2025.05.12

Is InvisMark Truly Robust?
: Experimental Analysis with the WAVES

국민대학교 이재형
jaehyeong8121@gmail.com

Contents

Pretrained Model Evaluation

(WACV2025) InvisMark: Invisible and Robust Watermarking for AI-generated Image Provenance

This repo contains source code used for the paper:
https://arxiv.org/pdf/2411.07795

Environment Requirement

```
pip install -r requirements.txt
```

Main Script

- Leverage the pretrained model ckpt
  - Download the pretrained model weights (with 100 encoded bits and no ECC included) from: https://1drv.ms/f/c/7882afab383c8474/Ei_Lasu5CrpHsrNIkYRLenYBmx662VSAovq5hD8r-NsB5A?e=gbHNVX
  - Follow the instruction in `Demo.ipynb`

[사용 모델]

- paper.ckpt
- 사전 학습된 모델

[입력 이미지]

- dalle 3에서 수집한 이미지 50장

[평가 지표]

- 이미지 품질 : PSNR, SSIM
- 워터마크 복원도 : Bit Accuracy

1.1 Clean Image Evaluation

Original_image

InvisMark 삽입

Watermaked_image

- dalle 3에서 수집한 50장 원본 이미지 저장
- 원본 이미지에 InvisMark 삽입 후 PSNR, SSIM, Bit Acc 측정
- 워터마크 삽입 이미지 또한 저장

01 Pretrained Model Evaluation

1.1 Clean Image Evaluation

```
# 1. 스트리밍 모드로 전체를 불러온다
ds_dalle = load_dataset("OpenDatasets/dalle-3-dataset", split='train', streaming=True)

# 2. transform 정의
transform = transforms.Compose([
    transforms.Resize((256, 256)),  # 사이즈 맞추기
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
])

# 3. 이미지 저장할 폴더 생성 (origin_img)
origin_img_dir = '/content/drive/MyDrive/origin_img'
os.makedirs(origin_img_dir, exist_ok=True)

# 4. 수동으로 50개 가져오고 저장
images = []
n = 50  # 사용할 이미지 수

for idx, example in enumerate(ds_dalle):
    img = example['image']
    if isinstance(img, Image.Image):
        # PIL 이미지 저장
        origin_save_path = os.path.join(origin_img_dir, f'original_{idx+1:02d}.png')
        img.save(origin_save_path)
        print(f"[{idx+1}/50] 원본 저장 완료 - {origin_save_path}")

        # 텐서로 변환해서 리스트에 저장
        img_tensor = transform(img)
        images.append(img_tensor)

    if len(images) >= n:
        break

# 5. 배치로 묶기
batch = torch.stack(images)  # (50, 3, 256, 256)

# 6. 데이터로더
dataloader = torch.utils.data.DataLoader(batch, batch_size=1, shuffle=False, num_workers=0)
```

- datasets 라이브러리에서 dalle 3 이미지 데이터셋을 불러옴
- 모델에 넣기 전에 이미지 전처리 과정을 거침
  - Resize : 256 x 256
  - ToTensor
  - Normalize

- 이미지 저장 폴더 : origin_img
- 루프를 통해서 이미지 50장 자동 수집

- 이미지 50장을 하나의 batch로 묶음
  - shape = (50, 3, 256, 256)
- 데이터로더 변환

01 Pretrained Model Evaluation

1.1 Clean Image Evaluation

```
save_dir = '/content/drive/MyDrive/watermarked_img'
psnr_list = []
ssim_list = []
bitacc_list = []
```

- 워터마킹된 이미지 저장 폴더 경로
- 각 이미지에 대한 실험 결과를 저장할 리스트 생성

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

    with torch.no_grad():
      final_output, enc_input, enc_output = wm_model._encode(inputs, secret)

      psnr = metrics.image_psnr(inputs, final_output.cpu()).item()
      ssim = metrics.image_ssim(inputs, final_output.cpu()).item()
      extracted_secret = wm_model._decode(final_output)
      bitAcc = metrics.bit_accuracy(extracted_secret, secret).item()

    psnr_list.append(psnr)
    ssim_list.append(ssim)
    bitacc_list.append(bitAcc)
```

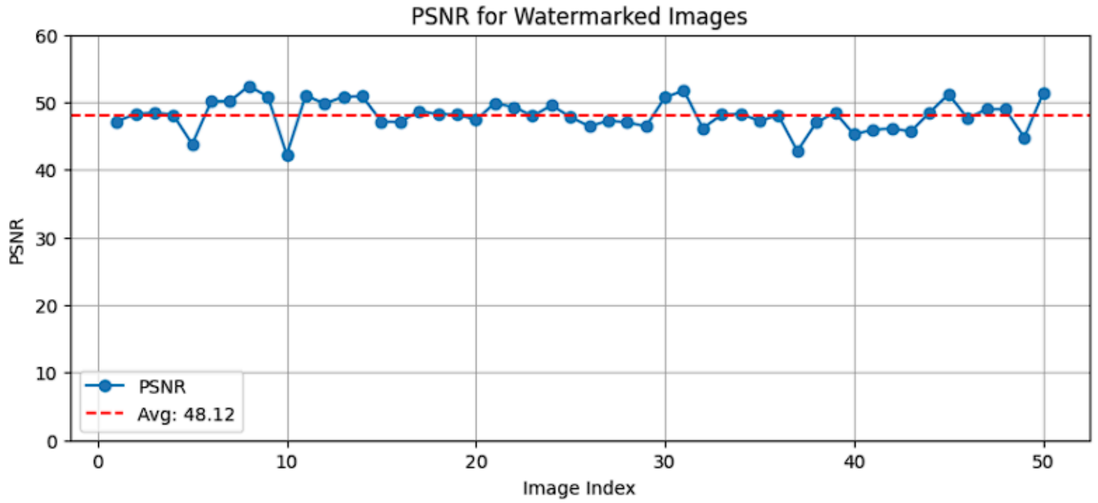
- **inputs**의 shape : (1, 3, 256, 256)
- **시크릿(비트 시퀀스) 생성**
  - 이미지 1장당 UUID 기반 100bit 워터마크를 생성
- **final_output** : 워터마크 삽입된 이미지
- **extracted_secret** : 복원된 워터마크 비트 시퀀스
- **PSNR, SSIM, Bit Acc 측정**
  - 각 리스트에 결과 저장

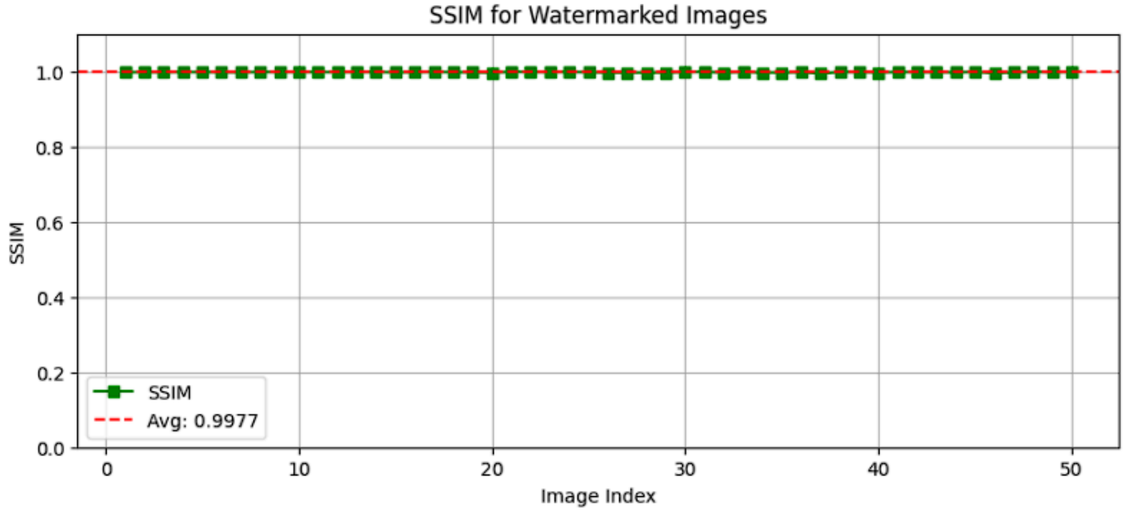
```
    # 이미지 저장
    img_tensor = final_output.squeeze(0).detach().cpu() * 0.5 + 0.5  # [0, 1] 범위로 복원
    img_pil = to_pil_image(img_tensor.clamp(0, 1))
    save_path = os.path.join(save_dir, f'wm_{idx+1:02d}.png')
    img_pil.save(save_path)

    print(f"[{idx+1}/50] 저장 완료 - {save_path}")
    print(f"[{idx+1}/{len(dataloader)}] 완료 - PSNR: {psnr:.2f}, SSIM: {ssim:.4f}, BitAcc: {bitAcc:.4f}")
```

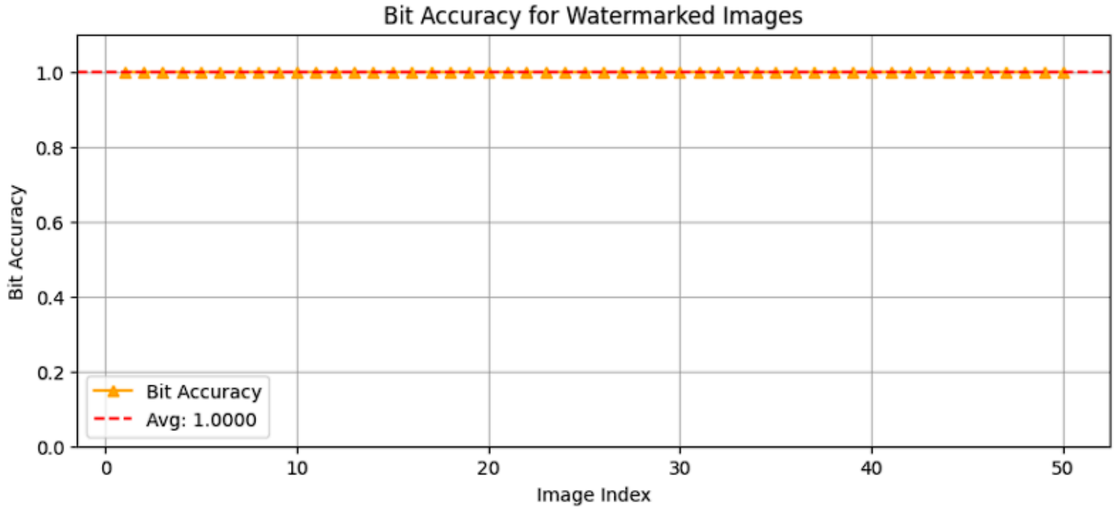
- 워터마킹된 이미지 저장
- 진행상황 출력 코드

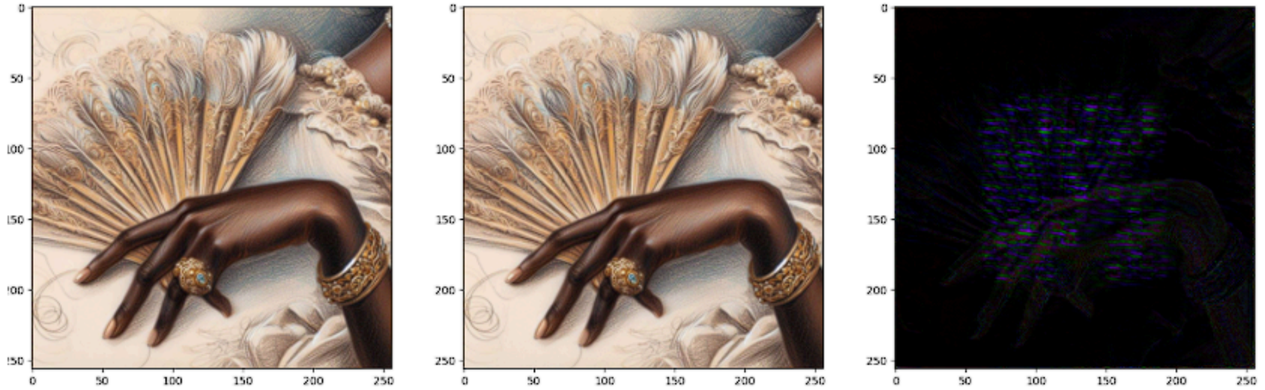
01 Pretrained Model Evaluation

1.1 Clean Image Evaluation









01 Pretrained Model Evaluation

1.2 WAVES attack Evaluation

```python
distortion_strength_paras = dict(
    rotation=(0, 45),
    resizedcrop=(1, 0.5),
    erasing=(0, 0.25),
    brightness=(1, 2),
    contrast=(1, 2),
    blurring=(0, 20),
    noise=(0, 0.1),
    compression=(90, 10),
)


def relative_strength_to_absolute(strength, distortion_type):
    assert 0 <= strength <= 1
    strength = (
        strength
        * (
            distortion_strength_paras[distortion_type][1]
            - distortion_strength_paras[distortion_type][0]
        )
        + distortion_strength_paras[distortion_type][0]
    )
    strength = max(strength, min(*distortion_strength_paras[distortion_type]))
    strength = min(strength, max(*distortion_strength_paras[distortion_type]))
    return strength
```

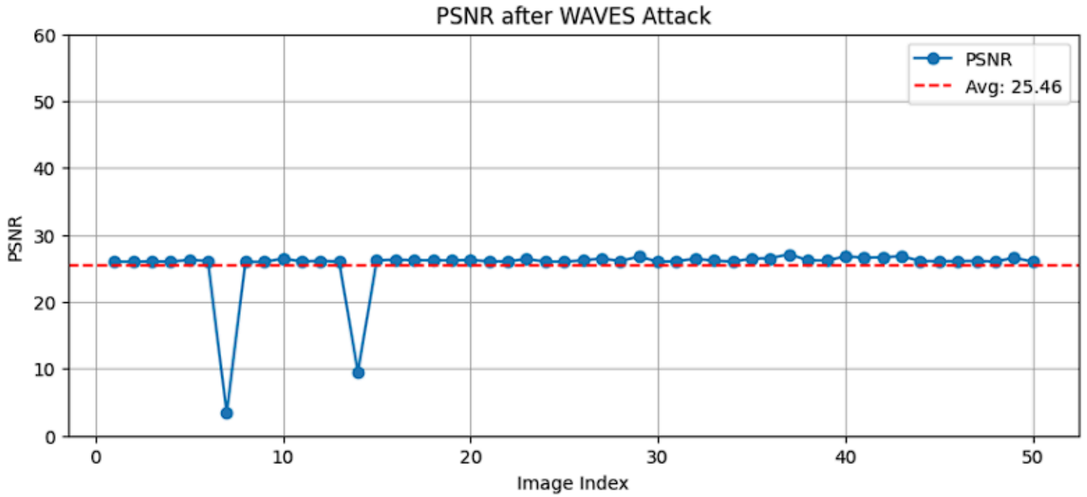
WAVES distortion.py

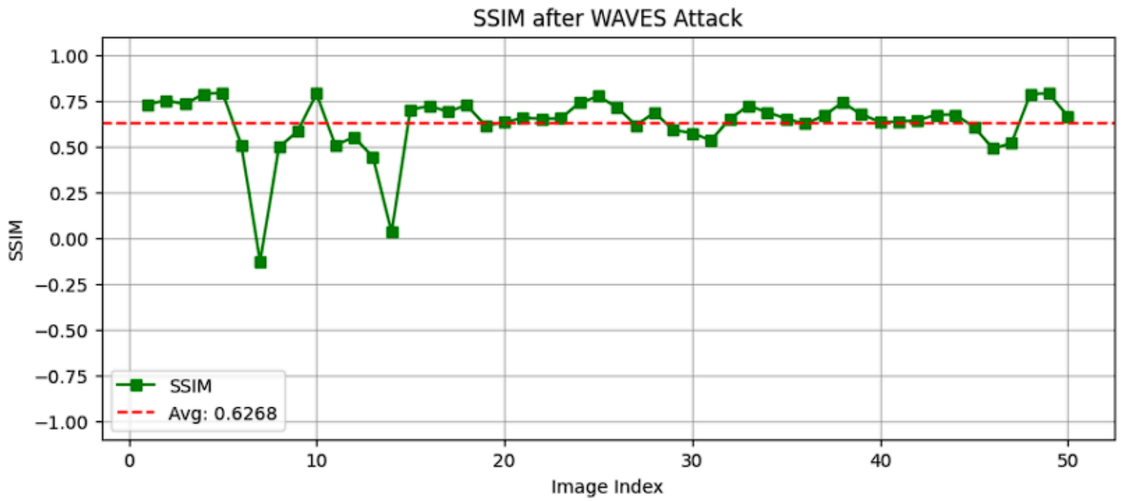
```python
# 1. 사용자 입력 받기
distortion_type = input("공격 유형을 입력하세요 (compression, noise, rotation, etc.): ").strip()
strength = float(input("강도 값을 입력하세요 (예: 0.7은 상대적 강도, 30은 절대값): "))
relative = input("강도를 상대값으로 적용할까요? (yes/no): ").strip().lower() == "yes"

# 2. 폴더 경로
input_dir = "/content/drive/MyDrive/watermarked_img"
output_dir = "/content/drive/MyDrive/attacked_img"
os.makedirs(output_dir, exist_ok=True)

# 3. 공격 실행 루프
for idx in range(1, 51):
    input_path = os.path.join(input_dir, f"wm_{idx:02d}.png")
    output_path = os.path.join(output_dir, f"attacked_{idx:02d}.png")

    # 이미지 로드 및 공격
    pil_img = Image.open(input_path).convert("RGB")
    attacked_img = distortions.apply_single_distortion(
        pil_img,
        distortion_type=distortion_type,
        strength=strength if not relative else None,
        distortion_seed=idx,
    ) if not relative else distortions.apply_distortion(
        [pil_img],
        distortion_type=distortion_type,
        strength=strength,
        distortion_seed=idx,
        same_operation=True,
        relative_strength=True,
        return_image=True,
    )[0]

    # 저장
    attacked_img.save(output_path)
    print(f"[{idx}/50] 저장 완료: {output_path}")
```

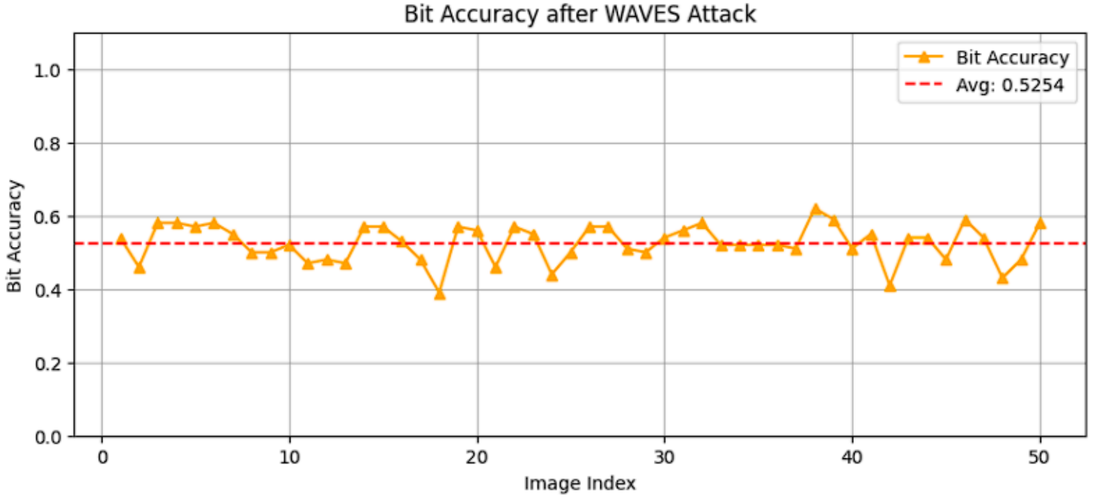
InvisMark 이미지 50장 WAVES 공격 후 저장

01 Pretrained Model Evaluation

1.2 WAVES attack Evaluation







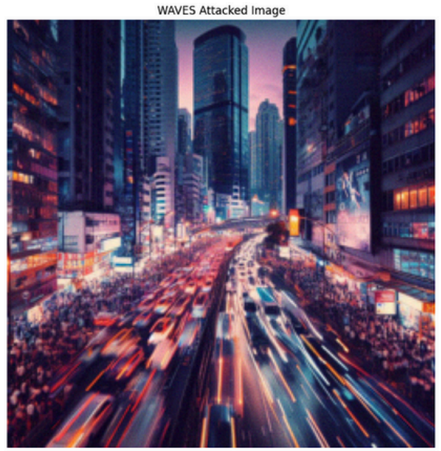
01 Pretrained Model Evaluation

1.2 WAVES attack Evaluation

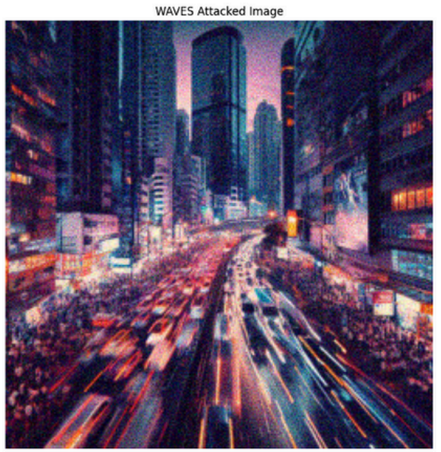
	Clean image	Attacked image
PSNR	48.12	25.46
SSIM	0.9977	0.6268
Bit Acc	1.0000	0.5254

이미지 품질 & Bit Acc 평균값 비교

원본



noise = 0.1



noise = 0.7

- WAVES의 8가지 공격 유형에 대해 강도를 10단계로 나누어서 공격을 진행
  - 오른쪽 그림의 noise=0.7의 경우 이미지 품질이 낮음을 육안으로 확인할 수 있음
  - 이미지 품질을 크게 저하시키지 않는 경우 비트 복원 정확도를 측정해보기 위하여 공격 강도를 10단계로 나눔

01 Pretrained Model Evaluation

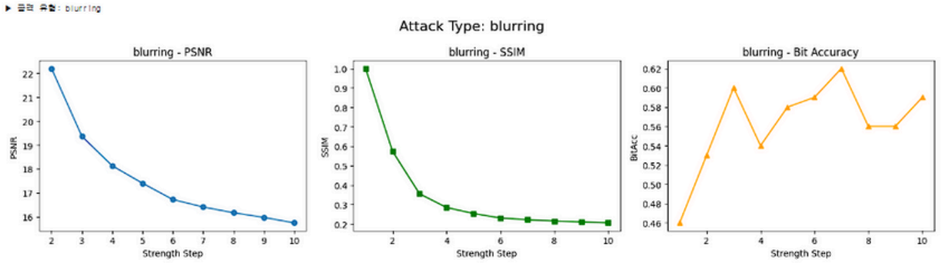
1.2 WAVES attack Evaluation

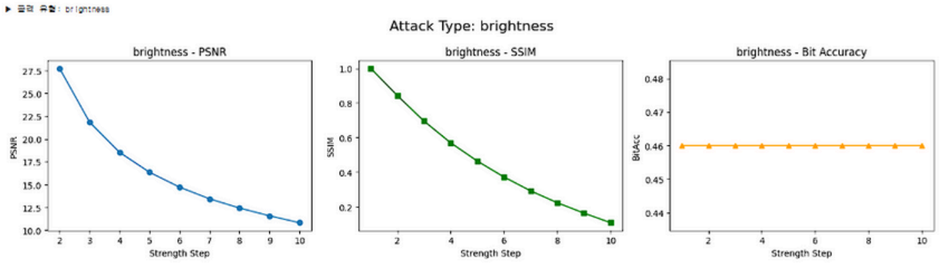
```python
# 1. 경로 설정
input_image_path = "/content/drive/MyDrive/watermarked_img/wm_01.png"
output_root_dir = "/content/drive/MyDrive/attack"
assert os.path.exists(output_root_dir), "attack 폴더가 존재하지 않습니다."

# 2. 공격 유형 정의
distortion_strength_paras = dict(
    rotation=(0, 45),
    resizedcrop=(1, 0.5),
    erasing=(0, 0.25),
    brightness=(1, 2),
    contrast=(1, 2),
    blurring=(0, 20),
    noise=(0, 0.1),
    compression=(90, 10),
)

# 3. 워터마크된 이미지 불러오기
base_img = Image.open(input_image_path).convert("RGB")

# 4. 공격 유형 반복
for distortion_type in distortion_strength_paras.keys():
    print(f"₩n▶ 공격 유형: {distortion_type}")

    # 기존 폴더 확인
    output_dir = os.path.join(output_root_dir, distortion_type)
    os.makedirs(output_dir, exist_ok=True)

    # 강도 10단계 반복
    for i, rel_strength in enumerate(np.linspace(0, 1, 10)):
        attacked_img = distortions.apply_distortion(
            [base_img],
            distortion_type=distortion_type,
            strength=rel_strength,
            distortion_seed=i,
            same_operation=True,
            relative_strength=True,
            return_image=True
        )[0]

        save_path = os.path.join(output_dir, f"{distortion_type}_{i+1:02d}.png")
        attacked_img.save(save_path)
        print(f"  - [{i+1}/10] 저장 완료: {save_path}")
```

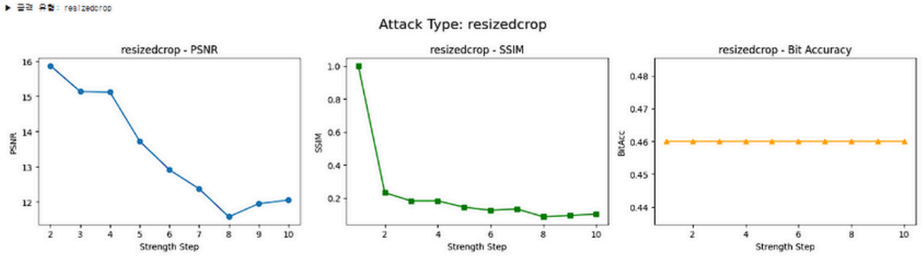
```
공격 유형을 입력하세요 (compression, noise, rotation, etc.): noise
강도 값을 입력하세요 (예: 0.7은 상대적 강도, 30은 절대값): 0.1
강도를 상대값으로 적용할까요? (yes/no): yes
[1/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_01.png
[2/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_02.png
[3/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_03.png
[4/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_04.png
[5/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_05.png
[6/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_06.png
[7/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_07.png
[8/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_08.png
[9/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_09.png
[10/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_10.png
[11/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_11.png
[12/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_12.png
[13/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_13.png
[14/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_14.png
[15/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_15.png
[16/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_16.png
[17/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_17.png
[18/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_18.png
[19/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_19.png
[20/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_20.png
[21/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_21.png
[22/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_22.png
[23/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_23.png
[24/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_24.png
[25/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_25.png
[26/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_26.png
[27/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_27.png
[28/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_28.png
[29/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_29.png
[30/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_30.png
[31/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_31.png
[32/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_32.png
[33/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_33.png
[34/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_34.png
[35/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_35.png
[36/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_36.png
[37/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_37.png
[38/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_38.png
[39/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_39.png
[40/50] 저장 완료: /content/drive/MyDrive/attacked_img/attacked_40.png
```

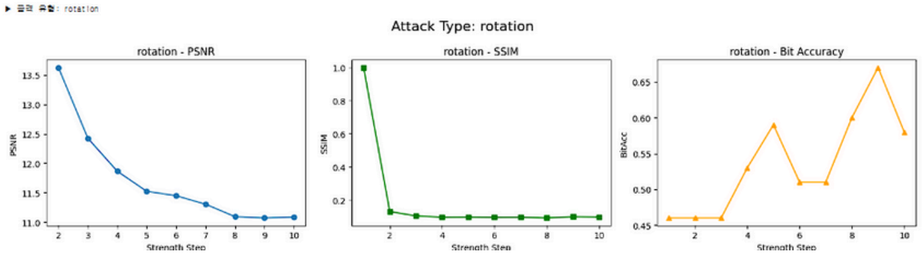
01 Pretrained Model Evaluation

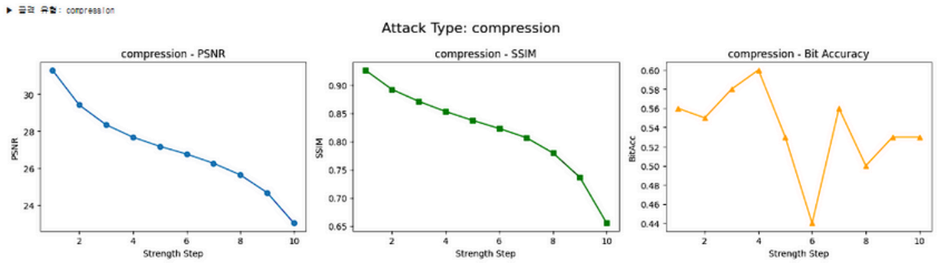
1.2 WAVES attack Evaluation

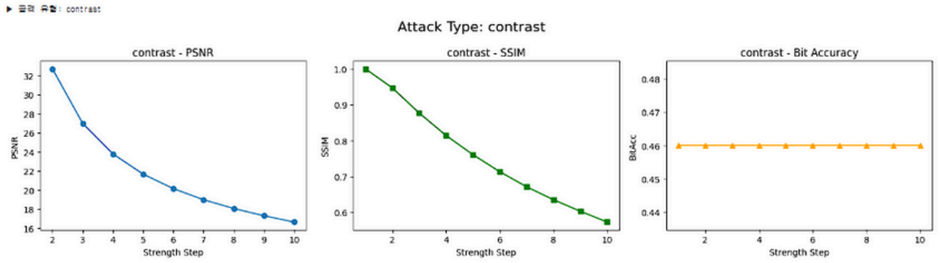


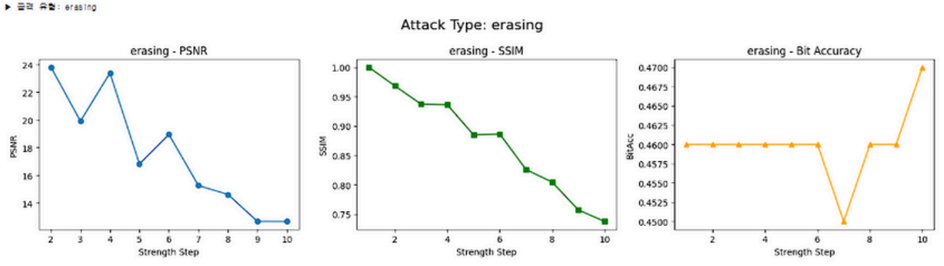


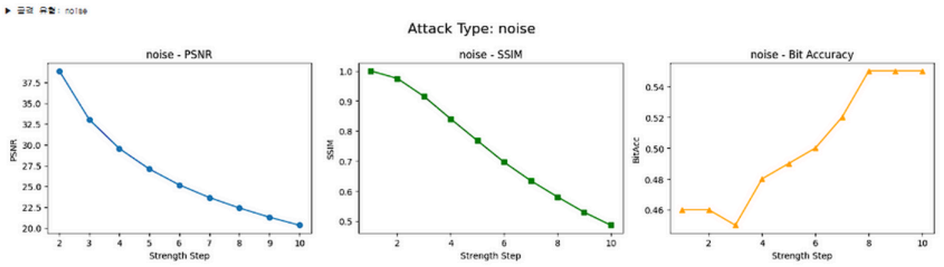












Robust Training Setup

[데이터 셋]

- dalle 3 dataset 10000장

[워터마크 비트 수]

- **100bit**
- **64bit UUID**

[평가 지표]

- 이미지 품질 : PSNR, SSIM
- 워터마크 복원도 : Bit Accuracy

```
cfg = ModelConfig(
    num_epochs=51,
    batch_size=16,
    enc_mode="uuid",
    num_encoded_bits=64,
    beta_transform=5.0,
    noise_start_epoch=5,
    image_shape=(256, 256),
    log_interval=1000,
    log_dir="/content/drive/MyDrive/InvisMark/logs",
    ckpt_path="/content/drive/MyDrive/InvisMark/checkpoints",
)
```

Results: Robustly Trained Model

이미지 1000장

```
cfg = ModelConfig(
    num_epochs=100,
    batch_size=16,
    enc_mode="uuid",
    num_encoded_bits=100,
    beta_transform=1.0,
    noise_start_epoch=5,
    image_shape=(128, 128),
```

- PSNR: 36.50
- SSIM: 0.9844
- Bit Accuracy: 0.5500

이미지 10000장

```
cfg = ModelConfig(
    num_epochs=50,
    batch_size=16,
    enc_mode="uuid",
    num_encoded_bits=100,
    beta_transform=1.0,
    noise_start_epoch=10,
    image_shape=(256, 256),
    log_interval=1000,
```

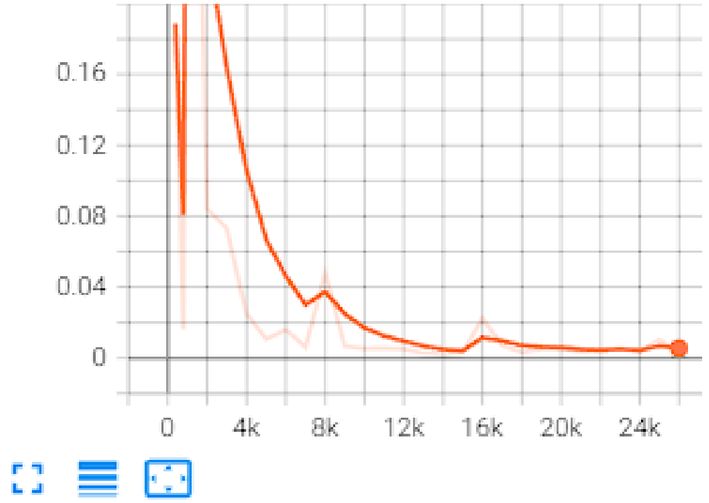
- PSNR: 40.73
- SSIM: 0.9922
- Bit Accuracy: 0.5700

```
cfg = ModelConfig(
    num_epochs=51,
    batch_size=16,
    enc_mode="uuid",
    num_encoded_bits=64,
    beta_transform=5.0,
    noise_start_epoch=5,
    image_shape=(256, 256),
    log_interval=1000,
```

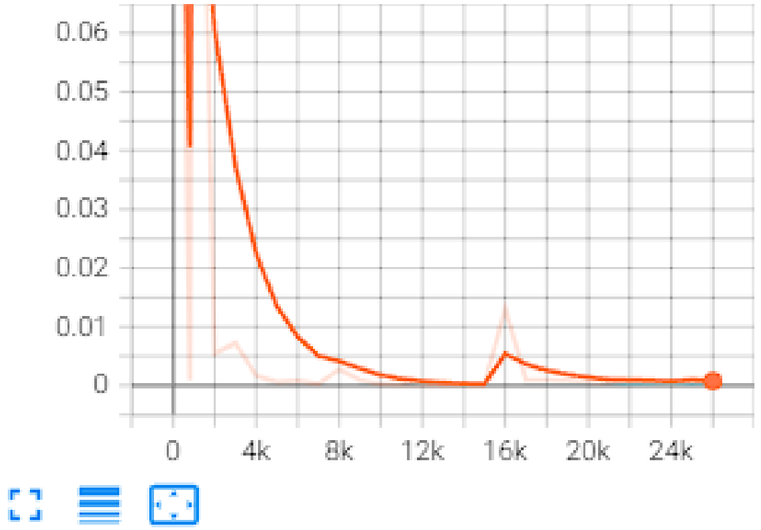
- PSNR: 45.04
- SSIM: 0.9976
- Bit Accuracy: 0.3906

03 Results: Robustly Trained Model

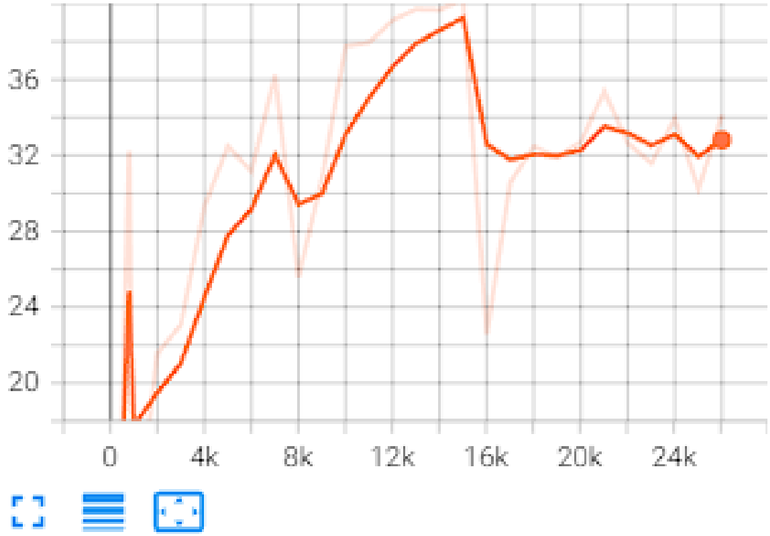
Train/mse_loss
tag: Train/mse_loss



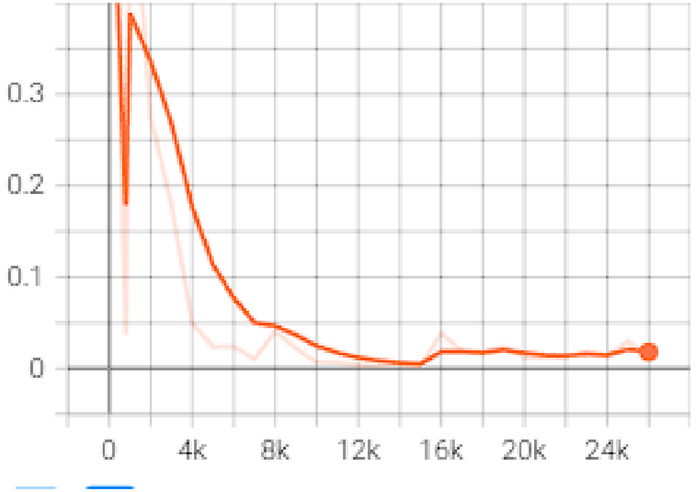
Train/ffl_loss
tag: Train/ffl_loss



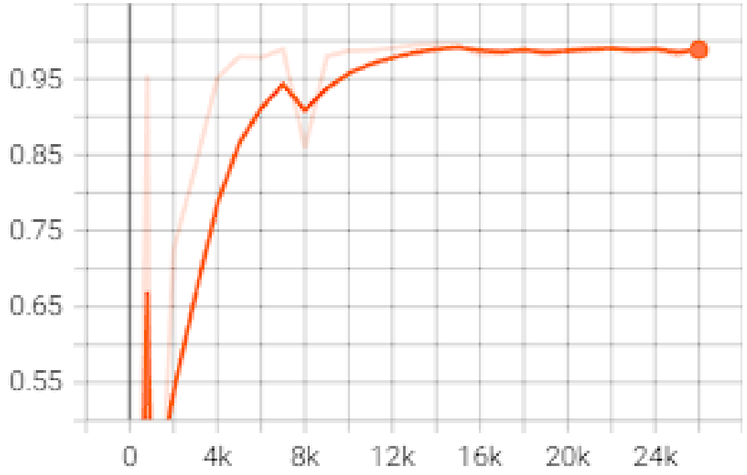
Train/psnr
tag: Train/psnr



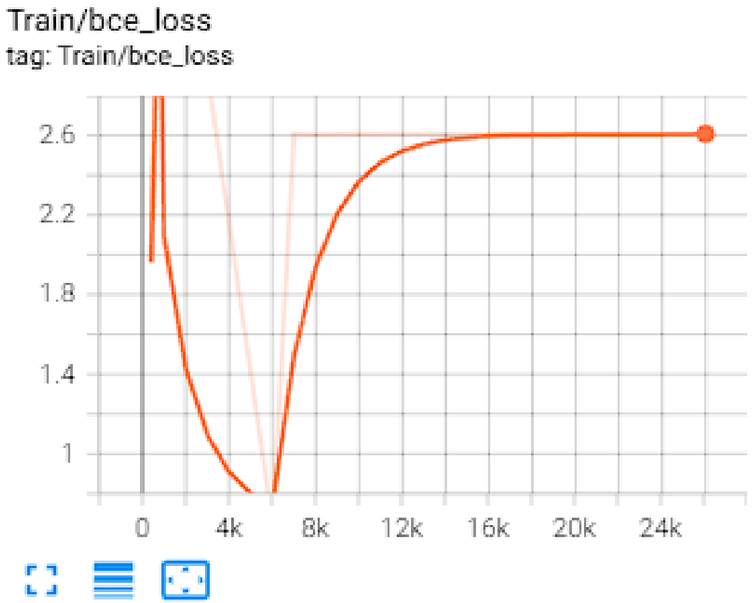
Train/lpips_loss
tag: Train/lpips_loss

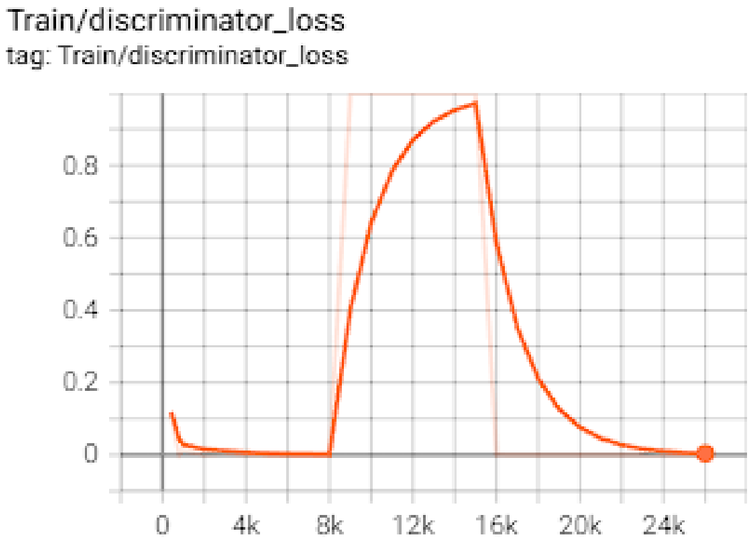


Train/ssim
tag: Train/ssim



03 Results: Robustly Trained Model





[BCE_Loss]

- BCE는 워터마크 비트 복원 정확도에 해당하는 Loss
- 학습 시 전체 50 epoch를 기준으로
  - 5 epoch에 noise 공격을 추가함

→ **워터마크 복원이 안정되기 전에 noise 공격이 들어가서 decoder가 제대로 학습을 못한 것으로 추측됨**

[Discriminator_Loss]

- GAN 구조에서 판별기가 이미지를 real인지 fake인지 구별하는 역할을 함
- 이미지 품질 관련 loss들이 너무 잘 수렴한게 오히려 판별기에게는 real/fake를 구분하게 힘들게 한 것으로 보임
- BCE Loss가 증가하는 동시에 D 학습도 폭주된 것을 보면
  - 노이즈로 이미지 품질이 흔들리는 순간 D가 이미지 구분 기준을 잃었을 가능성이 존재함

Conclusion & Future Work

$$\mathcal{L} = \alpha_q \cdot \mathcal{L}_q(x, \tilde{x}) + \alpha_r \cdot \mathcal{L}_r(\omega, \tilde{\omega})$$

$$\mathcal{L}_q = \mathcal{L}_{\text{YUV}} + \mathcal{L}_{\text{LPIPS}} + \mathcal{L}_{\text{FFL}} + \mathcal{L}_{\text{GAN}}$$

$$\mathcal{L}_r = \text{LBCE}(clean) + \sum_{i \in \text{top-k}} \gamma_i \cdot \text{LBCE}(\text{noisy}_i)$$

- 이미지 품질 손실보다 워터마크 복원 손실의 개선이 필요함

- 개선점
  - 워터마크 복원 안정화 전 공격 지연 : noise_start_epoch를 조정하여, 워터마크 복원이 안정화된 후에 noise 공격을 도입
  - 워터마크 복원에 더 집중하도록 유도 : α_r 고정, α_q를 초반엔 낮추고 후반에 키우기
  - 논문 기반 어려운 공격 중심 학습 유도 : top-k worst noise 기반 robust training 로직 추가

감사합니다