

Nama : Muhamad Rizq Rihaz

NIM : 1103210192

## Task 5

### 1. Instalasi dan Persiapan

```
pip install diffusers
```

Requirement already satisfied: diffusers in /usr/local/lib/python3.10/dist-packages (0.31.0)  
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.10/dist-packages (from diffusers) (8.5.0)  
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from diffusers) (3.16.1)  
Requirement already satisfied: huggingface-hub<0.23.2 in /usr/local/lib/python3.10/dist-packages (from diffusers) (0.27.0)  
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from diffusers) (1.26.4)  
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from diffusers) (2024.11.6)  
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from diffusers) (2.32.3)  
Requirement already satisfied: safetensors<0.3.1 in /usr/local/lib/python3.10/dist-packages (from diffusers) (0.4.5)  
Requirement already satisfied: Pillow in /usr/local/lib/python3.10/dist-packages (from diffusers) (11.0.0)  
Requirement already satisfied: fsspec<2023.5.0 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<0.23.2->diffusers) (2024.10.0)  
Requirement already satisfied: packaging<20.9 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<0.23.2->diffusers) (24.2)  
Requirement already satisfied: pyyaml<5.1 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<0.23.2->diffusers) (6.0.2)  
Requirement already satisfied: tqdm<4.42.1 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<0.23.2->diffusers) (4.67.1)  
Requirement already satisfied: typing-extensions<3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<0.23.2->diffusers) (4.12.2)  
Requirement already satisfied: zipp<3.20 in /usr/local/lib/python3.10/dist-packages (from importlib-metadata->diffusers) (3.21.0)  
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->diffusers) (3.4.0)  
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->diffusers) (3.10)  
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->diffusers) (2.2.3)  
Requirement already satisfied: certifi<2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->diffusers) (2024.12.14)

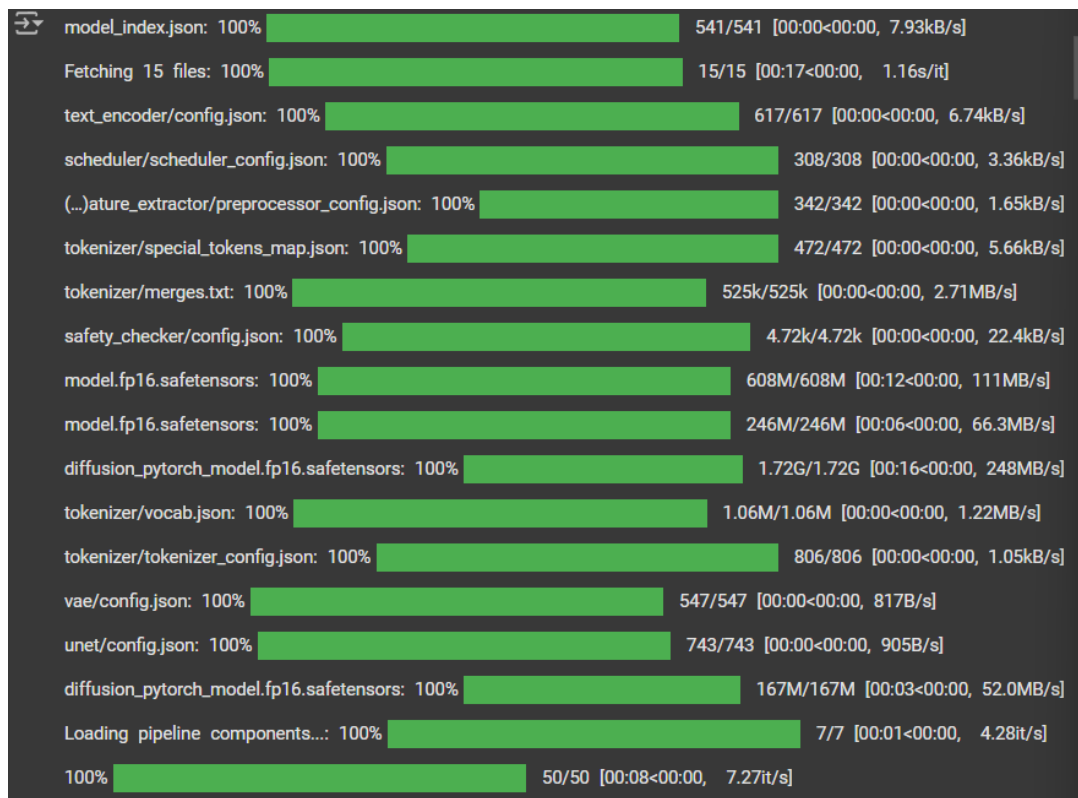
- **Penjelasan:** Perintah ini digunakan untuk menginstal pustaka diffusers, yang mendukung model difusi seperti Stable Diffusion untuk generasi gambar berbasis teks.

### 2. Image Generation with Stable Diffusion

```
from diffusers import AutoPipelineForText2Image
import torch

pipeline = AutoPipelineForText2Image.from_pretrained(
    "runwayml/stable-diffusion-v1-5", torch_dtype=torch.float16, variant="fp16"
).to("cuda")
generator = torch.Generator(device="cuda").manual_seed(31)
image = pipeline(
    "Astronaut in a jungle, cold color palette, muted colors, detailed, 8k",
    generator=generator,
).images[0]
```

- **Penjelasan:**
  - Memuat pipeline Stable Diffusion untuk generasi gambar berbasis teks.
  - Menggunakan GPU (cuda) untuk mempercepat proses.
  - Prompt digunakan untuk menggambarkan gambar yang ingin dihasilkan.



Gambar menunjukkan proses unduhan dan inisialisasi komponen untuk model Stable Diffusion. Semua file berhasil diunduh dan dimuat, termasuk konfigurasi, tokenizer, model utama (UNet, VAE), dan komponen pendukung seperti safety checker. Proses ini berjalan lancar tanpa error, dengan setiap tahap menunjukkan progres 100%.

### 3. Image Transformation and Manipulation

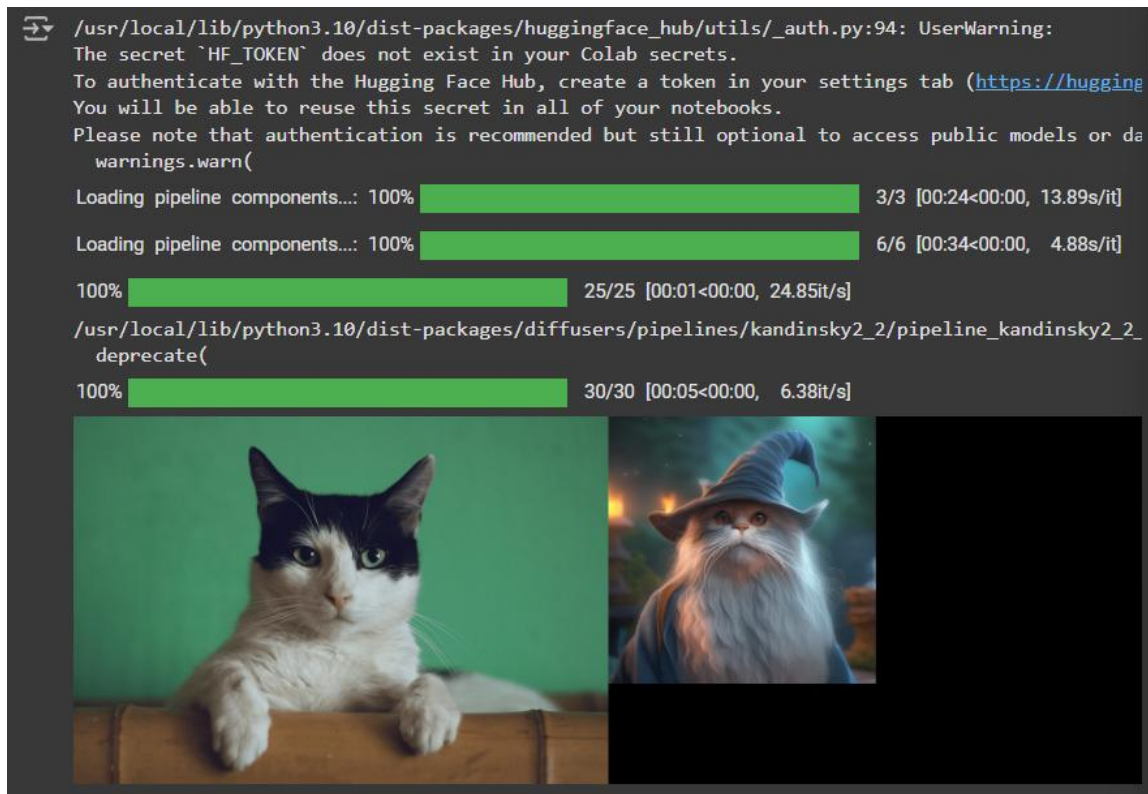
```
import torch
from diffusers import AutoPipelineForImage2Image
from diffusers.utils import load_image, make_image_grid

pipeline = AutoPipelineForImage2Image.from_pretrained(
    "kandinsky-community/kandinsky-2-2-decoder",
    torch_dtype=torch.float16,
    use_safetensors=True,
)

pipeline.enable_model_cpu_offload()
# remove following line if xFormers is not installed or you have PyTorch 2.0 or higher install
pipeline.enable_xformers_memory_efficient_attention()

# Load an image to pass to the pipeline:
init_image = load_image(
    "https://huggingface.co/datasets/huggingface/documentation-images/resolve/main/diffusers/c"
)

# Pass a prompt and image to the pipeline to generate an image:
prompt = "cat wizard, gandalf, lord of the rings, detailed, fantasy, cute, adorable, Pixar, Di
image = pipeline(prompt, image=init_image).images[0]
make_image_grid([init_image, image], rows=1, cols=2)
```



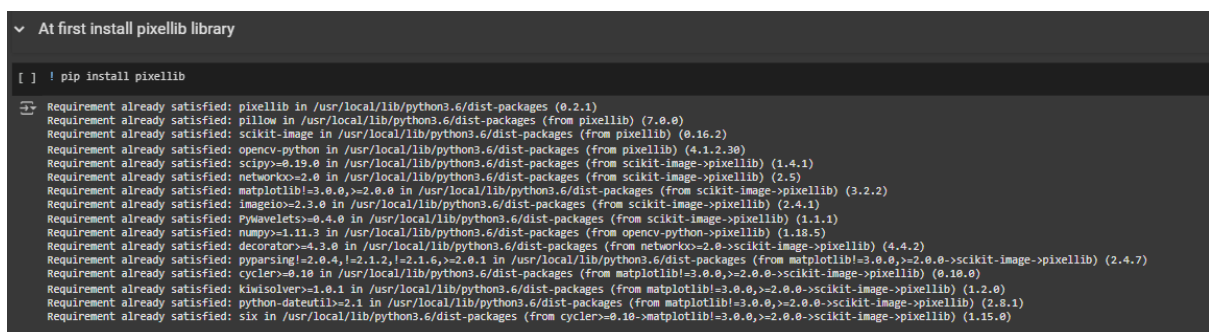
Gambar ini memperlihatkan log dari proses *pipeline* yang telah selesai dijalankan, serta hasil dari model *diffusion* yang berhasil menghasilkan gambar transformasi.

Penjelasan:

- Model Mask R-CNN digunakan untuk mendeteksi objek dalam gambar.
- predictions menyimpan informasi seperti mask (segmen), kotak pembatas, dan label objek.

#### 4. Object Detection and Image Segmentation using PixelLib library

- Instal pixrllib library



Ini bertujuan untuk menginstal pustaka *PixelLib*, yang digunakan untuk tugas segmentasi gambar, seperti segmentasi instance dan semantik.

#### Download Mask RCNN COCO Weights


```
[ ] !wget --quiet https://github.com/matterport/Mask_RCNN/releases/download/v2.0/mask_rcnn_coco.h5
```

- Perintah ini mengunduh weights Mask R-CNN untuk segmentasi instance berbasis dataset COCO.
- Setelah file diunduh, langkah selanjutnya biasanya adalah:
- Memuat model Mask R-CNN dengan bobot yang diunduh.
- Menggunakannya untuk segmentasi objek pada gambar atau video.

Proses pembacaan dan visualisasi gambar menggunakan pustaka OpenCV dan Matplotlib.

#### Show Input Image

```
[ ] import cv2
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (50,10)
image=cv2.imread("/content/drive/My Drive/images/sample.png")
plt.imshow(image)
```

 <matplotlib.image.AxesImage at 0x7fd96b2934a8>



Memuat pustaka PixelLib dan model Mask R-CNN yang dilatih pada dataset COCO.

#### ▼ Import Libraries and Use pixellib built in function to create object and load model

```
[ ] import pixellib
    from pixellib.instance import instance_segmentation
    segment_image = instance_segmentation()
    segment_image.load_model("mask_rcnn_coco.h5")
```

- Kode ini mengimpor pustaka PixelLib, menginisialisasi kelas untuk segmentasi instance, dan memuat weights Mask R-CNN.
- Model yang dimuat akan siap digunakan untuk tugas seperti mendeteksi dan memberi label pada objek dalam sebuah gambar.

Proses segmentasi instance dan deteksi objek pada sebuah gambar menggunakan PixelLib dengan model Mask R-CNN.

#### ▼ Image Segmentation and Object Detection

▼ Use segmentImage function for image segmentation, pass input image and output image name, and make show\_bboxes parameter True for bounding box object detection

```
[ ] segment_image.segmentImage("/content/drive/My Drive/images/sample.png", output_image_name = "/content/drive/My Drive/images/output_1.jpg", show_bboxes = True)
image=cv2.imread("/content/drive/My Drive/images/output_1.jpg")
plt.rcParams["figure.figsize"] = (50,10)
plt.imshow(image)
```

Processing image...  
Processed Image saved successfully in your current working directory.  
<matplotlib.image.AxesImage at 0x7fd969c03748>



## Hasil Visualisasi

### 1. Bounding Boxes:

- Setiap objek memiliki *bounding box* (kotak batas) di sekitar area yang terdeteksi.



- Kotak ini membantu memvisualisasikan lokasi dan dimensi objek dalam gambar.

## 2. Segmentasi Instance:

- Area dalam kotak batas diwarnai dengan warna berbeda untuk setiap kategori objek, menciptakan segmentasi visual yang jelas.

## 3. Label dan Confidence Score:

- Setiap objek diberi label sesuai dengan kategori yang terdeteksi, bersama dengan skor kepercayaan dari model.

## Kesimpulan

- Model berhasil mendeteksi dan melakukan segmentasi pada berbagai objek dalam gambar dengan akurasi tinggi.

## 5. inpainting

```
# Load the pipeline
import torch
from diffusers import AutoPipelineForInpainting
from diffusers.utils import load_image, make_image_grid

pipeline = AutoPipelineForInpainting.from_pretrained(
    "kandinsky-community/kandinsky-2-2-decoder-inpaint", torch_dtype=torch.float16
)
pipeline.enable_model_cpu_offload()
# remove following line if xformers is not installed or you have PyTorch 2.0 or higher installed
pipeline.enable_xformers_memory_efficient_attention()

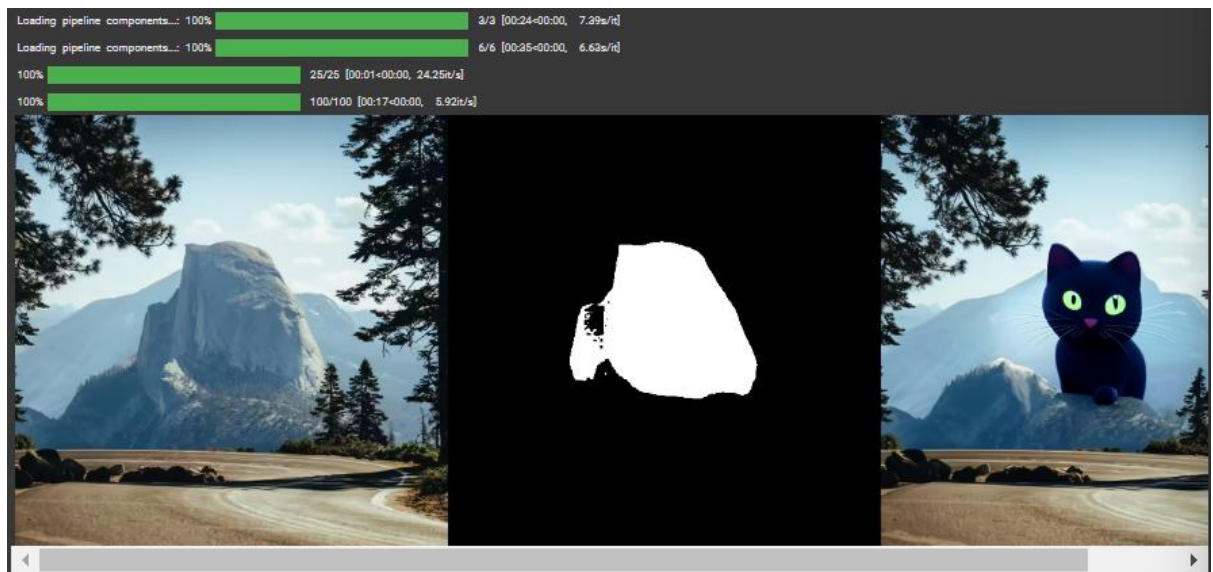
# Load the base and mask images:
init_image = load_image(
    "https://huggingface.co/datasets/huggingface/documentation-images/resolve/main/diffusers/inpaint.png"
)
mask_image = load_image(
    "https://huggingface.co/datasets/huggingface/documentation-images/resolve/main/diffusers/inpaint_mask.png"
)

# Create a prompt to inpaint the image with and pass it to the pipeline with the base and mask images:
prompt = (
    "a black cat with glowing eyes, cute, adorable, disney, pixar, highly detailed, 8k"
)
negative_prompt = "bad anatomy, deformed, ugly, disfigured"
image = pipeline(
    prompt=prompt,
    negative_prompt=negative_prompt,
    image=init_image,
    mask_image=mask_image,
).images[0]
make_image_grid([init_image, mask_image, image], rows=1, cols=3)
```

Kode ini bertujuan untuk melakukan *inpainting* menggunakan model "kandinsky-2-2-decoder-inpaint." Hasil akhirnya adalah mengisi bagian kosong (mask) dari sebuah gambar dengan objek baru berdasarkan *prompt*.

- AutoPipelineForInpainting digunakan untuk tugas *inpainting* (melengkapi bagian gambar yang hilang).
- Pipeline dimuat dari model *pre-trained* "kandinsky-2-2-decoder-inpaint."

- Model diatur untuk menggunakan tipe data float16 dan memanfaatkan fitur efisiensi memori seperti *xformers*.
- Gambar asli (*init\_image*) adalah lanskap pegunungan.
- Gambar *mask* (*mask\_image*) digunakan untuk menentukan area yang perlu diubah (bagian putih pada mask).
- *Prompt* mendeskripsikan gambar yang diinginkan (kucing hitam dengan mata bercahaya dalam gaya Pixar/Disney).
- *Negative prompt* digunakan untuk menghindari hasil yang tidak diinginkan, seperti anatomi buruk atau gambar jelek.
- Pipeline mengambil gambar asli dan mask, lalu menghasilkan gambar baru dengan objek sesuai deskripsi *prompt*.
- Gambar-gambar disusun dalam grid horizontal: gambar asli, gambar *mask*, dan gambar hasil *inpainting*.



Hasil:

1. Gambar Asli (Kiri):

- Lanskap pegunungan dengan jalan berliku di latar depan. Tidak ada perubahan pada bagian ini karena hanya digunakan sebagai referensi.

2. Gambar Mask (Tengah):

- Area putih pada mask menunjukkan bagian gambar yang akan diganti dengan hasil baru berdasarkan *prompt*.

3. Gambar Output (Kanan):

- Hasil *inpainting* berhasil mengisi bagian kosong dengan kucing hitam bergaya Pixar, memiliki mata bercahaya hijau. Objek terlihat sesuai deskripsi *prompt* (detail tinggi, gaya Pixar/Disney) dan terintegrasi dengan baik ke dalam latar belakang.

## 5. Basics of Linear Algebra for 3D Data

- Plot kubus 3d

```
import numpy as np
import matplotlib.pyplot as plt

def plot_cube(ax, cube, label, color="black"):
    ax.scatter3D(cube[0, :], cube[1, :], cube[2, :], label=label, color=color)
    lines = [
        [0, 1],
        [1, 2],
        [2, 3],
        [3, 0],
        [4, 5],
        [5, 6],
        [6, 7],
        [7, 4],
        [0, 4],
        [1, 5],
        [2, 6],
        [3, 7],
    ]
    for line in lines:
        ax.plot3D(cube[0, line], cube[1, line], cube[2, line], color=color)
    ax.set_xlabel("X")
    ax.set_ylabel("Y")
    ax.set_zlabel("Z")
    ax.legend()
    ax.set_xlim([-2, 2])
    ax.set_ylim([-2, 2])
    ax.set_zlim([-2, 2])
```

- Definisi Kubus:

- Kubus didefinisikan menggunakan koordinat *vertices* (simpul) dan garis (*edges*) yang menghubungkan simpul tersebut.
- Fungsi `plot_cube` digunakan untuk memvisualisasikan kubus dalam ruang 3D menggunakan `matplotlib`.

- Pewarnaan:

- Kubus diberi label dan warna, sehingga setiap bagian kubus dapat diidentifikasi secara unik pada plot.



- **Translasi Kubus dalam Ruang 3D**

```
# define 8 corners of our cube with coordinates (x,y,z,w) and w is always 1 in our case
cube = np.array(
    [
        [-1, -1, -1, 1],
        [1, -1, -1, 1],
        [1, 1, -1, 1],
        [-1, 1, -1, 1],
        [-1, -1, 1, 1],
        [1, -1, 1, 1],
        [1, 1, 1, 1],
        [-1, 1, 1, 1],
    ]
)

# translate to follow OpenGL notation
cube = cube.T

# set up figure
fig = plt.figure()
ax = fig.add_subplot(111, projection="3d")

# plot original cube
plot_cube(ax, cube, label="Original", color="blue")

# translation matrix (shift 1 in positive x and 1 in positive y-axis)
translation_matrix = np.array([[1, 0, 0, 1], [0, 1, 0, 1], [0, 0, 1, 0], [0, 0, 0, 1]])

# translation
translated_cube = translation_matrix @ cube
plot_cube(ax, translated_cube, label="Translated", color="red")
```

- **Koordinat Kubus:**

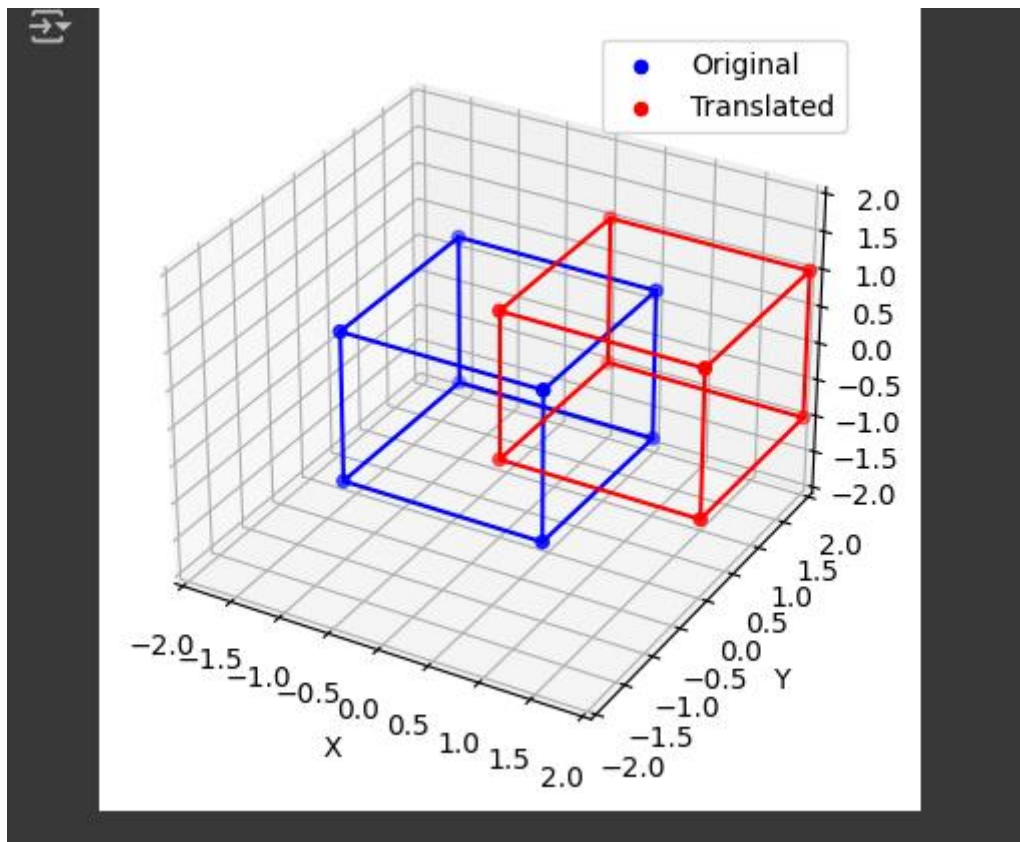
- Kubus didefinisikan dengan koordinat awal, yang disusun dalam array numpy.
- Kubus divisualisasikan dalam ruang koordinat 3D menggunakan matplotlib.

- **Transformasi Translasi:**

- Matriks translasi digunakan untuk memindahkan kubus ke posisi baru dalam ruang.
- Matriks translasi didefinisikan sebagai array numpy yang menambah nilai tertentu pada koordinat x, y, dan z.

- **Hasil:**

- Kubus awal dan kubus yang sudah ditranslasikan divisualisasikan bersama untuk menunjukkan perubahan posisi.



Gambar ini menunjukkan hasil visualisasi kubus dalam ruang 3D, dengan dua kubus yang dibandingkan: kubus asli (berwarna biru) dan kubus yang telah ditranslasikan (berwarna merah)

Kesimpulan:

- Visualisasi ini menunjukkan bagaimana transformasi geometris (translasi) memengaruhi posisi objek dalam ruang 3D tanpa mengubah bentuk atau ukurannya.

- Penerapan transformasi skala (scaling) pada kubus 3D

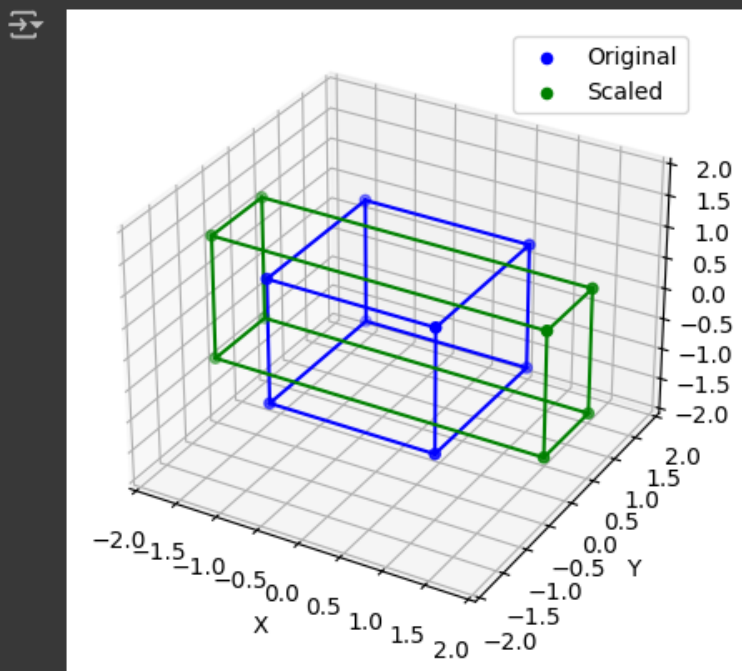
```
# set up figure
fig = plt.figure()
ax = fig.add_subplot(111, projection="3d")

# plot original cube
plot_cube(ax, cube, label="Original", color="blue")

# scaling matrix (scale by 2 along x-axis and by 0.5 along y-axis)
scaling_matrix = np.array([[2, 0, 0, 0], [0, 0.5, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]])

scaled_cube = scaling_matrix @ cube

plot_cube(ax, scaled_cube, label="Scaled", color="green")
```



### 1. Kubus Asli (Biru):

- Kubus biru menunjukkan bentuk dan ukuran asli sebelum transformasi.
- Kubus ini diplot dengan label "Original."

### 2. Matriks Skala:

- Sumbu X diperbesar dengan faktor 2.
- Sumbu Y diperkecil dengan faktor 0.5.
- Sumbu Z tetap tidak berubah (faktor skala 1).

### 3. Kubus yang Diskalikan (Hijau):

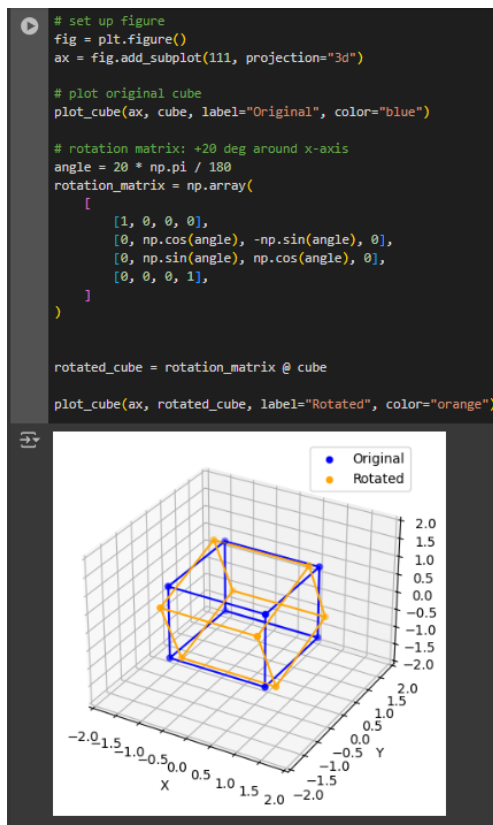
- Kubus hijau menunjukkan hasil dari transformasi skala.
- Kubus ini diperbesar di sumbu X dan diperkecil di sumbu Y, seperti yang didefinisikan oleh matriks skala.

### 4. Hasil Akhir:

- Kubus asli dan hasil transformasi ditampilkan dalam ruang koordinat 3D.
- Warna biru mewakili kubus asli, sedangkan warna hijau mewakili kubus yang telah diskalikan.
- Legenda menunjukkan perbedaan kedua kubus dengan label "Original" dan "Scaled."

### Kesimpulan:

- Transformasi skala berhasil mengubah dimensi kubus di sumbu X dan Y, sedangkan sumbu Z tetap sama.
- Penerapan transformasi rotasi pada kubus dalam ruang 3D.



### 1. Kubus Asli (Biru):

- Kubus biru mewakili posisi dan orientasi awal sebelum transformasi.
- Kubus ini diplot dengan label "Original" sebagai referensi.

### 2. Matriks Rotasi:

- Matriks rotasi didefinisikan untuk rotasi sebesar 20 derajat di sekitar sumbu X.
- Matriks yang digunakan:
  - angle dihitung dalam radian dengan konversi dari derajat:  $\text{angle} = 20 * \text{np.pi} / 180$ .
  - Sumbu X tetap, sedangkan koordinat pada sumbu Y dan Z berubah sesuai dengan fungsi trigonometri.

### 3. Kubus yang Dirotasi (Oranye):

- Kubus oranye adalah hasil dari rotasi kubus biru di sekitar sumbu X.
- Rotasi mengubah posisi simpul (vertices) pada sumbu Y dan Z, tetapi tidak memengaruhi sumbu X.

### 4. Hasil Visualisasi:

- Kubus asli dan kubus yang telah dirotasi diplot dalam ruang koordinat 3D.
- Warna biru menunjukkan kubus asli, sedangkan warna oranye menunjukkan kubus yang telah dirotasi.
- Legenda menjelaskan perbedaan kedua kubus dengan label "Original" dan "Rotated."

### Kesimpulan:

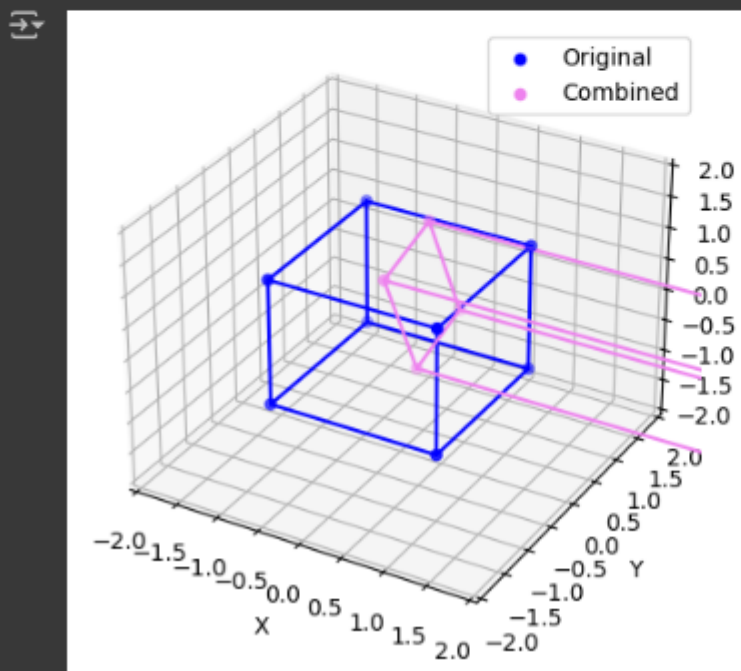
- Transformasi rotasi berhasil mengubah orientasi kubus dengan rotasi 20 derajat di sekitar sumbu X.

- Penerapan kombinasi transformasi (rotasi, skala, dan translasi) pada kubus dalam ruang 3D

```
# set up figure
fig = plt.figure()
ax = fig.add_subplot(111, projection="3d")

# plot original cube
plot_cube(ax, cube, label="Original", color="blue")

# combination of transforms
combination_transform = rotation_matrix.dot(scaling_matrix.dot(translation_matrix))
final_result = combination_transform.dot(cube)
plot_cube(ax, final_result, label="Combined", color="violet")
```



### 1. Kubus Asli (Biru):

- Kubus biru adalah bentuk awal sebelum transformasi apa pun diterapkan.
- Digunakan sebagai referensi untuk membandingkan hasil transformasi.

### 2. Kombinasi Transformasi:

- Kombinasi transformasi melibatkan:
  - Translasi: Memindahkan kubus ke posisi baru.
  - Skala: Mengubah ukuran kubus pada sumbu tertentu.



- **Rotasi: Memutar kubus di sekitar sumbu tertentu.**

### **3. Kubus Hasil Transformasi (Ungu):**

- **Kubus ungu adalah hasil dari kombinasi transformasi.**
- **Bentuknya telah berubah karena skala, posisinya bergeser karena translasi, dan orientasinya berubah karena rotasi.**
- **Transformasi ini memberikan bentuk dan posisi baru yang kompleks dibandingkan kubus asli.**

### **4. Hasil Visualisasi:**

- **Kubus asli (biru) dan hasil transformasi (ungu) diplot dalam ruang 3D.**
- **Perbedaan antara posisi dan orientasi kedua kubus terlihat jelas.**
- **Legenda membantu membedakan antara "Original" dan "Combined."**

## **Kesimpulan**

- **Kombinasi transformasi memungkinkan manipulasi posisi, orientasi, dan ukuran objek secara simultan dalam ruang 3D.**

- Implementasi *positional encoding* menggunakan PyTorch dan menampilkan hasilnya dalam bentuk grid visual

```
import torch
import matplotlib as mda
import numpy as np

def positional_encoding(in_tensor, num_frequencies, min_freq_exp, max_freq_exp):
    """Function for positional encoding"""
    # Scale input tensor to [0, 2 * pi]
    scaled_in_tensor = 2 * np.pi * in_tensor
    # Generate frequency spectrum
    freqs = 2 ** torch.linspace(
        min_freq_exp, max_freq_exp, num_frequencies, device=in_tensor.device
    )
    # Generate encodings
    scaled_inputs = scaled_in_tensor.unsqueeze(-1) * freqs
    encoded_inputs = torch.cat(
        [torch.sin(scaled_inputs), torch.cos(scaled_inputs)], dim=-1
    )
    return encoded_inputs.view(*in_tensor.shape[:-1], -1)

def visualize_grid(grid, encoded_images, resolution):
    """Helper function to visualize grid"""
    # Split the grid into separate channels for x and y
    x_channel, y_channel = grid[..., 0], grid[..., 1]
    # Show the original grid
    print("Input Values:")
    mda.show_images([x_channel, y_channel], cmap="plasma", border=True)
    # Show the encoded grid
    print("Encoded Values:")
    max_channels_to_visualize = min(
        8, encoded_images.shape[-1]
    )
    # Visualize up to 8 channels
    encoded_images_to_show = encoded_images.view(resolution, resolution, -1).permute(
        2, 0, 1
    )[:max_channels_to_visualize]
    mda.show_images(encoded_images_to_show, vmin=-1, vmax=1, cmap="plasma", border=True)

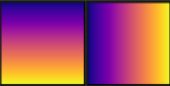
# Parameters similar to your NaFEncoding example
num_frequencies = 4
min_freq_exp = 0
max_freq_exp = 6
resolution = 128

# Generate a 2D grid of points in the range [0, 1]
x_samples = torch.linspace(0, 1, resolution)
y_samples = torch.linspace(0, 1, resolution)
grid = torch.stack(
    [torch.meshgrid(x_samples, y_samples), dim=-1]
) # (resolution, resolution, 2)

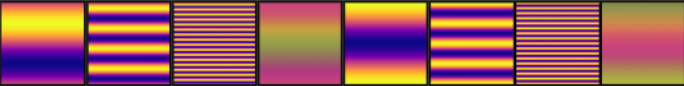
# Apply positional encoding
encoded_grid = positional_encoding(grid, num_frequencies, min_freq_exp, max_freq_exp)

# Visualize result
visualize_grid(grid, encoded_grid, resolution)
```

Input Values:



Encoded Values:



## 1. Import Library:

- torch*, *matplotlib*, dan *numpy* diimpor untuk memproses data tensor, visualisasi, dan manipulasi array numerik.

## 2. Fungsi positional\_encoding:

- Tujuan: Menghasilkan encoding posisi berdasarkan frekuensi sin dan cos.
- Parameter:
  - num\_frequencies:** Jumlah frekuensi yang digunakan untuk encoding.
  - min\_freq\_exp, max\_freq\_exp:** Rentang eksponen untuk frekuensi minimum dan maksimum.
  - input\_tensor:** Grid posisi yang akan di-encode.

- **Proses:**
  - Frekuensi dikalkulasi secara logaritmik dari `min_freq_exp` hingga `max_freq_exp`.
  - Tensor input diubah menjadi nilai sin dan cos dengan berbagai frekuensi, menghasilkan representasi posisi dengan dimensi tinggi.
- **Output:**
  - Tensor dengan dimensi tambahan yang berisi encoding posisi.

### 3. Fungsi `visualize_grid`:

- **Tujuan:** Membuat visualisasi grid dari hasil positional encoding.
- **Proses:**
  - Setiap channel pada hasil encoding divisualisasikan menggunakan *matplotlib* dalam format warna.
  - Warna mencerminkan nilai dari encoding (plasma colormap digunakan untuk meningkatkan visibilitas).

### 4. Definisi Grid dan Parameter:

- Grid 2D dibentuk dengan koordinat X dan Y dalam rentang `[0, 1]` menggunakan `torch.meshgrid`.
- Parameter encoding:
  - `num_frequencies = 6`: 6 frekuensi digunakan.
  - `min_freq_exp = 0`, `max_freq_exp = 8`: Eksponen frekuensi berkisar dari 0 hingga 8.
- Tensor grid diubah ke tensor encoding menggunakan fungsi `positional_encoding`.

### 5. Visualisasi:

- Gambar pertama menunjukkan grid asli dalam format koordinat X dan Y.
- Gambar-gambar berikutnya menunjukkan hasil encoding untuk setiap channel yang dihasilkan dari kombinasi sin dan cos pada berbagai frekuensi.

## Hasil Visualisasi

### 1. Grid Asli:

- Menampilkan pola warna berdasarkan posisi X dan Y dalam grid asli.
- Warna membantu memvisualisasikan perbedaan posisi dalam ruang 2D.

### 2. Encoded Values:

- Setiap kotak mewakili channel tertentu dari hasil encoding.
- Pola garis-garis (horizontal, vertikal, diagonal) muncul karena nilai-nilai sin dan cos dengan frekuensi berbeda.
- Encoding ini meningkatkan representasi posisi untuk model neural network.

## Kesimpulan

- Fungsi `positional_encoding` berhasil mengubah grid posisi menjadi representasi dengan dimensi tinggi menggunakan transformasi sin dan cos pada frekuensi logaritmik.