Draft version November 15, 2021
Typeset using LaTeX default style in AASTeX631

# Coronal Hole Detection using Machine Learning Techniques

Tamar Ervin,[1] Jacob Bortnik,[2] and Cooper Downs[3]

[1] *Department of Physics and Astronomy*
*University of California at Los Angeles*
*475 Portola Plaza, Los Angeles, CA 90095*
[2] *Department of Atmospheric and Oceanic Sciences*
*University of California at Los Angeles*
*Maths Science Building, 520 Portola Plaza, Los Angeles, CA 90095*
[3] *Predictive Science Inc.*
*9990 Mesa Rim Rd 170, San Diego, CA 92121*

## ABSTRACT

The detection and mapping of Coronal Holes (CH) in solar Extreme Ultraviolet (EUV) and X-Ray images is useful for a variety of scientific and space weather prediction applications. We build upon a previous pipeline for multi-spacecraft synchronic mapping and detection (Caplan et al. 2016), where long-term averages of EUVI/STEREO A/B (195 Å)[a)] and SDO/AIA (193 Å)[b)] images are used to compute data-derived corrections for center-to-limb variations in images and intensity differences among instruments. The calculation of these corrections has been greatly simplified through modern database storage and querying techniques implemented in the image processing pipeline. Full-Sun maps are built using a merge-mapping process which favors lower intensity data in areas of overlap, helping to mitigate the systematic issues of obscuration and evolution (Caplan et al. 2016). After image processing, CH regions are detected using two primary machine learning methods: a convolutional neural network (CNN) and K-means clustering. The CNN model is a supervised learning model for image segmentation while K-means is an unsupervised clustering model. These models both provide exceptional CH detections. Results are mapped on an image-by-image basis and merged to create EUV and CH maps. The merge-mapping process favors lower intensity data in areas of overlap, and an arbitrary number of images may be used, helping to mitigate the systematic issues of obscuration and evolution. With this flexible new framework and data-driven approach to CH detection, we explore methods for creating time-dependent coronal hole maps and evaluate their use for model comparisons.

*Keywords:* Coronal Holes — Data Processing — Convolutional Neural Network — K-Means Clustering

## 1. INTRODUCTION

The history of studying coronal holes dates back to the early 1900s. These dark regions on the Sun were first observed and named during solar eclipses in the 20th century. Astronomers noticed light rays emanating from the poles of the Sun, almost replicating the field lines surrounding a bar magnet. Similar rays were later noticed in the equatorial and mid-latitude regions of the sun, and these regions were then quantified and named. Coronal holes were re-discovered in the late 1960s to 1970s as dark areas in X-Ray and Ultraviolet (UV) images of the solar disk (S. 2009). Subsequent research showed correlation between the location of these dark regions and areas where magnetic fields opened into interplanetary space.

Coronal holes are regions of dark, low density plasma on the solar surface. They are the least active solar regions and are associated with expanding magnetic fields. The regions are a factor in the fast acceleration of the solar wind due to ionized atoms and electrons that flow along open the magnetic field lines. When measured in Extreme Ultraviolet

[a)] Courtesy of NASA/STEREO science teams.
[b)] Courtesy of NASA/SDO and the AIA, EVE, and HMI science teams.

(EUV) or X-Ray wavelengths, coronal holes are seen as dark, almost black, patches on the sun. They can also be observed with a coronagraph and during a solar eclipse show up as lower intensity regions (S. 2009). Solar coronal holes are often observed on the poles, especially during times of low solar activity. In times of higher solar activity, coronal holes show up at lower solar latitudes although with less temporal stability (S. 2009). These lower latitude coronal holes usually last only a few solar rotations before their magnetic configuration undergoes transformation.

In comparison with other magnetically dynamic regions of the Sun, coronal holes are simpler regions and thus often the starting point when modeling the corona. Understanding coronal holes and their evolution is important for our understanding of geomagnetic storms and how they affect Earth's atmosphere and magnetic field (S. 2009). Solar coronal holes are responsible for the high-speed solar wind and these are a primary component in geomagnetic storms. Additionally, coronal holes are pivotal in our understanding of the solar corona and its interactions with interplanetary space. Detecting and understanding oscillations of the solar corona is now a growing field in the search for a method to understand the coronal heating problem (Hegde et al. 2014). To create such oscillation detections, coronal holes must be detected and their boundaries can have an effect on the detected oscillations.

To understand and predict coronal holes, we must be able to detect them. This is a surprisingly difficult task due to a few main reasons. The first being the definition of a coronal hole. As previously described, coronal holes are often defined as dark regions in the solar corona. However, the boundaries of coronal holes are neither smooth nor uniform (S. 2009). Coronal hole boundaries vary in shape and sharpness and it is often difficult for observers to detect any intensity contrast between coronal holes and neighboring regions. Detecting and setting boundaries for coronal holes is thus difficult and the primary factor in the decision to take a machine learning approach to coronal hole detection.

To make accurate and meaningful coronal hole detections, a large amount of solar images are needed. The data used were from three different NASA instruments: SDO/AIA, STEREO-A/EUVI, and STEREO-B/EUVI. From each instrument, images were taken at a two-hour cadence from three different positions in space. This allows for a compilation of images to create full-Sun synchronic maps over a large timescale. The Atmospheric Imaging Assembly (AIA) in an instrument onboard the Solar Dynamics Observatory (SDO). It is an array of four normal-incidence reflecting telescopes that image the Sun in EUV and UV wavelengths (Boerner P. 2012). The Extreme Ultraviolet Imager (EUVI) is part of the instrument suite developed for the NASA STEREO mission. There are two identical telescopes on separate STEREO spacecraft that study the solar corona. EUVI has two viewing angles, ahead (STEREO-A) and behind (STEREO-B), along with a boost in image resolution in comparison to previous missions (Wuelser et al. 2004). For this project, the data used were 193 Å (EUV) data from the AIA instrument, and 195 Å (EUV) data from both STEREO spacecraft.

The data underwent extensive pre-processing analysis and corrections before being used for detection or mapping purposes. This is a primarily Python (Van Rossum & Drake 2009) based project and modules used include: `numpy` (Harris et al. 2020), `matplotlib` (Hunter 2007), `Tensorflow` (Abadi et al. 2015), and `scikit-learn` (Pedregosa et al. 2011). These pre-processing steps are described in more detail in section 2. The goal is to create uniform data across all three instruments such that CH detection is consistent between data from different spacecraft. Data processing removes center-to-limb intensity variations because CH detection relies heavily on intensity variation across images. It also creates a uniform intensity scale across all instruments by adjusting instrument intensity histograms to fit a chosen reference instrument.

Once processed, two different detection methods were applied and then full-Sun synchronic maps were produced. The first detection method is a convolutional neural network detailed in section 3. This is a supervised machine learning image segmentation model therefore the training data contains both the input (solar disk images) and corresponding label (CH detection mask). The second detection method is an unsupervised detection model: K-means clustering described in section 4. This model takes unlabeled data (solar disk images) and iteratively assigns each data point to a cluster based on image features. These clusters are then used to determine whether the pixel is a coronal hole or not. The advantage of an unsupervised method is that it relies entirely upon the machine's detection, with no recognition of an individual's definition of a coronal hole, thus removing human bias from the equation. Both of these models create beautiful coronal hole detections overlaid on solar maps and their accuracy and results are detailed in their corresponding sections.

## 2. DATA ANALYSIS

The data used for this project were solar disk images, however these images underwent extensive corrections before being used for coronal hole detection or map creation. Pre-processed images were stored in an HDF5 file system for

later querying. Here, long-term averages of EUVI/STEREO A/B (195 Å) and SDO/AIA (193 Å) images are used to compute data-derived corrections for center-to-limb variations in images and intensity differences among instruments. The calculation of these corrections has been greatly simplified through modern database storage and querying techniques implemented in the image processing pipeline. Each image has limb-brightening and inter-instrument intensity corrections applied before moving on to the creation of a segmentation mask.

### 2.1. *Limb-Brightening Correction*

Through the limb-brightening correction process, images were corrected for center-to-limb intensity variations. This was done through data-driven coefficient calculation of long term data. Since coronal holes are often defined as areas of lower intensity, it is imperative that intensity be uniform across the solar disk. The EUV emissions from the solar corona are optically thin meaning that path distance will affect intensity. A feature observed near the solar limb would thus appear less bright than that exact same feature observed at disk center. From the solar limb, the line of sight distance is further and thus resulting in lower intensity for that same feature which would effect the CH detection. The limb-brightening process seeks to correct this.

The time averaging of long term data is essential for the image processing pipeline to ensure fluidity of correction coefficients through time. Data is averaged over a six month period in correction coefficient calculation and calculated coefficients are then stored in the database with associated correction method and image information. This ensures efficient querying of coefficients when later building maps

The correction coefficients are calculated based their disk center equivalent. Limb brightening correction curves (LBCCs) are based on $\mu$ values. These $\mu$ values are defined such that

$$\mu = cos\,\theta,\ \theta\,\epsilon\,[0, \pi/2] \tag{1}$$

where $\theta$ is the angle from disk center ($\mu$=1) to solar limb ($\mu$=0).

The first step is the creation of 2D (intensity v. $\mu$) histograms. These histograms were created for each image over the entirety of the database. These histograms are then stored and tagged in the database with associated image information. Once all of these histograms have been calculated, the correction coefficient calculation can take place. The 2D histograms are queried from the database for the six-month time range in question and then averaged.

Assuming a spherically symmetric solar sphere, we are able to calculate the rate of intensity increase from solar limb to disk center. The calculation returns two coefficients: $y(\mu)$ and $\beta(\mu)$, both of which are $\mu$ dependent as seen in figure 1. The coefficients are calculated through least-squares optimization of normalized histograms at each $\mu$ value and the normalized disk-center histograms $\mathrm{psi}_p aper$.

The image transformation is non-linear, but linear in log space, such that $I_0$ is the $log_{10}$ of image intensity as seen in figure 2.

$$I_0 = \beta(\mu)I(\mu) + y(\mu) \tag{2}$$

The results of these calculation are $\beta(\mu)$ and $y(\mu)$ values that are similar between instruments, yet vary over time.

### 2.2. *Inter-Instrument Intensity Transformations*

Similar to the limb brightening correction process, the inter-instrument intensity transformation seeks to correct for intensity differences. This is important because intensity value plays a large role in CH detection and therefore the quality of the detection relies entirely upon the quality and uniformity of the data.

To create full-Sun maps, it is necessary to pull data from multiple instruments that view the Sun at the same time, yet from different angles. The three instruments used were SDO/AIA (193 Å), EUVI/STEREO A (195 Å), and EUVI/STEREO B (195 Å). The A and B denote which instrument is ahead and which behind. Different instruments have different rates of degradation and calibration differences that produce images with varying intensities. While CH detection on one disk image can occur after the limb-brightening correction, for the creation of any type of full-Sun map, it is imperative that intensities between different instruments are equated. This allows for the use of the same threshold values across all instruments in a purely intensity threshold based CH detection. Additionally, it allows for the creation of a uniform map, and to the naked eye it is often close to impossible to determine where one instrument's data starts and the other ends allowing for a more continuous picture.

The inter-instrument intensity correction coefficients were calculated following a similar time-averaging process as with the limb-brightening correction. In order to compute the inter instrument intensity correction, first the limb-brightening correction must be applied to the disk images. The limb-brightening $\beta$ and  values are queried from the

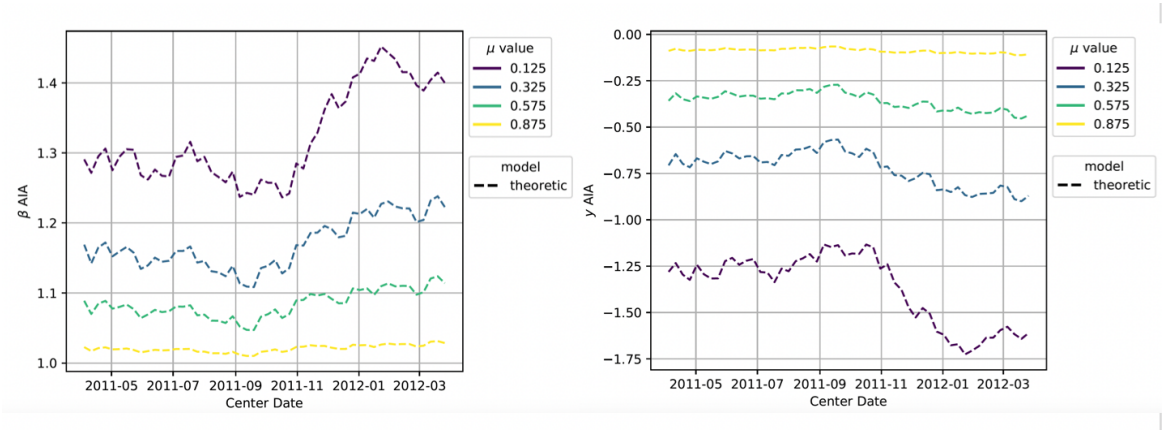


**Figure 1.** $\beta(\mu)$ and $y(\mu)$ limb-brightening correction coefficient values over time. These values are for the one-year moving average surrounding November 2011.

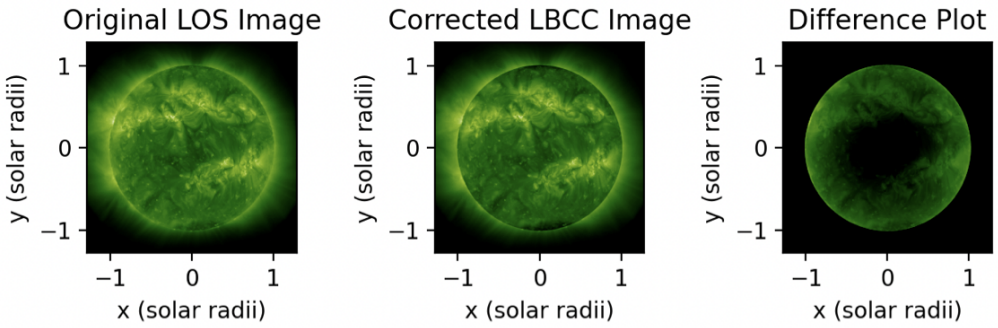


**Figure 2.** Comparison of Solar Disk Images before and after limb-brightening correction

database and applied to the original images following the previously defined non-linear transformation (Equation 2). Once the images have undergone limb-brightening correction, 1D (intensity v. $\mu$) histograms are calculated.

1D histograms were created for the entirety of the database images and these histograms were then stored in the database with the corresponding instrument and method information. This created a more efficient calculation process as now rather than continuously recalculating histograms, only a 1D array needed to be queried from the database.

For equating instrument intensities, it is important to ensure that each instrument is observing the same portion of the Sun. If one instrument had a portion of the Sun with an active region, while another instrument was observing quiet Sun, the intensity transformation was meaningless. To remedy this, rather than the purely time based six month span such as what was used in limb-brightening, a time span was taken based off Carrington rotation. Additionally, the latitude of disk data used was limited such that each instrument viewed the same region.

The necessary 1D histograms were queried from the database and then intensities were equated to a reference instrument. In this case, the reference instrument was EUVI-A. Correction coefficients of $\alpha$ and $x$ are saved in the database, where $x$ is the offset value.

The intensity correction was once again a non-linear (linear in log space) correction, identical to limb-brightening as seen in figure 3.

$$I_{ref} = \alpha I_0 + x \tag{3}$$

## 3. SUPERVISED MACHINE LEARNING DETECTION: CONVOLUTIONAL NEURAL NETWORK

### 3.1. *Data Preparation*

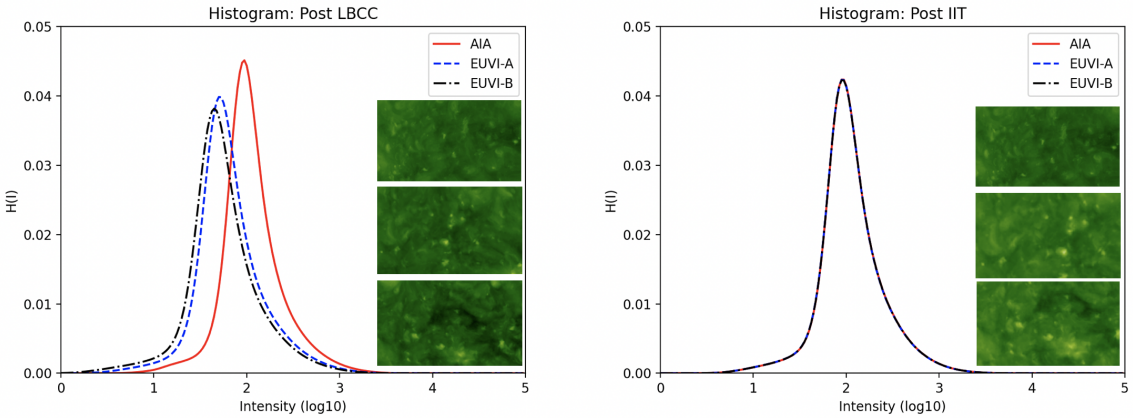


**Figure 3.** Comparison of 6-month time averaged, normalized intensity histograms before and after inter-instrument transformation. The histograms have 200 intensity bins and a latitude limit of $\pm 75.3°$. The intensity histograms for AIA and EUVI-B data have been equated to the EUVI-A histogram as seen in the post IIT plot.

In order to train a machine learning model, the data must be prepped such that it is compatible with the model in question. In this case, the TensorFlow package (Abadi et al. 2015) was used for building the model and thus TensorFlow compatible tensors were necessary to pass into the model.

For a supervised model, a set of training examples is mapped to an input-output pattern, then compared to the desired output. Since we were detecting coronal holes on solar disk images, this meant that to train the model disk images and a corresponding CH detection mask were needed. These coronal hole detection masks were created using Predictive Science's previously created two-threshold region growing algorithm (Caplan et al. 2016). This is a FORTRAN region growing algorithm that relies on a two-threshold method along with a pixel connectivity constraint to ensure coronal holes are not over detected. The algorithm ezseg, is freely available online and is based on intensity thresholding (Caplan et al. 2016). Due to the image pre-processing steps previously described, this algorithm is able to detect coronal holes without additional magnetic data.

The data used were solar disk images from the year 2011 at two hour cadence, with 3 images per timestamp (one from each instrument). Each image underwent the image pre-processing corrections and was then stored in a HDF5 file to decrease the amount of memory used when training the model. An additional advantage of the HDF5 format is it's ability to hierarchically structure the dataset and group together images and segmentation masks from the same timestamp. Coronal hole segmentation masks were created and stored in the HDF5 file for the entire data range that would be used in training this model.

The processed disk images and accompanying Coronal Hole detections were two-dimensional intensity arrays `(IMG_HEIGHT, IMG_WIDTH)`, however for the use in a CNN model, this data had to be converted to RGB (or RGB-A) arrays `(IMG_HEIGHT, IMG_WIDTH, N_CHANNELS)`. The conversion of this data from an intensity based array to RGB(-A) arrays was done using the `matplotlib.colors` (Hunter 2007) package.

Additionally, when training a supervised learning model, both training and validation data are needed. This is to ensure that the model is not drastically overfitting or underfitting the data when tested on data it has never seen before. To create a training and validation dataset, the original dataset from 2011 was split into training and validation datasets using the `sklearn.model_selection.train_test_split()` (Pedregosa et al. 2011) method. Once split, the data was converted into dictionaries of `image` and `segmentation_mask` for each datapoint. The tensorflow data library was used to convert the data into tensorflow compatible slices. `tf.data.Dataset.from_tensor_slices(dict)` to split the dictionaries into tensorflow compatible tensor slices to be used for training. This was the most time consuming pre-processing step, as converting numpy arrays to tensorflow compatible tensors is a computationally intensive process.

After being converted into tensor format, these datasets had to be mapped into the correct image size and normalized. Each dataset (training and validation) was mapped and normalized, such that it was the size of the input image the model desired.

### 3.2. *Model Creation and Training*

At its core, the model used was a U-Net model. This is an autoencoder model favored because of its ability to have a precise output, despite using fewer images. This model has a symmetric contraction and expansion path that give it the characteristic U-shape and name.

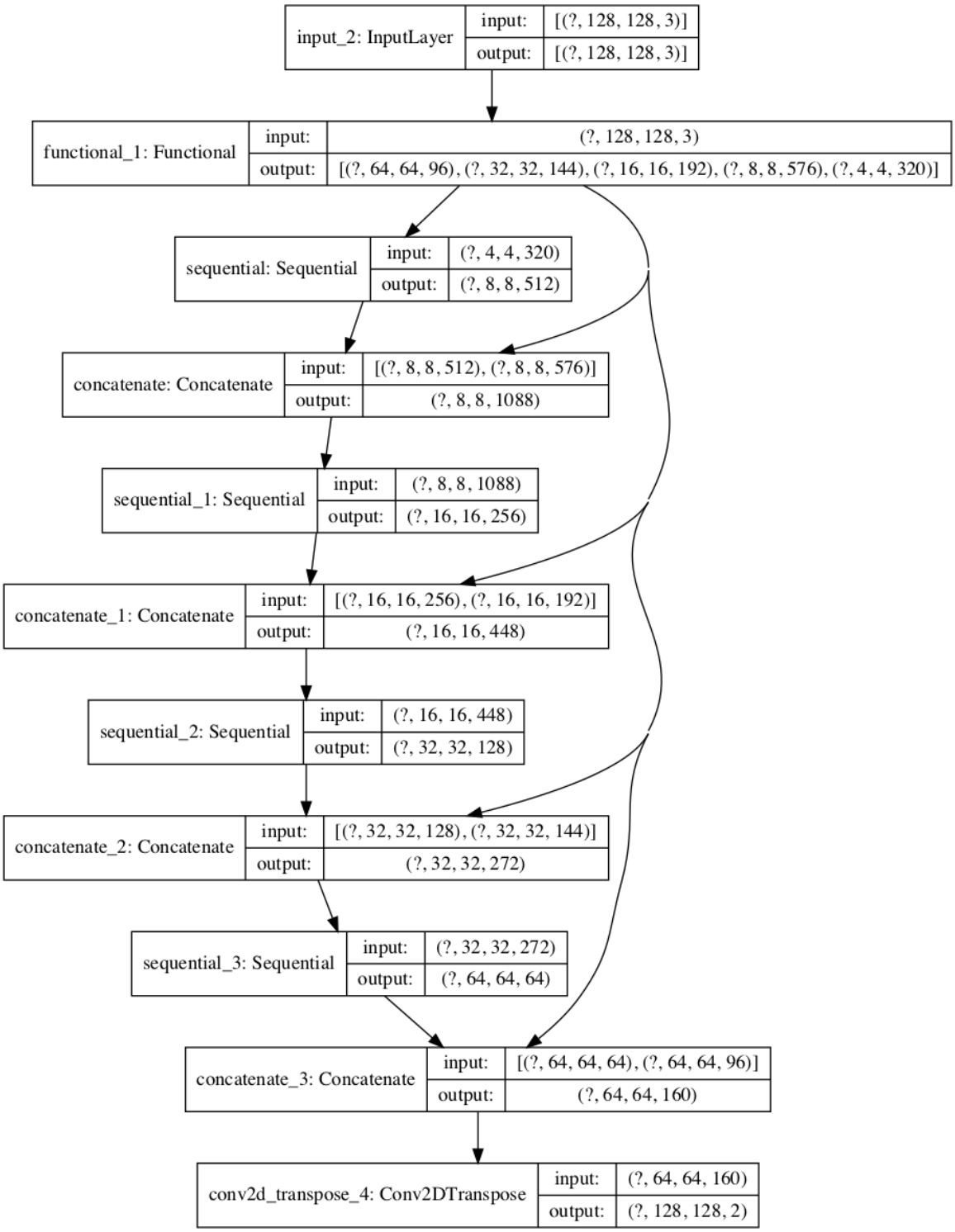


**Figure 4.** Full convolutional layers of the CNN model. Shows the movement from input layer to output layer and how each convolutional block affects array size.

The model input was a four dimensional tensor of size `(BATCH_SIZE, IMG_SIZE, IMG_SIZE, N_CHANNELS)` such that each size was user inputted and mutable. The model went through an encoder and decoder sequence. Each layer used two consecutive layers of convolution which were each followed sequentially by a ReLU activation function. This activation function was chosen over traditional sigmoid and tanh activation functions due to its speed and better performance. It allows for easy optimization of linear functions using gradient descent methods. The function limits the vanishing gradient problem posed by other activation functions by rectifying input values less than zero, forcing them to zero using a threshold operation (Nwankpa et al. 2018). A limitation of the ReLU activation function is its ability to easily overfit, this is prevented through the use of a dropout rate.

In the encoder (contraction) pathway, regular convolutions and max pooling were applied. The goal of pooling is to downsample an input image to extract dominant features. Max pooling is a downsampling method by which the maximum value is picked out of the receptive field of kernel size $(K_x, K_y)$. With max pooling, you look at the maximum presence of a feature rather than the average presence (such as in average pooling) and create position invariance over larger, local regions leading to a faster convergence rate (Nagi et al. 2011). Through the contraction pathway, the model learns pixel intensity information about the image, but loses spatial information in the process.

We then have the decoder (expansion) pathway. In the decoder path, we apply upsampling along with our regular convolutions. Upsampling is a process of transposed convolutions. The goal of this process is to regain spatial information for the image and create an output image of identical dimensions to the input image. The functions used to build both the encoder and decoder pathways were from the `tensorflow.keras.layers` package (Chollet 2015).

In addition to the encoder and decoder pathways that build the basic structure of the model, some additional pieces in model creation were important, specifically in regards to model training and compilation. The model is compiled as seen below:

```
model.compile(optimizer=Adam(), loss="binary_crossentropy", metrics=["accuracy"])
```

**Listing 1.** Compile Model

The optimizer and loss arguments are user defined and the `tensorflow.keras` (Chollet 2015) package has a multitude of options. The optimizer chosen was the Adam optimizer. The Adam optimizer is computationally efficient and easily implemented, even with large training datasets. It is a stochastic gradient descent method based on adaptive estimation of first and second order moments. This means that, not only does it base changes to the learning rate on the mean (first moment), it also uses the uncentered variance (second moment). It has a recommended learning rate of 0.001, which was used. In an optimization model, the learning rate is the parameter used to determine the step size of each iteration. Choosing the correct learning rate is important because it can mean the difference between overfitting and convergence of the model.

The binary cross entropy (log) loss function is a bit different than the more commonly used, traditional softmax loss function. It uses a sigmoid activation function coupled with cross entropy loss. This means that the loss calculated for each vector output in the CNN is independent of other values. Cross entropy is a measure of the difference between two randomly defined probability distributions. We can visualize this loss function as the log loss equation below:

$$J(\theta) = \frac{-1}{m} \sum_{i=1}^{m} y^{(i)} \log(y^{(i)}) + (1 - y^{(i)}) \log(1 - y^{(i)}) \tag{4}$$

The next step was fitting the model to training data:

```
results = model.fit(train_dataset, batch_size=BATCH_SIZE, epochs=EPOCHS, callbacks=callbacks,
    validation_data=val_dataset)
```

**Listing 2.** Fit Model

In addition to using the traditional `model.fit()` tensorflow function, callbacks were implemented using the `tensorflow.keras` keras library. The three functions used were: `EarlyStopping()`, `ReduceLROnPlateau()`, and `ModelCheckpoint()`.

`EarlyStopping()` was used to end model training early if the validation loss ceased to improve after 10 epochs. This was to protect against continuing model training without any improvement in accuracy. `ReduceLROnPlateau()` is a Keras function used for learning rate decay. If after 5 epochs (a used defined value) the validation loss had continued to plateau, the learning rate would decay by a factor of 0.1 (used defined) until the minimum learning rate of 0.00001 (used defined). This ensured a balance between a low learning rate, which allows for better prediction yet takes longer,

and a high learning rate, which may converge faster but could result in an inaccurate model. `ModelCheckpoint()` is the Keras function to save model weights as an HDF5 file. It was set to save the model after each epoch, yet only save the best model. This means that if the model in epoch 24 was not better than that of epoch 23, the HDF5 file would not be overwritten.

The model was then trained using the training dataset created, as described earlier. Training was set to halt after 50 epochs, however in some instances training ended early, around epoch 30. The learning curve from model training includes a red X which shows the best model (see Figure 5). This model is saved and used for validation and implementation. The training curve is in orchid and the validation curve is in light blue.

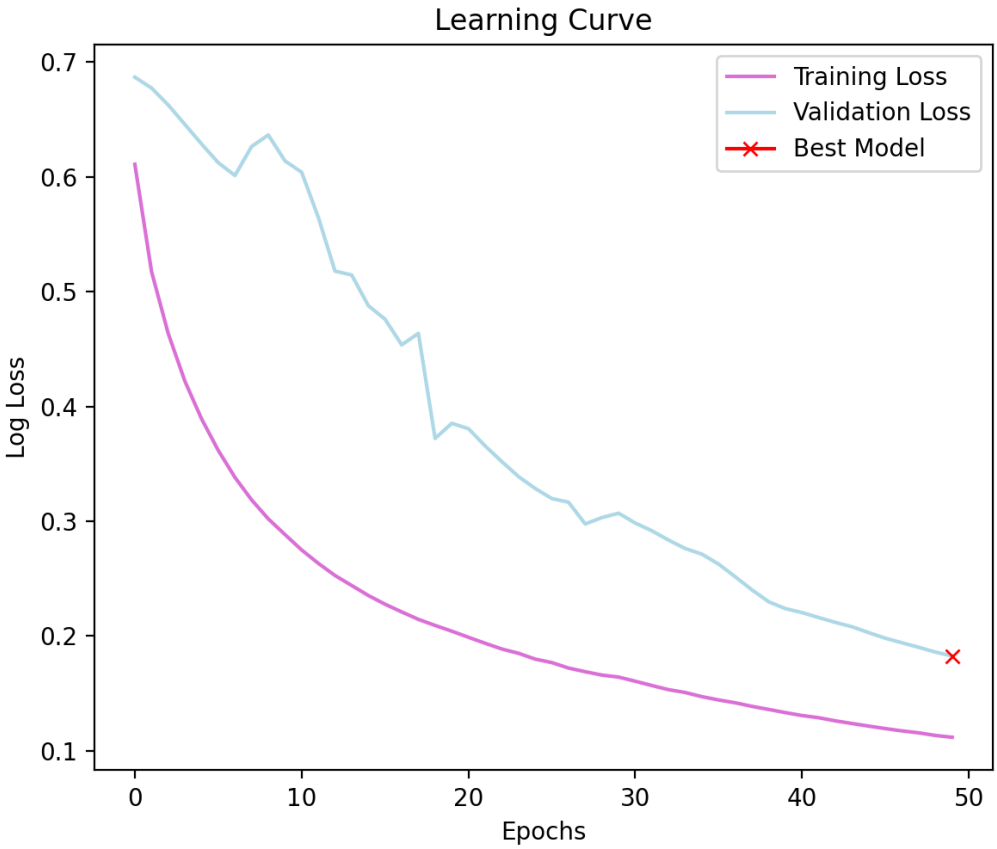


**Figure 5.** Learning curve for training of the convolutional neural network.

### 3.3. *Results*

The model was tested and validated using the training, validation, and test datasets. Each of these datasets was in the format of an HDF5 file that was read and parsed as needed. The predictions from each set of model validation steps is seen below. Each prediction has three images, the initial EUV Image, the coronal hole detection (CHD) segmentation mask, and the Model Prediction. Additionally, there is a binary model prediction which took the output prediction from the model and if the prediction was above 0.5, it decided that there indeed was a Coronal Hole at this pixel, and vice versa if the prediction was less than 0.5.

It is important to test results on data the model has never seen before to ensure that the model is not overfitting. This CNN model did an excellent job of detecting coronal holes in solar disk images. The ability for the model to create a floating point estimation of whether or not a coronal hole is present at a particular pixel is important for coronal hole prediction and tracking over time. This is an improvement upon the binary detection thresholding model that was used to create the segmentation masks for the dataset.

#### 3.3.1. *Prediction on Training Data*

This prediction was done on data the model saw while training. We see that the model prediction and segmentation mask are extremely similar, as because the model has seen this datapoint before.

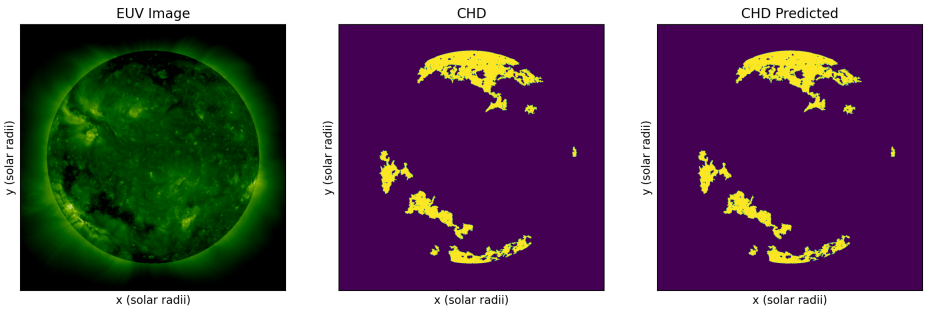


**Figure 6.** Prediction on Training Data: Compares CHD mask used in model training with the model output (CHD Predicted)

#### 3.3.2. *Prediction on Validation Data*

We see a prediction done on validation data below. This was the validation data that the model used when calculating the loss function and model accuracy.

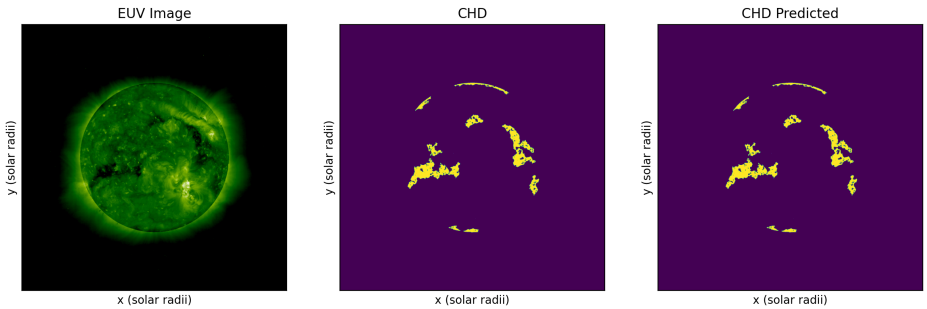


**Figure 7.** Prediction on Validation Data: Compares CHD mask used in model validation with the output predicted CHD

#### 3.3.3. *Prediction on Test Data*

Finally, we see below a prediction done on test data — completely new, never been seen before data for the model. We see that this prediction seems extremely accurate!

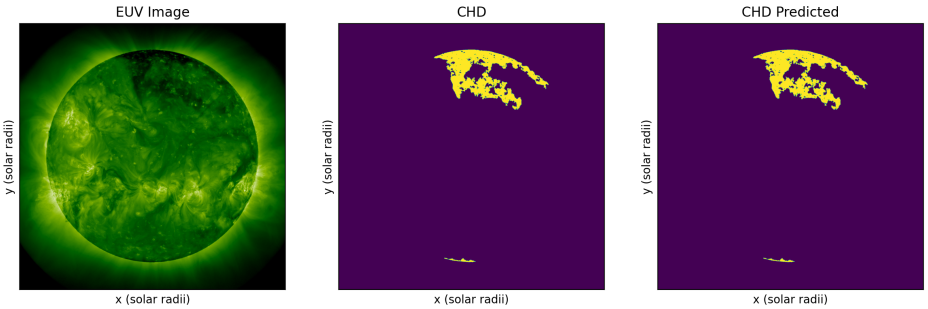


**Figure 8.** Prediction on Test Data: Compares prediction output from CNN model with output from the originally used FORTRAN CH detection algorithm (Caplan et al. 2016). The model has never seen this EUV image before, and never saw the CHD from the FORTRAN algorithm.

### 3.3.4. *Full Sun Maps*

Finally, using this supervised learning model, we are able to create full-Sun maps. These maps take the EUV image from each instrument at the timestamp in question. These images are then mapped to the proper coordinate system using a minimum merge mapping technique.

Maps are originally created for each instrument image individually through interpolation from disk image to Carrington map, then merged (Caplan et al. 2016). Areas of overlap between different instrument maps are resolved by choosing the data with the minimum intensity from each point of overlap. We include two $\mu$ value threshold values to ensure we choose the best data in areas of overlap, yet also extend to the limb. We visualize the merge mapping method with "quality maps" denoting the origin instrument and its mu dependence (darker color corresponds to better viewing angle).

The simplest merge map is an instantaneous "synchronic" map which combines three images (AIA, EUVI-A, and EUVI-B) taken at approximately the same time. We then overlay a coronal hole detection merged in the same manner as seen in figure 9. This results in a full-Sun map that can be used to determine where and when coronal holes were present.

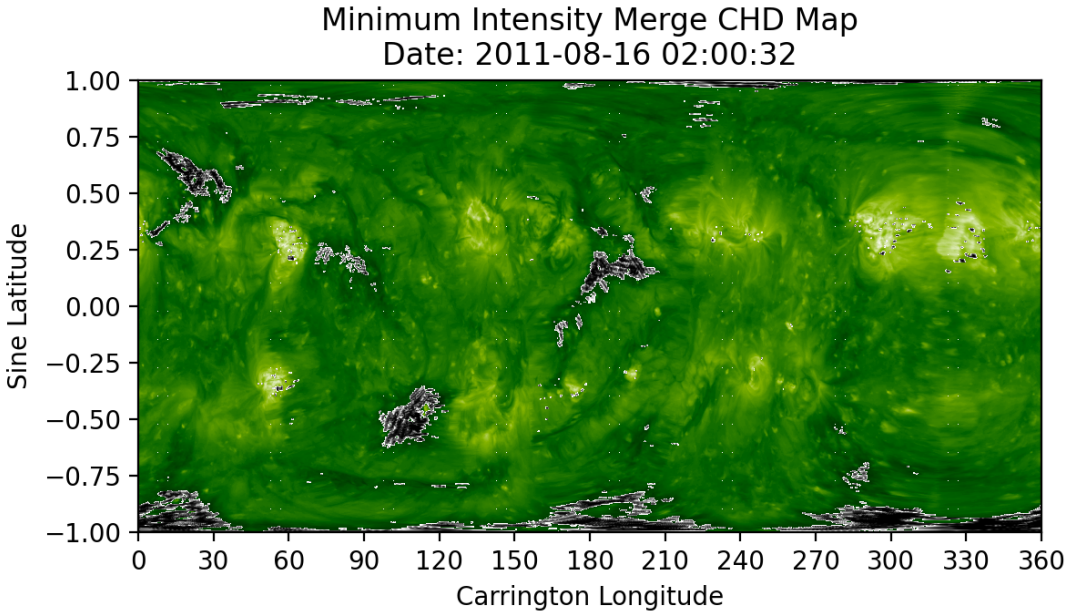


**Figure 9.** Full Sun Synchronic Map: 2011-08-16 02:00

## 4. K-MEANS CLUSTERING

Unsupervised learning is a machine learning method through which unlabeled data is labelled (Z. 2004). The advantage of using unsupervised learning to detect coronal holes and active regions is that it removes user interference and an individual's definition of a coronal hole from the process. Despite the minimal user interference, there are certain hyper parameters that must be selected by the individual and these have been optimized using various industry techniques to ensure that we have the least amount of user interference as possible in our detections (Z. 2004).

K-Means is one of the most popular unsupervised learning algorithm. It is a clustering algorithm which means that it iteratively takes observations and assigns them to a cluster. In the case of k-means, the algorithm takes N observations and assigns them to K clusters whereby each observation belongs to the cluster with the nearest cluster centroid (Qi et al. 2017). Each sample/observation receives a cluster label based on both spatial and intensity data, and these labels are then grouped into coronal holes and active regions. You can plot each detection individually, or both active regions and coronal holes on the same map.

The basic steps of k-means are (Qi et al. 2017):

1. Select cluster centers.
2. Calculate distance between each sample and cluster centers.
3. Assign data point to closest cluster center.
4. Calculate new cluster centers.
5. Recalculate the distance between each data point and newly obtained cluster centers.
6. Continue until convergence.

### 4.1. *Centroid Initialization*

The initialization of cluster centroids is the most volatile of the steps of this process as it determines both the speed, and accuracy of your results. This refers to the process of choosing the initial centroid locations and there are many commonly used methods to do this (Qi et al. 2017). Random initialization of cluster centroids is a commonly used method, however can result in output centroids that are ill-positioned in comparison to what is expected (Qi et al. 2017). Therefore, to select initial cluster centroids, we used `k-means++`. This is a method for centroid initialization that results in faster convergence of centroids (Qi et al. 2017) and reduces erroneous results. It is a probabilistic-based seeds method where the first centroid is chosen at random and each subsequent centroid is chosen from the remaining data points using a weighted probability distribution of distance from the closest existing center squared (Arthur & Vassilvitskii 2007). Although k-means++ does randomly initialize the first cluster centroid, the remaining centroids are not chosen at random which is what differentiates this method from purely random initialization, and leads to more accurate results.

The process for centroid initialization is as follows: (Arthur & Vassilvitskii 2007)

1. Randomly choose one center from among the data points.
2. For the remaining data points (x), compute the distance between x and the nearest already chosen centroid.
3. Choose a new data point as a new centroid using a weighted probability distribution proportional to distance squared.
4. Continue to repeat steps 2 and 3 until K cluster centroids have been chosen.
5. Proceed to use k-means clustering.

### 4.2. *Number of Clusters*

The last important user choice in the algorithm is the number of clusters (k). Much like the initialization of cluster centroids, there are many ways to choose the k value you will use. This is often situationally dependent and contingent upon how many groups you want to cluster your data into. For this case, we are looking at two main groups: coronal holes and active regions. However, there is a lot more going on on the solar surface than just these two phenomena. This means our choice of k value will have extreme impact on our results and thus is a crucial decision for an accurate model.

The value of k for this model was chosen using the elbow method. This is a method for choosing the optimal number of clusters (k) for our problem. This is done by running k-means on a range of k values and calculate the sum of squares error (SSE) for each value of k (Syakur et al. 2018). We then plot the SSE against the value of k. We isolate the elbow which is where the error starts becoming stable to determine the best value of k (Syakur et al. 2018). We choose a value of $k = 14$ for our problem.

### 4.3. *Region Detection*

The next step in the process is the actual detection of coronal holes and active regions. This entails converting our clustering output to specific coronal hole and active region labels. Since these regions are normally designated by intensities, we use average intensities of the cluster to determine what should be categorized as a coronal hole, active

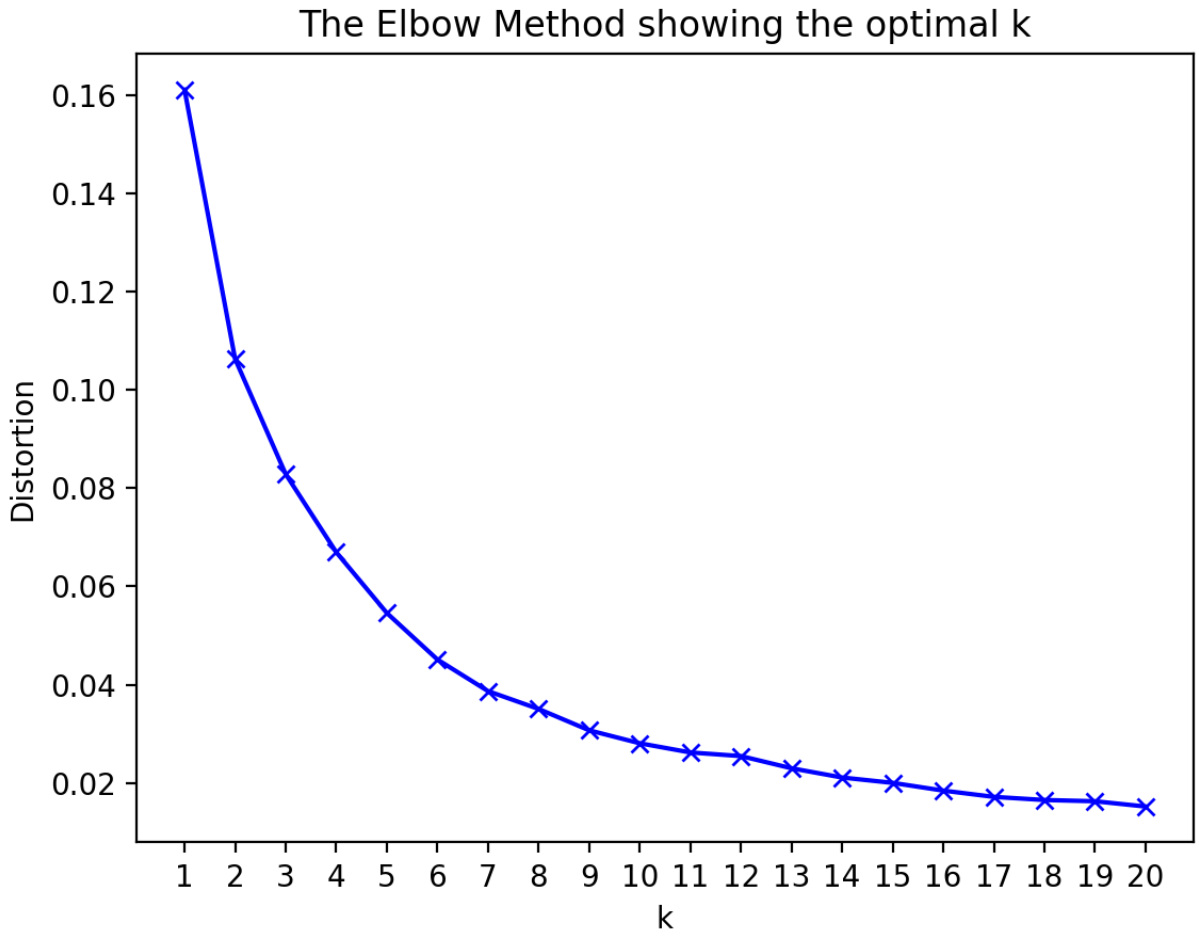


**Figure 10.** Elbow method for determining the optimal value of clusters (k).

region, or quiet sun. The average intensity of each cluster is calculated and from there clusters are chosen to be either coronal holes or active regions.

Coronal hole regions are designated as the two clusters of lowest intensities while active regions are the highest intensity cluster. To create a smooth coronal hole detection, we combine the two lowest intensity clusters into one conglomerate cluster (detection). We then build maps with these detections that are overlaid on full-Sun EUV maps. We denote coronal hole regions in red and active regions in blue as seen in figure 11.

### 4.4. *Model Validation*

The final step in this process is validation of our model. Because this is an unsupervised model, we do not have labelled data to compare our results to. However, we can do visual analysis of our results to determine how accurate they seem, as well as compare our unsupervised detection to previous methods of detection.

To achieve this, we look to compare the unsupervised detection with the previously used brute force detection algorithm as seen in figures 12 and 13 (Caplan et al. 2016).

We also look at comparing areas of detected coronal holes between these two methods. We see similar trends in area over time between the two methods, with the unsupervised method consistently detects less pixel-wise coronal hole area. There are some outliers which are the sharp spikes and these extras coronal holes often show up as flickering detections on large timescales.

## 5. CONCLUSION

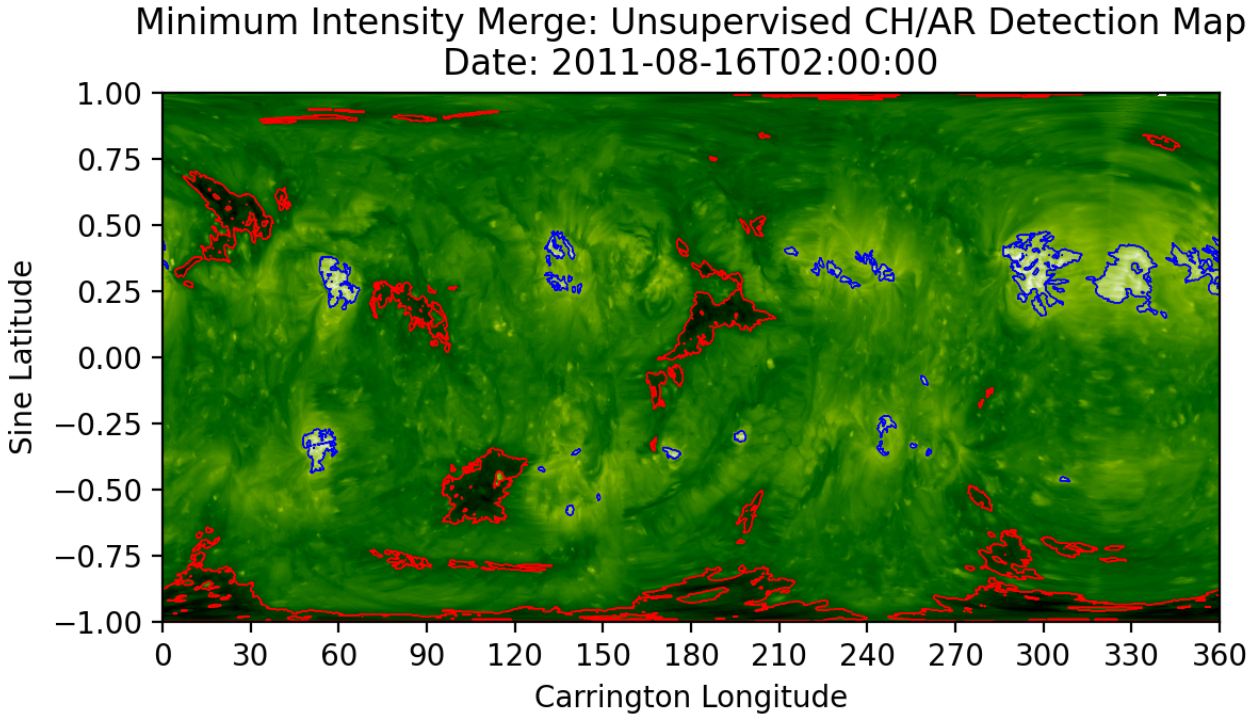


**Figure 11.** Full-Sun EUV map with overlaid coronal hole (red) and active region (blue) detections.

Through machine learning, the detection of coronal holes and the resulting applications has become a new frontier for gaining greater understanding of our closest star. These solar phenomena are not easily understood nor observed and more importantly are entirely subject to individual bias. Therefore, detecting coronal holes through machine learning techniques is an important step towards unbiased detection and greater scientific accuracy. The supervised convolutional neural network provides the necessary framework for CH detection, however due to the supervised nature of the model, it still contains the human aspect of threshold designation. The use of unsupervised learning to detect coronal holes allows us greater confidence in our results due to the limited human interference with our detection model. Although there is still user interaction due to the choice of parameters through dataset creation, choice of k value, and initialization of centroids, we find that through various methods of optimization, we are able to limit the effect that human choice has on these detections.

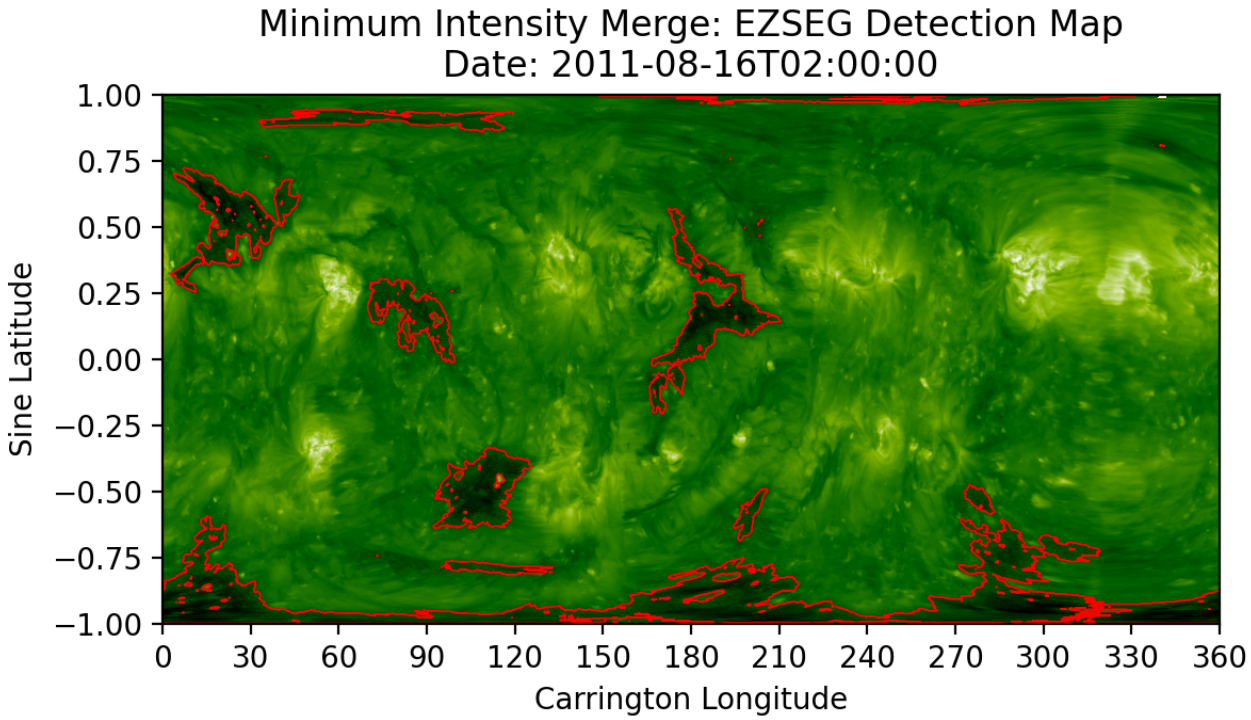


**Figure 12.** Brute force coronal hole detection using a Fortran region growing algorithm (Caplan et al. 2016).

## REFERENCES


Abadi, M., Agarwal, A., Barham, P., et al. 2015, TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/

Arthur, D., & Vassilvitskii, S. 2007, in Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (USA: Society for Industrial and Applied Mathematics), 1027–1035

Boerner P., Edwards C., L. J. 2012, SoPh, 275, 41, doi: 10.1007/s11207-011-9804-8

Caplan, R. M., Downs, C., & Linker, J. A. 2016, The Astrophysical Journal, 823, 53, doi: 10.3847/0004-637x/823/1/53

Chollet, F. 2015, Keras, https://github.com/fchollet/keras, GitHub

Harris, C. R., Millman, K. J., van der Walt, S. J., et al. 2020, Nature, 585, 357, doi: 10.1038/s41586-020-2649-2

Hegde, M., Hiremath, K., & Doddamani, V. H. 2014, Advances in Space Research, 54, 272, doi: https://doi.org/10.1016/j.asr.2014.04.001

Hunter, J. D. 2007, Computing in Science & Engineering, 9, 90, doi: 10.1109/MCSE.2007.55

Nagi, J., Ducatelle, F., Di Caro, G. A., et al. 2011, in 2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA), 342–347, doi: 10.1109/ICSIPA.2011.6144164

Nwankpa, C., Ijomah, W., Gachagan, A., & Marshall, S. 2018, arXiv e-prints, arXiv:1811.03378. https://arxiv.org/abs/1811.03378

Pedregosa, F., Varoquaux, G., Gramfort, A., et al. 2011, Journal of Machine Learning Research, 12, 2825

Qi, J., Yu, Y., Wang, L., Liu, J., & Wang, Y. 2017, International Journal of Distributed Sensor Networks, 13, 1550147717728627, doi: 10.1177/1550147717728627

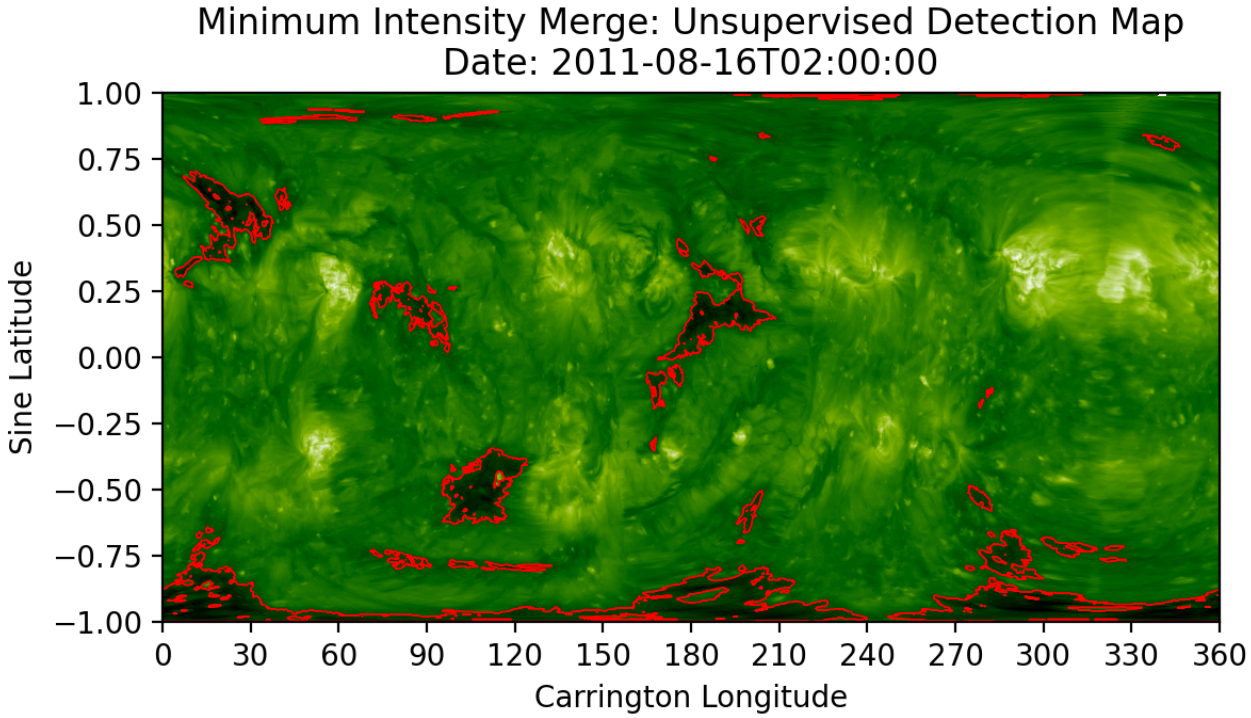


**Figure 13.** Unsupervised coronal hole detection using k-means clustering as previously described.

S., C. 2009, Living Rev. Solar Phys., 6, 66, doi: 10.12942/lrsp-2009-3

Syakur, M. A., Khotimah, B. K., Rochman, E. M. S., & Satoto, B. D. 2018, IOP Conference Series: Materials Science and Engineering, 336, 012017, doi: 10.1088/1757-899x/336/1/012017

Van Rossum, G., & Drake, F. L. 2009, Python 3 Reference Manual (Scotts Valley, CA: CreateSpace)

Wuelser, J.-P., Lemen, J. R., Tarbell, T. D., et al. 2004, 5171, 111 , doi: 10.1117/12.506877

Z., G. 2004, Unsupervised Learning, Vol. 3176 (Berlin, Heidelberg: Springer), doi: 10.1007/978-3-540-28650-9_5

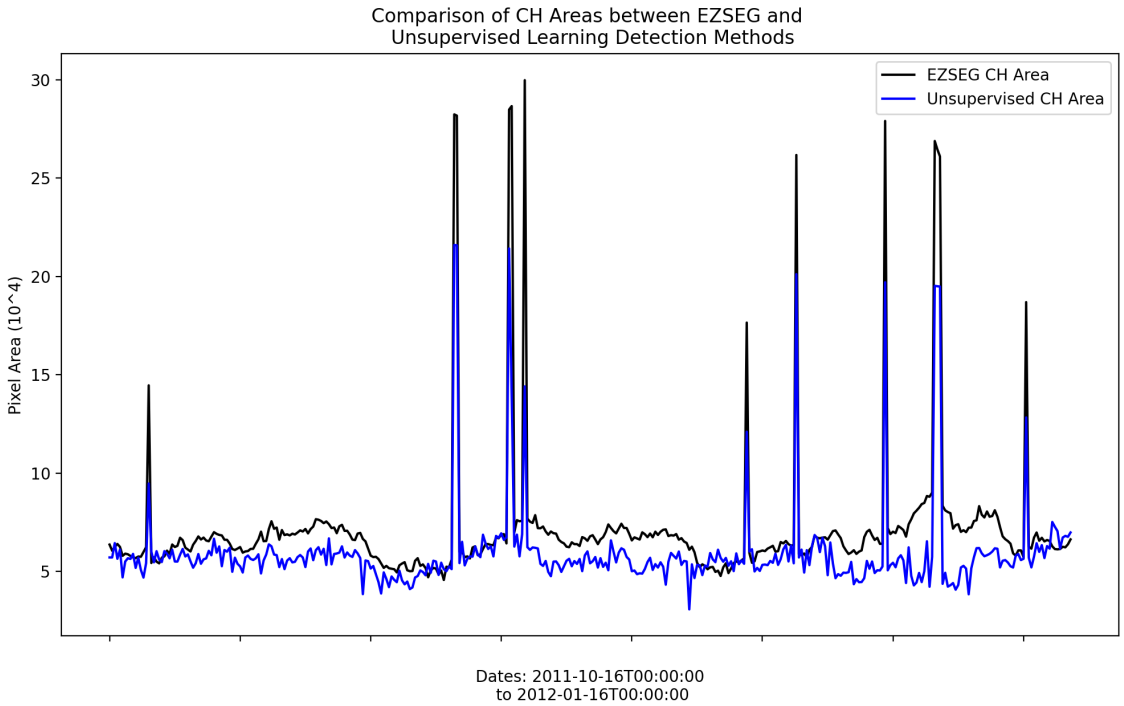


**Figure 14.** Comparison of detected coronal hole area between the previous detection method and the new unsupervised model.