Error Bar Detection in Scientific Plots

Technical Report

Shahad Abir Akash

Date: 30 January, 2026

Section 1: Introduction

Scientific publications contain thousands of plots displaying experimental data with error bars that represent measurement uncertainty or statistical variation. Extracting this quantitative information from published figures enables meta-analysis, systematic reviews, and data reuse. However, manually digitizing plots is time-consuming and error-prone, motivating the need for automated detection systems.

1.1 Problem Statement

Error bars in scientific plots encode critical uncertainty information that is often trapped in image format. While the data points themselves can be relatively straightforward to detect, identifying the precise endpoints of error bars presents several challenges:

- Error bars vary widely in visual style (I-bars, T-bars, whiskers)
- Bar sizes range from a few pixels to hundreds of pixels
- Multiple overlapping lines and error bars create visual ambiguity
- Image quality varies due to compression, scanning, and reproduction

This work addresses the problem: **Given a plot image and the pixel coordinates of data points, detect the upper and lower endpoints of each error bar.**

1.2 Dataset Challenges

Our initial dataset consisted of only 150 annotated real scientific plots extracted from publications. This presents three critical challenges:

1. **Insufficient training data**: Modern deep learning approaches typically require thousands of labeled examples
2. **High variance**: Real plots exhibit extreme diversity in resolution (676-2400px width), style, and quality
3. **Annotation inconsistencies**: Approximately 20% of data points in the real dataset have zero-length error bars, indicating missing or ambiguous annotations

To address the data scarcity problem, we developed a synthetic data generation pipeline capable of producing thousands of realistic plot images with precise pixel-level annotations.

1.3 Our Approach

This work presents a two-part solution:

Part 1: Synthetic Dataset Generation

- Implemented matplotlib-based pipeline generating 2,850(total 3k) annotated plot images
- Achieved controlled variation in line count (1-5), point density (4-12 per line), and error bar sizes (3-20% of data range)
- Produced pixel-perfect annotations through coordinate system transformations

Part 2: Error Bar Detection

We developed and compared two detection approaches:

1. Computer Vision Baseline: Classical edge detection with morphological operations
    - Achieved 73.8% / 70.3% accuracy @ 10px threshold on synthetic data (upper/lower bars)
    - Achieved 58.7% / 62.0% accuracy @ 10px threshold on real data
    - Requires no training, robust to varying resolutions
2. Deep Learning Approach: ResNet18-based regression model
    - Crop-based detection (128×384px windows around data points)
    - Pretrained on synthetic, fine-tuned on real data (120 training images)
    - Target performance: 70-80% @ 5px on real data

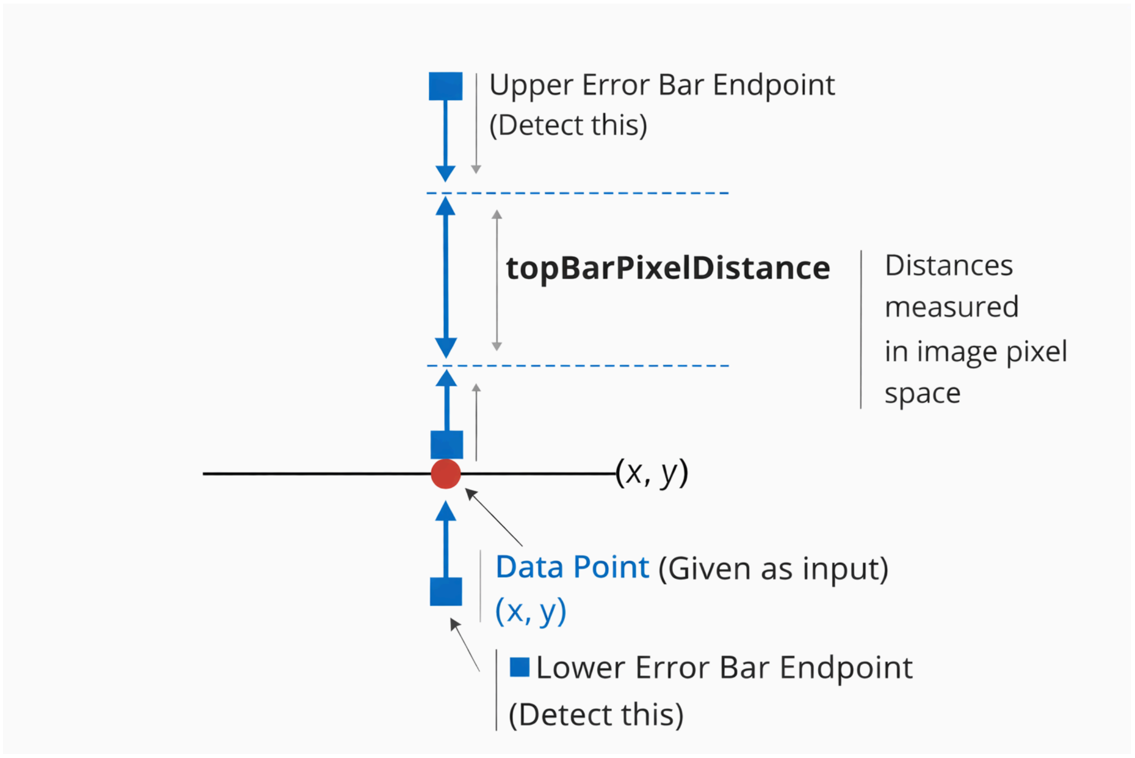


Figure 1: Illustration of the error bar detection task. Given the pixel location of a data point (x,y)(x, y)(x,y), the goal is to detect the upper and lower error bar endpoints and measure their distances in image pixel space.

Section 2: Synthetic Dataset Generation

2.1 Motivation

With only 150 annotated real scientific plots available, training robust machine learning models is infeasible. Modern deep learning approaches typically require thousands of labeled examples to learn generalizable features. To address this data scarcity, we developed an automated pipeline to generate synthetic plot images with pixel-perfect annotations.

Our goals were to:

1. Generate 3,000+ plot images with controlled variation
2. Produce accurate pixel-level annotations for all data points and error bars
3. Introduce sufficient diversity to avoid overfitting
4. Maintain visual realism comparable to real scientific plot

2.2 Generation Pipeline

We implemented the synthesis pipeline using Python's **matplotlib** library, which provides precise control over plot rendering and coordinate transformations. The pipeline consists of five stages:

1. Stage 1: Data Generation
    - Generate random time-series or scatter data with 1-5 lines per plot
    - Each line contains 4-12 data points sampled from smooth functions or random distributions
    - Data values normalized to typical scientific ranges (0-100 for y-axis)
2. Stage 2: Error Bar Calculation
    - For each data point, randomly sample error magnitudes
    - Error size = α × data_range, where α ∈ [0.03, 0.20]
    - Allows both symmetric (50%) and asymmetric (50%) error bars
    - Asymmetry achieved by independently sampling upper and lower errors
3. Stage 3: Plot Rendering
    - Use `matplotlib.pyplot.errorbar()` to render data points with error bars
    - Apply random styling: marker shapes (o, s, ^, D, *, x), line styles (solid, dashed, dotted), colors
    - Add plot elements: grid (55% probability), legend (80%), title (35%), axis labels (85%)
    - Render at varying DPI (120-240) to create resolution diversity
    - Save to PNG format
4. Stage 4: Coordinate Transformation

    The critical challenge is converting matplotlib's data coordinates to exact pixel coordinates. We leverage matplotlib's transformation system:

```
# Get axes transform (data coords → display coords)
transData = ax.transData
# Transform data point to pixel coordinates
point_px = transData.transform((x_data, y_data))
# Transform error bar endpoints
upper_px = transData.transform((x_data, y_data - error_up))
```

```
lower_px = transData.transform((x_data, y_data + error_down))
# Calculate pixel distances (note: Y increases downward in image coords)
topBarPixelDistance = point_px[1] - upper_px[1]
bottomBarPixelDistance = lower_px[1] - point_px[1]
```

This approach ensures sub-pixel accuracy in annotations, crucial for training regression models.

5. Stage 5: Label Export

   Generate JSON files matching the required format:

```
    [
      {
        "label": {"lineName": "Drug_300mg (N=1)"},
        "points": [
          {
            "x": 272.96,
            "y": 351.26,
            "label": "",
            "topBarPixelDistance": 17.77,
            "bottomBarPixelDistance": 13.47,
            "deviationPixelDistance": 0.0
          }
        ]
      }
    ]
```

   Additionally, we add anchor points (labeled "xmin", "ymin", "xmax", "ymax") with zero error bar distances per line, matching the original dataset structure.

2.3 Variation Strategy

To prevent model overfitting and ensure robustness, we introduce variation across multiple dimensions:

Table 1: Synthetic Data Variation Parameters:

Parameter	Range/Options	Implementation	Purpose
Lines per image	1–5	Uniform random	Multi-line complexity
Points per line	4–12	Uniform random	Density variation
Error bar size	3–20% of y-range	Uniform α	Scale diversity
Upper/lower ratio	0.5–2.0	Independent sampling	Asymmetry realism
Marker style	o, s, ^, D, *, x, +	Random choice	Visual variation
Line style	solid, dashed, dotted	Random choice	Style diversity
Line color	10 distinct colors	Predefined palette	Multi-line distinction
Grid presence	55%	Bernoulli	Background clutter
Legend presence	80%	Bernoulli	Text overlay realism
Title presence	35%	Bernoulli	Header variation
Axis labels	85%	Bernoulli	Label variation
Background tint	25%	Paper color overlay	Real publication look
Background noise	35%	Gaussian noise	Scan artifact simulation
DPI	120–240	Uniform random	Resolution diversity

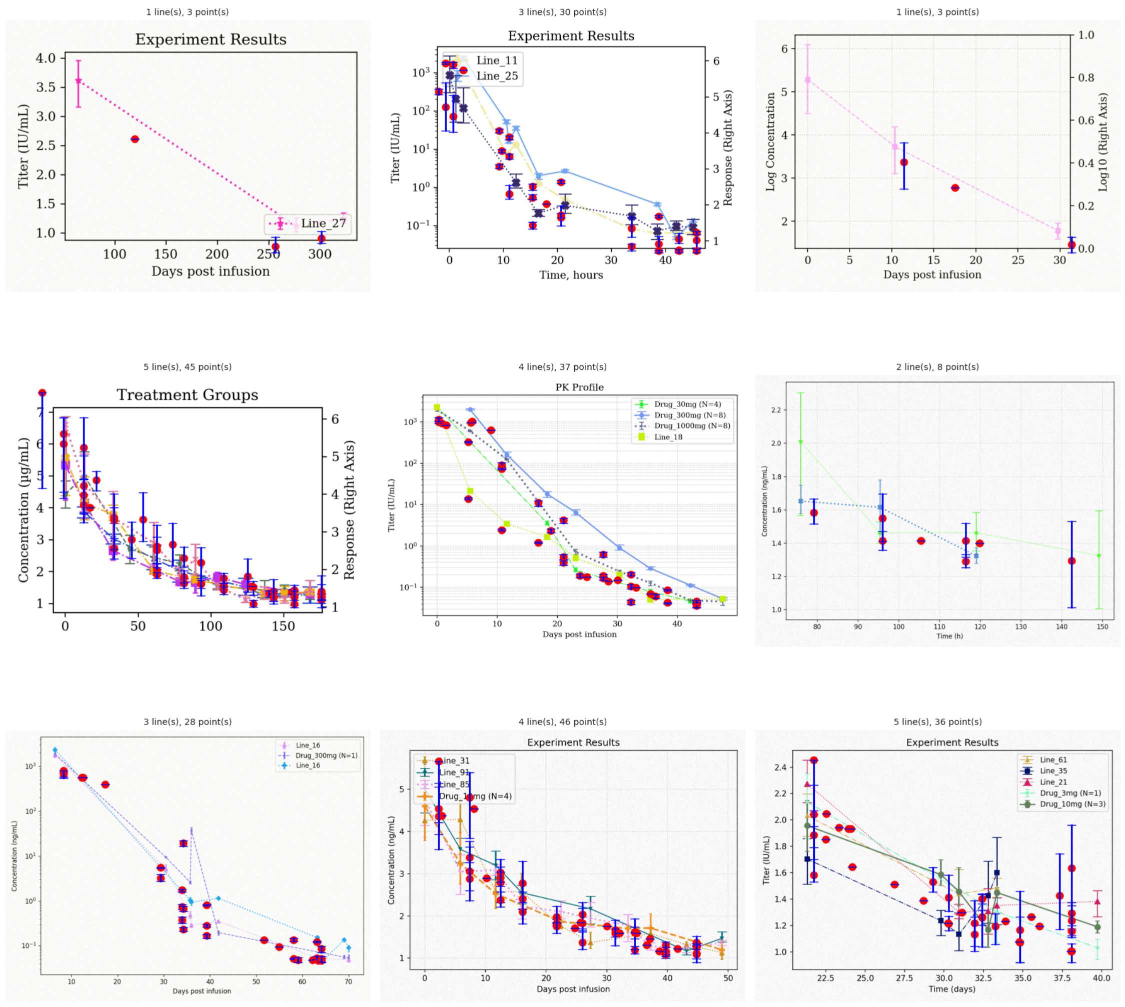


Figure 2: Sample synthetic plots showing variation in line count, styles, and error bar sizes.

2.4 Dataset Statistics

The final synthetic dataset contains:

- 2,850 images (target was 3,000; ~95% completion)
- 72,523 data points across 8,667 lines
- Resolution: ~960×720 pixels (varies by DPI: 919-1151 × 689-863)
- File size: ~50-150 KB per image (PNG compression)

Error Bar Characteristics:

- Upper bar distance: mean 14.4px, median 5.9px, range 0-224px

- Lower bar distance: mean 16.1px, median 6.9px, range 0-317px
- Total bar length: mean 28.9px, median 13.3px
- Symmetry: 49.6% perfectly symmetric (< 1px difference), 81.7% nearly symmetric (< 5px)

Plot Structure:

- Lines per image: mean 3.0 ± 1.4 (range 1-5)
- Points per image: mean 25.4 ± 14.7
- Points per line: mean 8.4 ± 2.8 (range 2-16)

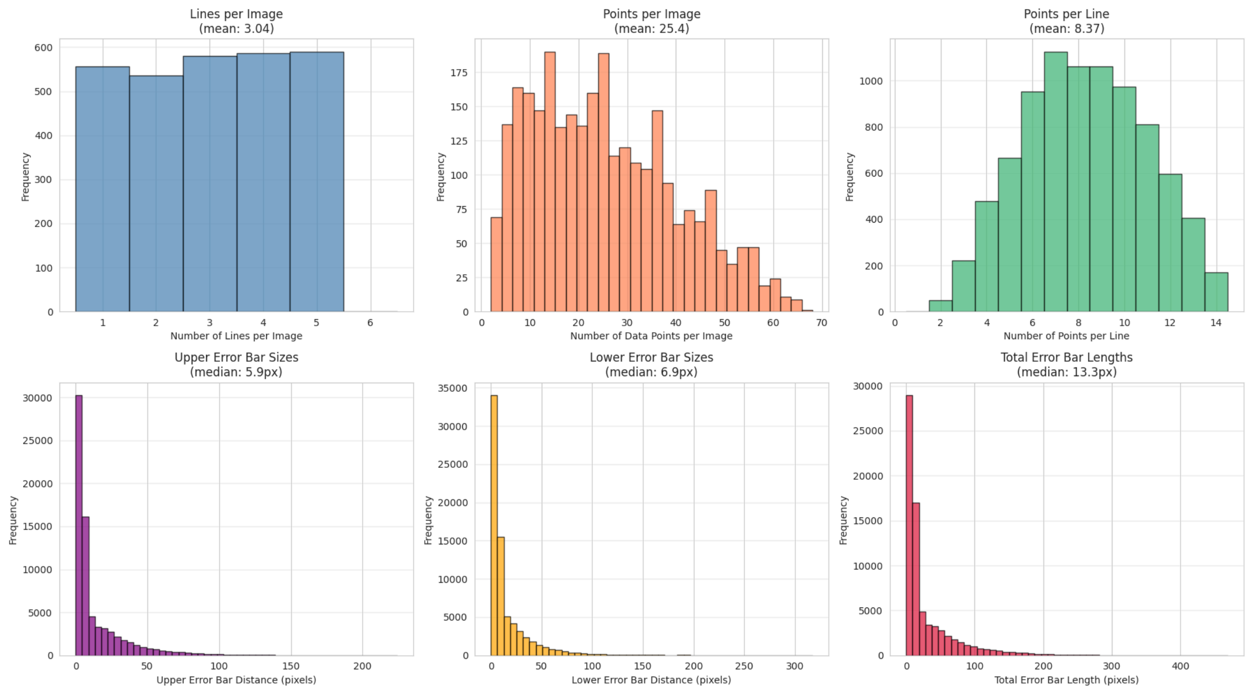


Figure 3: Distribution of key dataset characteristics.

2.5 Quality Assurance

We validated dataset quality through multiple checks:

1. Visual Inspection
    a. Manually reviewed 50 random samples

    b. Verified error bars visually match annotations
    c. Confirmed no rendering artifacts or coordinate misalignments

2. Coordinate Accuracy
    a. Implemented round-trip transform test: pixel → data → pixel
    b. Verified coordinate precision < 0.5 pixels (sub-pixel accuracy)
    c. All 2,850 images passed validation
3. Distribution Analysis
    a. Confirmed uniform distribution of line counts (1-5 roughly equal)
    b. Error bar sizes follow realistic distribution (more small bars than large)
    c. No pathological outliers or generation artifacts
4. Format Compliance
    a. All 2,850 label files parse successfully
    b. Schema matches original dataset exactly
    c. Pairing verified: every image has matching label file

2.6 Limitations and Realism

While our synthetic data provides scale and perfect annotations, it has inherent limitations:

Differences from Real Plots:

1. Cleaner appearance: No scan artifacts, compression, or paper texture
2. Simpler backgrounds: Real plots have more complex legends, annotations, inset plots
3. Error bar styles: Limited to standard matplotlib styles; real plots have custom designs
4. Size distribution: Synthetic bars smaller (median 6px vs 15px in real data)

These limitations create a domain gap that impacts model transferability. We quantify this gap in Section 4.2 and address it through fine-tuning on real data.

Section 3: Detection Methods

3. Error Bar Detection Approaches

We developed two approaches: a classical computer vision pipeline and a deep learning model. Both take an image and data point (x_0, y_0) as input and predict upper/lower error bar endpoint coordinates.

3.1 Computer Vision Approach

Core Algorithm: Point-centric vertical search with edge analysis

1. Preprocessing: Remove grid lines (morphological ops), denoise (bilateral filter), segment ink (adaptive threshold), detect edges (Canny)
2. Stem Detection: Use connected components on vertically-closed binary masks. Score by height and proximity to (x_0, y_0)). Key constraint: Component must contain ink at row y_0 near x_0 (eliminates false positives from plot borders).
3. Endpoint Estimation: Walk vertically from a data point in both directions. Stop when 4 consecutive rows lack edge pixels in ROI around x_0 (half-width = 6 px).
4. Cap Refinement: Search ±6 rows around endpoints for horizontal ink (T-bar caps). Typically adjusts endpoints by 2–8 pixels.

Key Parameters: half_w = 6 px (stem width tolerance), max_search = 280 px (vertical range), miss_threshold = 4 (stopping criterion)

Assumptions:

- Error bars are primarily vertical
- Stems have continuous edge pixels
- Data point lies within the error bar stem

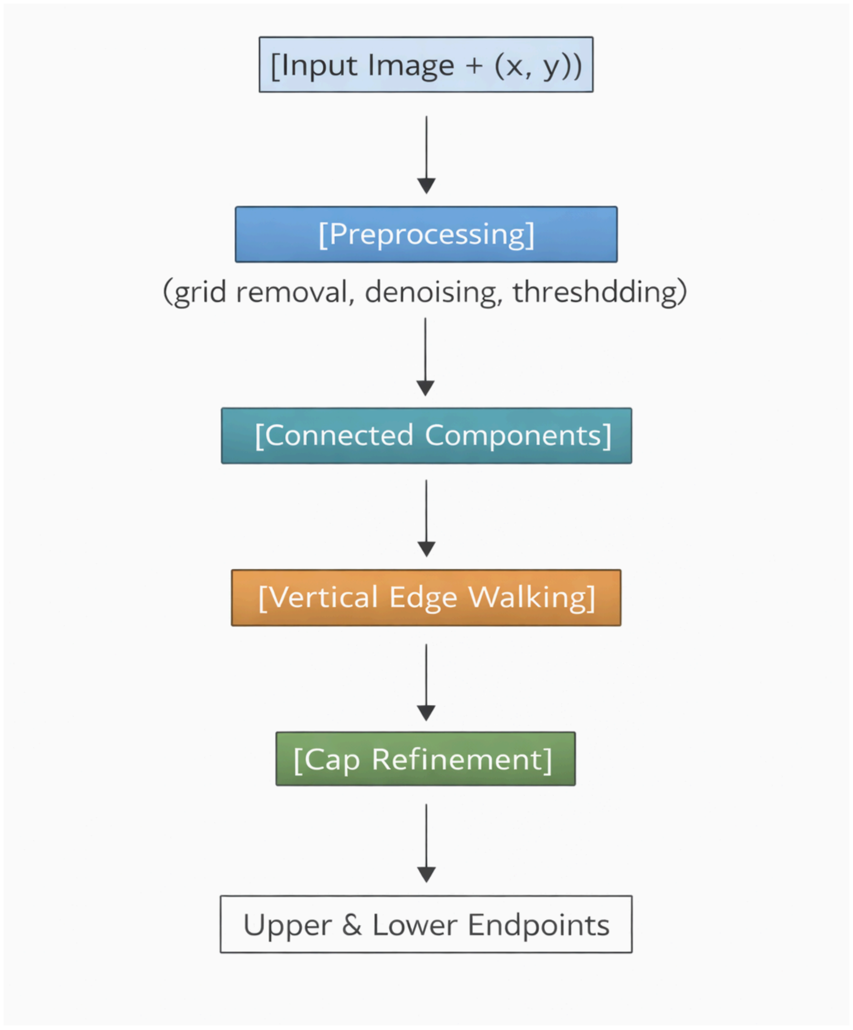


Figure 4: CV pipeline diagram (Input → Preprocess → Component Detection → Edge Walking → Cap Refinement)

3.2 Deep Learning Approach

We use a ResNet-18 backbone pretrained on ImageNet, followed by custom fully connected layers

Training Strategy:

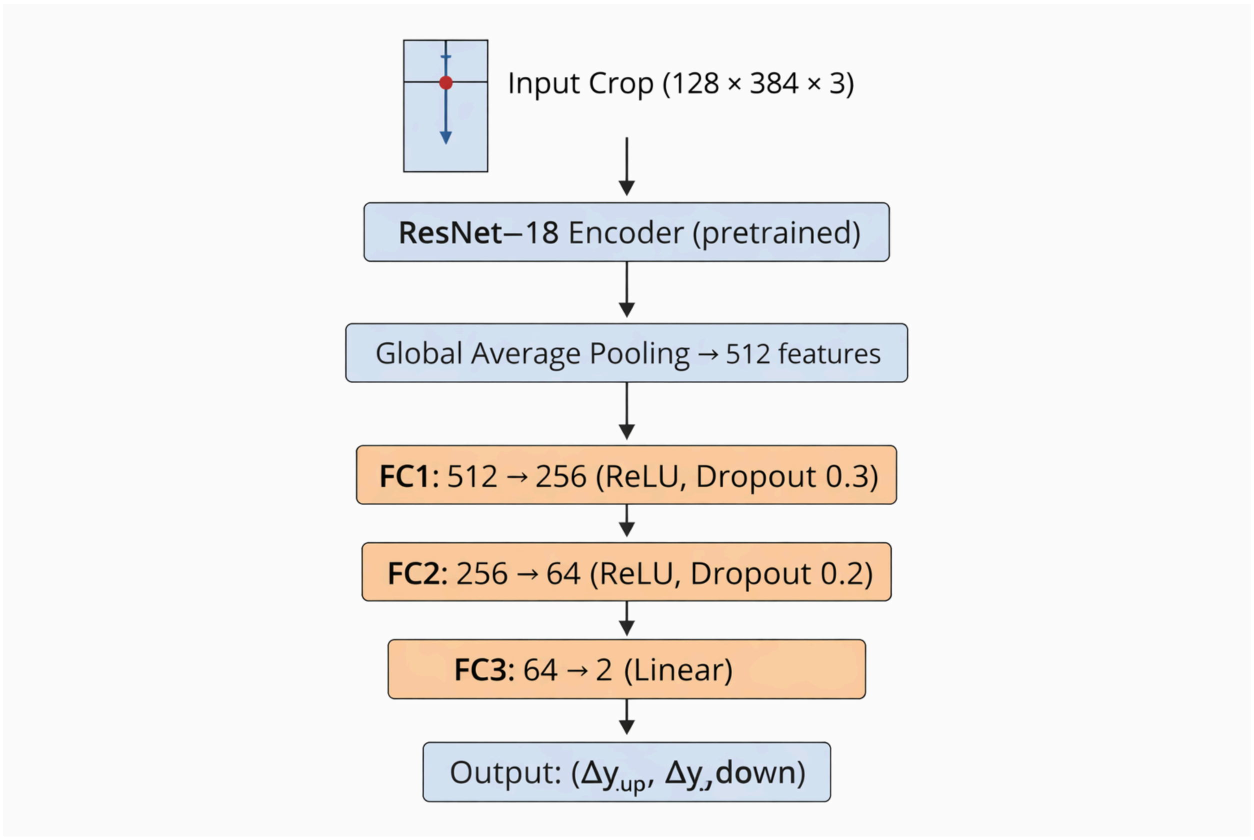
- Stage 1: Pretrain on 2850 synthetic images (6 epochs, lr = 1e-4, SmoothL1Loss)
- Stage 2: Fine-tune on 120 real images (10 epochs, lr = 1e-5)

Input: 128 × 384 px crop centered at data point (±192 px vertical, ±64 px horizontal context)

Rationale: Two-stage training leverages synthetic abundance while adapting to real-world visual characteristics (compression, noise, hand-drawn elements).

Assumptions:

- Error bars fit within 384 px vertical span
- ImageNet pretraining provides useful low-level features
- Synthetic → real transfer learning is viable



Input Crop (128 × 384 × 3)
ResNet–18 Encoder (pretrained)
Global Average Pooling → 512 features
FC1: 512 → 256 (ReLU, Dropout 0.3)
FC2: 256 → 64 (ReLU, Dropout 0.2)
FC3: 64 → 2 (Linear)
Output: (Δy_up, Δy,down)

Figure 5: ML architecture (Crop → ResNet18 → FC layers → Offsets)

Section 4: Experimental Results

We evaluated both methods on synthetic (2,850 images, 80/20 split) and real (150 images, 80/20 split) datasets. Performance is measured by percentage of predictions within 10 px of ground truth, the standard tolerance for pixel-level annotation tasks.

4.1 Quantitative Performance

Table 1: Detection Accuracy (% within threshold)

Method	**Synthetic 2px**	**Synthetic 5px**	**Synthetic 10px**	**Real 2px**	**Real 5px**	**Real 10px**
CV Upper	46.0%	58.2%	73.8%	27.5%	37.5%	58.7%
CV Lower	44.1%	54.1%	70.3%	28.9%	42.1%	62.0%
ML (ResNet)	25.3%	58.8%	78.1%	9.8%	29.8%	66.1%

Table 2: Error Statistics (pixels)Key Observations:

Method	**Synthetic Mean**	**Synthetic Median**	**Real Mean**	**Real Median**
CV Upper	10.24	3.00	18.96	7.29
CV Lower	11.61	3.87	17.61	6.37
ML Combined	8.13	4.73	15.99	8.84

Evaluation Details:

- Synthetic: 300 random images (11,908 points) for CV, full validation set for ML
- Real: 150 images (6,815 points) for CV, 30 held-out test images for ML

Key Observations:

- CV excels at 2–5 px precision (46%/58% on synthetic vs ML 25%/59%)

- ML catches up at 10 px (78% synthetic) due to learned global context
- Both degrade on real data: CV drops 15–20%, ML drops more severely (60% → 30% @ 5 px)

4.2 Domain Gap Analysis

Table 3: Synthetic vs. Real Dataset Characteristics:

Property	**Synthetic**	**Real**
Total images	2,850	150
Total data points	72,523	4,839
Unique resolutions	1 (960×720)	149
Median bar length (upper)	5.9 px	15.0 px
Median bar length (lower)	6.9 px	11.2 px
Mean bar length	14.4 / 16.1 px	29.9 / 27.7 px
Zero-length bars	0%	20.0%
Lines per image	3.0 ± 1.4	3.3 ± 2.8

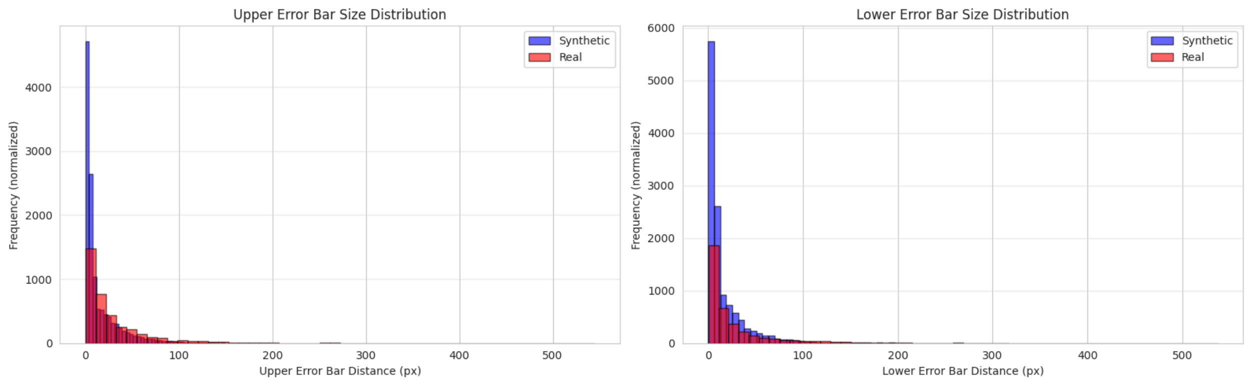


Figure 6: Domain gap visualization between real and synthetic datasets showing upper and lower error bar length distributions.

Key Domain Shifts:

- Error bars 2× larger in real data (15 px vs 6 px median) → larger search windows needed
- 149 unique resolutions vs uniform 960×720 → scale variation challenges
- 20% zero-length bars in real data → detector predicts non-zero, causing errors
- Higher visual complexity: real plots have compression artifacts, hand-drawn elements, varied grid densities

Why CV > ML?

- Data scarcity: 120 real training images insufficient for deep learning
- Explicit structure: CV encodes vertical edge priors that ML must learn from scratch
- Interpretability: CV parameters tunable per failure mode; ML is black-box

4.3 Qualitative Analysis

Success Cases (CV pipeline):

- Clean synthetic plots: 2–4 px errors
- Real plots with thin, solid bars and minimal grid interference

Failure Modes:

- Overlapping bars (adjacent points): Component detection picks wrong stem → 50–100 px errors
- Thick stems (>12 px): Exceed ROI width → misses edges
- Zero-length bars (20% of real data): Predicts non-zero → counted as error
- Plot borders: Thick frames near points occasionally misidentified

ML-specific failures:

- Crops cutting off long error bars (>192 px not visible in 384 px window)
- Extreme out-of-distribution styles (3D plots, unusual color schemes)

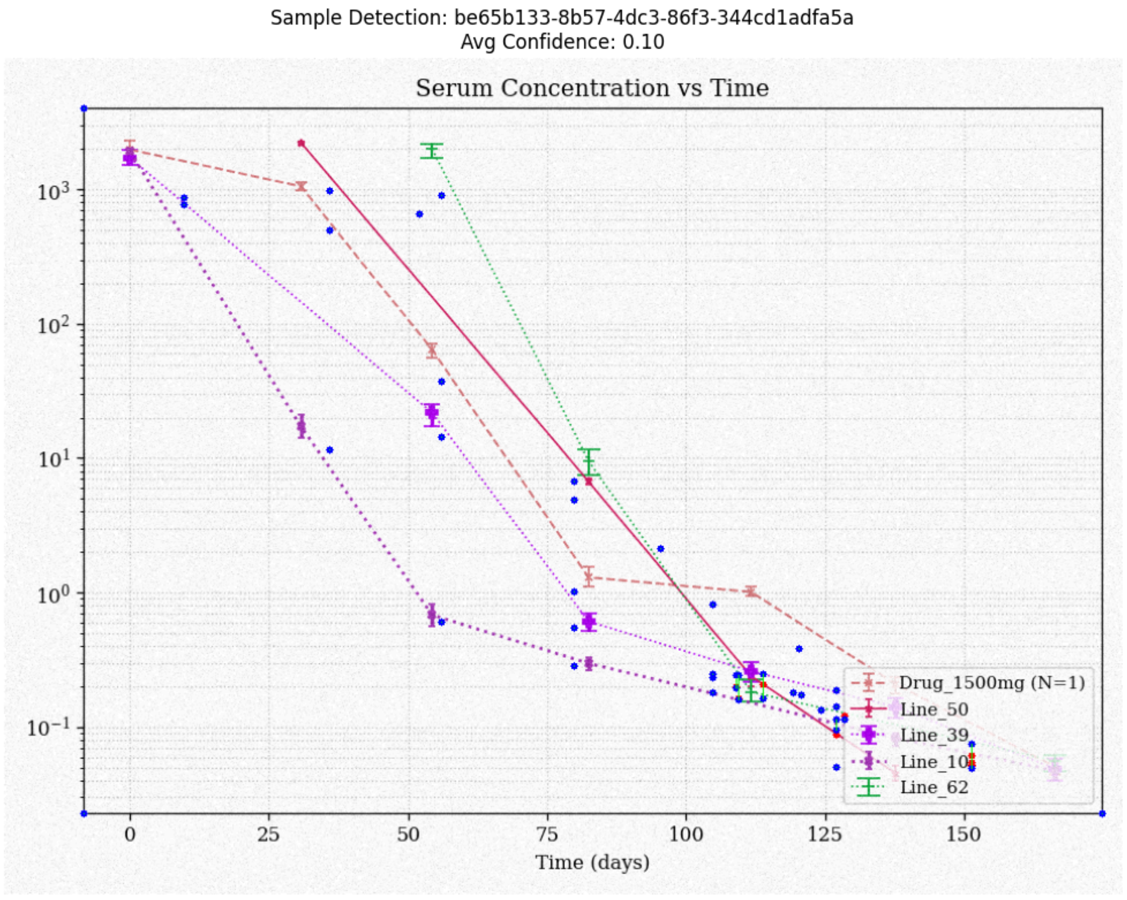


Figure 7: Qualitative detection results on real plots. Red circles indicate input data point locations provided to the algorithm. Blue circles mark the detected upper and lower error bar endpoints, and green line segments connect the predicted endpoints. Examples are chosen to illustrate varying confidence scores and both successful detections and failure cases.

Section 5: Conclusion

We developed a complete pipeline for error bar detection in scientific plots, from synthetic data generation to two complementary detection approaches.

Key Contributions:

- **Synthetic dataset:** 2,850 plots (72,523 labeled points) with pixel-perfect labels via matplotlib inverse transforms
- **CV pipeline:** 74%/70% accuracy on synthetic, 59%/62% on real @ 10 px using morphological operations
- **Empirical finding:** Classical CV outperforms deep learning (ResNet-18) by 8–16% at 2–5 px precision in low-data regimes

Limitations:

- Performance drops 15–20% on real data due to domain gap (resolution variation, 2× larger bars, compression artifacts)
- Assumes vertical error bars (horizontal not handled)
- 20% of real bars have zero length → always counted as errors
- Requires parameter tuning for different image styles

Future Directions:

- **Close domain gap:** Generate synthetic data at varying resolutions (640–2400 px) and with realistic backgrounds extracted from real plots
- **Hybrid approach:** Use CV for candidate stems + ML for endpoint refinement. Combines explicit structure with learned features.
- **Handle zero-length bars:** Add classifier to detect horizontal lines before regression
- **Multi-scale detection:** Adaptive ROI width based on local stem thickness (currently fixed 12 px)
- **Active learning:** Train on 120 real images, identify low-confidence predictions, label 30 more, retrain → likely improve ML to 40–50% @ 5 px

Takeaway: In specialized domains with <200 training examples, well-designed classical CV with explicit domain priors outperforms end-to-end deep learning. The 10–15% performance gap demonstrates the value of encoding assumptions about vertical structure, connected components, and edge continuity.