LINE SCAN EXTENSOMETER PROJECT METHODS

Gabe Morris

INTRODUCTION

A line scan camera is a technology used in high-strain rate material testing to capture precise strain data. This method enables highly accurate measurements by tracking deformation in real time. Given this contactless interface, it offers more reliable data compared to strain gauges, especially at high strain rates because the specimen tends to slip in its interface with the grips.

Figure 1 – Line Scan Principle

In Figure 1, a black reference mark is drawn on the specimen, while the grey line represents the scanning width of the camera. The line scan camera operates by continuously capturing changes in the length of the black mark at a known rate. Strain is calculated by measuring the change in pixel count relative to the original pixel measurement. If precise displacement tracking of a single edge is needed, then a pixel-to-distance ratio can be established to convert pixel shifts to physical distances.

PROBLEM DESCRIPTION

The primary challenge this project addresses is the tedious post-processing of line scan images, specifically determining the starting and ending points of the scanned line. Currently, this process requires manual input for initial row in the solver, making it time-consuming. By implementing a machine learning algorithm, the aim is to automate this process.

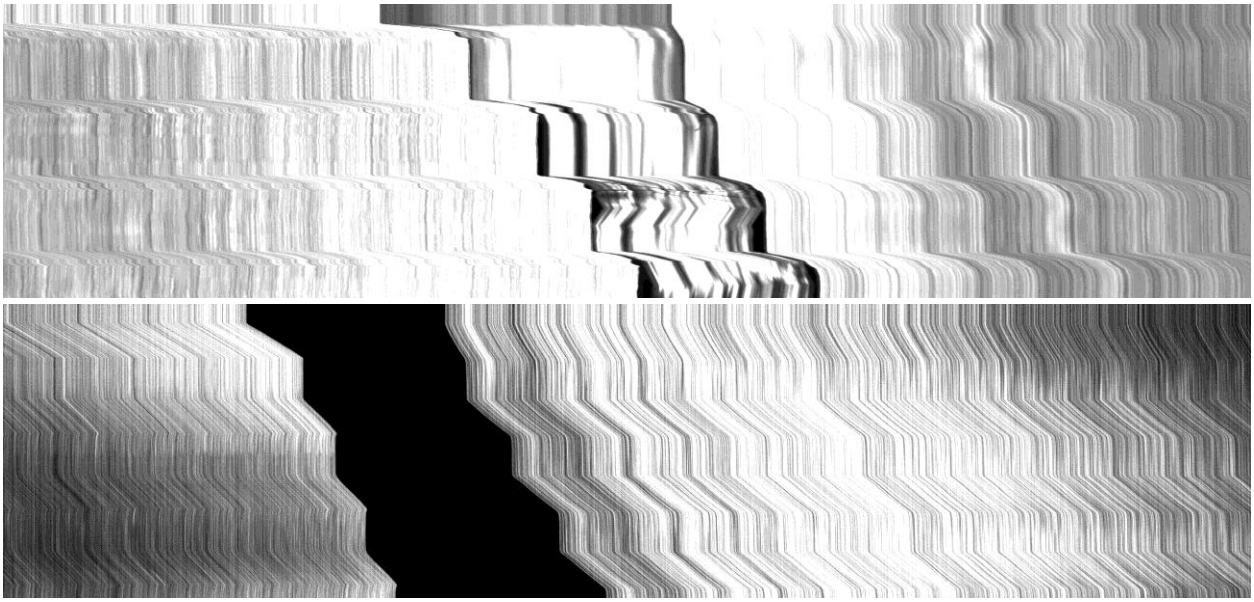

Figure 2 – Example of Noisy Images

While this problem may seem simple, line scan images tend to have atrocious noise. This can be clearly seen in Figure 2, where the noise surrounding the gauge section seems overwhelming. Before trying a machine learning algorithms, an attempt was made to find the edge by an algorithmic solution. The attempt included edge detection methods like Sobel and noise filtering with PCA or gaussian filtering, but the results were less than satisfactory.

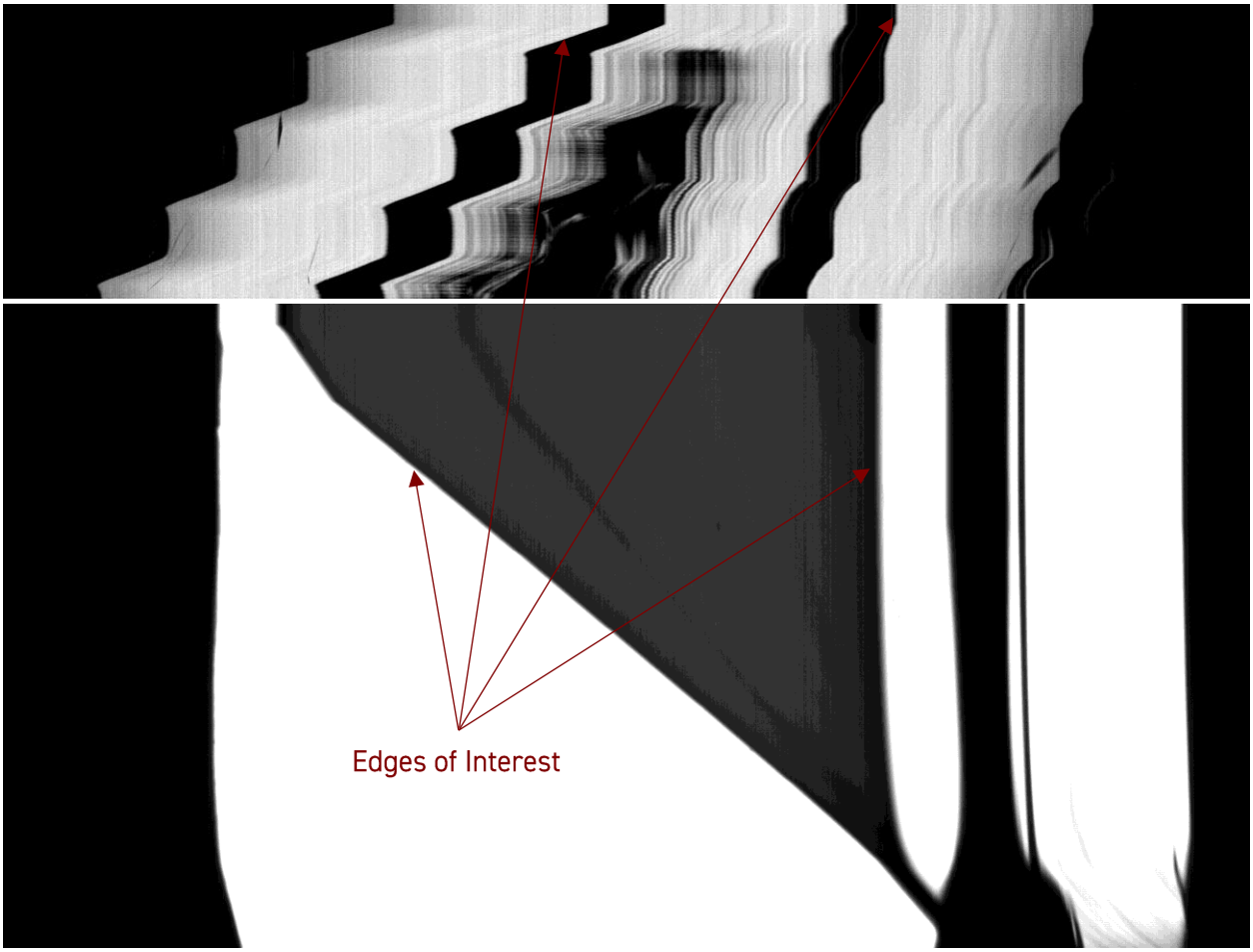

Figure 3 – Edge Identification Challenge

In addition to issues with noise, there is an issue of identifying the appropriate edge for the analysis. Figure 3 shows the correct edges that should be considered the border of the gauge section. An algorithmic solution would have a hard time reconciling the correct edge as it may be observed that there could be multiple gauge sections in these images.

In lieu of algorithmic approaches, it may be simpler to train a neural network to identify the starting and ending positions of the gauge section. The goal of this project is to develop an automated method for identifying the gauge-section edges in line-scan images so that strain vs. time curves can be extracted without manual intervention.

DATASET

Collecting this dataset requires manually processing images and identifying the gauge section. The images for this analysis were provided by Standard Mechanics. To accomplish the task of identifying

the gauge section, a custom GUI was created as seen in Figure 4. After specifying the target edges, an algorithm was applied for more refined positioning. The algorithm works by iterating across the row until the pixel value crosses over a user defined threshold. The threshold is typically around 150 (a gray value between 0 and 255), but this value changes depending on how black the gauge section is or how noisy the white section is.

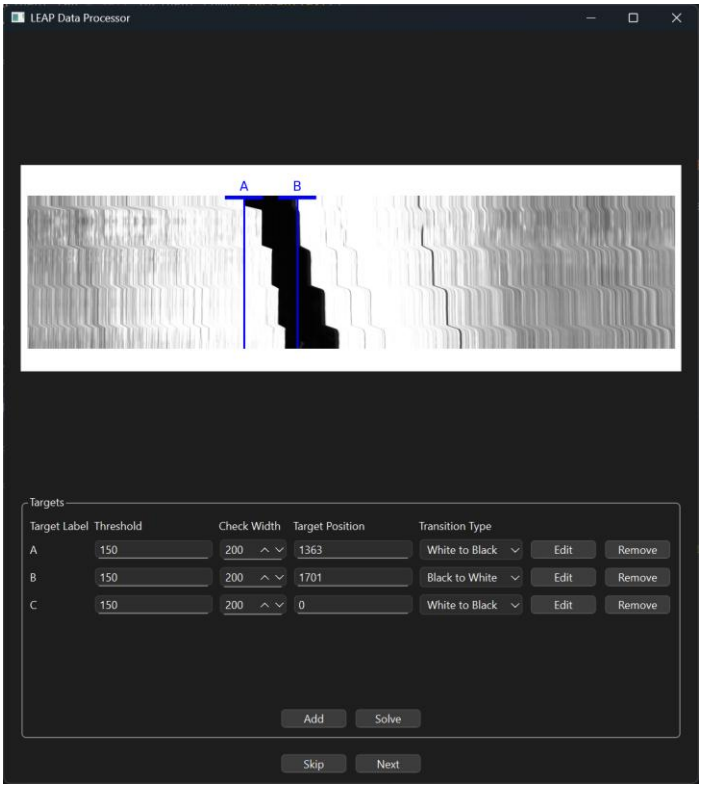

Figure 4 – GUI for Processing Images

The final solution of this process involves propagating down the image, repeating the procedure of finding the pixel position where the pixel value crosses the threshold value. A result of this is seen in Figure 5, which shows a correctly identified edge down the y-axis (time axis) of the image.

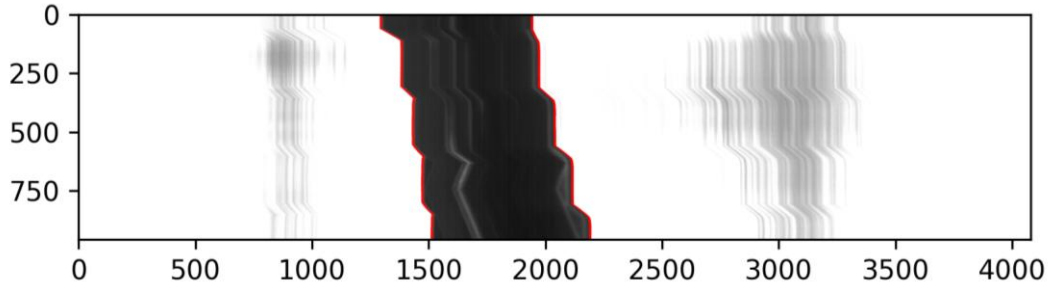

Figure 5 – Detected Edge

In this process, the pixel position where the crossing of the threshold occurs is recorded and fed into a linear regression calculation with its surrounding pixels. This results in a floating-point value for the pixel position. This method of using linear regression is beneficial for sub-pixel resolution.

As such, the final dataset includes 425 images that have a width of 4080 pixels and heights between 960 and 10,000 pixels. Additionally, the edge positions for each row are recorded in CSV files for each image along with the threshold values that were used. Since this model is being limited to identifying two edges, the CSV files contain two columns for each target edge. If there is a desire to have multiple strain targets (which would require marking the specimen with a barcode-like stamp), then a separate model can be made later.

There will be 4080 inputs for this model corresponding to a row of pixel values, and after normalization, the values will be between 0 and 1. Because I have fully resolved the edge for each image, I can make the number of entries to train over quite large as any row can be selected for each image. To begin with, I will select the first row of each image only for the training data, which means that I have a minimum of 425 entries, but can easily make this 42,500 images if I select 100 rows from each image.

MACHINE LEARNING APPROACH

To predict the two-pixel positions corresponding to the gauge-section edges, a fully connected feed-forward neural network will be employed. The network architecture is as follows:

- Input layer: A 1 × 4080 vector of normalized gray-scale intensities (values scaled from 0–1).
- Hidden layers: Two or more dense layers (for example, 1024 and 512 neurons) with ReLU activations to learn non-linear mappings from pixel patterns to edge positions. This portion will be experimented with to determine the best sizing and number of dense layers.
- Output layer: A dense layer with 2 linear neurons, outputting the starting and ending pixel indices along the width axis (values between 0 and 4079).
- Loss function: Mean squared error (MSE), optimized via Adam.
- Regression task: Because the outputs are continuous pixel positions, no activation (e.g., sigmoid) is applied at the output; raw linear values are directly compared against ground-truth labels.

This simple dense architecture is well suited to our fixed-length, one-dimensional input and the low-dimensional regression target. Pretrained convolutional backbones designed for 2D images are ill-suited to our one-dimensional, flattened scan-line input and would require extensive reshaping of both data and weights. Although 1D convolutions could detect local edge features, the fixed input length and global regression goal mean a fully connected network is sufficient. Therefore, all layers will be trained from scratch, avoiding the complexity of adapting incompatible pretrained filters.

All data preparation is managed through the previously mentioned Python GUI, which guides the user through threshold initialization, sub-pixel refinement, and CSV export. Initially, the user selects approximate edge locations on each image; the GUI then refines these positions by iterating along each row and applying local linear regression to determine the precise threshold crossing. Once the two floating-point edge positions per row are computed, the tool writes both the raw gray-scale scan line and its corresponding labels into CSV files. During model training, each row of pixel values is loaded, normalized by dividing by 255 to map intensities into the [0, 1] range, and cast into a tensor of shape [1 × 4080].

EVALUATION METRIC AND CHALLENGES

Model performance will be measured by the mean absolute error (MAE) between predicted and true edge positions, with a target MAE below 10 pixels. Hitting this threshold is sufficient for the downstream sub-pixel refinement algorithm to converge accurately on the final border locations. The primary risks are image noise and the modest size of the initial labeled set; to address these, I will apply on-the-fly noise augmentation, sample additional rows per image, and employ cross-validation to guard against overfitting.