Indic Script Style Adaption and Augmentation (Font Style Transfer)

Rahul Yadav

Dr. Kalidas Yeturu IIT TIRUPATI

Abstract. Optical character recognition for Devanagari becomes difficult when only a small annotated corpus is available. A recognizer trained on a narrow visual domain may learn the appearance of a particular scan, font, pen thickness, or background instead of learning stable character structure. This paper documents a practical low-resource OCR pipeline implemented in a Google Colab notebook. The method uses only the available project dataset stored in Google Drive. The dataset contains page images and JSON annotations from two folder pairs: `devnagriannaotation` with `devnagri.png`, and `shivannotation` with `shiv.png`. The notebook first extracts all valid text crops from the annotations and stores them in a crop list named `results`. The model-search section reuses this existing list, saves every valid crop, creates synthetic style variants from those crops only, and trains several CRNN-CTC OCR models. The generated visual styles include stroke thickening, stroke thinning, colored ink with textured backgrounds, font rendering, handwriting-like warping, and document degradation. The notebook also contains optional CycleGAN and diffusion image-to-image cells. CycleGAN uses the project-generated style variants as its target domain, while diffusion starts from the project crops and uses a pretrained image-to-image generator only when explicitly enabled. The base experiment compares three CRNN variants across seven data recipes, producing twenty-one model runs. If CycleGAN and diffusion outputs are generated, the training code automatically adds their recipes to the model search. The best model is selected by validation Character Error Rate (CER), and the output also reports Word Error Rate (WER), character accuracy, exact-match accuracy, and style-wise test performance.

Keywords: Devanagari OCR · Data Augmentation · CRNN · CTC Loss · Character Error Rate · Low-Resource OCR

1 Introduction

Devanagari is one of the major writing systems used across South Asia. It is used for Hindi, Sanskrit, Marathi, Nepali, and several other languages. The script has a horizontal headline, dependent vowel signs, consonant conjuncts, diacritics, and spacing patterns that make recognition more complex than simple isolated letter classification. A single written unit may contain marks above, below, before, or

after the main consonant body. In real document images, these marks can be blurred, broken, thickened, or shifted by scanning noise and handwriting.

Modern OCR systems usually learn from large labeled datasets. However, many project settings do not have such large data. The present work handles exactly this situation. The available data comes from a project folder in Google Drive, where page images are paired with JSON annotations. The original notebook already contains code to match the JSON files to page images, draw annotation boxes, and crop text regions. The final model-search code does not introduce a new dataset. It starts from the extracted crop list produced by the earlier cells.

The central problem is style sensitivity. If a model sees only one style, it may not generalize to thicker ink, faded ink, colored ink, textured paper, alternate fonts, or handwritten-like shapes. Collecting more real labeled data would be ideal, but this is often expensive. A practical alternative is to preserve the same transcript and create several visual forms of each crop. This does not add new language content, but it increases the range of visual appearances seen during training.

This paper documents the implemented pipeline and its evaluation design. The implementation uses traditional image operations and font-based rendering rather than external generative datasets. The pipeline is intentionally conservative: every synthetic image is generated from the project crops or their transcripts. This keeps the experiment aligned with the rule that only the available project dataset should be used.

The contributions are as follows:

- A complete crop-level OCR dataset is built from all valid annotations extracted in the notebook.
- A style augmentation pipeline generates visual variation while preserving the original transcript.
- The OCR recognizer is trained as a CRNN with CTC loss, allowing sequence learning without character-level alignment.
- Three CRNN variants are compared across seven base training-data recipes.
- Optional CycleGAN and diffusion image-to-image rows are integrated into the same experiment table when generated.
- The final notebook produces ranked CSV reports with comments explaining model behavior and data-recipe behavior.

2 Related Work

2.1 OCR with Convolutional Recurrent Networks

Text recognition from images is often treated as a sequence prediction problem. Instead of first segmenting a line into individual characters, a neural network can read the image as a sequence of visual features. Convolutional recurrent neural networks follow this idea. A convolutional backbone extracts feature maps from the image. These feature maps are collapsed into a left-to-right sequence. A

recurrent network, usually a bidirectional LSTM or GRU, models dependencies across the sequence. A final classifier predicts character probabilities.

The advantage of this design is that it does not require character-level bounding boxes. Many annotation tools provide only a crop and a transcript. This is exactly the situation in the present project. The JSON files give the transcript for each text region, but not the alignment between every character and every image column. CTC loss solves this alignment problem by summing over all possible frame-level paths that can collapse to the target transcript.

CRNN-CTC models are therefore a practical choice for low-resource OCR [2,1]. They are smaller and easier to train than transformer OCR systems, while still handling variable-length text. The present implementation uses this family of models because it matches the available labels and can be trained directly inside a Colab notebook.

2.2 Data Augmentation for Text Recognition

Data augmentation has long been used to improve OCR robustness. For text images, augmentation must preserve label correctness. A rotation that keeps text readable is safe. A deformation that changes one character into another is not safe. Therefore, augmentation for OCR should be designed around content preservation. The implementation uses OpenCV for mask and pixel operations [3] and Albumentations for random geometric transforms [4].

The present work uses augmentations that are directly connected to common document variations. Stroke-thickness changes simulate different writing instruments, scan thresholds, or ink spread. Texture and color changes simulate paper tone, ink color, and lighting variation. Font rendering introduces glyph shape diversity while preserving the transcript. Handwriting-like warps simulate slant and irregularity. Degradation simulates blur, broken strokes, and noisy scans.

The important point is that the label remains unchanged in every branch. Each variant is still paired with the original JSON transcript. This is what makes the generated data usable for supervised OCR training.

2.3 Indic and Devanagari Recognition

Indic scripts introduce additional OCR challenges because many characters are formed through combinations of base consonants, dependent vowels, and marks. Devanagari has the additional property of a headline connecting characters in a word. A recognizer must learn both local shape information and longer contextual patterns. If a model is trained only on a single printed style, it may fail when the headline is broken, the ink is thick, the matras are faint, or glyphs are warped by handwriting.

This makes style diversity valuable. However, style diversity should not come from unrelated datasets when the experiment is required to use only the project data. The implemented notebook follows this restriction strictly. It does not mix public handwriting datasets or manuscript datasets. Instead, it increases visual variety by transforming the available crops.

2.4 Generative Style Transfer

Generative adversarial networks introduced the generator–discriminator training framework [**?**]. Pix2Pix showed paired conditional image-to-image translation [**?**], while CycleGAN learns unpaired image-to-image translation using an adversarial loss and a cycle-consistency loss [**?**]. The method is useful when paired examples are unavailable. The present notebook includes CycleGAN as an optional experiment. To keep the dataset rule intact, the source domain is the project original crops and the target domain is one of the generated project styles, defaulting to `handwritten_param`. No outside target-style image dataset is used. This makes the experiment a project-local proxy for style transfer.

Diffusion models generate images through iterative denoising [**?**]. Latent diffusion models reduce the cost by operating in a compressed latent space and support conditioning through text prompts [**?**]. Image-to-image editing with stochastic differential equations motivates the use of low-strength diffusion editing from a source image [**?**]. The notebook includes a Stable Diffusion image-to-image cell as an optional experiment. It starts from the project crop and applies a low denoising strength. Because diffusion can alter letters, the notebook prints a warning that generated images should be manually inspected before being trusted.

3 Background and Motivation

OCR for printed Latin text has benefited from large datasets and mature engines, but low-resource Indic OCR remains challenging. Devanagari contains many compound forms, and the same base consonant can appear with several modifiers. The shirorekha, or headline, connects characters within words, which can make segmentation difficult. In noisy scans, this headline may merge with marks or break into fragments. Handwritten forms further increase variation because writers differ in slant, pressure, spacing, and joining behavior.

Data augmentation is a common solution when labeled data is limited. Basic geometric operations such as rotation, scaling, translation, and shear can help models tolerate layout changes. Pixel-level operations such as blur, contrast adjustment, dilation, erosion, and noise can help models handle scanner variation. For text images, augmentation must be careful: if it changes the character shape too much, it may change the label or make the text unreadable. The safest augmentations are those that preserve the transcript exactly.

The project pipeline follows this principle. It treats the transcript as the content and the image appearance as the style. The content remains fixed while style changes. The implemented styles include stroke thickness, color and background texture, font appearance, handwriting-like distortion, and document degradation. These styles correspond to common sources of variation in real OCR tasks.

Generative approaches such as CycleGAN and diffusion models can also create style transfer outputs. The notebook now contains both as optional cells. CycleGAN is kept dataset-local by using project-derived styles as the target domain. Diffusion uses a pretrained image-to-image model and is therefore off by

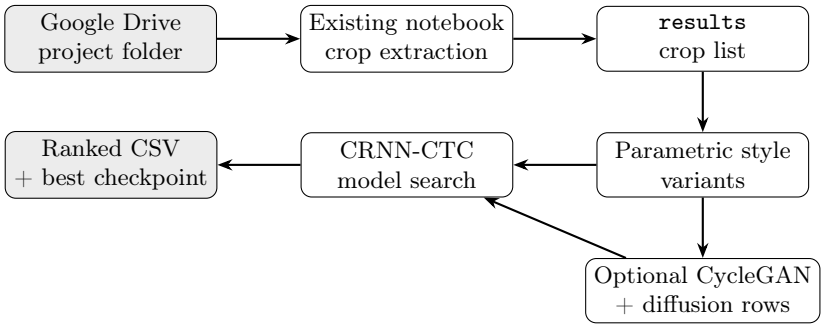

Fig. 1. Implemented pipeline. The model-search section reuses the existing `results` list, generates parametric style variants, optionally appends CycleGAN and diffusion rows, and then trains OCR models from the resulting metadata.

Table 1. Dataset sources used in the notebook.

Annotation folder	Image folder	Description
`devnagriannaotation`	`devnagri.png`	Annotated Devanagari pages
`shivannotation`	`shiv.png`	Annotated Shiva text pages

default. If either optional generator is enabled, its generated rows are appended to the metadata table and the OCR model search automatically includes additional recipes.

4 Dataset

4.1 Project Directory

The complete experiment uses the project directory:

```
/content/drive/MyDrive/indiciittp
```

This directory is mounted in Colab using Google Drive. The notebook assumes that the path variable `path` points to this folder. The final model-search section uses `PROJECT_ROOT = Path(path)` and writes all generated files to:

```
model_search_outputs
```

4.2 Folder Pairs

The dataset consists of annotation folders and image folders. The annotation files are JSON files, and the image files are page images. The notebook uses the folder pairs shown in Table 1.

Each JSON file stores annotation coordinates and text labels. The earlier cells in the notebook match each JSON file to the correct page image. The matching

logic uses filename patterns such as page numbers and trailing numeric IDs. Once a match is found, the notebook loads the page image and crops each annotated text region.

4.3 Crop Records

The extraction step stores crop information in a Python list named `results`. Each item in this list contains:

- `box`: annotation box identifier;
- `label`: ground-truth transcript;
- `crop`: NumPy array containing the cropped image;
- `coords`: crop coordinates;
- `source_image`: page image filename;
- `source_json`: source JSON filename.

The model-search section does not repeat the full extraction and visualization logic. Instead, it checks that `results` exists and then saves all valid crops to disk. This design avoids repeating code that is already present in the notebook. It also ensures that the training stage uses the same crops that were visually checked in the earlier cells.

4.4 Data Policy

No external image dataset is used. The model-search code does not download UCI Devanagari images, manuscript datasets, paper texture datasets, or handwritten style datasets. Fonts installed in the Colab environment are used only for rendering the existing project transcripts. The optional CycleGAN cell also uses only project-derived images: domain A is the original crop set and domain B is a generated style such as `handwritten_param`. The optional diffusion cell uses a pretrained image-to-image generator if enabled, but it still starts from the project crop images. Because diffusion may change text content, those images must be inspected before being treated as reliable training data.

5 Preprocessing

Before training, each valid crop is saved as a PNG image under:

`model_search_outputs/original_crops`

The corresponding metadata table is saved as:

`model_search_outputs/records_from_existing_results.csv`

The preprocessing stage normalizes labels using Unicode NFC normalization and collapses repeated whitespace. Empty labels are ignored. Very small crops are also ignored, because they are unlikely to contain usable text. If a crop is grayscale, it is converted to RGB before being saved.

Table 2. Style variants generated by the notebook.

Style	Purpose
`original`	Original crop from the project dataset
`stroke_thick`	Simulates heavier ink or thicker pen strokes
`stroke_thin`	Simulates thinner or faded writing
`color_texture`	Creates colored ink on synthetic paper texture
`font_render`	Renders the same transcript with Devanagari fonts
`handwritten_param`	Adds handwriting-like slant and elastic deformation
`doc_degrade`	Adds scan noise, blur, broken ink, and aging effects

The dataset is split into train, validation, and test partitions. When enough source JSON files are available, the code uses a group-aware split so that crops from the same source JSON file are less likely to be mixed across partitions. This reduces leakage between training and evaluation. If too few source pages are available, the code falls back to a record-level split.

5.1 Leakage Considerations

The notebook contains an earlier leakage audit section. This audit checks for duplicate page images, near-duplicate page images, repeated labels across files, out-of-bounds annotation coordinates, tiny crops, and empty labels. These checks are useful because a small dataset can produce misleading results if the same visual evidence appears in both training and testing.

The final model-search section continues this careful approach by preferring a group split based on the source JSON file. If crops from the same page are placed in different splits, a model may appear stronger than it really is because the layout, scan quality, and local background are shared. Group splitting reduces this risk when enough groups exist.

The code still includes a fallback record-level split because some project folders may contain too few source pages for a valid group split. In that case, the notebook prints a warning. This makes the experimental condition explicit.

6 Style Augmentation

The goal of augmentation is to generate style diversity while preserving text content. Each generated image keeps the same transcript as its parent crop. Table 2 lists the implemented styles.

6.1 Stroke Thickness

Stroke-thickness augmentation is implemented with morphological operations. The code first estimates the ink mask using grayscale conversion, Gaussian blur, and Otsu thresholding. The foreground mask is then dilated or eroded using

OpenCV operations [3]. Dilation expands foreground pixels and produces thicker strokes. Erosion removes foreground pixels near boundaries and produces thinner strokes.

For an image mask I and structuring element B, dilation can be written as:

$$(I \oplus B)(x, y) = \max_{(i,j)\in B} I(x - i, y - j). \tag{1}$$

Erosion can be written as:

$$(I \ominus B)(x, y) = \min_{(i,j)\in B} I(x + i, y + j). \tag{2}$$

The notebook samples kernel sizes randomly. This prevents all generated images from looking identical.

6.2 Color and Background Texture

The `color_texture` variant changes black text into colored ink and places it on a synthetic paper-like background. The background is generated from a base color, random noise, and smooth intensity gradients. The ink mask is then composited over this background using an alpha mask. This simulates red, blue, green, purple, brown, or dark ink on aged paper.

This step is important because OCR models trained only on black text over white backgrounds may fail when contrast, paper color, or ink color changes.

6.3 Font Rendering

The `font_render` style uses the crop transcript and renders it with available Devanagari fonts in the Colab environment. The code searches common font directories such as `/usr/share/fonts`. Rendering is performed with Pillow's `ImageFont` and `ImageDraw` [**?**]. When possible, the rendering uses the RAQM layout engine to improve shaping of complex text.

The font size is selected so that the rendered text fits inside the target crop size. The rendered text is then placed on either a light background or a synthetic paper texture. Since the transcript is typed directly, this variant preserves the intended label.

6.4 Handwriting-Like Warping

The `handwritten_param` style applies affine and elastic deformation. When Albumentations is available, the notebook applies affine rotation, translation, shear, and elastic transformation. If Albumentations fails, the code falls back to an OpenCV affine transform.

This does not create true human handwriting. It creates handwriting-like variation, including slant, slight distortion, and irregular shape. This is a parametric approximation and is included because no real external handwriting dataset is used.

6.5 Document Degradation

The `doc_degrade` style simulates noisy scans and aged documents. It randomly removes some foreground pixels, applies blur, adds Gaussian noise, and uses textured backgrounds. These changes imitate broken ink, scanner blur, and paper aging. The transcript remains unchanged.

7 Optional Generative Augmentation

The notebook now includes optional CycleGAN and diffusion cells. These cells are placed after `metadata = make_metadata(records)` and before OCR training. This placement is important because the generators append new rows to `metadata`. The OCR model-search cell reads `metadata` and adds extra recipes automatically if the new styles exist.

7.1 CycleGAN Experiment

CycleGAN is controlled by:

```
RUN_CYCLEGAN = False
```

To run it, the user changes this value to `True`. The default target style is:

```
CYCLEGAN_TARGET_STYLE = "handwritten_param"
```

The generated style name becomes:

```
cyclegan_handwritten_param
```

The CycleGAN data preparation uses:

- `trainA`: original training crops from the project dataset;
- `trainB`: generated target-style crops from the project dataset;
- `testA`: original crops to translate into the target style.

The output images are copied into:

```
model_search_outputs/augmented/cyclegan_<target-style>
```

Rows for these generated images are appended to `metadata` with the same transcripts as their parent crops.

The CycleGAN objective contains adversarial and cycle-consistency terms:

$$\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) = E_{y\sim Y}[\log D_Y(y)] + E_{x\sim X}[\log(1 - D_Y(G(x)))] \quad (3)$$

and

$$\mathcal{L}_{\text{cyc}}(G, F) = E_{x\sim X}[\|F(G(x)) - x\|_1] + E_{y\sim Y}[\|G(F(y)) - y\|_1]. \quad (4)$$

This objective encourages the generated image to look like the target domain while remaining recoverable back to the source domain.

7.2 Diffusion Image-to-Image Experiment

Diffusion image-to-image generation is controlled by:

```
RUN_DIFFUSION = False
```

To run it, the user changes this value to `True`. The current notebook uses:

```
DIFFUSION_MODEL_ID = "runwayml/stable-diffusion-v1-5"
```

and writes the generated rows under the style:

```
diffusion_img2img
```

The cell uses the original project crop as the input image. The prompt asks for legible Devanagari text with the same characters and layout, while the negative prompt discourages wrong letters, Latin letters, unreadable text, and extra characters. The denoising strength is kept low:

```
DIFFUSION_STRENGTH = 0.22
```

This low value is chosen to reduce the risk of changing the text. Even so, diffusion outputs are riskier than parametric augmentations. The notebook therefore prints a warning that these images should be inspected manually.

7.3 Generated Dataset Size

The notebook sets `VARIANTS_PER_STYLE = 4`. For each original crop, it keeps one original row and generates four variants for each of the six synthetic styles. Therefore, if the number of extracted valid crops is N, the metadata contains:

$$N_{\text{total}} = N \times (1 + 6 \times 4). \tag{5}$$

This equals $25N$ image rows. The value of N is not hard-coded in the paper because the notebook extracts all valid crops from the current annotation files. If more annotations are added to the Drive folder, the pipeline automatically uses them.

7.4 Why These Augmentations Are Safe

The chosen augmentations are safe because they either operate on the existing ink mask or render the same transcript. Stroke operations change foreground thickness. Color and texture change appearance but not the character identity. Font rendering uses the label string itself. Handwriting-like deformation is kept mild. Degradation changes scan quality but not the intended text. This design reduces the chance of assigning an incorrect label to a generated image.

8 Model Architecture

The recognizer is a convolutional recurrent neural network trained with CTC loss. The input image is converted to grayscale, resized while preserving aspect ratio, and padded to a fixed canvas of height 48 and width 512. The image is inverted so that text has high intensity and background has low intensity.

The CNN extracts a sequence of visual features. The recurrent layer processes this sequence from left to right and right to left. The final linear layer predicts character probabilities at each time step. CTC loss trains the model without requiring character-level bounding boxes.

The CTC objective is suitable because the dataset provides full crop transcripts but does not provide alignment between characters and image columns. Given an input image x and transcript y, CTC sums over all valid alignments π that collapse to y:

$$P(y|x) = \sum_{\pi \in \mathcal{B}^{-1}(y)} P(\pi|x), \tag{6}$$

where $\mathcal{B}$ removes repeated labels and blank symbols.

8.1 Input Representation

Every crop is resized to fit inside a fixed canvas of height 48 pixels and width 512 pixels. The aspect ratio is preserved. If the resized crop is narrower than the canvas, the remaining area is padded with white background. This keeps batch training simple while avoiding distortion from direct stretching.

The image is converted to grayscale and inverted. After inversion, text strokes have larger numerical values and the background has smaller values. This is a common convention for OCR models because it makes foreground information more prominent.

8.2 Character Vocabulary

The character vocabulary is built from the labels in the extracted project records. A blank class is added for CTC. This means the model predicts only characters present in the project transcripts. If future data contains new characters, rerunning the notebook will rebuild the vocabulary automatically.

8.3 Decoding

The implementation uses greedy decoding. At each time step, the most probable class is selected. Consecutive duplicate symbols are collapsed, and blank symbols are removed. Beam search is not used in the current notebook. Greedy decoding is simple and makes the comparison between experiments direct.

Table 3. OCR model variants used in the search.

Model	Recurrent unit	Comment
`crnn_lstm_light`	BiLSTM	Fast baseline for small data
`crnn_gru_light`	BiGRU	Fewer recurrent parameters
`crnn_lstm_base`	BiLSTM	Stronger CRNN configuration

Table 4. Training data recipes.

Recipe	Styles used for training	Condition
`original_only`	`original`	Always
`stroke`	`original`, `stroke_thick`, `stroke_thin`	Always
`color_texture`	`original`, `color_texture`	Always
`font`	`original`, `font_render`	Always
`handwriting`	`original`, `handwritten_param`	Always
`degradation`	`original`, `doc_degrade`	Always
`all_styles`	All parametric styles	Always
`cyclegan`	`original`, `cyclegan_*`	If CycleGAN is run
`all_styles_plus_cyclegan`	All parametric styles + `cyclegan_*`	If CycleGAN is run
`diffusion`	`original`, `diffusion_img2img`	If diffusion is run
`all_styles_plus_diffusion`	All parametric styles + `diffusion_img2img`	If diffusion is run
`all_styles_plus_cyclegan_diffusion`	All parametric + both optional styles	If both are run

9 Model Search

The notebook trains three model configurations. These are shown in Table 3.

Each model is trained with seven data recipes, listed in Table 4.

The base search therefore contains:

$$3 \text{ models} \times 7 \text{ recipes} = 21 \text{ experiments}. \tag{7}$$

If CycleGAN alone is enabled, the search contains nine recipes and therefore twenty-seven model runs. If diffusion alone is enabled, the search also contains twenty-seven model runs. If both optional generators are enabled, the search contains twelve recipes and therefore thirty-six model runs:

$$3 \text{ models} \times 12 \text{ recipes} = 36 \text{ experiments}. \tag{8}$$

9.1 Model Comments

The comparison table stores a short comment for each model. The light LSTM model is treated as a fast baseline. It is useful when the dataset is small or when GPU time is limited. The light GRU model uses fewer recurrent parameters and can sometimes generalize better on small data. The base LSTM model has a larger convolutional width and hidden state. It is expected to be stronger when the augmented dataset is large enough, but it can also overfit if validation data is too small.

9.2 Data Recipe Comments

The notebook also stores comments for each data recipe. The `original_only` recipe is the baseline. The `stroke` recipe tests ink thickness robustness. The `color_texture` recipe tests robustness to colored ink and non-white backgrounds. The `font` recipe tests whether alternate glyph shapes help. The `handwriting` recipe tests slanted and warped shapes. The `degradation` recipe tests blur, noise, and broken ink. The `all_styles` recipe tests whether combining all generated styles improves overall robustness. If CycleGAN rows exist, `cyclegan` and `all_styles_plus_cyclegan` are added. If diffusion rows exist, `diffusion` and `all_styles_plus_diffusion` are added. If both exist, the combined recipe `all_styles_plus_cyclegan_diffusion` is added.

10 Training Procedure

The implementation uses PyTorch [**?**]. Each experiment trains for a maximum number of epochs, with early stopping based on validation CER. The optimizer is AdamW with weight decay [**?**]. The training loop clips gradients to reduce instability. A learning rate scheduler reduces the learning rate when validation CER stops improving.

For each experiment, the notebook saves the best checkpoint according to validation CER. Checkpoints are stored in:

`model_search_outputs/checkpoints`

During decoding, the model uses greedy CTC decoding. The predicted class with the highest probability is selected at each time step. Repeated symbols are collapsed and blank symbols are removed.

10.1 Early Stopping

The training loop tracks validation CER. If validation CER improves, the current checkpoint is saved. If it does not improve for a fixed number of epochs, the training run stops early. This prevents wasting GPU time on experiments that are no longer improving. It also reduces overfitting on the small dataset.

10.2 Optimization

The optimizer is AdamW. Weight decay is used to regularize the model. Gradients are clipped to a maximum norm to avoid unstable updates. A ReduceLROnPlateau scheduler lowers the learning rate when the validation CER stops improving. These choices are standard for small neural OCR experiments.

10.3 Reproducibility

The notebook sets a fixed random seed for Python, NumPy, and PyTorch. CUDA seeds are also set when a GPU is available. Augmentation seeds are generated from the parent crop ID, style name, and variant index. This allows the generated variants to be reproduced when the notebook is rerun with the same input data.

11 Evaluation Metrics

The notebook evaluates OCR quality using Character Error Rate, Word Error Rate, character accuracy, and exact-match accuracy.

Character Error Rate is:

$$\mathrm{CER} = \frac{S + D + I}{N}, \tag{9}$$

where S is the number of substitutions, D is deletions, I is insertions, and N is the number of characters in the reference transcript.

Word Error Rate applies the same edit-distance calculation over word tokens:

$$\mathrm{WER} = \frac{S_w + D_w + I_w}{N_w}. \tag{10}$$

Character accuracy is computed as:

$$\text{Character Accuracy} = 1 - \mathrm{CER}. \tag{11}$$

Exact-match accuracy is the fraction of predictions that match the full reference transcript exactly.

11.1 Result Selection

The best experiment is selected by the lowest validation CER. Test CER is used as a secondary check. This is important because choosing directly by test performance would leak test information into model selection. The notebook sorts the final table by validation CER and then by test CER.

11.2 Style-Wise Evaluation

After selecting the winning checkpoint, the notebook evaluates it separately on each style present in the test split. This produces a style-wise report. The report shows whether the best model is strong on clean text, thick strokes, thin strokes, colored backgrounds, font-rendered samples, handwriting-like variants, and degraded samples. This is useful because one model may have the best overall CER while still being weak on a specific style.

12 Implemented Pipeline

12.1 Notebook Execution Order

The notebook should be run in order. First, the Drive is mounted and the project folder is listed. Second, the existing extraction and visualization cells should be executed so that `results` is created. Third, the model-search section should be run. If the user skips the extraction cells, the model-search section raises an error telling the user to run the existing crop extraction cell first.

This design avoids silently training on incomplete data. It also keeps the new section short and connected to the earlier notebook work.

Table 5. Main output files produced by the notebook.

Output path	Purpose
`records_from_existing_results.csv`	Saved crop metadata
`style_metadata.csv`	Metadata for original and generated styles
`model_comparison_with_comments.csv`	Ranked experiment table
`best_model_style_report.csv`	Style-wise report for best model
`checkpoints/`	Saved PyTorch model checkpoints
`cyclegan_data/`	CycleGAN train/test domains, if enabled
`cyclegan_checkpoints/`	CycleGAN checkpoints, if enabled
`cyclegan_results/`	Raw CycleGAN translated images, if enabled
`augmented/diffusion_img2img/`	Diffusion image-to-image outputs, if enabled

13 Outputs

The notebook writes several artifacts. The most important files are shown in Table 5.

The comparison table contains the model name, data recipe, checkpoint path, validation CER, validation WER, validation character accuracy, validation exact-match accuracy, test CER, test WER, test character accuracy, and test exact-match accuracy. It also includes comments explaining the model type, the data recipe, and whether the result improves over the original-only baseline.

13.1 How to Read the Comparison Table

The most important column is `val_cer`. Lower is better. The best row is the row with the smallest validation CER. The `test_cer` column is used to confirm that the selected model also performs well on held-out data. The `result_comment` column explains whether a model is the best, improves over the original-only baseline, or fails to beat the baseline.

The table should not be interpreted as a claim that one augmentation is always best for all Devanagari OCR tasks. It only answers the question for the current project dataset and split.

13.2 Expected Console Output

The notebook prints progress messages that confirm the pipeline state. The exact numbers depend on the number of extracted crops, but the output format follows the pattern below:

```
Device: cuda
Using only dataset under: /content/drive/MyDrive/indiciittp
All crops used: <N>
RUN_CYCLEGAN is False. Set it to True to train CycleGAN and append generated rows.
RUN_DIFFUSION is False. Set it to True to generate diffusion img2img rows.
```

Table 6. Schema of the ranked experiment output.

Column	Meaning
`model`	CRNN variant name
`recipe`	Data recipe used for training
`train_styles`	Styles included in the training set
`checkpoint`	Path to the saved best checkpoint
`model_note`	Explanation of model variant
`data_note`	Explanation of recipe
`train_rows`	Number of rows used for training
`val_cer`, `test_cer`	Character error rate
`val_wer`, `test_wer`	Word error rate
`val_char_accuracy`, `test_char_accuracy`	Character accuracy
`val_exact_accuracy`, `test_exact_accuracy`	Exact transcript match rate
`result_comment`	Automatic interpretation of result

```
Recipes to train: {...}
BEST MODEL
Model: <model-name>
Recipe: <recipe-name>
Validation CER: <value>
Validation char accuracy: <value>
Test CER: <value>
Test char accuracy: <value>
Checkpoint: <checkpoint-path>
```

When CycleGAN is enabled, the notebook prints the number of CycleGAN rows added. When diffusion is enabled, it prints the number of diffusion rows added and reminds the user to inspect the generated images manually.

14 Results Reporting

The notebook produces real numerical results after it is run in Colab. This paper does not invent accuracy numbers. Instead, it defines the exact table that must be filled from `model_comparison_with_comments.csv`. A result row has the fields shown in Table 6.

A publication-ready results table can be created from the first rows of that CSV. The template is shown in Table 7.

The style-wise report should be read after the best model is selected. It shows which visual style remains difficult. For example, if `doc_degrade` has high CER, the degradation recipe may need weaker noise. If `diffusion_img2img` has high CER, the diffusion strength may be too high or the generated characters may have changed.

Table 7. Result table template to fill after running the notebook.

Rank	Model	Recipe	Val CER	Test CER
1	from CSV	from CSV	from CSV	from CSV
2	from CSV	from CSV	from CSV	from CSV
3	from CSV	from CSV	from CSV	from CSV

15 Enabled and Disabled Experiment Conditions

The notebook supports three experiment modes.

Base mode. This is the default mode. It runs parametric and font-based augmentation, then trains the CRNN-CTC model search over seven recipes. CycleGAN and diffusion are not run because:

`RUN_CYCLEGAN = False, RUN_DIFFUSION = False.`

CycleGAN mode. If `RUN_CYCLEGAN` is set to `True`, the notebook trains a CycleGAN model from original crops to a project-generated style domain. This does not add an external target-style image dataset. It does clone the official CycleGAN implementation and trains a generator inside the notebook environment. After generation, the OCR model search adds the `cyclegan` and `all_styles_plus_cyclegan` recipes.

Diffusion mode. If `RUN_DIFFUSION` is set to `True`, the notebook uses Stable Diffusion image-to-image generation with the original crop as input. This requires a pretrained model download and therefore is not a dataset-only augmentation in the same strict sense as the parametric styles. The generated images should be manually inspected because text content may change. After generation, the OCR model search adds the `diffusion` and `all_styles_plus_diffusion` recipes.

If both optional generators are enabled, the notebook also adds `all_styles_plus_cyclegan_diffusion`. This conditional design keeps the base experiment reproducible while still allowing extended generative experiments when compute and inspection time are available.

16 Discussion

The implemented method is designed for a practical low-resource setting. It does not claim to solve all Devanagari OCR problems. Its main strength is that it uses all available annotation crops and compares several controlled augmentation recipes. This makes the experiment useful for identifying which visual variation helps the current dataset most.

The first limitation is that most synthetic styles are still generated from the available project crops. They may not fully represent real handwriting, historical manuscripts, or damaged archive documents. The handwriting-like variant is a

parametric approximation, not real human handwriting. CycleGAN can create a learned style mapping, but in the default configuration its target domain is also generated from project crops. The font variant is useful for shape diversity, but it may not match natural handwritten stroke flow.

The second limitation is that the validation and test sets are only as strong as the available data. If the dataset is very small, metric values may vary across splits. The code partly addresses this by using grouped splitting when enough source JSON files are available.

The third limitation is that aggressive augmentation can hurt OCR performance if it makes characters unreadable. This is especially important for diffusion outputs, because a diffusion model can create visually plausible text that no longer matches the original transcript. This is why the final comparison table is important. It does not assume that all augmentation helps. It trains each recipe and lets validation CER decide.

16.1 Expected Outcomes

The likely strong recipes are `all_styles` and the single augmentation recipes that best match the validation/test distribution. However, the notebook does not hard-code this assumption. If font rendering creates shapes that do not match the original data, the `font` recipe may perform worse. If handwriting-like deformation is too strong, the `handwriting` recipe may increase CER. If CycleGAN overfits to the proxy target domain, its recipe may not help. If diffusion changes letters, its recipe may hurt. If the original test data is very clean, `original_only` may remain competitive. These possibilities are why the full model search is run.

16.2 Practical Use

After the notebook finishes, the user should open `model_comparison_with_comments.csv`. The first row gives the best model according to validation CER. The checkpoint path in that row identifies the model file to reuse. The user should then inspect `best_model_style_report.csv` to see which style categories still have high error. If one style is weak, future work should focus on improving that style's augmentation quality.

16.3 Scope of the Claim

The experiment demonstrates a method for using the available project data more effectively. It does not claim state-of-the-art Devanagari OCR performance. It does not compare against large pretrained OCR systems. It does not use outside training data. Its purpose is to create a clear, reproducible baseline for the current dataset and to identify the best model/data recipe among the implemented options.

17 Conclusion

This paper documents a Devanagari OCR pipeline that matches the current implementation. The notebook uses the available project dataset in Google Drive. It extracts all valid crops from existing JSON annotations, generates style-preserving parametric variants from those crops, optionally adds CycleGAN and diffusion image-to-image rows, trains multiple CRNN-CTC recognizers, and ranks the results by validation CER. The final system produces a best model checkpoint, a full model-comparison CSV, and a style-wise error report.

The most important feature of the work is that it separates content from style without changing the transcript. The content is the JSON label, and the style is the visual appearance of the crop. By varying the style while preserving the label, the recognizer is encouraged to learn Devanagari character structure rather than a single image appearance.

Future work can add real target-style examples if the dataset rule changes. With real handwritten or manuscript images, CycleGAN could be trained against a more realistic domain instead of a project-generated proxy domain. Diffusion could be improved by adding automatic OCR-based filtering, where generated samples are rejected if their predicted transcript disagrees with the original label.

References

1. Graves, A., Fernandez, S., Gomez, F., Schmidhuber, J.: Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In: ICML, pp. 369–376 (2006)
2. Shi, B., Bai, X., Yao, C.: An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. IEEE TPAMI 39(11), 2298–2304 (2017)
3. Bradski, G.: The OpenCV Library. Dr. Dobb's Journal of Software Tools (2000)
4. Buslaev, A., Iglovikov, V.I., Khvedchenya, E., Parinov, A., Druzhinin, M., Kalinin, A.A.: Albumentations: fast and flexible image augmentations. Information 11(2), 125 (2020)

Algorithm 1 Implemented OCR Model Search Pipeline with Optional Generators

Require: Existing crop list `results`
Ensure: Ranked model comparison table and best checkpoint
Validate that `results` exists and contains crops
Save every valid crop to `original_crops`
Normalize labels and remove unusable records
Split records into train, validation, and test sets
for each crop **do**
    Add the original crop and transcript to metadata
    Generate thick-stroke and thin-stroke variants
    Generate colored-ink and textured-background variant
    Generate font-rendered variant
    Generate handwriting-like warped variant
    Generate degraded-document variant
end for
if `RUN_CYCLEGAN` is true **then**
    Build CycleGAN `trainA` from original training crops
    Build CycleGAN `trainB` from target generated style
    Train CycleGAN and translate original crops to target style
    Append generated rows to `metadata`
end if
if `RUN_DIFFUSION` is true **then**
    Run image-to-image diffusion from original project crops
    Append generated rows to `metadata`
    Mark diffusion outputs for manual inspection
end if
for each CRNN model configuration **do**
    for each available data recipe **do**
        Train with CTC loss
        Evaluate on validation set
        Save best checkpoint for the experiment
    end for
end for
Rank experiments by validation CER and test CER
Save the ranked comparison CSV with comments
Evaluate the winning model separately by style