure 4-18, this model incorrectly classified about one third of each class in the test set. This led to a test set accuracy of 0.68. This model clearly did not learn features that separated the classes.



(a) Accuracy of ATTN_1



(b) Loss of ATTN_1

Figure 4-17: Accuracy and loss of ATTN_1

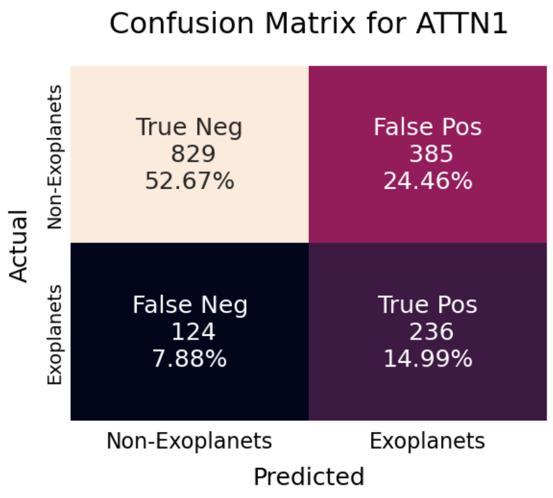
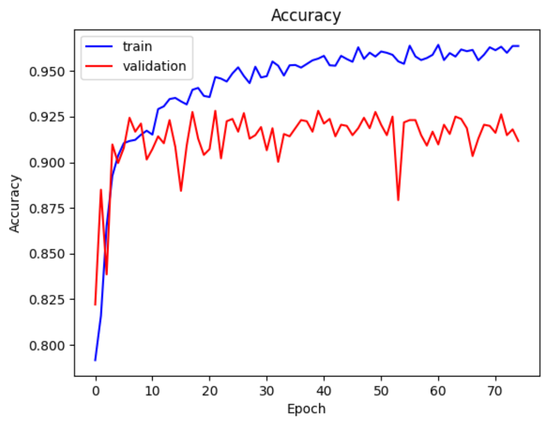
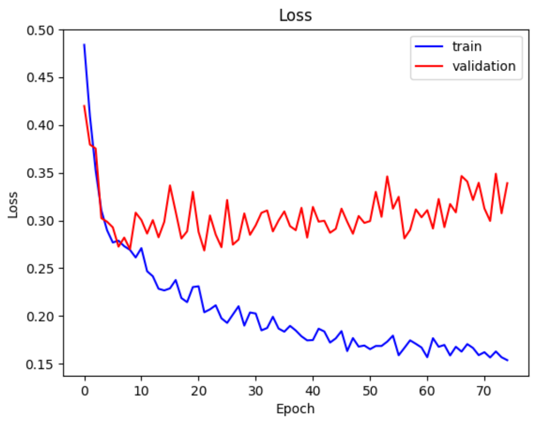


Figure 4-18: Confusion Matrix for ATTN_1

The ATTN2_N network had separate transformer layers for the separate inputs, which were then concatenated and fed through a couple of dense layers. Specifically, ATTN2_1 had transformer layers with 2 heads and key dimension of 2, each with L_2 regularization with a parameter of e-3. The outputs of these transformers were concatenated, put through a dense layer with 64 neurons and a dropout of 0.4, another dense layer with 64 neurons, a dense layer with 32 neurons, and finally the output layer. As can be seen in Figure 4-19, this network was overfit. The accuracy on the test set was 0.77, precision 0.25, recall 0.01, and AUC 0.44. Figure 4-20 shows that the network did not learn exoplanet transit patterns. It predicted that nearly all samples did not contain exoplanets.



(a) Accuracy of ATTN2_1



(b) Loss of ATTN2_1

Figure 4-19: Accuracy and loss of ATTN2_1

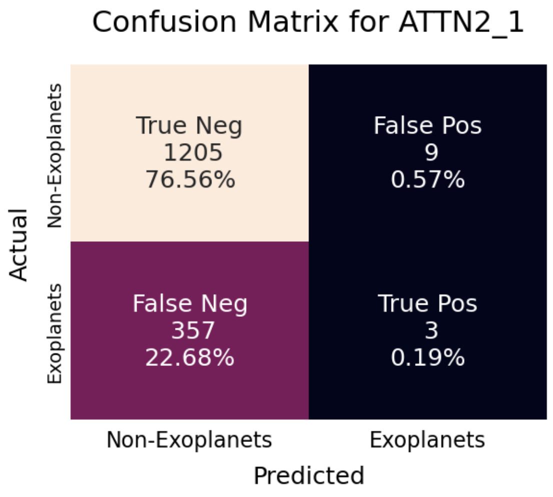


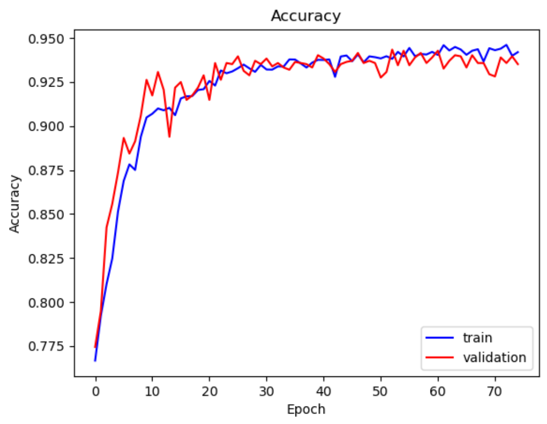
Figure 4-20: Confusion Matrix for ATTN2_1

Very few transformer models were produced and tested due to the time required to train each model. ATTN2_1 took about a week to train. Experiments with hyperopt on the ATTN2 architecture took a few weeks to complete about 15% of trials. More time was dedicated to the primary architectures (LSTM, CNN-LSTM, and CNN) due to computation cost and relatively low accuracy, precision, and recall from transformers.

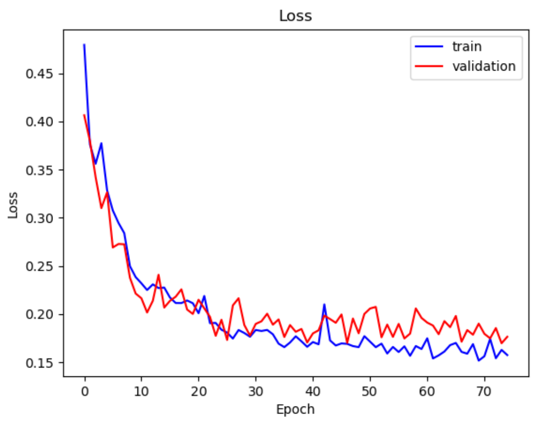
4.5.2 Other Models with Attention

Since the training time was significantly increased, not many LSTM models with attention were tested. All LSTM models with added attention were either extremely underfit or did not significantly increase performance on the test set. Due to marginal or poor performance and the increased training time, LSTM models with attention were abandoned in favor of high-performing and simpler LSTM models.

Attentional interfaces in the CNN-LSTM architecture also had marginal performance or did not significantly increase performance relative to the increased training time. One example, CNN-LSTM-ATTN2_3, had an Attention layer that took the output of each separate convolutional column (the global view column had two Conv1D layers with 32 filters of size 5, a MaxPooling1D layer with filters of size 2, two Conv1D layers with 64 filters of size 5, and a final MaxPooling1D layer with filters of size 2; the local view had just the first three layers of the global column) as input and produced an attention distribution that was fed into a single LSTM layer with 128 neurons and L_2 regularization of 0.0001. Following the LSTM layer was a dense layer with 64 neurons and a dropout of 0.4, a dense layer with 32 neurons, and the output layer. Figure 4-21 depicts the accuracy and loss during training on the training and validation sets.



(a) Accuracy of CNN-LSTM-ATTN2_3



(b) Loss of CNN-LSTM-ATTN2_3

Figure 4-21: Accuracy and loss of CNN-LSTM-ATTN2_3

As can be seen in Figure 4-22, CNN-LSTM-ATTN2_3 correctly predicted 321 of 360 samples with exoplanets and 1166 of 1214 samples without exoplanets. This model achieved an accuracy of 0.945, precision of 0.87, recall of 0.89, and AUC of 0.98. While this was the best performer of the CNN-LSTM models with attention, it did not perform as well as the best performing CNN-LSTMs described in section 4.3.

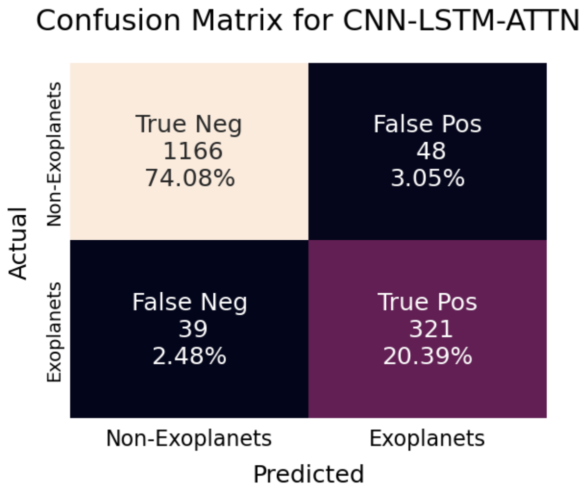
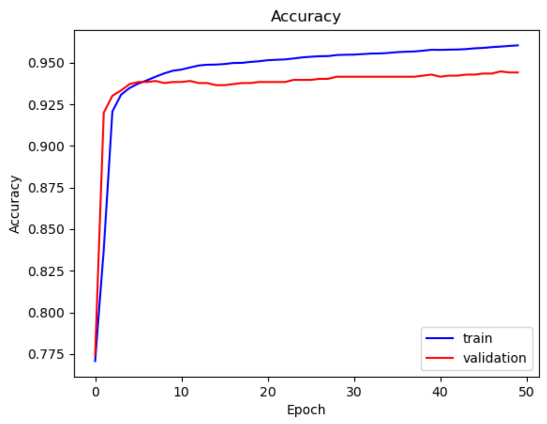


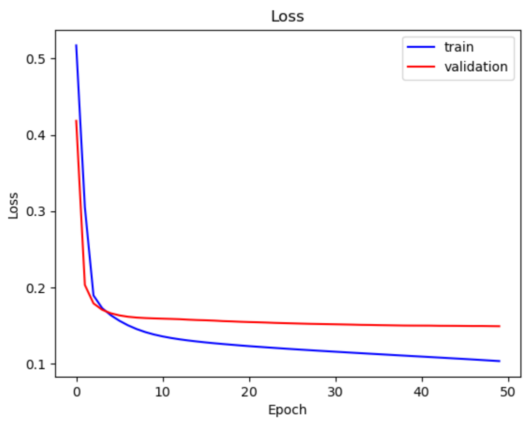
Figure 4-22: Confusion Matrix for CNN-LSTM-ATTN$_1$

4.6 Comparison of Best Performers to Astronet

Shallue and Vanderburg[40] reported that their best performing Astronet model achieved an accuracy of 0.96, precision of 0.90, recall of 0.95, and AUC of 0.99 on the test set. When Ansdell, et al.[6] recreated Astronet in PyTorch, they reported similar performance with test accuracy at 0.955. I recreated Astronet in Keras with the same model architecture as in Figure 2-2. As reported by Shallue and Vanderburg, no dropouts were used and the model was trained with the Adam optimizer with $\alpha = 10^{-5}$, default β_1 and β_2, and $\epsilon = 10^{-8}$ for 50 epochs with a batch size of 64. Figure 4-23 depicts the accuracy and loss during training and that the model is overfitted as noted by Ansdell, et al. Figure 4-24 is the confusion matrix for the Keras implementation of Astronet, which achieved 0.96 accuracy, 0.89 precision, 0.96 recall, and 0.99 AUC. The metrics of the two implementations are compared in Table 4.1.



(a) Accuracy of Astronet



(b) Loss of Astronet

Figure 4-23: Accuracy and loss of Astronet

Table 4.1: Reported Astronet vs Keras Implementation

Network	Accuracy	Precision	Recall	AUC
Astronet	0.96	0.90	0.95	0.99
Astronet$_K$	0.96	0.89	0.96	0.99

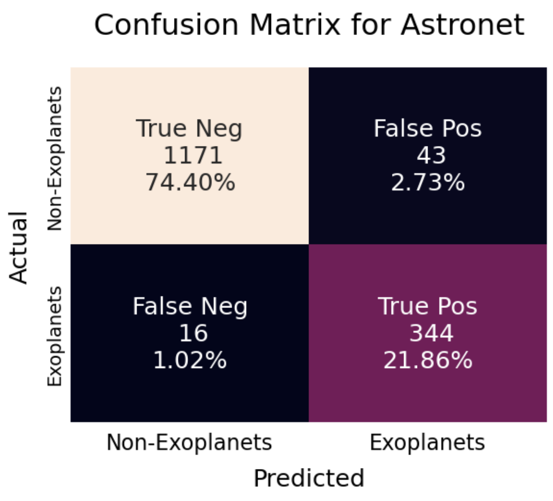


Figure 4-24: Confusion Matrix for Astronet

In Table 4.2 and Figure 4-25 I compare the performance of the top performer in the top three architectures (LSTM, CNN-LSTM, and CNN) with the performance of Astronet.

Table 4.2: Best Performers vs Astronet

Network	Accuracy	Precision	Recall	AUC	Parameters
Astronet	0.96	0.89	0.96	0.99	8.8M
CNN_{13}	0.96	0.91	0.93	0.99	1.2M
CNN-LSTM_1	0.95	0.90	0.91	0.98	124K
LSTM_6	0.92	0.79	0.87	0.97	143K

The best LSTM clearly did not perform as well as the other architectures; it is not worth spending time fine tuning an LSTM for exoplanet detection. CNN-LSTMs show promise with CNN-LSTM_1 close in performance to Astronet, which was just 1% behind in test set accuracy. This model has about 59 times fewer trainable parameters than Astronet. CNN_{13}, a scaled-down version of Astronet, has 8 times fewer trainable parameters. This model has comparable performance to Astronet with increased precision. The increased precision is very important, as the amount of false positives should be as low as possible so time and resources can be spent examining the best planet candidates. Clearly, the CNN-LSTMs and scaled down CNNs are competitive with Astronet.

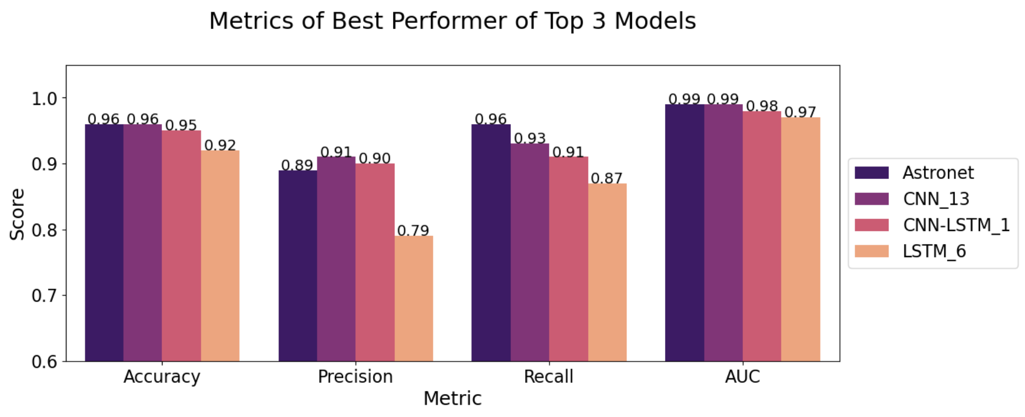


Figure 4-25: Metrics of Top 3 Performers vs Astronet

Chapter 5

Conclusion and Future Work

In this work, I explored two questions based on the work of Shallue and Vanderburg[40] and the necessity of automated methods for identifying potential exoplanet candidates in Kepler data. Firstly, would it be beneficial to use different neural network architectures, specifically those designed for sequential data such as recurrent networks, as an alternative to the current best performing convolutional neural networks for exoplanet detection? In other words, can these other network architectures achieve comparable or better performance than Astronet? Secondly, is it possible to achieve comparable or better performance than Astronet with a much smaller convolutional neural network that had far fewer trainable parameters? To test these alternative architectures against Astronet, I used the same dataset as was used to train Astronet and developed models to perform the same classification to determine if a light curve sample contained a transiting exoplanet.

To address the first question, I experimented with LSTMs, CNN-LSTMs, Transformers, and attentional interfaces with the LSTMs and CNN-LSTMs. Initially, the LSTM model had poor performance compared to Astronet. Further experimentation and hyperparameter tuning led to a model with about 0.92 accuracy; however, this was still not enough to be considered a success. It is most likely the case that, although the data was a time series, the temporal relationship was not as strong of an indicator of the existence of an exoplanet as the overall shape, or spatial relationship, of the data points. The primary reason for this may be the way the data was processed. Each light curve covered a long period with many periodic signals throughout the series. Shallue and Vanderburg processed the original light curve by folding the curve at specific intervals onto itself so that constructive interference would amplify any existing periodic signal and destructive interference would remove noise in the signal. They chose intervals for folding that resulted in one primary dip to exist in the data. There is no longer a periodic signal in the dataset. Instead, each data sample was focused on the shape of the strongest signal rather than temporal relationships. This is likely why the sequential-centric LSTMs failed. If they worked with the original light curve series, they likely would have been able to detect the periodic signal, the depth, and the period between dips. Another issue with the LSTMs may be the input sequence length. The global view had 2000 time steps, which is too long for reliable training. Simply trimming the global views may help with the LSTM performance. Future work will test the LSTMs on unfolded light curves and with the same dataset with truncated global views.

Adding convolutional columns to the LSTM network made a large improvement on the performance immediately, particularly in terms of precision. This supports the idea that the folded light curves emphasize spatial features in the dip. CNN-LSTM models always performed quite well in comparison to Astronet but never achieved better performance. However, they are still competitive. CNN-LSTM$_1$ achieved 0.95 accuracy compared to Astronet's 0.96 and contains 59 times fewer trainable parameters than Astronet. This model is significantly more efficient and very competitive, which is valuable for researchers with resource constraints.

Transformers were designed specifically for language tasks and, though they have been applied to numerical time series data, they are unreliable and research is limited in that area. Transformers operate on the entire input sequence at once, performing dot product operations for each input using every other input. This allows Transformers to learn over long-term dependencies without memory loss; however, this requires a lot of memory $((n^2))$ and computation. Transformers require a lot of resources and models in this research took far too long to train; the initial model took over a week and hyperparameter optimization experiments took a few weeks to complete about 15% of the experiments. Even training on a GPU in UWM's computing resources took several days to complete. Thus, Transformers were abandoned due to lack of time and resources and the relatively low performance. Attentional interfaces with the LSTMs and CNN-LSTMs had unreliable performance; sometimes, particularly with the LSTM models, there was a significant amount of underfitting. In a few cases with the CNN-LSTM, attention helped boost performance, but only marginally. Besides attention mechanisms not working well on the input data, perhaps some of the Transformer and attentional models were unnecessarily complicated. These models are not useful for exoplanet detection.

To address my second question, I developed a scaled-down version of Astronet. The fully convolutional neural network models contained separate convolutional columns for each of the input types, just like Astronet, then combined the output of these columns and fed this through a series of dense layers. These models were about 8 times smaller than Astronet, yet had competitive performance, which proves that Astronet is unnecessarily large and can be scaled down. In future work, the dataset will be expanded to include stellar parameters and centroid curves, which Ansdel, et al. proved to be essential for a model to discriminate between planets and eclipsing binary stars. Finely tuning a smaller CNN would greatly benefit the exoplanet research community. If a highly efficient and high performing model can be used by this community, it would make a big difference in the vetting of astrophysical false positives in the over 600 GB of data collected from Kepler and nearly 32 TB of data collected from TESS that is increasing by 30 GB every day. While exoplanet candidates must be manually certified by experts, the initial identification by an automated model will save invaluable time and resources.

Bibliography

[1] Cs231n convolutional neural networks for visual recognition. https://cs231n.github.io/convolutional-networks/.

[2] Understanding LSTM networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015.

[3] National Aeronautics and Space Administraion. The planet hunters. https://exoplanets.nasa.gov/discovery/missions/first-planetary-disk-observed.

[4] National Aeronautics and Space Administration. Kepler and k2. https://www.nasa.gov/mission_pages/kepler/main/index.html.

[5] National Aeronautics and Space Administration. Ways to find a planet. https://exoplanets.nasa.gov/alien-worlds/ways-to-find-a-planet/.

[6] Megan Ansdell, Yani Ioannou, Hugh P. Osborn, Michele Sasdelli, Jeffrey C. Smith, Douglas Caldwell, Jon M. Jenkins, Chedy Räissi, Daniel Angerhausen, and and. Scientific domain knowledge improves exoplanet transit classification with deep learning. *The Astrophysical Journal*, 869(1):L7, dec 2018.

[7] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

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

[9] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *25th annual conference on neural information processing systems (NIPS 2011)*, volume 24. Neural Information Processing Systems Foundation, 2011.

[10] James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International conference on machine learning*, pages 115–123. PMLR, 2013.

[11] William J Borucki, David Koch, Gibor Basri, Natalie Batalha, Timothy Brown, Douglas Caldwell, John Caldwell, Jørgen Christensen-Dalsgaard, William D Cochran,

Edna DeVore, et al. Kepler planet-detection mission: introduction and first results. *Science*, 327(5968):977–980, 2010.

[12] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. *arXiv preprint arXiv:1601.06733*, 2016.

[13] C. Darken, J. Chang, and J. Moody. Learning rate schedules for faster stochastic gradient search. In *Neural Networks for Signal Processing II Proceedings of the 1992 IEEE Workshop*, pages 3–12, 1992.

[14] Anne Dattilo, Andrew Vanderburg, Christopher J Shallue, Andrew W Mayo, Perry Berlind, Allyson Bieryla, Michael L Calkins, Gilbert A Esquerdo, Mark E Everett, Steve B Howell, et al. Identifying exoplanets with deep learning. ii. two new super-earths uncovered by a neural network in k2 data. *The Astronomical Journal*, 157(5):169, 2019.

[15] Ian Dewancker, Michael McCourt, and Scott Clark. Bayesian optimization primer.

[16] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[17] Jon Jenkins et al. Overview of the kepler science processing pipeline. 2009.

[18] Debra A. Fischer, Megan E. Schwamb, Kevin Schawinski, Chris Lintott, John Brewer, Matt Giguere, Stuart Lynn, Michael Parrish, Thibault Sartori, Robert Simpson, and et al. Planet hunters: the first two planet candidates identified by the public using the kepler public archive data. *Monthly Notices of the Royal Astronomical Society*, 419(4):2900–2911, Nov 2011.

[19] N. P. Gibson, S. Aigrain, S. Roberts, T. M. Evans, M. Osborne, and F. Pont. A Gaussian process framework for modelling instrumental systematics: application to transmission spectroscopy. *Monthly Notices of the Royal Astronomical Society*, 419(3):2683–2694, 01 2012.

[20] Daniel Giles and Lucianne Walkowicz. Systematic serendipity: a test of unsupervised machine learning as a method for anomaly detection. *Monthly Notices of the Royal Astronomical Society*, 484(1):834–849, 2019.

[21] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. JMLR Workshop and Conference Proceedings.

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

[23] Simon Haykin. *Neural networks and learning machines, 3/E.* Pearson Education India, 2010.

[24] Haibo He and Edwardo A Garcia. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, 21(9):1263–1284, 2009.

[25] Trisha A. Hinners, Kevin Tat, and Rachel Thorp. Machine learning techniques for stellar light curve classification. *The Astronomical Journal*, 156(1):7, jun 2018.

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

[27] Andrew W Howard, Geoffrey W Marcy, Stephen T Bryson, Jon M Jenkins, Jason F Rowe, Natalie M Batalha, William J Borucki, David G Koch, Edward W Dunham, Thomas N Gautier III, et al. Planet occurrence within 0.25 au of solar-type stars from kepler. *The Astrophysical Journal Supplement Series*, 201(2):15, 2012.

[28] Fazle Karim, Somshubra Majumdar, Houshang Darabi, and Shun Chen. Lstm fully convolutional networks for time series classification. *IEEE access*, 6:1662–1669, 2017.

[29] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[30] Will Koehrsen. A conceptual explanation of bayesian hyperparameter optimization for machine learning. https://towardsdatascience.com/a-conceptual-explanation-of-bayesian-model-based-hyperparameter-optimization-for-machine-learning-b8172278050f, 2018.

[31] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[32] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[33] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation, 2015.

[34] D Masters and C Luschi. Revisiting small batch training for deep neural networks. 2018.

[35] Chris Olah and Shan Carter. Attention and augmented recurrent neural networks. *Distill*, 2016.

[36] H. P. Osborn, M. Ansdell, Y. Ioannou, M. Sasdelli, D. Angerhausen, D. Caldwell, J. M. Jenkins, C. Räissi, and J. C. Smith. Rapid classification of tess planet candidates with convolutional neural networks. *Astronomy & Astrophysics*, 633:A53, Jan 2020.

[37] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[38] N Schanche, A Collier Cameron, G Hébrard, L Nielsen, A H M J Triaud, J M Almenara, K A Alsubai, D R Anderson, D J Armstrong, S C C Barros, and et al. Machine-learning approaches to exoplanet transit detection and candidate validation in wide-field ground-based surveys. *Monthly Notices of the Royal Astronomical Society*, 483(4):5534–5547, Nov 2018.

[39] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, Jan 2015.

[40] Christopher J. Shallue and Andrew Vanderburg. Identifying Exoplanets with Deep Learning: A Five-planet Resonant Chain around Kepler-80 and an Eighth Planet around Kepler-90. *Astronomical Journal*, 155(2):94, February 2018.

[41] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[42] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[43] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019.

[44] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[45] Qingsong Wen, Liang Sun, Xiaomin Song, Jingkun Gao, Xue Wang, and Huan Xu. Time series data augmentation for deep learning: A survey. *arXiv preprint arXiv:2002.12478*, 2020.

[46] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057. PMLR, 2015.

[47] Liang Yu, Andrew Vanderburg, Chelsea Huang, Christopher J Shallue, Ian JM Crossfield, B Scott Gaudi, Tansu Daylan, Anne Dattilo, David J Armstrong, George R Ricker, et al. Identifying exoplanets with deep learning. iii. automated triage and vetting of tess candidates. *The Astronomical Journal*, 158(1):25, 2019.

[48] Shay Zucker and Raja Giryes. Shallow transits—deep learning. i. feasibility study of deep learning to detect periodic transits of exoplanets. *The Astronomical Journal*, 155(4):147, 2018.