낙서에 기반한 그림 변환

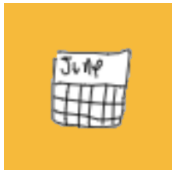
1조 : 강민정, 유지민, 김채연, 정수현, 김서윤

CONTENTS

01. 선행연구

02. 데이터 전처리

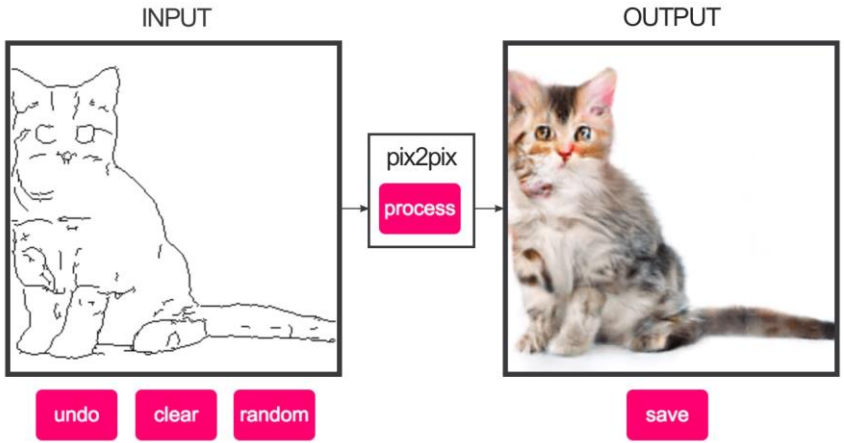
03. 모델링

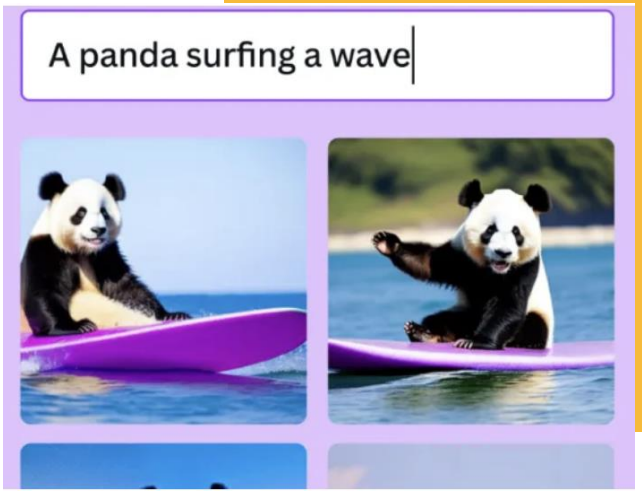


04. 결과분석

기존 생성 모델

Pix2Pix: Image-to-Image Translation with CGAN





Canva, 'Text-to-Image'

프로젝트 소개

사용자 중심의
간단한 입력

데이터셋 소개

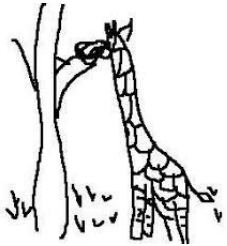
Quick, Draw!

간결한 선으로 표현되어 있으며,
노이즈가 많다

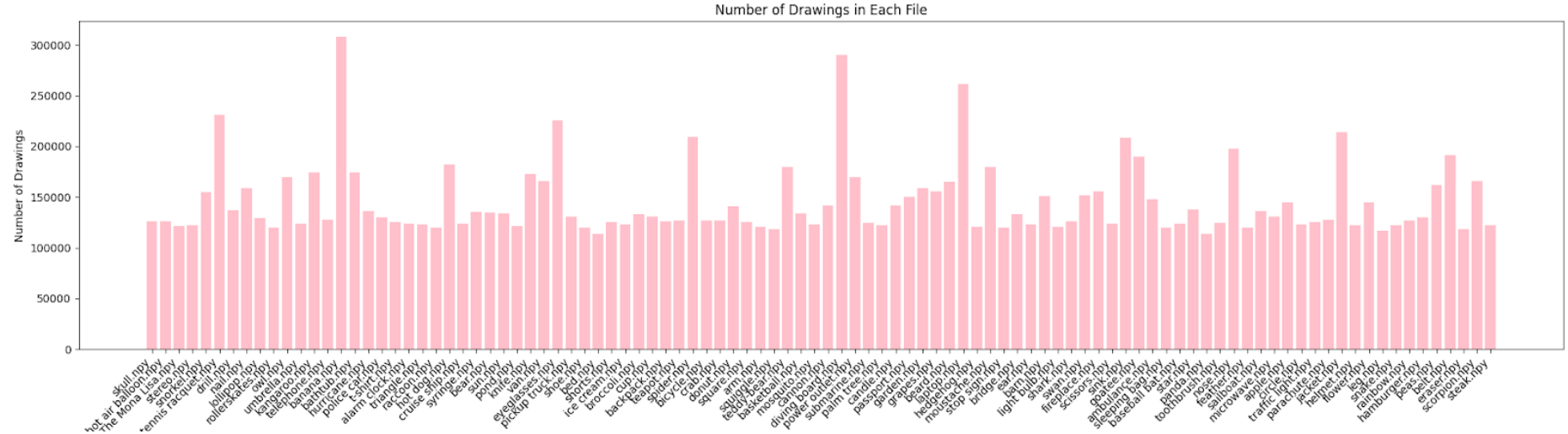
image (image)	text (string)
	"giraffe is eating leaves from the tree"
	"A zebra is eating grass"
	" people are riding on the horses"
	"two girafee"s eating the tree leaves"
	"An areoplane, airfranceis flying "

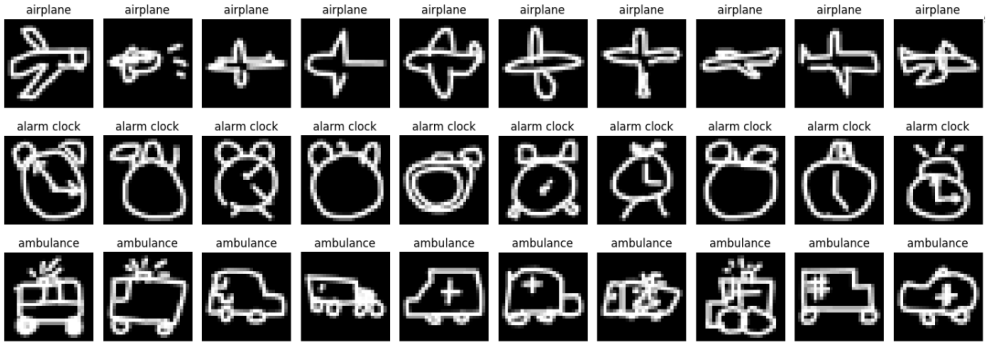
Sketch-Scene

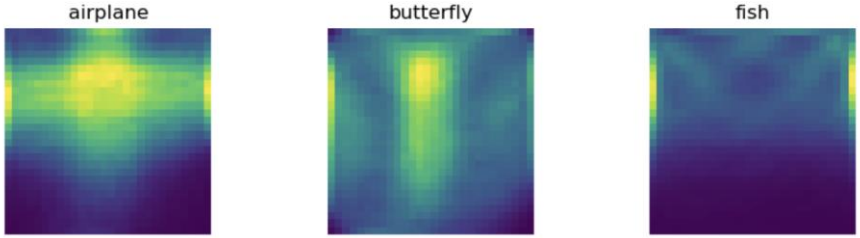
비전문가들이 그린 스케치로,
정교함이 떨어진다



Quick Draw EDA

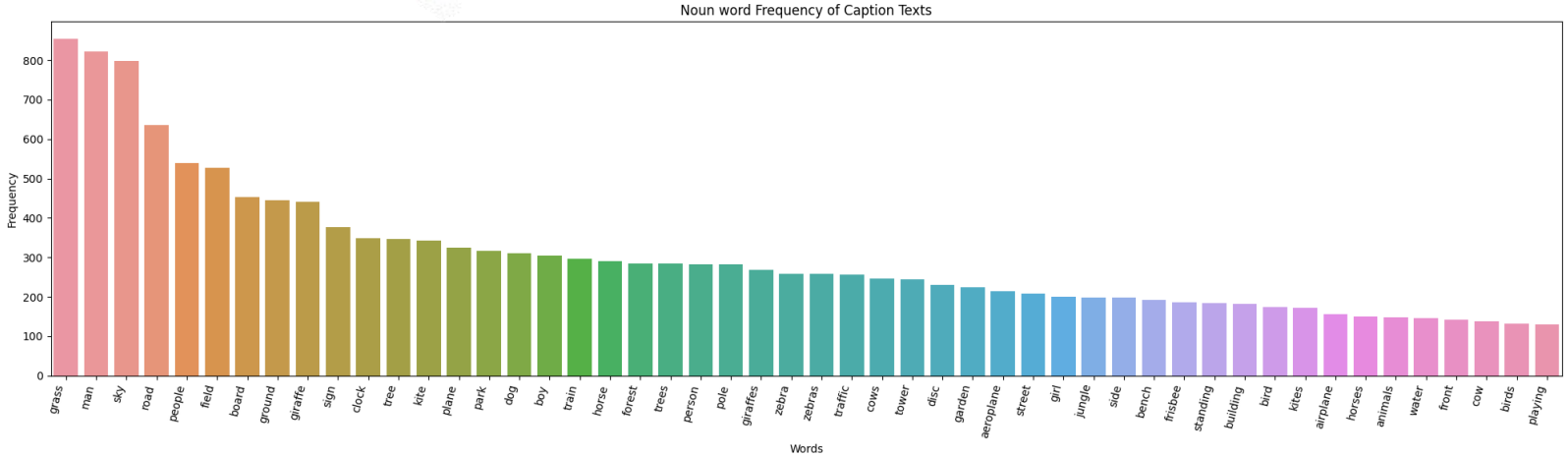






Sketch-Scene EDA


```python
noun_tokens = []

for text in texts:
    # Tokenize the text
    tokens = word_tokenize(text)

    # Perform POS tagging
    tagged_tokens = pos_tag(tokens)

    # Extract nouns from the tagged tokens
    nouns = [token for token, pos in tagged_tokens if pos.startswith('NN')]

    # Add the nouns to the noun_tokens list
    noun_tokens.extend(nouns)

#다 소문자로 변환
noun_word = [word.lower() for word in noun_tokens]

# 불용어
stopwords = ['s','a', 'the','in', 'a', 'drawing']

# 불용어 제거 후 빈도분석
noun_word=[word for word in noun_word if word not in stopwords]

c= Counter(noun_word)

c.most_common(100)
```



데이터 전처리

Resizing — Shuffle — augmentation

- ❑ **Resizing**
- ❑ **컬러 채널 복제**
- ❑ **데이터 순서 섞기**
- ❑ **Augmentation**
- ❑ **“there is a {category }” 형태의 캡션**

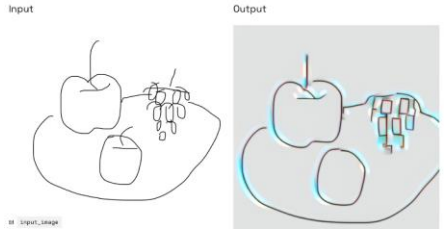
```python
# 이미지 확대 및 컬러 채널 복제 함수
def resize_and_replicate_channel(image):
    # 이미지 크기를 224x224로 리사이징
    img = Image.fromarray(image.reshape(28, 28))
    img_resized = img.resize((224, 224), Image.BILINEAR)


    # 컬러 채널 복제 (원본 이미지는 그레이스케일이므로 R, G, B 채널에 같은 값을 할당)
    img_rgb = Image.new("RGB", img_resized.size)
    img_rgb.paste(img_resized)

    return np.array(img_rgb)


# 데이터 생성기 설정
batch_size = 32  # 배치 크기
num_classes = len(class_names)

# 클래스별 이미지 데이터를 하나의 디렉토리로 합치기
all_resized_data = np.array([resize_and_replicate_channel(image) for class_name in class_
all_labels = np.repeat(np.arange(num_classes), 100)

# 이미지 데이터와 레이블의 순서를 섞기
random_indices = np.random.permutation(len(all_resized_data))
shuffled_data = all_resized_data[random_indices]
shuffled_labels = all_labels[random_indices]

# 데이터 생성기 생성
data_generator = ImageDataGenerator(
    preprocessing_function=preprocess_input,
    validation_split=0.2,
    rotation_range=20,  # 무작위 회전
    width_shift_range=0.2,  # 가로로 무작위 이동
    height_shift_range=0.2,  # 세로로 무작위 이동
    shear_range=0.2,  # 왜곡
    zoom_range=0.2,  # 확대/축소
    horizontal_flip=True,  # 수평 반전
    fill_mode='nearest'  # 빈 공간 채우기
)
```

03

모델링

모델링 – Task

Image-to-Image



낙서에 대한
학습 부족?
입력 이미지의
색깔, 구도 중요

사용자의
Prompt 필요

“Green apple”

모델링 – Task

Point:
그림의 인식!

Fine-Tuning

- 낙서 데이터 학습 (sketch scene, quick draw)
- 적은 양의 데이터로도 적합한 결과 생성 가능
- 기존 모델의 성능 활용 가능

Caption 생성

- Image Captioning을 통해 낙서를 자연어로 변환
- 자연어를 프롬프트로 이용해 다시 사진 생성 (사용자 입력 x)
- Image-to-Text + Text-to-Image

IMAGE — TEXT — IMAGE

모델링 – Image Captioning

- 기본적인 모델의 파라미터 수 적음
- 낮은 성능

Mobilenetv3small+ VGG 16

```
# 모델 로드 및 컴파일
mobilenet = tf.keras.applications.MobileNetV3Small(
    input_shape=(224, 224, 3),
    include_top=False,
    weights='imagenet',
    pooling='avg'
)
mobilenet.trainable = False

x = mobilenet.output
x = Dense(128, activation='relu')(x)
output_layer = Dense(num_classes, activation='softmax')(x)

model = Model(inputs=mobilenet.input, outputs=output_layer)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# 모델 학습
epochs = 20  # 학습 에폭 수
model.fit(train_generator, epochs=epochs, validation_data=validation_generator)
```

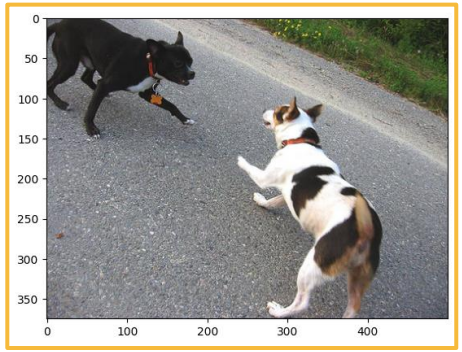
- 특성 추출까지는 good
- 특성 기반 문장 생성 bad

Visual Attention

```
class Captioner(tf.keras.Model):
  @classmethod
  def add_method(cls, fun):
    setattr(cls, fun.__name__, fun)
    return fun

  def __init__(self, tokenizer, feature_extractor, output_layer, num_layers=1,
               units=256, max_length=50, num_heads=1, dropout_rate=0.1):
    super().__init__()
    self.feature_extractor = feature_extractor
    self.tokenizer = tokenizer
    self.word_to_index = tf.keras.layers.StringLookup(
        mask_token="",
        vocabulary=tokenizer.get_vocabulary())
    self.index_to_word = tf.keras.layers.StringLookup(
        mask_token="",
        vocabulary=tokenizer.get_vocabulary(),
        invert=True)

    self.seq_embedding = SeqEmbedding(
        vocab_size=tokenizer.vocabulary_size(),
        depth=units,
        max_length=max_length)

    self.decoder_layers = [
        DecoderLayer(units, num_heads=num_heads, dropout_rate=dropout_rate)
        for n in range(num_layers)]

    self.output_layer = output_layer
```

모델링 – Image Captioning

Mobilenetv3small+ VGG 16 – 결과

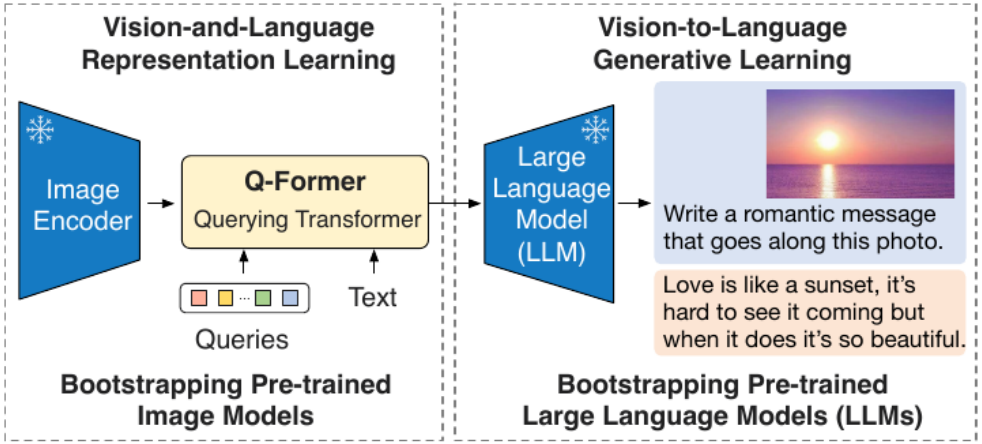



```
startseq two dogs play in the grass endseq
```

```
'startseq man is standing on top of mountain gazing at the sunset endseq'
```

=> 그림에서 인식률 낮은 모습 !

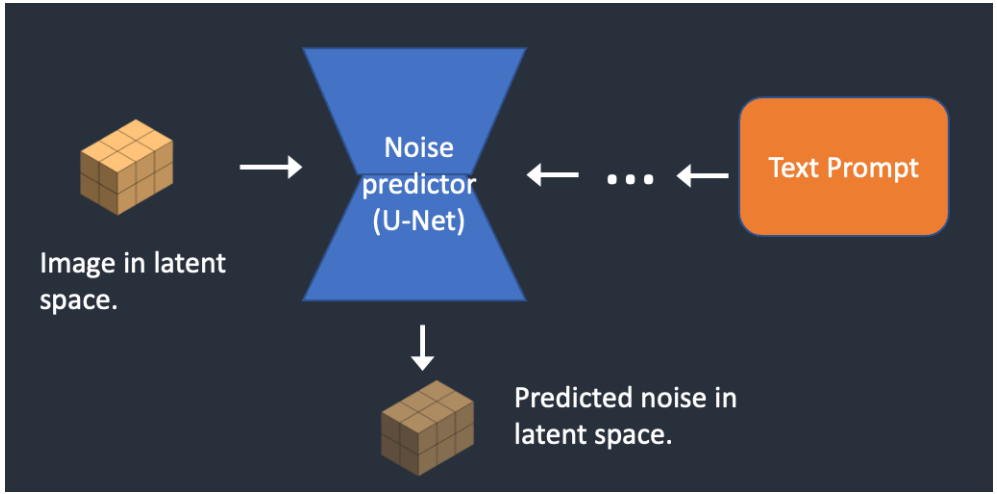
BLIP-2




```
from transformers import AutoProcessor, Blip2ForConditionalGeneration

processor = AutoProcessor.from_pretrained("Salesforce/blip2-opt-2.7b")
model = Blip2ForConditionalGeneration.from_pretrained("ybelkada/blip2-opt-2.7b-fp16-sharded", device_map="auto", load_in_8bit=True)
```

Stable Diffusion

Latent — U-net — VAE Decoder



PEFT - LoRA

BLIP-2 Finetuning

```
[ ]  from peft import LoraConfig, get_peft_model

     # Let's define the LoraConfig
     config = LoraConfig(
         r=16,
         lora_alpha=32,
         lora_dropout=0.05,
         bias="none",
         target_modules=["q_proj", "k_proj"]
     )

     model = get_peft_model(model, config)
     model.print_trainable_parameters()
```

PEFT (Parameter-Efficient Fine Tuning)

- 적은 수의 파라미터를 학습하는 것만으로 모델 전체를 파인 튜닝 하는 것과 유사한 효과!
- 적은 데이터 체제 & 도메인 밖의 데이터 일반화 가능

LoRA (Low-Rank Adaptation of Large Language Models)

- "미리 학습된 모델 가중치를 고정하고, 각 트랜스포머 블록에 새로운 학습 가능한 레이어를 추가한다"

```
trainable params: 5,242,880 || all params: 3,749,922,816 || trainable%: 0.13981301102065136
```

Data Loader

```python
from torch.utils.data import Dataset, DataLoader, random_split

class ImageCaptioningDataset(Dataset):
    def __init__(self, dataset, processor):
        self.dataset = dataset
        self.processor = processor

    def __len__(self):
        return len(self.dataset)

    def __getitem__(self, idx):
        item = self.dataset[idx]
        encoding = self.processor(images=item["image"], padding="max_length", return_tensors="pt")
        # remove batch dimension
        encoding = {k: v.squeeze() for k, v in encoding.items()}
        encoding["text"] = item["text"]
        return encoding
```

```python
# Loop through dataloader to print encodings
for batch in train_dataloader:
    print(batch)
    break
```

```
Image Size: torch.Size([1, 3, 224, 224])
Data Type: torch.float32
Image Size: torch.Size([1, 3, 224, 224])
Data Type: torch.float32
Image Size: torch.Size([1, 3, 224, 224])
Data Type: torch.float32
Image Size: torch.Size([1, 3, 224, 224])
Data Type: torch.float32
```

```
Key: pixel_values, Shape: torch.Size([50, 3, 224, 224])
Key: input_ids, Shape: torch.Size([50, 20])
Key: attention_mask, Shape: torch.Size([50, 20])
```

```python
train_dataloader = DataLoader(train_dataset, shuffle=True, batch_size=batch_size, collate_fn=collate_fn)
val_dataloader = DataLoader(val_dataset, batch_size=batch_size, collate_fn=collate_fn)
```

이미지 크기 : 224 * 224 텐서

encoding = self.Processor : 이미지 전처리, 인코딩
padding = “max_length” : 모델이 허용하는 최대 길이 패딩
squeeze() : 배치 차원 제거, [채널 수, 높이, 너비] 형태의 3차원 텐서 형성

Train & Eval

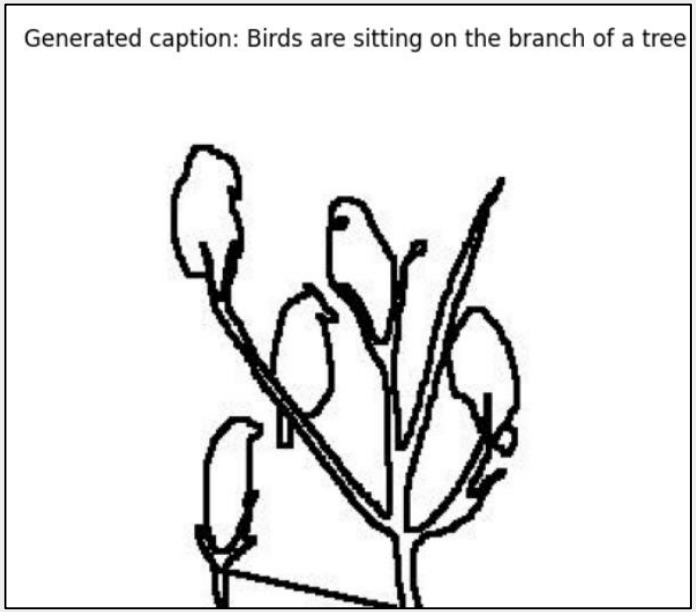
```
learning_rate = 5e-4
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

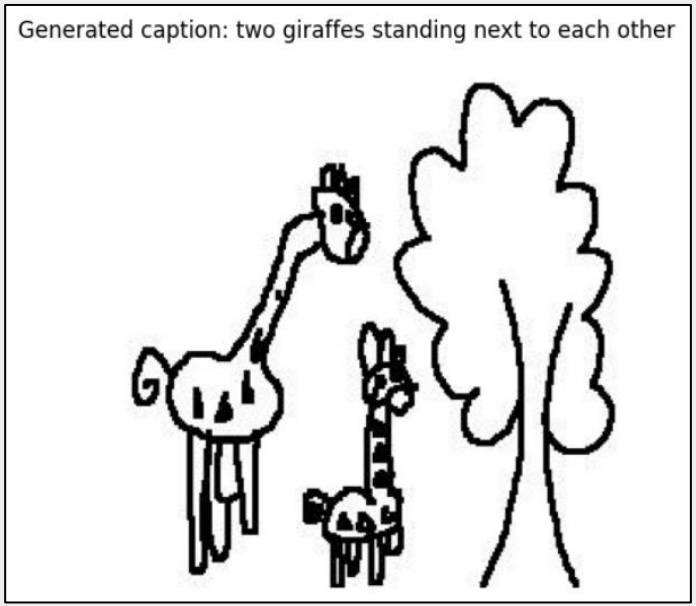
```
model.train()  # 모델을 훈련 모드로 설정

for idx, batch in enumerate(train_dataloader):
    # 훈련 데이터 배치 가져오기
    input_ids = batch.pop("input_ids").to(device)
    pixel_values = batch.pop("pixel_values").to(device, torch.float16)
    labels = input_ids

    optimizer.zero_grad()

    outputs = model(input_ids=input_ids,
                    pixel_values=pixel_values,
                    labels=input_ids)

    loss = outputs.loss
    loss.backward()
    optimizer.step()

    epoch_train_loss += loss.item()
```

- input_ids : 모델의 입력에 사용되는 텍스트 데이터
- pixel_values : 모델의 입력에 사용되는 이미지 데이터
- labels : 정답 레이블 설정
- optimizer.step() : 파라미터들을 업데이트

Blip-2 Caption

- 결과: 학습된 Blip-2 model로 생성된 캡션





Text to Image

- **Stable Diffusion**

```
import torch
from diffusers import StableDiffusionPipeline

# make sure you're logged in with `huggingface-cli login`
pipe = StableDiffusionPipeline.from_pretrained("CompVis/stable-diffusion-v1-4", revision="fp16", torch_dtype=torch.float16)
pipe = pipe.to("cuda")
```

```
Downloading (...)p16/model_index.json: 100% 543/543 [00:00<00:00, 16.3kB/s]
text_encoder/model.safetensors not found
Fetching 16 files: 100% 16/16 [00:30<00:00, 2.04s/it]
```

- **Trained blip-2를 활용해 생성된 caption 입력**

```
new_image = pipe(generated_caption).images[0]
```

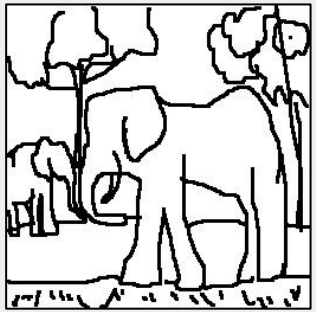
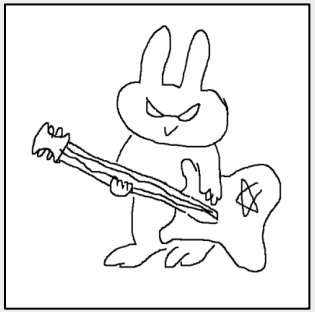
```
100% 50/50 [00:07<00:00, 7.11it/s]
```

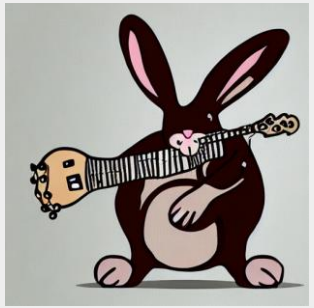
Text to Image

- **Input & Output**

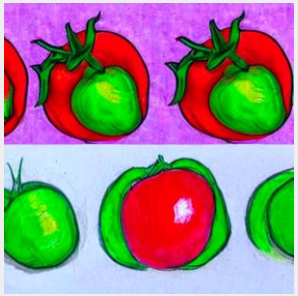
'A photo of Elephants walking in the jungle'

a rabbit playing with a guitar



- **Prompt text에 너무 의존하게 되면서 생기는 문제점?**

Img2Img

- **Stopwords 지정**

```
a cartoon drawing of a man holding balloons
```

```
stopwords= ['drawn in', 'drawing ', 'sketch ', 'draw ', 'doodle ', 'a drawing', 'a doodle', 'on a white background', 'white background', 'drew']
```

```
import re
pattern = re.compile(r'₩b(?:' + '|'.join(map(re.escape, stopwords)) + r')₩b', re.IGNORECASE); pattern
prompt = "A photo of " + pattern.sub('', generated_caption).strip(); prompt
```

```
'A photo of Elephants walking in the jungle'
```

- **Trained blip-2를 활용해 생성된 caption**

```
# prepare image for the model
inputs = processor(images=image, return_tensors="pt").to(device, torch.float16)
pixel_values = inputs.pixel_values

# Generate Caption -> using trained (finetuned) blip-2 model
generated_ids = model.generate(pixel_values=pixel_values, max_length=25)
generated_caption = processor.batch_decode(generated_ids, skip_special_tokens=True)[0]
print(generated_caption)
```

Img2Img

- **Image to Image Import**

```
import torch
import requests
from PIL import Image
from io import BytesIO
from diffusers import StableDiffusionImg2ImgPipeline

device = "cuda"
pipe = StableDiffusionImg2ImgPipeline.from_pretrained("nitrosocke/Ghibli-Diffusion",
                                                      torch_dtype=torch.float16).to(device)
```

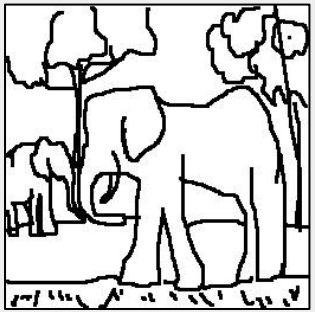
- **Output print (strength 조절)**

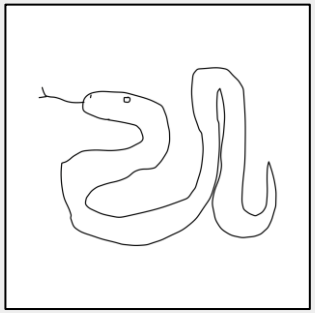
```
generator = torch.Generator(device=device).manual_seed(1024)
image = pipe(prompt=prompt, image=image_sketch, strength=0.95, guidance_scale= 10.5, generator=generator).images[0]
image
```

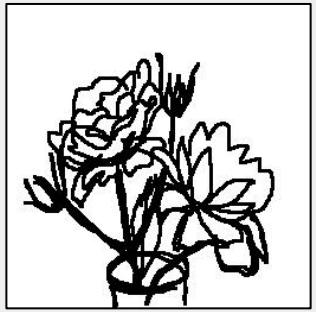
```
100% 47/47 [00:02<00:00, 17.40it/s]
```

Img2Img

- **Input & Output**





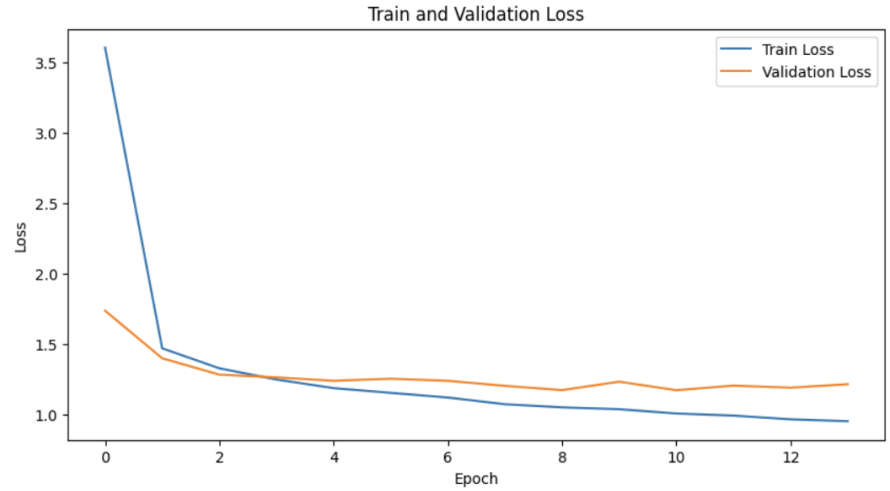


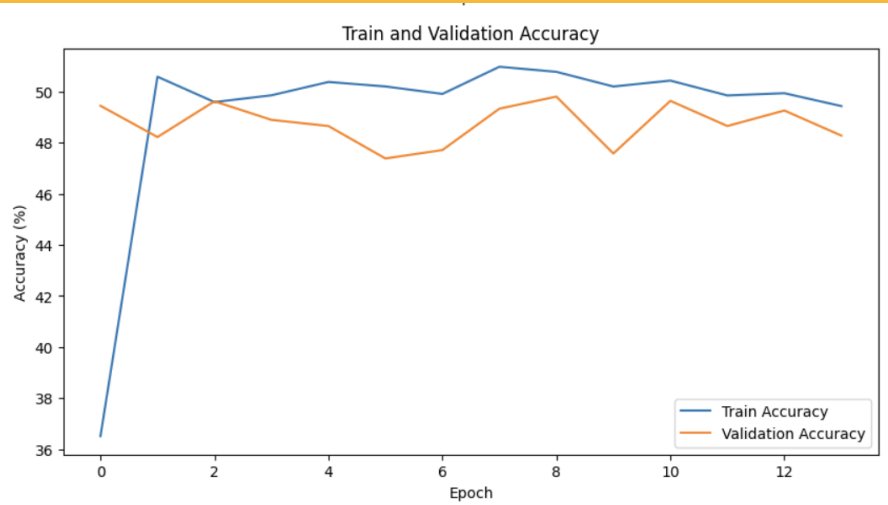
04

분석 및 개선점

Loss & Accuracy Plot

Learning_rate = 5e-4, dataset = 6000, batch_size=60, epoch=15





Loss & Accuracy Plot

- Dataset : 1000 ~ 6000
- Epoch : 10 ~ 20
- Batch Size : [16, 32, 60, 100] 등
- Learning Rate -> 5e-4에서 증가할 경우 제대로 수렴x, 감소할 경우 지나치게 느린 수렴
- Dropout (일부 뉴런 비활성화) : 0.05

개선점

Dataset

Sketch-scene dataset

- 이미지당 캡션 하나씩 존재
- class 분류 X
- 양과 다양성 부족

GPU

모델의 복잡도

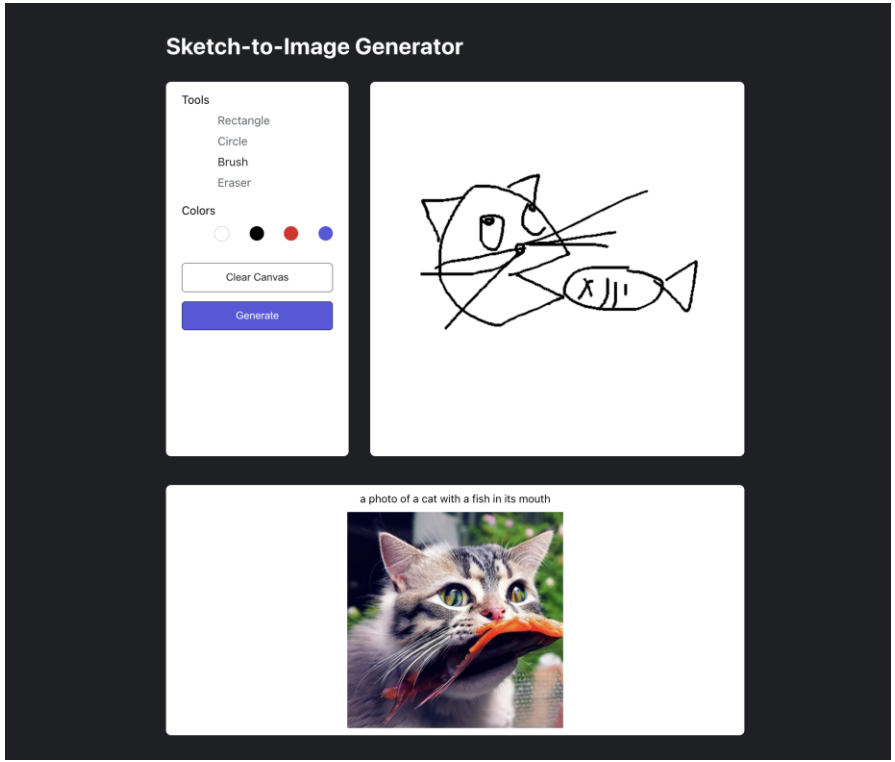
학습량의 한계

충분한 학습 부족

05

frontend

Frontend 시현 !


Sketch-to-Image Generator
Tools
Rectangle
Circle
Brush
Eraser
Colors
Clear Canvas
Generate
a photo of a cat with a fish in its mouth