###  **ABSTRACT**

We propose an automated system to detect errors in paper printing, including misalignment, missing or misprinted text, and orientation issues. The system uses a camera mounted above a conveyor belt to capture real-time images, processes them with a combination of text detection methods and a Convolutional Neural Network (CNN), and flags defective prints for rejection. Text detection integrates Maximally Stable Extremal Regions (MSER) and contour detection to localize text regions, while a CNN classifies and validates the text. Advanced preprocessing handles challenges like low-resolution images and complex backgrounds. Preliminary tests show high accuracy, making the system ideal for seamless quality control.

### **I. INTRODUCTION**

Ensuring flawless paper printing is critical for readability and customer satisfaction. Our system automates defect detection, tackling common issues such as text misalignment, smudges, and missing words. A camera mounted above the conveyor belt captures images of printed papers, which are analysed in real-time using deep learning and computer vision techniques. The solution combines cutting-edge detection and recognition.

###

### **Advanced Methodology for Printing Defect Detection**

#### **1. Image Acquisition**

**Setup:** Dual 4K Cameras: Capture high-resolution images of text and layout simultaneously. Lighting: Adjustable LED lighting with anti-reflective filters ensures consistent quality. Synchronization: High-speed cameras synchronized with conveyor motion prevent blur.

**Software**: Tools like Basler Pylon or FLIR Spinnaker SDK enable low-latency image capture in real-time.

 **2. Preprocessing**

**Objective: Standardize inputs and enhance defect visibility**.

**Steps**:

Normalization: Resize images to a fixed resolution (e.g., 1280x1280).

Noise Reduction: Apply advanced denoising filters for clearer text.

Contrast Enhancement: Use CLAHE to adjust lighting inconsistencies. Dynamic ROI Extraction: Automatically crop text regions based on templates. Software: OpenCV 5.0 for optimized image processing.

###

### **3. Text Detection and Region Proposal Objective**: Pinpoint and isolate text regions. Techniques: Transformer-Based Detection: Use DETR or DB Net++ for arbitrary-shaped text. Multi-Scale Features: Leverage Feature Pyramid Networks (FPN) to detect varying text sizes. Anchor-Free Models: Employ YOLOv8 for efficient region proposals. Software: PyTorch 2.1 for flexible and high-performance implementations.

### **4. Error Classification and Recognition**

**Objective**: Detect and classify errors like misalignment, missing text, or smudges.

**Model Architecture**:

1.Backbone: YOLOv8 with Coordinate Attention for precise object detection. 2. Contextual Analysis: Add Vision Transformer (ViT) to handle complex layouts or overlaps. Output: Real-time classification of defects for quality control

### **Software**

**NVIDIA Tensor RT**: Speeds up inference. **Weights & Biases (W&B)**: Tracks experiments and optimizes hyperparameters.

**5. Orientation and Layout Analysis**

**Objective**: Validate alignment and layout accuracy. Techniques: Pose Estimation: Use Hough Transform for detecting angles. Template Matching: Compare layouts using Structural Similarity Index (SSI). Alignment Validation: Analyse horizontal and vertical text alignment with Projection Histogram

### **6. Defect Decision and Rejection**

**Objective**: Automate defect handling.

**Process**: Assign a Defect ConfidenceScore based on detection confidence and defect severity. Minor defects are flagged for review; major defects trigger automatic rejection via actuated robotic arms or pneumatic pushers. Hardware: Controlled by Arduino Mega or Raspberry Pi.

###

### **7. Real-Time Integration**

**Objective**: Enable seamless operations.

**Techniques**: **Edge AI Processing**: Deploy models on Jetson Orin for low-latency inference. **IoT Monitoring**: Real-time defect tracking via MQTT protocol

**8. Evaluation Metrics**

**Accuracy**: **Mean Average Precision (mAP)** at IoU thresholds of 0.5 and 0.75. **Precision, Recall, and F1-Score** for individual defects.

**Performance**: **Processing Speed**: ≥60 FPS for real-time operation. **Rejection Accuracy**: ≥99% accuracy in detecting defective prints.

### **Unique Contributions**

**Advanced Models**: YOLOv8 + ViT for cutting-edge object detection and defect analysis.

**Edge Computing**: On-device inference with NVIDIA Jetson for minimal latency. **Dynamic Templates**: Automatically updates templates for different paper layouts. **Active Feedback**: Continuous model improvement through human-in-the-loop learning.

**Techniques Flowchart:**

****

###

### **IV. TEXT DETECTION**

Text detection is done in two stages: **1) General Text Detection** **2) Word-Level Detection and Refinement**. We combine **DETR** (Detection Transformer) with morphological operations for reliable and scalable detection across various printing scenarios.

 **General Text Detection**

We use **DETR**, a transformer-based model, to identify text regions in high-resolution images. DETR performs well in complex environments, detecting text without predefined anchors.

**1. Process**: The image is processed by DETR, which predicts bounding boxes for text regions based on global features.

**2. Advantages**: Handles **irregularly shaped text**, font variations, and complex backgrounds.

 **3.**  **Challenges & Solutions**: DETR may detect non-text elements in noisy images. We use **Otsu Thresholding** and **contour filtering** to reduce noise.

#### **Word-Level Detection and Refinement**

**I**. Feature Extraction with YOLOv8: Detected regions are passed to **YOLOv8**, which accurately locates and identifies words in real-time.

**II. Refinement**:

 **Binary Thresholding**: Converts regions to black and white. **Morphological Dilation**: Fills gaps to join characters within words.

**III. Advanced Filtering**: **Projection Histograms**: Analyze text alignment and grouping, ensuring words are correctly isolated and merging fragmented detections.

###

### **Comparison of DETR and YOLOv8**

**DETR** is used for detecting large, irregularly shaped text regions, while **YOLOv8** excels at fast and accurate word-level detection. Combining both ensures scalability and robustness in real-world printing environments.

### **Advantages of the Proposed Approach Multi-Scale Detection**: **DETR** handles cluttered, noisy environments, while **YOLOv8** refines individual words for better processing. **Noise Reduction**: Morphological operations and histogram alignment help minimize errors and false positives. **Context-Aware Analysis**: Combining **DETR** and **YOLOv8** improves handling of text variations compared to traditional methods. **Industrial Applicability**: The system ensures real-time detection, crucial for high-speed printing.

### **Pipeline Overview** Input Image **DETR** (Text Region Detection)

### Noise Reduction (Otsu + Contour Filtering) **YOLOv8** (Word Detection) Morphological Refinement (Thresholding + Dilation) Text and Word Isolation for Error Analysis This streamlined pipeline offers robust text detection

**A. Text Detection Results**

The text detection system was evaluated using metrics like precision, recall, and mean Average Precision (mAP), tailored to real-world industrial scenarios.

### **Evaluation Metrics**

**I. True Positives (TP)**:

A detected region is a true positive if it overlaps the ground truth by at least 50% (IoU). Partial overlaps are scored at 0.8.

**II. False Positives (FP)**:

Detected regions that don’t overlap with the ground truth are false positives.

**III. False Negatives (FN)**: Ground truth regions without a corresponding detection are false negatives.

**Partial Matches**: Regions with less than 50% IoU are weighted based on their overlap.

### **Dataset and Testing Environment**

**I) Dataset**:

 **Custom datasets** from real-world printing defects, and the **ICDAR 2013** dataset for standard comparison.  **Defects**: Smudged text, missing characters, misalignment, etc.

I**I) Environment**:

 **Hardware**: NVIDIA Jetson AGX Orin for fast inference. **Software**: PyTorch 2.1, TensorRT optimization for speed.

###

### **Evaluation Methodology**

**I) Ground Truth Preparation**: Text regions annotated with class labels (normal, misaligned, etc.). **Scoring Overlaps**: **IoU - based evaluation** for text detection. **Character Ratio** to calculate accuracy in character detection. **Error Class Handling**: Smudges, misalignment, and other defects are handled separately for accurate evaluation.

**Additionally**, the evaluation accounts for defect-specific categories like misaligned or smudged text, making the results more comprehensive for industrial applications. This methodology ensures robust and precise evaluation of the text detection module, setting a strong foundation for error analysis and defect classification in paper printing workflows.

### **V. CHARACTER SEGMENTATION**

After detecting words, the next step is segmenting them into individual characters, which is essential for identifying defects like missing characters or misalignment. The process combines binarization, morphological operations, and contour extraction for accurate segmentation.

**Character Segmentation Process**

**I. Dynamic Binarization**: **Adaptive Thresholding**: Adjusts thresholds based on local pixel intensity for better performance in varied lighting. **Gradient-Aware Thresholding**: Preserves edges for clear character differentiation.

**Morphological Refinement**: **Closing Operation**: Fills gaps in characters (e.g., "i" and "j") and addresses disconnected regions due to noise. **Opening Operation**: Removes small noise while preserving character structure.

**Contour Extraction and Validation**:

**Connected Component Analysis**: Identifies character boundaries based on pixel connectivity. **Aspect Ratio and Area Filtering**: Filters out non-character elements. **Edge Smoothing**: Reduces jagged edges from binarization.

**Overlapping Character Handling**:

**I) Projection Histograms**: Detects overlaps and guides segmentation using local minima.

**II) Watershed Segmentation**: Splits connected characters due to smudging or tight spacing.

###

### **Evaluation of Character Segmentation**

The segmentation model was tested on:

1. **Custom Dataset**: Captured from paper printing with added defects.
2. **ICDAR 2013 Recognition Dataset**: Standard dataset for text recognition.
3. **Oxford Synthetic Word Dataset**: Simulated dataset for character-level recognition.



 **Challenges and Observations**

I. Low-Resolution Images:

 Characters in low-resolution images or with heavy noise showed segmentation inaccuracies. To mitigate this, the method integrates gradient-aware preprocessing.

II. Overlapping Characters:

While watershed segmentation improved handling of merged characters, minor inaccuracies were observed in highly smudged text regions.

III. Complex Backgrounds:

Adaptive thresholding provided better performance under non-uniform backgrounds compared to static methods like Otsu thresholding.

**Examples of Successful and Failed Segmentation**

**I. Successful Cases**: Smudged text with moderate contrast was successfully segmented. Words with consistent spacing between characters were accurately isolated.

**II. Failed Cases**: Overlapping characters caused partial segmentation. Noisy images with background elements mimicking text features led to incorrect segmentation.

### **Improvements Over Traditional Methods**



###

### **Conclusion**

The proposed method significantly boosts segmentation accuracy by using dynamic thresholding, contour-based filtering, and advanced techniques like watershed segmentation and projection histograms. These innovations ensure reliable performance, even in challenging conditions like noisy images and smudged text. This methodology provides a solid foundation for accurate character.

 **VI. TEXT RECOGNITION**

Once the text is detected and segmented, it is passed through the text recognition module. This module classifies each character and reconstructs words by combining the recognized characters. The system uses a CNN-based architecture for character recognition, trained on a custom dataset designed to handle variations in fonts, colors, and backgrounds typical in paper printing.

###

### **Dataset Creation**

A custom dataset was created to train the recognition model, accounting for the diverse conditions in industrial paper printing.

**I. Fonts and Styles**:

Over **1500 fonts** from sources like Google Fonts and Adobe Fonts to cover the variety of typefaces. Fonts were scaled across multiple sizes to ensure adaptability to different character dimensions.

**II. Backgrounds and Noise**:

Real-world printing imperfections like **smudges**, **faded prints**, and **background artifacts** were simulated using preprocessing filters and synthetic overlays. Background textures from printing press images were used and augmented with random distortions.

**1.Dataset Specifications**:

**Classes**: 62 (A-Z, a-z, 0-9).

**Image Size**: 64x64 pixels per character image. **Samples**: ~350,000 images, evenly distributed across classes. **Augmentations**: Random rotations (-10° to +10°), varying contrast and brightness, and Gaussian blur to simulate poor quality prints.

### **CNN Architecture for Character Recognition**

The CNN is built to balance complexity and performance with modern optimizations.

**I. Model Structure**:

**Convolutional Layers**: Four layers (3x3 kernels, ReLU activation) to extract features.

**Pooling**: Three max-pooling layers (2x2 pool size) for dimensionality reduction while preserving important features.

**Dropout Layers**: Two dropout layers (rates 0.25 and 0.5) to prevent overfitting.

**Fully Connected Layers**: One dense layer (512 neurons, ReLU) for feature consolidation and an output layer (62 neurons, Softmax activation) for classification.

**II. Batch Normalization**: Applied after each convolution to stabilize training and improve model generalization.

**1.Training Details**:

**Optimizer**: Adam with a learning rate of 0.001 and decay of 1e-6. **Loss Function**: Sparse Categorical Cross-Entropy.  **Epochs**: 100. **Batch Size**: 256. **Hardware**: Trained on an **NVIDIA RTX 3090 GPU** for faster processing.

 **Text Reconstruction**

**1. Character Assembly**:
After recognizing individual characters, the system arranges them based on their positions and order in the image, ensuring accurate word reconstruction using bounding box coordinates.

**2. Error Handling**:
Misclassified characters are corrected by a language model trained on paper text, using common word patterns to guide error correction.

### **Evaluation**

**I. Character Recognition Metrics**:

**Precision**: Percentage of correctly classified characters out of all predicted ones.

**Recall**: Percentage of correctly classified characters out of all ground truth characters.

**F1-Score**: Harmonic mean of precision and recall.

**Results (Chars74K Dataset)**:

****

**Word Recognition Metrics**:
Evaluated on accurately segmented words from:

**ICDAR 2013**

**Custom Synthetic Dataset
Metric**: Word recognition accuracy based on correct word reconstruction.

**Results**: **ICDAR 2013**: 78.2% **Custom Dataset**: 82.7%

###

### **Challenges and Observations**

### **Complex Fonts and Noise**:Highly stylized fonts or smudged text led to misclassifications.**Solution**: Enhanced data augmentation and training with noisy samples improved robustness. **a. Character Overlap**:Poor printing led to merged or overlapping characters.**Solution**: **Projection histogram refinement** reduced this issue after segmentation. **b. Rare Characters**:Infrequent characters (e.g., special symbols) had lower precision due to insufficient samples.**Solution**: Oversampling rare classes during training improved recognition.

###

### **Examples of Successful and Unsuccessful Recognition**

**Successful Cases**:
Cleanly printed, well-segmented text with moderate noise or misalignment were handled effectively.

**Unsuccessful Cases**:
Severe smudging or faded text caused

misclassifications, as well as non-standard fonts with complex styles

###

### **VII. CONCLUSIONS**

The proposed system automates the detection and recognition of printing defects in newspapers, such as misalignment, missing characters, and smudges. By combining advanced machine vision and deep learning, it addresses challenges posed by various fonts, layouts, and noisy conditions. The **text recognition module** was the most effective, accurately classifying characters and reconstructing words. **DETR** for text detection and **YOLOv8 with Coordinate Attention** for word refinement performed well with different text orientations and backgrounds. **Morphological operations** and **projection histograms** improved segmentation, especially for overlapping characters.

**However, there were limitations:**

**I.Text Detection**: False positives in noisy images and struggles with non-horizontal or curved text.

 **II. Character Segmentation**: Issues with overlapping characters and low-resolution images.

**III. Recognition**: Decreased accuracy with highly stylized or smudged text.

**Future work should on:**

a. Enhancing detection with **transformer-based techniques** for varied text layouts.

b. Using **super-resolution** for better segmentation in low-res images.

c. Incorporating **language models** for better word reconstruction and faster **inference**.

###

### **REFERENCES**

Jaderberg, M., et al. (2016). "Reading text in the wild with convolutional neural networks." *Int. J. Comput. Vision*, 116(1), 1-20.

Zhu, Y., et al. (2016). "Scene text detection and recognition: Recent advances and future trends." *Frontiers of Computer Science*, 10(1), 19-36.

Karatzas, D., et al. (2013). "ICDAR 2013 robust reading competition." *12th Int. Conf. on Document Analysis and Recognition*, 1484–1493.

Neumann, L., & Matas, J. (2012). "Real-time scene text localization and recognition." *IEEE CVPR*, 3538–3545.

Hou, Q., et al. (2021). "Coordinate Attention for Efficient Mobile Network Design." *IEEE CVPR*, 13708–13717.