2025.04.28

Finding SOTA of image watermark

국민대학교 이재형
jaehyeong8121@gmail.com

Contents

What am i looking for?

1 **연구 목표** : SOTA 이미지 워터마킹 기법 찾기 + 실제 환경에서의 강건성 평가

2 **워터마크 역할** : 저작권 보호, 위변조 검출

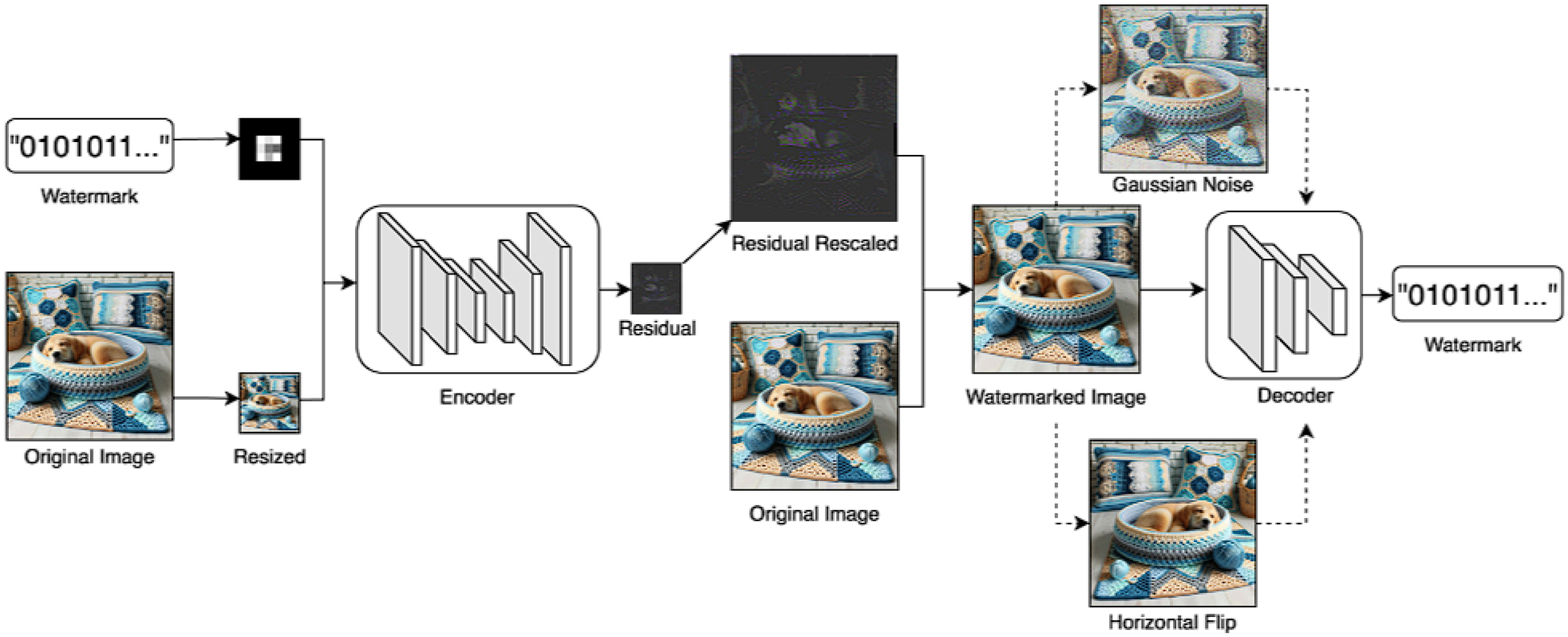
3 **핵심 평가 기준**:

- 워터마크 삽입 후 이미지 품질 유지
- 공격에 대한 강건성

What is InvisMark?

InvisMark: Invisible and Robust Watermarking for AI-generated Image Provenance

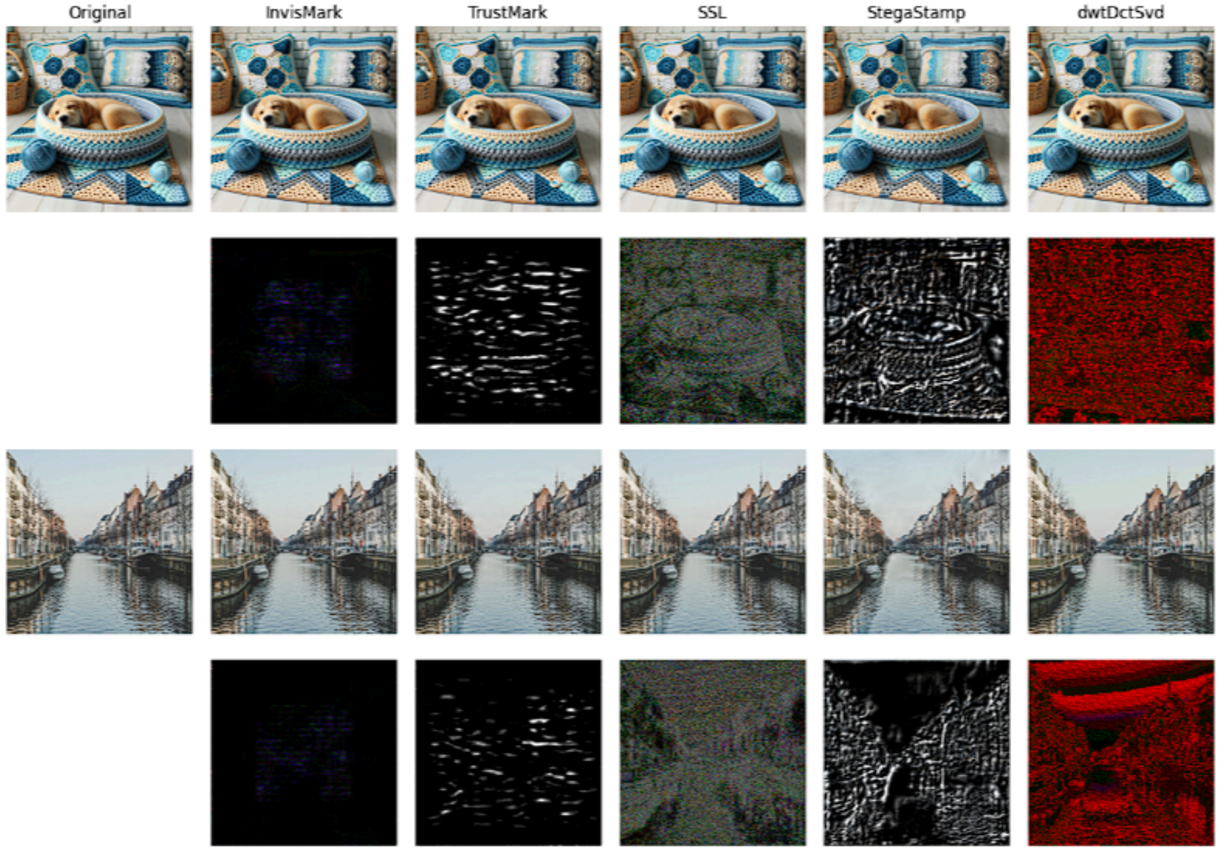
Rui Xu Mengya (Mia) Hu Deren Lei Yaxi Li David Lowe Alex Gorevski
Mingyu Wang Emily Ching
Alex Deng
Microsoft Responsible AI



2.1 Why InvisMark is SOTA?

1.Imperceptibility

	DIV2K					DALL-E 3				
	InvisMark	TrustMark	SSL	StegaStamp	dwtDctSvd	InvisMark	TrustMark	SSL	StegaStamp	dwtDctSvd
PSNR	51.4	41.9	42.6	37.4	38.3	51.4	41.9	42.7	37.5	40.2
SSIM	0.998	0.995	0.989	0.978	0.974	0.998	0.994	0.990	0.976	0.981



- **InvisMark**
  - PSNR : 51.4
  - SSIM : 0.998
- **다른 기법들보다 훨씬 높은 값**을 통해 뛰어난 imperceptibility 달성

- **Residual(잔차)**
  - InvisMark가 다른 기법들보다 **잔차가 훨씬 작음**
  - 워터마크가 눈에 띄지 않음을 시각적으로 입증

02 What is InvisMark?

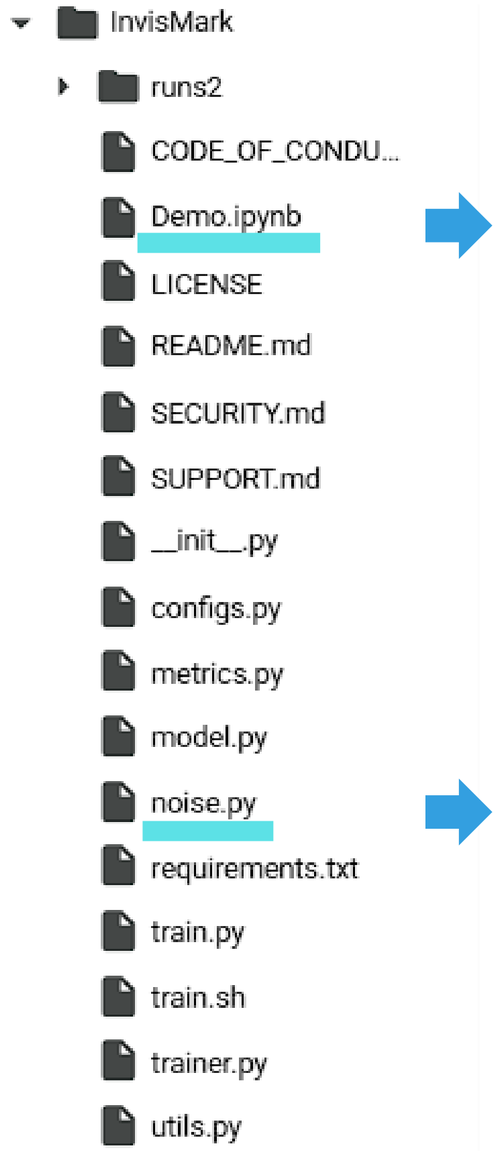
2.1 Why InvisMark is SOTA?

2.Robustness

	DIV2K					DALL-E 3				
	InvisMark	TrustMark	SSL	StegaStamp	dwtDctSvd	InvisMark	TrustMark	SSL	StegaStamp	dwtDctSvd
Clean Image	**100.0**	99.7	98.6	99.8	**100.0**	**100.0**	99.9	97.7	99.8	99.3
Jpeg Compression	99.5	89.7	53.9	**99.8**	80.8	97.5	92.9	52.6	**99.8**	79.6
Brightness	**99.7**	82.6	76.1	98.0	93.6	**99.8**	84.8	74.4	97.5	92.5
Contrast	**99.9**	81.8	71.7	99.4	62.8	**99.9**	82.8	69.1	99.1	56.3
Saturation	**100.0**	94.9	89.4	99.8	59.2	**100.0**	95.7	86.6	99.7	58.3
GaussianBlur	**100.0**	99.7	96.9	99.8	99.7	**100.0**	99.9	95.5	99.8	98.2
GaussianNoise	**100.0**	69.9	68.9	88.9	98.8	**100.0**	65.1	68.1	84.8	96.4
ColorJiggle	**100.0**	89.2	69.2	99.6	81.4	**100.0**	90.5	66.7	99.4	76.3
Posterize	**100.0**	94.9	81.8	99.7	99.5	**100.0**	96.0	80.3	99.7	98.6
RGBShift	**100.0**	90.2	70.4	99.8	53.5	**100.0**	92.5	67.8	99.8	47.8
Flip	**100.0**	99.7	94.8	50.9	52.6	**100.0**	99.8	91.4	50.7	54.3
Rotation	**97.4**	68.7	95.3	88.2	52.6	**98.7**	73.5	92.9	90.8	54.0
RandomErasing	**99.8**	98.5	98.6	99.5	**99.8**	**99.9**	97.4	97.6	99.5	99.1
Perspective	**100.0**	88.9	94.9	94.4	50.1	**100.0**	90.9	92.1	94.4	50.3
RandomResizedCrop	**97.3**	96.8	92.7	79.6	52.4	**99.8**	99.7	90.6	81.9	51.7

- 위쪽 그룹 : 픽셀 기반 공격 / 아래쪽 그룹 : 기하학적 변형 공격
- InvisMark가 **대부분의 변환에서 다른 방법들보다 훨씬 높은 비트 정확도를 달성**함
  - dwtDctSvd : Gaussian, Random Erasing 제외 노이즈에 취약
  - StegaStamp : 좋은 강건성, (이전 테이블) PSNR : 31.7 → 창작 분야에서 사용 어려움
  - SSL : JPEG 압축, ColorShift에 취약
  - TrustMark : Gaussian Noise, Rotation에 취약

2.2 File Structure of Invismark



사전학습 모델로 단순 워터마크 삽입 및 복원 실험

워터마크 삽입된 이미지에 공격 적용 코드 추가

다양한 이미지 변환(공격) 함수 모음

공격 기법 다양화 코드 추가

2.3 InvisMark Demo code operation

1.데이터셋 로딩

```
ds_dalle = load_dataset("OpenDatasets/dalle-3-dataset", split='train[:5%]')
```

2.사전학습된 모델 불러오기

```
state_dict = torch.load(ckpt_path)
cfg = state_dict['config']
wm_model = train.Watermark(cfg)
wm_model.load_model(ckpt_path)
```

3.Watermark 삽입 + 복원 실험

```
for idx, eval_batch in enumerate(dataloader):
    inputs = eval_batch['image']
    secret, _ = utils.uuid_to_bits(inputs.shape[0])
    secret = secret[:, :100].to(device)
    final_output, enc_input, enc_output = wm_model._encode(inputs, secret)
    extracted_secret = wm_model._decode(final_output)
```

2.3 InvisMark Demo code operation

4.결과 평가

```
psnr = metrics.image_psnr(inputs, final_output)
ssim = metrics.image_ssim(inputs, final_output)
bitAcc = metrics.bit_accuracy(extracted_secret, secret)
```

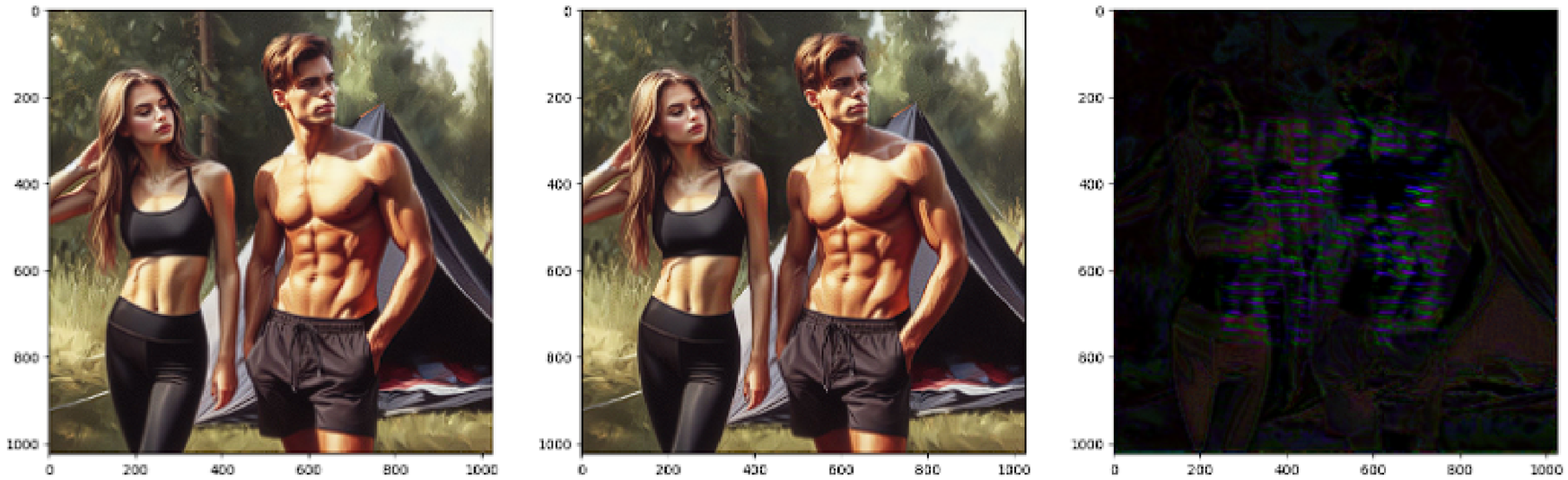
```
PSNR: 51.42175292968875, SSIM: 0.9988216161727905, BitAcc: tensor([1.])
```

PSNR	30dB 이상	고품질 (거의 구별 불가)	SSIM	0.95 이상	고품질 (거의 동일)
PSNR	20~30dB	약간 손상 (가끔 눈에 띔)	SSIM	0.9~0.95	양호 (미세한 차이)
PSNR	20dB 이하	심각한 손상 (명확히 보임)	SSIM	0.8 이하	손상 (구조 왜곡 눈에 띔)

5.시각화

```
fig, ax = plt.subplots(1, 3, figsize=(20,8))
```

```
<matplotlib.image.AxesImage at 0x7f78f4115a50>
```



Experiments

noise.py 분석

```python
# Medium level noise
def supported_transforms(image_size):
    return {
    'HFlip': aug.RandomHorizontalFlip(p=1.0),
    'VFlip': aug.RandomVerticalFlip(p=1.0),
    "Rotation":  aug.RandomRotation(degrees=[0.0, 10.0], p=1.0),
    #"Rotation":  aug.RandomRotation(degrees=[0.0, 45.0], p=1.0),
    'Perspective': aug.RandomPerspective(distortion_scale=0.1, p=1.0),
    'RandomResizedCrop':  aug.RandomResizedCrop(size=image_size, scale=(0.75, 1.0), ratio=(3.0 / 4.0, 4.0 / 3.0)),
    #'RandomResizedCrop':  aug.RandomResizedCrop(size=image_size, scale=(0.5, 1.0), ratio=(3.0 / 4.0, 4.0 / 3.0)),

    'Brighter':  aug.RandomBrightness(brightness=(1.25, 1.25)), # 0.5-1.5
    'Darker':  aug.RandomBrightness(brightness=(0.75, 0.75)), # 0.5-1.5
    'Contrast_p': aug.RandomContrast(contrast=(1.25, 1.25)), # 0.5 - 1.5
    'Contrast_m': aug.RandomContrast(contrast=(0.75, 0.75)), # 0.5 - 1.5
    'Saturation_p': aug.RandomSaturation(saturation=(1.25, 1.25)), #
    'Saturation_m': aug.RandomSaturation(saturation=(0.75, 0.75)), #
    'BoxBlur': aug.RandomBoxBlur(kernel_size=(5, 5), p= 1.0), # 7.
    'GaussianBlur': aug.RandomGaussianBlur(kernel_size=(5, 5), sigma=(1.0, 1.5), p=1.0), # 7, 0.1-2.0
    'GaussianNoise': aug.RandomGaussianNoise(mean=0.0, std=0.04, p=1.0), # 0.08
    "Jpeg": v2.JPEG(quality=[50, 50]), # it is expected to have dtype uint8, on CPU, and have [..., 3 or 1, H, W] shape 40
    'RandomErasing': aug.RandomErasing(scale=(0.02, 0.1), ratio=(0.5, 1.5), p=1.0),
    "Jiggle":  aug.ColorJiggle(brightness=0.1, contrast=0.1, saturation=0.1, hue=0.02),
    "Posterize": aug.RandomPosterize(bits=4),
    "RGBShift": aug.RandomRGBShift(r_shift_limit=0.05, g_shift_limit=0.05, b_shift_limit=0.05, p=1.0),
    }
```

```python
class Noiser(nn.Module):
    def __init__(self, num_transforms, device):
        super().__init__()
        self.device = device
        self.num_transforms = num_transforms
        keys = list(supported_transforms((256, 256)).keys())
        self.geo_transforms = keys[:5]
        self.pert_transforms = keys[5:]

    def forward(self, input, noises=None):
        if noises:
            for key in noises:
                input = self.apply_noise(input, key)
            return input
        for key in random.choices(self.geo_transforms, k = self.num_transforms):
            input = self.apply_noise(input, key)
        for key in random.choices(self.pert_transforms, k = self.num_transforms):
            input = self.apply_noise(input, key)
        return input

    def apply_noise(self, input, key):
        transforms = supported_transforms(input.shape[-2:])
        if key not in transforms:
            raise Exception(f"{key} is not a supported image transformation.")
        # JPEG dtype uint8, on CPU, and have [..., 3 or 1, H, W] shape
        if key == "Jpeg":
            input = ((input + 1.0)*127.5).to(torch.uint8).to('cpu')
            output = transforms[key](input)
            output = (output / 127.5 - 1.0).to(self.device)
            return output
        # Kornia assumes image data range [0, 1]
        else:
            input = (input + 1.0) / 2.0
            output = transforms[key](input)
            return output * 2.0 - 1.0
```

3.1 Testing InvisMark by Applying Random Attacks

```python
for idx, eval_batch in enumerate(dataloader):
    inputs = eval_batch['image']
    secret, _ = utils.uuid_to_bits(inputs.shape[0])
    secret = secret[:, :100].to(device)
    final_output, enc_input, enc_output = wm_model._encode(inputs, secret)

    psnr = metrics.image_psnr(inputs, final_output.cpu()).detach().numpy()
    ssim = metrics.image_ssim(inputs, final_output.cpu()).detach().numpy()
    extracted_secret = wm_model._decode(final_output)
    bitAcc = metrics.bit_accuracy(extracted_secret, secret)
    print(f"PSNR: {psnr}, SSIM: {ssim}, BitAcc: {bitAcc}")
    break
```

```python
# 다양한 이미지 변형(공격)을 적용할 수 있는 Noiser 객체 생성
noiser = noise.Noiser(num_transforms=1, device=device)

for idx, eval_batch in enumerate(dataloader):
    inputs = eval_batch
    secret, _ = utils.uuid_to_bits(inputs.shape[0])
    secret = secret[:, :100].to(device)
    final_output, enc_input, enc_output = wm_model._encode(inputs, secret)

    # 공격 추가
    attacked_output = noiser(final_output)

    psnr = metrics.image_psnr(inputs, attacked_output.cpu()).detach().numpy()
    ssim = metrics.image_ssim(inputs, attacked_output.cpu()).detach().numpy()
    extracted_secret = wm_model._decode(attacked_output)
    bitAcc = metrics.bit_accuracy(extracted_secret, secret)

    print(f"PSNR after attack: {psnr}, SSIM: {ssim}, BitAcc: {bitAcc}")
    break
```

3.1 Testing InvisMark by Applying Random Attacks

```
PSNR: 51.42175292968875, SSIM: 0.9988216161727905, BitAcc: tensor([1.])
```

```
PSNR after attack: 8.6186637878417797, SSIM: 0.17278034985065546, BitAcc: tensor([1.])
```

공격으로 이미지 품질은 심각하게 떨어졌지만, 워터마크 복원은 완벽하게 성공함

But, 실질적으로 효과적인 공격은 **이미지 품질을 최대한 유지하면서**

워터마크 비트 복원 정확도를 낮추는 공격 → **다양한 공격 시나리오의 필요성**

3.2 Enhancing Attack Scenarios by Modifying Noise Module

```python
# Medium level noise
def supported_transforms(image_size):
    return {
    'HFlip': aug.RandomHorizontalFlip(p=1.0),
    'VFlip': aug.RandomVerticalFlip(p=1.0),
    "Rotation":  aug.RandomRotation(degrees=[0.0, 10.0], p=1.0),
    #"Rotation":  aug.RandomRotation(degrees=[0.0, 45.0], p=1.0),
    'Perspective': aug.RandomPerspective(distortion_scale=0.1, p=1.0),
    'RandomResizedCrop':  aug.RandomResizedCrop(size=image_size, scale=(0.75, 1.0), ratio=(3.0 / 4.0, 4.0 / 3.0)),
    #'RandomResizedCrop':  aug.RandomResizedCrop(size=image_size, scale=(0.5, 1.0), ratio=(3.0 / 4.0, 4.0 / 3.0)),

    'Brighter':  aug.RandomBrightness(brightness=(1.25, 1.25)), # 0.5-1.5
    'Darker':  aug.RandomBrightness(brightness=(0.75, 0.75)), # 0.5-1.5
    'Contrast_p': aug.RandomContrast(contrast=(1.25, 1.25)), # 0.5 - 1.5
    'Contrast_m': aug.RandomContrast(contrast=(0.75, 0.75)), # 0.5 - 1.5
    'Saturation_p': aug.RandomSaturation(saturation=(1.25, 1.25)), #
    'Saturation_m': aug.RandomSaturation(saturation=(0.75, 0.75)), #
    'BoxBlur': aug.RandomBoxBlur(kernel_size=(5, 5), p= 1.0), # 7.
    'GaussianBlur': aug.RandomGaussianBlur(kernel_size=(5, 5), sigma=(1.0, 1.5), p=1.0), # 7, 0.1-2.0
    'GaussianNoise': aug.RandomGaussianNoise(mean=0.0, std=0.04, p=1.0), # 0.08
    "Jpeg": v2.JPEG(quality=[50, 50]), # it is expected to have dtype uint8, on CPU, and have [..., 3 or 1, H, W] shape 40
    'RandomErasing': aug.RandomErasing(scale=(0.02, 0.1), ratio=(0.5, 1.5), p=1.0),
    "Jiggle":  aug.ColorJiggle(brightness=0.1, contrast=0.1, saturation=0.1, hue=0.02),
    "Posterize": aug.RandomPosterize(bits=4),
    "RGBShift": aug.RandomRGBShift(r_shift_limit=0.05, g_shift_limit=0.05, b_shift_limit=0.05, p=1.0),
    }
```

```python
# Medium level noise
def supported_transforms(image_size, mode="combined"):
    all_transforms = {
    'HFlip': aug.RandomHorizontalFlip(p=1.0),
    'VFlip': aug.RandomVerticalFlip(p=1.0),
    "Rotation":  aug.RandomRotation(degrees=[0.0, 10.0], p=1.0),
    #"Rotation":  aug.RandomRotation(degrees=[0.0, 45.0], p=1.0),
    'Perspective': aug.RandomPerspective(distortion_scale=0.1, p=1.0),
    'RandomResizedCrop':  aug.RandomResizedCrop(size=image_size, scale=(0.75, 1.0), ratio=(3.0 / 4.0, 4.0 / 3.0)),
    #'RandomResizedCrop':  aug.RandomResizedCrop(size=image_size, scale=(0.5, 1.0), ratio=(3.0 / 4.0, 4.0 / 3.0)),

    'Brighter':  aug.RandomBrightness(brightness=(1.25, 1.25)), # 0.5-1.5
    'Darker':  aug.RandomBrightness(brightness=(0.75, 0.75)), # 0.5-1.5
    'Contrast_p': aug.RandomContrast(contrast=(1.25, 1.25)), # 0.5 - 1.5
    'Contrast_m': aug.RandomContrast(contrast=(0.75, 0.75)), # 0.5 - 1.5
    'Saturation_p': aug.RandomSaturation(saturation=(1.25, 1.25)), #
    'Saturation_m': aug.RandomSaturation(saturation=(0.75, 0.75)), #
    'BoxBlur': aug.RandomBoxBlur(kernel_size=(5, 5), p= 1.0), # 7.
    'GaussianBlur': aug.RandomGaussianBlur(kernel_size=(5, 5), sigma=(1.0, 1.5), p=1.0), # 7, 0.1-2.0
    'GaussianNoise': aug.RandomGaussianNoise(mean=0.0, std=0.04, p=1.0), # 0.08
    "Jpeg": v2.JPEG(quality=[50, 50]), # it is expected to have dtype uint8, on CPU, and have [..., 3 or 1, H, W] shape 40
    'RandomErasing': aug.RandomErasing(scale=(0.02, 0.1), ratio=(0.5, 1.5), p=1.0),
    "Jiggle":  aug.ColorJiggle(brightness=0.1, contrast=0.1, saturation=0.1, hue=0.02),
    "Posterize": aug.RandomPosterize(bits=4),
    "RGBShift": aug.RandomRGBShift(r_shift_limit=0.05, g_shift_limit=0.05, b_shift_limit=0.05, p=1.0),
    }

    if mode == 'mild':
        selected = {k: all_transforms[k] for k in ['HFlip', 'VFlip', 'Brighter', 'Darker', 'GaussianNoise']}
    elif mode == 'moderate':
        selected = {k: all_transforms[k] for k in ['Rotation', 'Perspective', 'GaussianBlur', 'Contrast_p', 'Contrast_m']}
    else:  # combined
        selected = all_transforms

    return selected
```

- **공격 강도 3단계 구분**
  - mild : 좌우/상하 뒤집기, 밝기 살짝 조정, 가우시안 노이즈
  - moderate : 회전, 원근 왜곡, 블러, 대비 조정
  - combined : Milde + Moderate + 나머지 전부 섞음

- **all_transforms** : 모든 augmentation 종류를 한 번에 정의
- **mode 인자**에 따라서,
  - mild : 가벼운 변형
  - moderate : 중간 난이도 변형 → 선택된 공격 리스트를 selected에 저장하여 리턴
  - combined : 전부 사용

3.2 Enhancing Attack Scenarios by Modifying Noise Module

```python
class Noiser(nn.Module):
    def __init__(self, num_transforms, device):
        super().__init__()
        self.device = device
        self.num_transforms = num_transforms
        keys = list(supported_transforms((256, 256)).keys())
        self.geo_transforms = keys[:5]
        self.pert_transforms = keys[5:]

    def forward(self, input, noises=None):
        if noises:
            for key in noises:
                input = self.apply_noise(input, key)
            return input
        for key in random.choices(self.geo_transforms, k = self.num_transforms):
            input = self.apply_noise(input, key)
        for key in random.choices(self.pert_transforms, k = self.num_transforms):
            input = self.apply_noise(input, key)
        return input

    def apply_noise(self, input, key):
        transforms = supported_transforms(input.shape[-2:])
        if key not in transforms:
            raise Exception(f"{key} is not a supported image transformation.")
        # JPEG dtype uint8, on CPU, and have [..., 3 or 1, H, W] shape
        if key == "Jpeg":
            input = ((input + 1.0)*127.5).to(torch.uint8).to('cpu')
            output = transforms[key](input)
            output = (output / 127.5 - 1.0).to(self.device)
            return output
        # Kornia assumes image data range [0, 1]
        else:
            input = (input + 1.0) / 2.0
            output = transforms[key](input)
            return output * 2.0 - 1.0
```

```python
class Noiser(nn.Module):
    def __init__(self, mode='combined', num_transforms=2, device='cpu'):
        super().__init__()
        self.device = device
        self.num_transforms = num_transforms
        self.mode = mode
        self.transforms = supported_transforms((256, 256), mode=self.mode)
        self.transform_keys = list(self.transforms.keys())

    def forward(self, input, noises=None):
        if noises:
            for key in noises:
                input = self.apply_noise(input, key)
            return input

        # 랜덤하게 self.num_transforms개 선택해서 적용
        selected = random.choices(self.transform_keys, k=self.num_transforms)
        for key in selected:
            input = self.apply_noise(input, key)
        return input

    def apply_noise(self, input, key):
        transforms = supported_transforms(input.shape[-2:], mode=self.mode)
        if key not in transforms:
            raise Exception(f"{key} is not a supported image transformation.")

        if key == "Jpeg":
            input = ((input + 1.0) * 127.5).to(torch.uint8).to('cpu')
            output = transforms[key](input)
            output = (output / 127.5 - 1.0).to(self.device)
            return output
        else:
            input = (input + 1.0) / 2.0
            output = transforms[key](input)
            return output * 2.0 - 1.0
```

- **수정된 Noiser 클래스**
  - 공격 강도 조절 가능 : 3가지 mode
  - transform 통합 관리 : geo/pert 분리 없이 하나의 transform 리스트 관리
  - 랜덤 transform 적용 로직 개선 : “geo/pert 각각 k개씩 선택 -> 하나의 리스트에서 k개 랜덤 선택“ ->> 적용 방식 간결 + 유연성 증가

Results & Conclusion

mild PSNR after attack: 33.89, SSIM: 0.9880, BitAcc: tensor([1.])

moderate PSNR after attack: 18.19, SSIM: 0.5496, BitAcc: tensor([1.])

combined PSNR after attack: 9.54, SSIM: 0.1894, BitAcc: tensor([1.])

- 공격이 강해질수록 (mild → combined) : PSNR과 SSIM은 점점 떨어짐(품질 저하)
- But, BitAcc(워터마크 복원 정확도)는 100% 유지됨

➡ 즉, InvisMark 워터마킹 시스템이 **공격 강도에 상관없이 복원력이 매우 뛰어남**

감사합니다 :)