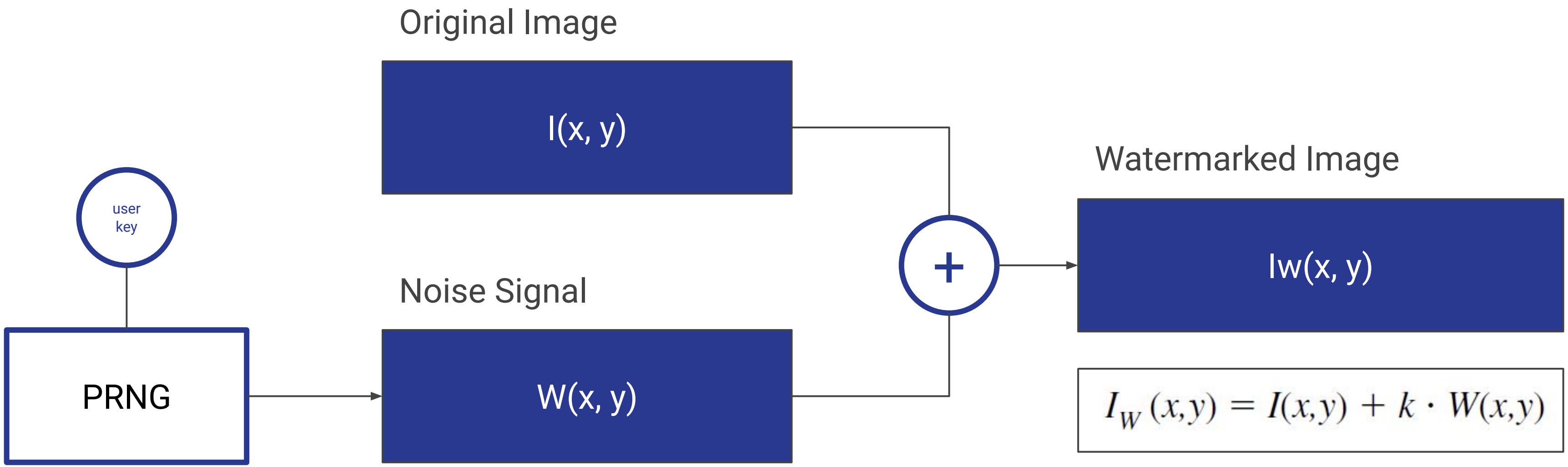
Image Watermarking

with spatial correlation technique using PRNG noise

by Fawwaz Abrial Saffa / 18221067

Table of Contents

1. Watermark Embedding Process



$$I_W(x,y) = I(x,y) + k \cdot W(x,y)$$

Eq 1. Formula to calculate a watermark image

1a. Open original image

Image will be opened using the **Pillow** library in grayscale, then image will be converted into a Numpy array for calculation

```
def open_image(self, file_path):
        original_image = Image.open(file_path).convert('L')
        original_array = np.array(original_image)
        return original_array
```

Fig 1. Sample original image with size 213x236

1b. Generate PRNG Noise Signal

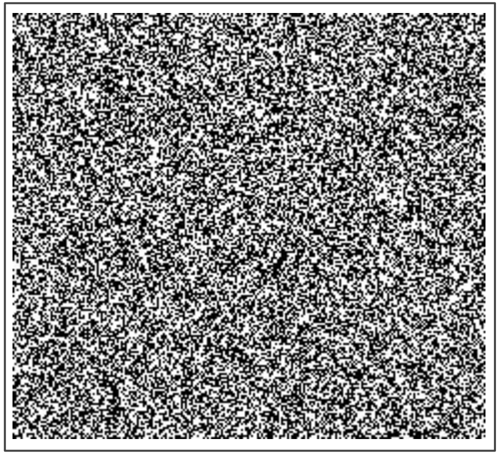
Noise should be:

- pseudo-randomly generated using a **user-key** as seed,
- a **binary pattern** consisting of {0, 1}
- the **same size** as original image
- a pattern where energy should be **uniformly distributed**
- **not correlated** with the original image content

Before adding to the original image, this watermark **should be mapped** to {-1, 1} to produce a zero energy signal

```
def generate_noisy_pattern(self):
        Generator = np.random.default_rng(self.seed)
        noisy_pattern = Generator.integers(0, 2, size=self.size)
        noisy_pattern = 2 * noisy_pattern - 1
        return noisy_pattern
```

Fig 2. Noisy pattern generated when image size is 213x236 and seed is 15012003



1c. Embed Watermark in Image

Mapped watermark will be **multiplied** by the **scaling factor k** then added to the original image to embed the watermark inside the image

```
def generate_watermarked_image(self):
    watermark_pattern = self.generate_noisy_pattern()
    watermarked_image = self.original_image +
                        self.scaling_factor * watermark_pattern
    watermarked_image = np.clip(watermarked_image, 0, 255)
    return watermarked_image
```

After embedding, the watermarked image will be **clipped** to the value of [0, 255] so that values higher than 255 will be dropped to 255 and those below 0 will be raised to 0

Fig 3. Watermarked image when k=10

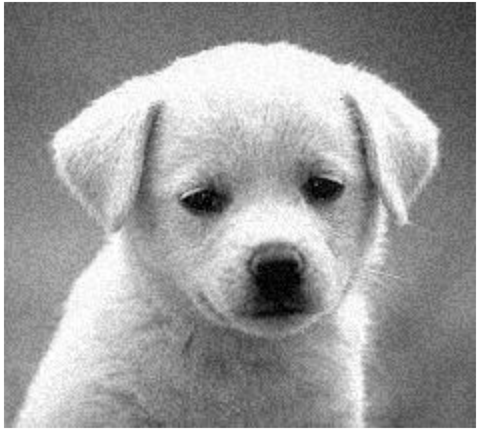
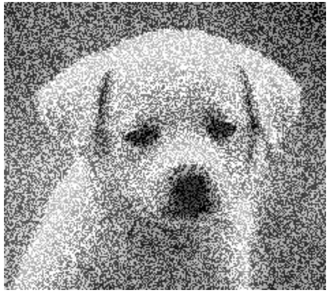


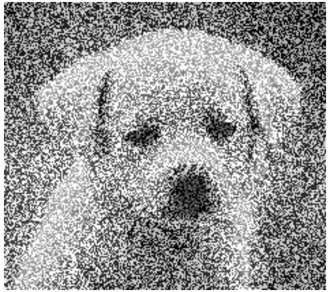
Fig 4. Watermarked image when k=25

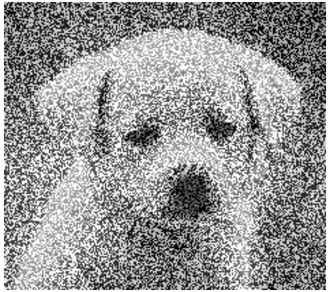

k=0
(similar to original image!)

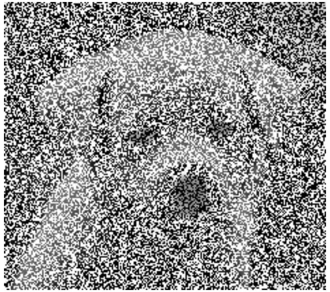

k=25

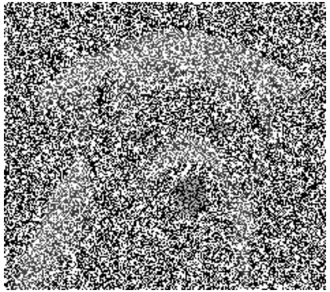

k=45

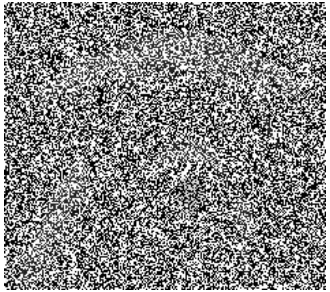

k=65

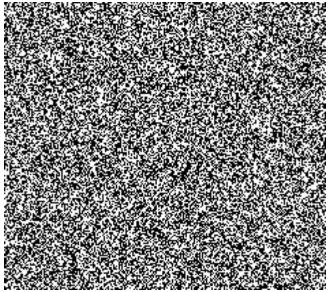

k=90


k=115


k=145


k=180


k=215


k=255
(similar to noisy pattern!)

2. Watermark Embedding Result

3. Watermark Detection

- Detection can be achieved by finding the **correlation** between the **watermarked image** with the **noisy pattern**
- The closer the correlation value is **to one** then the **more detectable** the watermark is; The closer it is **to zero** then the watermark may **not even be present**
- A threshold value may be used as a **success parameter**

$$
\begin{aligned}
R_{I_w(x,y)W(x,y)} &= \frac{1}{N}\sum_{i=1}^{N} I_W(x,y)\,W(x,y) \\
&= \frac{1}{N}\sum_{i=1}^{\frac{N}{2}} I_W(x,y)\,W^{+}(x,y) \;+\; \frac{1}{N}\sum_{i=1}^{\frac{N}{2}} I_W(x,y)\,W^{-}(x,y) \\
&= \frac{1}{2}\left(\mu I_W^{+}(x,y) \;+\; \mu I_W^{-}(x,y)\right)
\end{aligned}
$$

Eq 2. Formula to calculate the correlation between a watermarked image and noisy pattern

3. Watermark Detection

Before calculating, noisy pattern is **normalized to a zero mean** and the watermarked image will be passed through an edge-enhance filter. This is achieved by convolving the image with a convolution kernel

```
def detect_watermark(self):
        normalized_noisy_array = (noisy_pattern -
np.mean(noisy_pattern)) / np.std(noisy_pattern)
        edge_enhanced_image = convolve2d(self.watermarked_image,
np.divide(np.array([
            [-1, -1, -1],
            [-1, 8, -1],
            [-1, -1, -1]
        ]), 2), mode='same')
        normalized_edge_enhanced_image =
     (edge_enhanced_image - np.mean(edge_enhanced_image)) /
                                        np.std(edge_enhanced_image)
```

Fig 5. Convolved image when k=10

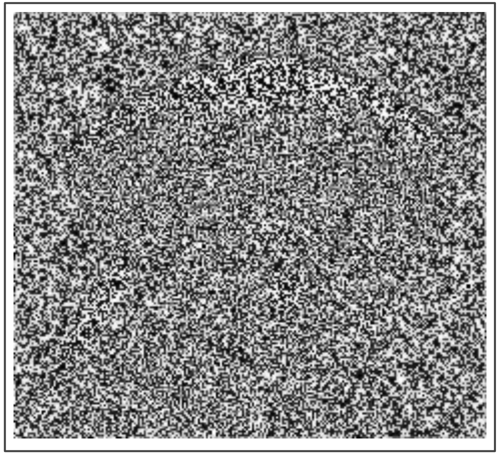
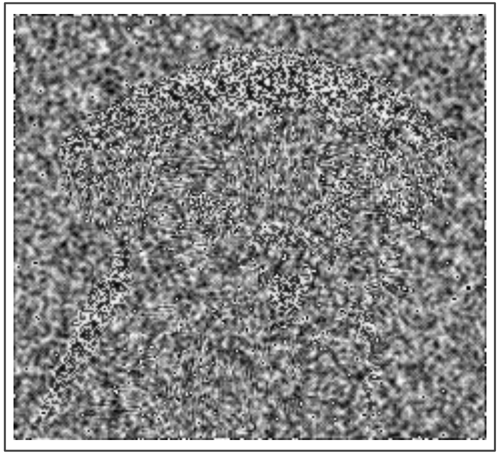
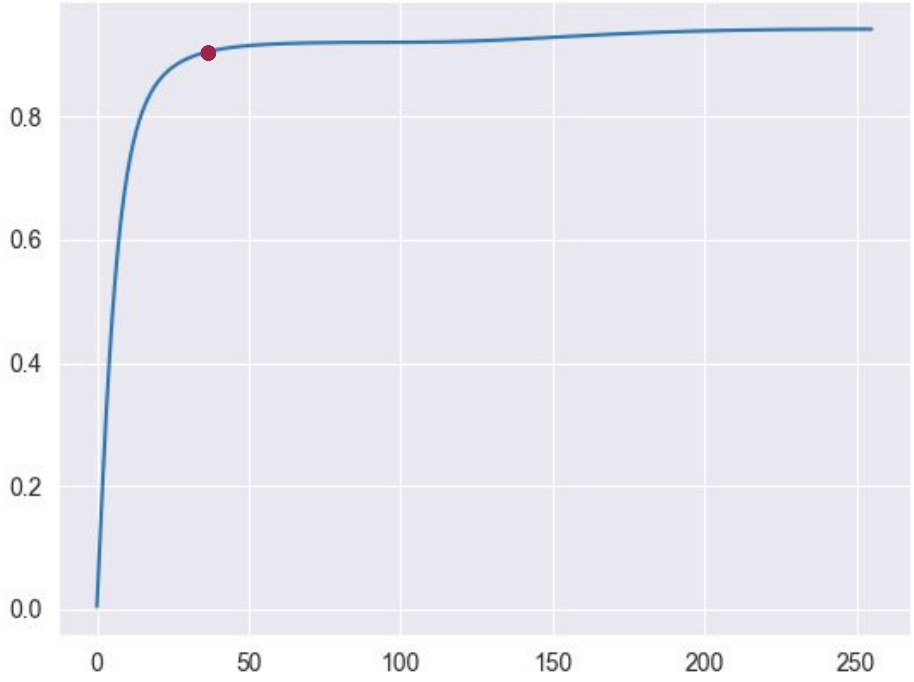


Fig 6. Convolved image when k=25



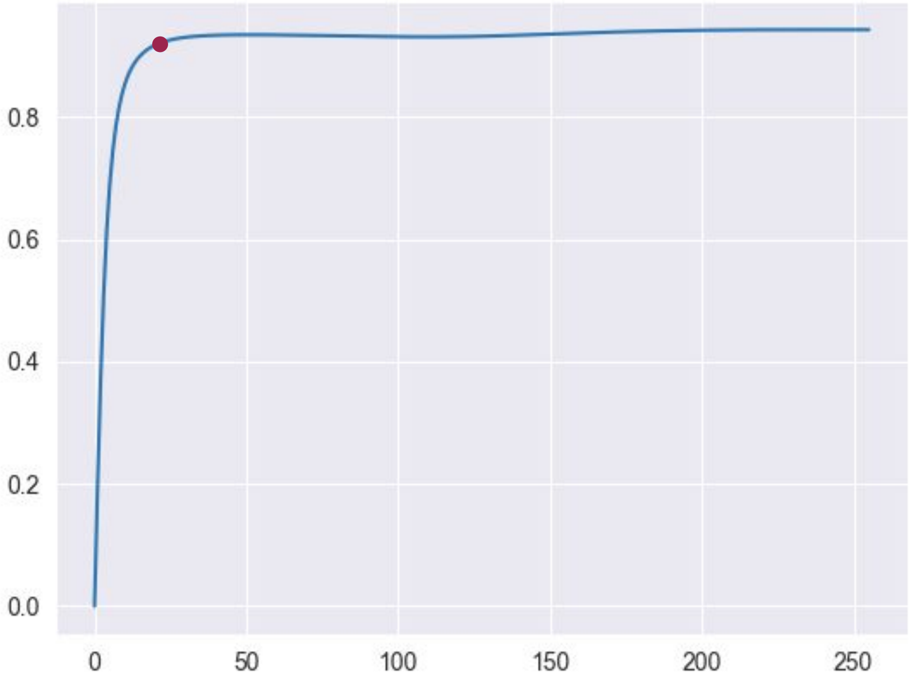
k=25; corr= 0.8805202033489109



At a certain point, increasing the scaling factor **will not improve** the correlation significantly. This suggests that a **threshold** can be used as an **indicator of watermark detection**

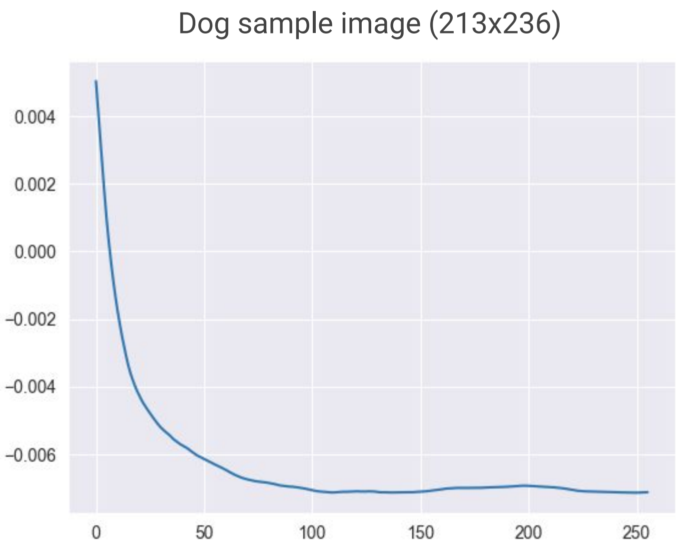
4. Watermark Detection Result

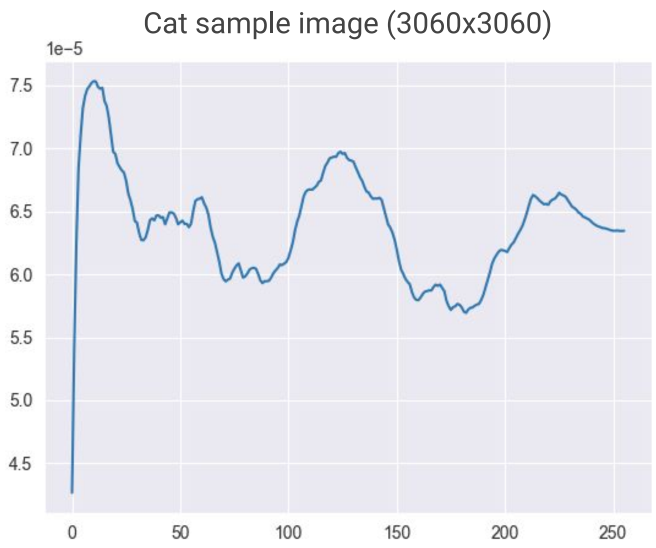
k=16; corr=0.9047595945084232



The same phenomena occurs when using a **different image** with size 3060x3060

4. Watermark Detection Result





When the **wrong key** is provided, the correlation result becomes **abysmally small**. This makes sense because the watermark for the key is **different** than the one embedded.

4. Watermark Detection Result

Conclusion

- The ideal factor gain seems to be around **15-40** since it **doesn't alter** the original image much but **still detectable** using correlation
- A threshold can be used to detect watermarking, using 2 samples a **threshold over 0.8** seems to be robust enough
- This study doesn't take into account **watermark attacks effect on correlation**

Terima kasih!

https://github.com/fawwazabrials/image-watermarking