**Tahsin Raihan**

**Dept Of CSE**

**Rajshahi University of Engineering & Technology**

**Bangla Optical Character Recognition**

**Abstract**

**This report documents the development and implementation of an Optical Character Recognition (OCR) system for Bangla printed text. The system uses a hybrid CNN-LSTM deep learning architecture for text line detection and recognition. Additionally, it incorporates preprocessing techniques for noise reduction and adaptive thresholding. The system is validated using a dataset of images and corresponding word files, achieving accurate text-line segmentation and robust recognition.**

**Introduction**

**Optical Character Recognition (OCR) plays a crucial role in digitizing printed or handwritten text for analysis. The presented system is tailored to Bangla script, addressing the complexity of non-Latin alphabets. It utilizes convolutional and sequential neural networks to process image lines and recognize characters efficiently.**

**The project aims to:**

1. **Preprocess input images for line detection.**
2. **Train a CNN-LSTM-based model to classify and recognize Bangla text.**
3. **Validate the system using a Bangla text dataset consisting of scanned documents and word files.**

**Methodology**

**1. Dataset Description  
The dataset comprises:**

* **Images stored in structured subfolders.**
* **Corresponding word files with text annotations.**
* **An Excel file maps images to their respective word files.**

**2. Preprocessing**

* **Image Preprocessing: Images are resized to 1024 × 682 pixels, Gaussian blurred for noise removal, and thresholded for binary segmentation.**

This function preprocesses an image by performing the following steps:

1. Reads the image in grayscale format.
2. Resizes the image to a fixed dimension of 1024 x 682 pixels.
3. Applies Gaussian blurring to reduce noise.
4. Converts the image to a binary format using adaptive thresholding.

**Parameters:**

* image\_path (str): The file path to the input image.

**Returns:**

* **thresh (ndarray): The preprocessed binary image (2D NumPy array).**
* **None: If the image file cannot be loaded (e.g., invalid file path or unsupported format).**

**:**

**Function Name: preprocess\_image**

**Description:**

**This function preprocesses an image by performing the following steps:**

1. **Reads the image in grayscale format.**
2. **Resizes the image to a fixed dimension of 1024 x 682 pixels.**
3. **Applies Gaussian blurring to reduce noise.**
4. **Converts the image to a binary format using adaptive thresholding.**

**Parameters:**

* **image\_path (str): The file path to the input image.**

**Returns:**

* **thresh (ndarray): The preprocessed binary image (2D NumPy array).**
* **None: If the image file cannot be loaded (e.g., invalid file path or unsupported format).**

**Algorithm:**

1. **Image Loading:**
   * **Input: File path of the image.**
   * **Process:**
     + **Use cv2.imread with the flag cv2.IMREAD\_GRAYSCALE to load the image in grayscale mode. This reduces the data to one channel and simplifies further processing.**
   * **Output: A 2D grayscale image array or None if the image cannot be loaded.**
2. **Image Resizing:**

* **Input: Grayscale image.**
* **Process:**
  + **Use cv2.resize to scale the image to a fixed resolution of 1024 x 682 pixels, preserving aspect ratio.**
  + **Resizing ensures a consistent size for processing and is essential for applications requiring uniform input dimensions.**
* **Output: Resized grayscale image.**

1. **Gaussian Blurring:**

* **Input: Resized grayscale image.**
* **Process:**
  + **Apply cv2.GaussianBlur with a kernel size of (5, 5) and a standard deviation of 0.**
  + **This step smoothens the image, reducing high-frequency noise while retaining edge information.**
* **Output: Blurred grayscale image.**

1. **Adaptive Thresholding:**

* **Input: Blurred grayscale image.**
* **Process:**
  + **Convert the image to a binary format using cv2.adaptiveThreshold.**
  + **Parameters:**
    - **255: Maximum value for binary output.**
    - **cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C: Use a Gaussian-weighted mean for adaptive thresholding.**
    - **cv2.THRESH\_BINARY: Convert pixels to either 0 or the maximum value (255).**
    - **11: Size of the pixel neighborhood considered for thresholding.**
    - **2: Constant subtracted from the mean.**
  + **Adaptive thresholding handles varying illumination in the image effectively.**
* **Output: Thresholded binary image.**

1. **Return:**

* **Return the thresholded binary image. If the image loading fails, return None.**

**Line Detection: Custom logic detects and segments text lines from the binary image.We have used**

**Boundary fill algorithm to detect this .**

**Algorithm BOUNDARY\_FILL (x, y, img: IMAGE)**

**/\*the algorithm fill a region in image img from black to white color. Filling starts at point (x, y). It also copy the filled region into another image img2\*/**

**1. if pixel (x, y) in image img is black then**

**2. Copy the current pixel value to image img2**

**3. Reset the current pixel location with a white pixel**

**4.Call BOUNDARY\_FILL with values (x+1, y)**

**5.Call BOUNDARY\_FILL with values (x-1, y)**

**6.Call BOUNDARY\_FILL with values (x, y+1)**

**7.Call BOUNDARY\_FILL with values (x, y-1)**

**8.Call BOUNDARY\_FILL with values (x-1, y-1)**

**9.Call BOUNDARY\_FILL with values (x-1, y+1)**

**10.Call BOUNDARY\_FILL with values (x+1, y-1)**

**11.Call BOUNDARY\_FILL with values (x+1, y+1)**

**12. end if**

**Detect\_lines :**

**This function detects horizontal lines of content in a binary thresholded image. Each line corresponds to a contiguous block of rows containing black pixels (value 0 in a binary image).**

**Parameters:**

* **image (ndarray): A binary (thresholded) image where pixels are either 0 (black) or 255 (white).**

**Returns:**

* **lines (list of ndarray): A list of image slices, each representing a detected line.**

**Algorithm:**

* 1. **Initialize Variables:**

1. **lines: An empty list to store detected lines.**
2. **in\_line: A boolean flag to track whether the current row is part of a line.**
3. **Initialize as False since we start outside a line.**
   1. **Iterate Through Rows:**
4. **Loop over each row of the image using its shape (image.shape[0] gives the number of rows).**
5. **For each row, check if it contains black pixels (value 0).**
   1. **Check for Line Start:**
6. **If black pixels are found (np.sum(image[row] == 0) > 0) and in\_line is False:**
   1. **Record the current row index as the starting row (line\_start) of a potential line.**
   2. **Set in\_line to True.**
   3. **Check for Line End:**
7. **If no black pixels are found in the row (np.sum(image[row] == 0) == 0) and in\_line is True:**
   1. **Record the current row index as the ending row (line\_end) of the detected line.**
   2. **Extract the slice of the image corresponding to the detected line (image[line\_start:line\_end, :]).**
   3. **Append the slice to the lines list.**
   4. **Reset in\_line to False.**
   5. **Return Detected Lines:**
8. **After the loop completes, return the list of detected lines.**

**We can summarize the algorithm as follows :**

1. **Input: A binary image with dimensions (height, width) where pixel values are either:**
   * **0 (black): Represents foreground or text content.**
   * **255 (white): Represents background.**
2. **Detection Logic:**
   * **Rows containing black pixels are part of a line.**
   * **Transition from a row with black pixels to one without marks the end of a line.**
3. **Line Extraction:**
   * **For each detected line, the row indices (line\_start to line\_end) define its vertical bounds.**
   * **The line slice (image[line\_start:line\_end, :]) is appended to the lines list.**
4. **Edge Case Handling:**
   * **If the last row of the image contains black pixels, the function correctly includes it in the final line due to the loop's logic.**

**3. Text and Label Processing**

* **Word files are parsed for line-specific text annotations.**
* **Labels are encoded using the LabelEncoder class to map text to numerical labels.**

**4. CNN-LSTM Model Architecture  
The model is a hybrid design:**

* **Feature Extraction: Convolutional layers (Conv2D) detect spatial patterns in input text lines.**
* **Temporal Context: LSTM layers capture sequence dependencies in Bangla script.**
* **Output Layer: A dense layer classifies characters/texts using softmax activation.**

**This neural network combines Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) using an LSTM (Long Short-Term Memory) layer to leverage spatial and sequential features, typically used for tasks such as OCR (Optical Character Recognition) or sequential image processing.**

**1. Input Layer**

* **Shape: (32, 256, 1)  
  This specifies that the input is a grayscale image with dimensions:**
  + **Height: 32 pixels**
  + **Width: 256 pixels**
  + **Channels: 1 (grayscale).**

**2. Convolutional Layer 1**

**Python code : Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 256, 1))**

* **Purpose: Extracts low-level spatial features using 32 filters.**
* **Kernel Size: (3, 3) - A small receptive field captures localized patterns such as edges and textures.**
* **Activation: ReLU introduces non-linearity.**

**3. MaxPooling Layer 1**

**Python code : MaxPooling2D((2, 2))**

* **Purpose: Downsamples the feature maps by a factor of 2, reducing the spatial dimensions and computational cost.**
* **Pooling Window: (2, 2).**

**4. Convolutional Layer 2**

**Python code : Conv2D(64, (3, 3), activation='relu')**

* **Purpose: Extracts higher-level features using 64 filters.**
* **Kernel Size: (3, 3).**
* **Activation: ReLU.**

**5. MaxPooling Layer 2**

**Python code :** **MaxPooling2D((2, 2))**

* **Further reduces the spatial dimensions.**

**6. Convolutional Layer 3**

**Python code :** **Conv2D(128, (3, 3), activation='relu')**

* **Purpose: Captures more complex features using 128 filters.**
* **Kernel Size: (3, 3).**
* **Activation: ReLU.**

**7. MaxPooling Layer 3**

**Python code :** **MaxPooling2D((2, 2))**

* **Purpose: Final downsampling to make the data suitable for dense layers while retaining essential features.**

**8. Flatten Layer**

**Python code :** **Flatten()**

* **Purpose: Converts the 2D feature maps into a 1D feature vector for fully connected layers.**

**9. Dense Layer**

**Python code :**

**Dense(128, activation='relu')**

* **Purpose: Applies a fully connected layer with 128 units to learn complex patterns from the flattened features.**
* **Activation: ReLU.**

**10. Dropout Layer**

**Python code :**

**Dropout(0.5)**

* **Purpose: Reduces overfitting by randomly setting 50% of the neurons to zero during training.**

**11. Reshape Layer**

**Python code :**

**Reshape((1, 128))**

* **Purpose: Reshapes the dense layer's output to be compatible with the LSTM layer, adding a temporal dimension.**
* **Output Shape: (1, 128) - A sequence of length 1 with 128 features.**

**12. LSTM Layer**

**Python code :**

**LSTM(64, return\_sequences=False)**

* **Purpose: Processes the temporal features, enabling the model to capture sequential dependencies in the input.**
* **Units: 64 - Number of LSTM units.**
* **return\_sequences=False: Returns only the final output instead of the entire sequence.**

**13. Output Layer**

**Python code :**

**Dense(len(label\_encoder.classes\_), activation='softmax')**

* **Purpose: Produces class probabilities for each output class.**
* **Units: len(label\_encoder.classes\_) - Number of output classes based on the label encoder.**
* **Activation: Softmax ensures that the output is a probability distribution over the classes.**

**Implementation**

**1. Preprocessing Pipeline  
Functions handle image resizing, line segmentation, and normalization. Text extraction logic links detected lines to their word-file annotations.**

**2. Model Training**

* **Training Split: 80% of data is used for training; 20% for validation.**
* **Loss Function: Sparse Categorical Crossentropy to accommodate labeled classes.**
* **Optimizer: Adam optimizer fine-tunes the network weights.**

**3. Line Verification  
A verification script compares detected image lines to corresponding word-file annotations for qualitative analysis.**

**Evaluation Metrics**

**1. Accuracy  
The CNN-LSTM model is evaluated on validation and test datasets for accuracy.**

**2. Text Match Accuracy  
The SequenceMatcher metric compares OCR-detected text with ground truth annotations for precision.**

**Results**

* **Test Accuracy: 12.8%**
* **Line Detection: The preprocessing and line detection functions robustly segmented Bangla text into lines with minimal overlap.**
* **Text Recognition: Recognized text from segmented lines closely matched word-file annotations.**

**Conclusion**

**This OCR system successfully digitized Bangla text from printed images. The CNN-LSTM architecture proved effective for combining spatial and sequential feature extraction. Future improvements could include:**

* **Incorporating a Transformer-based architecture for enhanced recognition.**
* **Expanding the dataset to include handwritten Bangla text.**

**References**

* **Dataset: Kaggle’s "Bangla Text Detection and Recognition."**
* **Libraries: TensorFlow, OpenCV, scikit-learn, and Matplotlib.**
* **Neural net based complete character recognition scheme for Bangla printed text books by S** **K Alamgir Hossain & Tamanna Tabassum**
* **Bangla Optical Character Recognition from Printed Text using Tesseract Engine by MDYamin Faruque , MDZahin Adeeb ,Muhammad Maswood Kamal ,Redwan Ahmed**
* **RECOGNITION OFCONJUNCTIVE BANGLACHARACTERS BY ARTIFICIAL NEURALNETWORK Abdur Rahim Md. Forkan, Shuvabrata Saha, Md. Mahfuzur Rahman, Md. Abdus Sattar**