CLEVELAND STATE UNIVERSITY

CIS 660 – DATA MINING

PROJECT PRESENTATION

GROUP MEMBERS:
RAHUL PADWANI - 2873059
JAYKUMAR MISTRY - 2875386

PROFESSOR: - DR. SUNNIE CHUNG

TEXT-TO-IMAGE SYNTHESIS WITH DEEP NEURAL NETWORKS

INTRODUCTION

In our project, we're turning the vision of words painting pictures into reality. Using a collection of **31,000 images** linked to descriptive sentences, we're training a computer to bridge the gap between words and visuals. This goes beyond art creation – it's about transforming language into vivid images through a sophisticated program, like a **pixel-by-pixel conversation** with a painter. The technology's potential spans graphic design creativity, accessibility revolution, and the transformation of education and entertainment into interactive **visual experiences**. Our project is more than technological advancement; it's about democratizing the power of visual storytelling, making it accessible to everyone.

GOAL OF THE PROJECT

- **Primary Goal**: Develop an advanced machine learning model capable of accurately generating images from textual descriptions.
- **Integration of Techniques**: Combine cutting-edge techniques in computer vision, natural language processing and stable diffusion model for a robust text-to-image synthesis system.
- **Overall Objective**: Create a versatile and effective tool that seamlessly translates descriptive language into visually compelling images.

SYSTEMS AND TOOLS USED

Development Environment: **Google Colab** and **Jupyter Notebook**

Programming Language: **Python**

Libraries: **TensorFlow, PyTorch, NLTK, PIL, OpenCV…**

Machine Learning Algorithms: **CNNs**, **GANs**, **Stable Diffusion**

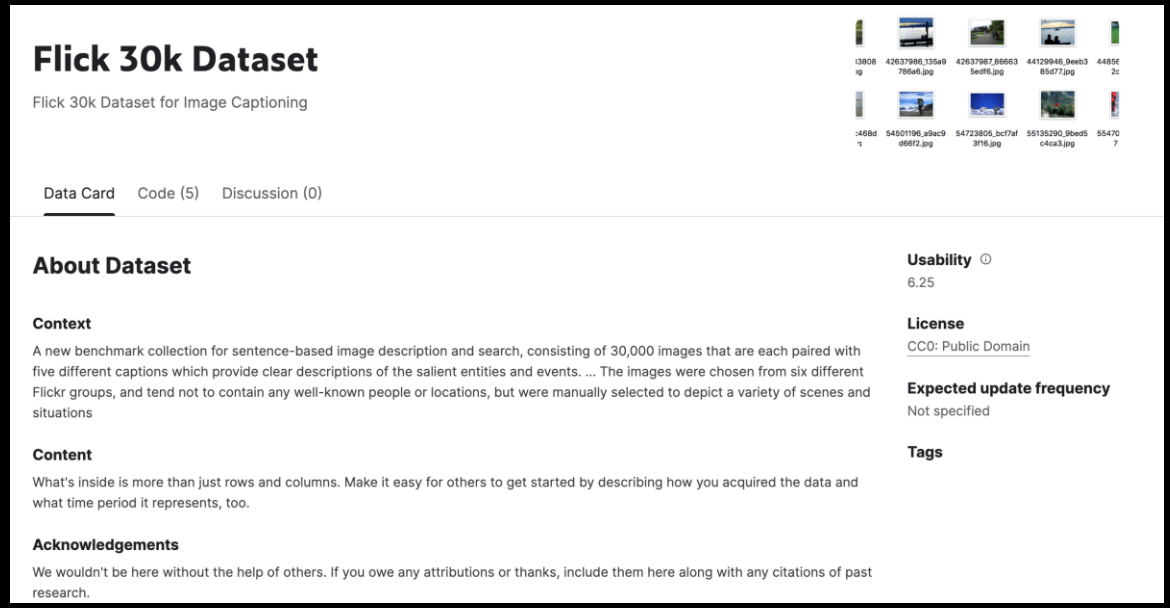
DATA COLLECTION

DATA COLLECTION

DATASET UTILIZATION: UTILIZE THE EXTENSIVE FLICKR30K DATASET TO ENABLE THE MODEL TO LEARN COMPLEX RELATIONSHIPS BETWEEN TEXT AND VISUAL ELEMENTS.

DATASET SOURCE:

KAGGLE -HTTPS://WWW.KAGGLE.COM/DATASETS/ADITYAJN105/FLICKR30K/DATA



DATASET OVERVIEW:

ROBUST REPRESENTATION WITH APPROXIMATELY **155,000 DATA POINTS.**

EACH IMAGE-DESCRIPTION PAIR IS TREATED AS AN INDIVIDUAL DATA POINT.

COMPRISES 31,000 IMAGES, WITH **EACH IMAGE PAIRED WITH FIVE DISTINCT TEXT DESCRIPTIONS.**

DATASET SIZE:

AROUND 155,000 TEXT-IMAGE PAIRS (31,000 IMAGES × 5 CAPTIONS/IMAGE).

SIZE OF THE DATASET IS APPROXIMATELY 8.86 GB.

STRUCTURE:

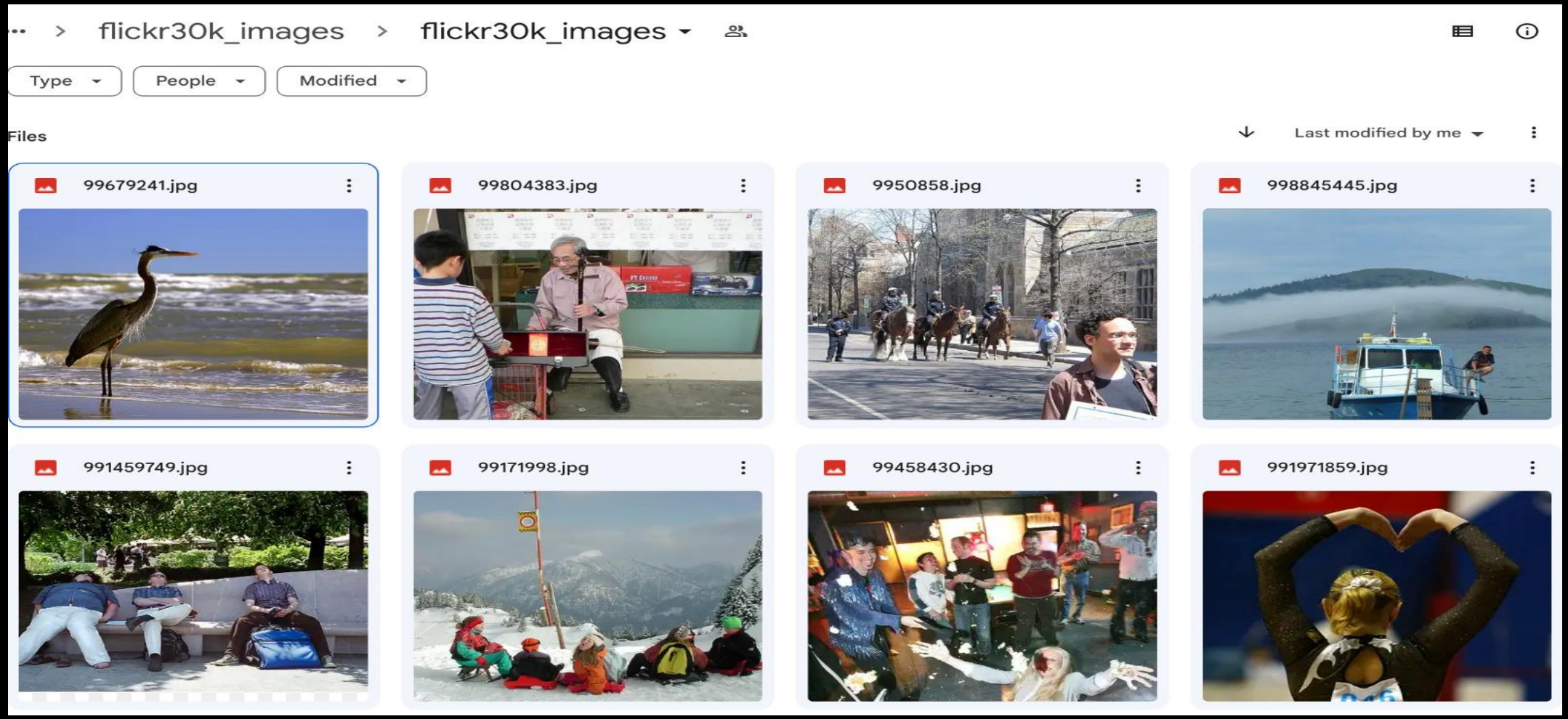
THE DATASET INCLUDES AN IMAGES FOLDER CONTAINING **FLICKER30K_IMAGES.**

THE "**CAPTION.TXT**" FILE HOLDS FIVE DIFFERENT TEXT DESCRIPTIONS PER IMAGE, PROVIDING DIVERSE PERSPECTIVES ON THE CONTENTS AND CONTEXT OF THE IMAGES.

CAPTION.TXT

image_name| comment_number| comment									
1000092795.jpg| 0| Two young guys with shaggy hair look at their hands while hanging out in the yard .										
1000092795.jpg| 1| Two young	White males are outside near many bushes .									
1000092795.jpg| 2| Two men in green shirts are standing in a yard .										
1000092795.jpg| 3| A man in a blue shirt standing in a garden .										
1000092795.jpg| 4| Two friends enjoy time spent together .										
10002456.jpg| 0| Several men in hard hats are operating a giant pulley system .										
10002456.jpg| 1| Workers look down from up above on a piece of equipment .										
10002456.jpg| 2| Two men working on a machine wearing hard hats .										
10002456.jpg| 3| Four men on top of a tall structure .										
10002456.jpg| 4| Three men on a large rig .										
1000268201.jpg| 0| A child in a pink dress is climbing up a set of stairs in an entry way .										
1000268201.jpg| 1| A little girl in a pink dress going into a wooden cabin .										
1000268201.jpg| 2| A little girl climbing the stairs to her playhouse .										
1000268201.jpg| 3| A little girl climbing into a wooden playhouse										
1000268201.jpg| 4| A girl going into a wooden building .										
1000344755.jpg| 0| Someone in a blue shirt and hat is standing on stair and leaning against a window .										
1000344755.jpg| 1| A man in a blue shirt is standing on a ladder cleaning a window .										
1000344755.jpg| 2| A man on a ladder cleans the window of a tall building .										
1000344755.jpg| 3| man in blue shirt and jeans on ladder cleaning windows										
1000344755.jpg| 4| a man on a ladder cleans a window										
1000366164.jpg| 0| Two men	one in a gray shirt	one in a black shirt	standing near a stove .							
1000366164.jpg| 1| Two guy cooking and joking around with the camera .										
1000366164.jpg| 2| Two men in a kitchen cooking food on a stove .										
1000366164.jpg| 3| Two men are at the stove preparing food .										
1000366164.jpg| 4| Two men are cooking a meal .										

FLICKER30K_IMAGES FOLDER



DATA PREPROCESSING

DATA PREPROCESSING

Part – 1 Image Preprocessing
Part – 2 Text Preprocessing
Part – 3 Combining Text and image in one result

PART – 1 IMAGE PREPROCESSING

```python
import os
from PIL import Image
from tqdm.notebook import tqdm
import numpy as np

# Directory where your original images are stored
image_dir = 'flickr30k_images/flickr30k_images'

# Directory where you want to save the preprocessed images
processed_image_dir = 'flickr30k_images/processed_images'
os.makedirs(processed_image_dir, exist_ok=True)

# List to keep track of processed files
processed_files = []
not_processed_files = []

# Function to preprocess images
def transform(image):
    return image
```

Path Description

```python
# Optionally, print the filenames of images not processed
if not_processed_files:
    print("Images not processed:")
    for filename in not_processed_files:
        print(filename)


num_images_to_print = 5  # Adjust as needed
for filename in processed_files[:num_images_to_print]:
    npy_filename = os.path.splitext(filename)[0] + '.npy'
    load_path = os.path.join(processed_image_dir, npy_filename)
    image_array = np.load(load_path)
    print(f"Array for {filename}:")
    print(image_array)
```

Printing the image array.

PART – 1 IMAGE PREPROCESSING

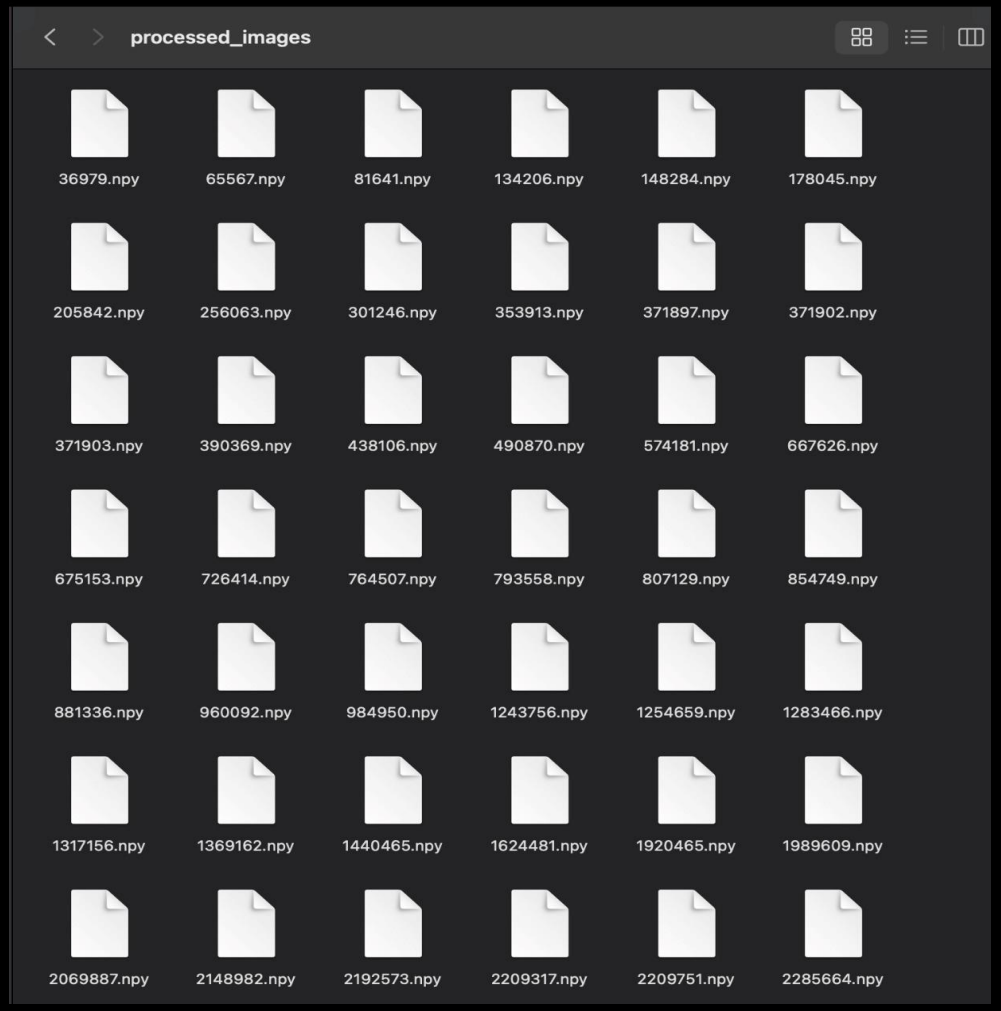
```python
# Process and save images as NumPy arrays
for filename in tqdm(os.listdir(image_dir)):
    if filename.lower().endswith('.jpg'):  # Check if the file is an image
        try:
            file_path = os.path.join(image_dir, filename)
            with Image.open(file_path) as img:
                transformed_img = transform(img)
                npy_filename = os.path.splitext(filename)[0] + '.npy'
                save_path = os.path.join(processed_image_dir, npy_filename)
                np.save(save_path, transformed_img)
                processed_files.append(filename)  # Add to the list of processed files
        except Exception as e:
            print(f"Error processing {filename}: {e}")
            not_processed_files.append(filename)  # Add to the list if not processed

# Print the results
print(f"Total number of images processed: {len(processed_files)}")
print(f"Total number of images not processed: {len(not_processed_files)}")
```

- Checking the image (jpg)
- Resizing the Image.
- Normalization.
- Noise Reduction.
- Splitting extension and adding with the NumPy array
- Printing total numbers of image processed and not processed

PART – 1 IMAGE PREPROCESSING

```
100% |████████████████████| 31785/31785 [02:06<00:00, 258.67it/s]

Error processing 4904808403.jpg: cannot identify image file 'flickr30k_images/flickr30k_images/4904808403.jpg'
Total number of images processed: 31782
Total number of images not processed: 1
Images not processed:
4904808403.jpg
Array for 2609797461.jpg:
[[[103 123 148]
  [105 125 150]
  [109 126 152]
  ...
  [198 203 206]
  [196 201 204]
  [196 201 204]]

 [[103 123 148]
  [105 125 150]
  [109 126 152]
```

Preprocessed Image



PART – 2 TEXT PREPROCESSING

```python
import pandas as pd
import re

# Path to your original captions file
captions_file_path = 'flickr30k_images/results.csv'

# Read the CSV file
captions_df = pd.read_csv(captions_file_path, delimiter='|')

# Rename the columns to remove leading/trailing spaces
captions_df.columns = captions_df.columns.str.strip()

# Function to clean text
def clean_text(text):
    text = text.lower()  # Convert text to lowercase
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation and numbers
    text = re.sub(r'\d+', '', text)  # Remove numbers
    text = re.sub(r'\s+', ' ', text).strip()  # Remove extra spaces
    return text

# Apply the cleaning function to the comment column
captions_df['comment'] = captions_df['comment'].astype(str).apply(clean_text)
```

- Separating columns with ' | '
- Removing punctuation and number
- Removing extra spaces

captions_df

	image_name	comment_number	comment
0	1000092795.jpg	0	two young guys with shaggy hair look at their ...
1	1000092795.jpg	1	two young white males are outside near many bu...
2	1000092795.jpg	2	two men in green shirts are standing in a yard
3	1000092795.jpg	3	a man in a blue shirt standing in a garden
4	1000092795.jpg	4	two friends enjoy time spent together
...	...	...	...
158910	998845445.jpg	0	a man in shorts and a hawaiian shirt leans ove...
158911	998845445.jpg	1	a young man hanging over the side of a boat wh...
158912	998845445.jpg	2	a man is leaning off of the side of a blue and...
158913	998845445.jpg	3	a man riding a small boat in a harbor with fog...
158914	998845445.jpg	4	a man on a moored blue and white boat with hil...

158915 rows × 3 columns

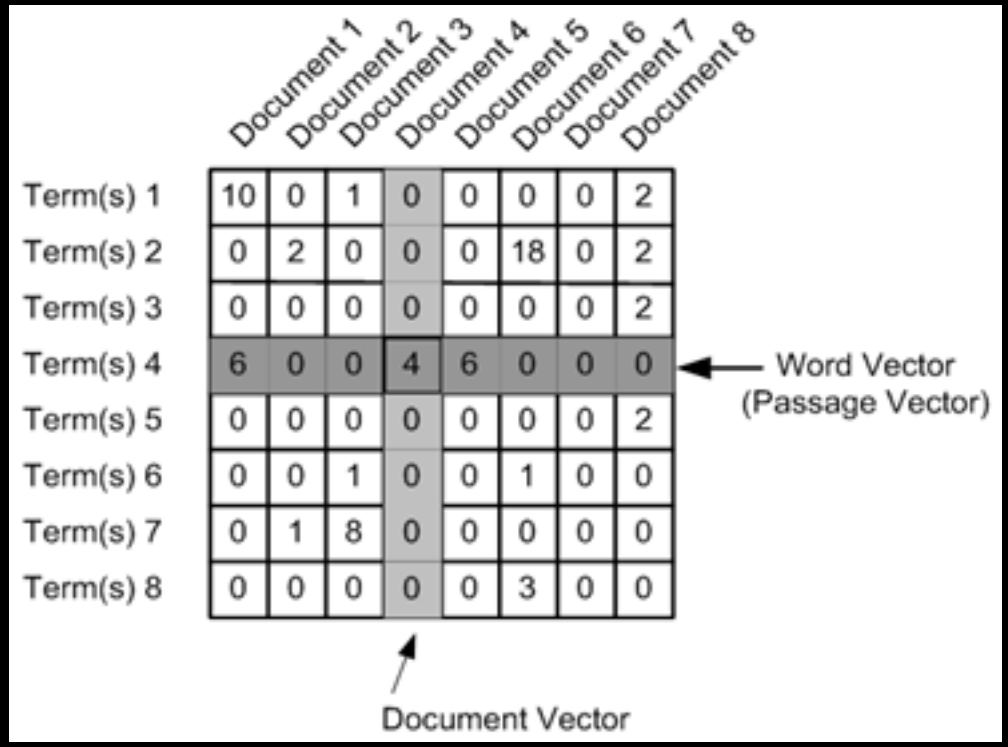
- Output of the cleaned data

PART – 2 TEXT PREPROCESSING

TOKENIZER AND VECTORIZATION ON TEXT DATA

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Tokenization and vectorization
MAX_NUM_WORDS = 10000
tokenizer = Tokenizer(num_words=MAX_NUM_WORDS)
tokenizer.fit_on_texts(captions_df['comment'])
sequences = tokenizer.texts_to_sequences(captions_df['comment'])
word_index = tokenizer.word_index
data = pad_sequences(sequences, maxlen=100)
```

Converting Comments in matrices for machine
To understand.



PART – 3 COMBINING TEXT AND IMAGE IN ARRAY

combined_dataset

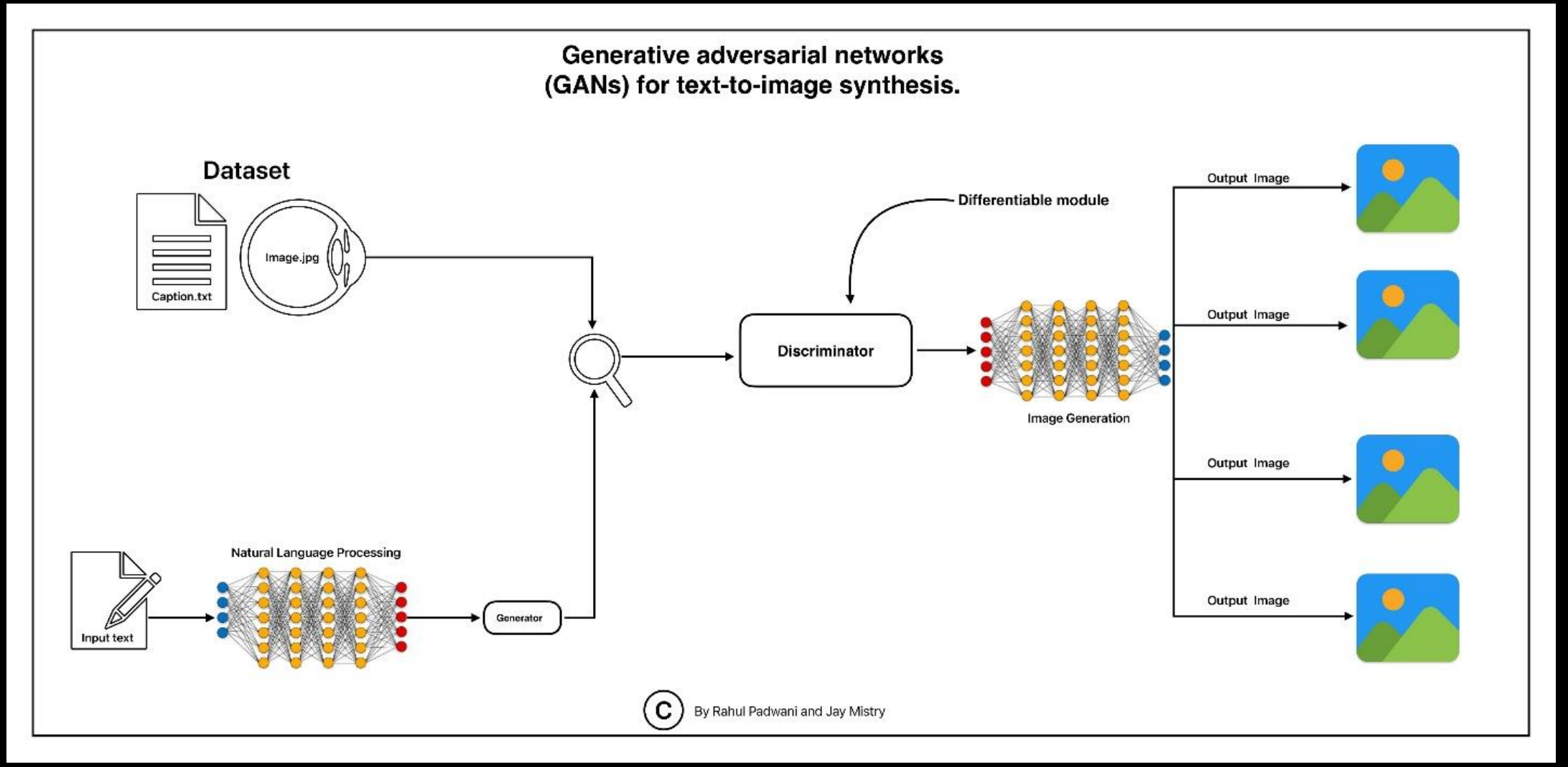
image_filename	text_data
516279719	[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 11 18 321 9 2098 107 183 15 60 155 22 320 3 485]
7610186624	[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 11 18 19 703 12 53 78 192 1416]
2735799584	[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 11 28 2 45 252 12 30 2 1 485]
5511458367	[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 6 2 1 24 20 30 2 1 705]
2044546977	[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 11 480 804 590 9961 134]
3726980861	[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 116 28 2 327 271 12 1303 1 8 3831 2622]

```python
image_filenames = [os.path.splitext(filename)[0] for filename in os.listdir(processed_image_dir) if filename.endswith('.npy')]

# Create DataFrame
export_df = pd.DataFrame({
    'image_filename': image_filenames,
    'text_data': [text for _, text in combined_dataset]
})

# Export to CSV
export_df.to_csv('flickr30k_images/combined_dataset.csv', index=False)
```

As per the image name the text data is mapped and converted into standard format.

GAN IMPLEMENTATION

GAN IMPLEMENTATION



GAN IMPLEMENTATION – GENERATOR

```python
def make_generator_model():
    model = tf.keras.Sequential()
    model.add(layers.Dense(8*8*256, use_bias=False, input_shape=(100,)))
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    model.add(layers.Reshape((8, 8, 256)))  # Reshape to a 3D tensor
    assert model.output_shape == (None, 8, 8, 256)

    # Upsample to 16x16
    model.add(layers.Conv2DTranspose(128, (5, 5), strides=(2, 2), padding='same', use_bias=False))
    assert model.output_shape == (None, 16, 16, 128)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    # Upsample to 32x32
    model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same', use_bias=False))
    assert model.output_shape == (None, 32, 32, 64)
    model.add(layers.BatchNormalization())
    model.add(layers.LeakyReLU())

    # Upsample to 64x64
    model.add(layers.Conv2DTranspose(3, (5, 5), strides=(2, 2), padding='same', use_bias=False, activation='tanh'))
    assert model.output_shape == (None, 64, 64, 3)

    return model
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 16384)             1638400

 batch_normalization_6 (Bat  (None, 16384)             65536
 chNormalization)

 leaky_re_lu_10 (LeakyReLU)  (None, 16384)             0

 reshape_2 (Reshape)         (None, 8, 8, 256)         0

 conv2d_transpose_6 (Conv2D  (None, 16, 16, 128)       819200
 Transpose)

 batch_normalization_7 (Bat  (None, 16, 16, 128)       512
 chNormalization)

 leaky_re_lu_11 (LeakyReLU)  (None, 16, 16, 128)       0

 conv2d_transpose_7 (Conv2D  (None, 32, 32, 64)        204800
 Transpose)

 batch_normalization_8 (Bat  (None, 32, 32, 64)        256
 chNormalization)

 leaky_re_lu_12 (LeakyReLU)  (None, 32, 32, 64)        0

 conv2d_transpose_8 (Conv2D  (None, 64, 64, 3)         4800
 Transpose)

=================================================================
Total params: 2733504 (10.43 MB)
Trainable params: 2700352 (10.30 MB)
Non-trainable params: 33152 (129.50 KB)
_________________________________________________________________
None
```

The generator model consists of a dense layer with 16,384 units (1,638,400 parameters), batch normalization (65,536 parameters), Leaky ReLU activation, a reshape layer for a 4D tensor, multiple Conv2D Transpose layers for upsampling, totaling 2,733,504 parameters (2,700,352 trainable).

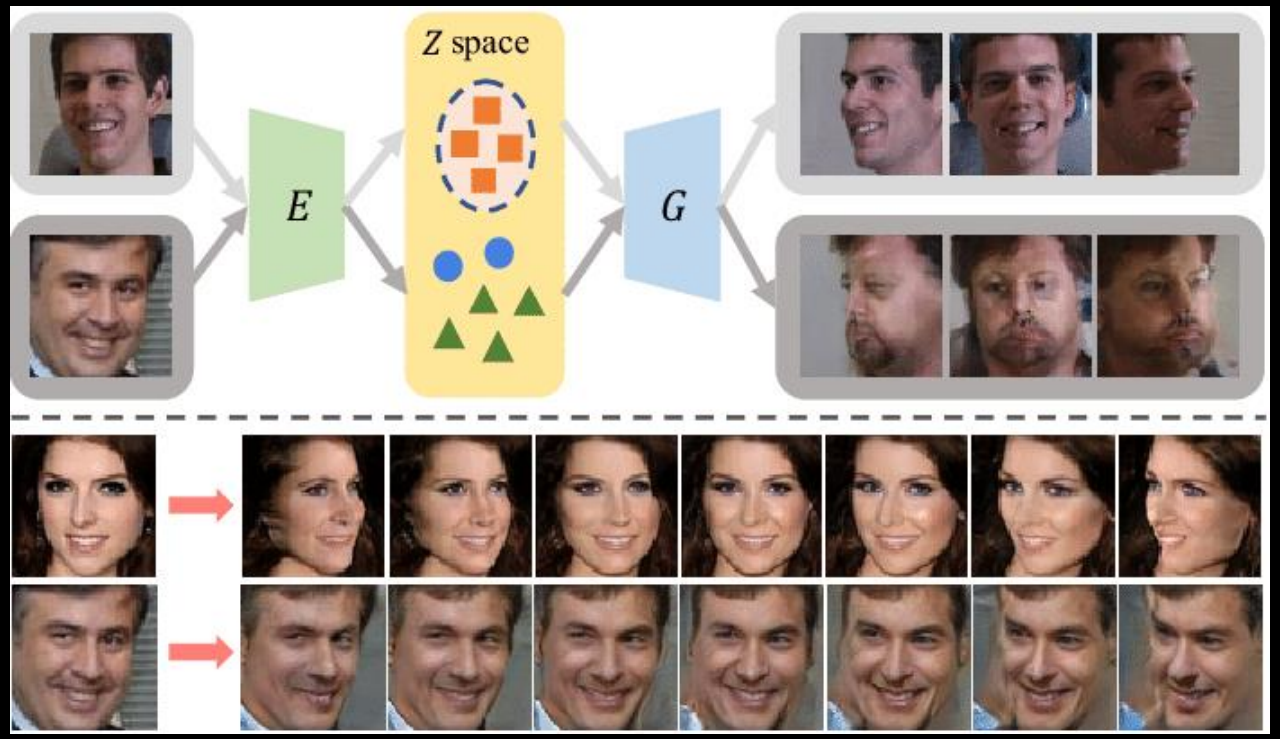
GAN IMPLEMENTATION – DISCRIMINATOR

```python
def make_discriminator_model():
    model = tf.keras.Sequential()
    model.add(layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same', input_shape=[64, 64, 3]))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Flatten())
    model.add(layers.Dense(1))

    return model

generator = make_generator_model()
discriminator = make_discriminator_model()

# Print model summaries
print(generator.summary())
print(discriminator.summary())
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 32, 32, 64)        4864

 leaky_re_lu_13 (LeakyReLU)  (None, 32, 32, 64)        0

 dropout_4 (Dropout)         (None, 32, 32, 64)        0

 conv2d_5 (Conv2D)           (None, 16, 16, 128)       204928

 leaky_re_lu_14 (LeakyReLU)  (None, 16, 16, 128)       0

 dropout_5 (Dropout)         (None, 16, 16, 128)       0

 flatten_2 (Flatten)         (None, 32768)             0

 dense_5 (Dense)             (None, 1)                 32769

=================================================================
Total params: 242561 (947.50 KB)
Trainable params: 242561 (947.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

The discriminator model features convolutional layers (Conv2D) for downsampling (209,792 parameters), Leaky ReLU activation, dropout for overfitting prevention, a flattening layer, and a dense layer for output, totaling 242,561 trainable parameters.

GAN IMPLEMENTATION – LIMITATION



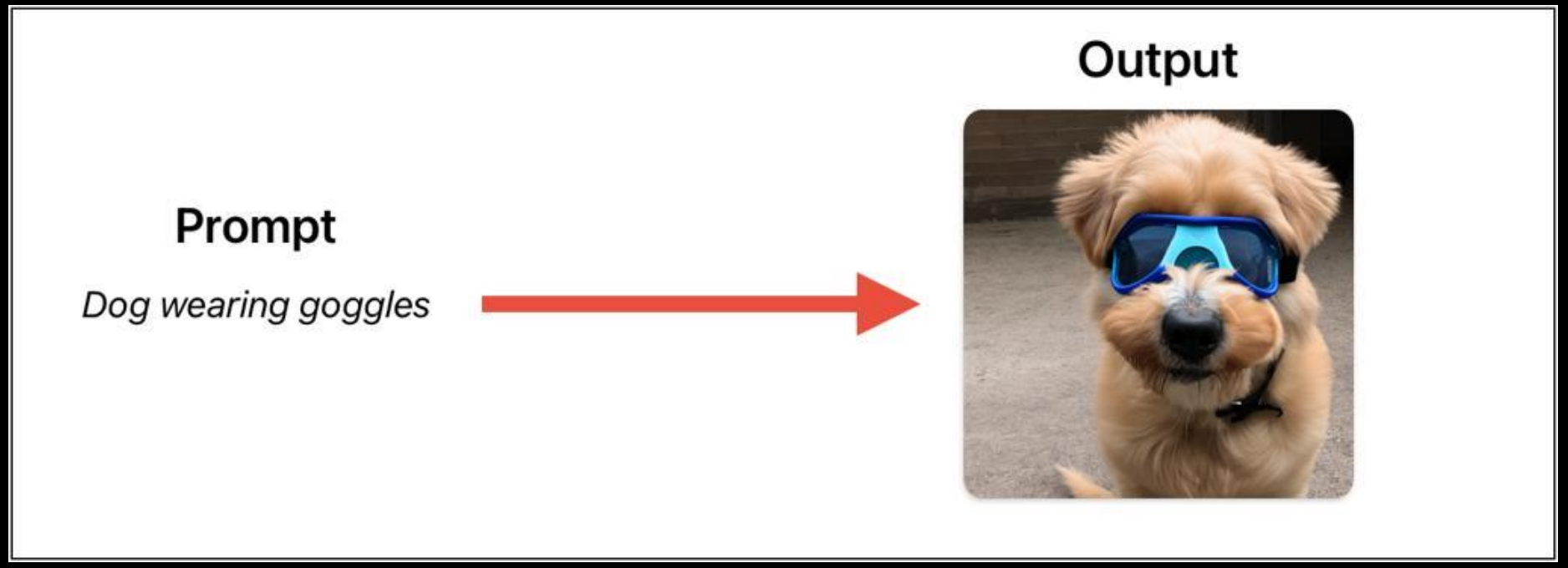
In contrast, the promising results of GANs have been revealed to be
mostly confined to data with comparably limited variability
as their adversarial learning procedure does not easily scale
to modeling complex, multi-modal distributions.

STABLE DIFFUSION

STABLE DIFFUSION

WHAT IS STABLE DIFFUSION?

STABLE DIFFUSION IS AN ADVANCED DEEP LEARNING MODEL USED FOR GENERATING HIGH-QUALITY IMAGES FROM TEXTUAL DESCRIPTIONS. IT OPERATES ON THE PRINCIPLES OF DIFFUSION MODELS, A TYPE OF GENERATIVE MODEL THAT GRADUALLY TRANSFORMS A RANDOM PATTERN OF PIXELS INTO A COHERENT IMAGE.



WHAT IS STABLE DIFFUSION?

Text-to-Image Synthesis: Stable Diffusion can create detailed and realistic images based on textual prompts, making it a powerful tool for artists, designers, and content creators.

Diffusion Process: It uses a process where it starts with a random noise image and gradually refines it to match the desired output, guided by the textual input.

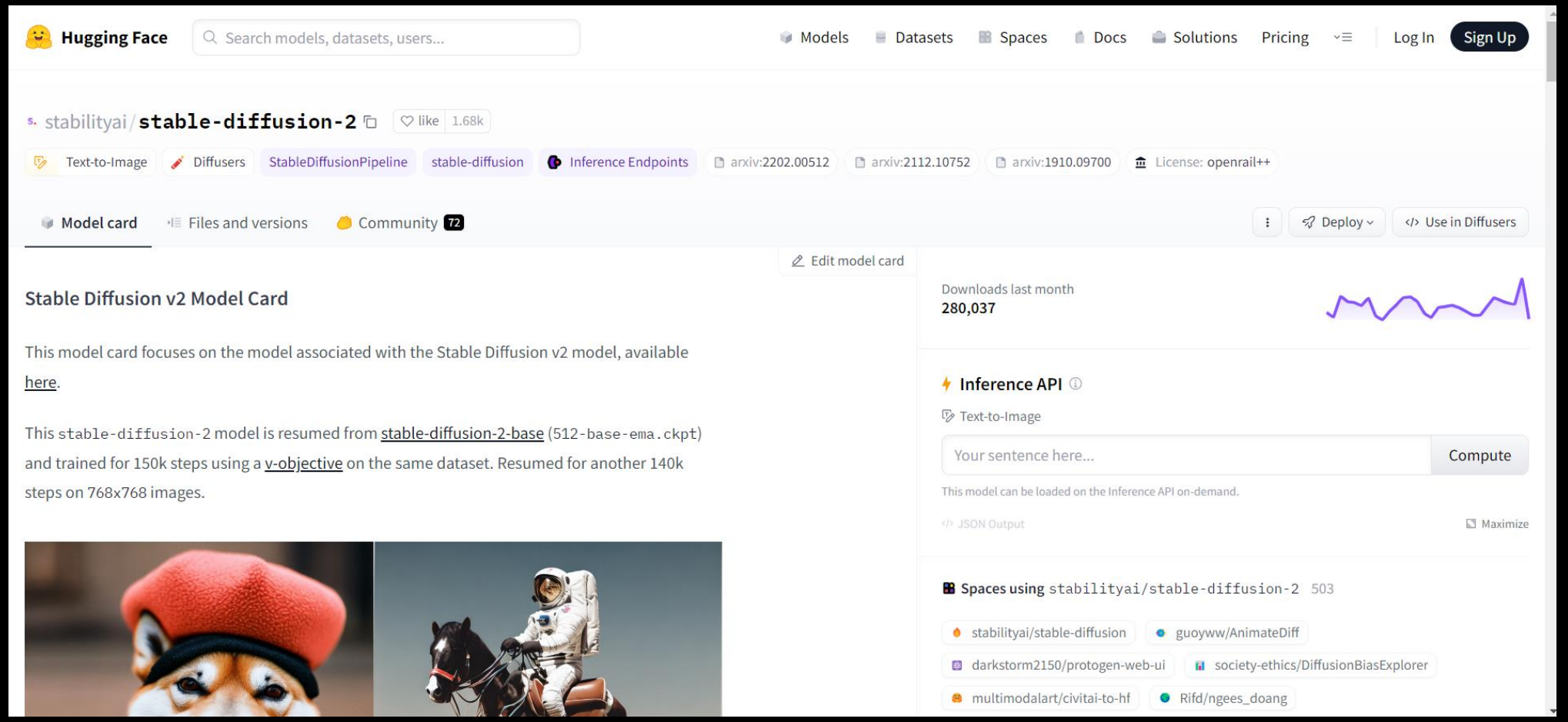
Versatility and Creativity: The model is capable of generating a wide range of images, from realistic photographs to artistic renderings, based on the specificity and creativity of the text prompt.

AI and Machine Learning: Stable Diffusion represents a significant advancement in AI and machine learning, showcasing the potential of these technologies in creative fields.

Accessibility: This technology is becoming increasingly accessible, enabling more people to experiment with AI-driven art and image generation. I.E. Dali3 in GPT4

STABLE DIFFUSION MODEL

WE ARE USING PRE-TRAINED MODEL FROM HUGGING FACE WEBSITE API NAMED AS STABLE-DIFFUSION-V2.



STABLE DIFFUSION MODEL

Step – 1 Setting up the environment
Step – 2 Importing Libraries
Step – 3 Combining Text and image in one result
Step-4 Configurations for Generation with Stable Diffusion
Step-5 Generating Text to Image.

STABLE DIFFUSION MODEL

1.Setting up the Environment

```
!pip install --upgrade diffusers transformers -q
```

```
[2] import torch
    print(torch.cuda.is_available())
```

```
False
```

```
[ ] !pip install torch torchvision torchaudio
```

```
[ ] import torch
    print(torch.cuda.is_available())
```

```
True
```

```
[ ] print(torch.__version__)
    print(torch.version.cuda)
```

```
2.1.0+cu118
11.8
```

```
!pip install diffusers
```

```
!nvcc --version
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2022 NVIDIA Corporation
Built on Wed_Sep_21_10:33:58_PDT_2022
Cuda compilation tools, release 11.8, V11.8.89
Build cuda_11.8.r11.8/compiler.31833905_0
```

```
[ ] !pip install torch==1.12.0+cu113 torchvision==0.13.0+cu113 torchaudio==0.12.0+cu113 -f https://download.pytorch.org/whl/torch_stable.html
```

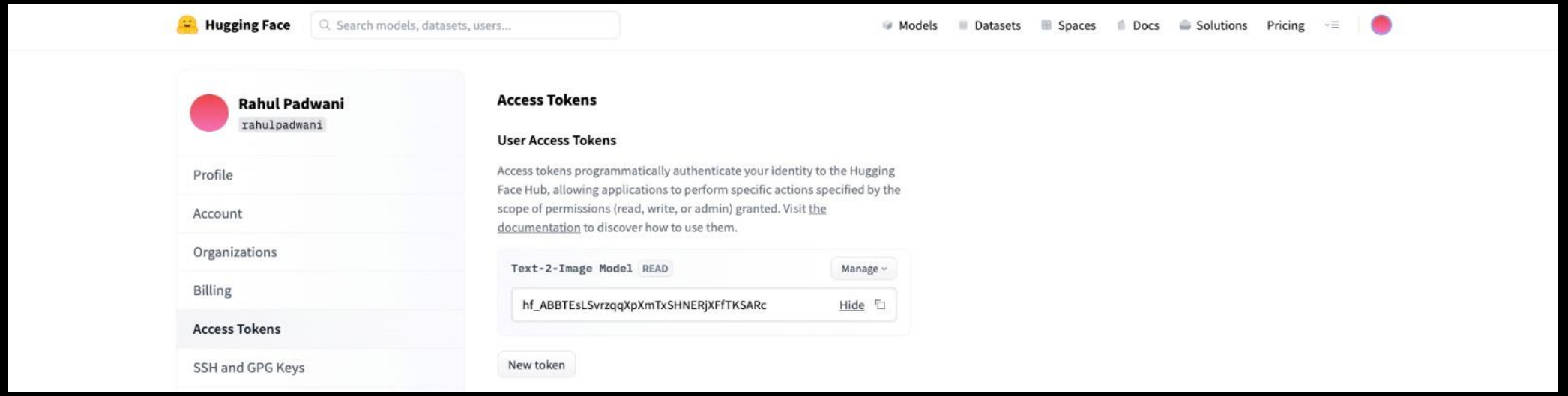
STABLE DIFFUSION MODEL

3.Configurations for AI-Driven Image and Prompt Generation with Stable Diffusion and GPT-2

```
[ ]  class CFG:
         device = "cuda"
         seed = 42
         generator = torch.Generator(device).manual_seed(seed)
         image_gen_steps = 50
         image_gen_model_id = "stabilityai/stable-diffusion-2"
         image_gen_size = (400,400)
         image_gen_guidance_scale = 9
         prompt_gen_model_id = "gpt2"
         prompt_dataset_size = 31000
         prompt_max_length = 12
```

Configuration of the Stable Diffusion.

STABLE DIFFUSION MODEL



4.Initializing Stable Diffusion Pipeline for Image Generation

```
[ ]
    image_gen_model = StableDiffusionPipeline.from_pretrained(
        CFG.image_gen_model_id, torch_dtype=torch.float16,
        revision="fp16", use_auth_token='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', guidance_scale=9
    )
    image_gen_model = image_gen_model.to(CFG.device)
```

```
vae/diffusion_pytorch_model.safetensors not found
Keyword arguments {'guidance_scale': 9} are not expected by StableDiffusionPipeline and will be ignored.
Cannot initialize model with low cpu memory usage because `accelerate` was not found in the environment. Defaulting to `low_cpu_mem_usage=False`. It is strongly recom
```

```
pip install accelerate
```

```
.
```

Loading pipeline components...: 100% 5/5 [00:39<00:00, 7.35s/it]

STABLE DIFFUSION MODEL

5.Image Generation Function Using Stable Diffusion Model

```
[ ]  def generate_image(prompt, model):
         image = model(
             prompt, num_inference_steps=CFG.image_gen_steps,
             generator=CFG.generator,
             guidance_scale=CFG.image_gen_guidance_scale
         ).images[0]

         image = image.resize(CFG.image_gen_size)
         return image
```

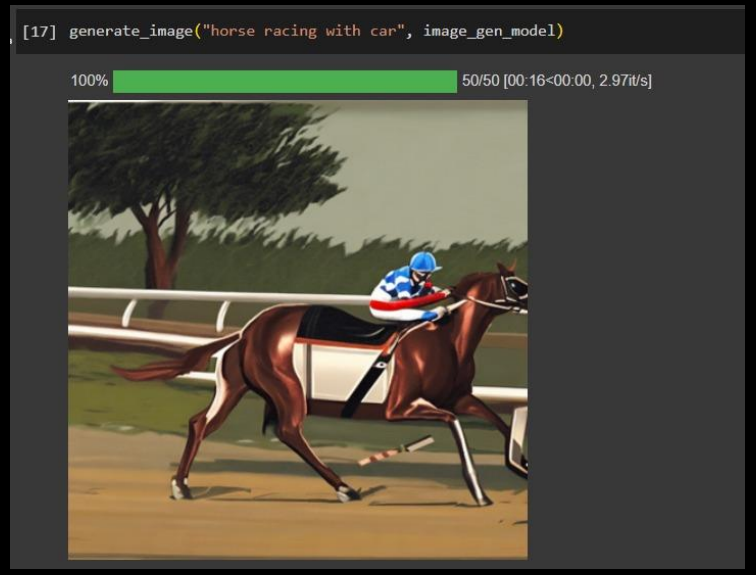
```
generate_image("Dog wearing goggles", image_gen_model)
```

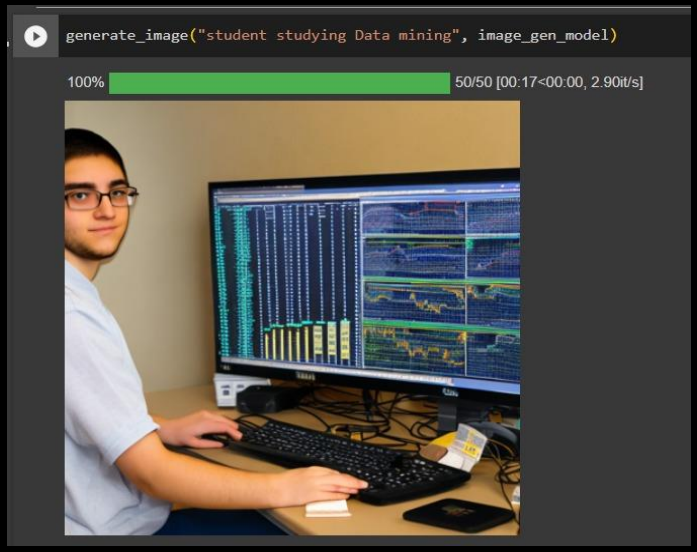
```
100% 50/50 [00:17<00:00, 2.77it/s]
```

Outcome of our model

STABLE DIFFUSION MODEL

- Accuracy of our project is 75-80%
- out of 10 images 8 images came out to be similar what we have given input.
- In the 2nd run of the code the model is giving more accurate description in the input.
- As we train more data the model will give positive result.





PROBLEMS WE FACED

PROBLEMS WE FACED

```
[12] # Verify the content of captions
     print(captions[:5])  # Print the first 5 captions to check their content

     # Verify the type of each caption to ensure they are strings
     print([type(caption) for caption in captions[:5]])
```

```
[nan, nan, nan, nan, nan]
[<class 'float'>, <class 'float'>, <class 'float'>, <class 'float'>, <class 'float'>]
```

```
captions_df
```

	image_name	comment_number	comment
0	image_namel comment_numberl comment	NaN	NaN
1	1000092795.jpgl 0l Two young guys with shaggy ...	NaN	NaN
2	1000092795.jpgl 1l Two young , White males are...	NaN	NaN
3	1000092795.jpgl 2l Two men in green shirts are...	NaN	NaN
4	1000092795.jpgl 3l A man in a blue shirt stand...	NaN	NaN
...	...	...	...
158911	998845445.jpgl 0l A man in shorts and a Hawaii...	NaN	NaN
158912	998845445.jpgl 1l A young man hanging over the...	NaN	NaN
158913	998845445.jpgl 2l A man is leaning off of the ...	NaN	NaN
158914	998845445.jpgl 3l A man riding a small boat in...	NaN	NaN
158915	998845445.jpgl 4l A man on a moored blue and w...	NaN	NaN

158916 rows × 3 columns

- NaN values removal
- Image not properly resized
- Model training .
- GAN Limitation
- GPU Problem
- Computation for processing images

REFERENCES

REFERENCES

1. High-Resolution Image Synthesis with Latent Diffusion Models
Robin Rombach1 * Andreas Blattmann1 ∗ Dominik Lorenz1 Patrick Esser Bjo¨rn Ommer1
2. https://github.com/Stability-AI/stablediffusion/blob/main/README.md
3. https://huggingface.co/stabilityai/stable-diffusion-2-1?text=dog+wearing+goggles
4. https://www.kaggle.com/datasets/adityajn105/flickr30k
5.GAN lecture notes (A Brief Introduction to Generative Adversarial Networks (GAN))

DEMO

THANK YOU!

Under Guidance: Dr. Sunnie Chung

By Rahul Padwani
Jaykumar Mistry