Deep Vision in Classification Tasks

April 29, 2024

1 Redes Neuronales y Deep Learning: "*Deep Vision in Classification Tasks*"

Elaborado por:

- **V. D. Betancourt**

1.1 Introducción

1.1.1 Objetivo

En este proyecto, se **evaluarán y compararán 2 estrategias** para la **Clasificación de Imágenes** empleando el `Vegetable Image Dataset`.

La propuesta de solución estará basada en **Redes Neuronales Convolucionales** (**CNNs**).

Pipeline

1. **Carga** del conjunto de datos
2. **Inspección** del conjunto de datos
3. **Acondicionamiento** del conjunto de datos
4. Desarrollo de la **arquitectura** de red neuronal y **entrenamiento** de la solución
5. **Monitorización** del proceso de **entrenamiento** para la toma de decisiones
6. **Evaluación** del modelo predictivo y planteamiento de la siguiente prueba experimental

1.1.2 Estrategia 1: Entrenar desde Cero (From Scratch)

La primera estrategia a comparar será una **Red Neuronal Profunda**.

Se expondrán la Arquitectura y los Hiperparámetros utilizados, y se aplicarán técnicas de **Regularización** para la mejora del rendimiento de la Red Neuronal tales como *weight regularization*, *dropout*, *batch normalization*, *data augmentation*, etc.

1.1.3 Estrategia 2: Usar Red Pre-entrenada

La segunda estrategia utilizará una **Red Pre-entrenada** con el dataset `ImageNet`, llevando a cabo tareas de ***Transfer Learning*** y ***Fine-Tuning*** para resolver la tarea de clasificación.

Se compararán al menos **2 tipos de Arquitecturas** entre las disponibles: VGGs, ResNet50, Xception, InceptionV3, InceptionResNetV2, MobileNetV2, DenseNet, ResNet. Y se seleccionará la que mayor precisión proporcione. (Información sobre las arquitecturas disponibles en https://keras.io/applications/).

De forma similar al procedimiento en la Estrategia 1, también se aplicarán técnicas de **Regularización** para la mejora del rendimiento de la Red Neuronal tales como *weight regularization*, *dropout*, *batch normalization*, *data augmentation*, etc.

1.2 Descripción

La siguiente información es extraída del repositorio de Kaggle donde se almacena el `Vegetable Image Dataset` original (https://www.kaggle.com/datasets/misrakahmed/vegetable-image-dataset).

1.2.1 Dataset: Vegetable Image Dataset

Clasificación y reconocimiento de vegetales

1.2.2 Contexto

- El experimento inicial se realiza con 15 tipos de hortalizas comunes en todo el mundo. Las verduras elegidas para el experimento son: judía, calabaza amarga, calabaza de botella, berenjena, brécol, col, pimiento, zanahoria, coliflor, pepino, papaya, patata, calabaza, rábano y tomate. Se utiliza un total de **21000** imágenes de **15 clases**, cada una de las cuales contiene 1400 imágenes de tamaño 224×224 y en formato `*.jpg`.
- El conjunto de datos se divide en un 70% para el entrenamiento, un 15% para la validación y un 15% para las pruebas.

1.2.3 Contenido

Este dataset contiene 3 carpetas:

- `train`: 15000 imágenes
- `test`: 3000 imágenes
- `validation`: 3000 imágenes

Cada una de las carpetas anteriores contiene subcarpetas para distintos vegetales en las que están presentes las imágenes de los vegetales/hortalizas respectivos.

De acuerdo con lo anterior, las 15 clases son las siguientes:

```
Y = ['Bean', 'Bitter Gourd', 'Bottle Gourd', 'Brinjal', 'Broccoli', 'Cabbage',
'Capsicum', 'Carrot', 'Cauliflower', 'Cucumber', 'Papaya', 'Potato', 'Pumpkin',
'Radish', 'Tomato']
```

1.3 Ejecución

No será posible ejecutar todos los modelos en una misma sesión de Colab.

- Lo anterior se debe a las limitaciones con la memoria RAM. Incluso, *tampoco es posible* ejecutar todos los modelos en una misma sesión usando una cuenta de Colab Pro.

Por lo tanto, debido a esta limitantes, se han incluido para cada modelo, las **instrucciones** para su correcta ejecución desde el entorno de **Google Colab**, que básicamente consisten (salvo que se especifique otra cosa) en ejecutar las Secciones:

1. Carga del Dataset.
2. Inspección del Dataset
3. Y en ocasiones el Acondicionamiento del Dataset.

Luego, se continuaría con la Sección específica del modelo que se desea ejecutar.

2 Carga del Dataset

En primer lugar, instalamos la API de Kaggle y procedemos a descargar el fichero vegetable-image-dataset.zip que incluye todas las imágenes que se utilizarán para entrenar a la red. Este fichero se descomprime en la carpeta **my_dataset** (en el entorno de la sesión actual de Colab).

[ ]:
```
%%capture
# Instalación de la última versión de la API de Kaggle en Colab
!pip install --upgrade --force-reinstall --no-deps kaggle
```

Al ejecutar la siguiente celda, se pedirá cargar el fichero **kaggle.json**, previamente obtenido del sitio **Kaggle/Usuario/Settings/API**.

[ ]:
```
# Seleccionar el API Token personal previamente descargado (fichero kaggle.json)
from google.colab import files
files.upload()
```

[ ]:
```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

En la siguiente celda de código, editar el **KAGGLE_USERNAME** y **KAGGLE_KEY**, tomando el **username** y **key** de la ejecución obtenida al cargar el fichero **kaggle.json** (arriba) y pegarlos en la siguiente línea de código correspondiente:

[ ]:
```
import os
os.environ['KAGGLE_USERNAME'] = "" # Poner aquí el username obtenido con el␣
 ↪kaggle.json
os.environ['KAGGLE_KEY'] = "" # Poner aquí la key obtenida con el kaggle.json
```

[ ]:
```
# Descargamos el dataset proveniente de Kaggle
!kaggle datasets download -d misrakahmed/vegetable-image-dataset
```

```
Downloading vegetable-image-dataset.zip to /content
100% 533M/534M [00:32<00:00, 20.7MB/s]
100% 534M/534M [00:32<00:00, 17.2MB/s]
```

[ ]:
```
# Creamos un directorio para descomprimir los datos
!mkdir my_dataset
```

```
[ ]: %%capture
     # Descomprimimos los datos y los dejamos listos para trabajar
     !unzip vegetable-image-dataset.zip -d my_dataset
```

3 Inspección del Dataset

En este apartado, se ejecuta una exploración inicial de los datos para confirmar que tienen la estructura adecuada.

3.1 Verificar Tamaño de las Imágenes

Warning!: Ejecutar la siguiente celda sólo la primera vez.

```
[ ]: #Verificamos si todas las imágenes tienen el mismo tamaño
     import os
     from PIL import Image

     def check_image_sizes_recursive(folder_path):
         for root, dirs, files in os.walk(folder_path):
             for file in files:
                 if file.lower().endswith('.jpg'):
                     file_path = os.path.join(root, file)
                     with Image.open(file_path) as img:
                         width, height = img.size
                         if width != 224 or height != 224:
                             print(f"File: {file_path} - Size: {width}x{height}")

     #Mostramos los datos y ubicación de las imágenes que no cumplen con el tamaño␣
      ↪de 224x224 píxeles
     folder_path = './drive/MyDrive/my_dataset'
     check_image_sizes_recursive(folder_path)
```

Se observa que algunas imágenes de la clase **Papaya** y otras de la clase **Bitter_Gourd** tienen unas un tamaño distinto a 224 x 224. Las imágenes en cuestión son las siguientes:

Clase **Bitter_Gourd**:

Carpeta **train**: 3 imágenes.

- 0526.jpg - Size: 224x205
- 0430.jpg - Size: 224x193
- 0609.jpg - Size: 224x200

Clase **Papaya**:

Carpeta **test**: 1 imagen. - 1246.jpg - Size: 224x207

Carpeta **train**: 3 imágenes

- 0741.jpg - Size: 224x210
- 0126.jpg - Size: 224x211

- 0176.jpg - Size: 224x198

Carpeta **validation**: 2 imágenes

- 1138.jpg - Size: 224x187
- 1150.jpg - Size: 224x223

Todas las imágenes tienen un width the 224px pero no todas tienen la misma altura. Al asignar el atributo **`padding=same`** a las primeras capas convolucionales, Tensorflow añadirá un padding de ceros (píxeles de color negro) a estas imágenes para que cumplan con la altura de 224px del resto de imágenes. Esto no debería suponer un inconveniente para la red debido a que son muy pocas imágenes a las que se hará esta padding con respecto al tamaño del dataset.

A continuación, se separan los datos en conjuntos de train, test y validation de acuerdo a como están estructurados en el fichero de Kaggle y se hace visualizan imágenes aleatorias para confirmar que se han cargado de manera correcta.

3.2 Guardar Imágenes en Tensores

Se crea una **función `load_dataset`** que toma 2 argumentos: - **`subfolder`**, que especifica el subdirectorio dentro de la ruta base donde se encuentran las imágenes específicas a cargar, y - **`batch_size`** con un valor por defecto de 32, que determina el número de imágenes a procesar en cada lote durante el entrenamiento del modelo.

Se crea un **diccionario `params`** que contiene varios parámetros necesarios para cargar el conjunto de datos de imágenes usando **`tf.keras.utils.image_dataset_from_directory`**, incluyendo: - **`directory`**: La ruta del directorio de donde cargar las imágenes. - **`seed`**: Un valor semilla para la aleatorización, asegurando reproducibilidad. - **`batch_size`**: El tamaño del lote para el conjunto de datos. - **`image_size`**: El tamaño al cual se redimensionarán las imágenes, especificado como (224, 224).

[ ]:
```python
# Creamos una función para guardar las imágenes en tensores
import tensorflow as tf
from pathlib import Path
path = '/content/my_dataset/Vegetable Images'

#Se fija el batch_size
def load_dataset(subfolder, batch_size=32):
  dataset_path = path + subfolder
  data_dir = Path(dataset_path)
  params = {
    'directory': data_dir,
    'seed': 0,
    'batch_size': batch_size,
    'image_size': (224,224)
    }
  dataset_params = {**params}
  dataset = tf.keras.utils.image_dataset_from_directory(**dataset_params)
  return dataset
```

3.3 Asignar Variables para las Imágenes: Train, Test, Validation

[ ]:
```
# Guardamos los diferentes conjuntos de imágenes en variables
print('>> Training Set:')
train_ds = load_dataset('/train')
print('\n')

print('>> Testing Set:')
test_ds = load_dataset('/test', None)
print('\n')

print('>> Validation Set:')
val_ds = load_dataset('/validation')
print('\n')
```

```
>> Training Set:
Found 15000 files belonging to 15 classes.


>> Testing Set:
Found 3000 files belonging to 15 classes.


>> Validation Set:
Found 3000 files belonging to 15 classes.
```

3.4 Definición de Clases

Se asigna la variable **class_names** para obtener un fácil acceso a los nombres de las categorías de imágenes que el modelo intentará aprender a clasificar.

[ ]:
```
# Definimos las clases
class_names = test_ds.class_names
class_names
```

[ ]:
```
['Bean',
 'Bitter_Gourd',
 'Bottle_Gourd',
 'Brinjal',
 'Broccoli',
 'Cabbage',
 'Capsicum',
 'Carrot',
 'Cauliflower',
 'Cucumber',
 'Papaya',
```

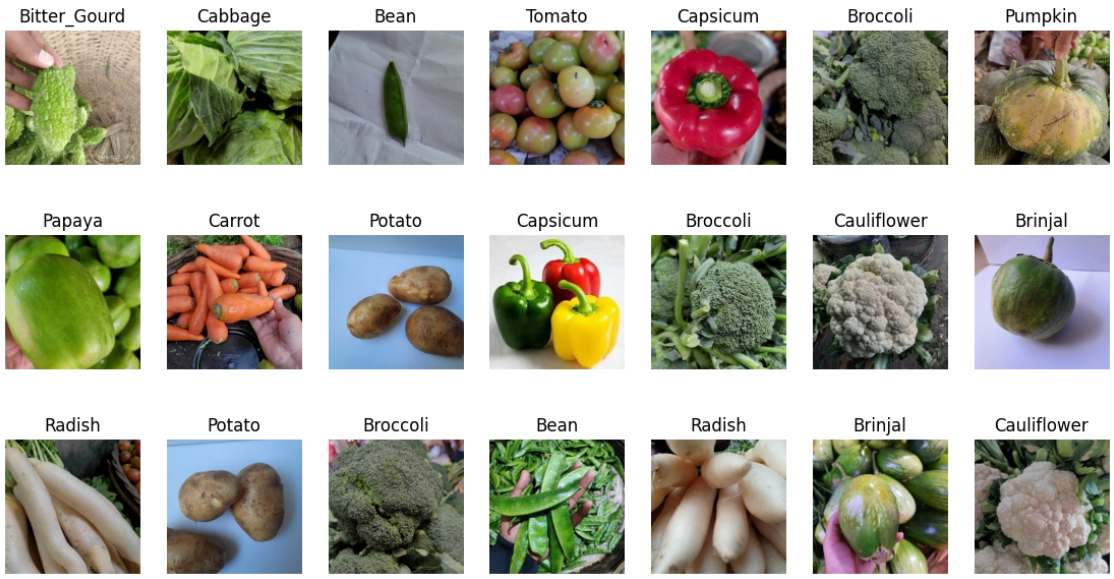
```
'Potato',
'Pumpkin',
'Radish',
'Tomato']
```

3.5 Visualización de las Imágenes: Train

Warning!: Ejecutar la visualización siguiente sólo la primera vez (para ahorrar memoria). No va a cambiar.

[ ]:
```
# Visualización de los datos de train
import matplotlib.pyplot as plt

plt.figure(figsize=(14,10))
for images, labels in train_ds.take(1):
  for i in range(21):
    ax = plt.subplot(4, 7, i+1)
    plt.imshow(images[i].numpy().astype('uint8'))
    plt.title(class_names[labels[i]])
    plt.axis('off')
```



4 Acondicionamiento del Dataset

Esta sección se ocupa de dos procesos de preprocesamiento fundamentales para el conjunto de datos de entrenamiento y validación: la **normalización** de las imágenes y la **conversión** de las etiquetas a formato one-hot encoding.

- **Normalización y One-Hot Encoding: Train y Validation**

La línea **train_ds.map(lambda x,y: (rescaling_layer(x), OHE(y)))** aplica la capa de re-escalado a cada imagen en el conjunto de entrenamiento (**train_ds**) para normalizar sus valores de píxeles. Además, aplica **one-hot** encoding a las etiquetas utilizando la capa **OHE**. El resultado es un nuevo conjunto de datos (**norm_train**) donde cada imagen está normalizada y cada etiqueta está en formato one-hot.

[ ]:
```
# Normalización y aplicación de One-hot Encoding
from tensorflow.keras.layers import Rescaling, CategoryEncoding

# Creación de una capa de re-escalado (normalización de datos)
rescaling_layer = Rescaling(1.0/255)

# Conversión a one-hot encoding
OHE = CategoryEncoding(num_tokens=15, output_mode="one_hot")

# Aplicamos normalización a cada imagen del dataset y OHE a las etiquetas
norm_train = train_ds.map(lambda x,y: (rescaling_layer(x), OHE(y)))
norm_val = val_ds.map(lambda x,y: (rescaling_layer(x), OHE(y)))
```

- **Separación de Imágenes y Etiquetas: Test**

Ahora, se prepara el conjunto de datos de prueba (**test_ds**) para la evaluación del modelo, separando las imágenes de sus etiquetas y aplicando normalización a las imágenes, pero dejando intactas las etiquetas.

Luego, la línea **test_ds.map(lambda x,y: (rescaling_layer(x), y))** aplica la capa de re-escalado a cada imagen en el conjunto de datos de prueba (**test_ds**) para normalizar sus valores de píxeles. Las etiquetas (**y**) se dejan intactas. Esto prepara las imágenes para la evaluación manteniendo el formato original de las etiquetas.

Finalmente, se convierten las listas **x_test** y **y_test** en arrays de NumPy.

[ ]:
```
# Separamos el conjunto de test entre imágenes y sus etiquetas
# Se utilizarán para evaluar la precisión del modelo
import numpy as np
x_test = []
y_test = []

norm_test = test_ds.map(lambda x,y: (rescaling_layer(x), y)) # dejamos intactas␣
 ↪las etiquetas (y)

for image, label in norm_test.take(len(norm_test)):
  x_test.append(image)
  y_test.append(label)

x_test = np.array(x_test)
y_test = np.array(y_test)

print("Test set shape:")
```

```
print(f'Images: {x_test.shape}')
print(f'Labels: {y_test.shape}')
```

```
Test set shape:
Images: (3000, 224, 224, 3)
Labels: (3000,)
```

5 Estrategia 1: Modelo From Scratch

En este apartado se procede a entrenar un modelo "**From Scratch**" definiendo la arquitectura desde cero, observando cómo se comporta en el entrenamiento y aplicando técnicas de regularización para mejorar su rendimiento.

Se han diseñado **4 variedades de Modelos From Scratch**:

1. Aplicando Max Pooling.
2. Aplicando Max Pooling, Batch Normalization y Dropout.
3. Aplicando Early Stopping.
4. Aplicando Data Augmentation.

Warning!: *No será posible ejecutar todos los modelos en una misma sesión de Colab.*

- Lo anterior se debe a las limitaciones con la memoria RAM. Incluso, *tampoco es posible* ejecutar todos los modelos en una misma sesión usando una cuenta de Colab Pro.

Por lo tanto, debido a esta limitantes, se han incluido para cada modelo, las **instrucciones** para su correcta ejecución desde el entorno de **Google Colab**, que básicamente consisten (salvo que se especifique otra cosa) en ejecutar las Secciones de Carga del Dataset, la Inspección del Dataset, y en ocasiones el Acondicionamiento del Dataset. Luego, se continuaría con la Sección específica del modelo que se desea ejecutar.

5.1 Modelo 1: From Scratch aplicando Max Pooling

Instrucciones para la Ejecución de esta Sección en Colab

- Primero, asegurarse de que ya se han ejecutado las siguientes Secciones del Notebook en la sesión actual:
  1. Carga del Dataset
  2. Inspección del Dataset
  3. Acondicionamiento
- Posteriormente, continuar en la presente **Sección Estrategia 1/Modelo 1/Definición del modelo**.

5.1.1 Definición del modelo

Se procede a definir el **BASE MODEL** de la siguiente manera:

1. **Entradas**: Se define la capa de entrada del modelo, especificando que se espera que cada imagen de entrada tenga un tamaño de **224x224 píxeles** con 3 canales de color (RGB), usando `Input(shape=(224, 224, 3))`.

2. **Primer Set de Capas Convolutivas**: El modelo comienza con un conjunto de capas convolutivas (`Conv2D`) con **32 filtros** de tamaño 3x3, seguidas por una capa de `MaxPooling2D` que reduce a la mitad las dimensiones espaciales de las características. Se aplica `Dropout` con una tasa de 0.25 para reducir el sobreajuste, descartando aleatoriamente el 25% de las conexiones entre las capas durante el entrenamiento.

3. **Segundo Set de Capas Convolutivas**: Continúa con otro bloque de capas convolutivas, esta vez con **64 filtros**. Similar al primer bloque, las capas `Conv2D` se aplican dos veces seguidas para extraer características más complejas, y se reduce la dimensión espacial mediante `MaxPooling2D`.

4. **Tercer Set de Capas Convolutivas**: Este bloque aumenta significativamente la profundidad a **256 filtros**. Nuevamente, se emplean dos capas `Conv2D` seguidas por `MaxPooling2D` para procesar y condensar aún más la información espacial de las imágenes.

5. **Top Model - Capas Densas**:

- `Flatten()`: Después de extraer y condensar las características con las capas convolutivas y de pooling, la salida se aplana para convertirse en un vector unidimensional.
- `Dense(512, activation="relu")`: Una capa densa (totalmente conectada) con **512 unidades** y la función de activación **ReLU** para introducir no linealidad y permitir que el modelo aprenda relaciones complejas.
- `predictions = Dense(15, activation="softmax")`: La última capa del modelo es otra capa densa con 15 unidades, una por cada clase objetivo. Se utiliza la función de activación `softmax` para obtener una distribución de probabilidad sobre las 15 categorías de salida, donde cada valor representa la probabilidad de que la imagen pertenezca a una de las clases.

[ ]:
```
# Importamos las librerías necesarias
from tensorflow.keras import backend as K
from tensorflow.keras.layers import Input, Conv2D, Activation, Flatten, Dense,␣
 ↪Dropout, BatchNormalization, MaxPooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import SGD, Adam
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt

########################################
###### Definimos la arquitectura ########
########################################
#BASE MODEL
# Definimos entradas
inputs = Input(shape=(224, 224, 3))

# Primer set de capas CONV
x1 = Conv2D(32, (3, 3), padding="same", activation="relu")(inputs)
x1 = Conv2D(32, (3, 3), padding="same", activation="relu")(x1)
x1 = MaxPooling2D(pool_size=(2, 2))(x1)
x1 = Dropout(0.25)(x1)
```

```
# Segundo set de capas CONV
x2 = Conv2D(64, (3, 3), padding="same", activation="relu")(x1) #(X)
x2 = Conv2D(64, (3, 3), padding="same", activation="relu")(x2) #(X)
x2 = MaxPooling2D(pool_size=(2, 2))(x2) #(X)


# Tercer set de capas CONV
x3 = Conv2D(256, (3, 3), padding="same", activation="relu")(x2) #(X)
x3 = Conv2D(256, (3, 3), padding="same", activation="relu")(x3) #(X)
x3 = MaxPooling2D(pool_size=(2, 2))(x3) #(X)


# TOP MODEL
# Primer (y único) set de capas FC => RELU
xfc = Flatten()(x3) #(X)
xfc = Dense(512, activation="relu")(xfc)
predictions = Dense(15, activation="softmax")(xfc) #(X)
```

5.1.2 Compilación del modelo

En esta sección, procede con la compilación y evaluación del Modelo 1 From Scratch, como sigue:

1. **Creación del Modelo**:

- **`Model(inputs=inputs, outputs=predictions)`** crea una instancia del modelo utilizando la API funcional de Keras. Se especifican las **`inputs`** y **`outputs`** del modelo, que fueron definidas en la sección anterior.

2. **Compilación**:

- **`model_cnn.compile(...)`**: Este paso es crucial para preparar el modelo para el entrenamiento.
- **`loss="categorical_crossentropy"`**: Se utiliza la entropía cruzada categórica como la función de pérdida, apropiada para problemas de clasificación multiclase donde las etiquetas están en formato one-hot encoding.
- **`optimizer=Adam(...)`**: Se elige Adam como el optimizador, con un **`learning_rate`** de 0.001 y valores predeterminados para los parámetros **`beta_1`**, **`beta_2`**, y **`epsilon`**. Adam es popular por su eficiencia en el ajuste de tasas de aprendizaje y su buena performance en problemas de visión por computadora.
- **`metrics=["accuracy"]`**: La precisión (accuracy) se utiliza como métrica para evaluar el rendimiento del modelo. Indica el porcentaje de etiquetas que el modelo predijo correctamente.

3. **Entrenamiento**:

- **`model_cnn.fit(...)`** entrena el modelo utilizando el conjunto de datos de entrenamiento normalizado (**`norm_train`**) y validación (**`norm_val`**).

- **validation_data=norm_val** especifica el conjunto de datos de validación sobre el cual evaluar el rendimiento del modelo después de cada época.
- **batch_size=128** define el tamaño del lote para el entrenamiento. Un tamaño de lote de 128 significa que el modelo actualizará sus pesos después de ver 128 muestras.
- **epochs=n_epochs** determina el número de veces que el algoritmo de entrenamiento trabajará a través de todo el conjunto de datos de entrenamiento. Se ha definido **n_epochs=20**, indicando que el conjunto de datos se pasará a través del modelo 20 veces.
- **verbose=1** muestra barras de progreso y otra información durante el entrenamiento, proporcionando visibilidad sobre el proceso de aprendizaje del modelo.

[ ]:
```
# Unimos las entradas y el modelo mediante la función Model con parámetros␣
 ↪inputs y outputs
model_cnn = Model(inputs=inputs, outputs=predictions) #(X)

# Compilar el modelo
print("[INFO]: Compilando el modelo...")
model_cnn.compile(loss="categorical_crossentropy",
                  optimizer=Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999,␣
 ↪epsilon=1e-08),
                  metrics=["accuracy"]) #(X)

n_epochs=20

# Entrenamiento de la red
print("[INFO]: Entrenando la red...")
#
H = model_cnn.fit(norm_train, validation_data=norm_val, batch_size=128,␣
 ↪epochs=n_epochs, verbose=1) #(X)
```

```
[INFO]: Compilando el modelo…
[INFO]: Entrenando la red…
Epoch 1/20
469/469 [==============================] - 128s 243ms/step - loss: 2.0135 -
accuracy: 0.3291 - val_loss: 1.2653 - val_accuracy: 0.5750
Epoch 2/20
469/469 [==============================] - 109s 232ms/step - loss: 0.7556 -
accuracy: 0.7553 - val_loss: 0.6341 - val_accuracy: 0.7883
Epoch 3/20
469/469 [==============================] - 109s 232ms/step - loss: 0.3685 -
accuracy: 0.8835 - val_loss: 0.4884 - val_accuracy: 0.8540
Epoch 4/20
469/469 [==============================] - 109s 232ms/step - loss: 0.2142 -
accuracy: 0.9329 - val_loss: 0.3413 - val_accuracy: 0.8913
Epoch 5/20
469/469 [==============================] - 107s 228ms/step - loss: 0.1145 -
accuracy: 0.9646 - val_loss: 0.3873 - val_accuracy: 0.8950
```

```
Epoch 6/20
469/469 [==============================] - 108s 231ms/step - loss: 0.0987 -
accuracy: 0.9699 - val_loss: 0.3601 - val_accuracy: 0.9143
Epoch 7/20
469/469 [==============================] - 108s 231ms/step - loss: 0.0775 -
accuracy: 0.9760 - val_loss: 0.3409 - val_accuracy: 0.9193
Epoch 8/20
469/469 [==============================] - 107s 228ms/step - loss: 0.0508 -
accuracy: 0.9852 - val_loss: 0.3722 - val_accuracy: 0.9163
Epoch 9/20
469/469 [==============================] - 107s 228ms/step - loss: 0.0515 -
accuracy: 0.9851 - val_loss: 0.3684 - val_accuracy: 0.9200
Epoch 10/20
469/469 [==============================] - 107s 228ms/step - loss: 0.0696 -
accuracy: 0.9803 - val_loss: 0.4473 - val_accuracy: 0.8820
Epoch 11/20
469/469 [==============================] - 107s 228ms/step - loss: 0.0310 -
accuracy: 0.9897 - val_loss: 0.5208 - val_accuracy: 0.9060
Epoch 12/20
469/469 [==============================] - 108s 229ms/step - loss: 0.0543 -
accuracy: 0.9846 - val_loss: 0.5342 - val_accuracy: 0.8807
Epoch 13/20
469/469 [==============================] - 108s 230ms/step - loss: 0.0261 -
accuracy: 0.9920 - val_loss: 0.5536 - val_accuracy: 0.9023
Epoch 14/20
469/469 [==============================] - 108s 229ms/step - loss: 0.0262 -
accuracy: 0.9915 - val_loss: 0.4210 - val_accuracy: 0.9183
Epoch 15/20
469/469 [==============================] - 107s 228ms/step - loss: 0.0671 -
accuracy: 0.9801 - val_loss: 0.3736 - val_accuracy: 0.9297
Epoch 16/20
469/469 [==============================] - 107s 227ms/step - loss: 0.0264 -
accuracy: 0.9913 - val_loss: 0.4332 - val_accuracy: 0.9060
Epoch 17/20
469/469 [==============================] - 107s 227ms/step - loss: 0.0236 -
accuracy: 0.9931 - val_loss: 0.5930 - val_accuracy: 0.8793
Epoch 18/20
469/469 [==============================] - 108s 229ms/step - loss: 0.0370 -
accuracy: 0.9891 - val_loss: 0.3482 - val_accuracy: 0.9250
Epoch 19/20
469/469 [==============================] - 107s 227ms/step - loss: 0.0182 -
accuracy: 0.9951 - val_loss: 0.4533 - val_accuracy: 0.9197
Epoch 20/20
469/469 [==============================] - 108s 229ms/step - loss: 0.0384 -
accuracy: 0.9900 - val_loss: 0.4066 - val_accuracy: 0.9200
```

5.1.3 Evaluación del modelo

Como era de esperar, el modelo presenta una cantidad importante de overfitting debido a que no se ha aplicado ninguna técnica de regularización más allá del max pooling. Se observa que en la última época el modelo no solo no mejora si no que parece tener tendencia a diverger.

[ ]:
```
from sklearn.metrics import classification_report

# Muestro gráfica de accuracy y losses
plt.style.use("ggplot")
plt.figure()

plt.plot(np.arange(0, 20), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, 20), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, 20), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, 20), H.history["val_accuracy"], label="val_acc")

plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()

plt.show()
```

```
[ ]: # Evaluación del modelo
print("[INFO]: Evaluando el modelo de red neuronal...")
# Efectuamos la predicción (empleamos el mismo valor de batch_size que en␣
 ↪training)
predictions = model_cnn.predict(x_test, batch_size=128) #(X)
# Sacamos el report para test
print(classification_report(y_test, predictions.argmax(axis=1),␣
 ↪target_names=class_names)) #(X)
```

```
[INFO]: Evaluando el modelo de red neuronal…
24/24 [==============================] - 18s 351ms/step
              precision    recall  f1-score   support

        Bean       0.91      0.93      0.92       200
Bitter_Gourd       0.95      0.90      0.92       200
Bottle_Gourd       0.90      0.97      0.94       200
     Brinjal       0.97      0.83      0.89       200
    Broccoli       0.92      0.90      0.91       200
     Cabbage       0.75      0.96      0.84       200
    Capsicum       0.94      0.97      0.96       200
      Carrot       0.94      0.99      0.96       200
 Cauliflower       0.88      0.85      0.87       200
    Cucumber       0.94      0.95      0.95       200
      Papaya       0.96      0.96      0.96       200
      Potato       0.98      0.94      0.96       200
     Pumpkin       0.88      0.86      0.87       200
      Radish       0.97      0.94      0.96       200
      Tomato       0.89      0.77      0.83       200

    accuracy                           0.92      3000
   macro avg       0.92      0.92      0.92      3000
weighted avg       0.92      0.92      0.92      3000
```

5.2 Modelo 2: From Scratch aplicando Max Pooling, Batch Normalization y Dropout

Instrucciones para la Ejecución de esta Sección en Colab

- Primero, ejecutar (en una nueva sesión) las siguientes Secciones del actual Notebook:
  1. Carga del Dataset
  2. Inspección del Dataset
  3. Acondicionamiento

  (excepto aquellas celdas para las cuales se especifique que no es necesario ejecutarlas).

- Posteriormente, continuar en la presente **Sección Estrategia 1/Modelo 2/Definición del modelo**.

5.2.1 Definición del modelo

Se procede a definir el **BASE MODEL** como sigue:

1. **Entradas**: Se especifica la forma de las imágenes de entrada, con un tamaño de **224x224 píxeles** y **3 canales** de color (RGB), usando `Input(shape=(224, 224, 3))`.
2. **Primer Set de Capas Convolutivas**: Este bloque inicia con una capa convolucional (`Conv2D`) con **32 filtros**, seguido por una capa de normalización por lotes (`BatchNormalization`), otra capa `Conv2D` con los mismos parámetros, y otra `BatchNormalization`. El uso de la normalización por lotes ayuda a estabilizar y acelerar el entrenamiento al normalizar las salidas de las capas convolutivas. Esto se complementa con una capa de `MaxPooling2D` para reducir la dimensionalidad y una capa de `Dropout` para mitigar el sobreajuste.
3. **Segundo Bloque de Capas Convolutivas**: Similar al primer bloque, pero con un aumento en la profundidad de los filtros a **64**. La estructura se repite: capas `Conv2D`, seguidas de `BatchNormalization`, y luego `Dropout` después del `MaxPooling2D`. Este diseño es común en las CNNs y ayuda a extraer características más complejas a medida que la información avanza a través de la red.
4. **Tercer Bloque de Capas Convolutivas**: Este bloque aumenta aún más la profundidad a **256 filtros**, siguiendo el mismo patrón de capas `Conv2D`, `BatchNormalization`, y `Dropout` después de `MaxPooling2D`. Este enfoque secuencial y profundo permite al modelo aprender una jerarquía de características espaciales de alto nivel.
5. **Capas Densas (Top Model)**:

- `Flatten()`: Convierte las características tridimensionales en un vector unidimensional, preparándolas para el procesamiento por capas densas.
- Se introduce una capa densa (`Dense`) de 512 unidades con activación `ReLU`, seguida por `BatchNormalization` y una capa de `Dropout` con una tasa más alta (0.5), buscando reducir el sobreajuste en esta parte más conectada y propensa al sobreajuste del modelo.
- La última capa es una densa con 15 unidades y activación `softmax`, que clasifica las imágenes en una de las 15 categorías.

[ ]:
```
# Importamos las librerías necesarias
from tensorflow.keras import backend as K
from tensorflow.keras.layers import Input, Conv2D, Activation, Flatten, Dense,␣
↪Dropout, BatchNormalization, MaxPooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import SGD, Adam
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt

########################################
###### Definimos la arquitectura ########
```

```
########################################
#BASE MODEL
# Definimos entradas
inputs = Input(shape=(224, 224, 3))

# Primer set de capas CONV => RELU => CONV => RELU => POOL
x1 = Conv2D(32, (3, 3), padding="same", activation="relu")(inputs)
x1 = BatchNormalization()(x1)
x1 = Conv2D(32, (3, 3), padding="same", activation="relu")(x1)
x1 = BatchNormalization()(x1)
x1 = MaxPooling2D(pool_size=(2, 2))(x1)
x1 = Dropout(0.25)(x1)

# Segundo set de capas CONV => RELU => CONV => RELU => POOL
x2 = Conv2D(64, (3, 3), padding="same", activation="relu")(x1) #(X)
x2 = BatchNormalization()(x2) #(X)
x2 = Conv2D(64, (3, 3), padding="same", activation="relu")(x2) #(X)
x2 = BatchNormalization()(x2) #(X)
x2 = MaxPooling2D(pool_size=(2, 2))(x2) #(X)
x2 = Dropout(0.25)(x2) #(X)

# Tercer set de capas CONV => RELU => CONV => RELU => POOL
x3 = Conv2D(256, (3, 3), padding="same", activation="relu")(x2) #(X)
x3 = BatchNormalization()(x3) #(X)
x3 = Conv2D(256, (3, 3), padding="same", activation="relu")(x3) #(X)
x3 = BatchNormalization()(x3) #(X)
x3 = MaxPooling2D(pool_size=(2, 2))(x3) #(X)
x3 = Dropout(0.25)(x3) #(X)

# TOP MODEL
# Primer (y único) set de capas FC => RELU
xfc = Flatten()(x3) #(X)
xfc = Dense(512, activation="relu")(xfc) #(X)
xfc = BatchNormalization()(xfc) #(X)
xfc = Dropout(0.5)(xfc) #(X)
# Clasificador softmax
predictions = Dense(15, activation="softmax")(xfc) #(X)
```

5.2.2 Compilación del modelo

En esta sección, procede con la compilación y evaluación del Modelo 2 From Scratch, como sigue:

1. **Creación del Modelo**:

- **`Model(inputs=inputs, outputs=predictions)`** crea una instancia del modelo utilizando la API funcional de Keras. Se especifican las **`inputs`** y **`outputs`** del modelo, que fueron definidas en la sección anterior.

2. **Compilación**:

- **`model_cnn.compile(...)`**: Prepara el modelo para el entrenamiento especificando la función de pérdida, el optimizador y las métricas para evaluar.
- **`loss="categorical_crossentropy"`**: Utiliza la entropía cruzada categórica como función de pérdida, adecuada para clasificación multiclase cuando las etiquetas están en formato one-hot.
- **`optimizer=Adam(...)`**: Selecciona Adam como el optimizador, con un learning rate de 0.001. Adam es eficaz para una amplia gama de problemas gracias a cómo ajusta dinámicamente el learning rate durante el entrenamiento.
- **`metrics=["accuracy"]`**: Establece la precisión como la métrica de evaluación del modelo, proporcionando una idea clara de su rendimiento en términos de porcentaje de predicciones correctas.

3. **Entrenamiento**:

- **`n_epochs=20`**: Define el número de épocas para el entrenamiento, indicando cuántas veces el algoritmo trabajará a través del conjunto de datos completo.
- **`H = model_cnn.fit(...)`**: Comienza el entrenamiento del modelo usando los conjuntos de datos normalizados de entrenamiento y validación. norm_train: El conjunto de datos de entrenamiento normalizado, donde cada imagen ha sido escalada a valores entre 0 y 1.
- **`validation_data=norm_val`**: El conjunto de datos de validación que se utiliza para evaluar el modelo después de cada época y ajustar los hiperparámetros sin sobreajustar al conjunto de entrenamiento.
- **`batch_size=128`**: Determina el tamaño del lote. Es la cantidad de muestras que el modelo trabajará antes de actualizar los pesos internos.
- **`epochs=n_epochs`**: Pasa el número de épocas definido anteriormente.
- **`verbose=1`**: Activa la salida detallada durante el entrenamiento, mostrando barras de progreso y métricas después de cada época.

[ ]:
```
# Unimos las entradas y el modelo mediante la función Model con parámetros␣
↪inputs y ouputs
model_cnn = Model(inputs=inputs, outputs=predictions) #(X)
```

[ ]:
```
# Resumen del Modelo
model_cnn.summary()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 conv2d (Conv2D)             (None, 224, 224, 32)      896

 batch_normalization (Batch  (None, 224, 224, 32)      128
 Normalization)
```

```
conv2d_1 (Conv2D)           (None, 224, 224, 32)      9248

batch_normalization_1 (Bat  (None, 224, 224, 32)      128
chNormalization)

max_pooling2d (MaxPooling2  (None, 112, 112, 32)      0
D)

dropout (Dropout)           (None, 112, 112, 32)      0

conv2d_2 (Conv2D)           (None, 112, 112, 64)      18496

batch_normalization_2 (Bat  (None, 112, 112, 64)      256
chNormalization)

conv2d_3 (Conv2D)           (None, 112, 112, 64)      36928

batch_normalization_3 (Bat  (None, 112, 112, 64)      256
chNormalization)

max_pooling2d_1 (MaxPoolin  (None, 56, 56, 64)        0
g2D)

dropout_1 (Dropout)         (None, 56, 56, 64)        0

conv2d_4 (Conv2D)           (None, 56, 56, 256)       147712

batch_normalization_4 (Bat  (None, 56, 56, 256)       1024
chNormalization)

conv2d_5 (Conv2D)           (None, 56, 56, 256)       590080

batch_normalization_5 (Bat  (None, 56, 56, 256)       1024
chNormalization)

max_pooling2d_2 (MaxPoolin  (None, 28, 28, 256)       0
g2D)

dropout_2 (Dropout)         (None, 28, 28, 256)       0

flatten (Flatten)           (None, 200704)            0

dense (Dense)               (None, 512)               102760960

batch_normalization_6 (Bat  (None, 512)               2048
chNormalization)

dropout_3 (Dropout)         (None, 512)               0
```

```
 dense_1 (Dense)             (None, 15)                7695

=================================================================
Total params: 103576879 (395.11 MB)
Trainable params: 103574447 (395.11 MB)
Non-trainable params: 2432 (9.50 KB)
_________________________________________________________________
```

- **Compilación**

[ ]:
```
# Compilación del modelo
print("[INFO]: Compilando el modelo...")
model_cnn.compile(loss="categorical_crossentropy",
#                  loss='sparse_categorical_crossentropy',
                  optimizer=Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999,␣
 ↪epsilon=1e-08),
                  metrics=["accuracy"]) #(X)
```

```
[INFO]: Compilando el modelo…
```

Nótese que la versión de la función de pérdida **loss='sparse_categorical_crossentropy'** se utiliza cuando las etiquetas están en un formato de enteros en lugar de one-hot encoded. Pero en este caso, las etiquetas se esperan en formato one-hot, por eso **loss='categorical_crossentropy'**.

- **Entrenamiento**

[ ]:
```
n_epochs=20

# Entrenamiento de la red
print("[INFO]: Entrenando la red...")

#
H = model_cnn.fit(norm_train, validation_data=norm_val, batch_size=128,␣
 ↪epochs=n_epochs, verbose=1) #(X)
```

```
[INFO]: Entrenando la red…
Epoch 1/20
469/469 [==============================] - 180s 344ms/step - loss: 0.9143 -
accuracy: 0.7309 - val_loss: 1.2400 - val_accuracy: 0.6513
Epoch 2/20
469/469 [==============================] - 153s 326ms/step - loss: 0.2554 -
accuracy: 0.9185 - val_loss: 0.7399 - val_accuracy: 0.7950
Epoch 3/20
469/469 [==============================] - 154s 328ms/step - loss: 0.1522 -
accuracy: 0.9514 - val_loss: 0.4091 - val_accuracy: 0.8810
Epoch 4/20
469/469 [==============================] - 153s 327ms/step - loss: 0.1155 -
accuracy: 0.9646 - val_loss: 2.1997 - val_accuracy: 0.6393
Epoch 5/20
```

```
469/469 [==============================] - 153s 326ms/step - loss: 0.0837 -
accuracy: 0.9735 - val_loss: 0.1226 - val_accuracy: 0.9643
Epoch 6/20
469/469 [==============================] - 154s 329ms/step - loss: 0.0642 -
accuracy: 0.9794 - val_loss: 0.2438 - val_accuracy: 0.9413
Epoch 7/20
469/469 [==============================] - 153s 325ms/step - loss: 0.0777 -
accuracy: 0.9764 - val_loss: 0.1391 - val_accuracy: 0.9563
Epoch 8/20
469/469 [==============================] - 153s 326ms/step - loss: 0.0646 -
accuracy: 0.9783 - val_loss: 0.1818 - val_accuracy: 0.9493
Epoch 9/20
469/469 [==============================] - 153s 326ms/step - loss: 0.0459 -
accuracy: 0.9854 - val_loss: 0.7278 - val_accuracy: 0.8203
Epoch 10/20
469/469 [==============================] - 152s 325ms/step - loss: 0.0427 -
accuracy: 0.9859 - val_loss: 0.8244 - val_accuracy: 0.8467
Epoch 11/20
469/469 [==============================] - 153s 326ms/step - loss: 0.0363 -
accuracy: 0.9883 - val_loss: 0.3272 - val_accuracy: 0.9190
Epoch 12/20
469/469 [==============================] - 153s 326ms/step - loss: 0.0538 -
accuracy: 0.9823 - val_loss: 0.9355 - val_accuracy: 0.8290
Epoch 13/20
469/469 [==============================] - 153s 326ms/step - loss: 0.0502 -
accuracy: 0.9843 - val_loss: 0.1121 - val_accuracy: 0.9723
Epoch 14/20
469/469 [==============================] - 153s 325ms/step - loss: 0.0360 -
accuracy: 0.9882 - val_loss: 0.1583 - val_accuracy: 0.9587
Epoch 15/20
469/469 [==============================] - 152s 324ms/step - loss: 0.0245 -
accuracy: 0.9923 - val_loss: 0.1531 - val_accuracy: 0.9610
Epoch 16/20
469/469 [==============================] - 153s 325ms/step - loss: 0.0330 -
accuracy: 0.9893 - val_loss: 0.1136 - val_accuracy: 0.9747
Epoch 17/20
469/469 [==============================] - 152s 324ms/step - loss: 0.0210 -
accuracy: 0.9935 - val_loss: 0.1329 - val_accuracy: 0.9673
Epoch 18/20
469/469 [==============================] - 153s 325ms/step - loss: 0.0355 -
accuracy: 0.9883 - val_loss: 0.8938 - val_accuracy: 0.8473
Epoch 19/20
469/469 [==============================] - 152s 325ms/step - loss: 0.0304 -
accuracy: 0.9907 - val_loss: 0.0749 - val_accuracy: 0.9823
Epoch 20/20
469/469 [==============================] - 153s 325ms/step - loss: 0.0173 -
accuracy: 0.9941 - val_loss: 0.0689 - val_accuracy: 0.9850
```

5.2.3 Evaluación del modelo

Puede notarse una mejora considerable del overfitting visto en el modelo anterior, donde en la última epoch la diferencia entre validation loss y training loss es menor a 0.1. Es interesante destacar el pico validation loss en la epoch, que puede ser causa de diversos factores, como sensibilidad del modelo durante la inicialización o una variabilidad marcada en el batch de validación para esa epoch.

En este modelo se ha alcanzado un F1-score perfecto para la detección de capsicum (pimiento). En el modelo anterior el valor de este vegetal también fue el más alto con un valor de 0.97.

[ ]:
```
# Muestro gráfica de accuracy y losses
plt.style.use("ggplot")
plt.figure()

plt.plot(np.arange(0, 20), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, 20), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, 20), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, 20), H.history["val_accuracy"], label="val_acc")

plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()

plt.show()
```

[ ]:
```
from sklearn.metrics import classification_report

# Evaluación del modelo
print("[INFO]: Evaluando el modelo de red neuronal...")

# Efectuamos la predicción (empleamos el mismo valor de batch_size que en␣
 ↪training)
predictions = model_cnn.predict(x_test, batch_size=128) #(X)

# Sacamos el report para test
print(classification_report(y_test, predictions.argmax(axis=1),␣
 ↪target_names=class_names)) #(X)
```

```
[INFO]: Evaluando el modelo de red neuronal…
24/24 [==============================] - 21s 418ms/step
              precision    recall  f1-score   support

        Bean       0.97      0.97      0.97       200
Bitter_Gourd       0.99      0.99      0.99       200
Bottle_Gourd       0.99      0.99      0.99       200
```

```
      Brinjal       0.99      0.96      0.98       200
    Broccoli        0.97      0.99      0.98       200
     Cabbage        0.99      0.97      0.98       200
    Capsicum        1.00      1.00      1.00       200
      Carrot        1.00      0.98      0.99       200
 Cauliflower        0.98      0.97      0.98       200
    Cucumber        0.99      0.98      0.98       200
      Papaya        0.96      0.99      0.98       200
      Potato        0.99      0.99      0.99       200
     Pumpkin        0.96      0.98      0.97       200
      Radish        0.99      1.00      1.00       200
      Tomato        0.96      0.95      0.95       200

    accuracy                            0.98      3000
   macro avg        0.98      0.98      0.98      3000
weighted avg        0.98      0.98      0.98      3000
```

5.3 Modelo 3: From Scratch aplicando Early Stopping

Instrucciones para la Ejecución de esta Sección en Colab

- Primero, ejecutar (en una nueva sesión) las siguientes Secciones del actual Notebook:
  1. Carga del Dataset
  2. Inspección del Dataset
  3. Acondicionamiento

  (excepto aquellas celdas para las cuales se especifique que no es necesario ejecutarlas).
- Posteriormente, continuar en la presente **Sección Estrategia 1/Modelo 3/Definición del modelo**.

5.3.1 Definición del modelo

Para el Modelo 3, se parte de la estructura del Modelo 2.

[ ]:
```
# Importamos las librerías necesarias
from tensorflow.keras import backend as K
from tensorflow.keras.layers import Input, Conv2D, Activation, Flatten, Dense,␣
↪Dropout, BatchNormalization, MaxPooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import SGD, Adam
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt

########################################
###### Definimos la arquitectura ########
########################################
#BASE MODEL
```

```
# Definimos entradas
inputs = Input(shape=(224, 224, 3))

# Primer set de capas CONV => RELU => CONV => RELU => POOL
x1 = Conv2D(32, (3, 3), padding="same", activation="relu")(inputs)
x1 = BatchNormalization()(x1)
x1 = Conv2D(32, (3, 3), padding="same", activation="relu")(x1)
x1 = BatchNormalization()(x1)
x1 = MaxPooling2D(pool_size=(2, 2))(x1)
x1 = Dropout(0.25)(x1)

# Segundo set de capas CONV => RELU => CONV => RELU => POOL
x2 = Conv2D(64, (3, 3), padding="same", activation="relu")(x1) #(X)
x2 = BatchNormalization()(x2) #(X)
x2 = Conv2D(64, (3, 3), padding="same", activation="relu")(x2) #(X)
x2 = BatchNormalization()(x2) #(X)
x2 = MaxPooling2D(pool_size=(2, 2))(x2) #(X)
x2 = Dropout(0.25)(x2) #(X)

# Tercer set de capas CONV => RELU => CONV => RELU => POOL
x3 = Conv2D(256, (3, 3), padding="same", activation="relu")(x2) #(X)
x3 = BatchNormalization()(x3) #(X)
x3 = Conv2D(256, (3, 3), padding="same", activation="relu")(x3) #(X)
x3 = BatchNormalization()(x3) #(X)
x3 = MaxPooling2D(pool_size=(2, 2))(x3) #(X)
x3 = Dropout(0.25)(x3) #(X)

# TOP MODEL
# Primer (y único) set de capas FC => RELU
xfc = Flatten()(x3) #(X)
xfc = Dense(512, activation="relu")(xfc) #(X)
xfc = BatchNormalization()(xfc) #(X)
xfc = Dropout(0.5)(xfc) #(X)
# Clasificador softmax
predictions = Dense(15, activation="softmax")(xfc) #(X)
```

5.3.2 Compilación del modelo

[ ]:
```
# Unimos las entradas y el modelo mediante la función Model con parámetros␣
↪inputs y ouputs
model_cnn = Model(inputs=inputs, outputs=predictions) #(X)
```

[ ]:
```
# Resumen del Modelo
model_cnn.summary()
```

```
Model: "model"
_________________________________________________________________
```

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0

conv2d (Conv2D)              (None, 224, 224, 32)      896

batch_normalization (Batch   (None, 224, 224, 32)      128
Normalization)

conv2d_1 (Conv2D)            (None, 224, 224, 32)      9248

batch_normalization_1 (Bat   (None, 224, 224, 32)      128
chNormalization)

max_pooling2d (MaxPooling2   (None, 112, 112, 32)      0
D)

dropout (Dropout)            (None, 112, 112, 32)      0

conv2d_2 (Conv2D)            (None, 112, 112, 64)      18496

batch_normalization_2 (Bat   (None, 112, 112, 64)      256
chNormalization)

conv2d_3 (Conv2D)            (None, 112, 112, 64)      36928

batch_normalization_3 (Bat   (None, 112, 112, 64)      256
chNormalization)

max_pooling2d_1 (MaxPoolin   (None, 56, 56, 64)        0
g2D)

dropout_1 (Dropout)          (None, 56, 56, 64)        0

conv2d_4 (Conv2D)            (None, 56, 56, 256)       147712

batch_normalization_4 (Bat   (None, 56, 56, 256)       1024
chNormalization)

conv2d_5 (Conv2D)            (None, 56, 56, 256)       590080

batch_normalization_5 (Bat   (None, 56, 56, 256)       1024
chNormalization)

max_pooling2d_2 (MaxPoolin   (None, 28, 28, 256)       0
g2D)

dropout_2 (Dropout)          (None, 28, 28, 256)       0
```

```
 flatten (Flatten)           (None, 200704)            0

 dense (Dense)               (None, 512)               102760960

 batch_normalization_6 (Bat  (None, 512)               2048
 chNormalization)

 dropout_3 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 15)                7695

=================================================================
Total params: 103576879 (395.11 MB)
Trainable params: 103574447 (395.11 MB)
Non-trainable params: 2432 (9.50 KB)
_________________________________________________________________
```

- **Compilación**

```
[ ]: # Compilar el modelo
print("[INFO]: Compilando el modelo...")
model_cnn.compile(loss="categorical_crossentropy",
#                 loss='sparse_categorical_crossentropy',
                  optimizer=Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999,␣
 ↪epsilon=1e-08),
                  metrics=["accuracy"]) #(X)
```

```
[INFO]: Compilando el modelo…
```

Nótese que la versión de la función de pérdida **`loss='sparse_categorical_crossentropy'`** se utiliza cuando las etiquetas están en un formato de enteros en lugar de one-hot encoded. Pero en este caso, las etiquetas se esperan en formato one-hot, por eso **`loss='categorical_crossentropy'`**.

- **Definición de Callbacks para Early Stopping**

Se definen los siguientes **Callbacks**:

1. **EarlyStopping**:

- Monitorea el valor de val_loss durante el entrenamiento. Si **`val_loss`** no mejora (disminuye en este caso, ya que el modo es min) después de un número específico de épocas (**`patience=3`**), el entrenamiento se detiene prematuramente. Esto es útil para evitar el sobreajuste al detener el entrenamiento cuando el modelo deja de mejorar en el conjunto de validación.

2. **ModelCheckpoint**:

- Guarda los pesos del modelo en el filepath especificado (**`/content/`**) cada vez que el **`val_accuracy`** mejora. Esto significa que solo se guardan los pesos del modelo en su mejor estado de acuerdo con la precisión de validación, lo cual es útil para recuperar el modelo en su punto de mejor rendimiento sin necesidad de reentrenar.

- **save_weights_only=True** indica que solo se guardarán los pesos del modelo, no toda la arquitectura del modelo.

3. **TensorBoard**:

- Proporciona una visualización poderosa del proceso de entrenamiento. **log_dir='./logs'** especifica dónde guardar los logs que TensorBoard usará para generar las visualizaciones. Esto puede incluir métricas de rendimiento, gráficos de la función de pérdida y precisión a lo largo del tiempo, y mucho más. TensorBoard es una herramienta invaluable para el análisis y la interpretación del comportamiento del modelo durante el entrenamiento.

[ ]:
```
# Callbacks
my_callbacks = [
    tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                     patience=3, # Valor de patience con mejor␣
 ↪resultado entre 2-5
                                     mode='min'),
    tf.keras.callbacks.ModelCheckpoint(filepath='/content/',
                                       monitor='val_accuracy',
                                       save_weights_only=True,
                                       mode='max'),
    tf.keras.callbacks.TensorBoard(log_dir='./logs')
]
```

- **Entrenamiento**

Se aumenta el número de épocas a **n_epochs=50**, lo que permite más iteraciones del proceso de entrenamiento para ajustar los pesos del modelo, con la seguridad de que el callback de **EarlyStopping** puede detener el entrenamiento si el modelo deja de mejorar.

Al invocar **model_cnn.fit**, se incluye el parámetro **callbacks=my_callbacks**, asegurando que los callbacks definidos sean ejecutados durante el entrenamiento.

[ ]:
```
n_epochs=50

# Entrenamiento de la red
print("[INFO]: Entrenando la red...")
#
H = model_cnn.fit(norm_train, validation_data=norm_val, batch_size=128,␣
 ↪epochs=n_epochs, verbose=1, callbacks=my_callbacks) #(X)
```

```
[INFO]: Entrenando la red…
Epoch 1/50
469/469 [==============================] - 182s 350ms/step - loss: 0.9475 -
accuracy: 0.7207 - val_loss: 2.1302 - val_accuracy: 0.5080
Epoch 2/50
469/469 [==============================] - 169s 360ms/step - loss: 0.2715 -
accuracy: 0.9115 - val_loss: 0.4091 - val_accuracy: 0.8643
Epoch 3/50
469/469 [==============================] - 163s 347ms/step - loss: 0.1552 -
```

```
accuracy: 0.9529 - val_loss: 0.2398 - val_accuracy: 0.9233
Epoch 4/50
469/469 [==============================] - 167s 355ms/step - loss: 0.1096 -
accuracy: 0.9646 - val_loss: 0.6776 - val_accuracy: 0.8323
Epoch 5/50
469/469 [==============================] - 167s 356ms/step - loss: 0.0884 -
accuracy: 0.9739 - val_loss: 0.8624 - val_accuracy: 0.7823
Epoch 6/50
469/469 [==============================] - 168s 358ms/step - loss: 0.0772 -
accuracy: 0.9735 - val_loss: 0.2498 - val_accuracy: 0.9280
```

5.3.3 Evaluación del modelo

El modelo se detiene en la **Epoch 6** (lo cual podría variar en otra ejecución). Aunque no hay mejora en el overfitting con respecto al modelo anterior, el tiempo de entrenamiento se ha reducido a la mitad.

[ ]:
```python
# Muestro gráfica de accuracy y losses
plt.style.use("ggplot")
plt.figure()

# Usar la longitud de H.history["loss"] para el rango de x
epochs_range = np.arange(len(H.history["loss"])) # Por si cambian las Epochs␣
 ↪del entrenamiento

plt.plot(epochs_range, H.history["loss"], label="train_loss")
plt.plot(epochs_range, H.history["val_loss"], label="val_loss")
plt.plot(epochs_range, H.history["accuracy"], label="train_acc")
plt.plot(epochs_range, H.history["val_accuracy"], label="val_acc")

plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()

plt.show()
```

[ ]:
```
# Evaluación del modelo
print("[INFO]: Evaluando el modelo de red neuronal...")

# Efectuamos la predicción (empleamos el mismo valor de batch_size que en␣
 ↪training)
predictions = model_cnn.predict(x_test, batch_size=128) #(X)

# Sacamos el report para test
print(classification_report(y_test, predictions.argmax(axis=1),␣
 ↪target_names=class_names)) #(X)
```

```
[INFO]: Evaluando el modelo de red neuronal…
24/24 [==============================] - 18s 360ms/step
              precision    recall  f1-score   support

        Bean       0.99      0.94      0.97       200
Bitter_Gourd       0.94      0.97      0.96       200
Bottle_Gourd       0.97      0.96      0.97       200
     Brinjal       0.95      0.95      0.95       200
    Broccoli       1.00      0.86      0.92       200
```

Cabbage	0.92	0.71	0.81	200
Capsicum	0.93	1.00	0.96	200
Carrot	0.95	0.95	0.95	200
Cauliflower	0.76	0.97	0.86	200
Cucumber	0.94	0.95	0.95	200
Papaya	0.94	0.94	0.94	200
Potato	0.83	0.99	0.90	200
Pumpkin	0.91	0.97	0.94	200
Radish	0.99	0.92	0.95	200
Tomato	0.96	0.78	0.86	200
accuracy			0.93	3000
macro avg	0.93	0.93	0.93	3000
weighted avg	0.93	0.93	0.93	3000

En las pruebas anteriores del modelo se obtuvo resultados considerablemente buenos con una presencia de overfitting poco prominente, sin embargo, se aplicará la técnica de Data augmentation con el fin de reducirlo un poco más y realizar una comparación más profunda.

5.4 Modelo 4: From Scratch aplicando Data Augmentation

Instrucciones para la Ejecución de esta Sección en Colab

- Primero, ejecutar (en una nueva sesión) las siguientes Secciones del actual Notebook:

  1. Carga del Dataset
  2. Inspección del Dataset
  3. Acondicionamiento

  (excepto aquellas celdas para las cuales se especifique que no es necesario ejecutarlas).

- Posteriormente, continuar en la presente **Sección Estrategia 1/Modelo 4**.

5.4.1 Generación de contenedor DataGenerator para el aumento automático de muestras

La **Aumentación de Datos (Data Augmentation)** mediante el uso de `ImageDataGenerator`, permite incrementar la diversidad de los datos de entrenamiento aplicando transformaciones aleatorias pero realistas a las imágenes. He aquí algunas transformaciones específicas que se aplicarán:

- `rotation_range=12`: Esto permite que las imágenes se roten aleatoriamente durante el entrenamiento en un rango de -12 a +12 grados. La rotación ayuda al modelo a aprender a reconocer objetos en diferentes orientaciones.
- `width_shift_range=0.2` y `height_shift_range=0.2`: Estas opciones desplazan las imágenes aleatoriamente en el eje horizontal y vertical, respectivamente, hasta un 20% del ancho o alto de la imagen. Estos desplazamientos pueden forzar al modelo a aprender a reconocer objetos que no están perfectamente centrados.
- `horizontal_flip=True`: Al activar el volteamiento horizontal, las imágenes pueden ser espejadas a lo largo del eje vertical. Esto es particularmente útil para datos donde la orientación

horizontal no afecta la clasificación.

- **zoom_range=0.2**: Aplica un zoom aleatorio a las imágenes dentro de un rango de +/- 20%. El zoom ayuda al modelo a aprender a reconocer objetos a diferentes escalas.

[ ]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    rotation_range=12, # grados de rotacion aleatoria
    width_shift_range=0.2, # fraccion del total (1) para mover la imagen
    height_shift_range=0.2, # fraccion del total (1) para mover la imagen
    horizontal_flip=True, # girar las imagenes horizontalmente (eje vertical)
    # shear_range=0, # deslizamiento
    zoom_range=0.2, # rango de zoom
    # fill_mode='nearest', # como rellenar posibles nuevos pixeles
    # channel_shift_range=0.2 # cambios aleatorios en los canales de la imagen
)
```

- **Variables para Test**

[ ]:
```
# Carga del conjunto de test
# Los datos de train y validation se cargan durante el entrenamiento del modelo
import numpy as np

test_aug = load_dataset('/test', None)
x_test_aug = []
y_test_aug = []

# norm_test = test_aug.map(lambda x,y: (rescaling_layer(x), y))
for image, label in test_aug.take(len(test_aug)):
  x_test_aug.append(image)
  y_test_aug.append(label)

x_test_aug = np.array(x_test_aug)
y_test_aug = np.array(y_test_aug)

# Guardamos las clases en una variable
class_names_aug = test_aug.class_names
```

```
Found 3000 files belonging to 15 classes.
```

5.4.2 Inspección de muestras generadas sintéticamente

Utilizando el **ImageDataGenerator** previamente definido (**datagen**), se generan versiones transformadas de la imagen seleccionada. Para esto, se utiliza el método **flow**, que recibe la imagen expandiendo sus dimensiones (para ajustarse al formato esperado por **datagen**) y produce lotes de imágenes transformadas.

Se itera sobre los lotes generados por **datagen.flow**, visualizando las primeras 4 imágenes transformadas en una cuadrícula 2x2. Cada imagen es convertida de un tensor a una imagen utilizable

mediante **image.array_to_img**, y luego se muestra con **imshow**.

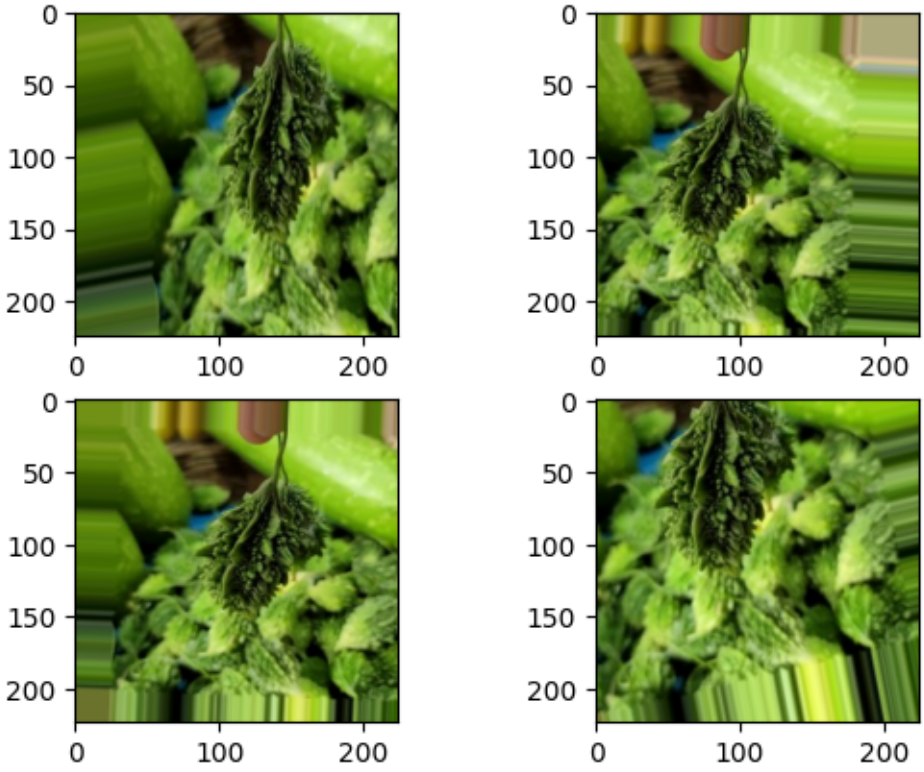
Este proceso demuestra visualmente el efecto de las transformaciones especificadas en **ImageDataGenerator**, como rotaciones, desplazamientos, zoom y volteo horizontal, proporcionando una vista previa de cómo estas técnicas de aumentación pueden enriquecer el conjunto de datos de entrenamiento al introducir variabilidad.

[ ]:
```
from tensorflow.keras.preprocessing import image
from tensorflow.keras.backend import expand_dims
import matplotlib.pyplot as plt
%matplotlib inline

# Tomamos uno de los ejemplos de imágenes de train
for images, labels in train_ds.take(1):
  sample = np.random.randint(len(images))
  sample_train_x = images[sample]
  sample_class = class_names_aug[labels[sample]]
plt.imshow(sample_train_x.numpy().astype('uint8'))
plt.title(sample_class)
plt.axis('off')
plt.show()

# Visualizamos las transformaciones hechas por la técnica de Data Augmentation
fig, axes = plt.subplots(2,2)
i = 0
for batch in datagen.flow(expand_dims(sample_train_x, axis=0),batch_size=1):
    #plt.figure(i)
    axes[i//2,i%2].imshow(image.array_to_img(batch[0]))
    i += 1
    if i == 4:
        break
plt.show()
```


Bitter_Gourd


0
50
100
150
200
0
100
200

5.4.3 Definición del modelo

Se usa el mismo Modelo 2, definido anteriormente.

1. **Entrada**

- Se define la capa de entrada con el tamaño esperado para las imágenes (224x224 píxeles con 3 canales de color).

2. **Primer Set de Capas**

- Comienza con dos capas convolutivas (`Conv2D`) con **32 filtros** cada una, utilizando activación ReLU y padding `same` para mantener el tamaño de la imagen. La inclusión de capas de normalización por lotes después de cada capa convolutiva ayuda a estabilizar y acelerar el entrenamiento.
- Se añade una capa de agrupamiento (`MaxPooling2D`) para reducir la dimensión espacial y una capa de abandono (`Dropout`) con una tasa del 25% para reducir el riesgo de sobreajuste.

3. **Segundo Set de Capas**

- Similar al primero, pero incrementando el número de filtros a **64**. Este set también incluye capas de normalización por lotes y una capa de abandono tras el agrupamiento, siguiendo el mismo patrón de diseño para profundizar en la extracción de características.

4. **Tercer Set de Capas**

- Aumenta significativamente la profundidad a **256 filtros**, manteniendo la estructura de capas convolutivas, de normalización por lotes, y de abandono. Este bloque está diseñado para extraer características aún más complejas y abstractas de las imágenes.

5. **Top Model**

- Las características extraídas son aplanadas (`Flatten`) para ser procesadas por capas densas. La primera capa densa tiene **512 unidades** y utiliza activación `ReLU`, seguida de normalización por lotes y una capa de abandono con una tasa más alta (50%) para mitigar aún más el sobreajuste.
- La última capa es una densa con **15 unidades** y activación `softmax`, destinada a clasificar las imágenes entre las 15 categorías posibles, basándose en las características aprendidas por el modelo.

[ ]:
```
# Importamos las librerías necesarias
from tensorflow.keras import backend as K
from tensorflow.keras.layers import Input, Conv2D, Activation, Flatten, Dense,␣
↪Dropout, BatchNormalization, MaxPooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import SGD, Adam
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt
```

```
########################################
###### Definimos la arquitectura ########
########################################
#BASE MODEL
# Definimos entradas
inputs = Input(shape=(224, 224, 3))

# Primer set de capas CONV => RELU => CONV => RELU => POOL
x1 = Conv2D(32, (3, 3), padding="same", activation="relu")(inputs)
x1 = BatchNormalization()(x1)
x1 = Conv2D(32, (3, 3), padding="same", activation="relu")(x1)
x1 = BatchNormalization()(x1)
x1 = MaxPooling2D(pool_size=(2, 2))(x1)
x1 = Dropout(0.25)(x1)

# Segundo set de capas CONV => RELU => CONV => RELU => POOL
x2 = Conv2D(64, (3, 3), padding="same", activation="relu")(x1) #(X)
x2 = BatchNormalization()(x2) #(X)
x2 = Conv2D(64, (3, 3), padding="same", activation="relu")(x2) #(X)
x2 = BatchNormalization()(x2) #(X)
x2 = MaxPooling2D(pool_size=(2, 2))(x2) #(X)
x2 = Dropout(0.25)(x2) #(X)

# Tercer set de capas CONV => RELU => CONV => RELU => POOL
x3 = Conv2D(256, (3, 3), padding="same", activation="relu")(x2) #(X)
x3 = BatchNormalization()(x3) #(X)
x3 = Conv2D(256, (3, 3), padding="same", activation="relu")(x3) #(X)
x3 = BatchNormalization()(x3) #(X)
x3 = MaxPooling2D(pool_size=(2, 2))(x3) #(X)
x3 = Dropout(0.25)(x3) #(X)

# TOP MODEL
# Primer (y único) set de capas FC => RELU
xfc = Flatten()(x3) #(X)
xfc = Dense(512, activation="relu")(xfc) #(X)
xfc = BatchNormalization()(xfc) #(X)
xfc = Dropout(0.5)(xfc) #(X)
# Clasificador softmax
predictions = Dense(15, activation="softmax")(xfc) #(X)
```

5.4.4 Compilación del modelo

[ ]:
```
# Unimos las entradas y el modelo mediante la función Model con parámetros␣
  ↪inputs y ouputs
model_aug = Model(inputs=inputs, outputs=predictions) #(X)
```

[ ]:
```
model_aug.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 conv2d (Conv2D)             (None, 224, 224, 32)      896

 batch_normalization (Batch  (None, 224, 224, 32)      128
 Normalization)

 conv2d_1 (Conv2D)           (None, 224, 224, 32)      9248

 batch_normalization_1 (Bat  (None, 224, 224, 32)      128
 chNormalization)

 max_pooling2d (MaxPooling2  (None, 112, 112, 32)      0
 D)

 dropout (Dropout)           (None, 112, 112, 32)      0

 conv2d_2 (Conv2D)           (None, 112, 112, 64)      18496

 batch_normalization_2 (Bat  (None, 112, 112, 64)      256
 chNormalization)

 conv2d_3 (Conv2D)           (None, 112, 112, 64)      36928

 batch_normalization_3 (Bat  (None, 112, 112, 64)      256
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 56, 56, 64)        0
 g2D)

 dropout_1 (Dropout)         (None, 56, 56, 64)        0

 conv2d_4 (Conv2D)           (None, 56, 56, 256)       147712

 batch_normalization_4 (Bat  (None, 56, 56, 256)       1024
 chNormalization)

 conv2d_5 (Conv2D)           (None, 56, 56, 256)       590080

 batch_normalization_5 (Bat  (None, 56, 56, 256)       1024
 chNormalization)
```

```
max_pooling2d_2 (MaxPoolin  (None, 28, 28, 256)       0
g2D)

dropout_2 (Dropout)         (None, 28, 28, 256)       0

flatten (Flatten)           (None, 200704)            0

dense (Dense)               (None, 512)               102760960

batch_normalization_6 (Bat  (None, 512)               2048
chNormalization)

dropout_3 (Dropout)         (None, 512)               0

dense_1 (Dense)             (None, 15)                7695

=================================================================
Total params: 103576879 (395.11 MB)
Trainable params: 103574447 (395.11 MB)
Non-trainable params: 2432 (9.50 KB)
_________________________________________________________________
```

- **Compilación**

```
[ ]: # Compilación del modelo
print("[INFO]: Compilando el modelo...")
model_aug.compile(loss="categorical_crossentropy",
#                 loss='sparse_categorical_crossentropy',
                  optimizer=Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999,␣
 ↪epsilon=1e-08),
                  metrics=["accuracy"]) #(X)
```

```
[INFO]: Compilando el modelo…
```

Nótese que la versión de la función de pérdida **loss='sparse_categorical_crossentropy'** se utiliza cuando las etiquetas están en un formato de enteros en lugar de one-hot encoded. Pero en este caso, las etiquetas se esperan en formato one-hot, por eso **loss='categorical_crossentropy'**.

- **Callbacks**

Se definen los siguientes **Callbacks**:

1. **ModelCheckpoint**:

- Guarda el modelo en el directorio **content/data_aug** cada vez que el valor de **val_loss** mejora, es decir, disminuye, lo que indica un mejor rendimiento del modelo en el conjunto de validación.
- **save_best_only=True** asegura que solo se guarde el modelo con el mejor rendimiento hasta el momento, basado en val_loss.

- **monitor='val_loss'** y **mode='min'** indican que el objetivo es minimizar la pérdida de validación.

2. **TensorBoard**:

- Proporciona herramientas de visualización para el entrenamiento del modelo, almacenando los logs en **'./logs'**. TensorBoard puede mostrar métricas de rendimiento, visualizar la arquitectura del modelo, y más, facilitando el análisis y la optimización del entrenamiento

[ ]:
```
# Callbacks
callbacks_aug = [
    tf.keras.callbacks.ModelCheckpoint(filepath='/content/data_aug',
                                       monitor='val_loss',
                                       save_best_only=True,
                                       mode='min'),
    tf.keras.callbacks.TensorBoard(log_dir='./logs')
]
```

- **Entrenamiento**

Algunas consideraciones importantes son:

1. **model_aug.fit(...)**: El modelo se entrena utilizando **datagen.flow_from_directory** para ambos, el conjunto de entrenamiento (**train**) y de validación (**validation**). Esta función carga las imágenes directamente desde el directorio especificado, aplicando las transformaciones definidas en **ImageDataGenerator** (**datagen**) sobre la marcha.
2. **target_size=(224, 224)**: Todas las imágenes se redimensionarán a 224x224 píxeles.
3. **color_mode='rgb'**: Las imágenes se cargarán en formato RGB.
4. **classes=class_names_aug**: Especifica los nombres de las clases, asegurando que las etiquetas se asignen de manera consistente.
5. **class_mode='categorical'**: Indica que las etiquetas se codificarán en formato categórico (**one-hot encoding**).
6. **batch_size=32**: Define el tamaño del lote para el entrenamiento y la validación.
7. **shuffle=True** y **seed=0**: Aseguran que las imágenes se barajen de manera reproducible.
8. **epochs=num_epoch**: Se especifican 20 épocas para el entrenamiento.

[ ]:
```
num_epoch=20

# Entrenamiento de la red
print("[INFO]: Entrenando la red...")
#
H_aug = model_aug.fit(datagen.flow_from_directory(path + '/train',
                                                  target_size=(224, 224),
                                                  color_mode='rgb',
                                                  classes=class_names_aug,
                                                  class_mode='categorical',
```

```
                                                batch_size=32,
                                                shuffle=True,
                                                seed=0),
                        # steps_per_epoch = len(norm_train)/ 128,
                        validation_data=datagen.flow_from_directory(path + '/
↪validation',
                                                target_size=(224, 224),
                                                color_mode='rgb',
                                                classes=class_names_aug,
                                                class_mode='categorical',
                                                batch_size=32,
                                                shuffle=True,
                                                seed=0),
                        epochs=num_epoch, verbose=1, callbacks=callbacks_aug)
```

```
[INFO]: Entrenando la red…
Found 15000 images belonging to 15 classes.
Found 3000 images belonging to 15 classes.
Epoch 1/20
469/469 [==============================] - 323s 651ms/step - loss: 1.4024 -
accuracy: 0.5881 - val_loss: 5.5661 - val_accuracy: 0.2697
Epoch 2/20
469/469 [==============================] - 296s 630ms/step - loss: 0.6094 -
accuracy: 0.8059 - val_loss: 0.9590 - val_accuracy: 0.7223
Epoch 3/20
469/469 [==============================] - 286s 609ms/step - loss: 0.4011 -
accuracy: 0.8707 - val_loss: 0.4981 - val_accuracy: 0.8530
Epoch 4/20
469/469 [==============================] - 290s 618ms/step - loss: 0.3050 -
accuracy: 0.9023 - val_loss: 0.2312 - val_accuracy: 0.9287
Epoch 5/20
469/469 [==============================] - 286s 610ms/step - loss: 0.2518 -
accuracy: 0.9213 - val_loss: 0.2704 - val_accuracy: 0.9167
Epoch 6/20
469/469 [==============================] - 286s 609ms/step - loss: 0.2257 -
accuracy: 0.9305 - val_loss: 0.2241 - val_accuracy: 0.9283
Epoch 7/20
469/469 [==============================] - 274s 585ms/step - loss: 0.1934 -
accuracy: 0.9399 - val_loss: 0.3378 - val_accuracy: 0.8910
Epoch 8/20
469/469 [==============================] - 274s 585ms/step - loss: 0.1591 -
accuracy: 0.9494 - val_loss: 0.4027 - val_accuracy: 0.8867
Epoch 9/20
469/469 [==============================] - 328s 700ms/step - loss: 0.1431 -
accuracy: 0.9567 - val_loss: 0.1447 - val_accuracy: 0.9587
Epoch 10/20
469/469 [==============================] - 297s 633ms/step - loss: 0.1557 -
```

```
accuracy: 0.9509 - val_loss: 0.1048 - val_accuracy: 0.9733
Epoch 11/20
469/469 [==============================] - 275s 586ms/step - loss: 0.1466 -
accuracy: 0.9521 - val_loss: 1.2557 - val_accuracy: 0.7520
Epoch 12/20
469/469 [==============================] - 277s 591ms/step - loss: 0.1165 -
accuracy: 0.9642 - val_loss: 0.4390 - val_accuracy: 0.8977
Epoch 13/20
469/469 [==============================] - 278s 592ms/step - loss: 0.1282 -
accuracy: 0.9591 - val_loss: 0.6652 - val_accuracy: 0.8567
Epoch 14/20
469/469 [==============================] - 277s 590ms/step - loss: 0.1281 -
accuracy: 0.9597 - val_loss: 0.5640 - val_accuracy: 0.8593
Epoch 15/20
469/469 [==============================] - 272s 580ms/step - loss: 0.1116 -
accuracy: 0.9651 - val_loss: 0.2530 - val_accuracy: 0.9337
Epoch 16/20
469/469 [==============================] - 271s 578ms/step - loss: 0.0844 -
accuracy: 0.9737 - val_loss: 0.6834 - val_accuracy: 0.8407
Epoch 17/20
469/469 [==============================] - 271s 578ms/step - loss: 0.0811 -
accuracy: 0.9741 - val_loss: 0.2398 - val_accuracy: 0.9330
Epoch 18/20
469/469 [==============================] - 273s 582ms/step - loss: 0.0939 -
accuracy: 0.9709 - val_loss: 0.7671 - val_accuracy: 0.8577
Epoch 19/20
469/469 [==============================] - 272s 580ms/step - loss: 0.0707 -
accuracy: 0.9773 - val_loss: 0.2786 - val_accuracy: 0.9220
Epoch 20/20
469/469 [==============================] - 271s 577ms/step - loss: 0.0824 -
accuracy: 0.9743 - val_loss: 0.1596 - val_accuracy: 0.9563
```

5.4.5 Evaluación del modelo

En este modelo se han obtenido valores de f1-score perfectos para la clasificación de patata y calabaza amarga y casi perfecta para el pimiento, la zanahoria y el pepino. Sin embargo, la clasificación de otros vegetales como la calabaza o el coliflor se ha visto impactada negativamente. Esto puede deberse a que las imágenes generadas durante el proceso de Data Augmentation han tenido poca representación de estos vegetales.

Destacamos neuvamente el pico de validation loss en la época ocho. El hecho de que el pico se repita en la misma época puede deberse a que el bache de validación (que no cambia entre modelos) contiene imágenes especialmente complicadas de detectar para la red.

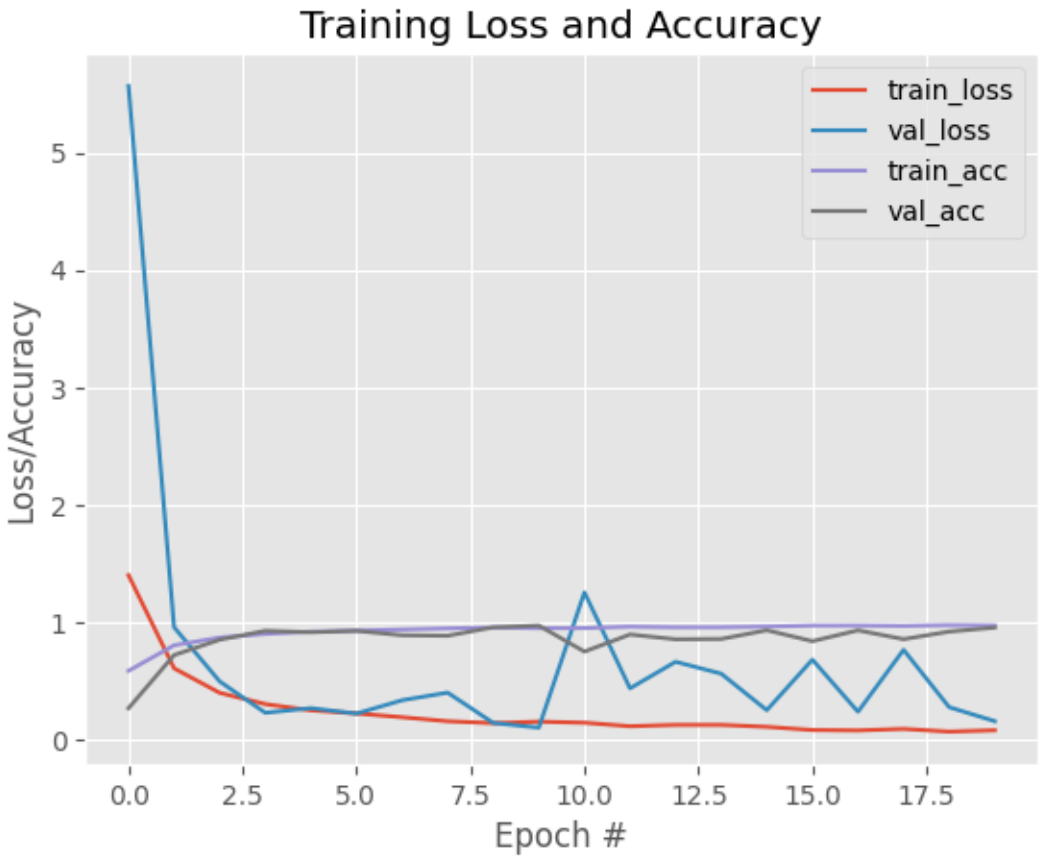
[ ]:
```
# Gráfica de accuracy y losses
plt.style.use("ggplot")
plt.figure()
```

```
# Usar la longitud de H.history["loss"] para el rango de x
epochs_range = np.arange(len(H_aug.history["loss"])) # Por si cambian las␣
 ↪Epochs del entrenamiento

plt.plot(epochs_range, H_aug.history["loss"], label="train_loss")
plt.plot(epochs_range, H_aug.history["val_loss"], label="val_loss")
plt.plot(epochs_range, H_aug.history["accuracy"], label="train_acc")
plt.plot(epochs_range, H_aug.history["val_accuracy"], label="val_acc")

plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()

plt.show()
```



[ ]:
```
# Evaluación del modelo
print("[INFO]: Evaluando el modelo de red neuronal...")
```

```
# Efectuamos la predicción (empleamos el mismo valor de batch_size que en␣
 ↪training)
predictions = model_aug.predict(x_test_aug, batch_size=128) #(X)

# Sacamos el report para test
print(classification_report(y_test_aug, predictions.argmax(axis=1),␣
 ↪target_names=class_names_aug))
```

```
[INFO]: Evaluando el modelo de red neuronal…
24/24 [==============================] - 20s 401ms/step
              precision    recall  f1-score   support

        Bean       0.93      0.94      0.94       200
Bitter_Gourd       0.79      0.98      0.88       200
Bottle_Gourd       0.99      0.94      0.97       200
     Brinjal       0.96      0.96      0.96       200
    Broccoli       0.95      0.97      0.96       200
     Cabbage       0.88      0.93      0.90       200
    Capsicum       0.95      0.99      0.97       200
      Carrot       0.99      0.98      0.98       200
 Cauliflower       0.99      0.96      0.98       200
    Cucumber       1.00      0.80      0.89       200
      Papaya       0.99      0.95      0.97       200
      Potato       0.98      0.98      0.98       200
     Pumpkin       0.89      0.98      0.93       200
      Radish       0.97      0.97      0.97       200
      Tomato       0.99      0.82      0.90       200

    accuracy                           0.95      3000
   macro avg       0.95      0.95      0.95      3000
weighted avg       0.95      0.95      0.95      3000
```

En caso de que se requiera utilizar el modelo recién entrenado (con el cual se aplicó Data augmentation), se ha guardado en la siguiente ruta.

[ ]:
```
# Guardar el modelo (opcional)
model_aug.save('/content/model_aug.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103:
UserWarning: You are saving your model as an HDF5 file via `model.save()`. This
file format is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

Luego de subirlo al entorno de colab, se puede volver a cargar el modelo y seguir entrenándolo si es necesario.

```
[ ]: from tensorflow.keras.models import load_model
     new_model_aug = load_model('/content/model_aug.h5') # debe verificar la ruta

     # Verifique que el estado esté preservado
     # new_predictions = new_model_aug.predict(x_test_aug)
     # np.testing.assert_allclose(predictions, new_predictions, rtol=1e-6, atol=1e-6)
```

.

6 Estrategia 2: Modelo con Red Pre-Entrenada

Para la estrategia con redes pre-entrenadas, se han seleccionado 2 de ImageNet:

1. VGG16
2. ResNet50

Adicionalmente, para reducir el overfitting, se les aplicará:

- Fine Tuning
- Early Stopping o Dropout (según el modelo)

Warning!: *No será posible ejecutar todos los modelos en una misma sesión de Colab.*

- Lo anterior se debe a las limitaciones con la memoria RAM. Incluso, *tampoco es posible* ejecutar todos los modelos en una misma sesión usando una cuenta de Colab Pro.

Por lo tanto, debido a esta limitantes, se han incluido para cada modelo, las **instrucciones** para su correcta ejecución desde el entorno de **Google Colab**, que básicamente consisten (salvo que se especifique otra cosa) en ejecutar las Secciones de Carga del Dataset, la Inspección del Dataset, y en ocasiones el Acondicionamiento del Dataset. Luego, se continuaría con la Sección específica del modelo que se desea ejecutar.

6.1 Modelo con Red VGG16

Instrucciones para la Ejecución de esta Sección en Colab

- Primero, ejecutar las siguientes Secciones del actual Notebook:

  1. Carga del Dataset
  2. Inspección del Dataset

  (excepto aquellas celdas para las cuales se especifique que no es necesario ejecutarlas).

- Posteriormente, continuar en la presente **Sección Estrategia 2/Acondicionamiento con Red VGG16**.

6.1.1 Acondicionamiento con Red VGG16

En primer lugar, se normaliza el set de datos de la misma forma en la que los creadores de la red lo hicieron para su entrenamiento.

[ ]:
```
import numpy as np
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import SGD, Adam
from sklearn.metrics import classification_report
from google.colab import drive
from tensorflow.keras.layers import CategoryEncoding
from tensorflow.keras import backend as K
from tensorflow.keras.layers import Input, Conv2D, Activation, Flatten, Dense,␣
↪Dropout, BatchNormalization, MaxPooling2D
from tensorflow.keras.applications import VGG16, imagenet_utils
```

- **Normalización y One-Hot Encoding: Train y Validation**

Los conjuntos de entrenamiento (**train_ds**) y validación (**val_ds**) se procesan mediante la función **map** para aplicar dos transformaciones:

1. **imagenet_utils.preprocess_input(x)**: Normaliza las imágenes x de acuerdo con los requerimientos de las redes preentrenadas de ImageNet, como VGG16. Esta normalización ajusta los valores de los píxeles de las imágenes para que estén en el rango que el modelo VGG16 espera, basado en cómo fue entrenado originalmente en ImageNet.
2. **OHE(y)**: Convierte las etiquetas y en formato one-hot, usando la capa de one-hot encoding definida con **CategoryEncoding**.

[ ]:
```
# Conversión a one-hot encoding
OHE = CategoryEncoding(num_tokens=15, output_mode="one_hot")
```

[ ]:
```
# Aplicamos one-hot encoding a las etiquetas
norm_train = train_ds.map(lambda x, y: (imagenet_utils.preprocess_input(x),␣
↪OHE(y)))
norm_val = val_ds.map(lambda x, y: (imagenet_utils.preprocess_input(x), OHE(y)))
```

Nótese que, para el acondicionamiento de los Modelos From Scratch, se utilizaba un **rescaling_layer = Rescaling(1.0/255)**, mientras que aquí se está utilizando **imagenet_utils.preprocess_input(x)**.

- **Separación de Imágenes y Etiquetas: Test**

El conjunto de datos de prueba (**test_ds**) se procesa para normalizar las imágenes sin alterar las etiquetas, usando **norm_test = test_ds.map(...)**.

[ ]:
```
# Separamos el conjunto de test entre imágenes y sus etiquetas
# Se utilizarán para evaluar la precisión del modelo
import numpy as np
x_test = []
y_test = []

norm_test = test_ds.map(lambda x,y: (imagenet_utils.preprocess_input(x), y)) #␣
↪dejamos intactas las etiquetas (y)
```

```
for image, label in norm_test.take(len(norm_test)):
  x_test.append(image)
  y_test.append(label)

x_test = np.array(x_test)
y_test = np.array(y_test)
```

6.1.2 Cargar el Base Model

Se carga el **modelo VGG16** con especificaciones particulares:

- **`weights='imagenet'`**: Se especifica que el modelo debe ser cargado con pesos que fueron entrenados previamente en el conjunto de datos ImageNet. Esto proporciona al modelo una comprensión previa de cómo "ver" características generales en imágenes.
- **`include_top=False`**: Esta opción excluye la parte superior (o cabeza) del modelo, que es la última parte de la red que está específicamente diseñada para la clasificación de las 1000 categorías de ImageNet. Al excluir esta parte, se puede adaptar la salida del modelo a un nuevo conjunto de clases específico del problema en cuestión, permitiendo el uso del modelo como un extractor de características en una nueva tarea de clasificación.
- **`input_shape=(224, 224, 3)`**: Define el tamaño de las imágenes de entrada que el modelo espera, en este caso, 224x224 píxeles con 3 canales de color (RGB). Esta especificación es importante para asegurar que todas las imágenes de entrada se redimensionen o preprocesen adecuadamente para coincidir con este formato.

[ ]:
```
# Seleccionar modelo preentrenado (VGG16 en este caso)
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224,␣
 ↪224, 3))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 0s 0us/step
```

[ ]:
```
base_model.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856
```

```
 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

=================================================================
Total params: 14714688 (56.13 MB)
Trainable params: 14714688 (56.13 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

6.1.3 Definir Top Model para Transfer Learning

Al definir el **Top Model** para incorporarlo al modelo base preentrenado de VGG16, se establece la estructura final del modelo para el aprendizaje por transferencia. Este enfoque permite adaptar el modelo preentrenado a una tarea específica, en este caso, la clasificación en 15 categorías distintas. A continuación, se desglosan las configuraciones clave:

1. **Congelar los Pesos del Base Model**

- `base_model.trainable = False`: Esta línea es crucial porque congela (o fija) los **pesos** del modelo base de VGG16, lo que significa que estos pesos no se actualizarán durante el entrenamiento del nuevo modelo.

2. **Crear el Top Model**

- Se inicia un modelo secuencial (**`Sequential()`**) que permite apilar las nuevas capas para la clasificación final sobre el modelo base de VGG16.
- **`pre_trained_model.add(base_model)`**: Añade el modelo base de **VGG16** como la primera capa del nuevo modelo. Este modelo base actuará como un extractor de características potente.
- **`pre_trained_model.add(layers.Flatten())`**: Después de las capas convolucionales y de agrupamiento del modelo base, se **aplana la salida** para convertirla en un vector unidimensional, lo que permite conectarla con capas densas.
- **`pre_trained_model.add(layers.Dense(256, activation='relu'))`**: Añade una capa densa (o totalmente conectada) con **256 unidades** y función de activación ReLU. Esta capa permite aprender combinaciones no lineales de las características extraídas por el modelo base.
- **`pre_trained_model.add(layers.Dense(15, activation='softmax'))`**: La última capa es una densa con 15 unidades, una por cada categoría objetivo, utilizando la función de activación **softmax**. Esto permite que el modelo produzca una distribución de probabilidad sobre las 15 clases, facilitando la clasificación multiclase.

[ ]:
```
# conectarlo a nueva parte densa
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers

base_model.trainable = False # Evitar que los pesos se modifiquen en la parte convolucional -> TRANSFER LEARNING
pre_trained_model = Sequential()
pre_trained_model.add(base_model)
pre_trained_model.add(layers.Flatten())
pre_trained_model.add(layers.Dense(256, activation='relu'))
pre_trained_model.add(layers.Dense(15, activation='softmax'))

pre_trained_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 256)               6422784

 dense_1 (Dense)             (None, 15)                3855

=================================================================
Total params: 21141327 (80.65 MB)
Trainable params: 6426639 (24.52 MB)
```

```
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

6.1.4 Compilación del modelo

- **Compilación**

Se prepara el modelo para el entrenamiento especificando cómo se debe medir el error (**`loss`**), cómo se ajustarán los parámetros del modelo (**`optimizer`**), y qué métrica se usará para evaluar el rendimiento (**`metrics`**).

1. **`loss="categorical_crossentropy"`**: Se utiliza la entropía cruzada categórica como la función de pérdida, adecuada para tareas de clasificación multiclase donde las etiquetas están en formato one-hot.
2. **`optimizer=Adam(...)`**: Se elige el optimizador Adam con un **`learning rate`** de 0.0005. El valor reducido del learning rate (en comparación con el valor por defecto de Adam que suele ser 0.001) se elige para hacer ajustes más finos a los pesos, ya que se parte de un modelo ya entrenado y se busca evitar alteraciones bruscas que puedan degradar las características aprendidas. Se especifica un **`weight_decay=0`** aunque no es un parámetro esperado por el optimizador Adam, pero puede ser útil para frameworks que sí incluyen decay del peso dentro de Adam.
3. **`metrics=["accuracy"]`**: La precisión se utiliza para monitorear el rendimiento del modelo durante el entrenamiento y la validación, proporcionando una idea clara de la proporción de etiquetas que el modelo predice correctamente.

[ ]:
```
# Import the necessary packages
import numpy as np
from tensorflow.keras import backend as K
from tensorflow.keras.layers import Input, Conv2D, Activation, Flatten, Dense,␣
 ↪Dropout, BatchNormalization, MaxPooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import SGD, Adam
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt
from google.colab import drive

# Compilar el modelo
print("[INFO]: Compilando el modelo...")
pre_trained_model.compile(loss="categorical_crossentropy",␣
 ↪optimizer=Adam(learning_rate=0.0005, weight_decay=0, beta_1=0.9, beta_2=0.
 ↪999, epsilon=1e-08), metrics=["accuracy"])
```

```
[INFO]: Compilando el modelo…
```

- **Entrenamiento**

Algunas consideraciones importantes son:

1. **batch_size=128** especifica el número de muestras que se procesarán antes de actualizar los pesos del modelo. Un tamaño de lote mayor puede acelerar el entrenamiento pero requiere más memoria.

2. **epochs=20** establece el número de veces que el algoritmo de entrenamiento trabajará a través de todo el conjunto de datos de entrenamiento. Este número de épocas se elige para permitir que el modelo aprenda de los datos gradualmente, sin sobreajustar.

3. **validation_data=norm_val** proporciona el conjunto de datos de validación contra el cual el modelo se evaluará después de cada época, permitiendo monitorear el rendimiento del modelo en datos no vistos durante el entrenamiento.

[ ]:
```
# Entrenamiento de la red
print("[INFO]: Entrenando la red...")
H_pre = pre_trained_model.fit(norm_train, batch_size=128, epochs=20,␣
 ↪validation_data=norm_val)
```

```
[INFO]: Entrenando la red…
Epoch 1/20
469/469 [==============================] - 103s 199ms/step - loss: 0.7056 -
accuracy: 0.9617 - val_loss: 0.3335 - val_accuracy: 0.9790
Epoch 2/20
469/469 [==============================] - 89s 188ms/step - loss: 0.1507 -
accuracy: 0.9915 - val_loss: 0.1230 - val_accuracy: 0.9943
Epoch 3/20
469/469 [==============================] - 81s 172ms/step - loss: 0.0602 -
accuracy: 0.9963 - val_loss: 0.1853 - val_accuracy: 0.9937
Epoch 4/20
469/469 [==============================] - 88s 187ms/step - loss: 0.1146 -
accuracy: 0.9949 - val_loss: 0.2717 - val_accuracy: 0.9917
Epoch 5/20
469/469 [==============================] - 81s 172ms/step - loss: 0.1590 -
accuracy: 0.9951 - val_loss: 0.1825 - val_accuracy: 0.9950
Epoch 6/20
469/469 [==============================] - 88s 187ms/step - loss: 0.0723 -
accuracy: 0.9973 - val_loss: 0.4383 - val_accuracy: 0.9913
Epoch 7/20
469/469 [==============================] - 88s 186ms/step - loss: 0.1761 -
accuracy: 0.9957 - val_loss: 0.1685 - val_accuracy: 0.9957
Epoch 8/20
469/469 [==============================] - 89s 189ms/step - loss: 0.0654 -
accuracy: 0.9979 - val_loss: 0.1311 - val_accuracy: 0.9963
Epoch 9/20
469/469 [==============================] - 88s 187ms/step - loss: 0.0306 -
accuracy: 0.9988 - val_loss: 0.0688 - val_accuracy: 0.9983
Epoch 10/20
469/469 [==============================] - 82s 174ms/step - loss: 0.0348 -
accuracy: 0.9993 - val_loss: 0.6508 - val_accuracy: 0.9910
Epoch 11/20
```

```
469/469 [==============================] - 81s 171ms/step - loss: 0.1406 -
accuracy: 0.9969 - val_loss: 0.2679 - val_accuracy: 0.9960
Epoch 12/20
469/469 [==============================] - 88s 188ms/step - loss: 0.1133 -
accuracy: 0.9977 - val_loss: 0.8754 - val_accuracy: 0.9917
Epoch 13/20
469/469 [==============================] - 81s 171ms/step - loss: 0.0563 -
accuracy: 0.9984 - val_loss: 0.2761 - val_accuracy: 0.9963
Epoch 14/20
469/469 [==============================] - 81s 173ms/step - loss: 0.0120 -
accuracy: 0.9997 - val_loss: 0.1259 - val_accuracy: 0.9977
Epoch 15/20
469/469 [==============================] - 81s 173ms/step - loss: 0.1275 -
accuracy: 0.9984 - val_loss: 1.0482 - val_accuracy: 0.9893
Epoch 16/20
469/469 [==============================] - 81s 172ms/step - loss: 0.0817 -
accuracy: 0.9983 - val_loss: 0.2128 - val_accuracy: 0.9977
Epoch 17/20
469/469 [==============================] - 82s 175ms/step - loss: 0.0395 -
accuracy: 0.9989 - val_loss: 0.1379 - val_accuracy: 0.9980
Epoch 18/20
469/469 [==============================] - 88s 187ms/step - loss: 0.0392 -
accuracy: 0.9995 - val_loss: 0.1323 - val_accuracy: 0.9983
Epoch 19/20
469/469 [==============================] - 80s 170ms/step - loss: 0.0065 -
accuracy: 0.9997 - val_loss: 0.0878 - val_accuracy: 0.9987
Epoch 20/20
469/469 [==============================] - 81s 172ms/step - loss: 0.0013 -
accuracy: 0.9999 - val_loss: 0.0980 - val_accuracy: 0.9983
```

6.1.5 Evaluación del modelo de Transfer Learning

El modelo da resultados excelentes en el test. Esto es sorprendente teniendo en cuenta la simplicidad del top model que se ha añadido y es muestra de la buena calidad del base model, cuyas capas convolucionales contienen pesos entrenados con una cantidad inmensa de datos y son excelentes detectando características base como bordes y texturas. Tenemos unos valores f1-score perfectos para la detección de repollo, pimiento, zanahoria, calabaza y rábano. Esto ya supera con creces los resultados obtenidos en la red from scratch, teniendo todos los f1-score mayores o iguales a 0.99.

El modelo entra en un mínimo local al no observar mejora de la pérdida tras completar la mitad del proceso del entrenamiento. Se buscará salir de este mínimo local aplicando Fine Tuning.

- **Definición de Clases**

[ ]:
```
# clases
class_names = test_ds.class_names
class_names
```

[ ]:
```
['Bean',
 'Bitter_Gourd',
 'Bottle_Gourd',
 'Brinjal',
 'Broccoli',
 'Cabbage',
 'Capsicum',
 'Carrot',
 'Cauliflower',
 'Cucumber',
 'Papaya',
 'Potato',
 'Pumpkin',
 'Radish',
 'Tomato']
```

- **Evaluación del Modelo**

[ ]:
```
# Evaluación del modelo
print("[INFO]: Evaluando el modelo...")

# Efectuamos la predicción (empleamos el mismo valor de batch_size que en␣
 ↪training)
predictions = pre_trained_model.predict(x_test, batch_size=128)

# Sacamos el report para test
print(classification_report(y_test, predictions.argmax(axis=1),␣
 ↪target_names=class_names))
```

```
[INFO]: Evaluando el modelo…
24/24 [==============================] - 46s 1s/step
              precision    recall  f1-score   support

        Bean       1.00      1.00      1.00       200
Bitter_Gourd       1.00      0.99      1.00       200
Bottle_Gourd       1.00      1.00      1.00       200
     Brinjal       1.00      1.00      1.00       200
    Broccoli       1.00      0.99      1.00       200
     Cabbage       1.00      1.00      1.00       200
    Capsicum       1.00      1.00      1.00       200
      Carrot       1.00      1.00      1.00       200
 Cauliflower       1.00      1.00      1.00       200
    Cucumber       1.00      1.00      1.00       200
      Papaya       1.00      0.99      1.00       200
      Potato       1.00      1.00      1.00       200
     Pumpkin       1.00      1.00      1.00       200
      Radish       1.00      1.00      1.00       200
      Tomato       1.00      1.00      1.00       200
```

```
    accuracy                           1.00      3000
   macro avg       1.00      1.00      1.00      3000
weighted avg       1.00      1.00      1.00      3000
```

- **Visualización del Desempeño del Modelo**

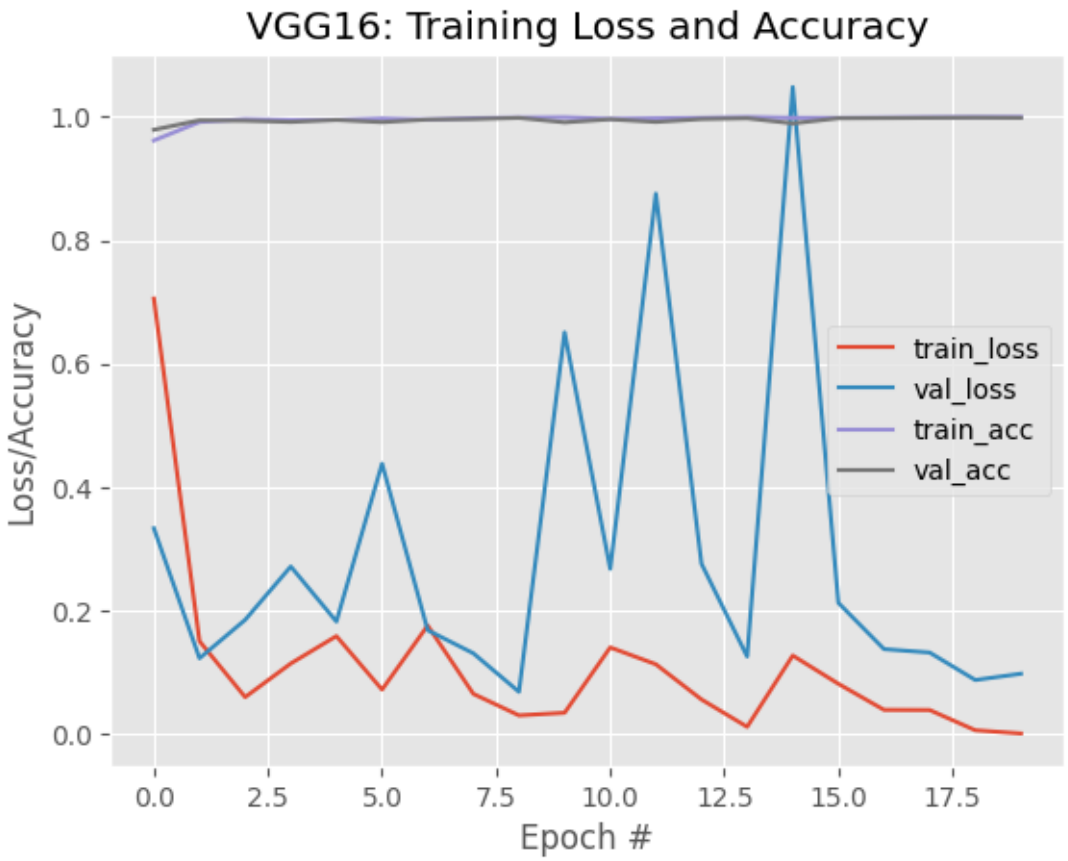
[ ]:
```python
# Gráfica de accuracy y losses
plt.style.use("ggplot")
plt.figure()

# Usar la longitud de H.history["loss"] para el rango de x
epochs_range = np.arange(len(H_pre.history["loss"])) # Por si cambian las␣
 ↪Epochs del entrenamiento

plt.plot(epochs_range, H_pre.history["loss"], label="train_loss")
plt.plot(epochs_range, H_pre.history["val_loss"], label="val_loss")
plt.plot(epochs_range, H_pre.history["accuracy"], label="train_acc")
plt.plot(epochs_range, H_pre.history["val_accuracy"], label="val_acc")

plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()

plt.show()
```



[ ]:
```
# Guardar el modelo (opcional)
pre_trained_model.save('/content/model_vgg16.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103:
UserWarning: You are saving your model as an HDF5 file via `model.save()`. This
file format is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

6.1.6 Aplicación de Fine Tuning y EarlyStopping para Reducir Overfitting

Si hasta este momento de la sesión, la RAM de Colab es suficiente (aproximadamente 7/12MB), entonces se puede continuar en la misma sesión. De no ser así, entonces hay que seguir las siguientes instrucciones.

Instrucciones para la Ejecución de esta Sección en Colab

- Primero, ejecutar (en una nueva sesión) las siguientes Secciones del actual Notebook:
  1. Carga del Dataset
  2. Inspección del Dataset

(excepto aquellas celdas para las cuales se especifique que no es necesario ejecutarlas).

- Luego, ejecutar las celdas de la **Sección Estrategia 2/Acondicionamiento con Red VGG16**.
- Posteriormente, continuar en la presente **Sección Estrategia 2/Aplicación de Fine Tuning y EarlyStopping**, con la carga del Base Model .

El Base Model se congela hasta una capa específica (`block4_pool`) para evitar ajustes innecesarios. Adicionalmente se aplica un Early Stopping con `patience=3`.

[ ]:
```
# Importar
from tensorflow.keras.applications import VGG16, imagenet_utils
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import optimizers
from tensorflow.keras.optimizers import SGD, Adam
from tensorflow.keras.layers import Dropout, Flatten, Dense
from tensorflow.keras import Model
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
import numpy as np
```

Se carga **VGG16** con pesos preentrenados en ImageNet, excluyendo la parte superior (clasificador original), y se establece la forma de entrada esperada para las imágenes.

[ ]:
```
####### BASE MODEL ######################
# Importamos VGG16 con pesos de imagenet y sin top_model especificando tamaño␣
 ↪de entrada de datos
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224,␣
 ↪224, 3))

# Mostramos la arquitectura
base_model.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0
```

```
 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

=================================================================
Total params: 14714688 (56.13 MB)
Trainable params: 14714688 (56.13 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

- **Congelar Pesos**

Se congelan todos los pesos en el modelo base de VGG16 para prevenir que se modifiquen durante el entrenamiento inicial, mediante la especificación: **`base_model.trainable = False`**.

[ ]:
```python
# conectarlo a nueva parte densa
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers

base_model.trainable = False # Evitar que los pesos se modifiquen en la parte␣
 ↪convolucional -> TRANSFER LEARNING
```

- **Congelar Capas**

Algunas consideraciones importantes son:

1. **Congelación Selectiva**:

- Se realiza una congelación selectiva de capas hasta la capa **`block4_pool`** incluida. Esto se hace iterando sobre las capas del modelo base y congelando las capas hasta (y no incluyendo) una capa específica.
- Las capas anteriores a **`block4_pool`** son congeladas para que sus pesos no se actualicen, lo que permite que el modelo mantenga las características generales aprendidas en las primeras capas, que suelen ser más abstractas y aplicables a una amplia variedad de tareas de visión por computadora.

2. **Clasificador Personalizado**:

- Se construye un nuevo clasificador para reemplazar la parte superior del modelo original de VGG16. Este clasificador está diseñado para adaptarse a la tarea específica, con **15 unidades de salida** correspondientes a las categorías del conjunto de datos objetivo.
- Este clasificador personalizado incluye una capa **`Flatten`** para convertir las características extraídas por VGG16 en un vector, seguido de dos capas densas con activación **`ReLU`** y una capa de dropout para reducir el sobreajuste. La última capa es una densa con activación softmax para clasificación multiclase.

5. **Modelo Final**:

- Se crea el modelo final combinando el modelo base de VGG16 (con las capas congeladas y las restantes ajustables) y el nuevo clasificador personalizado. Esto se logra tomando la entrada del modelo base y conectándola a la salida del clasificador personalizado, lo cual se indica con: **`model = Model(base_model.input, x)`**.

[ ]:
```
# congelar capas
for layer in base_model.layers:
  if layer.name == 'block4_pool':
    break
  layer.trainable = False
  print('Capa ' + layer.name + ' congelada...')
```

```
# Cogemos la última capa del model y le añadimos nuestro clasificador␣
  ↪(top_model)
last = base_model.layers[-1].output # ultima capa del base model

x = Flatten()(last)
x = Dense(1024, activation='relu', name='fc1')(x)
x = Dropout(0.3)(x)
x = Dense(256, activation='relu', name='fc2')(x)
x = Dense(15, activation='softmax', name='predictions')(x)

model = Model(base_model.input, x)
```

```
Capa input_2 congelada…
Capa block1_conv1 congelada…
Capa block1_conv2 congelada…
Capa block1_pool congelada…
Capa block2_conv1 congelada…
Capa block2_conv2 congelada…
Capa block2_pool congelada…
Capa block3_conv1 congelada…
Capa block3_conv2 congelada…
Capa block3_conv3 congelada…
Capa block3_pool congelada…
Capa block4_conv1 congelada…
Capa block4_conv2 congelada…
Capa block4_conv3 congelada…
```

[ ]:
```
# Compilamos el modelo
model.compile(optimizer=Adam(learning_rate=0.0005, weight_decay=0, beta_1=0.9,␣
  ↪beta_2=0.999, epsilon=1e-08), loss='categorical_crossentropy',␣
  ↪metrics=['accuracy'])

model.summary()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
input_2 (InputLayer)        [(None, 224, 224, 3)]     0
block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792
block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928
block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0
block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856
block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584
block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0
block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168
block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080
block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080
block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0
block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160
block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808
block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808
block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0
block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808
block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808
block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808
block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0
flatten_1 (Flatten)         (None, 25088)             0
fc1 (Dense)                 (None, 1024)              25691136
dropout_1 (Dropout)         (None, 1024)              0
fc2 (Dense)                 (None, 256)               262400
predictions (Dense)         (None, 15)                3855
```

```
=================================================================
Total params: 40672079 (155.15 MB)
Trainable params: 25957391 (99.02 MB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

- **Callbacks**

Se definen los siguientes **Callbacks**:

1. **EarlyStopping**:

- Monitorea la pérdida de validación (**`val_loss`**) y detiene el entrenamiento si no se observan mejoras después de un número especificado de épocas (**`patience=3`**). Esto previene el sobreajuste al detener el entrenamiento cuando el modelo ya no mejora con respecto al conjunto de validación.
- **`mode='min'`** indica que el proceso de EarlyStopping busca minimizar el valor de **`val_loss`**.

2. **ModelCheckpoint**:

- Guarda solo los pesos del modelo en el directorio especificado (**`/content/`**) cuando se observa una mejora en la precisión de validación (**monitor=`val_accuracy`**).
- **`save_weights_only=True`** indica que solo se guardarán los pesos del modelo, no toda la arquitectura del modelo.
- **`mode='max'`** indica que el objetivo es maximizar la precisión de validación.

3. **TensorBoard**:

- Permite la visualización del proceso de entrenamiento y otras estadísticas a través de TensorBoard, almacenando los logs en **`log_dir='./logs'`**. Esto es útil para monitorear el rendimiento del modelo, incluyendo la pérdida y la precisión a lo largo del tiempo, así como para realizar otros análisis post-entrenamiento.

[ ]:
```
my_callbacks = [
    tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                     patience=3, # Valor de patience con mejor␣
 ↪resultado entre 2-5
                                     mode='min'),
    tf.keras.callbacks.ModelCheckpoint(filepath='/content/',
```

```
                                        monitor='val_accuracy',
                                        save_weights_only=True,
                                        mode='max'),
    tf.keras.callbacks.TensorBoard(log_dir='./logs')
]
```

- **Entrenamiento**

Algunas consideraciones importantes son:

1. **validation_data=norm_val** permite evaluar el rendimiento del modelo en el conjunto de validación después de cada época, ofreciendo una visión de cómo el modelo generaliza a datos no vistos durante el entrenamiento.
2. **verbose=1** activa mensajes detallados durante el entrenamiento para proporcionar información sobre el progreso y el rendimiento del modelo.

[ ]:
```
n_epochs=20

# Entrenamos el modelo
H = model.fit(norm_train, validation_data = norm_val, batch_size=128,
              epochs=n_epochs, verbose=1, callbacks=my_callbacks)
```

```
Epoch 1/20
469/469 [==============================] - 84s 175ms/step - loss: 1.2776 -
accuracy: 0.9415 - val_loss: 0.2226 - val_accuracy: 0.9893
Epoch 2/20
469/469 [==============================] - 88s 186ms/step - loss: 0.3213 -
accuracy: 0.9853 - val_loss: 0.1669 - val_accuracy: 0.9937
Epoch 3/20
469/469 [==============================] - 90s 191ms/step - loss: 0.3221 -
accuracy: 0.9904 - val_loss: 0.2893 - val_accuracy: 0.9893
Epoch 4/20
469/469 [==============================] - 87s 184ms/step - loss: 0.3759 -
accuracy: 0.9894 - val_loss: 0.6077 - val_accuracy: 0.9880
Epoch 5/20
469/469 [==============================] - 90s 191ms/step - loss: 0.2393 -
accuracy: 0.9935 - val_loss: 0.1204 - val_accuracy: 0.9960
Epoch 6/20
469/469 [==============================] - 91s 194ms/step - loss: 0.0848 -
accuracy: 0.9969 - val_loss: 0.2307 - val_accuracy: 0.9950
Epoch 7/20
469/469 [==============================] - 93s 198ms/step - loss: 0.2636 -
accuracy: 0.9947 - val_loss: 0.1095 - val_accuracy: 0.9977
Epoch 8/20
```

```
469/469 [==============================] - 90s 191ms/step - loss: 0.2637 -
accuracy: 0.9949 - val_loss: 0.3819 - val_accuracy: 0.9940
Epoch 9/20
469/469 [==============================] - 87s 186ms/step - loss: 0.2529 -
accuracy: 0.9950 - val_loss: 0.3732 - val_accuracy: 0.9957
Epoch 10/20
469/469 [==============================] - 93s 197ms/step - loss: 0.2338 -
accuracy: 0.9950 - val_loss: 0.5691 - val_accuracy: 0.9960
```

6.1.7 Evaluación Post Fine Tuning

Aunque el overfitting haya mejorado, se observa que el accuracy en la etapa de test pasa de 0.99 a 0.94 al aplicar Fine Tuning. Esto puede deberse a que el hecho de reentrenar el último bloque convolucional redujo su capacidad de generalización, quizás por la relativa pequeña cantidad de imágenes usadas para entrenar o porque las imágenes usadas para entrenar al VGG16 hayan sido muy distintas a las usadas en estos entrenamientos.

[ ]:
```
# clases
class_names = test_ds.class_names
class_names
```

[ ]:
```
['Bean',
 'Bitter_Gourd',
 'Bottle_Gourd',
 'Brinjal',
 'Broccoli',
 'Cabbage',
 'Capsicum',
 'Carrot',
 'Cauliflower',
 'Cucumber',
 'Papaya',
 'Potato',
 'Pumpkin',
 'Radish',
 'Tomato']
```

- **Evaluación**

[ ]:
```
# Evaluación del modelo
print("[INFO]: Evaluando el modelo...")
predictions = model.predict(x_test, batch_size=128)
```

```
# Obtener el report de clasificación
print(classification_report(y_test, predictions.argmax(axis=1),␣
 ↪target_names=class_names))
```

```
[INFO]: Evaluando el modelo…
24/24 [==============================] - 12s 496ms/step
              precision    recall  f1-score   support

        Bean       1.00      0.99      0.99       200
Bitter_Gourd       1.00      0.99      1.00       200
Bottle_Gourd       0.98      1.00      0.99       200
     Brinjal       0.98      1.00      0.99       200
    Broccoli       1.00      1.00      1.00       200
     Cabbage       1.00      1.00      1.00       200
    Capsicum       1.00      0.99      1.00       200
      Carrot       1.00      0.99      1.00       200
 Cauliflower       1.00      1.00      1.00       200
    Cucumber       1.00      0.99      0.99       200
      Papaya       1.00      0.98      0.99       200
      Potato       1.00      0.99      1.00       200
     Pumpkin       1.00      1.00      1.00       200
      Radish       0.99      0.99      0.99       200
      Tomato       1.00      1.00      1.00       200

    accuracy                           1.00      3000
   macro avg       1.00      1.00      1.00      3000
weighted avg       1.00      1.00      1.00      3000
```

- **Visualización del Desempeño del Modelo**

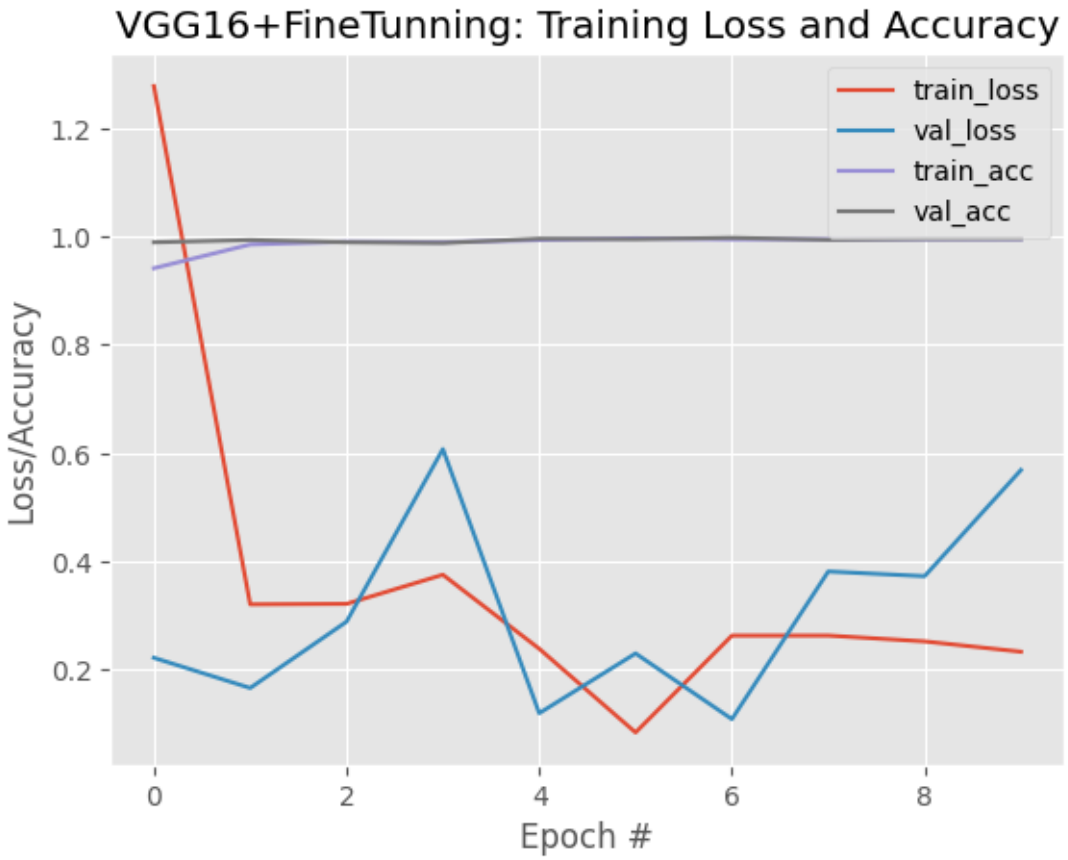
[ ]:
```
# Gráfica de accuracy y losses
plt.style.use("ggplot")
plt.figure()

# Usar la longitud de H.history["loss"] para el rango de x
epochs_range = np.arange(len(H.history["loss"])) # Por si cambian las Epochs␣
 ↪del entrenamiento

plt.plot(epochs_range, H.history["loss"], label="train_loss")
plt.plot(epochs_range, H.history["val_loss"], label="val_loss")
plt.plot(epochs_range, H.history["accuracy"], label="train_acc")
plt.plot(epochs_range, H.history["val_accuracy"], label="val_acc")
```

```
plt.title("VGG16+FineTunning: Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()

plt.show()
```




```
[ ]: # Guardar el modelo (opcional)
model.save('/content/model_vgg16_ftunning.h5')
```

.

6.2 Modelo con Red ResNet50

Instrucciones para la Ejecución de esta Sección en Colab

- Primero, ejecutar las siguientes Secciones del actual Notebook:

  1. Carga del Dataset
  2. Inspección del Dataset

  (excepto aquellas celdas para las cuales se especifique que no es necesario ejecutarlas).

- Posteriormente, continuar en la presente **Sección Estrategia 2/Acondicionamiento con Red ResNet50**.

6.2.1 Acondicionamiento con Red ResNet50

ResNet50 es una arquitectura de red profunda que utiliza bloques residuales para permitir el entrenamiento de redes mucho más profundas al abordar el problema de la desaparición de gradientes.

La inclusión de **ResNet50** en lugar de **VGG16** puede ofrecer varias ventajas, como una mayor eficiencia en términos de computación y potencialmente un mejor rendimiento, dado su diseño profundo y la capacidad de aprender características más complejas y abstractas.

[ ]:
```
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import classification_report
from google.colab import drive
from tensorflow.keras.layers import CategoryEncoding
from tensorflow.keras.applications import ResNet50, imagenet_utils
```

- **Normalización y One-Hot Encoding: Train y Validation**

Los conjuntos de entrenamiento (`train_ds`) y validación (`val_ds`) se procesan mediante la función `map` para aplicar dos transformaciones:

1. `imagenet_utils.preprocess_input(x)`: Se aplica a cada imagen en los conjuntos de datos para asegurar que las imágenes estén en el formato adecuado (normalización) para el modelo preentrenado. Esta normalización es específica para los modelos entrenados en ImageNet y ajusta las imágenes de acuerdo con la manera en que fue entrenado el modelo original.
2. **One-Hot Encoding**: Las etiquetas se convierten a formato one-hot usando `CategoryEncoding`, lo cual es necesario para la clasificación multiclase.

```
[ ]: # Conversión a one-hot encoding
     OHE = CategoryEncoding(num_tokens=15, output_mode="one_hot")
```

```
[ ]: # Aplicamos one-hot encoding a las etiquetas
     norm_train = train_ds.map(lambda x, y: (imagenet_utils.preprocess_input(x),␣
      ↪OHE(y)))
     norm_val = val_ds.map(lambda x, y: (imagenet_utils.preprocess_input(x), OHE(y)))
```

- **Separación de Imágenes y Etiquetas: Test**

Se preparan arrays de imágenes y etiquetas del conjunto de **`test`**, que serán utilizados para evaluar la precisión del modelo.

```
[ ]: # Separamos el conjunto de test entre imágenes y sus etiquetas
     # Se utilizarán para evaluar la precisión del modelo
     import numpy as np
     x_test = []
     y_test = []

     norm_test = test_ds.map(lambda x,y: (imagenet_utils.preprocess_input(x), y)) #␣
      ↪dejamos intactas las etiquetas (y)

     for image, label in norm_test.take(len(norm_test)):
       x_test.append(image)
       y_test.append(label)

     x_test = np.array(x_test)
     y_test = np.array(y_test)
```

6.2.2 Cargar el Base Model

Se carga el modelo **ResNet50** con **`weights='imagenet'`**, lo que significa que el modelo viene con pesos preentrenados en el conjunto de datos de ImageNet.

`include_top=False` excluye la parte superior (clasificador) del modelo, permitiendo personalizar el modelo para un número específico de clases (en este caso, 15).

`input_shape=(224, 224, 3)` define el tamaño y forma de las imágenes de entrada esperadas por el modelo.

[ ]:
```
# Seleccionar modelo preentrenado (ResNet50 en este caso)
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224,
 ↪224, 3))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 5s 0us/step
```

[ ]:
```
base_model.summary()
```

```
Model: "resnet50"
__________________________________________________________________________________
__________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================
==================
 input_1 (InputLayer)        [(None, 224, 224, 3)]        0         []

 conv1_pad (ZeroPadding2D)   (None, 230, 230, 3)          0
['input_1[0][0]']

 conv1_conv (Conv2D)         (None, 112, 112, 64)         9472
['conv1_pad[0][0]']

 conv1_bn (BatchNormalizati  (None, 112, 112, 64)         256
['conv1_conv[0][0]']
 on)

 conv1_relu (Activation)     (None, 112, 112, 64)         0
['conv1_bn[0][0]']

 pool1_pad (ZeroPadding2D)   (None, 114, 114, 64)         0
['conv1_relu[0][0]']

 pool1_pool (MaxPooling2D)   (None, 56, 56, 64)           0
['pool1_pad[0][0]']

 conv2_block1_1_conv (Conv2  (None, 56, 56, 64)           4160
['pool1_pool[0][0]']
 D)

 conv2_block1_1_bn (BatchNo  (None, 56, 56, 64)           256
['conv2_block1_1_conv[0][0]']
 rmalization)

 conv2_block1_1_relu (Activ  (None, 56, 56, 64)           0
['conv2_block1_1_bn[0][0]']
 ation)

 conv2_block1_2_conv (Conv2  (None, 56, 56, 64)           36928
```

```
['conv2_block1_1_relu[0][0]']
 D)

 conv2_block1_2_bn (BatchNo  (None, 56, 56, 64)           256
['conv2_block1_2_conv[0][0]']
 rmalization)

 conv2_block1_2_relu (Activ  (None, 56, 56, 64)           0
['conv2_block1_2_bn[0][0]']
 ation)

 conv2_block1_0_conv (Conv2  (None, 56, 56, 256)          16640
['pool1_pool[0][0]']
 D)

 conv2_block1_3_conv (Conv2  (None, 56, 56, 256)          16640
['conv2_block1_2_relu[0][0]']
 D)

 conv2_block1_0_bn (BatchNo  (None, 56, 56, 256)          1024
['conv2_block1_0_conv[0][0]']
 rmalization)

 conv2_block1_3_bn (BatchNo  (None, 56, 56, 256)          1024
['conv2_block1_3_conv[0][0]']
 rmalization)

 conv2_block1_add (Add)      (None, 56, 56, 256)          0
['conv2_block1_0_bn[0][0]',
'conv2_block1_3_bn[0][0]']

 conv2_block1_out (Activati  (None, 56, 56, 256)          0
['conv2_block1_add[0][0]']
 on)

 conv2_block2_1_conv (Conv2  (None, 56, 56, 64)           16448
['conv2_block1_out[0][0]']
 D)

 conv2_block2_1_bn (BatchNo  (None, 56, 56, 64)           256
['conv2_block2_1_conv[0][0]']
 rmalization)

 conv2_block2_1_relu (Activ  (None, 56, 56, 64)           0
['conv2_block2_1_bn[0][0]']
 ation)

 conv2_block2_2_conv (Conv2  (None, 56, 56, 64)           36928
```

```
['conv2_block2_1_relu[0][0]']
 D)

 conv2_block2_2_bn (BatchNo  (None, 56, 56, 64)           256
['conv2_block2_2_conv[0][0]']
 rmalization)

 conv2_block2_2_relu (Activ  (None, 56, 56, 64)           0
['conv2_block2_2_bn[0][0]']
 ation)

 conv2_block2_3_conv (Conv2  (None, 56, 56, 256)          16640
['conv2_block2_2_relu[0][0]']
 D)

 conv2_block2_3_bn (BatchNo  (None, 56, 56, 256)          1024
['conv2_block2_3_conv[0][0]']
 rmalization)

 conv2_block2_add (Add)      (None, 56, 56, 256)          0
['conv2_block1_out[0][0]',
'conv2_block2_3_bn[0][0]']

 conv2_block2_out (Activati  (None, 56, 56, 256)          0
['conv2_block2_add[0][0]']
 on)

 conv2_block3_1_conv (Conv2  (None, 56, 56, 64)           16448
['conv2_block2_out[0][0]']
 D)

 conv2_block3_1_bn (BatchNo  (None, 56, 56, 64)           256
['conv2_block3_1_conv[0][0]']
 rmalization)

 conv2_block3_1_relu (Activ  (None, 56, 56, 64)           0
['conv2_block3_1_bn[0][0]']
 ation)

 conv2_block3_2_conv (Conv2  (None, 56, 56, 64)           36928
['conv2_block3_1_relu[0][0]']
 D)

 conv2_block3_2_bn (BatchNo  (None, 56, 56, 64)           256
['conv2_block3_2_conv[0][0]']
 rmalization)

 conv2_block3_2_relu (Activ  (None, 56, 56, 64)           0
```

```
['conv2_block3_2_bn[0][0]']
 ation)

 conv2_block3_3_conv (Conv2  (None, 56, 56, 256)          16640
['conv2_block3_2_relu[0][0]']
 D)

 conv2_block3_3_bn (BatchNo  (None, 56, 56, 256)          1024
['conv2_block3_3_conv[0][0]']
 rmalization)

 conv2_block3_add (Add)      (None, 56, 56, 256)          0
['conv2_block2_out[0][0]',
'conv2_block3_3_bn[0][0]']

 conv2_block3_out (Activati  (None, 56, 56, 256)          0
['conv2_block3_add[0][0]']
 on)

 conv3_block1_1_conv (Conv2  (None, 28, 28, 128)          32896
['conv2_block3_out[0][0]']
 D)

 conv3_block1_1_bn (BatchNo  (None, 28, 28, 128)          512
['conv3_block1_1_conv[0][0]']
 rmalization)

 conv3_block1_1_relu (Activ  (None, 28, 28, 128)          0
['conv3_block1_1_bn[0][0]']
 ation)

 conv3_block1_2_conv (Conv2  (None, 28, 28, 128)          147584
['conv3_block1_1_relu[0][0]']
 D)

 conv3_block1_2_bn (BatchNo  (None, 28, 28, 128)          512
['conv3_block1_2_conv[0][0]']
 rmalization)

 conv3_block1_2_relu (Activ  (None, 28, 28, 128)          0
['conv3_block1_2_bn[0][0]']
 ation)

 conv3_block1_0_conv (Conv2  (None, 28, 28, 512)          131584
['conv2_block3_out[0][0]']
 D)

 conv3_block1_3_conv (Conv2  (None, 28, 28, 512)          66048
```

```
['conv3_block1_2_relu[0][0]']
 D)

 conv3_block1_0_bn (BatchNo  (None, 28, 28, 512)          2048
['conv3_block1_0_conv[0][0]']
 rmalization)

 conv3_block1_3_bn (BatchNo  (None, 28, 28, 512)          2048
['conv3_block1_3_conv[0][0]']
 rmalization)

 conv3_block1_add (Add)      (None, 28, 28, 512)          0
['conv3_block1_0_bn[0][0]',
'conv3_block1_3_bn[0][0]']

 conv3_block1_out (Activati  (None, 28, 28, 512)          0
['conv3_block1_add[0][0]']
 on)

 conv3_block2_1_conv (Conv2  (None, 28, 28, 128)          65664
['conv3_block1_out[0][0]']
 D)

 conv3_block2_1_bn (BatchNo  (None, 28, 28, 128)          512
['conv3_block2_1_conv[0][0]']
 rmalization)

 conv3_block2_1_relu (Activ  (None, 28, 28, 128)          0
['conv3_block2_1_bn[0][0]']
 ation)

 conv3_block2_2_conv (Conv2  (None, 28, 28, 128)          147584
['conv3_block2_1_relu[0][0]']
 D)

 conv3_block2_2_bn (BatchNo  (None, 28, 28, 128)          512
['conv3_block2_2_conv[0][0]']
 rmalization)

 conv3_block2_2_relu (Activ  (None, 28, 28, 128)          0
['conv3_block2_2_bn[0][0]']
 ation)

 conv3_block2_3_conv (Conv2  (None, 28, 28, 512)          66048
['conv3_block2_2_relu[0][0]']
 D)

 conv3_block2_3_bn (BatchNo  (None, 28, 28, 512)          2048
```

```
['conv3_block2_3_conv[0][0]']
 rmalization)

 conv3_block2_add (Add)      (None, 28, 28, 512)          0
['conv3_block1_out[0][0]',
'conv3_block2_3_bn[0][0]']

 conv3_block2_out (Activati  (None, 28, 28, 512)          0
['conv3_block2_add[0][0]']
 on)

 conv3_block3_1_conv (Conv2  (None, 28, 28, 128)          65664
['conv3_block2_out[0][0]']
 D)

 conv3_block3_1_bn (BatchNo  (None, 28, 28, 128)          512
['conv3_block3_1_conv[0][0]']
 rmalization)

 conv3_block3_1_relu (Activ  (None, 28, 28, 128)          0
['conv3_block3_1_bn[0][0]']
 ation)

 conv3_block3_2_conv (Conv2  (None, 28, 28, 128)          147584
['conv3_block3_1_relu[0][0]']
 D)

 conv3_block3_2_bn (BatchNo  (None, 28, 28, 128)          512
['conv3_block3_2_conv[0][0]']
 rmalization)

 conv3_block3_2_relu (Activ  (None, 28, 28, 128)          0
['conv3_block3_2_bn[0][0]']
 ation)

 conv3_block3_3_conv (Conv2  (None, 28, 28, 512)          66048
['conv3_block3_2_relu[0][0]']
 D)

 conv3_block3_3_bn (BatchNo  (None, 28, 28, 512)          2048
['conv3_block3_3_conv[0][0]']
 rmalization)

 conv3_block3_add (Add)      (None, 28, 28, 512)          0
['conv3_block2_out[0][0]',
'conv3_block3_3_bn[0][0]']

 conv3_block3_out (Activati  (None, 28, 28, 512)          0
```

```
['conv3_block3_add[0][0]']
 on)

 conv3_block4_1_conv (Conv2  (None, 28, 28, 128)          65664
['conv3_block3_out[0][0]']
 D)

 conv3_block4_1_bn (BatchNo  (None, 28, 28, 128)          512
['conv3_block4_1_conv[0][0]']
 rmalization)

 conv3_block4_1_relu (Activ  (None, 28, 28, 128)          0
['conv3_block4_1_bn[0][0]']
 ation)

 conv3_block4_2_conv (Conv2  (None, 28, 28, 128)          147584
['conv3_block4_1_relu[0][0]']
 D)

 conv3_block4_2_bn (BatchNo  (None, 28, 28, 128)          512
['conv3_block4_2_conv[0][0]']
 rmalization)

 conv3_block4_2_relu (Activ  (None, 28, 28, 128)          0
['conv3_block4_2_bn[0][0]']
 ation)

 conv3_block4_3_conv (Conv2  (None, 28, 28, 512)          66048
['conv3_block4_2_relu[0][0]']
 D)

 conv3_block4_3_bn (BatchNo  (None, 28, 28, 512)          2048
['conv3_block4_3_conv[0][0]']
 rmalization)

 conv3_block4_add (Add)      (None, 28, 28, 512)          0
['conv3_block3_out[0][0]',
'conv3_block4_3_bn[0][0]']

 conv3_block4_out (Activati  (None, 28, 28, 512)          0
['conv3_block4_add[0][0]']
 on)

 conv4_block1_1_conv (Conv2  (None, 14, 14, 256)          131328
['conv3_block4_out[0][0]']
 D)

 conv4_block1_1_bn (BatchNo  (None, 14, 14, 256)          1024
```

```
['conv4_block1_1_conv[0][0]']
 rmalization)

 conv4_block1_1_relu (Activ  (None, 14, 14, 256)          0
['conv4_block1_1_bn[0][0]']
 ation)

 conv4_block1_2_conv (Conv2  (None, 14, 14, 256)          590080
['conv4_block1_1_relu[0][0]']
 D)

 conv4_block1_2_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block1_2_conv[0][0]']
 rmalization)

 conv4_block1_2_relu (Activ  (None, 14, 14, 256)          0
['conv4_block1_2_bn[0][0]']
 ation)

 conv4_block1_0_conv (Conv2  (None, 14, 14, 1024)         525312
['conv3_block4_out[0][0]']
 D)

 conv4_block1_3_conv (Conv2  (None, 14, 14, 1024)         263168
['conv4_block1_2_relu[0][0]']
 D)

 conv4_block1_0_bn (BatchNo  (None, 14, 14, 1024)         4096
['conv4_block1_0_conv[0][0]']
 rmalization)

 conv4_block1_3_bn (BatchNo  (None, 14, 14, 1024)         4096
['conv4_block1_3_conv[0][0]']
 rmalization)

 conv4_block1_add (Add)      (None, 14, 14, 1024)         0
['conv4_block1_0_bn[0][0]',
'conv4_block1_3_bn[0][0]']

 conv4_block1_out (Activati  (None, 14, 14, 1024)         0
['conv4_block1_add[0][0]']
 on)

 conv4_block2_1_conv (Conv2  (None, 14, 14, 256)          262400
['conv4_block1_out[0][0]']
 D)

 conv4_block2_1_bn (BatchNo  (None, 14, 14, 256)          1024
```

```
['conv4_block2_1_conv[0][0]']
 rmalization)

 conv4_block2_1_relu (Activ  (None, 14, 14, 256)          0
['conv4_block2_1_bn[0][0]']
 ation)

 conv4_block2_2_conv (Conv2  (None, 14, 14, 256)          590080
['conv4_block2_1_relu[0][0]']
 D)

 conv4_block2_2_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block2_2_conv[0][0]']
 rmalization)

 conv4_block2_2_relu (Activ  (None, 14, 14, 256)          0
['conv4_block2_2_bn[0][0]']
 ation)

 conv4_block2_3_conv (Conv2  (None, 14, 14, 1024)         263168
['conv4_block2_2_relu[0][0]']
 D)

 conv4_block2_3_bn (BatchNo  (None, 14, 14, 1024)         4096
['conv4_block2_3_conv[0][0]']
 rmalization)

 conv4_block2_add (Add)      (None, 14, 14, 1024)         0
['conv4_block1_out[0][0]',
'conv4_block2_3_bn[0][0]']

 conv4_block2_out (Activati  (None, 14, 14, 1024)         0
['conv4_block2_add[0][0]']
 on)

 conv4_block3_1_conv (Conv2  (None, 14, 14, 256)          262400
['conv4_block2_out[0][0]']
 D)

 conv4_block3_1_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block3_1_conv[0][0]']
 rmalization)

 conv4_block3_1_relu (Activ  (None, 14, 14, 256)          0
['conv4_block3_1_bn[0][0]']
 ation)

 conv4_block3_2_conv (Conv2  (None, 14, 14, 256)          590080
```

```
['conv4_block3_1_relu[0][0]']
 D)

 conv4_block3_2_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block3_2_conv[0][0]']
 rmalization)

 conv4_block3_2_relu (Activ  (None, 14, 14, 256)          0
['conv4_block3_2_bn[0][0]']
 ation)

 conv4_block3_3_conv (Conv2  (None, 14, 14, 1024)         263168
['conv4_block3_2_relu[0][0]']
 D)

 conv4_block3_3_bn (BatchNo  (None, 14, 14, 1024)         4096
['conv4_block3_3_conv[0][0]']
 rmalization)

 conv4_block3_add (Add)      (None, 14, 14, 1024)         0
['conv4_block2_out[0][0]',
'conv4_block3_3_bn[0][0]']

 conv4_block3_out (Activati  (None, 14, 14, 1024)         0
['conv4_block3_add[0][0]']
 on)

 conv4_block4_1_conv (Conv2  (None, 14, 14, 256)          262400
['conv4_block3_out[0][0]']
 D)

 conv4_block4_1_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block4_1_conv[0][0]']
 rmalization)

 conv4_block4_1_relu (Activ  (None, 14, 14, 256)          0
['conv4_block4_1_bn[0][0]']
 ation)

 conv4_block4_2_conv (Conv2  (None, 14, 14, 256)          590080
['conv4_block4_1_relu[0][0]']
 D)

 conv4_block4_2_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block4_2_conv[0][0]']
 rmalization)

 conv4_block4_2_relu (Activ  (None, 14, 14, 256)          0
```

```
['conv4_block4_2_bn[0][0]']
 ation)

 conv4_block4_3_conv (Conv2  (None, 14, 14, 1024)         263168
['conv4_block4_2_relu[0][0]']
 D)

 conv4_block4_3_bn (BatchNo  (None, 14, 14, 1024)         4096
['conv4_block4_3_conv[0][0]']
 rmalization)

 conv4_block4_add (Add)      (None, 14, 14, 1024)         0
['conv4_block3_out[0][0]',
'conv4_block4_3_bn[0][0]']

 conv4_block4_out (Activati  (None, 14, 14, 1024)         0
['conv4_block4_add[0][0]']
 on)

 conv4_block5_1_conv (Conv2  (None, 14, 14, 256)          262400
['conv4_block4_out[0][0]']
 D)

 conv4_block5_1_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block5_1_conv[0][0]']
 rmalization)

 conv4_block5_1_relu (Activ  (None, 14, 14, 256)          0
['conv4_block5_1_bn[0][0]']
 ation)

 conv4_block5_2_conv (Conv2  (None, 14, 14, 256)          590080
['conv4_block5_1_relu[0][0]']
 D)

 conv4_block5_2_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block5_2_conv[0][0]']
 rmalization)

 conv4_block5_2_relu (Activ  (None, 14, 14, 256)          0
['conv4_block5_2_bn[0][0]']
 ation)

 conv4_block5_3_conv (Conv2  (None, 14, 14, 1024)         263168
['conv4_block5_2_relu[0][0]']
 D)

 conv4_block5_3_bn (BatchNo  (None, 14, 14, 1024)         4096
```

```
['conv4_block5_3_conv[0][0]']
 rmalization)

 conv4_block5_add (Add)      (None, 14, 14, 1024)         0
['conv4_block4_out[0][0]',
'conv4_block5_3_bn[0][0]']

 conv4_block5_out (Activati  (None, 14, 14, 1024)         0
['conv4_block5_add[0][0]']
 on)

 conv4_block6_1_conv (Conv2  (None, 14, 14, 256)          262400
['conv4_block5_out[0][0]']
 D)

 conv4_block6_1_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block6_1_conv[0][0]']
 rmalization)

 conv4_block6_1_relu (Activ  (None, 14, 14, 256)          0
['conv4_block6_1_bn[0][0]']
 ation)

 conv4_block6_2_conv (Conv2  (None, 14, 14, 256)          590080
['conv4_block6_1_relu[0][0]']
 D)

 conv4_block6_2_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block6_2_conv[0][0]']
 rmalization)

 conv4_block6_2_relu (Activ  (None, 14, 14, 256)          0
['conv4_block6_2_bn[0][0]']
 ation)

 conv4_block6_3_conv (Conv2  (None, 14, 14, 1024)         263168
['conv4_block6_2_relu[0][0]']
 D)

 conv4_block6_3_bn (BatchNo  (None, 14, 14, 1024)         4096
['conv4_block6_3_conv[0][0]']
 rmalization)

 conv4_block6_add (Add)      (None, 14, 14, 1024)         0
['conv4_block5_out[0][0]',
'conv4_block6_3_bn[0][0]']

 conv4_block6_out (Activati  (None, 14, 14, 1024)         0
```

```
['conv4_block6_add[0][0]']
 on)

 conv5_block1_1_conv (Conv2  (None, 7, 7, 512)           524800
['conv4_block6_out[0][0]']
 D)

 conv5_block1_1_bn (BatchNo  (None, 7, 7, 512)           2048
['conv5_block1_1_conv[0][0]']
 rmalization)

 conv5_block1_1_relu (Activ  (None, 7, 7, 512)           0
['conv5_block1_1_bn[0][0]']
 ation)

 conv5_block1_2_conv (Conv2  (None, 7, 7, 512)           2359808
['conv5_block1_1_relu[0][0]']
 D)

 conv5_block1_2_bn (BatchNo  (None, 7, 7, 512)           2048
['conv5_block1_2_conv[0][0]']
 rmalization)

 conv5_block1_2_relu (Activ  (None, 7, 7, 512)           0
['conv5_block1_2_bn[0][0]']
 ation)

 conv5_block1_0_conv (Conv2  (None, 7, 7, 2048)          2099200
['conv4_block6_out[0][0]']
 D)

 conv5_block1_3_conv (Conv2  (None, 7, 7, 2048)          1050624
['conv5_block1_2_relu[0][0]']
 D)

 conv5_block1_0_bn (BatchNo  (None, 7, 7, 2048)          8192
['conv5_block1_0_conv[0][0]']
 rmalization)

 conv5_block1_3_bn (BatchNo  (None, 7, 7, 2048)          8192
['conv5_block1_3_conv[0][0]']
 rmalization)

 conv5_block1_add (Add)      (None, 7, 7, 2048)          0
['conv5_block1_0_bn[0][0]',
'conv5_block1_3_bn[0][0]']

 conv5_block1_out (Activati  (None, 7, 7, 2048)          0
```

```
['conv5_block1_add[0][0]']
 on)

 conv5_block2_1_conv (Conv2  (None, 7, 7, 512)           1049088
['conv5_block1_out[0][0]']
 D)

 conv5_block2_1_bn (BatchNo  (None, 7, 7, 512)           2048
['conv5_block2_1_conv[0][0]']
 rmalization)

 conv5_block2_1_relu (Activ  (None, 7, 7, 512)           0
['conv5_block2_1_bn[0][0]']
 ation)

 conv5_block2_2_conv (Conv2  (None, 7, 7, 512)           2359808
['conv5_block2_1_relu[0][0]']
 D)

 conv5_block2_2_bn (BatchNo  (None, 7, 7, 512)           2048
['conv5_block2_2_conv[0][0]']
 rmalization)

 conv5_block2_2_relu (Activ  (None, 7, 7, 512)           0
['conv5_block2_2_bn[0][0]']
 ation)

 conv5_block2_3_conv (Conv2  (None, 7, 7, 2048)          1050624
['conv5_block2_2_relu[0][0]']
 D)

 conv5_block2_3_bn (BatchNo  (None, 7, 7, 2048)          8192
['conv5_block2_3_conv[0][0]']
 rmalization)

 conv5_block2_add (Add)      (None, 7, 7, 2048)          0
['conv5_block1_out[0][0]',
'conv5_block2_3_bn[0][0]']

 conv5_block2_out (Activati  (None, 7, 7, 2048)          0
['conv5_block2_add[0][0]']
 on)

 conv5_block3_1_conv (Conv2  (None, 7, 7, 512)           1049088
['conv5_block2_out[0][0]']
 D)

 conv5_block3_1_bn (BatchNo  (None, 7, 7, 512)           2048
```

```
['conv5_block3_1_conv[0][0]']
 rmalization)

 conv5_block3_1_relu (Activ  (None, 7, 7, 512)           0
['conv5_block3_1_bn[0][0]']
 ation)

 conv5_block3_2_conv (Conv2  (None, 7, 7, 512)           2359808
['conv5_block3_1_relu[0][0]']
 D)

 conv5_block3_2_bn (BatchNo  (None, 7, 7, 512)           2048
['conv5_block3_2_conv[0][0]']
 rmalization)

 conv5_block3_2_relu (Activ  (None, 7, 7, 512)           0
['conv5_block3_2_bn[0][0]']
 ation)

 conv5_block3_3_conv (Conv2  (None, 7, 7, 2048)          1050624
['conv5_block3_2_relu[0][0]']
 D)

 conv5_block3_3_bn (BatchNo  (None, 7, 7, 2048)          8192
['conv5_block3_3_conv[0][0]']
 rmalization)

 conv5_block3_add (Add)      (None, 7, 7, 2048)          0
['conv5_block2_out[0][0]',
'conv5_block3_3_bn[0][0]']

 conv5_block3_out (Activati  (None, 7, 7, 2048)          0
['conv5_block3_add[0][0]']
 on)

==================================================================================
==================
Total params: 23587712 (89.98 MB)
Trainable params: 23534592 (89.78 MB)
Non-trainable params: 53120 (207.50 KB)
----------------------------------------------------------------------------------
------------------
```

6.2.3 Definir Top Model para Transfer Learning

La definición del **Top Model** para transfer learning con la red ResNet50 como modelo base sigue un enfoque similar al empleado con VGG16, adaptando el modelo preentrenado a una tarea específica mediante la adición de un nuevo clasificador. Aquí se detallan los pasos para esta configuración:

1. **Congelar los Pesos del Base Model**

- `base_model.trainable = False`: Esta línea es crucial porque impide que los **pesos** del modelo base ResNet50 se actualicen durante el entrenamiento.

2. **Crear el Top Model**

- Se inicia un modelo secuencial (`Sequential()`) apilar fácilmente las capas necesarias para el clasificador.
- `pre_trained_model.add(base_model)`: Se añade el modelo base **ResNet50** (con los pesos congelados) al modelo secuencial. Este actúa como un potente extractor de características debido a su entrenamiento previo en el extenso conjunto de datos de ImageNet.
- `pre_trained_model.add(layers.Flatten())`: Después de las capas convolucionales y de pooling del modelo base, la **salida se aplana** para convertirla en un vector. Esto es necesario para pasar de la representación espacial de las características a una forma que se pueda alimentar a capas densas.
- `pre_trained_model.add(layers.Dense(256, activation='relu'))`: Se añaden dos capas densas: una con **256 unidades** y función de activación **ReLU**. Esta capa densa permite aprender combinaciones no lineales de las características extraídas.
- `pre_trained_model.add(layers.Dense(15, activation='softmax'))`: La función de activación `softmax` en la última capa densa es crucial para la clasificación multiclase, ya que produce un vector de probabilidades que suman 1, donde cada entrada del vector representa la probabilidad de que la imagen de entrada pertenezca a una de las clases.

[ ]:
```
# conectarlo a nueva parte densa
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers

base_model.trainable = False # Evitar que los pesos se modifiquen en la parte␣
 ↪convolucional -> TRANSFER LEARNING
pre_trained_model = Sequential()
pre_trained_model.add(base_model)
pre_trained_model.add(layers.Flatten())
pre_trained_model.add(layers.Dense(256, activation='relu'))
pre_trained_model.add(layers.Dense(15, activation='softmax'))
```

```
pre_trained_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 7, 7, 2048)        23587712

 flatten (Flatten)           (None, 100352)            0

 dense (Dense)               (None, 256)               25690368

 dense_1 (Dense)             (None, 15)                3855

=================================================================
Total params: 49281935 (188.00 MB)
Trainable params: 25694223 (98.02 MB)
Non-trainable params: 23587712 (89.98 MB)
_________________________________________________________________
```

6.2.4 Compilación del modelo aplicando Early Stopping

[ ]:
```
# Import the necessary packages
import numpy as np
from tensorflow.keras import backend as K
from tensorflow.keras.layers import Input, Conv2D, Activation, Flatten, Dense,␣
 ↪Dropout, BatchNormalization, MaxPooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import SGD, Adam
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt
from google.colab import drive
```

- **Compilación**

[ ]:
```
# Compilar el modelo
print("[INFO]: Compilando el modelo...")
```

```
pre_trained_model.compile(loss="categorical_crossentropy",␣
 ↪optimizer=Adam(learning_rate=0.0005, weight_decay=0, beta_1=0.9, beta_2=0.
 ↪999, epsilon=1e-08), metrics=["accuracy"])
```

```
[INFO]: Compilando el modelo…
```

- **Callbacks**

Se definen 2 **Callbacks**:

1. **EarlyStopping**:

- Este callback monitorea la **`val_loss`** (pérdida de validación) y detiene el entrenamiento si no se observa una mejora después de un número especificado de épocas (**`patience=3`**), indicando que el modelo ha dejado de mejorar y previniendo el sobreajuste.

2. **TensorBoard**:

- Proporciona una visualización poderosa del proceso de entrenamiento. **`log_dir='./logs'`** especifica dónde guardar los logs que TensorBoard usará para generar las visualizaciones.

[ ]:
```
# Callbacks
my_callbacks = [
    tf.keras.callbacks.EarlyStopping(monitor='val_loss', #Después de algunas␣
 ↪épocas el modelo no mejora
                                     patience=3,
                                     mode='min'),
    tf.keras.callbacks.TensorBoard(log_dir='./logs')
]
```

- **Entrenamiento**

[ ]:
```
# Entrenamiento de la red
print("[INFO]: Entrenando la red...")
H_pre = pre_trained_model.fit(norm_train, batch_size=128, epochs=20,␣
 ↪validation_data=norm_val, callbacks=my_callbacks)
```

```
[INFO]: Entrenando la red…
Epoch 1/20
```

```
469/469 [==============================] - 77s 147ms/step - loss: 0.3157 -
accuracy: 0.9691 - val_loss: 0.0406 - val_accuracy: 0.9940
Epoch 2/20
469/469 [==============================] - 67s 141ms/step - loss: 0.0319 -
accuracy: 0.9958 - val_loss: 0.1009 - val_accuracy: 0.9893
Epoch 3/20
469/469 [==============================] - 67s 141ms/step - loss: 0.0554 -
accuracy: 0.9945 - val_loss: 0.0273 - val_accuracy: 0.9973
Epoch 4/20
469/469 [==============================] - 66s 139ms/step - loss: 0.0518 -
accuracy: 0.9953 - val_loss: 0.1262 - val_accuracy: 0.9950
Epoch 5/20
469/469 [==============================] - 65s 139ms/step - loss: 0.0526 -
accuracy: 0.9965 - val_loss: 0.0159 - val_accuracy: 0.9983
Epoch 6/20
469/469 [==============================] - 68s 145ms/step - loss: 0.0271 -
accuracy: 0.9979 - val_loss: 0.1201 - val_accuracy: 0.9897
Epoch 7/20
469/469 [==============================] - 66s 140ms/step - loss: 0.0244 -
accuracy: 0.9983 - val_loss: 0.0568 - val_accuracy: 0.9970
Epoch 8/20
469/469 [==============================] - 75s 159ms/step - loss: 0.0314 -
accuracy: 0.9978 - val_loss: 0.0861 - val_accuracy: 0.9963
```

6.2.5 Evaluación del modelo de Transfer Learning

Los resultados de este modelo son muy similares a los obtenidos con el Transfer Learning del VGG16 y con menor overfitting. Tiene f1-score perfecto para seis vegetales (brócoli, pimiento, zanahora,patata, rábano y tomate).

- **Clases**

[ ]:
```
# clases
class_names = test_ds.class_names
class_names
```

[ ]:
```
['Bean',
 'Bitter_Gourd',
 'Bottle_Gourd',
 'Brinjal',
 'Broccoli',
 'Cabbage',
 'Capsicum',
```

```
 'Carrot',
 'Cauliflower',
 'Cucumber',
 'Papaya',
 'Potato',
 'Pumpkin',
 'Radish',
 'Tomato']
```

- **Evaluación**

[ ]:
```
# Evaluación del modelo
print("[INFO]: Evaluando el modelo...")

# Efectuamos la predicción (empleamos el mismo valor de batch_size que en␣
 ↪training)
predictions = pre_trained_model.predict(x_test, batch_size=128)

# Sacamos el report para test
print(classification_report(y_test, predictions.argmax(axis=1),␣
 ↪target_names=class_names))
```

```
[INFO]: Evaluando el modelo…
24/24 [==============================] - 19s 494ms/step
              precision    recall  f1-score   support

        Bean       1.00      1.00      1.00       200
Bitter_Gourd       1.00      0.98      0.99       200
Bottle_Gourd       1.00      1.00      1.00       200
     Brinjal       0.99      0.98      0.99       200
    Broccoli       1.00      1.00      1.00       200
     Cabbage       1.00      1.00      1.00       200
    Capsicum       1.00      0.99      1.00       200
      Carrot       1.00      1.00      1.00       200
 Cauliflower       1.00      1.00      1.00       200
    Cucumber       1.00      0.98      0.99       200
      Papaya       0.96      1.00      0.98       200
      Potato       1.00      0.98      0.99       200
     Pumpkin       0.97      1.00      0.98       200
      Radish       1.00      1.00      1.00       200
      Tomato       1.00      0.98      0.99       200

    accuracy                           0.99      3000
   macro avg       0.99      0.99      0.99      3000
weighted avg       0.99      0.99      0.99      3000
```

- **Visualización del Desempeño del Modelo**

[ ]:
```
# Gráfica de accuracy y losses
plt.style.use("ggplot")
plt.figure()

# Usar la longitud de H.history["loss"] para el rango de x
epochs_range = np.arange(len(H_pre.history["loss"])) # Por si cambian las␣
 ↪Epochs del entrenamiento

plt.plot(epochs_range, H_pre.history["loss"], label="train_loss")
plt.plot(epochs_range, H_pre.history["val_loss"], label="val_loss")
plt.plot(epochs_range, H_pre.history["accuracy"], label="train_acc")
plt.plot(epochs_range, H_pre.history["val_accuracy"], label="val_acc")

plt.title("ResNet50: Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()

plt.show()
```

6.2.6 Aplicación de Fine Tuning y Dropout para Reducir Overfitting

En esta etapa de Fine Tuning se congela el modelo hasta la capa `conv4_block6_out` y se redefine el top model para añadir una capa oculta adicional de **1024 neuronas** con un `dropout` del 30%.

[ ]:
```
# Imports que vamos a necesitar

from tensorflow.keras.applications import ResNet50, imagenet_utils
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import optimizers
from tensorflow.keras.layers import Dropout, Flatten, Dense
from tensorflow.keras import Model
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
```

```python
import numpy as np

####### BASE MODEL ######################
# Cargamos ResNet50 con pesos de imagenet y sin top_model especificando tamaño␣
 ↪de entrada de datos
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224,␣
 ↪224, 3))

# Mostramos la arquitectura
base_model.summary()
```

```
Model: "resnet50"
__________________________________________________________________________________
__________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================
==================
 input_3 (InputLayer)        [(None, 224, 224, 3)]        0         []

 conv1_pad (ZeroPadding2D)   (None, 230, 230, 3)          0
['input_3[0][0]']

 conv1_conv (Conv2D)         (None, 112, 112, 64)         9472
['conv1_pad[0][0]']

 conv1_bn (BatchNormalizati  (None, 112, 112, 64)         256
['conv1_conv[0][0]']
 on)

 conv1_relu (Activation)     (None, 112, 112, 64)         0
['conv1_bn[0][0]']

 pool1_pad (ZeroPadding2D)   (None, 114, 114, 64)         0
['conv1_relu[0][0]']

 pool1_pool (MaxPooling2D)   (None, 56, 56, 64)           0
['pool1_pad[0][0]']

 conv2_block1_1_conv (Conv2  (None, 56, 56, 64)           4160
['pool1_pool[0][0]']
 D)

 conv2_block1_1_bn (BatchNo  (None, 56, 56, 64)           256
['conv2_block1_1_conv[0][0]']
 rmalization)
```

```
 conv2_block1_1_relu (Activ  (None, 56, 56, 64)           0
['conv2_block1_1_bn[0][0]']
 ation)

 conv2_block1_2_conv (Conv2  (None, 56, 56, 64)           36928
['conv2_block1_1_relu[0][0]']
 D)

 conv2_block1_2_bn (BatchNo  (None, 56, 56, 64)           256
['conv2_block1_2_conv[0][0]']
 rmalization)

 conv2_block1_2_relu (Activ  (None, 56, 56, 64)           0
['conv2_block1_2_bn[0][0]']
 ation)

 conv2_block1_0_conv (Conv2  (None, 56, 56, 256)          16640
['pool1_pool[0][0]']
 D)

 conv2_block1_3_conv (Conv2  (None, 56, 56, 256)          16640
['conv2_block1_2_relu[0][0]']
 D)

 conv2_block1_0_bn (BatchNo  (None, 56, 56, 256)          1024
['conv2_block1_0_conv[0][0]']
 rmalization)

 conv2_block1_3_bn (BatchNo  (None, 56, 56, 256)          1024
['conv2_block1_3_conv[0][0]']
 rmalization)

 conv2_block1_add (Add)      (None, 56, 56, 256)          0
['conv2_block1_0_bn[0][0]',
'conv2_block1_3_bn[0][0]']

 conv2_block1_out (Activati  (None, 56, 56, 256)          0
['conv2_block1_add[0][0]']
 on)

 conv2_block2_1_conv (Conv2  (None, 56, 56, 64)           16448
['conv2_block1_out[0][0]']
 D)

 conv2_block2_1_bn (BatchNo  (None, 56, 56, 64)           256
['conv2_block2_1_conv[0][0]']
 rmalization)
```

```
 conv2_block2_1_relu (Activ  (None, 56, 56, 64)           0
['conv2_block2_1_bn[0][0]']
 ation)

 conv2_block2_2_conv (Conv2  (None, 56, 56, 64)           36928
['conv2_block2_1_relu[0][0]']
 D)

 conv2_block2_2_bn (BatchNo  (None, 56, 56, 64)           256
['conv2_block2_2_conv[0][0]']
 rmalization)

 conv2_block2_2_relu (Activ  (None, 56, 56, 64)           0
['conv2_block2_2_bn[0][0]']
 ation)

 conv2_block2_3_conv (Conv2  (None, 56, 56, 256)          16640
['conv2_block2_2_relu[0][0]']
 D)

 conv2_block2_3_bn (BatchNo  (None, 56, 56, 256)          1024
['conv2_block2_3_conv[0][0]']
 rmalization)

 conv2_block2_add (Add)      (None, 56, 56, 256)          0
['conv2_block1_out[0][0]',
'conv2_block2_3_bn[0][0]']

 conv2_block2_out (Activati  (None, 56, 56, 256)          0
['conv2_block2_add[0][0]']
 on)

 conv2_block3_1_conv (Conv2  (None, 56, 56, 64)           16448
['conv2_block2_out[0][0]']
 D)

 conv2_block3_1_bn (BatchNo  (None, 56, 56, 64)           256
['conv2_block3_1_conv[0][0]']
 rmalization)

 conv2_block3_1_relu (Activ  (None, 56, 56, 64)           0
['conv2_block3_1_bn[0][0]']
 ation)

 conv2_block3_2_conv (Conv2  (None, 56, 56, 64)           36928
['conv2_block3_1_relu[0][0]']
 D)
```

```
 conv2_block3_2_bn (BatchNo  (None, 56, 56, 64)           256
['conv2_block3_2_conv[0][0]']
 rmalization)

 conv2_block3_2_relu (Activ  (None, 56, 56, 64)           0
['conv2_block3_2_bn[0][0]']
 ation)

 conv2_block3_3_conv (Conv2  (None, 56, 56, 256)          16640
['conv2_block3_2_relu[0][0]']
 D)

 conv2_block3_3_bn (BatchNo  (None, 56, 56, 256)          1024
['conv2_block3_3_conv[0][0]']
 rmalization)

 conv2_block3_add (Add)      (None, 56, 56, 256)          0
['conv2_block2_out[0][0]',
'conv2_block3_3_bn[0][0]']

 conv2_block3_out (Activati  (None, 56, 56, 256)          0
['conv2_block3_add[0][0]']
 on)

 conv3_block1_1_conv (Conv2  (None, 28, 28, 128)          32896
['conv2_block3_out[0][0]']
 D)

 conv3_block1_1_bn (BatchNo  (None, 28, 28, 128)          512
['conv3_block1_1_conv[0][0]']
 rmalization)

 conv3_block1_1_relu (Activ  (None, 28, 28, 128)          0
['conv3_block1_1_bn[0][0]']
 ation)

 conv3_block1_2_conv (Conv2  (None, 28, 28, 128)          147584
['conv3_block1_1_relu[0][0]']
 D)

 conv3_block1_2_bn (BatchNo  (None, 28, 28, 128)          512
['conv3_block1_2_conv[0][0]']
 rmalization)

 conv3_block1_2_relu (Activ  (None, 28, 28, 128)          0
['conv3_block1_2_bn[0][0]']
 ation)
```

```
 conv3_block1_0_conv (Conv2  (None, 28, 28, 512)          131584
['conv2_block3_out[0][0]']
 D)

 conv3_block1_3_conv (Conv2  (None, 28, 28, 512)          66048
['conv3_block1_2_relu[0][0]']
 D)

 conv3_block1_0_bn (BatchNo  (None, 28, 28, 512)          2048
['conv3_block1_0_conv[0][0]']
 rmalization)

 conv3_block1_3_bn (BatchNo  (None, 28, 28, 512)          2048
['conv3_block1_3_conv[0][0]']
 rmalization)

 conv3_block1_add (Add)      (None, 28, 28, 512)          0
['conv3_block1_0_bn[0][0]',
'conv3_block1_3_bn[0][0]']

 conv3_block1_out (Activati  (None, 28, 28, 512)          0
['conv3_block1_add[0][0]']
 on)

 conv3_block2_1_conv (Conv2  (None, 28, 28, 128)          65664
['conv3_block1_out[0][0]']
 D)

 conv3_block2_1_bn (BatchNo  (None, 28, 28, 128)          512
['conv3_block2_1_conv[0][0]']
 rmalization)

 conv3_block2_1_relu (Activ  (None, 28, 28, 128)          0
['conv3_block2_1_bn[0][0]']
 ation)

 conv3_block2_2_conv (Conv2  (None, 28, 28, 128)          147584
['conv3_block2_1_relu[0][0]']
 D)

 conv3_block2_2_bn (BatchNo  (None, 28, 28, 128)          512
['conv3_block2_2_conv[0][0]']
 rmalization)

 conv3_block2_2_relu (Activ  (None, 28, 28, 128)          0
['conv3_block2_2_bn[0][0]']
 ation)
```

```
 conv3_block2_3_conv (Conv2  (None, 28, 28, 512)          66048
['conv3_block2_2_relu[0][0]']
 D)

 conv3_block2_3_bn (BatchNo  (None, 28, 28, 512)          2048
['conv3_block2_3_conv[0][0]']
 rmalization)

 conv3_block2_add (Add)      (None, 28, 28, 512)          0
['conv3_block1_out[0][0]',
'conv3_block2_3_bn[0][0]']

 conv3_block2_out (Activati  (None, 28, 28, 512)          0
['conv3_block2_add[0][0]']
 on)

 conv3_block3_1_conv (Conv2  (None, 28, 28, 128)          65664
['conv3_block2_out[0][0]']
 D)

 conv3_block3_1_bn (BatchNo  (None, 28, 28, 128)          512
['conv3_block3_1_conv[0][0]']
 rmalization)

 conv3_block3_1_relu (Activ  (None, 28, 28, 128)          0
['conv3_block3_1_bn[0][0]']
 ation)

 conv3_block3_2_conv (Conv2  (None, 28, 28, 128)          147584
['conv3_block3_1_relu[0][0]']
 D)

 conv3_block3_2_bn (BatchNo  (None, 28, 28, 128)          512
['conv3_block3_2_conv[0][0]']
 rmalization)

 conv3_block3_2_relu (Activ  (None, 28, 28, 128)          0
['conv3_block3_2_bn[0][0]']
 ation)

 conv3_block3_3_conv (Conv2  (None, 28, 28, 512)          66048
['conv3_block3_2_relu[0][0]']
 D)

 conv3_block3_3_bn (BatchNo  (None, 28, 28, 512)          2048
['conv3_block3_3_conv[0][0]']
 rmalization)
```

```
 conv3_block3_add (Add)      (None, 28, 28, 512)          0
['conv3_block2_out[0][0]',
'conv3_block3_3_bn[0][0]']

 conv3_block3_out (Activati  (None, 28, 28, 512)          0
['conv3_block3_add[0][0]']
 on)

 conv3_block4_1_conv (Conv2  (None, 28, 28, 128)          65664
['conv3_block3_out[0][0]']
 D)

 conv3_block4_1_bn (BatchNo  (None, 28, 28, 128)          512
['conv3_block4_1_conv[0][0]']
 rmalization)

 conv3_block4_1_relu (Activ  (None, 28, 28, 128)          0
['conv3_block4_1_bn[0][0]']
 ation)

 conv3_block4_2_conv (Conv2  (None, 28, 28, 128)          147584
['conv3_block4_1_relu[0][0]']
 D)

 conv3_block4_2_bn (BatchNo  (None, 28, 28, 128)          512
['conv3_block4_2_conv[0][0]']
 rmalization)

 conv3_block4_2_relu (Activ  (None, 28, 28, 128)          0
['conv3_block4_2_bn[0][0]']
 ation)

 conv3_block4_3_conv (Conv2  (None, 28, 28, 512)          66048
['conv3_block4_2_relu[0][0]']
 D)

 conv3_block4_3_bn (BatchNo  (None, 28, 28, 512)          2048
['conv3_block4_3_conv[0][0]']
 rmalization)

 conv3_block4_add (Add)      (None, 28, 28, 512)          0
['conv3_block3_out[0][0]',
'conv3_block4_3_bn[0][0]']

 conv3_block4_out (Activati  (None, 28, 28, 512)          0
['conv3_block4_add[0][0]']
 on)
```

```
 conv4_block1_1_conv (Conv2  (None, 14, 14, 256)          131328
['conv3_block4_out[0][0]']
 D)

 conv4_block1_1_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block1_1_conv[0][0]']
 rmalization)

 conv4_block1_1_relu (Activ  (None, 14, 14, 256)          0
['conv4_block1_1_bn[0][0]']
 ation)

 conv4_block1_2_conv (Conv2  (None, 14, 14, 256)          590080
['conv4_block1_1_relu[0][0]']
 D)

 conv4_block1_2_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block1_2_conv[0][0]']
 rmalization)

 conv4_block1_2_relu (Activ  (None, 14, 14, 256)          0
['conv4_block1_2_bn[0][0]']
 ation)

 conv4_block1_0_conv (Conv2  (None, 14, 14, 1024)         525312
['conv3_block4_out[0][0]']
 D)

 conv4_block1_3_conv (Conv2  (None, 14, 14, 1024)         263168
['conv4_block1_2_relu[0][0]']
 D)

 conv4_block1_0_bn (BatchNo  (None, 14, 14, 1024)         4096
['conv4_block1_0_conv[0][0]']
 rmalization)

 conv4_block1_3_bn (BatchNo  (None, 14, 14, 1024)         4096
['conv4_block1_3_conv[0][0]']
 rmalization)

 conv4_block1_add (Add)      (None, 14, 14, 1024)         0
['conv4_block1_0_bn[0][0]',
'conv4_block1_3_bn[0][0]']

 conv4_block1_out (Activati  (None, 14, 14, 1024)         0
['conv4_block1_add[0][0]']
 on)
```

```
 conv4_block2_1_conv (Conv2  (None, 14, 14, 256)          262400
['conv4_block1_out[0][0]']
 D)

 conv4_block2_1_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block2_1_conv[0][0]']
 rmalization)

 conv4_block2_1_relu (Activ  (None, 14, 14, 256)          0
['conv4_block2_1_bn[0][0]']
 ation)

 conv4_block2_2_conv (Conv2  (None, 14, 14, 256)          590080
['conv4_block2_1_relu[0][0]']
 D)

 conv4_block2_2_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block2_2_conv[0][0]']
 rmalization)

 conv4_block2_2_relu (Activ  (None, 14, 14, 256)          0
['conv4_block2_2_bn[0][0]']
 ation)

 conv4_block2_3_conv (Conv2  (None, 14, 14, 1024)         263168
['conv4_block2_2_relu[0][0]']
 D)

 conv4_block2_3_bn (BatchNo  (None, 14, 14, 1024)         4096
['conv4_block2_3_conv[0][0]']
 rmalization)

 conv4_block2_add (Add)      (None, 14, 14, 1024)         0
['conv4_block1_out[0][0]',
'conv4_block2_3_bn[0][0]']

 conv4_block2_out (Activati  (None, 14, 14, 1024)         0
['conv4_block2_add[0][0]']
 on)

 conv4_block3_1_conv (Conv2  (None, 14, 14, 256)          262400
['conv4_block2_out[0][0]']
 D)

 conv4_block3_1_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block3_1_conv[0][0]']
 rmalization)
```

```
 conv4_block3_1_relu (Activ  (None, 14, 14, 256)          0
['conv4_block3_1_bn[0][0]']
 ation)

 conv4_block3_2_conv (Conv2  (None, 14, 14, 256)          590080
['conv4_block3_1_relu[0][0]']
 D)

 conv4_block3_2_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block3_2_conv[0][0]']
 rmalization)

 conv4_block3_2_relu (Activ  (None, 14, 14, 256)          0
['conv4_block3_2_bn[0][0]']
 ation)

 conv4_block3_3_conv (Conv2  (None, 14, 14, 1024)         263168
['conv4_block3_2_relu[0][0]']
 D)

 conv4_block3_3_bn (BatchNo  (None, 14, 14, 1024)         4096
['conv4_block3_3_conv[0][0]']
 rmalization)

 conv4_block3_add (Add)      (None, 14, 14, 1024)         0
['conv4_block2_out[0][0]',
'conv4_block3_3_bn[0][0]']

 conv4_block3_out (Activati  (None, 14, 14, 1024)         0
['conv4_block3_add[0][0]']
 on)

 conv4_block4_1_conv (Conv2  (None, 14, 14, 256)          262400
['conv4_block3_out[0][0]']
 D)

 conv4_block4_1_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block4_1_conv[0][0]']
 rmalization)

 conv4_block4_1_relu (Activ  (None, 14, 14, 256)          0
['conv4_block4_1_bn[0][0]']
 ation)

 conv4_block4_2_conv (Conv2  (None, 14, 14, 256)          590080
['conv4_block4_1_relu[0][0]']
 D)
```

```
 conv4_block4_2_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block4_2_conv[0][0]']
 rmalization)

 conv4_block4_2_relu (Activ  (None, 14, 14, 256)          0
['conv4_block4_2_bn[0][0]']
 ation)

 conv4_block4_3_conv (Conv2  (None, 14, 14, 1024)         263168
['conv4_block4_2_relu[0][0]']
 D)

 conv4_block4_3_bn (BatchNo  (None, 14, 14, 1024)         4096
['conv4_block4_3_conv[0][0]']
 rmalization)

 conv4_block4_add (Add)      (None, 14, 14, 1024)         0
['conv4_block3_out[0][0]',
'conv4_block4_3_bn[0][0]']

 conv4_block4_out (Activati  (None, 14, 14, 1024)         0
['conv4_block4_add[0][0]']
 on)

 conv4_block5_1_conv (Conv2  (None, 14, 14, 256)          262400
['conv4_block4_out[0][0]']
 D)

 conv4_block5_1_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block5_1_conv[0][0]']
 rmalization)

 conv4_block5_1_relu (Activ  (None, 14, 14, 256)          0
['conv4_block5_1_bn[0][0]']
 ation)

 conv4_block5_2_conv (Conv2  (None, 14, 14, 256)          590080
['conv4_block5_1_relu[0][0]']
 D)

 conv4_block5_2_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block5_2_conv[0][0]']
 rmalization)

 conv4_block5_2_relu (Activ  (None, 14, 14, 256)          0
['conv4_block5_2_bn[0][0]']
 ation)
```

```
 conv4_block5_3_conv (Conv2  (None, 14, 14, 1024)         263168
['conv4_block5_2_relu[0][0]']
 D)

 conv4_block5_3_bn (BatchNo  (None, 14, 14, 1024)         4096
['conv4_block5_3_conv[0][0]']
 rmalization)

 conv4_block5_add (Add)      (None, 14, 14, 1024)         0
['conv4_block4_out[0][0]',
'conv4_block5_3_bn[0][0]']

 conv4_block5_out (Activati  (None, 14, 14, 1024)         0
['conv4_block5_add[0][0]']
 on)

 conv4_block6_1_conv (Conv2  (None, 14, 14, 256)          262400
['conv4_block5_out[0][0]']
 D)

 conv4_block6_1_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block6_1_conv[0][0]']
 rmalization)

 conv4_block6_1_relu (Activ  (None, 14, 14, 256)          0
['conv4_block6_1_bn[0][0]']
 ation)

 conv4_block6_2_conv (Conv2  (None, 14, 14, 256)          590080
['conv4_block6_1_relu[0][0]']
 D)

 conv4_block6_2_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block6_2_conv[0][0]']
 rmalization)

 conv4_block6_2_relu (Activ  (None, 14, 14, 256)          0
['conv4_block6_2_bn[0][0]']
 ation)

 conv4_block6_3_conv (Conv2  (None, 14, 14, 1024)         263168
['conv4_block6_2_relu[0][0]']
 D)

 conv4_block6_3_bn (BatchNo  (None, 14, 14, 1024)         4096
['conv4_block6_3_conv[0][0]']
 rmalization)
```

```
 conv4_block6_add (Add)      (None, 14, 14, 1024)         0
['conv4_block5_out[0][0]',
'conv4_block6_3_bn[0][0]']

 conv4_block6_out (Activati  (None, 14, 14, 1024)         0
['conv4_block6_add[0][0]']
 on)

 conv5_block1_1_conv (Conv2  (None, 7, 7, 512)            524800
['conv4_block6_out[0][0]']
 D)

 conv5_block1_1_bn (BatchNo  (None, 7, 7, 512)            2048
['conv5_block1_1_conv[0][0]']
 rmalization)

 conv5_block1_1_relu (Activ  (None, 7, 7, 512)            0
['conv5_block1_1_bn[0][0]']
 ation)

 conv5_block1_2_conv (Conv2  (None, 7, 7, 512)            2359808
['conv5_block1_1_relu[0][0]']
 D)

 conv5_block1_2_bn (BatchNo  (None, 7, 7, 512)            2048
['conv5_block1_2_conv[0][0]']
 rmalization)

 conv5_block1_2_relu (Activ  (None, 7, 7, 512)            0
['conv5_block1_2_bn[0][0]']
 ation)

 conv5_block1_0_conv (Conv2  (None, 7, 7, 2048)           2099200
['conv4_block6_out[0][0]']
 D)

 conv5_block1_3_conv (Conv2  (None, 7, 7, 2048)           1050624
['conv5_block1_2_relu[0][0]']
 D)

 conv5_block1_0_bn (BatchNo  (None, 7, 7, 2048)           8192
['conv5_block1_0_conv[0][0]']
 rmalization)

 conv5_block1_3_bn (BatchNo  (None, 7, 7, 2048)           8192
['conv5_block1_3_conv[0][0]']
 rmalization)
```

```
 conv5_block1_add (Add)      (None, 7, 7, 2048)           0
['conv5_block1_0_bn[0][0]',
'conv5_block1_3_bn[0][0]']

 conv5_block1_out (Activati  (None, 7, 7, 2048)           0
['conv5_block1_add[0][0]']
 on)

 conv5_block2_1_conv (Conv2  (None, 7, 7, 512)            1049088
['conv5_block1_out[0][0]']
 D)

 conv5_block2_1_bn (BatchNo  (None, 7, 7, 512)            2048
['conv5_block2_1_conv[0][0]']
 rmalization)

 conv5_block2_1_relu (Activ  (None, 7, 7, 512)            0
['conv5_block2_1_bn[0][0]']
 ation)

 conv5_block2_2_conv (Conv2  (None, 7, 7, 512)            2359808
['conv5_block2_1_relu[0][0]']
 D)

 conv5_block2_2_bn (BatchNo  (None, 7, 7, 512)            2048
['conv5_block2_2_conv[0][0]']
 rmalization)

 conv5_block2_2_relu (Activ  (None, 7, 7, 512)            0
['conv5_block2_2_bn[0][0]']
 ation)

 conv5_block2_3_conv (Conv2  (None, 7, 7, 2048)           1050624
['conv5_block2_2_relu[0][0]']
 D)

 conv5_block2_3_bn (BatchNo  (None, 7, 7, 2048)           8192
['conv5_block2_3_conv[0][0]']
 rmalization)

 conv5_block2_add (Add)      (None, 7, 7, 2048)           0
['conv5_block1_out[0][0]',
'conv5_block2_3_bn[0][0]']

 conv5_block2_out (Activati  (None, 7, 7, 2048)           0
['conv5_block2_add[0][0]']
 on)
```

```
 conv5_block3_1_conv (Conv2  (None, 7, 7, 512)           1049088
['conv5_block2_out[0][0]']
 D)

 conv5_block3_1_bn (BatchNo  (None, 7, 7, 512)           2048
['conv5_block3_1_conv[0][0]']
 rmalization)

 conv5_block3_1_relu (Activ  (None, 7, 7, 512)           0
['conv5_block3_1_bn[0][0]']
 ation)

 conv5_block3_2_conv (Conv2  (None, 7, 7, 512)           2359808
['conv5_block3_1_relu[0][0]']
 D)

 conv5_block3_2_bn (BatchNo  (None, 7, 7, 512)           2048
['conv5_block3_2_conv[0][0]']
 rmalization)

 conv5_block3_2_relu (Activ  (None, 7, 7, 512)           0
['conv5_block3_2_bn[0][0]']
 ation)

 conv5_block3_3_conv (Conv2  (None, 7, 7, 2048)          1050624
['conv5_block3_2_relu[0][0]']
 D)

 conv5_block3_3_bn (BatchNo  (None, 7, 7, 2048)          8192
['conv5_block3_3_conv[0][0]']
 rmalization)

 conv5_block3_add (Add)      (None, 7, 7, 2048)          0
['conv5_block2_out[0][0]',
'conv5_block3_3_bn[0][0]']

 conv5_block3_out (Activati  (None, 7, 7, 2048)          0
['conv5_block3_add[0][0]']
 on)

==================================================================================
==================
Total params: 23587712 (89.98 MB)
Trainable params: 23534592 (89.78 MB)
Non-trainable params: 53120 (207.50 KB)
----------------------------------------------------------------------------------
------------------
```

- **Congelar Capas**

Algunas consideraciones importantes son:

1. Se iterará sobre las capas del modelo base (ResNet50 en este caso), congelando las capas hasta encontrar una capa específica identificada por su nombre (`conv4_block6_out`). Esto se hace estableciendo `layer.trainable = False` para cada capa hasta el punto de interrupción, lo cual evita que los pesos de estas capas se actualicen durante el entrenamiento.
2. Se utilizará la salida de la última capa del **modelo base** como punto de partida.
3. `Flatten()`: La salida se aplana para convertirla de un tensor multidimensional a un vector unidimensional, lo cual es necesario para conectarla con capas densas.
4. Se añadirán dos capas densas con unidades **1024** y **256**, respectivamente, ambas con activación ReLU. Entre estas dos capas densas, se introduce una capa de Dropout(0.3) para reducir el riesgo de sobreajuste al descartar aleatoriamente un porcentaje de las conexiones entre las capas durante el entrenamiento.
5. La última capa es una densa con **15 unidades** y activación `softmax`, configurada para clasificar las imágenes en una de las 15 categorías.
6. `model = Model(base_model.input, x)`: Finalmente, se creará el modelo completo combinando el modelo base con el nuevo clasificador, especificando las entradas del modelo base y las salidas del clasificador personalizado.

[ ]:
```
# Congelar Capas
for layer in base_model.layers:
  if layer.name == 'conv4_block6_out':
    break
  layer.trainable = False
  print('Capa ' + layer.name + ' congelada...')

# Cogemos la última capa del model y le añadimos nuestro clasificador␣
 ↪(top_model)
last = base_model.layers[-1].output # ultima capa del base model
x = Flatten()(last)
x = Dense(1024, activation='relu', name='fc1')(x)
x = Dropout(0.3)(x)
x = Dense(256, activation='relu', name='fc2')(x)
x = Dense(15, activation='softmax', name='predictions')(x)
model = Model(base_model.input, x)
```

```
Capa input_3 congelada…
Capa conv1_pad congelada…
Capa conv1_conv congelada…
Capa conv1_bn congelada…
```

```
Capa conv1_relu congelada…
Capa pool1_pad congelada…
Capa pool1_pool congelada…
Capa conv2_block1_1_conv congelada…
Capa conv2_block1_1_bn congelada…
Capa conv2_block1_1_relu congelada…
Capa conv2_block1_2_conv congelada…
Capa conv2_block1_2_bn congelada…
Capa conv2_block1_2_relu congelada…
Capa conv2_block1_0_conv congelada…
Capa conv2_block1_3_conv congelada…
Capa conv2_block1_0_bn congelada…
Capa conv2_block1_3_bn congelada…
Capa conv2_block1_add congelada…
Capa conv2_block1_out congelada…
Capa conv2_block2_1_conv congelada…
Capa conv2_block2_1_bn congelada…
Capa conv2_block2_1_relu congelada…
Capa conv2_block2_2_conv congelada…
Capa conv2_block2_2_bn congelada…
Capa conv2_block2_2_relu congelada…
Capa conv2_block2_3_conv congelada…
Capa conv2_block2_3_bn congelada…
Capa conv2_block2_add congelada…
Capa conv2_block2_out congelada…
Capa conv2_block3_1_conv congelada…
Capa conv2_block3_1_bn congelada…
Capa conv2_block3_1_relu congelada…
Capa conv2_block3_2_conv congelada…
Capa conv2_block3_2_bn congelada…
Capa conv2_block3_2_relu congelada…
Capa conv2_block3_3_conv congelada…
Capa conv2_block3_3_bn congelada…
Capa conv2_block3_add congelada…
Capa conv2_block3_out congelada…
Capa conv3_block1_1_conv congelada…
Capa conv3_block1_1_bn congelada…
Capa conv3_block1_1_relu congelada…
Capa conv3_block1_2_conv congelada…
Capa conv3_block1_2_bn congelada…
Capa conv3_block1_2_relu congelada…
Capa conv3_block1_0_conv congelada…
Capa conv3_block1_3_conv congelada…
Capa conv3_block1_0_bn congelada…
Capa conv3_block1_3_bn congelada…
Capa conv3_block1_add congelada…
Capa conv3_block1_out congelada…
Capa conv3_block2_1_conv congelada…
```

```
Capa conv3_block2_1_bn congelada…
Capa conv3_block2_1_relu congelada…
Capa conv3_block2_2_conv congelada…
Capa conv3_block2_2_bn congelada…
Capa conv3_block2_2_relu congelada…
Capa conv3_block2_3_conv congelada…
Capa conv3_block2_3_bn congelada…
Capa conv3_block2_add congelada…
Capa conv3_block2_out congelada…
Capa conv3_block3_1_conv congelada…
Capa conv3_block3_1_bn congelada…
Capa conv3_block3_1_relu congelada…
Capa conv3_block3_2_conv congelada…
Capa conv3_block3_2_bn congelada…
Capa conv3_block3_2_relu congelada…
Capa conv3_block3_3_conv congelada…
Capa conv3_block3_3_bn congelada…
Capa conv3_block3_add congelada…
Capa conv3_block3_out congelada…
Capa conv3_block4_1_conv congelada…
Capa conv3_block4_1_bn congelada…
Capa conv3_block4_1_relu congelada…
Capa conv3_block4_2_conv congelada…
Capa conv3_block4_2_bn congelada…
Capa conv3_block4_2_relu congelada…
Capa conv3_block4_3_conv congelada…
Capa conv3_block4_3_bn congelada…
Capa conv3_block4_add congelada…
Capa conv3_block4_out congelada…
Capa conv4_block1_1_conv congelada…
Capa conv4_block1_1_bn congelada…
Capa conv4_block1_1_relu congelada…
Capa conv4_block1_2_conv congelada…
Capa conv4_block1_2_bn congelada…
Capa conv4_block1_2_relu congelada…
Capa conv4_block1_0_conv congelada…
Capa conv4_block1_3_conv congelada…
Capa conv4_block1_0_bn congelada…
Capa conv4_block1_3_bn congelada…
Capa conv4_block1_add congelada…
Capa conv4_block1_out congelada…
Capa conv4_block2_1_conv congelada…
Capa conv4_block2_1_bn congelada…
Capa conv4_block2_1_relu congelada…
Capa conv4_block2_2_conv congelada…
Capa conv4_block2_2_bn congelada…
Capa conv4_block2_2_relu congelada…
Capa conv4_block2_3_conv congelada…
```

```
Capa conv4_block2_3_bn congelada…
Capa conv4_block2_add congelada…
Capa conv4_block2_out congelada…
Capa conv4_block3_1_conv congelada…
Capa conv4_block3_1_bn congelada…
Capa conv4_block3_1_relu congelada…
Capa conv4_block3_2_conv congelada…
Capa conv4_block3_2_bn congelada…
Capa conv4_block3_2_relu congelada…
Capa conv4_block3_3_conv congelada…
Capa conv4_block3_3_bn congelada…
Capa conv4_block3_add congelada…
Capa conv4_block3_out congelada…
Capa conv4_block4_1_conv congelada…
Capa conv4_block4_1_bn congelada…
Capa conv4_block4_1_relu congelada…
Capa conv4_block4_2_conv congelada…
Capa conv4_block4_2_bn congelada…
Capa conv4_block4_2_relu congelada…
Capa conv4_block4_3_conv congelada…
Capa conv4_block4_3_bn congelada…
Capa conv4_block4_add congelada…
Capa conv4_block4_out congelada…
Capa conv4_block5_1_conv congelada…
Capa conv4_block5_1_bn congelada…
Capa conv4_block5_1_relu congelada…
Capa conv4_block5_2_conv congelada…
Capa conv4_block5_2_bn congelada…
Capa conv4_block5_2_relu congelada…
Capa conv4_block5_3_conv congelada…
Capa conv4_block5_3_bn congelada…
Capa conv4_block5_add congelada…
Capa conv4_block5_out congelada…
Capa conv4_block6_1_conv congelada…
Capa conv4_block6_1_bn congelada…
Capa conv4_block6_1_relu congelada…
Capa conv4_block6_2_conv congelada…
Capa conv4_block6_2_bn congelada…
Capa conv4_block6_2_relu congelada…
Capa conv4_block6_3_conv congelada…
Capa conv4_block6_3_bn congelada…
Capa conv4_block6_add congelada…
```

- **Compilación**

[ ]:
```
# Compilamos el modelo
model.compile(optimizer=Adam(learning_rate=0.0005, weight_decay=0, beta_1=0.9,␣
 ↪beta_2=0.999, epsilon=1e-08), loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

model.summary()
```

```
Model: "model"
__________________________________________________________________________________
__________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================
==================
 input_3 (InputLayer)        [(None, 224, 224, 3)]        0         []

 conv1_pad (ZeroPadding2D)   (None, 230, 230, 3)          0
['input_3[0][0]']

 conv1_conv (Conv2D)         (None, 112, 112, 64)         9472
['conv1_pad[0][0]']

 conv1_bn (BatchNormalizati  (None, 112, 112, 64)         256
['conv1_conv[0][0]']
 on)

 conv1_relu (Activation)     (None, 112, 112, 64)         0
['conv1_bn[0][0]']

 pool1_pad (ZeroPadding2D)   (None, 114, 114, 64)         0
['conv1_relu[0][0]']

 pool1_pool (MaxPooling2D)   (None, 56, 56, 64)           0
['pool1_pad[0][0]']

 conv2_block1_1_conv (Conv2  (None, 56, 56, 64)           4160
['pool1_pool[0][0]']
 D)

 conv2_block1_1_bn (BatchNo  (None, 56, 56, 64)           256
['conv2_block1_1_conv[0][0]']
 rmalization)

 conv2_block1_1_relu (Activ  (None, 56, 56, 64)           0
['conv2_block1_1_bn[0][0]']
 ation)
```

```
 conv2_block1_2_conv (Conv2  (None, 56, 56, 64)           36928
['conv2_block1_1_relu[0][0]']
 D)

 conv2_block1_2_bn (BatchNo  (None, 56, 56, 64)           256
['conv2_block1_2_conv[0][0]']
 rmalization)

 conv2_block1_2_relu (Activ  (None, 56, 56, 64)           0
['conv2_block1_2_bn[0][0]']
 ation)

 conv2_block1_0_conv (Conv2  (None, 56, 56, 256)          16640
['pool1_pool[0][0]']
 D)

 conv2_block1_3_conv (Conv2  (None, 56, 56, 256)          16640
['conv2_block1_2_relu[0][0]']
 D)

 conv2_block1_0_bn (BatchNo  (None, 56, 56, 256)          1024
['conv2_block1_0_conv[0][0]']
 rmalization)

 conv2_block1_3_bn (BatchNo  (None, 56, 56, 256)          1024
['conv2_block1_3_conv[0][0]']
 rmalization)

 conv2_block1_add (Add)      (None, 56, 56, 256)          0
['conv2_block1_0_bn[0][0]',
'conv2_block1_3_bn[0][0]']

 conv2_block1_out (Activati  (None, 56, 56, 256)          0
['conv2_block1_add[0][0]']
 on)

 conv2_block2_1_conv (Conv2  (None, 56, 56, 64)           16448
['conv2_block1_out[0][0]']
 D)

 conv2_block2_1_bn (BatchNo  (None, 56, 56, 64)           256
['conv2_block2_1_conv[0][0]']
 rmalization)

 conv2_block2_1_relu (Activ  (None, 56, 56, 64)           0
['conv2_block2_1_bn[0][0]']
 ation)
```

```
 conv2_block2_2_conv (Conv2  (None, 56, 56, 64)           36928
['conv2_block2_1_relu[0][0]']
 D)

 conv2_block2_2_bn (BatchNo  (None, 56, 56, 64)           256
['conv2_block2_2_conv[0][0]']
 rmalization)

 conv2_block2_2_relu (Activ  (None, 56, 56, 64)           0
['conv2_block2_2_bn[0][0]']
 ation)

 conv2_block2_3_conv (Conv2  (None, 56, 56, 256)          16640
['conv2_block2_2_relu[0][0]']
 D)

 conv2_block2_3_bn (BatchNo  (None, 56, 56, 256)          1024
['conv2_block2_3_conv[0][0]']
 rmalization)

 conv2_block2_add (Add)      (None, 56, 56, 256)          0
['conv2_block1_out[0][0]',
'conv2_block2_3_bn[0][0]']

 conv2_block2_out (Activati  (None, 56, 56, 256)          0
['conv2_block2_add[0][0]']
 on)

 conv2_block3_1_conv (Conv2  (None, 56, 56, 64)           16448
['conv2_block2_out[0][0]']
 D)

 conv2_block3_1_bn (BatchNo  (None, 56, 56, 64)           256
['conv2_block3_1_conv[0][0]']
 rmalization)

 conv2_block3_1_relu (Activ  (None, 56, 56, 64)           0
['conv2_block3_1_bn[0][0]']
 ation)

 conv2_block3_2_conv (Conv2  (None, 56, 56, 64)           36928
['conv2_block3_1_relu[0][0]']
 D)

 conv2_block3_2_bn (BatchNo  (None, 56, 56, 64)           256
['conv2_block3_2_conv[0][0]']
 rmalization)
```

```
 conv2_block3_2_relu (Activ  (None, 56, 56, 64)           0
['conv2_block3_2_bn[0][0]']
 ation)

 conv2_block3_3_conv (Conv2  (None, 56, 56, 256)          16640
['conv2_block3_2_relu[0][0]']
 D)

 conv2_block3_3_bn (BatchNo  (None, 56, 56, 256)          1024
['conv2_block3_3_conv[0][0]']
 rmalization)

 conv2_block3_add (Add)      (None, 56, 56, 256)          0
['conv2_block2_out[0][0]',
'conv2_block3_3_bn[0][0]']

 conv2_block3_out (Activati  (None, 56, 56, 256)          0
['conv2_block3_add[0][0]']
 on)

 conv3_block1_1_conv (Conv2  (None, 28, 28, 128)          32896
['conv2_block3_out[0][0]']
 D)

 conv3_block1_1_bn (BatchNo  (None, 28, 28, 128)          512
['conv3_block1_1_conv[0][0]']
 rmalization)

 conv3_block1_1_relu (Activ  (None, 28, 28, 128)          0
['conv3_block1_1_bn[0][0]']
 ation)

 conv3_block1_2_conv (Conv2  (None, 28, 28, 128)          147584
['conv3_block1_1_relu[0][0]']
 D)

 conv3_block1_2_bn (BatchNo  (None, 28, 28, 128)          512
['conv3_block1_2_conv[0][0]']
 rmalization)

 conv3_block1_2_relu (Activ  (None, 28, 28, 128)          0
['conv3_block1_2_bn[0][0]']
 ation)

 conv3_block1_0_conv (Conv2  (None, 28, 28, 512)          131584
['conv2_block3_out[0][0]']
 D)
```

```
 conv3_block1_3_conv (Conv2  (None, 28, 28, 512)          66048
['conv3_block1_2_relu[0][0]']
 D)

 conv3_block1_0_bn (BatchNo  (None, 28, 28, 512)          2048
['conv3_block1_0_conv[0][0]']
 rmalization)

 conv3_block1_3_bn (BatchNo  (None, 28, 28, 512)          2048
['conv3_block1_3_conv[0][0]']
 rmalization)

 conv3_block1_add (Add)      (None, 28, 28, 512)          0
['conv3_block1_0_bn[0][0]',
'conv3_block1_3_bn[0][0]']

 conv3_block1_out (Activati  (None, 28, 28, 512)          0
['conv3_block1_add[0][0]']
 on)

 conv3_block2_1_conv (Conv2  (None, 28, 28, 128)          65664
['conv3_block1_out[0][0]']
 D)

 conv3_block2_1_bn (BatchNo  (None, 28, 28, 128)          512
['conv3_block2_1_conv[0][0]']
 rmalization)

 conv3_block2_1_relu (Activ  (None, 28, 28, 128)          0
['conv3_block2_1_bn[0][0]']
 ation)

 conv3_block2_2_conv (Conv2  (None, 28, 28, 128)          147584
['conv3_block2_1_relu[0][0]']
 D)

 conv3_block2_2_bn (BatchNo  (None, 28, 28, 128)          512
['conv3_block2_2_conv[0][0]']
 rmalization)

 conv3_block2_2_relu (Activ  (None, 28, 28, 128)          0
['conv3_block2_2_bn[0][0]']
 ation)

 conv3_block2_3_conv (Conv2  (None, 28, 28, 512)          66048
['conv3_block2_2_relu[0][0]']
 D)
```

```
 conv3_block2_3_bn (BatchNo  (None, 28, 28, 512)          2048
['conv3_block2_3_conv[0][0]']
 rmalization)

 conv3_block2_add (Add)      (None, 28, 28, 512)          0
['conv3_block1_out[0][0]',
'conv3_block2_3_bn[0][0]']

 conv3_block2_out (Activati  (None, 28, 28, 512)          0
['conv3_block2_add[0][0]']
 on)

 conv3_block3_1_conv (Conv2  (None, 28, 28, 128)          65664
['conv3_block2_out[0][0]']
 D)

 conv3_block3_1_bn (BatchNo  (None, 28, 28, 128)          512
['conv3_block3_1_conv[0][0]']
 rmalization)

 conv3_block3_1_relu (Activ  (None, 28, 28, 128)          0
['conv3_block3_1_bn[0][0]']
 ation)

 conv3_block3_2_conv (Conv2  (None, 28, 28, 128)          147584
['conv3_block3_1_relu[0][0]']
 D)

 conv3_block3_2_bn (BatchNo  (None, 28, 28, 128)          512
['conv3_block3_2_conv[0][0]']
 rmalization)

 conv3_block3_2_relu (Activ  (None, 28, 28, 128)          0
['conv3_block3_2_bn[0][0]']
 ation)

 conv3_block3_3_conv (Conv2  (None, 28, 28, 512)          66048
['conv3_block3_2_relu[0][0]']
 D)

 conv3_block3_3_bn (BatchNo  (None, 28, 28, 512)          2048
['conv3_block3_3_conv[0][0]']
 rmalization)

 conv3_block3_add (Add)      (None, 28, 28, 512)          0
['conv3_block2_out[0][0]',
'conv3_block3_3_bn[0][0]']
```

```
 conv3_block3_out (Activati  (None, 28, 28, 512)          0
['conv3_block3_add[0][0]']
 on)

 conv3_block4_1_conv (Conv2  (None, 28, 28, 128)          65664
['conv3_block3_out[0][0]']
 D)

 conv3_block4_1_bn (BatchNo  (None, 28, 28, 128)          512
['conv3_block4_1_conv[0][0]']
 rmalization)

 conv3_block4_1_relu (Activ  (None, 28, 28, 128)          0
['conv3_block4_1_bn[0][0]']
 ation)

 conv3_block4_2_conv (Conv2  (None, 28, 28, 128)          147584
['conv3_block4_1_relu[0][0]']
 D)

 conv3_block4_2_bn (BatchNo  (None, 28, 28, 128)          512
['conv3_block4_2_conv[0][0]']
 rmalization)

 conv3_block4_2_relu (Activ  (None, 28, 28, 128)          0
['conv3_block4_2_bn[0][0]']
 ation)

 conv3_block4_3_conv (Conv2  (None, 28, 28, 512)          66048
['conv3_block4_2_relu[0][0]']
 D)

 conv3_block4_3_bn (BatchNo  (None, 28, 28, 512)          2048
['conv3_block4_3_conv[0][0]']
 rmalization)

 conv3_block4_add (Add)      (None, 28, 28, 512)          0
['conv3_block3_out[0][0]',
'conv3_block4_3_bn[0][0]']

 conv3_block4_out (Activati  (None, 28, 28, 512)          0
['conv3_block4_add[0][0]']
 on)

 conv4_block1_1_conv (Conv2  (None, 14, 14, 256)          131328
['conv3_block4_out[0][0]']
 D)
```

```
 conv4_block1_1_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block1_1_conv[0][0]']
 rmalization)

 conv4_block1_1_relu (Activ  (None, 14, 14, 256)          0
['conv4_block1_1_bn[0][0]']
 ation)

 conv4_block1_2_conv (Conv2  (None, 14, 14, 256)          590080
['conv4_block1_1_relu[0][0]']
 D)

 conv4_block1_2_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block1_2_conv[0][0]']
 rmalization)

 conv4_block1_2_relu (Activ  (None, 14, 14, 256)          0
['conv4_block1_2_bn[0][0]']
 ation)

 conv4_block1_0_conv (Conv2  (None, 14, 14, 1024)         525312
['conv3_block4_out[0][0]']
 D)

 conv4_block1_3_conv (Conv2  (None, 14, 14, 1024)         263168
['conv4_block1_2_relu[0][0]']
 D)

 conv4_block1_0_bn (BatchNo  (None, 14, 14, 1024)         4096
['conv4_block1_0_conv[0][0]']
 rmalization)

 conv4_block1_3_bn (BatchNo  (None, 14, 14, 1024)         4096
['conv4_block1_3_conv[0][0]']
 rmalization)

 conv4_block1_add (Add)      (None, 14, 14, 1024)         0
['conv4_block1_0_bn[0][0]',
'conv4_block1_3_bn[0][0]']

 conv4_block1_out (Activati  (None, 14, 14, 1024)         0
['conv4_block1_add[0][0]']
 on)

 conv4_block2_1_conv (Conv2  (None, 14, 14, 256)          262400
['conv4_block1_out[0][0]']
 D)
```

```
 conv4_block2_1_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block2_1_conv[0][0]']
 rmalization)

 conv4_block2_1_relu (Activ  (None, 14, 14, 256)          0
['conv4_block2_1_bn[0][0]']
 ation)

 conv4_block2_2_conv (Conv2  (None, 14, 14, 256)          590080
['conv4_block2_1_relu[0][0]']
 D)

 conv4_block2_2_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block2_2_conv[0][0]']
 rmalization)

 conv4_block2_2_relu (Activ  (None, 14, 14, 256)          0
['conv4_block2_2_bn[0][0]']
 ation)

 conv4_block2_3_conv (Conv2  (None, 14, 14, 1024)         263168
['conv4_block2_2_relu[0][0]']
 D)

 conv4_block2_3_bn (BatchNo  (None, 14, 14, 1024)         4096
['conv4_block2_3_conv[0][0]']
 rmalization)

 conv4_block2_add (Add)      (None, 14, 14, 1024)         0
['conv4_block1_out[0][0]',
'conv4_block2_3_bn[0][0]']

 conv4_block2_out (Activati  (None, 14, 14, 1024)         0
['conv4_block2_add[0][0]']
 on)

 conv4_block3_1_conv (Conv2  (None, 14, 14, 256)          262400
['conv4_block2_out[0][0]']
 D)

 conv4_block3_1_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block3_1_conv[0][0]']
 rmalization)

 conv4_block3_1_relu (Activ  (None, 14, 14, 256)          0
['conv4_block3_1_bn[0][0]']
 ation)
```

```
 conv4_block3_2_conv (Conv2  (None, 14, 14, 256)          590080
['conv4_block3_1_relu[0][0]']
 D)

 conv4_block3_2_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block3_2_conv[0][0]']
 rmalization)

 conv4_block3_2_relu (Activ  (None, 14, 14, 256)          0
['conv4_block3_2_bn[0][0]']
 ation)

 conv4_block3_3_conv (Conv2  (None, 14, 14, 1024)         263168
['conv4_block3_2_relu[0][0]']
 D)

 conv4_block3_3_bn (BatchNo  (None, 14, 14, 1024)         4096
['conv4_block3_3_conv[0][0]']
 rmalization)

 conv4_block3_add (Add)      (None, 14, 14, 1024)         0
['conv4_block2_out[0][0]',
'conv4_block3_3_bn[0][0]']

 conv4_block3_out (Activati  (None, 14, 14, 1024)         0
['conv4_block3_add[0][0]']
 on)

 conv4_block4_1_conv (Conv2  (None, 14, 14, 256)          262400
['conv4_block3_out[0][0]']
 D)

 conv4_block4_1_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block4_1_conv[0][0]']
 rmalization)

 conv4_block4_1_relu (Activ  (None, 14, 14, 256)          0
['conv4_block4_1_bn[0][0]']
 ation)

 conv4_block4_2_conv (Conv2  (None, 14, 14, 256)          590080
['conv4_block4_1_relu[0][0]']
 D)

 conv4_block4_2_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block4_2_conv[0][0]']
 rmalization)
```

```
 conv4_block4_2_relu (Activ  (None, 14, 14, 256)          0
['conv4_block4_2_bn[0][0]']
 ation)

 conv4_block4_3_conv (Conv2  (None, 14, 14, 1024)         263168
['conv4_block4_2_relu[0][0]']
 D)

 conv4_block4_3_bn (BatchNo  (None, 14, 14, 1024)         4096
['conv4_block4_3_conv[0][0]']
 rmalization)

 conv4_block4_add (Add)      (None, 14, 14, 1024)         0
['conv4_block3_out[0][0]',
'conv4_block4_3_bn[0][0]']

 conv4_block4_out (Activati  (None, 14, 14, 1024)         0
['conv4_block4_add[0][0]']
 on)

 conv4_block5_1_conv (Conv2  (None, 14, 14, 256)          262400
['conv4_block4_out[0][0]']
 D)

 conv4_block5_1_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block5_1_conv[0][0]']
 rmalization)

 conv4_block5_1_relu (Activ  (None, 14, 14, 256)          0
['conv4_block5_1_bn[0][0]']
 ation)

 conv4_block5_2_conv (Conv2  (None, 14, 14, 256)          590080
['conv4_block5_1_relu[0][0]']
 D)

 conv4_block5_2_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block5_2_conv[0][0]']
 rmalization)

 conv4_block5_2_relu (Activ  (None, 14, 14, 256)          0
['conv4_block5_2_bn[0][0]']
 ation)

 conv4_block5_3_conv (Conv2  (None, 14, 14, 1024)         263168
['conv4_block5_2_relu[0][0]']
 D)
```

```
 conv4_block5_3_bn (BatchNo  (None, 14, 14, 1024)         4096
['conv4_block5_3_conv[0][0]']
 rmalization)

 conv4_block5_add (Add)      (None, 14, 14, 1024)         0
['conv4_block4_out[0][0]',
'conv4_block5_3_bn[0][0]']

 conv4_block5_out (Activati  (None, 14, 14, 1024)         0
['conv4_block5_add[0][0]']
 on)

 conv4_block6_1_conv (Conv2  (None, 14, 14, 256)          262400
['conv4_block5_out[0][0]']
 D)

 conv4_block6_1_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block6_1_conv[0][0]']
 rmalization)

 conv4_block6_1_relu (Activ  (None, 14, 14, 256)          0
['conv4_block6_1_bn[0][0]']
 ation)

 conv4_block6_2_conv (Conv2  (None, 14, 14, 256)          590080
['conv4_block6_1_relu[0][0]']
 D)

 conv4_block6_2_bn (BatchNo  (None, 14, 14, 256)          1024
['conv4_block6_2_conv[0][0]']
 rmalization)

 conv4_block6_2_relu (Activ  (None, 14, 14, 256)          0
['conv4_block6_2_bn[0][0]']
 ation)

 conv4_block6_3_conv (Conv2  (None, 14, 14, 1024)         263168
['conv4_block6_2_relu[0][0]']
 D)

 conv4_block6_3_bn (BatchNo  (None, 14, 14, 1024)         4096
['conv4_block6_3_conv[0][0]']
 rmalization)

 conv4_block6_add (Add)      (None, 14, 14, 1024)         0
['conv4_block5_out[0][0]',
'conv4_block6_3_bn[0][0]']
```

```
 conv4_block6_out (Activati  (None, 14, 14, 1024)         0
['conv4_block6_add[0][0]']
 on)

 conv5_block1_1_conv (Conv2  (None, 7, 7, 512)            524800
['conv4_block6_out[0][0]']
 D)

 conv5_block1_1_bn (BatchNo  (None, 7, 7, 512)            2048
['conv5_block1_1_conv[0][0]']
 rmalization)

 conv5_block1_1_relu (Activ  (None, 7, 7, 512)            0
['conv5_block1_1_bn[0][0]']
 ation)

 conv5_block1_2_conv (Conv2  (None, 7, 7, 512)            2359808
['conv5_block1_1_relu[0][0]']
 D)

 conv5_block1_2_bn (BatchNo  (None, 7, 7, 512)            2048
['conv5_block1_2_conv[0][0]']
 rmalization)

 conv5_block1_2_relu (Activ  (None, 7, 7, 512)            0
['conv5_block1_2_bn[0][0]']
 ation)

 conv5_block1_0_conv (Conv2  (None, 7, 7, 2048)           2099200
['conv4_block6_out[0][0]']
 D)

 conv5_block1_3_conv (Conv2  (None, 7, 7, 2048)           1050624
['conv5_block1_2_relu[0][0]']
 D)

 conv5_block1_0_bn (BatchNo  (None, 7, 7, 2048)           8192
['conv5_block1_0_conv[0][0]']
 rmalization)

 conv5_block1_3_bn (BatchNo  (None, 7, 7, 2048)           8192
['conv5_block1_3_conv[0][0]']
 rmalization)

 conv5_block1_add (Add)      (None, 7, 7, 2048)           0
['conv5_block1_0_bn[0][0]',
'conv5_block1_3_bn[0][0]']
```

```
 conv5_block1_out (Activati  (None, 7, 7, 2048)           0
['conv5_block1_add[0][0]']
 on)

 conv5_block2_1_conv (Conv2  (None, 7, 7, 512)            1049088
['conv5_block1_out[0][0]']
 D)

 conv5_block2_1_bn (BatchNo  (None, 7, 7, 512)            2048
['conv5_block2_1_conv[0][0]']
 rmalization)

 conv5_block2_1_relu (Activ  (None, 7, 7, 512)            0
['conv5_block2_1_bn[0][0]']
 ation)

 conv5_block2_2_conv (Conv2  (None, 7, 7, 512)            2359808
['conv5_block2_1_relu[0][0]']
 D)

 conv5_block2_2_bn (BatchNo  (None, 7, 7, 512)            2048
['conv5_block2_2_conv[0][0]']
 rmalization)

 conv5_block2_2_relu (Activ  (None, 7, 7, 512)            0
['conv5_block2_2_bn[0][0]']
 ation)

 conv5_block2_3_conv (Conv2  (None, 7, 7, 2048)           1050624
['conv5_block2_2_relu[0][0]']
 D)

 conv5_block2_3_bn (BatchNo  (None, 7, 7, 2048)           8192
['conv5_block2_3_conv[0][0]']
 rmalization)

 conv5_block2_add (Add)      (None, 7, 7, 2048)           0
['conv5_block1_out[0][0]',
'conv5_block2_3_bn[0][0]']

 conv5_block2_out (Activati  (None, 7, 7, 2048)           0
['conv5_block2_add[0][0]']
 on)

 conv5_block3_1_conv (Conv2  (None, 7, 7, 512)            1049088
['conv5_block2_out[0][0]']
 D)
```

```
 conv5_block3_1_bn (BatchNo  (None, 7, 7, 512)           2048
['conv5_block3_1_conv[0][0]']
 rmalization)

 conv5_block3_1_relu (Activ  (None, 7, 7, 512)           0
['conv5_block3_1_bn[0][0]']
 ation)

 conv5_block3_2_conv (Conv2  (None, 7, 7, 512)           2359808
['conv5_block3_1_relu[0][0]']
 D)

 conv5_block3_2_bn (BatchNo  (None, 7, 7, 512)           2048
['conv5_block3_2_conv[0][0]']
 rmalization)

 conv5_block3_2_relu (Activ  (None, 7, 7, 512)           0
['conv5_block3_2_bn[0][0]']
 ation)

 conv5_block3_3_conv (Conv2  (None, 7, 7, 2048)          1050624
['conv5_block3_2_relu[0][0]']
 D)

 conv5_block3_3_bn (BatchNo  (None, 7, 7, 2048)          8192
['conv5_block3_3_conv[0][0]']
 rmalization)

 conv5_block3_add (Add)      (None, 7, 7, 2048)          0
['conv5_block2_out[0][0]',
'conv5_block3_3_bn[0][0]']

 conv5_block3_out (Activati  (None, 7, 7, 2048)          0
['conv5_block3_add[0][0]']
 on)

 flatten_1 (Flatten)         (None, 100352)              0
['conv5_block3_out[0][0]']

 fc1 (Dense)                 (None, 1024)                1027614
['flatten_1[0][0]']
                                                         72

 dropout (Dropout)           (None, 1024)                0
['fc1[0][0]']

 fc2 (Dense)                 (None, 256)                 262400
['dropout[0][0]']
```

```
 predictions (Dense)         (None, 15)                3855
['fc2[0][0]']

==================================================================================
==================
Total params: 126615439 (483.00 MB)
Trainable params: 118003727 (450.15 MB)
Non-trainable params: 8611712 (32.85 MB)
----------------------------------------------------------------------------------
------------------
```

- **Entrenamiento**

[ ]:
```
# Entrenamos el modelo
H = model.fit(norm_train, validation_data = norm_val, batch_size=128,␣
 ↪epochs=20, verbose=1)
```

```
Epoch 1/20
469/469 [==============================] - 103s 190ms/step - loss: 0.4959 -
accuracy: 0.9575 - val_loss: 0.4203 - val_accuracy: 0.9723
Epoch 2/20
469/469 [==============================] - 97s 207ms/step - loss: 0.1053 -
accuracy: 0.9896 - val_loss: 0.3107 - val_accuracy: 0.9787
Epoch 3/20
469/469 [==============================] - 87s 184ms/step - loss: 0.0516 -
accuracy: 0.9948 - val_loss: 0.0227 - val_accuracy: 0.9960
Epoch 4/20
469/469 [==============================] - 87s 184ms/step - loss: 0.1570 -
accuracy: 0.9894 - val_loss: 0.0053 - val_accuracy: 0.9990
Epoch 5/20
469/469 [==============================] - 98s 208ms/step - loss: 0.0194 -
accuracy: 0.9965 - val_loss: 0.0024 - val_accuracy: 0.9990
Epoch 6/20
469/469 [==============================] - 88s 186ms/step - loss: 0.0105 -
accuracy: 0.9985 - val_loss: 0.0019 - val_accuracy: 0.9997
Epoch 7/20
469/469 [==============================] - 88s 186ms/step - loss: 0.0145 -
accuracy: 0.9984 - val_loss: 0.0044 - val_accuracy: 0.9990
Epoch 8/20
469/469 [==============================] - 97s 207ms/step - loss: 0.0142 -
accuracy: 0.9981 - val_loss: 0.0558 - val_accuracy: 0.9967
Epoch 9/20
469/469 [==============================] - 97s 207ms/step - loss: 0.0814 -
accuracy: 0.9944 - val_loss: 5.8247 - val_accuracy: 0.9057
```

```
Epoch 10/20
469/469 [==============================] - 87s 186ms/step - loss: 0.1878 -
accuracy: 0.9919 - val_loss: 0.3777 - val_accuracy: 0.9787
Epoch 11/20
469/469 [==============================] - 88s 186ms/step - loss: 0.0401 -
accuracy: 0.9957 - val_loss: 0.0030 - val_accuracy: 0.9990
Epoch 12/20
469/469 [==============================] - 98s 207ms/step - loss: 0.0189 -
accuracy: 0.9985 - val_loss: 0.0083 - val_accuracy: 0.9987
Epoch 13/20
469/469 [==============================] - 88s 187ms/step - loss: 0.0359 -
accuracy: 0.9969 - val_loss: 0.0183 - val_accuracy: 0.9973
Epoch 14/20
469/469 [==============================] - 87s 184ms/step - loss: 0.0035 -
accuracy: 0.9989 - val_loss: 0.0063 - val_accuracy: 0.9990
Epoch 15/20
469/469 [==============================] - 87s 185ms/step - loss: 0.0137 -
accuracy: 0.9987 - val_loss: 0.0084 - val_accuracy: 0.9990
Epoch 16/20
469/469 [==============================] - 97s 206ms/step - loss: 0.0154 -
accuracy: 0.9988 - val_loss: 0.0174 - val_accuracy: 0.9980
Epoch 17/20
469/469 [==============================] - 88s 188ms/step - loss: 0.0056 -
accuracy: 0.9992 - val_loss: 0.0183 - val_accuracy: 0.9977
Epoch 18/20
469/469 [==============================] - 97s 207ms/step - loss: 0.0090 -
accuracy: 0.9993 - val_loss: 0.0062 - val_accuracy: 0.9987
Epoch 19/20
469/469 [==============================] - 88s 186ms/step - loss: 0.0036 -
accuracy: 0.9996 - val_loss: 0.0105 - val_accuracy: 0.9990
Epoch 20/20
469/469 [==============================] - 89s 188ms/step - loss: 0.0039 -
accuracy: 0.9996 - val_loss: 0.0199 - val_accuracy: 0.9980
```

6.2.7 Evaluación del modelo tras el Fine Tuning

Este es, con diferencia el mejor modelo generado en este proyecto, con valores f1-score perfectos para todos los vegetales, aunque presenta picos de validation loss.

- **Clases**

[ ]:
```
# clases
class_names = test_ds.class_names
class_names
```

[ ]:
```
['Bean',
 'Bitter_Gourd',
 'Bottle_Gourd',
 'Brinjal',
 'Broccoli',
 'Cabbage',
 'Capsicum',
 'Carrot',
 'Cauliflower',
 'Cucumber',
 'Papaya',
 'Potato',
 'Pumpkin',
 'Radish',
 'Tomato']
```

- **Evaluación**

[ ]:
```
# Evaluación del modelo
print("[INFO]: Evaluando el modelo...")
predictions = model.predict(x_test, batch_size=128)

# Obtener el report de clasificación
print(classification_report(y_test, predictions.argmax(axis=1),␣
 ↪target_names=class_names))
```

```
[INFO]: Evaluando el modelo…
24/24 [==============================] - 10s 380ms/step
              precision    recall  f1-score   support

        Bean       1.00      1.00      1.00       200
Bitter_Gourd       0.99      0.99      0.99       200
Bottle_Gourd       1.00      1.00      1.00       200
     Brinjal       1.00      0.99      0.99       200
    Broccoli       1.00      1.00      1.00       200
     Cabbage       1.00      1.00      1.00       200
    Capsicum       1.00      1.00      1.00       200
      Carrot       1.00      1.00      1.00       200
 Cauliflower       1.00      1.00      1.00       200
    Cucumber       0.99      1.00      0.99       200
      Papaya       1.00      0.99      0.99       200
```

```
              Potato       1.00      1.00      1.00       200
             Pumpkin       1.00      1.00      1.00       200
              Radish       1.00      1.00      1.00       200
              Tomato       1.00      1.00      1.00       200

            accuracy                           1.00      3000
           macro avg       1.00      1.00      1.00      3000
        weighted avg       1.00      1.00      1.00      3000
```

- **Visualización del Desempeño del Modelo**

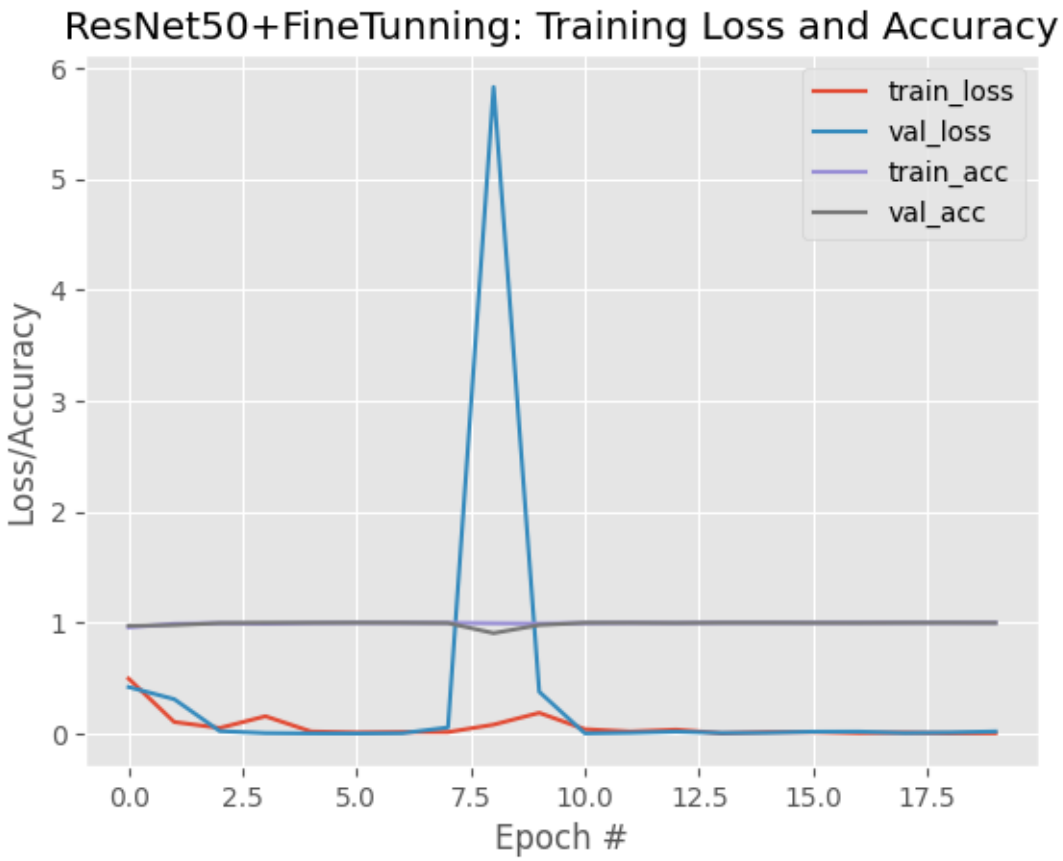
[ ]:
```python
# Gráfica de accuracy y losses
plt.style.use("ggplot")
plt.figure()

# Usar la longitud de H.history["loss"] para el rango de x
epochs_range = np.arange(len(H.history["loss"])) # Por si cambian las Epochs␣
 ↪del entrenamiento

plt.plot(epochs_range, H.history["loss"], label="train_loss")
plt.plot(epochs_range, H.history["val_loss"], label="val_loss")
plt.plot(epochs_range, H.history["accuracy"], label="train_acc")
plt.plot(epochs_range, H.history["val_accuracy"], label="val_acc")

plt.title("ResNet50+FineTunning: Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()

plt.show()
```



.

7 Conclusiones

En este proyecto, se desarrollaron varios modelos de redes neuronales convolucionales empleando distintas técnicas de regularización para clasificar imágenes de quince variedades de vegetales, haciendo uso del **Vegetable Image Dataset**.

Los Modelos diseñados **From Scratch** tendieron al sobreajuste, aunque mostraron mejoras significativas con la implementación de técnicas como la normalización por lotes (batch normalization) y el abandono (dropout). La técnica de **Detención Temprana (Early Stopping)** no mejoró el rendimiento de manera significativa, posiblemente debido a la variabilidad observada en la pérdida de validación (validation loss) en la mayoría de los modelos. Esto sugiere que detener prematuramente el entrenamiento puede impedir que la red se ajuste adecuadamente después de los picos de pérdida y converja de manera exitosa. Respecto a la técnica de **Aumento de Datos (Data Augmentation)**, si bien mejoró la clasificación para ciertos vegetales, redujo la precisión general en la fase de prueba. El uso de Data Augmentation corre el riesgo de causar sobreajuste en los datos generados sintéticamente, lo cual podría haber perjudicado a los vegetales menos representados durante la generación de datos.

Los modelos que utilizaron **Aprendizaje por Transferencia (Transfer Learning)** arrojaron resultados superiores en comparación con los construidos desde cero. El modelo que empleó **ResNet50** se destacó, logrando una precisión perfecta en todas las variedades de vegetales. Esto se debe probablemente a la mayor complejidad y profundidad arquitectónica de ResNet50 en comparación con VGG16; ResNet50 incluye 50 capas convolucionales frente a las 16 de VGG16. Se cree que ResNet50 se benefició más del **Ajuste Fino (Fine Tuning)** ya que, al tener las capas que detectan características básicas más distantes de las capas especializadas, es posible reentrenar las últimas capas para que se adapten mejor a los nuevos datos. En el caso de **VGG16**, reentrenar el último bloque convolucional podría implicar modificar pesos esenciales para la extracción de características básicas, dada su menor profundidad.

En conclusión, este proyecto subraya la importancia de experimentar con distintas arquitecturas de redes neuronales convolucionales, especialmente cuando se enfrentan limitaciones como la disponibilidad reducida de datos y capacidad de procesamiento limitada.