

Human character with animal features : full report

1. Description of the Product and Scope

The project is about creating images of human females with cat ears using a text-to-image model. Users can describe how they want the image to look and adjust the size of the cat ears. The images show the head and upper body with clothes, and the cat ears look like a natural part of the person. This is done by fine-tuning a diffusion model.

2. Data Sources, Preprocessing, and Progress

Data Sources:

For this project, I created my own dataset because I couldn't find a ready-made one available. I used generative AI tools like Leonardo.ai and DALL-E to produce high-quality images of females with cat ears. Each image was carefully described with captions that matched the visuals. The captions included details about the person's hair color, eye color, outfit, and the appearance of the cat ears.

Here's an example of how the data was organized:

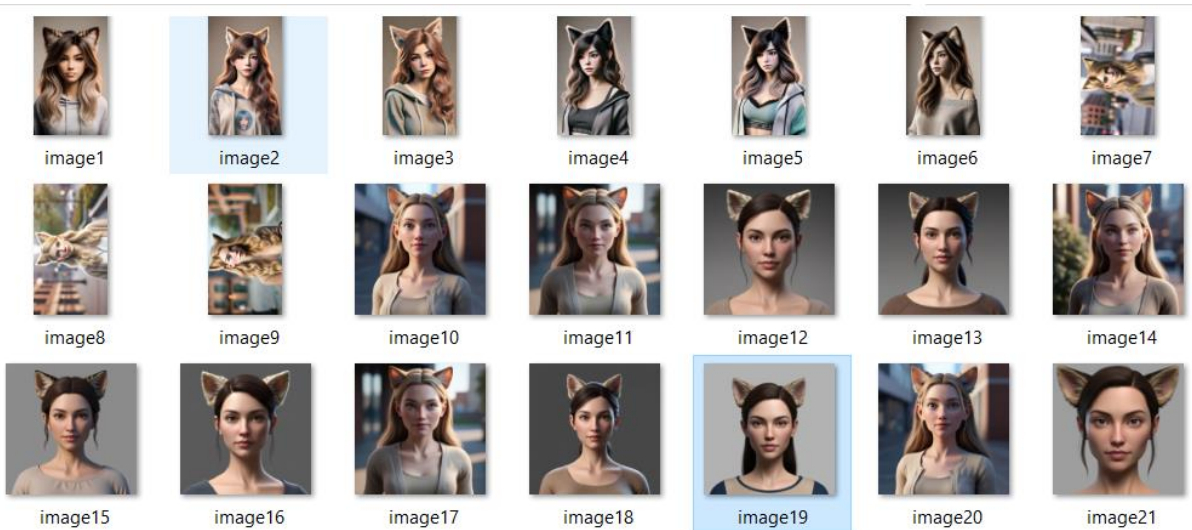
Captions: The captions included specific details like "A girl with long brown hair, wearing a hoodie, and having prominent black-and-yellow cat ears".

(a part of my_dataset/captions.txt)

```
image1.jpg  A girl with long brown hair and brown eyes, wearing a hoodie. She
has prominent feline ears covered in yellow and black fur, with a lighter
beige inner fur.
image2.jpg  An Asian girl with long brown hair and brown eyes, wearing a
hoodie. She has big fluffy blond cat ears with soft white inner fur.
image3.jpg  A girl with long blond hair and blue eyes, wearing a hoodie and a
white shirt. She has big fluffy blond cat ears with soft white inner fur.
image4.jpg  An Asian girl with long black hair and black eyes, wearing
fitness clothing with a hoodie on top. She has big fluffy black cat ears with
brown inner fur.
image5.jpg  A girl with short black hair featuring pinkish ends and black
eyes, wearing fitness clothing with a hoodie on top. She has big fluffy black
cat ears with brown inner fur.
```

Images: Each image showed a female with realistic cat ears integrated into her features, covering the head and upper body.

(a part of my_dataset/images)



This dataset allowed me to train the model to generate similar images based on text prompt

Preprocessing:

- **Image Preprocessing:** Images were resized to 512x512 to match the requirements of the Stable Diffusion XL model. Non-image files were excluded, and all images were normalized to a $[0, 1]$ pixel range.
- **Text Processing:** Captions were tokenized and aligned with the image data. Data augmentation techniques ensured the model could generalize over diverse prompts.
- **Progress:** The dataset was effectively preprocessed, and a data pipeline was established using PyTorch and Hugging Face libraries. The training pipeline was validated on small batches, confirming functionality.

3. Deep Learning Model

- **Model Selection:** The "Stable Diffusion XL Base 1.0" model was used as the base, comprising a UNet2DConditionModel, AutoencoderKL, CLIPTextModel, and DDPM noise scheduler.
- **Fine-Tuning:** The UNet model was fine-tuned using a custom dataset. Training leveraged the accelerate library for mixed precision and distributed training, optimizing performance on GPU.
- **Training Details:**
 - o Loss Function: Mean Squared Error (MSE) between predicted noise and added noise in images.
 - o Optimizer: Adam optimizer.
 - o Training Steps: 3 epochs, gradient accumulation every 2 steps.
 - o Output: Fine-tuned model saved locally, ready for deployment.
- **Inference Pipeline:** A web interface was implemented to let users input text prompts and adjust feature intensity. The model generated images directly in response to the inputs.

- How i proceeded:

This part of the code sets up the environment for running Stable Diffusion. It starts by cloning the project's repository from GitHub and moving into its directory. Then, it updates the Python package manager pip and installs all required libraries listed in the requirements.txt file. Finally, it upgrades keras to avoid compatibility issues and removes torchtext to prevent conflicts with the PyTorch version used in the project.

```
1 !git clone https://github.com/justinpinkney/stable-diffusion.git
2 %cd stable-diffusion
3 !pip install --upgrade pip
4 !pip install -r requirements.txt
5 |
6 !pip install --upgrade keras
7 !pip uninstall -y torchtext
```

This part sets up the paths to the dataset and reads the image file paths and their corresponding captions from a text file. It first defines the location of the images and captions, then opens the captions file to process each line. For each line, it splits the image file name and its caption using a delimiter (:). If the image exists, it adds the image path and caption to a data list. If an image file is missing or the line is malformed, it skips that entry and prints a message

```
1 import os
2 from PIL import Image
3 import numpy as np
4 from datasets import Dataset
5
6
7 base_path = "/mnt/batch/tasks/shared/LS_root/mounts/clusters/execinstance/code/Users/1155143473/humananimalears"
8
9 # Paths to the dataset and captions file
10 image_dir = os.path.join(base_path, "my_dataset/images")
11 captions_file = os.path.join(base_path, "my_dataset/captions.txt")
12
13 # Step 1: Read Captions File and Prepare Dataset
14 data = []
15 with open(captions_file, "r") as f:
16     for line in f:
17         try:
18             # Split using ': ' as the delimiter, accounting for extra spaces
19             image_name, caption = map(str.strip, line.strip().split(':', 1)) # Use maxsplit=1 to avoid splitting the caption
20             image_path = os.path.join(image_dir, image_name)
21             if os.path.exists(image_path): # Verify that the image file exists
22                 data.append({"image": image_path, "text": caption})
23             else:
24                 print(f"Image file not found: {image_path}")
25         except ValueError:
26             print(f"Skipping malformed line: {line.strip()}")
27
28 print(f"Loaded {len(data)} samples from local dataset.")
```

This part of the code first checks if valid data exists and then converts it into a Hugging Face Dataset object for easier handling. The dataset stores image paths and their corresponding captions. Then, it defines a preprocessing function for images. This function opens each image, converts it to RGB format, resizes it to 512x512 pixels, and normalizes its pixel values to a range between 0 and 1. This ensures all images are prepared in the right format for training the model.

```

31 if data:
32     dataset = Dataset.from_dict({
33         "image": [item["image"] for item in data],
34         "text": [item["text"] for item in data],
35     })
36
37 # Step 3: Preprocess Images
38 def preprocess_images(sample):
39     image = Image.open(sample["image"]).convert("RGB")
40     image = image.resize((512, 512)) # Resize to 512x512
41     sample["image"] = np.array(image) / 255.0 # Normalize to [0,

```

This code logs into the Hugging Face Hub. It installs the `huggingface_hub` library and uses the `notebook_login` function to authenticate the user, enabling access to models and datasets.

```

!pip install huggingface_hub
from huggingface_hub import notebook_login

notebook_login()

```

downloading checkpoint file for the Stable Diffusion model from the Hugging Face Hub. It uses the `hf_hub_download` function to fetch the file (`sd-v1-4-full-ema.ckpt`) from the specified repository, authenticating with the user's token. The downloaded file path is stored in `ckpt_path`.

```

from huggingface_hub import hf_hub_download
ckpt_path = hf_hub_download(repo_id="CompVis/stable-diffusion-v1-4-original", filename="sd-v1-4-full-ema.ckpt", use_auth_token=True)

```

This code sets up configuration parameters for training. `BATCH_SIZE` defines the number of samples processed at a time, `N_GPUS` specifies the number of GPUs used, and `ACCUMULATE_BATCHES` determines how many batches to accumulate before updating the model weights. The `gpu_list` variable creates a string listing the GPUs available for use.

```

BATCH_SIZE = 3
N_GPUS = 2
ACCUMULATE_BATCHES = 1

gpu_list = ",".join((str(x) for x in range(N_GPUS))) + ","

```

This command runs the main training script for fine-tuning Stable Diffusion. It specifies the configuration file, GPU settings, learning rate scaling, checkpoint to fine-tune from, and batch size. It also sets parameters for validation checks and gradient accumulation.

```

!(python /mnt/batch/tasks/shared/LS_root/mounts/clusters/execinstance/code/Users/1155143473/humananimalears/stable-diffusion/main.py \
-t \
--base_configs /mnt/batch/tasks/shared/LS_root/mounts/clusters/execinstance/code/Users/1155143473/humananimalears/stable-diffusion/configs/s
--gpus "0" \
--scale_lr false \
--num_nodes 1 \
--check_val_every_n_epoch 10 \
--finetune_from "${ckpt_path}" \
data.params.batch_size="${BATCH_SIZE}" \
lightning.trainer.accumulate_grad_batches="${ACCUMULATE_BATCHES}" \
data.params.validation.params.n_gpus="${N_GPUS}" \
)

```

This part of the code generates images using the fine-tuned Stable Diffusion model. It takes a text prompt (A young blond woman with big cat ears), specifies the output directory, image size (512x512), number of samples, the configuration file, and the trained model checkpoint. The generated images are saved in the specified output folder.

```

!(python /mnt/batch/tasks/shared/LS_root/mounts/clusters/execinstance/code/Users/1155143473/humananimalears/stable-diffusion/scripts/txt2img.py
--prompt 'A young blond women with big cat ears' \
--outdir /mnt/batch/tasks/shared/LS_root/mounts/clusters/execinstance/code/Users/1155143473/humananimalears/outputs \
-H 512 -W 512 \
-n_samples 4 \
--config /mnt/batch/tasks/shared/LS_root/mounts/clusters/execinstance/code/Users/1155143473/humananimalears/stable-diffusion/configs/stabl
--ckpt /mnt/batch/tasks/shared/LS_root/mounts/clusters/execinstance/code/Users/1155143473/humananimalears/checkpoints/sd-v2-full-ema.ckpt'

```

This code creates an interactive interface using `ipywidgets` for generating and displaying images based on user-provided prompts. It includes a text input box where users can enter a prompt (e.g., "A young blonde woman with big cat ears") and a slider to adjust the intensity of specific features, such as the size of the cat ears. A "Generate Image" button allows users to

trigger the image generation process, which is based on the input prompt and slider value. The generated image is displayed in the output area below the interface.

```
import ipywidgets as widgets
from IPython.display import display
from PIL import Image
import matplotlib.pyplot as plt

# Widgets
prompt_textbox = widgets.Text(
    value='Type your prompt here...',
    placeholder='Type your prompt here...',
    description='Prompt:',
    layout=widgets.Layout(width='80%', margin="10px 0")
)

feature_intensity = widgets.IntSlider(
    value=50,
    min=0,
    max=100,
    step=1,
    description='Feature Intensity:',
    continuous_update=False,
    style={'description_width': 'initial'},
    layout=widgets.Layout(width='80%', margin="10px 0")
)

generate_button = widgets.Button(
    description='Generate Image',
    button_style='info',
    tooltip='Click to generate an image based on the prompt',
    icon='magic',
    layout=widgets.Layout(width='40%', margin="10px auto")
)
```

The interface components are organized using a layout that includes a title, input fields, and output area, ensuring a clean and user-friendly design. The `on_button_click` function links the button to the image generation process, making the application interactive and responsive. This setup allows users to easily experiment with different prompts and feature adjustments, simplifying the process of generating custom images.

```
image_output = widgets.Output(layout=widgets.Layout(width="80%", margin="10px auto"))

# Link button click to function
def on_button_click(b):
    generate_and_display(prompt_textbox.value, feature_intensity.value)

generate_button.on_click(on_button_click)

# Layout organization
form_layout = widgets.VBox([
    widgets.HTML("<h2 style='text-align: center;'>Human With Animal Features</h2>"),
    prompt_textbox,
    feature_intensity,
    generate_button,
    image_output
], layout=widgets.Layout(border='solid 1px gray', padding='20px', width='90%', margin='20px auto'))

# Display the form
display(form_layout)
```


4. Screenshots of the Product

Human With Animal Features

Prompt:

Feature Intensity: 50

Generating image for: 'An asian female with subtle beige-furred cat ears' with intensity 50
100% 50/50 [00:42<00:00, 1.13it/s]




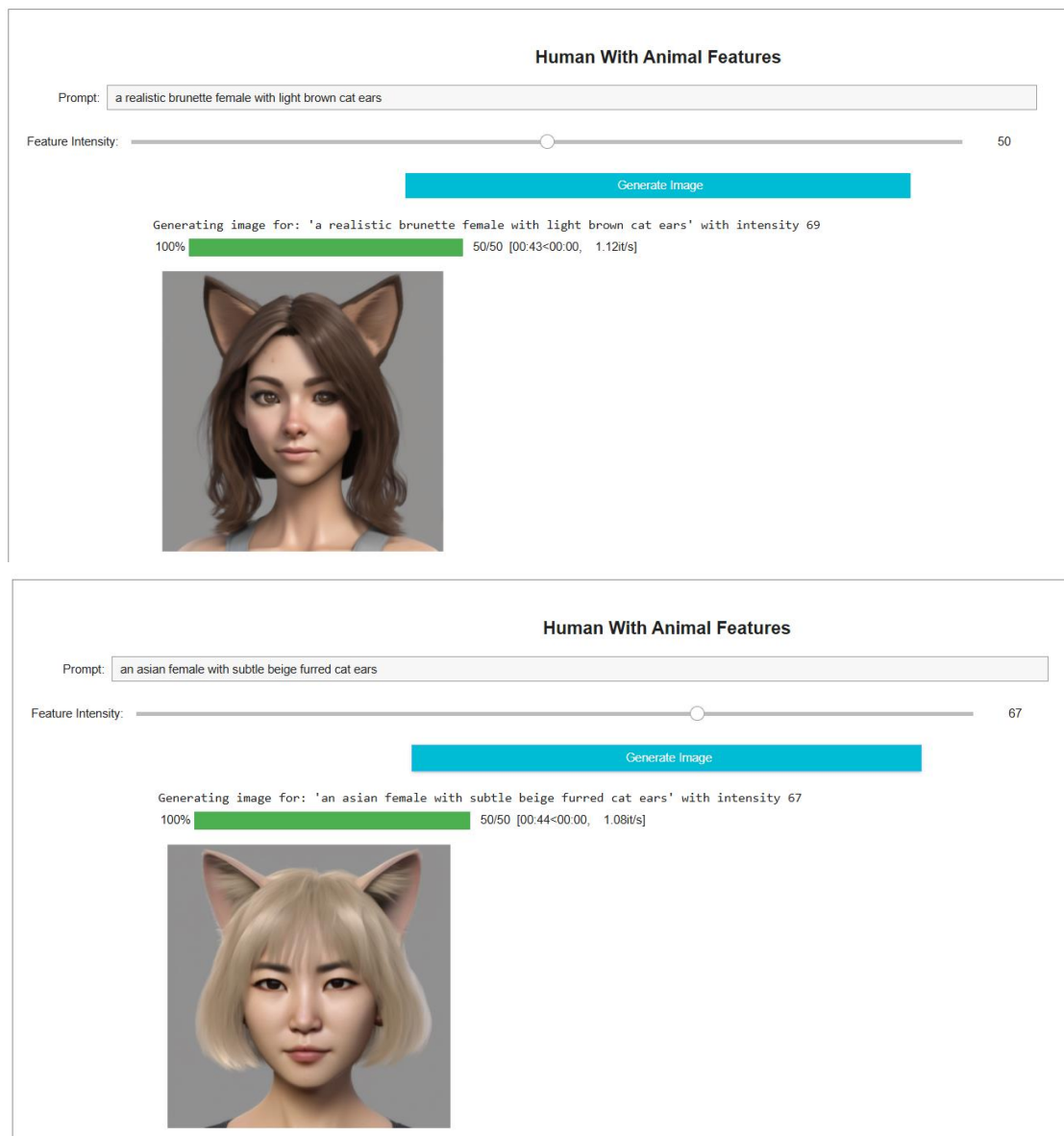
Human With Animal Features

Prompt:

Feature Intensity: 50

Token indices sequence length is longer than the specified maximum sequence length for this model (80 > 77). Running this sequence through the model will result in indexing errors
The following part of your input was truncated because CLIP can only handle sequences up to 77 tokens: ['not an accessory']
Generating image for: 'A blond girl with long blond hair, she has fluffy blond cat ears with soft beige inner fur' with intensity 50
Token indices sequence length is longer than the specified maximum sequence length for this model (80 > 77). Running this sequence through the model will result in indexing errors
The following part of your input was truncated because CLIP can only handle sequences up to 77 tokens: ['not an accessory']
100% 50/50 [00:45<00:00, 1.08it/s]





5. Summary of Challenges or Difficulties

- **Computational Resources:** Training a large-scale model like Stable Diffusion required extensive GPU resources. Fine-tuning the model on consumer-grade GPUs was particularly time-consuming, which slowed down experimentation and training iterations. Using cloud services like Microsoft Azure helped mitigate this to some extent but added its own complexities.
- **Model Performance:** Achieving a balance between realism and proper integration of features such as animal ears into the generated images was challenging. Ensuring that the animal features appeared natural and anatomically integrated, rather than looking artificial or out of place, required careful fine-tuning and experimentation.
- **Environment Setup:** Adapting the code to work seamlessly on the Microsoft Azure platform posed additional challenges. Setting up the proper environment, managing dependencies, and troubleshooting platform-specific issues required significant effort.

6. Declaration for the Use of AI Tools

This project utilized AI tools in the following ways:

- **Programming Assistance:** Tools like ChatGPT were used to refine portions of the code for better readability and efficiency. It also provided suggestions for resolving errors and improving implementation.
- **Data Collection:** Generative AI tools such as Leonardo.ai and DALL-E were used to create custom images for training. These tools allowed the generation of high-quality, realistic images with detailed captions.
- **Report Refinement:** GPT was used to refine the language of the report to ensure grammatical accuracy and improve readability, making it more professional and polished.