CrispyMcMark

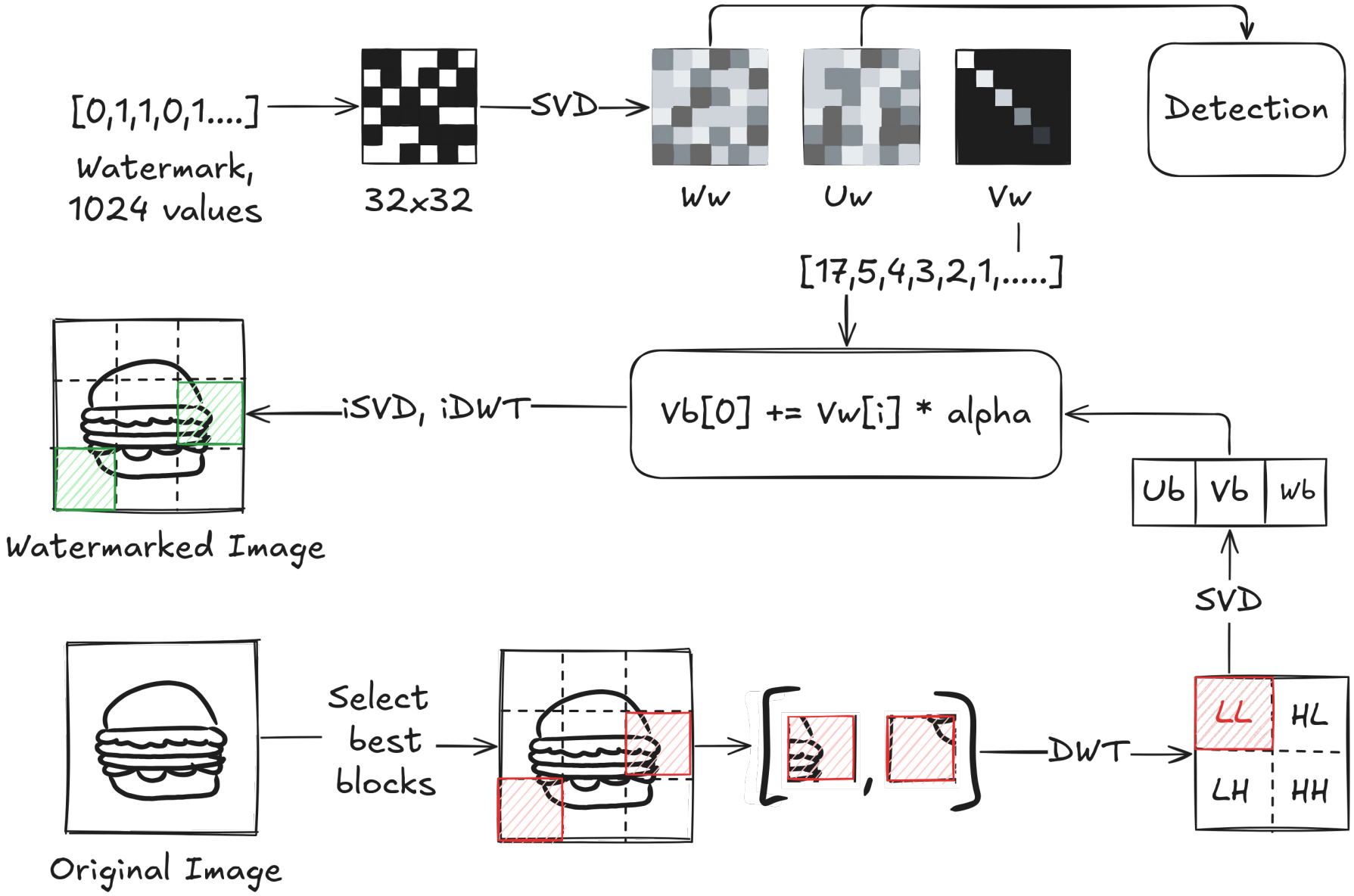
- Elia Gatti
- Pietro Ventrucci
- Filippo Marcon

How we reached the final implementation

- initial DWT-SVD implementation
- multiplicative embedding
- initial block implementation
- final block implementation

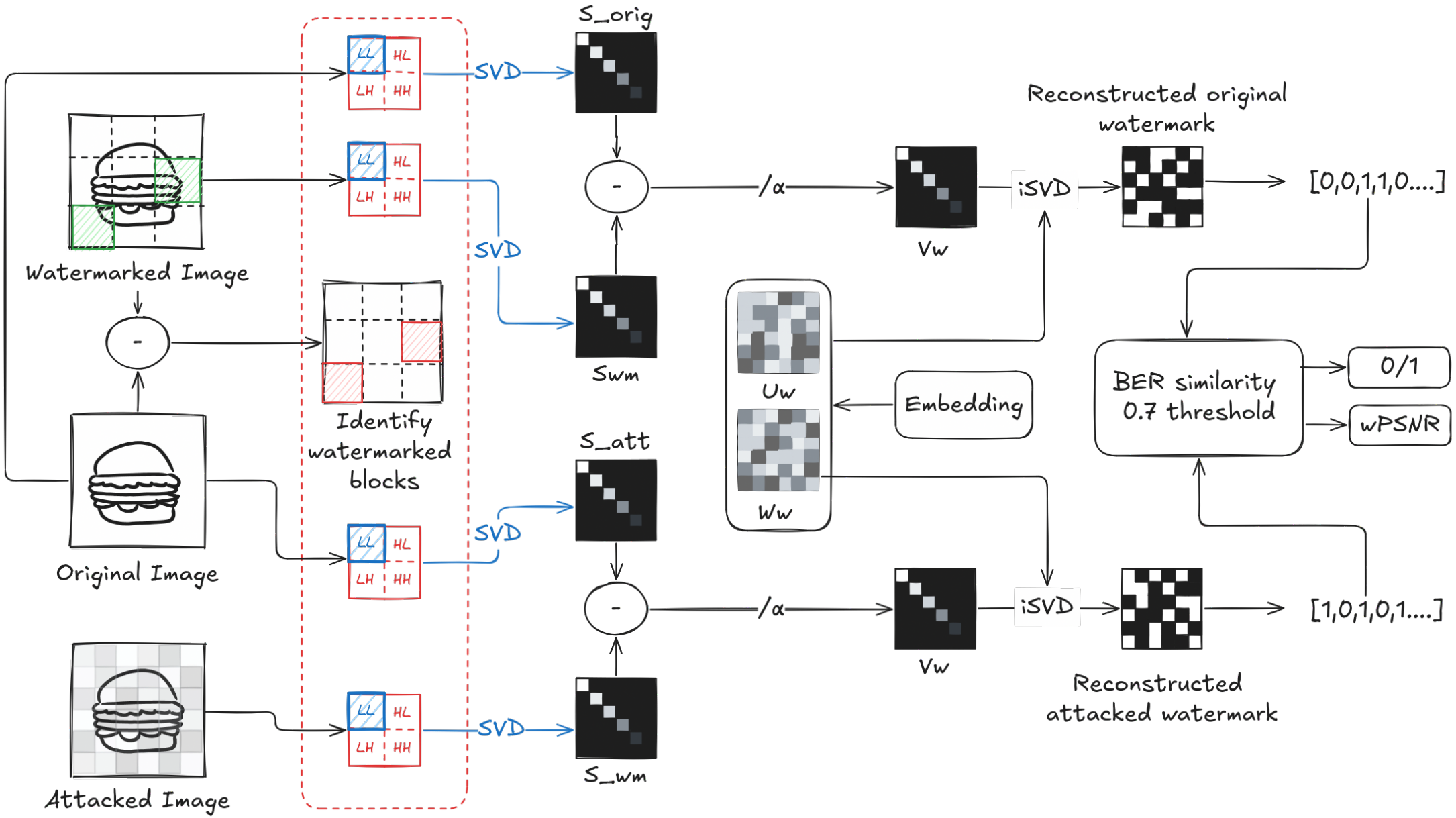
Embedding

- reshape watermark to 32x32 and take $U_w, V_w, W_w = SVD(watermark)$
- U_w, W_w hardcoded in the detection, V_w are the singular values
- choose x square blocks
- forall $i < x$:
  - take LL_b of the DWT of $blocks[i]$
  - compute the singular values V_b of its LL
  - embed $V[i]$ into the first singular value($V_b[0]$)
  - inverse the first two steps to reconstruct the block, and put it back into the image


[0,1,1,0,1....]
Watermark,
1024 values
32x32
SVD
Ww
Uw
Vw
Detection
[17,5,4,3,2,1,.....]
Vb[0] += Vw[i] * alpha
iSVD, iDWT
Watermarked Image
Ub
Vb
Wb
SVD
Original Image
Select
best
blocks
DWT
LL
HL
LH
HH

Detection

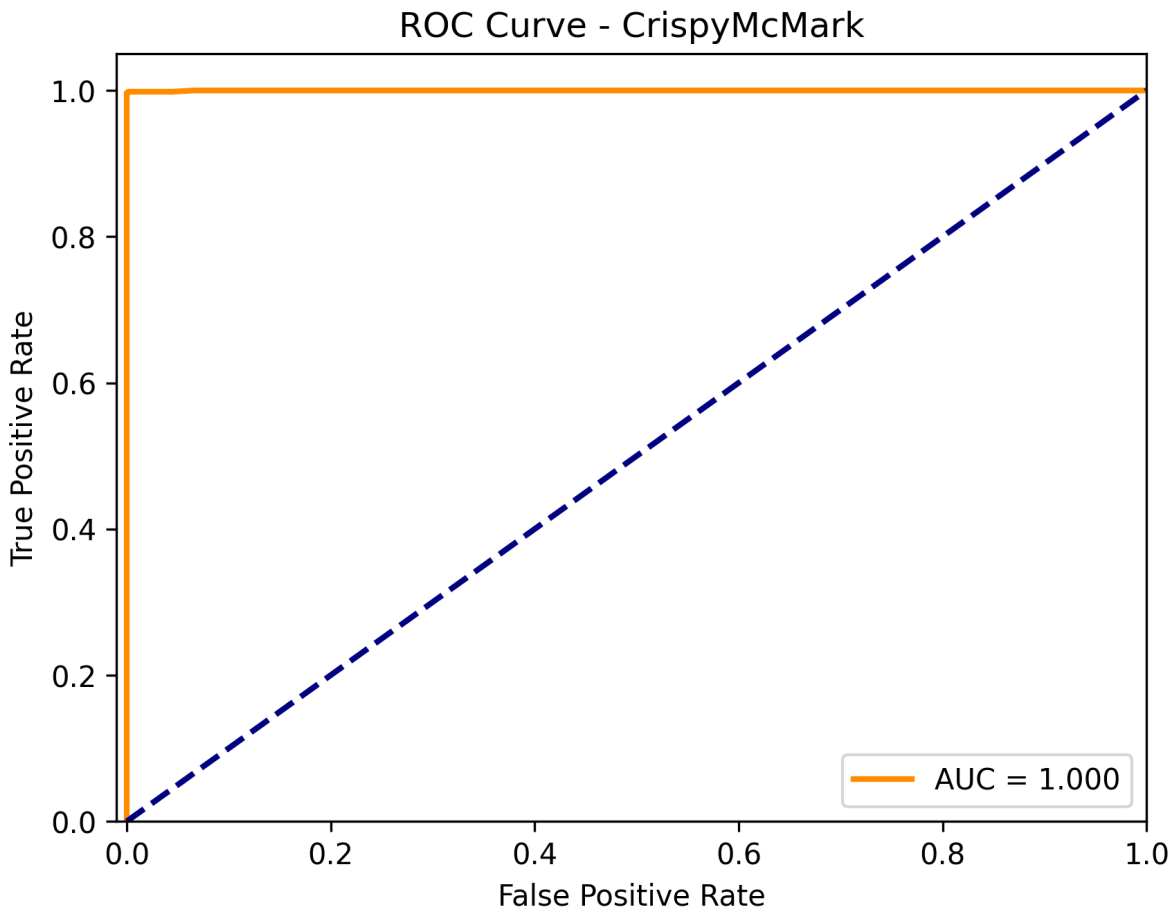
- use watermarked - original image to find x watermarked blocks
- attacked watermark = difference between singular values of attacked blocks and original blocks
- original watermark = difference between singular values of watermarked blocks and original blocks
- detection compares the two extracted watermarks using threshold


Watermarked Image
Original Image
Attacked Image
Identify watermarked blocks
LL
HL
LH
HH
SVD
S_orig
Swm
S_att
S_wm
-
/α
Vw
iSVD
Reconstructed original watermark
[0,0,1,1,0....]
Uw
Ww
Embedding
BER similarity 0.7 threshold
0/1
wPSNR
[1,0,1,0,1....]
Reconstructed attacked watermark

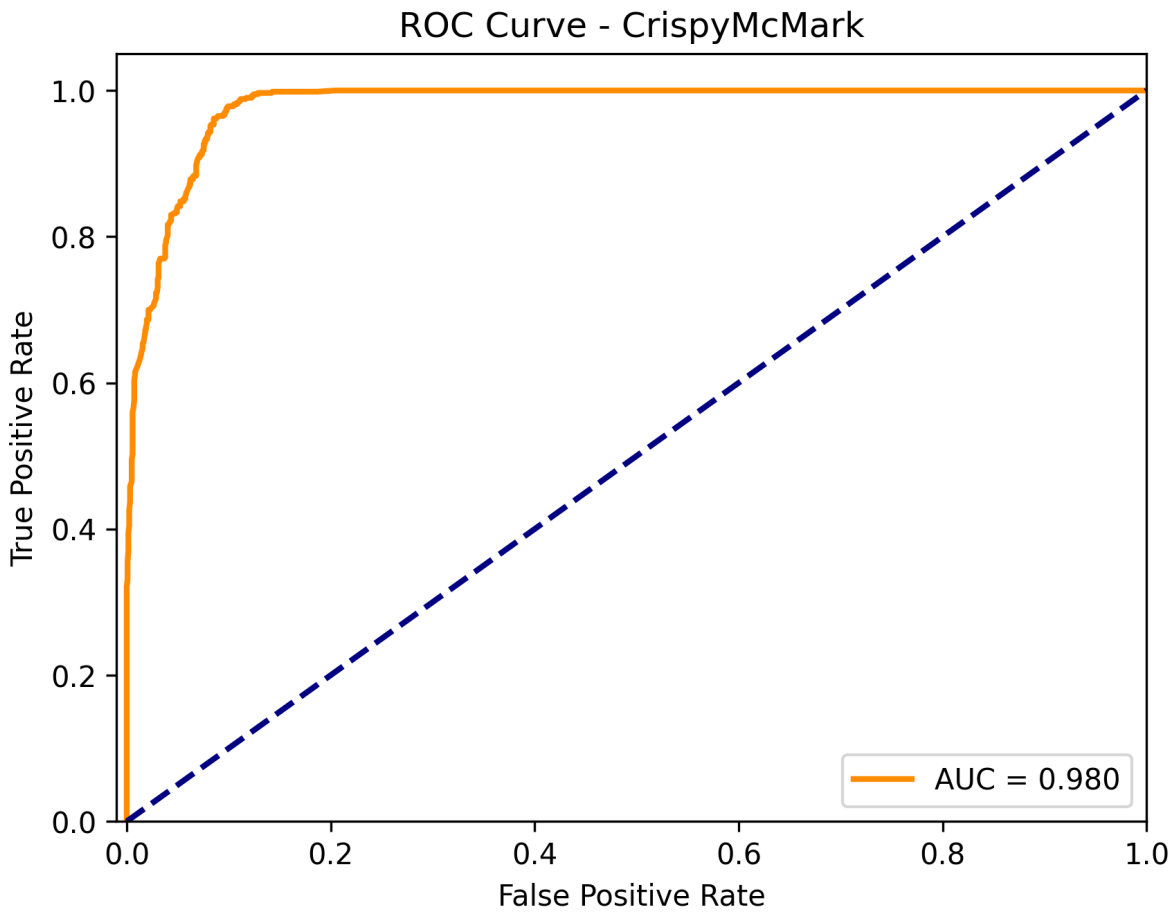
Current Limitations

- Embedding quality performed better on images with high entropy zones
- On low entropy images embedding was more visible
- Not enough time to refine the design and try different techniques
- Understanding how the algorithm performed/finding bugs based on the ROC function.

ROC

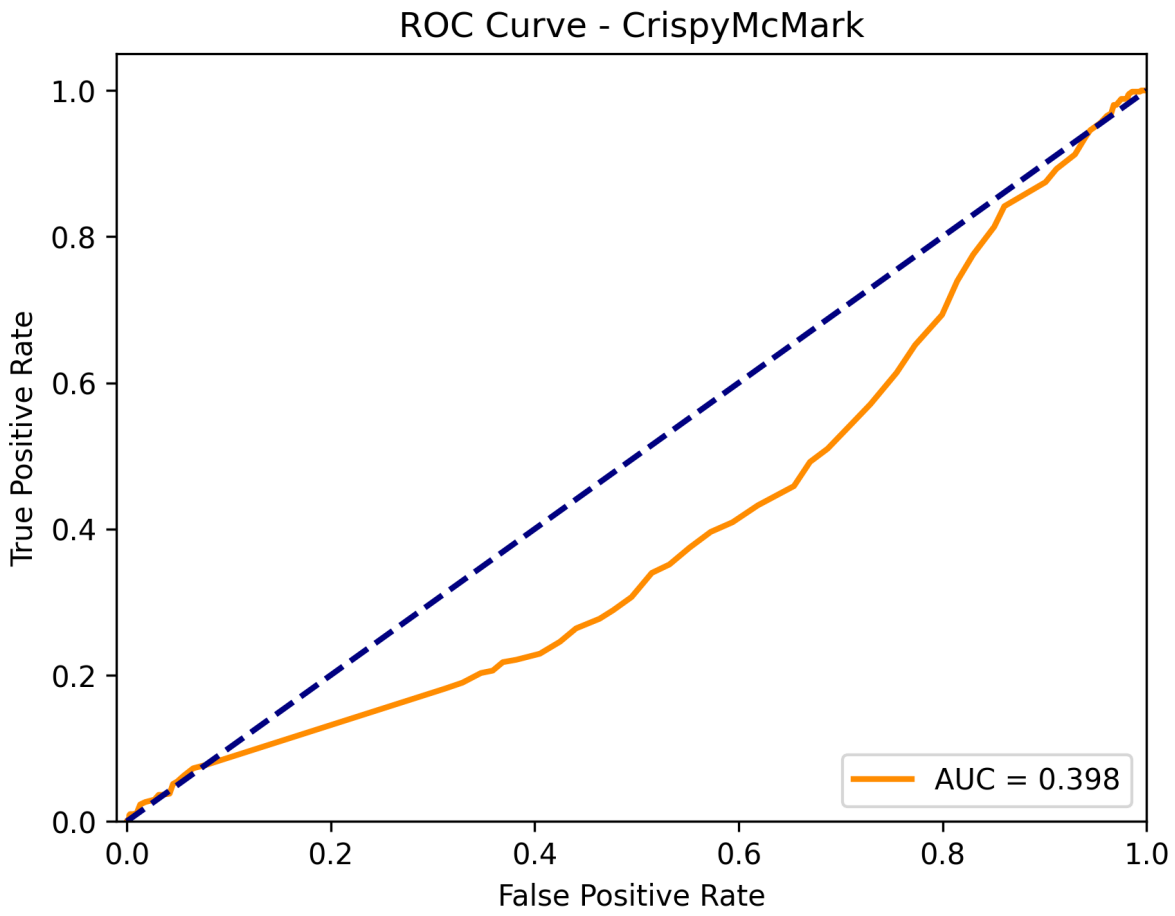

ROC Curve - CrispyMcMark
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
AUC = 1.000

ROC



ROC + label 0 for original (attacked) images + label 0 for destroyed

Effects of hardcoding the watermark



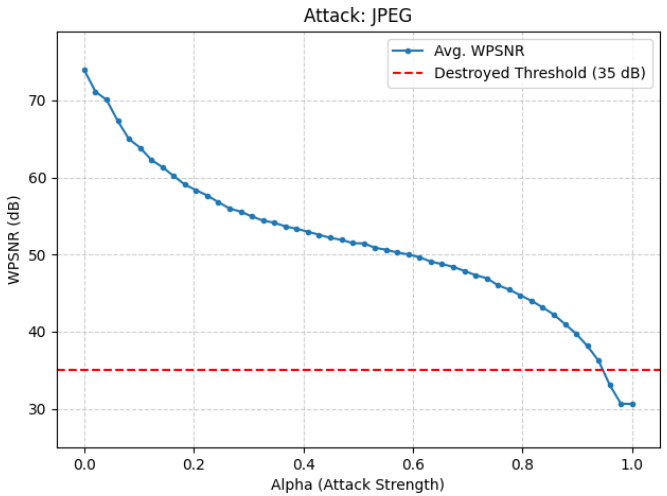
Possible Improvements

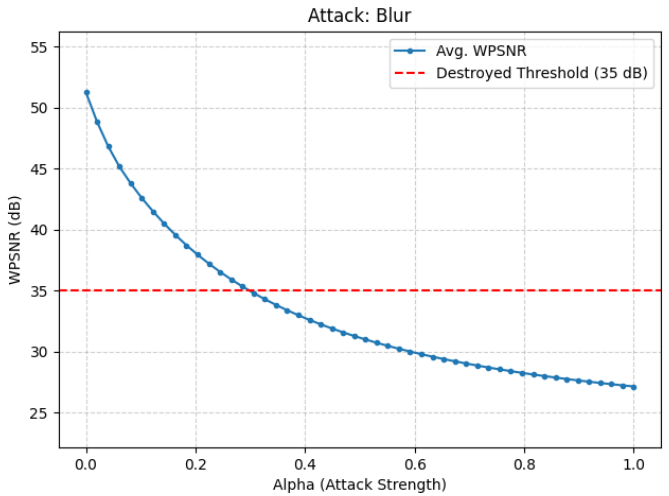
- Add redundancy based on singular value importance
- Improve the invisibility of attack squares, either by block choice or embedding strength
- Explore alternative block embedding techniques

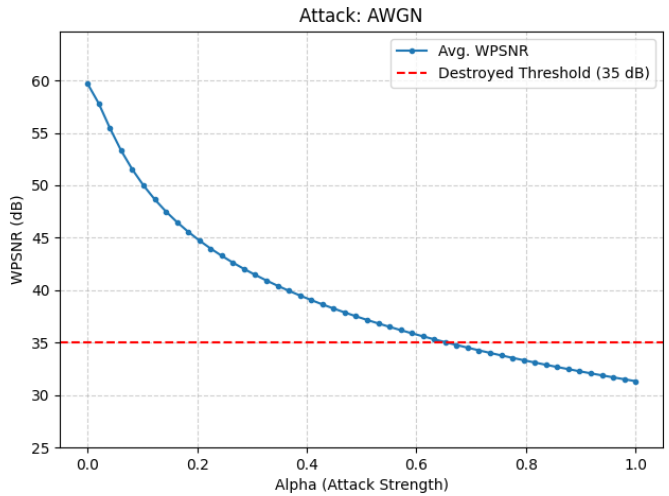
Attack Strategy

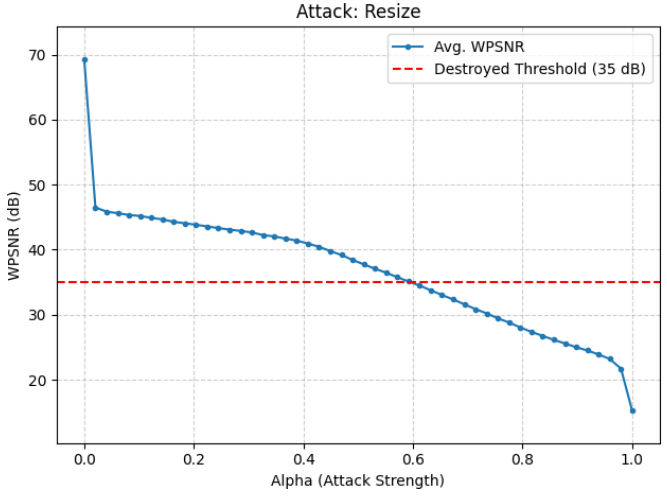
- binary search to find optimal attack strength
- attack functions tweaked to accept parameter $0 \leq \beta \leq 1$
- use of masks to attack different areas of the image
- parallelization to improve execution speed

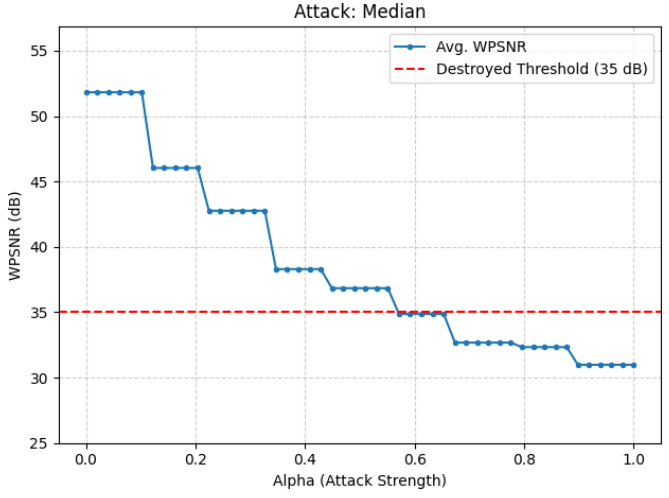
Attack Strategy - binary search

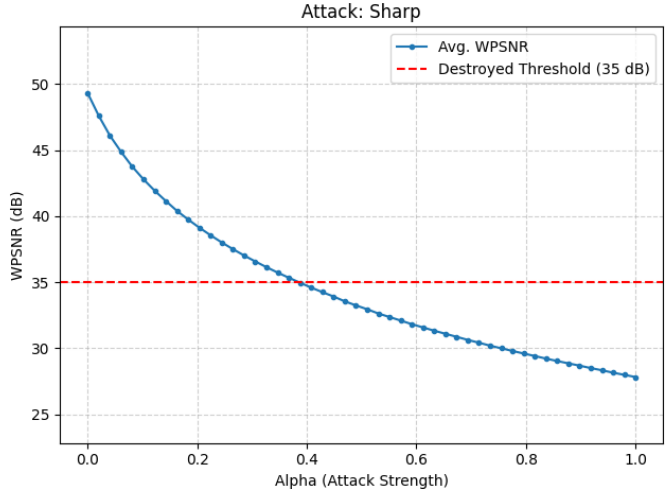












Attack Strategy - masks

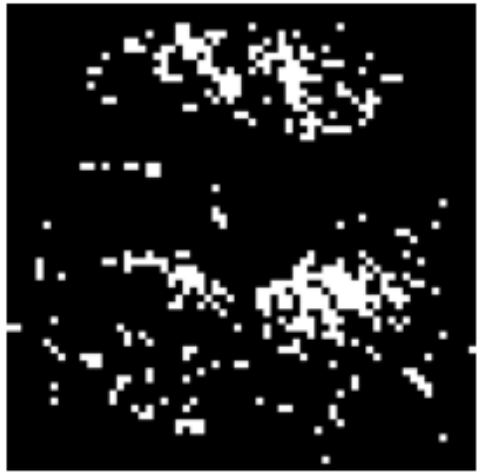
original_img

edges_mask

noisy_mask

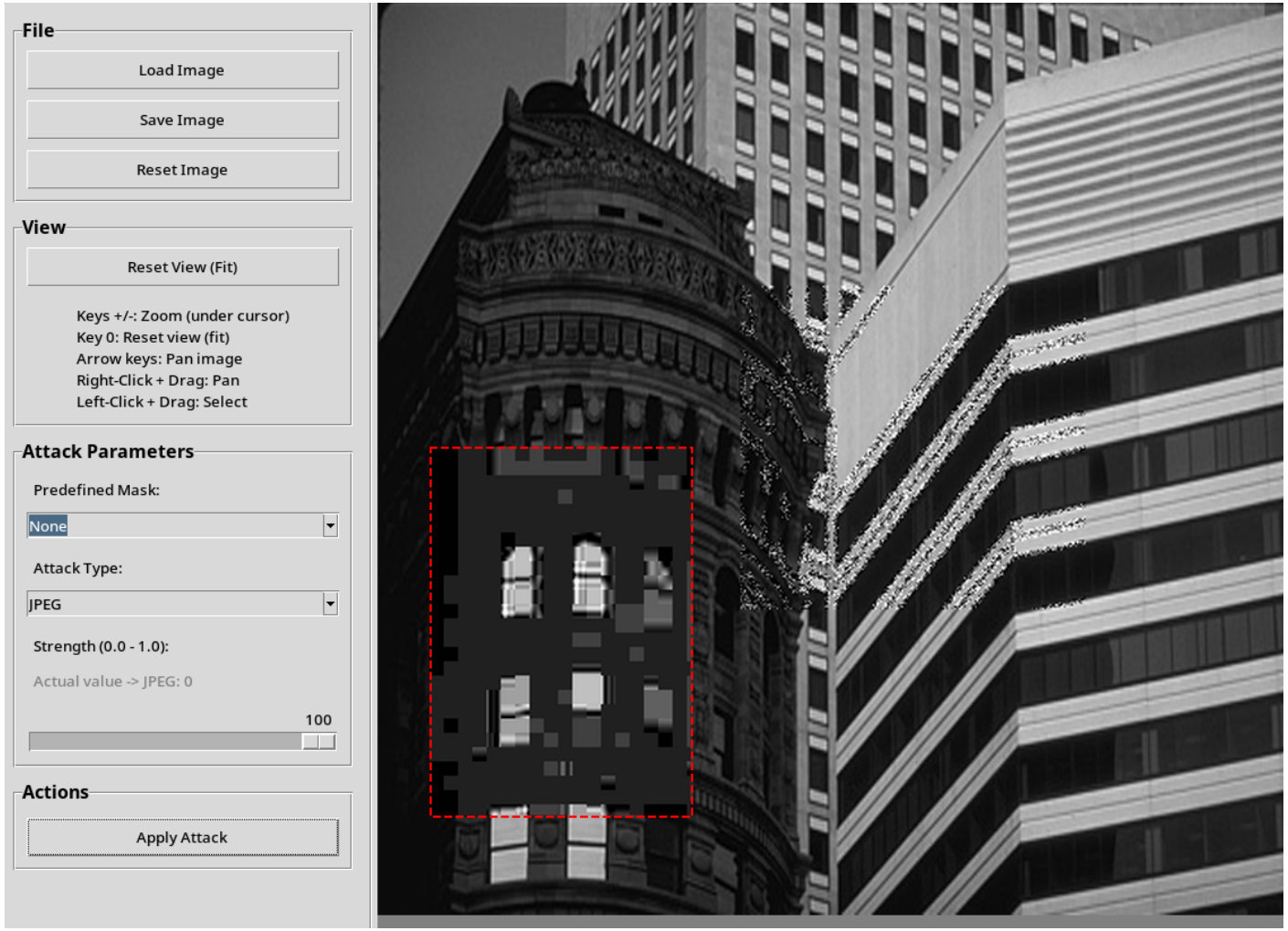
attack_mask

frequency_mask



saliency_mask

Attack Strategy - GUI tool



Questions?