Assignment 02

Kamil Ilyas —— 20i-2371

October 2023

1 Question 1: Experimental Results - Determination of Clock Time

1.1 Introduction

This technical report discusses the proposed solution for determining the current time displayed on an analog clock using image processing techniques. We explore the code provided and evaluate its performance, while also discussing its relevance to advanced Digital Image Processing (DIP) topics.

1.2 Code Description

The code defines a Python function called `detect_clock_time(image_path)` to process an input image of an analog clock and calculate the current time displayed on it. The code follows these main steps:

1. Load the input image.
2. Convert the image to grayscale for edge detection.
3. Apply Gaussian blur and Canny edge detection.
4. Identify and sort contours in the image, selecting the two largest contours, which represent the clock's hands.
5. Fit ellipses to the selected contours to determine the angles of the clock hands.
6. Calculate the angles of the hour and minute hands.
7. Translate the angles into hour and minute values to determine the current time.
8. Display the image with detected clock hands, including a visual representation of the hand angles.
9. Print the calculated time on the console.

1.3 Experimental Results

1.3.1 Data Preprocessing

In this experiment, we used an example clock image (`'Images//2-2.jpg'`) as input. The image is loaded and processed as described in the code.

1.3.2 Model Architecture

The code does not employ a traditional machine learning model; instead, it applies image processing techniques to extract clock hand angles and calculate the time. Therefore, there is no specific model architecture involved.

1.3.3 Training Process and Evaluation Metrics

The code does not involve training or evaluation metrics, as it focuses on image analysis rather than machine learning.

1.3.4 Results and Accuracy

The code successfully identifies and calculates the angles of the clock's hour and minute hands, then converts these angles into a current time reading. The calculated time is printed to the console and displayed visually on the image. The accuracy of the time calculation depends on the accuracy of the angle detection and may vary based on image quality and clock hand visibility.

1.3.5 Model Convergence and Overfitting

The code does not exhibit model convergence or overfitting characteristics because it does not involve training data or a machine learning model.

1.3.6 Addressing Class Imbalance

Class imbalance is not applicable to this code, as it primarily pertains to classification tasks where certain classes are underrepresented in the data, which is not the case here.

1.3.7 Hyperparameter Tuning

Hyperparameter tuning is not relevant to this code, as it does not involve machine learning models with hyperparameters that need adjustment.

1.3.8 Conclusion

The provided code effectively processes analog clock images, identifies clock hands, and calculates the current time. It does not involve machine learning models but relies on traditional image processing techniques. The results may vary based on image quality, hand visibility, and clock design.

This code serves as an interesting and relevant example of how image processing can be applied to solve real-world problems, such as extracting information from images, which is a common topic in advanced Digital Image Processing (DIP).

2 Question 2: Experimental Results - Character Search Match

2.1 Introduction

This technical report discusses the proposed solution for character recognition and matching within an image using the provided code. We explore the code and evaluate its performance, while also discussing its relevance to advanced Digital Image Processing (DIP) topics.

2.2 Code Description

The code is designed to identify a character within a specified input region and draw bounding boxes around matching characters in the lower section of the image. It uses a dictionary of character templates and employs template matching (`cv2.matchTemplate`) to find the most similar character to the one in the input image.

2.3 Experimental Results

2.3.1 Data Preprocessing

In this experiment, we used an example input image (`'Images\1-3.jpg'`) and character templates (`'Images\A.png'`, `'Images\8.png'`, `'Images\W.png'`). The input image and templates are loaded for processing.

2.3.2 Model Architecture

The code does not involve a traditional machine learning model. Instead, it utilizes template matching to recognize and match characters.

2.3.3 Training Process and Evaluation Metrics

The code does not include a training process or evaluation metrics, as it primarily focuses on image processing and template matching.

2.3.4 Results and Accuracy

The code successfully recognizes characters within the specified input region and draws bounding boxes around matching characters in the lower section of the image. The results depend on the quality of the character templates and the

similarity of the characters in the input image to the templates. The threshold for similarity is set at 0.95, meaning characters with a match score above this threshold are considered matching.

2.3.5 Model Convergence and Overfitting

The code does not exhibit model convergence or overfitting characteristics because it does not involve training or learning processes. Template matching relies on predefined templates.

2.3.6 Addressing Class Imbalance

The code does not address class imbalance, as it is not applicable to the task of character recognition and matching.

2.3.7 Hyperparameter Tuning

Hyperparameter tuning is not explicitly performed in this code. However, the code includes parameters such as the template matching threshold (0.95), which can be adjusted to control the level of similarity required for a character match.

2.3.8 Fine-Tuned Parameters

The following parameters are used in the code:

- Template Matching Threshold: 0.95

Fine-tuning these parameters can affect the character quality assessment results.

2.3.9 Conclusion

The provided code effectively identifies characters within an input region and highlights matching characters in the lower section of the image. It serves as a practical example of template matching in image processing and is relevant to DIP topics related to object recognition and matching within images. The code can be extended and customized for a wide range of character recognition tasks.

3 Question 3: Experimental Results - Test Scores Determination

3.1 Introduction

This technical report discusses the proposed solution for assessing the quality of handwritten characters within an image using the provided code. We explore the code and evaluate its performance, while also discussing its relevance to advanced Digital Image Processing (DIP) topics.

3.2 Code Description

The code is designed to assess the quality of handwritten characters within an image. It does so by computing a score based on the percentage of filled pixels in a region of interest (ROI). The code follows these main steps:

1. Load the input image and convert it to grayscale.
2. Apply Canny edge detection to the grayscale image.
3. Identify and sort contours in the image, selecting the three largest contours.
4. For each of these three contours:
    - Extract the ROI of the character.
    - Calculate a quality score for the character using the `calculate_score` function.
    - Annotate the image with the character's score.
5. Display the annotated image using `matplotlib`.

3.3 Experimental Results

3.3.1 Data Preprocessing

In this experiment, we used an example input image (`'Images\3-1.jpg'`) containing handwritten characters. The input image is loaded and processed as described in the code.

3.3.2 Model Architecture

The code does not involve a traditional machine learning model. Instead, it applies image processing techniques to assess character quality.

3.3.3 Training Process and Evaluation Metrics

The code does not involve training or evaluation metrics in the conventional sense. Instead, it calculates a quality score for each character based on the percentage of filled pixels in a region of interest.

3.3.4 Results and Accuracy

The code successfully identifies and assesses the quality of handwritten characters within the input image. The assessment is based on the percentage of filled pixels within a character's bounding box, which serves as a measure of the character's "filledness." The code annotates the image with the computed quality scores, providing a visual representation of character quality.

3.3.5 Model Convergence and Overfitting

The code does not exhibit model convergence or overfitting characteristics because it does not involve training data or a machine learning model.

3.3.6 Addressing Class Imbalance

Class imbalance is not applicable to this code, as it focuses on character quality assessment rather than classification tasks.

3.3.7 Hyperparameter Tuning

Hyperparameter tuning is not explicitly performed in this code. However, the code includes parameters such as the Canny edge detection thresholds and the score calculation threshold.

3.3.8 Fine-Tuned Parameters

The following parameters are used in the code:

- Canny Edge Detection Thresholds (threshold1 and threshold2)
- Score Calculation Threshold

Fine-tuning these parameters can affect the character quality assessment results.

3.3.9 Conclusion

The provided code effectively assesses the quality of handwritten characters in an image. It is a valuable tool for evaluating handwritten text in Optical Character Recognition (OCR) scenarios and other character quality assessment tasks. The code is relevant to advanced DIP topics related to image analysis and quality assessment.

4 Question 4: Experimental Results - Image Processing Software

4.1 Introduction

This technical report discusses the proposed solution for the Image Processing Software, a user-friendly interface that provides various image processing functionalities. We explore the code and evaluate its performance, while also discussing its relevance to advanced Digital Image Processing (DIP) topics.

4.2 Code Description

The provided code implements an image processing software with the following functionalities:

1. Open Image: Allows users to load an image using a file dialog and displays it in the software's canvas.
2. Save Image: Enables users to save the modified image using a file dialog.
3. Brightness Adjustment: Adjusts the brightness of the loaded image based on user input.
4. Logarithmic Transformation: Applies a logarithmic transformation to enhance image contrast.
5. Thresholding: Applies binary thresholding to the image based on the specified threshold value.
6. Grayscale Conversion: Converts the image to grayscale.
7. Filter Application: Applies various filters, including Gaussian, Median, and Bilateral, to the image.
8. Laplacian Sharpening: Enhances image edges using the Laplacian filter.
9. Unsharp Masking: Applies unsharp masking to enhance image details and edges.
10. Morphological Operations: Performs operations such as Erosion, Dilation, Opening, and Closing on the image.
11. Contour Detection: Detects and highlights contours in the image.
12. Color-Based Segmentation: Segments the image based on specified color ranges.
13. Line Detection: Detects and highlights lines in the image using the Hough Line Transform.
14. Connected Component Analysis: Identifies and annotates connected components in the image.

The code offers a user-friendly graphical interface with buttons and sliders to interact with these functionalities.

4.3 Experimental Results

4.3.1 Data Preprocessing

In this experiment, we used the image loading functionality to load various images and apply different processing operations.

4.3.2 Model Architecture

The code does not involve a traditional machine learning model. Instead, it applies a range of image processing techniques.

4.3.3 Training Process and Evaluation Metrics

The code does not involve training or evaluation metrics in the conventional sense, as it focuses on image processing.

4.3.4 Results and Functionality

The code successfully implements various image processing operations. Users can open, adjust, and save images, apply filters, enhance edges, perform morphological operations, and more. The software's graphical user interface (GUI) makes these operations accessible and user-friendly.

4.3.5 Model Convergence and Overfitting

The code does not exhibit model convergence or overfitting characteristics since it does not involve machine learning or training.

4.3.6 Addressing Class Imbalance

Class imbalance is not applicable to this code, as it primarily focuses on image processing and manipulation.

4.3.7 Hyperparameter Tuning

The code allows users to adjust parameters such as brightness, threshold values, and filter types. These parameter adjustments affect the image processing results and can be fine-tuned based on specific image processing requirements.

4.3.8 Conclusion

The provided Image Processing Software offers a user-friendly interface with a wide range of image processing functionalities. It is a valuable tool for enhancing, analyzing, and manipulating images. The code is relevant to advanced DIP topics related to image analysis, enhancement, and segmentation. It provides a platform for users to interact with various image processing techniques in a practical and intuitive manner.