

# Vision Transformers

Docentes:

Esp. Abraham Rodriguez - FIUBA

Mg. Oksana Bokhonok - FIUBA

# Programa de la materia

1. Arquitectura de Transformers e imágenes como secuencias.
2. Arquitecturas de ViT y el mecanismo de Attention.
3. Ecosistema actual, Huggingface y modelos pre entrenados.
4. GPT en NLP e ImageGPT.
5. Modelos multimodales: combinación de visión y lenguaje (CLIP, DALL-E, ..)
6. Segmentación con SAM y herramientas de auto etiquetado multimodales.
7. OCR y detección con modelos multimodales.
8. Presentación de proyectos.

# Evaluación

## 1. Entrega de trabajos prácticas obligatoria (trabajo Individual). Plazos de entrega:

- **Ejercicio 1:** Debe ser entregado antes de la **Clase 3**.
- **Ejercicio 2:** Debe ser entregado antes de la **Clase 4**.
- **Ejercicio 3:** Debe ser entregado antes de la **Clase 5**.

## 2. Entrega del proyecto obligatoria (trabajo en grupo):

- Código funcional.
- Informe técnico que contenga los pasos seguidos, las decisiones de diseño del modelo, el análisis de los resultados y las visualizaciones generadas.
- Presentación final de 15 minutos, mostrando los resultados más destacados, las visualizaciones de atención y cómo el modelo podría aplicarse en un contexto real.




El código y el informe deben ser entregados a más tardar el **lunes anterior a la clase 8**.

### Evaluación del proyecto:

- Claridad conceptual (25%): Explicación de los Vision Transformers.
- Calidad del código (25%): Correctitud y claridad del código implementado.
- Evaluación y análisis (25%): Uso de métricas y análisis en profundidad de los resultados.
- Explicabilidad y visualización (25%): Capacidad para explicar los mecanismos de atención y generar visualizaciones claras y útiles.

$$\text{Evaluación Global} = 0.4 * \text{Prácticas} + 0.6 * \text{Proyecto}$$

# Cronograma de la materia

	1:15H	20M	1:15H
Teoría			
Break			
Teoría/Práctica			

# Tecnologías y herramientas

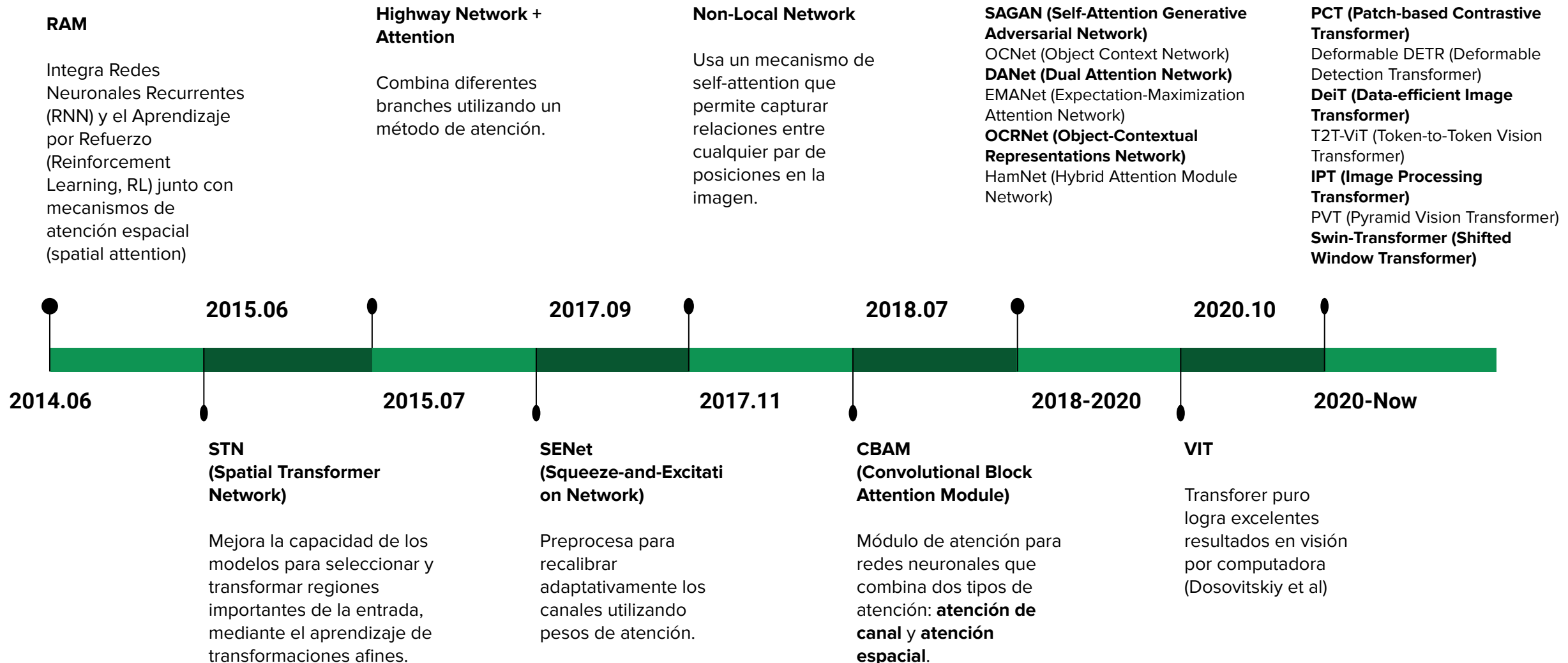


 PyTorch



**Hugging Face**

# Evolución de Transformers en NLP y Computer Vision



## Origen de ViT

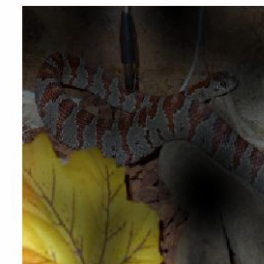
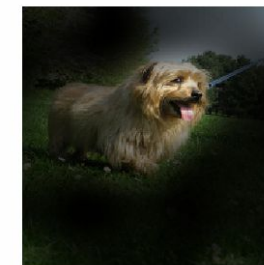
El Vision Transformer (ViT) oficialmente fue presentado en 2021 en el artículo "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale" por Alexey Dosovitskiy et al.

- ❑ El artículo demuestra que no siempre es necesario utilizar CNNs para lograr buenos resultados en clasificación.
- ❑ El modelo ViT utiliza entre 2 y 4 veces menos recursos computacionales en comparación con CNNs tradicionales como ResNet.
- ❑ El ViT se basa en un mecanismo de atención que permite modelar relaciones globales entre los píxeles, lo que efectivamente le permite identificar y "atender" regiones relevantes en la imagen durante la clasificación.

Input



Attention

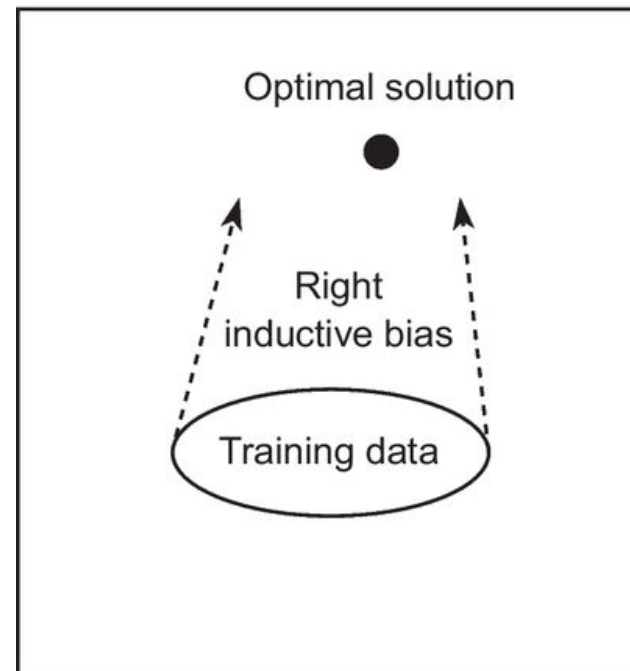


## Bias inductivo

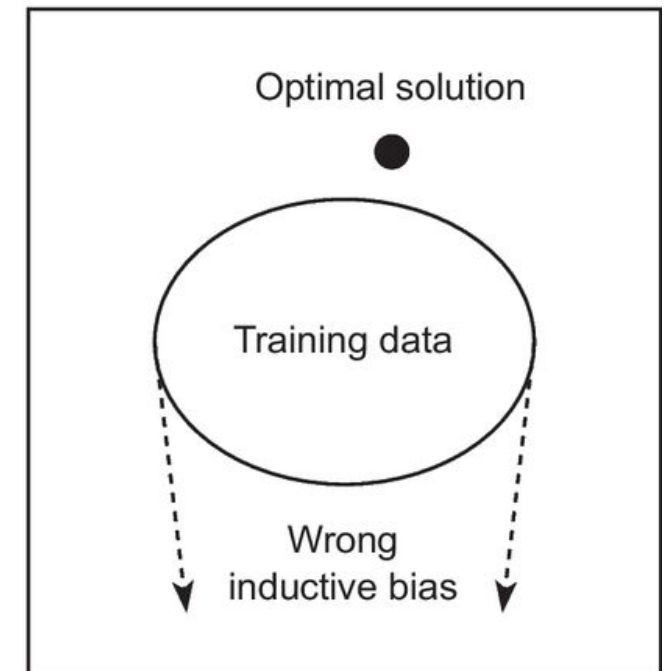
Todo modelo de aprendizaje automático necesita un diseño arquitectónico y suposiciones iniciales sobre los datos, lo que constituye un **sesgo inductivo**.

Los Transformers carecen de ciertos sesgos inductivos presentes en las CNN, como la invariancia traslacional y el aprovechamiento explícito de la estructura local de las imágenes.

Esto los hace menos efectivos con datos limitados, pero más robustos con conjuntos más grandes.



Ayuda en un contexto con pocos datos.



Perjudica la generalización, incluso en un contexto con muchos datos.

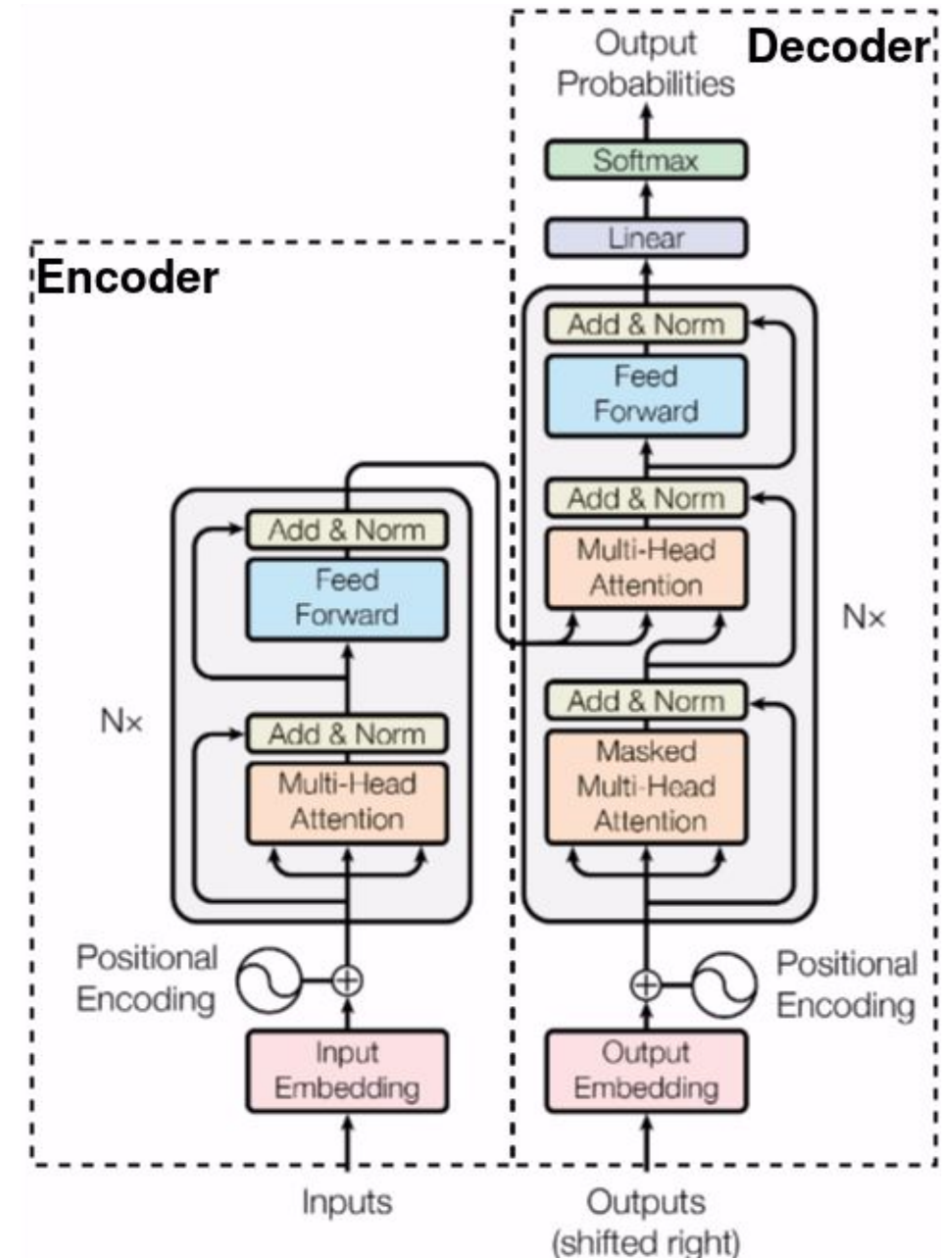


## Arquitectura del Transformer

**Embeddings:** Las imágenes se dividen en parches y se convierten en embeddings. Estos embeddings son vectores que representan cada parche de la imagen, permitiendo que el modelo trate la imagen como una secuencia, similar a cómo se procesan las palabras en NLP.

**Codificación Posicional (Positional Encoding):** Como los Transformers no tienen un sentido inherente del orden de las secuencias, se añaden embeddings posicionales para que el modelo sepa la posición de cada parche dentro de la imagen.

**Multi-Head Attention (MHA):** El corazón de la arquitectura de ViTs es el mecanismo de **Multi-Head Attention**. Aquí, múltiples cabezas de atención permiten al modelo enfocarse en diferentes partes de la imagen al mismo tiempo, capturando relaciones complejas dentro de la imagen.



## Mecanismo de Atención en Transformers

**Scaled Dot-Product Attention:** Es el mecanismo de atención que se utiliza para calcular la atención en base a las entradas Q (Queries), K (Keys), y V (Values).

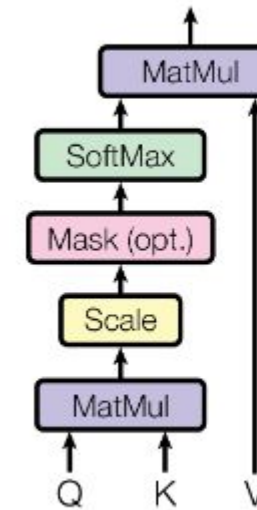
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \cdot V$$

**Multi-Head Attention:** Es una extensión del mecanismo de atención que aplica múltiples capas de atención en paralelo. Esto implica realizar proyecciones lineales de los vectores de entrada en varios subespacios, cada uno alimentando una capa de atención independiente. Los resultados de estas capas se concatenan y se proyectan nuevamente a una dimensión reducida, permitiendo que el modelo capture diferentes aspectos de las relaciones entre elementos en la secuencia.

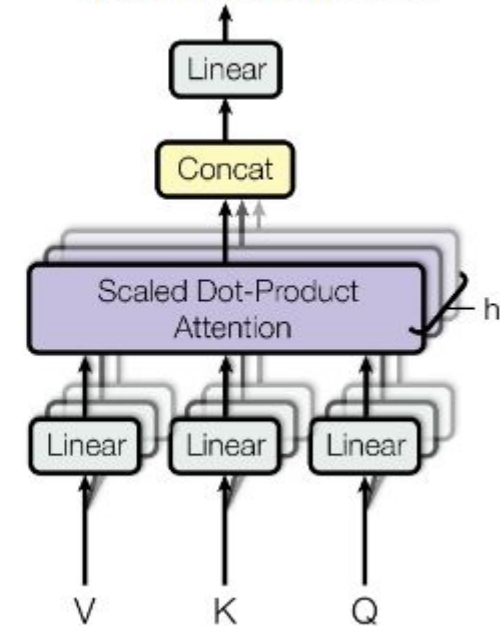
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h) W^O$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Scaled Dot-Product Attention



Multi-Head Attention

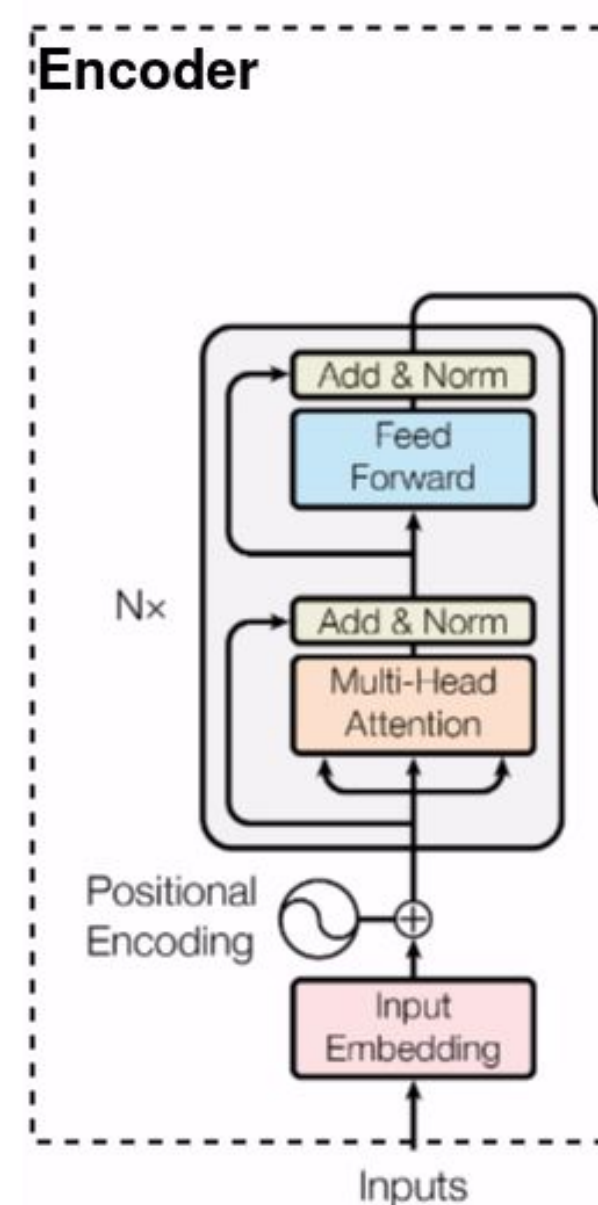


## Arquitectura del Transformer Encoder

**Suma y Normalización (Add & Norm):** Después de aplicar el MHA, se realiza una operación de suma y normalización en los resultados para estabilizar y mejorar el aprendizaje. Esto garantiza que las señales permanezcan dentro de un rango adecuado antes de pasar al siguiente bloque.

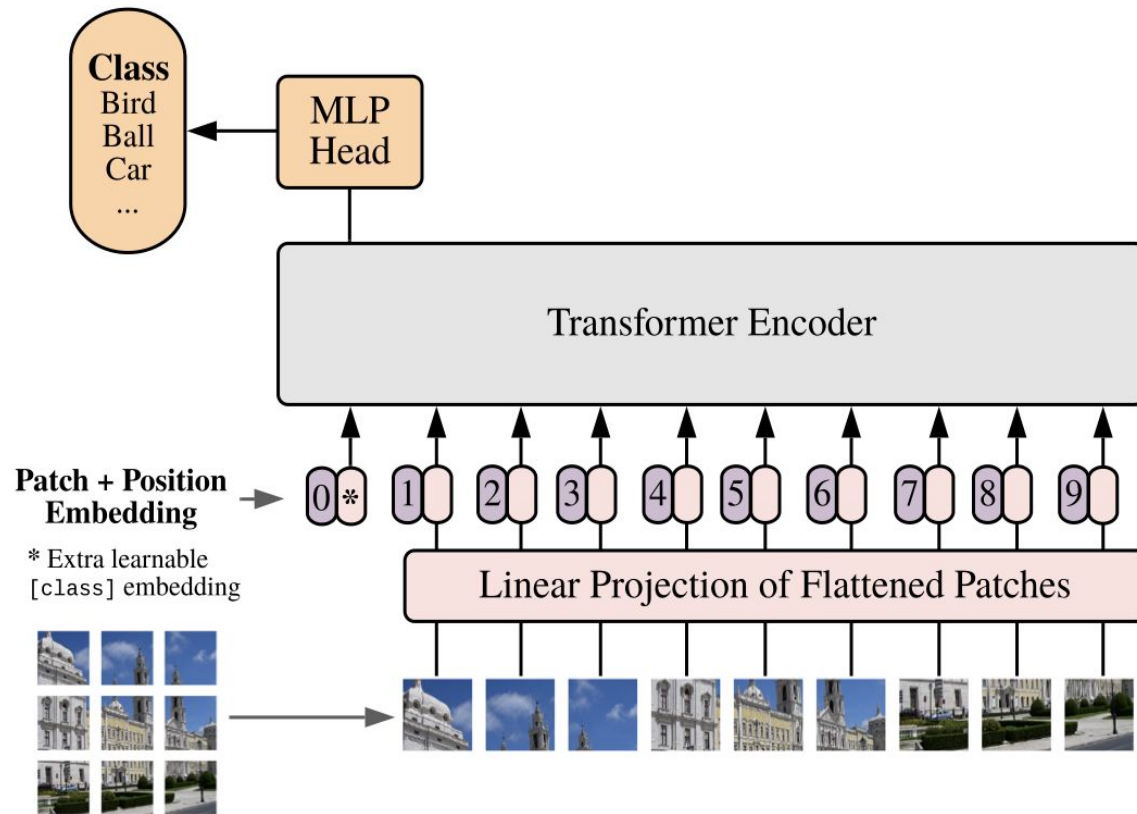
**Red Lineal Feed-Forward (Feed Forward):** Cada capa de atención es seguida por una red lineal feed-forward, que realiza transformaciones no lineales en los datos para extraer características más complejas.

**Iteración en Capas (Nx):** El proceso de MHA seguido de la red feed-forward se repite múltiples veces (Nx) en el encoder, lo que permite al modelo aprender representaciones más ricas de la imagen.

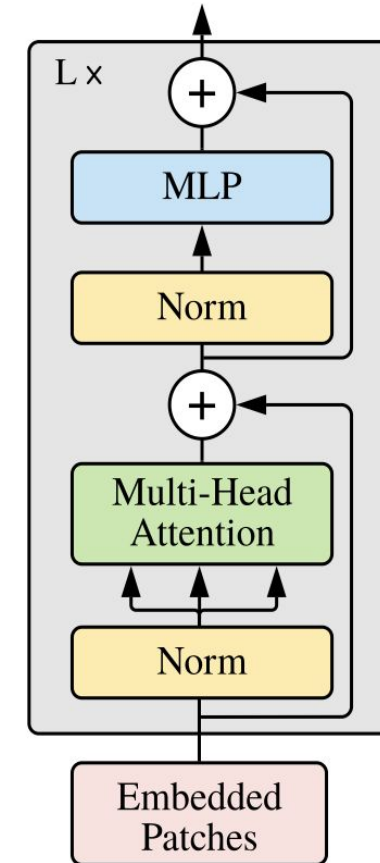


# Arquitectura ViT

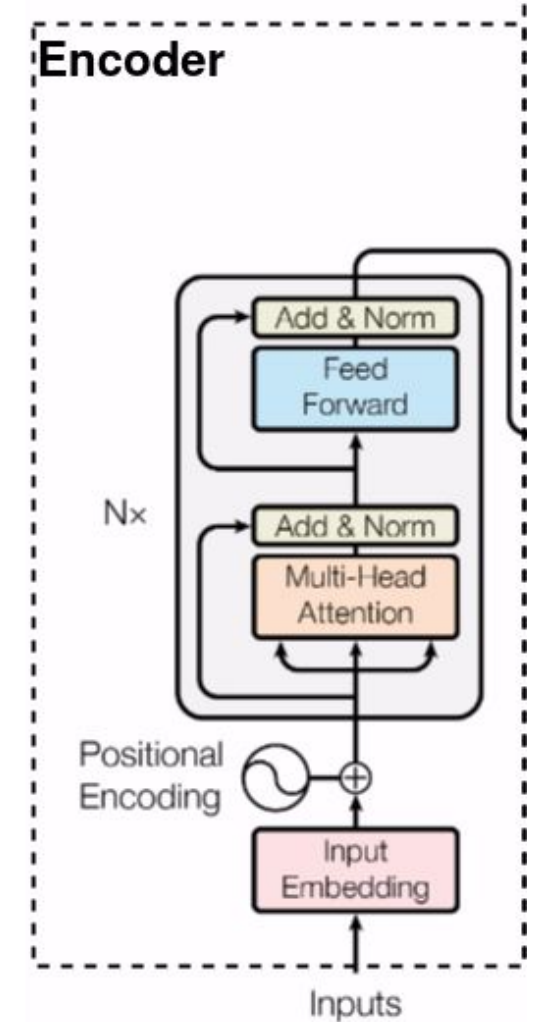
## Vision Transformer (ViT)



## Transformer Encoder



=



Fuente: Dosovitskiy, A., et al. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929.



## Entrada de datos ViT

### “Tokenización” de Imágenes

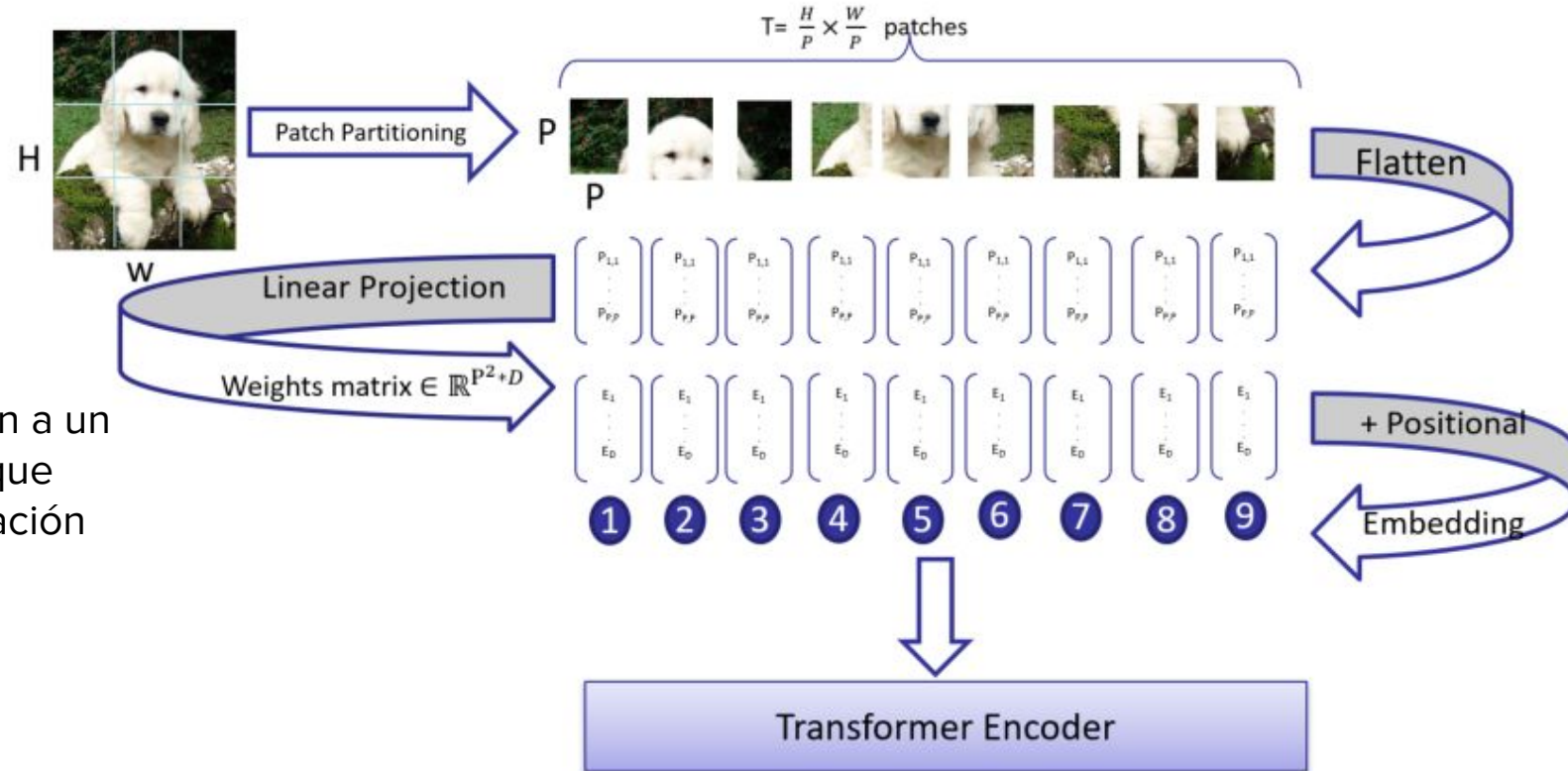
Imagen → parches (patches) → vector

### Patch Embeddings

Los vectores de los parches se proyectan a un espacio latente mediante embeddings, que permiten al modelo interpretar la información de cada parche de manera efectiva.

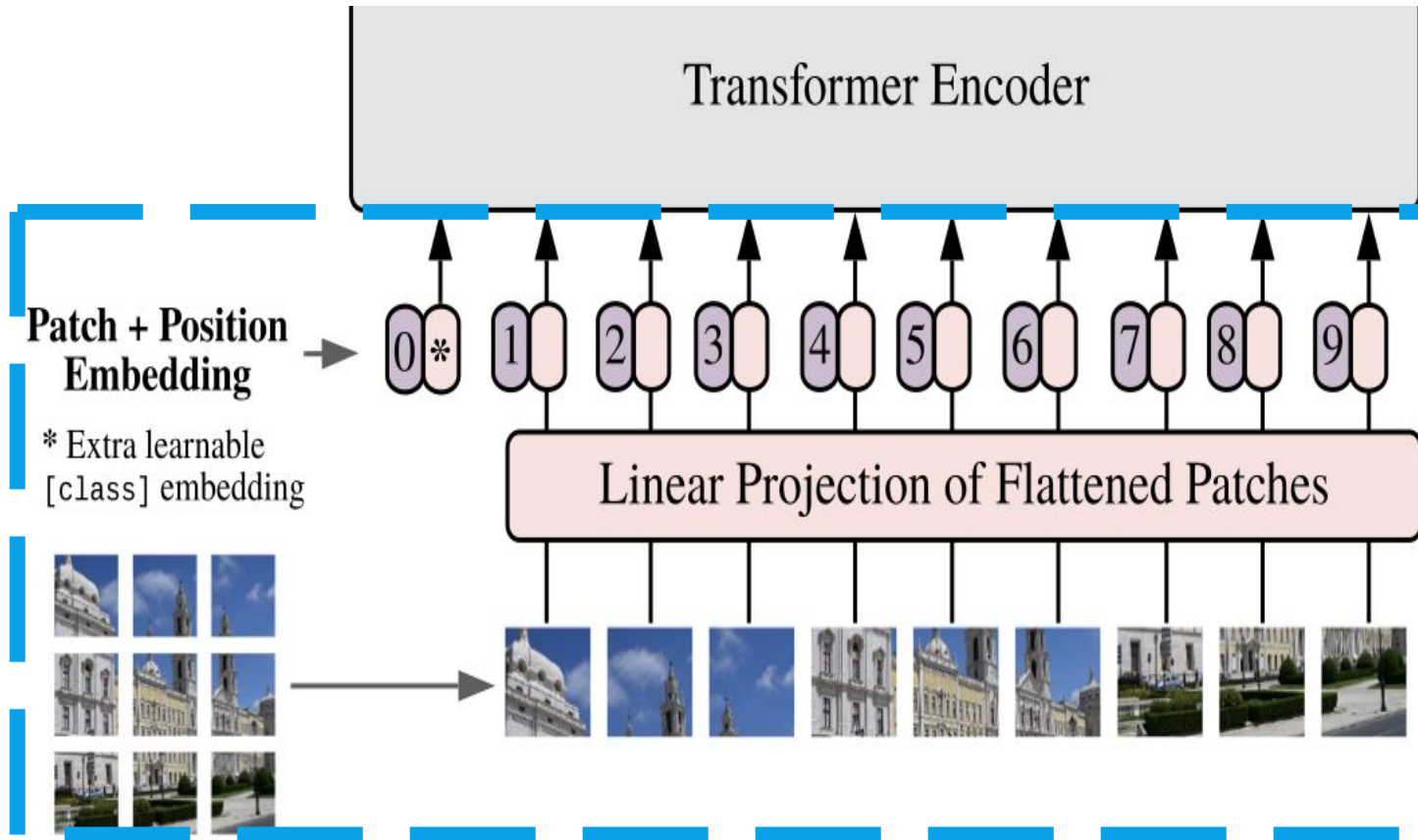
### Positional Embedding

Para conservar la información espacial de los parches, se añade una codificación posicional a los embeddings, lo que permite al modelo entender la ubicación de cada parche dentro de la imagen.



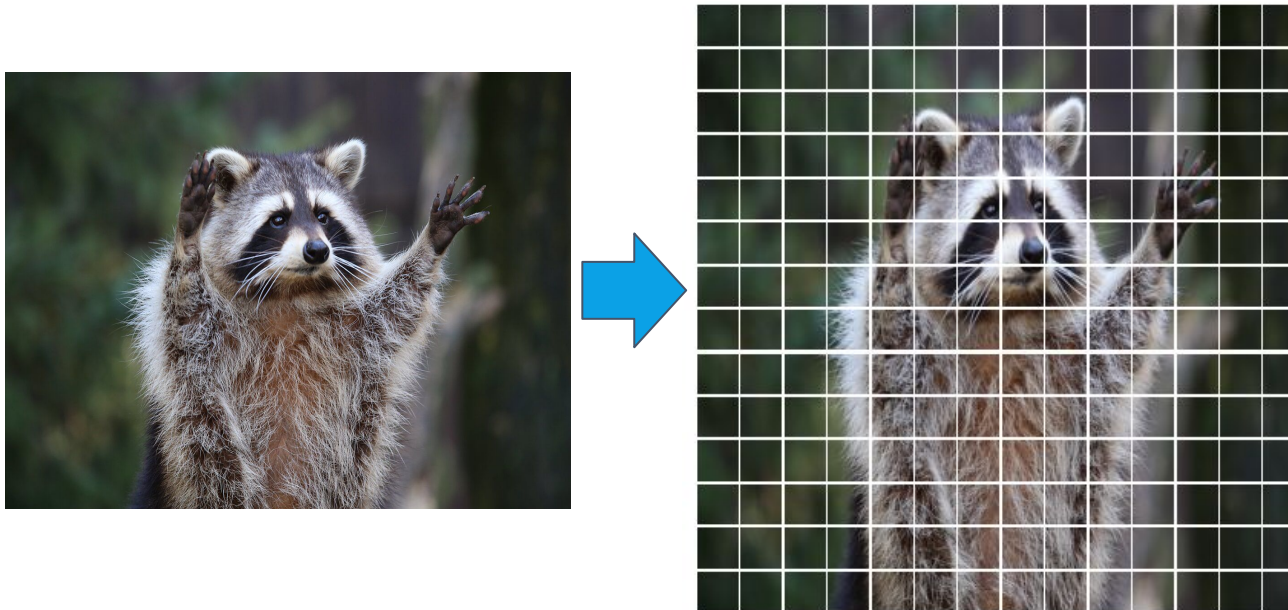
Fuente: [Converting weights of 2D Vision Transformer for 3D Image Classification - DLMA: Deep Learning for Medical Applications - BayernCollab \(dvv.bayern\)](#)

## Patch Embeddings



- ❑ La imagen se divide en parches de tamaño fijo.
- ❑ Cada parche se proyecta a través de una capa lineal, convirtiéndose en un vector.

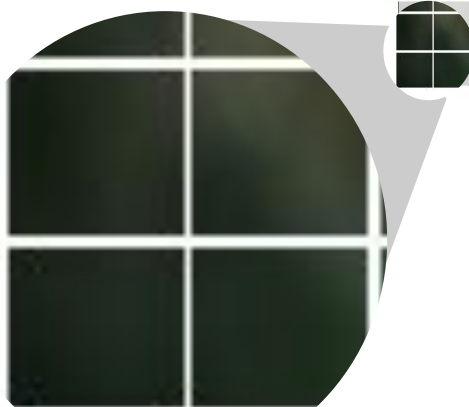
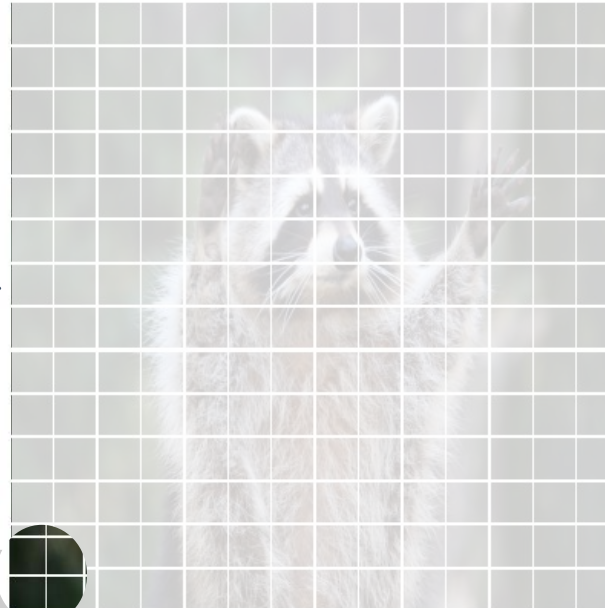
## Patch Embeddings



Sea una imagen  $x \in \mathbb{R}^{H \times W \times C}$ , donde  $H$ ,  $W$  y  $C$  representan la altura, el ancho y el número de canales, respectivamente.

Reestructuramos la imagen en una secuencia de parches 2D aplanados  $x_p \in \mathbb{R}^{N \times (P \cdot P \cdot C)}$ , donde  $(P, P)$  es la resolución de cada parche.

## Patch Embeddings



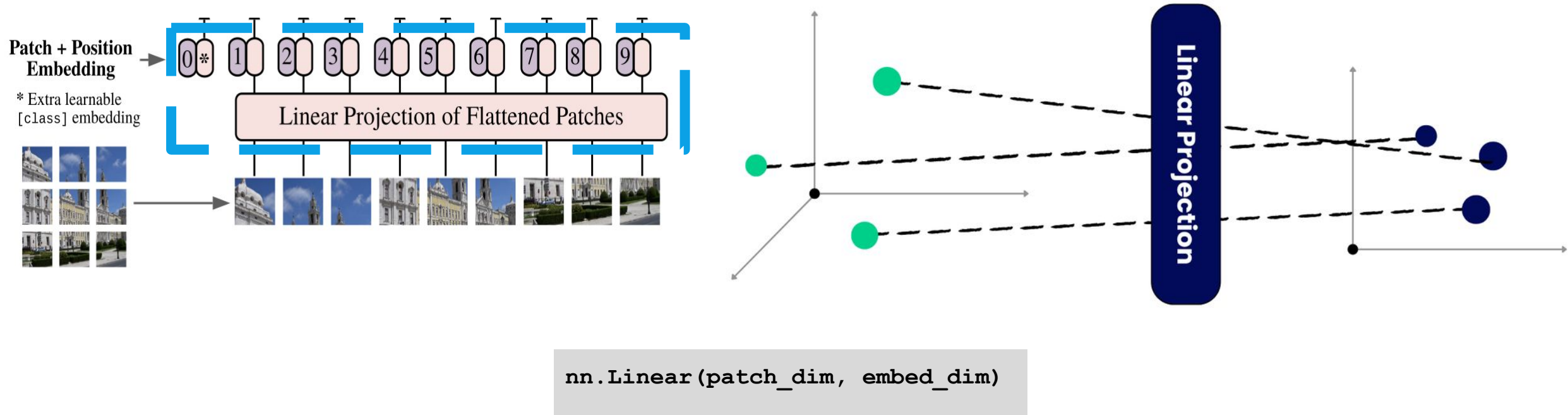
Cada patch  
tiene un  
área de  $P^2$ .

$N = HW/P^2$   $N \in \mathbb{Z}$ , es el número resultante de parches, que también sirve como la longitud efectiva de la secuencia de entrada para el Transformer.



## Patch Embeddings Linear Projection

Una vez tenemos  $N$  patches se debe de realizar la proyección de cada parche al espacio latente (embeddings). Siguiendo la implementación original se aplica una red neuronal lineal.



## Patch Embeddings Convolutional Projection

Cuando las imágenes no son cuadradas, la proyección lineal requiere de padding ya que  $N \in \mathbb{Z}$  o en otras palabras la resolución debe de ser divisible entre el tamaño de cada patch y ser un número entero.

Sin embargo se puede sustituir mediante una convolución 2D donde:

1. Kernel (K) = patchSize
2. Stride (S) = patchSize

La convolución es una operación altamente optimizada y vuelve más simple la implementación de Patch Embedding.

```
nn.Conv2d(in_channels, embed_dim, kernel_size=patch_size, stride=patch_size)
```

## Positional Embeddings (NLP vs ViT)

El Transformer por naturaleza **desconoce el contexto espacial** de los datos de entrada. En NLP se sigue el orden de tokens de entrada siendo necesaria la información posicional de los tokens.

En la investigación original de ViT, optaron a utilizar un vector 1D, ya que no se vio beneficio utilizar embeddings avanzados.

Sea:

$P \in \mathbb{R}^{1 \times N \times D}$  la matriz de PE un **parámetro entrenable** inicializado de manera **aleatoria**.

Sequence	Index of token, $k$	Positional Encoding Matrix with $d=4$ , $n=100$			
		$i=0$	$i=0$	$i=1$	$i=1$
I	0	$P_{00}=\sin(0)$ = 0	$P_{01}=\cos(0)$ = 1	$P_{02}=\sin(0)$ = 0	$P_{03}=\cos(0)$ = 1
am	1	$P_{10}=\sin(1/1)$ = 0.84	$P_{11}=\cos(1/1)$ = 0.54	$P_{12}=\sin(1/10)$ = 0.10	$P_{13}=\cos(1/10)$ = 1.0
a	2	$P_{20}=\sin(2/1)$ = 0.91	$P_{21}=\cos(2/1)$ = -0.42	$P_{22}=\sin(2/10)$ = 0.20	$P_{23}=\cos(2/10)$ = 0.98
Robot	3	$P_{30}=\sin(3/1)$ = 0.14	$P_{31}=\cos(3/1)$ = -0.99	$P_{32}=\sin(3/10)$ = 0.30	$P_{33}=\cos(3/10)$ = 0.96

Positional Encoding Matrix for the sequence 'I am a robot'

$$PE_{(pos, 2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

$$PE_{(pos, 2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

## Juntando los Embeddings (Transformer Encoder Input)

Sea:

$X$  la matriz de patch embeddings.

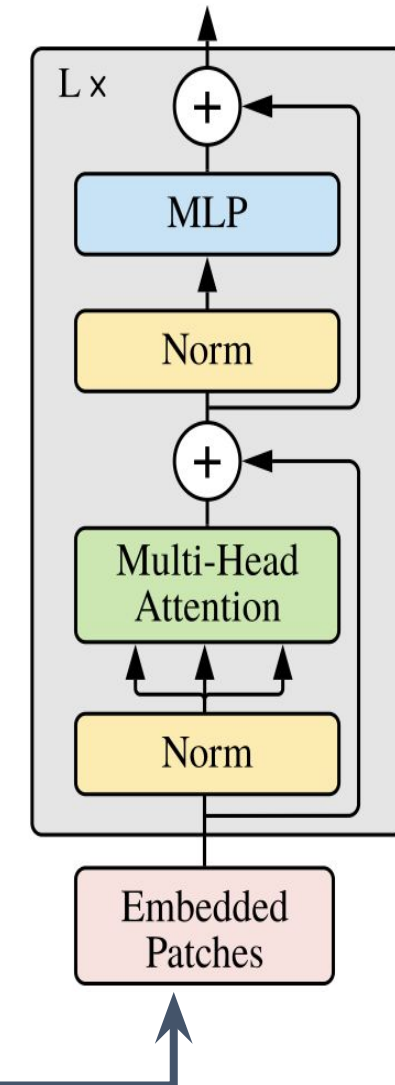
$P$  la matriz de positional embeddings.

$Z$  la matriz de entrada al Transformer, donde  $B$  es el tamaño del batch.

Entonces la entrada al encoder es:

$$Z = X + P \text{ donde } Z \in \mathbb{R}^{B \times N \times D}$$

Transformer Encoder




## ViT vs CNN: Comparativa y beneficios de enfoques híbridos (ViT+CNN)

Característica	ViT	CNN (Convolutional Neural Networks)	Híbrido ViT+CNN
<i>Estructura</i>	Basado en self-attention. Sin operación de convolución.	Basado en convoluciones para la extracción de características.	Combinación de convolución y self-attention.
<i>Procesamiento de datos</i>	Procesa la imagen como una secuencia de parches.	Procesa la imagen usando filtros convolucionales.	Usa convolución para características locales y ViT para relaciones globales.
<i>Ventajas</i>	<ul style="list-style-type: none"> <li>- Captura relaciones globales entre píxeles.</li> <li>- Escalabilidad con datos grandes.</li> </ul>	<ul style="list-style-type: none"> <li>- Excelente para capturar características locales.</li> </ul>	Captura características locales y globales.
<i>Desventajas</i>	<ul style="list-style-type: none"> <li>- Necesita más datos para entrenar.</li> </ul>	Dificultad para capturar relaciones globales a gran escala.	Mayor complejidad y demanda computacional.
<i>Necesidad de datos</i>	Alta cantidad de datos para entrenar eficientemente.	Menor cantidad de datos en comparación con los ViT.	Depende del balance entre la parte CNN y la parte ViT.
<i>Ejemplos</i>	Diagnóstico médico como detección de patologías en una radiografía.	Reconocimiento facial, donde se debe identificar características y detalles faciales.	Vehículos autónomos para identificar y clasificar objetos en tiempo real.

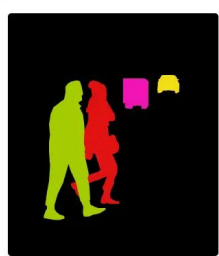
# Aplicaciones de ViT: Casos de Uso

Aplicación	Ejemplos de Uso
Tareas de Reconocimiento	Clasificación de imágenes: Categorizar imágenes (e.g., especies de animales). Detección de objetos: Identificación de objetos en imágenes. Segmentación: Dividir imágenes en segmentos significativos.
Tareas Multimodales	Respuesta a preguntas visuales: Responder preguntas sobre el contenido de una imagen. Razonamiento visual: Realizar inferencias basadas en imágenes. Visual grounding: Asociar texto con regiones específicas de una imagen.


### Types of Image Segmentation



SEMANTIC IMAGE SEGMENTATION




INSTANCE SEGMENTATION



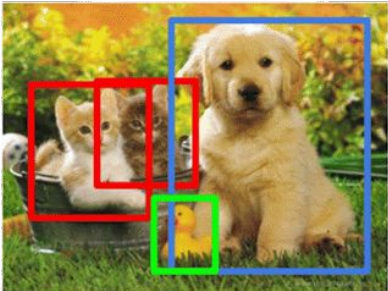
PANOPTIC SEGMENTATION

### Classification

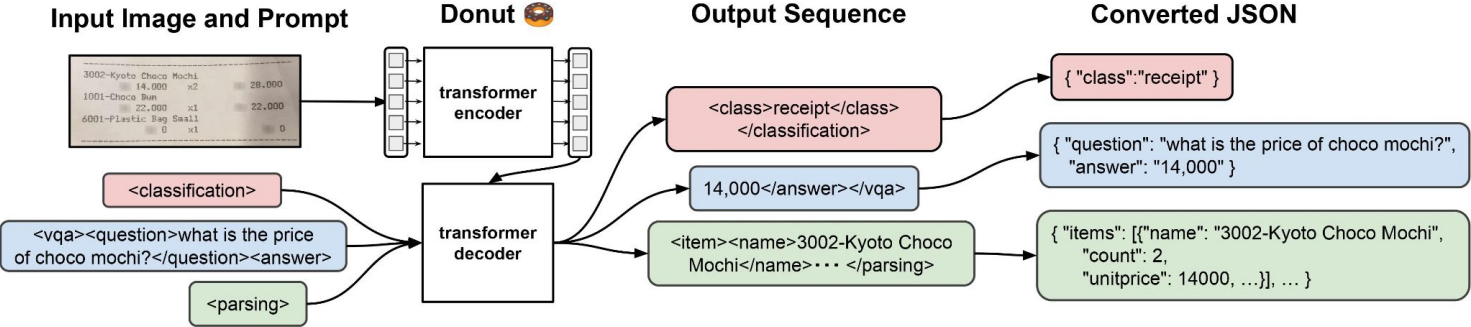


CAT

### Object Detection



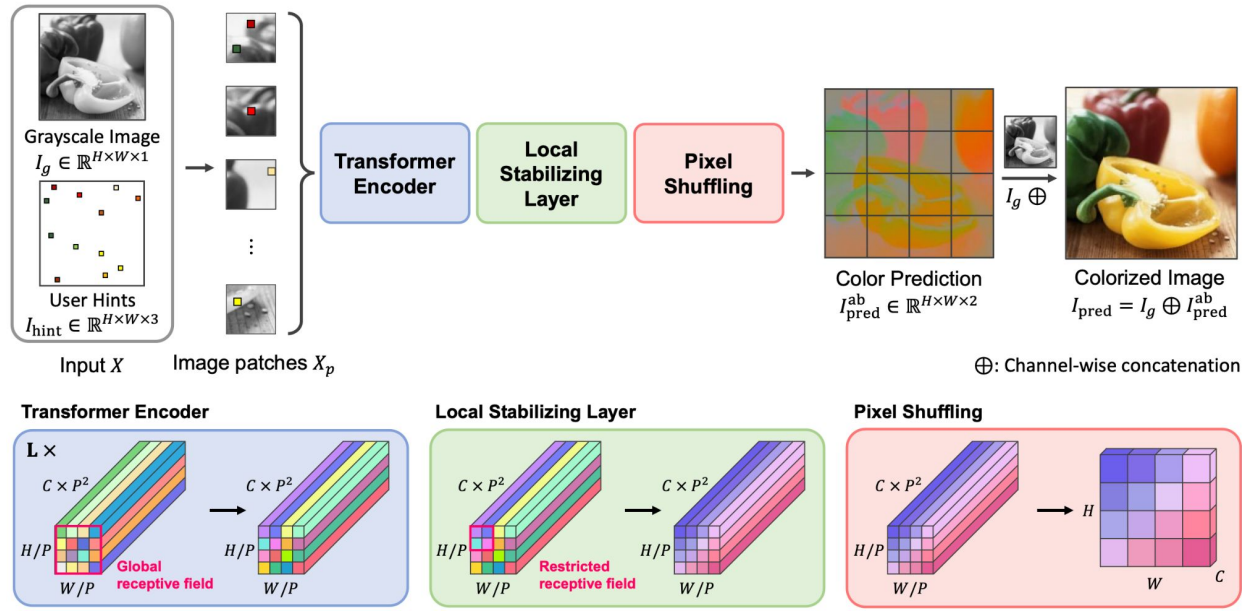
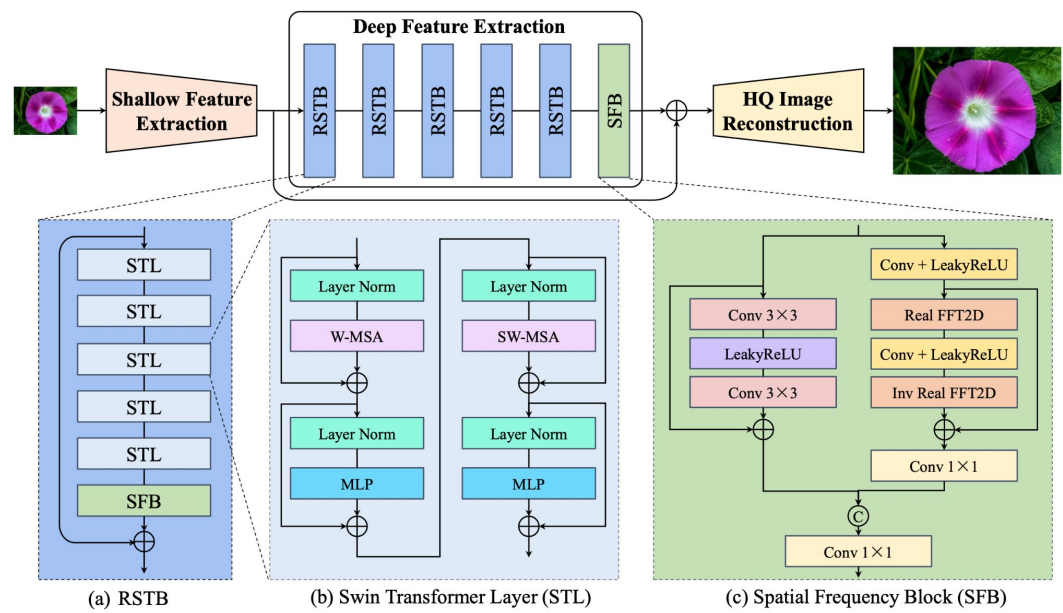
CAT, DOG, DUCK





# Aplicaciones de ViT: Casos de Uso

Aplicación	Ejemplos de Uso
Procesamiento de Video	Reconocimiento de actividades: Identificar acciones en secuencias de video. Pronóstico de video: Predecir frames futuros en un video.
Visión de Bajo Nivel	Súper-resolución de imágenes: Mejorar la resolución de imágenes borrosas. Mejora de imágenes: Aumentar la calidad visual. Colorización: Asignar colores a imágenes en blanco y negro.
Análisis 3D	Clasificación y segmentación de nubes de puntos: Clasificación y segmentación de datos 3D.



# Preparando el ambiente (Linux/MacOS)

- Verificar GPU driver (Linux).
- Instalar Pytorch mediante el [link](#).
- Copiar y ejecutar:

```
device = 'cuda' if torch.cuda.is_available() else 'mps' if torch.backends.mps.is_available() else 'cpu'  
print(device) # Expected: 'cuda' if Linux with available GPU else 'mps' if MacOS else
```

## Nota:

**En Linux revisar la versión de CUDA. 12.1+ es recomendado.**



# Preparando el ambiente (Windows Nativo)

- Verificar GPU driver (NVIDIA Control Panel).
- Instalar Pytorch mediante el [link](#).
- Copiar y ejecutar:

```
device = 'cuda' if torch.cuda.is_available() else 'cpu'  
print(device) # Expected: 'cuda' if GPU is available'
```

## Nota:

**Revisar la versión de CUDA. 12.1+ es recomendado.**

# Preparando el ambiente (Windows WSL + Docker)

## 1. Configuración de WSL2 (Command Prompt): [Guia-WSL](#)

```
wsl --install  
wsl --set-default-version 2  
wsl -l -v (verifica la versión de WSL)  
wsl --update (actualiza WSL2 a la última versión si es necesario)  
Instala una distribución Linux (ej: Ubuntu):  
wsl --install -d Ubuntu  
wsl --set-default Ubuntu
```

## 2. Instalar Docker Desktop:

Descargar Docker Desktop desde [Link-Docker](#)  
Configurar Docker con WSL2 (Docker de escritorio):  
En Settings > General y habilitar "Use the WSL2 based engine".  
En Settings > Resources > WSL Integration, seleccionar distribución Linux  
**Verificación de la instalación de Docker (Command Prompt-wsl):**  

```
docker --version  
docker info
```

**Agregar permisos de usuario al grupo docker (command prompt-wsl):**  

```
sudo groupadd docker  
sudo usermod -aG docker <usuario>
```

## 3. Instalar drivers NVIDIA para WSL2:

Descargar e instala los drivers NVIDIA WSL2 desde [Link-Nvidia](#)  
Verificar que los drivers se instalaron correctamente (Command Prompt - wsl):  

```
nvidia-smi
```

Instalar NVIDIA Container Toolkit:

```
sudo apt-get install -y nvidia-container-toolkit  
sudo systemctl restart docker
```

## 4. Configurar y verificar Docker con CUDA: [Guia-GPU support | Docker Docs](#)

Verificar si Docker está ejecutándose correctamente: `docker ps`  
Probar Docker con un contenedor básico: `sudo docker run -d --name test-container hello-world`  
Verificar que Docker tenga acceso a la GPU y CUDA:  

```
sudo docker run --rm --gpus all nvidia/cuda:12.1.1-base-ubuntu22.04 nvidia-smi
```

Verificar la instalación de `nvcc` (CUDA Compiler):

```
sudo docker run --rm --gpus all nvidia/cuda:12.1.1-devel-ubuntu22.04 nvcc --version
```

## 5. Ejecutar PyTorch con soporte CUDA en Docker: [Guia-PyTorch | NVIDIA NGC](#)

Instalar un contenedor de PyTorch: `docker pull pytorch/pytorch:latest`  
Veamos el nombre de docker: `docker ps` (`docker ps -a`, si no esta en ejecución)  
Cambiemos el nombre: `docker rename <viejo_nombre> <nuevo_nombre>`  
Habilitar los permisos sobre la carpeta que vamos a usar: `chmod -R 777 /home/<usuario>/<carpeta_proyecto>`  
Ejecutar el contenedor de PyTorch habilitando la conexión a la carpeta ubicada en linux:  
`docker run -v /mnt/wsl/Ubuntu/home/<tu_usuario>/<carpeta_proyecto>:/app/<carpeta_proyecto> -it <nombre_docker>`  
Dentro del contenedor, verifica si PyTorch detecta la GPU:  

```
python3 -c "import torch; print(torch.cuda.is_available())"
```

## 6. Configuración de VS Code dentro de Docker: [Guia-VS Code](#)

Instalar Visual Studio Code en Windows [Link-VS Code](#)  
Instalar la extensión Remote - WSL en VS Code, que permite editar y depurar código dentro de tu distribución WSL2 (tendría que detectar la existencia de wsl y sugerirlo al abrir VS Code).  
Configurar VS Code para Docker:  
- Instala la extensión Remote - Containers desde el marketplace de VS Code.  
- Una vez dentro de un proyecto, abre el entorno de Docker en VS Code:  
- Usa el comando F1 o Ctrl+Shift+P y selecciona Dev Containers: Attach to Running Container o Dev Containers:  
Open Folder in Container, según operación a ser realizada.

Si no se necesita docker, usar la guía de instalación de WSL, NVIDIA y VSCode sin la configuración de Docker

# TP-I

El trabajo práctico puede ser encontrado en el GitHub de la materia CEIA-ViT: [LINK](#)

Plazo de entrega antes de la clase 3.

**Recuerden formar los grupos!**

# Algunas ideas para los proyectos (pueden elegir otro tema)

1. Segmentación de Imágenes Médicas con ViTs	Aplicar un Vision Transformer a una tarea de segmentación de imágenes médicas. La segmentación consiste en dividir una imagen en regiones de interés, como identificar tumores en imágenes de resonancias magnéticas.
2. Detección de Anomalías en Imágenes de Producción Industrial	El objetivo es detectar defectos en productos industriales utilizando ViTs, Identificar áreas de las imágenes donde el producto tiene defectos (por ejemplo, un rayón en una pieza de metal).
3. Traducción de Imágenes a Texto (Scene Text Recognition)	El objetivo de este proyecto es construir un modelo basado en ViTs que reconozca texto en imágenes (OCR, Optical Character Recognition).  Dataset sugerido: COCO-text
4. Generación de Imágenes a partir de Descripciones Textuales	Este proyecto involucra la transformación de una descripción textual (por ejemplo, "un gato sobre un sofá") en una imagen correspondiente usando ViTs  Dataset sugerido: COCO Captions Dataset <a href="https://cocodataset.org/#captions-2015">https://cocodataset.org/#captions-2015</a>
5. Generación de Videos a partir de Imágenes Estáticas	Opción 1: Generar un video de corta duración a partir de una sola imagen estática. El objetivo es crear un modelo que pueda predecir una secuencia de cuadros que continúe una imagen fija  Opción 2: Entrenar un modelo que permita describir las acciones realizadas en un video.  Dataset sugerido: UCF101 Dataset  <a href="https://www.crcv.ucf.edu/research/data-sets/ucf101/">https://www.crcv.ucf.edu/research/data-sets/ucf101/</a>
Además recordemos algunas de las fuentes de datos disponibles:  <a href="https://cocodataset.org/">COCO - Common Objects in Context (cocodataset.org)</a>  <a href="https://image-net.org/">ImageNet (image-net.org)</a>  Kaggle datasets: <a href="https://www.kaggle.com/datasets">Find Open Datasets and Machine Learning Projects   Kaggle</a>  <a href="https://huggingface.co/datasets">GitHub - huggingface/datasets: 🤗 The largest hub of ready-to-use datasets for ML models with fast, easy-to-use and efficient data manipulation tools</a>  <a href="https://ultralytics.com/datasets/">Datasets Overview - Ultralytics YOLO Docs</a>	

**Gracias por su atención y  
dedicación.**

*Recuerden que los grandes retos traen grandes aprendizajes.*

**¡Nos vemos en la próxima clase!**