EXPLORING DEEPFAKE DETECTION IN INSTANTID IMAGES GENERATED

- Name: Myllena de Almeida Prado
- Class: CS.7389F Secure CPS - Spring 2025

INTRODUCTION

- Image Generation:
  - Diffusion Models: Common approach for creating images based on a reference picture.

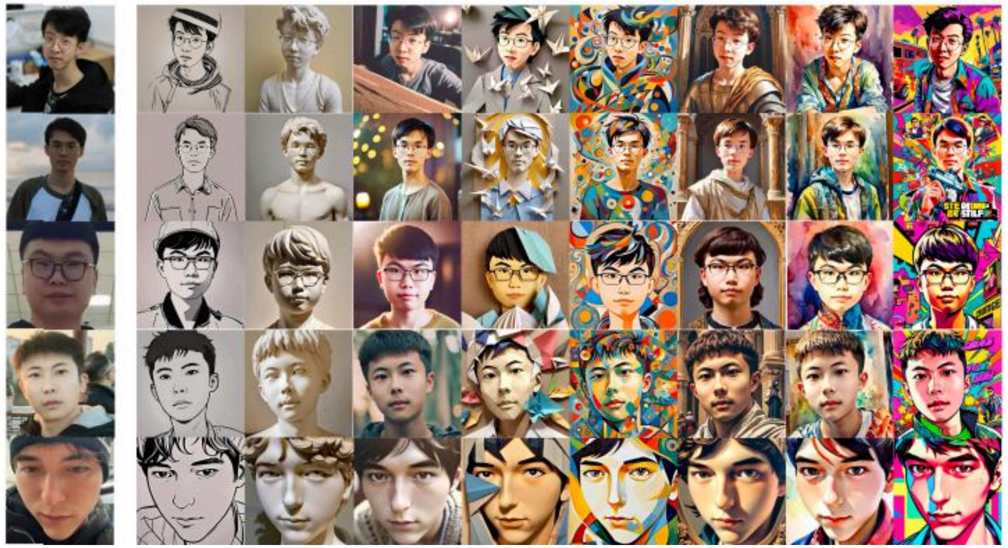
Two Types of Models:

- Fine-tuned Pre-trained Models: High cost, but high fidelity.
- Data-Driven Lightweight Adapters: Efficient, but lower fidelity.

InstantID:

- A plug-and-play module that bridges the gap between fidelity and efficiency.
- Preserves facial identity from a single image, with improved performance.

Images from InstantID paper illustrating the output of model[1]



RISK OF GENERATIVE AI IMAGES

- Privacy Concerns: Single-image models can compromise personal identity.
- DeepFake: Using a single facial image, privacy can be violated
- Research proposal:
  - Objective: Analyze the effectiveness of deepfake detection on InstantID-generated images.
  - Application: Social media and privacy concerns.

Images from InstantID paper illustrating the output of model[1]

METHODOLOGY OVERVIEW

- Dataset
  - LPFF-dataset: 1000 high-quality large-pose facial images.
- Image Generation
  - Use InstantID to generate faces with different poses based on the LPFF dataset.
- Deepfake Detection Models
  - A Comprehensive Benchmark for AI-generated Image Detection: Evaluate the detection models specifically for CNN-generated images.
  - DIRE (ICCV 2023): Detect diffusion-generated images.

DATASET

- LPFF: A Portrait Dataset for Face Generators Across Large Poses [2]
- A dataset containing 19,590 high-quality large-pose face images from Flickr and 19,321 images from Unsplash and Pexels.
- Designed to address pose imbalance in existing face datasets.
- We used 1000 random photos from the Pexels base.

Images of LPFF dataset[2]

IMAGE GENERATION

- InstantID model to generate images based on 5 types of pose.

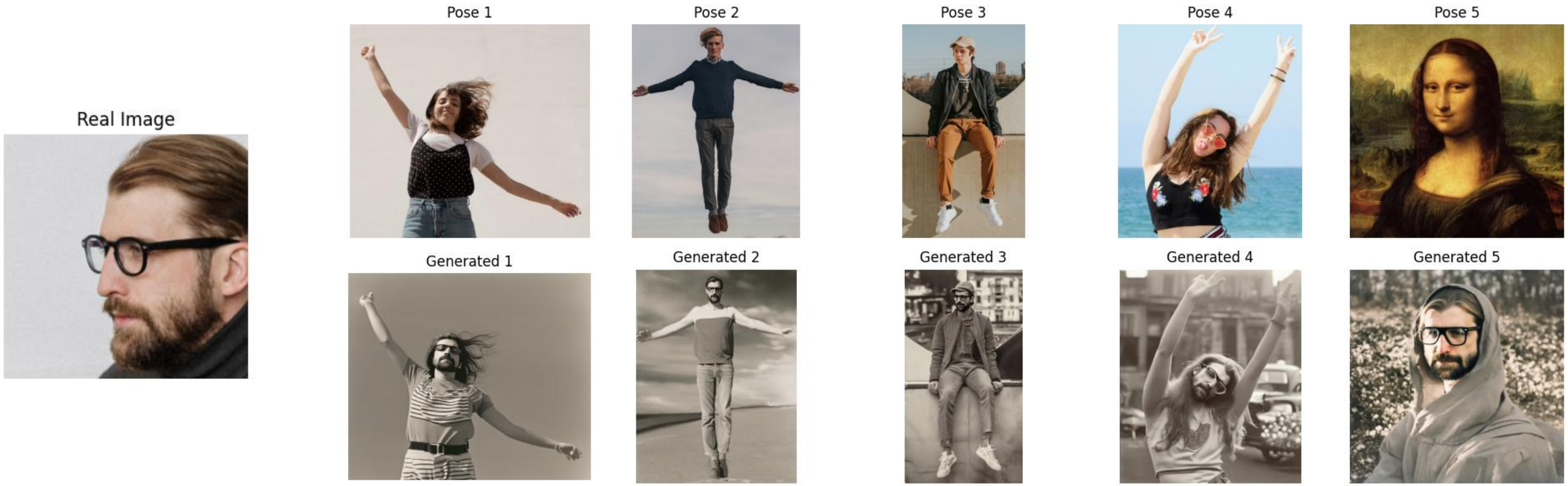


IMAGE GENERATION

- Code: InstantID/infer_fake.py
- Based on the demo available in the github of InstantID: https://github.com/instantX-research/InstantID
- Steps:
  - Read image and pose reference
  - Identify a face in the images
  - Enhance face region
  - Input the images in the model

```python
face_image = load_image(f"real_dataset/{img}")
face_image = resize_img(face_image)

face_info = app.get(cv2.cvtColor(np.array(face_image), cv2.COLOR_RGB2BGR))
face_info = sorted(face_info, key=lambda x:(x['bbox'][2]-x['bbox'][0])*(x['bbox'][3]-x
face_emb = face_info['embedding']

# use another reference image
pose_image = load_image("./examples/poses/{pose}")
pose_image = resize_img(pose_image)

face_info = app.get(cv2.cvtColor(np.array(pose_image), cv2.COLOR_RGB2BGR))
pose_image_cv2 = convert_from_image_to_cv2(pose_image)
face_info = sorted(face_info, key=lambda x:(x['bbox'][2]-x['bbox'][0])*(x['bbox'][3]-x
face_kps = draw_kps(pose_image, face_info['kps'])
width, height = face_kps.size

# use depth control
processed_image_midas = midas(pose_image)
processed_image_midas = processed_image_midas.resize(pose_image.size)

# enhance face region
control_mask = np.zeros([height, width, 3])
x1, y1, x2, y2 = face_info["bbox"]
x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
control_mask[y1:y2, x1:x2] = 255
control_mask = Image.fromarray(control_mask.astype(np.uint8))

image = pipe(
    prompt=prompt,
    negative_prompt=n_prompt,
    image_embeds=face_emb,
    control_mask=control_mask,
    image=[face_kps, processed_image_midas],
    controlnet_conditioning_scale=[0.8,0.8],
    ip_adapter_scale=0.8,
    num_inference_steps=30,
    guidance_scale=5,
).images[0]
name_pose = pose.split('.')[0]
image.save(f'fake_dataset/{name_pose}_{img}')
```

AIGCDETECTBENCHMARK

- Model based on CNN(CNNSpot) [3]
- Code: AIGCDetectBenchmark/eval_all.py
- Based on the demo available in the github : https://github.com/Ekko-zn/AIGCDetectBenchmark/tree/main
- Command:
  - python eval_all.py --model_path models/CNNSpot.pth --detect_method CNNSpot --noise_type blur --blur_sig 1.0 --no_resize --no_crop --batch_size 5
- Dataset configuration: eval_config.py

```
opt = TestOptions().parse(print_options=True) #获取参数类

model_name = os.path.basename(opt.model_path).replace('.pth', '')
results_dir=f"./results/{opt.detect_method}"
mkdir(results_dir)

rows = [["{} model testing on...".format(model_name)],
        ['testset', 'accuracy', 'avg precision', 'r_acc', 'f_acc']]

print("{} model testing on...".format(model_name))
for v_id, val in enumerate(vals):
    opt.dataroot = '{}/{}'.format(dataroot, val)


    model = get_model(opt)
    state_dict = torch.load(opt.model_path, map_location='cpu')


    try:        You, 2 weeks ago • Add folders for each model
        if opt.detect_method in ["FreDect","Gram"]:
            try:
                model.load_state_dict(state_dict['netC'], strict=True)
            except:
                model.load_state_dict({k.replace('module.', ''): v for k, v in s
        elif opt.detect_method == "UnivFD":
            model.fc.load_state_dict(state_dict)
        else:
            model.load_state_dict(state_dict['model'],strict=True)
    except:
        print("[ERROR] model.load_state_dict() error")
    model.to('cuda:1')
    model.eval()


    opt.process_device = torch.device("cpu")
    acc, ap, r_acc, f_acc ,_, _ = validate(model, opt)
    rows.append([val, acc, ap, r_acc, f_acc])
    print("({}) acc: {}; ap: {};  r_acc: {}, f_acc: {}".format(val, acc, ap, r_a
```

UNIVERSAL MODEL

```
parser = argparse.ArgumentParser(formatter_class=argparse.ArgumentDefaultsHelpFormatter)
parser.add_argument('--real_path', type=str, default=None, help='dir name or a pickle')
parser.add_argument('--fake_path', type=str, default=None, help='dir name or a pickle')
parser.add_argument('--data_mode', type=str, default=None, help='wang2020 or ours')
parser.add_argument('--max_sample', type=int, default=1000, help='only check this number of images for both fake/real')

parser.add_argument('--arch', type=str, default='res50')
parser.add_argument('--ckpt', type=str, default='./pretrained_weights/fc_weights.pth')

parser.add_argument('--result_folder', type=str, default='result', help='')
parser.add_argument('--batch_size', type=int, default=128)

parser.add_argument('--jpeg_quality', type=int, default=None, help="100, 90, 80, ... 30. Used to test robustness of our model. Not apply if None")
parser.add_argument('--gaussian_sigma', type=int, default=None, help="0,1,2,3,4.     Used to test robustness of our model. Not apply if None")
```

- Code: UniversalFakeDetect/validate.py
- Based on the demo available in the github : https://github.com/ZhendongWang6/DIRE?tab=readme-ov-file
- Command:
  - python validate.py --arch=CLIP:ViT-L/14 --ckpt=pretrained_weights/fc_weights.pth   --result_folder=clip_vitl14 --real_path Data_InstantID/ --fake_path poseMona/ --data_mode 'ours' --batch_size 10

```
r dataset_path in (dataset_paths):
    set_seed()

    dataset = RealFakeDataset(  dataset_path['real_path'],
                                dataset_path['fake_path'],
                                dataset_path['data_mode'],
                                opt.max_sample,
                                opt.arch,
                                jpeg_quality=opt.jpeg_quality,
                                gaussian_sigma=opt.gaussian_sigma,
                                )

    loader = torch.utils.data.DataLoader(dataset, batch_size=opt.batch_size, shuffle=False, num_workers=4)
    ap, r_acc0, f_acc0, acc0, r_acc1, f_acc1, acc1, best_thres = validate(model, loader, find_thres=True)
    print('ap:', ap, ' r_acc0:', r_acc0, ' f_acc0:', f_acc0, ' acc0:', acc0, ' r_acc1:', r_acc1, ' f_acc1:', f_acc1
    with open( os.path.join(opt.result_folder,'ap.txt'), 'a') as f:
        f.write('InstantID: ' + str(round(ap*100, 2))+'\n' )
        print('InstantID: ' + str(round(ap*100, 2))+'\n' )

    with open( os.path.join(opt.result_folder,'acc0.txt'), 'a') as f:
        f.write('InstantID: ' + str(round(r_acc0*100, 2))+'  '+str(round(f_acc0*100, 2))+'  '+str(round(acc0*100, 2
        print('InstantID: ' + str(round(r_acc0*100, 2))+'  '+str(round(f_acc0*100, 2))+'  '+str(round(acc0*100, 2))
```

RESULTS

- Code: results_analyses.ipynb

```
import pandas as pd
from sklearn.metrics import classification_report

# Load the CSV files
agi_df = pd.read_csv('AIGCDetectBenchmark/results.csv')
universal_df = pd.read_csv('UniversalFakeDetect/results.csv')

# Extract labels and predictions
agi_labels = agi_df["Label"]
agi_preds = agi_df["y_pred"]
universal_labels = universal_df["Label"]
universal_preds = universal_df["y_pred"]

# Generate classification reports
agi_report = classification_report(agi_labels, agi_preds, output_dict=True)
universal_report = classification_report(universal_labels, universal_preds, output_dict=True)
```

	Precision	Recall	F1
Real	0.5	0.6	0.55
Fake	0.5	0.4	0.45

AIGCDetectBenchmark – CNNSpot[3]

	Precision	Recall	F1
Real	0.5	0.9	0.64
Fake	0.5	0.1	0.17

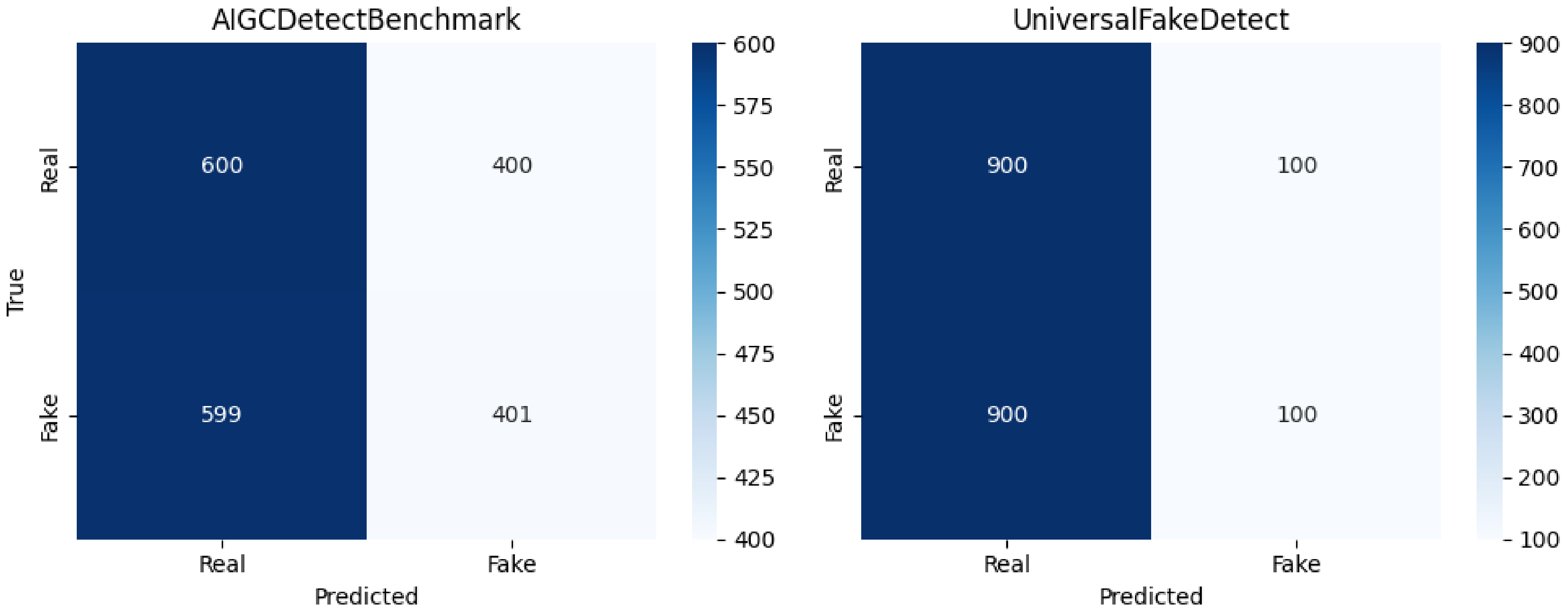
Universal Model [4]

RESULTS

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Compute confusion matrices
agi_cm = confusion_matrix(agi_labels, agi_preds)
universal_cm = confusion_matrix(universal_labels, universal_preds)
```

- Code: results_analyses.ipynb



CONCLUSION

- Deepfake Detection Challenges:
  - The detection models face significant difficulty in accurately identifying generated images
- Importance of Further Development:
  - There is a critical need for the continued advancement of deepfake detection technologies to stay ahead of rapidly improving generative models.

REFERENCES

- [1]Wang, Q., “InstantID: Zero-shot Identity-Preserving Generation in Seconds”, <i>arXiv e-prints</i>, Art. no. arXiv:2401.07519, 2024. doi:10.48550/arXiv.2401.07519.
- [2]Wu, Y., Zhang, J., Fu, H., and Jin, X., “LPFF: A Portrait Dataset for Face Generators Across Large Poses”, <i>arXiv e-prints</i>, Art. no. arXiv:2303.14407, 2023. doi:10.48550/arXiv.2303.14407.
- [3] Wang, S.-Y., Wang, O., Zhang, R., Owens, A., and Efros, A. A., “CNN-generated images are surprisingly easy to spot... for now”, <i>arXiv e-prints</i>, Art. no. arXiv:1912.11035, 2019. doi:10.48550/arXiv.1912.11035.
- [4] Wang, Z., “DIRE for Diffusion-Generated Image Detection”, <i>arXiv e-prints</i>, Art. no. arXiv:2303.09295, 2023. doi:10.48550/arXiv.2303.09295.