Cake-able diffusion

: Cake, not Fake

Spring 2025 Generative AI

온재현, 이현서, 조효원 (아트&테크놀로지)

프로젝트 배경

Generative AI

ANT5010/AATG010

Autoregressive Models

Haedong Jeong

Creative Artificial Intelligence
ART&TECHNOLOGY SOGANG UNIVERSITY
SOGANG UNIVERSITY

Some materials from lectures by Prof. Volodymyr Kuleshov from Cornell University and Prof. Seungchul Lee from KAIST

프로젝트 배경

NADE: Neural Autoregressive Density Estimation

- This yields a model called Neural Autoregressive Density Estimation (NADE)

- Samples on the left (binarized MNIST)
- Conditional probabilities $\hat{x}_i$ on the right (probability for each pixel)

프로젝트 배경

Summary

- Autoregressive model

$$p_\theta(x) = \prod_{i=1}^{d} p_\theta(x_i | x_{1:i-1})$$

- Train a network by using cross entropy loss like classification task
- Tokenization is key components to represent the autoregressive model
  - Any modality (text or Image or …) can be quantized → tokens
- Causal Masked Neural Models
  - E.g., Transformer with causal mask
  - Generation is still slow but generation flow is resonable

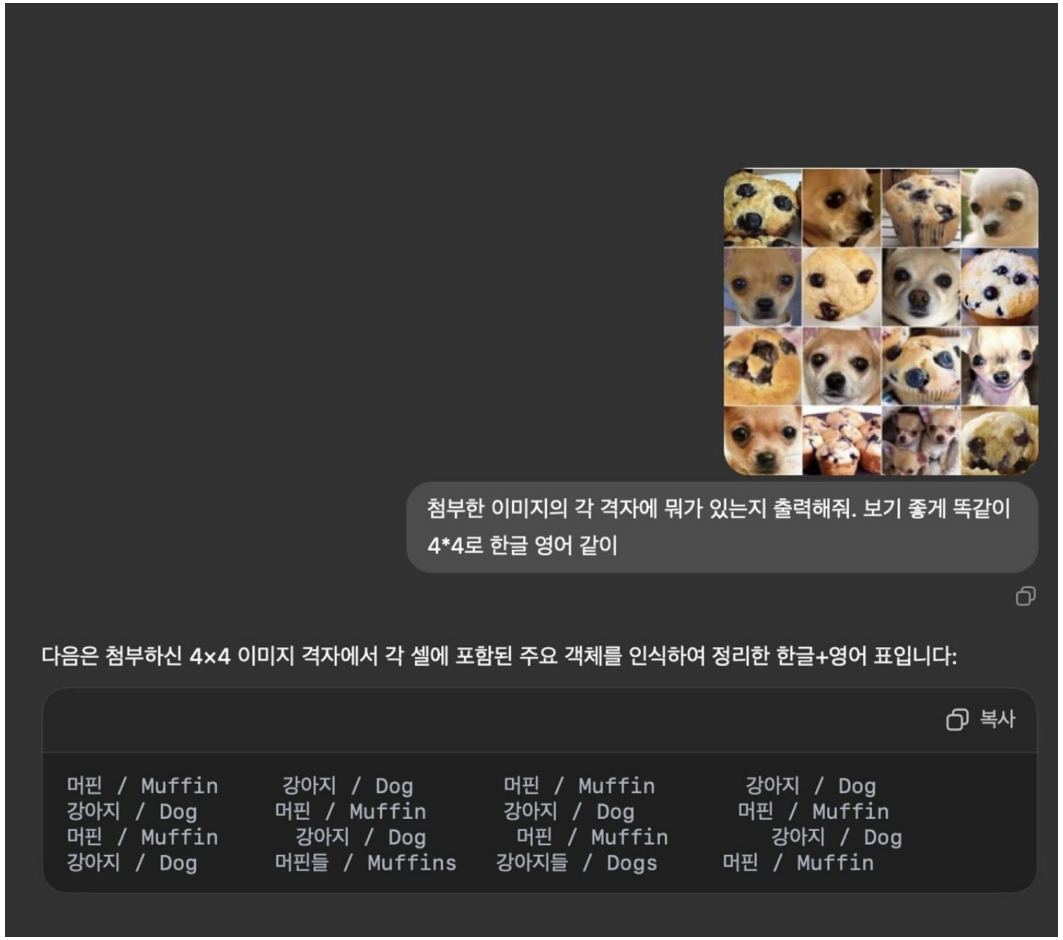
프로젝트 배경

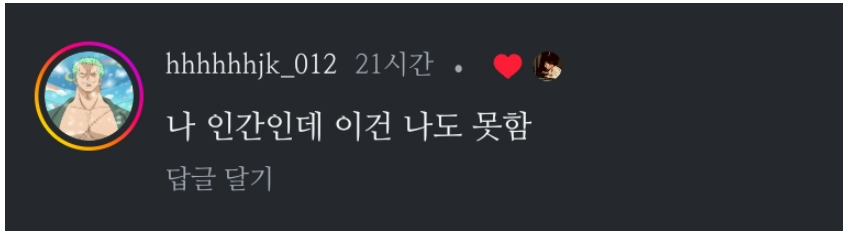
재밌는 Classification Task?

프로젝트 배경

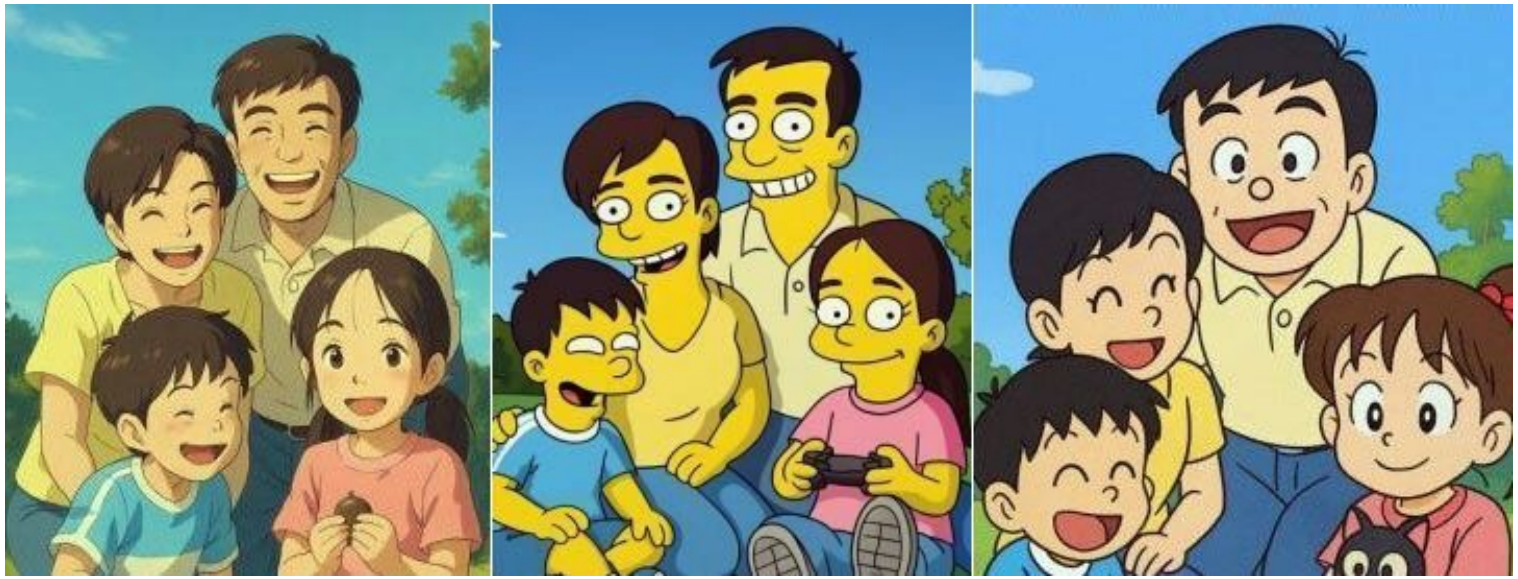
Chihuahua or muffin?

프로젝트 배경


첨부한 이미지의 각 격자에 뭐가 있는지 출력해줘. 보기 좋게 똑같이 4*4로 한글 영어 같이
다음은 첨부하신 4×4 이미지 격자에서 각 셀에 포함된 주요 객체를 인식하여 정리한 한글+영어 표입니다:
복사
머핀 / Muffin
강아지 / Dog
머핀 / Muffin
강아지 / Dog
강아지 / Dog
머핀 / Muffin
강아지 / Dog
머핀 / Muffin
머핀 / Muffin
강아지 / Dog
머핀 / Muffin
강아지 / Dog
강아지 / Dog
머핀들 / Muffins
강아지들 / Dogs
머핀 / Muffin


hhhhhhjk_012 21시간
나 인간인데 이건 나도 못함
답글 달기

프로젝트 배경

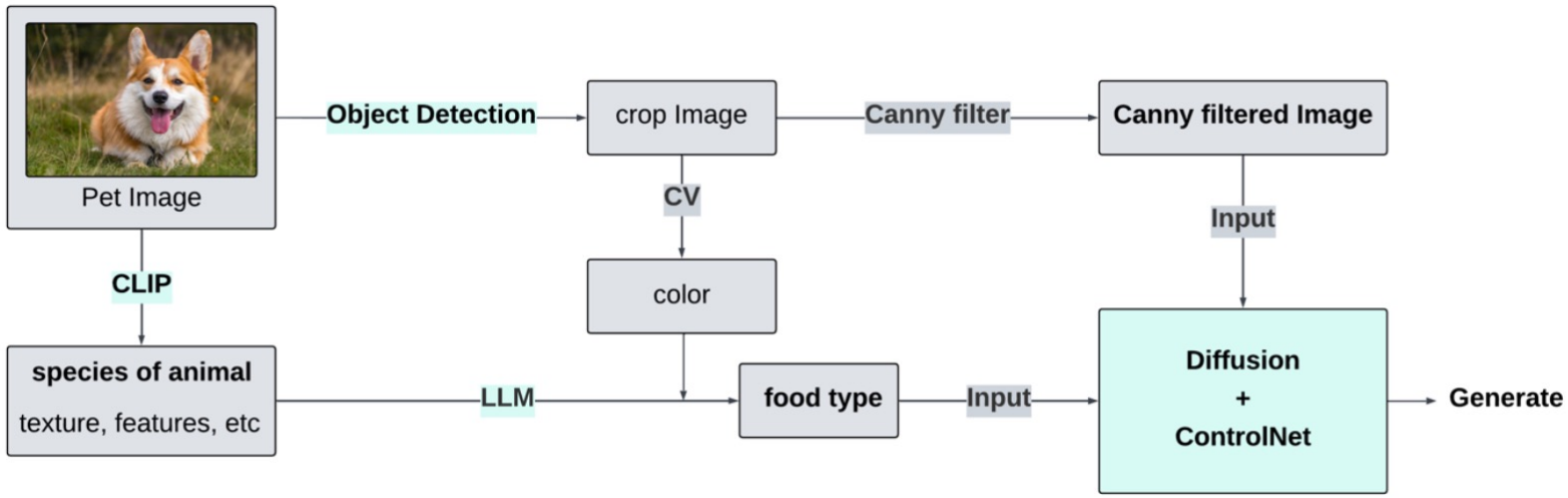


프로젝트

Cake-able diffusion

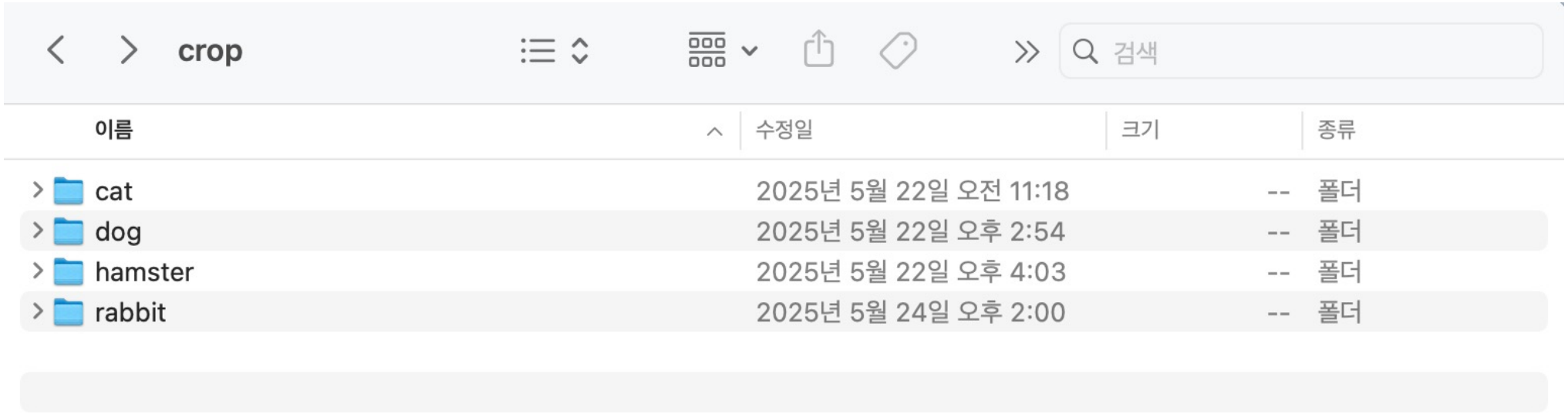
: Cake, not Fake

Project Pipeline

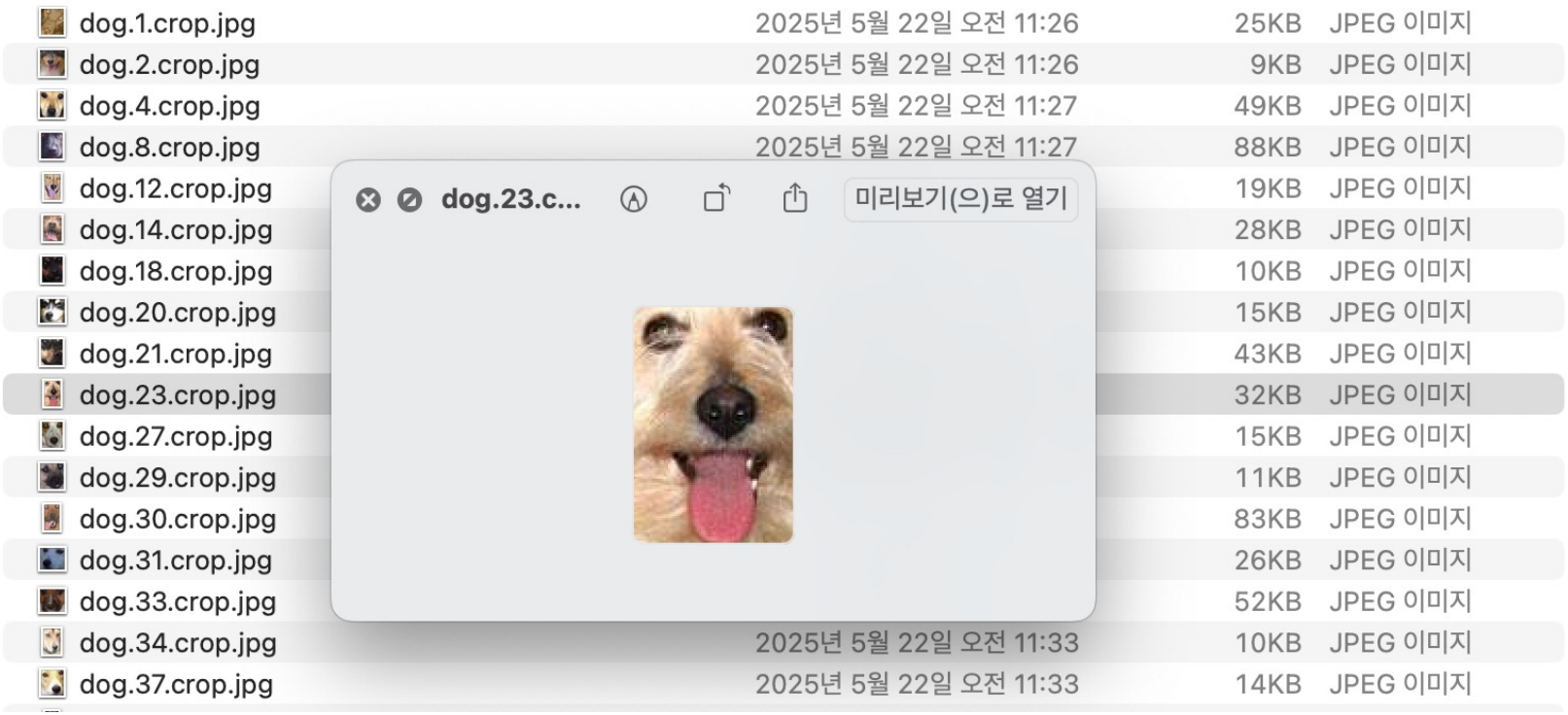


Object Detection / Crop Image (YOLO)

Object Detection / Crop Image (YOLO)



Object Detection / Crop Image (YOLO)



Object Detection / Crop Image (YOLO)

```
[ ]  # 00. cropset 전처리 과정 (좌표 뽑아서 csv 변환)
```

```python
def make_coordinate(original_image_path, crop_image_path, min_match_count=10):
    original_img = cv2.imread(original_image_path)
    crop_img = cv2.imread(crop_image_path)
    orb = cv2.ORB_create(nfeatures=1000)
    kp1, des1 = orb.detectAndCompute(crop_img, None)
    kp2, des2 = orb.detectAndCompute(original_img, None)

    if des1 is None or des2 is None or len(des1) < min_match_count or len(des2) < min_match_count :
        return None

    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    matches = bf.match(des1, des2)
    matches = sorted(matches, key=lambda x: x.distance)

    if len(matches) > min_match_count:
        src_pts = np.float32([kp1[m.queryIdx].pt for m in matches]).reshape(-1, 1, 2)
        dst_pts = np.float32([kp2[m.trainIdx].pt for m in matches]).reshape(-1, 1, 2)
        M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)

        if M is None:
            return None

        h, w = crop_img.shape[:2]
        pts = np.float32([[0, 0], [0, h - 1], [w - 1, h - 1], [w - 1, 0]]).reshape(-1, 1, 2)
        dst_corners = cv2.perspectiveTransform(pts, M)
        x_coords = dst_corners[:,0,0]
        y_coords = dst_corners[:,0,1]

        bbox = (int(np.min(x_coords)), int(np.min(y_coords)),
                int(np.max(x_coords)), int(np.max(y_coords)))

        return bbox
```

Object Detection / Crop Image (YOLO)

```
annotations = [] # 이미지 경로, 좌표 4개, 클래스 이름 형태

# for animal in ['dog', 'cat', 'rabbit', 'hamster'] :
for animal in ['dog'] :
  for i in range(len(get_file_names("crop", animal))):
    file_list = get_file_names("crop", animal)
    crop_image_path = Path(f'/content/cropset/crop/{animal}/{file_list[i]}')
    original_image_path = Path(f'/content/cropset/train/{animal}/{file_list[i][:-8] + file_list[i][-3:]}')
    try :
      bbox = make_coordinate(original_image_path, crop_image_path)
    except :
      print(crop_image_path)
    if bbox:
        annotations.append({
          "image_path": original_image_path,
          "x_min": bbox[0],
          "y_min": bbox[1],
          "x_max": bbox[2],
          "y_max": bbox[3],
          "class_name": f"{animal}_face"
        })

import csv

with open(output_annotations_file, 'w', newline='') as f:
    writer = csv.writer(f)
    writer.writerow(["image_path", "x_min", "y_min", "x_max", "y_max", "class_name"])
    for ann in annotations:
        writer.writerow([ann["image_path"], ann["x_min"], ann["y_min"], ann["x_max"], ann["y_max"], ann["class_name"]])
```

Object Detection / Crop Image (YOLO)

```
results = model.train(
    data='/content/animal_dataset.yaml',
    epochs=500,
    imgsz=1280,
    batch=24,
    name='animal_face_yolov11n_py_run1',
    device=0
)
```

```
      229/500      13.1G      1.572      2.088      1.987         19       1280: 100%|██████████| 6/6 [00:04<00:00,  1.30it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00,  2.07it/s]                 all         34         34      0.521

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      230/500      13.1G      1.673      2.181      2.023         17       1280: 100%|██████████| 6/6 [00:04<00:00,  1.30it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00,  2.02it/s]                 all         34         34      0.622

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      231/500      13.1G      1.639      2.166      2.036         22       1280: 100%|██████████| 6/6 [00:04<00:00,  1.30it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00,  2.05it/s]                 all         34         34      0.552

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      232/500      13.1G      1.632      2.171      1.993         24       1280: 100%|██████████| 6/6 [00:04<00:00,  1.31it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00,  2.04it/s]                 all         34         34      0.452

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      233/500      13.1G      1.602      2.153       1.98         20       1280: 100%|██████████| 6/6 [00:04<00:00,  1.30it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00,  2.06it/s]                 all         34         34      0.308

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      234/500      13.1G      1.609      2.094       1.94         18       1280: 100%|██████████| 6/6 [00:04<00:00,  1.31it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00,  2.06it/s]                 all         34         34      0.267

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      235/500      13.1G      1.698      2.159      2.032         30       1280: 100%|██████████| 6/6 [00:04<00:00,  1.30it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00,  2.06it/s]                 all         34         34      0.166

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      236/500      13.1G      1.706       2.26      2.084         22       1280: 100%|██████████| 6/6 [00:04<00:00,  1.30it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00,  2.04it/s]                 all         34         34      0.302

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      237/500      13.1G      1.677      2.308      2.031         19       1280: 100%|██████████| 6/6 [00:04<00:00,  1.30it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00,  2.07it/s]                 all         34         34      0.492
EarlyStopping: Training stopped early as no improvement observed in last 100 epochs. Best results observed at epoch 137, best model saved as best.pt.
To update EarlyStopping(patience=100) pass a new patience value, i.e. `patience=300` or use `patience=0` to disable EarlyStopping.
```

Object Detection / Crop Image (YOLO)

```
yolo_model = YOLO('best.pt')

def image_crop(image_path) :

  results = yolo_model(image_path, conf=0.03)
  boxes = results[0].boxes
  names = yolo_model.names
  image = Image.open(image_path)

  if boxes and boxes.conf.numel() > 0:
    top_idx = torch.argmax(boxes.conf)
    top_box = boxes[top_idx]

    cls_id = int(top_box.cls[0])
    conf = float(top_box.conf[0])
    x1, y1, x2, y2 = map(int, top_box.xyxy[0])

    return image.crop((x1, y1, x2, y2))

  return None
```

Image Description (CLIP)

```
device = "cuda" if torch.cuda.is_available() else "cpu"
clip_model, clip_preprocess = clip.load('ViT-B/32', device=device)

animal_vocabulary = [
    "dog", "cat", "rabbit", "hamster"
]

texture_vocabulary = [
    "fluffy", "furry", "smooth", "silky", "woolly", "shaggy", "sleek", "curly", "soft", "plush",
    "velvety", "bald", "short-haired", "long-haired", "fuzzy", "matted", "bristly", "puffy", "glossy", "patchy",
    "damp", "hairless", "downy", "dense-coated", "fine-haired", "frizzy", "straight-haired", "wavy", "coarse", "tufted",
    "scruffy", "greasy", "soaked", "clumpy", "velvety-coated", "semi-long-haired", "floppy-coated", "clean-coated", "spotty", "blotchy",
    "shiny", "messy", "wispy", "unkempt", "even-coated", "double-coated", "thin-coated", "thick-coated", "baby-furred", "powder-soft"
]


feature_vocabulary = [
    "tiny", "chubby", "round", "pointy", "droopy", "bright-eyed", "snub-nosed", "button-nosed",
    "stubby", "floppy-eared", "perky-eared", "tail-wagging", "bushy-tailed", "alert", "lazy-looking", "wrinkled",
    "freckled", "big-headed", "mini-sized", "wide-eyed", "long-legged", "short-legged", "pudgy", "nimble", "squat",
    "stocky", "grinning", "toothy", "squishy", "slinky", "chubby-cheeked", "stiff-looking", "dopey-eyed",
    "hunchbacked", "arched", "lanky", "beady-eyed", "double-chinned", "jawless", "clumsy", "gap-toothed", "bug-eyed", "pigeon-toed"
]

animal_inputs = clip.tokenize(animal_vocabulary).to(device)
texture_inputs = clip.tokenize(texture_vocabulary).to(device)
feature_inputs = clip.tokenize(feature_vocabulary).to(device)
```

Image Description (CLIP)

```python
def image_to_vocab(image_path, top_n : int = 5):
  image = clip_preprocess(Image.open(image_path)).unsqueeze(0).to(device)

  with torch.no_grad():
    image_features = clip_model.encode_image(image)
    animal_features = clip_model.encode_text(animal_inputs)
    texture_features = clip_model.encode_text(texture_inputs)
    feature_features = clip_model.encode_text(feature_inputs)

  image_features /= image_features.norm(dim=-1, keepdim=True)
  animal_features /= animal_features.norm(dim=-1, keepdim=True)
  animal_similarity = (image_features @ animal_features.T).squeeze(0)

  texture_features /= texture_features.norm(dim=-1, keepdim=True)
  texture_similarity = (image_features @ texture_features.T).squeeze(0)

  feature_features /= feature_features.norm(dim=-1, keepdim=True)
  feature_similarity = (image_features @ feature_features.T).squeeze(0)

  animal_threshold = max(mean([float(i) for i in list(animal_similarity)]), 0.2)
  animal_similarity = [0 if value < animal_threshold else value.item() for value in animal_similarity]
  animal_indexes = sorted(range(len(animal_similarity)), key=lambda i: animal_similarity[i], reverse=True)[:1]

  texture_threshold = max(mean([float(i) for i in list(texture_similarity)]), 0.2)
  texture_similarity = [0 if value < texture_threshold else value.item() for value in texture_similarity]
  texture_indexes = sorted(range(len(texture_similarity)), key=lambda i: texture_similarity[i], reverse=True)[:top_n]

  feature_threshold = max(mean([float(i) for i in list(feature_similarity)]), 0.2)
  feature_similarity = [0 if value < feature_threshold else value.item() for value in feature_similarity]
  feature_indexes = sorted(range(len(feature_similarity)), key=lambda i: feature_similarity[i], reverse=True)[:top_n]

  if(mean([float(i) for i in list(animal_similarity)]) == 0.0) : return None, [None], [None]

  return animal_vocabulary[animal_indexes[0]], [texture_vocabulary[i] for i in texture_indexes], [feature_vocabulary[i] for i in feature_indexes]
```

Image Color (CV2 + KMeans)

```
def rgb_to_hex(rgb): return '#%02x%02x%02x' % rgb

def get_image_color(image_path) :
  kmeans = KMeans(n_clusters=3, n_init='auto')
  kmeans.fit(cv2.resize(cv2.cvtColor(cv2.imread(image_path), cv2.COLOR_BGR2RGB), (64, 64)).reshape((-1, 3)))
  unique, counts = np.unique(kmeans.labels_, return_counts=True)
  return rgb_to_hex(tuple(kmeans.cluster_centers_[unique[np.argmax(counts)]].astype(int)))
```

Making ideas (GPT Assistants API)

```python
assistant = client.beta.assistants.retrieve("asst_[redacted]")

class SimpleHandler(AssistantEventHandler):
    @override
    def on_text_delta(self, delta, snapshot):
        pass

def get_dessert_list(animal_label, texture_labels, feature_labels, dom_hex, image_path) :
  image_file = client.files.create(file=open(image_path, "rb"), purpose="assistants")

  user_prompt = f"""
  Suggest 10 dessert names that visually or conceptually resemble the following animal traits.

  Animal: {animal_label}
  Textures: {', '.join(texture_labels)}
  Features: {', '.join(feature_labels)}
  Dominant Color: {dom_hex}

  Respond ONLY in JSON list like:
  ["dessert_name1", "dessert_name2", ..., "dessert_name10"]
  """

  thread = client.beta.threads.create()
  client.beta.threads.messages.create(
      thread_id=thread.id,
      role="user",
      content=[
          {
              "type": "text",
              "text": user_prompt
          },
          {
              "type": "image_file",
              "image_file": {
                  "file_id": image_file.id
              }
          }
      ]
  )

  with client.beta.threads.runs.stream(
      thread_id=thread.id,
      assistant_id=assistant.id,
      instructions="Respond with only the dessert name as JSON. No explanation or extra formatting.",
      event_handler=SimpleHandler(),
  ) as stream:
      stream.until_done()

  thread_messages = client.beta.threads.messages.list(thread_id=thread.id)


  content_blocks = thread_messages.data[0].content
  for block in content_blocks:
      if block.type == "text":
          text = block.text.value
          break

  return json.loads(text)
```

Image Generation (cv2 + Stable Diffusion + ControlNet)

```python
class SD3CannyImageProcessor(VaeImageProcessor):
    def __init__(self):
        super().__init__(do_normalize=False)
    def preprocess(self, image, **kwargs):
        image = super().preprocess(image, **kwargs)
        image = image * 255 * 0.5 + 0.5
        return image
    def postprocess(self, image, do_denormalize=True, **kwargs):
        do_denormalize = [True] * image.shape[0]
        image = super().postprocess(image, **kwargs, do_denormalize=do_denormalize)
        return image

def make_canny_image(image):
  image_np = np.array(image)
  if image_np.ndim == 3 and image_np.shape[2] == 3:
      gray = cv2.cvtColor(image_np, cv2.COLOR_RGB2GRAY)
  else:
      gray = image_np
  edges = cv2.Canny(gray, 200, 400)
  edge_image = Image.fromarray(edges, mode='L')
  return edge_image.convert("RGB")

imgProcessor = SD3CannyImageProcessor()
controlnet = ControlNetModel.from_pretrained("diffusers/controlnet-canny-sdxl-1.0", torch_dtype=torch.float16)
vae = AutoencoderKL.from_pretrained("madebyollin/sdxl-vae-fp16-fix", torch_dtype=torch.float16)
pipe = StableDiffusionXLControlNetPipeline.from_pretrained("stabilityai/stable-diffusion-xl-base-1.0", controlnet=controlnet, vae=vae, torch_dtype=torch.float16)
pipe = pipe.to('cuda')
pipe.image_processor = imgProcessor
```

Deploy (Gradio)

```
import gradio as gr

def main_function(image_path : str):
  animal_label, texture_labels, feature_labels = image_to_vocab(image_path)
  dessert_list = get_dessert_list(animal_label, texture_labels, feature_labels, get_image_color(image_path), image_path)
  return pipe(dessert_list[0], image = make_canny_image(image_crop(image_path)), num_inference_steps=60, controlnet_conditioning_scale=0.8, guidance_scale=7.5).images[0]

iface = gr.Interface(fn=main_function, inputs=gr.Image(type="filepath"), outputs=gr.Image(type="numpy"))
iface.launch(debug=True, share=True)
```

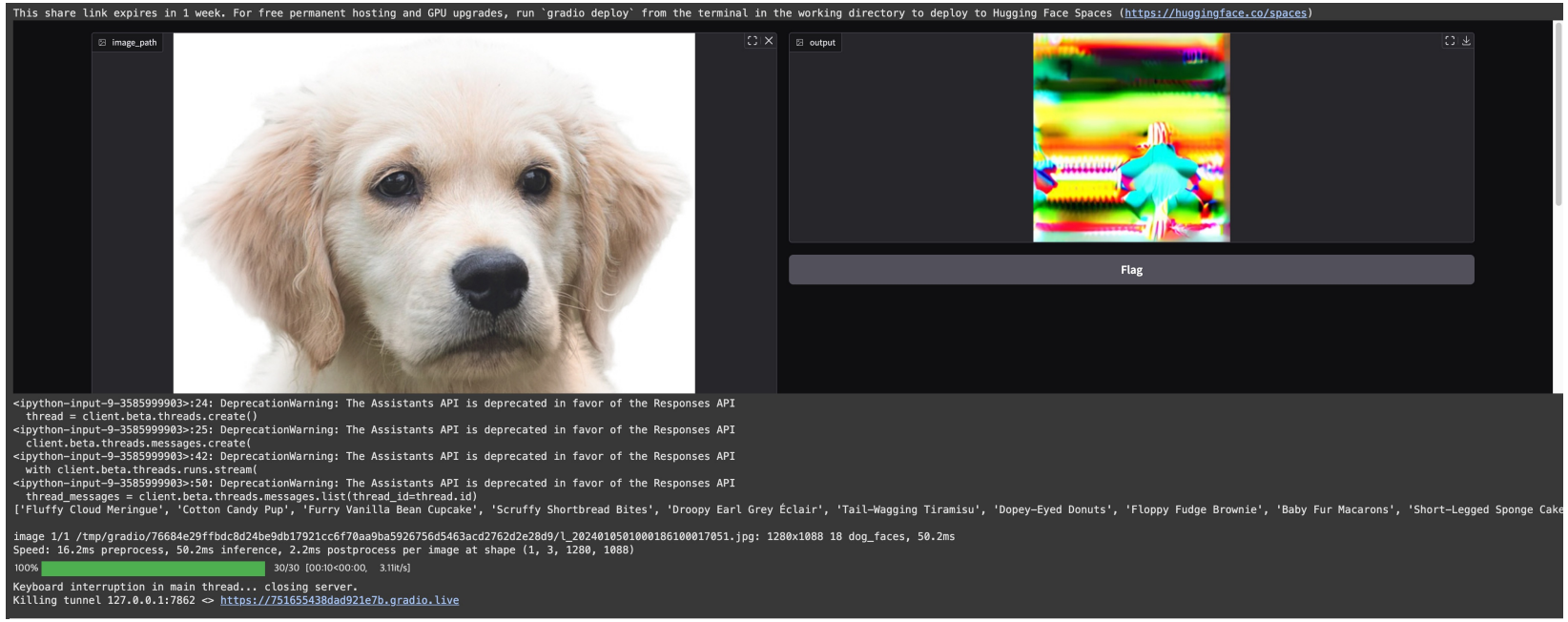
프로젝트 중 겪은 문제점

Cropped Image와 원본 Image의 크기 문제

프로젝트 중 겪은 문제점

이미지 생성 오류

프로젝트 중 겪은 문제점


This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to deploy to Hugging Face Spaces (https://huggingface.co/spaces)
image_path
output
Flag
<ipython-input-9-3585999903>:24: DeprecationWarning: The Assistants API is deprecated in favor of the Responses API
thread = client.beta.threads.create()
<ipython-input-9-3585999903>:25: DeprecationWarning: The Assistants API is deprecated in favor of the Responses API
client.beta.threads.messages.create(
<ipython-input-9-3585999903>:42: DeprecationWarning: The Assistants API is deprecated in favor of the Responses API
with client.beta.threads.runs.stream(
<ipython-input-9-3585999903>:50: DeprecationWarning: The Assistants API is deprecated in favor of the Responses API
thread_messages = client.beta.threads.messages.list(thread_id=thread.id)
['Fluffy Cloud Meringue', 'Cotton Candy Pup', 'Furry Vanilla Bean Cupcake', 'Scruffy Shortbread Bites', 'Droopy Earl Grey Éclair', 'Tail-Wagging Tiramisu', 'Dopey-Eyed Donuts', 'Floppy Fudge Brownie', 'Baby Fur Macarons', 'Short-Legged Sponge Cake
image 1/1 /tmp/gradio/76684e29ffbdc8d24be9db17921cc6f70aa9ba5926756d5463acd2762d2e28d9/l_20240105010001861000017051.jpg: 1280x1088 18 dog_faces, 50.2ms
Speed: 16.2ms preprocess, 50.2ms inference, 2.2ms postprocess per image at shape (1, 3, 1280, 1088)
100% 30/30 [00:10<00:00, 3.11it/s]
Keyboard interruption in main thread... closing server.
Killing tunnel 127.0.0.1:7862 <> https://751655438dad921e7b.gradio.live

생성 결과물

생성 결과물

생성 결과물

생성 결과물

생성 결과물

생성 결과물

생성 결과물

생성 결과물

기대 효과

나의 귀여운 반려동물을 맛있게 자랑해보세요!

기대 효과

멋있게

나의 귀여운 반려동물을 ~~맛있게~~ 자랑해보세요!

감사합니다.