Image Based Classifier for Detecting Poetic Content
Can we find poetic content in historic newspapers based on visual signals alone?
Motivation

• Advance work on the use of digital images

• Making data more readily available for study
Why poetry?

• Scale

• Visual distinctness

• Interest and significance
Visual features of a poem

- Whitespace between stanzas
- Content blocks with jagged right-side edges/varying line lengths
- Left margin whitespace
Teaching a computer to see poetry

- Pre-processing

- Features extraction

- Using artificial neural network
Pre-Processing Stage

- Blurring
- Bi-Gaussian binarization
- Pixel consolidation
RGB

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>G</td>
<td>G</td>
<td>G</td>
<td>B</td>
<td>B</td>
</tr>
</tbody>
</table>
public void convertToBinaryImage(){
    for (int i = 0; i < this.vertical; i++){
        for(int j= 0; j < this.horizontal; j++){
            if(this.blurredImagePixels[i][j] < this.threshold){
                binaryImagePixels[i][j] = 0;
            } else {
                binaryImagePixels[i][j] = 255;
            }
        }
    }
}
Finding the threshold

**Global Thresholding** = Choose threshold \( T \) that separates object from background.

Image Histogram

```java
public static long[] computeHist(int[][] img, int maxIntensity) {
    long[] hist = new long[maxIntensity];
    assert img.length > 0;
    int height = img.length;
    int width = img[0].length;
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            int value = img[i][j];
            if ((value >= 0) && (value < maxIntensity)) {
                hist[value]++;
            }
        }
    }
    return hist;
}
```
Otsu’s method - Demonstration

- Assumes a bimodal distribution of gray-level values
- Given 6x6 image
Otsu’s method - Demonstration

Weight $W_b = \frac{8 + 7 + 2}{36} = 0.4722$

Mean $\mu_b = \frac{(0 \times 8) + (1 \times 7) + (2 \times 2)}{17} = 0.6471$

Variance $\sigma_b^2 = \frac{((0 - 0.6471)^2 \times 8) + ((1 - 0.6471)^2 \times 7) + ((2 - 0.6471)^2 \times 2)}{17} = \frac{(0.4187 \times 8) + (0.1246 \times 7) + (1.8304 \times 2)}{17}$

$$= 0.4637$$

Threshold = 3

Weight $W_f = \frac{6 + 9 + 4}{36} = 0.5278$

Mean $\mu_f = \frac{(3 \times 6) + (4 \times 9) + (5 \times 4)}{19} = 3.8947$

Variance $\sigma_f^2 = \frac{((3 - 3.8947)^2 \times 6) + ((4 - 3.8947)^2 \times 9) + ((5 - 3.8947)^2 \times 4)}{19} = \frac{(4.8033 \times 6) + (0.0997 \times 9) + (4.8864 \times 4)}{19}$

$$= 0.5152$$
The next step is to calculate the 'Within-Class Variance'. This is simply the sum of the two variances multiplied by their associated weights.

\[
\text{Within Class Variance} \quad \sigma_W^2 = W_b \sigma_b^2 + W_f \sigma_f^2 = 0.4722 \times 0.4637 + 0.5278 \times 0.5152 = 0.4909
\]

Otsu's thresholding method involves iterating through all the possible threshold values.

The aim is to find the threshold value where the sum of foreground and background spreads is at its minimum.
Otsu’s method

By a bit of manipulation, we can calculate what is called the *between class* variance, which is far quicker to calculate.

Luckily, the threshold with the maximum *between class* variance also has the minimum *within class* variance.

Within Class Variance \( \sigma_W^2 = W_b \sigma_b^2 + W_f \sigma_f^2 \) (as seen above)

Between Class Variance \( \sigma_B^2 = \sigma^2 - \sigma_W^2 \)

\[ = W_b(\mu_b - \mu)^2 + W_f(\mu_f - \mu)^2 \quad (\text{where } \mu = W_b \mu_b + W_f \mu_f) \]

\[ = W_b W_f (\mu_b - \mu_f)^2 \]

```
// Total number of pixels
int total = srcData.length;

float sum = 0;
for (int t=0 ; t<256 ; t++) sum += t * histData[t];

float sumB = 0;
int wB = 0;
int wF = 0;

float varMax = 0;
threshold = 0;

for (int t=0 ; t<256 ; t++) {
    wB += histData[t]; // Weight Background
    if (wB == 0) continue;
    wF = total - wB; // Weight Foreground
    if (wF == 0) break;

    sumB += (float) (t * histData[t]);

    float mB = sumB / wB;
    float mF = (sum - sumB) / wF; // Mean Foreground

    // Calculate Between Class Variance
    float varBetween = (float)wB * (float)wF * (mB - mF) * (mB - mF);

    // Check if new maximum found
    if (varBetween > varMax) {
        varMax = varBetween;
        threshold = t;
    }
```
- Remove stray black spots are cleared.
- For each pixel in a row counts the total object pixels (black) in that row.
- If the total number of object pixels in a row is greater than a given threshold, all of the pixels from the start index to the end index in the row are assigned to object pixels (Black).
Features extraction

• Computation of:
  • Column widths
  • Row depths

• Calculating statistics (mean, std, min, max, range) of:
  • Margin on the left
  • Jaggedness
  • Stanzas
  • Row lengths
Column widths computation

The algorithm counts both:

- length of background (white) pixels prior to the first object (black) pixel
- length of background pixels after the final object pixel in a row

```java
public void computeColumnWidths() {
    boolean stillBackground;
    int numPixSoFarFromLeft;
    int numPixSoFarFromRight;

    for (int i = 0; i < DEPTH; i++) {
        // going from left to right
        numPixSoFarFromLeft = 0;
        stillBackground = true;
        int j = WOFFSET;
        while (stillBackground && j < WIDTH - WOFFSET) {
            if (image.getBinaryImagePixels()[i][j] == OBJECT)
                stillBackground = false;
            else
                numPixSoFarFromLeft++;
            j++;
        }

        // going from right to left
        numPixSoFarFromRight = 0;
        stillBackground = true;
        j = WIDTH - WOFFSET;
        while (stillBackground && j >= WOFFSET) {
            if (image.getBinaryImagePixels()[i][j] == OBJECT)
                stillBackground = false;
            else
                numPixSoFarFromRight++;
            j--;
        }

        leftColumnWidths[i] = numPixSoFarFromLeft;
        rightColumnWidths[i] = numPixSoFarFromRight;
    }
} // end computeColumnWidths
```

* WIDTH = image.getHorizontal(); DEPTH = image.getVertical();
WOFFSET = (int) (WIDTH*0.1); DOFFSET = (int) (DEPTH*0.1);
Row depths computation

The algorithm counts the continuous background (white) pixels in each column and stores the values in a 2D integer matrix.

```java
public void computeRowDepths() {
    int numPixSoFar;
    int k; // to hold the index to store each gap's depth
    for (int j = WOFFSET; j < WIDTH - WOFFSET; j++) {
        k = 0; // first index

        // going from top to bottom
        numPixSoFar = 0;
        int i = DOFFSET;
        while (i < DEPTH - DOFFSET) {
            if (image.getBinaryImagePixels()[i][j] == OBJECT){
                // we have found the first break of background pixels gap
                if (numPixSoFar > 0) { // only update if we have accumulated
                    rowDepths[k][j] = numPixSoFar;
                    numPixSoFar = 0; // reset for the next gap
                    k++;
                    // update index for the next gap
                }
                else
                    numPixSoFar++;
                i++;
            }
        }
    }
} // end for j loop
} // end computeRowDepths
```

* WIDTH = image.getHorizontal(); DEPTH = image.getVertical();
  WOFFSET = (int) (WIDTH*0.1); DOFFSET = (int) (DEPTH*0.1);
Calculating statistics  
(mean, std, min, max, range)

• Margin on the left - using the column widths on the left of each image

• Stanzas - looking for whitespace between stanzas, using row depths

• Jaggedness - measures of the background pixels after the final object pixel (using the column widths on the right of each image)

• Compute length of columns
ANN – Artificial Neural Network

- Inspired by the human brain

- The basic computational unit of the brain is a **neuron**.

- The node/neuron receives input from some other nodes and computes an output.

- Each input has an associated weight \( w \).

- The node applies a function to the weighted sum of its inputs.

- The idea is that the synaptic strengths (the weights \( w \)) are learnable and control the strength of influence.

- If the final sum is above a certain threshold, the neuron can **fire**, sending a spike along its axon.
ANN – Multi-layer Perceptron

- Consists of multiple layers of computational units
- Each neuron in one layer has directed connections to the neurons of the subsequent layer.
- Usually using sigmoid function as an activation function.
- MLP are able to learn non-linear representations
**ANN – More details**

**Activation function** - has to be a non-linear function, otherwise the neural network will only be able to learn linear models.

**Error function** - The goal is to learn the weights of the network automatically from data such that the predicted output $y_{\text{output}}$ is close to the target $y_{\text{target}}$ for all inputs $x_{\text{input}}$. To measure how far we are from the goal, we use an error function.

A commonly used error function is $E(y_{\text{output}}, y_{\text{target}}) = \frac{1}{2} (y_{\text{output}} - y_{\text{target}})^2$.

**Backpropagation** - Backpropagation minimizing the loss function, where the loss function determines how wrong the result is from what it’s suppose to be.
ANN – In our case

- Attributes are translated into individual nodes that communicate with a hidden layer.
- These connections are initially weighted
- After a predetermined number of iterations the weights are increased or decreased depending on the prediction each node makes
- Over the iterations, the ANN is optimized such that the attributes that contribute most to determining the instance have the most weight, and through back propagation, the ANN reduces the weight of the less deterministic attributes.
## Results

<table>
<thead>
<tr>
<th>Class</th>
<th>Training</th>
<th>Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>true &amp; predicted true</td>
<td>79.44%</td>
<td>61.84%</td>
</tr>
<tr>
<td>true &amp; predicted false</td>
<td>20.56%</td>
<td>36.18%</td>
</tr>
<tr>
<td>false &amp; predicted true</td>
<td>8.26%</td>
<td>20.70%</td>
</tr>
<tr>
<td>false &amp; predicted false</td>
<td>91.75%</td>
<td>79.30%</td>
</tr>
</tbody>
</table>

**Precision**

**Recall**

Recall = 

Precision =
False positive – sample 1

7: (a) the original image snippet, (b) the binary image, and (c) the consolidated binary in Our classifier mistakenly identified it as a poem image.
False positive – sample 2

\[ \text{\textbf{MARRIED}} \]
On Monday afternoon, March 21st, by the rector of the church, Miss Lucey and Miss Lucey, aged 18 years and 8 days.

\[ \text{\textbf{DIED}} \]
On Saturday morning, 16th inst., in White church, after a short illness, Miss Lucey and Miss Lucey, aged 18 years and 8 days.

The sudden demise of Miss Lucey has caused much sorrow to her friends, and much to Miss Lucey’s relatives and friends. She was evidently in her prime, but a few days, and during that time would spend all her time reading, writing, and until within a few hours of her death. Although it was expected that she was soon to die, she would not be, her death was a surprise to all. But she died not as one having no hope for her early resurrection, but on a bed of death with sighs of lamentation, and to parents and many relatives and friends.

---

\[ \text{\textbf{Our classifier mistakenly identified it as a poem image.}} \]
False negative – sample 3
False negative – sample 4

Figure 10: The original image snippet with significant bleed-through. Our algorithm failed to identify a viable threshold to classify the image into object and background pixels. Our classifier mistakenly identified the snippet as a non-poem image.
Improvements

• Improve extraction algorithms (for example binarization)

• Page segmentation

• More visual features

• Enlarge scaling
Simple thresholding is not always possible:

- Many objects at different gray levels.
- Variations in background gray level.
- Noise in image.
Local Thresholding - 4 Thresholds

Divide image into regions.

Perform thresholding independently in each region.
Adaptive Thresholding

Every pixel in image is thresholded according to the histogram of the pixel neighborhood.
Image segmentation

Image segmentation is defined as a process of partitioning a digital image into multiple smaller segments called regions.
Image segmentation

Algorithm: Page Segmentation

Input: an original image, $I_{\text{original}}$, of a newspaper page

Output: a set of image snippets, $\langle i_{\text{original}} \rangle$

1. Compute average intensity of $I_{\text{original}}$

2. If $\text{AveIntensity}(I_{\text{original}})$ is too bright then
   a. Perform contrast enhancement on $I_{\text{original}}$ to obtain $I_{\text{enhanced}}$

3. Perform binarization on $I_{\text{enhanced}}$ to obtain $I_{\text{binary}}$

4. Perform morphological cleaning on $I_{\text{binary}}$ to obtain $I_{\text{binary_cleaned}}$ to clean up image noise

5. $\text{ColumnBreaks} \leftarrow \text{FindColumnBreaks}(I_{\text{binary_cleaned}})$

6. $\langle i_{\text{original}} \rangle \leftarrow \text{GenerateSnippets}(\text{ColumnBreaks}, I_{\text{original}})$
Conclusion & Final thoughts