

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
6. Segmentación con SAM y herramientas de auto etiquetado multimodales.
7. OCR y detección con modelos multimodales.
8. Presentación de proyectos.

Cronograma de la materia

Opción 1

	1:30H	20M	1:30H
Teoría			
Break			
Teoría/Práctica			

Opción 2

	1:00H	10M	50Min	10M	50Min
Teoría					
Break					
Teoría/Práctica					

Evaluación

Entrega del Proyecto (Obligatoria - Grupal)

El proyecto debe incluir los siguientes elementos:

- Proyecto estructurado en git que contenga:
 - Código Código funcional y modular (nivel-preproducción).
 - Informe técnico (pdf): Debe contener
 - Objetivo del proyecto
 - Arquitectura general (diagrama de flujo + descripción de componentes)
 - Implementación técnica (herramientas, módulos clave)
 - Evaluación (métricas de desempeño de modelos, agentes y RAG)
 - Resultados y ejemplos
 - Conclusiones y mejoras futuras
 - Planificación del equipo (tabla con tareas, responsables y estado)
 - README orientativo.
- Presentación final de 15 minutos en la clase 8: Enfocada en
 - Análisis de los resultados más relevantes, con énfasis en las métricas utilizadas.
 - Visualizaciones del modelo.
 - Explicación de cómo el modelo puede aplicarse en un contexto real.
- El código y el informe deben ser entregados a más tardar el día de la clase 7.

2. Encuesta Intragrupal.

La entrega tardía tiene una penalización de 2 puntos.

Evaluación - encuesta

1. Cumplimiento y responsabilidad

Cumplió sus tareas de forma puntual y con la calidad esperada, manteniendo una actitud responsable y confiable durante el desarrollo del trabajo.

2. Participación e iniciativa

Mantuvo una participación activa en las actividades del grupo, aportando ideas, buscando información y proponiendo mejoras de manera autónoma.

3. Calidad técnica de sus aportes

Sus contribuciones fueron técnicamente correctas, relevantes y aportaron valor al resultado final del trabajo.

4. Resolución de problemas y autonomía

Mostró capacidad para enfrentar dificultades técnicas o de organización, encontrando soluciones efectivas sin depender en exceso del resto del equipo.

5. Comunicación y colaboración

Se comunicó de forma clara y respetuosa, favoreciendo la coordinación y el buen clima de trabajo dentro del grupo.

Código-Estructura

[Cookiecutter Data Science](#)

[Folder Structure for Machine Learning Projects](#)

├─ LICENSE	
├─ Makefile	<- Makefile with commands like `make data` or `make train`
├─ README.md	<- The top-level README for developers using this project.
├─ data	
│ ├─ external	<- Data from third party sources.
│ ├─ interim	<- Intermediate data that has been transformed.
│ ├─ processed	<- The final, canonical data sets for modeling.
│ └─ raw	<- The original, immutable data dump.
├─ docs	<- A default Sphinx project; see sphinx-doc.org for details
├─ models	<- Trained and serialized models, model predictions, or model summaries
├─ notebooks	<- Jupyter notebooks. Naming convention is a number (for ordering), the creator's initials, and a short '-' delimited description, e.g. `1.0-jqp-initial-data-exploration`.
├─ references	<- Data dictionaries, manuals, and all other explanatory materials.
├─ reports	<- Generated analysis as HTML, PDF, LaTeX, etc.
│ └─ figures	<- Generated graphics and figures to be used in reporting
├─ requirements.txt	<- The requirements file for reproducing the analysis environment, e.g. generated with `pip freeze > requirements.txt`
├─ setup.py	<- Make this project pip installable with `pip install -e`
├─ src	<- Source code for use in this project.
│ └─ __init__.py	<- Makes src a Python module
│ └─ data	<- Scripts to download or generate data
│ │ └─ make_dataset.py	
│ └─ features	<- Scripts to turn raw data into features for modeling
│ │ └─ build_features.py	
│ └─ models	<- Scripts to train models and then use trained models to make predictions
│ │ └─ predict_model.py	
│ │ └─ train_model.py	
└─ visualization	<- Scripts to create exploratory and results oriented visualizations
│ └─ visualize.py	
└─ tox.ini	<- tox file with settings for running tox; see tox.readthedocs.io

Código-Etapas

EDA, Prototipo, Preproducción, Producción

Aspecto	EDA	Productivo
Estructura	Notebook sin organización formal.	Scripts modulares con carpetas organizadas.
Código Reutilizable	Código acoplado, poco modular.	Funciones y clases reutilizables.
Configuración	Parámetros hardcoded en el código.	Configuración externa con archivos .yaml o .json.
Logs y Errores	Uso de print() para debugging.	Sistema de logging robusto con niveles (INFO, ERROR, DEBUG).
Pruebas	Sin pruebas o validaciones.	Pruebas unitarias y de integración.
Escalabilidad	Procesamiento limitado (archivos pequeños, sin paralelismo).	Optimización para grandes volúmenes (e.g., paralelismo, uso de GPU/CPU).
Documentación	Comentarios básicos o inexistentes.	Docstrings detallados y README explicativo.
Automatización	Manual (ejecución interactiva).	Pipelines automáticos (e.g., Prefect, Airflow).

Código-EDA (Exploratory Data Analysis)

Objetivo:	Entender los datos, explorar patrones, identificar problemas y validar hipótesis iniciales.
Estructura del Código:	<ul style="list-style-type: none">• Notebook poco estructurado, con celdas ejecutadas en orden arbitrario.• Código redundante o fragmentado (copiar/pegar es común).• Depuración y visualización inmediatas (print(), matplotlib, seaborn).
Enfoque:	<ul style="list-style-type: none">• Experimentación rápida.• Uso intensivo de visualizaciones.• Pruebas de hipótesis rápidas sin preocuparse por optimización o escalabilidad.
Problemas Comunes:	<ul style="list-style-type: none">• Falta de reproducibilidad. El orden de ejecución puede afectar los resultados.• Código no modular, difícil de reutilizar.• Operaciones no optimizadas.• Falta de manejo de errores y logs.



Código-Prototipo

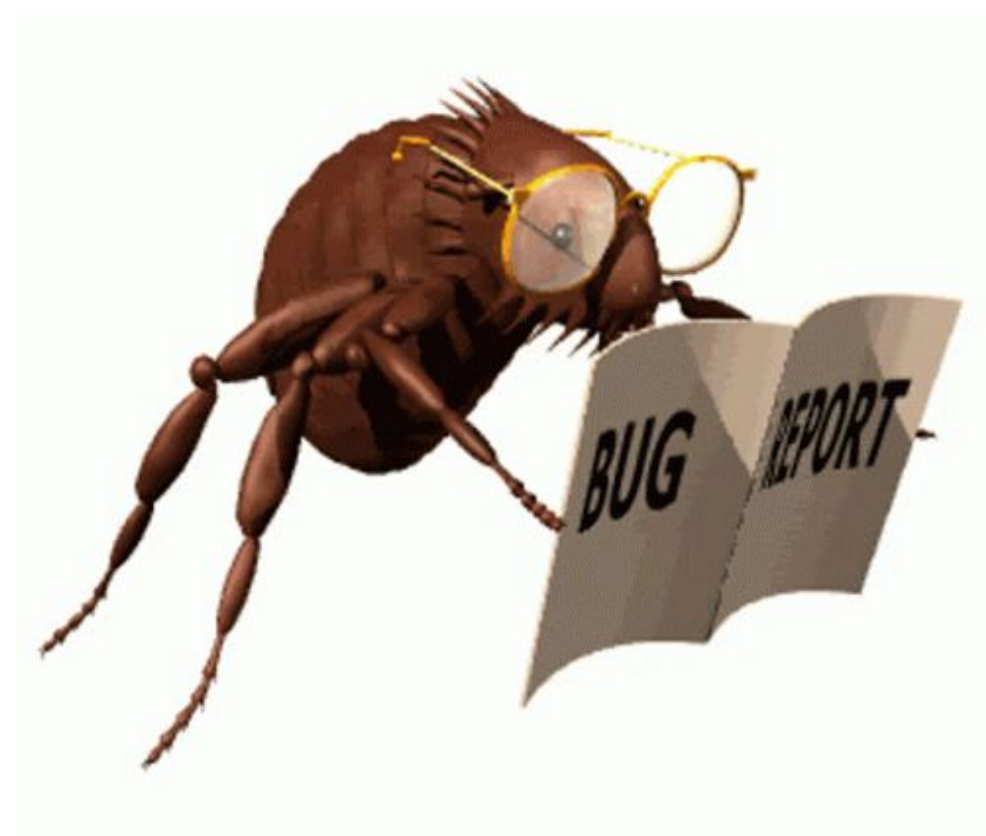
Objetivo:	Probar pipelines iniciales y validar resultados.
Estructura del Código:	<ul style="list-style-type: none">• Código más organizado con funciones reutilizables dentro del notebook.• Separación inicial de las secciones del notebook (EDA, preprocesamiento, modelado, evaluación).• Uso de pipelines rudimentarios (e.g., con scikit-learn o scripts ad-hoc).
Enfoque:	<ul style="list-style-type: none">• Funciones básicas para limpieza y transformación de datos.• Uso de configuraciones fijas (hardcoded) para hiperparámetros y rutas de archivos.
Problemas Comunes:	<ul style="list-style-type: none">• Limitada modularidad: Aún atado al notebook.• Falta de control de versiones: Cambios en datos o código pueden no rastrearse.• Errores no manejados: Dependencia excesiva en la ejecución interactiva.

Código-Preproducción

Objetivo:	Convertir el prototipo en un flujo reproducible y parcialmente automatizado.
Estructura del Código:	<ul style="list-style-type: none">• Código dividido en scripts o módulos (e.g., <code>data_preprocessing.py</code>, <code>train_model.py</code>).• Incorporación de configuraciones externas (<code>config.yaml</code> o <code>.json</code>).• Uso de herramientas de pruebas como Pytest para validar partes críticas del flujo.
Enfoque:	<ul style="list-style-type: none">• Modularidad: Separar claramente las etapas (carga de datos, preprocesamiento, modelado).• Uso de bibliotecas especializadas (e.g., joblib para guardar modelos).• Implementación inicial de logs (e.g., logging).• Control de versiones del código (e.g., Git).• Manejo de errores básicos (try/except en funciones críticas).
Problemas Comunes:	<ul style="list-style-type: none">• Falta de pruebas exhaustivas: Limitada cobertura de pruebas unitarias.• Manejo limitado de datos: No considera grandes volúmenes o datos en tiempo real.

Código-Producción

Objetivo:	Crear un sistema robusto, reproducible y escalable, listo para implementarse en producción.
Estructura del Código:	<ul style="list-style-type: none">• Código organizado en módulos y paquetes• Uso de pipelines automatizados con herramientas como scikit-learn Pipeline, Prefect, o Dagster.• Gestión de dependencias con requirements.txt (herramientas como <i>poetry</i>).
Enfoque:	<ul style="list-style-type: none">• Escalabilidad: Procesamiento eficiente (e.g., paralelismo, uso de recursos en la nube).• Robustez: Manejo completo de errores, logs, y validaciones de entrada/salida.• Automatización: Integración de workflows con cron jobs o sistemas de orquestación (e.g., Airflow).
Problemas Comunes:	<ul style="list-style-type: none">• --- S.O.S. de un fin de semana



Código-Proyecto Final

Obligatorio	Limpieza y claridad de código. Comments correspondientes a un código. La explicación más exhaustiva debe ir en el reporte como parte de documentación. Modularidad
Importante	<ul style="list-style-type: none">• Código más organizado con funciones reutilizables dentro del notebook.• Separación inicial de las secciones del notebook (EDA, preprocesamiento, modelado, evaluación).• Manejo de errores básicos (try/except en funciones críticas).• Manejo de warnings
Favorable	<ul style="list-style-type: none">• No imprimir nada por pantalla. Todo con salida a un archivo• Manejo de MLflow o similar para realizar el seguimiento de métricas y modelos

Código-Manejo de Errores y Warnings

Uso de bloques try/except para capturar errores críticos y proporcionar mensajes útiles. ([8. Errors and Exceptions — Python 3.11.10 documentation](#))

Warnings: Filtrar o personalizar los warnings para evitar ruido innecesario. ([warnings — Warning control — Python 3.11.10 documentation](#))

Logs: Implementar un sistema de logging con niveles como INFO, WARNING, ERROR y DEBUG. Los logs se guardan en archivos para referencia posterior. ([logging — Logging facility for Python — Python 3.11.10 documentation](#), [Logging HOWTO — Python 3.11.10 documentation](#))

Código-Registro


[MLflow Overview](#)

[LLMs](#)

[Fine-Tuning Transformers with MLflow for Enhanced Model Management](#)

Tecnologías y herramientas

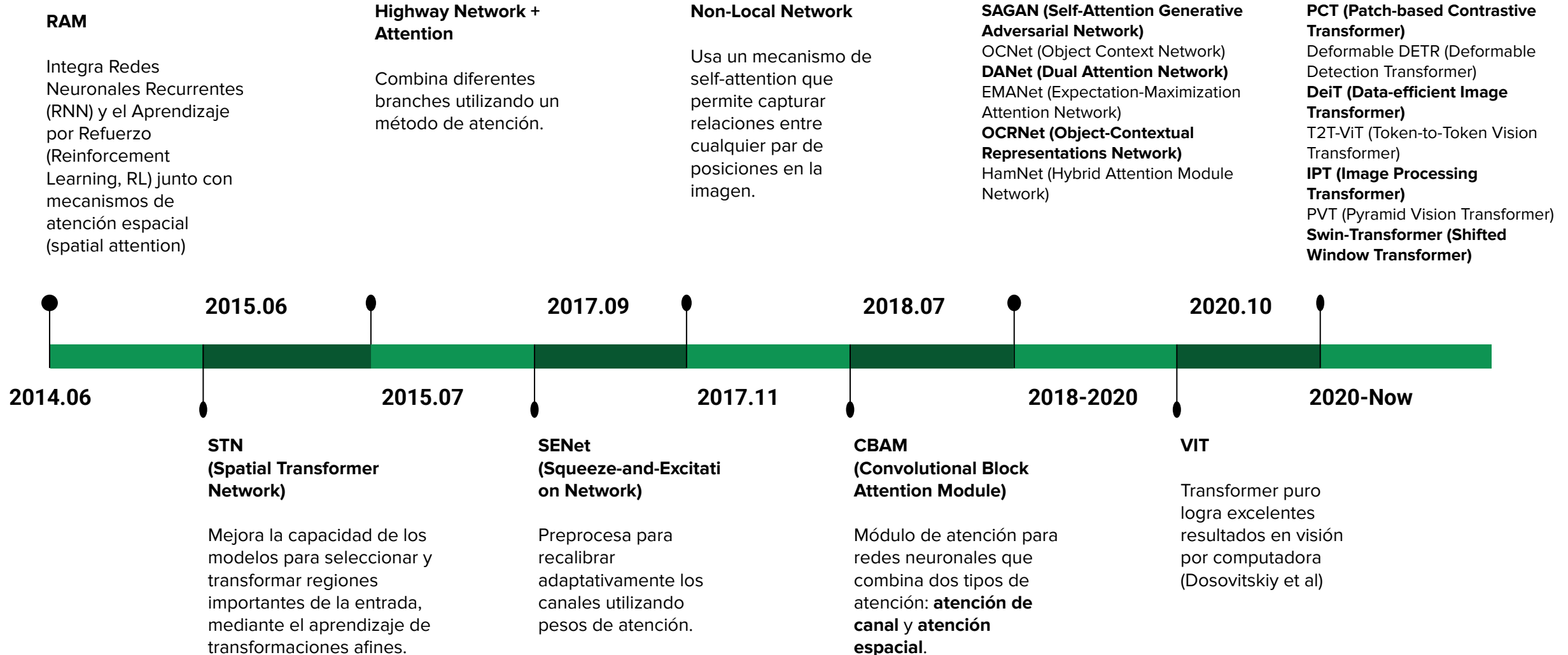


 PyTorch



Hugging Face

Evolución de Transformers en NLP y Computer Vision



Origen de ViT

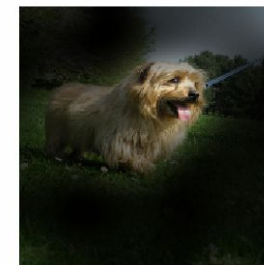
El Vision Transformer (ViT) oficialmente fue presentado en 2021 en el artículo "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale" por Alexey Dosovitskiy et al.

- ❑ 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

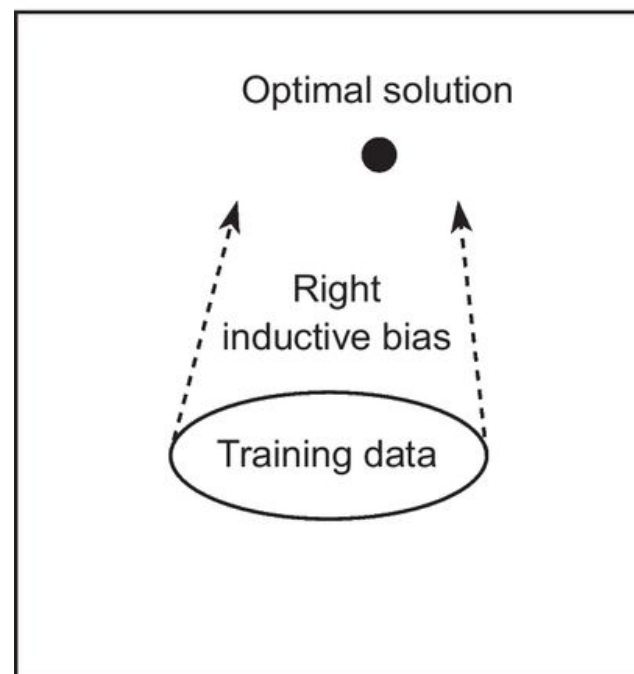


Bias inductivo

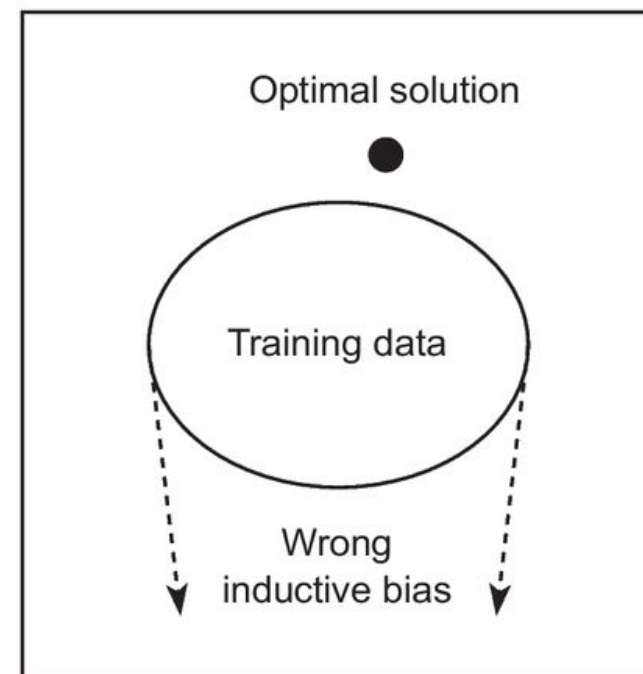
Todo modelo de aprendizaje automático necesita un diseño arquitectónico y suposiciones iniciales sobre los datos, lo que constituye un **sesgo inductivo**.

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.

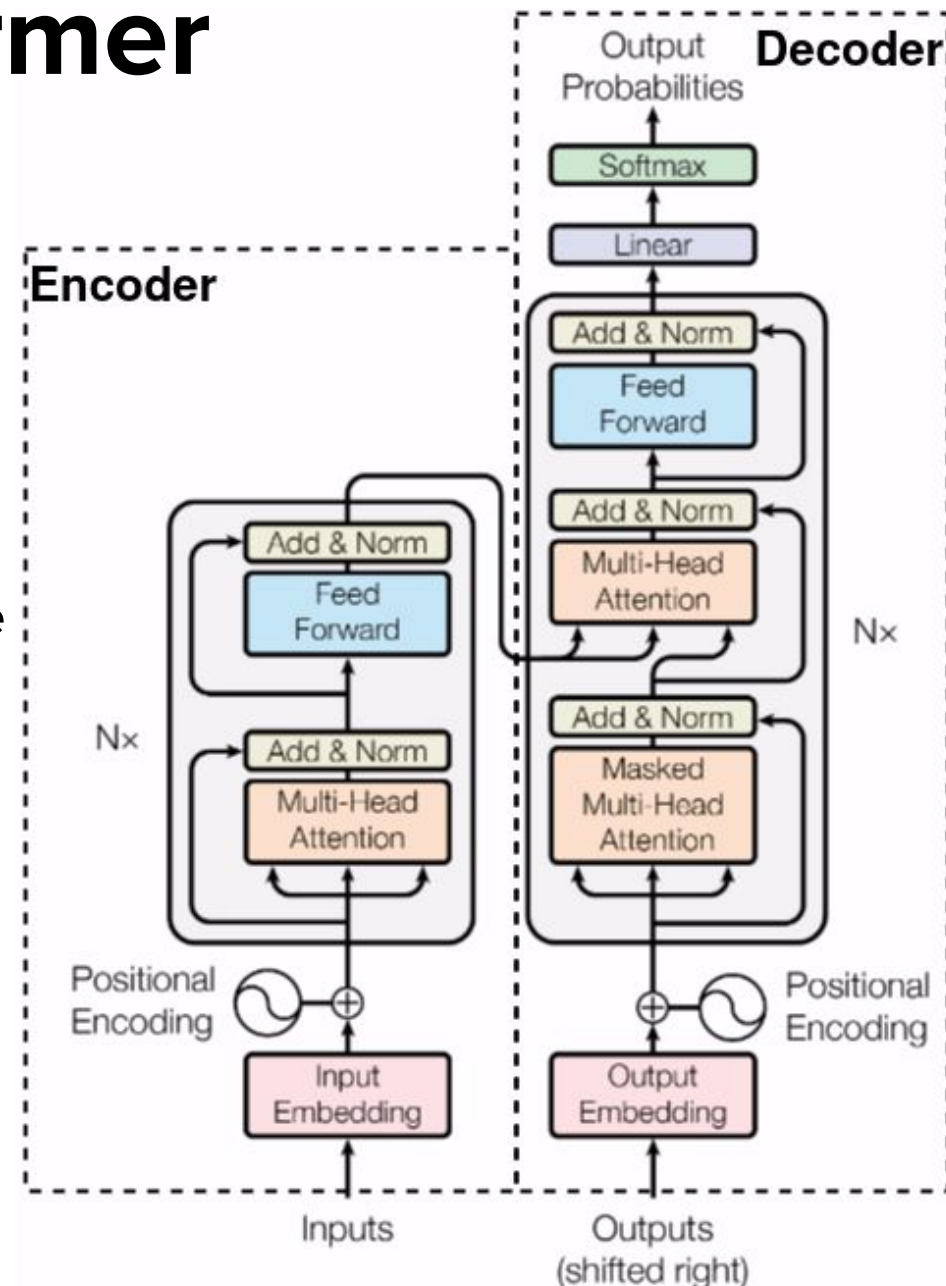


Perjudica la generalización, incluso en un contexto con muchos datos.

Arquitectura del Transformer

Propuesto por Vaswani et al. en su paper "[Attention Is All You Need](#)" (2017), es una arquitectura de redes neuronales principalmente para tareas de procesamiento de lenguaje natural (NLP).

Su innovación clave es el uso del **mecanismo de atención**, que permite al modelo **enfocarse** en diferentes partes de la entrada de manera dinámica y eficiente, sin depender de la estructura secuencial de modelos como las redes recurrentes (**RNNs**) o **LSTMs**.

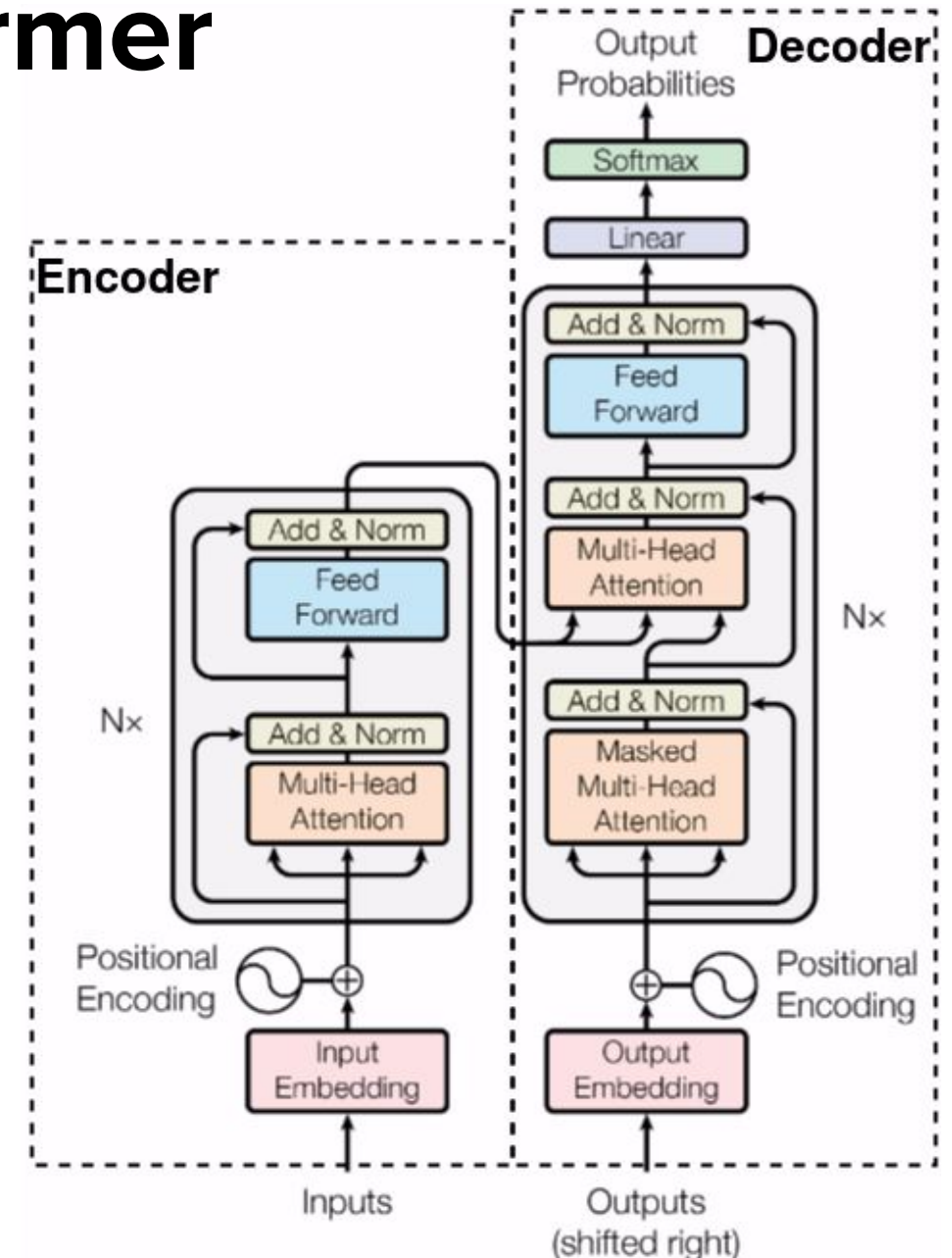


Arquitectura del Transformer

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.



Scaled Dot-Product Attention

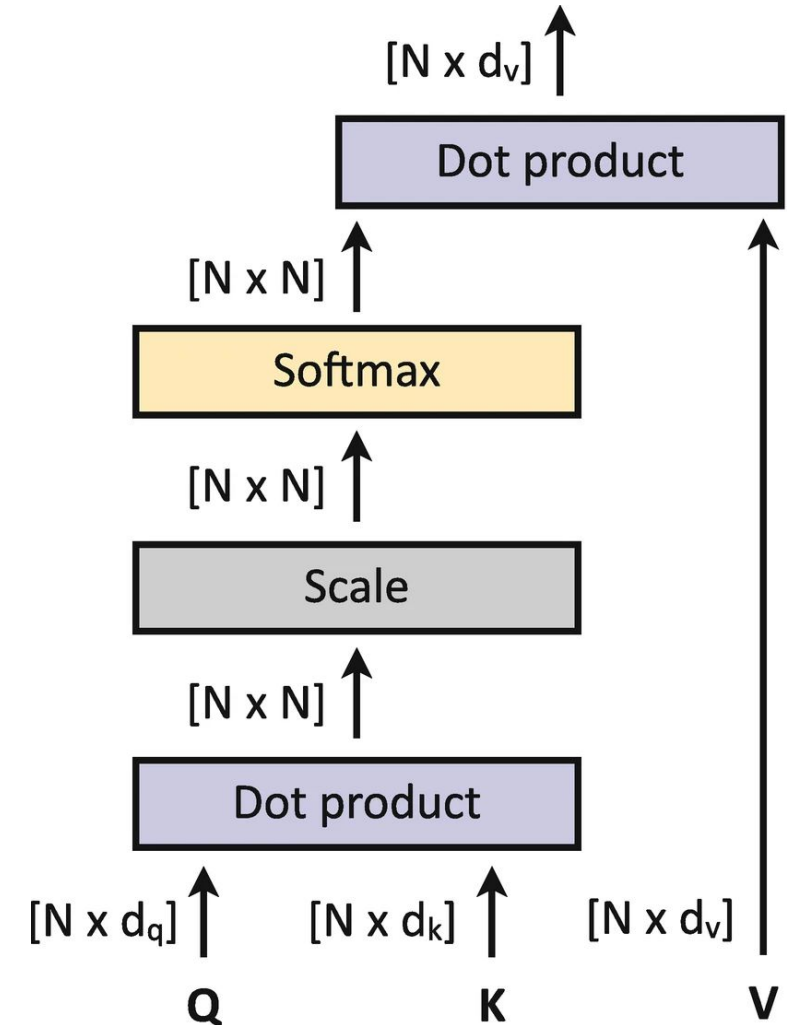
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \cdot V$$

- Dadas dos listas de tokens, X e Y, la atención codifica información de Y en X, donde N es la longitud de las entradas X e Y.

Luego, definimos la consulta Q, la clave K y el valor V como:

$$Q = XW^Q \quad K = YW^K \quad V = YW^V$$

- La atención se utiliza para permitir que cada parche de la imagen (X) "vea" y "preste atención" a otros **parches (Y)** usando el producto punto. Luego, se normaliza con la función softmax. Para evitar gradientes pequeños con grandes valores de d_k , se escala dividiendo por la raíz cuadrada de d_k .
- Cross Attention (CA) versus Self Attention (SA). Para SA, $X=Y$



Arquitectura del Transformer

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).

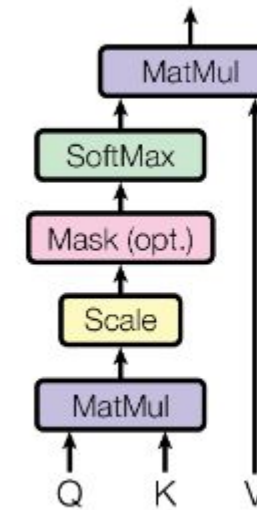
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \cdot V$$

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.

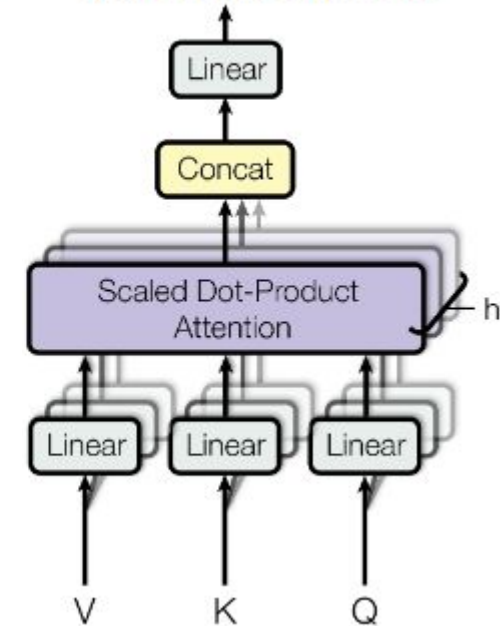
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h) W^O$$

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

Scaled Dot-Product Attention



Multi-Head Attention

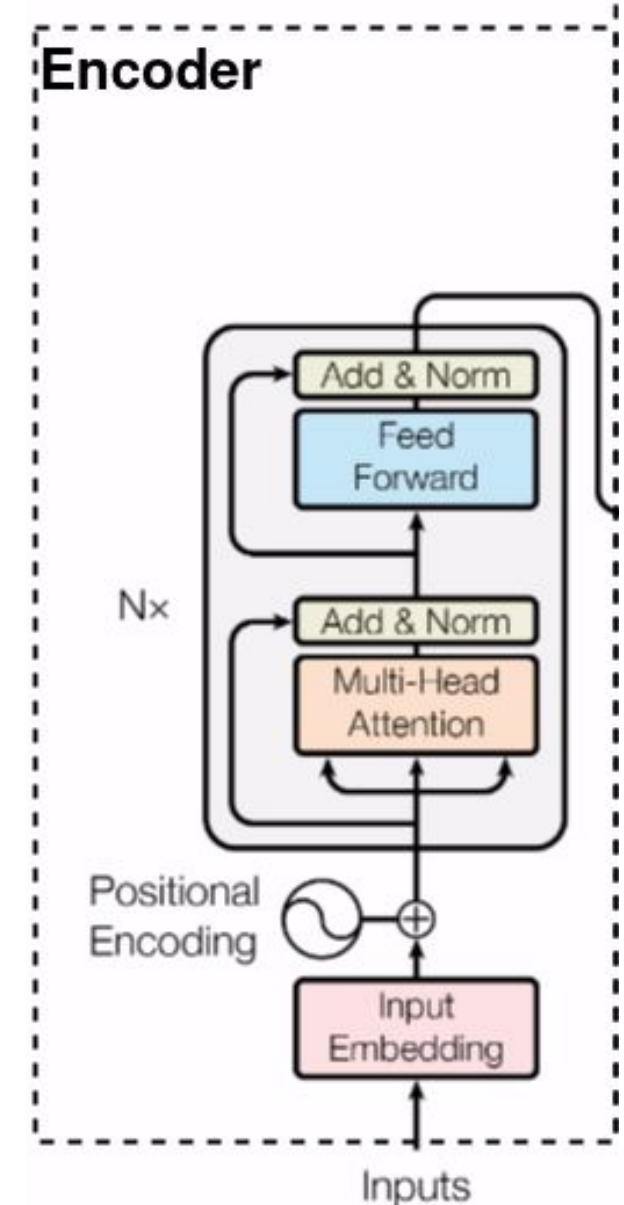


Arquitectura del Transformer

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.

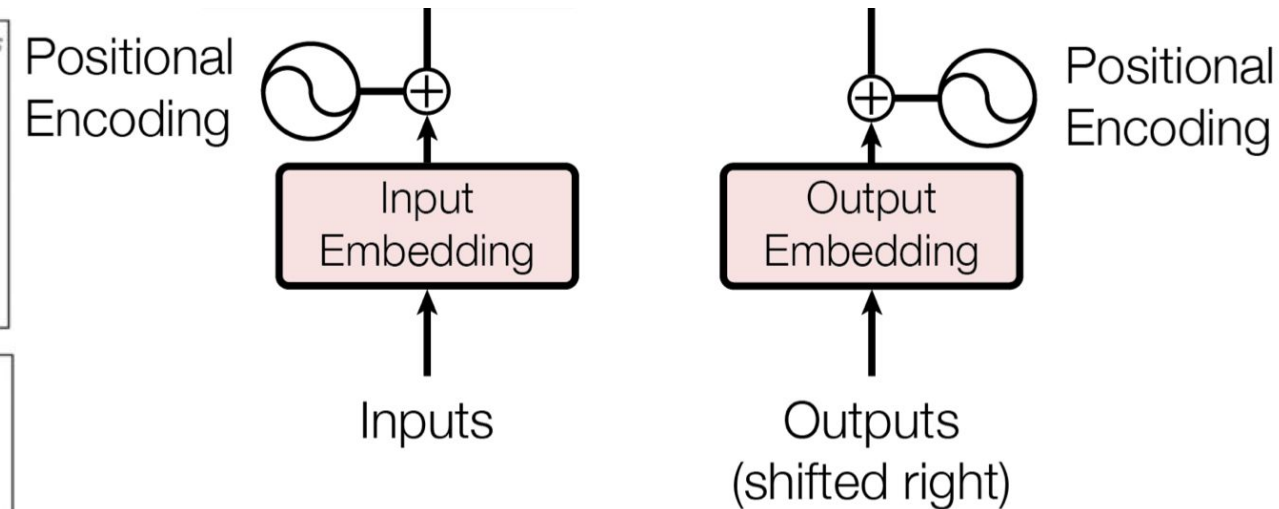
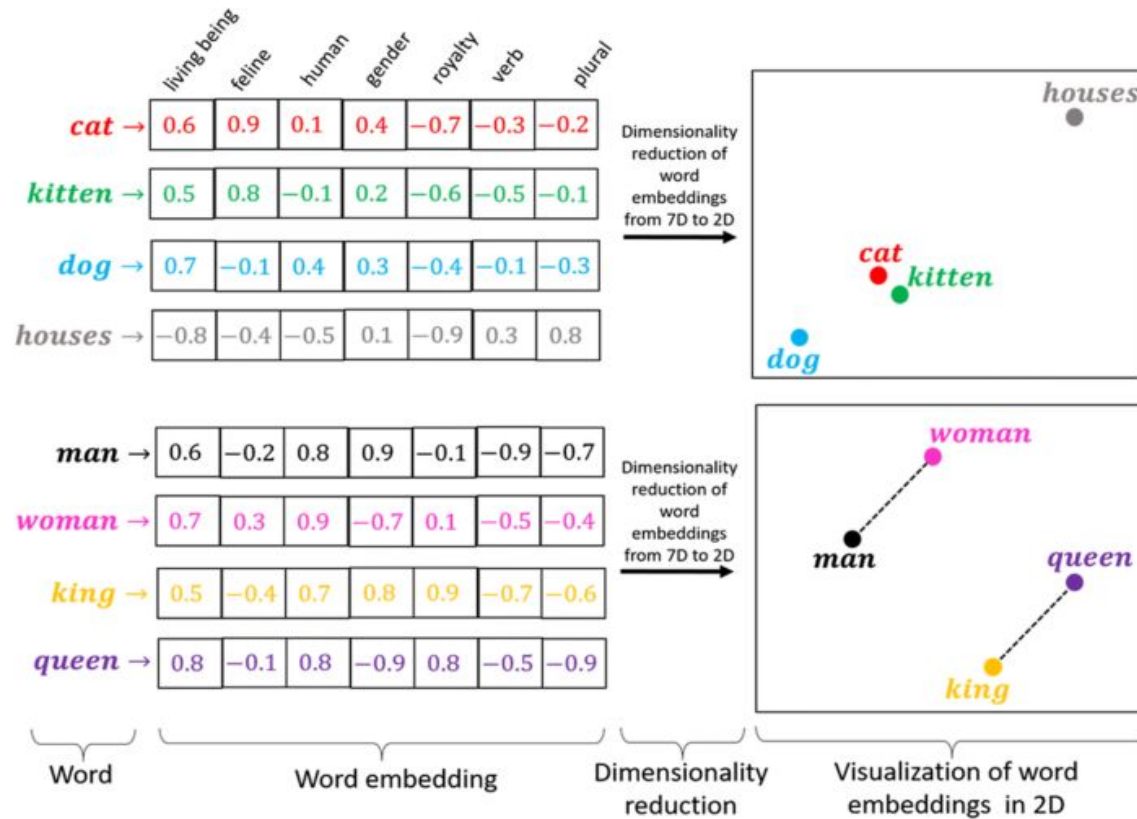
Iteración en Capas (Nx): El proceso de MHA seguido de la red feed-forward se repite múltiples veces (Nx) en el encoder, lo que permite al modelo aprender representaciones más ricas de la imagen.



Transformers Embeddings

Embeddings: Los tokens se transforman en vectores y se convierten en embeddings. Estos embeddings son vectores de un espacio latente permitiendo la relación entre tokens.

Codificación Posicional (Positional Encoding): Los Transformers desconocen el orden secuencial. Se añaden embeddings posicionales para que el modelo sepa la posición de cada token dentro de la secuencia.

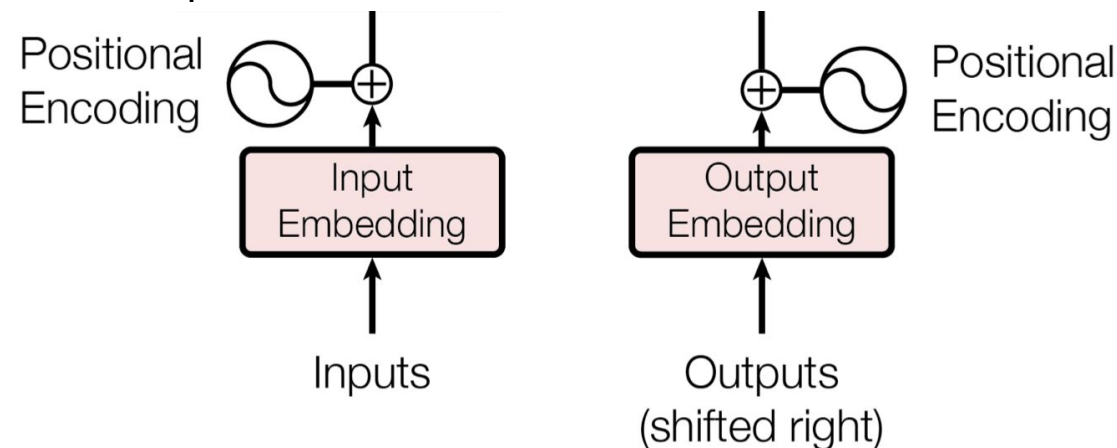


Positional Embeddings

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.

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'



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

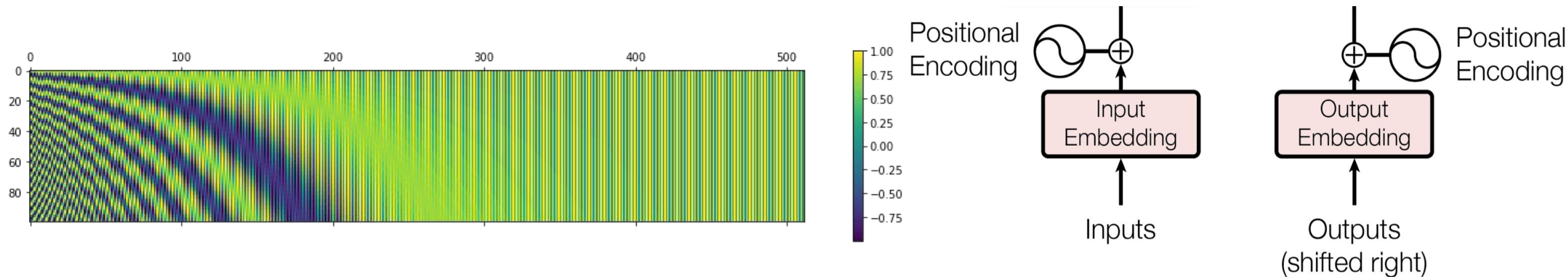
$$PE_{(pos, 2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

Positional Embeddings

[Positional Embeddings in Transformers Explained](#)

[Adding vs. concatenating positional embeddings & Learned positional encodings](#)

[Gentle introduction to positional encoding in transformer models](#)



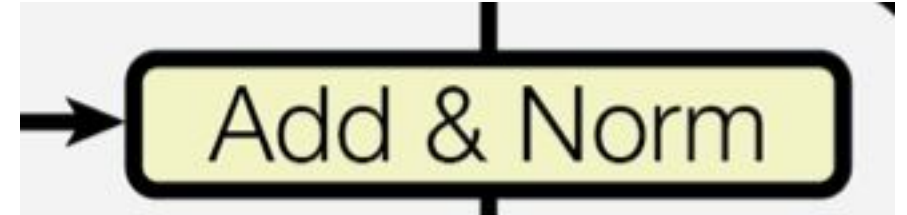
$$PE_{(pos, 2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \quad PE_{(pos, 2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

LayerNorm

Suma y Normalización (Add & Norm): Se realiza una operación de suma y normalización en los resultados para **estabilizar y mejorar** el aprendizaje. Esto estabiliza y optimiza la convergencia durante entrenamiento.

[Torch LayerNorm](#)

[Attention-is-all-you-need-layer-norm](#)

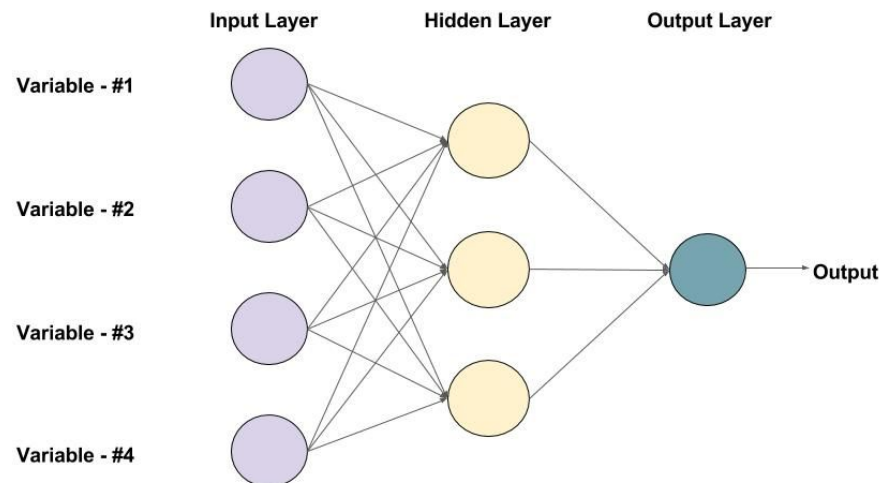


$$y = \frac{x - \mathbb{E}[x]}{\sqrt{\text{Var}[x] + \epsilon}} * \gamma + \beta$$

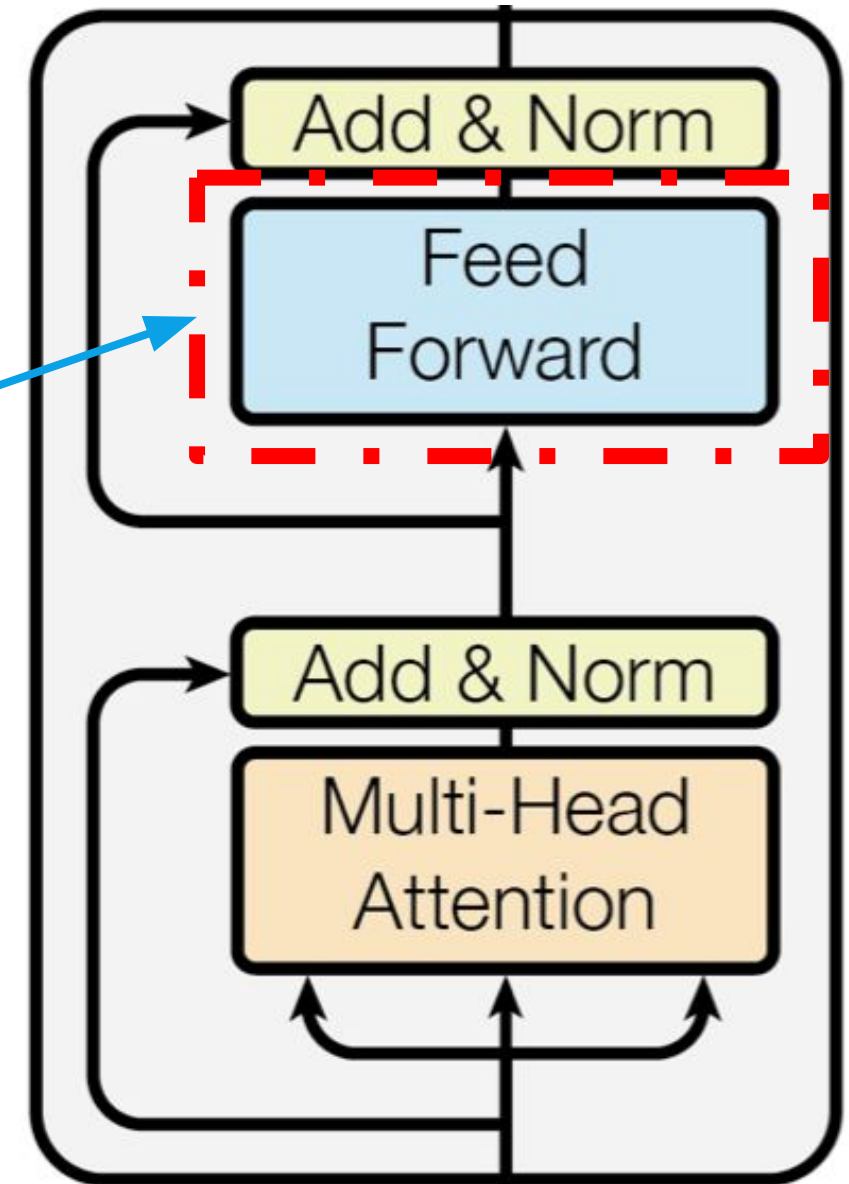
FeedForward (MLP)

Red Lineal Feed-Forward (Feed Forward): Cada capa de atención es seguida por una **red lineal**, que extrae características complejas internas e introduce no-linealidad.

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$



An example of a Feed-forward Neural Network with one hidden layer (with 3 neurons)

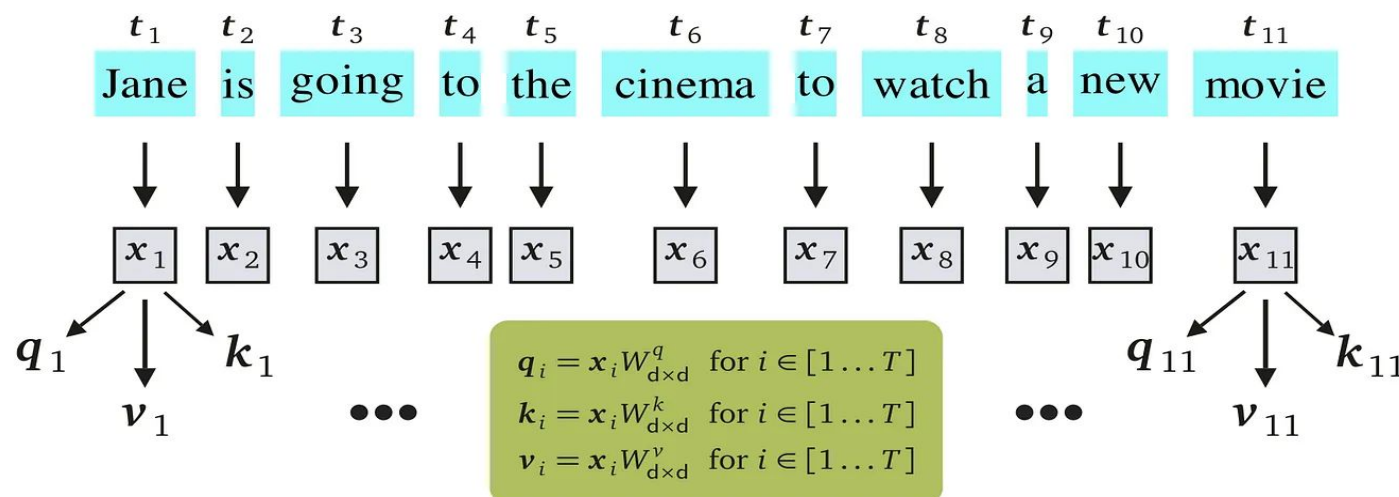


Scaled Dot-Product Attention

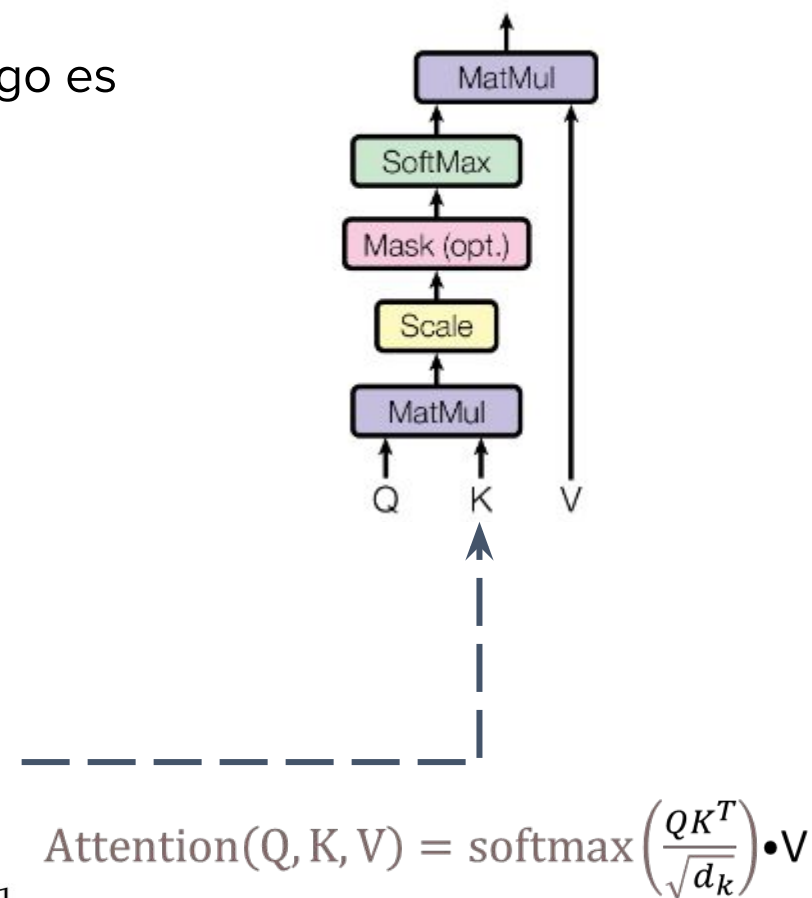
El mecanismo de atención consta de operaciones simples (multiplicación matricial, escalamiento y máscaras). Sin embargo es computacionalmente costoso $O(N^2)$.

Formalmente los vectores son:

- $Q \text{ y } K \in \mathbb{R}^{dk}$
- $V \in \mathbb{R}^{dv}$

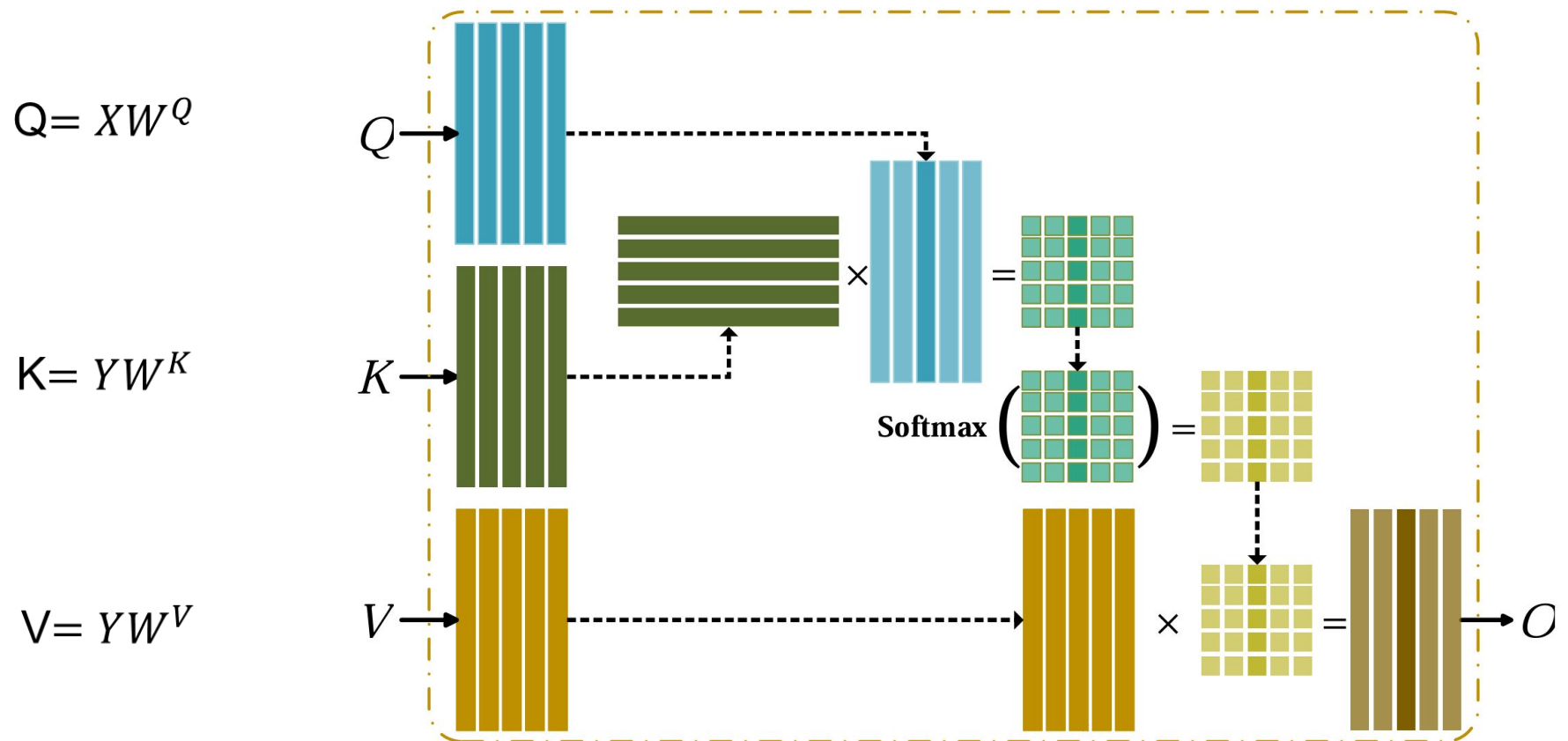


Scaled Dot-Product Attention



Scaled Dot-Product Attention

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



Multi-head Self-Attention (MSA)

$$\begin{aligned} \text{MCA}(\mathbf{X}, \mathbf{Y}) &= \text{Concat}(\text{head}_1(\mathbf{X}, \mathbf{Y}), \dots, \text{head}_h(\mathbf{X}, \mathbf{Y})) \mathbf{W}^O, \\ \text{head}_i(\mathbf{X}, \mathbf{Y}) &= \text{CA}(\mathbf{X}, \mathbf{Y}), \forall i \in \{1, h\}. \end{aligned}$$

Si $\mathbf{X}=\mathbf{Y}$:

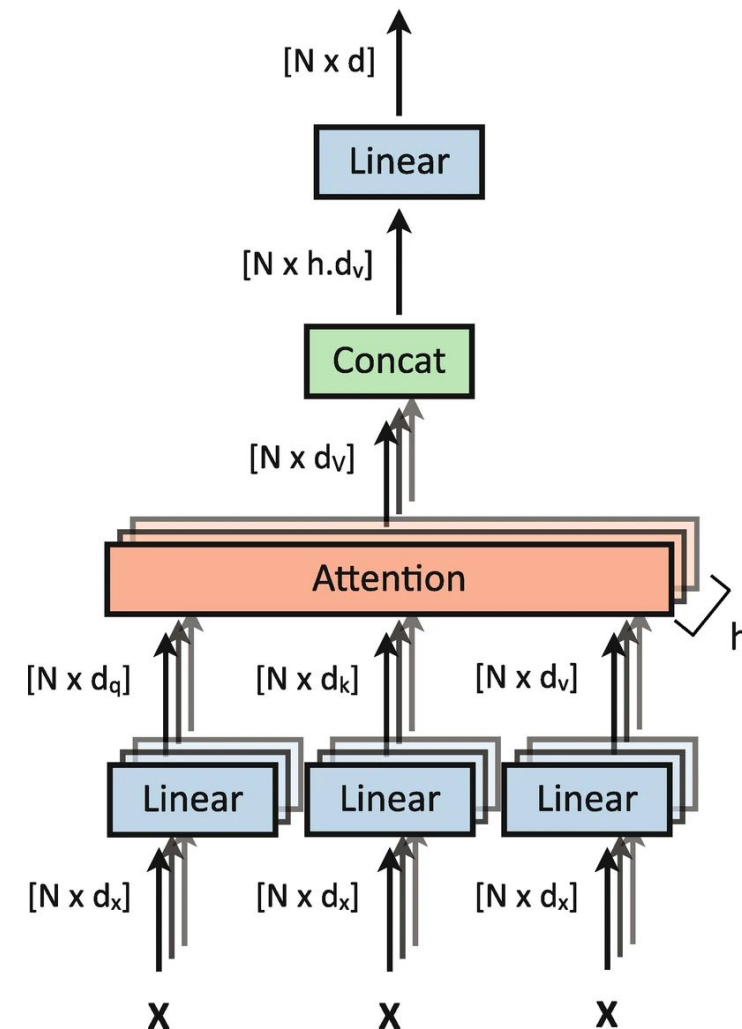
$$\begin{aligned} \text{MSA}(\mathbf{X}) &= \text{Concat}(\text{head}_1(\mathbf{X}), \dots, \text{head}_h(\mathbf{X})) \mathbf{W}^O, \\ \text{head}_i(\mathbf{X}) &= \text{SA}(\mathbf{X}), \forall i \in \{1, h\}, \end{aligned}$$

Donde head es:

$$\mathbf{H}_i = \mathbf{W}_v^{(i)} \mathbf{V} \times \text{softmax}\left((\mathbf{W}_k^{(i)} \mathbf{K})^T \mathbf{W}_q^{(i)} \mathbf{Q}\right) \in \mathbb{R}^{q_i \times n}$$

Recomendable ver el ejemplo de

[Transformers-for-NLP-and-Computer-Vision-3rd-Edition](#)



Mecanismo de Atención Vanilla en ViT

Scaled Dot-Product Attention

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

Multi-Head Self Attention (MSA):

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

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

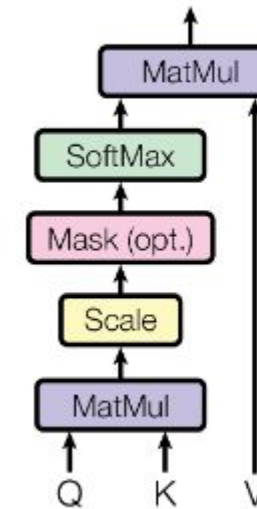
MSA es computacionalmente costoso $O(N^2)$. Es ineficiente en **aplicaciones** de visión artificial en tiempo real. Su complejidad temporal está dada por:

$$\Omega(\text{MSA}) = 4hwc^2 + 2(hw)^2C$$

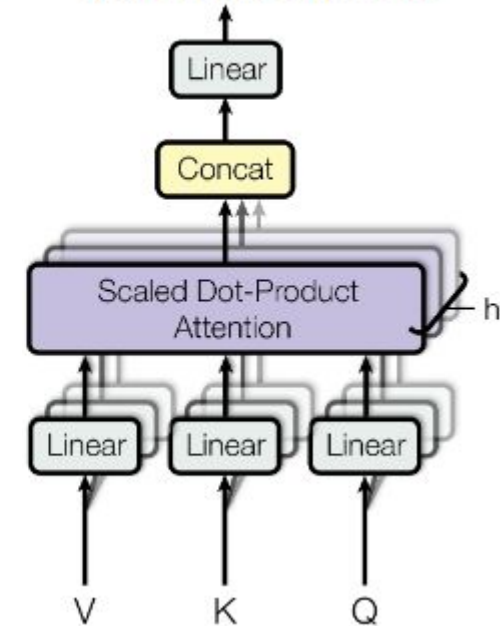
Donde hwc corresponden a (Height, Weight, Channel).

El mecanismo de atención es uno de los mayores cuellos de botella en modelos basados en Transformers.

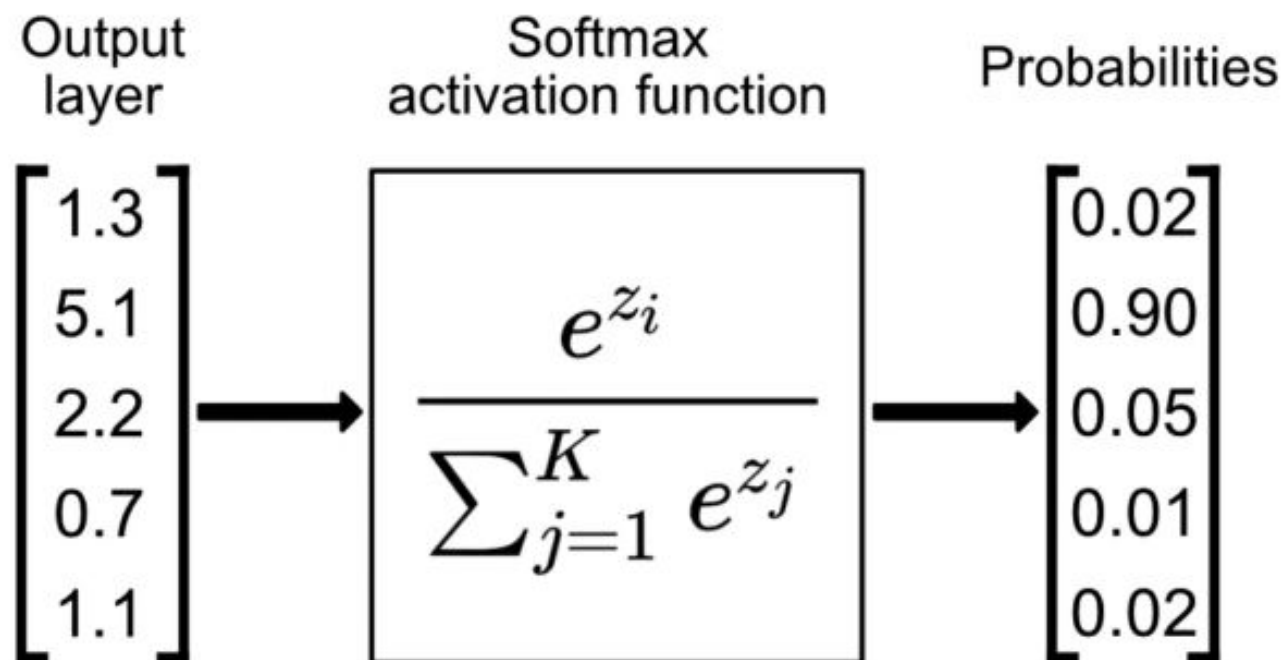
Scaled Dot-Product Attention



Multi-Head Attention



Escalamiento y Softmax



Si la dimensión d_k es grande, entonces $Q \cdot K$ puede dar como resultado valores muy altos. Eso hace que **softmax** tenga gradientes muy pequeños, dificultando el aprendizaje.

Los valores grandes pueden hacer que la *softmax* sea excesivamente “confiada” y puede desestabilizar el entrenamiento.

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

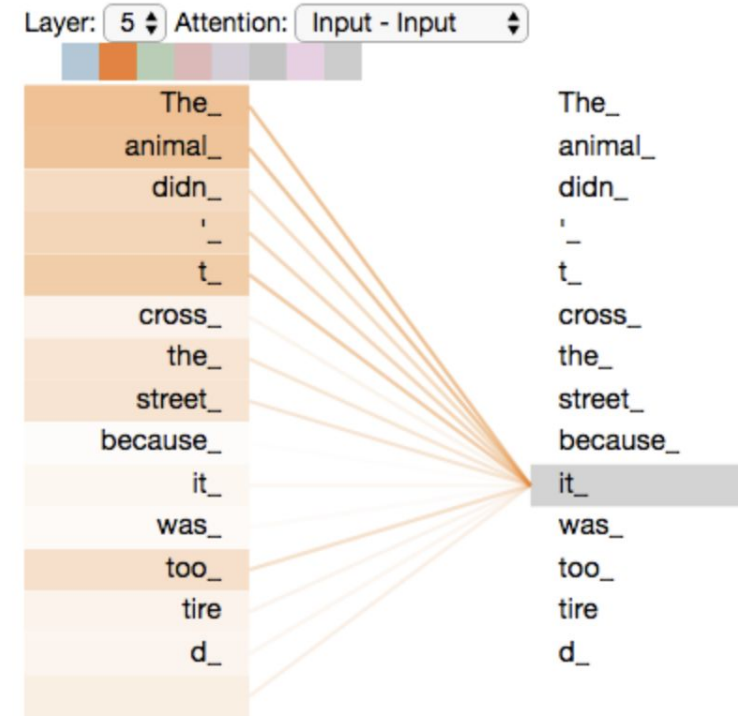
Visualizando Attention en ViT

[Unofficial Walkthrough of Vision Transformer](#)

Recomendable leer!

[ViT explainability](#)

[timm attention visualization](#)



The Annotated Transformer

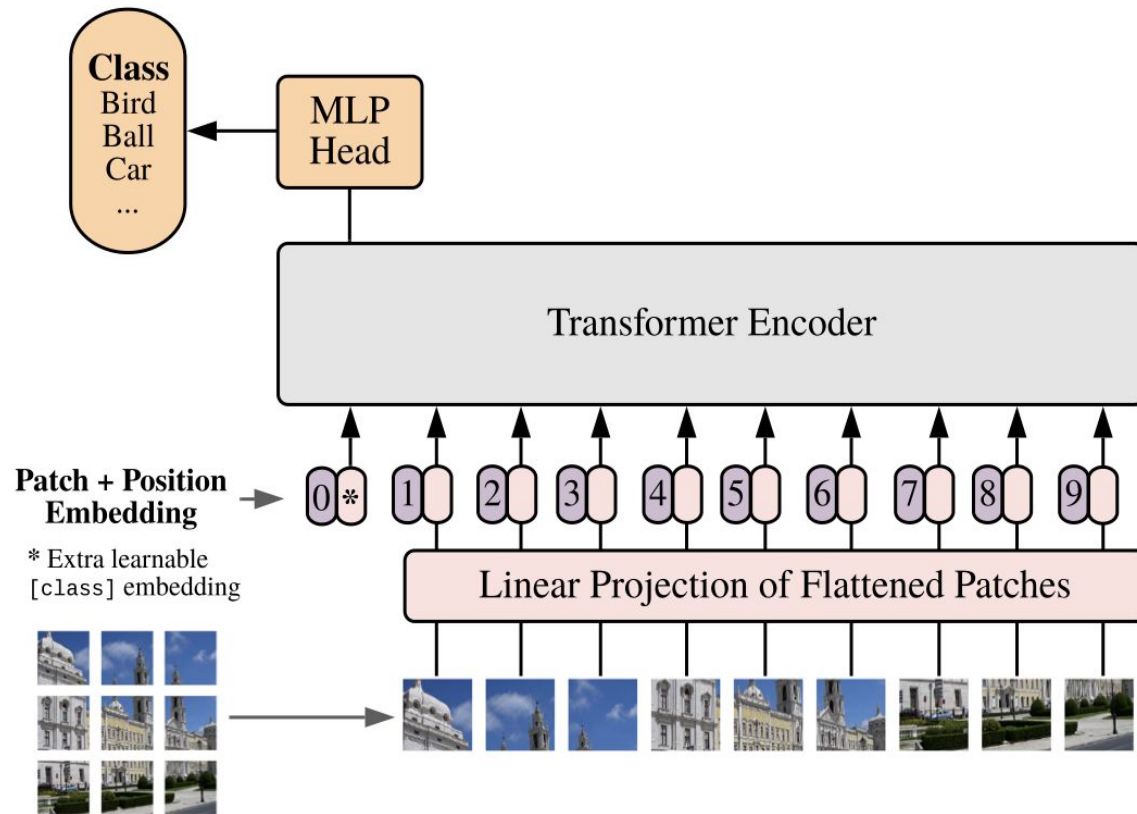
[The annotated Transformer](#) es una de las primeras guías de implementación y explicativas acerca del Transformer.

Recomendable leer!

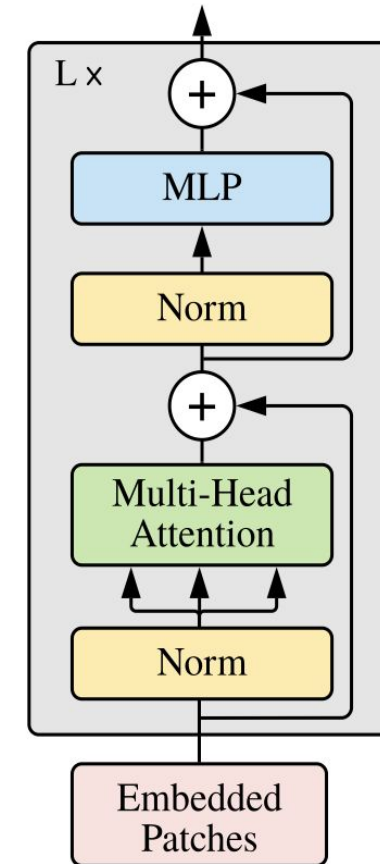
Arquitectura ViT

Arquitectura ViT

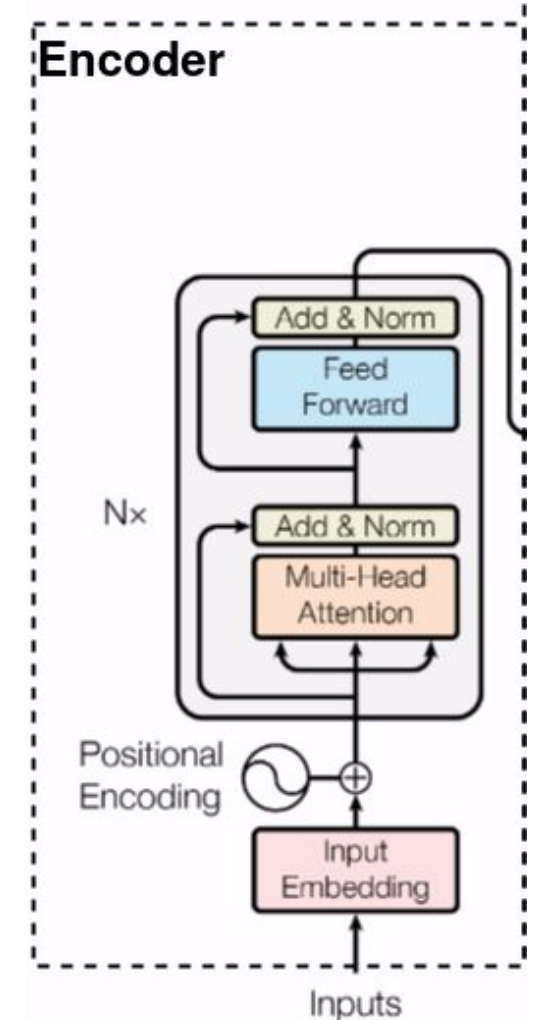
Vision Transformer (ViT)



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

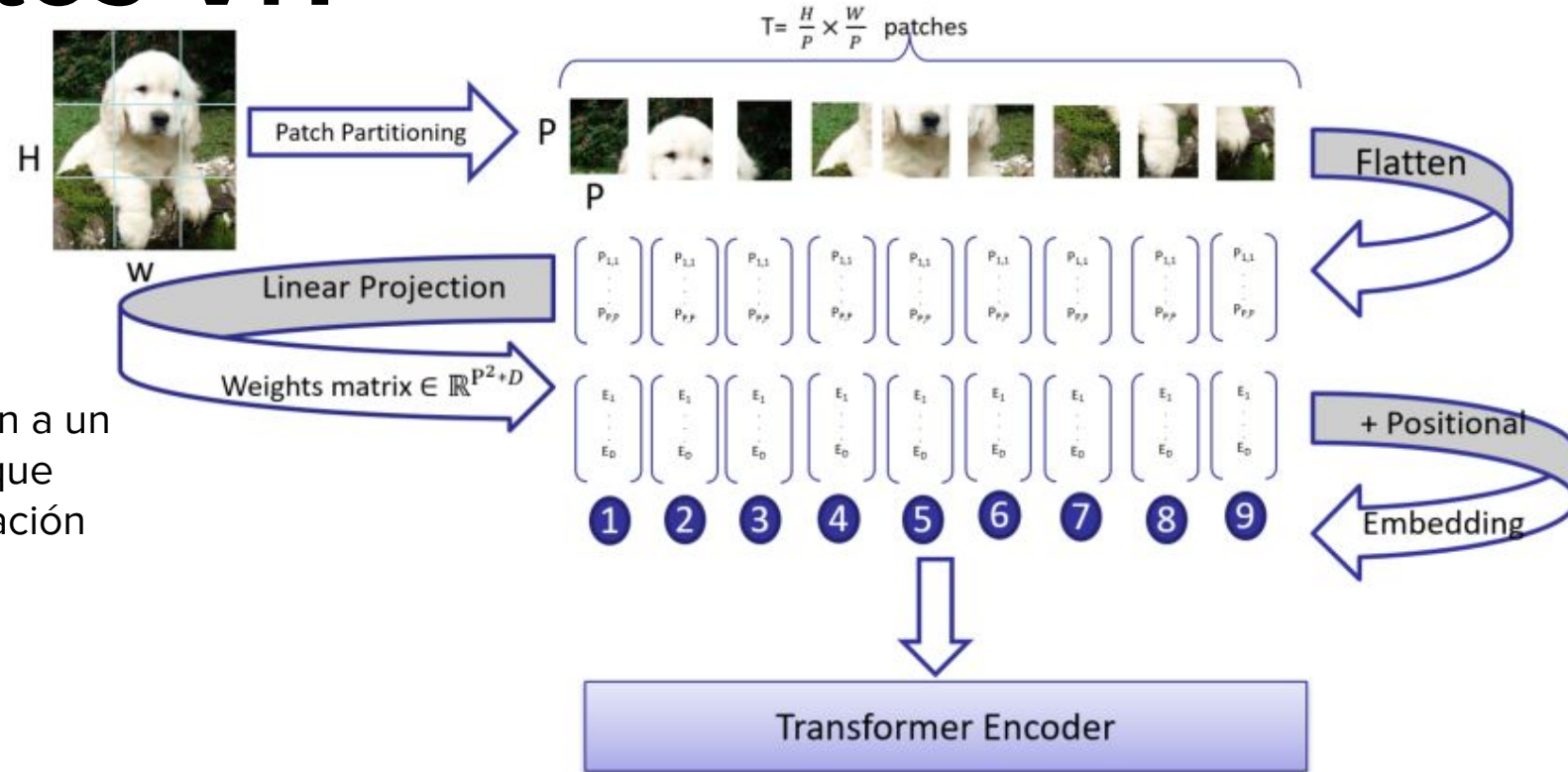
Imagen → parches (patches) → vector

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.

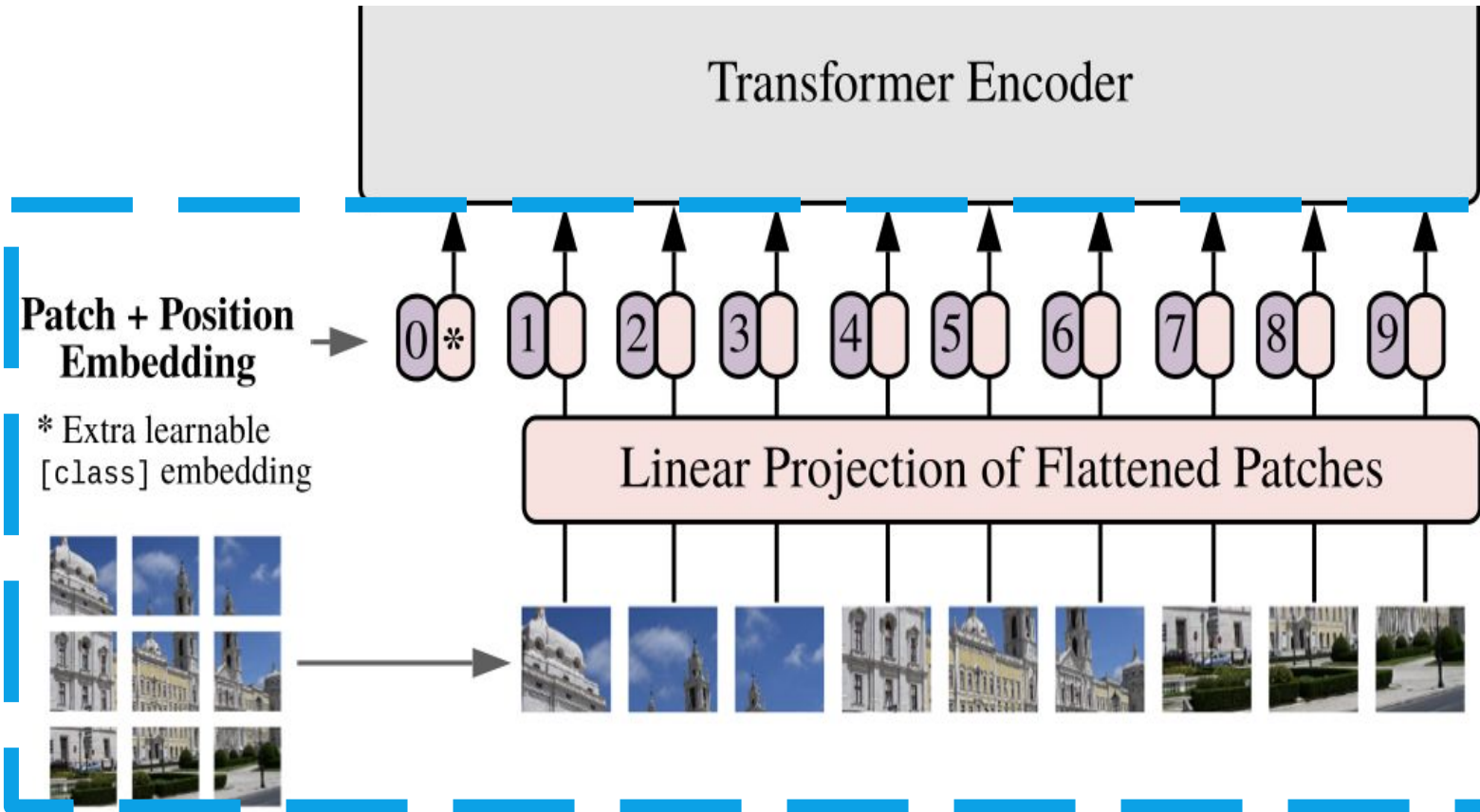
Positional Embedding

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.



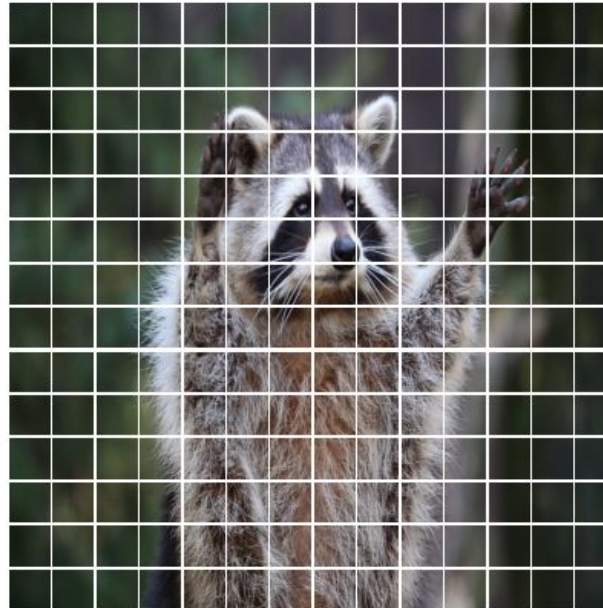
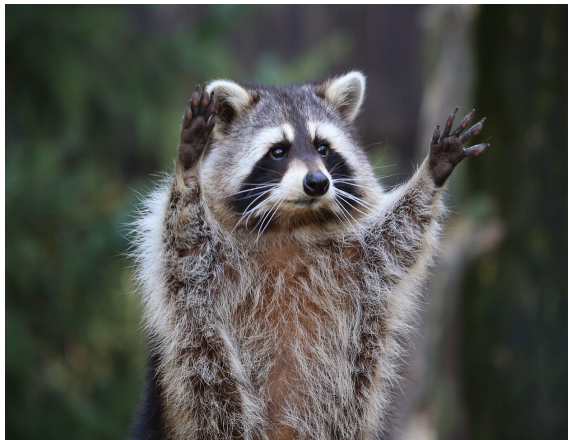
Fuente: [Converting weights of 2D Vision Transformer for 3D Image Classification - DLMA: Deep Learning for Medical Applications - BayernCollab \(dvv.bayern\)](#)

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.

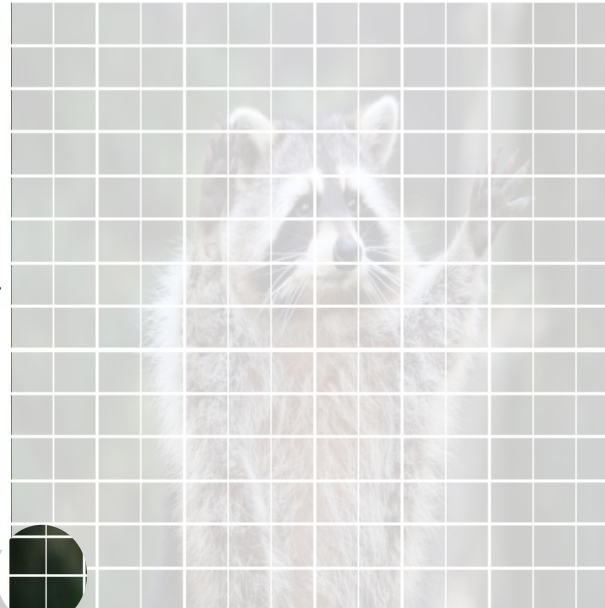
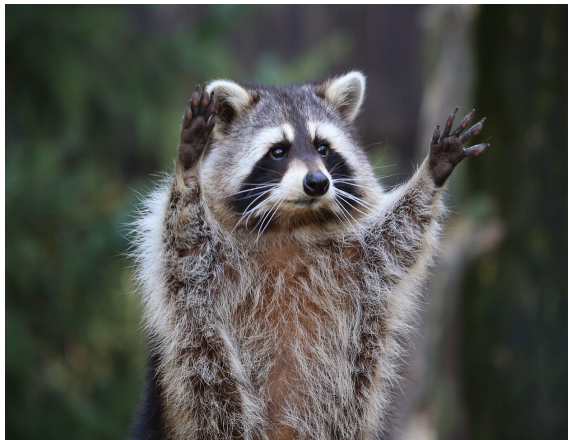
Patch Embeddings



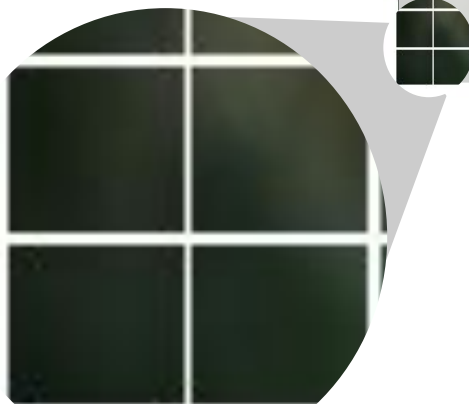
Sea una imagen $x \in \mathbf{R}^{H \times W \times C}$, donde H , W y C representan la altura, el ancho y el número de canales, respectivamente.

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

Patch Embeddings



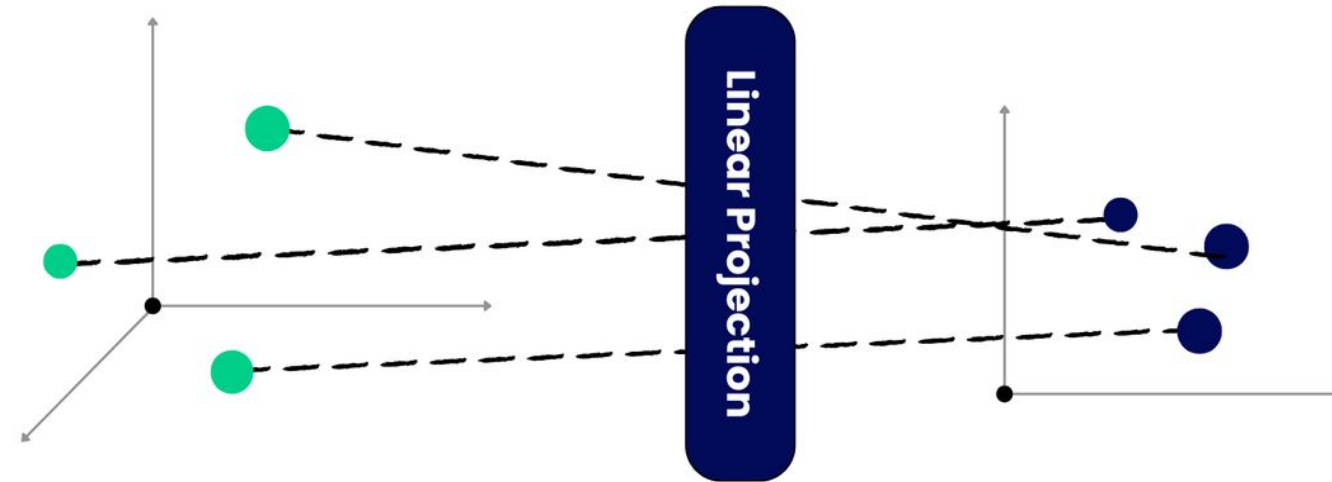
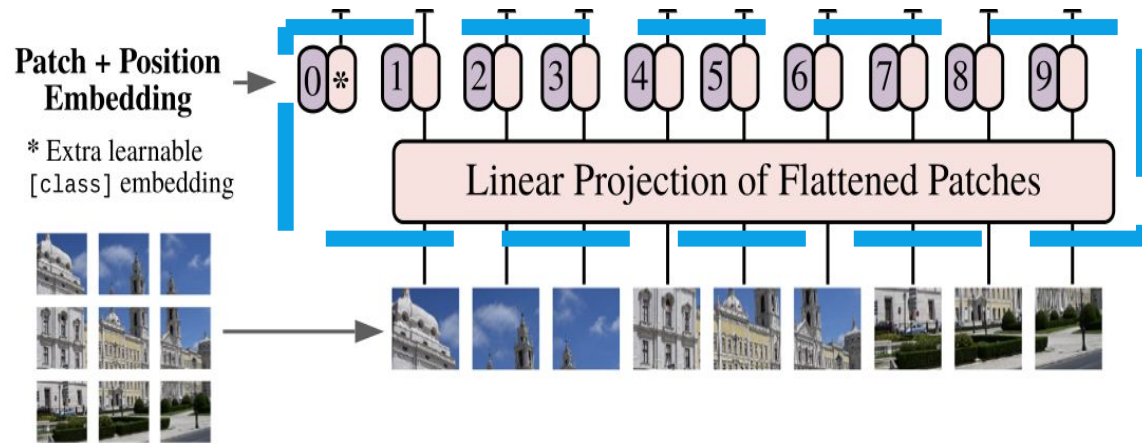
Cada patch
tiene un
área de P^2 .



$N = HW/P^2$ $N \in \mathbb{Z}$, 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 FeedForward.



```
nn.Linear(patch_dim, embed_dim)
```

TP-I

El trabajo práctico se encuentra en el [GitHub](#) de la materia CEIA-ViT:

Plazo de entrega antes de la clase 3.

*Tomemos **15 minutos** en grupos para realizar el primer punto del ejercicio.*

Instrucciones:

Modificar los parámetros: cambiar el tamaño de los parches y la cantidad de dimensiones del embedding. Investigar y describir las ventajas y desventajas de tener más o menos parches/dimensiones.

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:

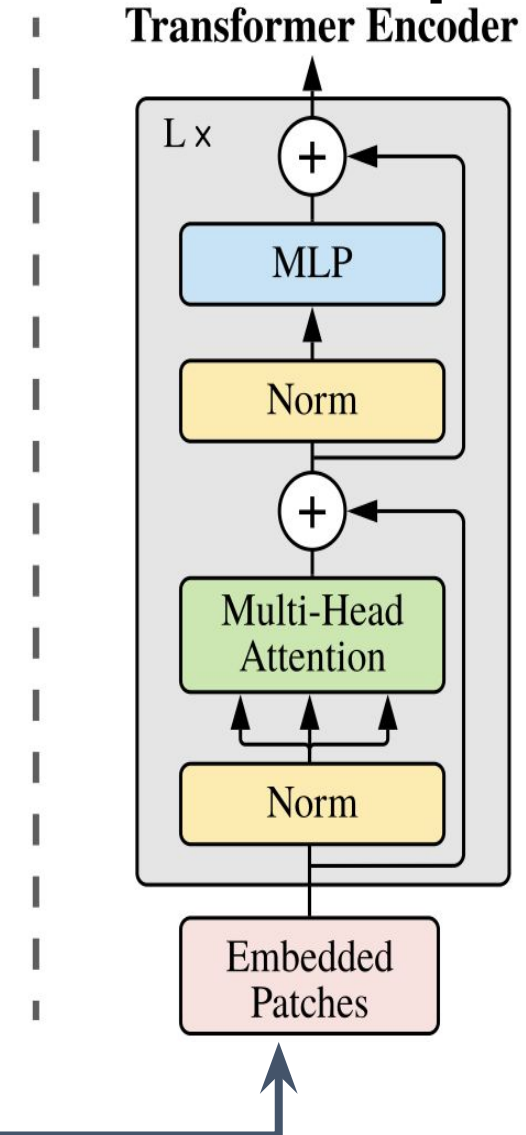
\mathbf{X} la matriz de patch embeddings.

\mathbf{P} la matriz de positional embeddings.

\mathbf{Z} la matriz de entrada al Transformer, donde \mathbf{B} es el tamaño del patch.

Entonces la entrada al encoder es:

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

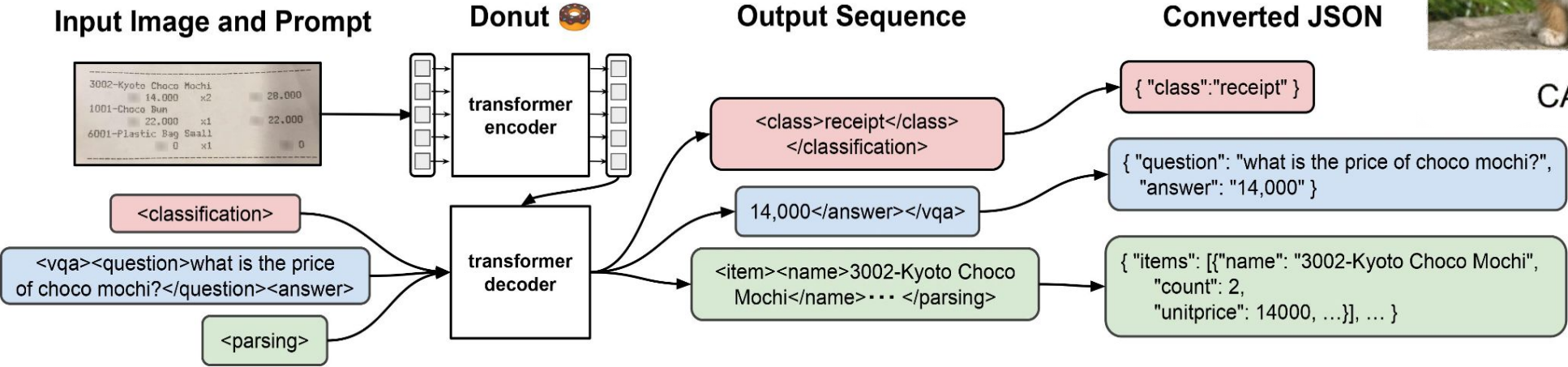
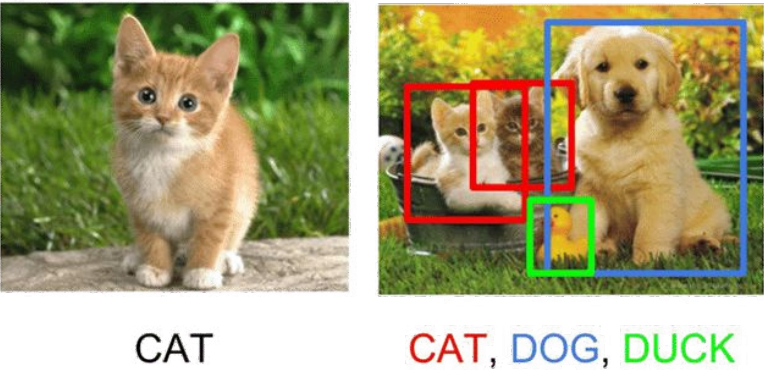
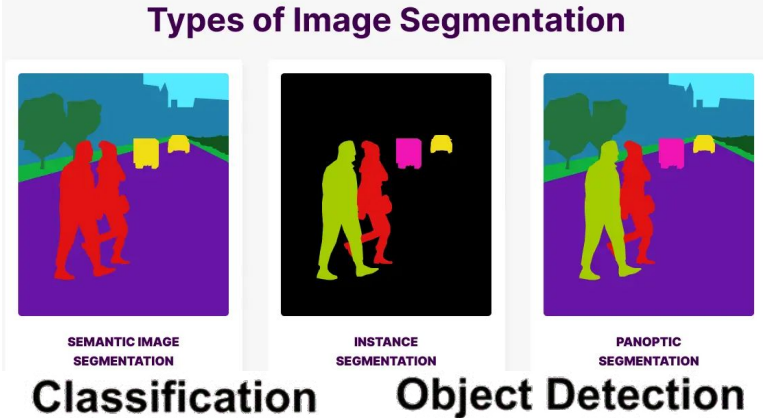


ViT vs CNN: Comparativa y beneficios de enfoques híbridos (ViT+CNN)

Característica	ViT	CNN (Convolutional Neural Networks)	Híbrido ViT+CNN
Estructura	Basado en attention . Sin operación de convolución.	Basado en convoluciones para la extracción de características.	Combinación de convolución y self-attention .
Procesamiento de datos	Procesa la imagen como una secuencia de parches.	Procesa la imagen usando filtros convolucionales.	Usa CNN para características locales y ViT para relaciones globales .
Ventajas	<ul style="list-style-type: none"> • Captura relaciones globales entre píxeles. • Escalabilidad con datos grandes. 	Excelente para capturar características locales .	Captura características locales y globales .
Desventajas	Necesita una alta cantidad más datos para entrenar.	Dificultad para capturar relaciones globales a gran escala.	Mayor complejidad y demanda computacional.
Necesidad de datos		Menor cantidad de datos en comparación con los ViT.	Depende del balance entre la parte CNN y la parte ViT.
Ejemplos	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.

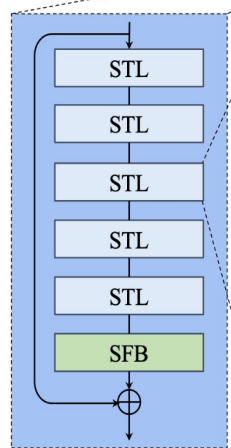
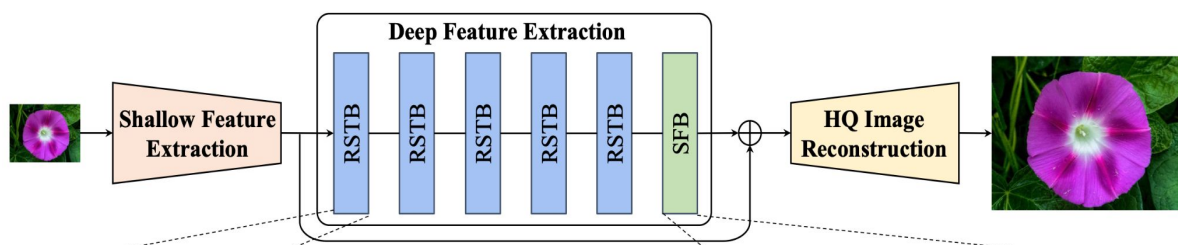
Aplicaciones de ViT: Casos de Uso

Aplicación	Ejemplos de Uso
Tareas de Reconocimiento	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• 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.

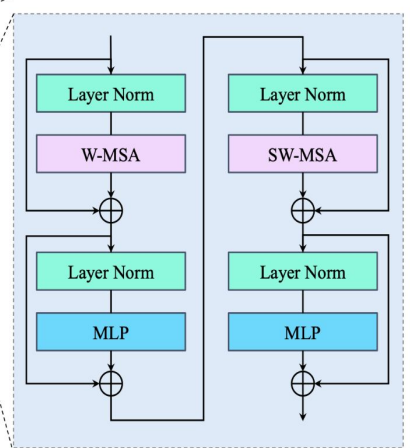


Aplicaciones de ViT: Casos de Uso

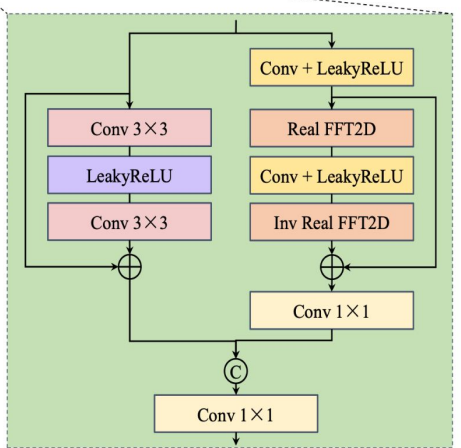
Aplicación	Ejemplos de Uso
Procesamiento de Video	<ul style="list-style-type: none">● Reconocimiento de actividades: Identificar acciones en secuencias de video.● Pronóstico de video: Predecir frames futuros en un video.
Visión de Bajo Nivel	<ul style="list-style-type: none">● Súper-resolución de imágenes: Mejorar la resolución de imágenes borrosas.● Mejora de imágenes: Aumentar la calidad visual.● Colorización: Asignar colores a imágenes en blanco y negro.
Análisis 3D	<ul style="list-style-type: none">● Clasificación y segmentación de nubes de puntos: Clasificación y segmentación de datos 3D.



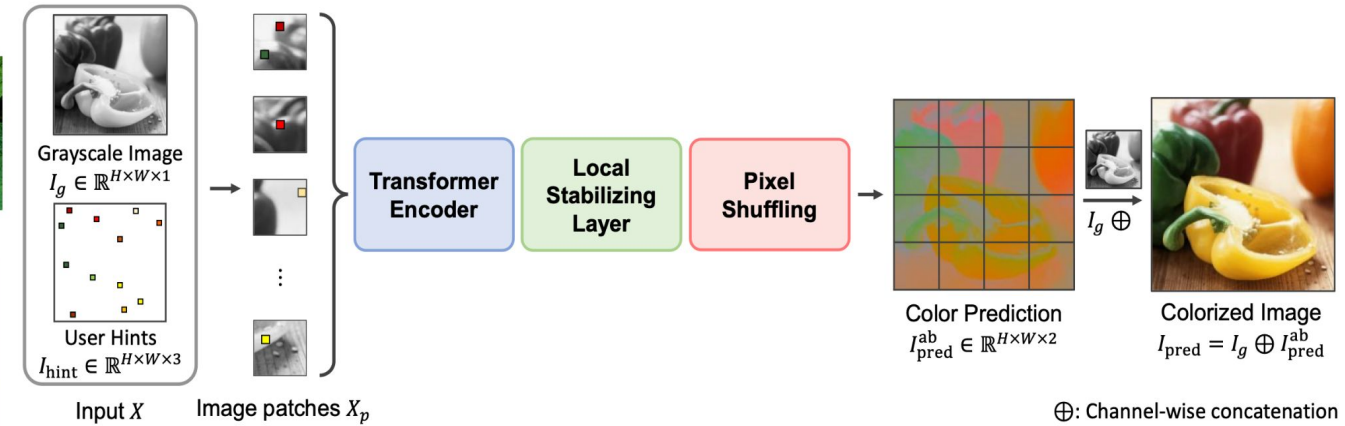
(a) RSTB



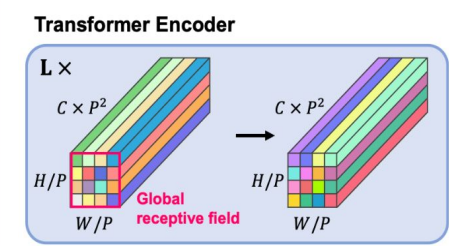
(b) Swin Transformer Layer (STL)



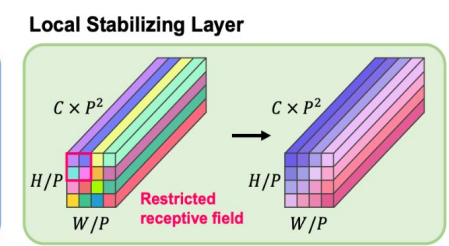
(c) Spatial Frequency Block (SFB)



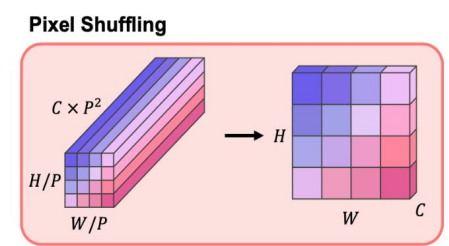
\oplus : Channel-wise concatenation



Transformer Encoder



Local Stabilizing Layer



Pixel Shuffling

Recuerden formar los grupos!

Bibliografía

Rothman, D. (2024) "Transformers for Natural Language Processing and Computer Vision: Explore Generative AI and Large Language Models with Hugging Face, ChatGPT, GPT-4V, and DALL-E." Packt Publishing; 3rd edition.

Dosovitskiy, A., et al. (2020) "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale." arXiv preprint arXiv:2010.11929. [Link](#)

Vaswani, A., et al. (2017) "Attention is All You Need." Advances in Neural Information Processing Systems (NeurIPS). [Link](#)

Haoran Z., et al. () "Understanding Why ViT Trains Badly on Small Datasets: An Intuitive Perspective"
[Link](#)

**Gracias por su atención y
dedicación.**

Recuerden que los grandes retos traen grandes aprendizajes.

¡Nos vemos en la próxima clase!