

✓ Máster en Inteligencia Artificial Aplicada

Unidad: Deep Learning - Caso Final

Clasificación de Radiografías de Tórax para COVID 19

Nombre: Patricio Galván

Fecha: 07 de Noviembre de 2024

Contenido:

1. Preparacion del Entorno
2. Lectura de Datos y Preparacion de Datos
3. Modelo 1: CNN Entrenado desde Cero
4. Modelo 2: VGG16
 - 4.1. Transfer Learning
 - 4.2. Fine Tuning
5. Modelo 3: ResNet50
 - 5.1. Transfer Learning
 - 5.2. Fine Tuning

Descripción

En este notebook, usaremos Redes CNN, y técnicas de Transfer Learning y Fine Tuning para clasificación de imágenes del Dataset Radiografías Covid 19.

La finalidad de este encargo es conseguir el mejor resultado posible utilizando las técnicas aprendidas a lo largo del curso de Deep Learning. Para Transfer Learning y Fine Tuning utilizaremos los modelos VGG16 y ResNet50.

Asimismo, como tenemos muy pocos datos utilizaremos técnicas de Data Augmentation.

Conclusion

De este trabajo hemos podido observar las grandes diferencias que existen al realizar diferentes técnicas de entrenamiento en redes neuronales pre-entrenadas y no entrenadas.

Cada modelo tiene sus propias características y lo que se aplica en uno, no funciona en otros, por lo que se debe proceder en cada uno con una metodología exploratoria única, pero siempre partiendo de lo más simple a lo más complejo.

El modelo que ha mostrado mejor comportamiento fue el VGG16, tanto para Transfer Learning como para Fine Tuning. El Modelo ResNet50 tiene comportamientos muy fluctuantes y explosivos con fácil tendencia al sobreajuste para este dataset.

En resumen, con el Modelo VGG16 hemos logrado un accuracy de **92.8%**!

	Train	Val
Loss	1.273390	1.351335
Acc	0.928287	0.848485

Descripción de los Modelos VGG16 y Resnet50

VGG16: es una red neuronal convolucional profunda diseñada para la clasificación de imágenes. Se caracteriza por:

- Arquitectura sencilla y uniforme: Consiste en 16 capas con pesos entrenables (13 capas convolucionales y 3 capas densas).
- Filtros pequeños (3x3): Utiliza convoluciones con filtros pequeños que permiten capturar características detalladas.
- Pooling: Después de bloques convolucionales, se aplican capas de max-pooling (reducción de dimensionalidad).
- Aplicación: VGG16 ha sido entrenado en el conjunto de datos ImageNet, que contiene más de un millón de imágenes clasificadas en 1000 categorías. Funciona bien en problemas de visión por computador como la clasificación de imágenes.

ResNet50: ResNet50 es una red más profunda con 50 capas entrenables y se basa en una técnica innovadora llamada "residual learning":

- Bloques residuales: Cada bloque incluye conexiones residuales ("skip connections") que permiten que el gradiente fluya mejor durante el entrenamiento, solucionando el problema de la degradación en redes muy profundas.
- Mayor precisión: ResNet50 es más eficiente que redes anteriores debido a la capacidad de entrenar modelos más profundos sin perder precisión.
- Aplicación: Al igual que VGG16, ResNet50 está preentrenado en ImageNet, lo que lo hace útil para tareas como clasificación de imágenes y transferencia de aprendizaje en otros problemas de visión.

Descripcion del Dataset Radiografías de COVID-19

El dataset de radiografías de COVID-19 incluye radiografías de tórax que se utilizan para entrenar modelos de aprendizaje automático y redes neuronales con el fin de diagnosticar la infección por COVID-19.

- Imágenes: Incluye radiografías de tórax en formato JPEG o PNG.
- Clases: Las imágenes suelen estar clasificadas en varias categorías, como:
 - COVID-19 positivo: Radiografías de pacientes diagnosticados con COVID-19.
 - Neumonía viral: Radiografías de pacientes con neumonía de otras causas virales.
 - Normal: Radiografías de pacientes sin infecciones pulmonares.
- Tamaño: X imagenes de Train y x de Test
- Resolución: Las imágenes pueden tener diferentes resoluciones, pero suelen estar en el rango de 450x450 a 1200x1200 píxeles, aunque es común trabajar con un tamaño uniforme para el entrenamiento de modelos. Formato: Las imágenes pueden ser en color (3 canales) o en blanco y negro (1 canal). Muchos datasets relacionados con radiografías utilizan imágenes en escala de grises.

1.0 Preparamos el Entorno

Load Libraries

```
# @title
# Importamos las librerias
import numpy as np
import pandas as pd
import time
import os
import math
import matplotlib.pyplot as plt


from tensorflow.keras.applications import VGG16
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.layers import GlobalAveragePooling2D

import keras
from keras.applications import VGG16
from keras.models import Sequential
from keras.layers import Flatten, Dense, Dropout, Conv2D, MaxPooling2D
from keras.optimizers import Adam

from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

# Otras
import warnings
warnings.filterwarnings("ignore")
```

```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

 Mounted at /content/drive

Definimos Funciones y Herramientas

```
# @title
# Funcion para graficar el proceso de entrenamiento

def training_plot(history):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(len(acc))

    plt.figure(figsize=(12,4))
    plt.subplot(1,2,1)
    plt.plot(epochs, acc, 'b.', label='Training accuracy') # Plot training accuracy in blue with dots
    plt.plot(epochs, val_acc, 'r-', label='Validation accuracy') # Plot validation accuracy in red with line
    plt.title('Training and validation accuracy')
    plt.legend()
```

```
plt.subplot(1,2,2) # Create subplot for loss
plt.plot(epochs, loss, 'b.', label='Training loss') # Plot training loss in blue with dots
plt.plot(epochs, val_loss, 'r-', label='Validation loss') # Plot validation loss in red with line
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

```
# @title
# Funcion para Generar Matriz de Confusion

def plot_confusion_matrix(cm, classes,
                           normalize=False,
                           title='Confusion Matrix',
                           cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Matriz de Confusion Normalizada")
    else:
        print('Matriz de Confusion Normalizada')

    import itertools
    print(cm)
    plt.figure(figsize=(6, 6))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('Clase Real')
    plt.xlabel('Clase Predicha')
    plt.tight_layout()
```

✓ 2.0 Lectura de Datos y Preparacion de Datos

Obtenemos los data flows desde las carpetas directamente con ayuda de la clase *ImageDataGenerator*. El método *flow_from_directory* permite utilizar directamente los datos, asignando a cada carpeta una clase/etiqueta.

```
# @title
# Directorio que contiene las carpetas de entrenamiento y prueba
PATH = "/content/drive/MyDrive/Colab Notebooks/Deep Learning/Final/Covid19-dataset"
```

```
# @title
#Generamos los Path a cada carpeta de train y test
train_dir = os.path.join(PATH, 'train')
test_dir = os.path.join(PATH, 'test')

# Cuentamos el número de archivos de imagen en los directorios de train y test
num_train_files = sum(len(files) for _, _, files in os.walk(train_dir))
num_test_files = sum(len(files) for _, _, files in os.walk(test_dir))

# Imprime el número de archivos
print(f"Número de imágenes de entrenamiento: {num_train_files}.")
print(f"Número de imágenes de test: {num_test_files}.")
```

↗ Número de imágenes de entrenamiento: 251.
Número de imágenes de test: 66.

```
# @title
%time
# Configurar los generadores de datos para entrenamiento y prueba
batch_size1 = 32

train_datagen = ImageDataGenerator(rescale=1. / 255,
                                    rotation_range=20,
                                    width_shift_range=0.1,
                                    height_shift_range=0.1,
```

```

        shear_range=0.2,
        zoom_range=0.1,
        horizontal_flip=True,
        fill_mode='nearest')

test_datagen = ImageDataGenerator(rescale=1./255)

# @title
%time
# Configurar los generadores de datos para entrenamiento y prueba

train_datagen = ImageDataGenerator(rescale=1. / 255,
                                   rotation_range=30,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.2,
                                   zoom_range=0.4,
                                   horizontal_flip=True,
                                   vertical_flip=True,
                                   fill_mode='nearest')

test_datagen = ImageDataGenerator(rescale=1./255)

# Especifica cómo leer los datos de entrenamiento y prueba desde el directorio
train_generator = train_datagen.flow_from_directory(
    directory = train_dir,
    target_size = (224, 224), # Tamaño de las imágenes
    batch_size = 16,
    shuffle=True,
    class_mode = 'categorical') # Modo de clasificación categórica

test_generator = test_datagen.flow_from_directory(
    directory = test_dir,
    target_size = (224, 224), # Tamaño de las imágenes
    batch_size = 16,
    shuffle=False,
    class_mode = 'categorical') # Modo de clasificación categórica

images_batch, labels_batch = next(train_generator) # Obtiene un lote de datos

```

```

CPU times: user 4 µs, sys: 0 ns, total: 4 µs
Wall time: 8.34 µs
CPU times: user 4 µs, sys: 0 ns, total: 4 µs
Wall time: 7.87 µs
Found 251 images belonging to 3 classes.
Found 66 images belonging to 3 classes.

```

Analizamos las formas de los vectores de Imagen y Label

```

# @title
print(f'- Images data Vector shape: {images_batch.shape}') # Forma de los datos de imágenes
print(f'- Labels data Vector shape: {labels_batch.shape}') # Forma de los datos de etiquetas
class_indices = train_generator.class_indices # Muestra el mapeo de nombres de clases a índices
print("- Clases en los datos: ", class_indices)

target_names = list(class_indices.keys())
print("- Nombres de Clases: ", target_names)

```

```

- Images data Vector shape: (16, 224, 224, 3)
- Labels data Vector shape: (16, 3)
- Clases en los datos: {'Covid': 0, 'Normal': 1, 'Viral Pneumonia': 2}
- Nombres de Clases: ['Covid', 'Normal', 'Viral Pneumonia']

```

```

# Definimos el generador para la exploracion de informacion
original_datagen = ImageDataGenerator(rescale=1./255)

```

```

# Create the generator for original images
original_generator = original_datagen.flow_from_directory(
    directory=train_dir,
    target_size=(224, 224),
    batch_size=16,
    shuffle=True,
    class_mode='categorical'
)

```

```

images_ori, labels_ori = next(original_generator)

```

```

Found 251 images belonging to 3 classes.

```

Visualización Exploratoria de las Imágenes

```
# @title
# Crear una figura con subplots
fig, axes = plt.subplots(2, 5, figsize=(11, 5)) # 2 filas, 5 columnas (10 imágenes)
fig.tight_layout() # Ajustar el espaciado entre subplots

# lista de nombres de clases
classes = list(original_generator.class_indices.keys())

# Iterar sobre las clases y graficar ejemplos
for i in range(10):
    # Seleccionar la imagen y etiqueta de ejemplo para la clase i
    image = images_ori[i] # Obtener la imagen en la posición i
    label = labels_ori[i] # Obtener la etiqueta en la posición i

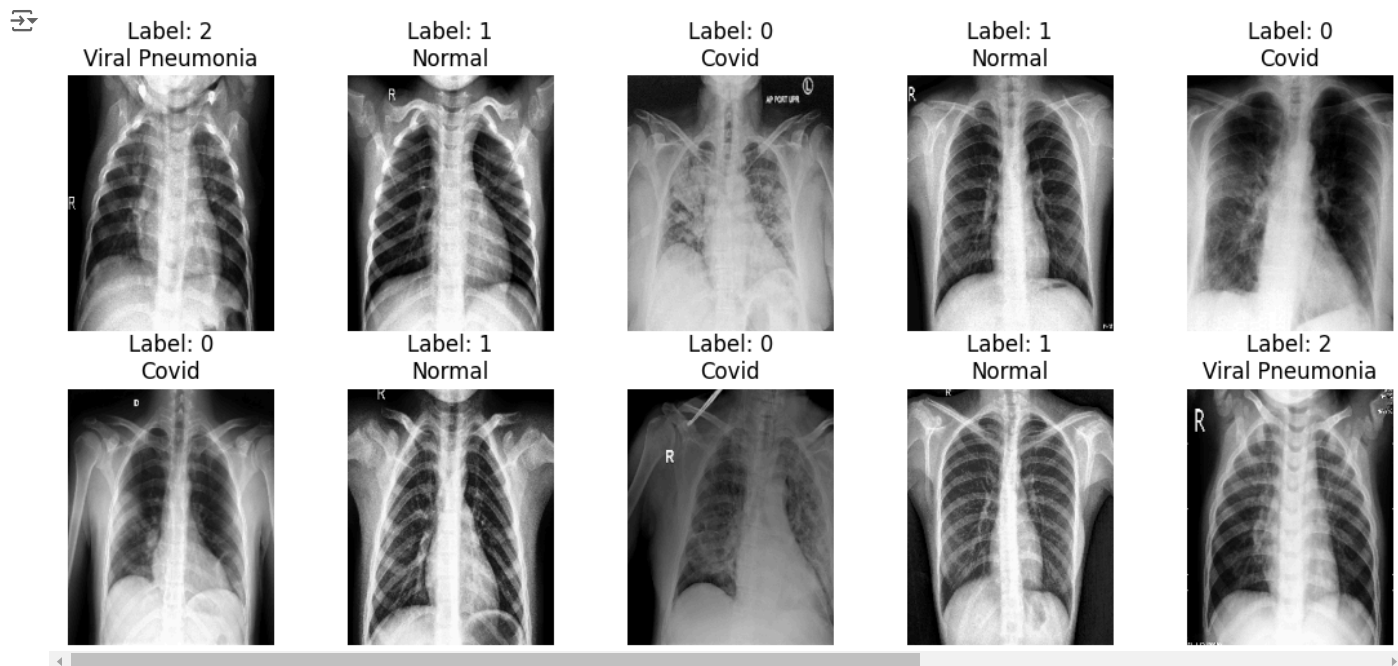
    # Obtener el índice de la clase a partir de la etiqueta one-hot
    class_index = np.argmax(label)

    # Graficar la imagen en el subplot correspondiente
    row = i // 5 # Calcular la fila del subplot
    col = i % 5 # Calcular la columna del subplot
    axes[row, col].imshow(image)

    # Asignar el título con el nombre de la clase
    axes[row, col].set_title(f"Label: {class_index}\n{classes[class_index]}")
    axes[row, col].axis("off") # Ocultar los ejes

# Ajustar el tamaño de las imágenes para que se vean más pequeñas
for ax in axes.flat:
    ax.set_aspect('auto')

# Mostrar la figura
plt.show()
```



3.0 Modelo 1: CNN Entrenado desde Cero

Primero vamos a generar un modelo de CNN desde cero y entrenarlo con los datos. Aunque podríamos probar con redes más complejas, va a ser difícil conseguir un accuracy bueno dado que tenemos un numero limitado de imágenes.

```
# @title
%%time


# Definimos el modelo de la CNN
model = Sequential()

# Bloque 1
model.add(Conv2D(50, (3, 3), activation='relu', input_shape=(224, 224, 3)))
model.add(Conv2D(50, (2, 2), activation='relu'))
#model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(MaxPooling2D((2, 2)))
```

```
# Bloque 2
model.add(Conv2D(100, (2, 2), activation='relu'))
model.add(Conv2D(100, (2, 2), activation='relu'))
#model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(MaxPooling2D((2, 2)))

# Perceptron Multicapa para Clasificacion
model.add(Flatten())
model.add(Dense(100, activation='relu', kernel_regularizer='l2')) #
model.add(Dropout(0.5))
model.add(Dense(3, activation='softmax'))


# Compilar el modelo
model.compile(optimizer=Adam(learning_rate=0.0005),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

 CPU times: user 111 ms, sys: 174 µs, total: 111 ms
Wall time: 108 ms

```
# Callback para guardar el mejor modelo basado en val_accuracy
checkpoint = ModelCheckpoint(
    filepath='model/CNN_best.keras',      # Ruta donde se guardará el mejor modelo
    monitor='val_accuracy',               # Monitorea la precisión en el conjunto de validación
    verbose=1,                           # Mensajes de progreso
    mode='max',                           # Guarda el modelo con la precisión más alta
    save_best_only=True                   # Solo guarda el modelo si es el mejor encontrado hasta ahora
)

# Callback para detener el entrenamiento tempranamente si no mejora el desempeño
early_stopping = EarlyStopping(
    monitor='val_loss',                  # Monitorea la pérdida en el conjunto de validación
    patience=5,                          # Detiene el entrenamiento después de 5 épocas sin mejora
    restore_best_weights=True            # Restaura los pesos del mejor modelo al detenerse
)
```

```
# Entrenar el modelo
history_CNN = model.fit(
    train_generator,
    epochs=15,
    validation_data=test_generator,
    callbacks=[checkpoint, early_stopping],
    shuffle=False,
    verbose=2)
```

 Epoch 1: val_accuracy improved from -inf to 0.65152, saving model to model/CNN_best.keras
16/16 - 26s - 2s/step - accuracy: 0.3745 - loss: 3.2944 - val_accuracy: 0.6515 - val_loss: 2.2316
Epoch 2/15

Epoch 2: val_accuracy did not improve from 0.65152
16/16 - 9s - 562ms/step - accuracy: 0.4980 - loss: 2.0372 - val_accuracy: 0.6515 - val_loss: 1.8572
Epoch 3/15

Epoch 3: val_accuracy did not improve from 0.65152
16/16 - 9s - 537ms/step - accuracy: 0.5339 - loss: 1.7012 - val_accuracy: 0.5909 - val_loss: 1.5799
Epoch 4/15

Epoch 4: val_accuracy did not improve from 0.65152
16/16 - 8s - 530ms/step - accuracy: 0.5618 - loss: 1.4058 - val_accuracy: 0.5000 - val_loss: 1.3837
Epoch 5/15

Epoch 5: val_accuracy did not improve from 0.65152
16/16 - 8s - 522ms/step - accuracy: 0.5339 - loss: 1.3597 - val_accuracy: 0.4242 - val_loss: 1.4068
Epoch 6/15

Epoch 6: val_accuracy did not improve from 0.65152
16/16 - 9s - 532ms/step - accuracy: 0.5179 - loss: 1.3160 - val_accuracy: 0.5909 - val_loss: 1.3664
Epoch 7/15

Epoch 7: val_accuracy did not improve from 0.65152
16/16 - 8s - 530ms/step - accuracy: 0.5657 - loss: 1.2120 - val_accuracy: 0.6212 - val_loss: 1.1896
Epoch 8/15

Epoch 8: val_accuracy did not improve from 0.65152
16/16 - 8s - 526ms/step - accuracy: 0.5538 - loss: 1.2019 - val_accuracy: 0.5303 - val_loss: 1.2603
Epoch 9/15

Epoch 9: val_accuracy did not improve from 0.65152
16/16 - 9s - 534ms/step - accuracy: 0.5817 - loss: 1.1047 - val_accuracy: 0.6364 - val_loss: 1.1806
Epoch 10/15

Epoch 10: val_accuracy improved from 0.65152 to 0.69697, saving model to model/CNN_best.keras

```
Epoch 11: val_accuracy did not improve from 0.69697
16/16 - 9s - 575ms/step - accuracy: 0.6335 - loss: 0.9697 - val_accuracy: 0.6667 - val_loss: 1.0241
Epoch 12/15

Epoch 12: val_accuracy did not improve from 0.69697
16/16 - 8s - 521ms/step - accuracy: 0.5498 - loss: 1.1174 - val_accuracy: 0.6515 - val_loss: 1.1413
Epoch 13/15

Epoch 13: val_accuracy did not improve from 0.69697
16/16 - 8s - 523ms/step - accuracy: 0.6056 - loss: 1.0122 - val_accuracy: 0.6364 - val_loss: 1.1355
Epoch 14/15

Epoch 14: val_accuracy did not improve from 0.69697
16/16 - 8s - 524ms/step - accuracy: 0.6096 - loss: 1.0975 - val_accuracy: 0.5152 - val_loss: 1.2503
Epoch 15/15

Epoch 15: val_accuracy did not improve from 0.69697
16/16 - 8s - 521ms/step - accuracy: 0.5777 - loss: 1.0490 - val_accuracy: 0.5606 - val_loss: 1.1662
```

```
# Resumen de Resultados
```

```
train_result = model.evaluate(train_generator)
```

```
test_result = model.evaluate(test_generator)
```

```
CNN_df = pd.DataFrame(zip(train_result,test_result),columns=['Train','Val'],index=['Loss','Acc'])
CNN_df
```

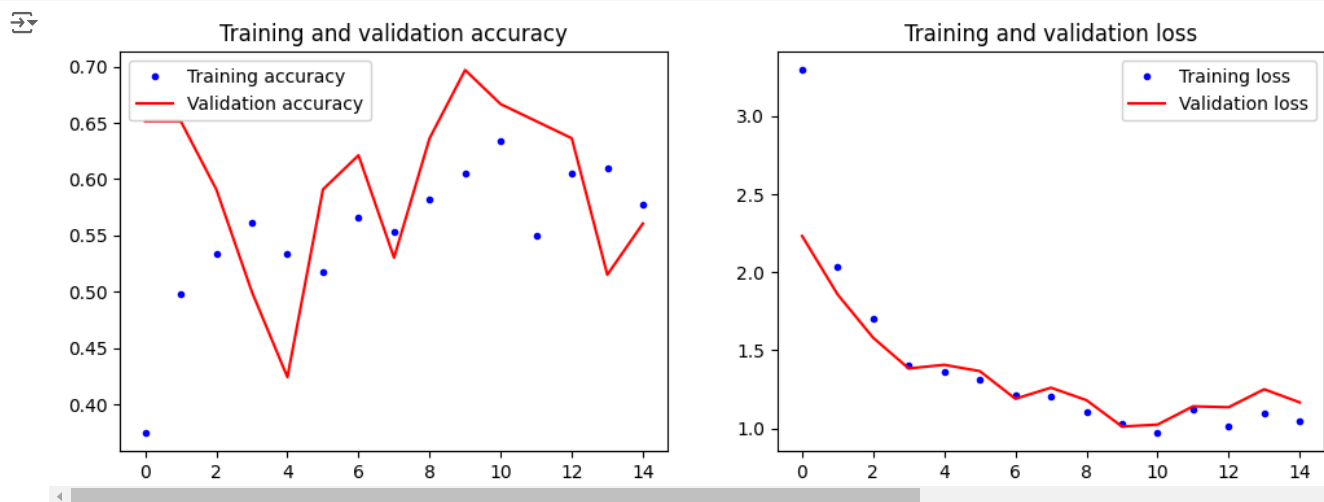
```
16/16 ----- 10s 590ms/step - accuracy: 0.6831 - loss: 1.0406
5/5 ----- 1s 156ms/step - accuracy: 0.7358 - loss: 0.9692
```

	Train	Val
Loss	1.053487	1.011685
Acc	0.649402	0.696970

```
# @title
```

```
# Visualizamos el proceso de Entrenamiento
```

```
training_plot(history_CNN)
```



Metricas de Clasificación y Matriz de Confusion

```
# @title
```

```
# Calculamos las metricas de Clasificación
```

```
num_of_validation_samples = test_generator.samples
```

```
Y_pred_res = model.predict(test_generator, steps=num_of_validation_samples + 1)
```

```
y_pred_resnet = np.argmax(Y_pred_res, axis=1)
```

```
print('Matriz de Confusión')
```

```
conf_matrix_res = confusion_matrix(test_generator.classes, y_pred_resnet)
```

```
cm_res = np.array2string(conf_matrix_res)
```

```
print(conf_matrix_res)
```

```
print("=====")
```

```
print('Resumen de Clasificación')
```

```
class_rep_res = classification_report(test_generator.classes, y_pred_resnet, target_names=target_names)
```

```
print(class_rep_res)
```

```
67/67 ----- 1s 11ms/step
```

```
Matriz de Confusión
```

```
[[25  1  0]
 [ 0  2 18]
 [ 0  1 19]]
```

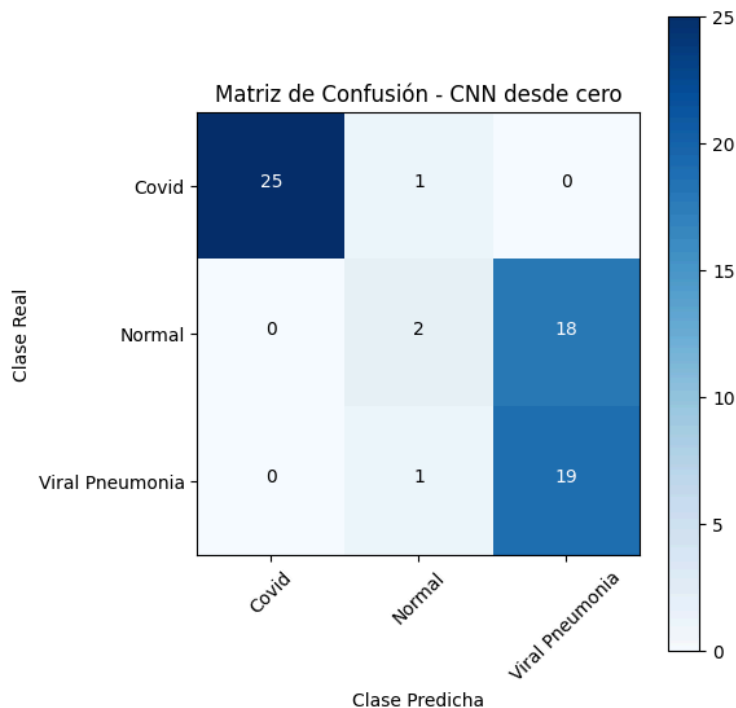
```
=====
Resumen de Clasificación
```

```
precision    recall  f1-score   support
```

Covid	1.00	0.96	0.98	26
Normal	0.50	0.10	0.17	20
Viral Pneumonia	0.51	0.95	0.67	20
accuracy			0.70	66
macro avg	0.67	0.67	0.60	66
weighted avg	0.70	0.70	0.64	66

```
# @title
# Ploteamos la Matriz de Clasificación
print('Matriz de Confusión')
conf_matrix_res = confusion_matrix(test_generator.classes, y_pred_resnet)
plot_confusion_matrix(conf_matrix_res, classes=target_names, title='Matriz de Confusión - CNN desde cero')
```

```
Matriz de Confusión
Matriz de Confusion Normalizada
[[25  1  0]
 [ 0  2 18]
 [ 0  1 19]]
```



Analisis de Resultados del Modelo 1: Entrenamiento CNN

Aquí tienes la tabla en formato Markdown:

	Train	Val
Loss	1.053487	1.011685
Acc	0.649402	0.696970

Analisis graficos:

En el grafico de Accuracy podemos una tendencia al alza de los datos de entrenamiento, mientras que los datos de validacion muestran una forma muy fluctuante, lo cual se puede deber a los pocos datos que tenemos o sobre ajuste.

El grafico de "perdida" muestra una tendencia descendente de los datos, con bastante buena uniformidad, tanto en los datos de entrenamiento como de evaluacion.

Conclusion

No obstante tenemos muy pocos datos para entrenar hemos logrado un accuracy de 64.9% en los datos de entrenamiento y un 69.6% en los datos de validacion. Lo cual es un resultado medianamente aceptable, sobre todo pensando que la presicion para detectar Covid a sido un 100% con un recall de 96%

El modelo empleado tiene una arquitectura sencilla, para evitar sobreajustes. Se realizaron muchas pruebas buscando la mejor configuración posible, hemos usado 2 bloques con 2 redes hidden cada uno y un bloque final para clasificación. Tambien hemos introducido algo de Dropout para evitar sobreajuste y suavizar el entrenamiento. Hemos empleado un batch lo mas bajo posible para evitar saltos en el entrenamiento. Finalmente hemos tenido que considerar un Learning Rate bajo, para evitar tanta fluctuacion en los valores.

4.0 Modelo 2: VGG16

✓ 4.1 VGG16 - Transfer Learning

En esta sección entrenaremos el modelo VGG16 con los pesos del modelo entrenado en imagenet. Primero lo hacemos con Transfer Learning, congelando todas las capas, y luego con Fine Tuning. El último bloque se modificará para adaptarlo a la clasificación de 3 elementos.

```
# @title
%%time

# Cargar el modelo VGG16 pre-entrenado sin la capa de clasificación
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Congelamos las capas del modelo base
for layer in base_model.layers:
    layer.trainable = False

# Definimos el modelo secuencial sobre el modelo base
model = Sequential()
model.add(base_model) # modelo base
model.add(GlobalAveragePooling2D())

# Bloque Salida
model.add(Flatten()) # Aplanamos las salidas
model.add(BatchNormalization())
model.add(Dense(300, activation='relu', kernel_regularizer='l2'))
model.add(Dropout(0.5))
model.add(Dense(300, activation='relu', kernel_regularizer='l2'))
model.add(Dropout(0.5))
model.add(Dense(len(train_generator.class_indices), activation='softmax')) # Capa de salida

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

⚡ CPU times: user 247 ms, sys: 18.1 ms, total: 265 ms
Wall time: 261 ms

```
# Callback para guardar el mejor modelo basado en val_accuracy
checkpoint = ModelCheckpoint(
    filepath='model/vgg16T_best.keras', # Ruta donde se guardará el mejor modelo
    monitor='val_accuracy',           # Monitorea la precisión en el conjunto de validación
    verbose=1,                        # Mensajes de progreso
    mode='max',                       # Guarda el modelo con la precisión más alta
    save_best_only=True               # Solo guarda el modelo si es el mejor encontrado hasta ahora
)

# Callback para detener el entrenamiento tempranamente si no mejora el desempeño
early_stopping = EarlyStopping(
    monitor='val_loss',               # Monitorea la pérdida en el conjunto de validación
    patience=5,                      # Detiene el entrenamiento después de 5 épocas sin mejora
    restore_best_weights=True        # Restaura los pesos del mejor modelo al detenerse
)
```

```
# @title
%%time
# Entrenamos el modelo
history_VGG16_tr = model.fit(
    x = train_generator,
    validation_data=test_generator,
    epochs=25,
    callbacks=[checkpoint, early_stopping],
    shuffle=False,
    verbose=2)
```



```

16/16 - 9s - 530ms/step - accuracy: 0.8247 - loss: 2.2177 - val_accuracy: 0.8030 - val_loss: 2.3403
Epoch 17/25

Epoch 17: val_accuracy did not improve from 0.86364
16/16 - 8s - 529ms/step - accuracy: 0.8765 - loss: 1.9960 - val_accuracy: 0.8030 - val_loss: 2.2043
Epoch 18/25

Epoch 18: val_accuracy did not improve from 0.86364
16/16 - 9s - 537ms/step - accuracy: 0.8606 - loss: 1.8951 - val_accuracy: 0.7879 - val_loss: 2.0690
Epoch 19/25

Epoch 19: val_accuracy did not improve from 0.86364
16/16 - 8s - 527ms/step - accuracy: 0.8327 - loss: 1.8318 - val_accuracy: 0.8182 - val_loss: 1.9202
Epoch 20/25

Epoch 20: val_accuracy did not improve from 0.86364
16/16 - 8s - 531ms/step - accuracy: 0.8685 - loss: 1.6644 - val_accuracy: 0.8333 - val_loss: 1.7956
Epoch 21/25

Epoch 21: val_accuracy did not improve from 0.86364
16/16 - 9s - 537ms/step - accuracy: 0.8606 - loss: 1.6170 - val_accuracy: 0.8333 - val_loss: 1.6766
Epoch 22/25

Epoch 22: val_accuracy improved from 0.86364 to 0.87879, saving model to model/vgg16T_best.keras
16/16 - 9s - 551ms/step - accuracy: 0.8805 - loss: 1.4875 - val_accuracy: 0.8788 - val_loss: 1.5987
Epoch 23/25

Epoch 23: val_accuracy did not improve from 0.87879
16/16 - 9s - 537ms/step - accuracy: 0.8566 - loss: 1.4196 - val_accuracy: 0.8333 - val_loss: 1.4927
Epoch 24/25

Epoch 24: val_accuracy did not improve from 0.87879
16/16 - 8s - 528ms/step - accuracy: 0.8765 - loss: 1.4357 - val_accuracy: 0.8636 - val_loss: 1.4075
Epoch 25/25

Epoch 25: val_accuracy did not improve from 0.87879
16/16 - 8s - 524ms/step - accuracy: 0.8685 - loss: 1.3196 - val_accuracy: 0.8485 - val_loss: 1.3513
CPU times: user 3min 34s, sys: 8.4 s, total: 3min 42s
Wall time: 3min 45s

```

```
# Resumen de Resultados
```

```
train_result = model.evaluate(train_generator)
```

```
test_result = model.evaluate(test_generator)
```

```
VGG16T_df = pd.DataFrame(zip(train_result, test_result), columns=['Train', 'Val'], index=['Loss', 'Acc'])
```

```
VGG16T_df
```

```

16/16 ----- 9s 542ms/step - accuracy: 0.9151 - loss: 1.2814
5/5 ----- 1s 164ms/step - accuracy: 0.9052 - loss: 1.3115

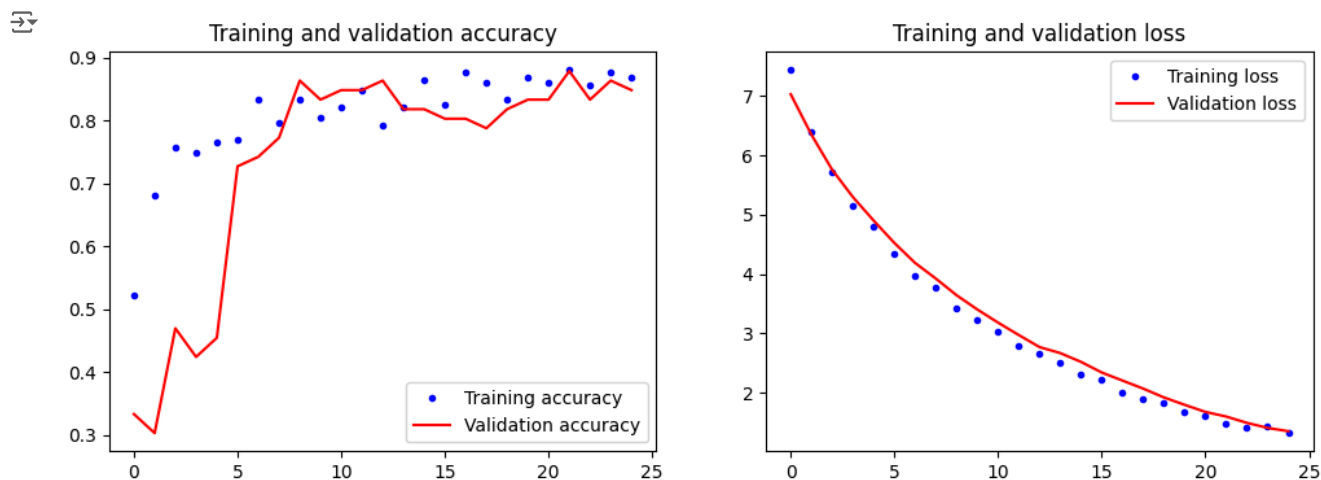
```

	Train	Val
Loss	1.273390	1.351335
Acc	0.928287	0.848485

```
# @title
```

```
# Visualizamos el proceso de Entrenamiento
```

```
training_plot(history_VGG16_tr)
```



Métricas de Clasificación y Matriz de Confusion

```
# @title
```

```
# Calculamos las metricas de Clasificación
```

```
num_of_validation_samples = test_generator.samples
```

```

Y_pred_VGG16T = model.predict(test_generator, steps=num_of_validation_samples + 1)
y_pred_VGG16T = np.argmax(Y_pred_VGG16T, axis=1)
print('Matriz de Confusión')
conf_matrix_res = confusion_matrix(test_generator.classes, y_pred_VGG16T)
cm_res = np.array2string(conf_matrix_res)
print(conf_matrix_res)
print("=====")
print('Resumen de Clasificación')
class_rep_res = classification_report(test_generator.classes, y_pred_resnet, target_names=target_names)
print(class_rep_res)

```

67/67 2s 18ms/step

Matriz de Confusión

```

[[26  0  0]
 [ 0 16  4]
 [ 0  6 14]]

```

=====
Resumen de Clasificación

	precision	recall	f1-score	support
Covid	1.00	0.88	0.94	26
Normal	0.49	1.00	0.66	20
Viral Pneumonia	0.50	0.05	0.09	20
accuracy			0.67	66
macro avg	0.66	0.64	0.56	66
weighted avg	0.69	0.67	0.60	66

```

# @title
# Ploteamos la Matriz de Clasificación
print('Matriz de Confusión')
conf_matrix_res = confusion_matrix(test_generator.classes, y_pred_VGG16T)
plot_confusion_matrix(conf_matrix_res, classes=target_names, title='Matriz de Confusión - VGG16 Transfer Learning')

```

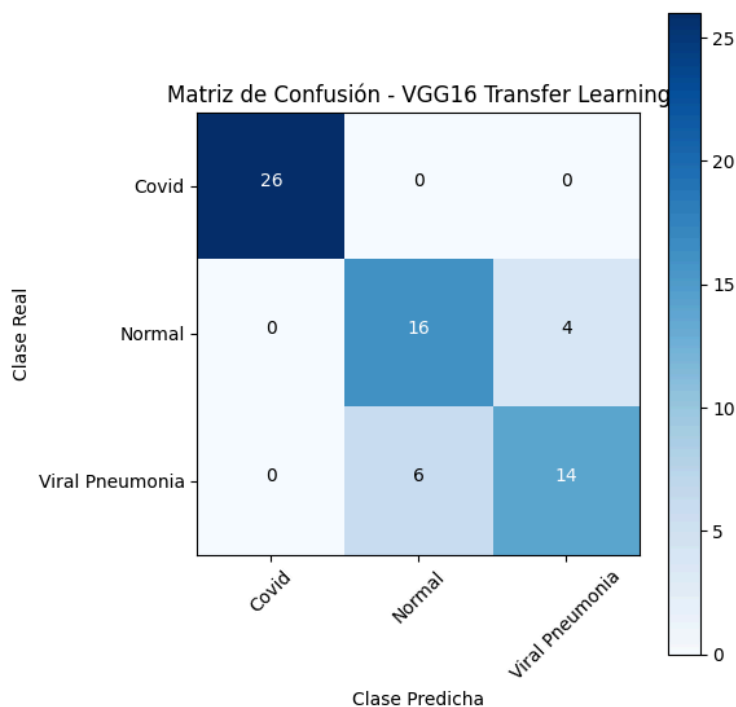
Matriz de Confusión

Matriz de Confusion Normalizada

```

[[26  0  0]
 [ 0 16  4]
 [ 0  6 14]]

```



Análisis de Resultados del VGG16 con Transfer Learning

	Train	Val
Loss	1.273390	1.351335
Acc	0.928287	0.848485

Análisis gráficos:

En el gráfico de Accuracy podemos ver que los datos de entrenamiento tienen un comportamiento casi monótonicamente creciente, mientras que los datos de validación lo hacen en forma más fluctuante.

El gráfico de "perdida" para ambos casos de datos muestra una curva monótonicamente decreciente, que parece converger.

Conclusion

Usando dos bloques de salida de 300 neuronas con regularización L2 y dropout de 0.5 hemos logrado un **Accuracy de 92.8%** empleando

Transfer Learning en VGG16.

4.2 VGG16 - Fine Tuning

A continuacion realizaremos la técnica de Fine Tunning sobre el mismo modelo VGG16.

```
# @title
# Cargamos el modelo VGG16 pre-entrenado sin la capa de clasificación
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Permitimos modificar las capas del modelo base
for layer in base_model.layers:
    layer.trainable = True

# Definimos el modelo secuencial sobre el modelo base
model = Sequential()
model.add(base_model) # Añadimos el modelo base
model.add(GlobalAveragePooling2D())

model.add(Flatten()) # Aplanamos las salidas
model.add(Dense(64, activation='relu', kernel_regularizer='l2'))
model.add(Dropout(0.50))

model.add(Dense(len(train_generator.class_indices), activation='softmax')) # Capa de salida

# @title
# Compilamos el modelo
model.compile(optimizer=Adam(learning_rate=0.0003),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Callback para guardar el mejor modelo basado en val_accuracy
checkpoint = ModelCheckpoint(
    filepath='model/vgg16F_best.keras', # Ruta donde se guardará el mejor modelo
    monitor='val_accuracy', # Monitorea la precisión en el conjunto de validación
    verbose=1, # Mensajes de progreso
    mode='max', # Guarda el modelo con la precisión más alta
    save_best_only=True # Solo guarda el modelo si es el mejor encontrado hasta ahora
)

# Callback para detener el entrenamiento tempranamente si no mejora el desempeño
early_stopping = EarlyStopping(
    monitor='val_loss', # Monitorea la pérdida en el conjunto de validación
    patience=5, # Detiene el entrenamiento después de 5 épocas sin mejora
    restore_best_weights=True # Restaura los pesos del mejor modelo al detenerse
)

# @title
%%time
# Entrenamos el modelo
history_VGG16_ft = model.fit(
    train_generator,
    epochs=20,
    validation_data=test_generator,
    callbacks=[checkpoint, early_stopping],
    shuffle=False,
    verbose=2)
```



```

16/16 - 8s - 528ms/step - accuracy: 0.6494 - loss: 1.0265 - val_accuracy: 0.6970 - val_loss: 0.8499
Epoch 13/20

Epoch 13: val_accuracy did not improve from 0.74242
16/16 - 9s - 541ms/step - accuracy: 0.6414 - loss: 1.0019 - val_accuracy: 0.5152 - val_loss: 1.0979
Epoch 14/20

Epoch 14: val_accuracy improved from 0.74242 to 0.83333, saving model to model/vgg16F_best.keras
16/16 - 9s - 578ms/step - accuracy: 0.6574 - loss: 0.9378 - val_accuracy: 0.8333 - val_loss: 0.7437
Epoch 15/20

Epoch 15: val_accuracy did not improve from 0.83333
16/16 - 9s - 550ms/step - accuracy: 0.6773 - loss: 0.9587 - val_accuracy: 0.6212 - val_loss: 0.8736
Epoch 16/20

Epoch 16: val_accuracy did not improve from 0.83333
16/16 - 8s - 530ms/step - accuracy: 0.6056 - loss: 0.9750 - val_accuracy: 0.6212 - val_loss: 0.8853
Epoch 17/20

Epoch 17: val_accuracy did not improve from 0.83333
16/16 - 9s - 534ms/step - accuracy: 0.5777 - loss: 0.9011 - val_accuracy: 0.7879 - val_loss: 0.6781
Epoch 18/20

Epoch 18: val_accuracy did not improve from 0.83333
16/16 - 8s - 526ms/step - accuracy: 0.6614 - loss: 0.9285 - val_accuracy: 0.7576 - val_loss: 0.7690
Epoch 19/20

Epoch 19: val_accuracy did not improve from 0.83333
16/16 - 9s - 533ms/step - accuracy: 0.6175 - loss: 0.8668 - val_accuracy: 0.6212 - val_loss: 0.8348
Epoch 20/20

Epoch 20: val_accuracy did not improve from 0.83333
16/16 - 9s - 537ms/step - accuracy: 0.6534 - loss: 0.8624 - val_accuracy: 0.6515 - val_loss: 0.7271
CPU times: user 3min 42s, sys: 9.14 s, total: 3min 51s
Wall time: 3min 16s

```

```

# @title
# Evaluamos el modelo
# Resumen de Resultados
train_result = model.evaluate(train_generator)
test_result = model.evaluate(test_generator)

VGG16_ft_df = pd.DataFrame(zip(train_result,test_result),columns=['Train','Val'],index=['Loss','Acc'])
VGG16_ft_df

```

```

16/16 ————— 8s 521ms/step - accuracy: 0.6457 - loss: 0.8768
5/5 ————— 1s 156ms/step - accuracy: 0.8885 - loss: 0.5409

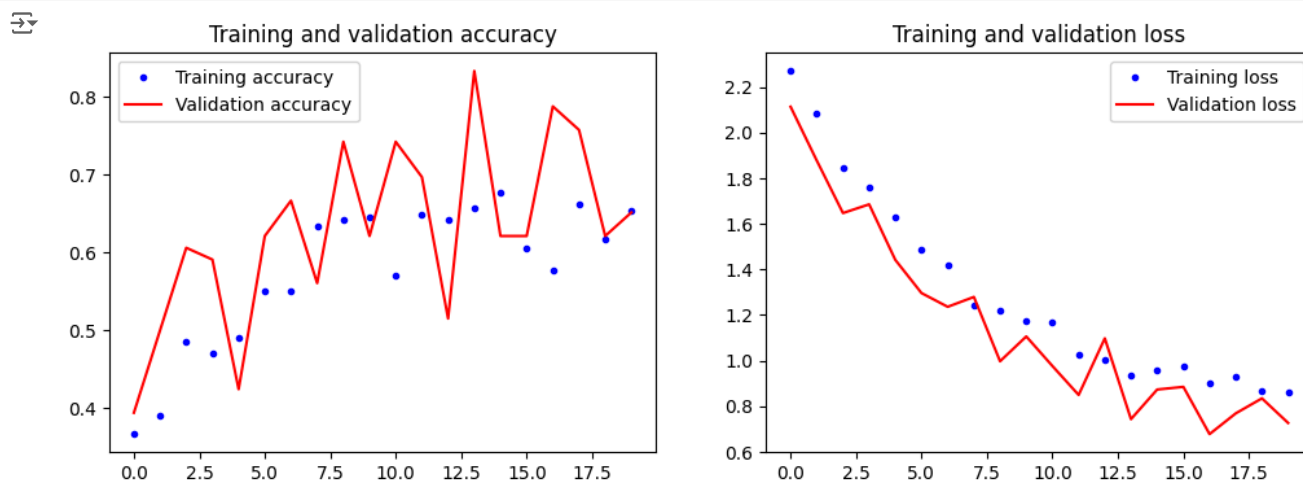
```

	Train	Val
Loss	0.848435	0.678111
Acc	0.677291	0.787879

```

# @title
# Visualizamos el proceso de Entrenamiento
training_plot(history_VGG16_ft)

```



Métricas de Clasificación y Matriz de Confusion

```

# @title
# Calculamos las metricas de Clasificación

num_of_validation_samples = test_generator.samples

Y_pred_VGG16F = model.predict(test_generator, steps=num_of_validation_samples // batch_size1 + 1)

```

```

y_pred_VGG16F = np.argmax(Y_pred_res, axis=1)
print('Matriz de Confusión')
conf_matrix_res = confusion_matrix(test_generator.classes, y_pred_VGG16F)
cm_res = np.array2string(conf_matrix_res)
print(conf_matrix_res)
print("=====")
print('Resumen de Clasificación')
class_rep_res = classification_report(test_generator.classes, y_pred_resnet, target_names=target_names)
print(class_rep_res)

```

3/3 1s 138ms/step

Matriz de Confusión

```

[[23  2  1]
 [ 0 20  0]
 [ 0 19  1]]

```

Resumen de Clasificación

	precision	recall	f1-score	support
Covid	1.00	0.88	0.94	26
Normal	0.49	1.00	0.66	20
Viral Pneumonia	0.50	0.05	0.09	20
accuracy			0.67	66
macro avg	0.66	0.64	0.56	66
weighted avg	0.69	0.67	0.60	66

```

# @title
# Ploteamos la Matriz de Clasificación
print('Matriz de Confusión')
conf_matrix_res = confusion_matrix(test_generator.classes, y_pred_VGG16F)
plot_confusion_matrix(conf_matrix_res, classes=target_names, title='Matriz de Confusión - VGG16 Fine Tuning')

```

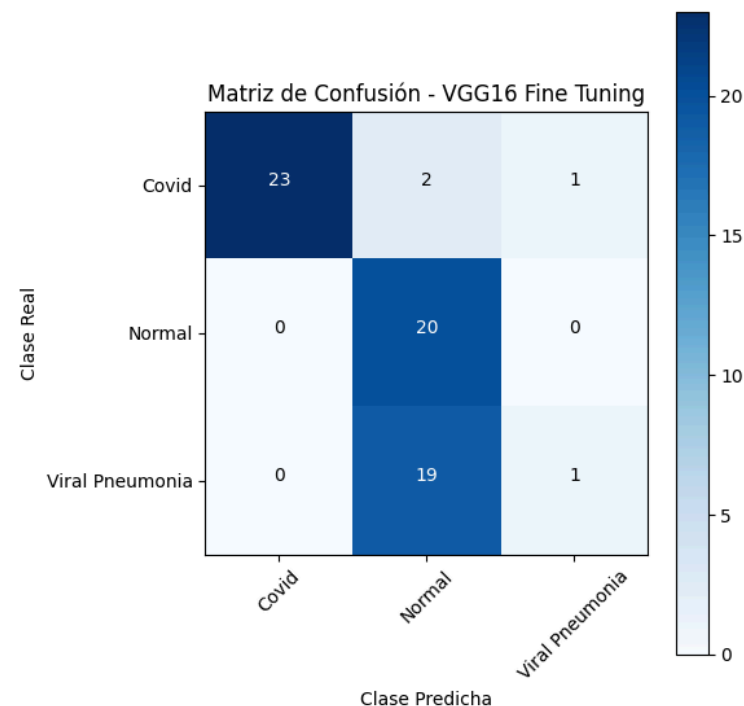
Matriz de Confusión

Matriz de Confusion Normalizada

```

[[23  2  1]
 [ 0 20  0]
 [ 0 19  1]]

```



Análisis de Resultados del VGG16 con Fine Tuning

	Train	Val
Loss	0.848435	0.678111
Acc	0.677291	0.787879

Análisis gráficos:

En el gráfico de Accuracy podemos ver que los datos de entrenamiento tienen una tendencia en crecimiento con baja fluctuación, mientras que los datos de Evaluación muestran una fuerte fluctuación. Hemos intentado mejorar este comportamiento, que creemos se debe a sobreajuste y quizás sea necesario congelar algunas capas.

El gráfico de "perdida" muestra una tendencia decreciente monotonica. Para los datos de Validación se observa una tendencia decreciente con algo de fluctuación pero aceptable.

Conclusion

Usando un bloque de salida de 64 neuronas con regularizacion L2 y dropout hemos logrado un **Accuracy de 78.8%** empleando Fine Tuning en VGG16.

✓ 5.0 Modelo 3 - ResNet50

✓ 5.1. ResNet - Transfer Learning

En esta seccion entrenaremos el modelo ResNET con los pesos del modelo.

Primero lo hacemos con Transfer Learning, congelando todas las capas, y luego con Fine Tuning. Usaremos el mismo bloque de salida que en el modelo VGG16.

```
# @title
from keras.applications import ResNet50

# Cargar el modelo ResNet50 pre-entrenado sin la capa de clasificación
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Congelamos las capas del modelo base
for layer in base_model.layers:
    layer.trainable = False

# Definimos el modelo secuencial sobre el modelo base
resnet1_model = Sequential()
resnet1_model.add(base_model) # Añadimos el modelo base
resnet1_model.add(GlobalAveragePooling2D())

# Bloque de salida
resnet1_model.add(Flatten()) # Aplanamos las salidas
resnet1_model.add(BatchNormalization())
resnet1_model.add(Dense(256, activation='relu', kernel_regularizer='l2'))
resnet1_model.add(Dropout(0.05))
resnet1_model.add(Dense(128, activation='relu', kernel_regularizer='l2'))
resnet1_model.add(Dropout(0.2))
resnet1_model.add(Dense(len(train_generator.class_indices), activation='softmax')) # Capa de salida con número de clases

# Compilar el modelo
resnet1_model.compile(optimizer=Adam(learning_rate=0.005),
                      loss='categorical_crossentropy',
                      metrics=['accuracy'])

# Callback para guardar el mejor modelo basado en val_accuracy
checkpoint = ModelCheckpoint(
    filepath='model/ResNet50T_best.keras', # Ruta donde se guardará el mejor modelo
    monitor='val_accuracy', # Monitorea la precisión en el conjunto de validación
    verbose=1, # Mensajes de progreso
    mode='max', # Guarda el modelo con la precisión más alta
    save_best_only=True # Solo guarda el modelo si es el mejor encontrado hasta ahora
)

# Callback para detener el entrenamiento tempranamente si no mejora el desempeño
early_stopping = EarlyStopping(
    monitor='val_loss', # Monitorea la pérdida en el conjunto de validación
    patience=5, # Detiene el entrenamiento después de 5 épocas sin mejora
    restore_best_weights=True # Restaura los pesos del mejor modelo al detenerse
)
```

Entrenamos el modelo

```
# @title
%%time
resnet1_training = resnet1_model.fit(
    train_generator,
    validation_data=test_generator,
    epochs=25,
    callbacks=[checkpoint, early_stopping],
    shuffle=False,
    verbose=2)
```

↗ Epoch 1/25

```
Epoch 1: val_accuracy improved from -inf to 0.39394, saving model to model/ResNet50T_best.keras
16/16 - 28s - 2s/step - accuracy: 0.5857 - loss: 5.0668 - val_accuracy: 0.3939 - val_loss: 3.5080
Epoch 2/25
```

```
Epoch 2: val_accuracy did not improve from 0.39394
16/16 - 9s - 547ms/step - accuracy: 0.6295 - loss: 2.9931 - val_accuracy: 0.3030 - val_loss: 3.5745
```

Epoch 3/25

Epoch 3: val_accuracy did not improve from 0.39394

16/16 - 9s - 536ms/step - accuracy: 0.6932 - loss: 2.1291 - val_accuracy: 0.3030 - val_loss: 2.1618

Epoch 4/25

Epoch 4: val_accuracy did not improve from 0.39394

16/16 - 8s - 531ms/step - accuracy: 0.6813 - loss: 1.6375 - val_accuracy: 0.3030 - val_loss: 1.9130

Epoch 5/25

Epoch 5: val_accuracy did not improve from 0.39394

16/16 - 8s - 530ms/step - accuracy: 0.7131 - loss: 1.4205 - val_accuracy: 0.3030 - val_loss: 1.7387

Epoch 6/25

Epoch 6: val_accuracy did not improve from 0.39394

16/16 - 9s - 535ms/step - accuracy: 0.6853 - loss: 1.2532 - val_accuracy: 0.3030 - val_loss: 1.5699

Epoch 7/25

Epoch 7: val_accuracy did not improve from 0.39394

16/16 - 8s - 529ms/step - accuracy: 0.6813 - loss: 1.1685 - val_accuracy: 0.3485 - val_loss: 1.5568

Epoch 8/25

Epoch 8: val_accuracy did not improve from 0.39394

16/16 - 8s - 528ms/step - accuracy: 0.6972 - loss: 1.1200 - val_accuracy: 0.3030 - val_loss: 1.4240

Epoch 9/25

Epoch 9: val_accuracy did not improve from 0.39394

16/16 - 8s - 524ms/step - accuracy: 0.7171 - loss: 0.9977 - val_accuracy: 0.3182 - val_loss: 1.4472

Epoch 10/25

Epoch 10: val_accuracy did not improve from 0.39394

16/16 - 8s - 530ms/step - accuracy: 0.7092 - loss: 0.9822 - val_accuracy: 0.3030 - val_loss: 1.3415

Epoch 11/25

Epoch 11: val_accuracy did not improve from 0.39394

16/16 - 8s - 525ms/step - accuracy: 0.6972 - loss: 0.9800 - val_accuracy: 0.3030 - val_loss: 1.3601

Epoch 12/25

Epoch 12: val_accuracy did not improve from 0.39394

16/16 - 8s - 523ms/step - accuracy: 0.7052 - loss: 0.9180 - val_accuracy: 0.3030 - val_loss: 1.3702

Epoch 13/25

Epoch 13: val_accuracy did not improve from 0.39394

16/16 - 9s - 533ms/step - accuracy: 0.6892 - loss: 0.9808 - val_accuracy: 0.3030 - val_loss: 1.2602

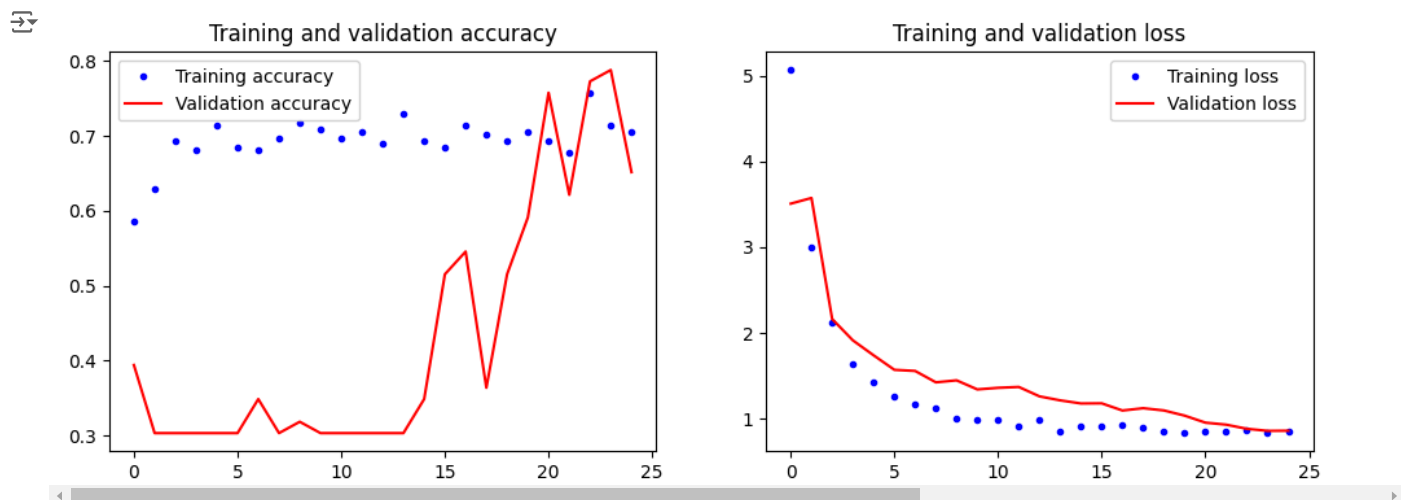
Epoch 14/25

Epoch 14: val_accuracy did not improve from 0.39394

16/16 - 9s - 536ms/step - accuracy: 0.7291 - loss: 0.8535 - val_accuracy: 0.3030 - val_loss: 1.2136

Epoch 15/25

```
# @title
# Visualizamos el proceso de entrenamiento
training_plot(resnet1_training)
```



```
# Resumen de Resultados
train_result = resnet1_model.evaluate(train_generator)
test_result = resnet1_model.evaluate(test_generator)

ResNet50T_df = pd.DataFrame(zip(train_result, test_result), columns=['Train', 'Val'], index=['Loss', 'Acc'])
ResNet50T_df
```


16/16 ————— 9s 594ms/step - accuracy: 0.6444 - loss: 0.9349
 5/5 ————— 1s 154ms/step - accuracy: 0.8173 - loss: 0.8361

	Train	Val
Loss	0.907734	0.858087
Acc	0.689243	0.787879

Metricas de Clasificación y Matriz de Confusion

```
# @title
# Calculamos las metricas de Clasificación

num_of_validation_samples = test_generator.samples

Y_pred_res1 = resnet1_model.predict(test_generator, steps=num_of_validation_samples + 1)
y_pred_resnet1 = np.argmax(Y_pred_res1, axis=1)
print('Matriz de Confusión')
conf_matrix_res = confusion_matrix(test_generator.classes, y_pred_resnet1)
cm_res = np.array2string(conf_matrix_res)
print(conf_matrix_res)
print("=====")
print('Resumen de Clasificación')
class_rep_res = classification_report(test_generator.classes, y_pred_resnet, target_names=target_names)
print(class_rep_res)
```

67/67 ————— 7s 54ms/step

Matriz de Confusión

```
[[21  4  1]
 [ 0 20  0]
 [ 0  9 11]]
```

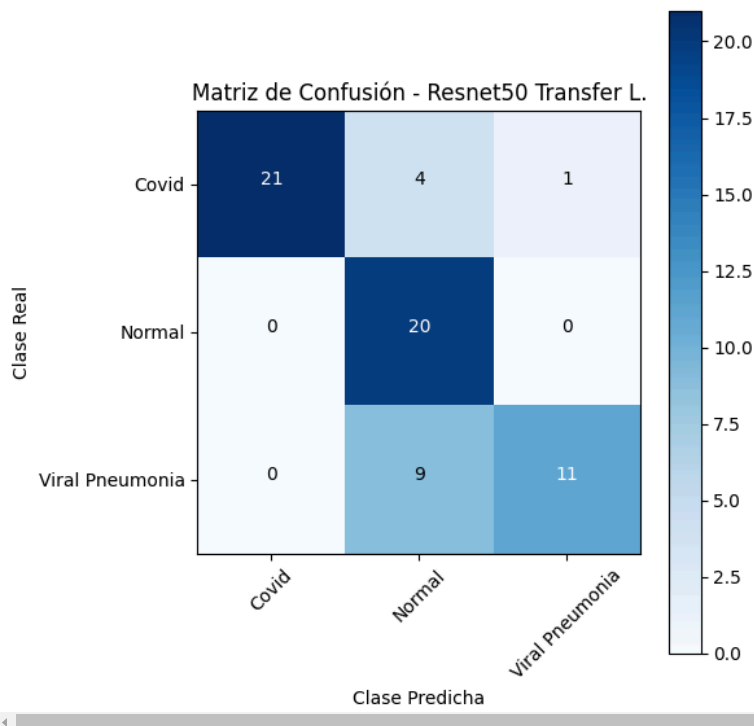
=====
 Resumen de Clasificación

	precision	recall	f1-score	support
Covid	1.00	0.88	0.94	26
Normal	0.49	1.00	0.66	20
Viral Pneumonia	0.50	0.05	0.09	20
accuracy			0.67	66
macro avg	0.66	0.64	0.56	66
weighted avg	0.69	0.67	0.60	66

```
# @title
# Ploteamos la Matriz de Clasificación
print('Matriz de Confusión')
conf_matrix_res = confusion_matrix(test_generator.classes, y_pred_resnet1)
plot_confusion_matrix(conf_matrix_res, classes=target_names, title='Matriz de Confusión - Resnet50 Transfer L.')
```

Matriz de Confusión
Matriz de Confusion Normalizada

```
[[21  4  1]
 [ 0 20  0]
 [ 0  9 11]]
```



Analisis de Resultados del ResNet50 Transfer Learning

	Train	Val
Loss	0.907734	0.858087
Acc	0.689243	0.787879

Analisis graficos Para los datos de entrenamiento la curva de accuracy ha mostrado comenzar con una tendencia creciente para estabilizarse en pocos epochs en torno al 69%. Los datos de Validacion han tenido un comportamiento mas Fluctuante y a la vez explosivo. Para la curva de perdida ambos datos muestran un comportamiento decreciente con vistas a converger. Por parte de los datos de entrenamiento lo hacen monotonicamente y los datos de entrenamiento lo hacen con una leve fluctuacion.

Conclusion

No obstante hemos intentado usar el mismo bloque de salida que el modelo de Transfer Learning para VGG16, este se ha comportado muy mal. Finalmente hemos llegado a usar un bloque menos potencia, pero tambien menos dropout. Con esta configuración hemos llegado a tener un **Accuracy de 78.7%**.

5.2 Resnet - Fine Tune

A continuación implementaremos la tecnica de Fine Tuning con Resnet50. Usaremos el mismo bloque de salida que en el modelo VGG16.

```
# Modelo base ResNet50
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Congelamos las primeras capas del modelo base
for layer in base_model.layers[:100]: # Ajustamos el índice según el tamaño de los datos
    layer.trainable = False

# Definimos el modelo final
model_resnet2 = Sequential([
    base_model,
    GlobalAveragePooling2D(),

    Flatten(),
    Dense(32, activation='relu'), # , kernel_regularizer='l2'
    Dropout(0.5), # Aumenta dropout
    Dense(len(train_generator.class_indices), activation='softmax')
])

# Compilamos el modelo
model_resnet2.compile(optimizer=Adam(learning_rate=0.0003),
                      loss='categorical_crossentropy',
```

```
metrics=['accuracy'])
```

```
from tensorflow.keras.callbacks import ReduceLRonPlateau

# Callback para guardar el mejor modelo basado en val_accuracy
checkpoint = ModelCheckpoint(
    filepath='model/ResNet50F_best.keras', # Ruta donde se guardará el mejor modelo
    monitor='val_accuracy',               # Monitorea la precisión en el conjunto de validación
    verbose=1,                           # Mensajes de progreso
    mode='max',                           # Guarda el modelo con la precisión más alta
    save_best_only=True                   # Solo guarda el modelo si es el mejor encontrado hasta ahora
)

# Callback para detener el entrenamiento tempranamente si no mejora el desempeño
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

# Callback para ,anejar la tasa de aprendizaje
reduce_lr = ReduceLRonPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr=1e-6, verbose=1)
```

Entrenamos el Modelo

```
# @title
%%time
resnet_training2 = model_resnet2.fit(
    train_generator,
    validation_data=test_generator,
    epochs=20,
    callbacks=[checkpoint, early_stopping, reduce_lr],
    shuffle=True,
    verbose=2)
```

```
Epoch 7: val_accuracy improved from 0.60606 to 0.62121, saving model to model/ResNet50F_best.keras
16/16 - 11s - 685ms/step - accuracy: 0.5817 - loss: 0.8560 - val_accuracy: 0.6212 - val_loss: 0.8878 - learning_rate: 1.5000e-04
Epoch 8/20

Epoch 8: val_accuracy improved from 0.62121 to 0.69697, saving model to model/ResNet50F_best.keras
16/16 - 11s - 659ms/step - accuracy: 0.5139 - loss: 0.9104 - val_accuracy: 0.6970 - val_loss: 0.7907 - learning_rate: 1.5000e-04
Epoch 9/20

Epoch 9: val_accuracy did not improve from 0.69697
16/16 - 10s - 604ms/step - accuracy: 0.5697 - loss: 0.8742 - val_accuracy: 0.5303 - val_loss: 0.7559 - learning_rate: 1.5000e-04
Epoch 10/20

Epoch 10: val_accuracy did not improve from 0.69697
16/16 - 9s - 552ms/step - accuracy: 0.5259 - loss: 0.8823 - val_accuracy: 0.5152 - val_loss: 0.8404 - learning_rate: 1.5000e-04
Epoch 11/20

Epoch 11: val_accuracy did not improve from 0.69697
16/16 - 9s - 559ms/step - accuracy: 0.5618 - loss: 0.8453 - val_accuracy: 0.6818 - val_loss: 0.6736 - learning_rate: 1.5000e-04
Epoch 12/20

Epoch 12: val_accuracy did not improve from 0.69697
16/16 - 9s - 551ms/step - accuracy: 0.5657 - loss: 0.8495 - val_accuracy: 0.6212 - val_loss: 1.4547 - learning_rate: 1.5000e-04
Epoch 13/20

Epoch 13: val_accuracy did not improve from 0.69697
16/16 - 9s - 561ms/step - accuracy: 0.5936 - loss: 0.7919 - val_accuracy: 0.6515 - val_loss: 0.6172 - learning_rate: 1.5000e-04
```

```
Epoch 20: val_accuracy did not improve from 0.11212
```

```
16/16 - 9s - 537ms/step - accuracy: 0.6135 - loss: 0.7908 - val_accuracy: 0.6061 - val_loss: 0.8327 - learning_rate: 7.5000e-05
```

```
CPU times: user 3min 36s, sys: 10.6 s, total: 3min 46s
```

```
Wall time: 3min 53s
```

```
train_result = model_resnet2.evaluate(train_generator)
```

```
test_result = model_resnet2.evaluate(test_generator)
```

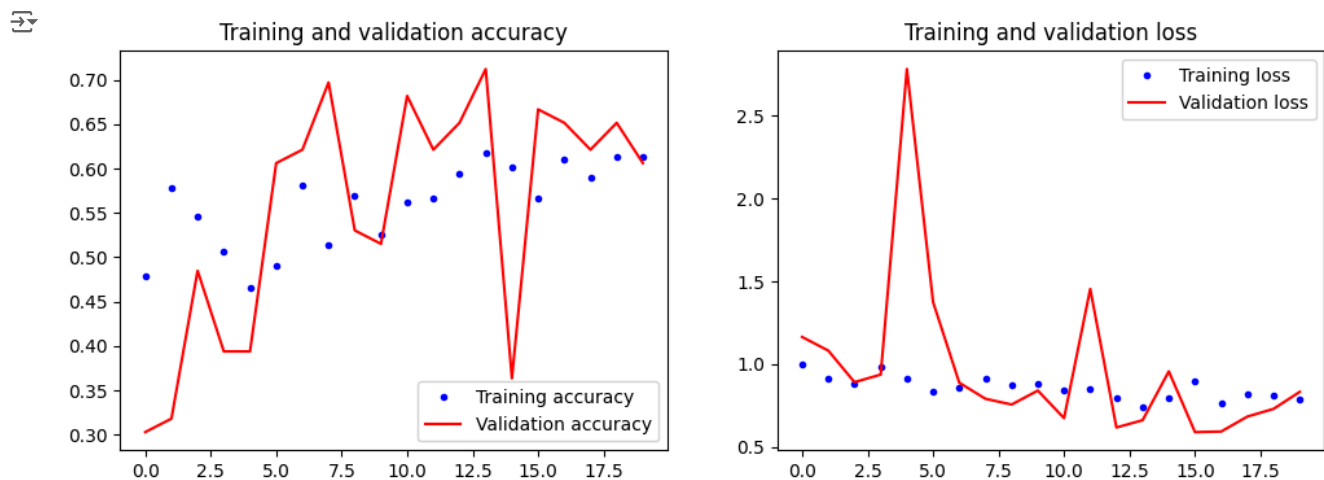
```
ResNet50F_df = pd.DataFrame(zip(train_result, test_result), columns=['Train', 'Val'], index=['Loss', 'Acc'])
```

```
ResNet50F_df
```

```
16/16 ----- 9s 590ms/step - accuracy: 0.6943 - loss: 0.7145
5/5 ----- 1s 158ms/step - accuracy: 0.7665 - loss: 0.4782
```

	Train	Val
Loss	0.787406	0.589307
Acc	0.677291	0.666667

```
# @title
# Visualizamos el proceso de Entrenamiento
training_plot(resnet_training2)
```



Métricas de Clasificación y Matriz de Confusión

```
# @title
# Calculamos las metricas de Clasificación

num_of_validation_samples = test_generator.samples

Y_pred_res2 = model_resnet2.predict(test_generator, steps=num_of_validation_samples + 1)
y_pred_resnet2 = np.argmax(Y_pred_res2, axis=1)
print('Matriz de Confusión')
conf_matrix_res = confusion_matrix(test_generator.classes, y_pred_resnet2)
cm_res = np.array2string(conf_matrix_res)
print(conf_matrix_res)
print("=====")
print('Resumen de Clasificación')
class_rep_res = classification_report(test_generator.classes, y_pred_resnet2, target_names=target_names)
print(class_rep_res)
```

```
67/67 ----- 7s 54ms/step
```

```
Matriz de Confusión
```

```
[[23  2  1]
 [ 0 18  2]
 [ 0 17  3]]
```

```
=====
Resumen de Clasificación
```

	precision	recall	f1-score	support
Covid	1.00	0.88	0.94	26
Normal	0.49	1.00	0.66	20
Viral Pneumonia	0.50	0.05	0.09	20
accuracy			0.67	66
macro avg	0.66	0.64	0.56	66
weighted avg	0.69	0.67	0.60	66

```
# @title
# Ploteamos la Matriz de Clasificación
print('Matriz de Confusión')
conf_matrix_res = confusion_matrix(test_generator.classes, y_pred_resnet2)
plot_confusion_matrix(conf_matrix_res, classes=target_names, title='Matriz de Confusión - Resnet50 Fine Tuning')
```

```
Matriz de Confusión  
Matriz de Confusion Normalizada  
[[23  2  1]  
 [ 0 18  2]  
 [ 0 17  3]]
```

