

Trabajo Final

Visión por Computadora II

CEIA 2025 - Coh17


- Agustina Quirós (agustinaqr@gmail.com)
 - Agustín de la Vega (delavega.agus@gmail.com)
 - Florentino Arias (florito.arias@gmail.com)
- 



Tabla de contenidos

01

**Dataset Elegido y
Problema a Resolver**

02

**Exploración y
Comprensión de los datos**

03

Balance del Dataset

04

Experimentos

05

Conclusiones





01

Dataset Elegido y Problema a Resolver





Dataset Elegido

El dataset utilizado para este trabajo se denomina **PlantVillage Dataset**, fue obtenido de la plataforma Kaggle.

[LINK](#)

Problema a Resolver

Entrenar modelos de visión por computadora que identifiquen y clasifiquen hojas de plantas a partir de imágenes, para detectar si están sanas o enfermas y, en este último caso, de qué enfermedad se trata, ayudando a los agricultores a tomar medidas preventivas y reducir pérdidas.





02

Exploración y Comprensión de los datos



Exploración y Comprensión de los datos

Contenido de las carpetas del dataset PlantVillage

Data Explorer

2.18 GB

- ▶ color
- ▶ grayscale
- ▶ segmented

- Cada carpeta contiene un conjunto de carpetas, nombradas siguiendo el patrón:
NombrePlanta[_(alguna aclaracion)]_Nombre_enfermedad[_(alguna aclaracion)] o NombrePlanta[_(alguna aclaracion)]_healthy.
- Cada planta tiene una carpeta healthy y una o más carpetas relacionadas a enfermedades (una carpeta corresponde a una sola enfermedad).
- Cada subcarpeta en la carpeta 'color' tiene su correspondiente subcarpeta en 'grayscale' y en 'segmented'.
- La cantidad de imágenes dentro de cada carpeta es variable (hay carpetas que tienen cerca de 300 y otras que tienen más de 1000).

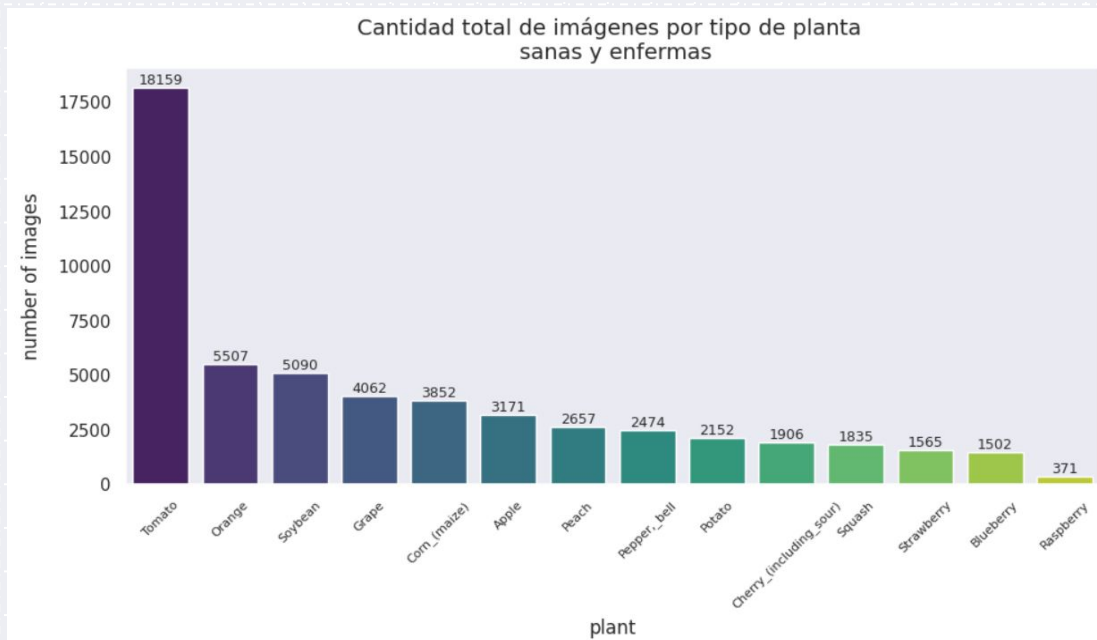
Exploración y Comprensión de los datos



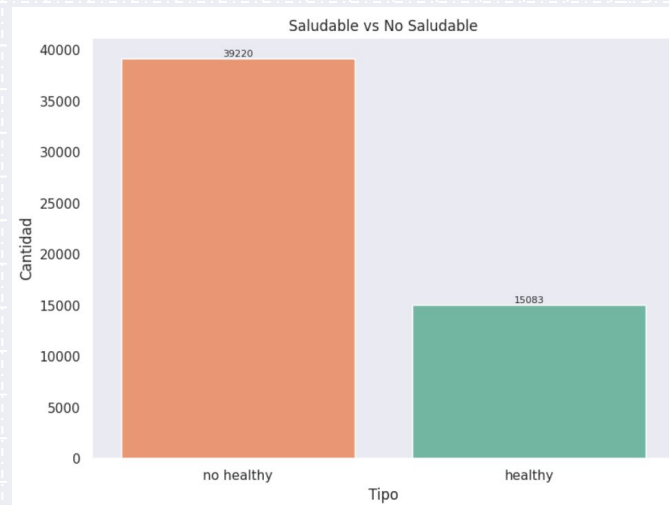
Tamaño promedio de imagen: 256.0 x 256.0

Exploración y Comprensión de los datos

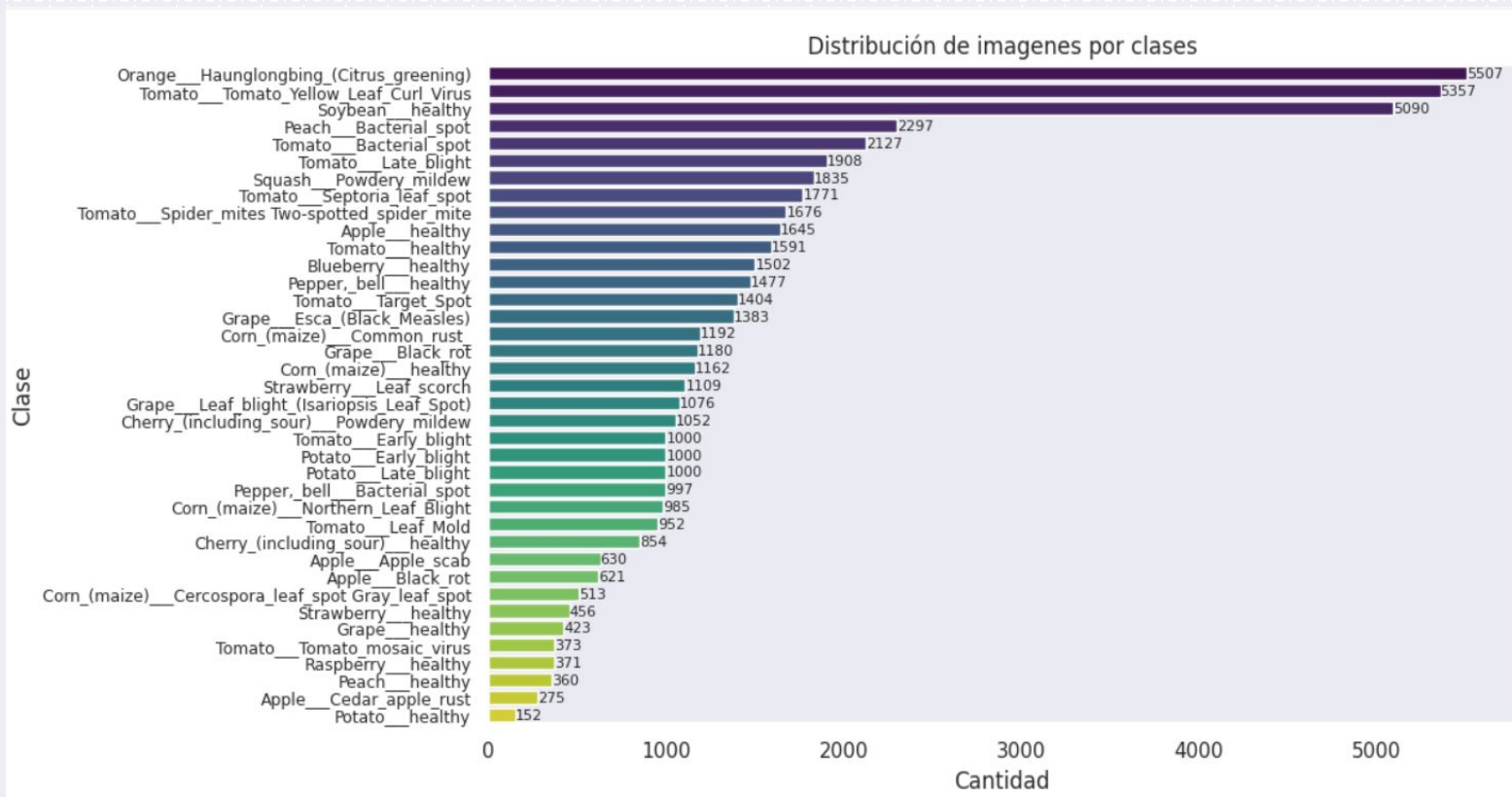
Variedades



Clases



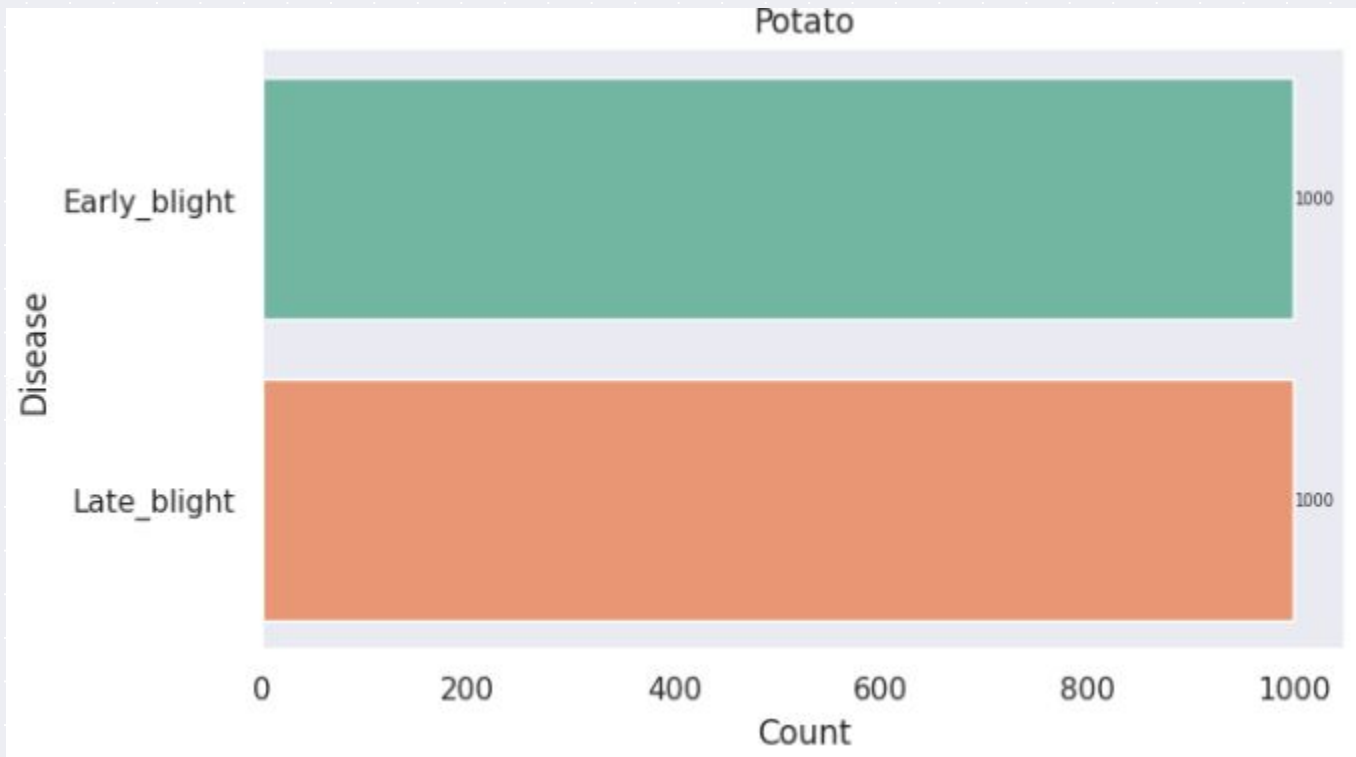
Exploración y Comprensión de los datos



Exploración y Comprensión de los datos



Exploración y Comprensión de los datos



03

Balance del Dataset



Data Augmentation y Class Weight

- Queremos evitar los problemas que podría traernos el desbalance de datos (5500 contra 152).
- Los modelos podrían inclinarse hacia las clases más grandes.
- El objetivo es aumentar la variabilidad y balancear el dataset.
- Normalizamos las imágenes.
- Aplicamos transformaciones aleatorias.
- Aplicamos pesos a las clases.



Probamos con las siguientes transformaciones:

- Rotaciones
- Desplazamientos
- Zoom
- Volteo horizontal y vertical

Damos pesos a las clases:

`class_weight='balanced'` compensa las clases minoritarias **dándole más peso al error** cuando se equivoca con ellas.

04

Experimentos

Entrenamiento de una CNN desde Cero - v1

Dividimos el dataset para entrenar y evaluar correctamente

Train (80%): se usa para ajustar los pesos del modelo.

Validation (20%): se usa durante el entrenamiento para ver si el modelo **está generalizando** o solo “memorizando” el train.

```
train_datagen = ImageDataGenerator(  
    rescale=1./255,  
    rotation_range=40,  
    width_shift_range=0.2,  
    height_shift_range=0.2,  
    shear_range=0.2,  
    zoom_range=0.2,  
    horizontal_flip=True,  
    validation_split=0.2 # 20%  
para validación  
)
```


Entrenamiento de una CNN desde Cero - v1

```
# Generadores
train_generator = train_datagen.flow_from_directory(
    path_de_trabajo,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='training'
)

val_generator = train_datagen.flow_from_directory(
    path_de_trabajo,
    target_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    subset='validation'
)
```

Entrenamiento de una CNN desde Cero - v1

Creamos una **red convolucional** básica con 3 capas Conv2D seguidas de MaxPooling2D.

Luego una capa Dense (512 neuronas), Dropout para regularización, y Dense final para clasificación (softmax para multiclase).

Entrenamiento de una CNN desde Cero - v1

```
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(IMG_SIZE[0], IMG_SIZE[1], 3)),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(num_classes, activation='softmax') # num_classes definido por
flow_from_directory
])

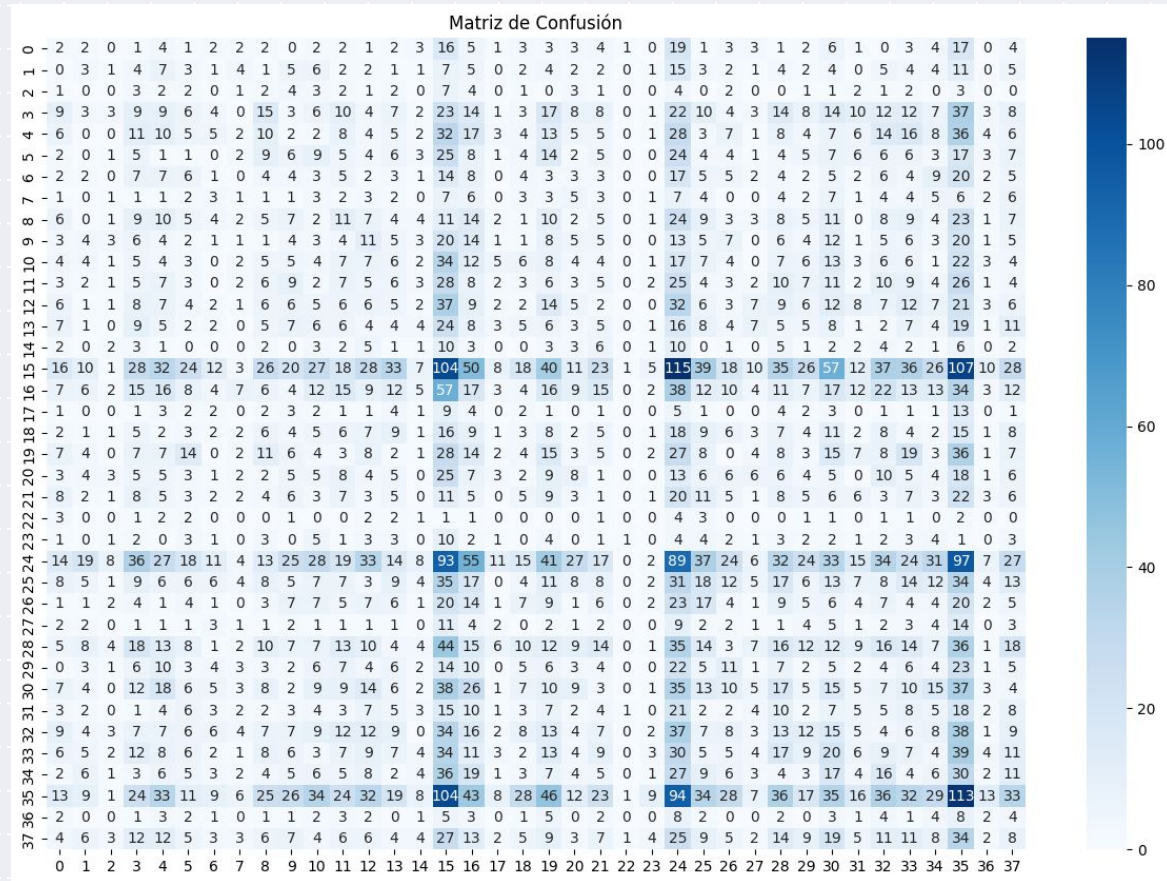
model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

Entrenamiento de una CNN desde Cero - v1

```
early_stop = EarlyStopping(monitor='val_loss', patience=3)

history = model.fit(
    train_generator,
    epochs=30,
    validation_data=val_generator,
    callbacks=[early_stop]
)
```

Entrenamiento de una CNN desde Cero - v1



Entrenamiento de una CNN desde Cero - v1

		precision	recall	f1-score	support
Orange	Tomato__Late_blight	0.01	0.02	0.01	126
	Tomato__healthy	0.02	0.02	0.02	124
	Grape__healthy	0.00	0.00	0.00	55
	Haunglongbing (Citrus greening)	0.03	0.03	0.03	329
Tomato	Strawberry__healthy	0.02	0.02	0.02	221
	Apple__healthy	0.01	0.01	0.01	91
	Grape__Black_rot	0.04	0.04	0.04	425
	Potato__Early_blight	0.01	0.01	0.01	200
	Cherry_(including_sour)__healthy	0.03	0.04	0.04	381
	Corn_(maize)__Common_rust_	0.03	0.03	0.03	190
	Grape__Esca_(Black_Measles)	0.01	0.01	0.01	354
	Raspberry__healthy	0.02	0.02	0.02	335
	Tomato__Leaf_Mold	0.02	0.02	0.02	280
	Two-spotted_spider_mite	0.11	0.11	0.11	1071
	Pepper,_bell__Bacterial_spot	0.02	0.03	0.02	74
	Corn_(maize)__healthy	0.03	0.03	0.03	318
accuracy				0.05	10849
macro avg		0.03	0.03	0.03	10849
weighted avg		0.05	0.05	0.05	10849

Entrenamiento de una CNN desde Cero - v2

Mejoras

Dividimos el dataset para entrenar y evaluar correctamente

Train (70%): se usa para ajustar los pesos del modelo.

Validation (15%): se usa durante el entrenamiento para ver si el modelo **está generalizando** o solo “memorizando” el train.

Test (15%): se guarda para el final. El modelo no lo ve hasta después de entrenar. Evalúa el rendimiento **real**, como si lo usáramos en producción.

Entrenamiento de una CNN desde Cero - v2

Mejoras

```
# =====  
# PARÁMETROS Y PREPROCESAMIENTO  
# =====  
  
IMG_SIZE = (128, 128)  
BATCH_SIZE = 32  
  
datagen = ImageDataGenerator(  
    rescale=1./255,  
    rotation_range=40,  
    width_shift_range=0.2,  
    height_shift_range=0.2,  
    shear_range=0.2,  
    zoom_range=0.2,  
    horizontal_flip=True,  
    vertical_flip=True,  
    fill_mode='nearest'  
)
```


Entrenamiento de una CNN desde Cero - v2

```
train_generator = datagen.flow_from_directory(  
    "dataset_dividido/train" ,  
    target_size=IMG_SIZE,  
    batch_size=BATCH_SIZE,  
    class_mode= 'categorical'  
)  
  
val_generator = datagen.flow_from_directory(  
    "dataset_dividido/val" ,  
    target_size=IMG_SIZE,  
    batch_size=BATCH_SIZE,  
    class_mode= 'categorical'  
)  
  
test_generator = test_datagen.flow_from_directory(  
    "dataset_dividido/test" ,  
    target_size=IMG_SIZE,  
    batch_size=BATCH_SIZE,  
    class_mode= 'categorical' ,  
    shuffle=False  
)
```

Entrenamiento de una CNN desde Cero - v2

```
# =====  
# CLASS WEIGHTS  
# =====  
class_weights =  
compute_class_weight(  
    class_weight="balanced",  
  
    classes=np.unique(train_generator.c  
lasses),  
    y=train_generator.classes  
)  
class_weights =  
dict(enumerate(class_weights))
```

```

model = Sequential([
    Conv2D(32, (3, 3), activation='relu', kernel_regularizer=l2(0.001), input_shape=(128, 128, 3)),
    BatchNormalization(),
    MaxPooling2D(2, 2),

    Conv2D(64, (3, 3), activation='relu', kernel_regularizer=l2(0.001)),
    BatchNormalization(),
    MaxPooling2D(2, 2),

    Conv2D(128, (3, 3), activation='relu', kernel_regularizer=l2(0.001)),
    BatchNormalization(),
    MaxPooling2D(2, 2),

    GlobalAveragePooling2D(),
    Dense(256, activation='relu', kernel_regularizer=l2(0.001)),
    Dropout(0.5),
    Dense(num_classes, activation='softmax')
])

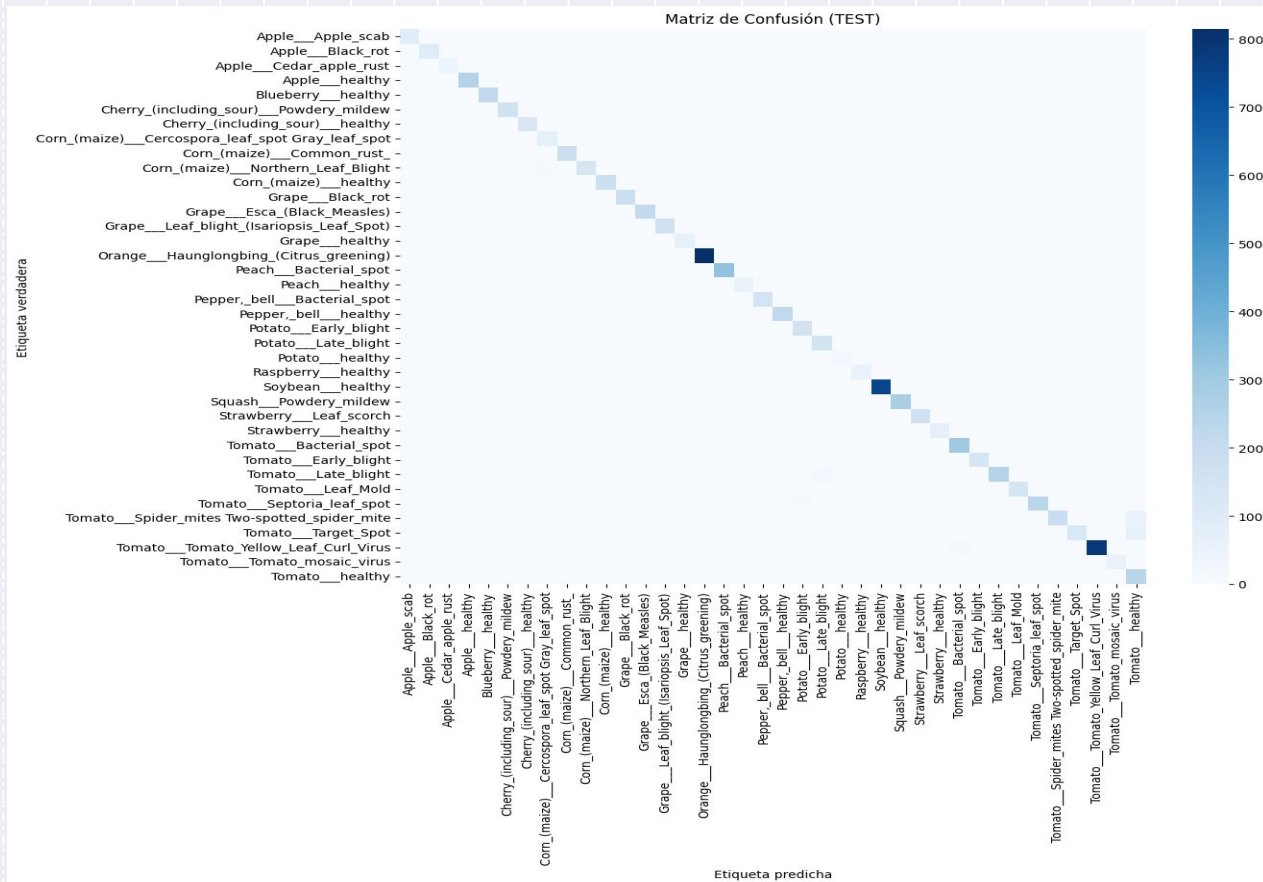
model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=[
        'accuracy',
        tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall'),
        tf.keras.metrics.AUC(name='auc')
    ])

```

Capa	Qué hace	Por qué se usa
Conv2D	Detecta patrones como bordes, manchas, texturas	Es la base del análisis visual
BatchNormalization	Normaliza la salida de la conv. para estabilizar el entrenamiento	Acelera y mejora el aprendizaje
MaxPooling2D	Reduce el tamaño de la imagen (submuestreo)	Hace que el modelo sea más rápido y generalice
GlobalAveragePooling 2D	Promedia cada mapa de activación	Reemplaza Flatten() con menos parámetros, evita overfitting
Dense(256)	Red neuronal completamente conectada	Toma las características extraídas y aprende combinaciones
Dropout(0.5)	Apaga aleatoriamente neuronas durante entrenamiento	Obliga a la red a no depender de neuronas específicas
Dense(num_classes) + softmax	Capa de salida que da la probabilidad de cada clase	Clasifica entre todas las clases

Entrenamiento de una CNN desde Cero - v2

```
# =====  
# CALLBACKS  
# =====  
  
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)  
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=2, min_lr=1e-6)  
checkpoint = ModelCheckpoint("mejor_modelo.h5", save_best_only=True, monitor="val_loss")  
  
# =====  
# ENTRENAMIENTO  
# =====  
  
history = model.fit(  
    train_generator,  
    epochs=50,  
    validation_data=val_generator,  
    callbacks=[early_stop, reduce_lr, checkpoint],  
    class_weight=class_weights  
)
```



Entrenamiento de una CNN desde Cero - v2

Resultados de las métricas

Tomato__healthy	0.64	0.99	0.78	240
accuracy			0.95	8179
macro avg	0.94	0.95	0.94	8179
weighted avg	0.96	0.95	0.95	8179

Transfer Learning

VGG16

- 16 Capas
- 138M params
- Año 2014

ResNet50

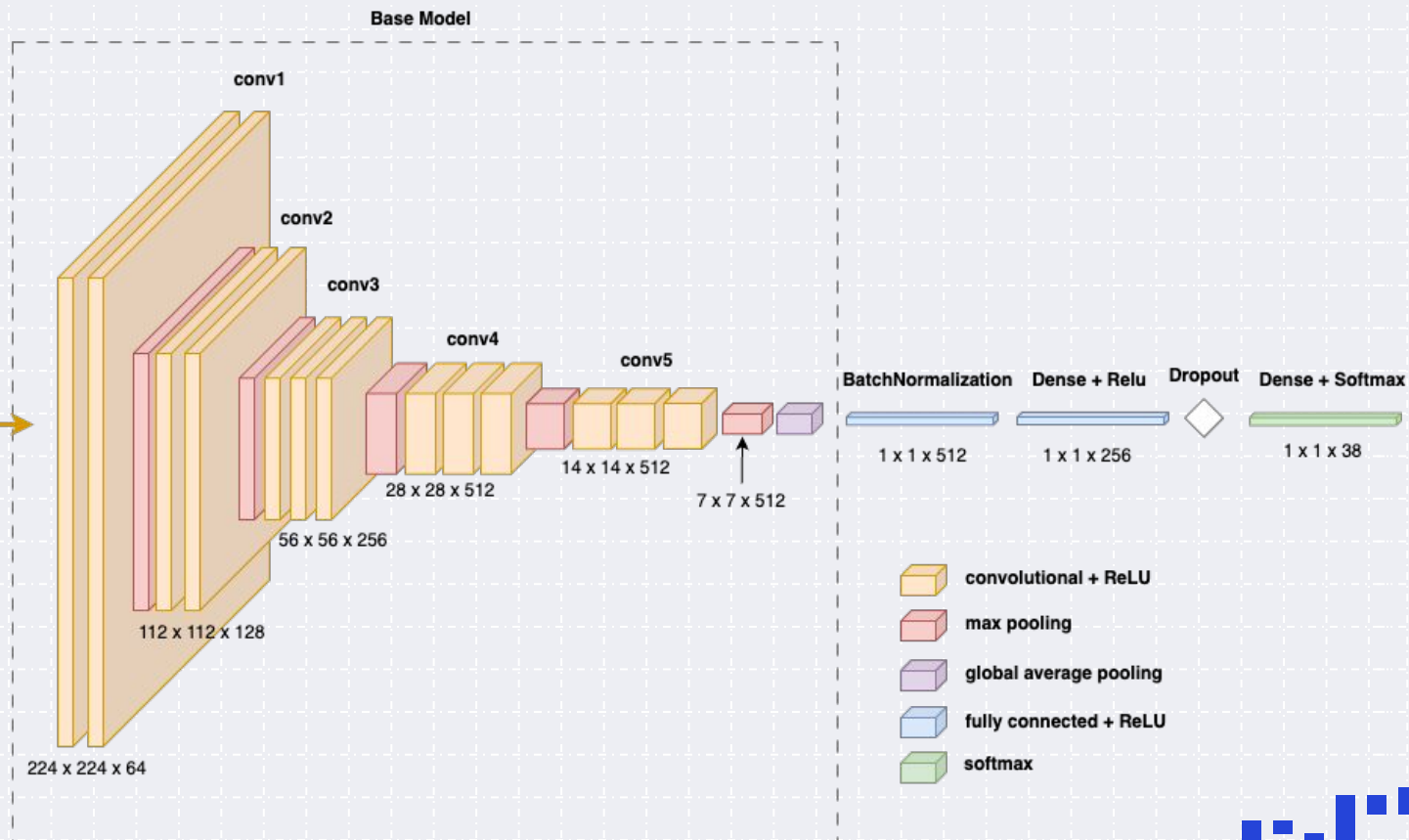
- 50 Capas
- 25.6M params
- Año 2015

EfficientNetB5

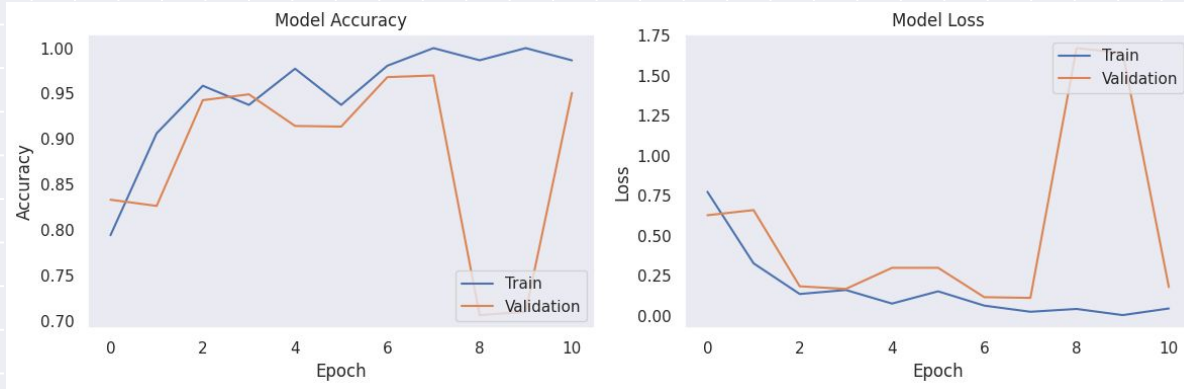
- 55 capas
- 30M params
- Año 2019



Transfer Learning



Transfer Learning- EfficientNetB5



Metrica/Modelo	Train Inicial	Tuneo Hyperparams
Accuracy	96.61%	97.31%
F1-Score	96.51%	97.32%

Clases con peor f1-score

- 5 clases: ~ 88%
- 2 clases: ~90%
- 3 clases: ~93%

Transfer Learning - ResNet50

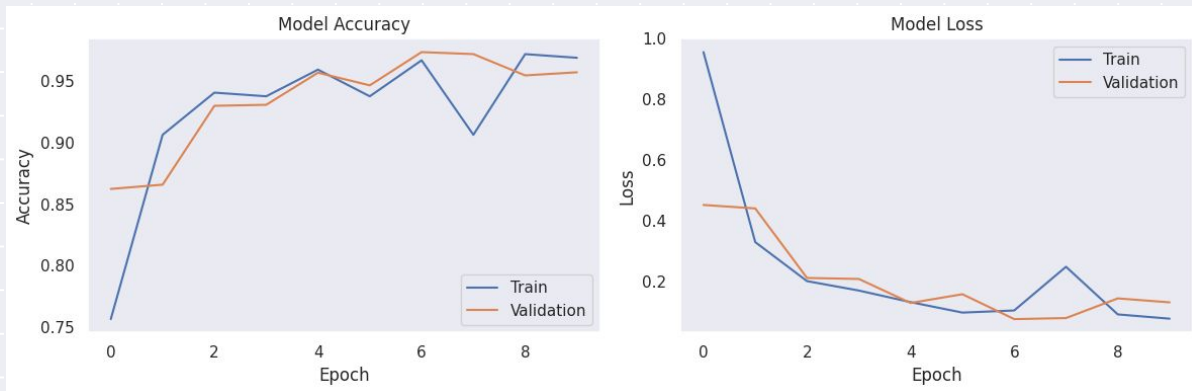


Metrica/Modelo	Train Inicial	Tuneo Hyperparams
Accuracy	98.44%	98.45%
F1-Score	98.43%	98.46%

Clases con peor f1-score

- Potato__healthy | Grape__Black_rot | Grape__Esca_(Black_Measles): ~ 90%
- Corn_(maize)__Northern_Leaf_Blight | Tomato__Early_blight: 95%
- Apple__Apple_scab, Peach__healthy | Tomato__Late_blight: 97%

Transfer Learning - VGG16



Metrica/Modelo	Train Inicial	Tuneo Hyperparams
Test Accuracy	97.37%	99.63%
F1-Score	97.34%	99.63%

Clases con peor f1-score

- Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot (513 imagenes): 96%
- Corn_(maize)___Northern_Leaf_Blight(985 imagenes): 97%

05

Conclusiones





Conclusiones

Modelo custom

- Versión inicial es una **implementación muy básica**. No cuenta con la capacidad para generalizar.
- V2 es una **versión más robusta**, incorpora mejoras en su arquitectura en el balanceo de clases e introduce capas de BatchNormalization. No son suficientes para utilizar el modelo en un ambiente productivo.

Transfer Learning

- Incorporan conocimiento adquirido de millones de imágenes. Requiere menor volumen de datos que el entrenamiento del modelo propio.
- **VGG16**, aunque sea la arquitectura más simple de los modelos base, es con diferencia la que mejor performa. 99.6% de accuracy, excelente balance de clases.
- Las otras dos arquitecturas resultan mejores que la propia, pero igualmente cuentan con un leve desbalance de clases. Podrían conseguirse mejores resultados si se entrena una “Cabeza” con más capas o se mejora el data augmentation y/o balance de clases.



FIN - MUCHAS GRACIAS