Deep Learning for In-Real-Time Stellar Transient Detection

A dissertation submitted in partial satisfaction of the requirements for the degree

Bachelors of Science in Physics

by

Natalie L. LeBaron

Committee in charge:

Professor Philip Lubin, Chair

June 2022
The Dissertation of Natalie L. LeBaron is approved.

Professor Philip Lubin, Committee Chair

June 2022
Deep Learning for In-Real-Time Stellar Transient Detection

Copyright © 2022

by

Natalie L. LeBaron
To all of the undergraduates who worked with me.

And to my parents.
Acknowledgements

I want to start by thanking Professor Philip Lubin and the members of the UCSB Experimental Cosmology Group for giving me the chance to begin research as an undergraduate freshman and providing me a welcoming space to grow and learn as a beginning astrophysicist.

I next want to acknowledge my SDI and II in the Sky team members, past and present, within the UCSB Experimental Cosmology group. I feel privileged that I was able to lead our group for as long as I did and I am so thankful for all the hardworking individuals that put in a ton of effort toward our goals and taught me how to be a scientist and project leader. I want to specifically thank Alex Polanski, Link Morgan, Varun Iyer, Prerana Kottapalli, Michael McDonald, Alex Thomas, Aidan Hogge, Kelvin Yip, Luca Angeleri, and Enda Mao for each of their contributions and guidance throughout my four years on the project.

Furthermore, I could not have done this project without the specific contributions of Benjamin Fogiel, Rachel Darlinger, and Cole Slenkovich. Thank you so much for your help and hard work.

Last and most importantly, I want to thank Kellan Colburn for helping me develop this idea and see it come to fruition. You have been my support and anchor throughout this entire endeavor, and I greatly appreciate all the advice, knowledge, and experience you have given me.

Funding for this project comes from the UCSB Faculty Research Assistance Program, NASA grants: NIAC Phase I DEEP-IN – 2015 NNX15AL91G and NASA NIAC Phase II DEIS – 2016 NNX16AL32G, the NASA California Space Grant (NASA NNX10AT93H), as well as a generous gift from the Emmett and Gladys W. fund. Additionally, this work makes use of observations from the Las Cumbres Observatory global telescope network.
Abstract

Deep Learning for In-Real-Time Stellar Transient Detection

by

Natalie L. LeBaron

In recent years, a number of large-scale sky observing facilities and surveys have been established which are bringing us into an era of unprecedented astrophysical data abundance and potential for finding transient phenomena. However, the speed of data collection generally outstrips the ability of a survey to process and analyze the collected images. Thus, we present a nominally accurate and fast first-look, real-time source extraction method for finding astronomical transients. This method is neural network based, specifically using YOLO Darknet [1], a pre-trained convolutional neural network that we optimized for transient source extraction from subtracted image residuals created from starfields simulated with Skymaker [2]. With this algorithm, we obtain a source extraction method that, once trained, operates within a few tens of milliseconds per image and identifies transient sources ≥16th magnitude with ≥99% accuracy and ≥90% confidence. Even dimmer sources (between 16-18.5 magnitude) were found with ~≥ 75% accuracy. This algorithm represents a promising start to a new regime of fast transient detection and multimessenger follow-up observation.
# Contents

Abstract vi

1 Introduction 1
   1.1 Large Scale Surveys ................................................. 1
   1.2 Classical Methods of Transient Detection: Difference Image Analysis .... 3
   1.3 Machine Learning from Industry to Astrophysics .......................... 3

2 Deep Learning and Overview of Neural Networks 8
   2.1 The Basics of Machine Learning and Computer Vision ..................... 8
   2.2 Neural Networks .................................................................. 11
   2.3 Convolutional Neural Networks .............................................. 20
   2.4 YOLO Darknet .................................................................... 21

3 Machine Learning in Astrophysics 25
   3.1 The Data Problem ............................................................ 25
   3.2 Transfer Learning ................................................................ 26
   3.3 Transient Simulations ......................................................... 29

4 Real-Time Neural Network Based Source Extraction 31
   4.1 Overview .......................................................................... 31
   4.2 Results ............................................................................. 33
   4.3 II in the Sky ...................................................................... 36

5 Conclusions and Going Forward 40
   5.1 Overview .......................................................................... 40
   5.2 Collection of and Testing on Real Data ..................................... 41
   5.3 Multimessenger Imagining .................................................... 41
   5.4 Conclusion ......................................................................... 42

Bibliography 44
Chapter 1

Introduction

1.1 Large Scale Surveys

In recent years, a number of large-scale sky observing facilities and surveys have been established which aim to stare longer and probe ever deeper into the universe and its contents. One such project is the Vera Rubin Observatory’s *Legacy Survey of Space and Time* (LSST, Figure 1.1, [3]). This deep-field survey’s 3.2 gigapixel detector will image 15 terabytes of data a night. Thus, over its 10 year lifetime, tens of thousands of terabytes of image data will be captured, covering the entire southern sky out to a 25 magnitude depth, which includes 37 billion stars and galaxies. With a number of other contemporary surveys achieving similar levels of data collection, we are entering an era of unprecedented astrophysical data abundance and potential for finding transient phenomena in the night sky. However, the speed of data collection generally outstrips the ability of a survey to process and analyze the collected images, especially considering the amount of data being observed by tens of concurrent projects and the amount of time it takes per image to process with anything but the best supercomputers available today.
This overload of data generally gets distributed over many different groups of researchers and computing setups or is put into a database for later processing. This strategy works well for phenomena that is not time sensitive, but there are some events which require immediate detection and classification so that study can begin right upon first detection. Generally, these are types of transient phenomena like supernovae, fast radio bursts, nearer objects like asteroids, and other sources that change location or brightness on human or faster timescales in the sky. Therefore, a quick, accurate, and multi-faceted method of processing the amount of data generated by these endeavours is required in order to produce useful results and catalog important phenomena as they are observed in real-time.
1.2 Classical Methods of Transient Detection: Difference Image Analysis

Historically, difference image analysis pipelines have been used to find transient phenomena. Difference image analysis starts with a set of pre-processed images (i.e. bias corrected and flat fielded) and combining them together into a median averaged template image. This template is then subtracted from each individual images in the set. This procedure removes all common (i.e. static) features from each image and leaves only the transient phenomena in the residual image. From there, the remaining sources can be readily located by a source extraction algorithm and then cataloged by their Right Ascension, Declination, and flux value so that the evolution of the individual sources can be tracked over time.

Such pipelines work well; however, the time it takes them to run is generally on the order of tens of seconds even when run on high performance computers meant for astrophysics analysis. From there, any sources found need to be identified based on their light curve, taking additional processing time. Even with a supercomputer that is able to process 350 MB of fits data per second, this number of steps is taxing as each operation will drastically increase the computation time. Only the most powerful supercomputers available today have a chance of keeping up with the mass of data from these surveys using traditional methods of processing. Therefore, novel methods are required to make this problem approachable and scalable to more limited hardware.

1.3 Machine Learning from Industry to Astrophysics

Recently, there have been many advancements in industry and academia in the field of machine learning based object detection, particularly for fields like self-driving cars,
The original image (a) was median combined with a set of 100 images of the same piece of the sky to create the template image. The template was then subtracted from (a) via the Bramich method of image subtraction to create the residual image.
facial identification, medical diagnosis, and other applications that rely on large scale pattern recognition. Following these innovations, astrophysicists have taken note and, in the past several years, have begun to use machine learning based methods to process large data sets, locate features indicative of interesting physics, and classify things like galaxies, supernovae, and objects moving across the night sky. For example, Duev et al. (2019) [6], Shallue and Vanderburg (2018) [7], Thomas and Kahn (2018) [8], Zhang and Bloom (2021) [9], Jamal and Bloom (2020) [10], Blickhan et al. (2018) [11].

We can further learn from industry innovation by studying fast paced machine learning applications like autonomous driving. Such industries require an algorithm which can identify, locate, and classify features such as road lines, pedestrians, and other vehicles quickly and accurately so that the desired classification can be outputted in a reasonable time frame for any follow-up actions to be taken. Such principles and algorithms can be directly applied to detecting and identifying locations and types of transient events from astronomical data. Further, this analysis can be done in real-time since such algorithms operate on the order of milliseconds. In this way, these algorithms can replace the source extraction and identification steps in a data processing pipeline leaving image subtraction as the only time-consuming step. Thereby contributing toward a pipeline that can analyze an image, detect and classify all interesting features, and determine follow-up procedures all within the span of a single integration time.

Moreover, by running several machine learning algorithms together, multiple of these classification and localization operations can be parallelized on several types of data input. Industry uses this ability to increase detection and classification accuracy by taking advantage of the cross-correlation between data types as seen in Figure 1.3. This has an immediate use case in multimessenger astrophysics which seeks to cross-reference phenomena across all types of data available to astrophysicists, including all electromagnetic frequencies as well as gravitational wave and neutrino data. With a transient detection
Figure 1.3: In the color and thermal images, the red bounding boxes represent false positive detections of pedestrians, green bounding boxes represent false negative detections, and yellow bounding boxes represent true positive detections. The yellow bounding boxes in the fusion images contain the resultant detections after cross-correlating the detections in the separate data types. Evidently, the algorithm is much more accurate when taking into account multiple data sources. [12]

A machine learning algorithm that operates similar to that in Figure 1.3, sources found in more than one wavelength can be compared and correlated, simultaneously collected spectral data can be included in rapid analysis, photometry can be performed simultaneously, light curves can be generated, and more, all within that same millisecond time scale. Thus, with these applications in mind, we explore an industry level algorithm’s
performance for real-time transient detection.
Chapter 2

Deep Learning and Overview of Neural Networks

2.1 The Basics of Machine Learning and Computer Vision

Over the last few decades, machine learning has arisen as a subfield of artificial intelligence that brings together concepts from computer science, data science, statistics, mathematics, and neuroscience to create algorithms capable of heuristically modeling and solving problems. This allows for previously untractable problems to be modeled and thus has become an important component of modern data science since if given sufficient data to learn from, a learning algorithm can be trained to make accurate classifications and predictions. Further, as the heuristic nature of these algorithms often leads to a dramatic reduction in the number of computations required for a trained network to model a problem, such algorithms are often incredibly lightweight when compared to conventional algorithms. In these ways, machine learning is a valuable tool for analyzing
vast amounts of data as once an algorithm is trained to pick up on the relevant patterns it is trained to recognize, it can run through gigabytes of data within seconds, outputting accurate predictions in a fraction of the time of more conventional algorithms.

Machine learning can be applied to many types of data, but here the focus is on using machine learning methods to process image data and recognize relevant features within those images. This is the problem of computer vision. Specifically, we want to replicate a living organism’s ability to perceive, comprehend, recognize, and then localize objects given an input picture (see Figure 2.1). This problem is quite easy for living creatures since eyes and brains have evolved to be able to complete this task quite efficiently for a creature’s survival. However, for a computer, this task is more difficult as it cannot be easily quantified via a mathematical or logical model capable of generalizing to a wide range of categories.

Figure 2.1: The goal is to create a computer vision algorithm capable of looking at an image and returning an overall classification and then localization (i.e. drawing a bounding box) of the classified object(s) in the image. From there, if more detail is needed in the localization, semantic image segmentation of each individual object can further differentiate the various features that the algorithm identifies. [13]

The goal is to get a computer algorithm to be able to take an input image, perhaps of a cat, and be able to categorize the image as containing a cat, and no other creature or object. However, cats come in a variety of shapes and sizes, as do other types of objects. Further, a single image may have multiple identifiable objects within it. Thus,
a computer must be able to learn what it is that makes a cat a cat and not a horse or a
dog so that the algorithm can correctly classify and localize identifying features. We can
create such an algorithm in several different ways.

The main types of machine learning models of interest for computer vision are di-
vided into supervised learning, unsupervised learning, and semi-supervised learning. Su-
pervised machine learning is characterized by labeled training data. In essence, the
algorithm is fed a number of pre-classified examples to learn from to classify a subse-
quent testing set of unlabeled data. The goal is for the algorithm to develop a mapping
function $f(x)$ that accurately resembles the transformation from each input to each as-
associated output so that it can be subsequently used to predict outputs for data that has
not yet been classified. A machine learning model that is meant to classify images as
either containing a cat or a dog would be given a large set of images that are labeled
either cat or dog to learn from so that it can subsequently compare its knowledge to
those examples when classifying new images.

Unsupervised machine learning is much more free-form. A machine learning model
is simply handed a dataset that has no explicit instructions, classifications, or correct
answers attached to it. From this set, the algorithm must find a pattern or structure
in the data on its own by extracting useful features and analyzing the structure of the
training data. Having to determine the relevant features and patterns itself allows the
algorithm to go beyond what its creators might have intended and find details that might
be overlooked if the model was only looking for pre-classified features. It is especially
useful if there are no recognizable features to pre-classify or if it is too difficult to make
a clean labeled dataset for helpful supervised learning.

Semi-supervised learning is somewhat of a happy medium between supervised and
unsupervised learning. In this regime, a machine learning model is given a training
dataset that contains both labeled and unlabeled data. Thus, an algorithm can combine
the direction provided by supervised learning with labeled training along with the freedom that unsupervised learning allows, letting the model both focus on a particular task while giving it room to innovate beyond where the labeled data might prevent it from doing so. Therefore, the algorithm bypasses problems with pre-classifying every piece of data since it can be difficult in large data sets with hundreds of thousands of data points or more (especially if they require manual classification), but can still benefit from a small proportion of labeled data and get much higher accuracies than purely unsupervised learning.

This project focuses on the classic machine learning model for classification and image feature detection, namely, supervised learning. This model is ideal for transient detection as supervised learning excels at predicting outcomes for new data, especially in regard to classifying new data based on the labeled examples that are given for training. For transient detection, we know exactly what the feature that we are trying to classify looks like and can compile ample pre-classified training data reflective of the features that we want the algorithm to detect. Such data can easily be accessed either by referencing existing transient catalogs of real stars or can be produced via simulation. Further, supervised learning algorithms are easy to work with as they usually do not require highly intensive computations and can be made in languages such as Python or R and run on relatively reasonable hardware.

2.2 Neural Networks

Once the overall machine learning model is chosen, the next step is to decide on or produce the most capable algorithm. There is a wide array of algorithms that all work based on the principles of supervised learning (the same is true for the other machine learning models). Some of the widest used supervised learning algorithms include linear
regression, logistic regression, k-nearest neighbors, decision trees, and neural networks. The first two are standard statistical processes that fit predictive models to given data to fill in gaps or estimate future or past evolution. k-nearest neighbors is a family of techniques that can be used for regression or classification. It works by producing an output $x$ for input $y$ by finding the $k$-nearest neighbors to $x$ in the training data and averaging over the corresponding $y$ values from the training data. Decision trees act by breaking the input space by various parameters at each node of the tree, thus dividing the data by its attributes and assigning predicted values based on where in the path an individual data point takes through the decision process. See Figure 2.2 for a graphical example of a decision tree.

Figure 2.2: Example of a decision tree that graphically illustrates response to directed mailing. Each node (large circles) divides the data pool by a different characteristic (the specific criterion is denoted along each arrow) until a final decision is reached (inside the triangles). [14]
However, due to recent innovations in computational power and memory, some of the best algorithms for computer vision are machine learning based neural networks. Neural networks in general are inspired by our understanding of the human mind. They are characterized by a hierarchical flow of data between various network layers which are themselves composed of processing nodes (also termed neurons). Data is passed from an input layer through a series of intermediate layers (or "hidden" layers) where the actual learning is performed before predictions are made via the output layer (see Figure 2.3). When a neural network has 3 or more hidden layers, we call it a deep learning network. Each node in the network implements an activation function, which, when given an
input function, decides to what degree the node is activated (if at all). All nodes are interconnected and each connection has an associated weight specifying the strength of the connection between two given nodes. Through this structure, termed multi-layer perception (see following example), weights are trained to appropriately navigate data through the layers such that a desired outcome is obtained.

For example, consider an input image such as Figure 2.4. The computer understands the image as essentially a rank 3 tensor or a matrix of color values (red, green, and blue or "RGB") which are each associated with a pixel in the image. For this image of 28 by 28 pixels, there is a matrix of 784 individual pixels which have been concatenated from RGB values into scalar values (usually a hexidesimal value where 0 is black and 255 is white, dubbed "grayscale") being stretched into essentially a rank 1 tensor which forms the input layer of the neural network.

Figure 2.4: How a computer perceives the image of a cat as a tensor. For a color image, the tensor is rank 3. Each vector corresponds to the RGB value of the associated pixel in the image. [15]
Pixel values are then normalized to be between 0 and 1 via an activation function. These functions form the connections between neurons, passing data from one neuron to another in a subsequent layer via a weight that determines the strength of a connection between the two neurons. Once at the second layer, the process repeats and the data is transferred to a subsequent layer with a new associated weight (and perhaps activation function).

There are a wide variety of activation functions, one of the simplest being a linear activation function (see Figure 2.8 for a graphical representation), or simple matrix multiplication by a weighting matrix and linear scaling of the input vector. However, linear activation functions are quite limited when compared to non-linear transformation functions as they are restricted to a 16-dimensional hypothesis space [16]. Thus, non-linear activation functions are the preferred methods for multi-layer neural networks so that they can take advantage of much deeper representations. Some of the more popular non-linear activation functions include the sigmoid function (see Figure 2.5), hyperbolic tangent, rectified linear unit (ReLU, see Figure 2.6), and leaky ReLU (see Figure 2.7).

Choosing the correct sequence and types of activation functions is a large component of the performance and accuracy of a neural network, however it can be a very tedious process to determine the most optimal configuration given the number of possible activation functions and layers in a given neural network. Only general rules of thumb exist for choosing the right activation functions based on the various application of the neural network [17].

At the end of the various neural network layers, there is a final layer made up of neurons corresponding to the classification categories. The corresponding label to the row of this output with the highest certainty is the network’s guess at the image’s classification. See Figure 2.8 for an example. Thus, we have inputted an image, and have received a statement of what the computer thinks that image contains.
In order to optimize a neural network and get it to output an accurate guess, it needs some way to learn and refine the method it uses to activate neurons in subsequent layers.
Deep Learning and Overview of Neural Networks

Figure 2.7: Graph of the leaky rectified liner unit activation function. [17]

Figure 2.8: An example of (linear) mapping an image (we assume it only has 4 monochrome pixels for brevity) to classification scores for three classes: cat (red), dog (green), ship (blue). The image’s pixels are arranged into a column vector and matrix multiplication by the weights matrix (that determines the strength each pixel is associated with a given class) is performed to get the scores for each class. In this example, matrix multiplication and the additional bias correction acts as the simple linear activation function that transforms the input vector into a new vector ready to be remapped by another neuron. Finally, this mapping results in an output vector which contains what corresponds to the algorithm’s certainty of classifying the image as a cat, dog, or ship. [15]
This is where the labeled training data comes into play by defining a loss score. Specifically, the neural network’s output vector (i.e. its guess for classification) is subtracted by the target values to determine a distance score—in essence how far off the algorithm is in its predictions. This score is then used as a feedback signal to update the training weights towards a more favorable loss score through a backpropagation algorithm that is instantiated by an optimizer. Weights are initialized as random, so the initial loss scores are naturally quite high. But after a number of loops of training via this method (typically over tens of iterations using thousands of labeled data points), a network that minimizes the loss function can be obtained. See Figure 2.9 for an example of this process.

![Diagram](image)

Figure 2.9: The loss score is used as a feedback signal to adjust the weights for each relevant layer. [16]

The type of loss function is another variable one can choose when designing a neural network. The overall goal is to minimize the loss score, so functions are chosen as part of
an optimization procedure involving both the loss function and some kind of optimization algorithm. Some well-known loss functions include softmax loss (or cross-entropy loss), modified softmax loss, angular softmax loss, additive-margin softmax loss, arcface loss, center loss, and focal loss. The softmax loss category of loss functions all operate based on equation 2.1, with some modifications in the cases of the later functions which are meant to improve performance. In equation 2.1, $L_S$ is the loss score and $x_i$ denotes the embedding value of the $i$-th training sample belonging to the $y_i$-th class. Further, $W$ is the weight matrix, $b$ is the bias term, $T$ defines the transpose of a given matrix, and $N$ and $C$ are the number of training samples and classes respectively [18].

$$L_S = -\frac{1}{N} \sum_{i=1}^{N} \log \frac{e^{W_{y_i}^T(x_i+b_{y_i})}}{\sum_{j=i}^{C} e^{W_{j}^T(x_i+b_{j})}}$$ \hspace{1cm} (2.1)

Center loss is typically combined with softmax loss to improve the latter’s classification ability since the former minimizes the distance between a class’s center and the corresponding features, providing a complimentary avenue of analyzing the loss. Focal loss is designed to alleviate training inefficiency that is due to imbalance between foreground and background classes so that the algorithm is trained more on the hard examples in the input data than the easy to identify samples [18].

Once a loss function is implemented, an optimization algorithm is used to help the loss function lower the prediction error by providing the network with information used to update the weights of each relevant layer via a method called backpropagation. Commonly, optimization is done by Stochastic Gradient Decent (SGD). This method uses the fact that the gradient of a given function always points in the direction of maximal increase, so if the direction opposite to the gradient is always followed, a function minimum will eventually be found. In this way, a machine learning loss function can be minimized. However, as seen in equation 2.1, when there are multiple training examples, the loss
function becomes a sum over all the various loss functions for each training example. The gradient must be calculated for each individual term in order to determine an overall gradient, but when the number of terms is high, the computational load needed to find the exact gradient can be great. SGD solves this problem by choosing only a few of the loss functions in the overall sum randomly to base the direction of optimization on. Further, SGD uses an adaptive learning rate that varies the optimization algorithm’s step size each iteration. This allows the algorithm to maintain a small step size that avoids noisy behavior which can lead to problems with converging to a final value while also sizing up the step size where needed to avoid plateaus or local minima where the algorithm can get stuck [19].

Additional common methods of optimization include the ADAM algorithm [20], the Broyden-Fletcher-Goldfarb-Shanno algorithm, and the Limited-Memory Broyden-Fletcher-Goldfarb-Shanno algorithm [21]. The latter two are known as second-order methods and work by calculating the Hessian matrix to account for the curvature of the overall function. ADAM is, in general, an improvement upon SGD and operates on the same principles, but SGD is a very simple and still extremely effective method for the purposes of most neural networks which is why it was chosen for this project.

### 2.3 Convolutional Neural Networks

Convolutional neural networks (CNNs) build on the capabilities of regular neural networks by optimizing for local feature detection. See Figure 2.10 for the basic structure of a CNN. Specifically, rather than associating a neuron with a single pixel, neurons are associated with a small nxm local feature map within the image. These local features are then convolved with the associated weighting matrix to produce a result that is a measure of how much the local features influence the overall output classification. This method
Deep Learning and Overview of Neural Networks Chapter 2

gives the algorithm the capability to detect objects that have been translated or rotated from the training examples. Further, it gives the algorithm a basis of features which can be used to identify an object when it is only the local features that differentiate two given objects or in the absence of the full set of features that make up a given category. For example, a human can look at a picture of a Bengal cat and a similarly colored leopard and be able to differentiate them as two different species, however a neural network without the ability to understand local features may not be able to recognize the two species since they appear superficially similar. Furthermore, a human can identify that an image contains a cat even if that image only shows the cat’s tail. A neural network needs a CNN’s understanding of local features (like various animal tails, ears, etcetera) if it has a hope of classifying such images appropriately.

![Figure 2.10: General structure of a CNN that is optimized for looking at local features to aid object detection and classification. The algorithm passes input data through a series of pooling (i.e. feature reduction) and convolutional layers until reaching a classification layer which outputs predictions. [22]](image)

**2.4 YOLO Darknet**

The specific convolutional neural network that we employ to detect transients in this project is the You Only Look Once (YOLO) algorithm for unified, real-time object
detection first presented in Redmon et al (2015) [23]. Most other algorithms take a
given model for an object and run a classifier that evaluates the image for the object
at various locations and scales. This means that to identify where in the image a given
object is, the input image would need to be cut into pieces of various sizes which are
then fed back into the network so that it can determine if any of these pieces contain
an object of interest. Unsurprisingly, this method takes a long time as it has to run
the same network over many different pieces of a single input image. In principle, a
neural network that partitions the input image can be written and combined with the
classifying network, however such methods for object detection are hard to optimize since
they require multiple neural networks to perform multiple components of analysis.

![Figure 2.11: YOLO operates by first resizing the input image to 416 by 416 pixels, then
it runs a single convolutional network on the image which outputs resulting detections
with the algorithm’s confidence. [23]](image)

YOLO reframes this problem via combining both the partitioning of the image and
the categorization into a single network as presented in Figure 2.11. Thus, YOLO func-
tions by only looking once at a given image and determines the location and classification
probability of the objects that it is trained to detect. This unified object detection op-
erates by using global feature detection and reasoning as shown in Figure 2.12. The
algorithm begins by dividing the input image into a SxS grid where a grid cell is respon-
sible for an object if it is at the center of the cell. These cells predict the bounding boxes
and their confidence scores. The latter reflects how confident the model is that the box
contains a given object and how well the box encompasses the object.
Figure 2.12: YOLO divides an input image into a SxS grid and for each cell predicts a number of bounding boxes and the confidence and class probability associated with each bounding box. [1]

Based on Figure 2.13 and 2.14, YOLO is on par with the mean average precision

Figure 2.13: YOLOv3 runs significantly faster than any other contemporary network while maintaining comparable mAP. [1]
YOLOv3 is much better than ResNet variants and comparable to models on the AP\(_{50}\) metric. Additionally, YOLOv3 is 3.8 times faster than RentinaNet. [1] [24] [25] [26] [27] [28] [29] [30]

The performance (mAP) of other one-step and two-step contemporary neural network based detection algorithms. However, YOLO runs approximately at least 3 times faster than the rest. Depending on hardware, YOLO runs at 45 frames per second but can be optimized to run at 150 frames per second. Therefore, it is an ideal method of in-real-time processing considering it has less than 25 milliseconds of latency once trained [31].

Further, for stellar transient detection, any loss in accuracy can be made up through cross correlation with images of the source captured in other frequencies of light. These multimessenger data can help to localize a given source and analyze further properties that lead to a more accurate identification. See Section 5.3 for more discussion.

Additionally, the numbers in Figures 2.13 and 2.14 are based on the Microsoft Common Objects in Context (COCO) dataset. [32] COCO is a set of 328 thousand images of over 90 unique categories of common objects in their natural context meant as a training dataset for object recognition algorithms like YOLO. Since neural networks generally perform worse the more categories they are trained to detect, detecting transients (which only involves a handful of categories at most) is apt to have higher accuracy than that of the COCO dataset.
Chapter 3

Machine Learning in Astrophysics

3.1 The Data Problem

In order to perform supervised learning to the scale as described in the previous chapter, large datasets that have been previously labeled are needed to train our chosen neural network to detect the features we are interested in finding. To train an algorithm that is robust and capable of generalizing pertinent features to novel images, it is optimal for a training set to contain at least upwards of several thousand (or more, upwards of a hundred thousand) examples of the object one wants the network to learn to detect. If the training dataset is too small, the algorithm risks simply memorizing the training set of images and their answers so that it cannot generalize to feature detection on a novel image (this is known as overfitting). Thus, the problem becomes one of gathering enough data, labeling it with the answers we want the algorithm to learn to detect, and finally, having the hardware capability to store and train that much data in a reasonable timeframe.

Astrophysics, as a field, has an exceptional backlog of data from a great number of surveys that have been digitized and made available to researchers, especially in the
internet age. Much of these data have been analyzed by various efforts, but most of it is not in an easily usable form for neural network training due to factors like not being the correct type of data for a given analysis, or more importantly, not being labeled correctly. Having appropriately labeled training data is paramount to training an effective algorithm, but until a computer is trained to locate objects in an image itself, a human must spend the time to create labeled data. When there are thousands of images in a given training set, the time it takes to label a full dataset becomes unmanageable.

Some projects solve this initial hurdle by crowdsourcing. For example, the Zooniverse [11] is a project that makes classifying galaxies, exoplanets, bird species, proteins, white blood cells, literature, and many other topics across disciplines into a game that engages the public to help researchers build training datasets through citizen science. According to their website, Zooniverse.org, as of 2022, they have achieved almost 7 hundred million classifications across their various efforts with help from over 2 million volunteers.

Other projects make do with smaller amounts of real data by employing a method termed transfer learning.

### 3.2 Transfer Learning

At its core, transfer learning is a method by which a neural network is trained on data that is adjacent to actual data of interest, and the knowledge from training on this adjacent data is transferred in such a way that the network is able to perform well on the actual data that it was not trained on. This method parallels human learning in the sense that if a person learns to play the piano, this person will likely have a much easier time learning to play a pipe organ since some of the skills cross over and can be used to accelerate mastery of this new skill. In a similar fashion, a neural network can be trained on say, generic object detection before being specialized into something that
specifically identifies a small subset of objects even if the neural network has never seen those specific objects before further training.

There are two types of transfer learning that are pertinent to this project. The first type is when a basic machine learning model is first pre-trained on a large number of generic data for the given task. For computer vision, this would mean that pre-training takes place on a large dataset of labeled everyday images, such as those compiled by Imagenet [33] or COCO [32]. With this training, the model gains the ability to extract low-level features from the dataset. From there, this base model with mature feature extraction abilities can be frozen and placed as the backbone of a second network. The final layers of this second network can then be trained on a much smaller amount of specialized data to fine-tune the algorithm to the specific task that the algorithm is being trained for (see Figure 3.1). Thus, the demand for labeled training data specific to the task of interest is reduced to only a few thousand images (even a few hundred in some cases). The accuracy of this method is mostly dependent on how well it was trained on the first set of training data and on the similarity of the initial and secondary

![Figure 3.1: A network is pre-trained on an initial generic dataset. From there, the network is copied and freezeed (made non-trainable) for learning on a second, smaller dataset. Only the final classification layers are trainable with the second dataset. [34]](image-url)
datasets, though the specific features a neural network identifies tend to be fairly generic, especially in computer vision (for example see Figure 3.2). This type of transfer learning is fairly standard throughout computer vision projects. YOLO itself is a pre-trained model based off of Imagenet, and this project follows from previous work conducted in the lab that used transfer learning for detection of exoplanets [36].

The other type of transfer learning of interest to this project is using simulated data to substitute for large scale training on real data. Simulated data can be used to either fully train a network from the ground up, or as the data that train the final layers that specialize a pre-trained network. If the simulations are close enough to the real thing, this technique can almost totally eliminate the need for having real data for training,
reserving it only for validation steps. This method is then extremely helpful when real data is scarce and the target objects are easily created artificially. See [36] for an example.

3.3 Transient Simulations

Due to the ease of simulating and labeling large numbers of artificial starfields with transients, as well as previous work by other lab members who showed that YOLO is capable of extrapolating to real starfields after training on such simulations, this project uses simulated data as our main training data.

Data is simulated using the Skymaker program developed by Bertin et al in 2009 [2]. We simulate 1000 image pairs per simulation setting to create a robust database of starfields for training. These images take the form of 400 by 400 pixel image pairs (conforming to YOLO’s input image size) with randomized starfields (sparse, dense, and middling) and are created assuming a 10 second exposure time in the program’s g filter with a background magnitude of 22.0. Each pair contains a specified number of transients,

Figure 3.3: Example starfield simulated for this project using Skymaker.
where the transient objects vary by 1 magnitude, 0.75 magnitudes, 0.5 magnitudes, or 0.25 magnitudes between images. Transients are given starting magnitudes between 6 and 19. See Figure 3.3 for an example image of these simulations.
Chapter 4

Real-Time Neural Network Based Source Extraction

4.1 Overview

To train a YOLO based neural network capable of real-time transient detection, we first assemble a database of simulated pairs with the specifications given in Section 3.3, then each pair is individually aligned, a median averaged template image is created combining the two images in each pair, and finally, this template is subtracted from both images using the subtraction method developed by Bramich (2008) [5]. At this point, we are left with a residual image that shows only the transient sources in a starfield (assuming good subtraction). See again Figure 1.2 and Figure 4.1 below for a residual of a simulated image. Such residual images are then converted to png images with an appropriate scaling and then divided into training and validation sets that are used to train YOLO. Each image is accompanied by an appropriate location label and bounding box for the transients contained in the image. These are easily compiled with a short Python script since Skymaker’s simulation configuration file contains the location of each
source in the image. The bounding boxes were calculated from the radius of the point spread function for a given magnitude.

Figure 4.1: Example of a residual image from a simulated pair of images. The circle shows the location of the inserted transient which varied from 16 to 17th magnitude between the two simulated starfields.

Multiple training sets were created to test YOLO’s efficacy at the following detection regimes. We started with images simulated with a single transient that varied by 1 magnitude between the images in a given pair. Pairs ranged from 6-7 magnitude to 18-19 magnitude. We then decided to focus on YOLO’s effectiveness at detecting dim objects and isolated the training set to 1 magnitude difference pairs between 16 and 19 magnitude. Finally, we tested on a full set of images with magnitude differences of 1, 0.75, 0.5, and 0.25 on the entire range of transient magnitudes (6 to 19).
The first set had a total of 15000 images where 4500 of those were used as the validation set (the other 10500 were training images). The second set had a total of 3000 images where 900 were used as the validation set and 2100 as the training set. The final set used a total of 69000 images over the full range of magnitudes and magnitude differences. 20700 of these images formed the validation set and the algorithm was trained on 48300 images. Thus, 30% to 40% of the total data was used as validation while the rest were used to train the algorithm. These ratios were chosen because with somewhat smaller datasets, the validation set needs to be comprised of a comprehensive set of example images that cover all varieties of images we would expect the network to be able to detect in order to sufficiently generalize the algorithm. With more data availability (in this case we are mostly constrained by computational resources and the time it takes to train a full neural network), such ratios can approach 10%/90% validation/training data, but \( \sim 30\% / 70\% \) remains a commonly accepted ratio.

### 4.2 Results

Our lightweight YOLO based neural network performs as expected and detects transients within 50 ms per image at a high confidence level for all but the dimmest transient differences. Transients brighter than 16th magnitude were detected almost 100% of the time with our network, and such findings had high \((\geq 90\%)\) confidence as true detections according to YOLO’s classification (See Figures 4.2 and 4.3). At the least, in this regime, the transient had the highest confidence detection over any other subtraction remnants in the residual image. Beyond 16th magnitude, the predictive power of our algorithm falls somewhat to \(\geq 75\%\) for transients between magnitudes of 16 and 18.5. Detection confidence also fell and the algorithm was no longer detecting the true transient in all images due to how faint the source appears. Still, for the vast majority \((\geq 75\%)\) of
Figure 4.2: Output image from the stars_6_19_v2 training version of YOLO. The background image is a residual created from an image with transient difference between magnitudes 10 and 11. Even though there are other source remnants in frame, YOLO still detects the singular true transient with 99% confidence and does not falsely classify any other source in the image as a transient.

Figure 4.3: Output image from the stars_16_19_v2 training version of YOLO. The background image is a residual created from an image with transient difference between magnitudes 17 and 18. Despite noise and how faint the transient is in this residual, YOLO is still able to find and classify the transient with high accuracy.
images, the true transient was detected and correctly identified. An overall graph of the performance of various training runs is found in Figure 4.5. We see that as the transients get fainter and magnitude difference between stages of the transient gets smaller, YOLO has a harder time detecting the correct source in the image. Considering that the transient in such images is quite undecidable even to a human eye, this is not surprising. See Figure 4.4 for an example of a non-detection.

Even with the lessened accuracy at dim magnitudes and small magnitude differences, these algorithms have a high accuracy for a quick look based detection pipeline. Operating in only a few 10s of milliseconds, this network already proves itself as a useful in-real-time first-look source extraction method for, at the very least, sources brighter than $\sim16$th magnitude, which includes much of the transient phenomena we are interested in.

![Figure 4.4: Example of a an image with a dim transient (in this case the magnitude difference is between 17 and 18) which the neural network could not detect. For this particular image, it may be also due to the transient being somewhat close to the edge of the image at the bottom right border.](image)
4.3 Π in the Sky

With the demonstrated capabilities, this neural network based extraction algorithm is being tested as a main feature of the real-time Π in the Sky survey pipeline being developed by an ancillary team in Professor Philip Lubin’s Experimental Cosmology Group at UCSB. Π in the Sky is a proposed wide-field full sky survey where an array of telescopes captures a view of the sky, horizon to horizon, every integration interval. This project has an overall goal of consistently monitoring the nearby sky for purposes such as planetary defense, comet and asteroid tracking, following bright stellar transients, and can even be used for observing plane or satellite flight paths. If we assume that this project will use a 10-60 second integration time for each observation, and given how large of an array of telescopes is necessary to get a π steradian field of view, the amount of image data collected within a few days of observing is astronomical. Further, since the prime directive of this project is meant as a planetary defense measure, identifying threats quickly is paramount. Our first-look source extraction method is an ideal step in this data pipeline as the neural network based source extraction takes up only a few milliseconds and detects anything bright that changes between two time points with ≥ 90% confidence. Asteroids (especially larger ones) are some of the brighter objects in the field of view of this array, so this use case is ideal for the neural network. Additionally, a trained YOLO algorithm is lightweight enough to fit and run on a small computer optimized for tensor operations that can be used as the control unit for each telescope in the array. An overall goal of this project is to make the transient detection pipeline operate within the proposed integration time itself, so that new transients can be found almost as soon as they are observed. This goal on any hardware that is reasonably attached to the telescopes in this proposed array will need to be accomplished by a primarily neural network based pipeline since almost any other type of algorithm will not operate fast enough to perform
tasks like image subtraction or source extraction within the integration time.

At this time, Π in the Sky is in an initial stage of building prototype observing domes and telescope setups which will subsequently be used to test the full real-time analysis pipeline for application to Π in the Sky. These domes are in construction at the top of Broida Hall at UCSB. See Figures 4.6 and 4.7 for images of the dome setup modeling and construction. We will use QHY268M Pro cameras on the Celestron 8" RASA telescope as our prototype observing setup due to generous donations by the Las Cumbres Global Telescope Network.
Figure 4.5: Percent of transients identified correctly by magnitude for 4 different training runs. stars_6_19_v1, stars_6_19_v2, and stars_16_19_v2 were trained on solely 1 magnitude differences while stars_6_19_v3 was trained on all images including the 0.25, 0.5, and 0.75 magnitude differences.
Figure 4.6: Image of the first partially constructed dome on the roof of Broida Hall at UCSB.

Figure 4.7: CAD render of the 4 domes at their proposed locations on the roof of Broida Hall at UCSB.
Chapter 5

Conclusions and Going Forward

5.1 Overview

Motivated by the amount of data contemporary astrophysical surveys are collecting and the need to process these data within a reasonable timeframe for reacting to time sensitive phenomena, we present a nominally accurate and fast first-look, real-time source extraction method. This method is neural network based, specifically using YOLO Darknet, a pre-trained convolutional neural network that we optimized for transient source extraction from subtracted image residuals created from starfields simulated with Sky-maker. With this algorithm, we obtain a source extraction method that, once trained, operates within a few tens of milliseconds per image and identifies transient sources \( \geq 16\text{th} \) magnitude with \( \geq 99\% \) accuracy and \( \geq 90\% \) confidence. Even dimmer sources (between 16-18.5 magnitude) were found with \( \sim \geq 75\% \) accuracy.
5.2 Collection of and Testing on Real Data

The results from our source extraction method are highly promising in our development of a neural network based transient detection pipeline for real-time first analysis of observations. From ancillary work in our group, we know that YOLO is very capable of generalizing from Skymaker’s simulated images to real image data. This ancillary work was done on temporally static datasets, so our project has the added complication of time series data. However, the similar results on this other effort shows that our current algorithm should have a high degree of success at detecting transients in real data. We use data archived by the Palomar Transient Factory [37] as well as images from LCOGT [4] to test our algorithms’ capabilities in this regime.

Further, in our work with Π in the Sky, we will be using the real data collected by this survey each night to incrementally train and test our algorithm each day once the observing night is over. This procedure will make the algorithm robust to this specific survey’s data and idiosyncrasies. Similar training can be performed to specify the algorithm to any instrument or set of instruments. The only limiting factor is labeling the newly collected data appropriately, which can be done by more classical transient detection pipelines if desired since this step need not be done in real-time.

5.3 Multimessenger Imagining

In addition to CNN based transient identification and classification, neural networks can be used to determine follow-up observing strategies for time sensitive observations. A recent movement in time domain astrophysics has defined the field of multimessenger astronomy which seeks to identify sources across wavelengths, neutrino data, and gravitational wave observations. For example, Margutti and Chornock (2021) [38], Coughlin
et al. (2018) [39], Bright et al. (2022) [40]. Neural networks can be trained to submit instantaneous follow-up requests on time sensitive objects that are detected using a fast algorithm such as the one we presented. Such functionality cuts out the normal time needed to reduce, analyze, and manually submit follow-up observations which could potentially miss time-sensitive features of various phenomena (for example, a neutrino burst accompanying a supernova) while also potentially making available matching observations that are practically simultaneous to the initial observation.

Furthermore, multimessenger data can be used to make classification algorithms much more robust and capable of identifying phenomena across data types. Most astrophysical phenomena emit stronger or more uniquely in certain wavelengths or data types more than others. Collecting, labeling, and using all the different data types available to astrophysicists in a multilayered network similar to industry utilized algorithms such as that shown in Figure 1.3 can highly improve our ability to detect transients and determine their classification.

Additionally, it would be advantageous to be able to observe in multiple wavelengths on a single telescope in order to lessen the need for follow-up observations and obtain multispectral data that is not time separated. Towards that goal, an ancillary team in Professor Philip Lubin’s Experimental Cosmology Lab at UCSB has been developing a multidichoric camera capable of imaging optical data in 3 filters simultaneously.

5.4 Conclusion

In order to reach the highest accuracies and incorporate the functionalities described above into an overall neural network based pipeline, we will eventually need to write our own algorithm that is optimized for astronomical data and transient phenomena. However, in the meantime, we have shown that even a basic pre-trained computer vision
algorithm is fully capable of contributing to transient science and performing at a high level of accuracy and confidence considering the relatively small amount of training data.

Our future work seeks to discover the full capabilities of these algorithms in regards to transient science. I will be attending University of California at Berkeley in the fall of 2022 to work under Professor Joshua Bloom and Professor Raffaella Margutti, who are leaders in machine learning based astrophysics and multimessenger time domain observation and analysis. I plan to continue this effort under my new mentors’ guidance.
Bibliography


