5. CLIP

```
!wget https://github.com/ichaparroc/IA-ESIS-UNJBG/raw/main/GenAI.zip
!unzip GenAI.zip
```

```
  inflating: data/cropped_flowers/sunflowers/5994572653_ea98afa3af_n.jpg
  inflating: data/cropped_flowers/sunflowers/5994586159_1dd99d66b4_n.jpg
  inflating: data/cropped_flowers/sunflowers/5995136822_8e1eed76f5_n.jpg
  inflating: data/cropped_flowers/sunflowers/6042014768_b57f0bfc79_n.jpg
  inflating: data/cropped_flowers/sunflowers/6080086410_17a02dcfb8.jpg
  inflating: data/cropped_flowers/sunflowers/6116210027_61923f4b64.jpg
  inflating: data/cropped_flowers/sunflowers/6198569587_23c3693328_m.jpg
  inflating: data/cropped_flowers/sunflowers/6199086734_b7ddc65816_m.jpg
  inflating: data/cropped_flowers/sunflowers/6250692311_cb60c85ee9_n.jpg
  inflating: data/cropped_flowers/sunflowers/6265084065_7a8b30cc6e_n.jpg
  inflating: data/cropped_flowers/sunflowers/6482016425_d8fab362f6.jpg
  inflating: data/cropped_flowers/sunflowers/6482016439_b0d06dac04.jpg
  inflating: data/cropped_flowers/sunflowers/6495554833_86eb8faa8e_n.jpg
  inflating: data/cropped_flowers/sunflowers/6495559397_61d01c0c57.jpg
  inflating: data/cropped_flowers/sunflowers/6606815161_3c4372760f.jpg
  inflating: data/cropped_flowers/sunflowers/6606817351_10f6e43a09.jpg
  inflating: data/cropped_flowers/sunflowers/6606820461_952c450f90_n.jpg
  inflating: data/cropped_flowers/sunflowers/6606823367_e89dc52a95_n.jpg
  inflating: data/cropped_flowers/sunflowers/6627521877_6e43fb3c49_m.jpg
  inflating: data/cropped_flowers/sunflowers/678714585_addc9aaaef.jpg
  inflating: data/cropped_flowers/sunflowers/7176723954_e41618edc1_n.jpg
  inflating: data/cropped_flowers/sunflowers/7176736574_14446539cb_n.jpg
  inflating: data/cropped_flowers/sunflowers/7369484298_332f69bd88_n.jpg
  inflating: data/cropped_flowers/sunflowers/7510240282_87554c7418_n.jpg
  inflating: data/cropped_flowers/sunflowers/7510262868_cf7d6f6f25_n.jpg
  inflating: data/cropped_flowers/sunflowers/7652532108_01ef94c476.jpg
  inflating: data/cropped_flowers/sunflowers/7728953426_abd179ab63.jpg
  inflating: data/cropped_flowers/sunflowers/7791014076_07a897cb85_n.jpg
  inflating: data/cropped_flowers/sunflowers/7820305664_82148f3bfb_n.jpg
  inflating: data/cropped_flowers/sunflowers/7935826214_9b57628203_m.jpg
  inflating: data/cropped_flowers/sunflowers/8038712786_5bdeed3c7f_m.jpg
  inflating: data/cropped_flowers/sunflowers/8174935013_b16626b49b.jpg
  inflating: data/cropped_flowers/sunflowers/8174935717_d19367d502.jpg
  inflating: data/cropped_flowers/sunflowers/821368661_4ab4343f5a.jpg
  inflating: data/cropped_flowers/sunflowers/8234846550_fdaf326dbe.jpg
  inflating: data/cropped_flowers/sunflowers/8249000137_eddfffa380_n.jpg
  inflating: data/cropped_flowers/sunflowers/8292914969_4a76608250_m.jpg
  inflating: data/cropped_flowers/sunflowers/8480886751_71d88bfdc0_n.jpg
  inflating: data/cropped_flowers/sunflowers/8563099326_8be9177101.jpg
  inflating: data/cropped_flowers/sunflowers/857698097_8068a2c135_n.jpg
  inflating: data/cropped_flowers/sunflowers/8705462313_4458d64cd4.jpg
  inflating: data/cropped_flowers/sunflowers/8929288228_6795bcb1fe.jpg
  inflating: data/cropped_flowers/sunflowers/9206376642_8348ba5c7a.jpg
  inflating: data/cropped_flowers/sunflowers/9240129413_f240ce7866_n.jpg
  inflating: data/cropped_flowers/sunflowers/9375675309_987d32f99e_n.jpg
  inflating: data/cropped_flowers/sunflowers/9384867134_83af458a19_n.jpg
  inflating: data/cropped_flowers/sunflowers/9410186154_465642ed35.jpg
  inflating: data/cropped_flowers/sunflowers/9432335346_e298e47713_n.jpg
  inflating: data/cropped_flowers/sunflowers/9448615838_04078d09bf_n.jpg
  inflating: data/cropped_flowers/sunflowers/9497774935_a7daec5433_n.jpg
  inflating: data/cropped_flowers/sunflowers/9558627290_353a14ba0b_m.jpg
  inflating: data/cropped_flowers/sunflowers/9558628596_722c29ec60_m.jpg
  inflating: data/cropped_flowers/sunflowers/9558630626_52a1b7d702_m.jpg
  inflating: data/cropped_flowers/sunflowers/9558632814_e78a780f4f.jpg
  inflating: data/cropped_flowers/sunflowers/9610373748_b9cb67bd55.jpg
  inflating: data/cropped_flowers/sunflowers/9610374042_bb16cded3d.jpg
  inflating: data/cropped_flowers/sunflowers/9681915384_b3b646dc92_m.jpg
  inflating: data/cropped_flowers/sunflowers/9783416751_b2a03920f7_n.jpg
```

Contrastive Language-Image Pre-Training o CLIP es una herramienta de codificación de texto e imagen utilizada con muchos modelos populares de IA Generativa como DALL-E y Stable Diffusion.

CLIP en sí mismo no es un modelo de IA Generativa, sino que se utiliza para alinear codificaciones de texto con codificaciones de imagen. Si existe la descripción textual perfecta de una imagen, el objetivo de CLIP es crear la misma incrustación vectorial tanto para la imagen como para el texto. Veamos qué significa esto en la práctica.

Los objetivos de este cuaderno son:

- Aprender a utilizar las codificaciones CLIP
    - Obtener una codificación de imagen
    - Obtener una codificación de texto
    - Calcular la similitud coseno entre ellas
- Usar CLIP para crear una red neuronal texto-imagen

5.1 Codificaciones

Primero, carguemos las librerías necesarias para este ejercicio.

```
!pip install einops
```

```
Collecting einops
  Downloading einops-0.8.0-py3-none-any.whl.metadata (12 kB)
Downloading einops-0.8.0-py3-none-any.whl (43 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 43.2/43.2 kB 3.4 MB/s eta 0:00:00
Installing collected packages: einops
Successfully installed einops-0.8.0
```

```
import glob
import numpy as np
import torch
import torch.nn.functional as F
from torch.optim import Adam
import torchvision.transforms as transforms
from torch.utils.data import Dataset, DataLoader

# Visualization tools
import matplotlib.pyplot as plt
from PIL import Image
from torchvision.utils import save_image, make_grid
from textwrap import wrap

# User defined libraries
from utils import other_utils
from utils import ddpm_utils
from utils import UNet_utils

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

Existen algunas variantes de CLIP basadas en las redes neuronales de reconocimiento de imágenes más conocidas:

```
pip install openai-clip
```

```
Collecting openai-clip
  Downloading openai-clip-1.0.1.tar.gz (1.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.4/1.4 MB 23.3 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Collecting ftfy (from openai-clip)
  Downloading ftfy-6.2.0-py3-none-any.whl.metadata (7.3 kB)
Requirement already satisfied: regex in /usr/local/lib/python3.10/dist-packages (from openai-clip) (2024.5.15)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from openai-clip) (4.66.4)
Requirement already satisfied: wcwidth<0.3.0,>=0.2.12 in /usr/local/lib/python3.10/dist-packages (from ftfy->openai-clip) (0.2.13)
Downloading ftfy-6.2.0-py3-none-any.whl (54 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 54.4/54.4 kB 4.5 MB/s eta 0:00:00
Building wheels for collected packages: openai-clip
  Building wheel for openai-clip (setup.py) ... done
  Created wheel for openai-clip: filename=openai_clip-1.0.1-py3-none-any.whl size=1368606 sha256=858a2c11560336edae5b5340c4139972d66b98fd7eb88a
  Stored in directory: /root/.cache/pip/wheels/08/77/8e/8d2f862df6bf7fb4e2007062d2cbaeae49862ec7b56d041229
Successfully built openai-clip
Installing collected packages: ftfy, openai-clip
Successfully installed ftfy-6.2.0 openai-clip-1.0.1
```

```
import clip

clip.available_models()
```

```
['RN50',
 'RN101',
 'RN50x4',
 'RN50x16',
 'RN50x64',
 'ViT-B/32',
 'ViT-B/16',
 'ViT-L/14',
 'ViT-L/14@336px']
```

Para este cuaderno, utilizaremos `ViT-B/32`, que se basa en la arquitectura Vision Transformer. Dispone de 512 características, que más tarde introduciremos en nuestro modelo de difusión.

```
clip_model, clip_preprocess = clip.load("ViT-B/32")
clip_model.eval()
CLIP_FEATURES = 512
```

```
100%|███████████████████████████████████| 338M/338M [00:02<00:00, 138MiB/s]
```

5.1.1 Codificaciones de imagen

Cuando carguemos CLIP, también vendrá con un conjunto de transformaciones de imagen que podemos utilizar para introducir imágenes en el modelo CLIP:

```
clip_preprocess
```

```
Compose(
    Resize(size=224, interpolation=bicubic, max_size=None, antialias=True)
```

```
        CenterCrop(size=(224, 224))
        <function _convert_image_to_rgb at 0x79e9f7cf4d30>
        ToTensor()
        Normalize(mean=(0.48145466, 0.4578275, 0.40821073), std=(0.26862954, 0.26130258, 0.27577711))
    )
```

Podemos probarlo con una de nuestras fotos de flores. Empecemos con una pintoresca **margarita**.

```
DATA_DIR = "data/cropped_flowers/"
img_path = DATA_DIR + "daisy/2877860110_a842f8b14a_m.jpg"
img = Image.open(img_path)
img.show()
```

Podemos encontrar la incrustación CLIP transformando primero nuestra imagen con `clip_preprocess` y convirtiendo el resultado en un tensor. Dado que el `clip_model` espera un lote de imágenes, podemos utilizar [np.stack]() para convertir la imagen procesada en un lote de un solo elemento.

```
clip_imgs = torch.tensor(np.stack([clip_preprocess(img)])).to(device)
clip_imgs.size()
```

```
torch.Size([1, 3, 224, 224])
```

A continuación, podemos pasar el lote a `clip_model.encode_image` para encontrar la incrustación de la imagen. Descomenta `clip_img_encoding` si quieres ver cómo es una codificación. Cuando imprimimos el tamaño, aparece `512` características para nuestra imagen `1`.

```
clip_img_encoding = clip_model.encode_image(clip_imgs)
print(clip_img_encoding.size())
#clip_img_encoding
```

```
torch.Size([1, 512])
```

5.1.2 Codificaciones de texto

Ahora que tenemos una codificación de imagen, veamos si podemos obtener una codificación de texto que se corresponda. A continuación se muestra una lista de diferentes descripciones de flores. Como en el caso de las imágenes, el texto debe ser preprocesado antes de ser codificado por CLIP. Para ello, CLIP incluye una función `tokenize` que convierte cada palabra en un número entero.

```
text_list = [
    "A round white daisy with a yellow center",
    "An orange sunflower with a big brown center",
    "A red rose bud"
]
text_tokens = clip.tokenize(text_list).to(device)
text_tokens
```

```
tensor([[49406,   320,  2522,  1579, 12865,   593,   320,  4481,  2119, 49407,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0],
        [49406,   550,  4287, 21559,   593,   320,  1205,  2866,  2119, 49407,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0],
        [49406,   320,   736,  3568, 10737, 49407,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0]], device='cuda:0',
       dtype=torch.int32)
```

A continuación, podemos pasar los tokens a `encode_text` para obtener nuestras codificaciones de texto. Descomenta `clip_text_encodings` si quieres ver el aspecto de una codificación. Similar a nuestra codificación de imagen, hay `512` características para cada una de nuestras `3` imágenes.

```
clip_text_encodings = clip_model.encode_text(text_tokens).float()
print(clip_text_encodings.size())
#clip_text_encodings
```

```
torch.Size([3, 512])
```

5.1.3 Similaridad

Para ver cuál de nuestras descripciones de texto describe mejor la margarita, podemos calcular la similitud de coseno entre las codificaciones de texto y las codificaciones de imagen. Cuando la similitud de coseno es `1`, es una coincidencia perfecta. Cuando la similitud de coseno es `-1`, las dos codificaciones son opuestas.

La similitud de coseno es equivalente a un producto escalar con cada vector normalizado por su magnitud. En otras palabras, la magnitud de cada vector se convierte en `1`.

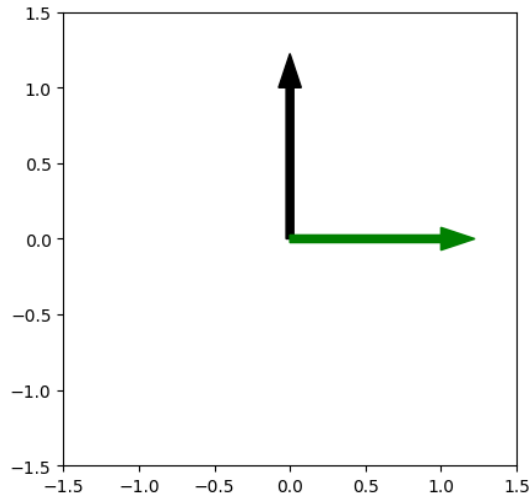
Podemos utilizar la siguiente fórmula para calcular el producto escalar:

$$X \cdot Y = \sum_{i=1}^{n} x_i y_i = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n$$

Practiquémoslo un poco.

Intentemos cambiar `x1`, `y1`, `x2` e `y2` por un valor entre `-1` y `1`. Cuando las flechas están alineadas, la similitud del coseno es `1`. Cuando las flechas apuntan en direcciones opuestas, la similitud es `-1`.

```
x1, y1 = [0, 1] # Change me
x2, y2 = [1, 0] # Change me

p1 = [x1, y1]
p2 = [x2, y2]

arrow_width = 0.05
plt.axis('square')
plt.xlim(-1.5, 1.5)
plt.ylim(-1.5, 1.5)
plt.arrow(0, 0, x1, y1, width=arrow_width, color="black")
plt.arrow(0, 0, x2, y2, width=arrow_width, color="green")
plt.show()

cosine = np.dot(p1, p2) / (np.linalg.norm(p1) * np.linalg.norm(p2))
print("Cosine Similarity:", cosine)
```




```
Cosine Similarity: 0.0
```

Ejercicio 1

Crea nuevas celdas con el código de la celda anterior, probando los siguientes valores.

```
x1, y1 = [0, 0.5]
x2, y2 = [0, 1]
```

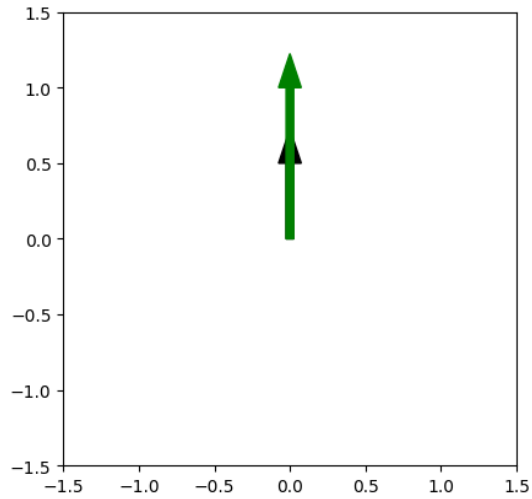
```
x1, y1 = [0, -1]
x2, y2 = [0, 0.5]
```

```
x1, y1 = [1, 1]
x2, y2 = [0, 1]
```

Para cada uno describe cuál es el significado del valor obtenido de la similaridad coseno.

```
x1, y1 = [0, 0.5]
x2, y2 = [0, 1]

p1 = [x1, y1]
p2 = [x2, y2]
```

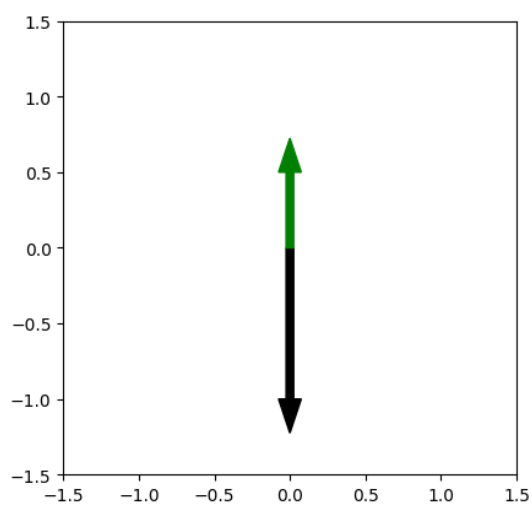
```
arrow_width = 0.05
plt.axis('square')
plt.xlim(-1.5, 1.5)
plt.ylim(-1.5, 1.5)
plt.arrow(0, 0, x1, y1, width=arrow_width, color="black")
plt.arrow(0, 0, x2, y2, width=arrow_width, color="green")
plt.show()

cosine = np.dot(p1, p2) / (np.linalg.norm(p1) * np.linalg.norm(p2))
print("Cosine Similarity:", cosine)
```




```
Cosine Similarity: 1.0
```

Caso 1: Los vectores x1, y1 = [0, 0.5] x2, y2 = [0, 1] están alineados en la misma dirección en el eje y. La similaridad coseno es 1, indicando que los vectores son perfectamente colineales y apuntan en la misma dirección.

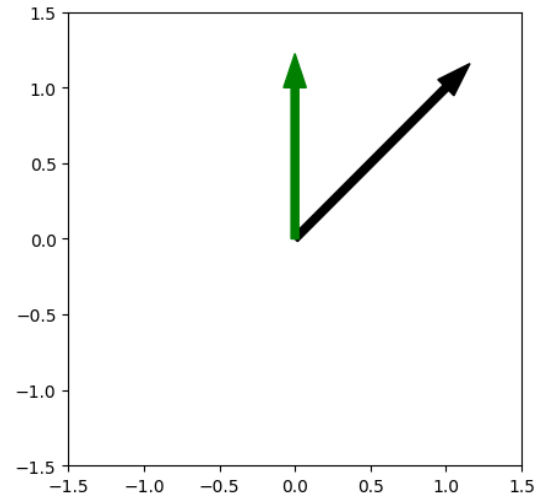
```
x1, y1 = [0, -1]
x2, y2 = [0, 0.5]

p1 = [x1, y1]
p2 = [x2, y2]

arrow_width = 0.05
plt.axis('square')
plt.xlim(-1.5, 1.5)
plt.ylim(-1.5, 1.5)
plt.arrow(0, 0, x1, y1, width=arrow_width, color="black")
plt.arrow(0, 0, x2, y2, width=arrow_width, color="green")
plt.show()

cosine = np.dot(p1, p2) / (np.linalg.norm(p1) * np.linalg.norm(p2))
print("Cosine Similarity:", cosine)
```




```
Cosine Similarity: -1.0
```

Caso 2: Los vectores x1, y1 = [0, -1] x2, y2 = [0, 0.5] están alineados en direcciones opuestas en el eje y. La similaridad coseno es -1, indicando que los vectores son perfectamente colineales pero apuntan en direcciones opuestas.

```
x1, y1 = [1, 1]
x2, y2 = [0, 1]

p1 = [x1, y1]
p2 = [x2, y2]

arrow_width = 0.05
plt.axis('square')
plt.xlim(-1.5, 1.5)
plt.ylim(-1.5, 1.5)
plt.arrow(0, 0, x1, y1, width=arrow_width, color="black")
plt.arrow(0, 0, x2, y2, width=arrow_width, color="green")
plt.show()

cosine = np.dot(p1, p2) / (np.linalg.norm(p1) * np.linalg.norm(p2))
print("Cosine Similarity:", cosine)
```




```
Cosine Similarity: 0.7071067811865475
```

Caso 3: Los vectores x1, y1 = [1, 1] x2, y2 = [0, 1] forman un ángulo de 45 grados entre ellos. La similaridad coseno es 0.707, indicando que los vectores están parcialmente alineados en la misma dirección.

La similitud de cosenos también funciona con vectores multidimensionales, aunque son más difíciles de representar gráficamente en una superficie 2D.

```
p1 = [1, 8, 6, 7]
p2 = [5, 3, 0, 9]

cosine = np.dot(p1, p2) / (np.linalg.norm(p1) * np.linalg.norm(p2))
print("Cosine Similarity:", cosine)
```

```
Cosine Similarity: 0.7004760286167305
```

Intentemos calcular la puntuación de similitud para nuestras codificaciones CLIP.

```
clip_img_encoding /= clip_img_encoding.norm(dim=-1, keepdim=True)
clip_text_encodings /= clip_text_encodings.norm(dim=-1, keepdim=True)
similarity = (clip_text_encodings * clip_img_encoding).sum(-1)
similarity
```

```
tensor([0.3707, 0.2473, 0.1768], device='cuda:0', grad_fn=<SumBackward1>)
```

A continuación imprimiremos cada texto seguido de su valor de similaridad.

```
for idx, text in enumerate(text_list):
    print(text, " - ", similarity[idx])
```

```
A round white daisy with a yellow center  -  tensor(0.3707, device='cuda:0', grad_fn=<SelectBackward0>)
An orange sunflower with a big brown center  -  tensor(0.2473, device='cuda:0', grad_fn=<SelectBackward0>)
A red rose bud  -  tensor(0.1768, device='cuda:0', grad_fn=<SelectBackward0>)
```

Ejercico 2

¿Cuál es el texto con mayor valor?

- El texto con mayor valor es "A round white daisy with a yellow center" con un valor de 0.3707.

¿El texto con mayor valor describe una margarita? ¿Por qué?

- Sí, el texto con mayor valor describe una margarita ("daisy" en inglés).

- Porque es una flor redonda y blanca con un centro amarillo. La alta puntuación (0.3707) indica que este texto tiene la mayor similitud o relevancia en el contexto específico evaluado, posiblemente en relación con un modelo de clasificación o similaridad de textos.

Practiquemos un poco más. A continuación, procesaremos la imagen de una **margarita, girasol y rosa**. hasta obtener su representación vectorial o embedding.

```
img_paths = [
    DATA_DIR + "daisy/2877860110_a842f8b14a_m.jpg",
    DATA_DIR + "sunflowers/2721638730_34a9b7a78b.jpg",
    DATA_DIR + "roses/8032328803_30afac8b07_m.jpg"
]

imgs = [Image.open(path) for path in img_paths]
for img in imgs:
    img.show()
```

```
def get_img_encodings(imgs):
    processed_imgs = [clip_preprocess(img) for img in imgs]
    clip_imgs = torch.tensor(np.stack(processed_imgs)).to(device)
    clip_img_encodings = clip_model.encode_image(clip_imgs)
    return clip_img_encodings
```

```
clip_img_encodings = get_img_encodings(imgs)
clip_img_encodings
```

```
tensor([[-0.2717, -0.0157, -0.1792,  ...,  0.5811,  0.0866, -0.1448],
        [ 0.2595, -0.1020, -0.3438,  ..., -0.0073,  0.4958,  0.0819],
        [-0.0620,  0.4136,  0.0090,  ...,  0.3269,  0.4626, -0.1388]],
       device='cuda:0', dtype=torch.float16, grad_fn=<MmBackward0>)
```

Ahora, calcularemos la representación vectorial o embedding de los siguientes textos.

```
text_list = [
    "A round white daisy with a yellow center", #"Una margarita blanca redonda con un centro amarillo",
    "An orange sunflower with a big brown center", #"Un girasol naranja con un gran centro marrón",
    "A deep red rose flower" #"Una rosa de color rojo intenso"
]
```

```
text_tokens = clip.tokenize(text_list).to(device)
clip_text_encodings = clip_model.encode_text(text_tokens).float()
clip_text_encodings
```

```
tensor([[-0.5107,  0.1919,  0.1963,  ...,  0.0949, -0.0848, -0.2783],
        [ 0.0224,  0.3889,  0.3506,  ...,  0.1219,  0.0778, -0.1910],
        [-0.1094,  0.0110,  0.3333,  ...,  0.1288, -0.1019, -0.1853]],
       device='cuda:0', grad_fn=<ToCopyBackward0>)
```

Sería bueno comparar cada combinación de texto e imagen. Para ello, podemos repetir cada codificación de texto para cada codificación de imagen. De manera similar, podemos repetir_interleave cada codificación de imagen para cada codificación de texto.

```
clip_img_encodings /= clip_img_encodings.norm(dim=-1, keepdim=True)
clip_text_encodings /= clip_text_encodings.norm(dim=-1, keepdim=True)

n_imgs = len(imgs)
n_text = len(text_list)
```

```
repeated_clip_text_encodings = clip_text_encodings.repeat(n_imgs, 1)
repeated_clip_text_encodings
```

```
tensor([[-0.0729,  0.0274,  0.0280,  ...,  0.0135, -0.0121, -0.0397],
        [ 0.0031,  0.0545,  0.0492,  ...,  0.0171,  0.0109, -0.0268],
        [-0.0135,  0.0014,  0.0410,  ...,  0.0158, -0.0125, -0.0228],
        ...,
        [-0.0729,  0.0274,  0.0280,  ...,  0.0135, -0.0121, -0.0397],
        [ 0.0031,  0.0545,  0.0492,  ...,  0.0171,  0.0109, -0.0268],
        [-0.0135,  0.0014,  0.0410,  ...,  0.0158, -0.0125, -0.0228]],
       device='cuda:0', grad_fn=<RepeatBackward0>)
```

```
repeated_clip_img_encoding = clip_img_encodings.repeat_interleave(n_text, dim=0)
repeated_clip_img_encoding
```

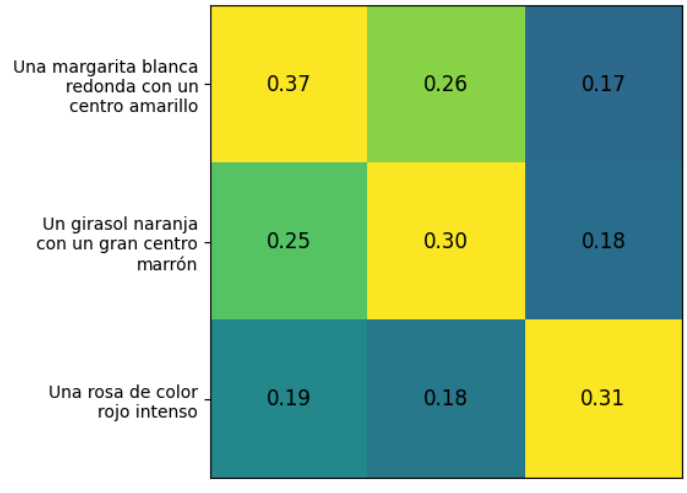
```
tensor([[-0.0247, -0.0014, -0.0163,  ...,  0.0528,  0.0079, -0.0132],
        [-0.0247, -0.0014, -0.0163,  ...,  0.0528,  0.0079, -0.0132],
        [-0.0247, -0.0014, -0.0163,  ...,  0.0528,  0.0079, -0.0132],
        ...,
        [-0.0053,  0.0356,  0.0008,  ...,  0.0282,  0.0399, -0.0120],
        [-0.0053,  0.0356,  0.0008,  ...,  0.0282,  0.0399, -0.0120],
        [-0.0053,  0.0356,  0.0008,  ...,  0.0282,  0.0399, -0.0120]],
       device='cuda:0', dtype=torch.float16, grad_fn=<ViewBackward0>)
```

```
similarity = (repeated_clip_text_encodings * repeated_clip_img_encoding).sum(-1)
similarity = torch.unflatten(similarity, 0, (n_text, n_imgs))
similarity
```

```
tensor([[0.3705, 0.2471, 0.1943],
        [0.2647, 0.2993, 0.1810],
        [0.1700, 0.1754, 0.3078]], device='cuda:0', grad_fn=<ViewBackward0>)
```

Comparemos. Lo ideal sería que la diagonal desde la parte superior izquierda hasta la inferior derecha fuera de un amarillo brillante, correspondiente a su valor alto. El resto de los valores deberían ser bajos y de color azul.

```
fig = plt.figure(figsize=(10, 10))
gs = fig.add_gridspec(2, 3, wspace=.1, hspace=0)

for i, img in enumerate(imgs):
    ax = fig.add_subplot(gs[0, i])
    ax.axis("off")
    plt.imshow(img)

ax = fig.add_subplot(gs[1, :])
plt.imshow(similarity.detach().cpu().numpy().T, vmin=0.1, vmax=0.3)

text_list = [
    "Una margarita blanca redonda con un centro amarillo",
    "Un girasol naranja con un gran centro marrón",
    "Una rosa de color rojo intenso"
]

labels = [ '\n'.join(wrap(text, 20)) for text in text_list ]
plt.yticks(range(n_text), labels, fontsize=10)
plt.xticks([])

for x in range(similarity.shape[1]):
    for y in range(similarity.shape[0]):
        plt.text(x, y, f"{similarity[x, y]:.2f}", ha="center", va="center", size=12)
```



Ejercicio 3

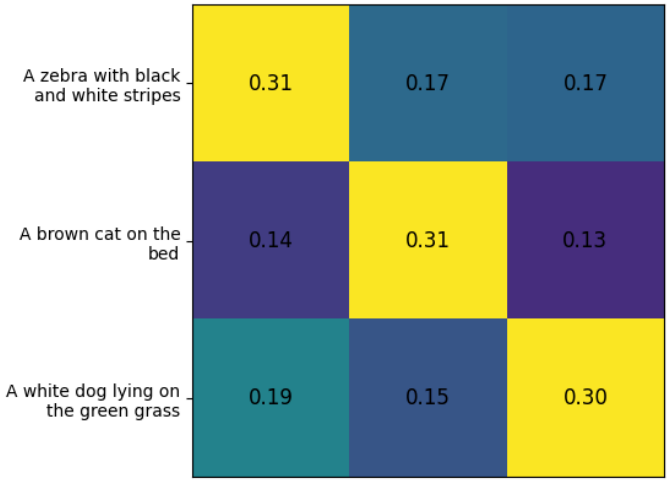
¿Son correctos los resultados? ¿Por qué?

- Los resultados son correctos, porque la diagonal tiene los valores más altos, indicando que cada texto es más similar a sí mismo, y los valores fuera de la diagonal son más bajos, indicando menor similaridad entre diferentes textos.

Ejercicio 4

Cree nuevas celdas en las cuales compare 3 imágenes de internet y 3 textos (en inglés). Ejemplo: Puede usar una foto de las pirámides de egipto, Machu Picchu y Cristo Redendor, luego crear 3 textos descriptivos en inglés e imprimir la matriz de similaridad (ya no vale usazr este ejemplo).

```
import torch
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import clip

#Definimos las  rutas de las imágenes y cargar imágenes.
img_paths = [
    "/content/zebra.jpg",
    "/content/gato.jpg",
    "/content/perro.jpg"
]

imgs = [Image.open(path) for path in img_paths]

# Mostrar las imagenes
for img in imgs:
    img.show()

# Cargar el modelo CLIP
device = "cuda" if torch.cuda.is_available() else "cpu"
clip_model, preprocess = clip.load('ViT-B/32', device=device)

def get_img_encodings(imgs):
    processed_imgs = [preprocess(img) for img in imgs]
    clip_imgs = torch.tensor(np.stack(processed_imgs)).to(device)
    clip_img_encodings = clip_model.encode_image(clip_imgs)
    return clip_img_encodings

clip_img_encodings = get_img_encodings(imgs)
clip_img_encodings

# Definir descripciones de texto en ingles y codificarlas.
text_list = [
    "A zebra with black and white stripes",
    "A brown cat on the bed",
    "A white dog lying on the green grass"
]

text_tokens = clip.tokenize(text_list).to(device)
clip_text_encodings = clip_model.encode_text(text_tokens).float()
clip_text_encodings

# Normalizar encodings
clip_img_encodings /= clip_img_encodings.norm(dim=-1, keepdim=True)
clip_text_encodings /= clip_text_encodings.norm(dim=-1, keepdim=True)

#Calcular similitud
n_imgs = len(imgs)
n_text = len(text_list)

repeated_clip_text_encodings = clip_text_encodings.repeat(n_imgs, 1)
repeated_clip_img_encoding = clip_img_encodings.repeat_interleave(n_text, dim=0)
similarity = (repeated_clip_text_encodings * repeated_clip_img_encoding).sum(-1)
similarity = torch.unflatten(similarity, 0, (n_text, n_imgs))

# Mostrar matriz de similitud
fig = plt.figure(figsize=(10, 10))
gs = fig.add_gridspec(2, 3, wspace=.1, hspace=0)

for i, img in enumerate(imgs):
    ax = fig.add_subplot(gs[0, i])
    ax.axis("off")
    plt.imshow(img)

ax = fig.add_subplot(gs[1, :])
plt.imshow(similarity.detach().cpu().numpy().T, vmin=0.1, vmax=0.3)

labels = [ '\n'.join(wrap(text, 20)) for text in text_list ]
plt.yticks(range(n_text), labels, fontsize=10)
plt.xticks([])

for x in range(similarity.shape[1]):
    for y in range(similarity.shape[0]):
        plt.text(x, y, f"{similarity[x, y]:.2f}", ha="center", va="center", size=12)

plt.show()
```



5.2 Un conjunto de datos CLIP

En el cuaderno anterior, usamos la categoría de flores como etiqueta. Esta vez, vamos a usar las codificaciones CLIP como nuestra etiqueta.

Si el objetivo de CLIP es alinear las codificaciones de texto con las codificaciones de imagen, ¿necesitamos una descripción de texto para cada una de las imágenes en nuestro conjunto de datos? Hipótesis: no necesitamos descripciones de texto y solo necesitamos las codificaciones CLIP de imagen para crear una secuencia de conversión de texto a imagen.

Para probar esto, agreguemos las codificaciones CLIP como la "etiqueta" a nuestro conjunto de datos. Ejecutar CLIP en cada lote de imágenes aumentadas con datos sería más preciso, pero también es más lento. Podemos acelerar las cosas preprocesando y almacenando las codificaciones con anticipación.

Podemos usar glob para enumerar todas las rutas de archivo de nuestras imágenes:

```
data_paths = glob.glob(DATA_DIR + '*/*.jpg', recursive=True)
data_paths[:5]
```

```
['data/cropped_flowers/sunflowers/4755705724_976621a1e7.jpg',
 'data/cropped_flowers/sunflowers/18972803569_1a0634f398_m.jpg',
 'data/cropped_flowers/sunflowers/1008566138_6927679c8a.jpg',
 'data/cropped_flowers/sunflowers/5004121118_e9393e60d0_n.jpg',
 'data/cropped_flowers/sunflowers/5028817729_f04d32bac8_n.jpg']
```

El siguiente bloque de código ejecuta el siguiente bucle para cada ruta de archivo:

- Abrir la imagen asociada con la ruta y almacenarla en `img`
- Preprocesar la imagen, buscar la codificación CLIP y almacenarla en `clip_img`
- Convertir la codificación CLIP de un tensor a una lista de Python
- Almacenar la ruta de archivo y la codificación CLIP como una fila en un archivo csv

```
import csv

csv_path = 'clip.csv'

with open(csv_path, 'w', newline='') as csvfile:
    writer = csv.writer(csvfile, delimiter=',')
    for idx, path in enumerate(data_paths):
        img = Image.open(path)
        clip_img = torch.tensor(np.stack([clip_preprocess(img)])).to(device)
        label = clip_model.encode_image(clip_img)[0].tolist()
        writer.writerow([path] + label)
```

Puede que el procesamiento del conjunto de datos completo tarde unos segundos. Cuando haya terminado, abra clip.csv para ver los resultados.

Podemos utilizar las mismas transformaciones de imágenes que utilizamos con el otro cuaderno:

```
IMG_SIZE = 32 # Due to stride and pooling, must be divisible by 2 multiple times
IMG_CH = 3
BATCH_SIZE = 128
INPUT_SIZE = (IMG_CH, IMG_SIZE, IMG_SIZE)

pre_transforms = [
    transforms.Resize(IMG_SIZE),
    transforms.ToTensor(),  # Scales data into [0,1]
    transforms.Lambda(lambda t: (t * 2) - 1)  # Scale between [-1, 1]
]
pre_transforms = transforms.Compose(pre_transforms)
random_transforms = [
    transforms.RandomCrop(IMG_SIZE),
    transforms.RandomHorizontalFlip(),
]
random_transforms = transforms.Compose(random_transforms)
```

A continuación se muestra el código para inicializar nuestro nuevo conjunto de datos. Dado que hemos preprocesado el clip, lo cargaremos previamente en nuestra GPU con la función **init**. Hemos mantenido la codificación CLIP "sobre la marcha" como ejemplo. Producirá resultados ligeramente mejores, pero es mucho más lento.

```
class MyDataset(Dataset):
    def __init__(self, csv_path, preprocessed_clip=True):
        self.imgs = []
        self.preprocessed_clip = preprocessed_clip
        if preprocessed_clip:
            self.labels = torch.empty(
                len(data_paths), CLIP_FEATURES, dtype=torch.float, device=device
            )

        with open(csv_path, newline='') as csvfile:
            reader = csv.reader(csvfile, delimiter=',')
            for idx, row in enumerate(reader):
                img = Image.open(row[0])
                self.imgs.append(pre_transforms(img).to(device))
                if preprocessed_clip:
                    label = [float(x) for x in row[1:]]
                    self.labels[idx, :] = torch.FloatTensor(label).to(device)

    def __getitem__(self, idx):
        img = random_transforms(self.imgs[idx])
        if self.preprocessed_clip:
            label = self.labels[idx]
        else:
            batch_img = img[None, :, :, :]
            encoded_imgs = clip_model.encode_image(clip_preprocess(batch_img))
            label = encoded_imgs.to(device).float()[0]
        return img, label

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


train_data = MyDataset(csv_path)
dataloader = DataLoader(train_data, batch_size=BATCH_SIZE, shuffle=True, drop_last=True)
```

El modelo U-Net es la misma arquitectura que la última vez, pero con una pequeña diferencia. En lugar de utilizar la cantidad de clases como `c_embed_dim`, utilizaremos la cantidad de `CLIP_FEATURES`. La última vez, `c` podría haber significado "clase", pero esta vez, significa "contexto". Afortunadamente, ambos comienzan con `c`, por lo que no necesitamos refactorizar el código para reflejar este cambio en la intención.

```
T = 400
B_start = 0.0001
B_end = 0.02
B = torch.linspace(B_start, B_end, T).to(device)

ddpm = ddpm_utils.DDPM(B, device)
model = UNet_utils.UNet(
    T, IMG_CH, IMG_SIZE, down_chs=(256, 256, 512), t_embed_dim=8, c_embed_dim=CLIP_FEATURES
)
print("Num params: ", sum(p.numel() for p in model.parameters()))
model_flowers = torch.compile(model.to(device))
```

```
Num params:  44900355
```

The `get_context_mask` function will change a little bit. Since we're replacing our categorical input with a CLIP embedding, we no longer need to one-hot encode our label. We'll still randomly set values in our encoding to `0` to help the model learn without context.

```
def get_context_mask(c, drop_prob):
    c_mask = torch.bernoulli(torch.ones_like(c).float() - drop_prob).to(device)
    return c_mask
```

Recreemos también la función `sample_flowers`. Esta vez, tomará nuestra `text_list` como parámetro y la convertirá a una codificación CLIP. La función `sample_w` sigue siendo prácticamente la misma y se ha movido al final de ddpm_utils.py.

```
def sample_flowers(text_list):
    text_tokens = clip.tokenize(text_list).to(device)
    c = clip_model.encode_text(text_tokens).float()
    x_gen, x_gen_store = ddpm_utils.sample_w(model, ddpm, INPUT_SIZE, T, c, device)
    return x_gen, x_gen_store
```

¡Es hora de entrenar! Después de aproximadamente 50 épocas, el modelo comenzará a generar algo reconocible y, en 100, alcanzará su máximo potencial. ¿Qué opinas? ¿Las imágenes generadas coinciden con tus descripciones?

```
epochs=100
c_drop_prob = 0.1
lrate = 1e-4
save_dir = "05_images/"


optimizer = torch.optim.Adam(model.parameters(), lr=lrate)

model.train()
for epoch in range(epochs):
    for step, batch in enumerate(dataloader):
        optimizer.zero_grad()
        t = torch.randint(0, T, (BATCH_SIZE,), device=device).float()
        x, c = batch
        c_mask = get_context_mask(c, c_drop_prob)
        loss = ddpm.get_loss(model_flowers, x, t, c, c_mask)
        loss.backward()
        optimizer.step()

    print(f"Epoch {epoch} | Step {step:03d} | Loss: {loss.item()}")
    if epoch % 5 == 0 or epoch == int(epochs - 1):
        x_gen, x_gen_store = sample_flowers(text_list)
        grid = make_grid(x_gen.cpu(), nrow=len(text_list))
        save_image(grid, save_dir + f"image_ep{epoch:02}.png")
        print("saved images in " + save_dir + f" for episode {epoch}")
```

```
Epoch 92 | Step 008 | Loss: 0.09365662932395935
Epoch 93 | Step 008 | Loss: 0.08323424309492111
Epoch 94 | Step 008 | Loss: 0.08237650990486145
Epoch 95 | Step 008 | Loss: 0.07228102535009384
saved images in 05_images/ for episode 95
Epoch 96 | Step 008 | Loss: 0.0799870491027832
Epoch 97 | Step 008 | Loss: 0.07181591540575027
Epoch 98 | Step 008 | Loss: 0.09468146413564682
Epoch 99 | Step 008 | Loss: 0.08761724829673767
saved images in 05_images/ for episode 99
```

Ahora que el modelo está entrenado, ¡juguemos con él! ¿Qué sucede cuando le damos un mensaje de algo que no está en el conjunto de datos?

- El modelo intentará generar una imagen basada en la interpretación más cercana de ese mensaje.

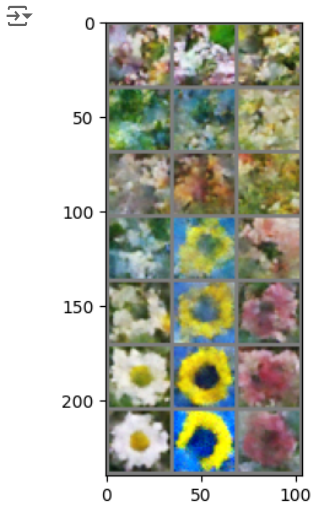
¿O puedes crear el mensaje perfecto para generar una imagen que puedas imaginar?

- Los resultados pueden variar en precisión y relevancia, dependiendo de qué tan bien el modelo pueda extrapolar de datos similares que ha visto durante el entrenamiento.

El arte de crear un mensaje para obtener los resultados que deseas se llama **ingeniería de mensajes** y, como se muestra aquí, depende del tipo de datos con los que se entrena el modelo.

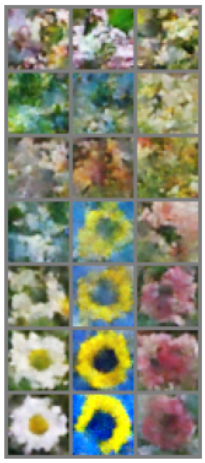
```
# Change me
text_list = [
    "A daisy",
    "A sunflower",
    "A rose"
]
```

```
model.eval()
x_gen, x_gen_store = sample_flowers(text_list)
grid = make_grid(x_gen.cpu(), nrow=len(text_list))
other_utils.show_tensor_image([grid])
plt.show()
```



Una vez que haya encontrado un conjunto de imágenes que le gusten, ejecute la celda a continuación para convertirla en una animación. Se guardará en 05_images/flowers.gif

```
grids = [other_utils.to_image(make_grid(x_gen.cpu(), nrow=len(text_list))) for x_gen in x_gen_store]
other_utils.save_animation(grids, "05_images/flowers.gif")
```



Ejercicio 5

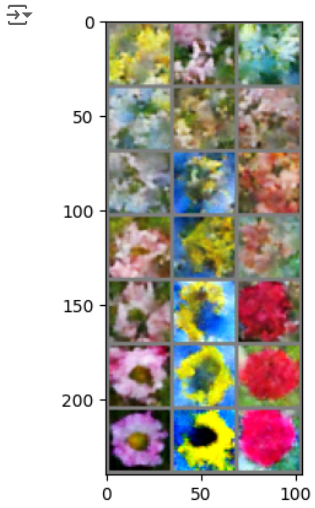
Cree nuevas celdas en las que genere 3 flores a partir de texto en inglés (A daisy..., A sunflower..., A rose...) pero con características especiales.

Ejemplo (no vale usar el mismo):

```
 text_list = [
     "A daisy with big petals",
     "A sunflower with small center",
     "A rose with dark tones"
 ]
```

```
text_list = [
    "A daisy with pink petals in half",
    "A bright sunflower with a light blue background",
    "A rose with red edges"
]
```

```
model.eval()
x_gen, x_gen_store = sample_flowers(text_list)
grid = make_grid(x_gen.cpu(), nrow=len(text_list))
other_utils.show_tensor_image([grid])
plt.show()
```



5.3 Siguientes pasos

1 celda oculta