Juan Ali Ruiz Guzman
EE345
24 September 2025

Automatic Thresholding

Project Overview

This project utilized knowledge about histographs and binary conversions that was learned from a previous project and expanded by adding thresholding values and Variance Plots. To get the desired results from three input images given to us, a python program was developed to automatically determine the best threshold value to perform the binary conversion.

Code Used To Generate Results

```
#!/usr/bin/env python3
"""

Project Name: Automatic Thresholding
Class: EE345
Date: 24 September 2025
Author: Juan Ali Ruiz Guzman
Version: 1.2 -Fixed Output Image Bug where only binary image would save instead of side to side
File Name: thresholding.py

Code Overview
---------------
Automatic thresholding by minimizing within-class variance (Otsu-style),
restricted to thresholds in [25, 230] by default.

- Loads one or more images from the command line.
- Converts to grayscale.
- Computes histogram H(x) and probability distribution p(x).
- Sweeps thresholds 25..230 to compute σw²(t).
- Finds the optimal threshold t* that minimizes σw²(t).
- Saves:
    <stem>_hist.png         : histogram with threshold marked
    <stem>_variance.png     : σw²(t) vs threshold plot
    <stem>_binary.png       : thresholded image
    <stem>_before_after.png : side-by-side comparison
"""

import argparse
import os
import numpy as np
from PIL import Image
```

```
import matplotlib
matplotlib.use("Agg")  # Used to just save as PNGs
import matplotlib.pyplot as plt


# ---------------- Math Section of the Code ---------------- #

def compute_histogram_and_probs(gray: np.ndarray):
    """
    Compute histogram H(x) and probability distribution p(x).
    - H[x] = number of pixels with intensity x (0–255).
    - p[x] = normalized probability (H[x] / total_pixels).
    """
    H, _ = np.histogram(gray, bins=256, range=(0, 255))
    total = H.sum()
    p = H / total if total > 0 else np.zeros_like(H, dtype=float)
    return H.astype(np.int64), p


def within_class_variance_curve(p: np.ndarray, tmin: int, tmax: int):
    """
    Compute σw²(t) curve, restricted to [tmin,tmax].
    σw²(t) = q0*σ0² + q1*σ1²
    - q0, q1: probabilities of background/foreground classes
    - σ0², σ1²: variances of background/foreground classes
    """
    x = np.arange(256, dtype=float)
   # Cumulative sums allow fast computation of means and variances
    P = np.cumsum(p)              # cumulative probability up to intensity x
    M = np.cumsum(x * p)          # cumulative first moment (mean numerator)
    S = np.cumsum((x**2) * p)     # cumulative second moment (variance numerator)

   # Background stats (pixels 0..t)
    q0 = P
    mu0 = np.divide(M, q0, out=np.zeros_like(M), where=q0 > 0)       # background mean
    E0x2 = np.divide(S, q0, out=np.zeros_like(S), where=q0 > 0)      # E[x^2] background
    var0 = np.where(q0 > 0, E0x2 - mu0**2, 0.0)                      # σ0²

    # Foreground stats (pixels t+1..255), computed as totals - background
    q1 = 1.0 - P
    M1 = M[-1] - M
    S1 = S[-1] - S
    mu1 = np.divide(M1, q1, out=np.zeros_like(M1), where=q1 > 0)     # foreground mean
    E1x2 = np.divide(S1, q1, out=np.zeros_like(S1), where=q1 > 0)    # E[x^2] foreground
    var1 = np.where(q1 > 0, E1x2 - mu1**2, 0.0)                      # σ1²


    # Within-class variance curve across all thresholds
    sigma_w2 = q0 * var0 + q1 * var1

    # Only keep values in the range [tmin,tmax], NaN elsewhere (for plotting clarity)
    curve = np.full(256, np.nan, dtype=float)
    curve[tmin:tmax+1] = sigma_w2[tmin:tmax+1]
    return curve
```

```
def find_optimal_threshold(curve: np.ndarray, tmin: int, tmax: int) -> int:
    """
    Find the threshold t* in [tmin,tmax] that minimizes σw²(t).
    """
    window = curve[tmin:tmax+1]              # restrict to allowed window
    t_local = int(np.nanargmin(window))      # index of min variance
    return tmin + t_local                    # shift back to global threshold


def apply_threshold(gray: np.ndarray, t: int) -> np.ndarray:
    """
    Apply binary threshold: pixels >= t → 255 (white), else 0 (black).
    """
    return np.where(gray >= t, 255, 0).astype(np.uint8)


# ---------------- Plotting helpers ---------------- #

def save_histogram(H, t, out_png):
    """Save histogram of pixel intensities with threshold marked."""
    plt.figure(figsize=(8, 4), dpi=120)
    plt.bar(np.arange(256), H, width=1.0)
    plt.axvline(t, linestyle="--", color="red")
    plt.title(f"Histogram (t* = {t})")
    plt.xlabel("Intensity (x)")
    plt.ylabel("Count H(x)")
    plt.tight_layout()
    plt.savefig(out_png)
    plt.close()


def save_variance(curve, t, out_png):
    """Save plot of within-class variance vs threshold with minimum marked."""
    plt.figure(figsize=(8, 4), dpi=120)
    plt.plot(np.arange(256), curve)
    plt.axvline(t, linestyle="--", color="red")
    plt.xlim([0, 255])
    plt.title(r"Within-Class Variance $\sigma_w^2(t)$ vs Threshold")
    plt.xlabel("Threshold t")
    plt.ylabel(r"$\sigma_w^2(t)$")
    plt.tight_layout()
    plt.savefig(out_png)
    plt.close()


def save_before_after(gray, binary, out_png):
    """
    Save a side-by-side comparison of the original grayscale and binary image.
    A small gray separator line is added in between.
    """
    g3 = np.stack([gray, gray, gray], axis=-1)      # make grayscale into 3-channel
    b3 = np.stack([binary, binary, binary], axis=-1)
```

```
    sep = np.full((g3.shape[0], 2, 3), 128, dtype=np.uint8)  # vertical separator
    side_by_side = np.concatenate([g3, sep, b3], axis=1)
    Image.fromarray(side_by_side).save(out_png)


# ---------------- Main pipeline ---------------- #

def process_image(path, outdir, tmin, tmax):
    """
    Process a single image:
    - Convert to grayscale
    - Compute histogram + variance curve
    - Find optimal threshold
    - Save plots and binary image
    """
    gray = np.array(Image.open(path).convert("L"), dtype=np.uint8)
    H, p = compute_histogram_and_probs(gray)
    curve = within_class_variance_curve(p, tmin, tmax)
    t_star = find_optimal_threshold(curve, tmin, tmax)
    binary = apply_threshold(gray, t_star)

    stem = os.path.splitext(os.path.basename(path))[0]
    save_histogram(H, t_star, os.path.join(outdir, f"{stem}_hist.png"))
    save_variance(curve, t_star, os.path.join(outdir, f"{stem}_variance.png"))
    Image.fromarray(binary).save(os.path.join(outdir, f"{stem}_binary.png"))
    save_before_after(gray, binary, os.path.join(outdir, f"{stem}_before_after.png"))

    # Print result to terminal for reference
    print(f"- {path}")
    print(f"  Optimal threshold t* = {t_star}")


def main():
    """Parse command-line arguments and run the pipeline on each image."""
    parser = argparse.ArgumentParser(description="Automatic thresholding by minimizing within-class variance.")
    parser.add_argument("images", nargs="+", help="Path(s) to image file(s).")
    parser.add_argument("--tmin", type=int, default=25, help="Minimum threshold (default 25).")
    parser.add_argument("--tmax", type=int, default=230, help="Maximum threshold (default 230).")
    parser.add_argument("--outdir", type=str, default="results", help="Output directory (default: results/).")
    args = parser.parse_args()

    os.makedirs(args.outdir, exist_ok=True)  # make sure output folder exists
    for img in args.images:
        try:
            process_image(img, args.outdir, args.tmin, args.tmax)
        except Exception as e:
            print(f"! Error processing {img}: {e}")


if __name__ == "__main__":
    main()
```

Results

Image 1:

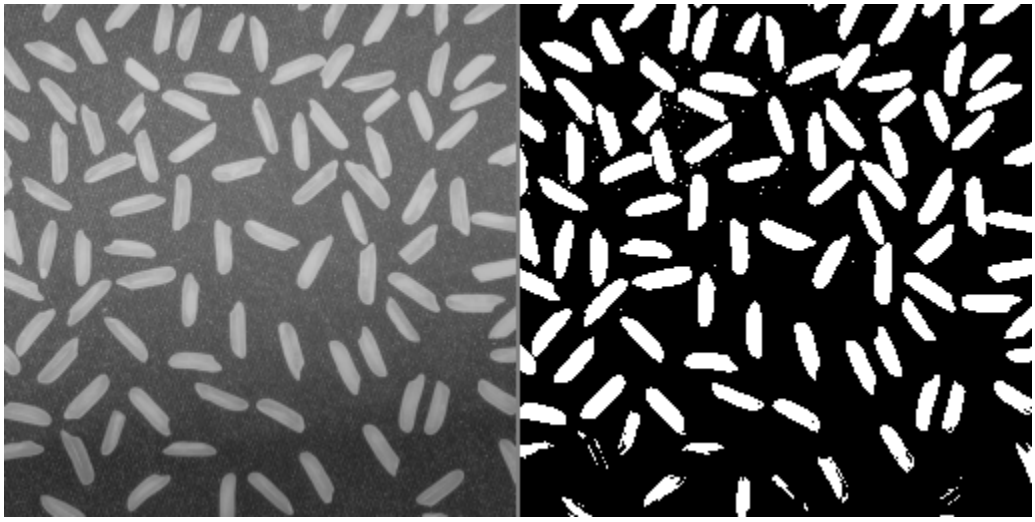

Figure 1: Before and After of Binary Conversion of Image 1

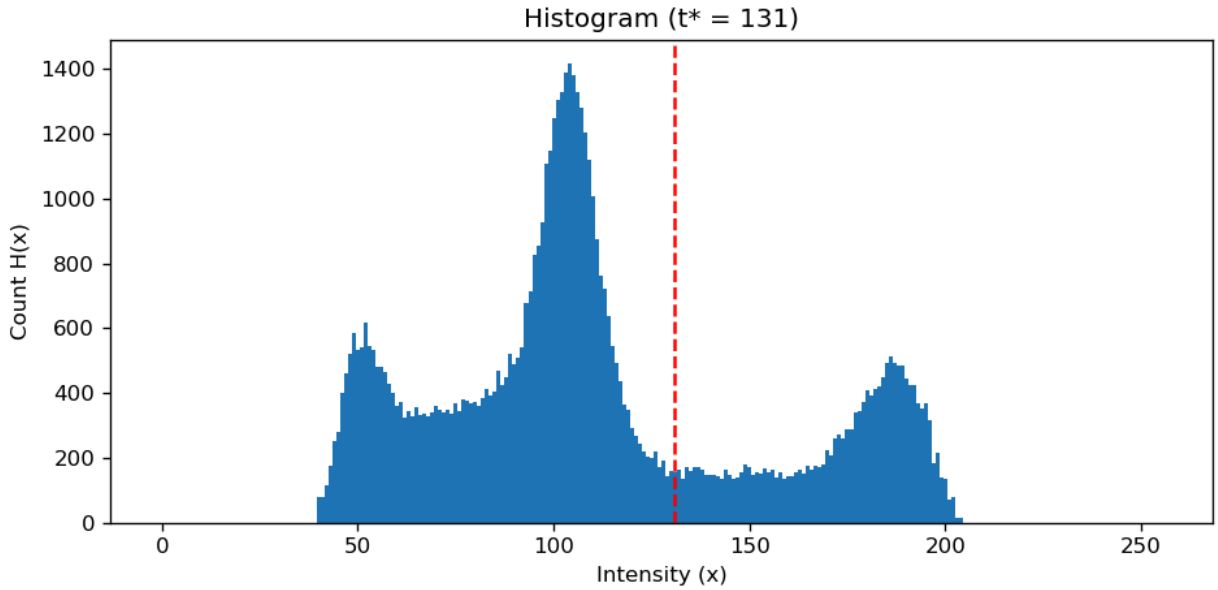

Figure 2: Histogram of Image 1

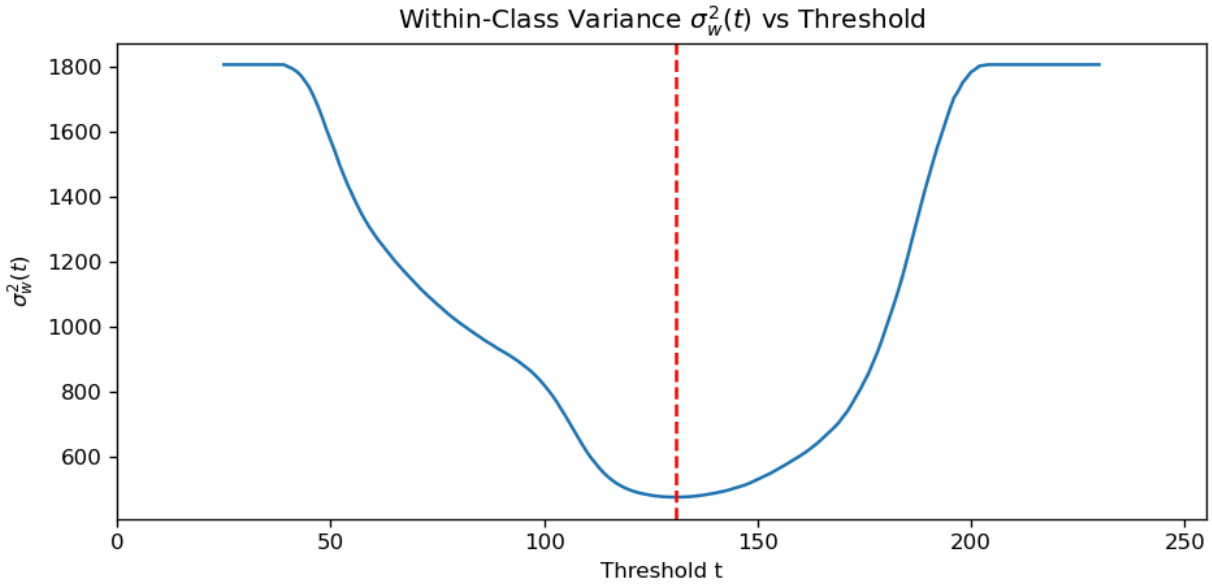

Figure 3: Variance Plot of Image 1

Image 2:

Figure 4: Before and After Binary Conversion of Image 2

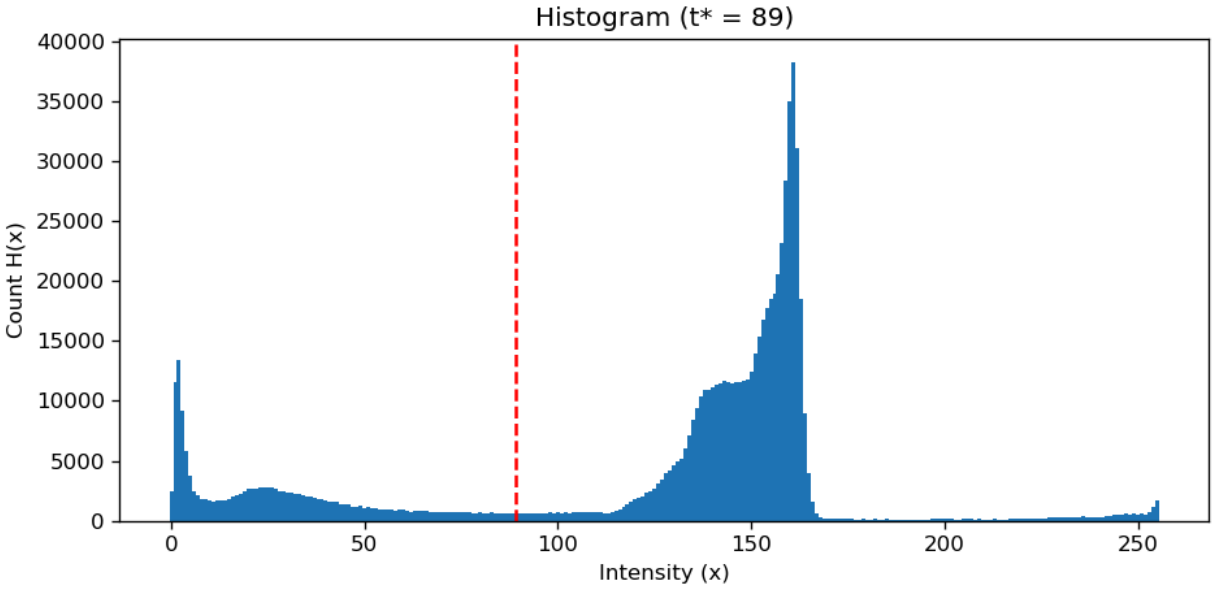

Figure 5: Histogram of of Image 2

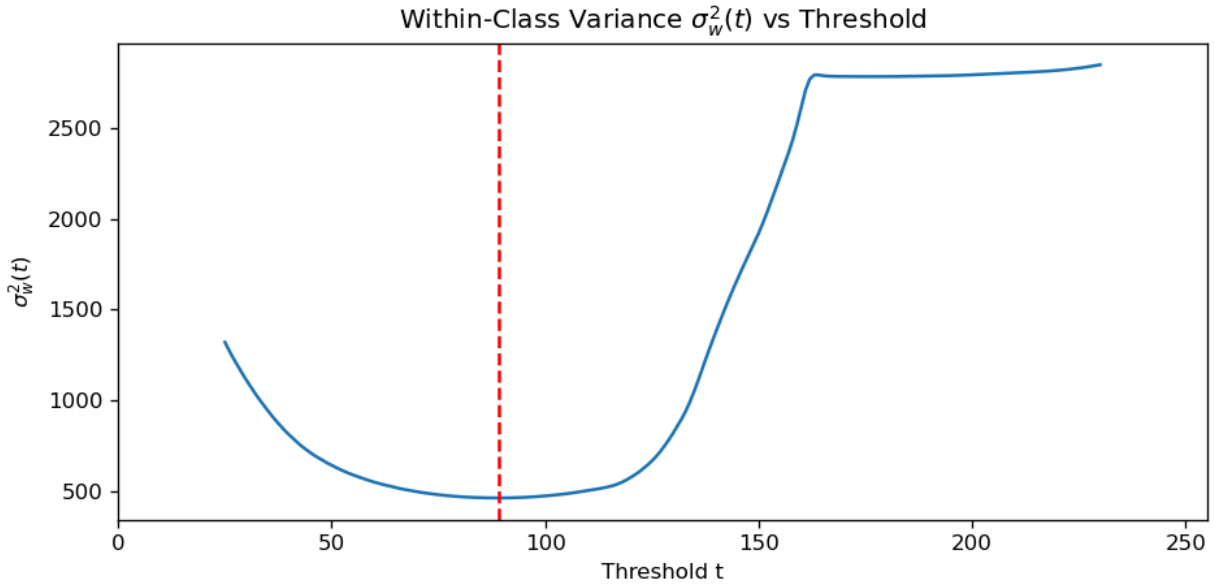

Figure 6: Variance Plot of Image 2

Image 3:

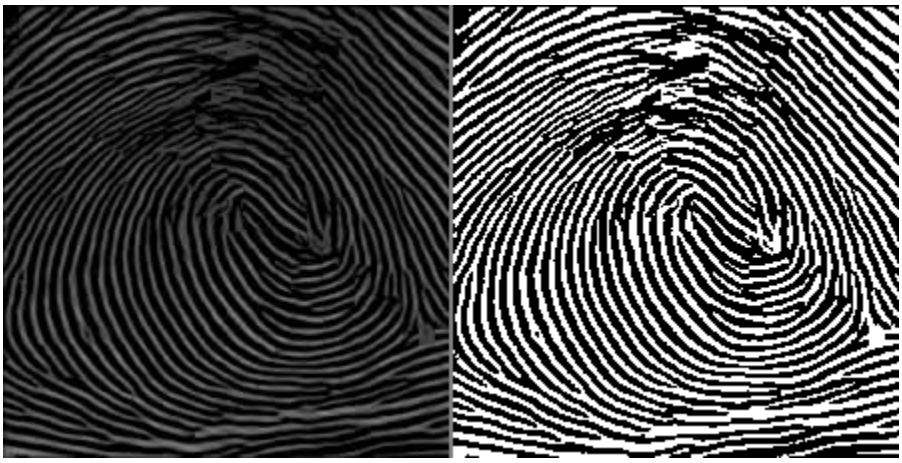

Figure 7: Before and After Binary conversion of Image 3

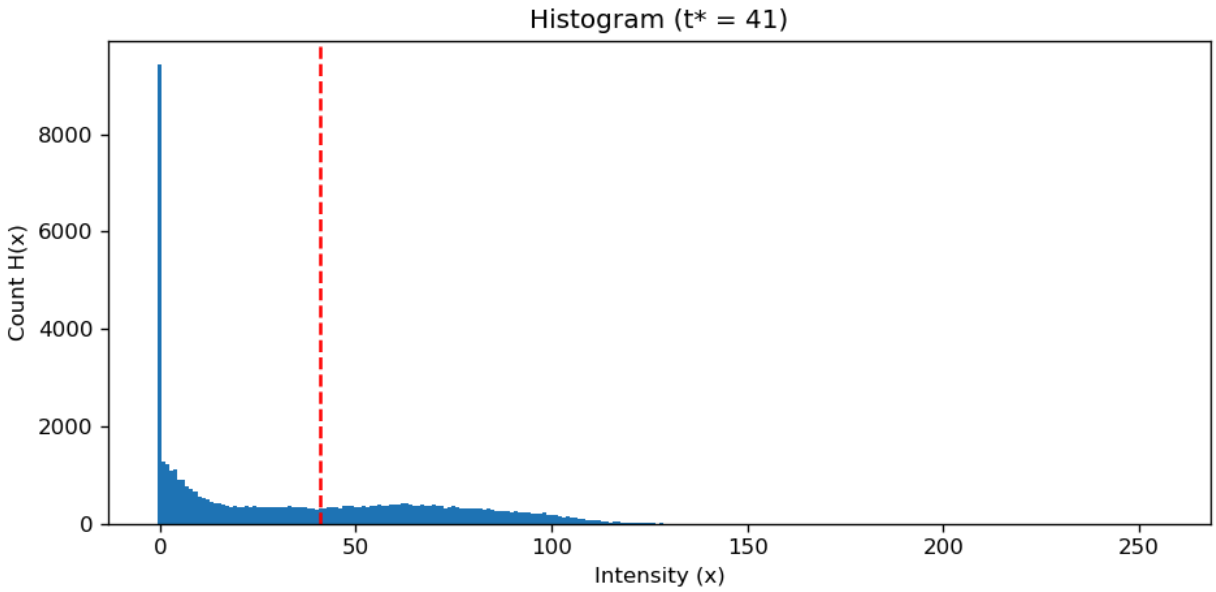

Figure 9: Histogram of Image 3

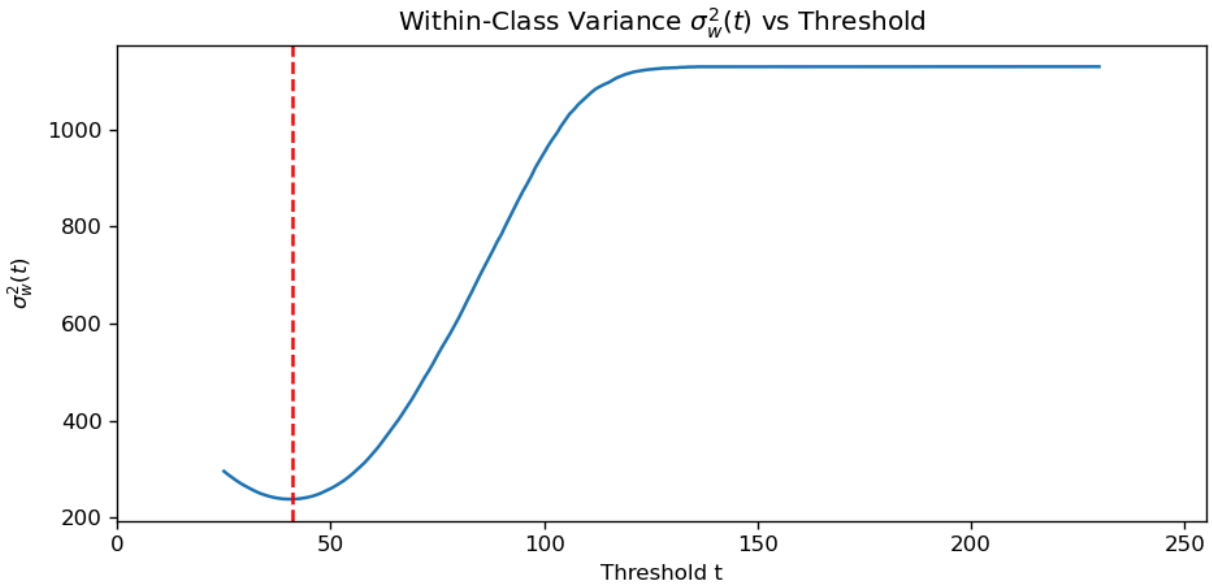

Figure 10: Variance Plot of Image 3

Threshold Values From Code Output:

```
(base) aliruiz@Juans-MacBook-Pro image_thersholding_project % python3 thresholdi
ng.py Project2_22.png Project2_51.jpg Project2_52.jpg --outdir results

- Project2_22.png
  Optimal threshold t* = 131
- Project2_51.jpg
  Optimal threshold t* = 89
- Project2_52.jpg
  Optimal threshold t* = 41
(base) aliruiz@Juans-MacBook-Pro image_thersholding_project %
```

Figure 11: Output Threshold Values as Seen on Variance Plots of Images

Conclusion

This project demonstrated how automatic thresholding can be achieved by minimizing the within-class variance of pixel intensities. By applying the algorithm to the three images given, optimal thresholds were determined in the restricted range of 25–23. This restriction was used to successfully separate background from foreground. The results validate the effectiveness of the algorithmic approach to handling binary conversions.