Modelos multimodales

Clase 22

Dra. Wendy Aguilar

Modelos Generativos Profundos

UN ENFOQUE DESDE LA CREATIVIDAD COMPUTACIONAL

Modelos representativos (2022-2023)

- DALL·E 2 (OpenAI) → Codifica texto e imagen en el espacio CLIP y usa *diffusion* para la generación.
- Imagen (Google Brain) → Usa un modelo de lenguaje muy potente y un decodificador de difusión para lograr fotorealismo extremo.
- Stable Diffusion (Stability AI, CompVis, Runway) → Modelo de difusión latente, eficiente y abierto, que popularizó la generación de imágenes creativas.
- Flamingo (DeepMind) → Modelo multimodal *few-shot* capaz de razonar y generar lenguaje condicionado por imágenes o video.

CompVis

runway

Google DeepMind

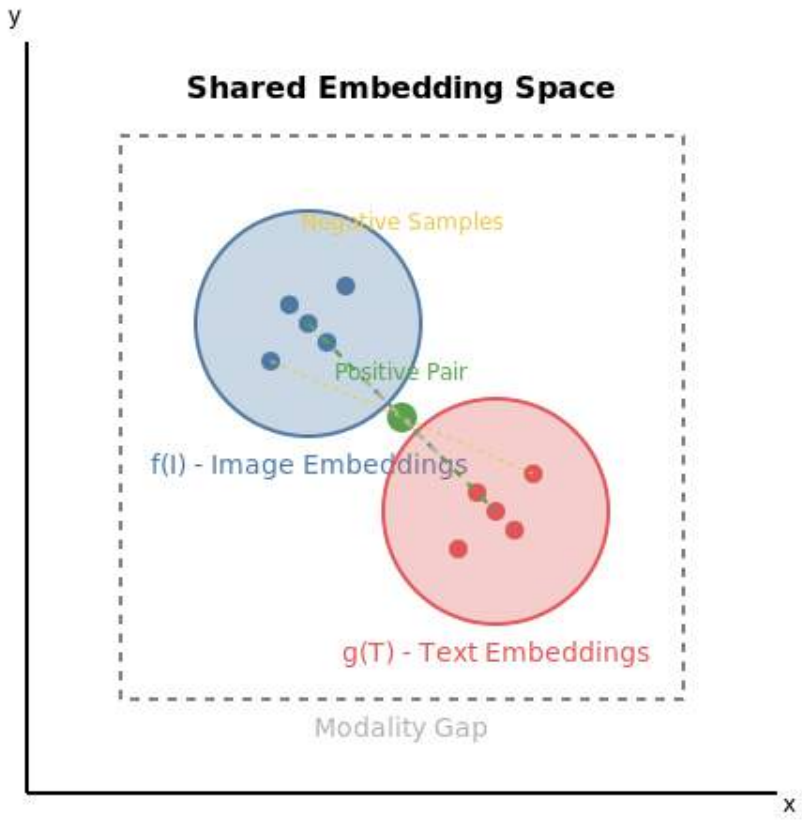
DALL-E 2

- Desarrollado por **OpenAI** para la **generación de imágenes a partir de texto** (*text-to-image generation*).
- Es la **segunda versión** de DALL·E (la primera se lanzó en **febrero de 2021**).
- Publicado en **abril de 2022**, representó un salto cualitativo en la generación multimodal.
- Despertó enorme interés en la comunidad científica y artística al mostrar que la IA puede **crear imágenes originales y coherentes** a partir de descripciones textuales complejas.

DALL-E 2

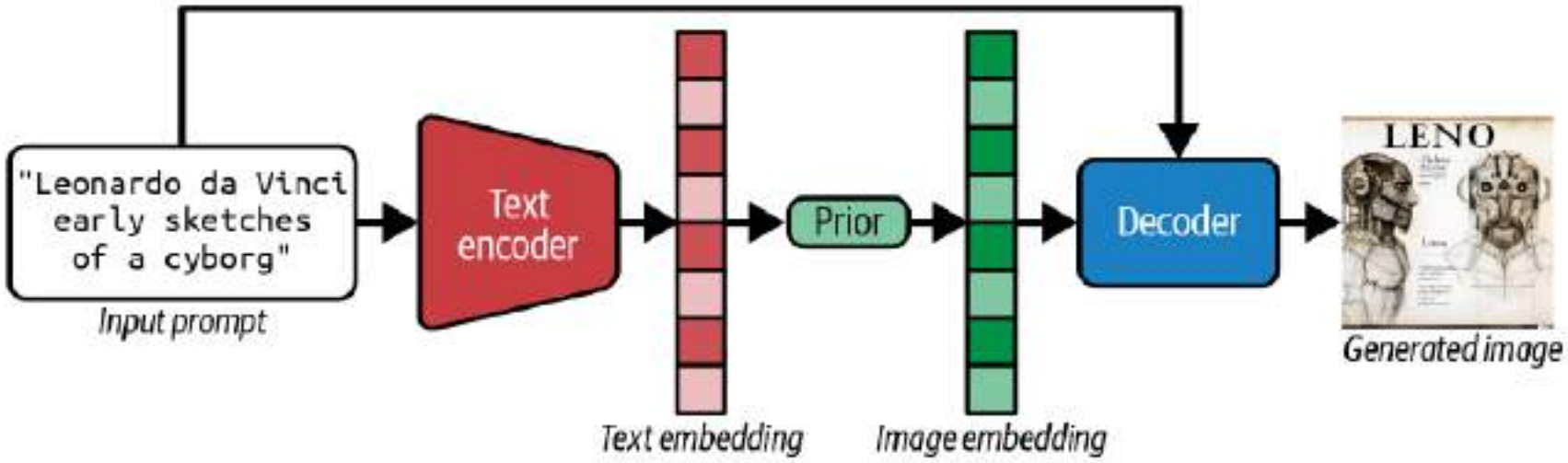
Fundamentos conceptuales

- DALL·E 2 combina **procesamiento de lenguaje natural** y **visión computacional** en un mismo marco generativo.
- Aprende una **representación compartida entre texto e imagen** (espacio latente multimodal).
- Basado en **CLIP (Contrastive Language–Image Pretraining)**:
  - CLIP aprende a asociar descripciones textuales con imágenes.
  - DALL·E 2 usa este conocimiento para **guiar la generación visual**.
- Introduce la idea de **"difusión guiada por CLIP"**: un proceso iterativo que transforma ruido en imágenes coherentes con el texto.



DALL-E 2

Arquitectura

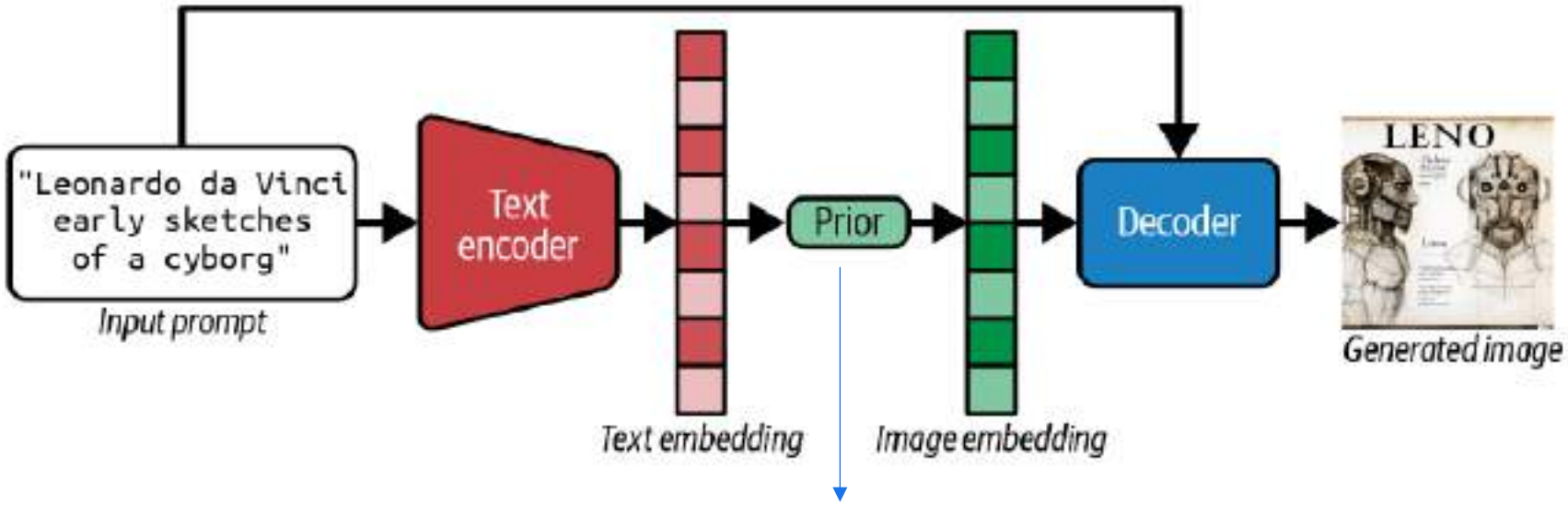


El texto se convierte en un **vector numérico de significado**, o **text embedding**.

Este vector ocupa una posición en el **espacio semántico multimodal**, donde palabras con significados similares están más cerca.

DALL-E 2

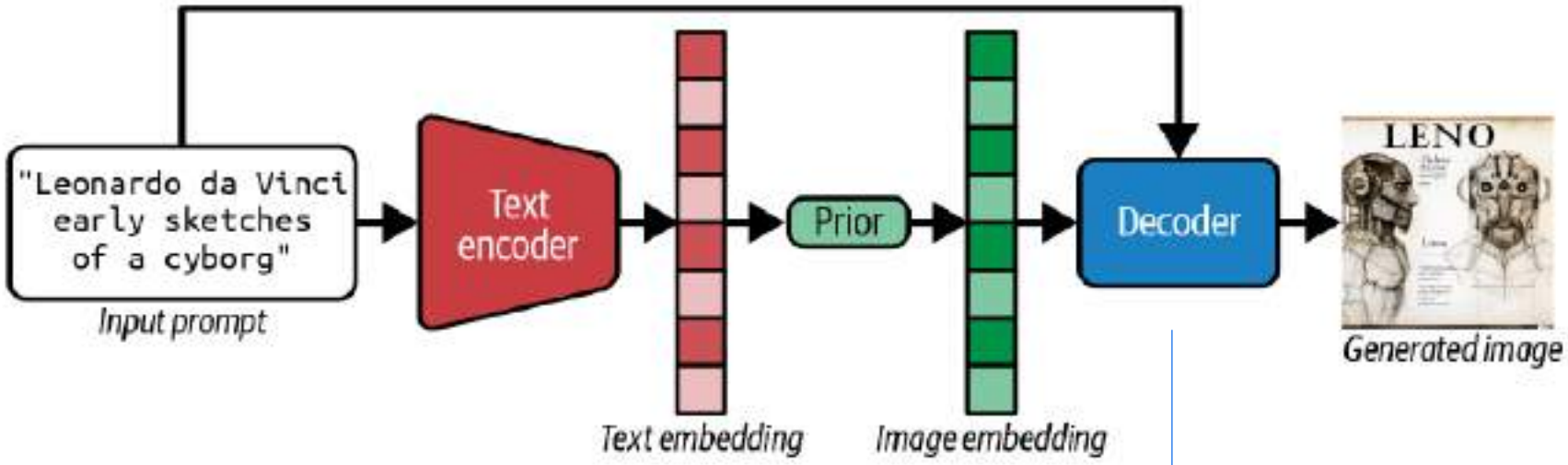
Arquitectura



- El módulo *Prior* aprende la **correspondencia estadística entre texto e imagen**.
- Traduce el *text embedding* a un *image embedding* — una representación visual latente que conserva el significado del texto.
- Aquí ocurre la **"traducción conceptual"**: el modelo infiere cómo debería "verse" el texto.

DALL-E 2

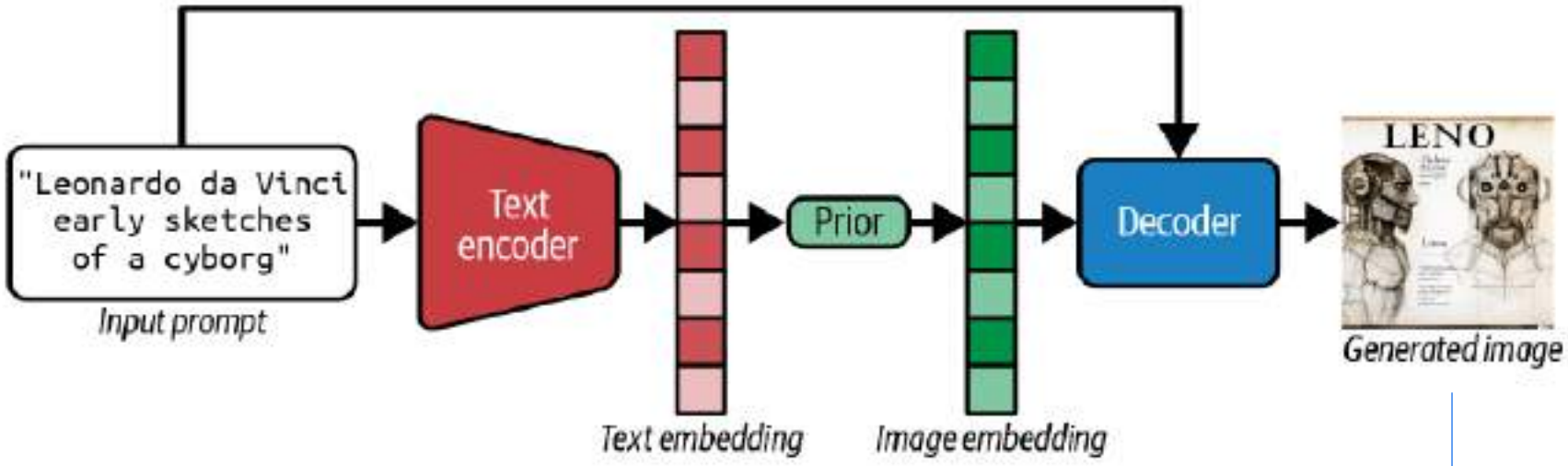
Arquitectura



- Convierte el *image embedding* en una **imagen visible** mediante un **modelo de difusión** o **autoencoder generativo**.
- Nota que recibe tanto el texto de entrada como el embedding de la imagen.

DALL-E 2

Arquitectura

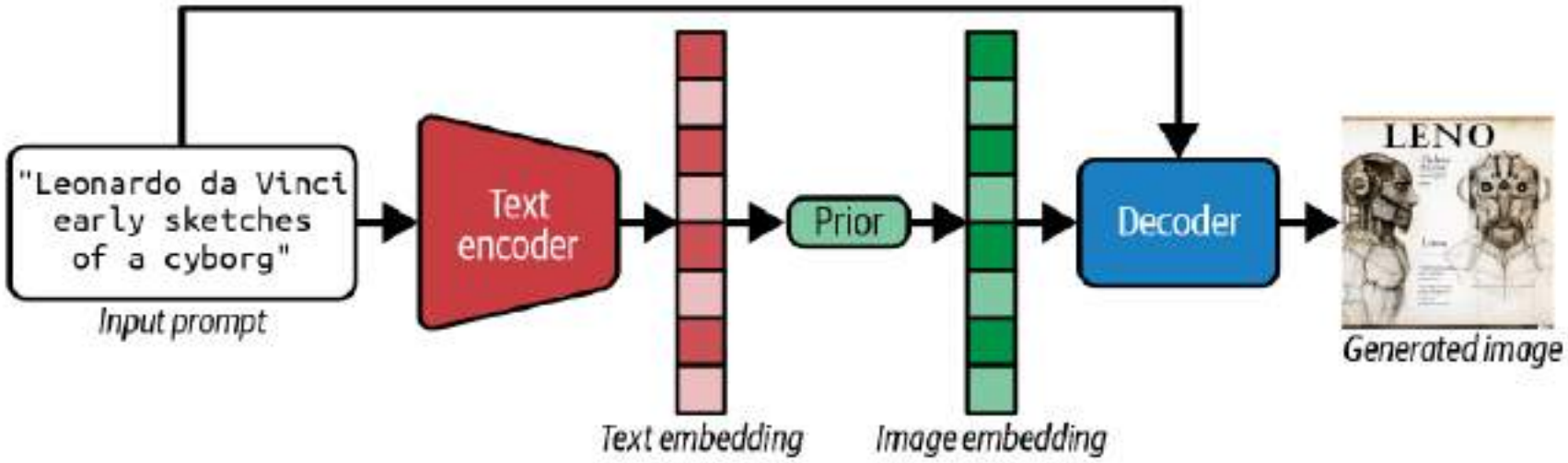


El resultado es una imagen que combina **contenido textual + conocimiento visual aprendido**.

Refleja la **capacidad creativa** de la IA al fusionar dominios conceptuales distintos.

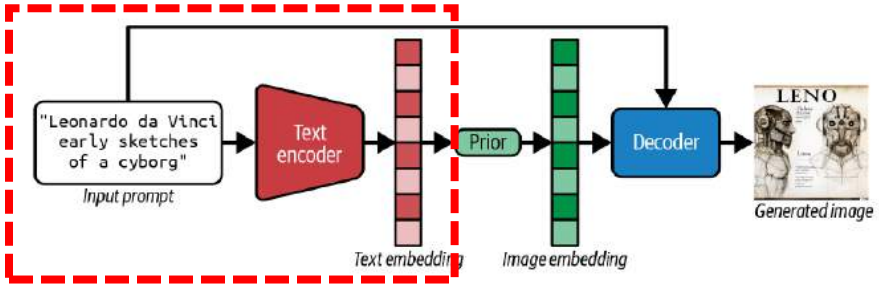
DALL-E 2

Arquitectura



Concepto clave: espacio latente compartido

- Tanto texto como imagen se proyectan en un **espacio de representación común**.
- En este espacio, las distancias reflejan **similitud semántica** entre descripciones y visuales.
- Este alineamiento entre modalidades es el corazón de los **modelos multimodales generativos**.



DALL-E 2

Text Encoder

Propósito

- **Convertir el texto de entrada (prompt)** en un **vector numérico (embedding)** que capture el **significado conceptual** del texto.
- Este vector representa el texto dentro de un **espacio latente continuo**, donde las relaciones semánticas pueden ser medidas y manipuladas.

Uso de un encoder preentrenado

- En lugar de entrenar un codificador de texto desde cero, DALL·E 2 utiliza uno ya existente: **CLIP (Contrastive Language–Image Pre-training)**.

CLIP

Contrastive Language–Image Pretraining

- Desarrollado por:

OpenAI y publicado en **febrero de 2021**, pocos días después del primer *DALL·E*.

- Su propósito:

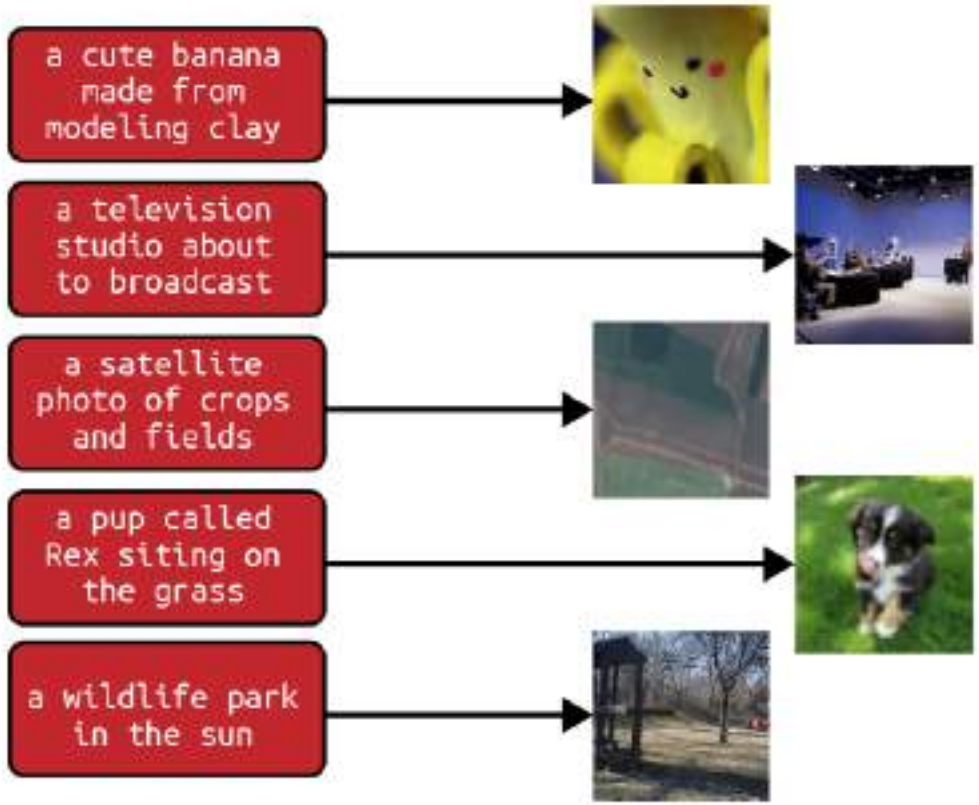
Aprender conceptos visuales a partir de lenguaje natural, sin depender de etiquetas humanas específicas.

- **Define un puente entre texto e imagen**:

Creando un espacio latente donde ambos pueden representarse y compararse.

- Entrenado con:

400 millones de pares texto–imagen obtenidos de Internet —una escala mucho mayor que conjuntos tradicionales como ImageNet (14 millones).

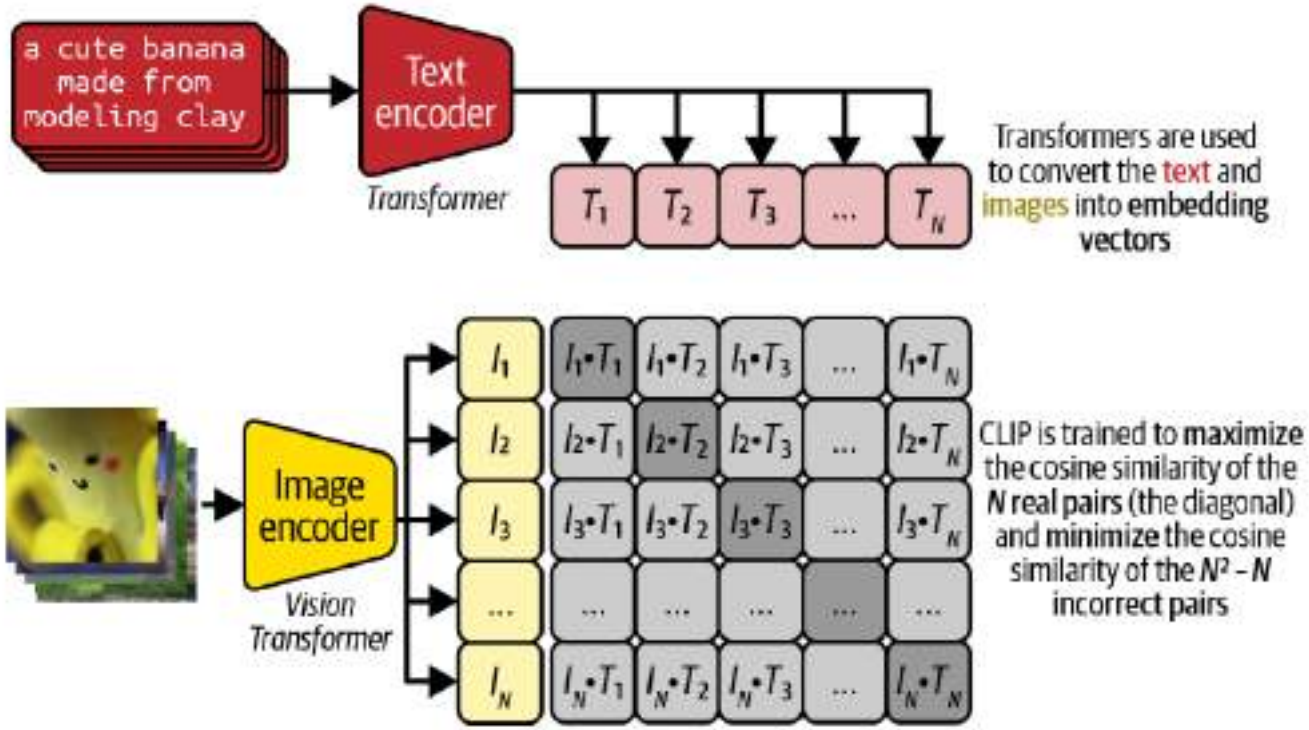


CLIP

Aprendizaje contrastivo

- CLIP aprende a **emparejar correctamente textos e imágenes**.
  - Utiliza **dos redes neuronales independientes**:
    - Un **text encoder** (basado en Transformer) para generar *text embeddings*.
    - Un **image encoder** (Vision Transformer o *ViT*) para generar *image embeddings*.
  - En cada lote de entrenamiento, se calculan las similitudes entre **todas las combinaciones** de embeddings de texto e imagen.
  - El objetivo del entrenamiento:
    Maximizar la similitud (cosine similarity) entre pares correctos.

    Minimizar la similitud entre pares incorrectos.



Este proceso crea un **espacio latente compartido** donde descripciones y visuales con el mismo significado están próximos.

CLIP

CLIP_Embeddings.ipynb

1. Instalar CLIP

```
!pip install git+https://github.com/openai/CLIP.git
```

Ejecutar:

2. Importar librerías

3. Carga de imágenes y descripciones

CLIP

CLIP_Embeddings.ipynb

4. Obtención de embeddings de texto e imagen

```
# Selección del dispositivo y carga del modelo CLIP
device = "cuda" if torch.cuda.is_available() else "cpu"
model, preprocess = clip.load("ViT-B/32", device=device) # Transformer (texto) + Vision Transformer
print("Modelo CLIP cargado en:", device)
```

```
100%|████████████████████████████████████| 338M/338M [00:02<00:00, 147MiB/s]
Modelo CLIP cargado en: cpu
```

CLIP

CLIP_Embeddings.ipynb

```
# Preprocesamiento de las imágenes
images = [preprocess(Image.open(path)).unsqueeze(0).to(device) for path in image_paths]

# Preprocesamiento de los prompts
texts = clip.tokenize(text_prompts).to(device)
```

Aplica **todas las transformaciones necesarias** para que la imagen sea compatible con el modelo CLIP:

- Redimensiona la imagen a **224×224 píxeles**.
- La convierte a **tensor de PyTorch**.
- Normaliza los valores de color (R, G, B) según los parámetros con los que CLIP fue entrenado.

Envía el tensor resul tante al **dispositivo** definido previamente (GPU o CPU).

Convierte cada frase de la lista `text_prompts` (por ejemplo, `"a cat"`, `"a dog"`) en una **secuencia de tokens numéricos** que el modelo de texto de CLIP puede interpretar.

Internamente usa:

- ftfy → corrige caracteres especiales o acentos.
- regex → separa palabras, signos, números, etc.
- un *tokenizer BPE (Byte Pair Encoding)* → convierte subpalabras en índices enteros.

- Envía el tensor de texto al mismo dispositivo que las imágenes (GPU o CPU).

```
print(text_prompts[0])
print(texts[0])
```

Imprime el texto original

Imprime una secuencia de números (tokens) en la que CLIP convierte esa frase.

- Cada número representa una subpalabra o símbolo dentro del vocabulario del modelo.

El resultado es un tensor con 77 posiciones que el transformer de texto puede procesar.

```
... a lion
tensor([49406,   320,  5567, 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,     0,     0], dtype=torch.int32)
```

CLIP

CLIP_Embeddings.ipynb


```python
# Generar los embeddings de imagen y texto con CLIP, y luego normalizarlos
# para que puedan compararse dentro del mismo espacio latente.
with torch.no_grad():
    image_features = [model.encode_image(img) for img in images]
    image_features = torch.vstack(image_features)
    text_features = model.encode_text(texts)

# Normalización
image_features /= image_features.norm(dim=-1, keepdim=True)
text_features /= text_features.norm(dim=-1, keepdim=True)
```

Pasa cada imagen por el **Vision Transformer (ViT)** de CLIP.
El resultado es un **vector de embedding** (ViT-B/32) que representa **el contenido visual semántico** de la imagen.

Une todos los vectores de imagen en una sola matriz (batch).

Pasa el conjunto de frases tokenizadas (`texts`) por el **transformer de texto** de CLIP.

Ambos tensores están en un **mismo espacio latente multimodal**, lo que permite compararlos directamente mediante medidas de similitud (como el coseno).

- Convierte cada embedding en un **vector unitario (longitud = 1)**.
- Permite comparar los vectores con **producto punto o coseno de forma equivalente**, ya que todos estarán en la **misma escala**.
- La similitud coseno será entonces directamente el **producto punto** entre los embeddings normalizados.

CLIP

CLIP_Embeddings.ipynb

5. Cálculo de similitud entre texto e imagen

```
# Calcular la similitud entre embeddings de imagen y texto
# - image_features @ text_features.T es el producto punto
# - Se multiplica por 100 para amplificar los valores de similitud antes de aplicar el softmax.
similarity = (100.0 * image_features @ text_features.T).softmax(dim=-1)
similarity = similarity.cpu().numpy()
```

Ejecutar

```
# Crear etiquetas automáticas a partir de las rutas de imagen
# (solo el nombre del archivo, sin la ruta completa)
image_labels = [path.split("/")[-1] for path in image_paths]

# Crear DataFrame de forma dinámica
df = pd.DataFrame(similarity, columns=text_prompts, index=image_labels)

# Mostrar con formato y gradiente de color
display(df.style.background_gradient(cmap="Blues").format("{:.2f}"))
```

CLIP

CLIP_Embeddings.ipynb

	a lion	a cat	a dog	a wild animal	a domestic animal	a pet
leon1.jpg	0.99	0.00	0.00	0.00	0.00	0.01
leon2.jpg	0.95	0.01	0.00	0.00	0.01	0.02
leon3.jpg	0.99	0.00	0.00	0.00	0.00	0.00
leon4.jpg	0.99	0.00	0.00	0.01	0.01	0.00
leon5.jpg	0.92	0.00	0.00	0.01	0.02	0.05
gato1.jpg	0.00	0.72	0.00	0.00	0.15	0.12
gato2.jpg	0.00	0.56	0.00	0.03	0.19	0.22
gato3.jpg	0.00	0.61	0.00	0.00	0.19	0.19
gato4.jpg	0.01	0.54	0.01	0.01	0.30	0.14
gato5.jpg	0.01	0.79	0.01	0.01	0.07	0.11
perro1.jpg	0.00	0.00	0.41	0.01	0.46	0.12
perro2.jpg	0.00	0.00	0.60	0.00	0.27	0.12
perro3.jpg	0.00	0.00	0.34	0.02	0.41	0.22
perro4.jpg	0.04	0.00	0.58	0.01	0.25	0.11
perro5.jpg	0.01	0.00	0.65	0.00	0.24	0.10

Ejecutar

6. Visualización del espacio latente con PCA

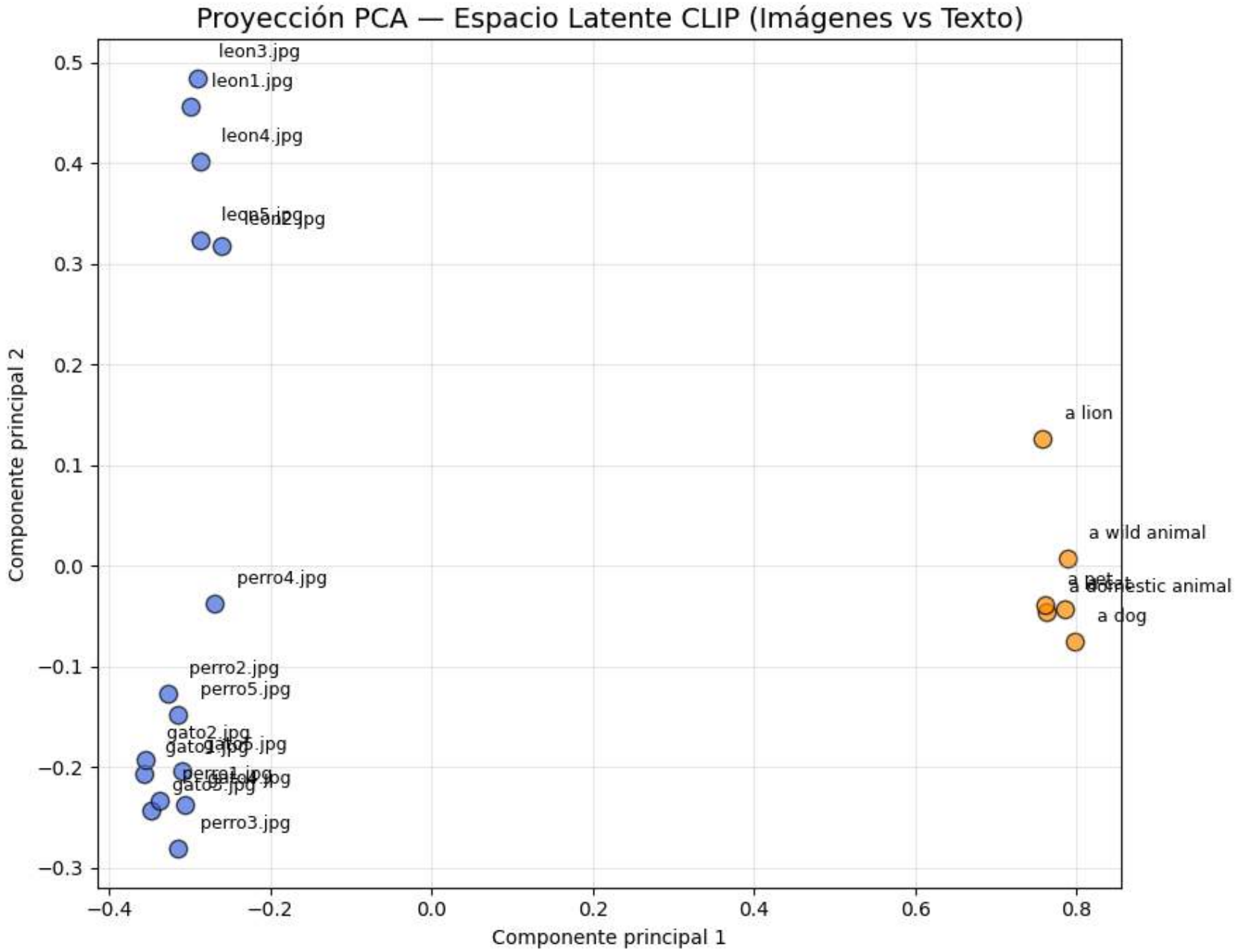
Cada punto del gráfico es un vector de características (embedding) que CLIP

una imagen (puntos azules ■), o

una descripción textual (puntos naranjas ■).

CLIP los entrena para que ambos tipos de datos —imagen y texto— vivan en u

"significan lo mismo" están cerca, aunque sean de modalidades distintas.



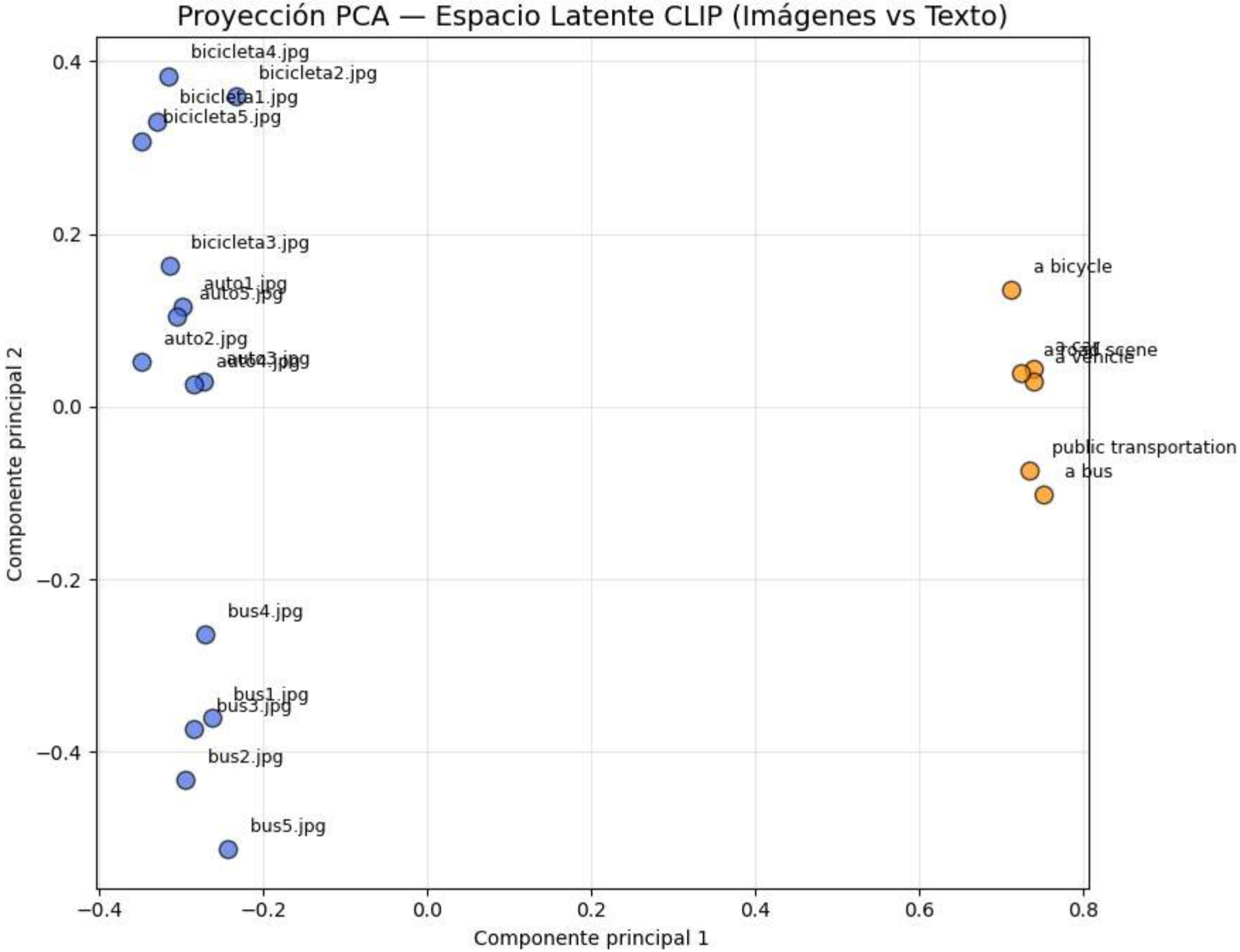
CLIP

CLIP_Embeddings.ipynb

Google colab

```
# Carga y muestra las imágenes de ejemplo (puedes reemplazarlas por otras)
image_paths, text_prompts = carga_imagenes_ejercicio2()
mostrar_imagenes(image_paths)
```

	a car	a bus	a bicycle	a vehicle	public transportation	a road scene
auto1.jpg	0.84	0.00	0.00	0.14	0.00	0.02
auto2.jpg	0.75	0.00	0.00	0.25	0.00	0.00
auto3.jpg	0.70	0.00	0.00	0.29	0.00	0.00
auto4.jpg	0.93	0.00	0.00	0.07	0.00	0.00
auto5.jpg	0.89	0.00	0.00	0.11	0.00	0.00
bus1.jpg	0.00	0.94	0.00	0.02	0.03	0.00
bus2.jpg	0.00	0.96	0.00	0.00	0.04	0.00
bus3.jpg	0.00	0.90	0.00	0.01	0.09	0.00
bus4.jpg	0.04	0.63	0.00	0.29	0.03	0.00
bus5.jpg	0.00	0.89	0.00	0.00	0.10	0.00
bicicleta1.jpg	0.00	0.00	0.96	0.03	0.00	0.00
bicicleta2.jpg	0.00	0.00	0.98	0.01	0.01	0.01
bicicleta3.jpg	0.00	0.00	0.99	0.01	0.00	0.00
bicicleta4.jpg	0.00	0.00	0.99	0.01	0.00	0.00
bicicleta5.jpg	0.00	0.00	0.99	0.01	0.00	0.00


Proyección PCA — Espacio Latente CLIP (Imágenes vs Texto)
bicicleta4.jpg
bicicleta2.jpg
bicicleta1.jpg
bicicleta5.jpg
bicicleta3.jpg
auto1.jpg
auto5.jpg
auto2.jpg
auto3.jpg
auto4.jpg
bus4.jpg
bus1.jpg
bus3.jpg
bus2.jpg
bus5.jpg
a bicycle
a car
a road scene
a vehicle
public transportation
a bus
Componente principal 2
Componente principal 1
0.4
0.2
0.0
−0.2
−0.4
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8

CLIP

CLIP_Embeddings.ipynb

Ejercicio

Crea tu propio conjunto de imágenes y descripciones

Implementar una función **`carga_imagenes_ejercicio3`** que:

- cargue un conjunto de imágenes elegidas por ti y
- una lista de descripciones (*text prompts*) asociadas,

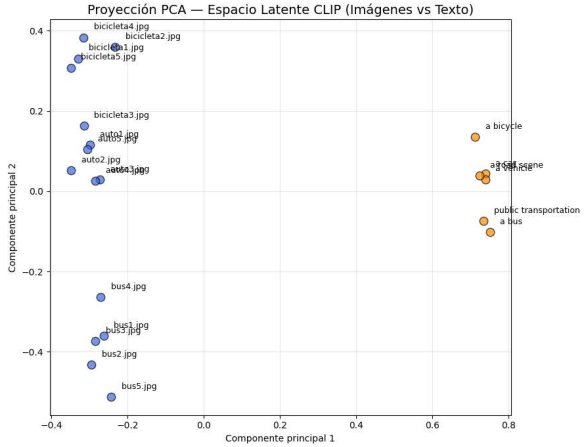
para explorar cómo CLIP representa distintos dominios semánticos.

Ejemplos:

- Comida/bebida
- Arquitectura
- Arte y diseño
- Tecnología
- Personas y profesiones
- Imágenes médicas (rayos x o de resonancia) de diferentes partes del cuerpo
- Etc.

Crea una imagen así:

	a car	a bus	a bicycle	a vehicle	public transportation	a road scene
auto1.jpg	0.84	0.00	0.00	0.14	0.00	0.02
auto2.jpg	0.75	0.00	0.00	0.25	0.00	0.00
auto3.jpg	0.70	0.00	0.00	0.29	0.00	0.00
auto4.jpg	0.93	0.00	0.00	0.07	0.00	0.00
auto5.jpg	0.89	0.00	0.00	0.11	0.00	0.00
bus1.jpg	0.00	0.94	0.00	0.02	0.03	0.00
bus2.jpg	0.00	0.96	0.00	0.00	0.04	0.00
bus3.jpg	0.00	0.90	0.00	0.01	0.09	0.00
bus4.jpg	0.04	0.63	0.00	0.29	0.03	0.00
bus5.jpg	0.00	0.89	0.00	0.00	0.10	0.00
bicicleta1.jpg	0.00	0.00	0.96	0.03	0.00	0.00
bicicleta2.jpg	0.00	0.00	0.98	0.01	0.01	0.01
bicicleta3.jpg	0.00	0.00	0.99	0.01	0.00	0.00
bicicleta4.jpg	0.00	0.00	0.99	0.01	0.00	0.00
bicicleta5.jpg	0.00	0.00	0.99	0.01	0.00	0.00



Enviarla a wendysan@hotmail.com

Poner a ejecutar todo el notebook:

DALLE2_Decoder_Kandinsky.ipynb

¿CLIP es generativo?

CLIP

No es generativo

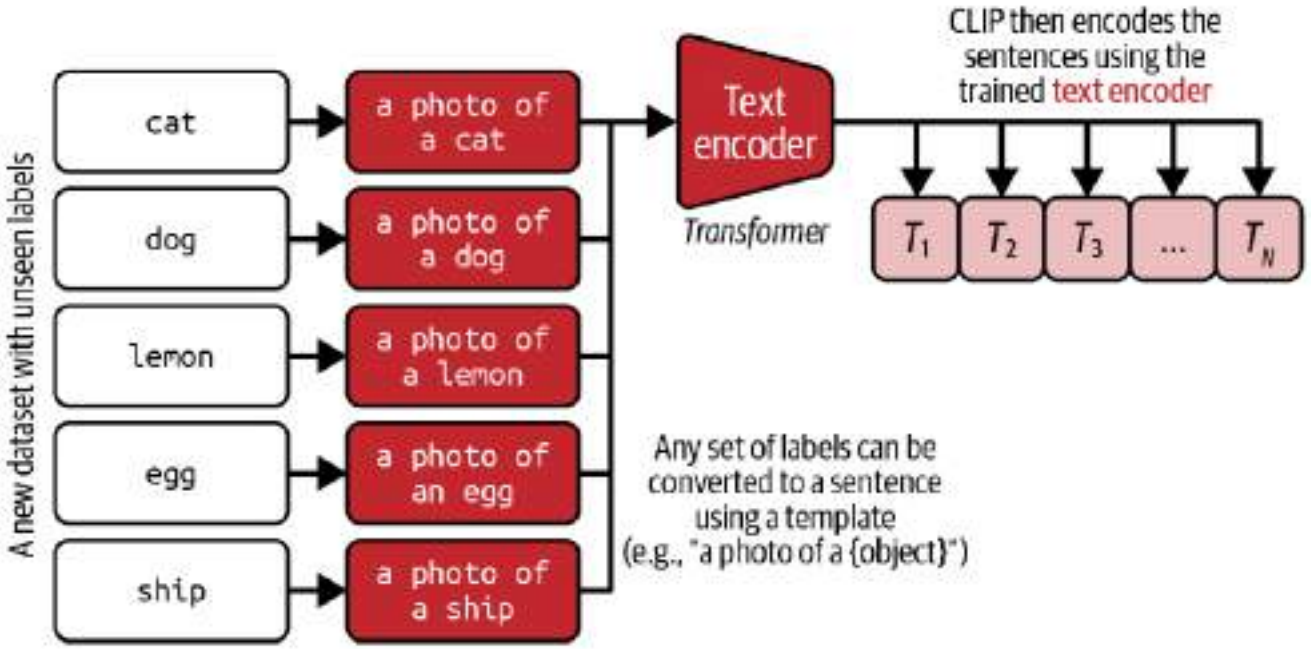
- CLIP **no genera** ni texto ni imágenes.
- Es un modelo **discriminativo**: predice qué texto corresponde mejor a una imagen (o viceversa).
- Su salida es una **comparación de similitudes**, no una nueva muestra generada.
- En esencia, aprende el idioma común entre palabras e imágenes.

… pero, enseña a otros modelos —como DALL·E 2— a entender cómo una descripción lingüística puede transformarse en una representación visual coherente."

CLIP

Capacidad de generaliación: Zero-Shot Learning

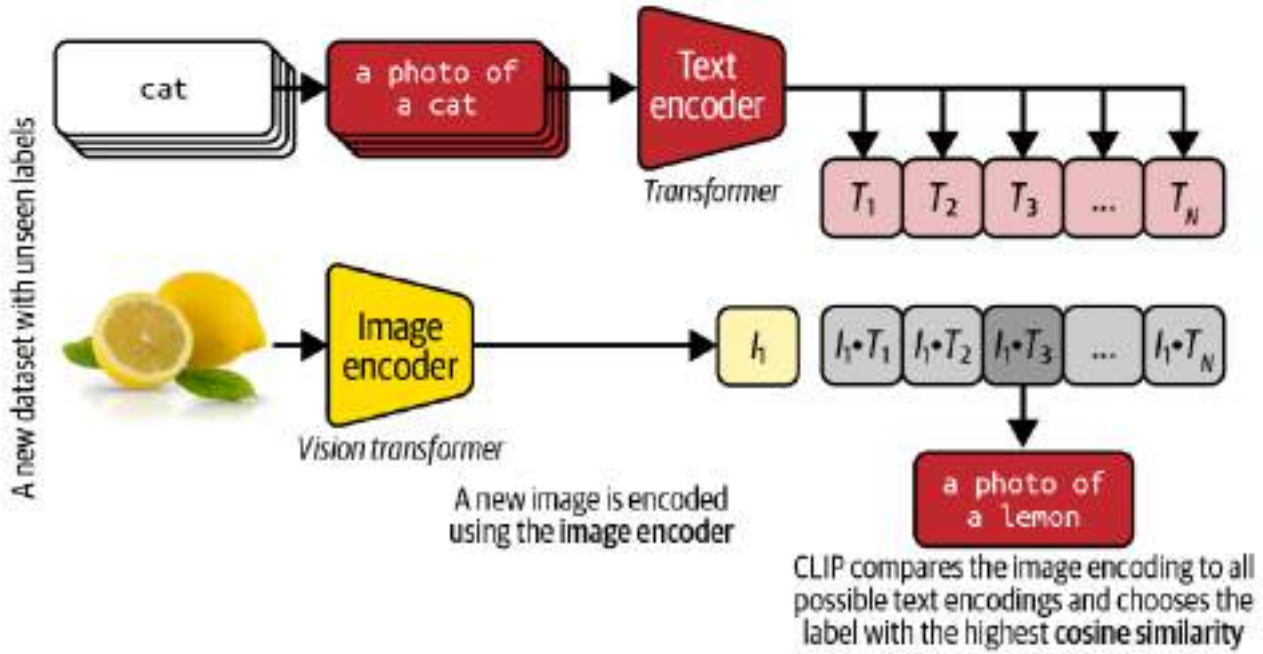
- CLIP puede **reconocer imágenes en tareas nunca vistas** sin reentrenarse.
    - Ejemplo: para clasificar imágenes de *ImageNet*, se usan templates para transformar las etiquetas en frases como (“a photo of a <label>”).



CLIP

Capacidad de generaliación: Zero-Shot Learning

- Luego, para predecir la etiqueta de una imagen:
  - La pasamos por el codificador de imágenes de CLIP y calculamos la similitud coseno entre el embedding de la imagen y todos los posibles embeddings de texto
  - Selecciona la de **mayor similitud**.
- Nota que no necesitamos entrenar ninguna de las dos redes para que pueda aplicarse a tareas nuevas.
- Usa el lenguaje como el dominio común a través del cual cualquier conjunto de etiquetas se puede expresar.



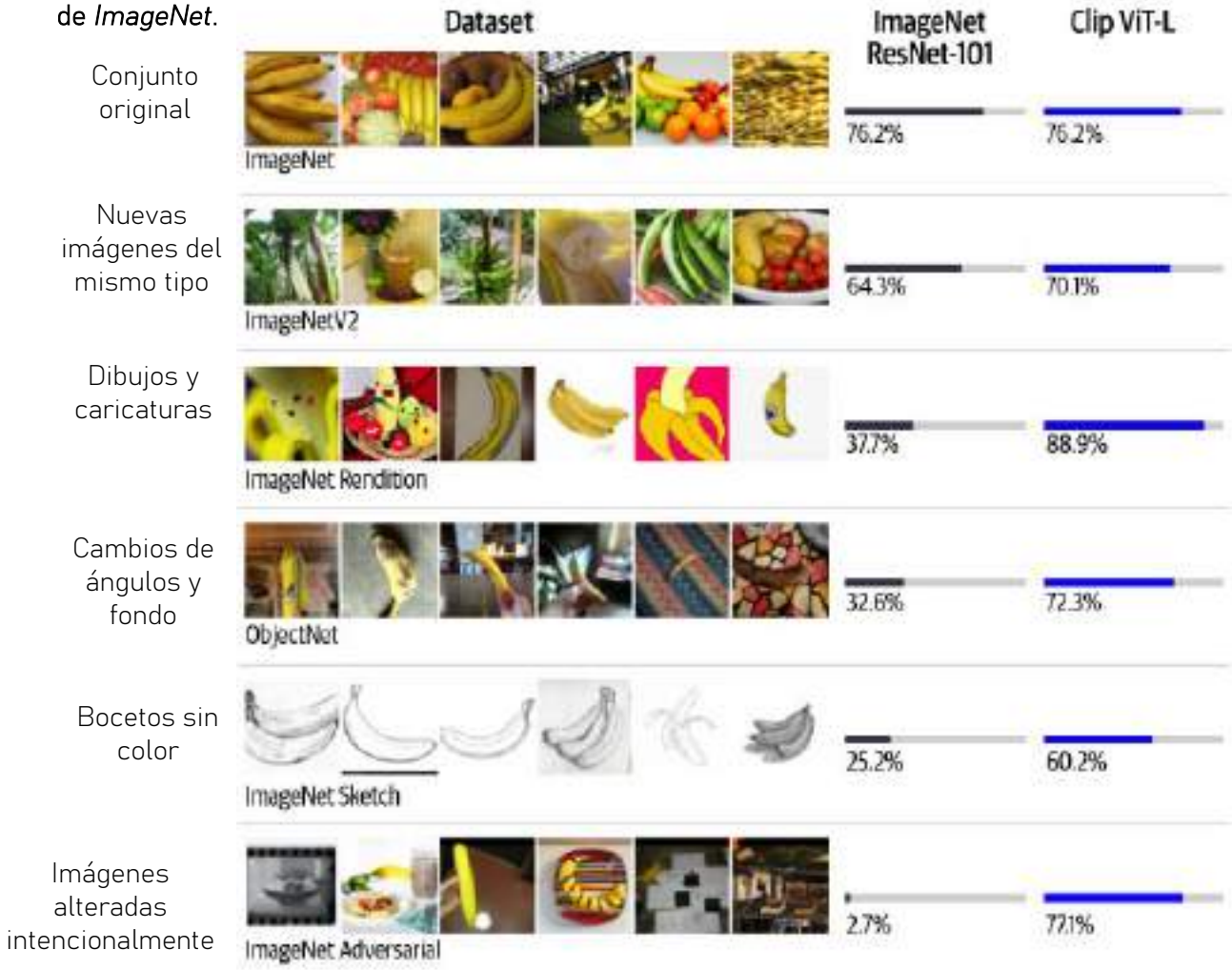
CLIP

Capacidad de generaliación: Zero-Shot Learning

- CLIP demuestra **mayor capacidad de generalización** que los modelos tradicionales entrenados en un solo conjunto de datos.
- Puede **reconocer correctamente conceptos visuales** incluso cuando cambian el estilo, el contexto o la representación de las imágenes.

Cada fila representa una **variación del mismo concepto** (“bananas”) bajo diferentes condiciones visuales.

Distintas versiones de *ImageNet*.

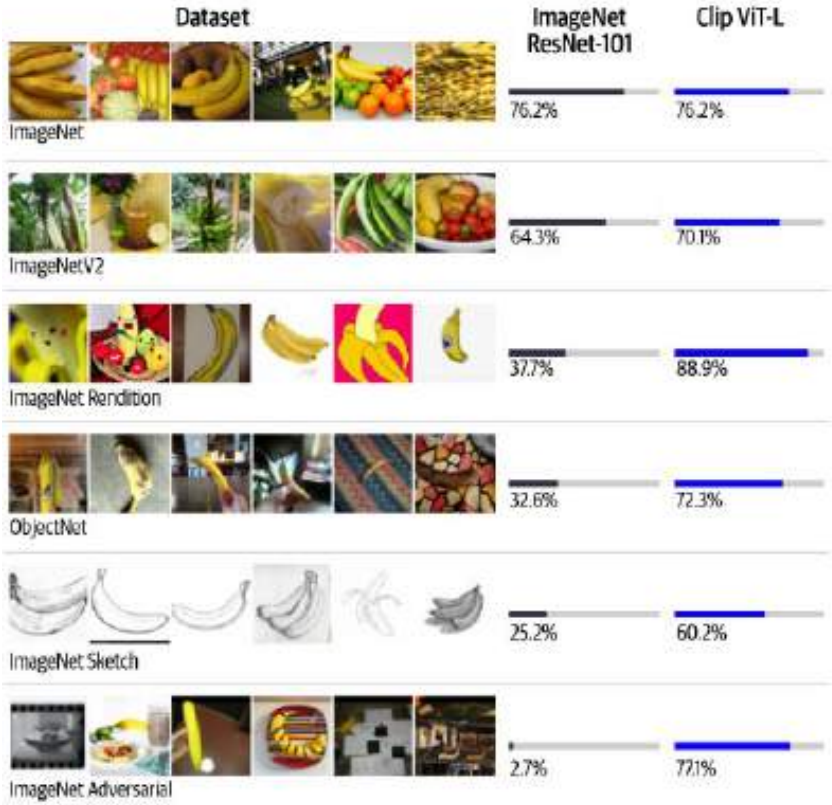


CLIP

Capacidad de generaliación: Zero-Shot Learning

Razón de esta robustez

- CLIP fue entrenado con **400 millones de pares texto–imagen**, lo que le permite **entender conceptos de manera abstracta**, no solo patrones visuales específicos.
- Aprende a **asociar significados** (“banana”) con muchas representaciones posibles, no solo con un tipo de imagen.
- Por eso, puede reconocer una banana aunque esté dibujada, en blanco y negro, o deformada.



Mientras que los modelos tradicionales reconocen solo lo que han visto, CLIP comprende más la idea detrás de lo que ve.

Su aprendizaje contrastivo con lenguaje natural le da una comprensión conceptual más profunda y flexible.