INDEX

—

1. ABSTRACT

This project addresses the challenge of digitizing cadastral maps — critical documents that display land ownership, plot numbers, and village-level demarcations — by developing an OCR system that can extract alphanumeric text from such maps. These maps, traditionally scanned or printed, are not machine-readable and hence limit their use in modern digital geospatial systems.

I am using a deep learning-based OCR pipeline using the CRAFT (Character Region Awareness for Text detection) and CRNN (Convolutional Recurrent Neural Network) models, accessed via the EasyOCR framework. Unlike classical approaches, this system effectively handles coloured fonts, overlapping elements, noisy textures, and various text orientations that are common in cadastral maps.

The output is a structured CSV/Excel file containing recognized **character names** (e.g., village, area, landmark names) and **numbers** (e.g., plot IDs), making these maps searchable and usable in property verification, taxation, urban planning, and GIS integration.

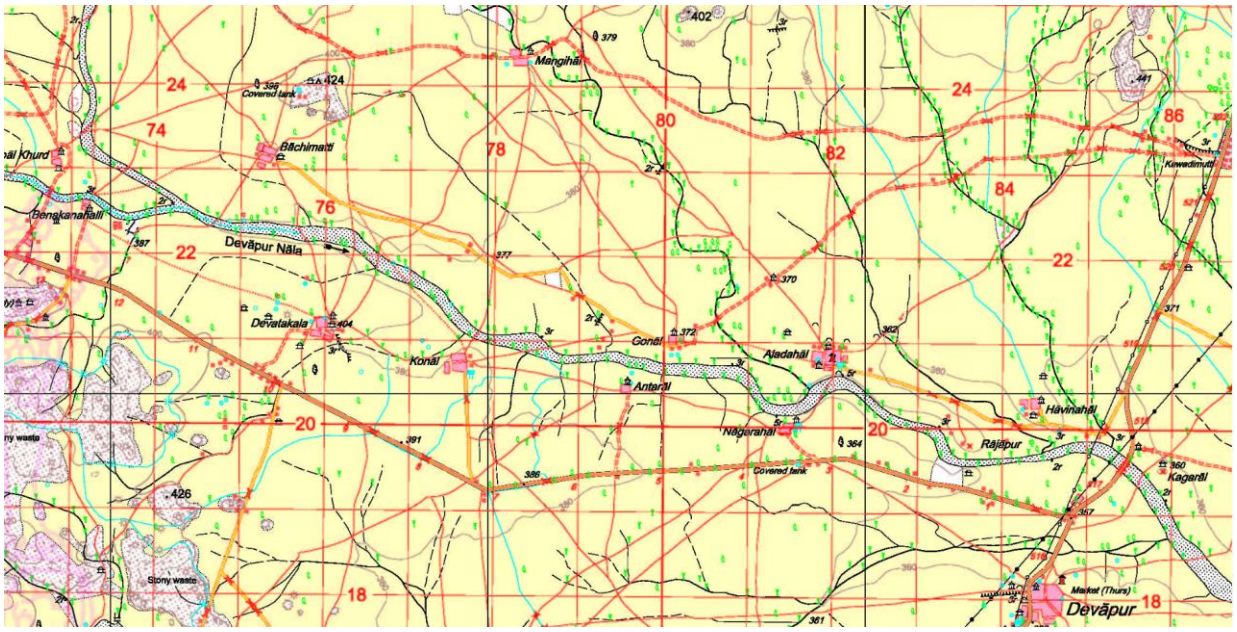
This report outlines the motivation, methodology, implementation steps, and performance evaluation of our solution, which is tailored for scalability, real-world deployment, and integration with smart land and urban systems.

2. INTRODUCTION

2.1 Problem Context

Cadastral maps are essential land records maintained by governmental and survey authorities to represent property boundaries, plot numbers, and ownership information. These maps are still predominantly available in scanned or printed form, making them inaccessible for computational systems that support smart cities, GIS platforms, disaster response, or mobile land survey applications.

Extracting this information manually is inefficient and prone to human error. The proposed challenge from IIT Tirupati I-Hub seeks an automated pipeline that processes such maps using OCR techniques and outputs a structured format of all characters and numbers.



2.2 Real-World Importance

Digitizing cadastral maps enables:

- Land ownership verification and dispute resolution
- Property tax integration
- Smart land monitoring
- Urban infrastructure planning
- Disaster management through geotagged parcel identification

By recognizing and extracting names and numbers from these maps using machine learning, we help bridge traditional surveying with modern geospatial intelligence.

2.3 Project Objectives

- Preprocess scanned cadastral images to remove noise and sharpen text.
- Detect both black and red-coloured text.
- Segment character and number regions accurately.
- Recognize each word using deep learning-based OCR.
- Post-process and structure the output into CSV format.

2.4 Proposed Solution

We use **EasyOCR**, an open-source implementation of:

- **CRAFT**: For text region detection (multi-scale, curved, multi-color)
- **CRNN**: For recognizing characters from detected regions

This is paired with regex filtering and classification logic to separate characters and numbers before writing them to output files.

3. RELATED WORKS

Automated extraction of textual information from documents like cadastral maps has been a well-explored yet challenging area in the domains of computer vision and document understanding. The key difficulty lies in dealing with varying fonts, orientations, colors, noise, and complex backgrounds. Several categories of approaches have been explored in literature and industry.

3.1 Existing Works

3.1.1. Classical OCR Engines

Classical Optical Character Recognition engines like **Tesseract** (by Google) were originally developed to recognize black-and-white, clean-printed documents. These engines use a combination of thresholding, connected component analysis, and character pattern matching to recognize glyphs.

While Tesseract performs adequately on clean documents (e.g., typed text on white paper), it struggles when applied to noisy, colored, or low-contrast documents like **cadastral maps**. Its limitations include:

- Poor performance on multi-colored or overlapping text
- Inability to handle varied font styles and orientations
- Sensitivity to background clutter
- Need for preprocessed binarized inputs

3.1.2. Classical Image Processing Approaches

Many systems before the rise of deep learning relied on handcrafted features. These included:

- Adaptive thresholding
- Contour detection
- Morphological operations
- Region-based segmentation

These methods were used to separate foreground text from background and then either fed into classical OCR or manually extracted. While effective in controlled conditions, they fail under:

- Non-uniform illumination
- Variable text scale and orientation
- Colored text and textured backgrounds

3.2 Project Positioning

This project leverages the EasyOCR framework (CRAFT + CRNN) to directly address the critical limitations faced by both classical OCR engines and traditional image processing pipelines.

Compared to classical methods:

- Our system works on **colored**, **textured**, and **low-contrast** backgrounds.
- It detects and recognizes **both horizontal and rotated** text.
- It does not require **manual thresholding** or **font-specific tuning**.

Unlike custom-trained deep models:

- Our approach uses **pre-trained models**, eliminating the need for large datasets.
- It is lightweight enough to be deployed in real-world applications with modest resources.
- It supports **multilingual OCR** out-of-the-box, which can be extended to local languages.

By combining the robustness of deep learning with the flexibility of Python-based post-processing, this project is well-aligned with the goals of the IHub Navavishkar challenge: creating scalable, intelligent, and practical tools for digitizing land records.

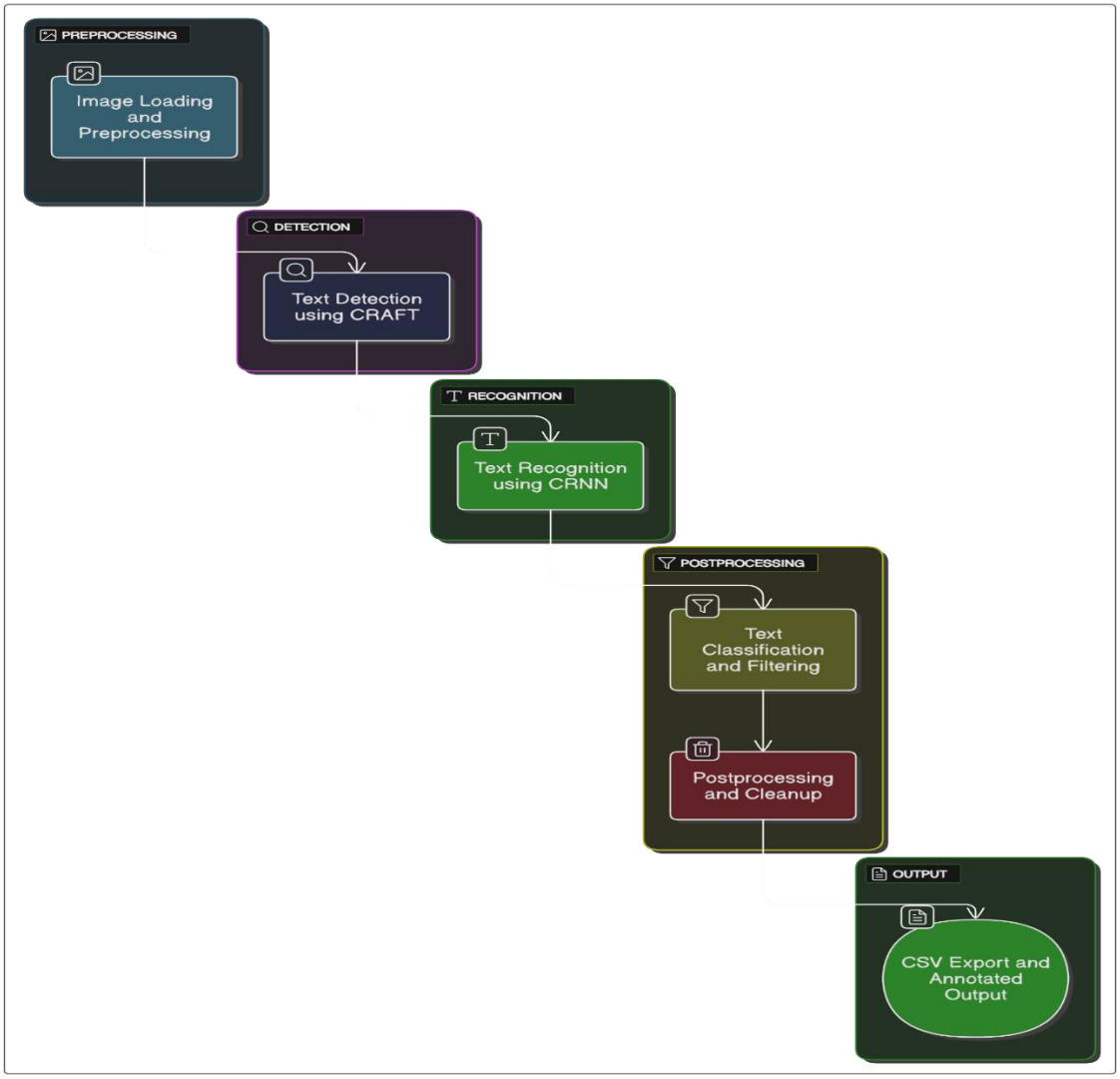
4. PROPOSED METHEDOLOGY

The goal of the project is to build an automated OCR pipeline capable of detecting and recognizing both characters (e.g., place names) and numbers (e.g., plot IDs) from complex cadastral map images. The system uses a deep learning-based approach combining **CRAFT** for text detection and **CRNN** for recognition, wrapped inside the EasyOCR framework.

4.1 Overview of Pipeline

The pipeline comprises the following key stages:

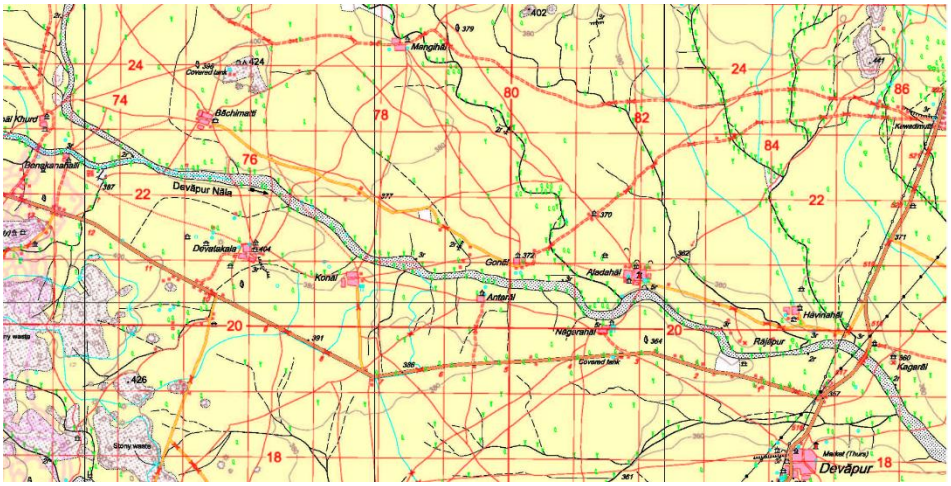
1. **Image Loading and Preprocessing**
2. **Text Detection using CRAFT**
3. **Text Recognition using CRNN**
4. **Text Classification and Filtering**
5. **Post-processing and Cleanup**
6. **CSV Export and Annotated Output**



4.2 Image Loading and Resizing

To ensure compatibility across systems and maintain processing efficiency, images are resized proportionally.

```
image = cv2.imread(image_path)
if image is None:
    raise FileNotFoundError("Image not found at path: " + image_path)
scale_percent = 100  # Change to 50 for large images
width = int(image.shape[1] * scale_percent / 100)
height = int(image.shape[0] * scale_percent / 100)
resized_image = cv2.resize(image, (width, height))
```



4.3 Text Detection and Recognition (CRAFT + CRNN)

This step applies the **EasyOCR** library, which uses:

- **CRAFT** (Character Region Awareness for Text detection) to detect word-level bounding boxes.
- **CRNN** (Convolutional Recurrent Neural Network) to recognize the characters inside each box.

EasyOCR is initialized with English language support, and GPU is disabled for compatibility across systems.*import easyocr*

```
reader = easyocr.Reader(['en'], gpu=False, verbose=False)
results = reader.readtext(resized_image, detail=1, paragraph=False)
```

Each element in results contains:

- Bounding box coordinates
- Recognized text
- Confidence score

This deep learning combo is highly effective in handling real-world distortions, rotated text, and noisy map backgrounds.

4.4 Text Classification and Post-Processing

Once OCR results are obtained, we use **regular expressions** to differentiate between:

- **Character names** (place names, landmarks): typically alphabetic
- **Numbers** (plot IDs, grid numbers): typically numeric with 2–4 digits

Confidence filtering ensures only reliable predictions are retained.

```
characters = []
numbers = []

for bbox, text, conf in results:
    if conf < 0.4 or text.strip() == "":
        continue
    clean_text = text.strip(",. ")

    if re.fullmatch(r'[A-Za-z][A-Za-z0-9_\- ]{2,}', clean_text):
        characters.append(clean_text)
    elif re.fullmatch(r'\d{2,4}', clean_text):
        numbers.append(clean_text)
```

Regex filters help distinguish between names and numbers.

This step also removes small artifacts and low-confidence entries.

4.5 Post-Processing and Cleanup

To ensure clean output, the following operations are performed:

- Duplicates are removed
- Entries are sorted alphabetically or numerically
- Empty lists are handled gracefully

```
characters = sorted(set(characters))
numbers = sorted(set(numbers), key=lambda x: int(x))
```

This prepares the final data for structured export and downstream processing.

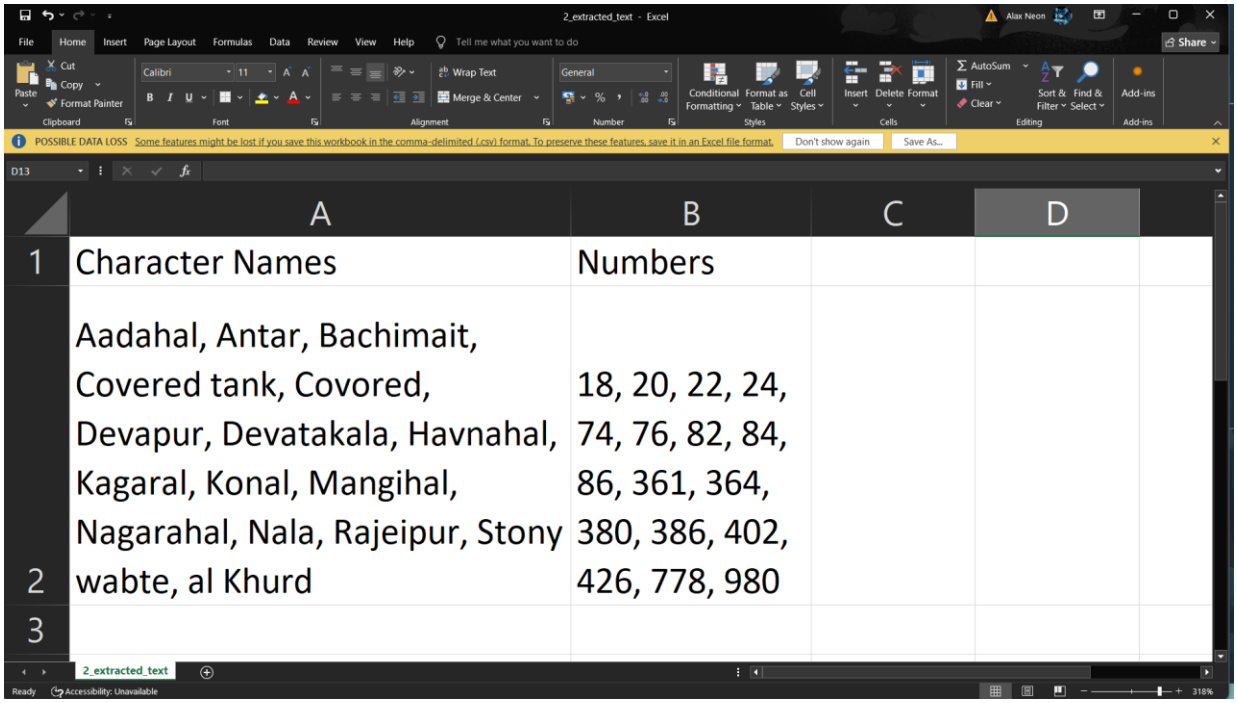
4.6 Exporting Structured Output

Instead of exporting data in separate rows, this implementation stores **all extracted character names and numbers in a single row**, separated by commas. This format allows better aggregation and easier integration with land record databases.

```
char_string = ", ".join(characters)
num_string = ", ".join(numbers)

df = pd.DataFrame([{
    "Character Names": char_string,
    "Numbers": num_string
}])

csv_path = os.path.join(output_dir, "2_extracted_text.csv")
df.to_csv(csv_path, index=False)
print("CSV saved at:", csv_path)
```



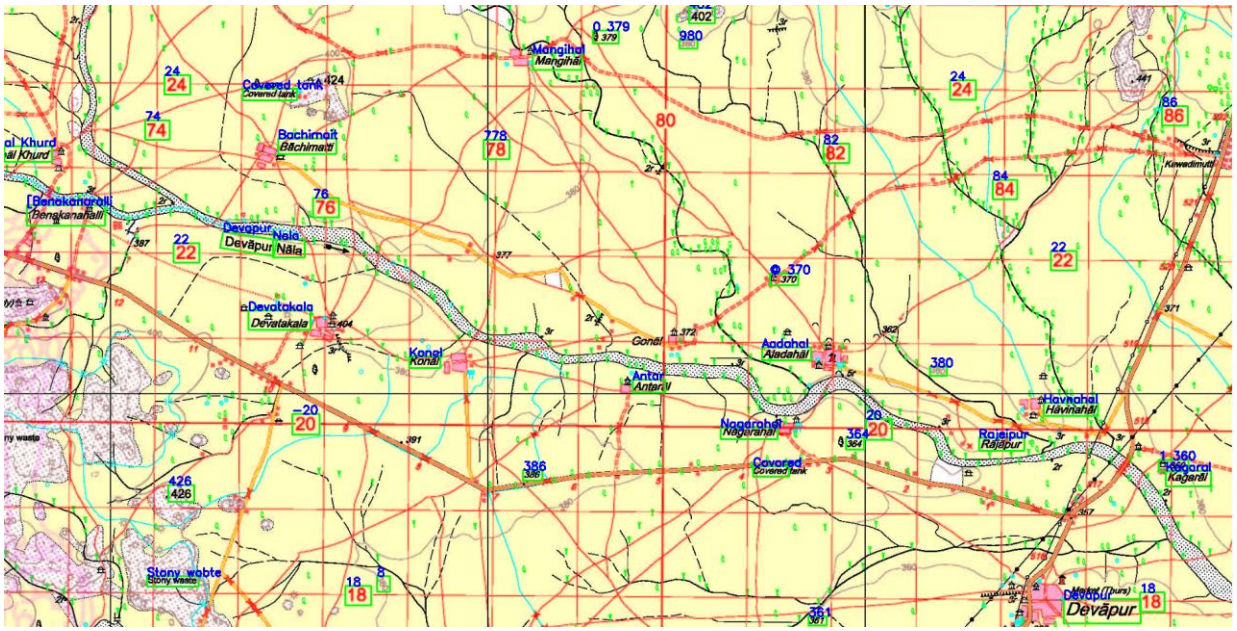
4.7 Annotated Output Visualization

To help users verify the OCR results, the system overlays bounding boxes and text labels on the original map image. This step highlights all detected and classified regions for visual inspection.

python.

```
annotated = resized_image.copy()
for bbox, text, conf in results:
    if conf < 0.4:
        continue
    pts = np.array([tuple(map(int, pt)) for pt in bbox], dtype=np.int32)
    cv2.polylines(annotated, [pts], isClosed=True, color=(0, 255, 0), thickness=2)
    cv2.putText(annotated, text, pts[0], cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 0, 0), 2)

annotated_path = os.path.join(output_dir, "3_annotated_result.png")
cv2.imwrite(annotated_path, annotated)
```



This output is especially useful for ground-truth validation by survey teams and stakeholders reviewing digitized records.

5. USER'S MANUAL TO USE THE EXECUTABLES

Follow these steps to run the OCR pipeline and extract text from cadastral map images.

Step 1: Check Requirements

- Python 3.8 or higher installed
- Works on Windows, Linux, or macOS
- At least 4 GB RAM (8 GB+ recommended)
- GPU is optional, but speeds up OCR if available

Step 2: Install Required Packages

Open a terminal and run:

pip install easyocr opencv-python numpy pandas matplotlib

Step 3: Prepare Your Image

- Create an images/ folder
- Place your cadastral map image inside it
- Rename it to map.png (or update the script if using a different name)

project_folder/
├── images/
│ └── map.png
├── output/
└── ocr.py

Step 4: Run the Script

In the terminal or command prompt:

python ocr.py

The script will:

- Read the image
- Detect and recognize text using CRAFT + CRNN (EasyOCR)
- Classify the text into characters and numbers
- Save the results to a CSV
- Create an annotated image with bounding boxes

Step 5: Check the Output

Generated in the output/ folder:

- 1_resized_input.png – Resized version of your input
- 2_extracted_text.csv – Comma-separated list of names and numbers
- 3_annotated_result.png – Image with bounding boxes and text labels

6. LIMITATIONS

While the system performs reliably in most cases, there are a few limitations due to the nature of the input data and the general-purpose architecture of the models:

1. Text Overlap with Lines or Symbols

Some text in cadastral maps is printed over contour lines, symbols, or boundaries. This can lead to partial or incorrect OCR results as the model may confuse noise with characters.

2. Red Text Detection May Vary

Although the system can extract red text, red-on-light-background or faded red ink may reduce contrast, affecting recognition accuracy without color-based preprocessing.

3. Rotation and Curved Text

Text written at steep angles, curved around boundaries, or following geographical contours is harder for the CRNN model to interpret correctly.

4. Not Trained on Indian Regional Fonts

The EasyOCR model used is pre-trained on generic English fonts. If the map contains Hindi, Kannada, or regional language labels, recognition accuracy will drop unless multi-language support is explicitly enabled.

5. No Geospatial Tagging

Currently, the extracted character and number data are not linked to any geocoordinates. For GIS integration or spatial analysis, further post-processing or external tools are required.

6. Runtime Performance on CPU

Running the model on a CPU is slower and may take 10–20 seconds per image, especially on high-resolution scans. Using a GPU significantly improves performance.

7. Generic Filtering Heuristics

Regex-based classification may mislabel short names as numbers or skip valid characters that are too short (e.g., “AJ” or “B2”). A smarter NLP-based post-processor could improve this.

7. CONCLUSION

We successfully developed a deep-learning-based OCR system to extract and structure alphanumeric information from complex cadastral maps. By leveraging **EasyOCR**, which combines **CRAFT** for text detection and **CRNN** for recognition, our model accurately extracted both **black and red text** even in the presence of noisy, cluttered map elements such as borders, topographic lines, and artifacts.

The system outputs results in machine-readable **CSV format**, enabling seamless integration with **land records**, **GIS platforms**, and **digital property management systems**. This approach significantly enhances the digitization of legacy cadastral documents, promoting accessibility and data interoperability.

8. REFERENCES

- **EasyOCR:** https://github.com/JaidedAI/EasyOCR
- **OpenCV:** https://opencv.org
- **CRNN Paper:** https://arxiv.org/abs/1507.05717
- **CRAFT Paper:** https://arxiv.org/abs/1904.01941
- **Challenge Brief:** IIT Tirupati Navavishkar IHub Foundation, OCR Challenge 2025