Generative AI (AATG010-01)

Team 3 : 온재현, 이현서, 조효원

Project Name : Cake-able Diffusion

Project Overview

A few years ago, AI's inability to distinguish between a dog and a muffin went viral as an example of its limitations. Today, the tables have turned—AI can now make clear distinctions, while humans struggle to tell AI-generated images from real ones.

This project draws inspiration from the 2025 trend of “Ghibli-style artwork generation using GPT.” It aims to create cognitive confusion by transforming pet photos into dessert-like images that are hard to distinguish, resulting in humorous, shareable meme content for social media.

The main goals of the project are:

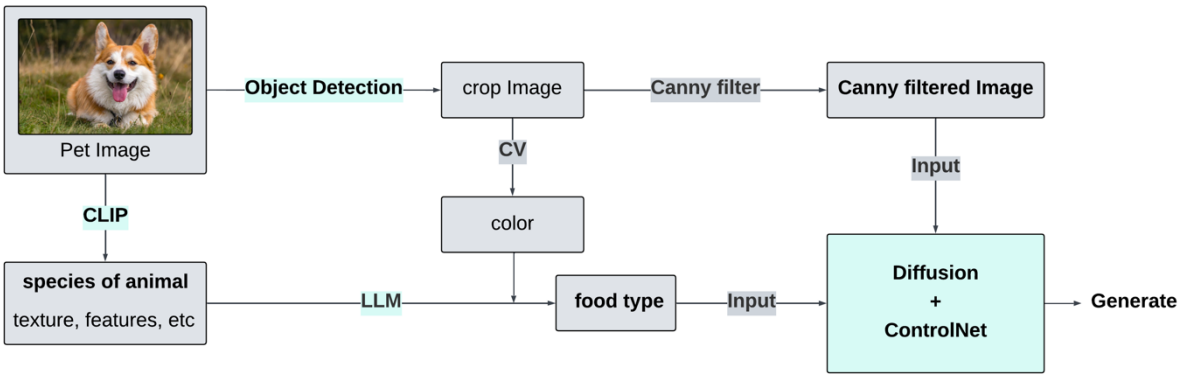
1. To generate images structurally similar to a user's pet that trigger visual illusions.
2. To produce meme-worthy content that encourages sharing and social engagement.

Expected Outcomes

- Showcase the advancement of AI, which can now intentionally generate confusing images.
- Promote viral content through user participation and organic sharing on social platforms.

Based on recent trends, the project supports the idea that interactive AI content brings more enjoyment and engagement than static results, offering users a playful and intuitive way to interact with generative technology.

Project Pipeline



- The CLIP model is used to extract semantic features from the user-provided pet image, including species, texture, and physical traits. From predefined vocabularies (e.g., "dog",

"cat", "rabbit", "hamster" for species), the most semantically similar keywords are selected. These are then used as part of the input prompt to GPT in order to determine a suitable dessert/food type. The system also extracts the image's dominant color using computer vision techniques, which is incorporated into the GPT prompt for improved context.

- A fine-tuned Ultralytics YOLO model is used to detect the main object (the pet) in the image and crop it accordingly. The cropped image is then processed with Canny edge detection to generate a line-art representation that captures the image's structural contours. The object detection model was fine-tuned on approximately 180 training samples, and the version with the lowest validation error was selected for deployment.
- The final image is generated using a diffusion model that takes two inputs: the processed Canny edge map and the food-type prompt obtained from GPT. This results in a structurally similar dessert image that visually resembles the pet input.

Process Revisions and Enhancements

- To enhance generation quality, the system limits the acceptable pet species in input images to commonly owned domestic animals: dogs, cats, rabbits, and hamsters.
- Initially, the cropping process focused on detecting and isolating the facial area (eyes, nose, mouth) of the pet. However, this led to a significant mismatch between the generated food images and the original input. As a result, the cropping logic was revised to detect and crop the full pet body rather than just facial features.

Technology Stack Overview

- **CLIP**: Image embedding and semantic similarity
- **YOLO**: Object detection and cropping
- **OpenCV + KMeans**: Dominant color extraction
- **OpenAI GPT**: Dessert concept generation from features
- **Stable Diffusion XL + ControlNet**: Image generation based on structure

Phase 1: Feature Extraction & Analysis

Animal Species, Texture, and Feature Extraction via CLIP

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

/* define animal_vocabulary, texture_vocabulary , feature_vocabulary (skipped) */

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

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]
```

The CLIP model (ViT-B/32) extracts the most semantically similar words from the provided vocabularies (animal, texture, features) based on cosine similarity to the image embedding.

Dominant Color Extraction

```
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)))


client = OpenAI(api_key=API_KEY)
```

```
assistant = client.beta.assistants.retrieve(ASSISTANT_ID)
```

Using OpenCV and KMeans clustering, the system computes the most dominant RGB and HEX color from the cropped image to serve as a visual reference.

Phase 2: Object Detection & Preprocessing

Face/Pet Detection and Cropping via YOLO

```
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
```

In this project, a YOLO model was fine-tuned specifically for pet face detection by constructing a custom dataset. Initially, cropped face images and corresponding full images were aligned using ORB feature matching to automatically generate bounding box annotations, which were then saved in YOLO-compatible format. The dataset was organized into training and validation splits, and label files were generated by converting bounding box coordinates into normalized center-based YOLO format. Using this prepared dataset, the model was trained on a pre-trained YOLOv11n checkpoint (yolo11n.pt) for 500 epochs with an image size of 1280 and a batch size of 24, leveraging the T4 GPU environment.

```
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
```

The model detects the pet's face and crops the region of interest from the original image for use in further processing like color and edge extraction.

Canny Edge Filtering

```
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")
```

The cropped image is converted to grayscale and passed through a Canny edge detector to produce a structure map for ControlNet.

Phase 3: LLM-based Dessert Suggestion

```
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)
```

Prompt Creation with Extracted Features for GPT

Combining the extracted animal label, texture and features, along with dominant color, a prompt is created to ask GPT to return a JSON list of dessert names that visually resemble the original image.

Phase 4: Image Generation using ControlNet & Main Pipeline

Stable Diffusion XL + ControlNet Pipeline Setup

```
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
```

The selected dessert prompt and the Canny edge image are used as inputs for the ControlNet-conditioned Stable Diffusion XL pipeline to generate a structurally similar dessert image.

```
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)
  print(dessert_list[0])
  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)
```

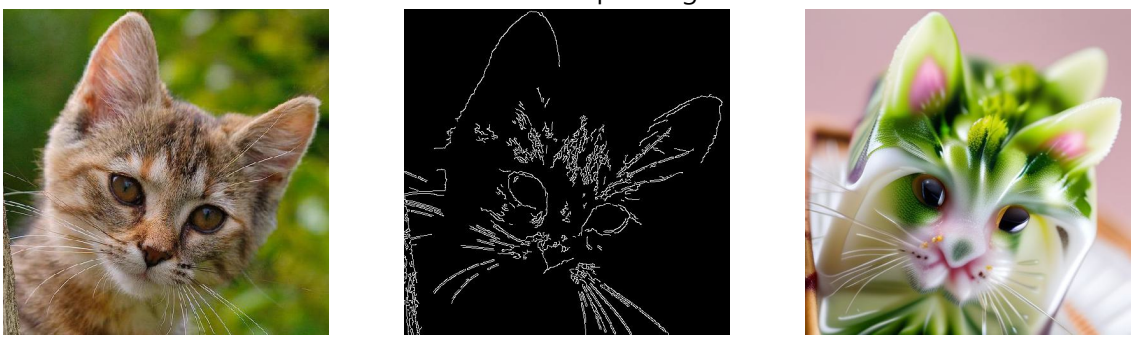
Using all previously implemented functions, construct the main pipeline with the Gradio library. By adjusting model parameters such as guidance_scale and controlnet_conditioning_scale through iterative experimentation, the generated results can be fine-tuned for better quality.

Image Generation Result

There was a noticeably high frequency of dessert recommendations such as chocolate, marshmallow, and macaron when using the LLM. This appears to be a result of relying on a predefined vocabulary, which limited the diversity of suggested desserts. While this constraint reduced the variety in prompt generation, it also contributed to more stable and reliable image outputs.

During image generation, feeding the model with Canny edge images produced using a more sensitive detection threshold—highlighting fine-grained structural features—consistently led to improved output quality over those generated with conservative settings.

butterscotch pudding

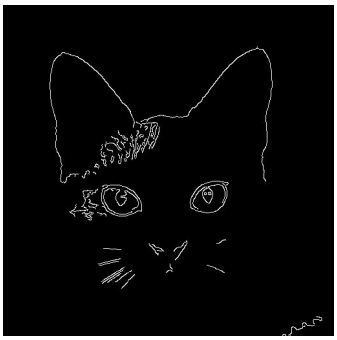


matcha mochi

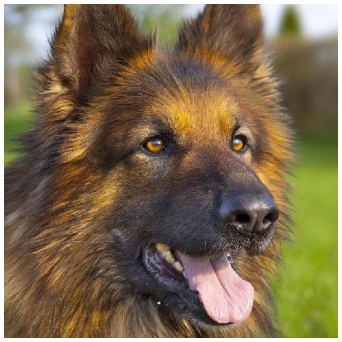
Chocolate Chip Cookies



Chocolate Lava Cake


Velvet Choux



Chocolate Truffle



Salted Caramel Blondies