Documentation

Himeksh Malhotra | 2022218 | himeksh22218@iiitd.ac.in

Lakshay Kumar | 2022266 | lakshay22266@iiitd.ac.in

Guided by : Dr. A V Subramanyam | subramanyam@iiitd.ac.in

Table of Contents

6. **Overall Project Pipeline and Inter-module Synergy**
7. **General Conclusion and Broader Future Directions**

---

1. General Introduction and Literature Review

The study of inscriptions, or epigraphy, is crucial for understanding historical, cultural, and linguistic aspects of past civilizations. Traditionally, this involves manual transcription, translation, and analysis, which are laborious and require specialized expertise. The digitization of epigraphic artifacts and the application of computational methods offer transformative potential for the field.

Challenges in Inscription Analysis:

Inscriptions often present numerous challenges for automated analysis:

- **Degradation:** Weathering, erosion, physical damage, and fading of pigments can obscure characters.
- **Variability:** Scripts evolve over time, and handwriting styles vary significantly. Ligatures, abbreviations, and non-standard character forms are common.
- **Material and Background:** Inscriptions appear on diverse materials (stone, metal, clay, papyrus) with varying textures, colors, and potential occlusions (e.g., lichen, cracks, later markings).
- **Imaging Conditions:** Illumination, perspective, and resolution during image acquisition can greatly impact readability and downstream processing.
- **Data Scarcity:** Large, well-annotated datasets for specific ancient scripts are often unavailable, hindering the training of robust machine learning models.

Digital Approaches to Epigraphy:

Early digital approaches focused on image enhancement techniques to improve human readability. More recently, computer vision and machine learning have enabled more automated analyses:

- **Text Detection:** Identifying the regions in an image that contain text. This is a precursor to recognition and often involves techniques ranging from traditional edge and morphology-based methods to sophisticated deep learning models like Faster R-CNN, SSD, EAST, and more recently, DBNet and its variants, which excel at arbitrarily shaped text instances.
- **Character Segmentation:** Isolating individual characters from detected text lines or regions. This can be challenging with connected scripts or degraded inscriptions.

- **Character Recognition:** Classifying segmented characters into their respective graphemes. Optical Character Recognition (OCR) and Handwritten Text Recognition (HTR) are active research areas. For historical documents, methods often need to handle large character sets and significant intra-class variation. Convolutional Neural Networks (CNNs) are standard for feature extraction, often combined with Recurrent Neural Networks (RNNs) like LSTMs for sequence modeling in CRNN architectures, especially for line-level recognition or when contextual information is beneficial.
- **Image Inpainting/Restoration:** Digitally filling in missing or damaged parts of an inscription. Generative Adversarial Networks (GANs), particularly models like DeepFill with contextual attention and gated convolutions, have shown promise in generating plausible reconstructions of missing regions, which can aid both human experts and subsequent automated analysis.
- **3D Modeling and Analysis:** Using techniques like Reflectance Transformation Imaging (RTI) or 3D scanning to capture surface details, which can reveal features not visible in 2D images.

Role of Computer Vision and Deep Learning:

The modules documented below leverage state-of-the-art deep learning techniques to address specific tasks within the broader goal of inscription analysis. The CRNN aims at classifying individual characters, DeepFill at restoring damaged inscription images, MMOCR at detecting text regions, and a foundational image processing script at isolating specific markings. Together, these tools can form a powerful pipeline for extracting information from epigraphic sources. The successful application of these methods often relies on careful dataset preparation, domain-specific fine-tuning, and an understanding of the inherent challenges posed by historical artifacts.

---

2. Module 1: Pen Mark Extraction from Inscriptions

overview:

This script serves as a crucial preprocessing or analytical step for inscription images where modern pen marks (e.g., annotations by researchers, tracing for emphasis) need to be isolated from the original inscription or the substrate. By creating a binary mask of these marks, subsequent analyses can either focus on these marks, exclude them, or use them to guide other processes. The script employs fundamental image processing operations to achieve robust segmentation of these often high-contrast additions.

Literature Context for Segmentation:

The task of isolating pen marks is essentially a foreground-background segmentation problem.

- **Thresholding:** This is one of the simplest and most effective methods when the foreground (pen marks) has a distinct intensity range compared to the background. Global thresholding, as used here, applies a single threshold value across the entire image. Adaptive thresholding methods (e.g., Otsu's method, adaptive mean/Gaussian) can be more robust if illumination varies significantly, though they might be overly sensitive for this specific task if the pen marks are consistently dark. The use of `cv2.THRESH_BINARY_INV` is common when darker regions need to be segmented as white (foreground).
- **Morphological Operations:** Noise, small gaps in strokes, or irrelevant small dark spots can be problematic. Morphological operations like erosion, dilation, opening, and closing are standard techniques for refining binary masks.
    - **Opening (Erosion followed by Dilation):** As used in the script, this is effective for removing small noise objects (salt noise) and breaking thin connections without significantly affecting the size of the main objects.
    - **Closing (Dilation followed by Erosion):** Useful for filling small holes within objects (pepper noise) and connecting nearby components.

Detailed Workflow and Parameter Justification:

1. **Image Reading and Grayscaling:**
    - `cv2.imread(img_path)`: Loads the image.
    - `cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)`: Converts the image to grayscale. Color information is typically not necessary for distinguishing dark pen marks from a lighter background and simplifies the thresholding process.
2. **Thresholding:**
    - `cv2.threshold(gray_image, threshold_value, 255, cv2.THRESH_BINARY_INV)`: This is the core step.
        - `threshold_value`: Pixels with intensity *below* this value (due to `THRESH_BINARY_INV`) will be set to 255 (white), and those above will be set to 0 (black). The default of 50 assumes pen marks are significantly darker than this value. This parameter is critical and may need tuning based on the dataset (ink color, paper/stone color, scanning brightness).
        - `cv2.THRESH_BINARY_INV`: Inverted binary thresholding. This is chosen because pen marks are typically dark, and the goal is to make them white (foreground) in the mask.

3. **Morphological Opening:**
    - `kernel = np.ones((3,3), np.uint8)`: Defines a 3x3 square structuring element. The size and shape of the kernel can influence the aggressiveness of the noise removal.
    - `cv2.morphologyEx(thresh_image, cv2.MORPH_OPEN, kernel)`: Performs the opening operation. This helps remove small, isolated white pixels (noise) that might have resulted from the thresholding of tiny dark specks in the original image, while generally preserving the shape and size of the actual pen strokes.
4. **Saving Output:**
    - The script maintains the input folder structure, which is good practice for dataset management. The "pen_" prefix clearly indicates the nature of the processed image.

Limitations and Considerations:

- **Threshold Value Dependency:** The fixed `threshold_value` might not be optimal for all images, especially if lighting conditions or ink intensity vary. A very low threshold might pick up faint parts of the original inscription if it's dark, while a very high threshold might miss fainter pen strokes.
- **Color Overlap:** If the original inscription itself contains very dark elements similar in intensity to the pen marks, this script will not distinguish between them. It purely works on intensity.
- **Mark Thickness and Kernel Size:** If pen marks are very thin, an aggressive morphological opening (e.g., larger kernel) might inadvertently remove parts of them. The 3x3 kernel is generally a safe choice for moderately sized noise.
- **Complex Backgrounds:** If the background is highly textured or has its own dark patterns, the simple thresholding might produce a noisy mask.

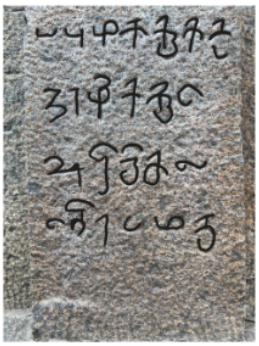
Future Enhancements for this Module:

- **Adaptive Thresholding:** Explore `cv2.adaptiveThreshold` if global thresholding proves insufficient across a diverse dataset.
- **Color-based Segmentation:** If pen marks have a distinct color (e.g., blue or red ink on a monochrome inscription), convert the image to a color space like HSV or LAB and threshold on hue or chromaticity channels for more robust segmentation.
- **Connected Components Analysis:** After thresholding, use `cv2.connectedComponentsWithStats` to filter out components based on size or aspect ratio, providing more refined noise removal than morphological opening alone.

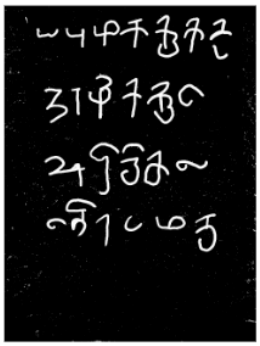
- **User Interface for Threshold Tuning:** For interactive use, a simple GUI slider to adjust `threshold_value` and see a live preview could be very helpful.
- **Batch Processing with Parameter Sweep:** Allow testing a range of threshold values on a subset of images to determine an optimal setting.

8 incriscription in total : 50 images, multiple images of same inscription from many different angles

Raw image



Manually inked



mask from inked image

---

3. Module 2: MMOCR Custom Text Detection Pipeline

overview:

This module leverages the MMOCR toolbox for the critical task of text detection in images of stone inscriptions. Text detection identifies the locations (typically as bounding boxes or polygons) of text within an image, which is a fundamental first step before any recognition or further analysis can occur. The choice of DBNet++, particularly the dbnetpp_resnet50-oclip_fpnc_1200e_icdar2015.py configuration, indicates a focus on robust and accurate detection, likely capable of handling the irregular text layouts and orientations often found in inscriptions.

Literature Context for Text Detection:

Text detection has evolved significantly:

- **Traditional Methods:** Early approaches relied on edge detection, connected component analysis, and Maximally Stable Extremal Regions (MSERs). These methods often struggled with complex backgrounds, variable fonts, and lighting.

- **Deep Learning Era - Bounding Box Based:** Models like Faster R-CNN and SSD adapted object detection frameworks for text, primarily predicting rectangular bounding boxes.
- **Deep Learning Era - Segmentation Based / Arbitrary Shapes:** For non-horizontal or curved text (common in inscriptions or scene text), segmentation-based approaches became prominent.
    - **EAST (Efficient and Accurate Scene Text Detector):** Directly predicts quadrilaterals.
    - **PixelLink, TextSnake:** Segment text regions and link pixels belonging to the same text instance.
    - **DBNet (Differentiable Binarization):** This model, and its successor DBNet++, are particularly relevant here. They predict a probability map and a threshold map. The differentiability of the binarization process allows for end-to-end training and often leads to more precise boundaries, especially for arbitrarily shaped text. DBNet++ enhances DBNet with an Adaptive Scale Fusion module for better handling of text instances at various scales.
- **Transformer-based Models:** More recently, Vision Transformers (ViTs) and DETR-like architectures are being explored for text detection, showing promising results.

The selection of DBNet++ with a ResNet-50 backbone and Feature Pyramid Network (FPN) components (indicated by `fpnc`) is a strong choice, balancing accuracy and efficiency. The "o-clip" likely refers to a specific pre-training strategy or an architectural variant that might involve contrastive language-image pre-training (like OpenAI's CLIP) adapted for OCR tasks, potentially improving feature representation.**Deep Dive into Configuration Choices:**

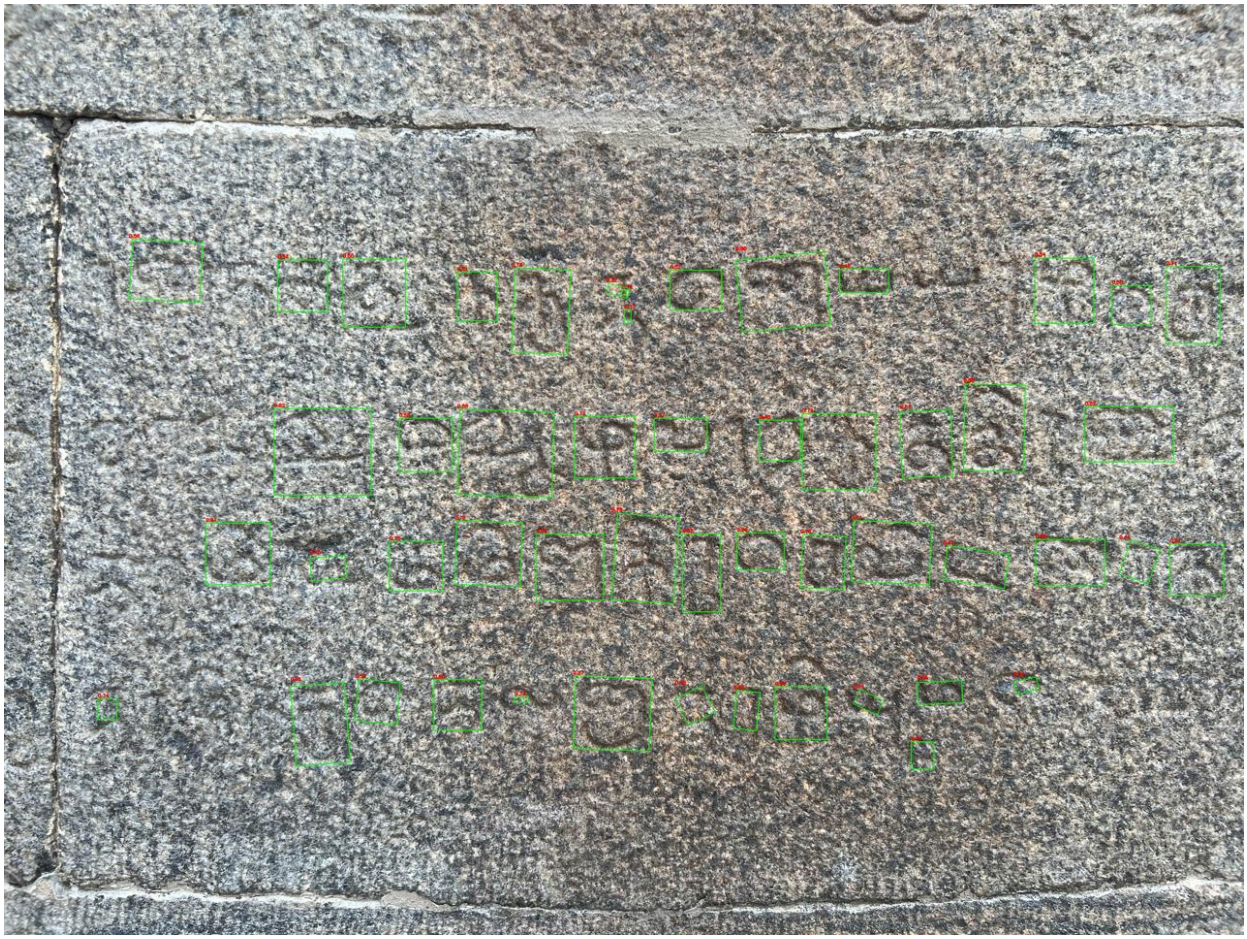
- `dbnetpp_resnet50-oclip_fpnc_1200e_icdar2015.py` **(Best Results):**
    - `dbnetpp`: Uses the DBNet++ architecture, known for its robustness with arbitrarily shaped text and improved performance over the original DBNet.
    - `resnet50`: A ResNet-50 backbone is a common choice, offering a good trade-off between feature extraction power and computational cost. `with_cp = True` (checkpointing) is a smart move to reduce memory usage during training by trading some compute for memory, essential for deeper models or larger batch sizes.
    - `oclip`: Suggests the use of a ResNet-50 variant pre-trained with a CLIP-like methodology, which can provide richer semantic features beneficial for distinguishing text from complex backgrounds found on inscriptions.

    - `fpnc`: Implies the use of a Feature Pyramid Network (FPN) for neck, which helps in detecting text at multiple scales by combining features from different layers of the backbone.
    - `1200e_icdar2015`: Indicates the model was potentially pre-trained or its training schedule is inspired by a long training run (1200 epochs) on a standard OCR dataset like ICDAR2015. This long pre-training helps in learning robust features.
- `init_cfg = None`**:** Training from scratch on the custom dataset. This might be chosen if the custom dataset is significantly different from common OCR datasets, or if a specific pre-trained model wasn't available/suitable for the `oclip` variant. However, if `oclip` itself implies a pre-trained backbone, `init_cfg=None` might refer to initializing only the detection head from scratch.
- **Learning Rate (0.002) and Scheduler (LinearLR warmup + PolyLR decay):** This is a standard and effective learning rate strategy. Linear warm-up prevents early divergence by starting with a small LR, and polynomial decay gradually reduces the LR over epochs, allowing for finer adjustments as training progresses.
- **Epochs (1000):** A substantial number of epochs, indicating an expectation that the model requires significant training on the custom dataset to achieve good performance.
- **Visualizer (LocalVisBackend, TensorboardVisBackend):** Essential for monitoring training progress, debugging, and visualizing model predictions.
- **Hooks (LoggerHook, CheckpointHook, VisualizationHook):** Standard MMOCR hooks for managing training, saving models, and visualizing at set intervals.

Interpreting Results and Metrics:

- **Recall (~67%):** Out of all the actual text instances in the dataset, the model correctly identified 67% of them. A lower recall means more missed text (false negatives).
- **Precision (~72%):** Of all the text instances the model predicted, 72% of them were actually correct. Lower precision means more incorrect detections (false positives).
- **F1-Score (Implicit):** While not stated, the harmonic mean of Precision and Recall (F1-Score = 2×Precision+RecallPrecision×Recall) is often the primary metric. For these values, F1 ≈69.4%. These results are decent, especially for challenging inscription data. The balance between recall and precision can be tuned (e.g., by adjusting the detection score threshold). The low `pred_score_thr` (e.g., 0.0) during inference for visualization ensures all potential detections are

seen, but for actual metric calculation, a higher, optimized threshold would be used.



Further Experimentation:

- **Schedulers and Learning Rates:** Experiment with different warm-up lengths, decay strategies (e.g., cosine annealing), or even cyclical learning rates.
- **Backbones:**
    - **DCNv2 (Deformable Convolutional Networks v2):** Integrating DCNv2 into the backbone or detection head can improve the model's ability to handle irregular text shapes and scales, as indicated by the other config files explored.
    - **ViT (Vision Transformer):** Exploring newer transformer-based backbones could offer performance gains, though they might require more data and computational resources. MMOCR is increasingly supporting these.
- **Data Augmentation:** MMOCR provides extensive data augmentation pipelines. Tailoring these specifically for inscriptions (e.g., simulating cracks, color variations, perspective distortions relevant to stone surfaces) could significantly boost robustness.
- **Loss Function Weights:** If the model struggles with a particular aspect (e.g., distinguishing text from non-text, or accurately delineating boundaries), adjusting the weights of different components in the DBNet++ loss function might help.
- **Cross-Dataset Evaluation:** Test the trained model on other inscription datasets (if available) or even general scene text datasets to understand its generalization capabilities.

---

4. Module 3: DeepFill Image Inpainting Model

overview:

This module details the fine-tuning of the DeepFill v2 model for image inpainting specifically on a custom dataset of segmented inscriptions. Image inpainting aims to plausibly fill in missing or corrupted regions of an image. For inscriptions, this could mean repairing cracks, filling areas obscured by lichen or damage, or even reconstructing missing characters or parts of characters. Fine-tuning a powerful pre-trained model like DeepFill v2 allows leveraging its learned knowledge of general image structures while adapting it to the specific textures, patterns, and characteristics of inscriptions.

Literature Context for Image Inpainting:

- **Traditional Methods:** Diffusion-based, patch-based (e.g., PatchMatch), and examplar-based methods were common. They work well for small gaps or textured regions but often fail on large missing areas or complex structures.
- **Deep Learning - CNN based:** Early deep learning approaches used CNNs to directly regress pixel values for missing regions, often resulting in blurry outputs.
- **Deep Learning - GAN based:** Generative Adversarial Networks (GANs) revolutionized inpainting by learning to generate realistic content.
    - **Context Encoders:** An early GAN approach with an encoder-decoder structure and an adversarial loss.
    - **Partial Convolutions:** Proposed using convolutions only on valid pixels, along with an automatic mask updating mechanism.
    - **Gated Convolutions (used in DeepFill v1/v2):** Learns a soft mask for each channel and location dynamically, offering more flexibility than partial convolutions. This helps handle irregular holes and diverse image content.
    - **Contextual Attention (used in DeepFill v1/v2):** Allows the model to borrow or copy feature information from distant spatial locations within the image (known valid regions) to fill missing patches. This is crucial for generating coherent structures and repeating patterns, which can be relevant for lines of text or decorative elements in inscriptions.
- **Loss Functions:** A combination of pixel-wise reconstruction loss (L1 or L2), perceptual loss (measuring differences in feature spaces of pre-trained classification networks like VGG), and adversarial loss is standard for achieving both fidelity and realism. WGAN-GP (Wasserstein GAN with Gradient Penalty) is often used for more stable training of the adversarial component.

DeepFill v2, with its gated convolutions and contextual attention mechanism, is a state-of-the-art model well-suited for inpainting large, irregular missing regions, making it a strong candidate for restoring damaged inscriptions.

Rationale for Fine-tuning and Dataset Strategy:

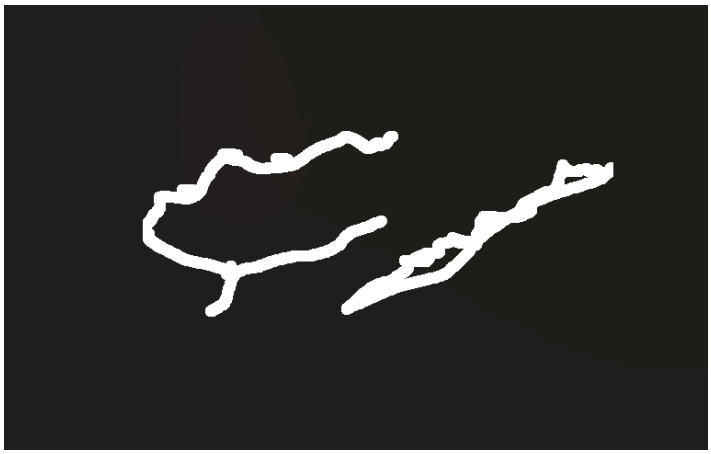
- **Pre-trained Base Model (`states_tf_celebahq.pth`):** The CelebA-HQ dataset contains high-resolution face images. While faces are different from inscriptions, pre-training on such a large and diverse dataset allows the model to learn fundamental concepts of image structure, texture, color, and coherence. This foundational knowledge is invaluable.
- **Fine-tuning on Custom Dataset:** Inscriptions have unique characteristics (stone textures, character shapes, specific types of degradation). Fine-tuning adapts the general knowledge from CelebA-HQ to these specific features. This domain adaptation is crucial for achieving high-quality results on inscriptions.
- **Mask Generation (Random Rectangular Masks 20-40%):**

    - **Artificial Masks:** Since inscriptions might not come with "ground truth" undamaged versions, creating artificial masks on existing images is a common strategy for training inpainting models.
    - **Random Rectangles:** While simple, they provide a good starting point for training the model to fill diverse missing regions. The variable size (20-40%) ensures the model learns to handle both smaller and larger gaps.
    - **Future Work Implication:** The "Dynamic Masks: Incorporate irregular free-form masks" is a key future step, as real-world damage is rarely perfectly rectangular.

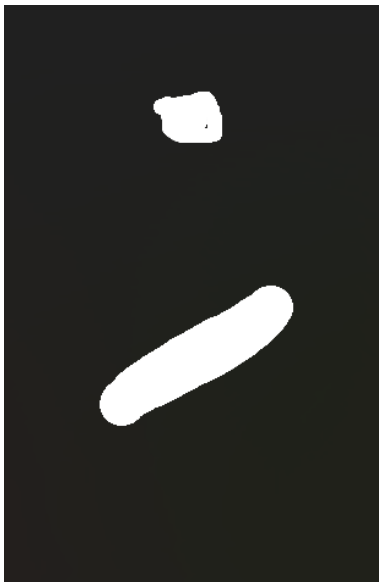
Understanding Losses and Evaluation:

- **Losses:**
    - **Reconstruction (L1):** `loss_rec = L1_loss(outputs, imgs, masks)` measures the pixel-wise absolute difference between the inpainted image and the original image (in the masked regions). It encourages pixel-accurate reconstructions. L1 is often preferred over L2 as it tends to produce less blurry results.
    - **Adversarial (WGAN-GP):** `loss_adv = adversarial_loss(...)` forces the generator (the inpainting model) to produce images that are indistinguishable from real images for a discriminator network. This pushes the output towards realism. WGAN-GP helps stabilize GAN training. The weight of 0.1 suggests it's a significant but not dominant component of the total loss.
    - **Perceptual (VGG-based):** `loss_perc = perceptual_loss(...)` measures differences in high-level feature representations extracted by a pre-trained network (like VGG). This helps ensure that the inpainted regions are semantically consistent with the surrounding content, even if not pixel-perfect. The small weight (0.01) suggests it's used for fine-tuning the perceptual quality.
- **Optimizer (**`Adam(beta1=0.5, beta2=0.999)`**):** These beta values are common in GAN training.
- **Evaluation Metrics:**
    - **PSNR (Peak Signal-to-Noise Ratio):** Measures the ratio between the maximum possible power of a signal and the power of corrupting noise.[1] Higher PSNR generally indicates better reconstruction quality in terms of pixel fidelity. 28.45 dB is a respectable value.
    - **SSIM (Structural Similarity Index Measure):** Measures the similarity between two images based on luminance, contrast, and structure. It often

aligns better with human perception of image quality than PSNR. 0.912 (out of 1) is a good score.



Original Mask Inpainted



Original Mask
Inpainted

Future Work :

- **Longer Fine-tuning (10k iterations):** Especially with a small custom dataset, more iterations can allow the model to better adapt to the nuances of the inscription data, provided overfitting is monitored and managed.
- **Dynamic Masks:** This is critical. Using tools to generate more realistic, free-form masks (e.g., based on typical crack patterns, flaking, etc.) or even using masks derived from actual damaged examples will lead to a model that performs better on real-world tasks.
- **Perceptual Metrics and Loss Weighting:** Systematically experimenting with the weights of `loss_adv` and `loss_perc` can fine-tune the balance between pixel accuracy, overall realism, and semantic coherence. User studies might be needed to determine optimal perceptual quality.
- **Domain Adaptation (Soft-modulation):** Instead of just fine-tuning all layers, explore techniques like AdaIN (Adaptive Instance Normalization) or other feature modulation strategies to more explicitly adapt the pre-trained style/features to the inscription domain without catastrophically forgetting the general knowledge.
- **Application-Specific Evaluation:** Evaluate inpainted results not just by PSNR/SSIM, but by how much they improve downstream tasks (e.g., does inpainting improve character recognition rates on damaged characters?).

---

5. Module 4: Classification using CRNN

overview:

This module focuses on the classification of segmented inscription characters into one of 170 classes using a Convolutional Recurrent Neural Network (CRNN). This is a core task in deciphering and analyzing inscriptions. The model architecture cleverly combines Convolutional Neural Networks (CNNs) for robust spatial feature extraction from the character images and a Bidirectional Long Short-Term Memory (LSTM) network to capture sequential or contextual patterns, even if the input is a single character image (the LSTM here might be learning to model stroke sequences or implicit features that have a sequential nature).

Literature Context for Character Recognition:

- **Traditional OCR:** Relied on hand-crafted features and simpler classifiers (e.g., SVMs). Often struggled with font variations and noise.
- **Deep Learning - CNNs:** For isolated character recognition, CNNs (e.g., LeNet, AlexNet, VGG, ResNet) became the standard, automatically learning hierarchical features directly from pixel data. Models like the one described use a series of convolutional blocks for this purpose.

- **Deep Learning - RNNs for Sequences:** For recognizing lines of text (Handwritten Text Recognition - HTR), RNNs, particularly LSTMs and GRUs, are used to model the sequential dependencies between characters.
- **CRNNs (Convolutional Recurrent Neural Networks):** This architecture, popularized by Shi et al. (2015) for scene text recognition, is a powerful combination:
    - **CNN Backbone:** Extracts a sequence of feature vectors from the input image (each vector representing a "receptive field" or frame).
    - **RNN Layer (LSTM/GRU):** Processes this sequence of feature vectors to capture contextual information and make predictions. Bidirectional LSTMs are common as they consider both past and future context.
    - **Transcription Layer:** Converts per-frame predictions from the RNN into a final label sequence (often using Connectionist Temporal Classification - CTC loss for sequence-to-sequence tasks). In *this specific CRNN architecture*, the LSTM is applied *after* `GlobalAveragePooling2D` and `Reshape`. This means the LSTM is not processing a sequence of spatial features across the width of an image (as in typical line HTR). Instead, `GlobalAveragePooling2D` collapses spatial dimensions into a single feature vector per image. `Reshape` then makes this compatible with LSTM input (a sequence of length 1). The Bidirectional LSTM here might be learning complex relationships within this global feature vector or acting as a powerful classifier on top of the CNN features. For single characters, its "sequential" role is more abstract than in HTR line recognition.

Architectural Deep Dive and Rationale:

- **Input Layer (`input_shape=(64, 64, 3)`):** Expects 64x64 pixel RGB images. The size is a common trade-off for character recognition, retaining enough detail while being computationally manageable.
- **Convolutional Blocks (1-4):**
    - `Conv2D` (32, 64, 128, 256 filters, (3,3) kernel, 'relu' activation, 'same' padding): Standard convolutional layers to extract increasingly complex features. Filter counts increase with depth, a common pattern. (3,3) kernels are efficient. 'relu' is a standard activation. 'same' padding ensures output feature maps have the same spatial dimensions as input (before pooling).
    - `BatchNormalization()`: Stabilizes training, allows higher learning rates, and acts as a regularizer. Applied after convolution and before/after activation (practice varies, here it's after).

    - `MaxPooling2D(pool_size=(2, 2))`: Reduces spatial dimensions by half, making the network more robust to small translations and reducing computational load.
- `GlobalAveragePooling2D()`**:** Instead of flattening the final feature maps (which can lead to a huge number of parameters), GAP averages each feature map into a single value. This drastically reduces parameters, helps prevent overfitting, and is more robust to spatial variations.
- `Reshape(target_shape=(-1, x.shape[-1]))`**:** The output of GAP is `(batch_size, channels)`. The LSTM expects a sequence input of shape `(batch_size, timesteps, features)`. Reshaping to `(-1, x.shape[-1])` likely means `(batch_size, 1, channels)`, treating the single global feature vector as a sequence of length 1.
- `Bidirectional(LSTM(256, return_sequences=False))`**:**
    - `LSTM(256)`: An LSTM layer with 256 units, capable of learning long-range dependencies (though here, on a single "timestep" from the global features).
    - `Bidirectional`: Processes the input sequence (of length 1) in both forward and backward directions. For a single timestep, this means two separate LSTMs (forward and backward) process the input, and their outputs are concatenated or summed. This doubles the parameters for the LSTM part but can provide richer representations.
    - `return_sequences=False`: Since this is the final LSTM layer before classification and we need one output per input image, `False` ensures it outputs only the final state.
- `Dropout(0.5)`**:** Regularization technique to prevent overfitting by randomly setting a fraction of input units to 0 during training. 0.5 is a common and relatively high dropout rate.
- **Output Layer (**`Dense(num_classes, activation='softmax')`**):** A fully connected layer that maps the LSTM output to the 170 character classes. `softmax` activation provides a probability distribution over the classes.
- **Compiler (**`Adam`**,** `categorical_crossentropy`**):**
    - `Adam`: An efficient and popular optimizer.
    - `categorical_crossentropy`: Standard loss function for multi-class classification problems where classes are mutually exclusive and data is one-hot encoded.

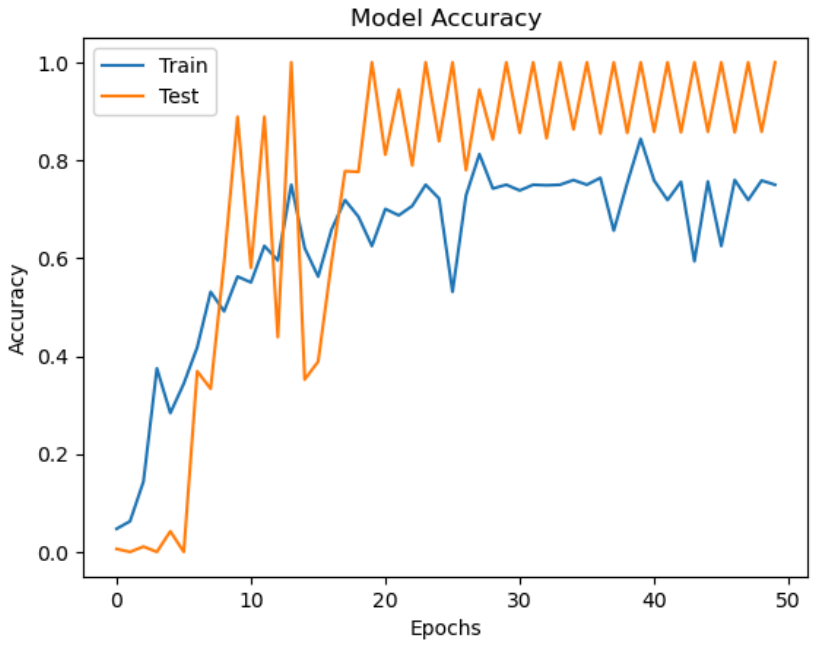
Data Preparation and Augmentation Strategy:

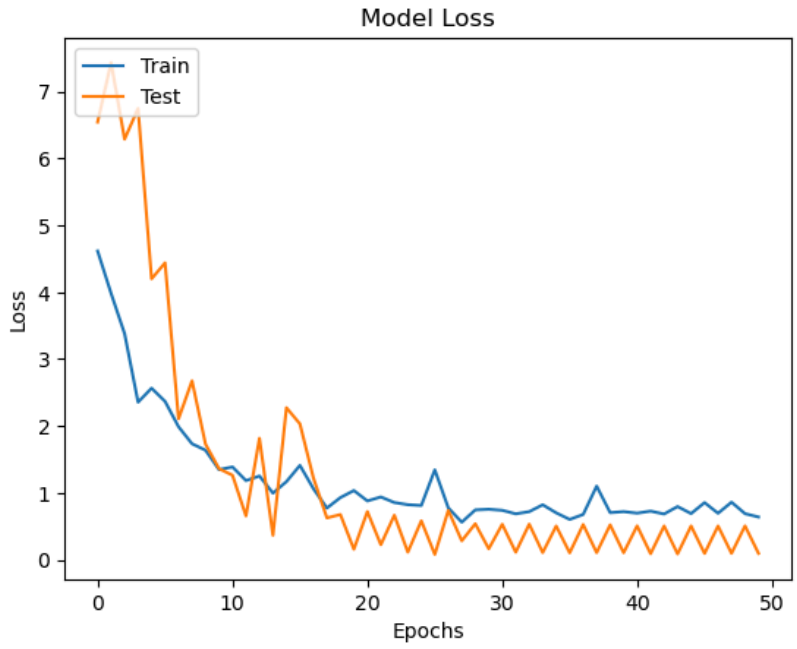
- **Image Size (64x64), Batch Size (32):** Standard choices.
- `ImageDataGenerator`**:** Keras utility for loading images and applying real-time data augmentation.

- **Training Data Augmentation:**
    - `rescale=1./255`: Normalizes pixel values to [0, 1].
    - `rotation_range=20`, `width_shift_range=0.2`, `height_shift_range=0.2`, `shear_range=0.2`, `zoom_range=0.2`, `horizontal_flip=True`:[2] These are excellent augmentation choices to make the model robust to variations in character appearance, orientation, and minor translations. For characters, `horizontal_flip` might be useful for some scripts but detrimental for others where mirrored characters mean different things (e.g., 'b' vs 'd'). This should be considered based on the specific script. `fill_mode='nearest'` handles pixels outside boundaries during transformations.
- **Test Data Preprocessing (`rescale=1./255` only):** Crucially, no augmentation is applied to the test set, only rescaling, to ensure evaluation reflects performance on unmanipulated data.
- `flow_from_directory`**:** Efficiently loads images from directories where each subdirectory is a class. `class_mode='categorical'` ensures labels are one-hot encoded. `shuffle=True` for training is important.

Training, Evaluation, and Future Improvements (Elaborated):

- **Epochs (50), Callbacks (`ReduceLROnPlateau`):** A good starting point for training length. `ReduceLROnPlateau` is a useful callback to adjust learning rate dynamically, helping to fine-tune convergence when validation loss stagnates.
- **Results (Training Acc: 72.25%, Test Acc: 72.72% after 26 epochs):** The test accuracy being slightly higher or very close to training accuracy is good; it suggests the model is not significantly overfitting, possibly due to effective regularization (Batch Norm, Dropout, Augmentation) and the early stopping point.





- **Future Improvements (Elaborated):**
  - `EarlyStopping` **and** `ModelCheckpoint`**:** These are essential. `EarlyStopping` prevents overfitting and saves training time. `ModelCheckpoint` saves the best model based on `val_loss` or `val_accuracy`.
  - **Optimizers and Learning Rate Schedules:** Experiment with `AdamW`, `RMSprop`, or `SGD` with momentum. Try different initial LRs or more aggressive decay schedules like cosine annealing.

- **Dropout in Conv Blocks:** Adding `SpatialDropout2D` after pooling layers in the CNN blocks can further regularize and improve generalization, as it drops entire feature maps rather than individual activations.
- **Hyperparameter Tuning:** Use tools like KerasTuner, Optuna, or Ray Tune to systematically search for optimal filter sizes, LSTM units, dropout rates, learning rates, and batch sizes.
- **Transfer Learning (Pre-trained CNN Backbones):** This is a very powerful technique. Replace the custom CNN blocks with a pre-trained backbone (e.g., ResNet50, EfficientNet, MobileNetV2) initialized with weights from ImageNet. Then, fine-tune this backbone or just train the newly added LSTM and dense layers. This can significantly boost performance, especially with limited inscription character data.
- **Attention Mechanisms:** Consider adding attention mechanisms after the CNN feature extraction or within the LSTM to allow the model to focus on more salient parts of the character image.
- **CTC Loss (if applicable):** If the model were adapted for sequences of characters (like words or short phrases) instead of isolated characters, Connectionist Temporal Classification (CTC) loss would be the standard. For single character classification, `categorical_crossentropy` is appropriate.
- **Class Imbalance:** With 170 classes, there might be class imbalance. Investigate techniques like weighted loss functions or oversampling/undersampling if this is an issue.

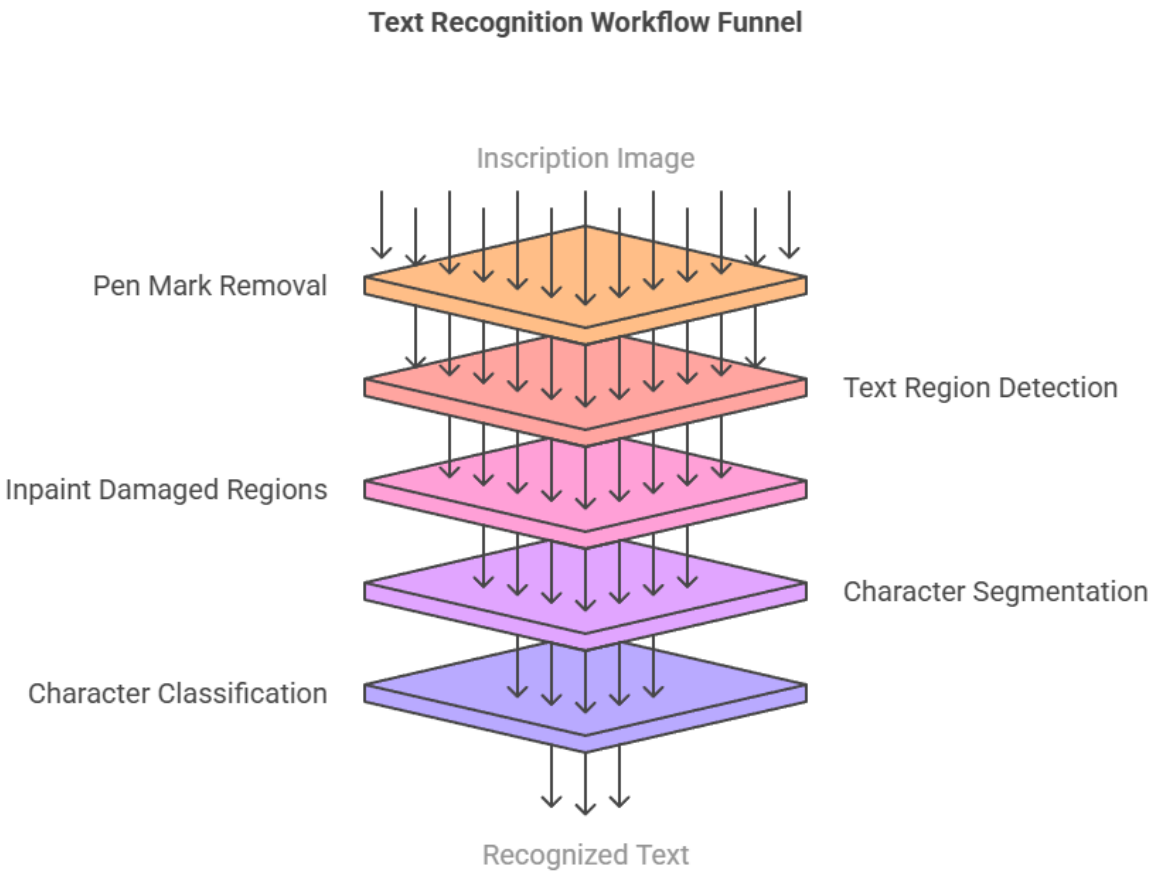
---

6. Overall Project Pipeline and Inter-module Synergy

While each documented module addresses a specific task, they can be envisioned as components of a larger, integrated pipeline for comprehensive inscription analysis. The potential synergy is as follows:

1. **Initial Image Acquisition & Optional Preprocessing:**
    - An image of an inscription is acquired.
    - **(Optional) Pen Mark Extraction (Module 1):** If the inscription image contains modern annotations or markings that need to be either analyzed separately or removed to avoid confusing subsequent models, the "Pen Mark Extraction Script" could be used. The outputted mask could identify these regions.
2. **Text Detection (Module 2 - MMOCR):**

    - The (potentially cleaned) inscription image is fed into the fine-tuned MMOCR (DBNet++) model.
    - This model detects the locations of text lines or individual words/characters, outputting bounding boxes or polygons.
3. **Image Inpainting (Module 3 - DeepFill v2) - Conditional Application:**
    - If detected text regions (from MMOCR) are found to be damaged, occluded, or incomplete (e.g., parts of characters in a crack), these specific regions can be targeted for inpainting.
    - The DeepFill v2 model, fine-tuned on inscriptions, would take the image and a mask (identifying the damaged part within the detected text region) as input.
    - It would output an inpainted version of that region, aiming to restore missing content plausibly. This could improve the legibility for human experts or the recognizability for automated systems.
4. **Character Segmentation (Implicit Step, not fully documented but necessary for CRNN):**
    - The detected text regions (either original or inpainted) need to be further segmented into individual character images. This could involve traditional contour analysis, projection profiles, or more advanced segmentation models if characters are touching or script is complex.
    - Each segmented character image would be resized to match the input requirements of the CRNN model (e.g., 64x64 pixels).
5. **Character Classification (Module 4 - CRNN):**
    - The segmented (and potentially inpainted and resized) character images are fed into the CRNN model.
    - The CRNN classifies each character, assigning it one of the 170 predefined labels.
6. **Post-processing and Interpretation:**
    - The sequence of classified characters from a detected text line can be assembled to form words or phrases.
    - This output can then be used for further linguistic analysis, database indexing, comparison with other inscriptions, etc.



This pipeline demonstrates how specialized models can contribute to a complex analysis task. The output of one stage becomes the input for the next, with decision points (e.g., whether to inpaint) potentially integrated based on quality metrics or human intervention.

---

7. General Conclusion and Broader Future Directions

The documented modules represent significant steps towards building a comprehensive computational toolkit for the analysis of stone inscriptions. The use of sophisticated deep learning models like CRNNs for classification, DeepFill v2 for inpainting, and DBNet++ (via MMOCR) for text detection, all tailored or fine-tuned for inscription data, demonstrates a commitment to leveraging cutting-edge AI for epigraphic research. The foundational image processing script for pen mark extraction addresses a practical need in handling annotated artifacts.

Broader Future Directions for the Overall Project:

1. **End-to-End System Development:** Integrate these modules into a seamless end-to-end pipeline, potentially with a user interface for epigraphers to upload images, trigger analyses, and review results.
2. **Dataset Expansion and Curation:** Continuously expand the custom datasets for training and fine-tuning. Consider collaborative efforts to build larger, more diverse, and richly annotated inscription datasets. Standardized annotation formats would be beneficial.
3. **Uncertainty Quantification:** For classification and inpainting, provide confidence scores or uncertainty estimates alongside predictions. This helps users understand the reliability of the automated results.
4. **Human-in-the-Loop Systems:** Develop interactive systems where human experts can easily correct errors made by the models (e.g., incorrect detections, classifications, or implausible inpainting). These corrections can then be used to further fine-tune the models (active learning).
5. **Multi-modal Analysis:** If 3D scans or RTI data are available for inscriptions, explore fusing information from these modalities with 2D image analysis to improve robustness, especially for highly degraded inscriptions.
6. **Historical Script Evolution Modeling:** For scripts that evolve over time, explore models that can account for temporal variations in character shapes or even assist in dating inscriptions.
7. **Cross-Lingual and Cross-Script Adaptation:** Investigate transfer learning techniques to adapt models trained on one script or language to others with limited data.
8. **Deployment and Accessibility:** Make these tools accessible to the wider epigraphic community, perhaps through web platforms, plugins for existing digital humanities tools, or open-source code releases with clear documentation.

By addressing these challenges and pursuing these future directions, the project can significantly contribute to the preservation, understanding, and accessibility of invaluable historical knowledge contained within inscriptions.

References

- Zhang, K., Jiang, X., Madadi, M., Chen, L., Savitz, S., & Shams, S. (2021). DBNet: A novel deep learning framework for mechanical ventilation prediction using electronic health records. In *Proceedings of the 12th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics (BCB '21)* (Art. 9, pp. 1–8). Association for Computing Machinery. https://doi.org/10.1145/3459930.3469551

- Kuang, Z., Sun, H., Li, Z., Yue, X., Lin, T. H., Chen, J., Wei, H., Zhu, Y., Gao, T., Zhang, W., Chen, K., Zhang, W., & Lin, D. (2021). MMOCR: A comprehensive toolbox for text detection, recognition and understanding. In *Proceedings of the 29th ACM International Conference on Multimedia (MM '21)* (pp. 3791–3794). Association for Computing Machinery. https://doi.org/10.1145/3474085.3478328
- Xuan, Z., Yang, Z., Lei, C., Yu, Z., Jin, Z., Luo, Q., Zheng, W., Guo, Y., Zhu, S., Wang, N., Chen, Z., & Ding, Y. (2024). Image inpainting for ECEI based on DeepFillv2 model. *Fusion Engineering and Design, 202*, Article 114378. https://doi.org/10.1016/j.fusengdes.2024.114378
- Fu, X., Ch'ng, E., Aickelin, U., & See, S. (2017). CRNN: A joint neural network for redundancy detection. In *2017 IEEE International Conference on Smart Computing (SMARTCOMP)* (pp. 1–8). IEEE. https://doi.org/10.1109/SMARTCOMP.2017.7946996
- Vijayalakshmi, R., & Gnanasekar, J. M. (2022). A review on character recognition and information retrieval from ancient inscriptions. In *2022 8th International Conference on Smart Structures and Systems (ICSSS)* (pp. 1–7). IEEE. https://doi.org/10.1109/ICSSS54381.2022.9782241
- Manigandan, T., Vidhya, V., Dhanalakshmi, V., & Nirmala, B. (2017). Tamil character recognition from ancient epigraphical inscription using OCR and NLP. In *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)* (pp. 1008–1011). IEEE. https://doi.org/10.1109/ICECDS.2017.8389589
- Zhang, H., Qi, Y., Xue, X., & Nan, Y. (2021). Ancient stone inscription image denoising and inpainting methods based on deep neural networks. *Discrete Dynamics in Nature and Society, 2021*, Article 7675611. https://doi.org/10.1155/2021/7675611