Лабораторна робота № 13. Використання генеративних змагальних нейронних мереж (GANs) для створення зображень

Мета: ознайомитися з принципами роботи та можливостями генеративних моделей на прикладі Stable Diffusion, навчитися генерувати та аналізувати зображення, а також експериментувати з різними параметрами моделі.

Завдання для всіх варіантів

1. ***Завдання 1:*** *дослідіть документацію встановлених бібліотек та поясніть, для чого кожна з них використовується в цьому проєкті.*
2. ***Завдання 2:*** *знайдіть інші моделі на Hugging Face, які можна використати замість "`runwayml/stable-diffusion-v1-5`". Спробуйте завантажити та порівняти їх.*
3. ***Завдання 3:*** *модифікуйте функцію* **`show_images`***, щоб вона могла відображати довільну кількість зображень у сітці (наприклад, 3x3 або 4x4).*
4. ***Завдання 4:*** *додайте ще один параметр для контролю генерації зображень (наприклад: Width та Height, Negative Prompt, температуру тощо) та відповідний віджет.*
5. ***Завдання 5:*** *реалізуйте функцію для інтерполяції між трьома промптами та створіть відповідний віджет.*
6. ***Завдання 6:*** *додайте інші методи аналізу зображень (наприклад, гістограма кольорів або виявлення об'єктів) та відповідні кнопки.*
7. ***Завдання 7:*** *реалізуйте функцію для завантаження користувацького зображення та його аналізу.*

Базова структура роботи

Блок 1: Встановлення та імпорт бібліотек

```
!pip install transformers torch torchvision ipywidgets matplotlib pillow
!pip install git+https://github.com/huggingface/diffusers.git
!pip install opencv-python

import torch
import torchvision.transforms as transforms
from torchvision.utils import make_grid
import matplotlib.pyplot as plt
from PIL import Image
import numpy as np
from diffusers import StableDiffusionPipeline, DPMSolverMultistepScheduler
import ipywidgets as widgets
from IPython.display import display, clear_output
import cv2
import io
import base64
import glob
import os
```

Завдання 1: *дослідіть документацію встановлених бібліотек та поясніть, для чого кожна з них використовується в цьому проєкті.*

Блок 2: Завантаження моделі

```
model_id = "runwayml/stable-diffusion-v1-5"
pipe = StableDiffusionPipeline.from_pretrained(model_id,
torch_dtype=torch.float16)
pipe.scheduler =
DPMSolverMultistepScheduler.from_config(pipe.scheduler.config)
pipe = pipe.to("cuda")
```

Завдання 2: *знайдіть інші моделі на Hugging Face, які можна використати замість* ***"runwayml/stable-diffusion-v1-5"****. Спробуйте завантажити та порівняти їх.*

Блок 3: Функції для генерації та відображення зображень

```
def generate_image(prompt, num_images=1, guidance_scale=7.5,
num_inference_steps=50, seed=None):
    if seed is not None:
        generator = torch.Generator("cuda").manual_seed(seed)
    else:
        generator = None
    with torch.no_grad():
        images = pipe(prompt, num_images_per_prompt=num_images,
guidance_scale=guidance_scale,
                      num_inference_steps=num_inference_steps,
generator=generator).images
    return images

def show_images(images):
    fig, axes = plt.subplots(1, len(images), figsize=(15, 5))
    for i, img in enumerate(images):
        if len(images) == 1:
            axes.imshow(img)
            axes.axis('off')
        else:
            axes[i].imshow(img)
            axes[i].axis('off')
    plt.tight_layout()
    plt.show()

def slerp(t, v0, v1, DOT_THRESHOLD=0.9995):
    dot = torch.sum(v0 * v1 / (torch.norm(v0) * torch.norm(v1)))
    if abs(dot) > DOT_THRESHOLD:
        v2 = (1 - t) * v0 + t * v1
    else:
        theta_0 = torch.acos(dot)
        sin_theta_0 = torch.sin(theta_0)
        theta_t = theta_0 * t
        sin_theta_t = torch.sin(theta_t)
        s0 = torch.sin(theta_0 - theta_t) / sin_theta_0
        s1 = sin_theta_t / sin_theta_0
        v2 = s0 * v0 + s1 * v1
    return v2
```

Завдання 3: *модифікуйте функцію* ***`show_images`****, щоб вона могла відображати довільну кількість зображень у сітці (наприклад, 3x3 або 4x4).*

Блок 4: Інтерфейс для генерації зображень

```
output_images = widgets.Output()

def generate_images_button_clicked(b):
    with output_images:
        clear_output(wait=True)
        images = generate_image(prompt.value, num_images.value,
guidance_scale.value, num_inference_steps.value, seed_widget.value)
        show_images(images)
        generate_button.images = images  # зберігаємо згенеровані
зображення

prompt = widgets.Text(value="A portrait of a smiling person",
description="Prompt:", style={'description_width': 'initial'})
num_images = widgets.IntSlider(min=1, max=4, step=1, value=1,
description="Number of images:", style={'description_width': 'initial'})
guidance_scale = widgets.FloatSlider(min=1, max=20, step=0.5, value=7.5,
description="Guidance scale:", style={'description_width': 'initial'})
num_inference_steps = widgets.IntSlider(min=10, max=100, step=10,
value=50, description="Inference steps:", style={'description_width':
'initial'})
seed_widget = widgets.IntText(value=42, description="Seed:",
style={'description_width': 'initial'})
random_seed_button = widgets.Button(description="Random Seed")
generate_button = widgets.Button(description="Generate Images")

def on_random_seed_click(b):
    seed_widget.value = np.random.randint(0, 1000000)

random_seed_button.on_click(on_random_seed_click)
generate_button.on_click(generate_images_button_clicked)
```

Завдання 4: *додайте ще один параметр для контролю генерації зображень (наприклад: Width та Height, Negative Prompt, температуру тощо) та відповідний віджет.*

Блок 5: Функції для інтерполяції промптів

```
output_interpolation = widgets.Output()

def interpolate_prompts(prompt1, prompt2, steps, guidance_scale,
num_inference_steps, seed=None, transition_speed=1.0):
    images = []
    for i in range(steps + 1):
        t = (i / steps) ** transition_speed
        interpolated_prompt = f"{prompt1} {(1-t):.2f}, {prompt2} {t:.2f}"
        if seed is not None:
            seed_i = seed + i  # змінюємо seed для кожного кроку, щоб
отримати різні, але послідовні зображення
        else:
            seed_i = None
        image = generate_image(interpolated_prompt, 1, guidance_scale,
num_inference_steps, seed_i)[0]
        images.append(image)
    return images

def interpolate_prompts_button_clicked(b):
    with output_interpolation:
        clear_output(wait=True)
        seed = np.random.randint(0, 1000000) if seed_widget_interp.value
== 0 else seed_widget_interp.value
        images = interpolate_prompts(prompt1.value, prompt2.value,
steps.value,
                                     guidance_scale_interp.value,
num_inference_steps_interp.value,
                                     seed, transition_speed.value)
        show_images(images)
        interpolate_button.images = images  # зберігаємо інтерпольовані
зображення

prompt1 = widgets.Text(value="A portrait of a young person",
description="Prompt 1:", style={'description_width': 'initial'})
prompt2 = widgets.Text(value="A portrait of an old person",
description="Prompt 2:", style={'description_width': 'initial'})
steps = widgets.IntSlider(min=2, max=20, step=1, value=10,
description="Steps:", style={'description_width': 'initial'})
guidance_scale_interp = widgets.FloatSlider(min=1, max=20, step=0.5,
value=7.5, description="Guidance scale:", style={'description_width':
'initial'})
num_inference_steps_interp = widgets.IntSlider(min=10, max=100, step=10,
value=50, description="Inference steps:", style={'description_width':
'initial'})
seed_widget_interp = widgets.IntText(value=0, description="Seed (0 for
random):", style={'description_width': 'initial'})
```

```
transition_speed = widgets.FloatSlider(min=0.1, max=2.0, step=0.1,
value=1.0, description="Transition speed:", style={'description_width':
'initial'})
interpolate_button = widgets.Button(description="Interpolate Prompts")

interpolate_button.on_click(interpolate_prompts_button_clicked)
```

Завдання 5: *реалізуйте функцію для інтерполяції між трьома промптами та створіть відповідний віджет.*

Блок 6: Функції для обробки та аналізу зображень

```
def image_to_base64(image):
    buffered = io.BytesIO()
    image.save(buffered, format="PNG")
    return base64.b64encode(buffered.getvalue()).decode()

def analyze_image(image):
    img_array = np.array(image)
    gray = cv2.cvtColor(img_array, cv2.COLOR_RGB2GRAY)
    edges = cv2.Canny(gray, 100, 200)
    plt.figure(figsize=(10, 5))
    plt.subplot(121), plt.imshow(image), plt.title('Original')
    plt.subplot(122), plt.imshow(edges, cmap='gray'), plt.title('Edge
Detection')
    plt.show()

def analyze_image_button_clicked(b):
    if hasattr(generate_button, 'images') and len(generate_button.images)
> 0:
        analyze_image(generate_button.images[0])
    else:
        print("No image generated yet. Please generate an image first.")

analyze_button = widgets.Button(description="Analyze Last Generated
Image")
analyze_button.on_click(analyze_image_button_clicked)
```

Завдання 6: *додайте інші методи аналізу зображень (наприклад, гістограма кольорів або виявлення об'єктів) та відповідні кнопки.*

Блок 7: Функції для збереження та завантаження зображень/відео

```
def save_image_button_clicked(b):
    if hasattr(generate_button, 'images') and len(generate_button.images)
> 0:
        generate_button.images[0].save("generated_image.png")
        print("Image saved as 'generated_image.png'")
    else:
        print("No image generated yet. Please generate an image first.")

save_button = widgets.Button(description="Save Last Generated Image")
save_button.on_click(save_image_button_clicked)

def create_smooth_video(images, output_path, fps):
    height, width, _ = np.array(images[0]).shape
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    out = cv2.VideoWriter(output_path, fourcc, fps, (width, height))

    for i in range(len(images)-1):
        img1 = cv2.cvtColor(np.array(images[i]), cv2.COLOR_RGB2BGR)
        img2 = cv2.cvtColor(np.array(images[i+1]), cv2.COLOR_RGB2BGR)

        flow = cv2.calcOpticalFlowFarneback(cv2.cvtColor(img1,
cv2.COLOR_BGR2GRAY),
                                            cv2.cvtColor(img2,
cv2.COLOR_BGR2GRAY),
                                            None, 0.5, 3, 15, 3, 5, 1.2,
0)

        for j in range(fps):
            t = j / fps
            warped = cv2.remap(img1, (flow[..., 0] * t +
np.arange(width)).astype(np.float32),
                               (flow[..., 1] * t + np.arange(height)[:,
np.newaxis]).astype(np.float32),
                               cv2.INTER_LINEAR)
            out.write(warped)

    out.release()

def create_video_button_clicked(b):
    if hasattr(interpolate_button, 'images') and
len(interpolate_button.images) > 0:
        output_path = "interpolation_video.mp4"
        create_smooth_video(interpolate_button.images, output_path,
fps.value)
        print(f"Video saved as '{output_path}'")
    else:
        print("No interpolated images yet. Please interpolate prompts
first.")
```

```
create_video_button = widgets.Button(description="Create Smooth Video")
create_video_button.on_click(create_video_button_clicked)

fps = widgets.IntSlider(min=10, max=60, step=1, value=30,
description="FPS:", style={'description_width': 'initial'})
```

Завдання 7: *реалізуйте функцію для завантаження користувацького зображення та його аналізу.*

Блок 8: Головний інтерфейс

```
# @title Головний інтерфейс
tab = widgets.Tab()
tab.children = [
    widgets.VBox([
        widgets.HBox([prompt, num_images]),
        widgets.HBox([guidance_scale, num_inference_steps]),
        widgets.HBox([seed_widget, random_seed_button]),
        generate_button,
        output_images,
        widgets.HBox([analyze_button, save_button])
    ]),
    widgets.VBox([
        widgets.HBox([prompt1, prompt2]),
        widgets.HBox([steps, guidance_scale_interp,
num_inference_steps_interp]),
        widgets.HBox([seed_widget_interp, transition_speed]),
        interpolate_button,
        output_interpolation,
        widgets.HBox([fps, create_video_button])
    ])
]
tab.set_title(0, "Generate Images")
tab.set_title(1, "Interpolate Prompts")
display(tab)
```

Рисунок 1.1. Результат використання SD 1.5 в базовому прикладі

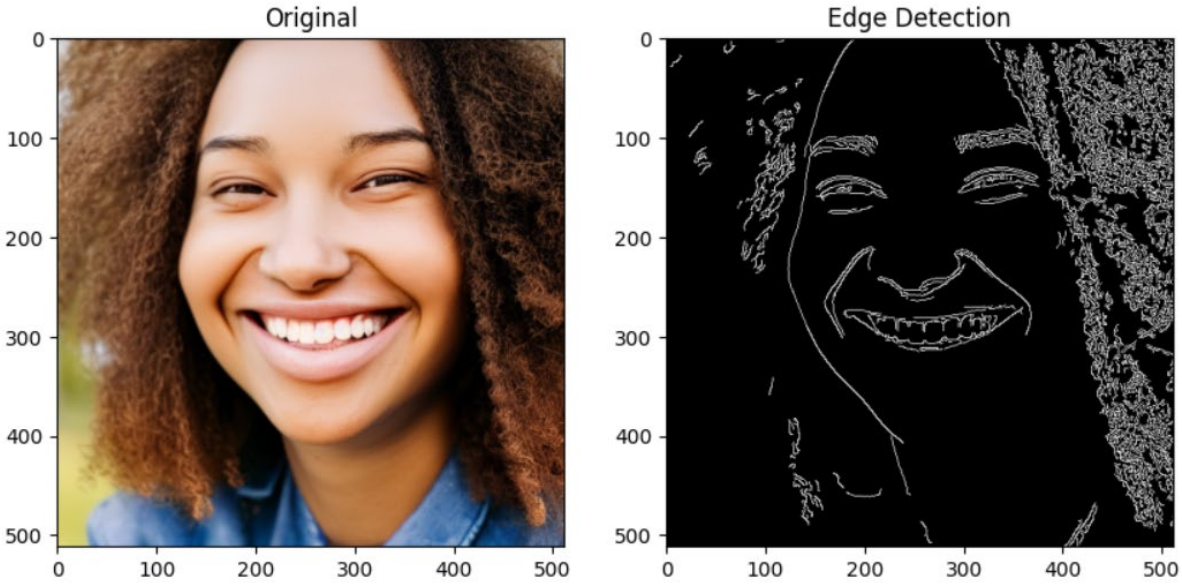

Рисунок 1.2. Результат аналізу зображення в базовому прикладі

Рисунок 1.3. Результат інтерполяції зображень в базовому прикладі

Приклад виконання роботи

Блок 1: Встановлення та імпорт бібліотек

```
!pip install transformers torch torchvision ipywidgets matplotlib pillow
!pip install git+https://github.com/huggingface/diffusers.git
!pip install opencv-python

import torch
import torchvision.transforms as transforms
from torchvision.utils import make_grid
import matplotlib.pyplot as plt
from PIL import Image
import numpy as np
from diffusers import StableDiffusionPipeline, DPMSolverMultistepScheduler
import ipywidgets as widgets
from IPython.display import display, clear_output
import cv2
import io
import base64
import glob
import os
```

Завдання 1: *дослідіть документацію встановлених бібліотек та поясніть, для чого кожна з них використовується в цьому проєкті.*

Відповідь:

`torch` – ...

`torchvision.transforms` – ...

...

Блок 2: Завантаження моделі

```
model_id = "runwayml/stable-diffusion-v1-5"
pipe = StableDiffusionPipeline.from_pretrained(model_id,
torch_dtype=torch.float16)
pipe.scheduler =
DPMSolverMultistepScheduler.from_config(pipe.scheduler.config)
pipe = pipe.to("cuda")
```

Завдання 2: *знайдіть інші моделі на Hugging Face, які можна використати замість "**runwayml/stable-diffusion-v1-5**". Спробуйте завантажити та порівняти їх.*

Відповідь:

```
models = {
    "SD 1.5": "runwayml/stable-diffusion-v1-5",
    "SD 2.1": "stabilityai/stable-diffusion-2-1"
}

current_model = "SD 1.5"
pipe = StableDiffusionPipeline.from_pretrained(models[current_model],
torch_dtype=torch.float16)
pipe.scheduler =
DPMSolverMultistepScheduler.from_config(pipe.scheduler.config)
pipe = pipe.to("cuda")

def change_model(model_name):
    global pipe, current_model
    current_model = model_name
    pipe = StableDiffusionPipeline.from_pretrained(models[model_name],
torch_dtype=torch.float16)
    pipe.scheduler =
DPMSolverMultistepScheduler.from_config(pipe.scheduler.config)
    pipe = pipe.to("cuda")
```

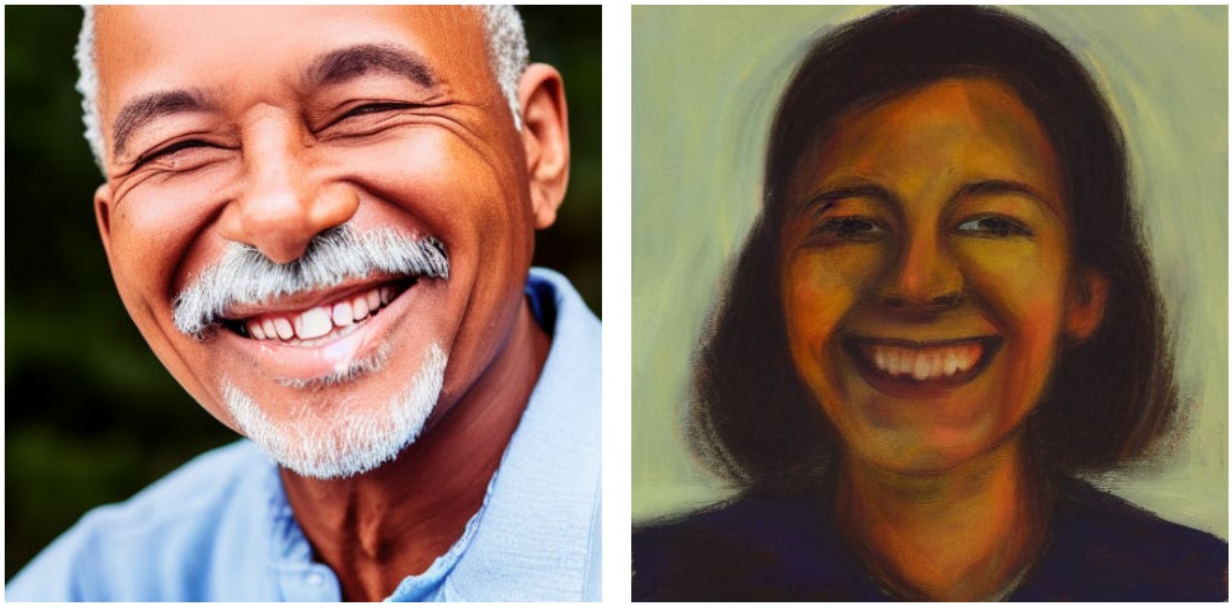

Рисунок 1.4. Результат використання SD 1.5 (за однакових параметрів)

Рисунок 1.5. Результат використання SD 2.1 (за однакових параметрів)

Блок 3: Функції для генерації та відображення зображень

```
def generate_image(prompt, num_images=1, guidance_scale=7.5,
num_inference_steps=50, seed=None):
    if seed is not None:
        generator = torch.Generator("cuda").manual_seed(seed)
    else:
        generator = None
    with torch.no_grad():
        images = pipe(prompt, num_images_per_prompt=num_images,
guidance_scale=guidance_scale,
                      num_inference_steps=num_inference_steps,
generator=generator).images
    return images

def show_images(images):
    fig, axes = plt.subplots(1, len(images), figsize=(15, 5))
    for i, img in enumerate(images):
        if len(images) == 1:
            axes.imshow(img)
            axes.axis('off')
        else:
            axes[i].imshow(img)
            axes[i].axis('off')
    plt.tight_layout()
    plt.show()

def slerp(t, v0, v1, DOT_THRESHOLD=0.9995):
    dot = torch.sum(v0 * v1 / (torch.norm(v0) * torch.norm(v1)))
    if abs(dot) > DOT_THRESHOLD:
        v2 = (1 - t) * v0 + t * v1
    else:
        theta_0 = torch.acos(dot)
        sin_theta_0 = torch.sin(theta_0)
        theta_t = theta_0 * t
        sin_theta_t = torch.sin(theta_t)
        s0 = torch.sin(theta_0 - theta_t) / sin_theta_0
        s1 = sin_theta_t / sin_theta_0
        v2 = s0 * v0 + s1 * v1
    return v2
```

Завдання 3: *модифікуйте функцію* ***`show_images`****, щоб вона могла відображати довільну кількість зображень у сітці (наприклад, 3x3 або 4x4).*

Відповідь:

```
def show_images(images, rows=None, cols=None):
    n = len(images)
    if n == 0:
        print("No images to display")
```

```
        return

    if rows is None and cols is None:
        cols = int(np.ceil(np.sqrt(n)))
        rows = int(np.ceil(n / cols))
    elif rows is None:
        rows = int(np.ceil(n / cols))
    elif cols is None:
        cols = int(np.ceil(n / rows))

    fig, axes = plt.subplots(rows, cols, figsize=(cols*5, rows*5))

    if n == 1:
        axes = np.array([axes])  # Перетворюємо в масив для уніфікації
обробки
    else:
        axes = axes.flatten()

    for i, img in enumerate(images):
        if i < n:
            axes[i].imshow(img)
            axes[i].axis('off')
        else:
            fig.delaxes(axes[i])

    for i in range(n, len(axes)):
        fig.delaxes(axes[i])

    plt.tight_layout()
    plt.show()
```

Блок 4: Інтерфейс для генерації зображень

```
output_images = widgets.Output()

def generate_images_button_clicked(b):
    with output_images:
        clear_output(wait=True)
        images = generate_image(prompt.value, num_images.value,
guidance_scale.value, num_inference_steps.value, seed_widget.value)
        show_images(images)
        generate_button.images = images  # зберігаємо згенеровані
зображення

prompt = widgets.Text(value="A portrait of a smiling person",
description="Prompt:", style={'description_width': 'initial'})
num_images = widgets.IntSlider(min=1, max=4, step=1, value=1,
description="Number of images:", style={'description_width': 'initial'})
guidance_scale = widgets.FloatSlider(min=1, max=20, step=0.5, value=7.5,
description="Guidance scale:", style={'description_width': 'initial'})
num_inference_steps = widgets.IntSlider(min=10, max=100, step=10,
value=50, description="Inference steps:", style={'description_width':
'initial'})
seed_widget = widgets.IntText(value=42, description="Seed:",
style={'description_width': 'initial'})
random_seed_button = widgets.Button(description="Random Seed")
generate_button = widgets.Button(description="Generate Images")

def on_random_seed_click(b):
    seed_widget.value = np.random.randint(0, 1000000)

random_seed_button.on_click(on_random_seed_click)
generate_button.on_click(generate_images_button_clicked)
```

Завдання 4: *додайте ще один параметр для контролю генерації зображень (наприклад: Width та Height, Negative Prompt, температуру тощо) та відповідний віджет.*

Відповідь:

```
def generate_image(prompt, negative_prompt="", num_images=1,
guidance_scale=7.5, num_inference_steps=50, seed=None, width=512,
height=512, temperature=1.0):
    if seed is not None:
        generator = torch.Generator("cuda").manual_seed(seed)
    else:
        generator = None

    with torch.no_grad():
        # Генеруємо латентний простір з типом float16
        latents = torch.randn((num_images, pipe.unet.config.in_channels,
height // 8, width // 8),
```

```
                                generator=generator, device="cuda",
dtype=torch.float16) * temperature

        # Перетворюємо промпти в тензори з типом float16
        text_inputs = pipe.tokenizer(
            prompt,
            padding="max_length",
            max_length=pipe.tokenizer.model_max_length,
            truncation=True,
            return_tensors="pt",
        )
        text_input_ids = text_inputs.input_ids.to(pipe.device)
        prompt_embeds = pipe.text_encoder(text_input_ids)[0].half()

        # Обробляємо негативний промпт
        if negative_prompt:
            uncond_input = pipe.tokenizer(
                negative_prompt,
                padding="max_length",
                max_length=text_input_ids.shape[-1],
                truncation=True,
                return_tensors="pt",
            )
            uncond_input_ids = uncond_input.input_ids.to(pipe.device)
            negative_prompt_embeds =
pipe.text_encoder(uncond_input_ids)[0].half()
        else:
            # Якщо негативний промпт не вказано, створюємо пустий ембединг
            negative_prompt_embeds = torch.zeros_like(prompt_embeds)

        # Переконуємось, що форми співпадають
        if prompt_embeds.shape[1] != negative_prompt_embeds.shape[1]:
            negative_prompt_embeds = negative_prompt_embeds.repeat(1,
prompt_embeds.shape[1], 1)

        images = pipe(prompt_embeds=prompt_embeds,
                      negative_prompt_embeds=negative_prompt_embeds,
                      num_images_per_prompt=num_images,
                      guidance_scale=guidance_scale,
                      num_inference_steps=num_inference_steps,
                      latents=latents,
                      width=width,
                      height=height).images
    return images

output_images = widgets.Output()

def generate_images_button_clicked(b):
    with output_images:
```

```
        clear_output(wait=True)
        images = generate_image(prompt.value, negative_prompt.value,
num_images.value,
                                guidance_scale.value,
num_inference_steps.value, seed_widget.value,
                                width.value, height.value,
temperature.value)
        show_images(images)
        generate_button.images = images

prompt = widgets.Text(value="A portrait of a smiling person",
description="Prompt:", style={'description_width': 'initial'})
negative_prompt = widgets.Text(value="", description="Negative Prompt:",
style={'description_width': 'initial'})
num_images = widgets.IntSlider(min=1, max=9, step=1, value=1,
description="Number of images:", style={'description_width': 'initial'})
guidance_scale = widgets.FloatSlider(min=1, max=20, step=0.5, value=7.5,
description="Guidance scale:", style={'description_width': 'initial'})
num_inference_steps = widgets.IntSlider(min=10, max=100, step=10,
value=50, description="Inference steps:", style={'description_width':
'initial'})
seed_widget = widgets.IntText(value=42, description="Seed:",
style={'description_width': 'initial'})
width = widgets.IntSlider(min=256, max=1024, step=64, value=512,
description="Width:", style={'description_width': 'initial'})
height = widgets.IntSlider(min=256, max=1024, step=64, value=512,
description="Height:", style={'description_width': 'initial'})
temperature = widgets.FloatSlider(min=0.1, max=2.0, step=0.1, value=1.0,
description="Temperature:", style={'description_width': 'initial'})

random_seed_button = widgets.Button(description="Random Seed")
generate_button = widgets.Button(description="Generate Images")

def on_random_seed_click(b):
    seed_widget.value = np.random.randint(0, 1000000)

random_seed_button.on_click(on_random_seed_click)
generate_button.on_click(generate_images_button_clicked)
```

Блок 5: Функції для інтерполяції промптів

```
output_interpolation = widgets.Output()

def interpolate_prompts(prompt1, prompt2, steps, guidance_scale,
num_inference_steps, seed=None, transition_speed=1.0):
    images = []
    for i in range(steps + 1):
        t = (i / steps) ** transition_speed
        interpolated_prompt = f"{prompt1} {(1-t):.2f}, {prompt2} {t:.2f}"
        if seed is not None:
            seed_i = seed + i  # змінюємо seed для кожного кроку, щоб
отримати різні, але послідовні зображення
        else:
            seed_i = None
        image = generate_image(interpolated_prompt, 1, guidance_scale,
num_inference_steps, seed_i)[0]
        images.append(image)
    return images

def interpolate_prompts_button_clicked(b):
    with output_interpolation:
        clear_output(wait=True)
        seed = np.random.randint(0, 1000000) if seed_widget_interp.value
== 0 else seed_widget_interp.value
        images = interpolate_prompts(prompt1.value, prompt2.value,
steps.value,
                                     guidance_scale_interp.value,
num_inference_steps_interp.value,
                                     seed, transition_speed.value)
        show_images(images)
        interpolate_button.images = images  # зберігаємо інтерпольовані
зображення

prompt1 = widgets.Text(value="A portrait of a young person",
description="Prompt 1:", style={'description_width': 'initial'})
prompt2 = widgets.Text(value="A portrait of an old person",
description="Prompt 2:", style={'description_width': 'initial'})
steps = widgets.IntSlider(min=2, max=20, step=1, value=10,
description="Steps:", style={'description_width': 'initial'})
guidance_scale_interp = widgets.FloatSlider(min=1, max=20, step=0.5,
value=7.5, description="Guidance scale:", style={'description_width':
'initial'})
num_inference_steps_interp = widgets.IntSlider(min=10, max=100, step=10,
value=50, description="Inference steps:", style={'description_width':
'initial'})
seed_widget_interp = widgets.IntText(value=0, description="Seed (0 for
random):", style={'description_width': 'initial'})
```

```
transition_speed = widgets.FloatSlider(min=0.1, max=2.0, step=0.1,
value=1.0, description="Transition speed:", style={'description_width':
'initial'})
interpolate_button = widgets.Button(description="Interpolate Prompts")

interpolate_button.on_click(interpolate_prompts_button_clicked)
```

Завдання 5: *реалізуйте функцію для інтерполяції між трьома промптами та створіть відповідний віджет.*

Відповідь:

```
output_interpolation = widgets.Output()

def interpolate_prompts(prompt1, prompt2, prompt3, steps, guidance_scale,
num_inference_steps, seed=None, transition_speed=1.0):
    images = []
    total_steps = steps * 2

    for i in range(total_steps + 1):
        t = (i / total_steps) ** transition_speed
        if i <= steps:
            interpolated_prompt = f"{prompt1} {(1-t*2):.2f}, {prompt2}
{t*2:.2f}"
        else:
            t = (i - steps) / steps
            interpolated_prompt = f"{prompt2} {(1-t):.2f}, {prompt3}
{t:.2f}"

        if seed is not None:
            seed_i = seed + i
        else:
            seed_i = None
        image = generate_image(interpolated_prompt, num_images=1,
guidance_scale=guidance_scale,
                               num_inference_steps=num_inference_steps,
seed=seed_i)[0]
        images.append(image)
    return images

def interpolate_prompts_button_clicked(b):
    with output_interpolation:
        clear_output(wait=True)
        seed = np.random.randint(0, 1000000) if seed_widget_interp.value
== 0 else seed_widget_interp.value
        images = interpolate_prompts(prompt1.value, prompt2.value,
prompt3.value, steps.value,
                                     guidance_scale_interp.value,
num_inference_steps_interp.value,
```

```
                                        seed, transition_speed.value)
        show_images(images)
        interpolate_button.images = images

prompt1 = widgets.Text(value="A portrait of a young person",
description="Prompt 1:", style={'description_width': 'initial'})
prompt2 = widgets.Text(value="A portrait of a middle-aged person",
description="Prompt 2:", style={'description_width': 'initial'})
prompt3 = widgets.Text(value="A portrait of an old person",
description="Prompt 3:", style={'description_width': 'initial'})
steps = widgets.IntSlider(min=2, max=20, step=1, value=10,
description="Steps per transition:", style={'description_width':
'initial'})
guidance_scale_interp = widgets.FloatSlider(min=1, max=20, step=0.5,
value=7.5, description="Guidance scale:", style={'description_width':
'initial'})
num_inference_steps_interp = widgets.IntSlider(min=10, max=100, step=10,
value=50, description="Inference steps:", style={'description_width':
'initial'})
seed_widget_interp = widgets.IntText(value=0, description="Seed (0 for
random):", style={'description_width': 'initial'})
transition_speed = widgets.FloatSlider(min=0.1, max=2.0, step=0.1,
value=1.0, description="Transition speed:", style={'description_width':
'initial'})
interpolate_button = widgets.Button(description="Interpolate Prompts")

interpolate_button.on_click(interpolate_prompts_button_clicked)
```

Блок 6: Функції для обробки та аналізу зображень

```
def image_to_base64(image):
    buffered = io.BytesIO()
    image.save(buffered, format="PNG")
    return base64.b64encode(buffered.getvalue()).decode()

def analyze_image(image):
    img_array = np.array(image)
    gray = cv2.cvtColor(img_array, cv2.COLOR_RGB2GRAY)
    edges = cv2.Canny(gray, 100, 200)
    plt.figure(figsize=(10, 5))
    plt.subplot(121), plt.imshow(image), plt.title('Original')
    plt.subplot(122), plt.imshow(edges, cmap='gray'), plt.title('Edge
Detection')
    plt.show()

def analyze_image_button_clicked(b):
    if hasattr(generate_button, 'images') and len(generate_button.images)
> 0:
        analyze_image(generate_button.images[0])
    else:
        print("No image generated yet. Please generate an image first.")

analyze_button = widgets.Button(description="Analyze Last Generated
Image")
analyze_button.on_click(analyze_image_button_clicked)
```

Завдання 6: *додайте інші методи аналізу зображень (наприклад, гістограма кольорів або виявлення об'єктів) та відповідні кнопки.*

Відповідь:

```
import traceback

def analyze_image(image):
    try:
        img_array = np.array(image)
        print(f"Image shape: {img_array.shape}")
        print(f"Image dtype: {img_array.dtype}")

        # Edge detection
        gray = cv2.cvtColor(img_array, cv2.COLOR_RGB2GRAY)
        edges = cv2.Canny(gray, 100, 200)

        # Color histogram
        color = ('r','g','b')
        plt.figure(figsize=(20, 15))
        plt.subplot(221)
        plt.title("Color Histogram")
```

```
        for i, col in enumerate(color):
            histr = cv2.calcHist([img_array],[i],None,[256],[0,256])
            plt.plot(histr, color = col)
            plt.xlim([0,256])

        # Display results
        plt.subplot(222)
        plt.imshow(image)
        plt.title('Original')
        plt.axis('off')

        plt.subplot(223)
        plt.imshow(edges, cmap='gray')
        plt.title('Edge Detection')
        plt.axis('off')

        plt.subplot(224)
        plt.imshow(img_array)
        plt.title('RGB Image')
        plt.axis('off')

        plt.tight_layout()
        plt.show()

        print("Image analysis completed successfully.")
    except Exception as e:
        print(f"An error occurred during image analysis: {str(e)}")
        print("Traceback:")
        traceback.print_exc()

def analyze_image_button_clicked(b):
    if hasattr(generate_button, 'images') and len(generate_button.images)
> 0:
        print("Starting image analysis...")
        analyze_image(generate_button.images[0])
    else:
        print("No image generated yet. Please generate an image first.")
```

```
Image shape: (512, 512, 3)
Image dtype: uint8
```

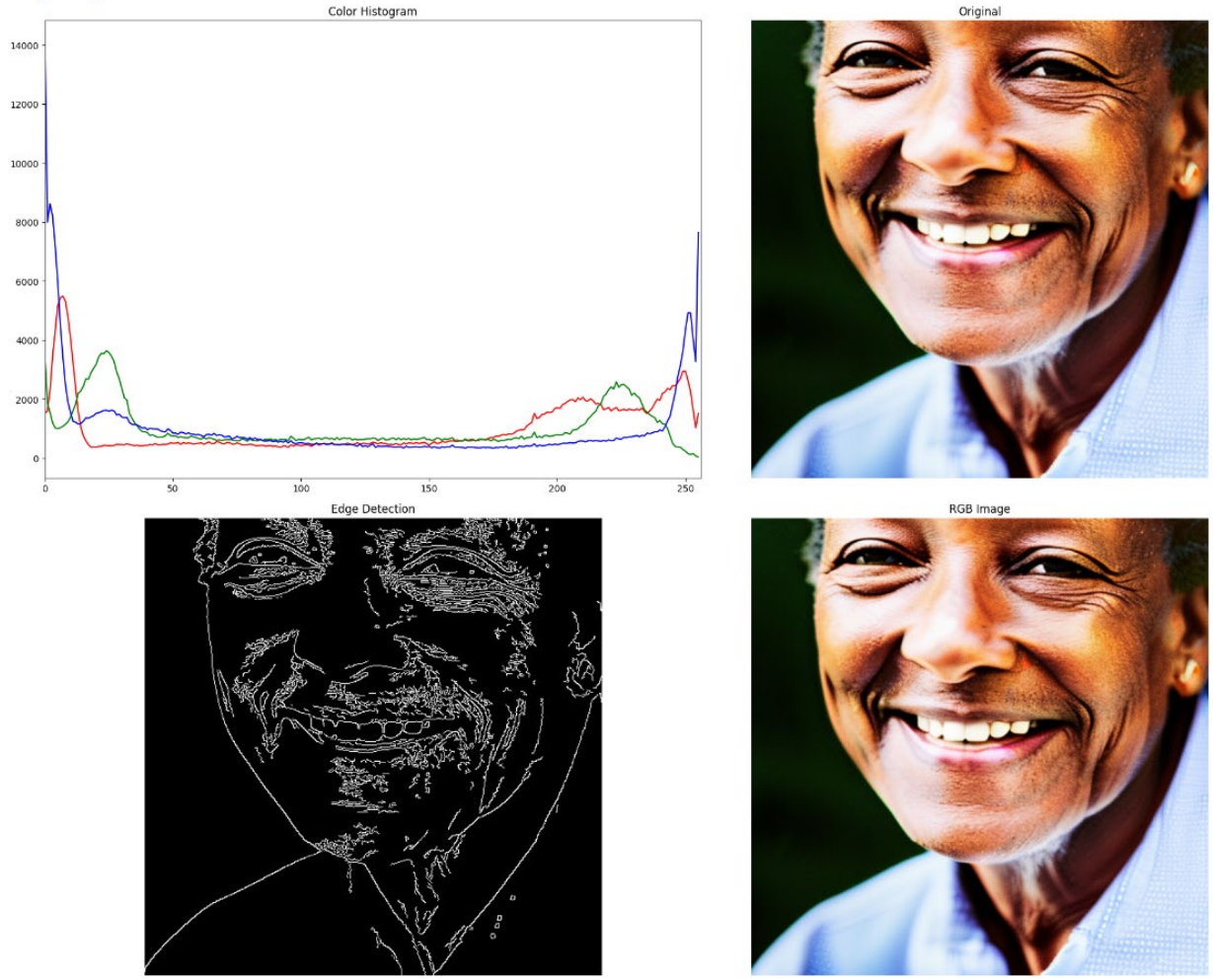

Рисунок 1.6. Приклад розширеного аналізу зображень

Блок 7: Функції для збереження та завантаження зображень/відео

```
def save_image_button_clicked(b):
    if hasattr(generate_button, 'images') and len(generate_button.images)
> 0:
        generate_button.images[0].save("generated_image.png")
        print("Image saved as 'generated_image.png'")
    else:
        print("No image generated yet. Please generate an image first.")

save_button = widgets.Button(description="Save Last Generated Image")
save_button.on_click(save_image_button_clicked)

def create_smooth_video(images, output_path, fps):
    height, width, _ = np.array(images[0]).shape
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    out = cv2.VideoWriter(output_path, fourcc, fps, (width, height))

    for i in range(len(images)-1):
        img1 = cv2.cvtColor(np.array(images[i]), cv2.COLOR_RGB2BGR)
        img2 = cv2.cvtColor(np.array(images[i+1]), cv2.COLOR_RGB2BGR)

        flow = cv2.calcOpticalFlowFarneback(cv2.cvtColor(img1,
cv2.COLOR_BGR2GRAY),
                                            cv2.cvtColor(img2,
cv2.COLOR_BGR2GRAY),
                                            None, 0.5, 3, 15, 3, 5, 1.2,
0)

        for j in range(fps):
            t = j / fps
            warped = cv2.remap(img1, (flow[..., 0] * t +
np.arange(width)).astype(np.float32),
                               (flow[..., 1] * t + np.arange(height)[:,
np.newaxis]).astype(np.float32),
                               cv2.INTER_LINEAR)
            out.write(warped)

    out.release()

def create_video_button_clicked(b):
    if hasattr(interpolate_button, 'images') and
len(interpolate_button.images) > 0:
        output_path = "interpolation_video.mp4"
        create_smooth_video(interpolate_button.images, output_path,
fps.value)
        print(f"Video saved as '{output_path}'")
    else:
        print("No interpolated images yet. Please interpolate prompts
first.")
```

```
create_video_button = widgets.Button(description="Create Smooth Video")
create_video_button.on_click(create_video_button_clicked)

fps = widgets.IntSlider(min=10, max=60, step=1, value=30,
description="FPS:", style={'description_width': 'initial'})
```

Завдання 7: *реалізуйте функцію для завантаження користувацького зображення та його аналізу.*

Відповідь:

```
def upload_and_analyze_image(b):
    clear_output()
    print("Please upload an image file.")
    uploaded = files.upload()

    for filename in uploaded.keys():
        content = uploaded[filename]
        image = Image.open(io.BytesIO(content))
        analyze_image(image)

upload_button = widgets.Button(description="Upload and Analyze Image")
upload_button.on_click(upload_and_analyze_image)
```

Зміни у графічному інтерфейсі:

```
model_dropdown = widgets.Dropdown(options=list(models.keys()),
value=current_model, description="Model:")
model_dropdown.observe(lambda change: change_model(change.new),
names='value')

tab = widgets.Tab()
tab.children = [
    widgets.VBox([
        model_dropdown,
        widgets.HBox([prompt, negative_prompt]),
        widgets.HBox([num_images, guidance_scale, num_inference_steps]),
        widgets.HBox([seed_widget, random_seed_button]),
        widgets.HBox([width, height, temperature]),
        generate_button,
        output_images,
        widgets.HBox([analyze_button, upload_button])
    ]),
    widgets.VBox([
        widgets.HBox([prompt1, prompt2, prompt3]),
        widgets.HBox([steps, guidance_scale_interp,
num_inference_steps_interp]),
        widgets.HBox([seed_widget_interp, transition_speed]),
```

```
        interpolate_button,
        output_interpolation
    ])
]
tab.set_title(0, "Generate Images")
tab.set_title(1, "Interpolate Prompts")
display(tab)
```

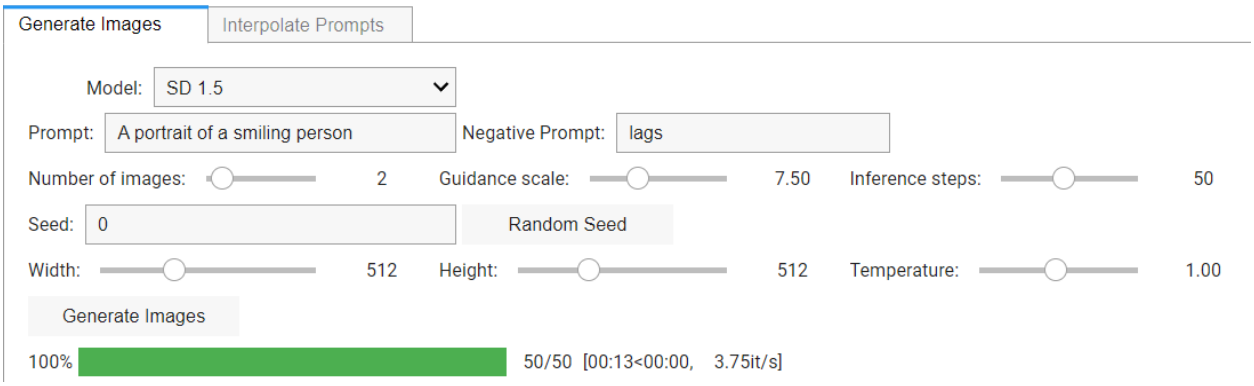

Рисунок 1.7. Вигляд інтерфейсу №1

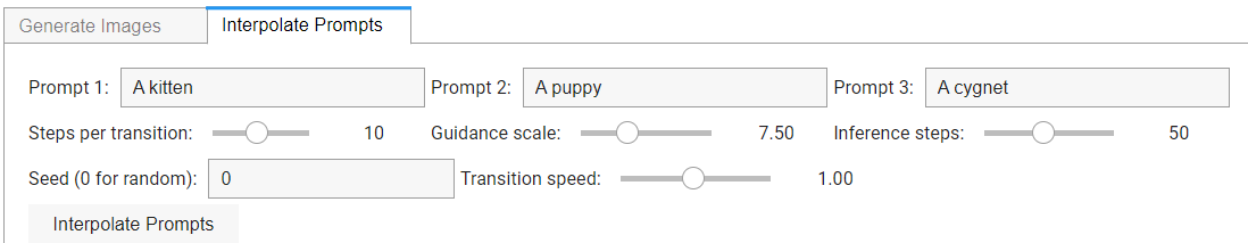

Рисунок 1.8. Вигляд інтерфейсу №2

```
Image shape: (1697, 1200, 3)
Image dtype: uint8
```

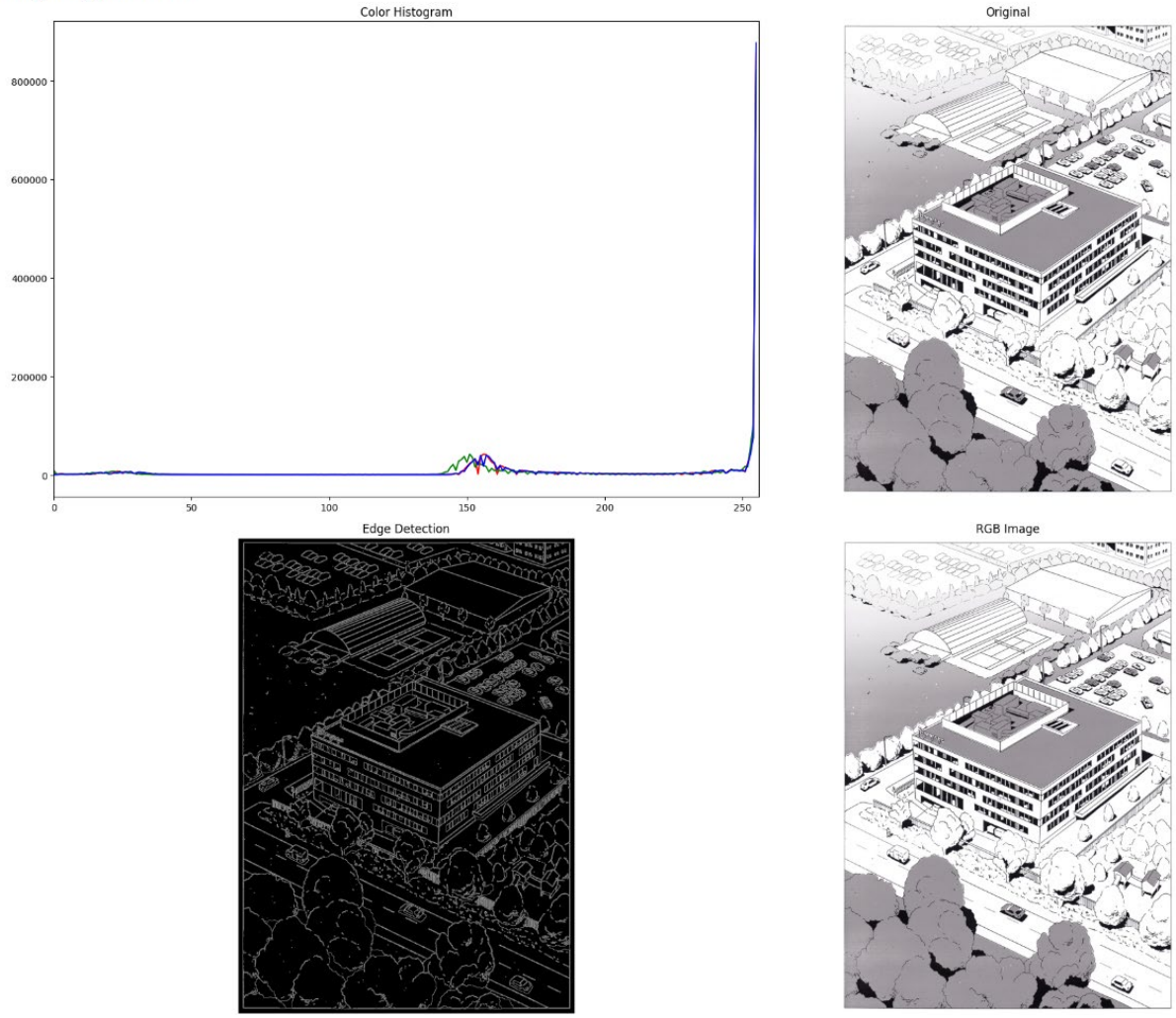

Рисунок 1.9. Приклад аналізу зображення, що завантажується

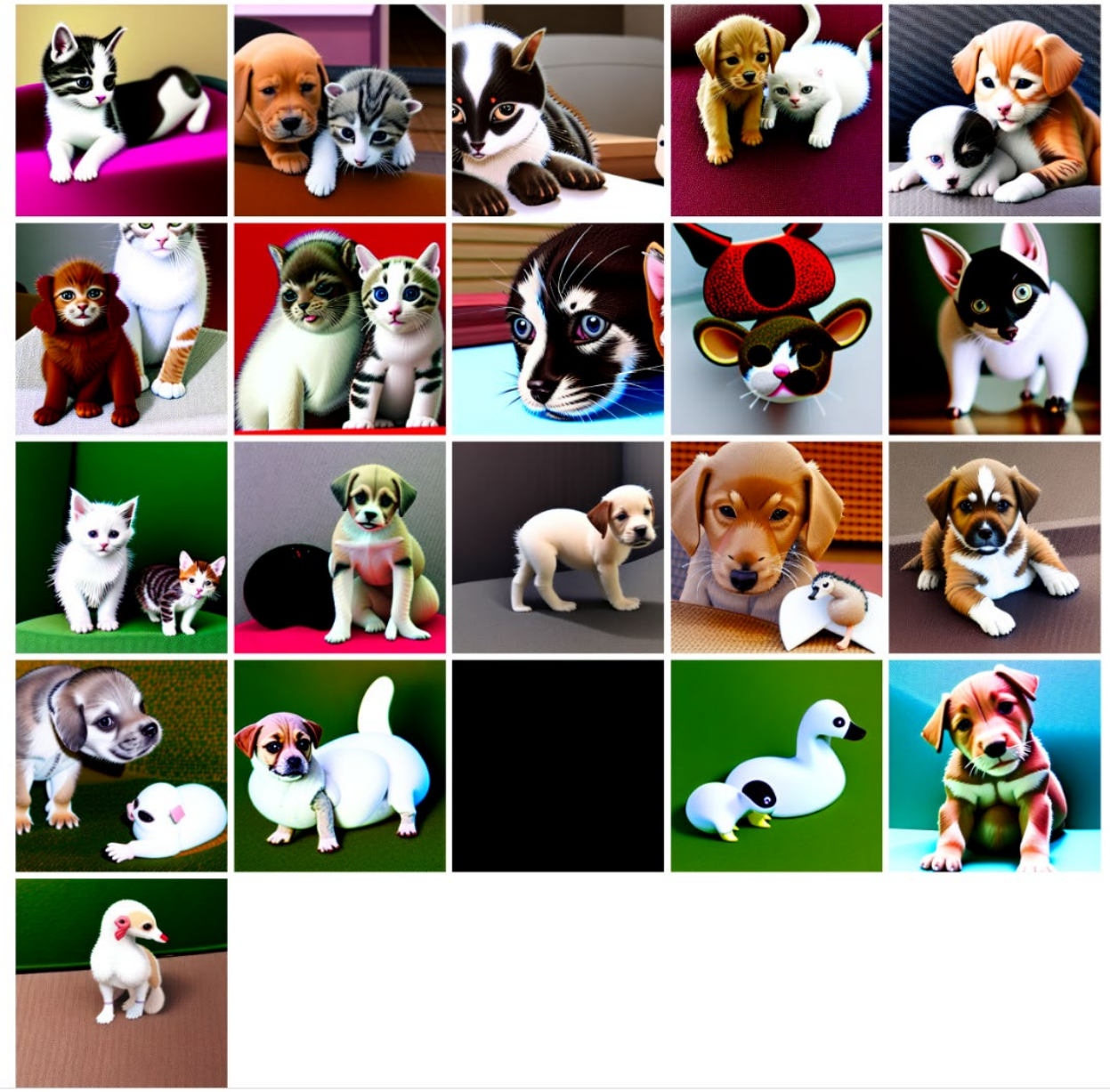

Рисунок 1.10. Приклад інтерполяції зображень за 3 промптами

Висновки: …

Welcome to Google Colab

https://colab.research.google.com/drive/1x2T9VPvZntyxEp45KsNist6yJJe4bqz4?usp=sharing