Leveraging Machine Learning for Source Detection with the NASA Roman Telescope

Hebu Patil
Ashish Mahabal and Roberta Paladini

Abstract

We focus on developing a convolutional neural network (CNN) machine learning model to identify transient objects in astrophysical images for later classification from the Nancy Gray Roman Space (Roman) Telescope[1]. The telescope is scheduled to launch in 2026, so the model is currently using simulated image pairs containing variable and transient objects, coupled with key metadata such as magnitude and coordinates from accompanying catalog files. Snapshots of images with transients were selected for training, followed by magnitude difference calculations using aperture photometry. The core of the project involves constructing a model designed to identify the locations and brightness of transients using celestial coordinates as ground truth. Results show the model can identify the coordinates of transients within a 2 pixel range with 98%. Future work includes refining the model with a larger dataset and implementing advanced preprocessing techniques for artifact detection, along with improving magnitude detection accuracy. This work aims to contribute to the broader goal of building a robust machine-learning pipeline for the identification and classification of celestial events from the Roman.

Background

The detection of transients and variable phenomena—brief astronomical events such as supernovae or gamma-ray bursts—plays a pivotal role in advancing our understanding of the dynamic universe. With the advent of next-generation telescopes like the NASA Roman Space Telescope, we are positioned to capture unprecedented datasets that will require sophisticated machine learning techniques for automated analysis specifically tailored to each telescope [5]. Traditional methods of transient detection often rely on manual or semi-automated processes, which are not scalable for the large volumes of data expected from upcoming surveys. This necessitates the development of advanced computational models capable of efficiently detecting and classifying transient events [1]with high accuracy. This is not the first transient detection algorithm, but a novel approach for the Roman and one of few machine learning approaches. Methods such as ZOGY [5] or TransiNet [5] provide an image differentiation algorithm to identify transients , which may be accurate but introduce "lag time" that a machine learning based inference model could cut down significantly.

The primary goal of the project was transient detection. We experimented with both an autoencoder architecture and a CNN. The autoencoder was tasked with processing

[1] The NASA Roman Telescope. https://roman.gsfc.nasa.gov/

pairs of non-variable and injected transient images, with the goal of isolating the transient signal. In the future, when the model is implemented there will no longer be injected images, but rather true transients, which might vary from the simulated data that we are currently feeding it. The autoencoder included convolutional layers in the encoder for feature extraction and upsampling layers in the decoder to reconstruct the transient signal. Despite careful tuning of hyperparameters—such as batch size, learning rate, and the choice of loss function—the model struggled to accurately reproduce the transients, often optimizing for recreating the background noise. Recognizing the limitations of the autoencode and that it was unnecessary to recreate an image when all we needed was delta magnitude, we shifted to a CNN-based approach designed to directly predict the coordinates and magnitudes of transients from image pairs. The CNN architecture, consisting of multiple convolutional layers and a dense output layer, allowed for more direct spatial and magnitude estimation. The model was evaluated on a variety of performance metrics, including the Euclidean distance between predicted and true transient positions, and the difference in predicted versus true magnitudes. Augmentation tests, such as distance versus magnitude and edge effects, provided deeper insights into the model's robustness. The CNN model demonstrated promising results in terms of special localization, with average positional deviations of less than 2 pixels for the majority of test cases. The model exhibited robust performance against small image augmentations, though further refinements are required to improve its sensitivity to faint transients and to handle transients near the edges of images.

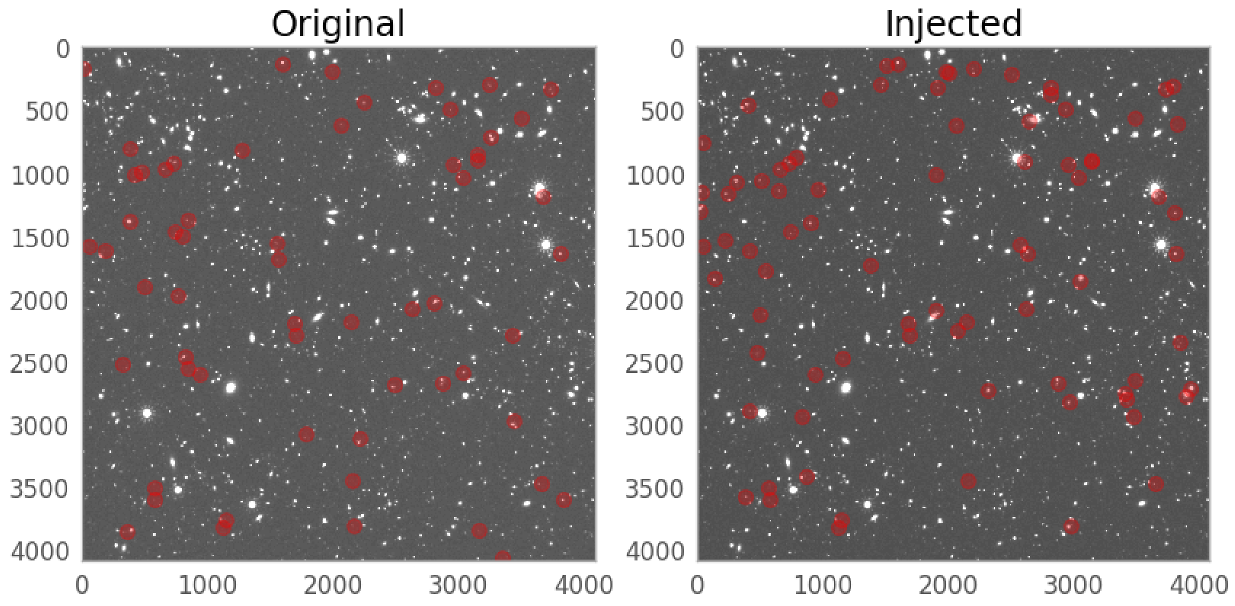


Figure 1. *Two 4088x4088 grayscale images representing an image with and without injected transients. The red scatter plots highlight the locations of transients in each image.*

This work builds on other machine learning efforts in the field of astrophysics, particularly in the domain of object classification and transient detection [2][5]. Our approach, while tailored to the Roman data, could be adapted to other large-scale astronomical surveys, paving the way for automated, scalable detection systems. Future work will explore more complex architectures, larger datasets, and expanding on detection with the identification of transients. In conclusion, this project represents a step forward in automating transient detection in large-scale astronomical datasets. The methodologies and findings outlined here have broader applications in astrophysical machine learning, particularly for future missions like Roman, where data volume will far exceed manual processing capabilities.

It is also important to acknowledge that prior to working on the main project, we worked on building a star/galaxy classifier to differentiate between static celestial objects and more dynamic features as a warm up to using astronomical data, much like other classification projects [2]. This classifier achieved a peak accuracy of 98% for relatively bright objects, providing a solid foundation for further exploration into transient detection. The results of the classifier were evaluated using standard metrics such as confusion matrices, accuracy curves, and ROC curves. The model's interpretability was enhanced through SHAP [3] analysis, helping to understand why certain objects were misclassified. These learnings were helpful to gain fluency in how to interact with the dataset and iterate on the model.

Code Architecture

The model implementation in Python was structured to allow flexibility in data handling, augmentation, and model training. The main components of the codebase were:

- Data Loading and Preprocessing: Two separate notebooks utilizing generalized functions were used to extract datasets from either AWS stored surveys or directly from FITS files with their corresponding catalog file.
- Model Creation: The function train_model.py constructed a CNN based on the architecture described above with modularization allowing for easy modification of hyperparameters such as the number of filters, kernel size, and dropout rates.
- Training and Testing: Training was made simple where .npy files containing images and coordinates could be selected and initiate the training process with the dataset prepped for training. During training batch size, epoch number, and learning rate can be altered to adjust the model. The model was trained initially on a CPU for exploratory runs, before shifting to a GPU for accelerated training.
- Evaluation and Metrics: Model evaluation functions provided an in-depth analysis of the model's performance, generating plots and metrics to compare the predicted versus true positions of the transients.

Dataset Generation

The dataset for training the Convolutional Neural Network (CNN) was obtained from the NASA Roman Space Telescope simulated survey data, accessed through the AWS Integrated Research System for Sustainability (IRSA) along with select files from another Roman Time Domain Survey. The data consisted of pairs of FITS files representing specific regions of the sky, in which transients would vary in brightness between each image. We attempted two methods to extract 64x64 image pairs from these large fits images for model training. The first method was extracting the locations of all transients using the catalog file, then plotting the transient with a random offset from the center and applying that same offset for its pair. This method was adequate in providing transients but would not work when we transitioned to a model that can identify whether or not there even exists a transient in the image. Thus we shifted to using a grid-based scanning approach from left to right and up to down, creating snapshots of 64x64 pixel images overlapping with the previous snapshot by 10 pixels.

Magdiff

The visibility of transients, in order to determine whether it would be fit to train on, was calculated using delta magnitude. Delta magnitude is calculated differently in each of the two cases: (1) the transient is visible in both images, and becomes brighter in one, and (2) the transient only appears in one image. If a transient is visible in both images, it is simple to subtract the magnitude values provided in the catalog file from each other to get the magnitude difference. But for those transients that appear only in one transient, one must find the brightness of the pixel area at the same coordinates in the image that does not have the transient. For this case, there is no entry in the catalog because there is no stellar object present, so we must perform aperture photometry using the flux values in the area (otherwise known as brightness values in the fits image file). By finding the average flux in the area using aperture photometry, we can then apply the magnitude formula (2), and then use that magnitude to find our magdiff by subtracting (1) the known value from the catalog for the brighter image.

$$|m_1 - m_2| = magdiff \tag{1}$$

$$m = -2.5 * log_{10} I + z \tag{2}$$

Z in (2) is “shift” or rather zero-point of the data, which is a constant that calibrates the magnitude scale. A way to calculate zero point when it is not provided in the header files of a FITS image, is by finding the difference between a known magnitude in a certain area (from the catalog) and manually performing aperture photometry on the same spot and finding that difference (3).

$$|m_{1_{cat}} - m_{1_{manual}}| = z \tag{3}$$

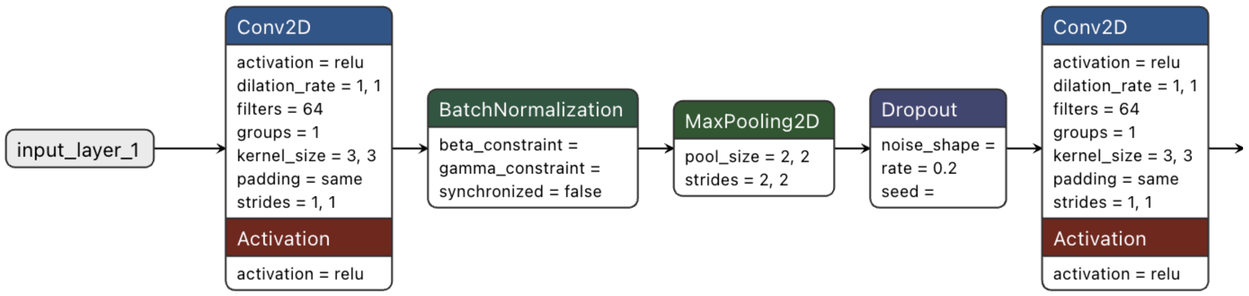
Magnitudes in both original and injected images were then subtracted to get the magnitude difference that indicates how much of a visible difference the transient had in the images. We must note here however, that in some cases the original image may have a brighter transient than the injected, in which case the pair should be reversed such that the brighter image is second.

Augmentation and Scaling

Given the initial scarcity of visible transients, which posed a significant challenge for training a robust model, an extensive augmentation strategy was also employed. This included geometric transformations such as horizontal and vertical flips, as well as rotations of 90, 180, and 270 degrees, thereby increasing the diversity of the dataset by a factor of five. In these cases the coordinates were also changed to match the augmentation of the image. Each augmentation was critical in ensuring that the model was exposed to various orientations and scales of astronomical transients, helping to mitigate overfitting and enhance generalization.

It is also important to note that there was scaling done to provide universalized brightnesses for the model to comprehend. Each pair is normalized to a 0-1 range where 1 corresponds to the brightest magnitude in any snapshot. This meant each image along with the coordinates was scaled down by a factor of 64.

Architecture



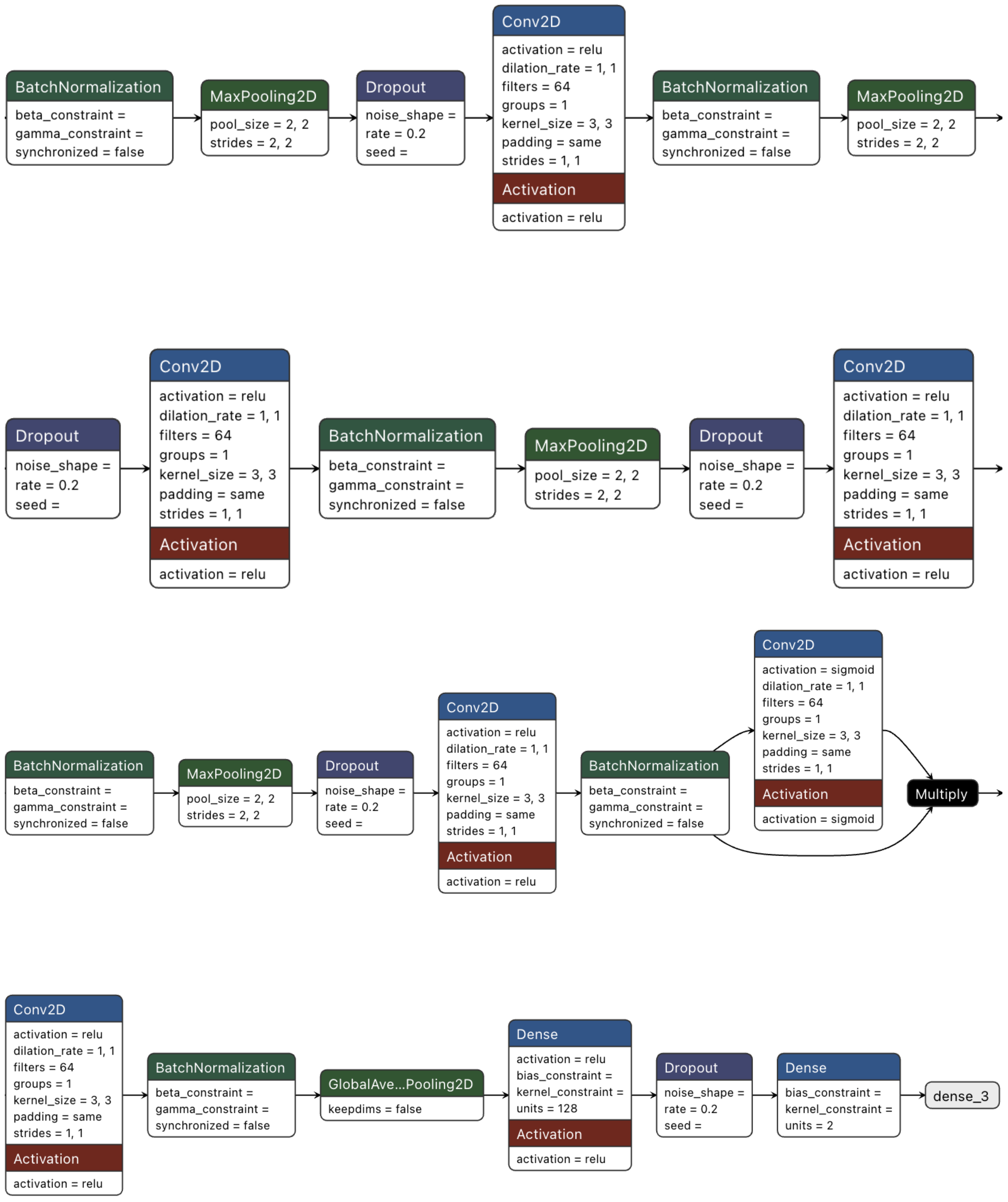

BatchNormalization
beta_constraint =
gamma_constraint =
synchronized = false
MaxPooling2D
pool_size = 2, 2
strides = 2, 2
Dropout
noise_shape =
rate = 0.2
seed =
Conv2D
activation = relu
dilation_rate = 1, 1
filters = 64
groups = 1
kernel_size = 3, 3
padding = same
strides = 1, 1
Activation
activation = relu
BatchNormalization
beta_constraint =
gamma_constraint =
synchronized = false
MaxPooling2D
pool_size = 2, 2
strides = 2, 2
Dropout
noise_shape =
rate = 0.2
seed =
Conv2D
activation = relu
dilation_rate = 1, 1
filters = 64
groups = 1
kernel_size = 3, 3
padding = same
strides = 1, 1
Activation
activation = relu
BatchNormalization
beta_constraint =
gamma_constraint =
synchronized = false
MaxPooling2D
pool_size = 2, 2
strides = 2, 2
Dropout
noise_shape =
rate = 0.2
seed =
Conv2D
activation = relu
dilation_rate = 1, 1
filters = 64
groups = 1
kernel_size = 3, 3
padding = same
strides = 1, 1
Activation
activation = relu
BatchNormalization
beta_constraint =
gamma_constraint =
synchronized = false
MaxPooling2D
pool_size = 2, 2
strides = 2, 2
Dropout
noise_shape =
rate = 0.2
seed =
Conv2D
activation = relu
dilation_rate = 1, 1
filters = 64
groups = 1
kernel_size = 3, 3
padding = same
strides = 1, 1
Activation
activation = relu
BatchNormalization
beta_constraint =
gamma_constraint =
synchronized = false
Conv2D
activation = sigmoid
dilation_rate = 1, 1
filters = 64
groups = 1
kernel_size = 3, 3
padding = same
strides = 1, 1
Activation
activation = sigmoid
Multiply
Conv2D
activation = relu
dilation_rate = 1, 1
filters = 64
groups = 1
kernel_size = 3, 3
padding = same
strides = 1, 1
Activation
activation = relu
BatchNormalization
beta_constraint =
gamma_constraint =
synchronized = false
GlobalAve...Pooling2D
keepdims = false
Dense
activation = relu
bias_constraint =
kernel_constraint =
units = 128
Activation
activation = relu
Dropout
noise_shape =
rate = 0.2
seed =
Dense
bias_constraint =
kernel_constraint =
units = 2
dense_3

Figure 2. *2D map representation of the CNN architecture where it can be seen there are 5 convolutional blocks (Conv2d, BatchNormalization, MaxPooling2D, Dropout) and an Attention layer ending with Dense to give an output.*

The final model consisted of five convolutional layers, strategically organized to extract meaningful spatial features from the 64x64 pixel input images. The core of the CNN is composed of repeated convolutional blocks, each with the following structure:

- Conv2D layers with a 3x3 kernel were applied, leveraging the ReLU activation function to introduce non-linearities. Padding was set to "same" to preserve spatial dimensions after each convolution.
- Batch Normalization followed each convolutional layer to stabilize training and accelerate convergence by normalizing the activations.
- Max Pooling with a 2x2 pool size was utilized to progressively reduce the spatial dimensions while retaining salient features.
- Dropout Layers with a rate of 0.2 were included after pooling to prevent overfitting by randomly omitting a subset of features during training.

Through experimentation, it was determined that five layers provided an optimal trade-off between model complexity and performance. Models with fewer than five layers lacked the representational power to capture fine-grained details, while deeper networks (greater than six layers) led to overfitting and diminishing returns, particularly given the relatively small dataset.

A spatial attention mechanism was incorporated into the model after the convolutional layers to enhance the model's ability to focus on regions of interest, such as transient objects. The attention layer generated a 2D weight map using a sigmoid-activated convolutional operation. This map was then element-wise multiplied with the feature maps from the previous layer, allowing the network to emphasize relevant areas while suppressing background noise. This mechanism proved especially useful in identifying faint transients, which could otherwise be overwhelmed by the surrounding data.

To further reduce the dimensionality of the feature maps, a Global Average Pooling (GAP) layer was employed. GAP provides a robust means of condensing spatial information without overfitting, as it collapses each feature map into a single value by averaging all spatial locations. The output was passed through a dense layer with 128 units and ReLU activation, introducing additional non-linearity. Another dropout layer followed this to maintain regularization.

The final layer of the CNN consisted of a fully connected Dense layer with two output units. These units represented the predicted x and y coordinates of the transient within the image. The model was compiled with the mean squared error (MSE) loss function, chosen for its ability to penalize large deviations in both the predicted coordinates and the intensity of the transients. MSE was ideal for this task as it provided smooth gradients, which are essential for optimizing both spatial localization and intensity matching.

Model Training

Training the model involved running experiments across varying epoch counts, from 100 to 10,000 epochs. Early experiments indicated that training for fewer than 1,000 epochs resulted in underfitting, as the model failed to capture sufficient detail in transient structures. However, extending training beyond 7,000 epochs offered diminishing returns, with marginal improvements in validation loss accompanied by increased risk of overfitting. Ultimately, a training regime of 7,000 epochs was selected as optimal.

During training, the Adam optimizer was used with an initial learning rate of 0.001. The learning rate remained constant throughout training, as reducing it led to slower convergence without noticeable performance gains. Batch sizes of 32 were chosen to balance memory constraints on the GPU with efficient gradient updates.

In prior stages of model development, extensive experimentation with different hyperparameters was conducted using Weights & Biases (W&B) [6] and TensorBoard. These tools were instrumental in tracking the model's performance across various configurations and enabling a structured comparison of the impact of different choices. Weights & Biases (W&B) was particularly useful in managing and visualizing experiment data, allowing the ability to systematically alter and monitor key hyperparameters such as the number of convolutional layers, filter sizes, dropout rates, and learning rates. W&B facilitated automated logging of metrics such as validation loss and mean squared error across epochs, making it easier to pinpoint which configurations led to underfitting, overfitting, or optimal performance. Early runs tested varying dropout rates (0.2 to 0.5), learning rate (0.1 to 0.0001), batch size (16, 32, 64), and epochs (100, 200, 500, 1000) to assess which hyperparameters to stick with. TensorBoard was also employed during preliminary training runs to visualize gradients, loss curves, and model graph architecture in real-time. This tool provided critical insights into gradient flow and parameter updates, helping refine the architecture by identifying inefficiencies in the network structure. By monitoring the loss curve trajectories on TensorBoard, we were able to quickly determine that larger learning rates caused unstable training, while lower learning rates led to excessively slow convergence, confirming that the learning rate of 0.001 was a stable midpoint. Furthermore, these visualizations helped in early identification of exploding or vanishing gradients, which led to adjustments such as the inclusion of batch normalization layers after each convolutional block. Using both W&B and TensorBoard for hyperparameter tuning significantly shortened the development cycle by automating many aspects of experimentation and visualization. The resulting optimized configuration of the CNN described above was ultimately chosen based on a comprehensive comparison of validation performance across hundreds of experiment runs tracked by these tools.

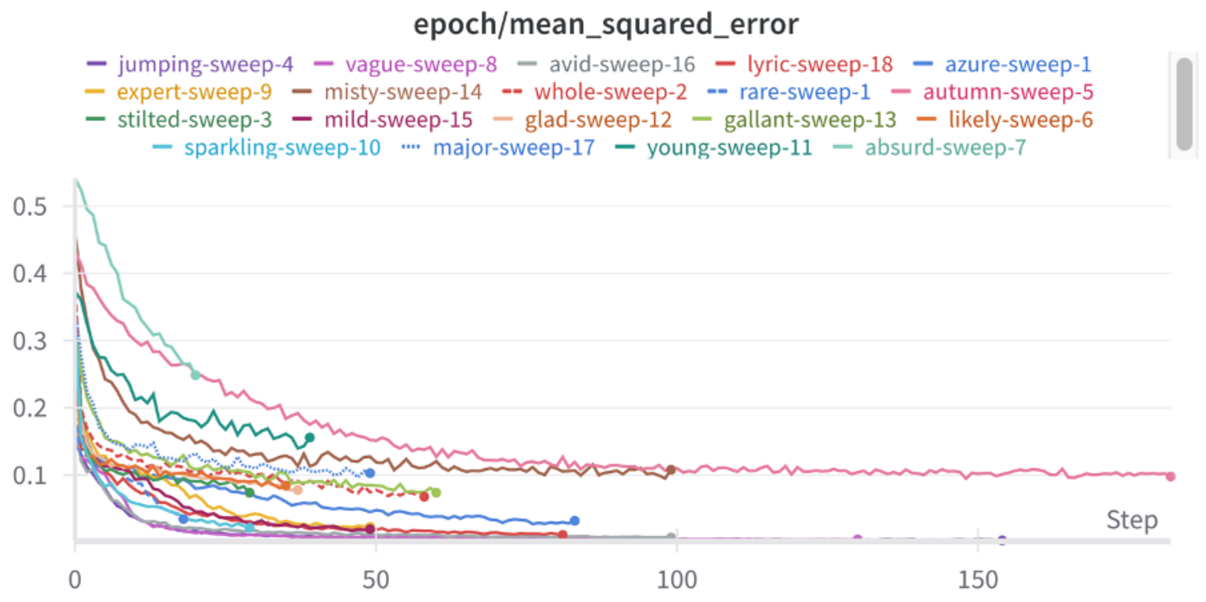


Figure 3. *Example of output created by running runs created by Weights and Biases, where one can track mean squared error over all sweeps containing different hyperparameters in order to determine which sweep produces the least error.*

Correctness

To formally evaluate the correctness and robustness of the model, we used quantitative metrics derived from the test set and qualitative assessments through visualizations of predictions. The primary measure of correctness was the mean squared error (MSE) computed between the predicted and true transient coordinates. It calculates the average of the squares of the errors, where each error is the difference between the predicted value and the true value. By squaring the errors, MSE emphasizes larger discrepancies, making it particularly sensitive to outliers, and provides a single value that reflects the model's overall performance in terms of accuracy.

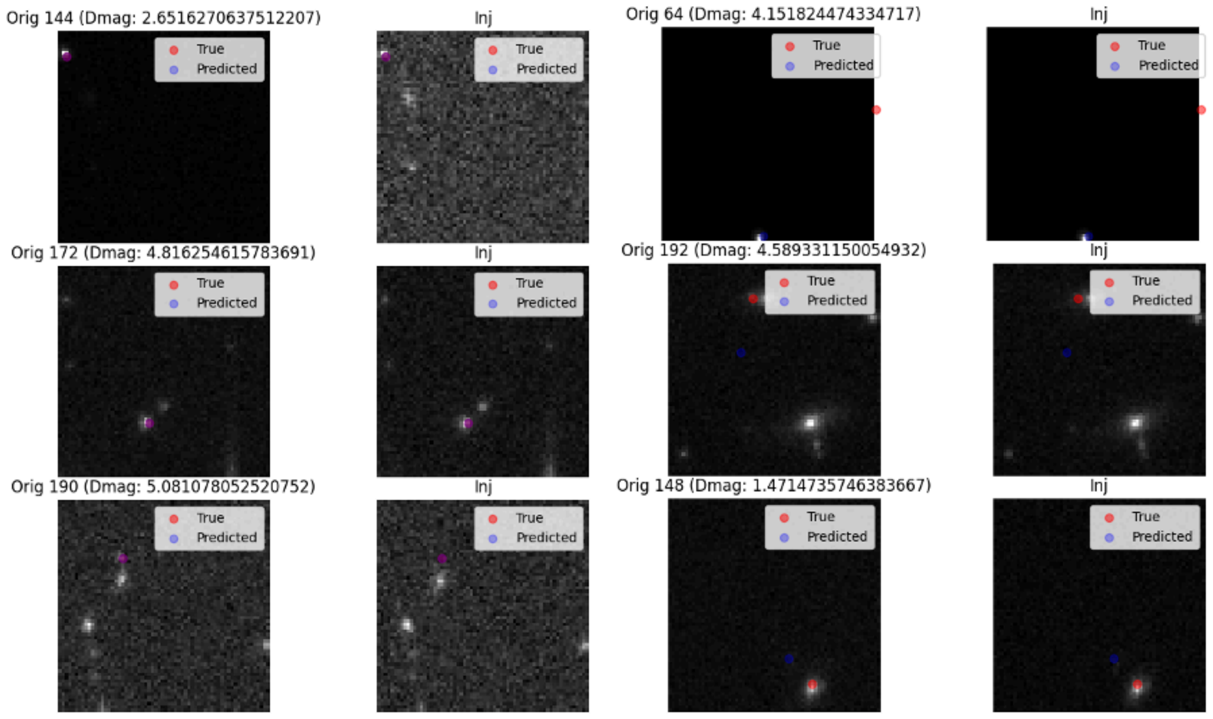


Figure 4. *To the left there are original and injected image pairs of the top 3 most accurate guesses, while the right 2 columns are original and injected image pairs of the top 3 least accurate guesses. This is the model's performance on a test set.*

From the test set results, the model demonstrated a high level of accuracy:

- 98.63% of predictions fell within 5 pixels of the true coordinates.
- 97.61% of predictions were within 2 pixels.
- 75.09% of predictions were within 1 pixel.

These results indicate a strong spatial accuracy, where nearly all predictions were consistently near the true locations. The overall mean distance between the predicted and actual positions was 0.99 pixels, with a standard deviation of 3.16 pixels, highlighting that the model generally performs well but has some variability for outliers. The maximum distance of 50.90 pixels is an outlier, suggesting a rare instance of significant misprediction, due to the lack of visibility of the transient. The minimum distance was near zero, showing that in some cases, the model pinpointed transients with near-perfect accuracy.

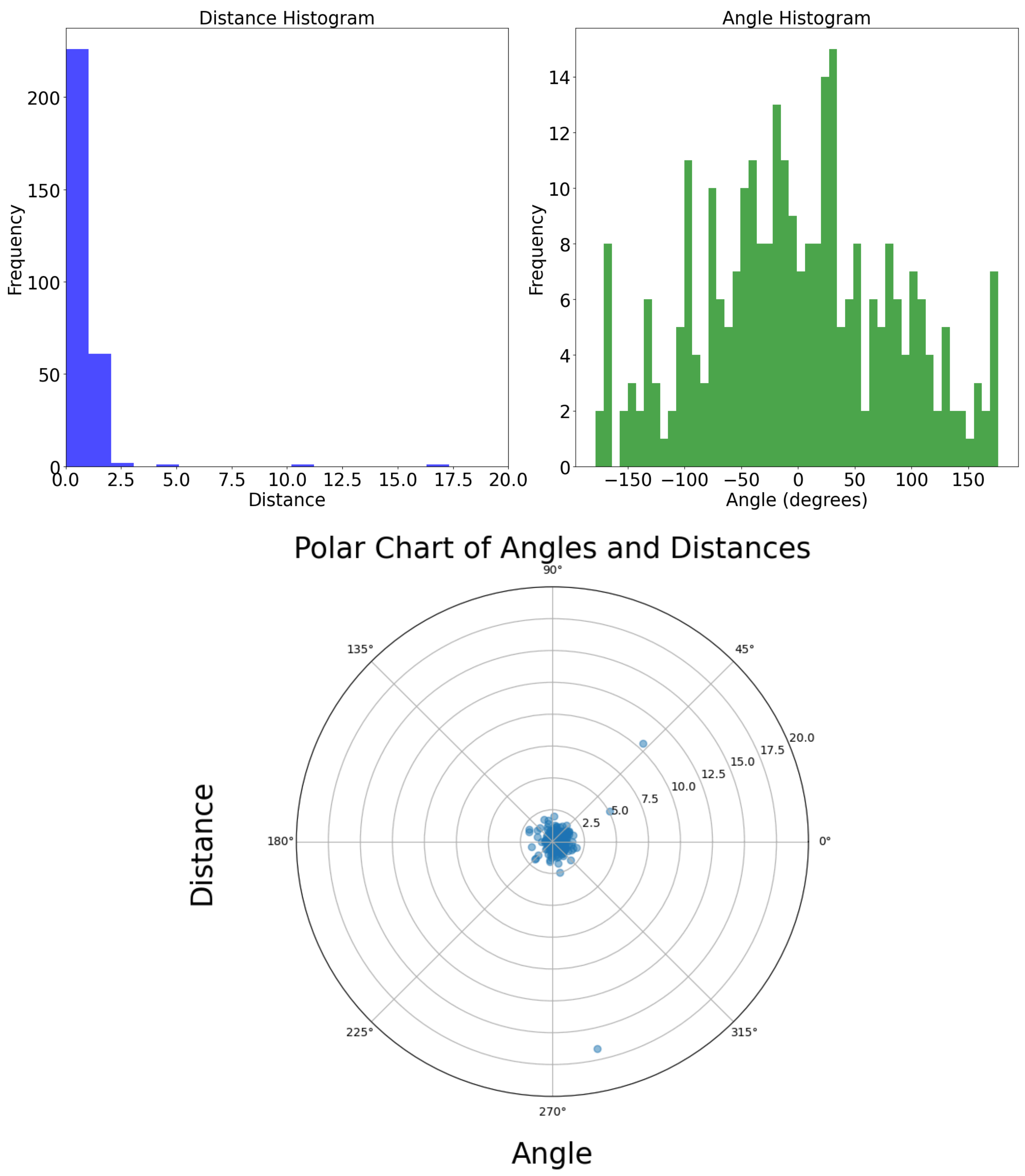


Figure 5. *Histograms representing the relationship between distance and angle of the guesses made by the model. One outlier above distance of 20 is redacted to get a better picture of the majority of guesses.*

The distance histogram shows that nearly all predictions fall under 5 pixels from the ground truth, reinforcing the model's ability to localize transients accurately. Additionally, the angle histogram confirms that there is no angular bias in the model's predictions—the model shows no preference for any particular direction when misidentifying the transients, which indicates uniform accuracy across all angular orientations. The polar chart of angles versus distances, with red dots representing the most accurate predictions, further supports this observation. The majority of predictions cluster around small distances with no angular pattern, demonstrating the model's consistency in localization. This chart was especially useful in batches when noticing the spread getting tighter.

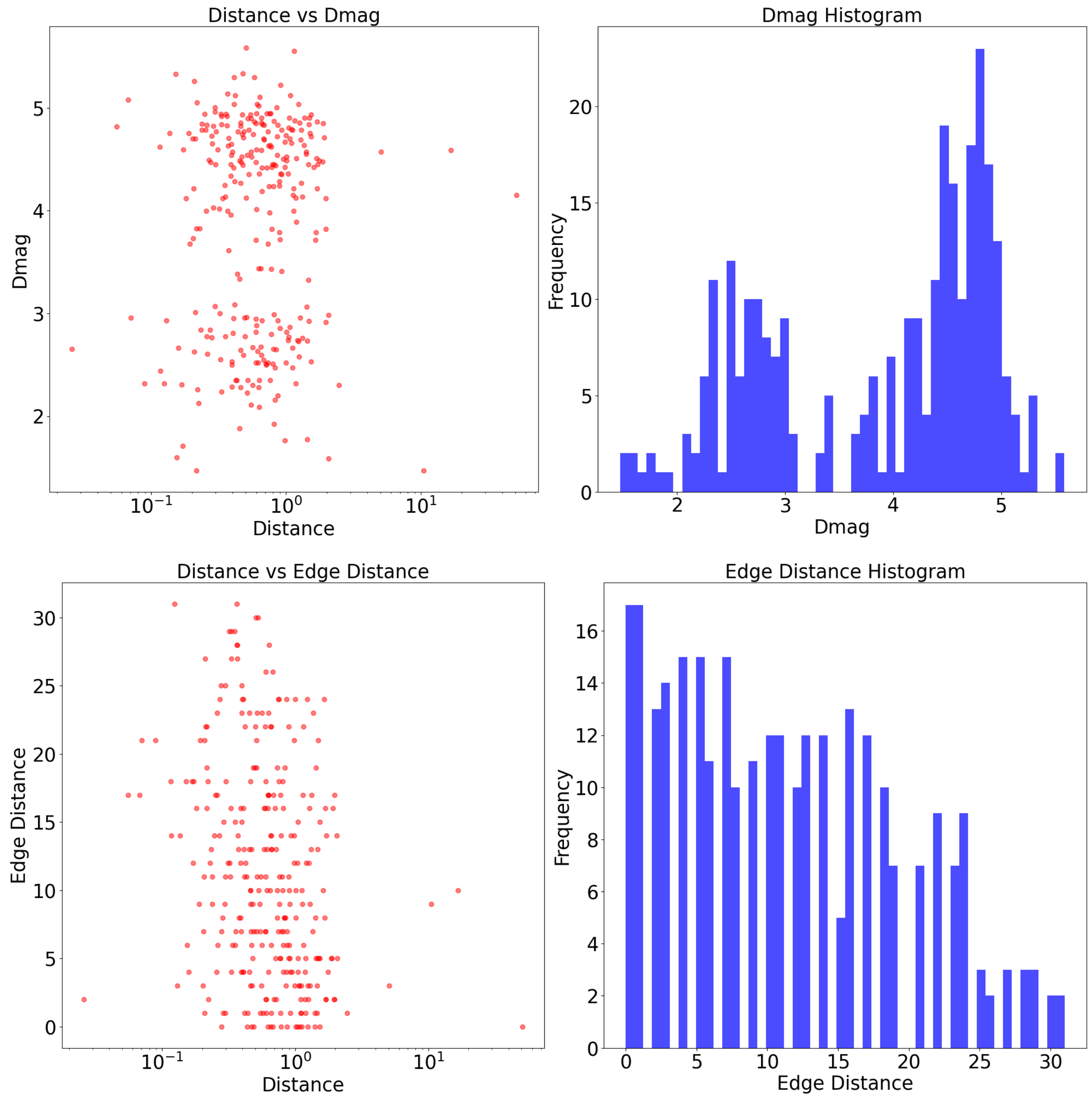


Figure 6. *Histograms representing the relationship between edge distance and delta magnitude with the distance of guesses. The x axis is in logarithmic scale to show distance's relationship with delta magnitude and edge distance.*

The relationship between distance and delta magnitude (dmag) was plotted, revealing that most predictions fall within the 0-5 dmag range. This indicates that the model is particularly accurate at identifying transients with moderate brightness differences. However, the dmag histogram suggests a limitation in detecting transients with low magnitude differences (fainter sources), where the model's performance diminishes. This is a critical insight for further refinement, as enhancing the model's sensitivity to faint transients will be key to improving its applicability to more challenging datasets. Alternatively, the distance vs edge test examines the accuracy of transient identification based on proximity to the image edge. The results show a slight correlation between the distance of the transient from the edge and the prediction error. Transients closer to the edge tend to exhibit slightly higher prediction errors, likely due to fewer surrounding pixels for context, which impacts the model's ability to accurately localize the source. While the effect is minor, it suggests a potential area for improving edge transient detection.

Robustness

To verify the robustness of our model, we conducted a series of experiments wherein each image in the test dataset underwent five augmentations (horizontal flip, vertical flip, 90, 180, 270 rotation). These augmentations included transformations such as rotation, flipping, and scaling, allowing us to assess the model's performance under varied conditions but with essentially the same image. For each original image and its augmentations, we calculated the median distance from the truth coordinates for each prediction. This process involved compiling results from the original image alongside its augmented versions and plotting these distances with standard deviation (STD) as error bars.

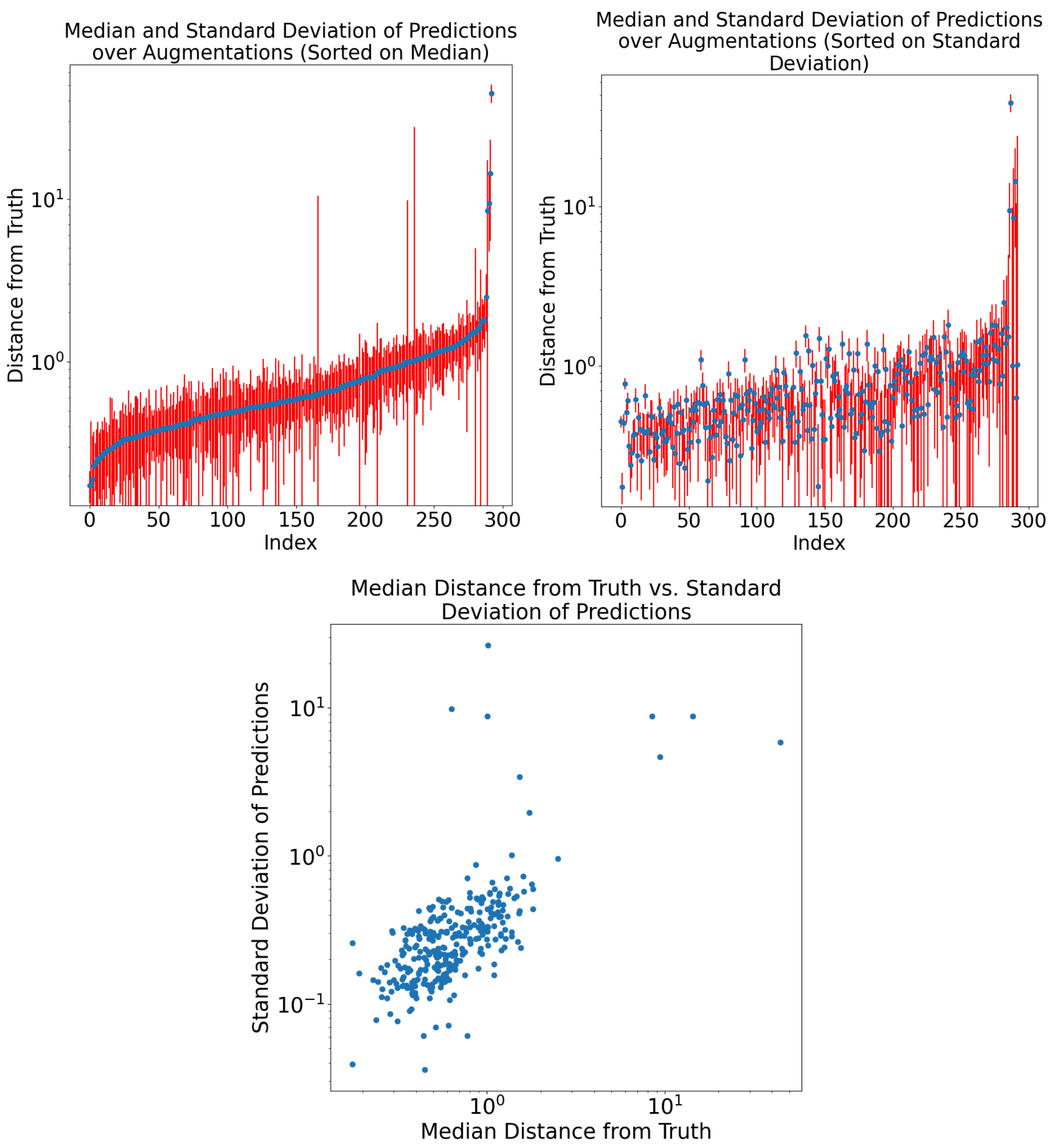


Figure 7. *Plots of the relationship between median distance and standard deviation to see if it follows an identifiable trend.*

In analyzing the distance from truth versus the index of guesses—sorted by median distance—we identified several outliers in the chart. These outliers exhibited significantly larger standard deviation lines, indicating higher variability in the predictions associated with these images. This variance suggests that these specific cases merit further examination to determine whether there are distinct characteristics within the images that contributed to the model's less accurate predictions. By sorting the plot by standard deviation rather than median distance, we effectively highlighted the instances where the model's performance fluctuated, revealing that some images consistently produced more erratic results.

We also plotted median distance from truth against standard deviation, which yielded a cluster concentrated in the bottom left corner of the graph. This observation indicates a clear relationship between the two metrics: lower median distances from the truth are associated with lower standard deviations. This clustering suggests that as the model's predictions become more accurate (i.e., the median distance decreases), the variability in those predictions also diminishes. This finding reinforces the notion that the model is robust in its capability to produce consistent predictions when applied to images that are not overly complex or ambiguous. Overall, these analyses provide valuable insights into the model's performance and identify areas for potential improvement, particularly in dealing with challenging image conditions.

Future Work and Challenges

Future work on this project will focus on addressing several critical challenges to enhance the model's capabilities in detecting and characterizing transients. One significant limitation of the current dataset is the visibility of transients; many are obscured by brighter objects or overlap with other transients, making them difficult to detect. A more comprehensive dataset that ensures all transients are visible, regardless of their luminosity or position, will be essential for improving the model's accuracy and robustness.

In addition to refining the dataset, the next step in our research involves calculating the magnitudes of the identified transients. Accurately determining the magnitude will provide insights into the intrinsic brightness of these objects, allowing for better classification and understanding of their nature. Furthermore, developing algorithms to identify the specific types of transients—such as supernovae, variable stars, or gravitational wave events—will be a vital component of future work.

Overall, while the current model demonstrates promising capabilities in pinpointing transient locations, optimizing its performance through iterative adjustments and leveraging larger, higher-quality datasets will significantly enhance its output. By addressing these challenges, we aim to develop a more robust framework for transient detection that can be applied in large-scale astronomical surveys, thereby contributing to our understanding of these elusive phenomena. This project is novel in its application to the upcoming Roman Space Telescope, which has not yet been launched, providing a unique opportunity to develop and refine algorithms tailored for its anticipated observational capabilities. As we prepare for the launch, it is crucial to recognize that the real space environment will introduce challenges and variables that were not fully captured in our simulation data. Consequently, our work must be an ongoing endeavor, continually adapting to the complexities of actual astronomical observations. By proactively addressing as many potential issues as possible now, we aim to mitigate the need for significant adjustments post-launch, ensuring that our model is well-prepared to navigate the dynamic and often unpredictable conditions of space imaging.

References

[1] Birla, D. (2021, January 20). *Autoencoders*. Medium. https://medium.com/@birla.deepak26/autoencoders-76bb49ae6a8f

[2] Duev, D. A., Mahabal, A. et al. (2019). Real-bogus classification for the Zwicky Transient Facility using deep learning. *Monthly Notices of the Royal Astronomical Society, 489*(3), 3582–3590. https://doi.org/10.1093/mnras/stz2357

[3] Lundberg, S. (n.d.). *SHAP (SHapley Additive exPlanations) documentation*. Retrieved September 27, 2024, from https://shap.readthedocs.io/en/latest/index.html

[4] Perrin, M. *Roman Space Telescope PSF simulation documentation*. (n.d.). In *WebbPSF*. Retrieved September 27, 2024, from https://webbpsf.readthedocs.io/en/stable/roman.html

[5] Sedaghat, N., & Mahabal, A. (2018). Effective image differencing with convolutional neural networks for real-time transient hunting. *Monthly Notices of the Royal Astronomical Society, 476*(4), 5365–5376. **https://doi.org/10.1093/mnras/sty613**

[6] Weights & Biases. (n.d.). *Weights & Biases documentation*. Retrieved September 27, 2024, from https://docs.wandb.ai/

Acknowledgments
I would like to thank Dr. Ashish Mahabal for his mentorship throughout this project, and my associate mentor Dr. Roberta Paladini along with members of the RAPID team at Caltech, specifically Jacob Jencson for technical support with the datasets. I also thank my sponsor Dr. Taylor Lawrence and the SURF program for financial support.

Appendix
It is necessary to mention that prior to the CNN, there was extensive work done in attempt to create an autoencoder that achieved the same output of a coordinate, but instead recreating the image with solely the transient. This would have been accomplished using a model (Fig 8) that would pass input pairs through a series of encoder layers much like a CNN, then flatten the output from the encoder in a bottleneck. Once these latent points are created, they can be upsampled through several layers of the decoder therefore producing a reconstructed image of just a transient [1].

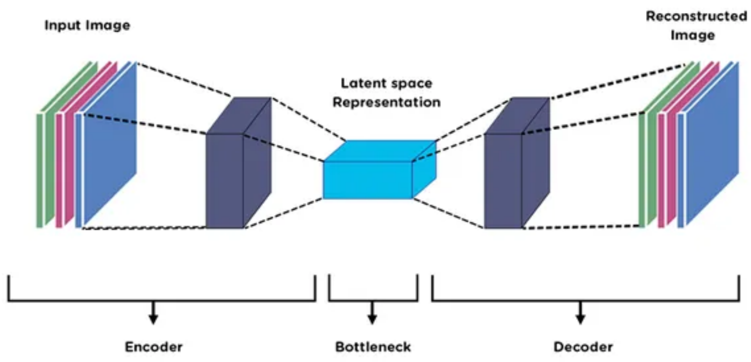


Figure 8. *Visual representation of a general autoencoder with encoder, bottleneck, and decoder.*

The truth images for such a model were generated using a Point Spread Function (PSF) on a blank image at each transient location for the pairs using webbpsf [4]. Initial development of PSFs had the shape of an "L" but further iterations of blurring and pixel dispersion allowed for a rounder, more blob-like transient. This process was applied for every pair such that each pair had a corresponding truth image.

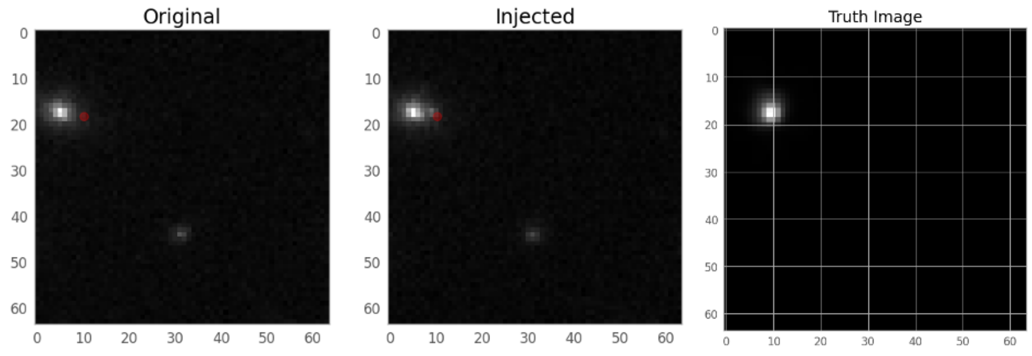


Figure 9. Red dot identifies the location of the coordinate. Due to scaling the truth image transient looks brighter than its true value.

This model did not provide helpful output, with each recreated image just reproducing the original and injected image. This was an unsuccessful attempt but will be revisited now that we have been supplied with cleaner and more visible transients that were used for the CNN.