STScI | SPACE TELESCOPE SCIENCE INSTITUTE

Instrument Science Report WFC3 2021-08

WFC3 IR Blob Classification with Machine Learning

F. Dauphin, J. V. Medina, P. R. McCullough

June 23, 2021

Abstract

IR blobs are small, circular, dark artifacts in WFC3 IR images caused by particulates that occasionally are deposited on a flat mirror that is nearly optically conjugate to the IR detector. Machine learning can potentially reduce the effort currently devoted to visually inspecting blobs. We describe how machine learning (ML) techniques have been implemented to develop software that will automatically find new IR blobs and notify the WFC3 Quicklook team. This report describes the data preparation, development of the ML model, and criteria for success. The results of our latest test cases demonstrate that the model finds blobs reliably, with the model correctly classifying blob and non-blob images 94% and 88% of the time, respectively. We also report tips and lessons learned from our experience in machine learning as a result of this project.

1. Introduction

The WFC3 Channel Select Mechanism (CSM) determines which channel (UVIS or IR) incoming light will travel through (Figure 1). In one configuration the CSM allows light to pass directly to the UVIS channel and in the other configuration, the CSM places a flat mirror in the beam to reflect the light to the IR channel. The CSM switches between the two configurations by rotating an arm that supports the mirror. A bellows-style coupler is secured with metal pins epoxied to the two shafts of the motor actuator and the CSM arm. It is theorized that some of the dried epoxy occasionally cracks and/or is crushed into small particulates that are then deposited inside the instrument. If an epoxy particulate lands on the CSM mirror, it produces a (new) blob-like artifact in the IR images. Hereafter, "blobs" refer to the small, circular, dark artifacts in WFC3 IR images caused by the particulates that occasionally are deposited on the CSM mirror that is nearly optically conjugate to the IR detector (Pirzkal et al., 2010).

There are two main reasons for why we monitor blobs:

1. Blobs appear as small circular patches on the frame with a throughput about 10-15% lower than in the surrounding pixels. Each blob is of variable strength and size which makes them hard to correct for in IR images. As such, science pixels that are contaminated with blobs are no longer usable. Standard practice is to identify the affected pixels, mask them, and overcome them by combining dithered exposures.

2. Insofar as new blobs may be a symptom of newly broken epoxy material which, if true, could impact the CSM's performance, monitoring for blobs is a high-priority activity. Any significant changes would be cause for an STScI/GSFC engineering and operations anomaly review board evaluation.

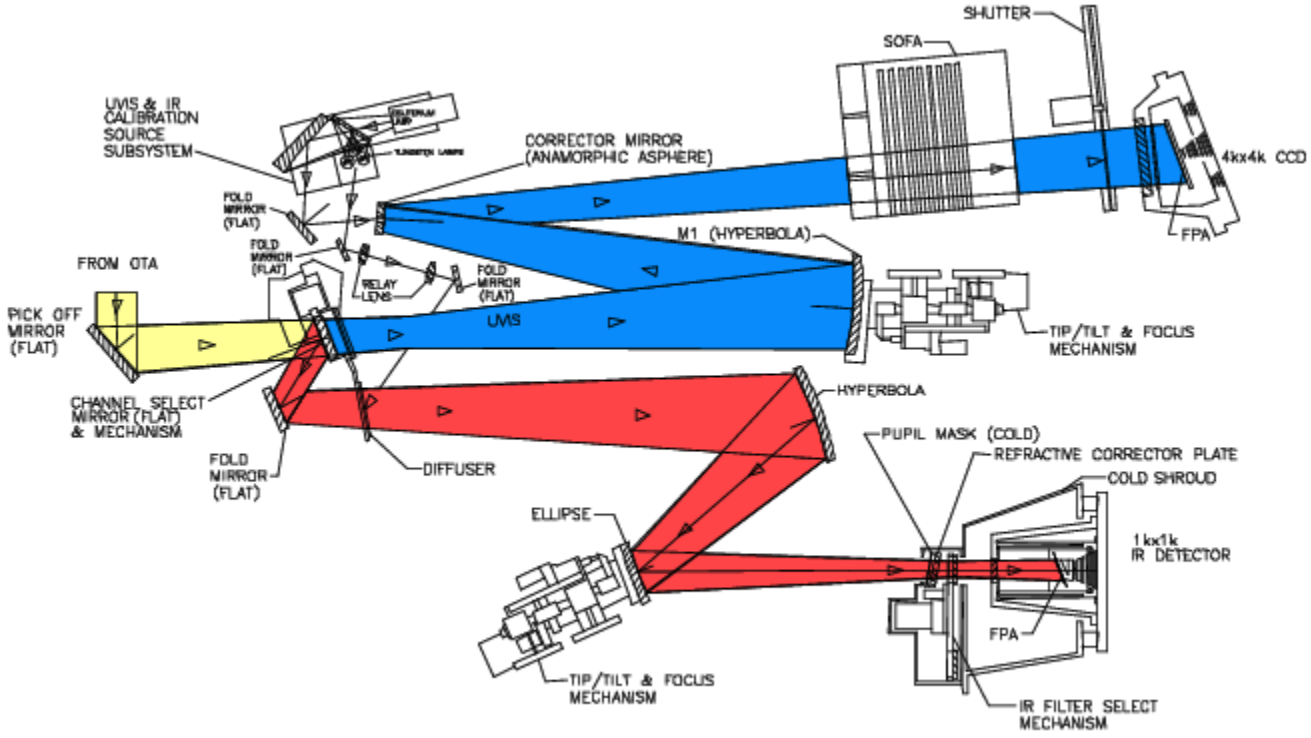

Fig. 1.—*A diagram of the WFC3 instrument's optics, showing the direction of incoming flux depending on which channel is selected (Dressel et al., 2019). Incoming light reflects off the pick off mirror and into either the UVIS channel (blue) or the IR channel (red). For IR, the incoming light is reflected off a flat mirror positioned into the beam by the CSM.*

2. Blob Monitoring: The Current Method

The CSM monitoring program is a routine WFC3 calibration program for the purpose of measuring offset/variability of the CSM angle over time, as well as to keep track of the current "map" of blobs that appear on the IR detector. The program consists of observations taken of dark-Earth airglow in F153M (Sunnquist, 2018) to create a smooth illuminated background across the IR detector that makes it easier to find and measure new blobs. Currently the method for checking for incoming IR blobs is to perform visual inspections of these blob images as they come in. The blob is then recorded in our system and measured for strength using an internal metric to produce a new IR bad pixel table for users.

WFC3's internal Quicklook system looks for new CSM monitor images on a daily basis and pre-processes them to create the final `blob.fits` files. These are the images used to track and record new IR blobs. Contents on the CSM monitor webpage include the most up-to-date version of the IR blob map (see Figure 2 - left), as well as a blob difference image (see Figure 2 - right) which is calculated by taking the most recent blob image and subtracting it with a median stack of the preceding 30 blob images taken from that cycle. The idea is that if no new blob has appeared, the latest image will look nearly identical to the preceding 30 blob images, so the difference will be zero plus or minus noise. However, if a new blob appears on the IR detector, subtracting the first image in which it appears from a median stack of the last 30 images in which it did not, will leave only the *newest* blob in the frame. Nominally the *image difference* will be input to the trained blob-classification model.

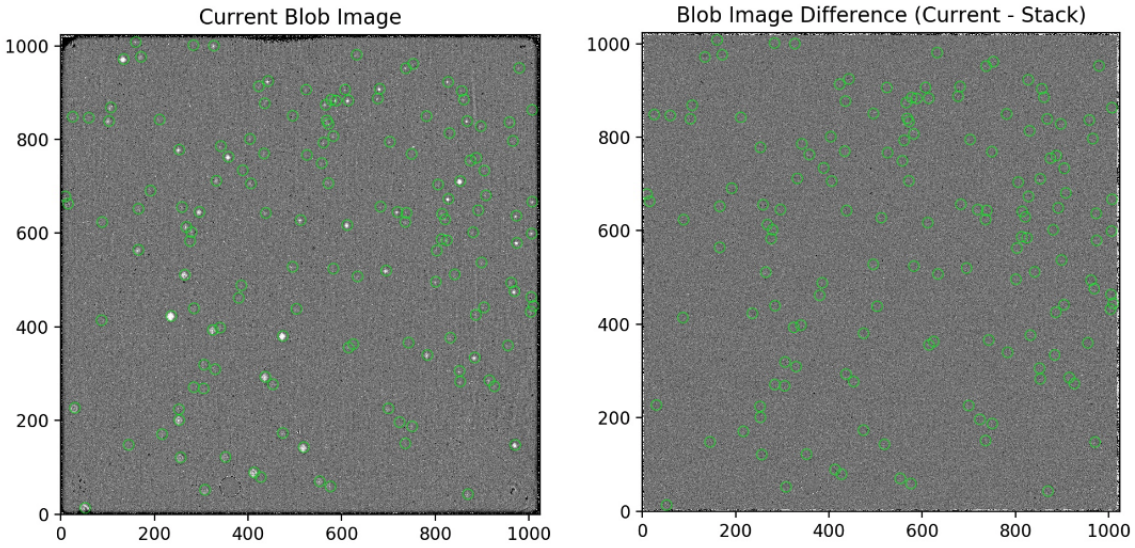

Fig. 2.—*Left: A map of every blob (white dots) from the most current blob image from the CSM monitor program. Right: A difference image that will only display a blob if it has not appeared within the last ≈30 days. The green circles denote blobs that have previously been found and recorded. These circles are identical in each panel. They are not part of the calibration image, but rather laid on top to be used as a point of reference.*

The blob classification model will ingest these blob difference images, and its prior knowledge of what a blob looks like will allow it to determine whether or not there is a new blob that needs to be recorded and measured. From there, the classification routine's integration into the WFC3 Quicklook system will allow it to notify the WFC3 team of a potential new blob along with its pixel coordinates. Because the blob monitoring is currently done by eye, using a model for blob searching will reduce the effort required to perform visual inspections on these processed images, as well as increase the accuracy of blob detection, potentially detecting blobs overlooked by the human eye.

3. Data Preparation

Because the model was trained using the supervised learning* [1] technique, the training set* was separated into two groups of labeled data: Blob images (labeled as 1) and non-blob images (labeled as 0). It is important to note that the blob images used for training will be blob difference images with blobs superimposed on them rather than single `blob.fits` files. This is to ensure that both the blob and non-blob images being trained on are within the same data domain. The data preparation is the same for both the blob and non-blob images and can be broken down into four basic steps. The following subsections outline each of these steps.

3.1. Outlier Clipping

We begin by taking a blob difference image that contains no new blobs (hereafter "non-blob" image) as identified by a human verifier. Figure 3 shows the pixel distribution of our example image, with outliers that can be attributed to the reference pixels on the edges of the detector, as well as bad pixels. To calculate our clipping thresholds, we first extract the z_{min} and z_{max} of the image using the `ginga.utils zscale` tool (Jeschke, 2017). These values are then multiplied by a factor F to determine the minimum and maximum clipping threshold. It can be empirically shown that a factor of 0.5, which we used, contains about +/- 3σ of the data centered around the mean. All values outside of the thresholds are clipped to a minimum or maximum value.

[1]Terms marked with a * have their definitions listed in the glossary under the Appendix of this report.

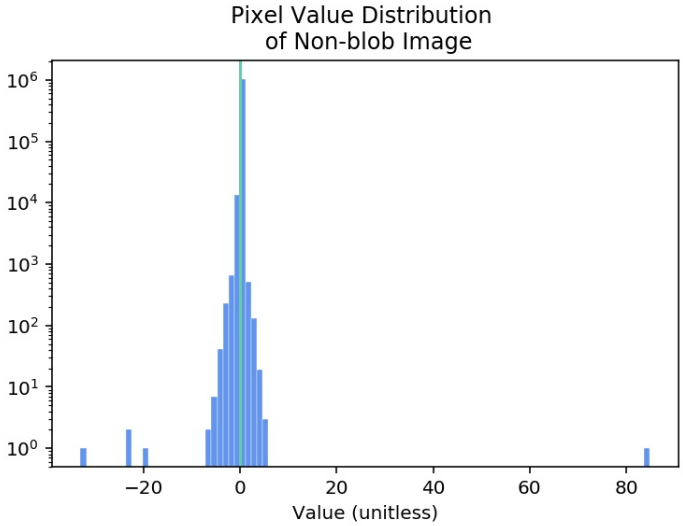

Fig. 3.—*Pixel value distribution of an unclipped, unscaled non-blob image (or blob difference image containing no new blobs). Outliers may be attributed to the edges of the image that were not removed in pre-processing, as well as bad pixels that were not filtered through the median stacking.*

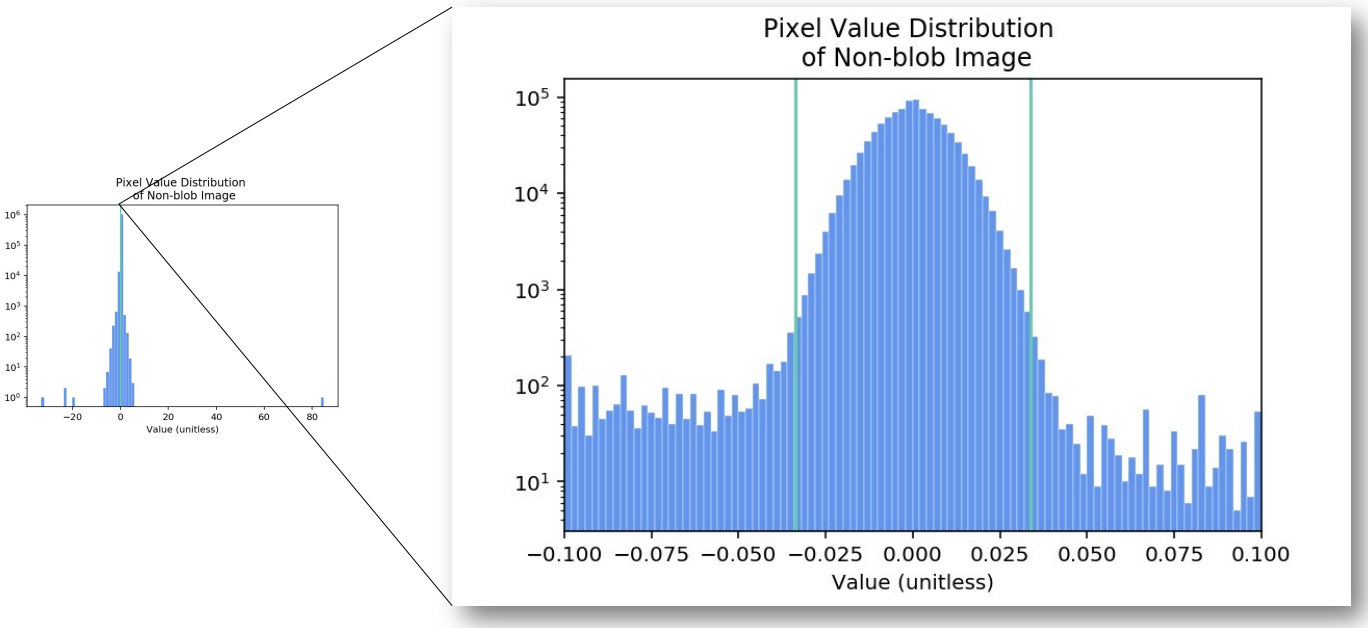

Fig. 4.—*Pixel value distribution of the previous image, now zoomed in to show the minimum and maximum clipping thresholds determined by the clipping factor multiplied by* z_{min} *and* z_{max}*, respectively. The clipping thresholds are denoted by the vertical lines.*

3.2. Feature Scaling

Once the major outliers are clipped, the next step is to perform z-scale normalization on the image. Z-scale normalization is a common normalization technique used in machine learning, intended to normalize the pixels so that their values are changed to represent the number of standard deviations they are away from the mean. For our case, z-scale normalization can be expressed with the following formula:

$$Scaled\ Pixel = \frac{Pixel - Clipped\ Mean\ Pixel}{Standard\ Deviation}$$

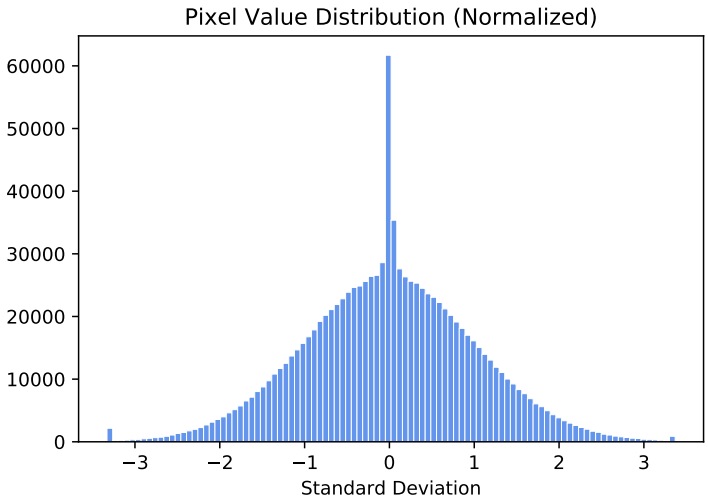

Fig. 5.—*Pixel distribution of the example image after clipping and z-scale normalization. The pixel value for each pixel now represents the number of standard deviations it is away from the mean of the distribution. Lastly, some pixels contain non-finite values as a result of the blob image pre-processing, and are thus replaced by the median because the model is only able to work with finite values.*

3.3. Image Cutouts

Once the entire image has been clipped for outliers and scaled, 256×256 subframes are taken from the left 75% of the full 1024×1024 image in random locations (while the right 25% of the image is used for validation). Rather than using entire blob images for the training and validation sets, subframes are fed into it for a few reasons: It allows us to preserve data for future testing and validation runs, so a single dataset can effectively return N number of images, with N being the number of subframes taken out. It also improves computational efficiency, as the blob subframes are a fraction of the entire 1024×1024 image. For the case of the blob images, this is when the blobs are superimposed onto the subframes. The blobs are blobs taken from a median stack of 30 real blob images, superimposed in random locations on the designated subframe. Figure 6 shows an example of a 256×256 non-blob subframe before and after the superimposed blobs are implemented.

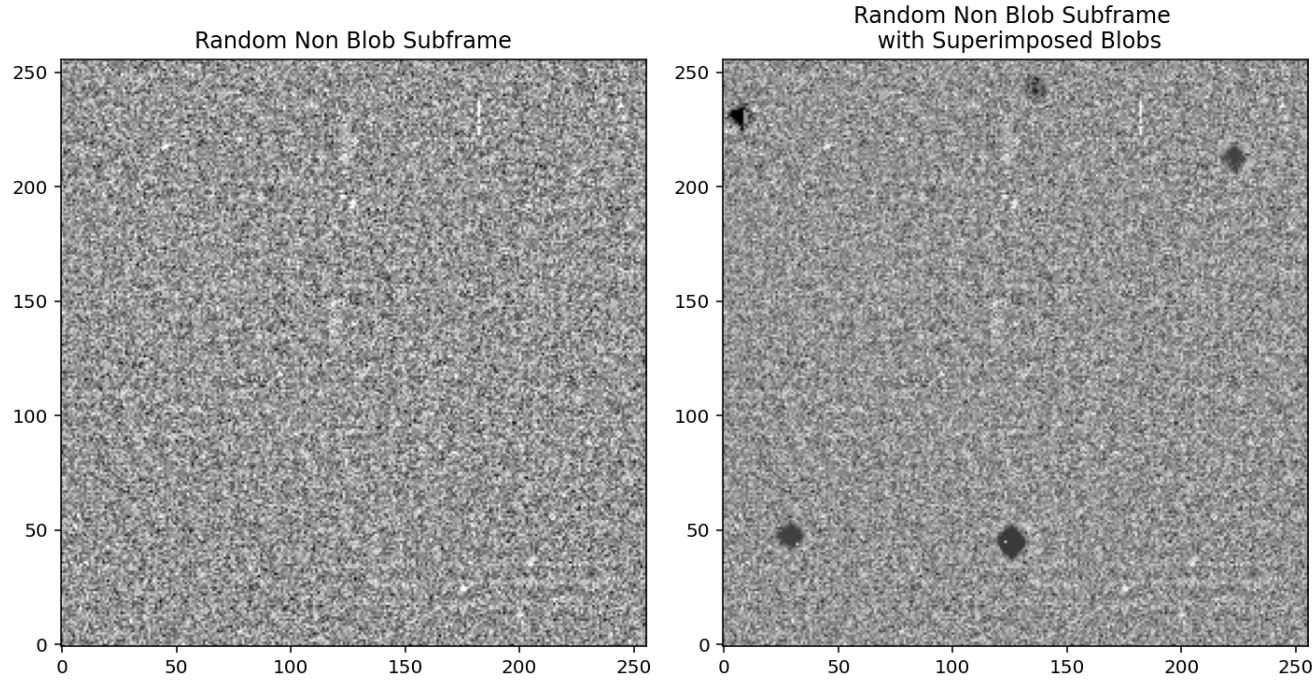

Fig. 6.—*An example of a non-blob subframe before and after the blobs are superimposed. The number of blobs that are placed on the subframe is random, as well as their physical characteristics such as size, strength, and location.*

3.4. Data Augmentation

Data augmentation, or creating new data by slightly modifying existing data, is critical in making the model more robust and preventing over-correction. The way in which we augment the data starts with the subframes: The subframes are taken in arbitrarily selected locations on the IR camera, to incorporate an element of randomness to the data. While there is only a limited number of blobs to work with, the following steps help to prevent over-correction:

Step 1: Once a subframe is taken, a level of noise is added at a normal distribution with a mean of 0 and spread of 0.75.
Step 2: The subframe is then randomly rotated by 0^o, 90^o, 180^o, or 270^o.
Step 3: Finally, it is randomly flipped (or not) across a random axis (X and/or Y).

The added noise, rotation, and flipping of the subframe add elements of randomness that make all the subframes different from one another, and thus mitigates over-correction.

4. Designing the Model

We chose a convolutional neural network* (CNN) to classify our images. A CNN, popularized in 2012 with AlexNet (Krizhevsky et. al 2012.), is a machine learning algorithm that specializes in computer vision. The model is built using convolutional layers and fully connected layers to extract important features from the images, making this model type ideal for blob observations. A convolutional layer uses kernels* to scan through input images and convolve a block of pixels (determined by the size of the kernel) to an output feature map. A fully connected layer takes input feature maps and connects them to output neurons* in a neural network. In the convolutional blocks, filters perform convolutions, the ReLU activation function* is applied, and max pool is used to condense the feature maps (see glossary for definitions). At the end of the convolutional blocks, the extracted features pass through fully connected layers. The network uses backpropagation to converge the CNN toward a minimum loss and learns important features for image classification through this process. At the end of the fully connected layers are two neurons, one for each classification of blob or non-blob. The final output neurons are the probabilities of each classification, from which the neuron with the max probability is the classification.

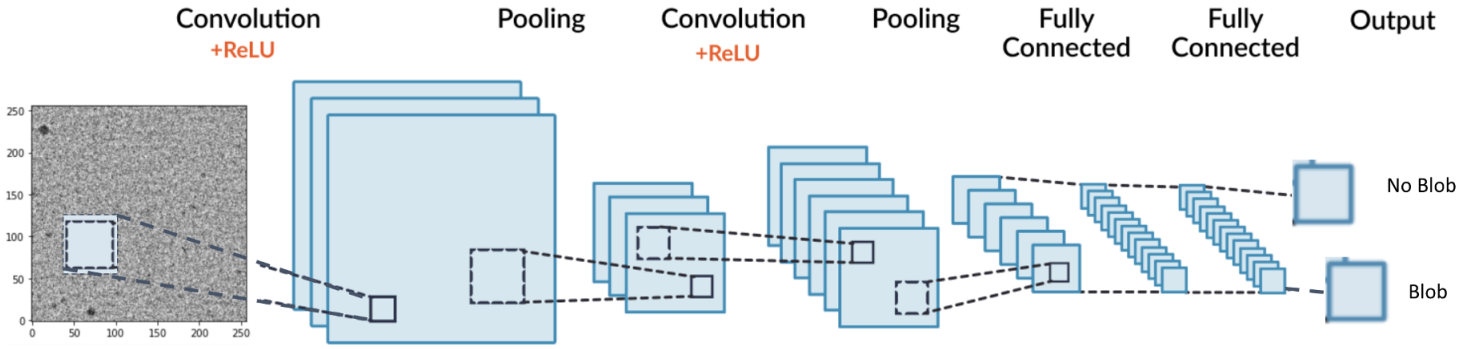

Fig. 7.—*Convolutional Neural Network Architecture. A 256x256 subframe passes through the network to be classified as having a blob on the subframe or not (LotusLabs, 2020).*

We chose a 2-layer CNN with 8 and 16 filters consecutively, 5×5 kernel and 1 pixel stride. We also used 2×2 zero padding, which adds extra zero pixels on the borders of our inputs to ensure the inputs and outputs are the same size. At the end of each convolutional layer, we used 4×4 max pooling* to condense the inputs, forcing the model to learn and vastly decreasing model size. After the second layer, we flatten the last feature maps to use as inputs to form two fully connected layers with output sizes of 128 neurons and 2 neurons. We implemented 0.15 and 0.3 dropout regularization at both fully connected layers to reduce overfitting. To introduce non-linearity, we chose the rectified linear unit (ReLU)* as an activation function as it empirically outperforms other activation functions in a majority of cases.

We sparingly tuned our hyperparameters in order to conserve training time, as exhaustively searching through the hyperparameter space would be taxing on the processor. We constructed a classifier using the Modified National Institute of Standards and Technology (MNIST)* handwritten digits database (Cohen et al., 2017) to gain intuition of how to construct our CNN. Even though three layers should perform better because there are more trainable parameters, the decrease in cross entropy* loss and increase in metrics were insufficient to compensate for increased complexity and training duration. For that reason, we chose a two-layer model. The dropout rates were adopted from MNIST hyperparameter tuning. The filter, kernel, max pool and neuron sizes are large enough for the model to learn important abstract features useful for classification and small enough to learn in a timely manner, approximately four hours. We can reduce the training time given better computational resources.

5. Training the Model

Batch* size is the number of samples the model uses to update its parameters once. Epochs* represent the number of times a model trains through an entire training set. Both of these parameters determine the total number of times the model will update. A batch size of 100 with 200 epochs sufficed for training the model. As a rule of thumb, choosing a batch size approximately 1/100th the size of the training set allows for the model to update around 100 times per epoch. With a training size of 16000 samples, the model updates 32000 times in training.

We used cross entropy as our loss function to optimize. This loss function interprets the output neurons as probabilities by assigning the probability of a classification to each neuron. Stochastic Gradient Descent (SGD)* and Adaptive Moment Estimation (Adam) are two iterative methods we used to optimize training. SGD finds the negative gradient of the loss function and uses the learning rate to slowly converge to a well-defined minimum. Adam implements a per parameter learning rate that adapts based on how fast the gradient converges and quickly converges to a well-approximated minimum. We utilized Adam to optimize our model for 5 epochs to find a local minimum, then use SGD to converge through the local minimum. By using Adam as a "head start" and finishing off with SGD, we achieved a well converged solution in a reasonable amount of time.

6. Results

The confusion matrix (Figure 8) summarizes the performance of our model on the validation data. Ideally, the confusion matrix is the identity matrix, i.e. its diagonal is unity and the off diagonals are zero, indicating a perfect classifier. However, this is unrealistic as a perfect classifier most likely overfits, or memorizes the training/validation set*. In practice, we aim for our model to produce a confusion matrix as close as practical to the identity matrix through model selection, design, and hyperparameter tuning. The true negative (correctly classified non blob subframes) and true positive (correctly classified blob subframes) rates are 0.88 and 0.94. The false positive (misclassified non-blob subframes as blobs) and negatives (misclassified blob subframes as non-blobs) are 0.12 and 0.06. Our model performs fairly well against the validation set. The majority of false positive samples contain "dips", which occur when the difference image is off by a few pixels.[2] Since the dips have blob structure to them, our model catches interest in it and misclassifies them as blobs. However, we are able to remove dips in the data preparation phase, meaning this rate will vastly decrease in practice. The majority of the false negatives are either small, faint, or off the subframe. These qualities would confuse a human so it is reassuring that the model's performance is comparable to a human. For testing, we used real samples of difference images during a blob finding period. The model's performance is comparable to that represented in the confusion matrix. We find that the false negative cases are comparable to those produced by visual inspection of a trained expert, and therefore is not a limitation of the network.

[2]Due to slight non-repeatability of positioning by the CSM, each blob image can be slightly mis-registered with respect to others. If the mis-registration is more than a small fraction of a pixel, it will produce noticeable residuals on either side of each blob.

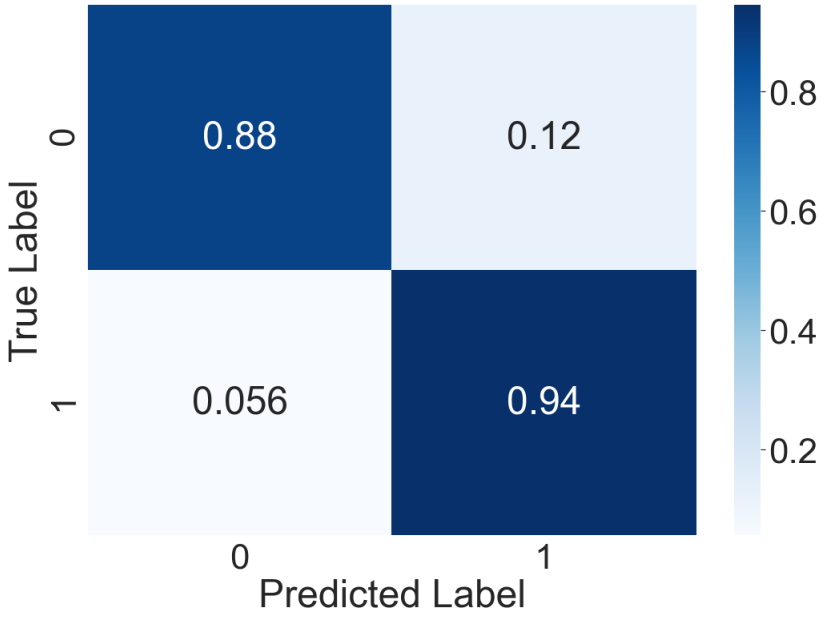

Fig. 8.—*CNN Confusion Matrix. The diagonal elements are the true negative (0.88) and positive (0.94) rates, which illustrate how well our model correctly classifies subframes. The closer the diagonals are to unity, the better the classifier is. 0 is for non-blob subframes and 1 is for blob subframes. The off-diagonal elements are the false positive (0.12) and false negative (0.056) rates, which illustrate how often our model misclassifies. The false positives mostly contain dips and the false negatives contain small blobs, faint blobs, or blobs that are near the edge of the image.*

Classification Type	**Rate**
True Negative	0.88
False Positive	0.12
True Positive	0.94
False Negative	0.056

Table 1:Results from the confusion matrix in table form.

Saliency maps are an evaluation technique used in computer vision that highlights which pixels the model uses for classification. When applying to saliency maps to non-blob frames, it randomly highlights the subframe, indicating it did not detect any useful pixels. When applied to blob frames, the blobs are almost always highlighted, even when misclassified if

it was a harder sample, such as it being too small, faint, or off the frame. Although our model may not be confident in classifying some of the more difficult samples, the saliency map almost always identifies blob-like features (see Figure 9 for an example).

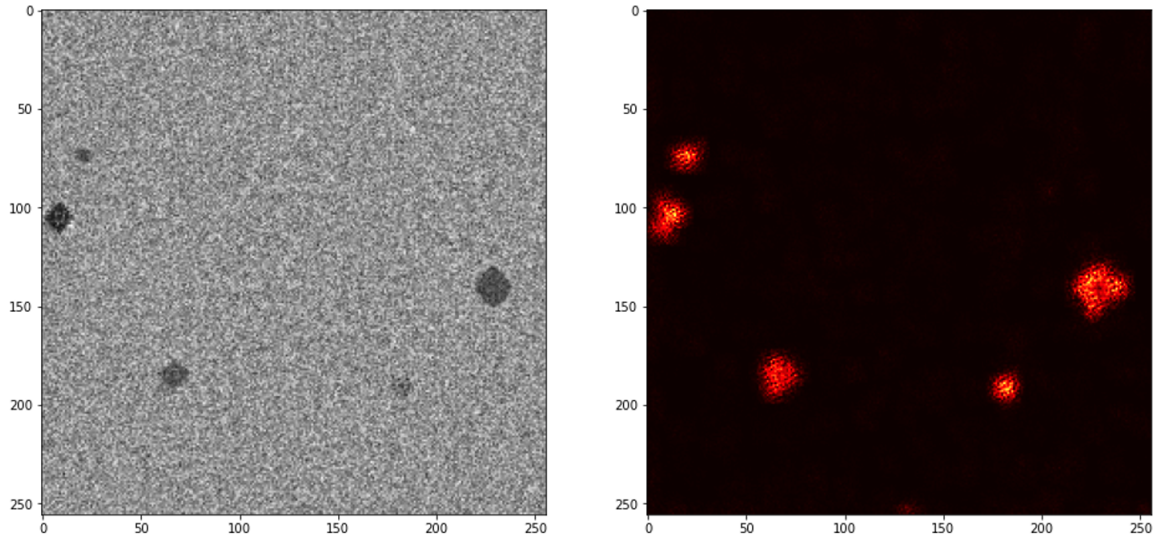

Fig. 9.—*Saliency Maps. The left panel is a subframe with five blobs superimposed. The right panel is the saliency map of the subframe, highlighting the location of the blobs. The model highlights pixels that it considered important for classification. The clear highlight of the input blobs indicates that our model is successful.*

7. Conclusions & Future Work

As a first version of automated blob detection, the results are successful and we plan to deploy the model into Quicklook. As an overview, every day the model will process the difference image, cut it into 16 subframes, and predict if there is a new blob or not for each subframe. It will log its prediction and saliency map of each subframe for periodic review by a team member. If a blob is detected, the team will be notified for further analysis and confirmation. Visual inspections will continue in parallel for a time to further vet the model's performance.

We plan to continuously improve our model through hyper parameter tuning and data exploration. As mentioned earlier, we were not able to tune our hyperparameters as thoroughly as we would have wanted. The tunable parameters include model depth, dropout rate, number of filters, kernel* size, max pooling size, number of neurons, batch size, and

epochs. The parameters we chose optimized training time on CPUs, but utilizing GPUs, which are around five times faster, will give us more leverage for possible grid or random search through hyperparameter space.

We also plan to try other algorithms to see how they compare to CNNs. Autoencoders can be trained for anomaly detection given specific types of training images. As mentioned earlier, it may be difficult to train for anomaly detection on non-blob subframes due to lack of any features. Training on blob subframes to detect non-blobs may also be difficult. Autoencoders perform well on training sets in which a majority of the image contains valuable information. Since blobs usually take up less than 1% of the image pixels, i.e. valuable information, it may be difficult for an autoencoder to detect the difference between blob and non-blob subframes with that small percentage of pixels.

We are also interested in utilizing dimensionality reduction techniques to decrease sample size and computational power. Algorithms such as principal component analysis (PCA), parametric t-distributed stochastic neighbor embedding (t-SNE), and uniform manifold approximation and projection (UMAP) embed high dimensional data into a lower dimension and has the potential to separate classes in the embedded space. Our inputs are 65536 pixels, where each pixel represents a varying dimension. If we can efficiently decrease the size of the inputs to a smaller embedded space and keep the classifications distinct, then our training time and model size can significantly decrease and possibly generalize better.

Lastly, one of the near-term future plans for the model involves performing a regression test that involves feeding it observed blob difference images in which one or more blobs have been found (previously from visual inspection) since the beginning of the CSM monitor program. There are currently over 100 such cases that can be used to assess the current state of the model, and determine if it is deployment-ready. After passing the regression test with sufficient accuracy, the blob classification model will be integrated into the Quicklook architecture and be part of the system's automated daily processes, where it will search for new blobs in the blob difference images as they appear.

Acknowledgements

We would like to thank Varun Bajaj, Vera Platais, Joel Green, and Sylvia Baggett for their revisions of this ISR. We would also like to acknowledge the weekly ML training provided by DSMO staff, especially Michelle Ntampaka, Josh Peek, and Josh Neuheisel. We thank them for the opportunity and also for specific consultation on this work.

References

Cohen, G., et al., 2017, "EMNIST: An Extension of MNIST to handwritten letters", arXiv 1702.05373 v1
https://arxiv.org/pdf/1702.05373v1.pdf

Dressel, L., et al., 2019, Wide Field Camera 3 Instrument Handbook for Cycle 27, Version 11.0 (Baltimore: STScI)
http://www.stsci.edu/hst/wfc3/documents/handbooks/currentIHB/wfc3_ihb.pdf

Durbin, M. J. & McCullough, P. R., 2015, "The Impact of Blobs on WFC3/IR Stellar Photometry", WFC3 ISR 2015-06 (Baltimore: STScI)
https://www.stsci.edu/files/live/sites/www/files/home/hst/instrumentation/wfc3/documentation/instrument-science-reports-isrs/_documents/2015/WFC3-2015-06.pdf

Jeschke, E., 2017, ginga ReadTheDocs v. 2.6.2, Revision c29f9214
https://ginga.readthedocs.io/en/v2.6.2/index.html

Krizhevsky, A. et al., 2012, "ImageNet Classification with Deep Convolutional Neural Networks"
https://papers.nips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

LotusLabs, 2020, "Intro to Convolutional Neural Networks and its Applications"
https://www.lotuslabs.ai/intro-to-convolutional-neural-networks-and-its-application

Mack, J., et al., 2021, "WFC3/IR Filter-Dependent Sky Flats", WFC3 ISR 2021-01 (Baltimore: STScI)
https://www.stsci.edu/files/live/sites/www/files/home/hst/instrumentation/wfc3/documentation/instrument-science-reports-isrs/_documents/2021/2021-01.pdf

McCullough, P. R., et al., 2014, "Infrared Blobs: Time-dependent Flags", WFC3 ISR 2014-21 (Baltimore: STScI)
https://www.stsci.edu/files/live/sites/www/files/home/hst/instrumentation/wfc3/documentation/instrument-science-reports-isrs/_documents/2014/WFC3-2014-21.pdf

Pirzkal, N., et al., 2010, "The WFC3 IR 'Blobs'", WFC3 ISR 2010-06 (Baltimore: STScI)
https://www.stsci.edu/files/live/sites/www/files/home/hst/instrumentation/wfc3/documentation/instrument-science-reports-isrs/_documents/2010/WFC3-2010-06.pdf

Pirzkal, N. & Hilbert, B., 2012, "The WFC3 IR 'Blobs' Monitoring", WFC3 ISR 2012-15 (Baltimore: STScI)
https://www.stsci.edu/files/live/sites/www/files/home/hst/instrumentation/wfc3/documentation/instrument-science-reports-isrs/_documents/2012/WFC3-2012-15.pdf

Russel, R. E. & McCullough, P. R., 2017, "Possible Overlaps Between Blobs, Grism Apertures, and Dithers", WFC3 ISR 2017-16 (Baltimore: STScI)
https://www.stsci.edu/files/live/sites/www/files/home/hst/instrumentation/wfc3/documentation/instrument-science-reports-isrs/_documents/2017/WFC3-2017-16.pdf

Sunnquist, B., 2018, "WFC3/IR Blob Monitoring", WFC3 ISR 2018-06 (Baltimore: STScI)
https://www.stsci.edu/files/live/sites/www/files/home/hst/instrumentation/wfc3/documentation/instrument-science-reports-isrs/_documents/2018/WFC3-2018-06.pdf

Sunnquist, B., et al. 2019, "Time-dependent WFC3/IR Bad Pixel Tables", WFC3 ISR 2019-03 (Baltimore: STScI)
https://www.stsci.edu/files/live/sites/www/files/home/hst/instrumentation/wfc3/documentation/instrument-science-reports-isrs/_documents/2019/WFC3-2019-03.pdf

8. Appendix

The following is a glossary of the machine learning terms used in this report to describe the model, its training process, and data preparation (and a few other general terms). The terms and their definitions are referenced from the Google Developers Machine Learning Glossary (**available here**). Each term is marked with an asterisks (*) at its first appearance in the report.

Activation function	A function that takes in the weighted sum of all of the inputs from the previous layer and then generates and passes an output value (typically nonlinear) to the next layer.
Batch	The set of examples used in one iteration (that is, one gradient update) of model training.
Convolution	In mathematics, casually speaking, a mixture of two functions. In machine learning, a convolution mixes the convolutional filter and the input matrix in order to train weights.
Cross-entropy	A generalization of Log Loss to multi-class classification problems. Cross-entropy quantifies the difference between two probability distributions. See also perplexity.
Drop out regularization	Dropout regularization works by removing a random selection of a fixed number of the units in a network layer for a single gradient step. The more units dropped out, the stronger the regularization. This is analogous to training the network to emulate an exponentially large ensemble of smaller networks.
Epoch	A full training pass over the entire dataset such that each example has been seen once. Thus, an epoch represents N/batch size training iterations, where N is the total number of examples.
Kernel	Also known as a convolutional filter, kernels convolve images and feature maps to extract useful features for modeling.

Feature set	The group of features your machine learning model trains on. For example, postal code, property size, and property condition might comprise a simple feature set for a model that predicts housing prices.
MNIST	MNIST is a canonical dataset for machine learning, often used to test new machine learning approaches. For details, see The MNIST Database of Handwritten Digits.
Neural Network	A model that, taking inspiration from the brain, is composed of layers (at least one of which is hidden) consisting of simple connected units or neurons followed by nonlinearities.
Neuron	A node in a neural network, typically taking in multiple input values and generating one output value.
Pooling	Reducing a matrix (or matrices) created by an earlier convolutional layer to a smaller matrix. Pooling usually involves taking either the maximum or average value across the pooled area.
ReLu	An activation function with the following rules: If input is negative or zero, output is 0. If input is positive, output is equal to input.
Stride	In a convolutional operation or pooling, the delta in each dimension of the next series of input slices.
Stochastic Gradient Descent (SGD)	A gradient descent algorithm in which the batch size is one. In other words, SGD relies on a single example chosen uniformly at random from a dataset to calculate an estimate of the gradient at each step.
Supervised learning	Training a model from input data and its corresponding labels. Supervised machine learning is analogous to a student learning a subject by studying a set of questions and their corresponding answers.
Training set	The subset of the dataset used to train a model.
Validation set	A subset of the dataset used in validation.