

PROYECTO FINAL

FRUIT DETECTION WITH SWIN TRANSFORMER.

Autor: Jordan Isaac Pérez D.
Profesor: Javier Ruiz del Solar
Auxiliar: Patricio Loncomilla Z.
Ayudantes: Christian Díaz Guerra
Danilo Moreira
Jonas Peñailillo P.
Stefano Schiappacasse

Fecha de entrega: 28 de diciembre de 2023
Santiago, Chile

Índice de Contenidos

1. Introducción	1
2. Marco Teórico [1]	3
2.1. Arquitectura General	3
2.2. Autoatención basada en ventanas desplazadas	4
3. Desarrollo y resultados	6
3.1. Dataset MinneApple y preparación del conjunto	6
3.2. Modelo preentrenado Swin Transformer en MMDetection [3]	9
3.2.1. Implementación	9
3.2.2. Resultados	12
3.3. Fine Tuning y ajuste de hyperparámetros	14
3.3.1. Implementación	15
3.3.2. Resultados	19
3.3.3. Análisis de resultados fine tuning a distintos learning rates y Drop Out	25
3.3.4. Pruebas al conjunto de test	27
4. Conclusiones	29
5. Anexo	30
6. Referencias	47

Lista de Figuras

1. (a) La arquitectura de un Swin Transformer (Swin-T); (b) dos bloques Swin Transformer sucesivos. W-MSA y SW-MSA son módulos de autoatención de múltiples cabezales con configuraciones de ventanas regulares y desplazadas, respectivamente. [1]	3
2. Enfoque de ventana desplazada para calcular la autoatención en la arquitectura Swin Transformer. En la capa l (izquierda), se adopta un esquema de partición de ventanas normal y la autoatención se calcula dentro de cada ventana. En la siguiente capa l + 1 (derecha), la partición de la ventana se desplaza, lo que da como resultado nuevas ventanas. El cálculo de autoatención en las nuevas ventanas cruza los límites de las ventanas anteriores en la capa l, proporcionando conexiones entre ellas. [1]	5
3. Enfoque eficiente de cálculo por lotes para la autoatención en la partición de ventanas desplazadas. [1]	5
4. a) Detecciones test 1	12
5. b) Detecciones test 2	12
6. c) Detecciones test 3	13
7. d) Detecciones test 4	13
8. e) Detecciones test 5	14
5. loss validación lr = 0.00025	20

6.	accuracy validación lr = 0.00025	20
7.	loss validación lr = 0.000025	21
8.	accuracy validación lr = 0.000025	22
9.	loss validación lr = 0.0000025	23
10.	accuracy validación lr = 0.00000025	23
11.	loss validación lr = 0.000025	24
12.	accuracy validación lr = 0.000025	24
13.	test para modelo entrenado con Minneapple.	27
14.	test para modelo entrenado con Minneapple.	27
15.	test para modelo entrenado con Minneapple.	28
16.	test para modelo entrenado con Minneapple.	28
17.	test para modelo entrenado con Minneapple.	29
18.	test para modelo entrenado con Minneapple.	30

Listas de Tablas

1.	lr = 0.00025 y drop_out = 0.0	19
2.	lr=0.000025 y drop_out = 0.0	21
3.	lr = 0.0000025 y drop_out = 0.0	22
4.	lr = 0.00025 y drop_out = 0.25	24

Listas de Códigos

1.	Manipulación de json (1).	6
2.	Manipulación de json (2).	7
3.	Manipulación de json (3).	7
4.	Preparación del Conjunto (1).	7
5.	Preparación del Conjunto (2).	8
6.	Instalación de MMDetection	9
7.	Verificación de dependencias de MMDetection	9
8.	Descarga del modelo preentrenado	10
9.	Inicialización del modelo preentrenado SwinT	10
10.	Pruebas del modelo preentrenado (1)	11
11.	Pruebas del modelo preentrenado (2)	11
12.	Pruebas del modelo preentrenado (3)	11
13.	Conjunto de entrenamiento a directorio de mmdetection.	15
14.	Conjunto de entrenamiento a directorio de mmdetection.	16
15.	Inicialización de la configuración.	17
16.	Inicialización de la configuración.	17
17.	Configuración de entrenamiento (1).	17
18.	Configuración de entrenamiento (2).	18
19.	Configuración de entrenamiento (3).	18
20.	Configuración de entrenamiento (4).	18
21.	Configuración de entrenamiento (5).	18

22. Configuración de entrenamiento (6).	18
23. Configuración de entrenamiento (7).	19
24. Configuración de entrenamiento (8).	19
25. Configuración de entrenamiento (9).	19
26. Configuración de entrenamiento (10).	19
27. Código de python utilizado para generar los resultados.	30

1. Introducción

Este informe revisita e implementa un nuevo *Vision Transformer*, llamado *Swin Transformer*, que sirve como columna vertebral de propósito general para la visión por computadora, el cual explora los desafíos para adaptar Transformer del lenguaje a la visión, los cuales surgen de diferencias entre los dos dominios, como grandes variaciones en la escala de las entidades visuales y la alta resolución de los píxeles en las imágenes en comparación con las palabras en el texto, utilizando la jerarquía en su estructura cuya representación se calcula con ventanas desplazadas, aportando una mayor eficiencia al limitar el cálculo de autoatención a ventanas locales que no se superponen y al mismo tiempo permite la conexión entre ventanas. Esta arquitectura jerárquica, además, tiene la flexibilidad de modelar a varias escalas y tiene una complejidad computacional lineal con respecto al tamaño de la imagen. Estas cualidades de Swin Transformer lo hacen compatible con una amplia gama de tareas de visión, incluida la clasificación de imágenes y tareas de predicción densa, como la detección de objetos y segmentación semántica. Su rendimiento supera el estado del arte anterior por un amplio margen de +2,7 box AP y +2,6 AP de mask AP en COCO, y +3,2 mIoU en ADE20K, lo que demuestra el potencial de los modelos basados en Transformer como columna vertebral de visión. El diseño jerárquico y el enfoque de ventana desplazada también resultan beneficiosos para las arquitecturas totalmente MLP.

En este informe, se explorará la aplicación de un modelo preentrenado de OpenMMLab para la detección de manzanas en el dataset Minneappl, realizando fine-tuning específico para adaptarlo a las características del conjunto conjunto de datos. Este proceso permitirá aprovechar el conocimiento previo del modelo en tareas relacionadas y acelerar el entrenamiento para la detección de manzanas.

Además, se abordarán estrategias avanzadas para mejorar el rendimiento del modelo y pruebas con diferentes hiperparámetros, con el objetivo de perfeccionar la capacidad del modelo para identificar manzanas en diversas condiciones y tamaños, contribuyendo así a su robustez y generalización, según métricas de evaluación coherentes.

En este contexto, se comprenderá la configuración detallada del modelo, la manipulación de datos, y la optimización de hiperparámetros para lograr un rendimiento óptimo en la tarea de detección de manzanas. A lo largo del informe, se presentarán resultados, análisis y conclusiones derivadas de estas exploraciones, proporcionando una visión integral de la evolución y mejora del modelo en el contexto específico de la detección de manzanas con el dataset Minneapple.

El informe aborda aspectos clave del proceso de entrenamiento, desde la preparación y descripción del conjunto de datos hasta la elección de hiperparámetros críticos. Además, se analiza el rendimiento del modelo en términos de pérdida tanto en el conjunto de entrenamiento como en el de validación, otorgando una interpretación y análisis de los resultados obtenidos. Se discutirán, además, las posibles mejoras y los desafíos encontrados durante el proceso.

De esta forma, este informe detalla el proceso de diseño, entrenamiento y evaluación del modelo, destacando las decisiones clave tomadas durante cada fase y proporcionando un análisis exhaustivo de los resultados obtenidos. Con estos esfuerzos, se busca no solo construir un modelo preciso, sino también comprender las complejidades asociadas con la tarea de detección con swin transformer.

A continuación, se presentan las secciones que seguirán en este informe:

- Marco teórico: Se introducen los conocimientos teóricos necesarios para una correcta comprensión de las estructuras Swin Transformer-
- Desarrollo y resultados: En esta sección se tratarán la implementación y modificaciones de código necesarios para generar las detecciones y la aplicación de una configuración óptima para el proceso de fine tuning del modelo preentrenado, junto con sus métricas de evaluación y ejemplos de resultados en el conjunto de test.
- Conclusión: En esta sección se señalan cuáles fueron los aprendizajes obtenidos al realizar la tarea, las dificultades que se encontraron durante su desarrollo y si los resultados coinciden con lo que se espera teóricamente.
- Anexo: En esta última sección se adjunta el código utilizado para la generación de los resultados, la disposición de tiempos para la realización del proyecto mediante una carta Gantt y el link a github.

Mediante la combinación de estos conceptos y técnicas, el objetivo final de este informe es una comprensión más profunda de cada una de estas metodologías y su importancia en el procesamiento de imágenes.

2. Marco Teórico [1]

2.1. Arquitectura General

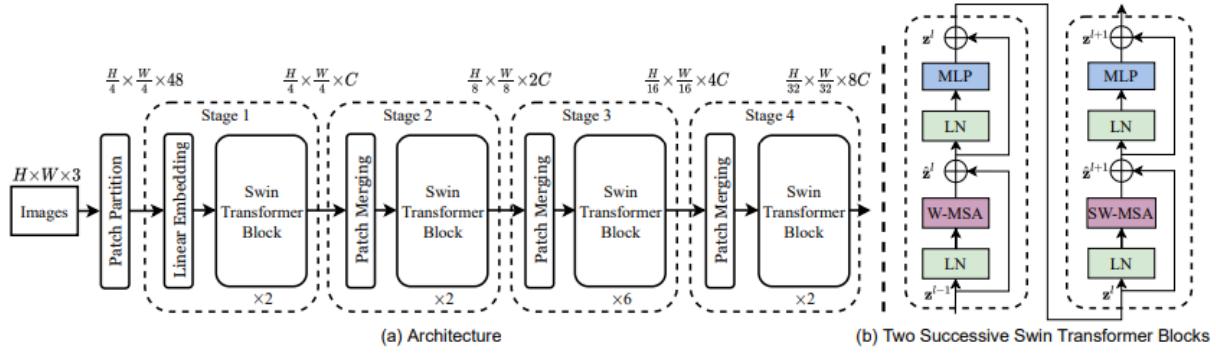


Figura 1: (a) La arquitectura de un Swin Transformer (Swin-T); (b) dos bloques Swin Transformer sucesivos. W-MSA y SW-MSA son módulos de autoatención de múltiples cabezales con configuraciones de ventanas regulares y desplazadas, respectivamente. [1]

En la Figura 1 se presenta una descripción general de la arquitectura Swin Transformer, que ilustra la versión pequeña *SwinT* y que es la utilizada en este trabajo de detección. Las etapas de la red se explican a continuación:

- **División en Patches (Patch Partition):** El primer paso consiste en dividir la imagen RGB de entrada en patches no superpuestos. Se utiliza un módulo de división de patches, como el utilizado en Vision Transformer. Cada uno de estos patches se considera y se procesa como un "token". En este caso, se utiliza un tamaño de patch de 4×4 , lo que significa que cada patch contiene información de píxeles de 4×4 regiones de la imagen. La característica de cada patch se representa como la concatenación de los valores RGB de los píxeles sin procesar dentro del patch, resultando en una dimensión de característica de $4 \times 4 \times 3 = 48$. Posteriormente, se aplica una capa de incrustación lineal a estas características para proyectarlas a una dimensión arbitraria, denotada como C .
- **Bloques Swin (Etapa 1):** Los tokens generados a partir de los patches se someten a varios bloques Transformer con cálculo de autoatención modificado, conocidos como bloques Swin Transformer. Estos bloques Transformer forman la “Etapa 1” del procesamiento. Es importante destacar que estos bloques Transformer mantienen el número de tokens, que es $\frac{H}{4} \times \frac{W}{4}$, donde H y W son las dimensiones de la imagen original. Además, junto con la incrustación lineal mencionada anteriormente, esta etapa constituye la primera capa de transformación y extracción de características en el modelo Swin Transformer.
- **Fusión de Patches y Transformación de Características (Etapa 2):** Para lograr una representación jerárquica, se reduce la cantidad de tokens fusionando capas de patches a

medida que la red se profundiza. En la primera capa de fusión de patches, se concatenan las características de cada grupo de patches vecinos de 2×2 y se aplica una capa lineal sobre las características concatenadas dimensionales 4C. Esto reduce la cantidad de tokens en un múltiplo de $2 \times 2 = 4$ (reducción de resolución $2 \times$), y la dimensión de salida se establece en 2C. Posteriormente, se aplican bloques Swin Transformer para la transformación, manteniendo la resolución en $\frac{H}{8} \times \frac{W}{8}$. Esta primera capa de fusión de parches y transformación de características se denomina “Etapa 2”.

- **Repetición del Proceso (Etapa 3 y 4):** Este procedimiento se repite dos veces más, generando así la “Etapa 3” y la “Etapa 4”. En cada etapa subsiguiente, la resolución de salida se reduce a $\frac{H}{16} \times \frac{W}{16}$ y $\frac{H}{32} \times \frac{W}{32}$, respectivamente.

Bloque Swin Transformer: El bloque Swin Transformer se construye reemplazando el módulo estándar de atención automática de cabezales múltiples (MSA) en un bloque Transformer por un módulo basado en ventanas desplazadas, descrito en la siguiente sección, manteniendo las demás capas igual. Como se ilustra en la Figura 3(b), un bloque Swin Transformer consta de un módulo MSA basado en ventana desplazada, seguido de un MLP de 2 capas con no linealidad GELU en el medio. Se aplica una capa LayerNorm (LN) antes de cada módulo MSA y cada MLP, y se aplica una conexión residual después de cada módulo.

2.2. Autoatención basada en ventanas desplazadas

- **Autoatención en ventanas no superpuestas:** Para un modelado eficiente, se calcula la autoatención dentro de ventanas locales. Las ventanas están dispuestas para dividir uniformemente la imagen sin superponerse.
- **Partición de ventanas desplazada en bloques sucesivos:** Para introducir conexiones entre ventanas manteniendo el cálculo eficiente de ventanas que no se superponen, se utiliza un enfoque de partición de ventana desplazada que alterna entre dos configuraciones de partición en bloques Swin Transformer consecutivos.

Como se ilustra en la Figura 2, el primer módulo utiliza una estrategia de partición de ventanas normal que comienza desde el píxel superior izquierdo, y el mapa de características de 8×8 se divide uniformemente en 2×2 ventanas de tamaño 4×4 ($M = 4$). Luego, el siguiente módulo adopta una configuración de ventanas que se desplaza respecto a la de la capa anterior, desplazando las ventanas en $(\lfloor \frac{M}{2} \rfloor, \lfloor \frac{M}{2} \rfloor)$ píxeles de las ventanas divididas regularmente.

- **Cálculo por lotes eficiente para la configuración desplazada:** Un problema con la partición de ventanas desplazadas es que dará como resultado más ventanas, desde $\frac{h}{M} \times \frac{w}{M}$ hasta $(\frac{h}{M} + 1) \times (\frac{w}{M} + 1)$ en la configuración desplazada, y algunas de las ventanas serán más pequeñas que $M \times M$. De esta forma, se utiliza un enfoque de cálculo por lotes eficiente mediante un desplazamiento cíclico hacia la dirección superior izquierda, como se ilustra en la 3. De esta forma, una ventana por lotes puede estar compuesta por varias subventanas que no son adyacentes en el mapa de características. Por lo tanto, se emplea un mecanismo de enmascaramiento para limitar el cálculo de la autoatención dentro de cada subventana.

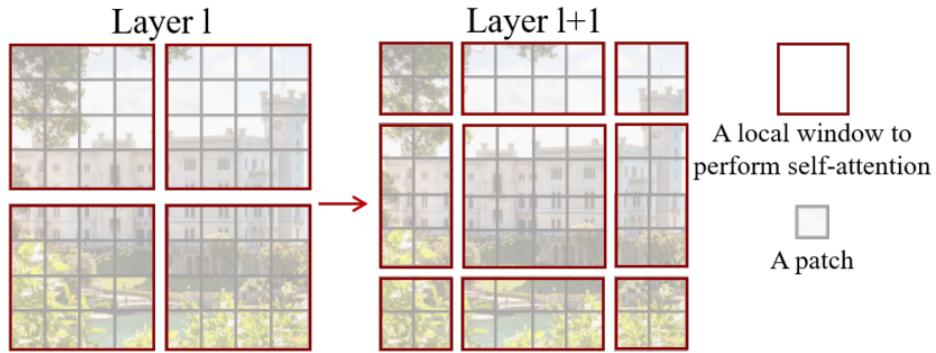


Figura 2: Enfoque de ventana desplazada para calcular la autoatención en la arquitectura Swin Transformer. En la capa 1 (izquierda), se adopta un esquema de partición de ventanas normal y la autoatención se calcula dentro de cada ventana. En la siguiente capa $l + 1$ (derecha), la partición de la ventana se desplaza, lo que da como resultado nuevas ventanas. El cálculo de autoatención en las nuevas ventanas cruza los límites de las ventanas anteriores en la capa 1, proporcionando conexiones entre ellas. [1]

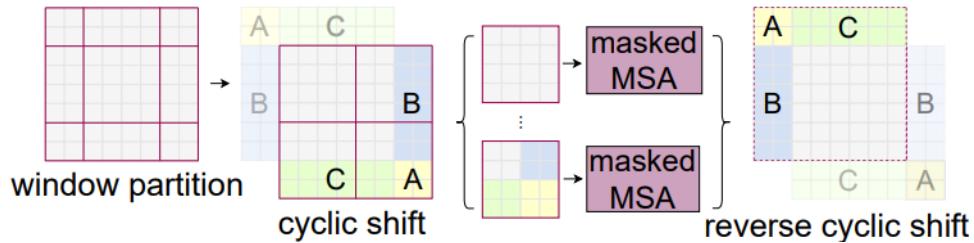


Figura 3: Enfoque eficiente de cálculo por lotes para la autoatención en la partición de ventanas desplazadas. [1]

- **Sesgo de posición relativa:** Al calcular la autoatención, se incluye un sesgo de posición relativa $B \in \mathbb{R}^{M^2 \times M^2}$ para cada cabeza al calcular la similitud:

$$\text{Attention}(Q, K, V) = \text{SoftMax}(QK^T / \sqrt{d} + B)V \quad (1)$$

donde $Q, K, V \in \mathbb{R}^{M^2 \times d}$ son las matrices de consulta, clave y valor; d es la dimensión de consulta/clave, y M^2 es el número de parches en una ventana. Dado que la posición relativa a lo largo de cada eje está en el rango $[-M + 1, M - 1]$, se parametriza una matriz de sesgo de tamaño más pequeño $\hat{B} \in \mathbb{R}^{(2M-1) \times (2M-1)}$, y los valores en B son tomados de \hat{B} .

3. Desarrollo y resultados

3.1. Dataset MinneApple y preparación del conjunto

MinneApple es un conjunto de datos que tiene como objetivo avanzar en la detección, segmentación y conteo de frutos en entornos de huertos a través de una gran variedad de imágenes de alta resolución adquiridas en huertos, junto con anotaciones humanas de los frutos de los árboles. Los objetos se etiquetan utilizando máscaras poligonales para cada instancia de objeto para ayudar en la detección, localización o segmentación precisa de objetos [2]. Para el propósito del proyecto, se utiliza el archivo de extensión tar.gz <https://conservancy.umn.edu/bitstream/handle/11299/206575/detection.tar.gz?sequence=2&isAllowed=y>, el cual tiene la siguiente estructura:

- train:
 - images: 670 imágenes de manzanas para entrenamiento.
 - masks: las máscaras de las 670 imágenes.
- test: 331 imágenes de manzanas para pruebas.

Además se cuenta con dos archivos Json: instances_train.json e instances_val.json, los cuales contienen las anotaciones asociadas las imágenes de *train* separados en entrenamiento y validación respectivamente, donde en primera instancia, se utiliza el siguiente código para subirlos al espacio de trabajo:

Código 1: Manipulación de json (1).

```
1 from google.colab import files
2 import json
3
4 # Función para cargar un archivo JSON
5 def cargar_json():
6     # Subir el archivo JSON
7     uploaded = files.upload()
8
9     # Obtener el nombre del archivo
10    file_name = list(uploaded.keys())[0]
11
12    # Cargar el contenido del archivo JSON
13    with open(file_name, 'r') as file:
14        data = json.load(file)
15
16    return data
17
18 # Función para mostrar información selectiva sobre el JSON
19 def mostrar_informacion_selectiva(data, keys_to_display=None, max_items=5):
20     print("Contenido del JSON (primeros {} elementos):".format(max_items))
21
```

```

22 # Mostrar solo algunas claves específicas o todas si no se especifica ninguna
23 keys_to_display = keys_to_display or list(data.keys())
24
25 for key in keys_to_display:
26     if key in data:
27         print(f"{key}: {data[key]}")
28
29 print("\nInformación sobre el JSON:")
30 print("Número total de elementos:", len(data))
31 print("Tipo de datos:", type(data))
32 print("Claves:", list(data.keys()))

```

De esta forma, los archivos json quedan guardados en variables definidas a conveniencia:

Código 2: Manipulación de json (2).

```
1 instance_train = cargar_json()
```

Código 3: Manipulación de json (3).

```
1 instance_val = cargar_json()
```

Por otro lado, para la utilización de este conjunto, primero se ha subido el set de datos a Google drive, para luego ser montado al espacio de trabajo de Google Colab.

De esta forma, la preparación del conjunto se lleva a cabo de la siguiente manera:

- Creación del directorio de validación: Primero se crea la carpeta val con las subcarpetas correspondientes según la estructura de la carpeta *train* preexistente en el archivo de datos descargado.

Código 4: Preparación del Conjunto (1).

```

1 import os
2
3 # Ruta del directorio principal
4 base_directory = '/content/drive/MyDrive/Procesamiento avanzado de imágenes/Proyecto
5           final/appledataset'
6
7 # Nombre de la carpeta que deseas crear
8 folder_name = 'val'
9
10 # Ruta completa de la carpeta a crear
11 folder_path = os.path.join(base_directory, folder_name)
12
13 # Verificar si la carpeta ya existe o no
14 if not os.path.exists(folder_path):
15     # Crear la carpeta si no existe
16     os.makedirs(folder_path)
17     print(f"Se ha creado la carpeta {folder_name} en {base_directory}")
18 else:
19     print(f"La carpeta {folder_name} ya existe en {base_directory}")

```

- Luego, se pasan las imágenes de validación según el archivo instances_val.json desde la carpeta de train al directorio recién creado.

Código 5: Preparación del Conjunto (2).

```

1 import os
2 import shutil
3
4 source_directory_images = '/content/drive/MyDrive/Procesamiento avanzado de imagenes/
   Proyecto final/appledataset/train/images'
5 source_directory_masks = '/content/drive/MyDrive/Procesamiento avanzado de imagenes/
   Proyecto final/appledataset/train/masks'
6
7 destination_directory_images = '/content/drive/MyDrive/Procesamiento avanzado de
   imagenes/Proyecto final/appledataset/val/images'
8 destination_directory_masks = '/content/drive/MyDrive/Procesamiento avanzado de imagenes
   /Proyecto final/appledataset/val/masks'
9
10 len_val = len(instance_val['images'])
11
12 for i in range(len_val):
13     val_image = instance_val['images'][i]['file_name']
14     source_path_images = os.path.join(source_directory_images, val_image)
15     source_path_masks = os.path.join(source_directory_masks, val_image)
16
17     destination_path_images = os.path.join(destination_directory_images, val_image)
18     destination_path_masks = os.path.join(destination_directory_masks, val_image)
19
20     # Verificar si el archivo ya existe en el directorio de destino
21     if not os.path.exists(destination_path_images) and not os.path.exists(
22         destination_path_masks):
23         # Mover el archivo solo si no existe en el directorio de destino
24         shutil.move(source_path_images, destination_path_images)
25         shutil.move(source_path_masks, destination_path_masks)
26
27         print(f'Se ha movido el archivo {val_image} de {source_directory_images} a {destination_directory_images}')
28         print(f'Se ha movido el archivo {val_image} de {source_directory_masks} a {destination_directory_masks}')
29     else:
30         print(f'El archivo {val_image} ya existe en {destination_directory_images} y {destination_directory_masks}. No se ha movido.')

```

De esta forma, se cuenta con 3 carpetas: *train*, *val* y *test* siempre disponibles en Google Drive para su utilización directa en el proyecto.

3.2. Modelo preentrenado Swin Transformer en MMDetection [3]

3.2.1. Implementación

OpenMMLab crea el sistema de algoritmos de visión por computadora de código abierto más influyente en la era del aprendizaje profundo. Sus objetivos son:

- Proporcionar bibliotecas de alta calidad para reducir las dificultades en la reimplementación de algoritmos.
- Crear cadenas de herramientas de implementación eficientes dirigidas a una variedad de backends y dispositivos
- Construir una base sólida para la investigación y el desarrollo de la visión por computadora
- Cerrar la brecha entre la investigación académica y las aplicaciones industriales con cadenas de herramientas completas

En particular, para este proyecto se utiliza la caja de herramientas de detección de objetos de código abierto basada en PyTorch llamado MMDetection cuyas características principales son:

- Diseño modular
- Soporte de múltiples tareas listo para usar
- Alta eficiencia

De esta forma, para utilizar un modelo preentrenado de MMDetection se hace lo siguiente:

- Se instalan las dependencias y ambiente en Google Colab.

Código 6: Instalación de MMDetection

```

1 !pip install -U openmim
2 !mim install mmengine
3 !mim install "mmcv>=2.0.0"
4
5 # Install mmdetection
6 !rm -rf mmdetection
7 !git clone https://github.com/open-mmlab/mmdetection.git
8 %cd mmdetection
9
10 !pip install -v -e .
11
12 !pwd

```

- Se verifican las versiones de algunas bibliotecas y su disponibilidad en el entorno.

Código 7: Verificación de dependencias de MMDetection

```

1 # Check Pytorch installation
2 import torch, torchvision
3 print("torch version:",torch.__version__, "cuda:",torch.cuda.is_available())

```

```

4
5 # Check MMDetection installation
6 import mmdet
7 print("mmdetection:",mmdet.__version__)
8
9 # Check mmcv installation
10 import mmcv
11 print("mmcv:",mmcv.__version__)
12
13 # Check mmengine installation
14 import mmengine
15 print("mmengine:",mmengine.__version__)

```

- Se utiliza el comando mim para descargar el modelo preentrenado mas simple de Swin Transformer (SwinT) de MMDetection y almacenarlo en el directorio específico checkpoints.

Código 8: Descarga del modelo preentrenado

```
1 !mim download mmdet --config mask-rcnn_swin-t-p4-w7_fpn_1x_coco --dest ./checkpoints
```

- Luego se inicializa un modelo de detección de objetos utilizando MMDetection con la arquitectura Swin Transformer preentrenado en COCO. Además se intenta inicializar el modelo en la GPU y, si hay un error, lo intenta en la CPU.

Código 9: Inicialización del modelo preentrenado SwinT

```

1 import mmcv
2 import mmengine
3 from mmdet.apis import init_detector, inference_detector
4 from mmdet.utils import register_all_modules
5 # Choose to use a config and initialize the detector
6 config_file = 'configs/swin/mask-rcnn_swin-t-p4-w7_fpn_1x_coco.py'
7 # Setup a checkpoint file to load
8 checkpoint_file = 'checkpoints/mask_rcnn_swin-t-p4-w7_fpn_1x_coco_20210902_120937-9
d6b7cfa.pth'
9
10 # register all modules in mmdet into the registries
11 register_all_modules()
12
13 # build the model from a config file and a checkpoint file
14 #model = init_detector(config_file, checkpoint_file, device='cuda:0') # or device='cpu'
15
16 try:
17     model = init_detector(config_file, checkpoint_file, device='cuda:0')
18     print("Modelo inicializado en GPU.")
19 except RuntimeError as e:
20     # Si hay un error, intenta inicializar el modelo en CPU
21     print(f"Error al inicializar en GPU: {e}")
22     print("Intentando inicializar en CPU.")
23     model = init_detector(config_file, checkpoint_file, device='cpu')
24     print("Modelo inicializado en CPU.")

```

De esta forma, se tiene un modelo preentrenado de Swin Transformer, que debería ser capaz de detectar algunas instancias de MinneApple, para esto, se hace una primera prueba de predicción con 5 imágenes del conjunto de test con el siguiente código:

Código 10: Pruebas del modelo preentrenado (1)

```
1 # Use the detector to do inference
2 image = mmcv.imread('/content/drive/MyDrive/Procesamiento avanzado de imagenes/Proyecto final
3                     /appledataset/test/images/dataset1_back_1.png',channel_order='rgb')
4 result = inference_detector(model, image)
5 print(result)
```

Código 11: Pruebas del modelo preentrenado (2)

```
1 # init visualizer(run the block only once in jupyter notebook)
2 visualizer = VISUALIZERS.build(model.cfg.visualizer)
3 # the dataset_meta is loaded from the checkpoint and
4 # then pass to the model in init_detector
5 visualizer.dataset_meta = model.dataset_meta
```

Código 12: Pruebas del modelo preentrenado (3)

```
1 # show the results
2 visualizer.add_datasample(
3     'result',
4     image,
5     data_sample=result,
6     draw_gt = False,
7     wait_time=0,
8     pred_score_thr=0.2
9 )
10 visualizer.show()
```

3.2.2. Resultados

Utilizando solo el modelo preentrenado directamente sobre 5 imágenes, donde se ha ajustado el umbral lo mas bajo posible dentro del rango tal que no se detecten otras instancias. Así, se obtienen las siguientes detecciones de la Figura 4:



a) Detecciones test 1



b) Detecciones test 2





e) Detecciones test 5

Figura 4: Detecciones imágenes de test modelo preentrenado

Dado que se ha utilizado un modelo preentrenado, en principio no se tiene una métrica que nos indique formalmente que tan bien el modelo hace las detecciones, pero gracias a las imágenes de prueba de la Figura 4, visualmente podemos concluir que el modelo a pesar de lograr algunas detecciones, la mayoría de las manzanas no logran detectarse, lo que no es viable para un modelo minimamente utilizable en la práctica. Para solucionar esto, es necesario emplear algunas técnicas avanzadas que permitan mejorar el rendimiento, y es en este punto donde el Proyecto muestra su utilidad real.

3.3. Fine Tuning y ajuste de hyperparámetros

El ajuste fino, conocido como "fine-tuning", es una estrategia crucial en el campo del aprendizaje profundo que permite adaptar modelos de inteligencia artificial preentrenados a tareas específicas. En lugar de entrenar un modelo desde cero, el fine tuning aprovecha los conocimientos aprendidos por un modelo en una tarea más general para mejorar su rendimiento en una tarea más específica o

relacionada. Este proceso implica tomar un modelo preentrenado, que ha aprendido patrones útiles en un conjunto de datos amplio, y ajustar sus pesos utilizando datos adicionales específicos para la tarea de interés.

En este caso, se ha utilizado el conjunto de datos MinneApple para ajustar los pesos del modelo preentrenado y obtener así una mejor detección de manzanas que las vistas hasta ahora.

3.3.1. Implementación

Pasos previos:

- Primeramente se crea la siguiente ruta en el directorio de mmdetection:
mmdetection/minneappledataset/minneapple
Además, en la carpeta *minneapple* se crean dos carpetas: train y val.
La creación de estas carpetas es análoga a lo mostrado anteriormente.
- Se pasan las imágenes (no las máscaras) desde las carpetas de train y val creadas en google colab al directorio de mmdetection.

Código 13: Conjunto de entrenamiento a directorio de mmdetection.

```

1 # Rutas de origen y destino
2 ruta_origen = '/content/drive/MyDrive/Procesamiento avanzado de imagenes/Proyecto final/
   appledataset/train/images'
3 ruta_destino = '/content/mmdetection/minneappledataset/minneapple/train'
4
5 # Verificar si la carpeta de destino existe
6 if not os.path.exists(ruta_destino):
7     os.makedirs(ruta_destino)
8
9 # Obtener la lista de archivos en la carpeta de origen
10 archivos_a_copiar = [archivo for archivo in os.listdir(ruta_origen) if os.path.isfile(os.path.join(
    ruta_origen, archivo))]
11
12 # Copiar cada archivo a la carpeta de destino si no existe en el destino
13 for archivo in archivos_a_copiar:
14     ruta_archivo_origen = os.path.join(ruta_origen, archivo)
15     ruta_archivo_destino = os.path.join(ruta_destino, archivo)
16
17     # Verificar si el archivo no existe en el destino antes de copiar
18     if not os.path.exists(ruta_archivo_destino):
19         shutil.copy(ruta_archivo_origen, ruta_archivo_destino)
20         print(f'Archivo copiado de "{ruta_origen}" a "{ruta_destino}": {archivo}')
21     else:
22         print(f'El archivo ya existe en "{ruta_destino}": {archivo}')
23
24 print('Proceso completado.')

```

- Además, se mueven los archivos json a su respectiva carpeta de train y val en el directorio de mmdetection. De esta forma, se tienen 536 imágenes de entrenamiento y 134 imágenes de validación.

- Como último paso antes de realizar el fine tuning, se verifica que los archivos json tengan el formato de COCO, formato necesario en mmdetection para realizar el entrenamiento de los modelos. Para esto se ha implementado el siguiente código:

Código 14: Conjunto de entrenamiento a directorio de mmdetection.

```

1 import json
2
3 def is_coco_format(json_data):
4     try:
5         # Verificar si el archivo tiene las claves y estructura típicas de COCO
6         required_keys = ['images', 'annotations', 'categories']
7         for key in required_keys:
8             if key not in json_data:
9                 return False
10
11         # Verificar si las claves 'id', 'image_id', 'category_id', 'bbox' están presentes en 'annotations'
12         annotation_keys = json_data['annotations'][0].keys()
13         annotation_required_keys = ['id', 'image_id', 'category_id', 'bbox']
14         for key in annotation_required_keys:
15             if key not in annotation_keys:
16                 return False
17
18         # Si todo está bien, devuelve True
19         return True
20
21     except Exception as e:
22         print(f"Error al verificar el formato COCO: {e}")
23         return False
24
25 # Ruta a tu archivo JSON
26 json_path = '/content/mmdetection/minneappledataset/minneapple/train/instances_train.json'
27
28 # Cargar el archivo JSON
29 with open(json_path, 'r') as f:
30     json_data = json.load(f)
31
32 # Verificar el formato COCO
33 if is_coco_format(json_data):
34     print("El archivo JSON está en formato COCO.")
35 else:
36     print("El archivo JSON no cumple con el formato COCO.")
37
38 # Ruta a tu archivo JSON
39 json_path = '/content/mmdetection/minneappledataset/minneapple/val/instances_val.json'
40
41 # Cargar el archivo JSON
42 with open(json_path, 'r') as f:
43     json_data = json.load(f)
44
45 # Verificar el formato COCO

```

```

46 if is_coco_format(json_data):
47     print("El archivo JSON está en formato COCO.")
48 else:
49     print("El archivo JSON no cumple con el formato COCO.")

```

el cual, define una función llamada `is_coco_format` que toma como entrada un diccionario JSON y verifica la presencia de claves esenciales ('`images`', '`annotations`', '`categories`') y la estructura típica de anotaciones COCO. Luego, verifica si las claves necesarias (`'id'`, `'image_id'`, `'category_id'`, `'bbox'`) están presentes en las anotaciones. Si todas estas verificaciones son exitosas, la función devuelve `True`, indicando que el formato es COCO.

Con estos pasos previos, el proceso de fine tuning se realiza de la siguiente forma:

- Se utiliza la biblioteca `mmengine` para cargar la configuración de un modelo de detección de objetos. La configuración se almacena en un archivo Python ubicado en la ruta `'./configs/swin/mask-rcnn_swin-t-p4-w7_fpn_1x_coco.py'`. La función `Config.fromfile()` de `mmengine` se encarga de cargar esta configuración desde el archivo y devolver un objeto de configuración (`cfg` en este caso).

Código 15: Inicialización de la configuración.

```

1 from mmengine import Config
2 cfg = Config.fromfile('./configs/swin/mask-rcnn_swin-t-p4-w7_fpn_1x_coco.py')

```

- Se explora el archivo de anotaciones para realizar correctamente la configuración necesaria:

Código 16: Inicialización de la configuración.

```

1 from pycocotools.coco import COCO
2
3 # Path to load the COCO annotation file
4 annotation_file = '/content/mmdetection/minneappledataset/minneapple/train/
5     instances_train.json'
6
7 # Initialise the COCO object
8 coco = COCO(annotation_file)
9
10 # Get all category tags and corresponding category IDs
11 categories = coco.loadCats(coco.getCatIds())
12 category_id_to_name = {cat['id']: cat['name'] for cat in categories}
13
14 # Print all category IDs and corresponding category names
15 for category_id, category_name in category_id_to_name.items():
16     print(f"Category ID: {category_id}, Category Name: {category_name}")

```

De esta forma, logramos conocer que tenemos una única categoría llamada *apple* de ID = 1.

- Se configura la semilla aleatoria para garantizar la reproducibilidad de los resultados del modelo.

Código 17: Configuración de entrenamiento (1).

```

1 from mmengine.runner import set_random_seed
2 set_random_seed(0, deterministic=False)

```

- Se definen las clases y colores específicos para el conjunto de datos, en este caso, solo la clase 'apple' y un color específico.

Código 18: Configuración de entrenamiento (2).

```

1 cfg.metainfo = {
2     'classes': ('apple', ),
3     'palette': [(220, 20, 60)]
4 }

```

- Se establecen las rutas y configuraciones específicas para el conjunto de datos de entrenamiento y validación.

Código 19: Configuración de entrenamiento (3).

```

1 cfg.data_root = './minneappledataset/minneapple'
2 cfg.train_dataloader.dataset.ann_file = 'train/instances_train.json'
3 cfg.train_dataloader.dataset.data_root = cfg.data_root
4 cfg.train_dataloader.dataset.data_prefix.img = 'train/'
5 cfg.train_dataloader.dataset.metainfo = cfg.metainfo

```

- Se configuran las rutas de anotaciones y evaluadores para la validación y prueba del modelo.

Código 20: Configuración de entrenamiento (4).

```

1 cfg.val_evaluator.ann_file = cfg.data_root + '/' + 'val/instances_val.json'
2 cfg.test_evaluator = cfg.val_evaluator

```

- Se establece el número de clases en las cabezas de caja y máscara del modelo a 1, ya que solo hay una clase ('apple').

Código 21: Configuración de entrenamiento (5).

```

1 cfg.model.roi_head.bbox_head.num_classes = 1
2 cfg.model.roi_head.mask_head.num_classes = 1

```

- Se especifica la ruta del modelo preentrenado y el directorio donde se guardarán los archivos y registros relacionados con el entrenamiento.

Código 22: Configuración de entrenamiento (6).

```

1 cfg.load_from = 'checkpoints/mask_rcnn_swin-t-p4-w7_fpn_1x_coco_20210902_120937-9
                 d6b7cfa.pth'
2 cfg.work_dir = './results'

```

- Se ajustan intervalos de evaluación, de guardado de puntos de control y parámetros de optimización.

Código 23: Configuración de entrenamiento (7).

```

1 cfg.train_cfg.val_interval = 3
2 cfg.default_hooks.checkpoint.interval = 3
3 cfg.optim_wrapper.optimizer.lr = 0.0002 / 8
4 cfg.model.backbone.drop_rate = 0.0
5 cfg.default_hooks.logger.interval = 10

```

- Se agrega el soporte para TensorBoard para realizar un seguimiento del proceso de entrenamiento.

Código 24: Configuración de entrenamiento (8).

```
1 cfg.visualizer.vis_backends.append({"type": "TensorboardVisBackend"})
```

- Se guarda la configuración actualizada en un archivo para su posterior referencia y uso en el entrenamiento del modelo.

Código 25: Configuración de entrenamiento (9).

```

1 config = './configs/swin/mask-rcnn_swin-t-p4-w7_fpn_1x_coco.py'
2 with open(config, 'w') as f:
3     f.write(cfg.pretty_text)

```

- Se procede a realizar el entrenamiento

Código 26: Configuración de entrenamiento (10).

```
1 !python tools/train.py {config}
```

3.3.2. Resultados

A continuación, se muestran las métricas proporcionadas por MMDetection a distintas combinaciones de inicialización de lr y Drop out en la configuración del modelo, teniendo en cuenta que dentro de las técnicas que implementa MMDetection, el lr es variable, se implementan ciertos tipos de dropout y otras técnicas avanzadas para mejorar el rendimiento y los resultados.

Tabla 1: lr = 0.00025 y drop_out = 0.0

Average Precision (AP)	[IoU = 0.5:0.95 area = all masDets = 100] = 0.415
	[IoU = 0.5 area = all masDets = 1000] = 0.827
	[IoU = 0.75 area = all masDets = 1000] = 0.371
	[IoU = 0.5:0.95 area = small masDets = 1000] = 0.400
	[IoU = 0.5:0.95 area = medium masDets = 1000] = 0.570
Average Recall (AR)	[IoU = 0.5:0.95 area = all masDets = 100] = 0.476
	[IoU = 0.5 area = all masDets = 300] = 0.476
	[IoU = 0.75 area = all masDets = 1000] = 0.476
	[IoU = 0.5:0.95 area = small masDets = 1000] = 0.456
	[IoU = 0.5:0.95 area = medium masDets = 1000] = 0.647

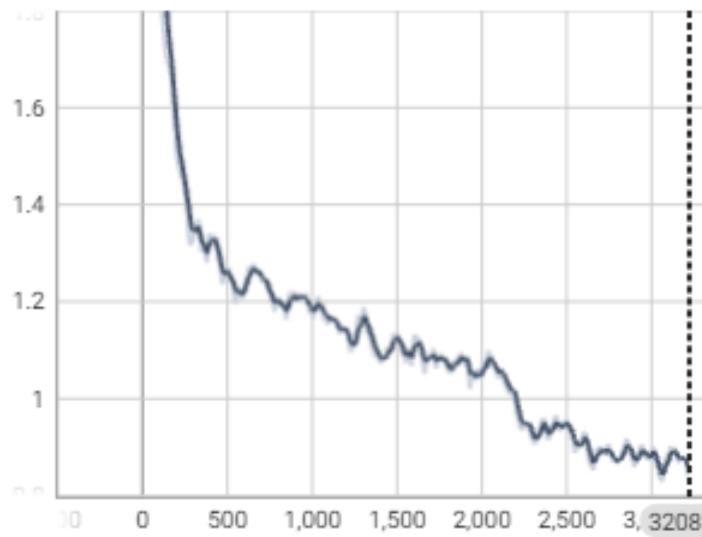


Figura 5: loss validación lr = 0.00025

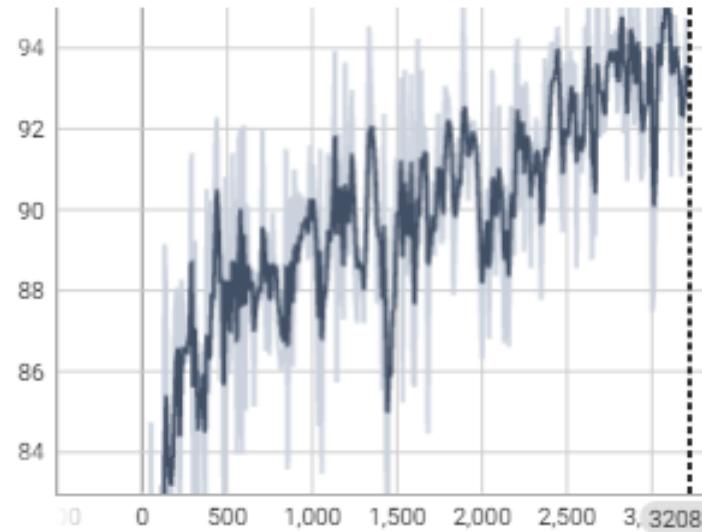


Figura 6: accuracy validación lr = 0.00025

Tabla 2: lr=0.000025 y drop_out = 0.0

Average Precision (AP)	[IoU = 0.5:0.95 area = all masDets = 100] = 0.405
	[IoU = 0.5 area = all masDets = 1000] = 0.818
	[IoU = 0.75 area = all masDets = 1000] = 0.347
	[IoU = 0.5:0.95 area = small masDets = 1000] = 0.388
	[IoU = 0.5:0.95 area = medium masDets = 1000] = 0.55
Average Recall (AR)	[IoU = 0.5:0.95 area = all masDets = 100] = 0.473
	[IoU = 0.5 area = all masDets = 300] = 0.473
	[IoU = 0.75 area = all masDets = 1000] = 0.454
	[IoU = 0.5:0.95 area = small masDets = 1000] = 0.641
	[IoU = 0.5:0.95 area = medium masDets = 1000] = 0.0

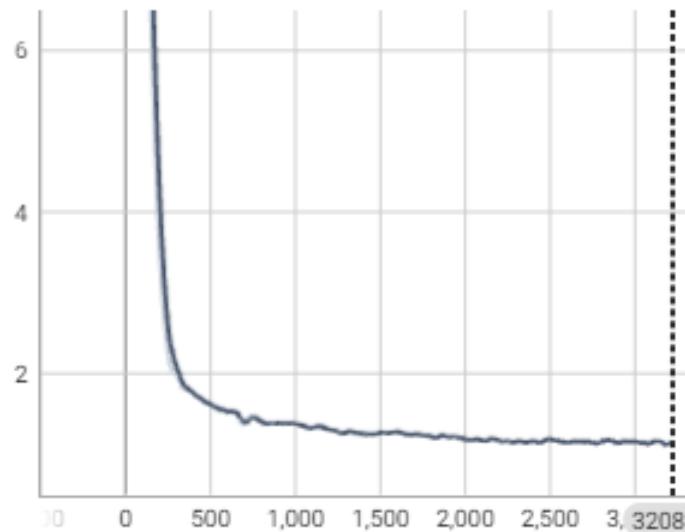


Figura 7: loss validación lr = 0.000025

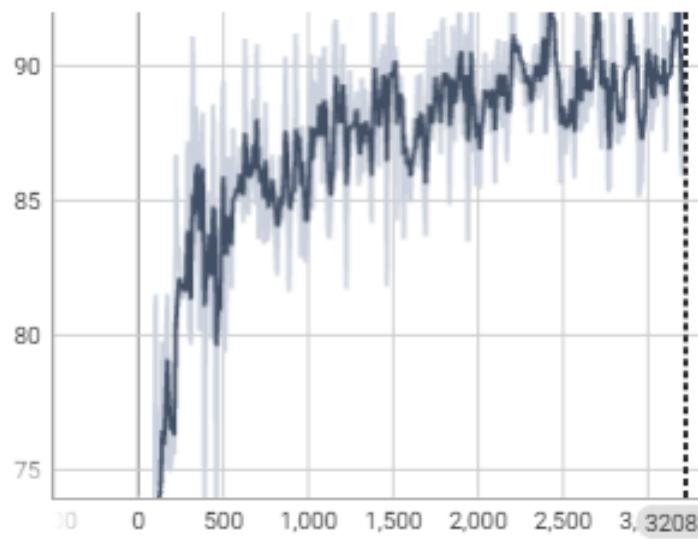


Figura 8: accuracy validación lr = 0.000025

Tabla 3: lr = 0.0000025 y drop_out = 0.0

Average Precision (AP)	[IoU = 0.5:0.95 area = all maxDets = 100] = 0.349
	[IoU = 0.5 area = all maxDets = 1000] = 0.753
	[IoU = 0.75 area = all maxDets = 1000] = 0.271
	[IoU = 0.5:0.95 area = small maxDets = 1000] = 0.325
	[IoU = 0.5:0.95 area = medium maxDets = 1000] = 0.531
Average Recall (AR)	[IoU = 0.5:0.95 area = all maxDets = 100] = 0.417
	[IoU = 0.5 area = all maxDets = 300] = 0.417
	[IoU = 0.75 area = all maxDets = 1000] = 0.417
	[IoU = 0.5:0.95 area = small maxDets = 1000] = 0.397
	[IoU = 0.5:0.95 area = medium maxDets = 1000] = 0.583

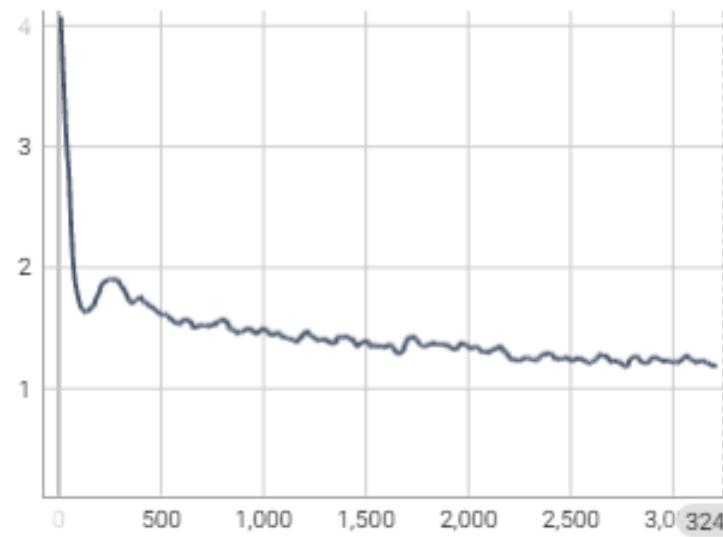


Figura 9: loss validación lr = 0.0000025

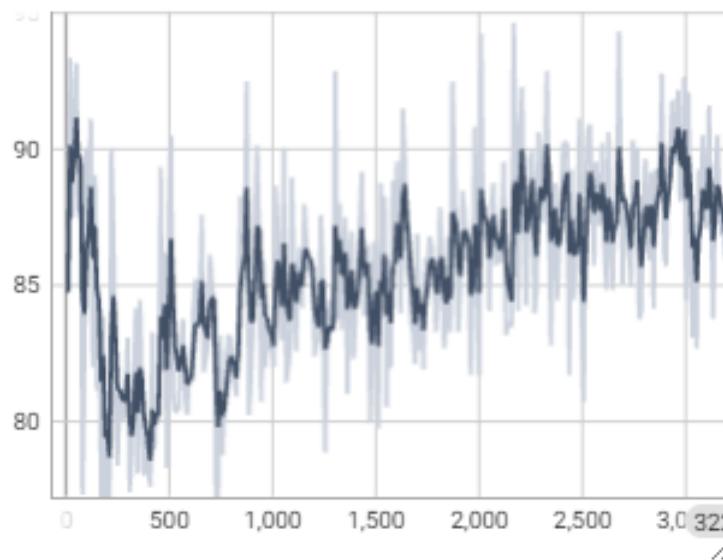


Figura 10: accuracy validación lr = 0.0000025

Tabla 4: lr = 0.00025 y drop_out = 0.25

Average Precision (AP)	[IoU = 0.5:0.95 area = all maxDets = 100] = 0.358
	[IoU = 0.5 area = all maxDets = 1000] = 0.763
	[IoU = 0.75 area = all maxDets = 1000] = 0.287
	[IoU = 0.5:0.95 area = small maxDets = 1000] = 0.332
	[IoU = 0.5:0.95 area = medium maxDets = 1000] = 0.55
Average Recall (AR)	[IoU = 0.5:0.95 area = all maxDets = 100] = 0.436
	[IoU = 0.5 area = all maxDets = 300] = 0.436
	[IoU = 0.75 area = all maxDets = 1000] = 0.436
	[IoU = 0.5:0.95 area = small maxDets = 1000] = 0.416
	[IoU = 0.5:0.95 area = medium maxDets = 1000] = 0.608

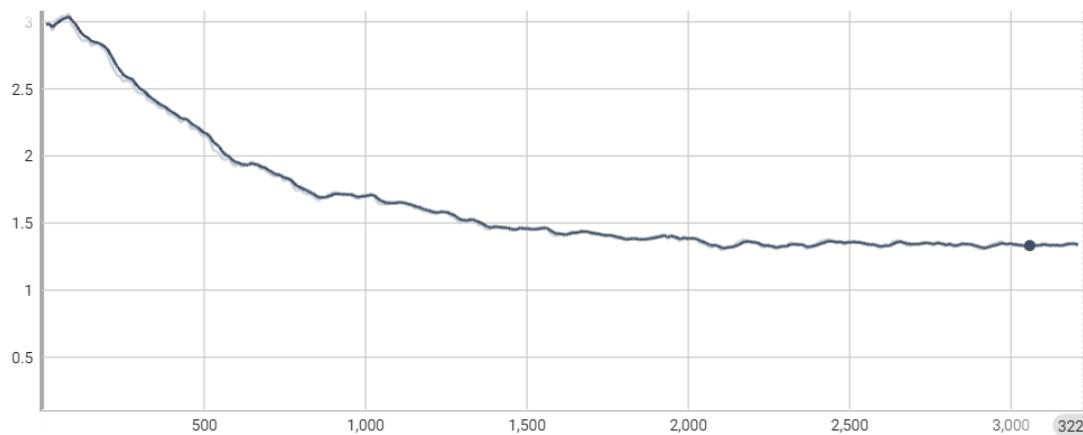


Figura 11: loss validación lr = 0.000025

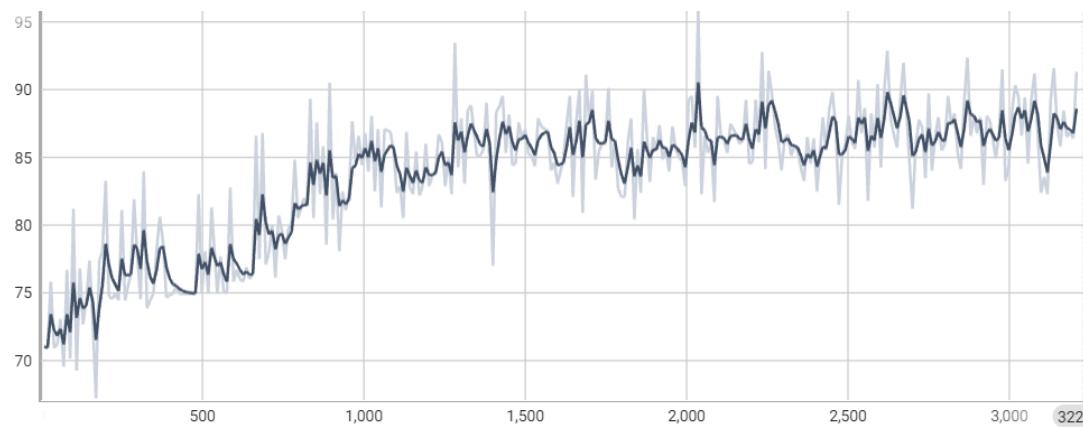


Figura 12: accuracy validación lr = 0.000025

¿Que significan estas métricas?

- Average Precision (AP):

- IoU=0.50:0.95: Mide la precisión promedio para diferentes niveles de superposición (IoU) entre las predicciones del modelo y las anotaciones de referencia. Se consideran todos los objetos y hasta 100 detecciones por imagen.
 - IoU=0.50: Mide la precisión promedio con un umbral de IoU del 50 %.
 - IoU=0.75: Mide la precisión promedio con un umbral de IoU del 75 %.
 - IoU=0.50:0.95 (small, medium, large): Mide la precisión promedio para objetos pequeños, medianos y grandes.
- Average Recall (AR):
- IoU=0.50:0.95: Mide el recall promedio para diferentes niveles de superposición (IoU) entre las predicciones y las anotaciones de referencia. Se consideran todos los objetos y hasta 100, 300 o 1000 detecciones por imagen, respectivamente.
 - IoU=0.50:0.95 (small, medium, large): Mide el recall promedio para objetos pequeños, medianos y grandes.

3.3.3. Análisis de resultados fine tuning a distintos learning rates y Drop Out

Observando los resultados, se obtienen las siguientes conclusiones:

- El ratio de aprendizaje de 0.0025 genera resultados muy deficientes, y no han sido agregados a este informe ya que su capacidad de detección es prácticamente cero, esto puede ser debido a que es muy grande como para que el modelo aprenda correctamente, ya sea porque la actualización de parámetros es demasiado grande y hace que el modelo diverja o salte alrededor de la región óptima en lugar de converger.
- Los mejores resultados se obtienen con el learning rate de 0.00025, mas pequeño que el anteriormente mencionado, cuyas métricas se muestran en la tabla 2 y sus loss y accuracy en las figuras 5 y 6 respectivamente, entregando una inicialización adecuada para que el modelo se ajuste correctamente a los nuevos datos.
- El loss y accuracy mostrados en las figuras para cada combinación de loss y accuracy, muestran que efectivamente el modelo se avanza correctamente en el entrenamiento, sin embargo, la fuente mas confiable para analizar el rendimiento del modelo al aplicar fine tuning siguen siendo las métricas mostradas en las tablas.
- Cabe mencionar, además, que la implementación de MMDetection utiliza muchas técnicas avanzadas como regularización, drop out, learning rate variable, optimizador Adam, etc, sin embargo, se tienen el poder de fijar las inicializaciones del modelo al entrenarse, por ejemplo, a pesar de que los mejores resultados sean con un drop_rate general de 0.0, existe otro parámetro de drop out asociado a las conexiones residuales (drop_path_rate) en el bloque del transformador, y en este caso, está configurada en 0.2, lo que indica que hay un dropout del 20 % en las conexiones residuales por defecto.
- No se han utilizado otras técnicas como data augmentation o similares ya que el entenamiento se realizó utilizando los archivos json de anotaciones, pudiendo ser necesario procesamiento extra para este archivo lo cual se escapa de los aprendizajes de este proyecto.

- Por otro lado, los datos de minneapple utilizados no son muchos, por lo tanto en principio no es preocupante el sobreajuste, sin embargo, es necesario comparar las losses de entrenamiento y validación para confirmar este aspecto, información que en primera instancia no se entrega por MMDetection.
- Existe una métrica de AP y AR que no se muestra en las tablas que está asociado al area=large, que tiene como valor -1, sin embargo esto no tiene sentido, puesto que al no haber instancias grandes que detectar la métrica se indefine no aportando información relevante.

3.3.4. Pruebas al conjunto de test

Utilizando la mejor configuración del modelo, se aplica para obtener predicciones nuevamente de las 5 imágenes del conjunto de test para visualizar si el modelo efectivamente ha mejorado con las técnicas aplicadas, lo cual definirá si el proyecto ha sido exitoso o no:



Figura 13: test para modelo entrenado con Minneapple.

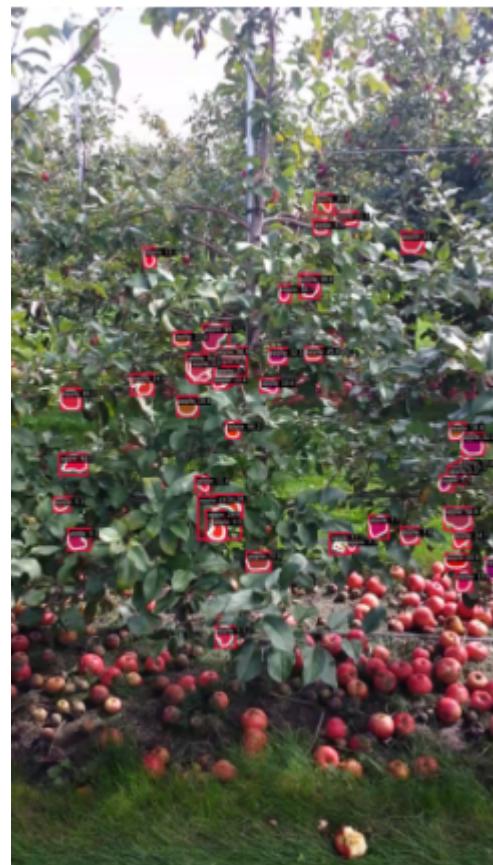


Figura 14: test para modelo entrenado con Minneapple.



Figura 15: test para modelo entrenado con Minneapple.



Figura 16: test para modelo entrenado con Minneapple.

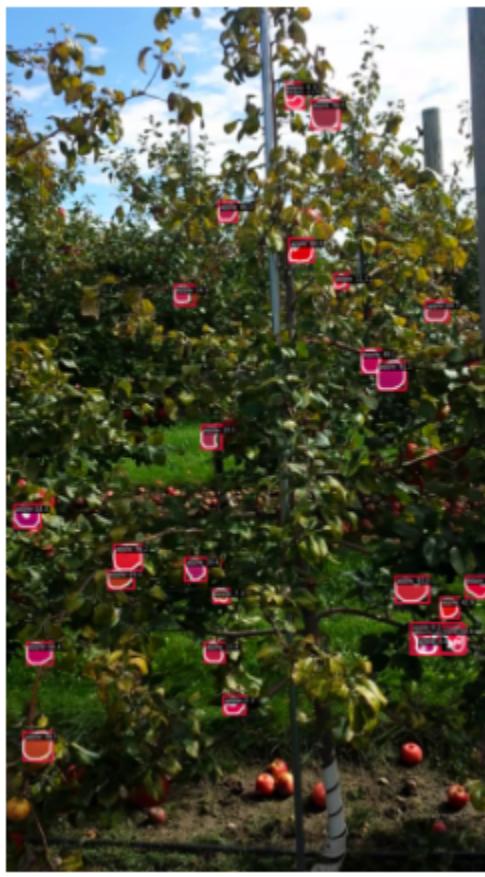


Figura 17: test para modelo entrenado con Minneapple.

Como podemos observar, efectivamente el modelo con fine tuning y con la mucha exploración de hiperparámetros, ha mejorado notablemente respecto a lo que se tenía en un inicio solo con el modelo preentrenado, mostrando ya una utilización práctica en el mundo real y alcanzando una proporción de detecciones más que suficiente según lo esperado.

4. Conclusiones

En esta implementación de detección de objetos utilizando MMDetection, se abordaron varios desafíos significativos. La aplicación efectiva del fine-tuning se presentó como un aspecto desafiante, ya que la correcta configuración de hiperparámetros y la exploración de las opciones adecuadas requirieron un tiempo considerable. El ajuste fino de modelos preentrenados es crucial, pero su éxito depende en gran medida de la adaptación a la naturaleza específica del conjunto de datos objetivo.

La exploración de hiperparámetros se vio limitada por el extenso tiempo de entrenamiento requerido para modelos de detección de objetos, lo que implicó un enfoque cuidadoso y selectivo para evitar tiempos de experimentación excesivos. Este desafío subraya la importancia de la eficiencia computacional al trabajar con modelos de aprendizaje profundo, especialmente en tareas de detección que suelen ser más intensivas en recursos.

A pesar de estos desafíos, los resultados obtenidos demostraron coherencia con la teoría subyacente. La aplicación exitosa de transfer learning, aprovechando modelos preentrenados en conjuntos de datos masivos como COCO, destacó la utilidad de esta técnica para tareas de detección de objetos en dominios específicos. Los modelos entrenados exhibieron capacidades de generalización y rendimiento que se alinearon con las expectativas teóricas.

El proyecto también proporcionó valiosos aprendizajes sobre la implementación práctica de modelos de detección de objetos utilizando MMDetection. Se adquirió experiencia en la manipulación de conjuntos de datos personalizados, la adaptación de arquitecturas de modelos y la comprensión de los desafíos específicos asociados con la tarea. La importancia de la selección adecuada de modelos preentrenados, la configuración de hiperparámetros y la gestión eficiente de recursos informáticos se destacaron como lecciones fundamentales.

5. Anexo

A continuación se muestra la disposición de tiempos en esta tarea:

	Bibliografía	Conjunto de datos	Modelo preentrenado	Fine tuning	Prueba de hiperparámetros	Exploración de técnicas	Informe y entregables
Semana 1							
Semana 2							
Semana 3							
Semana 4							
Semana 5							

Figura 18: test para modelo entrenado con Minneapple.

A continuación el link de guthub:

<https://github.com/IsaacJordan-bit/ProyectoFinal>

A continuación se adjunta el código utilizado:

Código 27: Código de python utilizado para generar los resultados.

```

1
2
3 # -*- coding: utf-8 -*-
4 """Proyecto_final.ipynb
5
6 Automatically generated by Colaboratory.
7
8 Original file is located at
9 https://colab.research.google.com/drive/1-mfKuSeWCInMxTpsQ7Gvryb0zr_KrbvB
10
11 # Preparación de los conjuntos
12 """
13
14 import shutil

```

```
15
16 from google.colab import files
17 import json
18
19 # Función para cargar un archivo JSON
20 def cargar_json():
21     # Subir el archivo JSON
22     uploaded = files.upload()
23
24     # Obtener el nombre del archivo
25     file_name = list(uploaded.keys())[0]
26
27     # Cargar el contenido del archivo JSON
28     with open(file_name, 'r') as file:
29         data = json.load(file)
30
31     return data
32
33 # Función para mostrar información selectiva sobre el JSON
34 def mostrar_informacion_selectiva(data, keys_to_display=None, max_items=5):
35     print("Contenido del JSON (primeros {} elementos):".format(max_items))
36
37     # Mostrar solo algunas claves específicas o todas si no se especifica ninguna
38     keys_to_display = keys_to_display or list(data.keys())
39
40     for key in keys_to_display:
41         if key in data:
42             print(f"\n{key}: {data[key]}")
43
44     print("\nInformación sobre el JSON:")
45     print("Número total de elementos:", len(data))
46     print("Tipo de datos:", type(data))
47     print("Claves:", list(data.keys()))
48
49 """
50 # Cargar el JSON
51 instance_train = cargar_json()
52
53 # Especificar las claves que te gustaría mostrar o dejar como None para mostrar todas las claves
54 claves_a_mostrar = ["clave1", "clave2", "clave3"]
55
56 # Mostrar información selectiva sobre el JSON
57 mostrar_informacion_selectiva(instance_train, keys_to_display=claves_a_mostrar)
58 """
59
60 """
61 # Cargar el JSON
62 instance_val = cargar_json()
63
64 # Especificar las claves que te gustaría mostrar o dejar como None para mostrar todas las claves
65 claves_a_mostrar = ["clave1", "clave2", "clave3"]
```

```
67 # Mostrar información selectiva sobre el JSON
68 mostrar_informacion_selectiva(instance_val, keys_to_display=claves_a_mostrar)
69 """
70 """
71 """
72 print(instance_val['images'][1])
73 print(len(instance_val['images']))
74 """
75 """
76 """
77 print(instance_val['annotations'][1])
78 print(len(instance_val['annotations']))
79 """
80
81 from google.colab import drive
82
83 # Monta Google Drive en el directorio '/content/drive'
84 drive.mount('/content/drive')
85
86 import os
87
88 # Ruta del directorio principal
89 base_directory = '/content/drive/MyDrive/Procesamiento avanzado de imagenes/Proyecto final/
90     appledataset'
91
92 # Nombre de la carpeta que deseas crear
93 folder_name = 'val'
94
95 # Ruta completa de la carpeta a crear
96 folder_path = os.path.join(base_directory, folder_name)
97
98 # Verificar si la carpeta ya existe o no
99 if not os.path.exists(folder_path):
100     # Crear la carpeta si no existe
101     os.makedirs(folder_path)
102     print(f'Se ha creado la carpeta {folder_name} en {base_directory}')
103 else:
104     print(f'La carpeta {folder_name} ya existe en {base_directory}')
105 import os
106
107 # Ruta del directorio principal
108 base_directory = '/content/drive/MyDrive/Procesamiento avanzado de imagenes/Proyecto final/
109     appledataset/val'
110
111 # Nombre de la carpeta que deseas crear
112 folder_name_images = 'images'
113 folder_name_masks = 'masks'
114
115 # Ruta completa de la carpeta a crear
116 folder_path_images = os.path.join(base_directory, folder_name_images)
```

```
117 folder_path_masks = os.path.join(base_directory, folder_name_masks)
118
119
120 # Verificar si la carpeta ya existe o no
121 if not os.path.exists(folder_path_images):
122     # Crear la carpeta si no existe
123     os.makedirs(folder_path_images)
124     print(f'Se ha creado la carpeta {folder_name_images} en {base_directory}')
125 else:
126     print(f'La carpeta {folder_name_images} ya existe en {base_directory}')
127
128
129 # Verificar si la carpeta ya existe o no
130 if not os.path.exists(folder_path_masks):
131     # Crear la carpeta si no existe
132     os.makedirs(folder_path_masks)
133     print(f'Se ha creado la carpeta {folder_name_masks} en {base_directory}')
134 else:
135     print(f'La carpeta {folder_name_masks} ya existe en {base_directory}')
136
137 instance_train = cargar_json()
138
139 instance_val = cargar_json()
140
141 import os
142 import shutil
143
144
145 source_directory_images = '/content/drive/MyDrive/Procesamiento avanzado de imagenes/Proyecto final/appledataset/train/images'
146 source_directory_masks = '/content/drive/MyDrive/Procesamiento avanzado de imagenes/Proyecto final/appledataset/train/masks'
147
148 destination_directory_images = '/content/drive/MyDrive/Procesamiento avanzado de imagenes/Proyecto final/appledataset/val/images'
149 destination_directory_masks = '/content/drive/MyDrive/Procesamiento avanzado de imagenes/Proyecto final/appledataset/val/masks'
150
151 len_val = len(instance_val['images'])
152
153 for i in range(len_val):
154     val_image = instance_val['images'][i]['file_name']
155     source_path_images = os.path.join(source_directory_images, val_image)
156     source_path_masks = os.path.join(source_directory_masks, val_image)
157
158     destination_path_images = os.path.join(destination_directory_images, val_image)
159     destination_path_masks = os.path.join(destination_directory_masks, val_image)
160
161     # Verificar si el archivo ya existe en el directorio de destino
162     if not os.path.exists(destination_path_images) and not os.path.exists(destination_path_masks):
163         # Mover el archivo solo si no existe en el directorio de destino
164         shutil.move(source_path_images, destination_path_images)
```

```
165     shutil.move(source_path_masks, destination_path_masks)
166
167     print(f'Se ha movido el archivo {val_image} de {source_directory_images} a {destination_directory_images}')
168     print(f'Se ha movido el archivo {val_image} de {source_directory_masks} a {destination_directory_masks}')
169 else:
170     print(f'El archivo {val_image} ya existe en {destination_directory_images} y {destination_directory_masks}. No se ha movido.')
171
172 !pip install -U openmim
173 !mim install mmengine
174 !mim install "mmcv>=2.0.0"
175
176 # Commented out IPython magic to ensure Python compatibility.
177 # Install mmdetection
178 !rm -rf mmdetection
179 !git clone https://github.com/open-mmlab/mmdetection.git
180 # %cd mmdetection
181
182 !pip install -v -e .
183
184 !pwd
185
186 # Check Pytorch installation
187 import torch, torchvision
188 print("torch version:",torch.__version__, "cuda:",torch.cuda.is_available())
189
190 # Check MMDetection installation
191 import mmdet
192 print("mmdetection:",mmdet.__version__)
193
194 # Check mmcv installation
195 import mmcv
196 print("mmcv:",mmcv.__version__)
197
198 # Check mmengine installation
199 import mmengine
200 print("mmengine:",mmengine.__version__)
201
202 !mim download mmdet --config mask-rcnn_swin-t-p4-w7_fpn_1x_coco --dest ./checkpoints
203
204 import mmcv
205 import mmengine
206 from mmdet.apis import init_detector, inference_detector
207 from mmdet.utils import register_all_modules
208 # Choose to use a config and initialize the detector
209 config_file = 'configs/swin/mask-rcnn_swin-t-p4-w7_fpn_1x_coco.py'
210 # Setup a checkpoint file to load
211 checkpoint_file = 'checkpoints/mask_rcnn_swin-t-p4-w7_fpn_1x_coco_20210902_120937-9d6b7cfa.pth'
212
```

```
213 # register all modules in mmdet into the registries
214 register_all_modules()
215
216 # build the model from a config file and a checkpoint file
217 #model = init_detector(config_file, checkpoint_file, device='cuda:0') # or device='cuda:0'
218
219 try:
220     model = init_detector(config_file, checkpoint_file, device='cuda:0')
221     print("Modelo inicializado en GPU.")
222 except RuntimeError as e:
223     # Si hay un error, intenta inicializar el modelo en CPU
224     print(f"Error al inicializar en GPU: {e}")
225     print("Intentando inicializar en CPU.")
226     model = init_detector(config_file, checkpoint_file, device='cpu')
227     print("Modelo inicializado en CPU.")
228
229 # Use the detector to do inference
230 image = mmcv.imread('/content/drive/MyDrive/Procesamiento avanzado de imagenes/Proyecto final
231             /appledataset/test/images/dataset1_back_1141.png',channel_order='rgb')
232 result = inference_detector(model, image)
233 print(result)
234
235 from mmdet.registry import VISUALIZERS
236 # init visualizer(run the block only once in jupyter notebook)
237 visualizer = VISUALIZERS.build(model.cfg.visualizer)
238 # the dataset_meta is loaded from the checkpoint and
239 # then pass to the model in init_detector
240 visualizer.dataset_meta = model.dataset_meta
241
242 # show the results
243 visualizer.add_datasample(
244     'result',
245     image,
246     data_sample=result,
247     draw_gt = False,
248     wait_time=0,
249     pred_score_thr=0.2
250 )
251 visualizer.show()
252
253 # Use the detector to do inference
254 image = mmcv.imread('/content/drive/MyDrive/Procesamiento avanzado de imagenes/Proyecto final
255             /appledataset/test/images/dataset1_back_1081.png',channel_order='rgb')
256 result = inference_detector(model, image)
257 print(result)
258
259 # init visualizer(run the block only once in jupyter notebook)
260 visualizer = VISUALIZERS.build(model.cfg.visualizer)
261 # the dataset_meta is loaded from the checkpoint and
262 # then pass to the model in init_detector
263 visualizer.dataset_meta = model.dataset_meta
```

```
263 # show the results
264 visualizer.add_datasample(
265     'result',
266     image,
267     data_sample=result,
268     draw_gt = False,
269     wait_time=0,
270     pred_score_thr=0.2
271 )
272 visualizer.show()
273
274 # Use the detector to do inference
275 image = mmcv.imread('/content/drive/MyDrive/Procesamiento avanzado de imagenes/Proyecto final
276 /appledataset/test/images/dataset1_back_1.png',channel_order='rgb')
277 result = inference_detector(model, image)
278 print(result)
279
280 # init visualizer(run the block only once in jupyter notebook)
281 visualizer = VISUALIZERS.build(model.cfg.visualizer)
282 # the dataset_meta is loaded from the checkpoint and
283 # then pass to the model in init_detector
284 visualizer.dataset_meta = model.dataset_meta
285
286 # show the results
287 visualizer.add_datasample(
288     'result',
289     image,
290     data_sample=result,
291     draw_gt = False,
292     wait_time=0,
293     pred_score_thr=0.001
294 )
295 visualizer.show()
296
297 # Use the detector to do inference
298 image = mmcv.imread('/content/drive/MyDrive/Procesamiento avanzado de imagenes/Proyecto final
299 /appledataset/test/images/dataset1_back_1021.png',channel_order='rgb')
300 result = inference_detector(model, image)
301 print(result)
302
303 # init visualizer(run the block only once in jupyter notebook)
304 visualizer = VISUALIZERS.build(model.cfg.visualizer)
305 # the dataset_meta is loaded from the checkpoint and
306 # then pass to the model in init_detector
307 visualizer.dataset_meta = model.dataset_meta
308
309 # show the results
310 visualizer.add_datasample(
311     'result',
312     image,
313     data_sample=result,
314     draw_gt = False,
```

```
313     wait_time=0,
314     pred_score_thr=0.1
315 )
316 visualizer.show()
317
318 # Use the detector to do inference
319 image = mmcv.imread('/content/drive/MyDrive/Procesamiento avanzado de imagenes/Proyecto final
320                 /appledataset/test/images/dataset1_back_1051.png',channel_order='rgb')
320 result = inference_detector(model, image)
321 print(result)
322
323 # init visualizer(run the block only once in jupyter notebook)
324 visualizer = VISUALIZERS.build(model.cfg.visualizer)
325 # the dataset_meta is loaded from the checkpoint and
326 # then pass to the model in init_detector
327 visualizer.dataset_meta = model.dataset_meta
328
329 # show the results
330 visualizer.add_datasample(
331     'result',
332     image,
333     data_sample=result,
334     draw_gt = False,
335     wait_time=0,
336     pred_score_thr=0.2
337 )
338 visualizer.show()
339
340 """# Modelo preentrenado en MMDetection"""
341
342 import os
343
344 # Ruta del directorio donde deseas crear la carpeta
345 directorio_mmdetection = '/content/mmdetection'
346
347 # Nombre de la carpeta que deseas crear
348 nombre_carpeta = 'minneappledataset'
349
350 # Ruta completa de la nueva carpeta
351 ruta_nueva_carpeta = os.path.join(directorio_mmdetection, nombre_carpeta)
352
353 # Verificar si la carpeta ya existe antes de intentar crearla
354 if not os.path.exists(ruta_nueva_carpeta):
355     # Crear la carpeta
356     os.makedirs(ruta_nueva_carpeta)
357     print(f'Carpetas "{nombre_carpeta}" creada en {directorio_mmdetection}')
358 else:
359     print(f'La carpeta "{nombre_carpeta}" ya existe en {directorio_mmdetection}')
360
361 import os
362
363 # Ruta del directorio donde deseas crear la carpeta
```

```
364 directorio_mmdetection = '/content/mmdetection/minneappledataset'  
365  
366 # Nombre de la carpeta que deseas crear  
367 nombre_carpeta = 'minneapple'  
368  
369 # Ruta completa de la nueva carpeta  
370 ruta_nueva_carpeta = os.path.join(directorio_mmdetection, nombre_carpeta)  
371  
372 # Verificar si la carpeta ya existe antes de intentar crearla  
373 if not os.path.exists(ruta_nueva_carpeta):  
374     # Crear la carpeta  
375     os.makedirs(ruta_nueva_carpeta)  
376     print(f'Carpeta "{nombre_carpeta}" creada en {directorio_mmdetection}')  
377 else:  
378     print(f'La carpeta "{nombre_carpeta}" ya existe en {directorio_mmdetection}')  
379  
380 import os  
381  
382 # Ruta del directorio donde deseas crear la carpeta  
383 directorio_mmdetection = '/content/mmdetection/minneappledataset/minneapple'  
384  
385 # Nombre de la carpeta que deseas crear  
386 nombre_carpeta = 'train'  
387  
388 # Ruta completa de la nueva carpeta  
389 ruta_nueva_carpeta = os.path.join(directorio_mmdetection, nombre_carpeta)  
390  
391 # Verificar si la carpeta ya existe antes de intentar crearla  
392 if not os.path.exists(ruta_nueva_carpeta):  
393     # Crear la carpeta  
394     os.makedirs(ruta_nueva_carpeta)  
395     print(f'Carpeta "{nombre_carpeta}" creada en {directorio_mmdetection}')  
396 else:  
397     print(f'La carpeta "{nombre_carpeta}" ya existe en {directorio_mmdetection}')  
398  
399  
400 # Ruta del directorio donde deseas crear la carpeta  
401 directorio_mmdetection = '/content/mmdetection/minneappledataset/minneapple'  
402  
403 # Nombre de la carpeta que deseas crear  
404 nombre_carpeta = 'val'  
405  
406 # Ruta completa de la nueva carpeta  
407 ruta_nueva_carpeta = os.path.join(directorio_mmdetection, nombre_carpeta)  
408  
409 # Verificar si la carpeta ya existe antes de intentar crearla  
410 if not os.path.exists(ruta_nueva_carpeta):  
411     # Crear la carpeta  
412     os.makedirs(ruta_nueva_carpeta)  
413     print(f'Carpeta "{nombre_carpeta}" creada en {directorio_mmdetection}')  
414 else:  
415     print(f'La carpeta "{nombre_carpeta}" ya existe en {directorio_mmdetection}')
```

```
416
417 # Rutas de origen y destino
418 ruta_origen = '/content/drive/MyDrive/Procesamiento avanzado de imagenes/Proyecto final/
419     appledataset/train/images'
420 ruta_destino = '/content/mmdetection/minneappledataset/minneapple/train'
421
422 # Verificar si la carpeta de destino existe
423 if not os.path.exists(ruta_destino):
424     os.makedirs(ruta_destino)
425
426 # Obtener la lista de archivos en la carpeta de origen
427 archivos_a_copiar = [archivo for archivo in os.listdir(ruta_origen) if os.path.isfile(os.path.join(
428     ruta_origen, archivo))]
429
430 # Copiar cada archivo a la carpeta de destino si no existe en el destino
431 for archivo in archivos_a_copiar:
432     ruta_archivo_origen = os.path.join(ruta_origen, archivo)
433     ruta_archivo_destino = os.path.join(ruta_destino, archivo)
434
435     # Verificar si el archivo no existe en el destino antes de copiar
436     if not os.path.exists(ruta_archivo_destino):
437         shutil.copy(ruta_archivo_origen, ruta_archivo_destino)
438         print(f'Archivo copiado de "{ruta_origen}" a "{ruta_destino}": {archivo}')
439     else:
440         print(f'El archivo ya existe en "{ruta_destino}": {archivo}')
441
442 # Rutas de origen y destino
443 ruta_origen = '/content/drive/MyDrive/Procesamiento avanzado de imagenes/Proyecto final/
444     appledataset/val/images'
445 ruta_destino = '/content/mmdetection/minneappledataset/minneapple/val'
446
447 # Verificar si la carpeta de destino existe
448 if not os.path.exists(ruta_destino):
449     os.makedirs(ruta_destino)
450
451 # Obtener la lista de archivos en la carpeta de origen
452 archivos_a_copiar = [archivo for archivo in os.listdir(ruta_origen) if os.path.isfile(os.path.join(
453     ruta_origen, archivo))]
454
455 # Copiar cada archivo a la carpeta de destino si no existe en el destino
456 for archivo in archivos_a_copiar:
457     ruta_archivo_origen = os.path.join(ruta_origen, archivo)
458     ruta_archivo_destino = os.path.join(ruta_destino, archivo)
459
460     # Verificar si el archivo no existe en el destino antes de copiar
461     if not os.path.exists(ruta_archivo_destino):
462         shutil.copy(ruta_archivo_origen, ruta_archivo_destino)
463         print(f'Archivo copiado de "{ruta_origen}" a "{ruta_destino}": {archivo}')
464     else:
465         print(f'El archivo ya existe en "{ruta_destino}": {archivo}'
```

```
464 print('Proceso completado.')
465
466
467 import shutil
468 import os
469
470 # Función para mover el archivo si no existe en la carpeta de destino
471 def move_if_not_exists(source_file, destination_folder):
472     destination_file = os.path.join(destination_folder, os.path.basename(source_file))
473
474     # Verificar si el archivo no existe en la carpeta de destino antes de mover
475     if not os.path.exists(destination_file):
476         # Mover el archivo
477         shutil.move(source_file, destination_folder)
478         print(f"El archivo {source_file} se ha movido a {destination_folder}.")
479     else:
480         print(f"El archivo ya existe en '{destination_folder}': {os.path.basename(source_file)}")
481
482 # Rutas de los archivos
483 source_file_train = '/content/instances_train.json'
484 destination_folder_train = '/content/mmdetection/minneappledataset/minneapple/train'
485 move_if_not_exists(source_file_train, destination_folder_train)
486
487 # Rutas de los archivos
488 source_file_val = '/content/instances_val.json'
489 destination_folder_val = '/content/mmdetection/minneappledataset/minneapple/val'
490 move_if_not_exists(source_file_val, destination_folder_val)
491
492 import os
493
494 # Ruta de la carpeta de imágenes de entrenamiento
495 folder_path = '/content/mmdetection/minneappledataset/minneapple/train'
496
497 # Contador de imágenes
498 image_count = 0
499
500 # Recorre todos los archivos en la carpeta
501 for filename in os.listdir(folder_path):
502     # Verifica si el archivo es una imagen (puedes ajustar las extensiones según tu necesidad)
503     if filename.lower().endswith('.png', '.jpg', '.jpeg'):
504         image_count += 1
505
506 print(f"El número total de imágenes en {folder_path} es: {image_count}")
507
508 import os
509
510 # Ruta de la carpeta de imágenes de entrenamiento
511 folder_path = '/content/mmdetection/minneappledataset/minneapple/val'
512
513 # Contador de imágenes
514 image_count = 0
515
```

```
516 # Recorre todos los archivos en la carpeta
517 for filename in os.listdir(folder_path):
518     # Verifica si el archivo es una imagen (puedes ajustar las extensiones según tu necesidad)
519     if filename.lower().endswith('.png', '.jpg', '.jpeg'):
520         image_count += 1
521
522 print(f"El número total de imágenes en {folder_path} es: {image_count}")
523
524 import json
525
526 def is_coco_format(json_data):
527     try:
528         # Verificar si el archivo tiene las claves y estructura típicas de COCO
529         required_keys = ['images', 'annotations', 'categories']
530         for key in required_keys:
531             if key not in json_data:
532                 return False
533
534         # Verificar si las claves 'id', 'image_id', 'category_id', 'bbox' están presentes en 'annotations'
535         annotation_keys = json_data['annotations'][0].keys()
536         annotation_required_keys = ['id', 'image_id', 'category_id', 'bbox']
537         for key in annotation_required_keys:
538             if key not in annotation_keys:
539                 return False
540
541         # Si todo está bien, devuelve True
542         return True
543
544     except Exception as e:
545         print(f"Error al verificar el formato COCO: {e}")
546         return False
547
548 # Ruta a tu archivo JSON
549 json_path = '/content/mmdetection/minneappledataset/minneapple/train/instances_train.json'
550
551 # Cargar el archivo JSON
552 with open(json_path, 'r') as f:
553     json_data = json.load(f)
554
555 # Verificar el formato COCO
556 if is_coco_format(json_data):
557     print("El archivo JSON está en formato COCO.")
558 else:
559     print("El archivo JSON no cumple con el formato COCO.")
560
561 # Ruta a tu archivo JSON
562 json_path = '/content/mmdetection/minneappledataset/minneapple/val/instances_val.json'
563
564 # Cargar el archivo JSON
565 with open(json_path, 'r') as f:
566     json_data = json.load(f)
567
```

```
568 # Verificar el formato COCO
569 if is_coco_format(json_data):
570     print("El archivo JSON está en formato COCO.")
571 else:
572     print("El archivo JSON no cumple con el formato COCO.")
573
574 from pycocotools.coco import COCO
575
576 # Path to load the COCO annotation file
577 annotation_file = '/content/mmdetection/minneappledataset/minneapple/train/instances_train.json'
578
579 # Initialise the COCO object
580 coco = COCO(annotation_file)
581
582 # Get all category tags and corresponding category IDs
583 categories = coco.loadCats(coco.getCatIds())
584 category_id_to_name = {cat['id']: cat['name'] for cat in categories}
585
586 # Print all category IDs and corresponding category names
587 for category_id, category_name in category_id_to_name.items():
588     print(f"Category ID: {category_id}, Category Name: {category_name}")
589
590 from mmengine import Config
591 cfg = Config.fromfile('./configs/swin/mask-rcnn_swin-t-p4-w7_fpn_1x_coco.py')
592
593 from mmengine.runner import set_random_seed
594
595 # Modify dataset classes and color
596 cfg.metainfo = {
597     'classes': ('apple', ),
598     'palette': [
599         (220, 20, 60),
600     ]
601 }
602
603 # Modify dataset type and path
604 cfg.data_root = './minneappledataset/minneapple'
605
606 cfg.train_dataloader.dataset.ann_file = 'train/instances_train.json'
607 cfg.train_dataloader.dataset.data_root = cfg.data_root
608 cfg.train_dataloader.dataset.data_prefix.img = 'train/'
609 cfg.train_dataloader.dataset.metainfo = cfg.metainfo
610
611 cfg.val_dataloader.dataset.ann_file = 'val/instances_val.json'
612 cfg.val_dataloader.dataset.data_root = cfg.data_root
613 cfg.val_dataloader.dataset.data_prefix.img = 'val/'
614 cfg.val_dataloader.dataset.metainfo = cfg.metainfo
615
616 cfg.test_dataloader = cfg.val_dataloader
617
618 # Modify metric config
619 cfg.val_evaluator.ann_file = cfg.data_root + '/' + 'val/instances_val.json'
```

```
620 cfg.test_evaluator = cfg.val_evaluator
621
622 # Modify num classes of the model in box head and mask head
623 cfg.model.roi_head.bbox_head.num_classes = 1
624 cfg.model.roi_head.mask_head.num_classes = 1
625
626 # We can still the pre-trained Mask RCNN model to obtain a higher performance
627 cfg.load_from = 'checkpoints/mask_rcnn_swin-t-p4-w7_fpn_1x_coco_20210902_120937-9d6b7cfaf.pth'
628
629 # Set up working dir to save files and logs.
630 cfg.work_dir = './results'
631
632
633 # We can set the evaluation interval to reduce the evaluation times
634 cfg.train_cfg.val_interval = 3
635 # We can set the checkpoint saving interval to reduce the storage cost
636 cfg.default_hooks.checkpoint.interval = 3
637
638 # The original learning rate (LR) is set for 8-GPU training.
639 # We divide it by 8 since we only use one GPU.
640 cfg.optim_wrapper.optimizer.lr = 0.002 / 8
641 cfg.model.backbone.drop_rate = 0.5
642 cfg.default_hooks.logger.interval = 10
643
644
645 # Set seed thus the results are more reproducible
646 # cfg.seed = 0
647 set_random_seed(0, deterministic=False)
648
649 # We can also use tensorboard to log the training process
650 cfg.visualizer.vis_backends.append({'type': 'TensorboardVisBackend'})
651
652 #-----
653 config=f'./configs/swin/mask-rcnn_swin-t-p4-w7_fpn_1x_coco.py'
654 with open(config, 'w') as f:
655     f.write(cfg.pretty_text)
656
657 !python tools/train.py {config}
658
659 # Commented out IPython magic to ensure Python compatibility.
660 # %pip install tensorboard -i https://mirrors.ustc.edu.cn/pypi/web/simple
661
662 # Commented out IPython magic to ensure Python compatibility.
663 # load tensorboard in jupyter notebook
664 # %load_ext tensorboard
665
666 # Commented out IPython magic to ensure Python compatibility.
667 # see curves in tensorboard
668 # if you see <IPython.core.display.HTML object> please run it again
669 # %tensorboard --logdir results/
670
```

```
671 import mmcv
672 from mmdet.apis import init_detector, inference_detector
673 img = mmcv.imread('/content/drive/MyDrive/Procesamiento avanzado de imagenes/Proyecto final/
674     /appledataset/test/images/dataset1_back_1141.png',channel_order='rgb')
675 checkpoint_file = 'results/epoch_12.pth'
676 model = init_detector(cfg, checkpoint_file, device='cuda')
677 new_result = inference_detector(model, img)
678 print(new_result)
679
680 import mmcv
681 import mmenGINE
682 from mmdet.apis import init_detector, inference_detector
683 from mmdet.utils import register_all_modules
684 # Choose to use a config and initialize the detector
685 # Setup a checkpoint file to load
686 checkpoint_file = 'results/epoch_12.pth'
687
688 # register all modules in mmdet into the registries
689 register_all_modules()
690
691 # build the model from a config file and a checkpoint file
692 #model = init_detector(config_file, checkpoint_file, device='cuda:0') # or device='cuda:0'
693
694 try:
695     model = init_detector(cfg, checkpoint_file, device='cuda:0')
696     print("Modelo inicializado en GPU.")
697 except RuntimeError as e:
698     # Si hay un error, intenta inicializar el modelo en CPU
699     print(f"Error al inicializar en GPU: {e}")
700     print("Intentando inicializar en CPU.")
701     model = init_detector(config_file, checkpoint_file, device='cpu')
702     print("Modelo inicializado en CPU.")
703
704 # Use the detector to do inference
705 image = mmcv.imread('/content/drive/MyDrive/Procesamiento avanzado de imagenes/Proyecto final/
706     /appledataset/test/images/dataset1_back_1141.png',channel_order='rgb')
707 result = inference_detector(model, image)
708 print(result)
709
710 from mmdet.registry import VISUALIZERS
711 # init visualizer(run the block only once in jupyter notebook)
712 visualizer = VISUALIZERS.build(model.cfg.visualizer)
713 # the dataset_meta is loaded from the checkpoint and
714 # then pass to the model in init_detector
715 visualizer.dataset_meta = model.dataset_meta
716
717 # show the results
718 visualizer.add_datasample(
719     'result',
720     image,
721     data_sample=result,
722     draw_gt = False,
```

```
721     wait_time=0,  
722     pred_score_thr=0.05  
723 )  
724 visualizer.show()  
725  
726 # Use the detector to do inference  
727 image = mmcv.imread('/content/drive/MyDrive/Procesamiento avanzado de imagenes/Proyecto final  
    /appledataset/test/images/dataset1_back_1081.png',channel_order='rgb')  
728 result = inference_detector(model, image)  
729 print(result)  
730  
731 from mmdet.registry import VISUALIZERS  
732 # init visualizer(run the block only once in jupyter notebook)  
733 visualizer = VISUALIZERS.build(model.cfg.visualizer)  
734 # the dataset_meta is loaded from the checkpoint and  
735 # then pass to the model in init_detector  
736 visualizer.dataset_meta = model.dataset_meta  
737  
738 # show the results  
739 visualizer.add_datasample(  
    'result',  
    image,  
    data_sample=result,  
    draw_gt = False,  
    wait_time=0,  
    pred_score_thr=0.05  
)  
740  
741 visualizer.show()  
742  
743  
744 # Use the detector to do inference  
745 image = mmcv.imread('/content/drive/MyDrive/Procesamiento avanzado de imagenes/Proyecto final  
    /appledataset/test/images/dataset1_back_1.png',channel_order='rgb')  
746 result = inference_detector(model, image)  
747 print(result)  
748  
749 # init visualizer(run the block only once in jupyter notebook)  
750 visualizer = VISUALIZERS.build(model.cfg.visualizer)  
751 # the dataset_meta is loaded from the checkpoint and  
752 # then pass to the model in init_detector  
753 visualizer.dataset_meta = model.dataset_meta  
754  
755 # show the results  
756 visualizer.add_datasample(  
    'result',  
    image,  
    data_sample=result,  
    draw_gt = False,  
    wait_time=0,  
    pred_score_thr=0.05  
)  
757  
758 visualizer.show()  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768 )  
769 visualizer.show()  
770
```

```
771 image = mmcv.imread('/content/drive/MyDrive/Procesamiento avanzado de imágenes/Proyecto final  
    /appledataset/test/images/dataset1_back_1021.png',channel_order='rgb')  
772 result = inference_detector(model, image)  
773 print(result)  
774  
775 # init visualizer(run the block only once in jupyter notebook)  
776 visualizer = VISUALIZERS.build(model.cfg.visualizer)  
777 # the dataset_meta is loaded from the checkpoint and  
778 # then pass to the model in init_detector  
779 visualizer.dataset_meta = model.dataset_meta  
780  
781 # show the results  
782 visualizer.add_datasample(  
    'result',  
    image,  
    data_sample=result,  
    draw_gt = False,  
    wait_time=0,  
    pred_score_thr=0.05  
)  
789 )  
790 visualizer.show()  
791  
792 image = mmcv.imread('/content/drive/MyDrive/Procesamiento avanzado de imágenes/Proyecto final  
    /appledataset/test/images/dataset1_back_1051.png',channel_order='rgb')  
793 result = inference_detector(model, image)  
794 print(result)  
795  
796 # init visualizer(run the block only once in jupyter notebook)  
797 visualizer = VISUALIZERS.build(model.cfg.visualizer)  
798 # the dataset_meta is loaded from the checkpoint and  
799 # then pass to the model in init_detector  
800 visualizer.dataset_meta = model.dataset_meta  
801  
802 # show the results  
803 visualizer.add_datasample(  
    'result',  
    image,  
    data_sample=result,  
    draw_gt = False,  
    wait_time=0,  
    pred_score_thr=0.05  
)  
810 )  
811 visualizer.show()
```

6. Referencias

- [1] Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, Baining Guo. <https://arxiv.org/abs/2103.14030>
- [2] Github MMDetection. <https://github.com/open-mmlab/mmdetection>