.UBAfiuba
FACULTAD DE INGENIERÍA

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.

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.

Evaluación Global=0.4*Prácticas + 0.6*Proyecto

Código

```
├── 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
```

[Cookiecutter Data Science]

[Folder Structure for Machine Learning Projects]

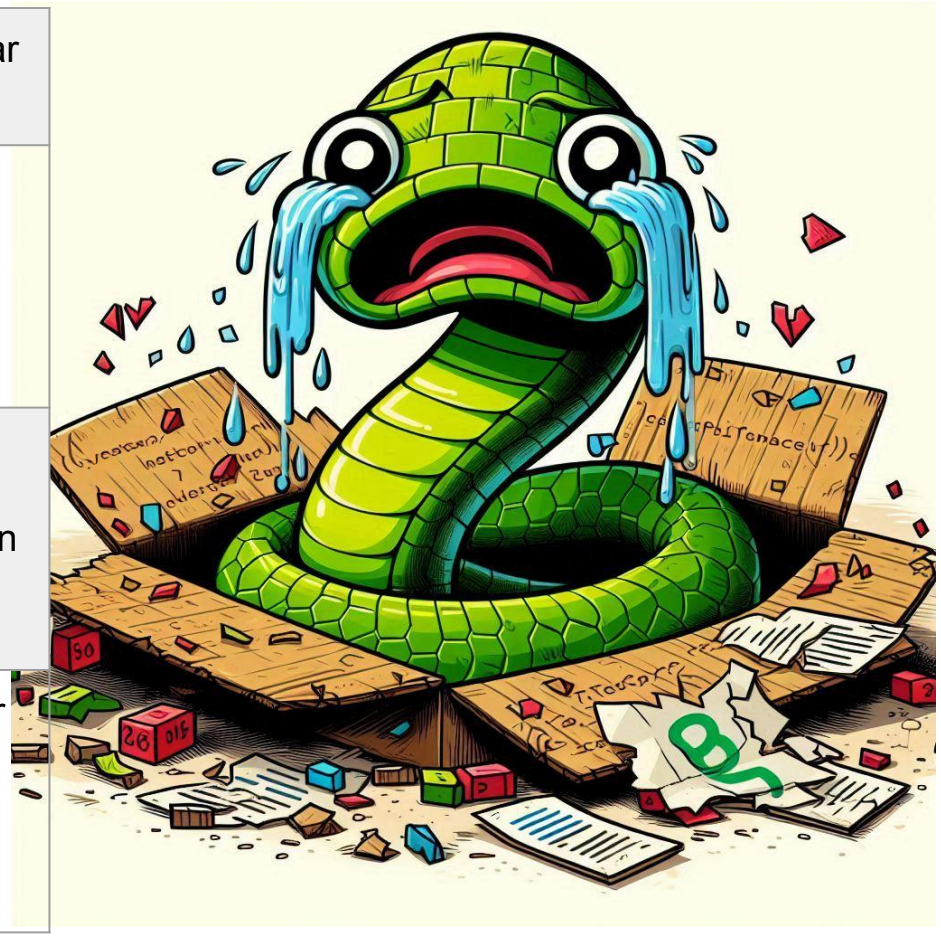
Código-Etapas

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:	• 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:	• Experimentación rápida. • Uso intensivo de visualizaciones. • Pruebas de hipótesis rápidas sin preocuparse por optimización o escalabilidad.
Problemas Comunes:	• 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.



Codigo-Prototipo

Objetivo:	Probar pipelines iniciales y validar resultados.
Estructura del Código:	• 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:	• Funciones básicas para limpieza y transformación de datos. • Uso de configuraciones fijas (hardcoded) para hiperparámetros y rutas de archivos.
Problemas Comunes:	• 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.

Codigo-Preproducción

Objetivo:	Convertir el prototipo en un flujo reproducible y parcialmente automatizado.
Estructura del Código:	• Código dividido en scripts o módulos (e.g., data_preprocessing.py, train_model.py). • Incorporación de configuraciones externas (config.yaml o .json). • Uso de herramientas de pruebas como pytest para validar partes críticas del flujo.
Enfoque:	• 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:	• Falta de pruebas exhaustivas: Limitada cobertura de pruebas unitarias. • Manejo limitado de datos: No considera grandes volúmenes o datos en tiempo real.

Codigo-Producción

Objetivo:	Crear un sistema robusto, reproducible y escalable, listo para implementarse en producción.
Estructura del Código:	• 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 o poetry.
Enfoque:	• 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:	• --- 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.
Importante	• 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	• No imprimir nada por pantalla. Todo con salida a un archivo • Manejo de MLflow o similar para realizar el seguimiento de métricas y modelos
Un plus absoluto	• Modularidad

Codigo-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 de 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

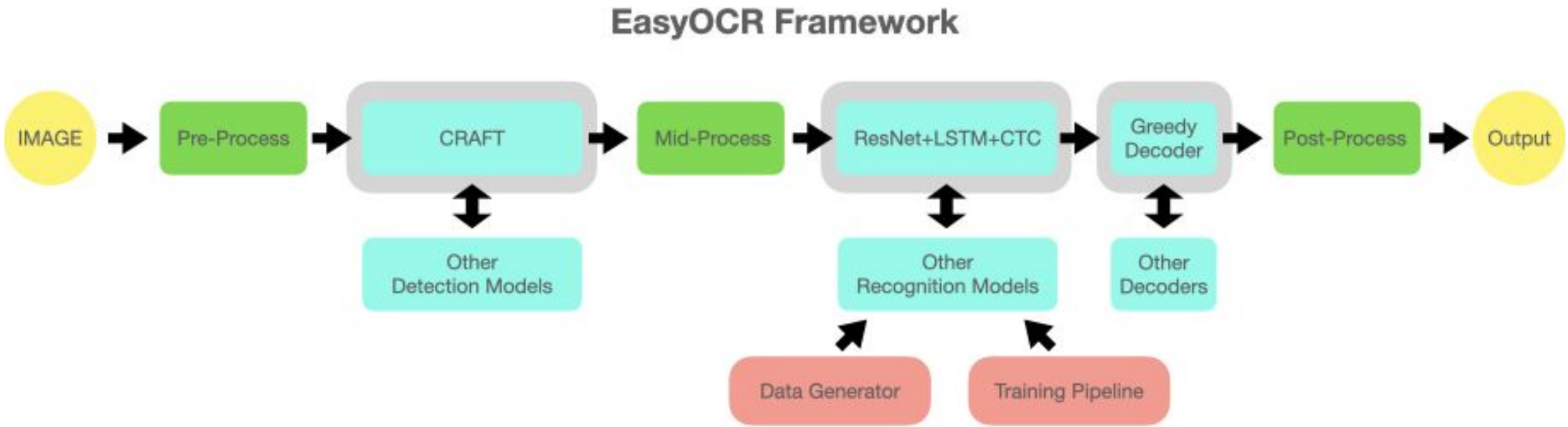
Optical Character Recognition

Optical Character Recognition (OCR)

El OCR, es un problema clásico (antes de 1980s), consiste en extraer texto de imágenes, documentos, etc. Hasta recientemente ha sido muy complejo y computacionalmente costoso. Previo a la IA, se utilizaban algoritmos complejos.

EasyOCR, es un framework que denota explícitamente el proceso tradicional de OCR.

Antes del Transformer, las CNN y LSTM servían como **backbone** para el OCR.

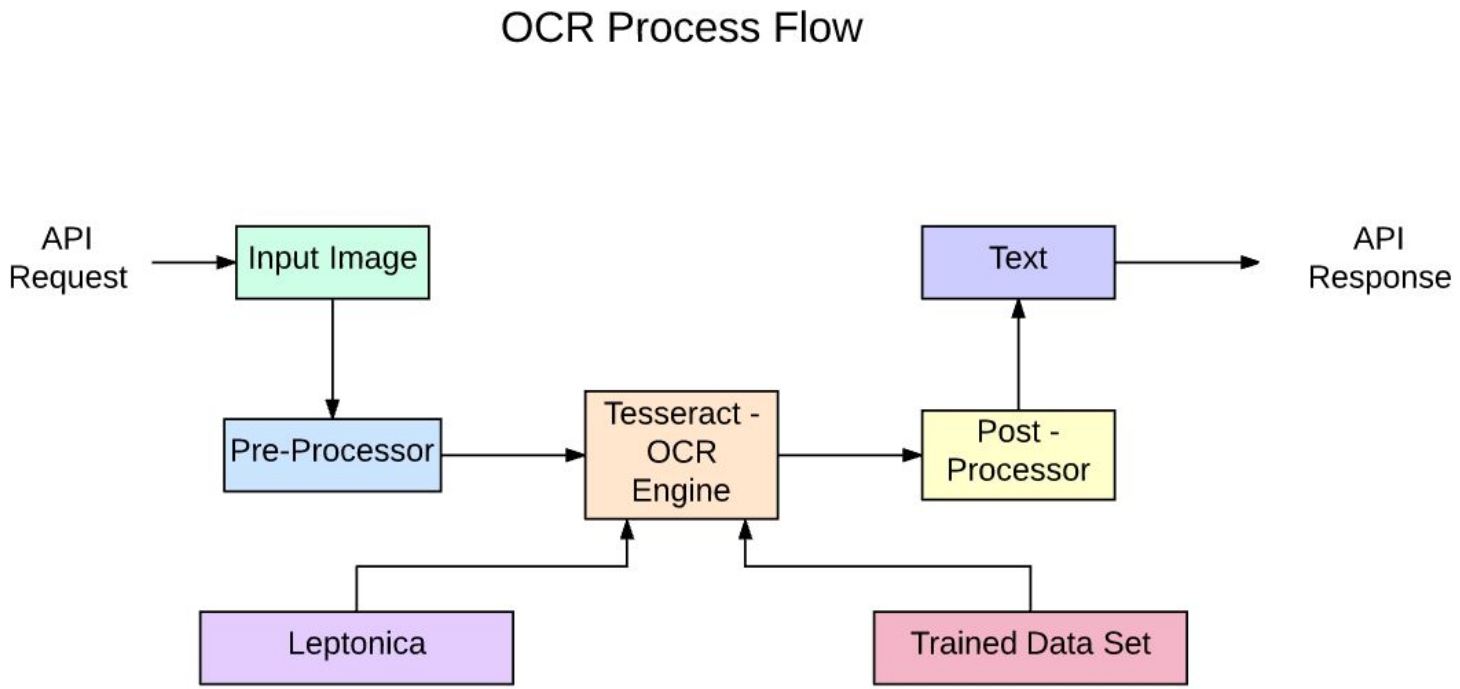


TesseractOCR

TesseractOCR es un proyecto inicialmente propietario de HP, desarrollado en los 1980s, pero finalmente fue open sourced en 2005 y patrocinado por Google en 2006.

En 2008, Tesseract ya utilizaba redes FeedForward, pero en 2018 se introdujo redes **LSTM como OCR engine.**

Cuenta con un wrapper de Python.

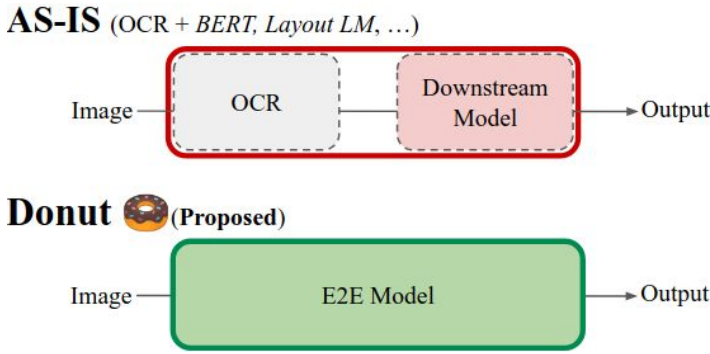


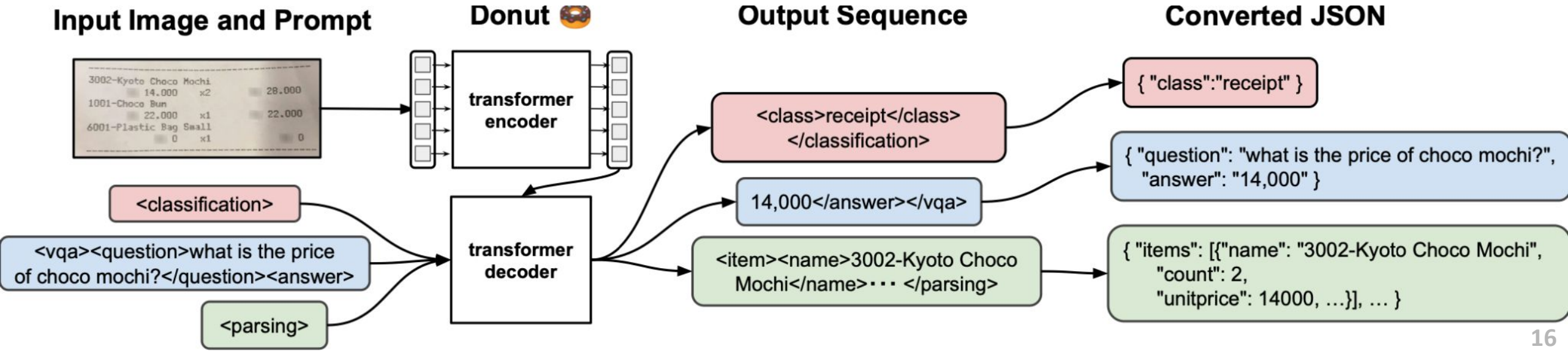
DONUT

Document Understanding Transformer (DONUT) fue presentado en 2022 en el paper "OCR-free Document Understanding Transformer", presenta una solución de OCR **end to end** mediante un ViT.

El proceso **convencional** de OCR, consiste en: detección de texto, reconocimiento de texto y parsing.

El proceso de DONUT no es nada más que introducir una imagen y un query para realizar la extracción.



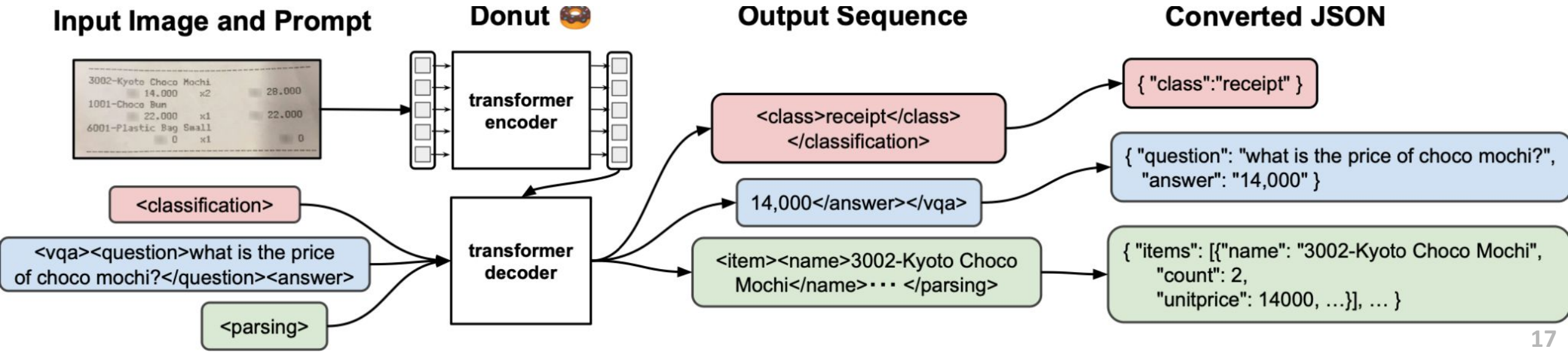


DONUT

El encoder (ViT) transforma una imagen de documento a **embeddings**, el decoder genera una secuencia de tokens que pueden ser transformados a un formato estructurado.

El encoder es un Swin-Transformer.

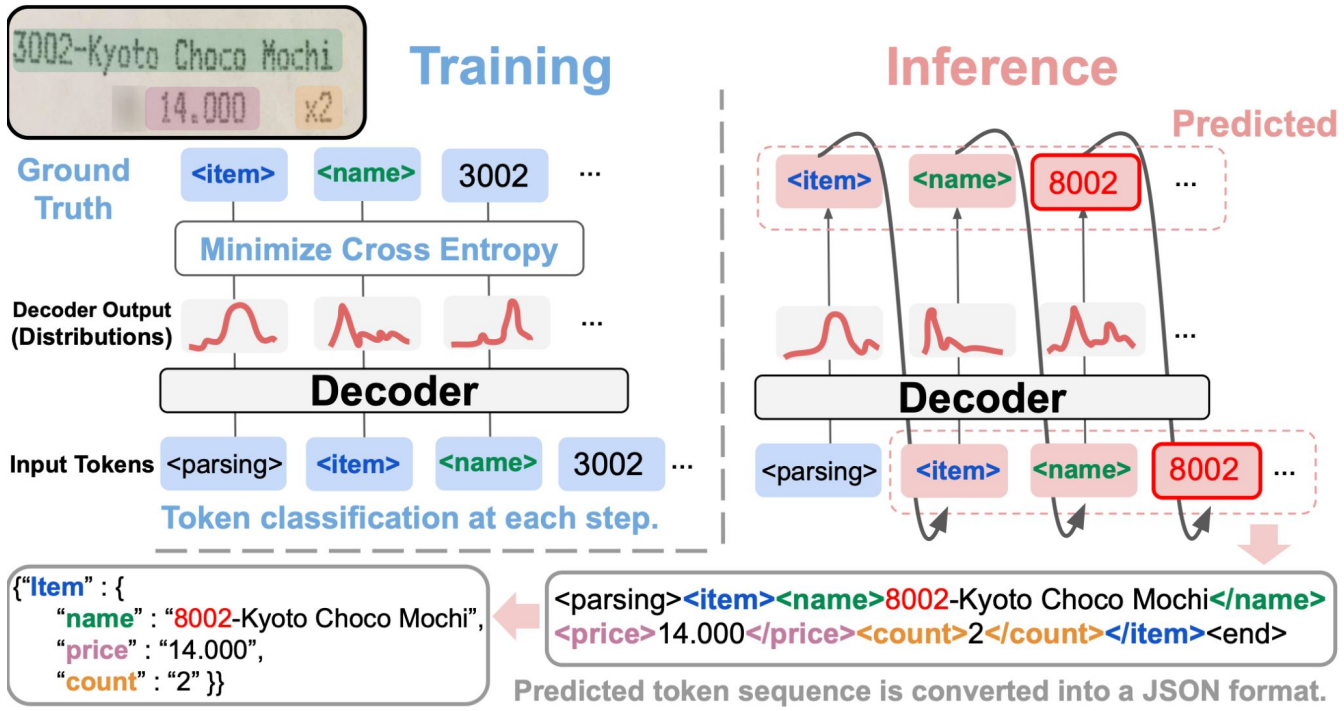
El decoder es BART preentrenado.



DONUT

Para DONUT, el encoder (ViT) transforma una imagen de documento a **embeddings**, el decoder genera una secuencia de tokens que pueden ser transformados a un formato estructurado.

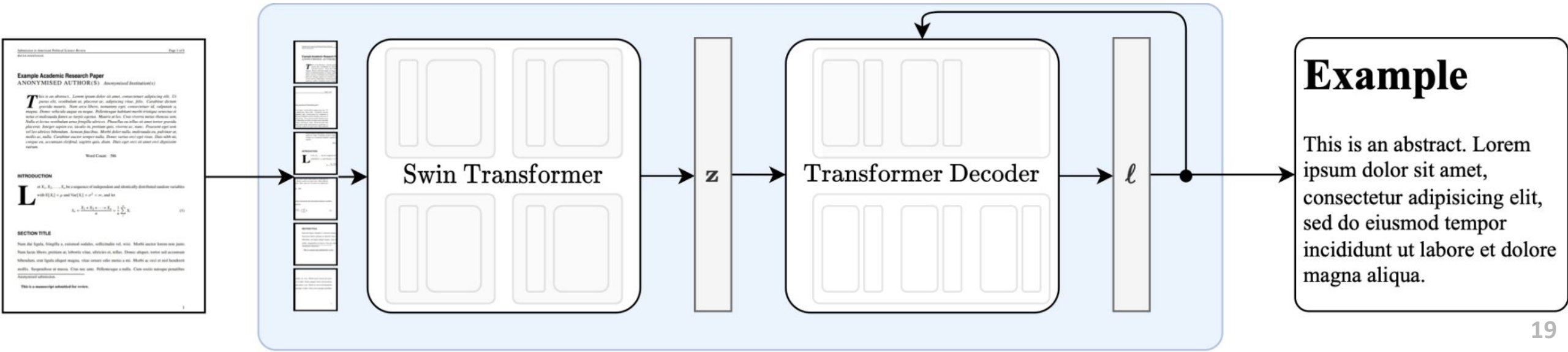
Github



NOUGAT

Presentado en el paper “Nougat: Neural Optical Understanding for Academic Documents”, consiste en el desarrollo de un modelo basado en la arquitectura de DONUT para documentos académicos. Menciona los problemas de OCR tradicional debido al nivel de granularidad linea por linea. NOUGAT realiza OCR en imágenes completas de documentos, capaz de convertir un PDF a un markup language.

Como image encoder utiliza Swin-T, como decoder utiliza mBART.



NOUGAT

Website
HuggingFace
Github

Name	Number of Pages
arXiv	7,511,745
PMC	536,319
IDL	446,777
Total	**8,204,754**

Table A.1: Dataset composition

Pixtral

Anteriormente vimos Pixtral, sin embargo puede utilizarse para la tarea de OCR, esto conlleva a grandes beneficios sobre NOUGAT y DONUT, el cual es integrar con información más allá del contexto de la imagen gracias a ser un vision language model.

Building OCR Systems Using Pixtral-12B

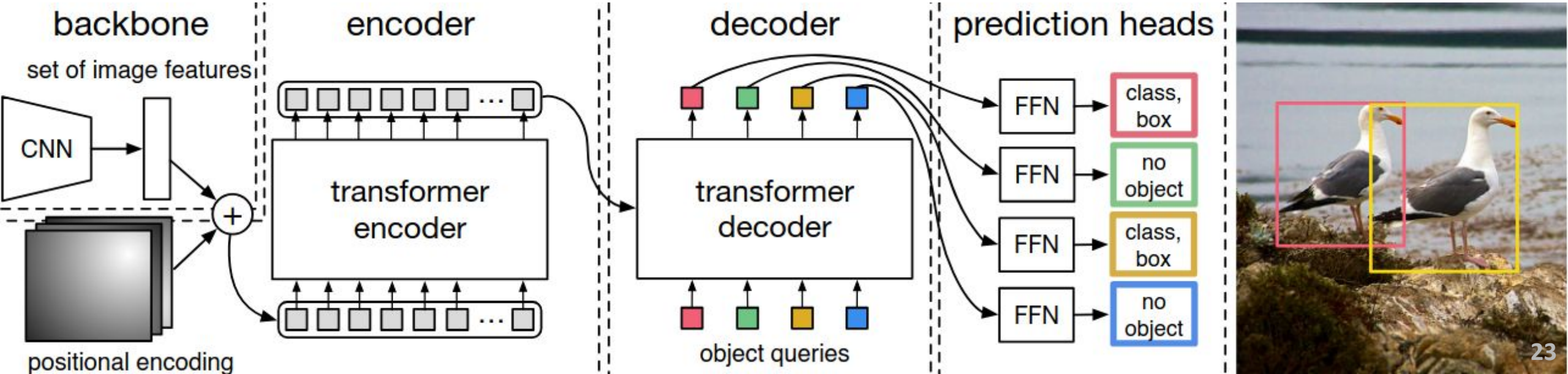
Pixtral Vision LLM

Detección

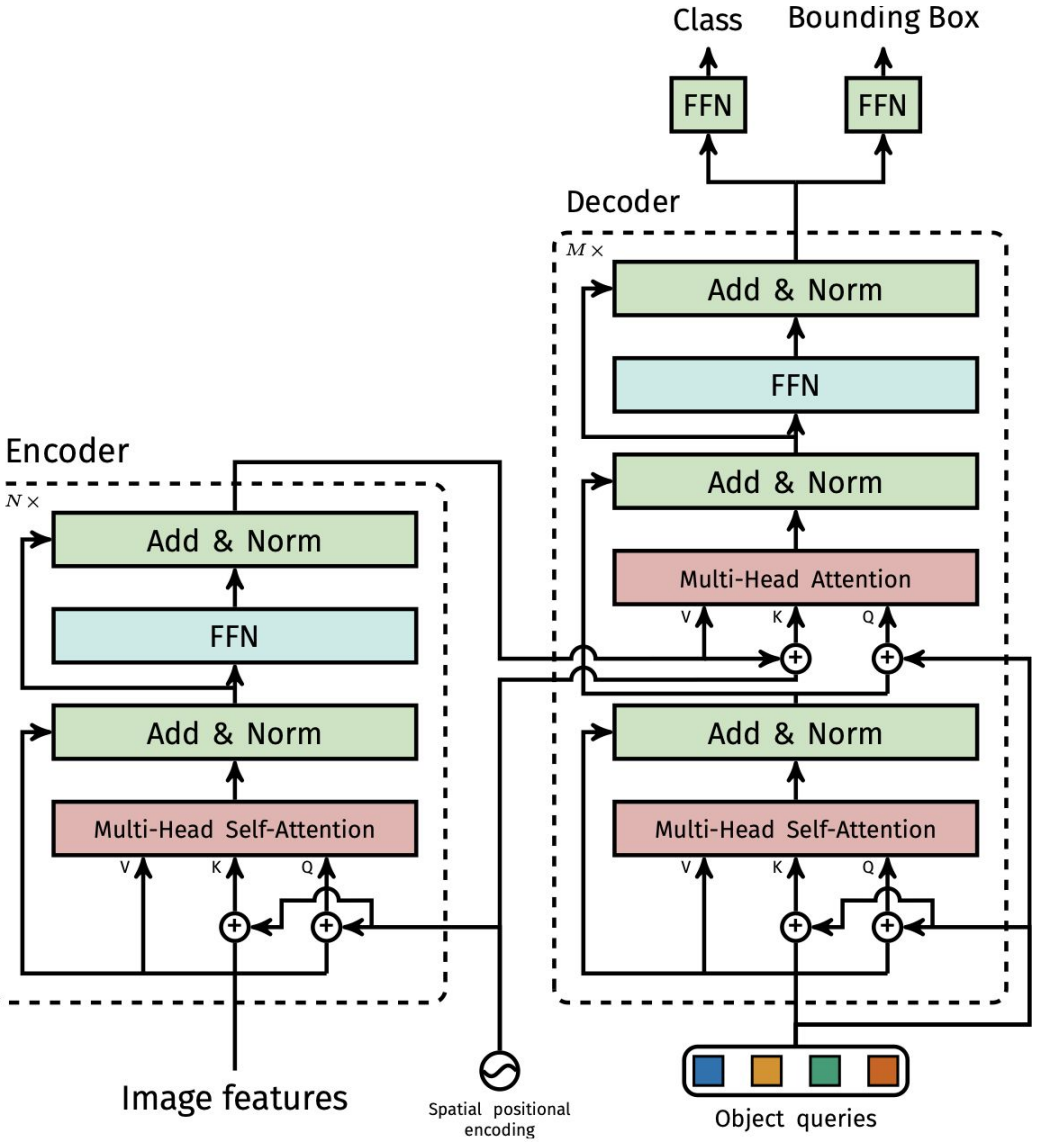
Detection Transformer (DETR)

Presentado en el paper "End-to-End Object Detection with Transformers", DETR consiste de un Transformer encoder y decoder en conjunto con una CNN para realizar predicción de bounding boxes.

Sea y el conjunto de ground truth e $\hat{y}=\{\hat{y}_i\}_{i=1}^{N}$ el conjunto de N predicciones. Asumiendo que N es mayor que la cantidad de objetos en la imagen, se considera y también como un conjunto de tamaño N, rellenado con $\varnothing$ (sin objeto).



Detection Transformer (DETR)



Para encontrar un bipartite match entre estos dos conjuntos, buscamos una permutación de N elementos $\sigma \in S_N$ con el costo más bajo:

$$\hat{\sigma} = \arg\min_{\sigma \in \mathfrak{S}_N} \sum_{i}^{N} \mathcal{L}_{\text{match}}(y_i, \hat{y}_{\sigma(i)})$$

Donde la función de pérdida es el matching cost entre ground truth y la predicción

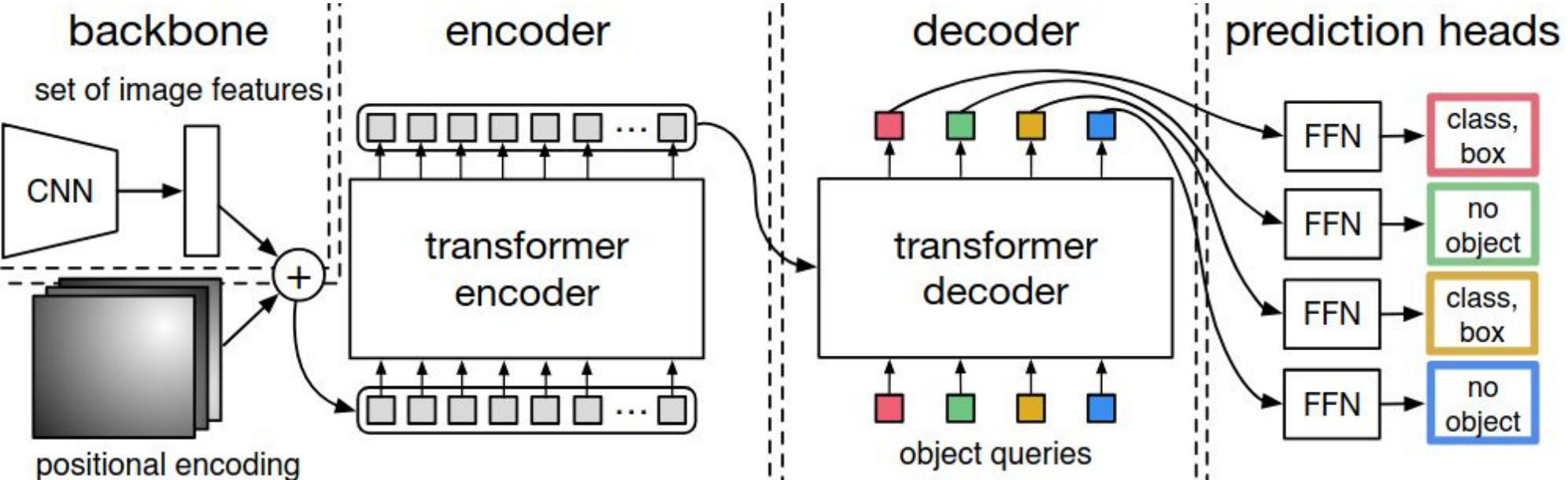
Detection Transformer (DETR)

El **backbone**, parte de una imagen inicial $x_{img} \in R^{3 \times H0 \times W0}$ con 3 canales de color, una CNN convencional genera un feature map de menor resolución $f \in R^{C \times H \times W}$. Los valores típicos que utiliza DETR son C=2048 y H,W= H0/32 ,W0/32. El backbone puede ser sustituido por ResNet u otros modelos convencionales.

El **encoder,** es un Transformer estándar que espera una secuencia como input, por lo tanto se transforman los features a d×HW

El **decoder**, es un Transformer estándar, la única diferencia es que realiza decodificación de N objetos en paralelo.

Los **prediction heads**, son FFNs de 3 capas, donde se da como output un set de N bounding boxes, se agrega la clase especial ∅ para representar que ningún objeto es detectado.



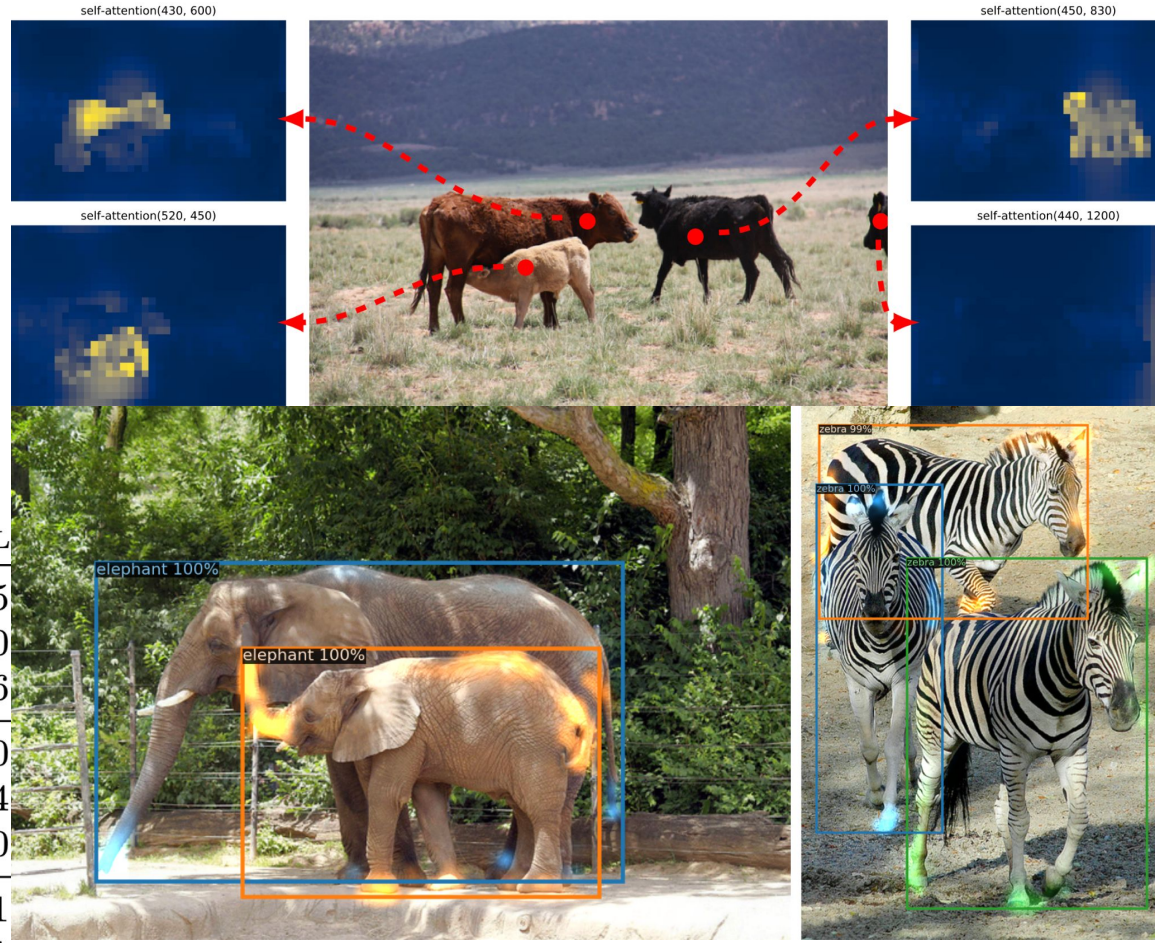
Detection Transformer (DETR)

[Huggingface]

[Github]

[Roboflow: What is DETR?]

[DETR vs YOLO for object detection]



Model	GFLOPS/FPS	#params	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Faster RCNN-DC5	320/16	166M	39.0	60.5	42.3	21.4	43.5	52.5
Faster RCNN-FPN	180/26	42M	40.2	61.0	43.8	24.2	43.5	52.0
Faster RCNN-R101-FPN	246/20	60M	42.0	62.5	45.9	25.2	45.6	54.6
Faster RCNN-DC5+	320/16	166M	41.1	61.4	44.3	22.9	45.9	55.0
Faster RCNN-FPN+	180/26	42M	42.0	62.1	45.5	26.6	45.4	53.4
Faster RCNN-R101-FPN+	246/20	60M	44.0	63.9	**47.8**	**27.2**	48.1	56.0
DETR	86/28	41M	42.0	62.4	44.2	20.5	45.8	61.1
DETR-DC5	187/12	41M	43.3	63.1	45.9	22.5	47.3	61.1
DETR-R101	152/20	60M	43.5	63.8	46.4	21.9	48.0	61.8
DETR-DC5-R101	253/10	60M	**44.9**	**64.7**	47.7	23.7	**49.5**	**62.3**

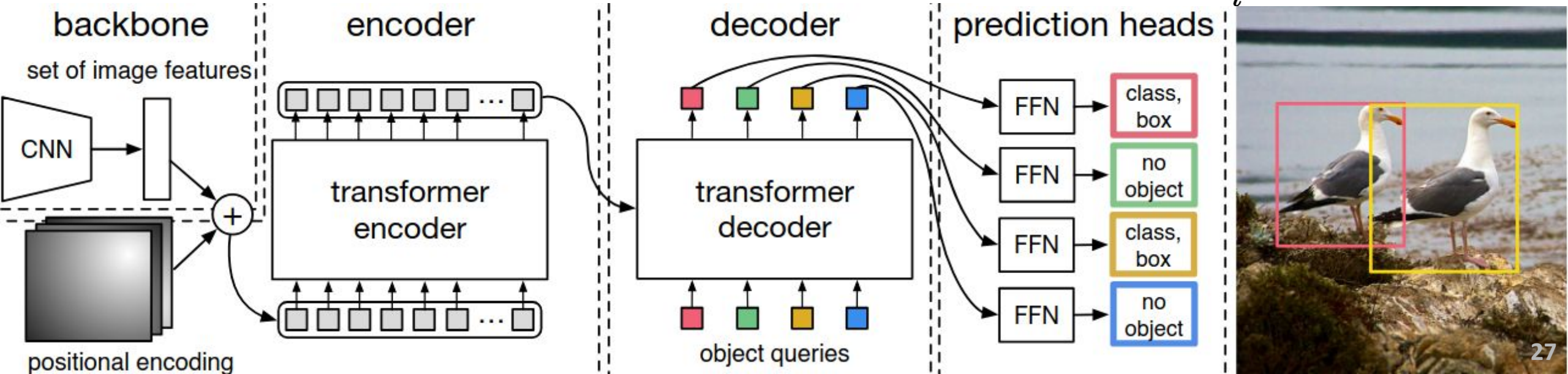
Detection Transformer (DETR)

https://medium.com/@faheemrustamy/detection-transformer-detr-vs-yolo-for-object-detection-baeb3c50bc3

https://keras.io/examples/vision/object_detection_using_vision_transformer/

Github

$$\hat{\sigma} = \arg\min_{\sigma \in \mathfrak{S}_N} \sum_{i}^{N} \mathcal{L}_{\text{match}}(y_i, \hat{y}_{\sigma(i)})$$



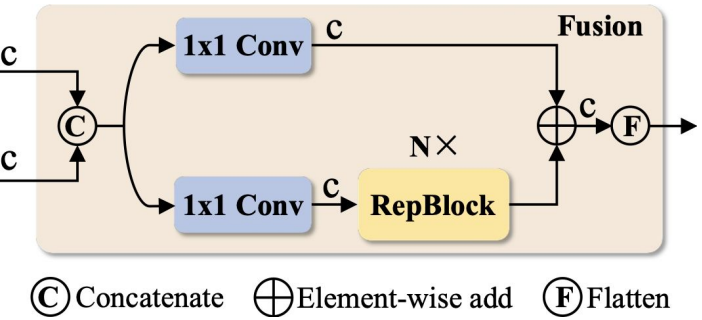
Real-Time DETR (RT-DETR)

Presentado en 2023 en el paper “DETRs Beat YOLOs on Real-time Object Detection”, mencionan los problemas que tienen YOLOs con Non-Maximum Suppresion (NMS), el cual se realiza en post-procesamiento y tiene un impacto negativo en inferencia.

Por otro lado DETR no utiliza NMS pero requiere recursos computacionales altos lo cual evita que sea utilizado en aplicaciones en tiempo real, el paper propone una modificacion de DETR para escenarios en tiempo real y superar a YOLO.

El paper menciona que el **cuello de botella se encuentra en el encoder**, para ello se introduce un encoder híbrido con la finalidad de aumentar el tiempo de procesamiento y mantener el accuracy de DETR.

Real-Time DETR (RT-DETR)

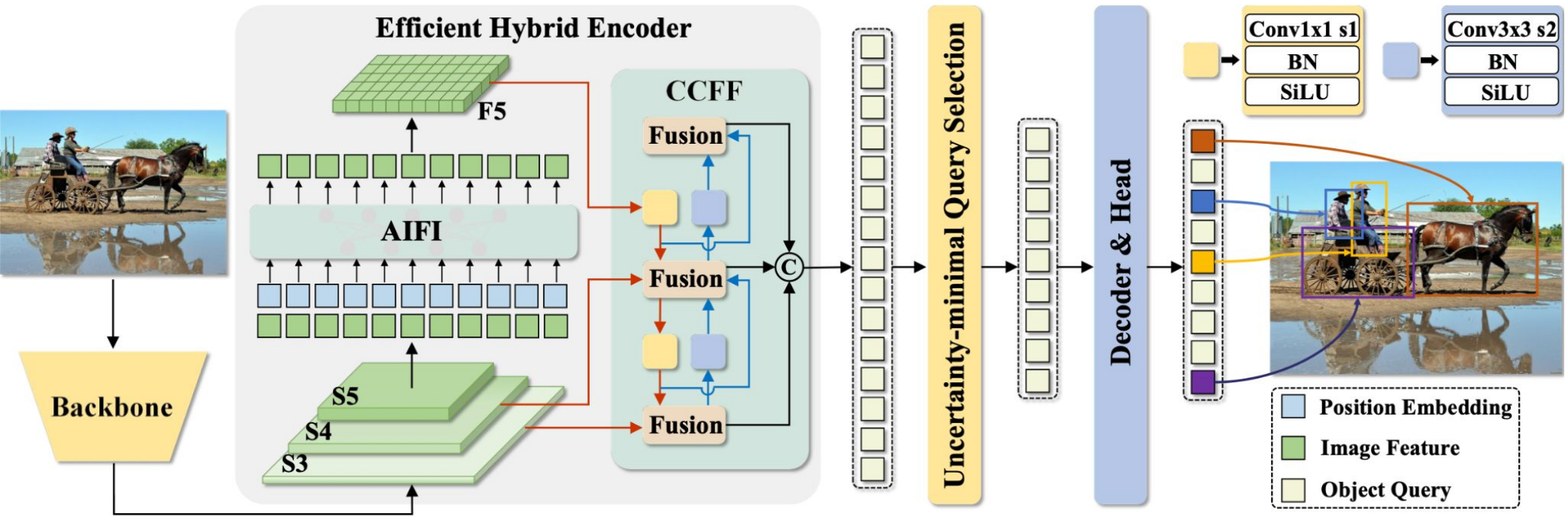


El encoder híbrido se divide en Attention-based Intra-scale Feature Interaction (AIFI) y CNN- based Cross-scale Feature Fusion (CCFF).

AIFI es un transformer al que se reduce el costo computacional al interactuar solamente con el último feature (S5) extraído del backbone, esto genera una reducción de latencia del 35%.

CCFF, fusiona features en un nuevo feature mediante conv.

$$\mathcal{Q} = \mathcal{K} = \mathcal{V} = \texttt{Flatten}(\mathcal{S}_5),$$
$$\mathcal{F}_5 = \texttt{Reshape}(\texttt{AIFI}(\mathcal{Q}, \mathcal{K}, \mathcal{V})),$$
$$\mathcal{O} = \texttt{CCFF}(\{\mathcal{S}_3, \mathcal{S}_4, \mathcal{F}_5\}),$$



Real-Time DETR (RT-DETR)

Model	Backbone	#Epochs	#Params (M)	GFLOPs	$FPS_{bs=1}$	AP^{val}	AP^{val}_{50}	AP^{val}_{75}	AP^{val}_{S}	AP^{val}_{M}	AP^{val}_{L}
Real-time Object Detectors											
YOLOv5-L [11]	-	300	46	109	54	49.0	67.3	-	-	-	-
YOLOv5-X [11]	-	300	86	205	43	50.7	68.9	-	-	-	-
PPYOLOE-L [40]	-	300	52	110	94	51.4	68.9	55.6	31.4	55.3	66.1
PPYOLOE-X [40]	-	300	98	206	60	52.3	69.9	56.5	33.3	56.3	66.4
YOLOv6-L [16]	-	300	59	150	99	52.8	70.3	57.7	34.4	58.1	70.1
YOLOv7-L [38]	-	300	36	104	55	51.2	69.7	55.5	35.2	55.9	66.7
YOLOv7-X [38]	-	300	71	189	45	52.9	71.1	57.4	36.9	57.7	68.6
YOLOv8-L [12]	-	-	43	165	71	52.9	69.8	57.5	35.3	58.3	69.8
YOLOv8-X [12]	-	-	68	257	50	53.9	71.0	58.7	35.7	59.3	70.7
End-to-end Object Detectors											
DETR-DC5 [4]	R50	500	41	187	-	43.3	63.1	45.9	22.5	47.3	61.1
DETR-DC5 [4]	R101	500	60	253	-	44.9	64.7	47.7	23.7	49.5	62.3
Anchor-DETR-DC5 [39]	R50	50	39	172	-	44.2	64.7	47.5	24.7	48.2	60.6
Anchor-DETR-DC5 [39]	R101	50	-	-	-	45.1	65.7	48.8	25.8	49.4	61.6
Conditional-DETR-DC5 [27]	R50	108	44	195	-	45.1	65.4	48.5	25.3	49.0	62.2
Conditional-DETR-DC5 [27]	R101	108	63	262	-	45.9	66.8	49.5	27.2	50.3	63.3
Efficient-DETR [42]	R50	36	35	210	-	45.1	63.1	49.1	28.3	48.4	59.0
Efficient-DETR [42]	R101	36	54	289	-	45.7	64.1	49.5	28.2	49.1	60.2
SMCA-DETR [9]	R50	108	40	152	-	45.6	65.5	49.1	25.9	49.3	62.6
SMCA-DETR [9]	R101	108	58	218	-	46.3	66.6	50.2	27.2	50.5	63.2
Deformable-DETR [45]	R50	50	40	173	-	46.2	65.2	50.0	28.8	49.2	61.7
DAB-Deformable-DETR [23]	R50	50	48	195	-	46.9	66.0	50.8	30.1	50.4	62.5
DAB-Deformable-DETR++ [23]	R50	50	47	-	-	48.7	67.2	53.0	31.4	51.6	63.9
DN-Deformable-DETR [17]	R50	50	48	195	-	48.6	67.4	52.7	31.0	52.0	63.7
DN-Deformable-DETR++ [17]	R50	50	47	-	-	49.5	67.6	53.8	31.3	52.6	65.4
DINO-Deformable-DETR [44]	R50	36	47	279	5	50.9	69.0	55.3	34.6	54.1	64.6
Real-time End-to-end Object Detector (ours)											
RT-DETR	R50	72	42	136	**108**	**53.1**	**71.3**	**57.7**	34.8	58.0	70.0
RT-DETR	R101	72	76	259	**74**	**54.3**	**72.7**	58.6	36.0	58.8	**72.1**

Github

Website

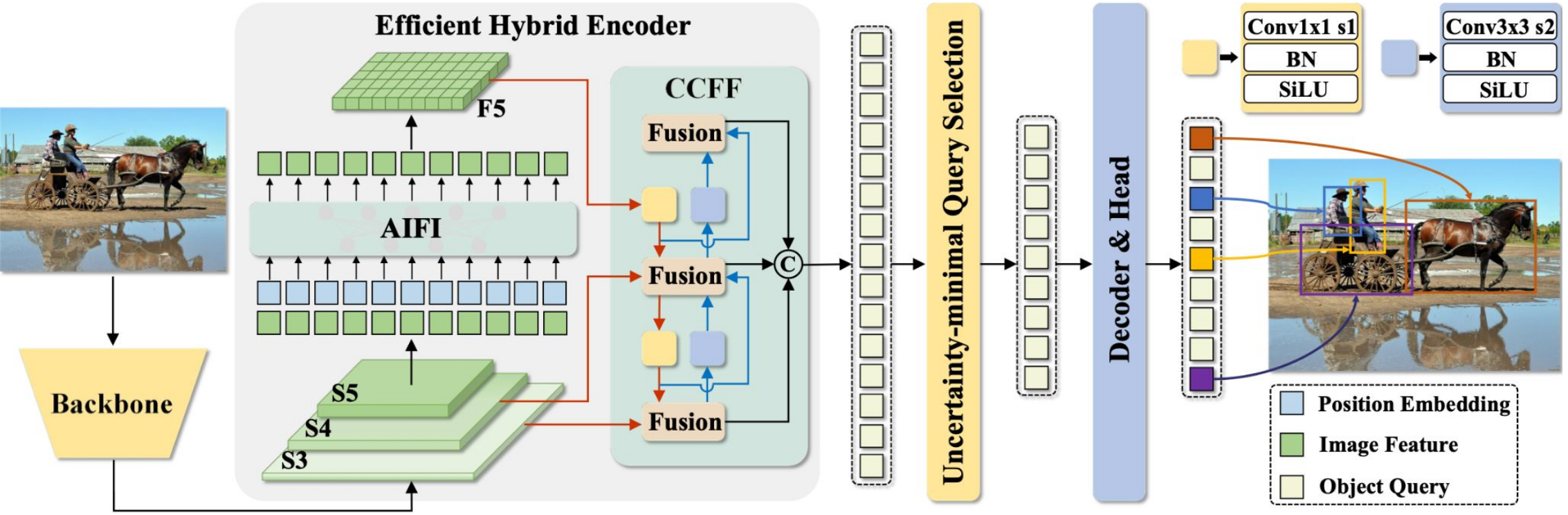
Huggingface

Ultralytics RT-DETR

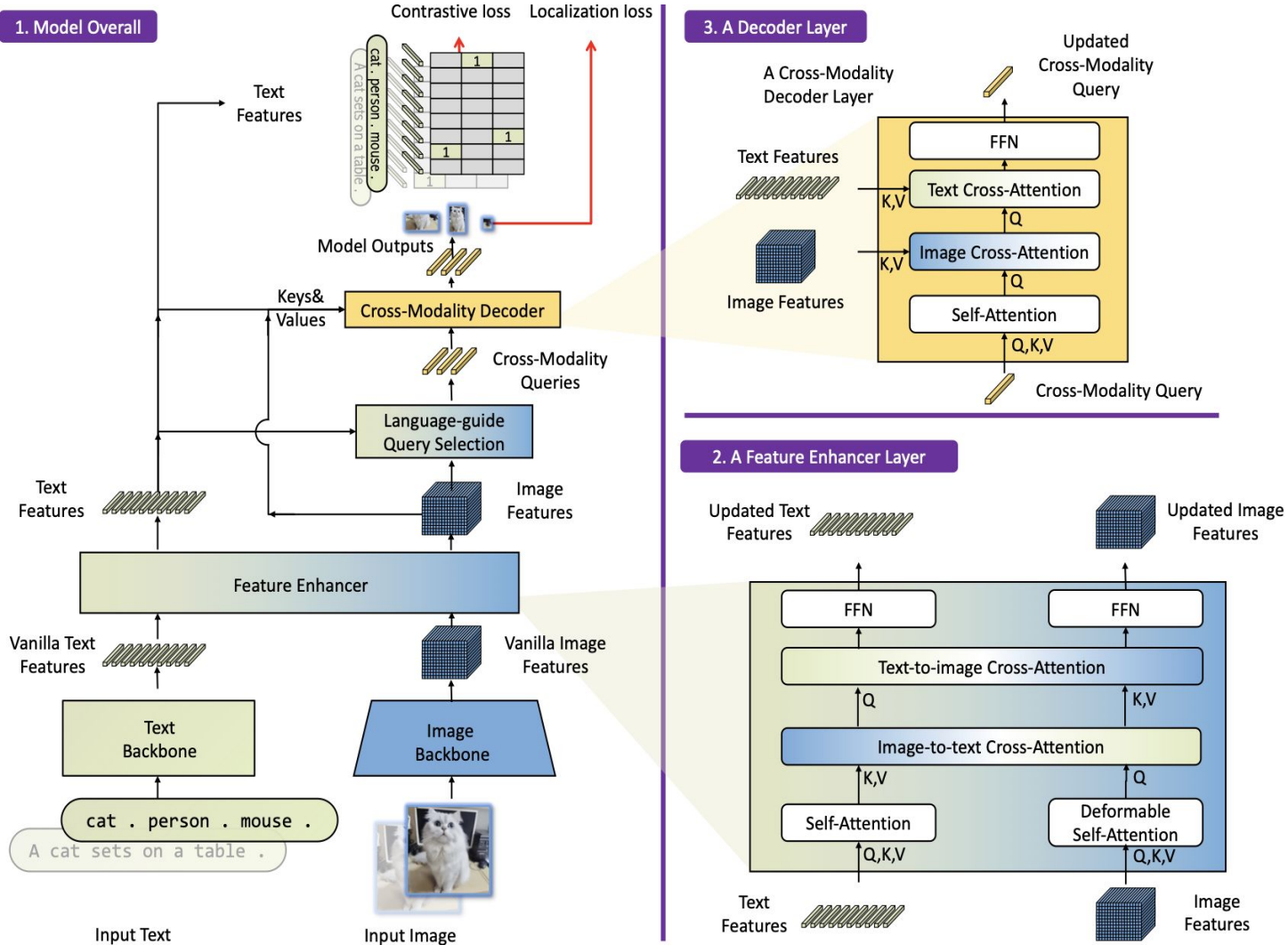
Actualmente se encuentra el paper corto de RT-DETRv2, el cual es idéntico al original, introduce mejoras de muestreo, data augmentation

Real-Time DETR (RT-DETR)

[Ultralytics RT-DETR]



Grounding DINO



Presentado en 2024 en el paper "Grounding DINO: Marrying DINO with Grounded Pre-Training for Open-Set Object Detection". Grounding DINO es basado en DINO (variante de DETR) el cual produce múltiples pares de cajas de objetos y frases nominales para un par dado de (Imagen, Texto).

Los image features son extraídos mediante un image backbone como Swin-T y los text features mediante BERT.

Grounding Dino realiza la tarea de detectar objetos en una imagen mediante un text input.

Grounding DINO

Model	Backbone	Pre-Training Data	Zero-Shot 2017val	Fine-Tuning 2017val/test-dev
Faster R-CNN	RN50-FPN	-	-	40.2 / -
Faster R-CNN	RN101-FPN	-	-	42.0 / -
DyHead-T [5]	Swin-T	-	-	49.7 / -
DyHead-L [5]	Swin-L	-	-	58.4 / 58.7
DyHead-L [5]	Swin-L	O365,ImageNet21K	-	60.3 / 60.6
SoftTeacher [50]	Swin-L	O365,SS-COCO	-	60.7 / 61.3
DINO(Swin-L) [57]	Swin-L	O365	-	62.5 / -
DyHead-T† [5]	Swin-T	O365	43.6	53.3 / -
GLIP-T (B) [25]	Swin-T	O365	44.9	53.8 / -
GLIP-T (C) [25]	Swin-T	O365,GoldG	46.7	55.1 / -
GLIP-L [25]	Swin-L	FourODs,GoldG,Cap24M	49.8	60.8 / 61.0
DINO(Swin-T)† [57]	Swin-T	O365	46.2	56.9 / -
Grounding DINO T (Ours)	Swin-T	O365	46.7	56.9 / -
Grounding DINO T (Ours)	Swin-T	O365,GoldG	48.1	57.1 / -
Grounding DINO T (Ours)	Swin-T	O365,GoldG,Cap4M	48.4	57.2 / -
Grounding DINO L (Ours)	Swin-L	O365,OI [19],GoldG	**52.5**	**62.6** / **62.7** (**63.0** / **63.0**)*
Grounding DINO L (Ours)	Swin-L	O365,OI,GoldG,Cap4M,COCO,RefC	**60.7**‡	**62.6** / -

[Roboflow: Grounding DINO]

[Notebook: Object Detection with Grounding DINO]

[Using Text Prompts for Image Annotation with Grounding DINO and Label Studio]

Preguntas?