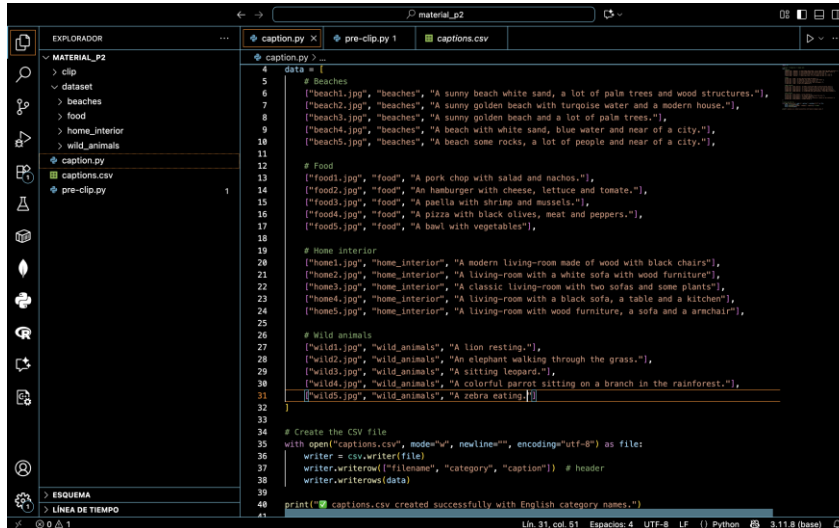


## Parte 1: dataset

Las 4 categorías que elegí fueron: “beaches”, “wild\_animals”, “food” y “home interior” y añadía 5 imágenes a cada una.



```
4 data = []
5 # Beaches
6 ["beach1.jpg", "beaches", "A sunny beach white sand, a lot of palm trees and wood structures."],
7 ["beach2.jpg", "beaches", "A sunny golden beach with turquoise water and a modern house."],
8 ["beach3.jpg", "beaches", "A sunny golden beach and a lot of palm trees."],
9 ["beach4.jpg", "beaches", "A beach with white sand, blue water and near of a city."],
10 ["beach5.jpg", "beaches", "A beach some rocks, a lot of people and near of a city."],
11
12 # Food
13 ["food1.jpg", "food", "A pork chop with salad and nachos."],
14 ["food2.jpg", "food", "An hamburger with cheese, lettuce and tomato."],
15 ["food3.jpg", "food", "A paella with shrimp and mussels."],
16 ["food4.jpg", "food", "A pizza with black olives, meat and peppers."],
17 ["food5.jpg", "food", "A bowl with vegetables"],
18
19 # Home interior
20 ["home1.jpg", "home_interior", "A modern living-room made of wood with black chairs"],
21 ["home2.jpg", "home_interior", "A living-room with a white sofa with wood furniture"],
22 ["home3.jpg", "home_interior", "A classic living-room with two sofas and some plants"],
23 ["home4.jpg", "home_interior", "A living-room with a black sofa, a table and a kitchen"],
24 ["home5.jpg", "home_interior", "A living-room with wood furniture, a sofa and a armchair"],
25
26 # Wild animals
27 ["wild1.jpg", "wild_animals", "A lion resting."],
28 ["wild2.jpg", "wild_animals", "An elephant walking through the grass."],
29 ["wild3.jpg", "wild_animals", "A sitting leopard."],
30 ["wild4.jpg", "wild_animals", "A colorful parrot sitting on a branch in the rainforest."],
31 ["wild5.jpg", "wild_animals", "A zebra eating."],
32
33
34 # Create the CSV file
35 with open("captions.csv", mode="w", newline="", encoding="utf-8") as file:
36     writer = csv.writer(file)
37     writer.writerow(["filename", "category", "caption"]) # header
38     writer.writerows(data)
39
40 print("captions.csv created successfully with English category names.")
```

Luego realicé este script en python para generar un csv con una pequeña descripción de cada imagen.

```
filename,category,caption
beach1.jpg,beaches,"A sunny beach white sand, a lot of palm trees and wood structures."
beach2.jpg,beaches,"A sunny golden beach with turquoise water and a modern house.
beach3.jpg,beaches,"A sunny golden beach and a lot of palm trees.
beach4.jpg,beaches,"A beach with white sand, blue water and near of a city."
beach5.jpg,beaches,"A beach some rocks, a lot of people and near of a city."
food1.jpg,food,"A pork chop with salad and nachos.
food2.jpg,food,"An hamburger with cheese, lettuce and tomato."
food3.jpg,food,"A paella with shrimp and mussels.
food4.jpg,food,"A pizza with black olives, meat and peppers."
food5.jpg,food,"A bowl with vegetables"
home1.jpg,home_interior,"A modern living-room made of wood with black chairs
home2.jpg,home_interior,"A living-room with a white sofa with wood furniture
home3.jpg,home_interior,"A classic living-room with two sofas and some plants
home4.jpg,home_interior,"A living-room with a black sofa, a table and a kitchen"
home5.jpg,home_interior,"A living-room with wood furniture, a sofa and a armchair"
wild1.jpg,wild_animals,"A lion resting.
wild2.jpg,wild_animals,"An elephant walking through the grass.
wild3.jpg,wild_animals,"A sitting leopard.
wild4.jpg,wild_animals,"A colorful parrot sitting on a branch in the rainforest.
wild5.jpg,wild_animals,"A zebra eating.
```

## Parte2: Pre-CLIP

```
# -----  
# CARGAR DATOS  
# -----  
df = pd.read_csv(csv_file)  
  
# Transformaciones de imagen  
transform = transforms.Compose([  
    transforms.Resize((224, 224)),  
    transforms.ToTensor(),  
    transforms.Normalize(mean=[0.485, 0.456, 0.406],  
                           | | | | | std=[0.229, 0.224, 0.225])  
])
```

1. Normalicé y transformé las imágenes para el formato que usa ResNet50.

```
# -----  
# MODELO DE IMAGEN: ResNet50 preentrenada  
# -----  
resnet = resnet50(pretrained=True)  
resnet = torch.nn.Sequential(*(list(resnet.children())[:-1]))  
resnet.eval()  
resnet.to(device)
```

2. Utilicé el modelo ya preentrenado de pytorch de ResNet50 pero quitando la última capa para obtener embeddings puros

```
# -----  
# MODELO DE TEXTO: SentenceTransformer  
# -----  
text_model = SentenceTransformer('distiluse-base-multilingual-cased-v1', device=device)
```

3. Utilicé el modelo de texto 'distiluse-base-multilingual-cased-v1'

```
# -----  
# OBTENER EMBEDDINGS DE IMÁGENES  
# -----  
image_embeddings = []  
for idx, row in df.iterrows():  
    category = row['category']  
    filename = row['filename']  
    img_path = os.path.join(dataset_dir, category, filename)  
  
    image = Image.open(img_path).convert("RGB")  
    image_tensor = transform(image).unsqueeze(0).to(device)  
  
    with torch.no_grad():  
        emb = resnet(image_tensor) # shape: (1, 2048, 1, 1)  
        emb = emb.flatten(1) # shape: (1, 2048)  
        emb = F.normalize(emb, p=2, dim=1) # normalizar  
        image_embeddings.append(emb)  
  
image_embeddings = torch.cat(image_embeddings, dim=0) # shape: (num_images, 2048)  
  
# -----  
# OBTENER EMBEDDINGS DE TEXTOS  
# -----  
captions = df['caption'].tolist()  
text_embeddings = text_model.encode(captions, convert_to_tensor=True, normalize_embeddings=True)
```

4. Normalicé embeddings de imágenes y textos.

```
# -----
# CALCULAR SIMILITUD COSENO
# -----
# Nota: ResNet y SentenceTransformer tienen dimensiones distintas, por simplicidad,
# podemos calcular similitud coseno usando la misma librería, ajustando dimensiones si es necesario
# Para este ejemplo, reducimos imagen a 512 dimensiones con PCA simple opcional
# o usamos torch.nn.functional.cosine_similarity en pares de vectores
# Aquí vamos a hacer un truco simple: proyectar imagen embeddings a 512 con una capa lineal
proj = torch.nn.Linear(image_embeddings.size(1), text_embeddings.size(1)).to(device)
with torch.no_grad():
    image_proj = F.normalize(proj(image_embeddings), p=2, dim=1) # shape: (num_images, 512)

# Similitud coseno
similarities = image_proj @ text_embeddings.T # shape: (num_images, num_texts)
```

5. Proyecté los embeddings de imagen a la dimensión de texto (512) para poder comparar con cosine similarity.

```
# -----
# EVALUACIÓN TOP-1
# -----
top1_correct = 0
for i in range(similarities.size(0)):
    top_idx = similarities[i].argmax().item()
    if top_idx == i:
        top1_correct += 1

top1_acc = top1_correct / similarities.size(0)
print(f"Top-1 accuracy (image matches its caption): {top1_acc*100:.2f}%")
```

6. Calculé la Top-1 accuracy, es decir, cuántas imágenes matchean con su caption correcto.

```
# -----
# Mostrar tabla de similitudes
# -----
sim_matrix = similarities.cpu().numpy()
sim_df = pd.DataFrame(sim_matrix, index=df['filename'], columns=df['filename'])
print("\nCosine similarity matrix (images x captions):")
print(sim_df.round(3))
```

7. Imprime la matriz de similitud, útil para ver qué imágenes están más cercanas a qué captions.

filename	beach1.jpg	beach2.jpg	beach3.jpg	beach4.jpg	...	wild2.jpg	wild3.jpg	wild4.jpg	wild5.jpg
filename					...				
beach1.jpg	-0.027	-0.000	-0.003	-0.004	...	0.005	0.000	-0.000	-0.049
beach2.jpg	-0.010	-0.000	0.013	-0.042	...	0.018	0.036	0.020	-0.048
beach3.jpg	0.005	-0.000	0.017	-0.032	...	0.018	0.023	-0.002	-0.035
beach4.jpg	-0.019	-0.020	-0.012	0.005	...	-0.014	0.003	0.002	-0.036
beach5.jpg	-0.032	-0.027	-0.020	-0.019	...	-0.012	-0.002	0.005	-0.049
food1.jpg	-0.025	-0.037	0.009	-0.016	...	-0.008	0.077	0.030	-0.047
food2.jpg	-0.057	-0.060	-0.045	-0.060	...	-0.039	0.023	0.005	-0.033
food3.jpg	-0.011	-0.049	-0.010	-0.012	...	-0.005	0.044	0.012	-0.066
food4.jpg	-0.009	-0.042	-0.006	-0.016	...	-0.018	0.027	0.024	-0.047
food5.jpg	-0.034	-0.031	-0.025	-0.008	...	-0.032	0.021	0.018	-0.070
home1.jpg	-0.059	-0.059	-0.062	-0.014	...	-0.061	-0.061	-0.046	-0.104
home2.jpg	-0.074	-0.015	-0.071	-0.019	...	-0.032	-0.022	-0.062	-0.080
home3.jpg	-0.055	-0.021	-0.069	0.003	...	-0.033	-0.027	-0.038	-0.088
home4.jpg	-0.029	-0.009	-0.041	0.016	...	-0.031	-0.013	-0.054	-0.096
home5.jpg	-0.033	-0.014	-0.036	0.008	...	-0.053	-0.013	-0.038	-0.098
wild1.jpg	-0.005	-0.023	-0.018	-0.007	...	-0.053	-0.019	-0.004	-0.089
wild2.jpg	-0.038	-0.029	-0.010	-0.012	...	-0.005	0.002	0.022	-0.053
wild3.jpg	-0.022	0.001	-0.004	-0.031	...	-0.026	0.019	-0.019	-0.088
wild4.jpg	-0.018	-0.000	-0.018	-0.030	...	-0.022	0.015	0.014	-0.073
wild5.jpg	-0.017	-0.019	-0.035	-0.036	...	-0.038	-0.009	0.005	-0.065

Esta sería la matriz de las distancias coseno. Los valores oscilan entre negativos y positivos muy bajos (-0.1 ... 0.1). No hay un patrón claro de “imagen coincide con su caption”: los máximos no están en la diagonal. Esto refuerza que los embeddings de imagen y texto no están directamente comparables, a menos que hagas un proyecto de alineación (como lo hace CLIP).

La Top-1 accuracy me da 5.00%. Esto significa que solo 1 de cada 20 imágenes coincide correctamente con su caption según la similitud coseno. Cómo el valor es tan bajo, la red de imagen y la de texto no están alineadas, porque cada una genera embeddings en espacios distintos.

El resultado demuestra la limitación que hay al utilizar modelos separados para imágenes y texto.

## Parte 3: Clip

1. Cargué cada imagen y su caption al igual que en el preclip

```
# -----  
# CARGAR MODELO CLIP  
# -----  
model_name = "openai/clip-vit-base-patch32"  
model = CLIPModel.from_pretrained(model_name).to(device)  
processor = CLIPProcessor.from_pretrained(model_name)
```

Usé CLIPProcessor para procesar imagen y texto simultáneamente.

```
# -----  
# EMBEDDINGS DE IMÁGENES Y TEXTOS  
# -----  
image_embeddings = []  
text_embeddings = []  
  
for idx, row in df.iterrows():  
    # Imagen  
    category = row['category']  
    filename = row['filename']  
    img_path = os.path.join(dataset_dir, category, filename)  
    image = Image.open(img_path).convert("RGB")  
  
    # Procesar imagen y caption  
    inputs = processor(text=row['caption'], images=image, return_tensors="pt", padding=True).to(device)  
  
    with torch.no_grad():  
        outputs = model(**inputs)  
        # image_embedding: [1, 512], text_embedding: [1, 512]  
        image_emb = F.normalize(outputs.image_embs, p=2, dim=1)  
        text_emb = F.normalize(outputs.text_embs, p=2, dim=1)  
  
        image_embeddings.append(image_emb)  
        text_embeddings.append(text_emb)  
  
# Convertir a tensor  
image_embeddings = torch.cat(image_embeddings, dim=0) # (num_images, 512)  
text_embeddings = torch.cat(text_embeddings, dim=0) # (num_images, 512)
```

2. Obtuve los embeddings **normalizados** de imagen y texto. A diferencia del pre-clip, ya obtuve ambos en el mismo bucle al utilizar el mismo modelo para ambos.

```
# -----
# SIMILITUD COSENO
# -----
similarities = image_embeddings @ text_embeddings.T # (num_images, num_texts)
```

3. Calculé similitud coseno entre cada imagen y todas las captions.

```
# -----
# TOP-1 ACCURACY
# -----
top1_correct = 0
for i in range(similarities.size(0)):
    top_idx = similarities[i].argmax().item()
    if top_idx == i:
        top1_correct += 1
top1_acc = top1_correct / similarities.size(0)
print(f"Top-1 accuracy (image matches its caption with CLIP): {top1_acc*100:.2f}%")
```

4. Calculé un **Top-1 accuracy**, que indica cuántas imágenes coinciden con su caption correcto.

```
# -----
# MATRIZ DE SIMILITUD
# -----
import pandas as pd
sim_matrix = similarities.cpu().numpy()
sim_df = pd.DataFrame(sim_matrix, index=df['filename'], columns=df['filename'])
print("\nC cosine similarity matrix (images x captions) with CLIP:")
print(sim_df.round(3))
```

5. Para imprimir la matriz de similitud.

```
Cosine similarity matrix (images x captions) with CLIP:
filename  beach1.jpg  beach2.jpg  beach3.jpg  beach4.jpg  ...  wild2.jpg  wild3.jpg  wild4.jpg  wild5.jpg
filename
beach1.jpg    0.276    0.249    0.283    0.289    ...    0.141    0.133    0.109    0.156
beach2.jpg    0.240    0.257    0.259    0.282    ...    0.154    0.159    0.121    0.143
beach3.jpg    0.271    0.258    0.290    0.277    ...    0.163    0.154    0.154    0.167
beach4.jpg    0.224    0.264    0.233    0.312    ...    0.131    0.125    0.079    0.136
beach5.jpg    0.230    0.249    0.232    0.297    ...    0.125    0.133    0.042    0.132
food1.jpg     0.082    0.125    0.089    0.123    ...    0.123    0.150    0.079    0.224
food2.jpg     0.092    0.158    0.128    0.106    ...    0.122    0.113    0.089    0.192
food3.jpg     0.100    0.177    0.136    0.135    ...    0.073    0.118    0.106    0.131
food4.jpg     0.108    0.135    0.127    0.115    ...    0.111    0.149    0.111    0.204
food5.jpg     0.091    0.131    0.105    0.105    ...    0.127    0.144    0.146    0.193
home1.jpg     0.113    0.216    0.118    0.138    ...    0.121    0.152    0.084    0.158
home2.jpg     0.135    0.224    0.140    0.150    ...    0.153    0.178    0.123    0.175
home3.jpg     0.128    0.207    0.148    0.130    ...    0.123    0.168    0.111    0.152
home4.jpg     0.150    0.207    0.129    0.138    ...    0.127    0.168    0.110    0.157
home5.jpg     0.155    0.227    0.162    0.162    ...    0.128    0.156    0.079    0.165
wild1.jpg     0.098    0.130    0.121    0.141    ...    0.161    0.240    0.101    0.204
wild2.jpg     0.115    0.127    0.119    0.132    ...    0.305    0.215    0.112    0.226
wild3.jpg     0.092    0.140    0.105    0.132    ...    0.204    0.306    0.146    0.224
wild4.jpg     0.120    0.141    0.129    0.160    ...    0.221    0.208    0.101    0.338
wild5.jpg     0.124    0.157    0.144    0.159    ...    0.184    0.205    0.095    0.251
```

La diagonal tiene valores más altos ( $\approx 0.25-0.34$ ) que el resto de la matriz. Esto indica que cada imagen está más cerca de su caption correcto que de otros captions. Las similitudes fuera de la diagonal son menores, aunque algunas categorías similares (ej. playas entre sí) pueden tener valores moderadamente altos, lo cual es normal.

Top-1 accuracy (image matches its caption with CLIP): 65.00%

Tiene bastante más que el pre-clip, esto es debido a que alinea embeddings de imagen y texto en el mismo espacio.