

Nama : Hero Kartiko

NIM : 1103210205

## UNIT 5

### Generative Models

#### 5.1 Introduction

Model matematika dibagi menjadi dua, yaitu model generatif dan model diskriminatif. Perbedaanannya adalah model diskriminatif mempelajari batas – batas yang memisahkan berbagai kelas sedangkan model generatif mempelajari distribusi dari berbagai kelas.

Model diskriminatif dapat diterapkan pada tugas *computer vision* standar seperti klasifikasi dan regresi. Tugas ini dapat diperluas menjadi proses yang lebih kompleks seperti segmentasi semantik atau deteksi objek.

Evaluasi model generatif sulit dilakukan karena sering tidak ada *ground truth* yang kelas, dan kualitas gambar sulit diukur secara kuantitatif. **ID (Fréchet Inception Distance)** adalah metrik yang paling umum digunakan meskipun tidak sempurna. FID mengukur jarak antara dua distribusi fitur (dari Inception-v3) yang dihitung dari data pelatihan dan gambar yang dihasilkan. Semakin rendah FID, semakin baik kualitas gambar yang dihasilkan. FID dianggap tahan terhadap noise dan artefak tertentu. Adapun metrik lain yang sering digunakan meliputi :

- **PSNR (Peak Signal-to-Noise Ratio):** Mirip dengan mean squared error. Nilai [25–34] dianggap cukup baik, sementara >34 sangat baik.
- **SSIM (Structural Similarity Index):** Metrik dalam rentang [0,1], dengan 1 menunjukkan kecocokan sempurna, dihitung dari luminansi, kontras, dan struktur.
- **Inception Score (IS):** Diperkenalkan dalam *Improved Techniques for Training GANs*. Semakin tinggi nilainya, semakin baik. Namun, metrik ini kini jarang digunakan.
- **CLIP Score :** Digunakan untuk model *text-to-image* dengan menghitung kesamaan kosinus antara gambar yang dihasilkan dan *text prompt*. Rentang nilainya [0–100], semakin tinggi semakin baik.

#### 5.2 Variational Autoencoders

Variational Autoencoders (VAEs) adalah jenis *neural network* yang digunakan untuk *unsupervised learning* dan pengurangan dimensi. Mereka mengatasi keterbatasan autoencoder tradisional dengan memperkenalkan pendekatan probabilistik dalam proses encoding dan decoding.

- **Encoder** : Bertugas mentransformasikan data masukan menjadi representasi terkompresi atau *latent space*. Biasanya terdiri dari satu atau lebih lapisan neuron secara progresif mengurangi dimensi data masukan. Tujuan utamanya adalah mengekstrak fitur penting data masukan
- **Decoder** : Mengambil representasi terkompresi yang dihasilkan oleh encoder dan mencoba merekonstruksi kembali data masukan asli. Sama seperti encoder, decoder biasanya terdiri dari satu atau lebih lapisan neuron tetapi dalam ukuran yang berlawanan. Tujuannya memastikan rekonstruksi data sedekat mungkin dengan data asli. Kegunaannya adalah : Reduksi dimensi, Deteksi Anomali.

Fungsi loss terdiri dari dua komponen, yaitu : Reconstruction loss dan KL Divergence. Dengan memasukkan KL Divergence ke dalam fungsi loss, VAEs didorong untuk mempelajari *latent space* dimana titik yang serupa berada lebih dekat, memastikan representasi yang bermakna dan terstruktur. *Latent loss* yang lebih kecil menghasilkan gambar yang menyerupai data pelatihan tetapi kurang bervariasi. *Reconstruction loss* yang lebih kecil menghasilkan rekonstruksi yang baik selama pelatihan tetapi menghambat generasi gambar baru.

### 5.3 Generative Adversarial Networks

GANs (Generative Adversarial Networks) adalah model *deep learning* yang diperkenalkan oleh Ian Goodfellow dan rekan – rekannya pada tahun 2014. GANs terdiri dari dua komponen utama : generator dan discriminator. Generator menghasilkan data sintesis yang menyerupai data nyata, sementara discriminator membedakan data nyata dan data yang dihasilkan. Kedua jaringan ini dilatih secara bersamaan dalam proses adversarial.

Perbandingan GANs dan VAEs :

- **GANs** : Mampu menghasilkan gambar berkualitas tinggi dengan detail tajam dan tekstur realistis. Namun pelatihan GANs bisa sulit dan rentan terhadap ketidakstabilan.

- **VAEs** : Lebih mudah dilatih dan lebih stabil dibandingkan GANs, tetapi cenderung menghasilkan gambar yang kurang tajam dan detail. VAEs sering digunakan untuk denoising gambar dan deteksi anomali.

Pelatihan GANs melibatkan proses adversarial dimana generator dan discriminator bermain dalam permainan minimax. Generator berusaha meminimalkan kemampuan discriminator dalam membedakan data palsu, sementara discriminator berusaha memaksimalkan kemampuan dalam mendeteksi data palsu. Proses akan terus berlanjut hingga generator mampu menghasilkan data yang hampir tidak dapat dibedakan dari data nyata.

#### 5.4 StyleGAN Variants

StyleGAN adalah arsitektur jaringan adversarial generatif (GAN) yang dikembangkan untuk menghasilkan gambar realistis dengan kontrol yang lebih besar terhadap fitur gambar yang dihasilkan. Tidak seperti GAN konvensional, StyleGAN memperkenalkan beberapa komponen kunci yang memungkinkan manipulasi atribut secara lebih terstruktur.

Komponen utama StyleGAN :

- **Mapping Network** : StyleGAN memetakan kode laten ini ke ruang gaya menggunakan jaringan pemetaan. Hal ini membantu dalam mendekorrelasi fitur dan memungkinkan kontrol yang lebih baik terhadap atribut gambar.
- **Adaptive Instance Normalization** : StyleGAN menggunakan mekanisme AdaIN untuk menerapkan gaya pada berbagai lapisan jaringan. Dengan menyesuaikan parameter normalisasi pada setiap lapisan, model dapat mengontrol atribut gambar pada berbagai skala, dari struktur global hingga detail lokal.
- **Penambahan Vektor Noise** : Selain kode gaya, vektor noise ditambahkan pada berbagai tahap proses generasi untuk memperkenalkan variasi stokastik, seperti tekstur halus dan detail kecil.

Evolusi StyleGAN :

- **StyleGAN2** : Memperbaiki artefak yang ada pada versi sebelumnya dan meningkatkan kualitas gambar dengan memperkenalkan arsitektur yang lebih efisien dan teknik normalisasi yang ditingkatkan.

- **StyleGAN3** : Mengatasi masalah aliasing dan meningkatkan keselarasan spasial dalam gambar yang dihasilkan, memungkinkan manipulasi yang lebih halus dan konsisten.

Aplikasi StyleGAN :

- Pengeditan gambar : digunakan untuk imprinting dan transfer gaya antar gambar
- Pelestarian privasi : menghasilkan data sintesis yang realistis untuk menggunakan informasi sensitif dalam pelatihan dan pengujian model.
- Eksplorasi kreatif : membantu pembuatan desain mode, lingkungan virtual realistis, dan aplikasi kreatif lainnya.

## 5.5 CycleGAN Introduction

CycleGAN adalah kerangka kerja yang dirancang untuk tugas translasi gambar ke gambar dimana pasangan yang tidak cocok tidak tersedia. Diperkenalkan oleh Zhu et al. pada 2017, CycleGAN memungkinkan translasi antar dua domain gambar tanpa memerlukan data berpasangan.

Dalam banyak skenario, kita memiliki dua set gambar yang tidak berpasangan, misalnya foto dan lukisan Monet, atau gambar kuda dan zebra. Tujuan CycleGAN adalah mempelajari pemetaan antara dua domain ini sehingga gambar dari satu domain dapat ditransformasikan ke gaya domain lain, dan sebaliknya, tanpa memerlukan pasangan gambar yang cocok.

Komponen Utama CycleGAN

- **Struktur Dual GAN** : CycleGAN menggunakan dua jaringan GAN; satu untuk mentranslasikan dari domain X ke Y (misalnya, dari zebra ke kuda) dan satu lagi untuk translasi sebaliknya (dari kuda ke zebra). Struktur ini memastikan realisme melalui proses adversarial dan menjaga konten melalui konsistensi siklus.
- **Discriminator PatchGAN** : Discriminator dalam CycleGAN menggunakan arsitektur PatchGAN, yang mengevaluasi patch kecil dari gambar daripada keseluruhan gambar, membantu dalam memastikan realisme pada level lokal.
- **Arsitektur Generator** : Generator dalam CycleGAN mengadopsi elemen dari arsitektur U-Net dan DCGAN, termasuk proses downsampling (encoding), upsampling (decoding), dan lapisan konvolusi dengan normalisasi batch dan aktivasi ReLU. Penambahan koneksi residual membantu dalam pembelajaran transformasi yang lebih dalam.

- **Loss Konsistensi Siklus** : CycleGAN memperkenalkan loss konsistensi siklus yang memastikan bahwa jika sebuah gambar ditranslasikan ke domain lain dan kemudian kembali ke domain asli, gambar tersebut harus serupa dengan aslinya. Ini membantu dalam menjaga konten gambar selama proses translasi.

Selain itu CycleGAN telah digunakan dalam berbagai aplikasi, termasuk:

- **Translasi Gaya Seni**: Mengubah foto menjadi lukisan dalam gaya seniman tertentu tanpa memerlukan pasangan data yang cocok.
- **Perubahan Musim**: Mengubah gambar pemandangan dari musim panas ke musim dingin dan sebaliknya.
- **Translasi Spesies Hewan**: Mengubah gambar kuda menjadi zebra atau sebaliknya.

Dengan kemampuan untuk bekerja tanpa data berpasangan, CycleGAN membuka peluang baru dalam translasi gambar-ke-gambar di berbagai domain di mana pengumpulan data berpasangan sulit atau tidak mungkin dilakukan.

## 5. 6 Introduction to Diffusion Models

Model difusi adalah jenis model generatif yang telah menunjukkan hasil memuaskan dalam pembuatan gambar. Model ini bekerja melalui dua tahap utama: tahap difusi maju dan tahap difusi mundur. Pada tahap difusi maju, model secara bertahap menambahkan noise Gaussian ke data pelatihan, mengubahnya menjadi gambar yang sangat bising. Kemudian, pada tahap difusi mundur, model belajar menghilangkan noise tersebut secara bertahap untuk merekonstruksi data asli. Proses ini memungkinkan model untuk menghasilkan sampel data baru yang realistis dengan menghapus noise dari input acak.

Salah satu perbedaan utama antara model difusi dan Generative Adversarial Networks (GANs) adalah stabilitas dalam pelatihan. GANs sering kali sulit dilatih karena tujuan adversarialnya dan rentan terhadap masalah seperti mode collapse, di mana model hanya menghasilkan variasi terbatas dari data. Sebaliknya, model difusi memiliki proses pelatihan yang lebih stabil dan cenderung menghasilkan variasi sampel yang lebih beragam karena pendekatan berbasis likelihood. Namun, model difusi memerlukan komputasi yang lebih intensif dan waktu inferensi yang lebih lama dibandingkan dengan GANs karena proses mundur yang bertahap.

Terdapat tiga kerangka kerja utama dalam pemodelan difusi :

- **Denoising Diffusion Probabilistic Models (DDPMs):** Model ini menggunakan variabel laten untuk memperkirakan distribusi probabilitas dan dapat dianggap sebagai jenis variational autoencoders (VAEs).
- **Noise Conditioned Score Networks (NCSNs):** Juga dikenal sebagai Score Matching with Langevin Dynamics (SMLD), model ini mempelajari skor data (gradien log-density) dan menggunakan dinamika Langevin untuk menghasilkan sampel.
- **Stochastic Differential Equations (SDEs):** Pendekatan ini memodelkan proses difusi sebagai persamaan diferensial stokastik, memungkinkan fleksibilitas dalam desain proses difusi dan sampling.

Model difusi memiliki beberapa aplikasi gambar, termasuk pembuatan gambar, pemulihan gambar, dan transfer gaya. Adapun kekurangan model difusi adalah : Kompleksitas Komputasi, waktu inferensi lama, desain proses difusi.

## 5. 7 Introduction to Stable Diffusion

Stable Diffusion adalah model generatif AI yang menghasilkan gambar fotorealistik berdasarkan teks atau gambar input. Diluncurkan pada tahun 2022, model ini dikembangkan melalui kolaborasi antara Stability AI, RunwayML, dan CompVis Group di LMU Munich. Komponen utama pada Stable Diffusion : Proses Difusi Maju dan Mundur, Model Difusi Laten dengan Variational Auto – Encoder (VAE), Penggabungan Teks dan Gambar melalui Penyekoran Teks.

```
[ ] 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]
```

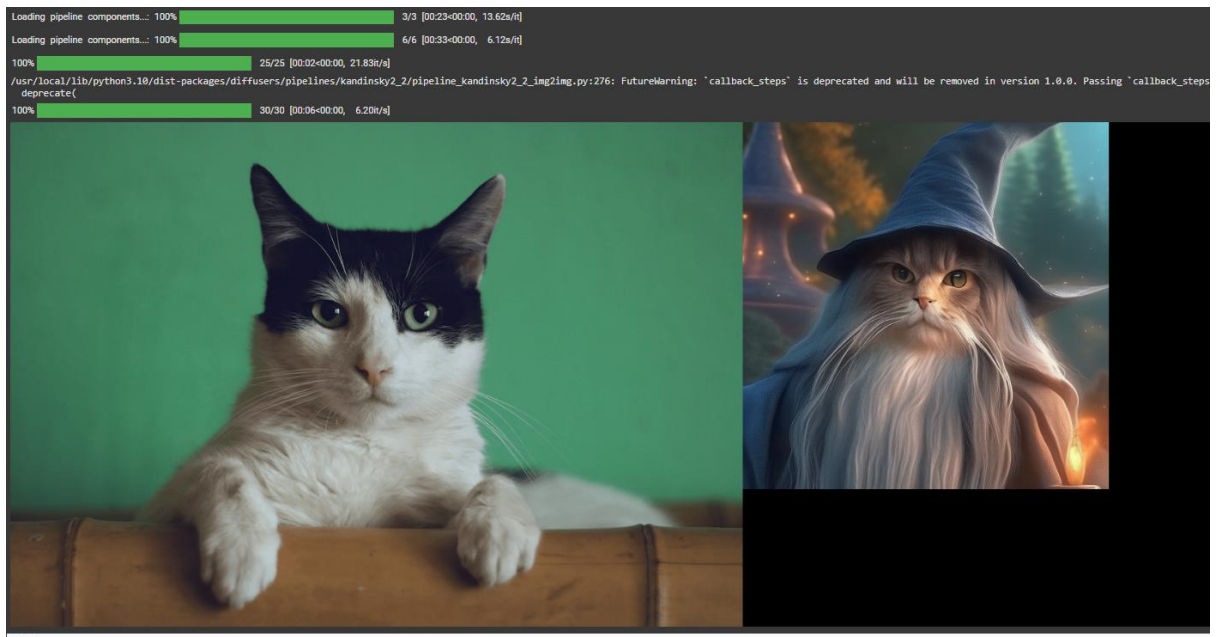
```
model_index.json: 100% ██████████ 541/541 [00:00<00:00, 28.1kB/s]
Fetching 15 files: 100% ██████████ 15/15 [00:43<00:00, 3.22s/it]
model.fp16.safetensors: 100% ██████████ 608M/608M [00:19<00:00, 49.4MB/s]
safety_checker/config.json: 100% ██████████ 4.72k/4.72k [00:00<00:00, 171kB/s]
model.fp16.safetensors: 100% ██████████ 246M/246M [00:07<00:00, 8.35MB/s]
text_encoder/config.json: 100% ██████████ 617/617 [00:00<00:00, 10.2kB/s]
(._ature_extractor/preprocessor_config.json: 100% ██████████ 342/342 [00:00<00:00, 8.04kB/s]
tokenizer/merges.txt: 100% ██████████ 525k/525k [00:00<00:00, 1.05MB/s]
scheduler/scheduler_config.json: 100% ██████████ 308/308 [00:00<00:00, 9.04kB/s]
tokenizer/special_tokens_map.json: 100% ██████████ 472/472 [00:00<00:00, 11.7kB/s]
tokenizer/tokenizer_config.json: 100% ██████████ 806/806 [00:00<00:00, 17.9kB/s]
diffusion_pytorch_model.fp16.safetensors: 100% ██████████ 1.72G/1.72G [00:42<00:00, 87.8MB/s]
tokenizer/vocab.json: 100% ██████████ 1.06M/1.06M [00:00<00:00, 1.56MB/s]
UNET/config.json: 100% ██████████ 743/743 [00:00<00:00, 13.7kB/s]
diffusion_pytorch_model.fp16.safetensors: 100% ██████████ 167M/167M [00:07<00:00, 18.8MB/s]
vae/config.json: 100% ██████████ 547/547 [00:00<00:00, 5.72kB/s]
Loading pipeline components....: 100% ██████████ 7/7 [00:02<00:00, 2.47it/s]
100% ██████████ 50/50 [00:07<00:00, 7.01it/s]
```

```
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 installed
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/cat.png"
)

# 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, Disney, 8k"
image = pipeline(prompt, image=init_image).images[0]
make_image_grid([init_image, image], rows=1, cols=2)
```



```
# 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)
```



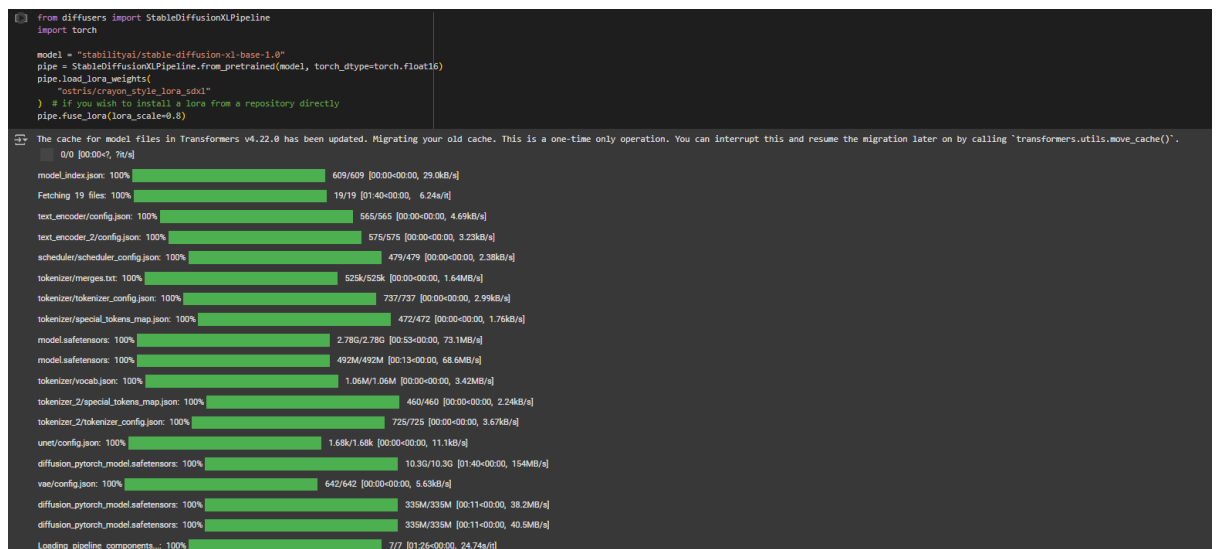


## 5. 8 Control over Diffusion Models

Model difusi adalah jenis model generatif yang bekerja dengan menambahkan noise secara bertahap ke data training dan kemudian mempelajari cara menghilangkan noise tersebut untuk menghasilkan data baru yang realistis. Proses ini terdiri dari dua tahap utama :

- Proses Maju : Noise Gaussian ditambahkan secara bertahap ke data pelatihan hingga data tersebut menjadi noise acak sepenuhnya.
- Proses Mundur : Model dilatih untuk menghilangkan noise secara bertahap , membalikkan proses maju sehingga dapat menghasilkan data baru dari noise acak.

Keunggulan model difusi dibanding GANs meliputi stabilitas dalam pelatihan dan kemampuan menghasilkan data dengan keragaman yang lebih tinggi. Namun, model difusi memerlukan waktu inferensi lebih lama karena proses penghilangan yang bertahap. Model difusi telah digunakan dalam banyak aplikasi seperti : Peningkatan resolusi gambar, pemulihan gambar, edit gambar.



## 5. 9 Privacy, Bias, and Societal Concerns

Penggunaan luas alat pengeditan yang memanfaatkan AI menimbulkan banyak keresahan terkait dengan privasi, bias, dan dampak sosial. Kemampuan alat ini untuk memanipulasi gambar 2D dan 3D dengan sangat realistis membuat kekhawatiran terkait etis. Adapun dampak di masyarakat : Menurunnya kepercayaan terhadap media, pelecehan dan pencemaran nama baik, standar kecantikan yang tidak realistis.

## UNIT 6

### BASIC CV TASKS

#### 6.1 Object Detection

Deteksi objek adalah tugas dalam *computer vision* bertujuan untuk mengidentifikasi dan menentukan lokasi objek tertentu gambar atau video. Tugas ini menggabungkan klasifikasi dan lokalisasi. Deteksi objek memiliki berbagai aplikasi termasuk sistem pengawasan, diagnosis medis, manufaktur, dll.

Untuk menilai kinerja deteksi model terdapat beberapa metrik yang digunakan sebagai acuan :

- **Intersection over Union (IoU):** Mengukur tingkat tumpang tindih antara kotak prediksi dan kotak referensi (ground truth) sebagai persentase. Semakin tinggi persentasenya, semakin baik akurasi model dalam menentukan lokasi objek.
- **Mean Average Precision (mAP):** Menilai efisiensi deteksi objek dengan mempertimbangkan presisi (rasio prediksi benar) dan recall (kemampuan mengidentifikasi positif sejati). mAP dihitung pada berbagai ambang batas IoU, memberikan gambaran menyeluruh tentang kinerja model.

```
[ ] from transformers import pipeline
from PIL import Image

pipe = pipeline("object-detection", model="facebook/detr-resnet-50")

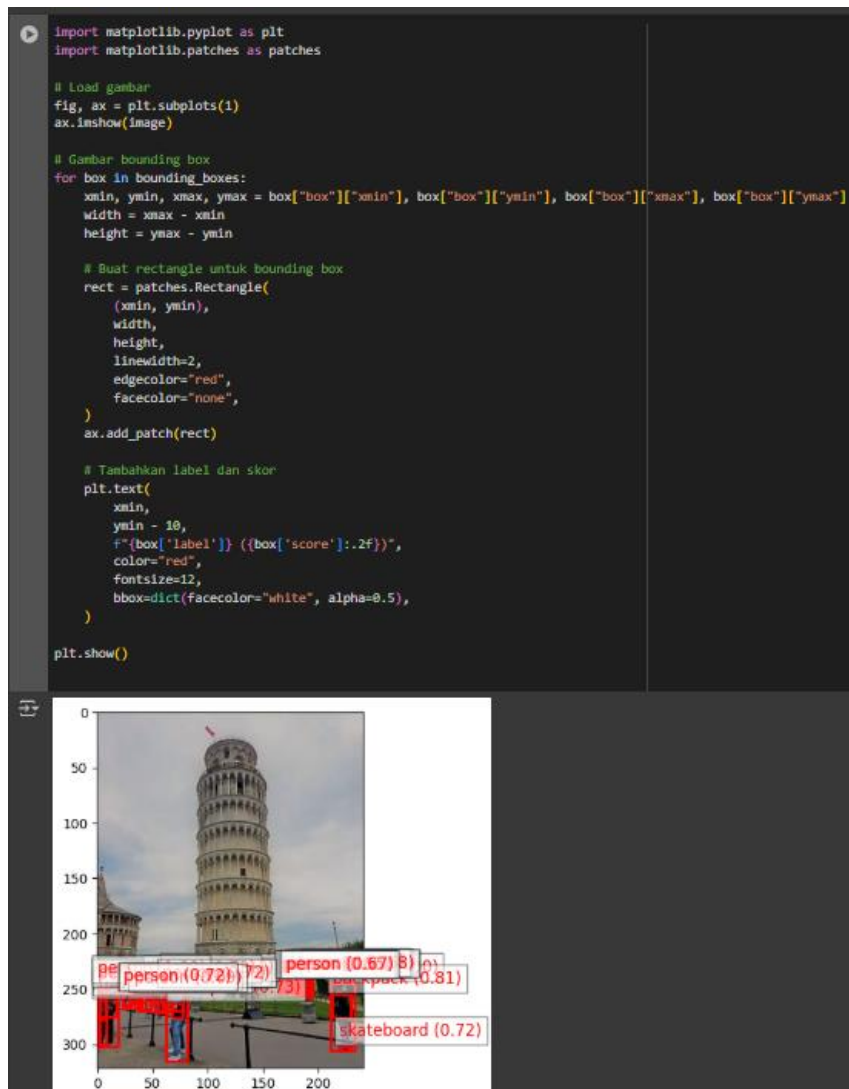
image = Image.open("img4.jpg").convert("RGB")

bounding_boxes = pipe(image)

Some weights of the model checkpoint at facebook/detr-resnet-50 were not used when initializing DetrForObjectDetection: ['model.backbone.conv_encoder.model.layer1.0.downsample.1.num_batches_tracked', 'model.backbone.conv_encoder.model.layer2.0.downsample.1.num_batches_tracked', 'model.backbone.conv_encoder.model.layer3.0.downsample.1.num_batches_tracked', 'model.backbone.conv_encoder.model.layer4.0.downsample.1.num_batches_tracked'].
- This is expected if you are initializing DetrForObjectDetection from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a BertForSequenceClassification model from a BertForPreTraining model).
- This is NOT expected if you are initializing DetrForObjectDetection from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification model).
Device set to use cpu

[ ] # Tampilkan hasil
for box in bounding_boxes:
    print(f"Label: {box['label']}, Confidence: {box['score']:.2f}")
    print(f"Bounding Box: {box['box']}")

Label: person, Confidence: 0.58
Bounding Box: ('min': 188, 'min': 241, 'max': 193, 'max': 258)
Label: person, Confidence: 0.41
Bounding Box: ('min': 185, 'min': 242, 'max': 198, 'max': 258)
Label: person, Confidence: 0.48
Bounding Box: ('min': 174, 'min': 241, 'max': 179, 'max': 253)
Label: person, Confidence: 0.59
Bounding Box: ('min': 52, 'min': 251, 'max': 60, 'max': 272)
Label: person, Confidence: 0.83
Bounding Box: ('min': 0, 'min': 248, 'max': 2, 'max': 302)
Label: kite, Confidence: 0.62
Bounding Box: ('min': 62, 'min': 254, 'max': 78, 'max': 270)
Label: backpack, Confidence: 0.81
Bounding Box: ('min': 215, 'min': 255, 'max': 229, 'max': 276)
Label: person, Confidence: 0.79
Bounding Box: ('min': 60, 'min': 250, 'max': 66, 'max': 271)
Label: person, Confidence: 0.67
Bounding Box: ('min': 178, 'min': 241, 'max': 183, 'max': 253)
Label: person, Confidence: 0.81
Bounding Box: ('min': 180, 'min': 241, 'max': 195, 'max': 254)
Label: person, Confidence: 1.00
Bounding Box: ('min': 62, 'min': 245, 'max': 81, 'max': 315)
Label: person, Confidence: 0.79
Bounding Box: ('min': 183, 'min': 240, 'max': 188, 'max': 254)
Label: person, Confidence: 0.51
Bounding Box: ('min': 0, 'min': 250, 'max': 8, 'max': 300)
Label: person, Confidence: 0.89
Bounding Box: ('min': 55, 'min': 250, 'max': 62, 'max': 271)
Label: person, Confidence: 0.71
Bounding Box: ('min': 57, 'min': 255, 'max': 44, 'max': 270)
Label: person, Confidence: 0.52
Bounding Box: ('min': 165, 'min': 242, 'max': 170, 'max': 253)
```



## 6. 2 Image Segmentation

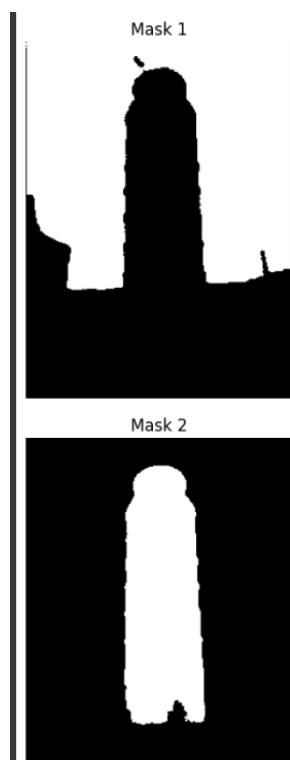
Segmentasi gambar adalah proses membagi gambar menjadi segmen – segmen bermakna dengan menetapkan label pada setiap piksel, sehingga setiap piksel dipetakan ke objek tertentu. Adapun, jenis-jenis segmentasi :

- Segmentasi semantik: Menetapkan label kelas yang sama untuk semua piksel yang mewakili objek serupa tanpa membedakan antara instance individu.
- Segmentasi Instance: Membedakan antara instance individu dari kelas yang sama dengan memberikan label unik pada setiap objek.
- Segmentasi Panoptik: Menggabungkan segmentasi semantik dan instance dengan menetapkan label kelas dan instance pada setiap piksel.

Model berbasis Transformer, seperti Vision Transformer (ViT), telah diterapkan dalam segmentasi gambar. Contohnya, **Segment Anything Model (SAM)** adalah model berbasis ViT yang diperkenalkan oleh Meta AI Research. SAM dilatih pada

dataset besar yang mencakup lebih dari 1 miliar mask pada 11 juta gambar, memungkinkan model ini melakukan segmentasi zero-shot pada gambar baru. Segmentasi Gambar sudah dapat diterapkan ke beberapa aplikasi di dunia nyata, diantaranya :

- **Pertanian:** berbagai bagian lahan dan menilai tahap pertumbuhan tanaman.
- **Kendaraan Otonom:** Mengenali jalur, trotoar, dan pengguna jalan lainnya untuk navigasi yang aman.
- **Penghapusan Latar Belakang:** Model segmentasi digunakan dalam kamera untuk menghapus latar belakang objek tertentu dan menerapkan filter.



## UNIT 7

### VIDEO AND VIDEO PROCESSING

#### 7.1 Introduction

Pada chapter ini akan memperkenalkan konsep dasar video dan pemrosesan video dalam visi komputer. Video adalah format multimedia yang menampilkan serangkaian frame atau gambar secara berurutan, menciptakan ilusi gerakan. Frame ini diperoleh melalui perangkat akusisi seperti kamera atau smartphone. Adapun, aspek – aspek video :

- **Resolusi:** Menunjukkan jumlah piksel dalam setiap frame, misalnya HD (1280x720 piksel), Full HD (1920x1080 piksel), dan Ultra HD atau 4K (3840x2160 piksel). Resolusi lebih tinggi memberikan detail lebih baik tetapi memerlukan lebih banyak ruang penyimpanan dan daya pemrosesan.
- **Frame Rate:** Jumlah frame yang ditampilkan per detik, diukur dalam frame per second (fps). Frame rate umum termasuk 24, 30, dan 60 fps; frame rate lebih tinggi menghasilkan gerakan yang lebih halus.
- **Bitrate:** Jumlah data yang diperlukan untuk menggambarkan audio dan video, biasanya dinyatakan dalam megabit per detik (Mbps) atau kilobit per detik (Kbps). Bitrate lebih tinggi menghasilkan kualitas lebih baik tetapi memerlukan lebih banyak ruang penyimpanan dan bandwidth.
- **Codec:** Singkatan dari "compressor-decompressor," codec adalah komponen perangkat lunak atau keras yang mengompresi dan mendekompresi media digital untuk mengurangi ukuran file sambil mempertahankan kualitas yang dapat diterima. Terdapat dua jenis utama codec: lossless (tanpa kehilangan kualitas) dan lossy (dengan kehilangan kualitas).

#### 7.2 Video Processing Basics

Vision Transformer (ViT) telah menjadi alat penting dalam berbagai tugas visi komputer, termasuk pemrosesan video. Memahami perbedaan antara pemrosesan gambar dan video dengan ViT sangat penting untuk mencapai kinerja optimal. Pemrosesan gambar dengan Vision Transformer dapat dibagi menjadi beberapa tahap:

- **Pembagian Patches:** Gambar dibagi menjadi patch non-overlapping, misalnya gambar 224x224 piksel dibagi menjadi patch 16x16 piksel.
- **Pemrosesan Patches:** Setiap patch diproses melalui lapisan self-attention dan jaringan saraf feed-forward untuk mengekstraksi fitur lokal dan global.

- **Encoding Posisi:** Untuk mempertahankan konteks spasial, ViT menambahkan encoding posisi yang menunjukkan posisi relatif setiap patch dalam gambar.

Adapun perbedaan antara pemrosesan gambar dan video yaitu dengan pendekatan utama diantaranya :

- **Uniform Frame Sampling:**
  - Sampling sejumlah frame secara uniform dari video.
  - Setiap frame diproses seperti gambar statis, dengan membaginya menjadi patch dan mengekstraksi fitur.
  - Token dari semua frame kemudian digabungkan untuk pemrosesan lebih lanjut.
- **Tubelet Embedding**
  - Memperluas embedding gambar ViT ke 3D, mirip dengan konvolusi 3D.
  - Menangkap "tubelet" spatiotemporal yang mencakup beberapa frame, menggabungkan informasi spasial dan temporal selama tokenisasi.

## UNIT 8

### 3D VISION, SCENE RENDERING AND RECONSTRUCTION

#### 8.1 Introduction

Pada kali ini kita akan mempelajari Visi Komputer 3D, yang berfokus pada data visual dalam tiga dimensi spesial:  $x$ ,  $y$ , dan  $z$ . Data 3D memungkinkan representasi dunia nyata yang lebih akurat. Aplikasi umum dari visi komputer 3D termasuk dalam bidang Mixed Reality, dimana dunia digital dan analog digabungkan.

#### 8.2 Applications of 3D Visions

*Computer Vision 3D* memungkinkan mesin untuk melihat dan memahami lingkungan dalam 3 Dimensi, membuka berbagai aplikasi di berbagai industri. Ada beberapa aplikasi yang telah diterapkan dari visi komputer 3D : Robotika dan otomasi, Navigas Otonom, Kesehatan, AR dan VR, serta Hiburan dan Permainan.

#### 8.3 A brief history of 3D Visions

*Computer Vision 3D* telah mengalami beberapa perkembangan sejak abad ke – 19, diantaranya : Stereoskopi (1838), Anaglyph 3D (1853), 3D terpolarisasi (1936), VR (1960-an), Autostereogram (1979), Imax 3D (1986), Sinema Digital 3D (2000-an), Televisi 3D (2010-an), serta perkembangan AR dan VR yang masih berlanjut hingga sekarang.

#### 8.4 Camera Models

Model kamera adalah representasi matematis yang menggambarkan bagaimana titik – titik dalam ruang 3D diproyeksi ke dalam gambar 2D. Pemahaman model ini penting dalam visi komputer seperti rekonstruksi 3D, kalibrasi kamera dan pemetaan citra. Kamera pinhole adalah model paling sederhana yaitu kamera lubang jarum yang terdiri dari kotak kedap cahaya dengan lubang kecil di satu sisi dan layar atau film fotografi di sisi lain. Cahaya yang melewati lubang ini membentuk gambar terbalik pada dinding kotak belakang.

Kamera lubang jarum mengalami transformasi dimana tiap titik dalam ruang 3D dipetakan ke satu titik bidang 2D, untuk menentukan peta antar koordinat 3D dan 2D, diperlukan parameter intrinsik kamera, yaitu panjang fokus pada sumbu dan koordinat titik pusat.

#### 8.5 Basic Linear Algebra for 3D Data

```

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])

```

```

# 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

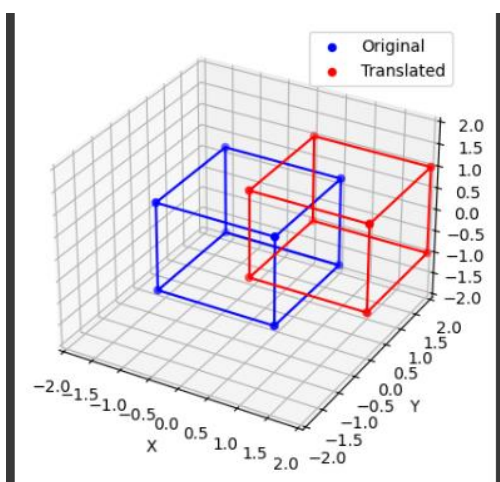
# set up figure
fig = plt.figure()
ax = fig.add_subplot(111, projection="3d")

# 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")

```





```

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

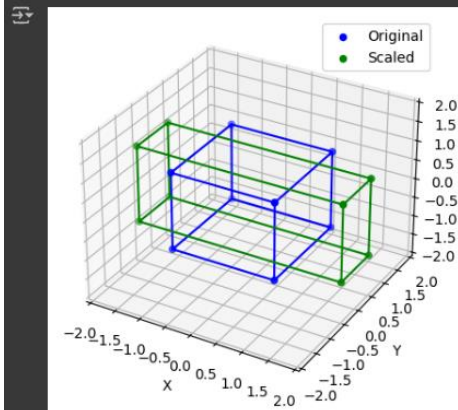
# plot original cube
plot_cube(ax, cube, label="Original", color="blue")

# 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")

```



```

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

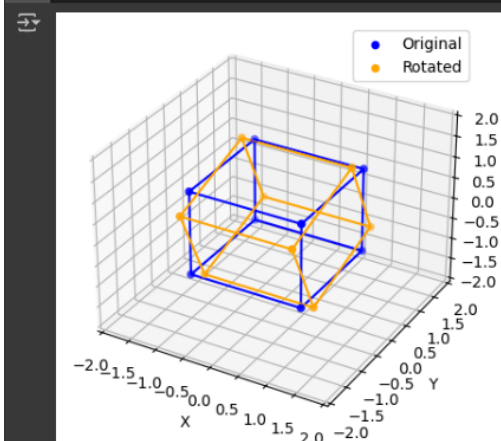
# plot original cube
plot_cube(ax, cube, label="Original", color="blue")

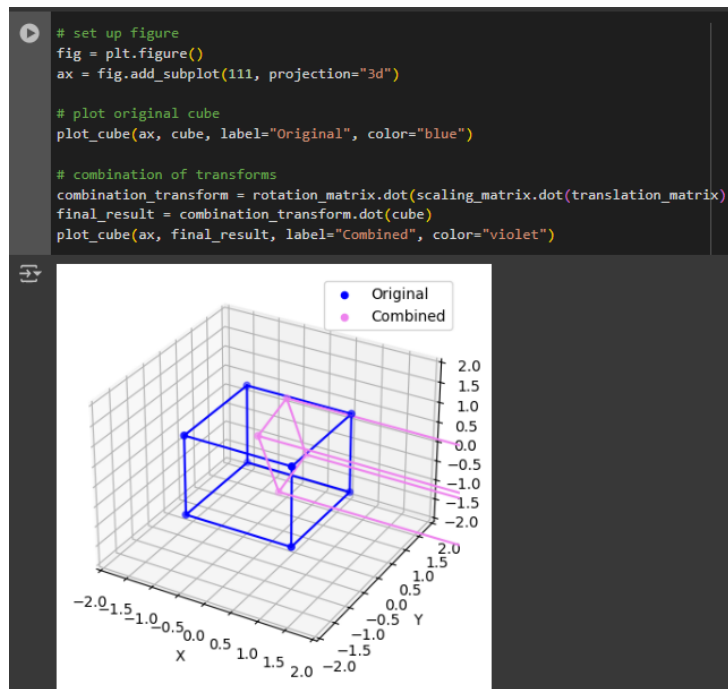
# rotation matrix: +20 deg around x-axis
angle = 20 * np.pi / 180
rotation_matrix = np.array(
    [
        [1, 0, 0, 0],
        [0, np.cos(angle), -np.sin(angle), 0],
        [0, np.sin(angle), np.cos(angle), 0],
        [0, 0, 0, 1],
    ]
)

rotated_cube = rotation_matrix @ cube

plot_cube(ax, rotated_cube, label="Rotated", color="orange")

```





## 8. 6 Representation for 3D Data

Dalam komputer vision, representasi data 3D digunakan sesuai dengan kebutuhan aplikasi. Berikut beberapa representasi umum :

- **Points Cloud** : merepresentasikan objek sebagai kumpulan titik dalam ruang 3D, masing-masing dengan koordinatnya. Representasi ini sering dihasilkan melalui teknik seperti pemindaian LIDAR. Namun, point cloud tidak menyertakan informasi tentang konektivitas antar titik, sehingga sulit untuk menentukan permukaan atau topologi objek.
- **Mashes** : merepresentasikan permukaan objek sebagai kumpulan wajah (biasanya berbentuk segitiga) yang menghubungkan titik-titik dalam ruang 3D. Informasi tambahan seperti normal, warna, atau koordinat tekstur dapat dikaitkan dengan titik atau wajah. Mesh efisien untuk menyimpan objek padat dan sering digunakan dalam grafis komputer, seperti pengembangan game.
- **Volumetric Data** : Data volumetrik merepresentasikan objek dengan fungsi  $f(x,y,z)$  yang memetakan posisi dalam ruang ke densitas, warna, atau atribut lain. Representasi ini digunakan untuk objek transparan seperti awan atau api. Salah satu metode umum adalah grid volumetrik, di mana data di setiap titik dihitung melalui interpolasi dari voxel di sekitarnya.
- **Implicit Surfaces** : Permukaan implisit merepresentasikan objek sebagai himpunan titik di mana fungsi  $f(x,y,z)$  bernilai nol. Jika fungsi ini berupa Signed Distance Function (SDF), nilai positif menunjukkan posisi di luar

objek, sementara nilai negatif berada di dalam objek. Representasi ini memungkinkan perhitungan cepat untuk menemukan interseksi antara garis lurus dan permukaan menggunakan algoritma seperti sphere tracing.

### **8.7 Novel View Synthesis**

Novel View Synthesis (NVS) adalah metode untuk menghasilkan tampilan objek dari sudut kamera baru yang konsisten dengan serangkaian gambar yang ada. Teknik ini berguna ketika hanya tersedia beberapa gambar atau bahkan satu gambar saja. Setelah memperoleh serangkaian gambar yang konsisten, algoritma seperti NeRF dapat digunakan untuk membangun representasi 3D.

Pendekatan NVS dibagi menjadi dua : Pendekatan dengan representasi 3D Intermediate, dan Pendekatan tanpa representasi 3D Intermediate. Tantangan utama NVS adalah sifatnya yang sering kali tidak terdefinisi dengan baik, terutama ketika objek tersembunyi atau tidak terlihat dalam gambar input.

### **8.8 Introduction to Stereo Vision**

Visi stereo adalah teknik yang memungkinkan penentuan struktur tiga dimensi objek dengan menggunakan dua atau lebih gambar dari sudut pandang berbeda. Dengan menganalisis perbedaan antara gambar-gambar ini, dimungkinkan untuk merekonstruksi koordinat 3D dari titik-titik dalam adegan.

Adapun prinsip dasar dalam Stereo Vision adalah :

- **Proyeksi 3D ke 2D:** Sebuah titik dalam ruang 3D diproyeksikan ke bidang gambar 2D melalui lubang jarum kamera. Proses ini mengakibatkan hilangnya informasi kedalaman, sehingga titik-titik yang terletak pada garis yang sama melalui lubang jarum akan diproyeksikan ke titik yang sama pada bidang gambar.
- **Ambiguitas Kedalaman:** Dengan hanya satu gambar, tidak mungkin menentukan posisi pasti titik dalam ruang 3D karena semua titik pada garis yang sama akan tampak identik dalam gambar.

## 8.9 Neural Radiance Fields (NeRFs)

```
import torch
import mediapy as media
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:")
    media.show_images([x_channel, y_channel], cmap="plasma", border=True)
    # Show the encoded grid
    print("Encoded Values:")
    num_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
    )[:num_channels_to_visualize]
    media.show_images(encoded_images_to_show, vmin=-1, vmax=1, cmap="plasma", border=True)

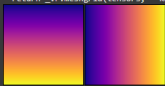
# Parameters similar to your NeRFEncoding example
num_frequencies = 4
min_freq_exp = 0
max_freq_exp = 6
resolution = 128

# 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]
```

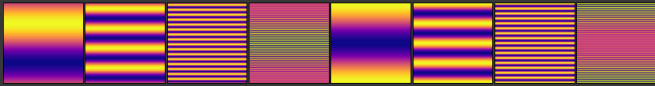
```
# Apply positional encoding
encoded_grid = positional_encoding(grid, num_frequencies, min_freq_exp, max_freq_exp)

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

Input Values:  
/usr/local/lib/python3.10/dist-packages/torch/functional.py:534: UserWarning: torch.meshgrid: in an upcoming release, it will be required to pass the indexing argument. (Triggered internally at ../aten/src/ATen/native/TensorShape.cpp:3595.)  
return \_VF.meshgrid(tensors, \*\*kwargs) # type: ignore[attr-defined]



Encoded Values:



### Daftar Pustaka

- [1] S. Zhao, J. He, X. Du, dan Y. Zhang, "SGFNet: Segmentation Guided Fusion Network for 3D Object Detection," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 2, pp. 1-12, 2023, doi: 10.1109/TNNLS.2023.10290931.
- [2] H. Liu, Z. Wang, C. Li, dan J. Chen, "Improving GAN-based Domain Adaptation for Object Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 3, pp. 789-798, 2023, doi: 10.1109/TPAMI.2023.9922138.
- [3] Q. Lin, X. Wang, dan L. Xu, "3D Domain Adaptive Instance Segmentation via Cyclic Segmentation GANs," *IEEE Transactions on Image Processing*, vol. 32, pp. 2345-2357, 2023, doi: 10.1109/TIP.2023.10138543.