Supplementary for Unveiling Text in Challenging Stone Inscriptions: A Character-Context-Aware Patching Strategy for Binarization*

Pratyush Jena, Amal Joseph, Arnav Sharma, Ravi Kiran Sarvadevabhatla
(pratyush.jena,amal.joseph,arnav.sharma)@research.iiit.ac.in,ravi.kiran@iiit.ac.in
Center for Visual Information Technology, International Institute of Information Technology, Hyderabad
Hyderabad, Telangana, India

ACM Reference Format:
Pratyush Jena, Amal Joseph, Arnav Sharma, Ravi Kiran Sarvadevabhatla. 2025. *Supplementary for* Unveiling Text in Challenging Stone Inscriptions: A Character-Context-Aware Patching Strategy for Binarization. In *Proceedings of 16th Indian Conference on Computer Vision, Graphics and Image Processing (ICVGIP'25).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 SELF-REFINING INFERENCE PIPELINE

During inference, our goal is to apply the learned binarization model to new, unseen stone inscription images. To ensure the patches are scaled appropriately to the character heights and to replicate the patching strategy used during training, we introduce a two-stage, self-refining inference pipeline. This pipeline first generates a coarse prediction of the text regions and then uses this information to guide a more precise, context-aware binarization. Refer to Algorithm. 1 for a detailed description of our method.

*Produces the permission block, and copyright information

ICVGIP'25, December 2025, Mandi, India

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Algorithm 1 Detailed Self-Refining Inference Pipeline (Algorithm S1)

Hyperparameters:

$S \leftarrow \{256, 384, 512, 768\}$, $k_{\text{range}} \leftarrow [4, 12]$,

$N_{\min} \leftarrow 50$, $N_{\max} \leftarrow 400$, $N_{bg\_max} \leftarrow 150$, MODEL_INPUT_SIZE $\leftarrow (512, 512)$, $\varepsilon \leftarrow 10^{-8}$.

function RunInference(I, M)
  $(H, W) \leftarrow \text{size}(I)$ ▷ **Stage 1: Initial Prediction (Coarse Map Generation)**
  $P_{\text{pyr}} \leftarrow 0^{|S| \times H \times W}$ ▷ Initialize prediction maps for each scale
  for each $s \in S$ **do**
    $(\text{Patches}, \text{Locs}) \leftarrow \text{SlidingWindow}(I, s, s/2)$
    Resized $\leftarrow$ Resize(Patches, MODEL_INPUT_SIZE) ▷ Resize patches to fit model's input size
    Preds $\leftarrow M(\text{Resized})$ ▷ Run inference on the batch of patches
    $P_{\text{pyr}}[s] \leftarrow \text{PlacePredictionsOnMap}(\text{Preds}, \text{Locs}, (H, W))$ ▷ Reassemble patch predictions onto a full-size map
  end for
  $P_{\text{coarse}} \leftarrow \max_{\text{pixel}}(P_{\text{pyr}})$ ▷ Fuse predictions across scales by taking the max probability per pixel
  $B_{\text{pseudo}} \leftarrow (P_{\text{coarse}} > 0.5)$ ▷ Create a binary pseudo-ground truth mask for the next stage
  $h_{cc} \leftarrow \text{CalcAvgIQRHeight}(B_{\text{pseudo}})$ ▷ **Stage 2: Context-Aware Refinement**
  $(\text{Patches}', \text{Locs}') \leftarrow \text{ContextAwarePatch}(I, B_{\text{pseudo}}, h_{cc})$ ▷ Generate adaptively scaled patches
  $\text{A} \leftarrow 0^{H \times W}$, $\text{C} \leftarrow 0^{H \times W}$ ▷ Initialize accumulators for predictions and counts
  Resized′ $\leftarrow$ Resize(Patches′, MODEL_INPUT_SIZE)
  $\text{Refined} \leftarrow M(\text{Resized}')$
  for each $(p, \ell) \in \text{zip}(\text{Refined}, \text{Locs}')$ **do**
    $p_{\text{orig}} \leftarrow \text{Resize}(p, \text{size}(\ell))$ ▷ Resize prediction back to its original patch size before merging
    $\text{A}[\ell] \mathrel{+}= p_{\text{orig}}$ ▷ Add prediction logits to the accumulator map
    $\text{C}[\ell] \mathrel{+}= 1$ ▷ Increment the count for overlapping pixels
  end for
  $P_{\text{final}} \leftarrow A \oslash (C + \varepsilon)$ ▷ Average overlapping predictions element wise to create a smooth, seamless final map
  $B_{\text{final}} \leftarrow (P_{\text{final}} > 0.5)$
  return B_{final}
end function

---

function CalcAvgIQRHeight(B_{mask})
  $C \leftarrow \text{FindConnectedComponents}(B_{\text{mask}})$
  $H \leftarrow \{c.\text{height} \mid c \in C\}$
  $(Q_1, Q_3) \leftarrow \text{Quartiles}(H, [25, 75])$ ▷ Calculate first and third quartiles of component heights
  $\text{Filt} \leftarrow \{h \in H \mid Q_1 \le h \le Q_3\}$ ▷ Filter out outliers to get a robust set of primary character heights
  return mean(Filt)
end function

---

function ContextAwarePatch($I, B_{\text{pseudo}}, h_{cc}$)
  $\text{BBoxMask} \leftarrow \text{CreateBBoxMask}(B_{\text{pseudo}})$ ▷ Start with a mask of all component bounding boxes
  $K_1 \leftarrow ((0.3h_{cc}), (0.9h_{cc}))$ ▷ Define adaptive kernels based on character height
  $K_2 \leftarrow ((0.9h_{cc}), (0.3h_{cc}))$
  $\text{Mask}_1 \leftarrow \text{MorphoDilate}(\text{BBoxMask}, K_1)$ ▷ Perform two-stage dilation to merge text lines and words
  $\text{Mask}_{fg} \leftarrow \text{MorphoDilate}(\text{Mask}_1, K_2)$
  $\text{Mask}_{bg} \leftarrow \neg\, \text{Mask}_{fg}$ ▷ Background is the inverse of the final foreground mask
  $C_v \leftarrow \text{FilterByIQR}(\text{FindConnectedComponents}(B_{\text{pseudo}}))$ ▷ Count only the valid, non-outlier characters
  $N'_{fg} \leftarrow |C_v| \times R_{\text{base}}$
  $N_{fg} \leftarrow \max(N_{\min}, \min(N'_{fg}, N_{\max}))$ ▷ Clamp the number of foreground patches to prevent imbalance
  $r_{bg} \leftarrow \frac{\sum \text{Mask}_{bg}}{H \times W}$ ▷ Calculate background area ratio
  $N_{bg} \leftarrow (r_{bg} \times N_{bg\_max})$ ▷ Set background patch count proportionally to its area
  $A_{fg} \leftarrow \text{Mask}_{fg}$, $A_{bg} \leftarrow \text{Mask}_{bg}$ ▷ Step C: Sample anchor points and extract patches
  $\text{Anch}_{fg} \leftarrow \text{UniformSample}(A_{fg}, N_{fg})$ ▷ Sample anchor points uniformly from foreground
  $\text{Anch}_{bg} \leftarrow \text{UniformSample}(A_{bg}, N_{bg})$ ▷ Sample anchor points uniformly from background
  $\text{AllAnch} \leftarrow \text{Anch}_{fg} \cup \text{Anch}_{bg}$
  for each $(x, y) \in \text{AllAnch}$ **do**
    $k \leftarrow \text{RandInt}(k_{\text{range}})$ ▷ Randomly select a scale multiplier for data augmentation
    $L \leftarrow \text{round}(k \cdot h_{cc})$ ▷ Determine adaptive patch size based on h_{cc} and k
    patch $\leftarrow$ ExtractPatch(I, $(x - L/2,\ y - L/2)$, (L, L), reflect) ▷ Extract patch with reflect padding at borders
    append `patch` to `OutputPatches` and its location to `OutputLocs`
  end for
  return (OutputPatches, OutputLocs)
end function