LINE SCAN EXTENSOMETER REPORT

Gabe Morris

INTRODUCTION

High-strain-rate material testing is essential for understanding how materials behave under extreme conditions—such as ballistic impact, industrial crashes, and automotive safety events. In these scenarios, accurately capturing rapid deformation is critical for both design and failure analysis. Line-scan extensometry provides a noncontact optical solution: by painting a high-contrast mark on the specimen and imaging it with a line-scan camera, one can record strain in real time without the slippage or inertial errors that often plague traditional strain gauges at rates above 500 s^{-1}.

Figure 1 – Line Scan Principle

As shown in Figure 1, a black reference mark is drawn across the gauge section, and the grey overlay indicates the camera's scan width. The line-scan camera continuously captures one-dimensional intensity profiles at a fixed sampling rate. Strain is computed by measuring changes in pixel count of the black mark relative to its initial length. When precise displacement tracking of a single edge is required, a pixel-to-distance calibration converts pixel shifts into physical units (e.g., millimeters), enabling the generation of accurate displacement-vs.-time curves.

PROBLEM DESCRIPTION

Despite the precision of line-scan extensometry, post-processing remains a major bottleneck. Each scan produces hundreds to thousands of line profiles, and manually specifying the start and end rows of the

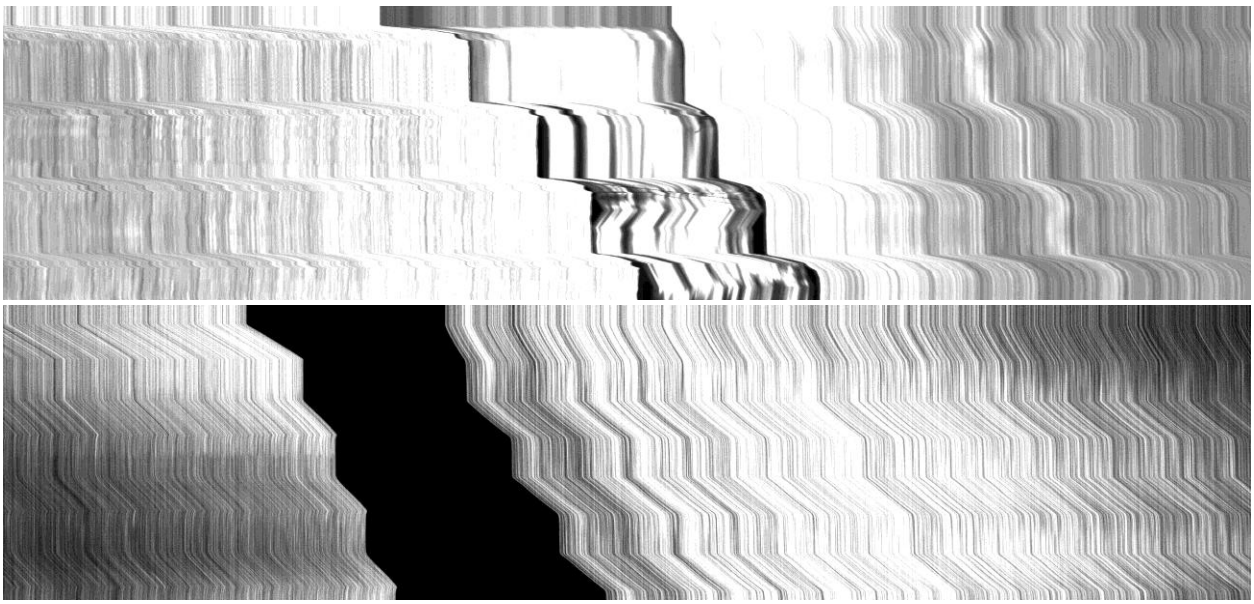

Figure 2 – Example of Noisy Images

gauge section for every image can demand several minutes of operator time per test. This tedious workflow slows data throughput and limits scalability in high volume testing environments.

Moreover, line-scan images frequently exhibit severe noise—stemming from shot noise, speckle from illumination, and motion blur at high strain rates—making edge detection challenging. Figure 2 highlights the overwhelming noise surrounding the gauge section, which degrades the performance of conventional edge-finding methods. An initial algorithmic attempt using Sobel filtering combined with PCA-based noise reduction or Gaussian smoothing yielded inconsistent and noisy boundary estimates.

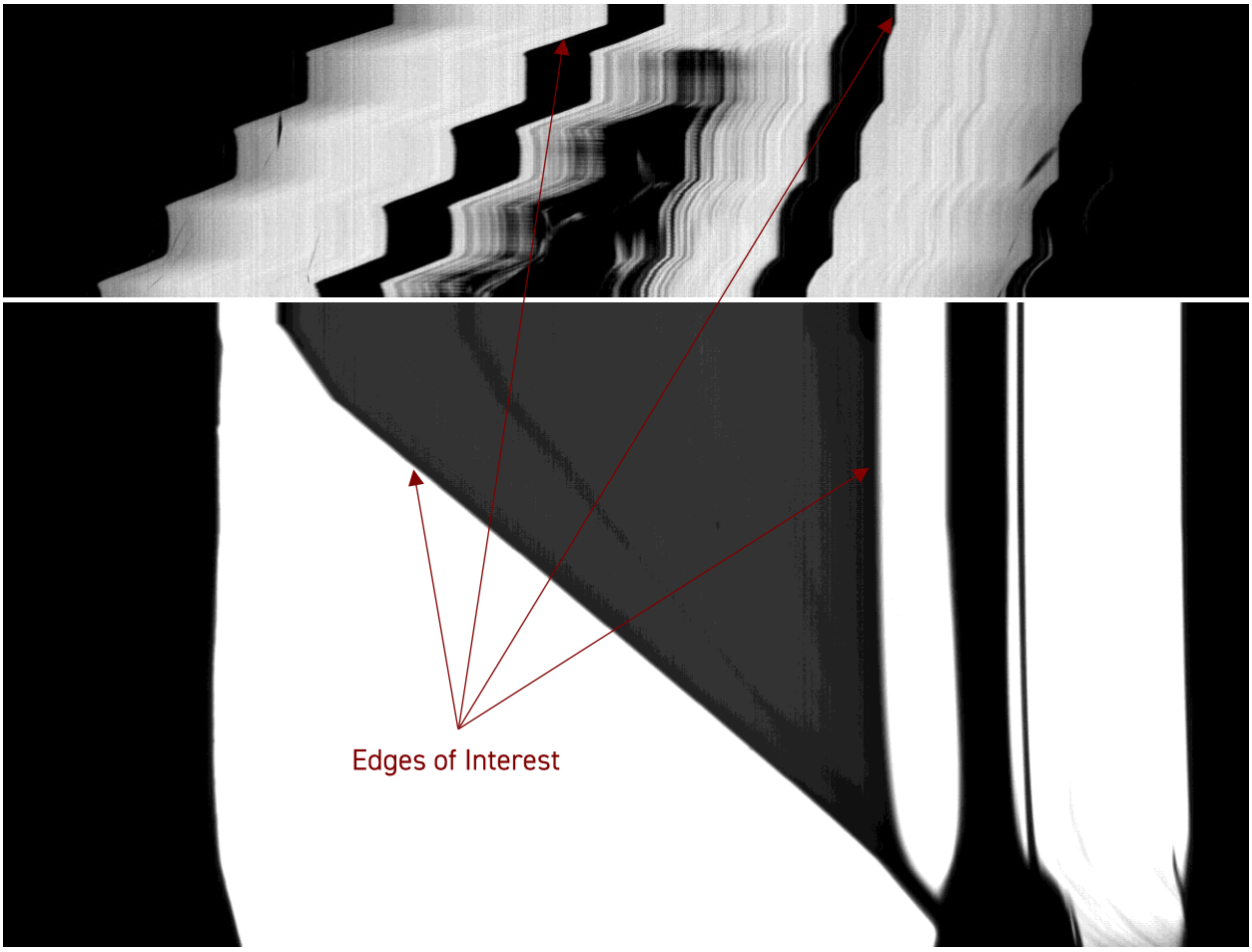

Figure 3 – Edge Identification Challenge

A second hurdle is correctly identifying the gauge-section edges among multiple spurious boundaries. As illustrated in Figure 3, only two specific edges define the true gauge section, yet an algorithmic approach can easily mistake adjacent features or tooling artifacts as valid edges.

Considering these challenges, this project proposes training a machine learning model to predict only the starting and ending guess values for the first row of each line-scan image. These coarse guesses are then fed into a threshold-type algorithmic solver: the solver applies a simple intensity threshold to the first row to locate the precise edge, and for each subsequent row it uses the previously resolved boundary as its initial guess before applying the threshold again. By combining the model's ability to

generate robust initial estimates in noisy data with the solver's deterministic, row-wise threshold refinement, this hybrid workflow minimizes manual input to accelerate the processing throughput.

DATASET

The image dataset was supplied by Standard Mechanics and comprises raw line-scan captures of tensile and compression specimens marked with a high-contrast stripe. To generate ground-truth edge annotations, a custom GUI (Figure 4) was created to specify approximate start and end positions of the gauge section on the first row. Once the two seed points are defined, the threshold-type solver—with a tunable gray-level threshold (typically around 150 on a 0–255 scale)—is applied to that row. The solver scans away from each seed until the pixel intensity crosses the threshold, then refines each boundary via linear regression over its immediate neighborhood to achieve sub-pixel accuracy.

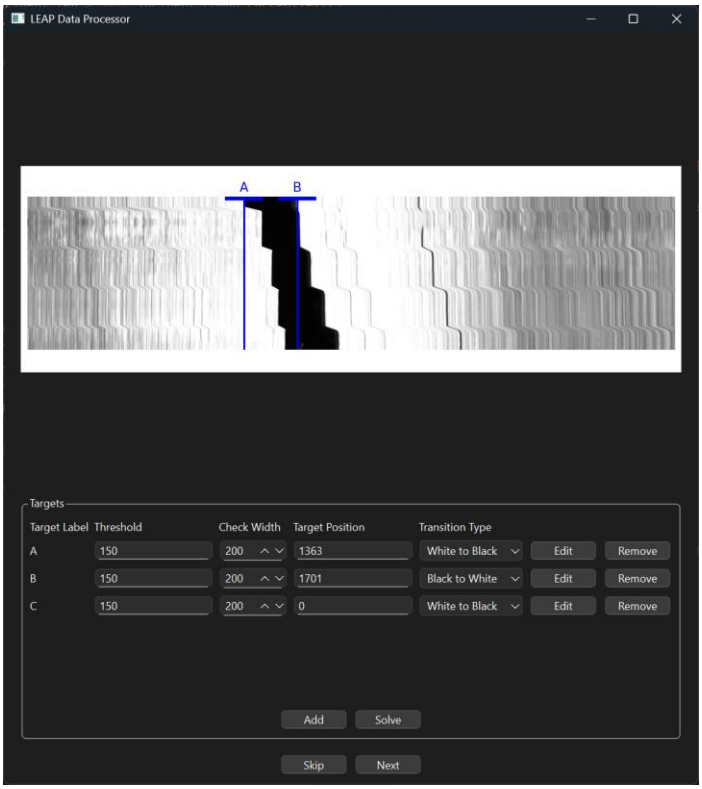

Figure 4 – GUI for Processing Images

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. All edge positions are recorded as floating-point pixel indices (between 0 and 4079) and exported to per-image CSV files, each containing two columns (one for the left edge, one for the right), along with the threshold value used.

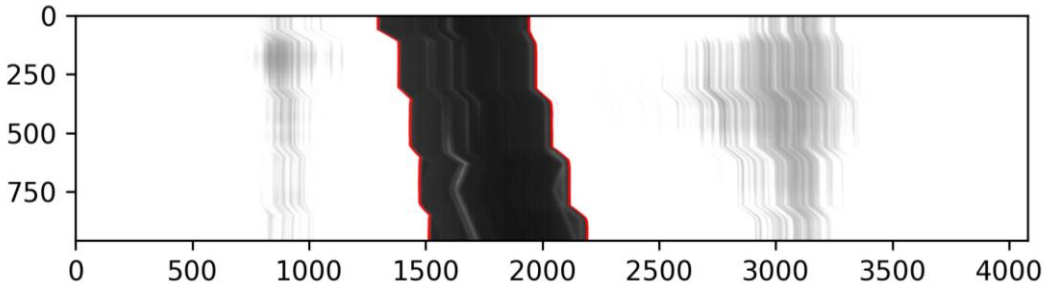

Figure 5 – Detected Edge

The final dataset comprises 425 images with a fixed width of 4,080 px and heights ranging from 960 to 10,000 px. For model training, each row of pixel intensities—normalized to [0, 1]—constitutes one input sample, paired with its two ground-truth edge positions as targets. While sampling only the first row of each image yields 425 examples, selecting multiple rows per image (e.g., 100 rows) expands the training set to 42,500 samples, facilitating robust fitting under diverse noise and contrast conditions.

The dataset thus comprises paired feature–target examples: the feature for each sample is the one-dimensional vector of normalized pixel intensities from a single row of the image, and the target is the corresponding pair of edge positions (left and right) recorded in the CSV file.

METHODS

Data preparation begins by extracting the first row of each 4,080-pixel-wide grayscale image and normalizing its intensity values to the [0, 1] range by dividing by 255. Corresponding ground-truth edge positions—recorded in CSV files and divided by 4,079 for normalization—serve as the regression targets. The dataset was partitioned into 340 training samples and 85 validation samples (an 80/20 split) via stratified random shuffling at the image level. During training, each one-dimensional input was reshaped to (4,080, 1) to satisfy the Conv1D input requirements. The snippet responsible for this is shown below.

```
# Loading data
x_train, x_test, y_train, y_test, train_paths, test_paths = load_data(train_size=0.2,
random_state=42)

# Normalize data
x_train_normal = x_train/255
y_train_normal = y_train/4079
x_test_normal = x_test/255
y_test_normal = y_test/4079

# For convolutional, we need to reshape the feature data
x_train_normal = x_train_normal.reshape(-1, x_train_normal.shape[1], 1)
x_test_normal = x_test_normal.reshape(-1, x_test_normal.shape[1], 1)
```

The core of the prediction pipeline is a one-dimensional convolutional neural network designed to learn local edge features while remaining compact enough for practical deployment. The final model consists of four Conv1D–MaxPooling blocks: the first with 32 filters and an 11-pixel kernel, the second with 64 filters and a 7-pixel kernel, the third with 128 filters and a 5-pixel kernel, and the fourth with 256 filters and a 3-pixel kernel, all using "same" padding and ReLU activations. The output of these blocks is flattened and passed through four fully connected layers of 1,024, 512, 256, and 128 units respectively, each followed by a 10 % dropout layer and ReLU activation. A final dense layer with two linear outputs

produces the normalized left- and right-edge guesses for the first row. The snippet below shows the code for the architecture used.

```
model = Sequential([
    Input(shape=(x_train.shape[1], 1)),

    Conv1D(32, kernel_size=11, padding='same', activation='relu'),
    MaxPooling1D(pool_size=8),

    Conv1D(64, kernel_size=7, padding='same', activation='relu'),
    MaxPooling1D(pool_size=6),

    Conv1D(128, kernel_size=5, padding='same', activation='relu'),
    MaxPooling1D(pool_size=4),

    Conv1D(256, kernel_size=3, padding='same', activation='relu'),
    MaxPooling1D(pool_size=2),

    Flatten(),

    Dense(1024, activation='relu'),
    Dropout(0.1),

    Dense(512, activation='relu'),
    Dropout(0.1),

    Dense(256, activation='relu'),
    Dropout(0.1),

    Dense(128, activation='relu'),
    Dropout(0.1),

    Dense(2, activation='linear')
])
```

Hyperparameter tuning proceeded by iterative trial and error and adjustments to architecture, batch size, and training duration were retained only if they reduced validation mean absolute error (MAE). The Adam optimizer with its default 0.001 learning rate and a batch size of thirty-two yielded the best convergence behavior. Although training was run for 5,000 epochs, inspection of the validation MAE curve in Figure 6 revealed that performance peaked near epoch 1,200, with slight overfitting evident beyond epoch 2,000.

An initial fully connected network comprising nine hidden layers (from 2,048 down to 16 units) was also evaluated but produced a larger saved model (>120 MB) and a validation MAE of approximately 70 pixels. In contrast, the convolutional design reduced model size to under 50 MB while lowering MAE to about 41 pixels. This is primarily because a structure comprising of only dense layers does not have any neighborhood or local continuity, so it cannot learn filters that look only at nearby pixels. By contrast, a 1D-CNN uses convolutional kernels (sliding windows) that explicitly enforce locality and can share weights across the entire line. That both reduces the number of parameters and makes it far easier to learn robust, noise-tolerant edge detectors. This less than satisfactory model of only dense layers is shown in the snippet below.

```
model = Sequential([
    Input(shape=(x_train.shape[1],)),
    Dense(2048, activation='relu'),
    Dense(1024, activation='relu'),
    Dense(512, activation='relu'),
    Dense(256, activation='relu'),
    Dense(128, activation='relu'),
```

```
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(16, activation='relu'),
    Dense(2)
])
```

RESULTS

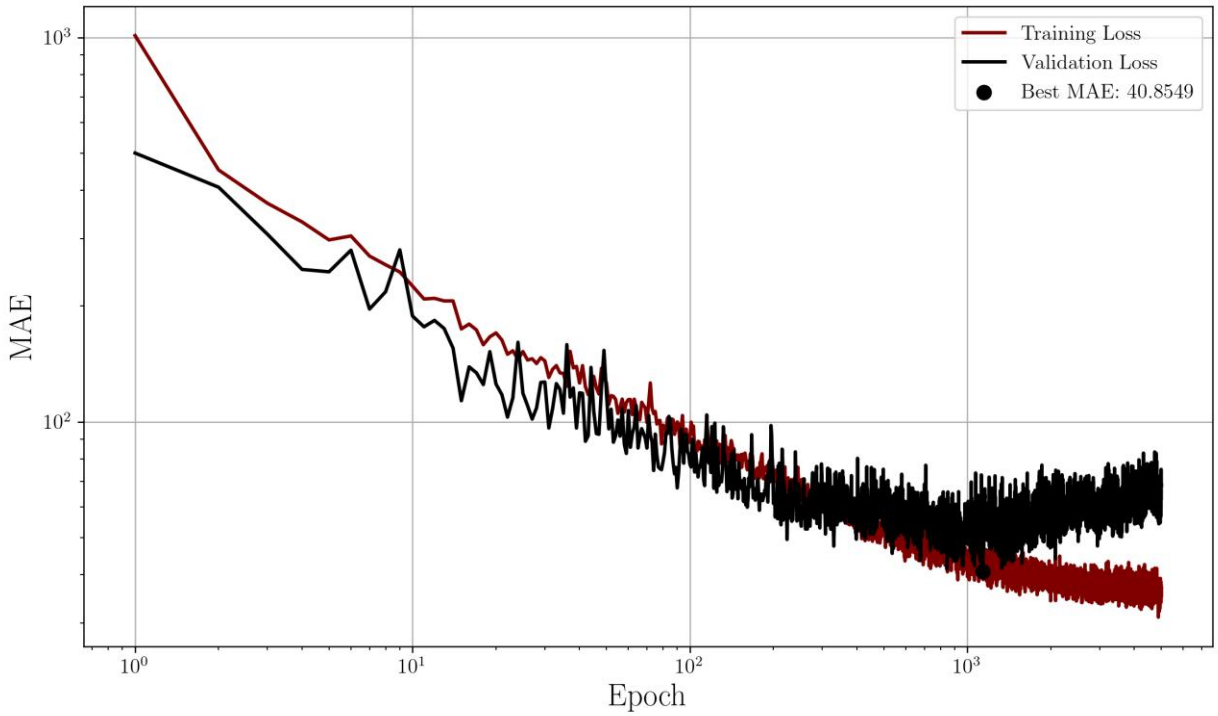

Figure 6 – Results

The convolutional model exhibited a steady decline in both training and validation MAE during early epochs, dropping from an initial validation MAE of 0.1227 (around 500 pixels) at epoch 1 to a minimum of 0.01 (around 41 pixels) at epoch 1,200, as shown in Figure 6. Beyond this point, validation error gradually increased, indicating mild overfitting. The final model file occupies less than 50 MB on disk, making it more suitable for deployment.

DISCUSSION

Despite substantial improvements over purely algorithmic or dense-only approaches, the hybrid workflow retains sensitivity to extreme noise conditions: when background noise overwhelms the gauge-section contrast, the initial guesses can drift, placing greater burden on the threshold solver to recover accurate edges. Moreover, the manual threshold parameter still requires tuning to match lighting and material-specific contrast, and misconfigured thresholds can propagate errors through the row-wise refinement process.

The achieved mean absolute error of approximately 41 pixels represents a sufficient initial guess for the threshold solver to converge reliably. Given that the gauge-section width typically spans several hundred pixels, a 41-pixel deviation remains well within the solver's capture range, allowing the

deterministic refinement routine to attain sub-pixel accuracy on the final edge positions. In practice, this level of initial-guess error translates to negligible impact on the final strain calculation.

To address remaining limitations, future work will explore adaptive threshold prediction, whereby the neural network outputs both seed positions and per-image threshold values, eliminating manual calibration. Expanding the model's output dimensionality to detect multiple gauge edges—enabled by barcode-style specimen markings—would also generalize the approach to multi-target strain measurement. Additionally, incorporating vertical context by feeding adjacent rows as separate input channels may improve robustness against localized artifacts and motion blur.

CONCLUSION

A one-dimensional convolutional regression model has been shown to generate high-quality initial edge estimates (around 41 pixels MAE) for line-scan extensometry, outperforming a dense network baseline and reducing model size. When integrated with a threshold-type solver that refines edges on a row-by-row basis, the hybrid workflow eliminates manual start-and-end specification, accelerates data throughput, and maintains sub-pixel precision. Ongoing efforts will focus on automated threshold selection, multi-edge detection, and leveraging spatiotemporal inputs to further enhance accuracy and adaptability across diverse testing conditions.