Using Machine Learning to Track the Location of the Shock Train in Hypersonic Engines

Alison Reynolds

Follow this and additional works at: https://scholarworks.wm.edu/honorstheses

Part of the Applied Statistics Commons

Recommended Citation
https://scholarworks.wm.edu/honorstheses/1631

This Honors Thesis -- Open Access is brought to you for free and open access by the Theses, Dissertations, & Master Projects at W&M ScholarWorks. It has been accepted for inclusion in Undergraduate Honors Theses by an authorized administrator of W&M ScholarWorks. For more information, please contact scholarworks@wm.edu.
Using Machine Learning to Track the Location of the Shock Train in Hypersonic Engines

A thesis submitted in partial fulfillment of the requirement for the degree of Bachelor of Science in Mathematics from William & Mary

by

Alison Reynolds

Accepted for (Honors)

Greg Hunt, Chair
Rex Kincaid
Jon Kay
Heather Sasinowksa

Williamsburg, VA
May 13, 2021
Using Machine Learning to Track the Location of the Shock Train in Hypersonic Engines

Alison Reynolds
Abstract

Proposed hypersonic vehicles would be able to travel at five to ten times the speed of sound, but there are still many problems that need to be solved to construct a functioning vehicle. One such problem involves shocks created in the engine isolator when the vehicle reaches high speeds. These shocks must be contained to the isolator to maximize performance and avoid potential failure. This project attempts to track the location of the leading shock given images of airflow from ground tests of engines using random forests and convolutional neural networks. When the models are trained and tested on data from the same facility, the average root mean squared testing error is approximately 18-20mm. However, the models struggle to adapt to the difference in lighting conditions and increase in noise when tested on data from a different facility, leading to much higher error.
Acknowledgements

I’d like to thank my advisor Dr. Greg Hunt for all of his guidance and assistance on this project. I’d also like to thank Dr. Jon Kay, Dr. Rex Kincaid, and Dr. Heather Sasinowska for serving on my committee, as well as Dr. Ryan Vinroot for always encouraging me to complete an honors thesis. Finally, I’d like to thank my family and friends, in particular Ashley Hernandez-Estrada who has been with me through every step of this.
Contents

I. Introduction .................................................................................................................. 1

II. Data .................................................................................................................................. 3

III. Methods .......................................................................................................................... 5

IV. Application to UMDCI Data .......................................................................................... 11

V. Transfer Learning .......................................................................................................... 20

VI. Conclusion ..................................................................................................................... 27
List of Figures

1 Dual-Mode Scramjet ........................................................................... 1
2 STLE Truth Over Time ........................................................................ 4
3 Random Forest Diagram .................................................................... 6
4 Example UMDCI Image ...................................................................... 7
5 Summed Column Values ..................................................................... 7
6 Convolutional Layer ......................................................................... 8
7 CNN Architecture .............................................................................. 9
8 Random Forest Predictions ................................................................. 12
9 Distribution of STLE Values ................................................................. 13
10 Random Forest/CNN Testing Error .................................................... 18
11 Random Forest/CNN Predictions ........................................................ 19
12 Example AFIT Image .......................................................................... 20
13 Transformed AFIT Image .................................................................... 21
14 AFIT Random Forest/CNN Predictions ............................................. 22
15 AFIT Cropped Random Forest/CNN Predictions ............................... 24
16 Standardized Column Sums ................................................................. 25
Section I

Introduction

Hypersonic vehicles have long been a goal for air travel. While existing supersonic planes such as the Concorde can travel at up to twice the speed of sound, a hypersonic vehicle could reach five to ten times the speed of sound, traveling from Washington D.C. to Micronesia in 30 minutes [1]. Since the Concorde first flew in 1969 there has been a spike in interest in commercial hypersonic technology, sparking advancements in fluid dynamics, material sciences, and aerodynamical control systems. However, some problems are still unsolved, including how to control the airflow in the engine. One major problem is in the engine isolator which connects the air intake to combustor where fuel is burned (Figure 1).

![Propulsion-airframe integrated scramjet](image)

**Figure 1:** In this diagram of dual-mode scramjet, the red X’s denote the shocks in the engine isolator. They typically move back and forth within the isolator [2].

When an aircraft operates at speeds between approximately three to seven times the speed of sound (known as Mach 3 or Mach 7), heat release and pressure in the combustor create
a series of shocks that move back and forth in the isolator that are known as a shock train [3]. Shocks are characterized by abrupt changes in flow properties and are the ultimate explanation for sonic booms which are caused by bow shocks coming off the front of a vehicle. The isolator shocks move during flight as a result of changes in trajectory or engine operations; however, it is imperative that the shock train is contained within the isolator, otherwise it could cause the engine to operate inefficiently or catastrophically fail. The location of the shocks can be studied by following the initial shock (known as the shock train leading edge, or STLE). It is of interest to track the location of the STLE for fundamental studies of airflow dynamics and real-time applications in controlling the location of the shock train. The goal of this work is to use Schlieren based images to track the shocks using machine learning.

There are two types of tests for the engines: ground tests and flight tests. Ground tests are performed in laboratories to study the engines, and flight tests are performed in the air to see how they react to real conditions. Ground tests are cheaper to run by orders of magnitude and thus are used much more often, especially in the earlier stages of development. This work is primarily designed to facilitate ground tests as that is where researchers have optical access and can produce the imagery used in this analysis. Additionally, this project helps demonstrate the general principles for using convolutional neural networks to study flow dynamics. Beyond a research tool, we hope that the methods and knowledge about how to accurately track shock location might be used to inform real engine design.

The next section will describe the data being used for this project. Section III will introduce the machine learning algorithms applied to the data. Section IV discusses results when the methods are applied to the initial data set, and Section V describes the adjustments and results for transfer learning. Finally, Section VI presents conclusions and ideas for future work.
Section II

Data

This project is a collaboration with the Hypersonic Airbreathing and Propulsion Branch at NASA Langley who provided the data for this analysis. Researchers can generate shock trains in conditions similar to those of hypersonic engines using a wind tunnel facility. The flow is imaged using high-speed Schlieren photography, producing 10,000 images per second. This data was collected from two sources: the University of Michigan Direct-Connect Isolator (UMDCI) facility and the Air Force Institute of Technology (AFIT) isolator facility. The UMDCI imagery has a resolution of 3.512 pixels per millimeter and are black and white images [4]. The dimensions of the UMDCI images are 800 pixels by 200 pixels. The AFIT data has a resolution of 4.331 pixels per millimeter and are color images [5]. As a result, the AFIT data is converted to grayscale using the Python Imaging Library. The dimensions of the AFIT images, after cropping to remove empty space, are 520 pixels by 140 pixels. The UMDCI images are grouped into runs of approximately 31,000 images based on the test run they were created in, as this is the storage capacity of the camera used. This analysis uses four runs of UMDCI data. The AFIT data consists of a single set of 3,000 images.
One important factor to note is that the shocks move at different speeds in each run. Downstream from the shocks is a butterfly valve that moves at different frequencies for each run, which changes the air flow in the isolator. This work analyzes runs 1, 2, 3, and 5 of the UMDCI data which use the frequencies 1 hertz, 3.33 hertz, 2 hertz, and 7.14 hertz, respectively (Figure 2). This changes how fast the shocks move in the isolator and as a consequence any location prediction model cannot simply rely on movement speed for consistent predictions.
Section III

Methods

This is a supervised learning problem, as each of the runs comes with the horizontal location of the STLE for each image. The primary methods used for analysis were random forests and convolutional neural networks. Random forest is a bagged machine learning model which averages the results of many different decision trees (Figure 3). A regression decision tree considers all variables in the training data and splits the data into two groups based on the value that minimizes the sum of the squared residuals where the predicted value for each group is the average of the data in that group. This continues recursively with each group until the groups are sufficiently small without being overfit. A random forest builds many decision trees and averages the predictions. For each tree, a random subset of that data is chosen and at each split in the tree, a random subset of the variables is available to be split on to decorrelate the trees. This prevents the model from building the same tree many times. In this analysis, the random forest builds 500 trees.
Figure 3: This figure illustrates the process of predicting a value using a random forest. The test data is input into each decision tree, and then the predictions are averaged [6].

Random forests need variables to split on, which need to be derived from the images in the training data. Since the goal is to predict the horizontal location of the STLE, the variables chosen were summed column values. For this method, the pixel values for each column of the image were summed to create a single value for that column. Many summation methods were explored to choose the most accurate and consistent predictors. Tests were done summing the values of all the rows, the top half, middle half, and bottom half, as well as the top, middle, and bottom third. Due to the noise at the edges of the images, the most consistent predictors of STLE location out of these options were found by summing the middle half of the image which will be discussed in section III. The variables were created by summing values from rows 50-150 out of the 200 rows in the image, as seen in Figures 4 and 5.
Figure 4: Image 20371 from run 1 of the UMDCI data. The values in between the white lines were summed for the column variables used in the random forest. The “X” shape furthest to the left of the image is the STLE.

Figure 5: Sum of middle 100 rows for each column for image 20371 from run 1 of the UMDCI data.

The other primary method used in this analysis was convolutional neural networks. These algorithms are loosely based off of the human brain, with interconnected nodes organized in layers. Each layer feeds information into the subsequent process based on the weights assigned to the connection between them. Convolutional neural networks (CNNs) are a specific type of neural network that are often used to analyze imagery. Instead of feeding summed column values
into the algorithm, CNNs take the entire image as the input. This may be a better method to analyze images because it looks at the image as a whole instead averaging pixel values of the vertical dimension, so it can encapsulate the spatial dependencies within the image.

The crucial feature of a convolutional neural network is the convolution layer, which makes the image more abstract to highlight features such as lines and curves. It takes a defined subset of the image matrix starting at the top left corner and computes the dot product with a filter of weights of the same dimensions as the subset (Figure 6). The weights control how important each area of the filter is to the convolved feature. Then the filter, also known as a kernel, is shifted over one unit and the dot product is computed for that subset. This process is repeated until the filter reaches the bottom right corner. Then, the entire procedure is repeated with a new filter with new weights to create another convolved feature. The filter values are generated using the Glorot normalized initialization, which draws values from a modified normal distribution whose standard deviation is based on the number of input and output units of the convolutional layer [7]. The output of the layer is several convolved features, one for each filter used. The final model for this project uses 16 filters with a three-by-three kernel.

Figure 6: The pixel values are shown here in black with the filter (or kernel) weights show in red in the bottom right corner of each box. The dot product of the pixels and weights are computed to create the smaller convolved feature that is input into the subsequent layer of the network [8].
Figure 7: This shows the architecture of the final convolutional neural network used in this analysis, with the order of the layers for the network.

The convolutional layer is followed by the activation layer, which applies an activation function to the convolved features (Figure 7). This layer will be discussed in more detail later, as several different activation functions were tested. The activation function accelerates the convergence of the model towards a stable solution and reduces the computations of the later layers, making the model more efficient [9]. It also introduces non-linearities to the model to
make the ultimate function fit of the network non-linear so it can learn non-linear relationships. After the activation layer comes the batch normalization which applies a z-score standardization, so the values are scaled to have a mean of zero and standard deviation of one. This helps account for noise in the image. Next comes the pooling layer, which reduces the size of the input. This model uses max pooling, which splits the input into two-by-two subsets and takes only the maximum value for each subset, reducing the size of the initial input to a quarter of the previous size. This again decreases the computational power needed to process the data and extracts the dominant features of the image. After the pooling layer comes the dropout layer which randomly sets input units to zero at a given rate to help prevent overfitting. The next layer is the flattening layer which compresses the data into a single dimension in order to connect all the layers. Finally, the dense layer (also known as the fully connected layer) connects all of the nodes from the previous layer using various weights which are updated as the model is trained. This layer makes the predictions for the model.
Section IV

Application to UMDCI Data

As mentioned earlier, the UMDCI data used in this analysis consists of four runs of 31,123 images each. This was the primary data used in the analysis as it is cleaner than the AFIT data and there are many more images. The first technique used to analyze the data was a random forest. As discussed above, the chosen variables for the random forest were summed column values using the middle 100 rows of data. However, other summing techniques were explored to reach this conclusion. The initial idea was to sum all of the values for each column to create the variables; however, there is a lot of noise at the top and bottom of the images that led to errors in the predictions. Using only a subset of the rows in the column sums would remove this noise and ideally lead to more accurate predictions. I tested four different summation methods: summing all row values for each column, values from the top half of the rows, the bottom half, and the middle half to see which method would result in the most consistent predictions. These methods were used in a cross-validation, where each model was trained using 5000 images from each of three of the runs, and then tested on 5000 images from the remaining run. This was repeated with each possible combination of three training runs and one test run, and the results from the testing are shown in Figure 8. For instance, to find the testing predictions for the top model of run 1, a random forest was trained using 5000 images from each of runs 2, 3, and 5 where the variables are the summed pixel values from the top 100 rows for each column. The model was then tested using 5000 images from run 1, and the results of that test are shown by the red dots in the top left graph of Figure 8.
Figure 8: This is a set of four scatterplots, with the true location of the STLE (based on column number of the image) on the x-axis and the predicted location from the random forest on the y-axis. Each model was trained using a random subset of images from the other three runs and tested on the run not used in training. The different colors represent different summing methods used to create the variables.

As shown in Figure 8, the models struggle to predict runs 1 and 5 when trained on the other runs. This is a product of the different frequencies between runs. The plots in Figure 2 show that as the frequency used in the run increases, the distance the shocks travel decreases. The shocks in run 1 travel the length of the isolator, so the STLE appears in all 800 columns of pixels. However, run 5 has a much higher frequency, so while the shocks all start in column 800 (the shocks move from right to left), they never go farther left than column 232. Similarly, values in run 2 only go as low as 176 and values in run 3 only reach 80. Therefore, as seen in Figure 9, the range and distribution of true values is different for each run, which causes problems for random forest since it can only predict values in the range of the training data. Thus, the model trained on runs 2, 3, and 5 cannot accurately predict values below 80, since none of the runs in
the training data achieve those values. Conversely, the leading shock in run 5 primarily stays in the center of the frame, while the other runs have a more “U-shaped” distribution. As the data for these models is randomly sampled from the runs, the training data mostly consists of images with the STLE at the edges of the frame, which causes difficulties when the testing data is centered in the frame.

![Distribution of STLE Values](image)

**Figure 9**: These four plots show histograms of the true horizontal location of the STLE for each run. Note that both the range and shape of the distribution is different for each run. This complicates the predictions of the random forest as it cannot predict values that do not exist in the training data.

The results from the vertical summing tests suggested that the most consistent predictions came from summing the bottom half or middle half of the columns, likely because most of the noise in the images is at the top. This is particularly visible in the plots for runs 2 and 3 in Figure 8. The bottom model generally outperformed the middle one, with an average RMSE of 66.880 compared to 87.641 for the middle model. However, after larger scale tests in which I trained
each model using 15,000 randomly selected images from each run and testing on 5000 randomly selected images from the test run (the run excluded from the training data), the middle model performed better than the bottom one. Therefore, I decided to move forward using the middle summing method as it was more consistent with large amounts of data.

The most important case for the model is when it is tested on a run that is completely excluded from the training data, as the model should be able to perform well on data it has never seen before. As a result, the metric used to evaluate the model is the average root mean squared testing error over each of the four cases, where the model is trained using three runs and tested on the remaining run. Overall, the testing error for the predictions of the middle summing model was 87.641 pixels, or 24.955 millimeters (the resolution of the data is 3.512 pixels per millimeter). However, as explained above, the model struggled to predict certain runs due to the differences between the distributions. To mitigate this error, I decided to choose the training data based on a uniform distribution of the range of true values. For each run, I broke the range of true values into 1000 evenly distanced points and found the image with the true STLE closest to each point. Due to the distribution of true values for the runs, this resulted in between 908 and 998 images with true values spaced across the range of true values. For these tests, each model was trained on the uniformly sampled images from three runs and tested on the entirety of the remaining run. Training the model using only these uniformly sampled images instead of a larger random subset decreased the root mean squared error of the random forest to 71.702 pixels, or 20.416 mm.

The goal with the convolutional neural network was to create a model that captured the spatial components of the images without being too complex or overfitting to the training data. As a result, the final network was rather shallow, with only seven layers, and there were several
parameters that I’ll discuss in the next paragraph that were set early on to ensure the model was not overfit. These parameters are primarily used in the process to train the weights, called backpropagation, which I’ll briefly explain here. Initially, the weights between each layer are randomly generated, and a subset of the training data is used to iterate through the network and calculate the loss function. The goal is to decide, based on the loss function, by how much to change the weights to improve the predictions. However, that information can only be determined one layer at a time, starting at the final layer of the network (since it is directly connected to the loss function). The answer to this problem is the chain rule. Starting at the final layer of the network and working backwards, the algorithm changes the value at a layer by the step size (one of the parameters set for the network) and calculates the derivative of the loss function based on that change. This continues for each layer, and the derivatives are strung together using the chain rule to find the derivative of the weights based on the loss function. The derivative determines how much to change the weights before the next iteration of the model. Backpropagation is also used to train the values used in the filters for the convolutional layer, using the same process with random initialization and updating based on derivatives.

The batch size and number of epochs determines how much data the network sees at one time and how many times it sees all the data. At the beginning of each epoch, the training data is randomly shuffled and split into batches based on the batch size set for the network (if the data does not split evenly into the batches, the last batch will have fewer samples). Then each batch is used to train the network, going through the backpropagation process described above. The epoch is complete once each batch has been used, so the network has seen all of the training data. Then this is repeated for each epoch. Limiting the batch size restricts the amount of memory needed to train the network, since it only needs to hold a small number of training samples at a
time. It also speeds up the training of the network as the network is updated more times per
epoch with a smaller batch size, so the network does lots of small computations instead of fewer
large computations. Most networks have a batch size between 16 and 256 (typically powers of
two) and between 10 and 1000 epochs. The most consistent model for this project based on the
same cross validation used for random forest was with a batch size of 32 and 50 epochs.

There were a few other parameters needed for the network. It uses the most common
optimizer for neural networks called the Adam optimization algorithm. It’s an adaptation of
stochastic gradient descent that is better for sparse gradients and averages recent gradient values
in the updates of the weights [10]. The step size used for the gradient descent, called the learning
rate, is set at $1 \times 10^{-5}$ as any increases or decreases in this value caused large increases in the
testing error.

Extensive testing was also done to choose the activation function used in the activation
layer (the third layer in the network). The classic method is Rectified Linear Unit (ReLU) which
sets all input values below zero to zero and leaves the rest unchanged. Since this is a piecewise
function, it adds nonlinearities to the model which allows it to learn more complex relationships
than simply linear functions [9]. I tested alternatives to this function, including Exponential
Linear Unit (ELU), Scaled Exponential Linear Unit (SELU), and Gaussian Error Linear Unit
(GELU). These are adaptations of ReLU that either zero-center the data or preserve more
information from the previous layers. While the GELU activation function led to a slightly lower
testing error (the RMSE was 0.432 pixels lower), this model uses the ReLU activation since this
difference is likely not statistically significant and ReLU is much simpler to implement (the
GELU function is very new and only available in the beta version of most neural network
packages).
The CNN is trained on the same 1,000 uniformly sampled images from three training runs as used for the random forest. The model is then tested on all images from the test run. The average test error from the runs was 15.390mm. However, as with the random forest, the CNN struggled to predict run 1 far more than the other runs. When the test error from run 1 is removed from this average, the error drops to 8.435mm. The difficulties with run 1 are likely a result of the CNN’s spatial focus; since the training data from runs 2, 3, and 5 does not include images with shocks in the first 80 columns of the image, the CNN learns that the shock is never in that area so it will not predict values there.

<table>
<thead>
<tr>
<th>Test Run</th>
<th>Random Forest RMSE (pixels/mm)</th>
<th>CNN RMSE (pixels/mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>R1</strong></td>
<td>137.246 / 39.079</td>
<td>127.328 / 36.255</td>
</tr>
<tr>
<td><strong>R2</strong></td>
<td>47.822 / 13.616</td>
<td>35.407 / 10.081</td>
</tr>
<tr>
<td><strong>R3</strong></td>
<td>56.078 / 15.967</td>
<td>33.903 / 9.653</td>
</tr>
<tr>
<td><strong>R5</strong></td>
<td>45.664 / 13.002</td>
<td>19.570 / 5.572</td>
</tr>
</tbody>
</table>

*Figure 10:* This table shows the testing error for each run using a model trained with data from the other three runs and tested on the entirety of the remaining run. The middle column shows the testing root mean squared error for the random forest model in both pixels and millimeters, and the third column shows the testing root mean squared error for the convolutional neural network.

Overall, the comparison for the two models in Figure 10 shows that the CNN performs better on all of the testing runs. As seen in Figure 11, the CNN predictions mostly follow the same pattern as the true location but are shifted a bit up or down. On the other hand, the random forest predictions are generally closer to the true locations, but they are noisier, especially at the tops and bottoms of the curves. Remarkably, the random forest may be closer to the true values because it is not as good at extrapolating as the CNN. The training data with true values in one area of the image do not affect the random forest predictions for training data with true values in
another part of the image because the variable space is split into groups. Usually this would detract from the model’s performance because it treats the variables independently while the CNN is a more global model, but the CNN’s predictions are skewed by the uneven distribution of true locations. Therefore, the CNN predictions are pulled to the modes of the truth range as that is where most of the observations lie and fail to predict at the edges. Conversely, the random forest predictions are not skewed by the distribution but are more sensitive to noise.

![Random Forest and CNN Predictions](image)

**Figure 11:** This figure shows the testing predictions for the random forest and convolutional neural network graphed against the true values for each run with elapsed time on the x-axis and the predicted or true location on the STLE on the y-axis.

Accuracy is not the only metric to measure an effective machine learning model. Speed and memory usage are also important factors to consider. The random forest model takes about an hour to train on a standard laptop, while the CNN takes about forty-five minutes when run on an online server. The CNN is typically run online because it requires more processing power and
runs most efficiently using a graphics processing unit (GPU), which allows it to run the computations parallelly. It can be run on a standard laptop, but it takes closer to six hours. The CNN also has slightly higher memory usage as there are more weights in the model. However, the accuracy benefits are likely worth the increased memory usage and computational requirements, especially if a GPU is available.
Section V

Transfer Learning

All of the analysis in the previous section was done using data from UMDCI. However, that is not the only facility working on the problem. For the prediction model to be used more broadly, it must be able to make accurate predictions using data from facilities it was not trained with. To test this transfer learning, I used a model trained on the uniformly spaced images from all four runs of the UMDCI data and tested it on data from the Air Force Institute of Technology (AFIT) isolator facility. As discussed in the data section and demonstrated in Figure 12, the AFIT data has several differences from the UMDCI data including different dimensions, a different pixel per millimeter resolution, and it is in color while the UMDCI data is in black and white. Additionally, the AFIT data was produced while replicating engine conditions at a different speed than the UMDCI data, so the shocks are a different shape (as seen in Figure 12, they are more curved than the shocks in Figure 4). As a result, I had to make some adjustments to the data and the model in order to make predictions for the AFIT data.

Figure 12: Image 419 from AFIT data produced on June 11th, 2006. The curved “X” shape to the left of the image is the STLE.
The data was initially in video form, so I used an algorithm to save an image of each frame resulting in 3,037 images. The data was then cropped to remove as much of the extra black space as possible, then transformed to black and white (Figure 13). However, the AFIT images were much darker than the UMDCI images, which caused problems for the models as the variables are pixel values and the AFIT data had much higher pixel values than the UMDCI data. To remedy this, before the UMDCI data was used to train the data, I standardized it by subtracting the mean of all the pixel values in the data for that run from each pixel value and dividing by the standard deviation of all the pixel values in the data for that run. I performed the same operation on the AFIT data using the mean and standard deviation for all of the AFIT images. This put all of the pixel values on the same scale to make them comparable. The AFIT images were also stretched to the same dimensions of the UMDCI images using nearest neighbor resampling. This matches the new pixel values in the expanded image to the closest pixel value from the original image.

![Figure 13: Image 419 from AFIT data, cropped and transformed to black and white to imitate the UMDCI data.](image)

After making these changes, the random forest and CNN models were trained using all runs of the UMDCI data and tested on the entirety of the AFIT data (while there are 3,037 images in the AFIT data, the first shock does not appear in the video until frame 132, so only 2,906 images are used to calculate testing error). The random forest model was trained using the
1000 evenly spaced images from each of the four UMDCI runs with 500 trees in the random forest. However, the random forest model did not translate well to the new data, with a RMSE of 413.248 pixels (note that while rows 50-150 were summed for the UMDCI data, rows 66-166 were summed for the AFIT data due to the framing of the image to center the summation vertically around the shocks). The CNN, which was trained on 1,000 images from each run following the same method as in the UMDCI analysis, had an even higher testing error, with a RMSE of 449.501 pixels. But as seen in Figure 14, the random forest model predicts the same value (608.045) for every image while the CNN follows the approximate pattern of the true values but shifted downwards.

**Figure 14:** Testing predictions for the AFIT data using z-scored UMDCI data from all runs. The horizontal axis is the frame of the video and the vertical axis is the location of the STLE. Random forest predictions are in blue and CNN predictions are in red, with the true values in black.
While these are not optimal results, they are consistent with the design of the models. Even though the pixel values have been standardized, due to the differences in lighting and color between the UMDCI data and the AFIT data, the pixel values of the shocks are different between the datasets. The random forest is incapable of extrapolating to values outside of the training data because they do not fit well into any of the branches. Meanwhile, the CNN recognizes the shape of the STLE but because the pixel values are different, it cannot pinpoint their location.

One idea to rectify this was to force the images to represent the same size across the datasets. The UMDCI images are 800 pixels by 200 pixels with a resolution of 3.512 pixels per millimeter. On the other hand, the AFIT images are 520 pixels by 140 pixels with a resolution of 4.330 pixels per millimeter. Therefore, while the UMDCI images represent 12,972.120 mm$^2$, the AFIT images represent a much smaller space at only 3,881.639 mm$^2$. In order to match the physical properties of the images, the UMDCI data was stretched to represent the same amount of area per pixel as the AFIT data (again using the nearest neighbor technique), and then cropped to match the size of the AFIT data. The truth values are also adjusted to match the size of the image. Since the AFIT data represents a much smaller area, this resulted in the loss of much of the UMDCI images. This did decrease the random forest RMSE all the way down to 68.007 pixels, but as seen in Figure 15, it is again picking approximately the same value for every image (with a bit more variation). Meanwhile, the CNN RMSE testing error increased to 470.116 pixels, but it is still following the approximate pattern of the true values.
Figure 15: Testing predictions for the AFIT data using z-scored and cropped UMDCI data from all runs. As in Figure 14, the horizontal axis is the frame of the video and the vertical axis is the location of the STLE with random forest predictions in blue, CNN predictions in red, and the true values in black.

Although the random forest has lower testing error in both cases, the CNN predictions are more correlated with the true values than the random forest predictions. The $R^2$ value for the uncropped CNN predictions with the true values is 0.638, while the $R^2$ value for the random forest model is 0 (since the predictions do not change). Similarly, the cropped CNN predictions have an $R^2$ value of 0.635 with the true values while the random forest predictions have an $R^2$ value of 0.007. Therefore, even though the RMSE is much lower for the random forest, the CNN does a much better job of following the relative location of the STLE as it moves across the frame, but it misses the true value because of the differences in pixel values between the datasets.

While resizing and cropping the UMDCI data to match the resolution and size of the AFIT data does not significantly change the CNN predictions, it does improve the random forest
predictions by quite a bit. One possible reason for this is that after the UMDCI data is resized and cropped, the distribution of the summed column values is much closer to the distribution of the summed column values for the AFIT data, as seen in Figure 16. The first column of the figure shows the input data for the original models and the second column shows the input data for the cropped models (note that since the AFIT data only has 520 columns, the number of variables input into the cropped random forest decreases from 800 to 520). Cropping the UMDCI data cuts off the smaller values in the early columns so the distribution of the summed pixel values more closely resembles the distribution of the AFIT data, which is darker.

![Standardized Column Sums](image)

**Figure 16:** This figure shows the summed column values for the training and testing images for the models. The top row shows the summed column values for image 13161 of run 3 of the UMDCI data and the bottom row shows the summed column values for frame 1500 of the AFIT data. The first column is the data used in the original models, where the AFIT data is resized to match the size of the UMDCI data and the second column is the data used in cropped models, where the UMDCI data is resized to match the resolution of the AFIT data and cropped to match the size of the AFIT data.
Overall, both models struggled to adapt to data from a new facility. The CNN was able to follow the relative pattern on the STLE as it moves across the image, but the predictions were shifted down (to the left of the image) by about 400 pixels. The random forest had a lower testing RMSE, but it predicted the same value for every image in the original model and approximately the same value for each image in the cropped model. The CNN could become a much more accurate model if the predictions could be shifted up, which is a topic for future research on this project.
Section VI

Conclusion

Hypersonic research has come a long way in the past several decades, but it still has a long way to go. The aim of this project was to support ground tests of hypersonic engines by building a machine learning model to predict the location of the STLE given an image of airflow in the engine isolator. The primary analysis was done using four sets of data from UMDCI to train and test random forests and convolutional neural networks. The input variables for the random forest were summed column values of the middle 100 rows of each image, as this removes the noise at the tops and bottoms of the images, and the inputs for the CNN were the entire images.

Since the distribution of the true STLE locations is vastly skewed and bimodal, both models were trained using a subset of 1,000 images from three of the runs that were equidistantly spaced along the range of true values and tested on the entirety of the remaining run. The random forest generally predicted within 20.416mm of the true value and the CNN within 18.564mm of the true value, though both models struggled to predict on run 1 as it had a larger range of true values than the other runs.

Both models were then trained using all four runs of data from UMDCI and tested on a run of data from AFIT. However, even after the AFIT images were transformed to black and white and resized to match the dimensions of the UMDCI images, both models struggled to predict the location of the STLE. The random forest initially predicted the same value for each image and the CNN, while able to follow the relative motion of the STLE, missed the true value by about 100mm for each image. This is likely due to the light differences between the images which changes the relative pixel values between the shock and the rest of the image, as well as

27
the higher noise ratio in the AFIT data. In an attempt to mitigate this, the UMDCI data was resized to match the resolution and size of the AFIT data. While this shifted the random forest predictions closer to the true values and added a bit more variation, they are still very inaccurate, and the CNN predictions stayed approximately the same.

The random forest is limited by its design as it cannot extrapolate outside of the training data, so future research on this topic involves improving the CNN to better adapt to data from new facilities. Some possibilities include adding more convolutional blocks, applying pre-trained models before fitting a new one, and doing more preprocessing on input data. These could help make the model more adaptable to new data. Additionally, a time variable could be added to the model to take into account the order of the images (as the current STLE location is dependent on the location in the image before).
References


