Classical-Based Approach Implementation

Milestone 2

German International University
Department of Mechanical Engineering
Faculty of Engineering

Project Members/IDs:

Yahia Adam (7005813)
Youssef Desouki (7001816)
Nour El Fewy (7002342)
Kiroll Farid (7005201)

Deadline: April 27th, 2024

Introduction:

We will implement a basic Automatic License Plate Recognition (ALPR) system in this milestone. To recognize the text on the license plates, it is necessary to preprocess images of English automobile license plates and integrate an Optical Character Recognition (OCR) library.

• **Dataset Acquisition**: We used the dataset that was provided on the CMS, and did the preprocessing on the images that were given to us and the rest of the processes.

• **Pre-processing Techniques**: The first technique used on sample images was the Gaussian blur, Gaussian blur is a popular image processing technique used to reduce image noise and detail, creating a smoother appearance by applying a Gaussian function. Which applications can be image denoising, image smoothing, and lastly image pre-processing which we will be needing.

Secondly, The Laplacian operator is a mathematical operator used to compute the second spatial derivative of an image. In simpler terms, it highlights regions of rapid intensity change in an image, such as edges or boundaries between objects.

The sharpening filter using the Laplacian operator is a powerful tool for enhancing image detail and clarity, particularly in areas with high contrast or edges. However, care should be taken to avoid excessive sharpening, which can lead to artifacts or unnatural-looking results.

Gamma correction is a technique used in image processing to adjust the brightness and contrast of an image by applying a non-linear transformation to the pixel values. It's primarily used to compensate for the nonlinear relationship between the intensity of light and the brightness perceived by the human eye.

Gamma correction is an essential technique in image processing for accurately representing the perceived brightness and contrast of digital images, ensuring consistent reproduction across different display devices and mediums.

These techniques are important to enhance the performance of the OCR process.

• **Integration of OCR Library**: EasyOCR library was the chosen library to work on this milestone.
EasyOCR is a software library or tool that provides Optical Character Recognition (OCR) capabilities in an accessible and user-friendly manner.

1. Optical Character Recognition (OCR):
OCR is a technology that enables computers to recognize and extract text from images or scanned documents. It involves analyzing the visual patterns of characters in an image and converting them into machine-readable text.

2. EasyOCR:
EasyOCR is a specific implementation or software library that simplifies the process of performing OCR tasks. It is designed to be easy to use and integrate into various applications or projects without requiring extensive knowledge of OCR algorithms or techniques.

3. Features:
EasyOCR typically offers the following features:
- **Text Detection**: Identifying regions of text within an image.
- **Text Recognition**: Extracting text from the detected regions.
- **Language Support**: Recognizing text in multiple languages.
- **Accuracy**: Providing reliable and accurate OCR results.
- **Flexibility**: Supporting various image formats and resolutions.
- **Speed**: Processing images efficiently, often in real-time or near-real-time.

4. How it Works:
EasyOCR employs advanced machine learning algorithms, often based on deep learning models such as Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs). These models are trained on large datasets of annotated images to learn the visual patterns of characters and words in different languages and fonts.

When you provide an image containing text to EasyOCR, it first detects regions of text using techniques like object detection. Then, it processes each text region individually, recognizing and extracting the text content using the trained OCR model. Finally, it returns the recognized text as output, which can be further processed or used as needed.

5. Applications:
EasyOCR can be used in various applications across different industries and domains, including:
- Document Scanning and Digitization
- Text Extraction from Images for Information Retrieval

- Automatic Number Plate Recognition (ANPR)
- Receipt and Invoice Processing
- Automated Data Entry
- Augmented Reality (AR) and Virtual Reality (VR) Applications

• **Explanation of Methodology**:

- **Theoretical Background**

The methodology employed in this project involves the use of various image processing techniques and Optical Character Recognition (OCR) to extract text from noisy images. This section discusses the theoretical background behind the chosen methodology, including pre-processing techniques and the OCR library used.

1. Image Pre-processing Techniques:

Image pre-processing is essential for enhancing the quality of images before performing OCR. The following techniques are utilized:

- **Colorspace Conversion**: The images are initially converted to grayscale using the `cv2.COLOR_BGR2GRAY` conversion method. Grayscale images simplify subsequent processing by representing each pixel's intensity as a single value.

- **Gaussian Blur**: Gaussian blur is applied to the grayscale images using `cv2.GaussianBlur` to reduce noise and smooth out irregularities. This step improves the quality of the images and enhances the effectiveness of subsequent processing.

- **Sharpening Filter with Laplacian Operator**: A sharpening filter is applied to further enhance the clarity of the images. This involves detecting edges using the Laplacian operator (`cv2.Laplacian`) and then sharpening the image by subtracting a fraction of the Laplacian result from the original grayscale image.

- **Gamma Correction**: Gamma correction is applied to adjust the brightness and contrast of the images. This non-linear transformation compensates for the display device's or human eye's response to light intensity.

2. OCR Library: EasyOCR

EasyOCR is employed as the OCR library due to its simplicity, accuracy, and support for multiple languages. EasyOCR utilizes deep learning-based models to perform text detection and recognition on images. Key features of EasyOCR include:

- **Ease of Use**: EasyOCR simplifies the process of integrating OCR capabilities into applications with its straightforward API.

- **Language Support**: EasyOCR supports a wide range of languages, making it suitable for multilingual applications.

- **Accuracy**: The deep learning models employed by EasyOCR are trained on large datasets, resulting in high accuracy in text recognition tasks.

- **OCR Process**

The OCR process involves the following steps:

1. **Image Pre-processing**: The noisy images are pre-processed using the mentioned techniques to enhance their quality and improve OCR accuracy.

2. **Text Detection and Recognition**: EasyOCR's `readtext` function is used to detect and recognize text in the pre-processed images. The function returns bounding boxes around detected text regions along with the recognized text content.

3. **Character Filtering**: To improve OCR accuracy and restrict recognition to alphanumeric characters, a list of allowed characters (`allowed_characters`) is defined and provided to the OCR reader. This ensures that only relevant text is extracted from the images.

4. **Text Extraction and Output**: The OCR results, including the recognized text and corresponding bounding boxes, are extracted and processed as needed for further analysis or application-specific tasks.

- **Conclusion**

The methodology outlined above leverages image pre-processing techniques and the EasyOCR library to accurately extract text from noisy images. By applying a series of transformations to enhance image quality and employing a robust OCR engine, the proposed methodology

facilitates efficient and accurate text extraction for various applications, including document scanning, information retrieval, and data entry automation.

Code Explanation:

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import cv2

from zipfile import ZipFile
file_name = '/content/Dataset.zip'

with ZipFile(file_name, 'r') as zip:
 zip.extractall()
 print('Done')
```

In this code snippet, we had to extract the file data from the zip file to start the preprocessing techniques. We import all necessary libraries like numpy for array operation, matplotlib for plotting purposes, and cv2 for computer vision tasks. Tensorflow was imported for the sake of the second milestone where convolutional neural networks are to be applied. Zipfile library was also imported to allow and facilitate the extraction and handling of images in the zip file.

The code imports the ZipFile class from Python's zipfile module and defines the path to a ZIP file named "Dataset.zip". It then opens the ZIP file, extracts all its contents into the current directory, and prints "Done" to indicate successful extraction.

```
import os
import cv2
import numpy as np

# Define the folder path
folder_path = "/content/noisy_dataset"

# Get a list of all image files in the folder
image_files = [f for f in os.listdir(folder_path) if f.endswith(".png")]
# Create an empty list to store the images
```

```
images = []

# Iterate over the image files and read them into the list
for image_file in image_files:
  image = cv2.imread(os.path.join(folder_path, image_file))
  images.append(image)
```

This Python code part uses the os module to navigate the filesystem and the cv2 library to read images. It defines a folder license plate path where it expects to find PNG images. The path is /content/noisy_dataset. This is where the images are stored in the zip file. Then, it retrieves the list of the license plate PNG images in the specified folder. Then, it iterates over each filename, reads the corresponding image using cv2.imread(), and appends it to a list. Finally, the list images contain NumPy arrays, each representing an image from the folder.

```
# Define the new colorspace you want to convert to (e.g., cv2.COLOR_BGR2GRAY)
new_colorspace = cv2.COLOR_BGR2GRAY
# Loop through the images and convert colorspace for each one
converted_images = [cv2.cvtColor(image, new_colorspace) for image in images]
```

In this code part, we have changed the colorspace of the image from BGR(Blue - Green - Red) to grayscale. This is to make the preprocessing techniques more computationally efficient. The colorspace conversion was applied to every single image via the usage of a for loop.

```
# Apply Gaussian blur to each image in the list
blurred_images = [cv2.GaussianBlur(image, (9, 9), 0) for image in converted_images]
```

In this code part, we have applied Gaussian Blur to every single image in the list via the usage of a for loop. This step was to reduce the noise since the original image contained salt and pepper noise mainly. Gaussian blur reduced that.

```
for idx, image in enumerate(blurred_images):
  plt.figure()
  plt.imshow(image)  # Convert BGR to RGB
  plt.title(f"Image {idx+1}")
  plt.axis('off')  # Turn off axis
  plt.show()
```

In this continued part of the code, we printed all images in the terminal via a for loop. To visualize.

```
sharpened_images = []

# Iterate over each image in the array
for image in blurred_images:
  # Convert the image to grayscale
  gray = image

  # Apply Laplacian filter to detect edges
  laplacian = cv2.Laplacian(gray, cv2.CV_64F, ksize=3)

  # Increase the strength of the sharpening effect
  sharpened_img = cv2.convertScaleAbs(gray - 0.1 * laplacian)

  # Append the sharpened image to the array
  sharpened_images.append(sharpened_img)

for idx, image in enumerate(sharpened_images):
  plt.figure()
  plt.imshow(image)
  plt.title(f"Image {idx+1}")
  plt.axis('off')  # Turn off axis
  plt.show()
```

In this code, an array was created which is called sharpened_images. This array is basically where we will store the images that have been processed via a sharpening filter. Using a for loop to iterate through all images, a laplacian filter was applied to detect edges in the grayscale image. Then the sharpness of the image was enhanced by subtracting a fraction of the Laplacian-filtered image from the original grayscale image. The result was appended in the array sharpened_images. Then using a for loop, all images that have been sharpened have been printed.

```
# Define gamma value for gamma correction
gamma = 0.9  # Adjust gamma value as needed
# Apply gamma correction to each image in the array
processed_images = []
for image in sharpened_images:
```

```
  # Calculate gamma correction table
  inv_gamma = 1.0 / gamma
  table = np.array([((i / 255.0) ** inv_gamma) * 255
                    for i in np.arange(0, 256)]).astype(np.uint8)

  # Apply gamma correction
  gamma_corrected_image = cv2.LUT(image, table)
  # Append the processed image to the list
  processed_images.append(gamma_corrected_image)

for idx, image in enumerate(processed_images):
  plt.figure()
  plt.imshow(image)
  plt.title(f"Image {idx+1}")
  plt.axis('off')  # Turn off axis
  plt.show()
```

In this part of the code, gamma correction was applied as a final step in the preprocessing stage. In this section of the code, a gamma value of 0.9 is defined for gamma correction, which can be adjusted according to the desired effect. It then iterates over a list of sharpened images, generating a lookup table for gamma correction based on the inverse of the gamma value. This table is applied to each pixel intensity of the image using OpenCV's LUT (Look-Up Table) function, resulting in gamma-corrected images. Finally, the processed images are stored in the list named processed_images. Adjustments to the gamma value can affect the overall brightness and contrast of the images, providing control over their visual appearance.

```
reader = easyocr.Reader(['en'], gpu = True)
# Define the list of characters you want to allow
allowed_characters = '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ'
for image in sharpened_images:
 results = reader.readtext(image,allowlist=allowed_characters)
 print(results)
 print(".")
```

In this final code section, all the preprocessed images are passed to the OCR library easyOCR using the function .Reader. A for loop was used so that the images are passed to OCR and the results are then printed.

Results:

In this section, the output from the terminal is shown. The output here shows the coordinate that the OCR cropped from. That's the first element in the tuple. Then it prints the detected text and it shows the probability in EasyOCR, the probability represents the confidence level or certainty of the detected text recognition result.

It indicates the likelihood that the recognized text is accurate. Higher probability values generally indicate higher confidence in the accuracy of the detected text, while lower probability values may suggest lower confidence or uncertainty.

Some license plate numbers were correctly extracted. Some license plates were extracted correctly but contained 1 or 2 letters extracted wrong. This can be explained because the OCR library does not employ any artificial intelligence technology and it can not differentiate between different texts. The OCR library approach is a bit outdated. This problem will be tackled in milestones where a CNN will be applied and will be trained using a large dataset which will be much more accurate and will give better results. The text of the licence plate is in in ‘’ speech marks. The rest of the output shows the coordinates. A full stop indicates the output of a separate image

```
WARNING:easyocr.easyocr:Neither CUDA nor MPS are available - defaulting to CPU. Note: This module is much faster with a GPU.
[([[23, 13], [225, 13], [225, 57], [23, 57]], 'DLZCN5617', 0.4679587525612845)]
.
[([[78, 70], [1274, 70], [1274, 262], [78, 262]], 'UP14CP6748', 0.6844606223094094)]
.
[([[2, 6], [226, 6], [226, 54], [2, 54]], 'DL12CG6648', 0.8437282022228374)]
.
[([[0, 5], [291, 5], [291, 78], [0, 78]], 'HH12DE143Z', 0.10871306495576424)]
.
[([[26, 6], [292, 6], [292, 75], [26, 75]], 'HPESXG00', 0.19572763395573692)]
.
[([[31, 10], [262, 10], [262, 61], [31, 61]], 'DLGCLB1237', 0.309604480034455)]
.
[([[16, 38], [361, 38], [361, 110], [16, 110]], 'KLIQAV6633', 0.6744147564027667)]
.
[([[0, 44], [38, 44], [38, 70], [0, 70]], 'ND', 0.7104303758224771), ([[52, 16], [446, 16], [446, 95], [52, 95]], 'DL49AK49', 0.9977750958728817)]
.
[([[16, 12], [228, 12], [228, 64], [16, 64]], 'MH03BS7778', 0.3708624112650662)]
.
[([[48, 74], [112, 74], [112, 106], [48, 106]], 'AND', 0.8017675229899069), ([[96, 24], [692, 24], [692, 145], [96, 145]], 'KA2929991', 0.6433184330076418)]
.
[([[42, 75], [1160, 75], [1160, 420], [42, 420]], 'KL31B', 0.9998817531255133), ([[1281, 79], [2055, 79], [2055, 405], [1281, 405]], '40CC', 0.40322476625442505)]
.
[([[23, 19], [187, 19], [187, 55], [23, 55]], 'HR696969', 0.9895477101516734)]
.
[([[13, 24], [289, 24], [289, 85], [13, 85]], 'HKI4DT8831', 0.2891933557412826)]
.
```