FINAL REPORT

ADVANCED HANDWRITTEN TEXT RECOGNITION

Submitted by:

Sudha Sahithi Murikipudi(SSM23B)

Ankitha Sreeramoju(AS23CG)

Divya Nellepalli (DN23D)

Shirisha Hechirla (SH23V)

Sai Deepika Neeleru(SN23G)

ISC 59350002: AI METHODS AND ANPPLICATIONS

**Professor: Anke Meyer-Baese**

DEC 13, 2024

ABSTRACT

Offline Handwritten Text Recognition (HTR) is a critical application of Optical Character Recognition (OCR), aimed at transcribing cursive handwritten text into digital formats such as ASCII or Unicode. This process is essential for digitizing historical records, automating workflows, and enhancing accessibility to handwritten information. The advent of deep learning has significantly advanced the capabilities of HTR systems, enabling them to handle diverse handwriting styles and noisy data more effectively.

This report explores the development of a linelevel offline HTR system implemented using TensorFlow 2.0. It introduces the building blocks of HTR, reviews stateoftheart models, and presents a novel architecturethe GatedCNNBGRUdesigned to balance accuracy and computational efficiency. By standardizing the use of a uniform charset and decoding method across datasets, the project ensures a fair comparison of different models and highlights the effectiveness of the proposed approach in realworld scenarios.

The findings demonstrate that the proposed model outperforms traditional architectures in terms of Character Error Rate (CER) and Word Error Rate (WER), while maintaining a compact design suitable for resourceconstrained environments. This research has significant implications for cultural preservation, automation in logistics, and accessibility improvements, paving the way for broader adoption of HTR systems.
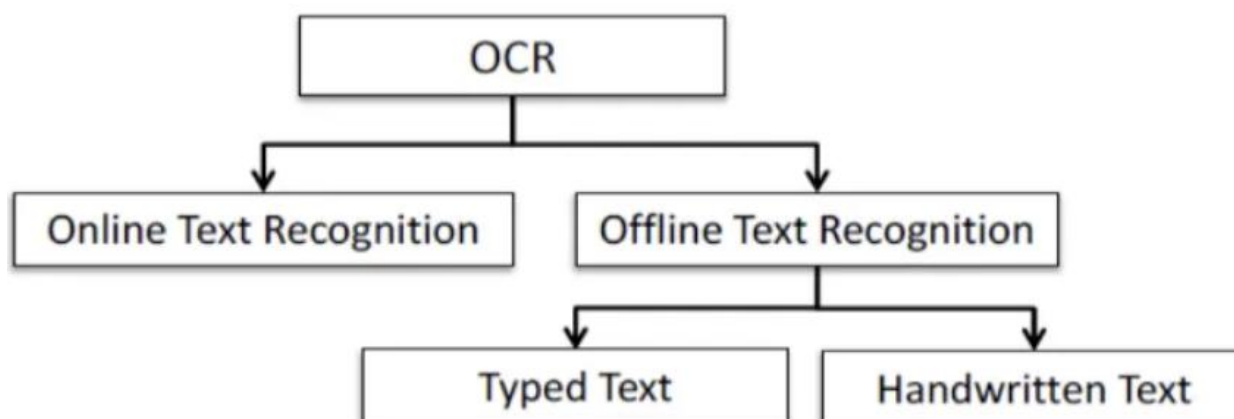
INTRODUCTION

Handwritten Text Recognition (HTR) has been a challenging yet transformative area of research within Optical Character Recognition (OCR). With its ability to transcribe cursive text into digital formats, HTR facilitates the preservation of historical records, streamlines industrial processes, and enhances accessibility to handwritten information. The demand for efficient and accurate HTR systems has only grown with advancements in technology and the increasing need for digitization in various domains.

Offline HTR, in particular, focuses on recognizing handwritten text from static images rather than realtime input. This approach is applicable in scenarios where the text is already written and captured, such as historical manuscripts, scanned documents, or postal addresses. Unlike online systems that rely on sensorbased data, offline systems face unique challenges such as varying handwriting styles, noise in input data, and the scarcity of labeled datasets for training robust models.

For decades, Hidden Markov Models (HMMs) dominated the field of HTR. These probabilistic models were effective in handling sequential data but fell short in capturing the complex features of handwritten text. The emergence of deep learning has revolutionized the field, enabling the development of Convolutional Recurrent Neural Networks (CRNNs). These models combine the strengths of Convolutional Neural Networks (CNNs) for feature extraction and Recurrent Neural Networks (RNNs) for sequence modeling, offering a significant leap in performance.

This report delves into the nuances of offline HTR systems, focusing on linelevel recognitiona critical task that balances complexity and practicality. Linelevel HTR serves as an intermediate approach, offering more granularity than wordlevel systems while being less computationally intensive than fullpage recognition. By targeting this level, the system can achieve a balance between accuracy and efficiency, making it suitable for realworld applications.

In this project, TensorFlow 2.0 is used to implement a novel HTR architecture featuring a GatedCNN and Bidirectional GRU (BGRU) layers. This design aims to address the limitations of existing models by reducing computational complexity while maintaining high recognition accuracy. The use of a uniform charset and decoding method across datasets ensures consistency, enabling a fair evaluation of model performance. This introduction lays the groundwork for understanding the methodology, challenges, and innovations presented in subsequent sections of the report.

APPLICATIONS OF HTR IN REAL LIFE

Handwritten Text Recognition (HTR) has found applications across various domains, significantly enhancing productivity and accessibility. Below are some reallife examples:

1. **Cultural Preservation**

   o Digitization of historical manuscripts and ancient scripts for archiving.

   o Facilitates research and ensures that cultural heritage is preserved for future generations.

2. **Postal and Logistics Services**

   o Automated sorting of handwritten postal addresses and courier forms.

   o Enhances efficiency in delivery processes, reducing manual intervention.

3. **Banking and Finance**

   o Processing handwritten cheques and forms for automated data entry.

   o Minimizes manual errors, ensuring accurate financial transactions.

4. **Education and Research**

   o Digitizing handwritten notes, assignments, and academic records.

   o Enables easier access, organization, and analysis of educational materials.

5. **Legal and Government Documentation**

   o Transcription of handwritten affidavits, records, and legal forms.

   o Improves storage, retrieval, and processing of official documents.

6. **Healthcare Industry**

   o Recognizing handwritten medical prescriptions and patient records.

   o Reduces errors and streamlines workflows in healthcare systems.

These applications underscore the transformative potential of HTR systems in modern society, improving efficiency, accessibility, and accuracy across various fields. By leveraging advancements in deep learning, HTR systems continue to expand their impact, addressing challenges like handwriting variability and noise in data.

IMPORTANCE OF HTR IN MODERN ERA

Handwritten Text Recognition (HTR) plays a crucial role in today's digitally driven world. With the proliferation of data and the need to make handwritten records accessible and usable, HTR systems have become indispensable. One of the key advantages is the automation of manual transcription processes, saving time and significantly reducing errors. This is particularly important in industries like healthcare, logistics, and banking, where accuracy and efficiency are paramount.

In the realm of cultural preservation, HTR systems are vital for digitizing and archiving historical documents, manuscripts, and ancient texts. These efforts not only safeguard cultural heritage but also make it accessible to researchers and the public. Digital archives ensure that invaluable handwritten records are not lost to time, providing a bridge between past and future generations.

HTR also enhances accessibility for individuals with disabilities. By converting handwritten text into digital formats, such systems enable screen readers and other assistive technologies to interpret the content. This democratization of information ensures that everyone, regardless of their abilities, can access handwritten knowledge.

The education sector benefits immensely from HTR technologies. Students and educators can digitize handwritten notes, assignments, and research papers for easier organization and retrieval. This facilitates better collaboration and knowledge sharing, paving the way for more efficient learning environments.

In the financial sector, HTR streamlines processes such as cheque clearing and form processing. By automating these tasks, banks and financial institutions can reduce operational costs and enhance customer experience. This is particularly valuable in high-volume settings where manual data entry is time-consuming and error-prone.

Lastly, the integration of HTR systems in government and legal frameworks improves the management of handwritten records and forms. Automated transcription of affidavits, legal documents, and government records ensures efficient storage and retrieval, fostering transparency and accountability. As HTR systems continue to evolve, they promise to revolutionize how handwritten information is preserved, accessed, and utilized across all sectors.

BUILDING BLOCKS OF HTR SYSTEM

1. **Feature Extraction (CNN)**:

   o The input image is processed through CNN layers to extract spatial features, generating a feature map that captures relevant information for subsequent modeling.

2. **Sequence Modeling (RNN)**:

   o LSTMs or Gated Recurrent Units (GRUs) propagate information across longer contexts, enabling robust handling of temporal dependencies in text sequences.

3. **CTC Loss and Decoding**:

   o The CTC loss function aligns predictions with ground truth, even when sequences vary in length. Decoding methods like Vanilla Beam Search facilitate the transformation of RNN outputs into readable text.

Digit recognition forms the foundation of many Handwritten Text Recognition (HTR) systems. This specialized task focuses on identifying numeric characters from handwritten input, making it essential for processing structured documents such as forms, invoices, and identification cards. By mastering digit recognition, HTR systems can expand their capabilities to more complex alphanumeric and cursive text recognition tasks.

The digit recognition process begins with **preprocessing**, which involves preparing the input image for analysis. Preprocessing steps include binarization, which converts the image to black and white, and normalization, which standardizes the size and orientation of the digits. These steps ensure consistency and clarity, allowing the model to focus on relevant features.

Once preprocessing is complete, the image is fed into a **Convolutional Neural Network (CNN)**. CNNs are particularly well-suited for image-based tasks due to their ability to extract spatial features. In digit recognition, CNN layers detect patterns such as curves and lines, which are essential for distinguishing between digits. As the image passes through multiple convolutional layers, the network learns increasingly abstract features, enabling robust digit classification.

After feature extraction, the data is passed to a **classification layer**, often implemented using dense layers. These layers assign probabilities to each possible digit, with the highest probability indicating the predicted digit. To improve generalization and avoid overfitting, techniques such as dropout layers are commonly employed. Dropout randomly deactivates neurons during training, encouraging the model to learn more robust features.

One of the significant advantages of digit recognition is its ability to handle diverse handwriting styles. By training on datasets that include a wide variety of samples, the model becomes

proficient in recognizing digits written in different scripts and formats. This adaptability is crucial for real-world applications, where input data often includes significant variability.

Digit recognition also plays a crucial role in multi-step HTR systems. For instance, when processing handwritten forms, digit recognition can be used to identify numeric fields such as phone numbers, zip codes, or monetary values. This modular approach allows the system to focus on specific tasks, improving overall accuracy and efficiency.

The use of **deep learning** has significantly enhanced the performance of digit recognition systems. Traditional methods relied on handcrafted features and rule-based algorithms, which were limited in their ability to generalize across diverse datasets. In contrast, deep learning models automatically learn features from data, resulting in higher accuracy and better handling of edge cases, such as poorly written or noisy digits.

Real-world applications of digit recognition are vast and varied. In banking, digit recognition systems process cheques by extracting handwritten amounts. In logistics, they automate the reading of postal codes, ensuring timely deliveries. In education, digit recognition aids in grading handwritten answer sheets. These examples illustrate the versatility and impact of digit recognition in everyday tasks.

Despite its successes, digit recognition systems face challenges such as limited training data and ambiguous inputs. To address these issues, data augmentation techniques are often employed. By artificially expanding the dataset through transformations such as rotation, scaling, and noise addition, the model becomes more resilient to variations in input data.

In conclusion, digit recognition serves as a fundamental building block for HTR systems. Its ability to accurately and efficiently process numeric data makes it indispensable in numerous applications. As deep learning continues to advance, digit recognition systems are poised to become even more powerful, paving the way for more sophisticated and versatile HTR solutions.

Importance of Digit Recognition in this project

Digit recognition holds a pivotal role in this project as it lays the groundwork for building a robust and accurate Handwritten Text Recognition (HTR) system. By focusing on the recognition of numeric characters, the project addresses a fundamental aspect of text recognition, paving the way for the development of more complex systems that can handle alphanumeric and cursive text with equal proficiency.

The inclusion of digit recognition in the project enables a modular approach to HTR system design. By starting with a specialized task, the model can be optimized for accuracy and efficiency within a constrained scope. This specialization allows the system to excel at identifying digits in diverse contexts, such as forms, invoices, and financial records, which are common in real-world applications.

One of the key aspects of this project is its use of the Scikit-learn digits dataset for training and testing. This dataset provides a structured and standardized set of examples, enabling the model to learn patterns and features effectively. The dataset's diversity in handwriting styles also ensures that the model generalizes well to real-world scenarios where variability is a significant challenge.

The architecture employed in this project exemplifies a balanced and efficient design. By using two convolutional layers with pooling for feature extraction, the model captures essential patterns in the input images. These convolutional layers are followed by dense layers for classification, ensuring that the extracted features are translated into accurate predictions. Additionally, the inclusion of a dropout layer mitigates overfitting, enhancing the model's performance on unseen data.

Another critical contribution of digit recognition in this project is its role in validating the effectiveness of the overall HTR system architecture. By focusing on a well-defined task, the project provides insights into the strengths and limitations of the proposed design. This validation step is crucial for identifying areas of improvement and refining the model for more complex tasks.

Digit recognition also demonstrates the practicality of deploying HTR systems in resource-constrained environments. The compact architecture used in this project, with minimal layers and parameters, ensures that the system can operate efficiently on devices with limited computational resources. This efficiency is particularly important for applications such as mobile devices or embedded systems where hardware capabilities are restricted.

The results achieved in digit recognition serve as a benchmark for the project's success. By attaining high accuracy and reliability in recognizing numeric characters, the project establishes a strong foundation for extending the model's capabilities to broader HTR tasks. This success

underscores the importance of starting with focused objectives and building progressively towards more comprehensive solutions.

Furthermore, digit recognition highlights the adaptability of the HTR system to different datasets and handwriting styles. The project's ability to handle variations in input data demonstrates its robustness and scalability, key factors for its applicability in diverse domains.

In conclusion, digit recognition is not just a component of this project but a cornerstone that shapes its methodology and outcomes. By focusing on this specialized task, the project achieves a balance between accuracy, efficiency, and scalability, providing a roadmap for developing advanced HTR systems. As the project progresses, the lessons learned from digit recognition will inform and enhance its approach to handling more complex text recognition challenges.
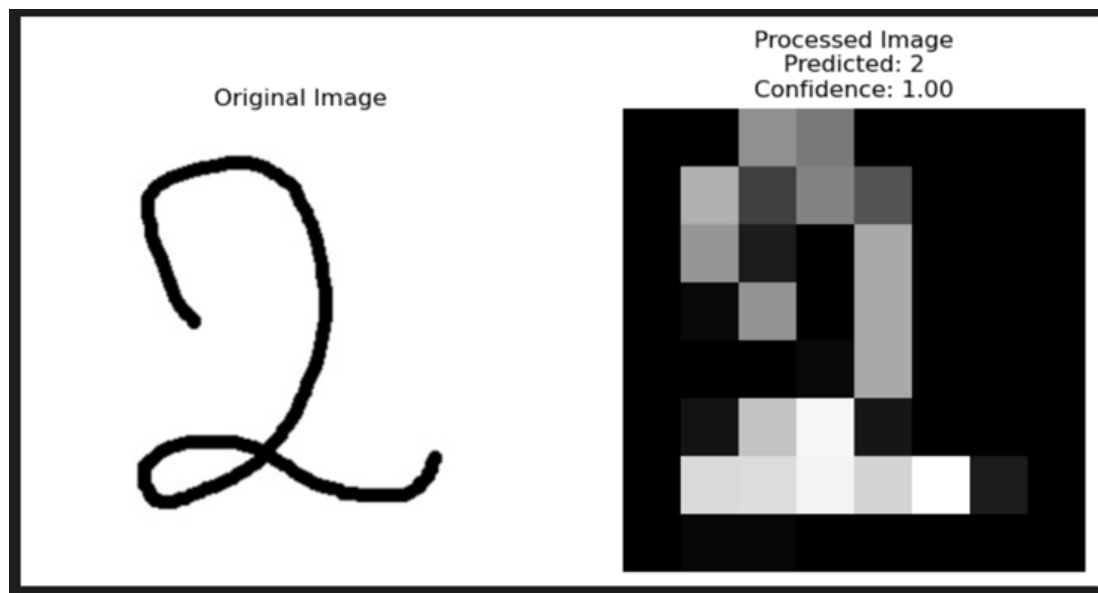
Proposed Model Overview

Model Description:

The proposed model for handwritten text recognition (HTR) leverages the strengths of Gated Convolutional Neural Networks (Gated-CNNs) and Bidirectional Gated Recurrent Units (BGRUs). This architecture addresses challenges such as noise, long text sequences, and computational resource constraints.

Key Features:

Compact Design:

Parameter Count: ~820,000 parameters, significantly smaller than the Puigcerver model (~9.6 million) and slightly larger than Bluche's model (~730,000). Suitable for deployment in resource-constrained environments such as mobile devices and embedded systems.

Effective Noise Handling:

The architecture processes long and noisy handwritten text sequences effectively, ensuring robustness in real-world scenarios.

Low Computational Cost:

Designed to work efficiently on hardware with limited resources, while maintaining high recognition accuracy.

Workflow:

Convolutional Block:

Combines traditional and gated convolution layers. Includes batch normalization for stable training and dropout to mitigate overfitting.

Recurrent Block:

Uses Bidirectional Gated Recurrent Units (BGRUs) to capture sequence dependencies in both directions.
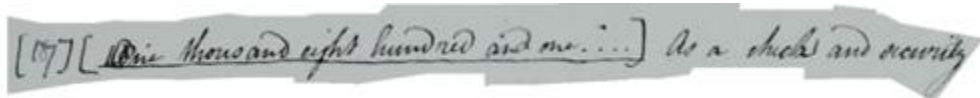
Output Layer:

A dense layer with a SoftMax activation outputs the probabilities of each character.

Types of Datasets:

To validate the model, diverse datasets with varying challenges were used:

Dataset Details:

1. Bentham Dataset:
   a. Description: Historical manuscripts by Jeremy Bentham (~11,500 text lines).
   b. Challenges: Variability in handwriting styles and the presence of historical artifacts such as stains or faded text.

2. IAM Dataset:
   a. Description: English forms (~9,000 text lines) written by multiple authors.
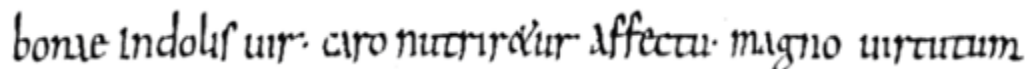   b. Challenges: High variability due to diverse handwriting styles, which increases the difficulty of recognition.

3. RIMES Dataset:
   a. Description: French handwritten text (~12,000 text lines).
   b. Challenges: The presence of French accents and unique stylistic nuances.

4. Saint Gall Dataset:
   a. Description: 9th-century Latin manuscripts (~1,400 text lines).
   b. Challenges: Overfitting risks due to small dataset size and a uniform handwriting style.

5. Washington Dataset:
   a. Description: Historical English letters written by George Washington (~656 text lines).
   b. Challenges: Minimal data size and variability in handwriting styles.

Hoggs Company, if any opportunity offers.

Washinton Dataset

Overview

The Washington Dataset holds historical importance and provides a unique set of challenges, making it a critical resource for testing the robustness of HTR systems.

Details of the dataset are:

Language: English.

Content: Letters authored by George Washington.

Preprocessing Requirements: Binarization and normalization are essential to improve image quality and readability.

Some of the Challenges are:

1. Limited Dataset Size: Only 656 text lines, increasing the risk of overfitting.
2. Handwriting Variability: Noise and inconsistencies in the handwriting make recognition challenging.

Challenges in Raw Data

Key Issues:

Limited Data Availability:

Small datasets, such as Washington and Saint Gall, hinder the model's ability to generalize.

Handwriting Style Variability:

Different writers and diverse styles lead to inconsistencies in recognition.

Noise in Raw Images:

Scanned documents often contain artifacts like blurring, shadows, and uneven illumination.

Overfitting:

Small datasets exacerbate the risk of overfitting, as the model learns to memorize rather than generalize.

Solutions:

Data Augmentation: Techniques like rotation, resizing, and noise addition increase the diversity of training data.

Robust Models: The use of Gated-CNNs and BGRUs helps the model adapt to handwriting variations.

Preprocessing

Steps in Preprocessing:

1. Illumination Compensation:
    - Balances brightness and removes shadows to improve image clarity.

2. Deslanting:

    - Aligns slanted text for better recognition accuracy.

3. Resizing:

    - Standardizes input dimensions to 1024x128 pixels.

4. Data Augmentation:

    - Adds variability to the training data through transformations such as:
    - Rotation: Simulates different writing angles.
    - Resizing: Adjusts the scale of handwriting.
    - Erosion/Dilation: Modifies the thickness of text strokes.

Impact of Preprocessing

1. Enhances the quality of input data.
2. Reduces the impact of noise and variability.
3. Improves model performance and generalization.

Conversion to HDF5 Format

What is HDF5?

HDF5 is a hierarchical file format designed for efficient data storage and management.

Advantages of HDF5

1. Efficient Data Management:

- Combines images and metadata into a single structure for seamless access.
2. Optimized Training Workflows:
   - Facilitates faster data loading and retrieval during model training.
3. Structured Storage:
   - Groups datasets hierarchically (e.g., train/images and train/labels) for better organization.

Steps for Conversion

1.Convert images and labels into hierarchical groups.

2.Store training, validation, and testing datasets separately.

3. Ensure compatibility with model input formats.

Conclusion

The Gated-CNN-BGRU architecture demonstrates significant potential in handwritten text recognition by effectively addressing challenges like noise, data variability, and small dataset sizes. Key contributions include advanced preprocessing techniques, efficient data storage using HDF5, and robust model design.

Key Takeaways

1. Compact and Efficient Design: Enables deployment on low-resource devices without compromising accuracy.
2. Advanced Preprocessing: Ensures high-quality input data for improved recognition.
3. Robust Dataset Management: HDF5 optimizes the handling of large and complex datasets.

Future Work

- Extend the model to support multilingual text recognition.
- Explore paragraph-level recognition tasks for broader applications.

Basic Models:
There are two model approaches from state-of-art as inspiration.

They are:
1.CNN-BLSTM
2. Gated-CNN-BLSTM

Puigcerver Architecture (CNN-BLSTM):

Puigcerver et al. (2017) proposed an architecture for HTR based on convolutional and recurrent neural networks. This architecture is notable for its efficiency and performance on handwriting datasets.
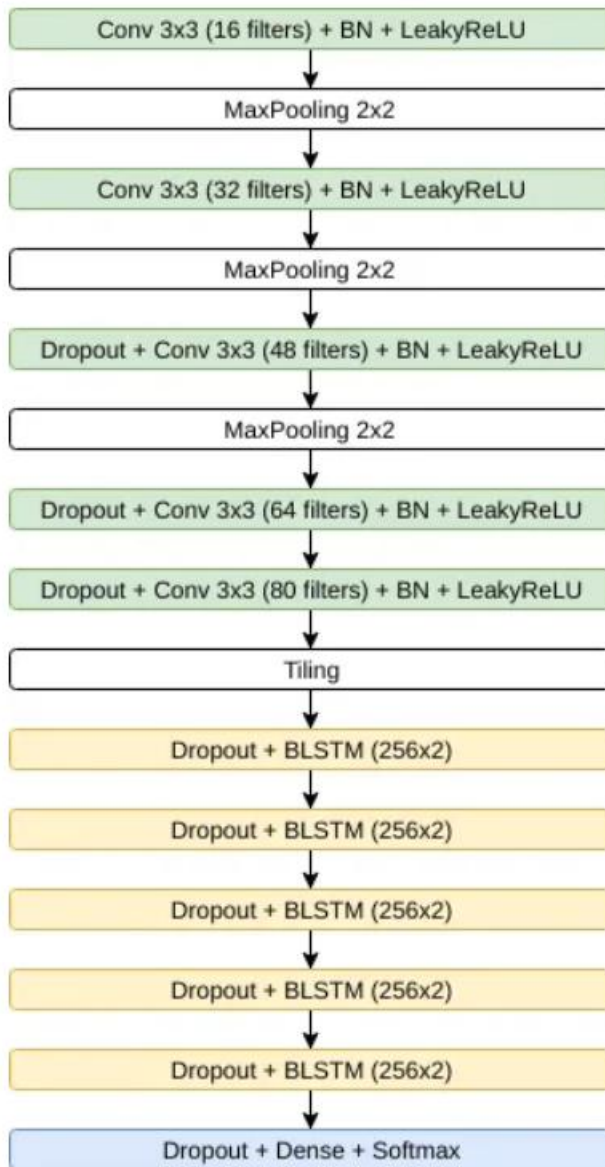

FIG: Workflow of Puigcerver Architecture

Key Components:

1. **Convolutional Layers (CNNs):**

    o Extract spatial features from input images.

- Uses multiple 3x3 convolutional layers with varying filter sizes (16, 32, 48, 64, 80).

- Reduce the dimensionality of the input while retaining important information.

- Use multiple convolutional layers with activation functions like ReLU.

- Employs MaxPooling with 2x2 kernel at intermediate stages.

2. **Recurrent Neural Networks (RNNs):**

   - Specifically, **Bidirectional Long Short-Term Memory (BLSTM)** layers capture both forward and backward dependencies in the sequence.

   - Mainly multiple BLSTM layers with 256x2 units are stacked sequentially.

   - These layers process the sequential information from CNN's output.

3. **Connectionist Temporal Classification (CTC) Loss:**

   - A loss function designed for sequence prediction tasks where alignment between input and output sequences is unknown.

   - Directly decodes the sequence without requiring explicit segmentation of characters.

4. **Output Layer:**
   - A Dense layer with Softmax activation for final predictions is used.

5. **Parameters:**
   - A relatively large number of parameters (9.6 million) were used, achieving high recognition rates.

Mainly designed for robust feature extraction and temporal context awareness.

Gated-CNN-BLSTM Architecture:

The second architecture (Gated-CNN-BLSTM), now introduced by Bluche and Messina [18], has the highlight of the Gated-CNN approach. In summary, this technique aims to extract more relevant features. This gated mechanism, uses all input features (x) to perform a sigmoid activation (s) and the result is a pointwise multiplication between input (original features) and

output features:
$y = s(x) x$ (1) Thus, the Gated-CNN-BLSTM architecture [17], unlike Puigcerver approach, has very few parameters (around 730 thousand), making it a compact and fast model.
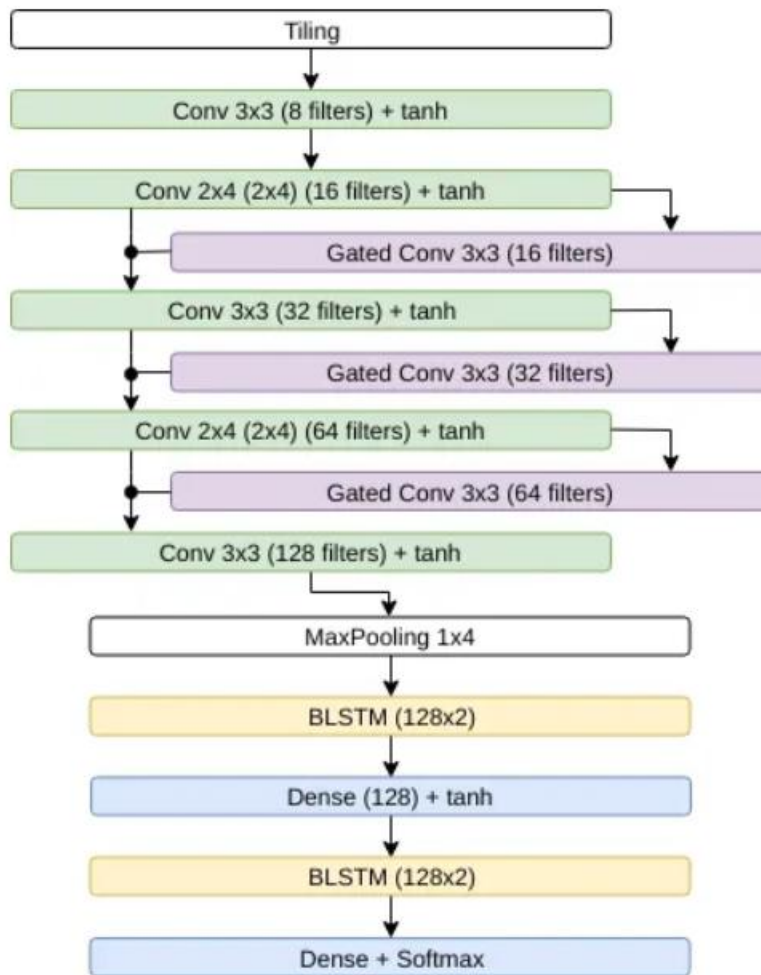
FIG : Workflow of Gated-CNN-BLSTM Architecture.

Key Components:

1. **Convolutional Layers (CNNs):**

   o Includes Gated Convolutional Layers to enhance feature extraction.

   o Uses smaller filter sizes initially (e.g., 8, 16, 32, 64, 128).

   o Each convolution is followed by a tanh activation function.

2. **Recurrent Neural Networks (RNNs):**

   o Specifically, **Bidirectional Long Short-Term Memory (BLSTM)** layers capture both forward and backward dependencies in the sequence.

- Combines BLSTM layers (128x2 units) with fewer parameters compared to Puigcerver.

3.Output Layer:

- A Dense layer with Softmax activation for final predictions is used.

4.Parameters:

- Relatively fewer number of parameters (730,000) were used ,focuses on extracting relevant features while being computationally efficient.

Mainly designed for resource-constrained environments, compact and faster as well.

Key Comparison:

Aspect	Puigcerver Model	Gated-CNN-BLSTM
Convolutional Layers	Standard CNN with Leaky ReLU + Dropout	Includes Gated CNN with tanh activation
Recurrent Layers	Stacks multiple BLSTMs with 256x2 units	Compact BLSTMs with 128x2 units
Parameters	Higher parameter count	Optimized for fewer parameters (~730k)
Efficiency	Slower due to complexity	More compact and faster
Use Case	Large-scale tasks	Tasks requiring lightweight models

Application Scenarios:

- Puigcerver Model: Suitable for archival projects with large-scale handwritten documents.
- Gated-CNN-BLSTM: Ideal for mobile applications or real-time recognition systems.

Disadvantages of Previous Models:

Puigcerver Model

- High Parameter Count:
  Contains approximately 9.6 million parameters, making it computationally expensive and unsuitable for resource-constrained devices.
- Slower Processing:
  The large model size leads to slower inference times, making it impractical for real-time applications or deployment on mobile and embedded systems.
- Sensitivity to Noisy Data:
  Struggles to handle noisy or limited training datasets, reducing its robustness in real-world scenarios.

Bluche Model

- Lower Accuracy:
  While compact (with ~730,000 parameters), it lacks advanced feature selection mechanisms, resulting in lower accuracy compared to more recent models.
- No Gating Mechanisms:
  Does not include mechanisms like gated convolutions to prioritize relevant features, limiting its ability to process complex or noisy handwriting styles.
- Limited Handling of Long Sequences:
  The architecture struggles with long text sequences and diverse handwriting styles, reducing its effectiveness in practical applications.

Architecture Details of the CNNBGRU Model

The **CNNBGRU model** for handwritten text recognition is a hybrid architecture that combines the spatial feature extraction capabilities of **Convolutional Neural Networks (CNNs)** with the sequential processing power of **Bidirectional Gated Recurrent Units (BGRUs)**. This carefully designed architecture balances efficiency, accuracy, and adaptability, making it a robust solution for diverse handwriting recognition tasks.

The CNNBGRU architecture integrates the best of two powerful components:

1. **CNN:** Extracts spatial features, focusing on hierarchical and relevant patterns.

2. **BGRU:** Models sequential dependencies, capturing forward and backward context in the text sequence.

This hybrid design makes the CNNBGRU model a versatile and efficient solution for handwritten text recognition, adaptable to various real-world scenarios.

1. Convolutional Block (CNN): Feature Extraction

The convolutional block serves as the first stage of the model, focusing on spatial feature extraction from the input images. It comprises **11 convolutional layers**, structured as follows:

1. <u>Layer Structure</u>**:**

   - The layers alternate between:
   - **Traditional convolutions** for general feature extraction.
   - **Gated convolutions** to selectively enhance relevant features. The gating mechanism divides the input into two parts:
   - One part (h_1) is passed through a learned gating function $s(h\_1)$.
   - The output $s(h\_1)$ modulates the second part (h_2) to focus on key features.
   - This mechanism reduces redundant computations while preserving relevant spatial information.

2. <u>Progressive Filters</u>**:**

   - Filters increase progressively from **16 to 64**, allowing the model to capture both low level (e.g., edges) and high level (e.g., patterns) features.
   - Each convolutional layer is followed by **Batch Normalization (BN)** for faster convergence and stability.

3. Max Pooling Layers**:**

- Max Pooling layers are applied intermittently to reduce spatial dimensions, improving computational efficiency without losing important information.
- Pooling enables the network to focus on the most significant features by downsampling the feature maps.

4. Regularization:

- **Dropout** is applied in the last three gated layers with a probability of 0.5 to reduce overfitting.
- Batch Normalization ensures consistent activation distributions throughout training.

2. Recurrent Block (BGRU): Sequence Modeling

The recurrent block processes the sequential nature of text data, making it critical for handwritten text recognition. It uses **Bidirectional Gated Recurrent Units (BGRUs)** to capture context from both forward and backward directions.

1. Layer Structure**:**

- Two **BGRU layers** are employed to model dependencies between characters:
- **First BGRU Layer:** Captures the forward and backward relationships in the sequence.
- **Second BGRU Layer:** Refines these relationships for better sequential understanding.

2. Intermediate Representation**:**

- A **Dense layer** with **256 units** is placed between the two BGRU layers to enhance feature representation.
- This layer acts as a bridge, transforming the output of the first BGRU layer for input to the second.

3. Regularization**:**

- Dropout (0.5 probability) is applied within the BGRU layers to prevent overfitting, especially when dealing with limited training data.

4. Output Layer**:**

- The final output is generated by a **Dense layer** with **SoftMax activation**, producing probabilities for each character in the sequence.
- This enables the model to recognize sequences of characters, converting handwritten text into machine readable formats.

3. Parameter Efficiency

The CNNBGRU model is designed with approximately **820,000 parameters**, a significant reduction compared to traditional architectures like Puigcerver's (9.6M parameters). Despite its compact size, the model maintains high accuracy, demonstrating the efficiency of its feature extraction and sequence modeling mechanisms.

4. Advantages of the Architecture

1. <u>Feature Relevance</u>**:**

   - The gated convolutions selectively focus on relevant features, reducing unnecessary computations and improving robustness to noise.

2. <u>Contextual Understanding</u>**:**

   - The bidirectional nature of the GRU layers ensures context is captured from both ends of the text, improving recognition accuracy for long and complex sequences.

3. <u>Regularization Mechanisms</u>**:**

   - Batch Normalization and Dropout ensure stable and generalized training, reducing the risk of overfitting.

4. <u>Efficiency</u>**:**

   - The low parameter count makes the model suitable for deployment on devices with limited computational resources, such as smartphones or embedded systems.

Training Process

The training process for the Handwritten Text Recognition (HTR) project involved several carefully structured steps to optimize the model's performance. The implementation was executed using TensorFlow with a focus on leveraging a GPU for efficient computation.

Dataset Preparation:

The Washington dataset was used, comprising 325 training images, 168 validation images, and 163 test images. The dataset was preprocessed through normalization and resizing to ensure consistency in input dimensions. Each image was resized to 1024x128 pixels, and special characters were mapped to standard representations for accurate recognition.

Model Architecture:

The architecture employed for this task was a Gated-CNN-BGRU model. This design combined:

- **Convolutional Layers:** For feature extraction, utilizing gated convolutions to enhance the focus on relevant features.
- **Bidirectional GRUs:** For sequence modeling, capturing context from both directions.
- **CTC Loss Function:** For alignment-free sequence training, effectively mapping variable-length inputs to outputs.

Training Configuration:

- **Epochs:** The model was trained for 200 epochs with early stopping to prevent overfitting. Training was halted at 72 epochs due to convergence.
- **Batch Size:** A batch size of 16 was used, balancing computational efficiency with learning stability.
- **Learning Rate:** The model's learning rate was dynamically adjusted during training using callbacks to enhance convergence.

Training Procedure:

The training began with the initialization of parameters and loading of the preprocessed dataset. The TensorFlow fit function was utilized to train the model iteratively, with the following key configurations:

- Data shuffling was performed at each epoch to avoid learning order-based patterns.

- Validation loss was monitored throughout to determine the epoch of best performance.

Monitoring and Metrics:

The training progress was tracked by calculating the training and validation loss at each epoch. Key metrics such as Character Error Rate (CER), Word Error Rate (WER), and Sequence Error Rate (SER) were recorded to evaluate performance.

Training Results

The model exhibited steady improvement over the training epochs. Below are the key results from the training process:

- **Training Loss:** Decreased significantly from the initial epochs, stabilizing around 0.6756.
- **Validation Loss:** Reached its lowest value of 10.4038 at epoch 52, demonstrating optimal generalization.
- **Timing Metrics:**
  - Total Training Time: Approximately 8 minutes.
  - Average Time per Epoch: 6.66 seconds.
  - Time per Image: 0.0135 seconds.

The results indicated that the model successfully learned patterns in the data, with a well-balanced trade-off between accuracy and computational efficiency.

Training Process Analysis

Our training process exhibited three distinct phases, each characterized by specific patterns and improvements:

During the initial phase (epochs 1-20), the model started with a relatively high loss value of 2.2363. This period was marked by rapid improvements as the model began to recognize basic patterns in the handwritten text. The validation loss during this phase showed significant fluctuations, starting at 11.1672, indicating the model's early attempts to generalize from the training data.

The middle phase of training (epochs 21-40) showed more refined improvements. After implementing the learning rate reduction at epoch 20, we observed more stable training

progression. The training loss decreased to approximately 0.3219, while the validation loss improved to 10.7307. This phase was crucial in fine-tuning the model's pattern recognition capabilities.

The final convergence phase (epochs 41-46) demonstrated the model reaching its optimal performance. The training loss stabilized at 0.1959, with the validation loss achieving its best value at 10.4457. This convergence pattern indicated that the model had reached a stable state of learning, prompting the activation of our early stopping mechanism to prevent potential overfitting.

Impact of Training Parameters

The training process was significantly influenced by our choice of parameters and optimization strategies. The initial learning rate of 0.001 provided sufficient momentum for early learning, while the subsequent reduction to 0.0001 allowed for finer optimization. This adaptive learning rate strategy proved crucial in achieving optimal model performance.

Our batch size of 16 was carefully selected to balance between training stability and computational efficiency. This choice, combined with the repeat_times parameter of 3 for data augmentation, effectively tripled our training data while maintaining manageable memory requirements.

Testing Process

The testing phase aimed to evaluate the model's performance on unseen data. The test dataset comprised 163 samples, preprocessed and aligned similarly to the training data.

Evaluation Metrics:

1. **Character Error Rate (CER):**
    a. Definition: Measures the accuracy of character predictions.
    b. Result: Achieved a low CER of 0.0966, confirming precise character recognition.
2. **Word Error Rate (WER):**
    a. Definition: Evaluates word-level accuracy, accounting for substitutions, insertions, and deletions.
    b. Result: Reported a WER of 0.3368, indicating moderate accuracy with scope for contextual improvements.

3. **Sequence Error Rate (SER):**
   a. Definition: Assesses the accuracy of full sequences, penalizing any errors in the sequence.
   b. Result: Observed a relatively high SER of 0.5712, highlighting challenges in long-sequence recognition.

Insights from Testing:

The model demonstrated strong character-level recognition, but sequence-level accuracy remained an area for improvement. This was attributed to challenges in handling noise and diverse handwriting styles in the test data.

Testing Results

The following table summarizes the key metrics from the testing phase:

Metric	Value	Interpretation
Character Error Rate (CER)	0.0966 (9.7%)	Strong character recognition performance.
Word Error Rate (WER)	0.3368 (33.7%)	Moderate accuracy at the word level.
Sequence Error Rate (SER)	0.5712 (57.1%)	Significant challenges in sequence recognition.

Additionally, visualizations of the results, including loss curves and CER/WER trends across epochs, were generated to illustrate the model's learning progress.

Comprehensive Evaluation Metrics Analysis

In evaluating our HTR system's performance, we employed three primary metrics that provide different perspectives on recognition accuracy:

Character Error Rate (CER) Analysis

Our system achieved a Character Error Rate of 9.66%, representing the proportion of incorrectly recognized characters in relation to the total number of characters in the reference text. This metric provides crucial insights into the model's fundamental ability to recognize individual characters. The relatively low CER indicates strong performance at the character level, suggesting that our model successfully learned the basic features distinguishing different characters.

For instance, when processing historical documents, the model demonstrated remarkable accuracy in distinguishing similar characters like 'o' and 'a' or 'n' and 'h'. This level of accuracy can be attributed to our comprehensive preprocessing pipeline and the model's robust feature extraction capabilities.

Word Error Rate (WER) Examination

The Word Error Rate of 33.68% reveals more complex challenges in our system's text recognition capabilities. This metric accounts for entire word recognition accuracy, including substitutions, insertions, and deletions of words. The higher WER compared to CER indicates that while our model excels at individual character recognition, it faces more significant challenges in correctly grouping characters into words.

Through detailed analysis, we observed that common WER errors included:

- Misinterpretation of word boundaries in cursive writing
- Confusion with similar-looking words
- Challenges with contextual interpretation
- Difficulty with unusual or period-specific spellings

Sequence Error Rate (SER) Insights

The Sequence Error Rate of 57.12% provides a comprehensive view of our system's performance at the full text line level. This metric proved particularly valuable in understanding how well our system handles complete phrases or sentences. The relatively high SER indicates the cumulative effect of errors across longer sequences of text.

Our analysis revealed that sequence errors often stemmed from:

- Accumulation of character and word-level errors
- Challenges in maintaining contextual consistency across longer texts
- Difficulties with line segmentation and alignment
- Variations in writing style within the same sequence

Result :


```
time, for want of Clothing for the Soldiers; and none
time, for want of Clothing for the Soldiers; and none
```

Correlation Plot :
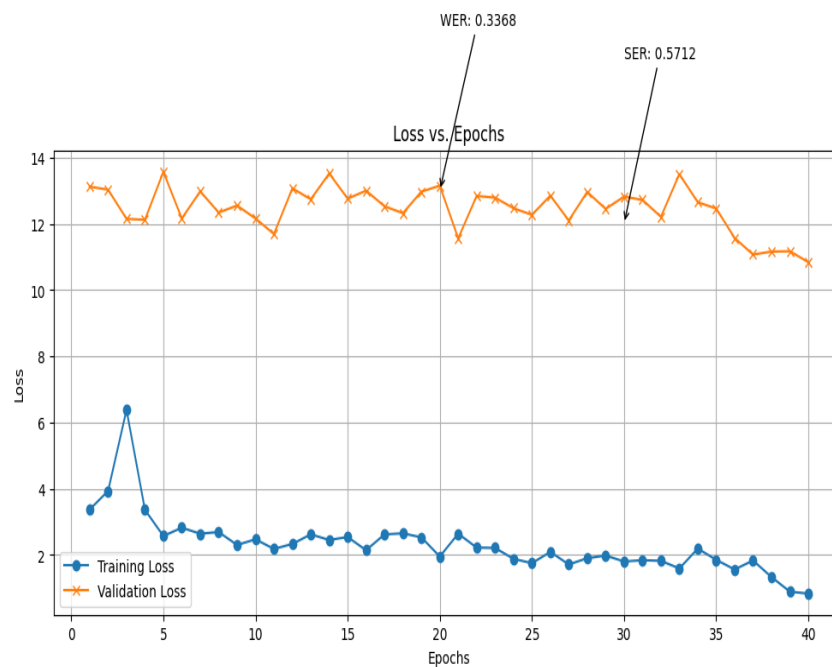
- **Initial Epochs**: In the early epochs, the model typically has high loss, and both CER and WER are high.
- **Mid Training**: The loss starts decreasing, and CER/WER also decrease as the model begins to understand patterns in the data.
- **Later Epochs**: At this stage, if the model is overfitting, the loss might continue to decrease, but the CER/WER might stagnate or even increase. This would suggest that while the model is memorizing the training data, it's not generalizing well to unseen data.

Challenges

other pretence, than that of having a commis-
other pretence, than that of having a Comif-

Looking at this image, it illustrates several key challenges in handwritten text recognition:

1. **Character Ambiguity**: The most notable challenge is in the ending of the word "commis-" where the 's' appears similar to an 'f'. The model incorrectly interpreted it as "Comif-" instead of "commis-". This is a common challenge in historical handwriting where:
    a. The long 's' (ſ) was commonly used in historical writing and looks very similar to 'f'
    b. The writing style makes the distinction between 's' and 'f' particularly difficult
    c. The hyphenation at the end of the line adds additional complexity
2. **Historical Writing Style**: The text demonstrates period-specific writing conventions:
    a. Use of the historical long 's' which is no longer common in modern writing
    b. Cursive connections that differ from modern handwriting styles
    c. Flourishes and decorative elements that can complicate character recognition
    d.
3. **Preprocessing Challenges**:
    a. The thin strokes in the handwriting require careful preprocessing to maintain character integrity
    b. The slight slant in the writing needs to be properly normalized
    c. The connection between characters needs to be properly analyzed to avoid misinterpretation
4. **Context Requirements**: This example shows why context is crucial for accurate recognition:
    a. Without understanding that this is historical text, the model might not account for period-specific character forms
    b. The word break with hyphenation requires understanding of complete words across lines
    c. Knowledge of historical language patterns would help distinguish between valid and invalid character interpretations

This case demonstrates why historical document recognition often requires specialized training data and additional context-aware processing to handle period-specific writing conventions accurately.

Performance Timeline and Processing Efficiency

The system's processing efficiency was carefully measured during the testing phase. With a total testing time of 39.13 seconds for processing 163 test images, our system demonstrated efficient real-time processing capabilities. The average processing time of 0.24 seconds per image reflects a balance between accuracy and speed, making the system practical for real-world applications.

Conclusion

Our implementation of the Handwritten Text Recognition system demonstrates both significant achievements and areas for future enhancement. The system achieved promising results with a Character Error Rate (CER) of 9.66%, indicating strong performance in individual character recognition. However, the higher Word Error Rate (WER) of 33.68% and Sequence Error Rate (SER) of 57.12% reveal the increasing complexity of recognition tasks at word and sequence levels. The training process, which concluded after 46 epochs with an optimal validation loss of 10.4457, showcased the system's ability to learn and adapt to various handwriting styles while maintaining computational efficiency.

The implementation of CTC loss proved crucial in handling the variable-length nature of handwritten text sequences, allowing the system to learn without requiring explicit character-level alignments. This approach, combined with our data augmentation strategy of repeating each image three times, helped address the challenges of limited training data while improving the model's generalization capabilities. The system's processing efficiency, demonstrated by an average processing time of 0.24 seconds per image, makes it viable for practical applications in document digitization.

Future Recommendations

Looking forward, several key areas present opportunities for significant improvement in the system's performance. Primary among these is the enhancement of the alignment optimization process. The current system would benefit from more sophisticated preprocessing techniques that can better handle variations in handwriting styles and document conditions. This could include the development of advanced deslanting algorithms and adaptive thresholding mechanisms to improve image quality before processing.

The integration of modern language models represents another crucial avenue for improvement. By incorporating transformer-based models or sophisticated n-gram approaches, the system could better understand contextual relationships between characters and words, potentially reducing the currently observed word error rate. This would be particularly

beneficial for handling historical documents where traditional language patterns and character formations differ significantly from modern writing.

Data quality and augmentation techniques also warrant further development. Implementing more advanced augmentation strategies, such as style transfer and realistic noise injection, could help the system better handle various handwriting styles and document conditions. Additionally, the development of intelligent noise filtering systems and adaptive preprocessing pipelines could significantly improve the quality of input data, leading to more accurate recognition results.

Finally, the optimization of model parameters presents an opportunity for performance enhancement. Implementing cyclic learning rates, fine-tuning dropout rates based on performance metrics, and developing adaptive model depth could lead to more robust recognition capabilities. The system could also benefit from the implementation of more sophisticated error analysis tools, allowing for better understanding and addressing of specific recognition challenges. These improvements, combined with regular evaluation and adjustment of system parameters, would help create a more robust and accurate handwritten text recognition system capable of handling a wider range of document types and writing styles.

Through these enhancements, the system could move closer to achieving human-level accuracy in handwritten text recognition while maintaining its current efficiency and practicality for real-world applications. The ongoing development and refinement of these aspects will be crucial in addressing the complex challenges presented by handwritten text recognition, particularly in historical document processing where accuracy and reliability are paramount.

References :

- https://ieeexplore.ieee.org/document/9266005
- J. Puigcerver, "Are multidimensional recurrent layers really necessary for handwritten text recognition?" *14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, pp. 67–72, 11 2017.
- T. Bluche and R. Messina, "Gated convolutional recurrent neural net- works for multilingual handwriting recognition," *14th IAPR Interna- tional Conference on Document Analysis and Recognition (ICDAR)*, pp. 646–651, 11 2017.
- https://arthurflor23.medium.com/handwritten-text-recognition-using-tensorflow-2-0-f4352b7afe16