CA3

Alireza Karimi : 810101492

Mehdi Vajhi : 810101558

Part 2

Template Matching Strategy for IC Recognition

In this report, we describe a strategy to detect the location of an IC component within a larger PCB image using template matching based on normalized cross-correlation. This approach involves comparing the template (IC image) with regions of the PCB image to find the best matches. Since the IC component might appear rotated by 180 degrees, we account for both original and rotated orientations of the template image.

The process can be broken down into the following main steps:

1. **Preprocessing**: Convert both the template (IC) and PCB images to grayscale for simpler computation.
2. **Normalized Cross-Correlation**: Perform template matching by calculating the normalized cross-correlation coefficient between the IC image and the PCB image for both original and 180-degree rotated templates.
3. **Thresholding and Local Maxima Detection**: Identify regions with a high correlation coefficient, exceeding a specified threshold, which indicates potential matches.
4. **Overlay Rectangles on Matches**: Mark the detected areas in the PCB image by drawing rectangles over the locations with high correlation.

Formula for Normalized Cross-Correlation

The normalized cross-correlation coefficient for a template image \(X \) and a target image \(Y \) at a given location \((x, y)\) is calculated as:

$$\text{Correlation Coeff}(x, y) = \frac{\sum_{n=1}^{N} X[n]Y[n]}{\sqrt{\sum_{n=1}^{N} X[n]^2} \times \sqrt{\sum_{k=1}^{N} Y[k]^2}}$$

where:

- X[n] and Y[n] are pixel values from the template and target image within the current region.
- The sum runs over all pixel values in the template.
- The result is a value between -1 and 1, where 1 indicates a perfect match, -1 indicates a perfect inverse match, and values close to 0 indicate poor matches.

Code Explanation

Here's an explanation of each main part of the MATLAB code used for this process:

1. Image Preprocessing

```
if size(templateImage, 3) == 3
    templateImage = rgb2gray(templateImage);
end
if size(mainImage, 3) == 3
    mainImage = rgb2gray(mainImage);
end
```

Explanation: This section checks if the images are in color (RGB) and, if so, converts them to grayscale using `rgb2gray`. Grayscale images simplify the computation, as they reduce the number of channels to process.

2. Compute Normalized Cross-Correlation

```
correlationMatrix =
normxcorr2(templateImage, mainImage);

templateRotated = imrotate(templateImage, 180);

correlationMatrixRotated =
normxcorr2(templateRotated, mainImage);
```

Explanation: `normxcorr2` is used to compute the normalized cross-correlation between the template and the target image. This function slides the template over the target image and computes the correlation coefficient at each position. We perform this twice: first with the original template (`correlationMatrix`), then with the template rotated by 180 degrees (`correlationMatrixRotated`), to account for possible rotation in the target.

3. Thresholding and Finding Local Maxima

```
significantLocalMaxima1 =
imregionalmax(correlationMatrix) &
(correlationMatrix > threshold);

significantLocalMaxima2 =
imregionalmax(correlationMatrixRotated) &
(correlationMatrixRotated > threshold);
```

Explanation: We use `imregionalmax` to find local maxima in the correlation matrices. Local maxima represent potential matches of the template in the target image. We also apply a threshold to filter out matches with low correlation values. Only maxima with correlation values higher than the specified threshold are considered significant matches.

4. Overlaying Rectangles on Matches

1. **Threshold and Find Local Maxima for Original Template**:

```
significantLocalMaxima1 = imregionalmax(correlationMatrix)
& (correlationMatrix > threshold);
[rows1, cols1] = find(significantLocalMaxima1);
```

- `imregionalmax(correlationMatrix)`: This function finds local maxima in the `correlationMatrix`. Local maxima are points where the correlation value is higher than all neighboring points, which suggests potential matches of the template image within the target image.
- `(correlationMatrix > threshold)`: We apply a threshold to the correlation values to keep only significant matches. This means we ignore low-correlation values that don't meet the threshold.
- `significantLocalMaxima1`: This is a logical matrix where only the positions of significant local maxima (above the threshold) are `true`.
- `[rows1, cols1] = find(significantLocalMaxima1)`: The `find` function extracts the row and column indices of these significant local maxima, which represent potential match locations.

2. **Adjust Coordinates for Padding**:

```
[rows1_adj, cols1_adj] = deal(rows1 - size(templateImage,
1) + 1, cols1 - size(templateImage, 2) + 1);
```

- Since `normxcorr2` adds padding to the correlation matrix (due to template sliding), the indices need adjustment to align with the original image coordinates.
- `size(templateImage, 1)` and `size(templateImage, 2)` are the dimensions of the template image (height and width, respectively).
- Subtracting these dimensions (and adding 1) adjusts the coordinates to match the original position of the template in the target image.

3. **Draw Blue Rectangles for Original Template Matches**:

```
for i = 1:length(rows1_adj)
    pos = [cols1_adj(i), rows1_adj(i), size(templateImage,
2), size(templateImage, 1)];
    rectangle('Position', pos, 'EdgeColor', 'b', 'LineWidt
h', 2);
end
```

- This loop iterates over each detected significant match position.
- `pos` defines the rectangle's position and size:
  - `cols1_adj(i), rows1_adj(i)`: The adjusted x and y coordinates of the match.
  - `size(templateImage, 2), size(templateImage, 1)`: The width and height of the rectangle, matching the template size.
- `rectangle('Position', pos, 'EdgeColor', 'b', 'LineWidth', 2)`: Draws a blue (`'b'`) rectangle around each match in the original correlation matrix, marking high-correlation regions for the original template orientation.

4. **Threshold and Find Local Maxima for Rotated Template**:

```
significantLocalMaxima2 = imregionalmax(correlationMatrixR
otated) & (correlationMatrixRotated > threshold);
[rows2, cols2] = find(significantLocalMaxima2);
```

- This part is similar to steps 1 and 2 but applied to the rotated template. We find significant local maxima in the `correlationMatrixRotated` and retrieve the coordinates of potential matches for the 180-degree rotated template.

5. **Adjust Coordinates for Rotated Template Padding**:

```
[rows2_adj, cols2_adj] = deal(rows2 - size(templateImage,
1) + 1, cols2 - size(templateImage, 2) + 1);
```

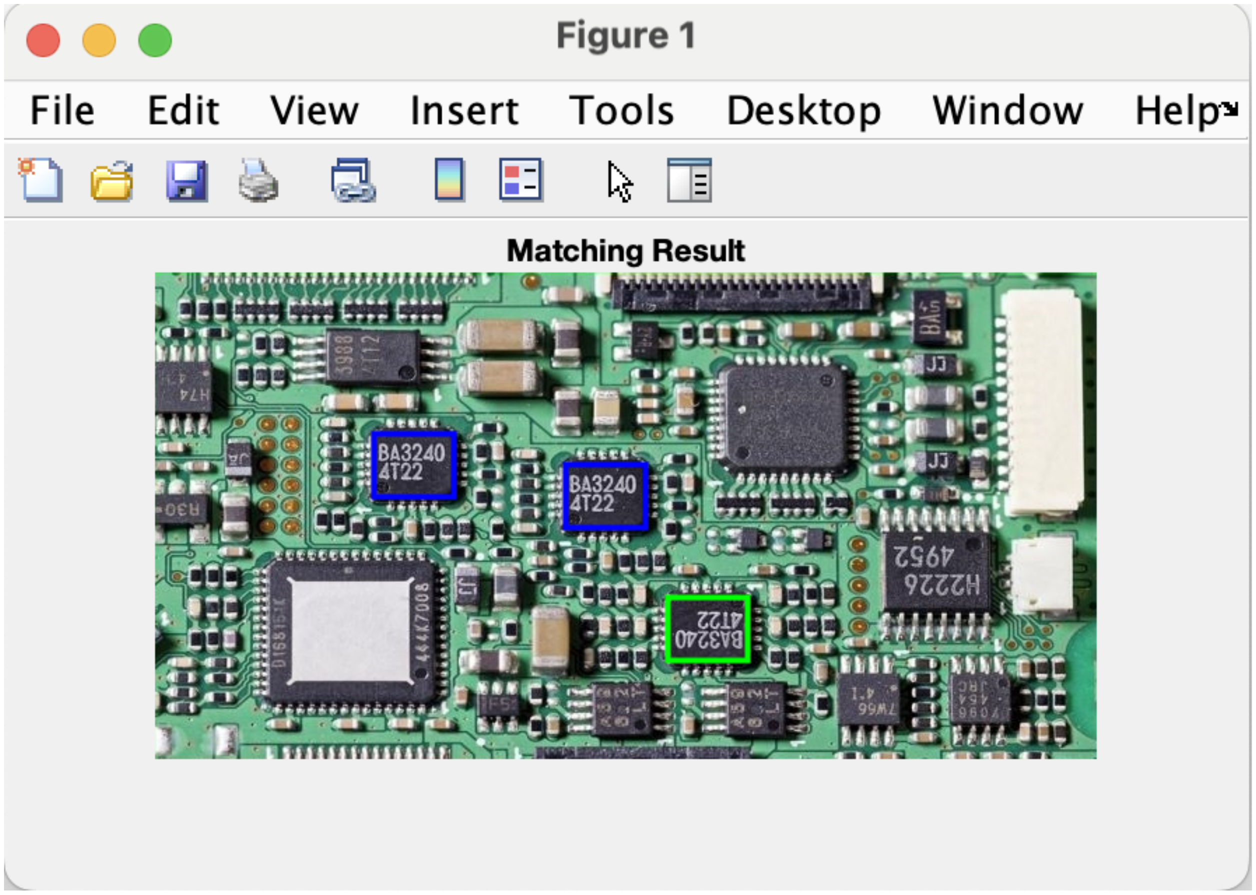
- Just like with the original template, we adjust the coordinates for the rotated template to account for padding.

6. **Draw Green Rectangles for Rotated Template Matches**:

```
for j = 1:length(rows2_adj)
    pos = [cols2_adj(j), rows2_adj(j), size(templateImage,
2), size(templateImage, 1)];
    rectangle('Position', pos, 'EdgeColor', 'g', 'LineWidt
h', 2);
end
```

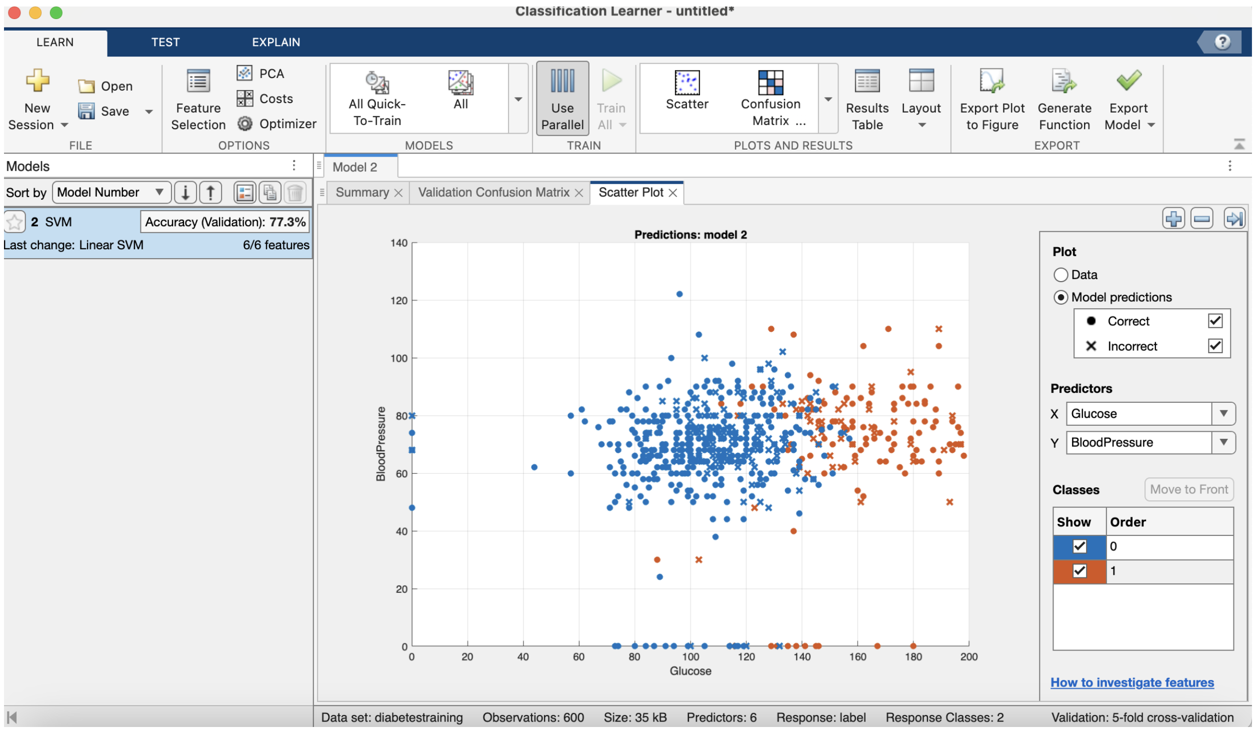
- This loop draws green (`'g'`) rectangles over the locations in `mainImage` where the rotated template has a high correlation.
- Each rectangle highlights the region where the rotated IC component likely appears in the PCB image.

The result:



Part 3

Accuracy with all features:



2 SVM — Accuracy (Validation): 77.3%

Last change: Linear SVM — 6/6 features

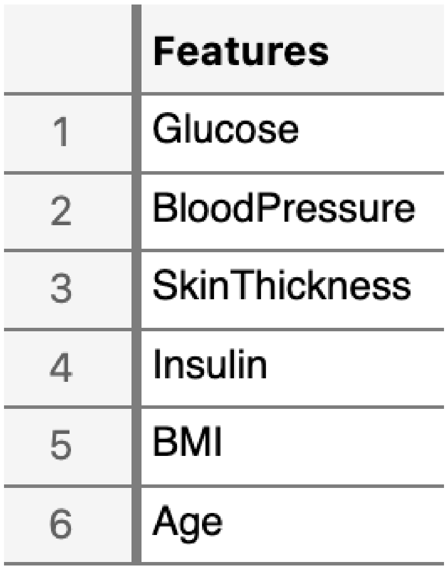
Plot: Model predictions (Correct ✓, Incorrect ✓)

Predictors: X Glucose, Y BloodPressure

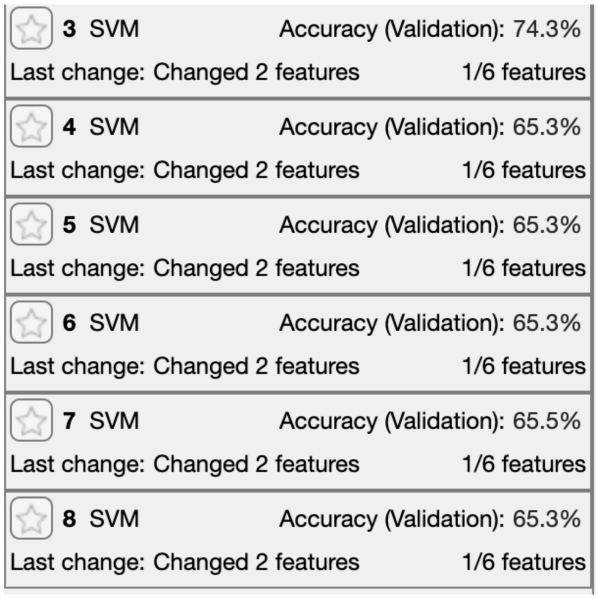
Show	Order
✓	0
✓	1

Data set: diabetestraining Observations: 600 Size: 35 kB Predictors: 6 Response: label Response Classes: 2 Validation: 5-fold cross-validation

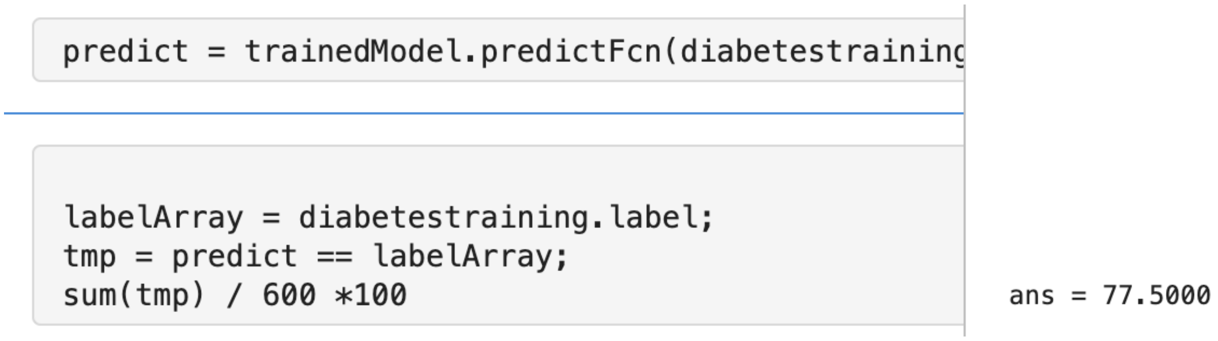
accuracy per feature: (same order)



	Features
1	Glucose
2	BloodPressure
3	SkinThickness
4	Insulin
5	BMI
6	Age



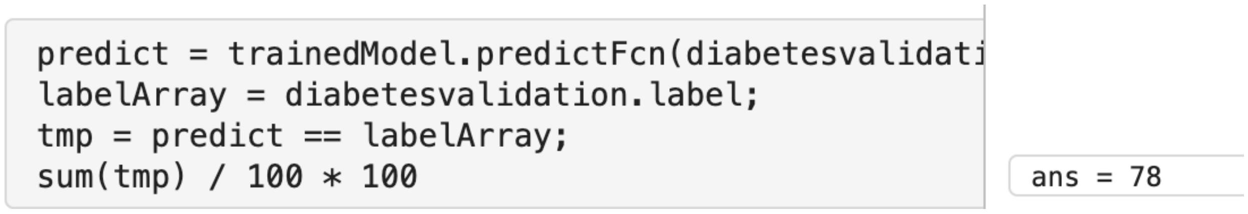
```
3 SVM        Accuracy (Validation): 74.3%
Last change: Changed 2 features    1/6 features
4 SVM        Accuracy (Validation): 65.3%
Last change: Changed 2 features    1/6 features
5 SVM        Accuracy (Validation): 65.3%
Last change: Changed 2 features    1/6 features
6 SVM        Accuracy (Validation): 65.3%
Last change: Changed 2 features    1/6 features
7 SVM        Accuracy (Validation): 65.5%
Last change: Changed 2 features    1/6 features
8 SVM        Accuracy (Validation): 65.3%
Last change: Changed 2 features    1/6 features
```



```
predict = trainedModel.predictFcn(diabetestraining
```

```
labelArray = diabetestraining.label;
tmp = predict == labelArray;
sum(tmp) / 600 *100
```

```
ans = 77.5000
```



```
predict = trainedModel.predictFcn(diabetesvalidati
labelArray = diabetesvalidation.label;
tmp = predict == labelArray;
sum(tmp) / 100 * 100
```

```
ans = 78
```

Explanation: The obtained accuracy is 0.2 percent higher than previously achieved. This slight discrepancy arises because the program uses the k-fold method for calculating accuracy, where only 1/5 of the data is left out for validation during each fold. The model's accuracy is predicted on this held-out data, and the average is taken as the final accuracy. However, in our calculation, we used the entire training dataset to assess accuracy, which tends to yield a slightly higher value. This explains the difference.