Open in Colab

# Ejercicio 3-a. Transfer learning de modelos convolucionales

El apartado a del ejercicio 3 es una continuación del ejercicio 2, donde deberás analizar la técnica de transfer learning sobre el dataset de mascarillas. Por tanto, necesitarás hacer uso de lo visto en la práctica 5.3.

## 0. Cabecera: JOSE MARIA GARCIA QUIJADA

## 1. Preparación de los datos para ser usados en Keras

En esta sección, prepararemos el entorno y los datos necesarios para entrenar nuestros modelos de detección de mascarillas faciales.

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Descargamos el conjunto de datos de mascarillas faciales y establecemos la ruta del directorio de datos. Definimos el tamaño de las imágenes a 128x128 píxeles para estandarizar las entradas al modelo.

In [2]:
```
import pathlib
import keras

data_path = keras.utils.get_file(
    "mask_dataset.zip",
    "https://hdvirtual.us.es/discovirt/index.php/s/dijNRMPYfybYd3N/download",
    extract=True,
    archive_format='zip'
)
DATA_FOLDER = pathlib.Path(data_path).parent / 'mask_ds'
IMAGE_SIZE = (128, 128)
```

Utilizamos ImageDataGenerator para cargar y preprocesar las imágenes. Aplicamos una normalización rescalando los valores de los píxeles y dividimos el conjunto de datos en entrenamiento y validación con una proporción de 80/20.

In [ ]:
```
datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2
)

train_data = datagen.flow_from_directory(
    DATA_FOLDER,
    target_size=IMAGE_SIZE,
    batch_size=16,
    subset='training',
    class_mode='binary'
)

val_data = datagen.flow_from_directory(
    DATA_FOLDER,
    target_size=IMAGE_SIZE,
    batch_size=16,
    subset='validation',
    class_mode='binary'
)
```

```
Found 6043 images belonging to 2 classes.
Found 1510 images belonging to 2 classes.
```

## 2. Modelos y configuraciones creados en Keras basados en 3 modelos pre-entrenados

En esta sección, crearemos tres modelos basados en arquitecturas pre-entrenadas populares para clasificación de imágenes: MobileNetV2, EfficientNetB0 y ResNet50.

Definimos una función create_model que toma un modelo base pre-entrenado y le agrega capas adicionales para adaptar el modelo a nuestra tarea de clasificación binaria (con o sin mascarilla). Se seleccionaron los siguientes modelos por sus características:

-**MobileNetV2**: Elegido por su eficiencia y tamaño reducido, ideal para dispositivos móviles y aplicaciones en tiempo real.

-**EfficientNetB0**: Ofrece un buen equilibrio entre precisión y eficiencia computacional gracias a su escalamiento compuesto.

-**ResNet50**: Conocido por su gran profundidad y capacidad para manejar características complejas gracias a las conexiones residuales.

In [4]:
```
from keras.applications import MobileNetV2, EfficientNetB0, ResNet50
from keras.models import Model
from keras.layers import Dense, GlobalAveragePooling2D, Dropout
from keras.optimizers import Adam
from keras import Input

def create_model(base_model):
    base_model.trainable = False
    inputs = Input(shape=(128, 128, 3))
    x = base_model(inputs, training=False)
    x = GlobalAveragePooling2D()(x)
    x = Dropout(0.2)(x)
    outputs = Dense(1, activation='sigmoid')(x)
    model = Model(inputs, outputs)
    return model


mobilenet_model = create_model(MobileNetV2(weights='imagenet', include_top=False, input_shape=(128, 128, 3)))
efficientnet_model = create_model(EfficientNetB0(weights='imagenet', include_top=False, input_shape=(128, 128, 3)))
resnet_model = create_model(ResNet50(weights='imagenet', include_top=False, input_shape=(128, 128, 3)))
```

Inicialmente, congelamos las capas de los modelos base para utilizar las características pre-entrenadas. Sin embargo, decidimos descongelar todas las capas para permitir un ajuste fino completo y mejorar potencialmente el rendimiento en nuestro conjunto de datos específico.

In [5]:
```
for layer in mobilenet_model.layers:
    layer.trainable = True

for layer in efficientnet_model.layers:
    layer.trainable = True

for layer in resnet_model.layers:
    layer.trainable = True
```

## 3. Entrenamiento y evaluación de cada modelo creado

En esta sección, entrenaremos cada modelo y analizaremos sus resultados.

### 3.1. Implementación de métricas personalizadas y funciones auxiliares

Implementamos una métrica personalizada de F1-score para evaluar mejor el rendimiento de los modelos en la clasificación binaria, especialmente útil en casos de desequilibrio de clases.

In [ ]:
```
import tensorflow as tf
import matplotlib.pyplot as plt

class F1Score(tf.keras.metrics.Metric):
    def __init__(self, name='f1_score', threshold=0.5, **kwargs):
        super(F1Score, self).__init__(name=name, **kwargs)
        self.threshold = threshold
        self.tp = self.add_weight(name='tp', initializer='zeros')
        self.fp = self.add_weight(name='fp', initializer='zeros')
        self.fn = self.add_weight(name='fn', initializer='zeros')

    def update_state(self, y_true, y_pred, sample_weight=None):

        y_pred = tf.cast(y_pred > self.threshold, tf.float32)
        y_true = tf.cast(y_true, tf.float32)

        tp = tf.reduce_sum(tf.cast(y_true * y_pred, tf.float32))
        fp = tf.reduce_sum(tf.cast((1 - y_true) * y_pred, tf.float32))
        fn = tf.reduce_sum(tf.cast(y_true * (1 - y_pred), tf.float32))

        self.tp.assign_add(tp)
        self.fp.assign_add(fp)
        self.fn.assign_add(fn)

    def result(self):
        precision = self.tp / (self.tp + self.fp + tf.keras.backend.epsilon())
        recall = self.tp / (self.tp + self.fn + tf.keras.backend.epsilon())
        f1 = 2 * precision * recall / (precision + recall + tf.keras.backend.epsilon())
        return f1

    def reset_states(self):
        self.tp.assign(0)
        self.fp.assign(0)
        self.fn.assign(0)
```

Creamos una función plot_training para visualizar las métricas durante el entrenamiento y facilitar el análisis de resultados.

In [ ]:
```
def plot_training(history, model_name):
    plt.figure(figsize=(18, 12))

    plt.subplot(2, 2, 1)
    plt.plot(history.history['loss'], label='Entrenamiento')
    plt.plot(history.history['val_loss'], label='Validación')
    plt.title(f'Pérdida durante el entrenamiento - {model_name}')
    plt.xlabel('Epochs')
    plt.ylabel('Pérdida')
    plt.legend()
```

```
    plt.subplot(2, 2, 2)
    plt.plot(history.history['accuracy'], label='Entrenamiento')
    plt.plot(history.history['val_accuracy'], label='Validación')
    plt.title(f'Precisión durante el entrenamiento - {model_name}')
    plt.xlabel('Epochs')
    plt.ylabel('Precisión')
    plt.legend()

    plt.subplot(2, 2, 3)
    plt.plot(history.history['precision'], label='Entrenamiento')
    plt.plot(history.history['val_precision'], label='Validación')
    plt.title(f'Precision durante el entrenamiento - {model_name}')
    plt.xlabel('Epochs')
    plt.ylabel('Precision')
    plt.legend()

    plt.subplot(2, 2, 4)
    plt.plot(history.history['recall'], label='Entrenamiento')
    plt.plot(history.history['val_recall'], label='Validación')
    plt.title(f'Recall durante el entrenamiento - {model_name}')
    plt.xlabel('Epochs')
    plt.ylabel('Recall')
    plt.legend()

    plt.tight_layout()
    plt.show()

    if 'f1_score' in history.history:
        plt.figure(figsize=(6, 4))
        plt.plot(history.history['f1_score'], label='Entrenamiento')
        plt.plot(history.history['val_f1_score'], label='Validación')
        plt.title(f'F1-Score durante el entrenamiento - {model_name}')
        plt.xlabel('Epochs')
        plt.ylabel('F1-Score')
        plt.legend()
        plt.show()
```

3.2. Entrenamiento de MobileNetV2

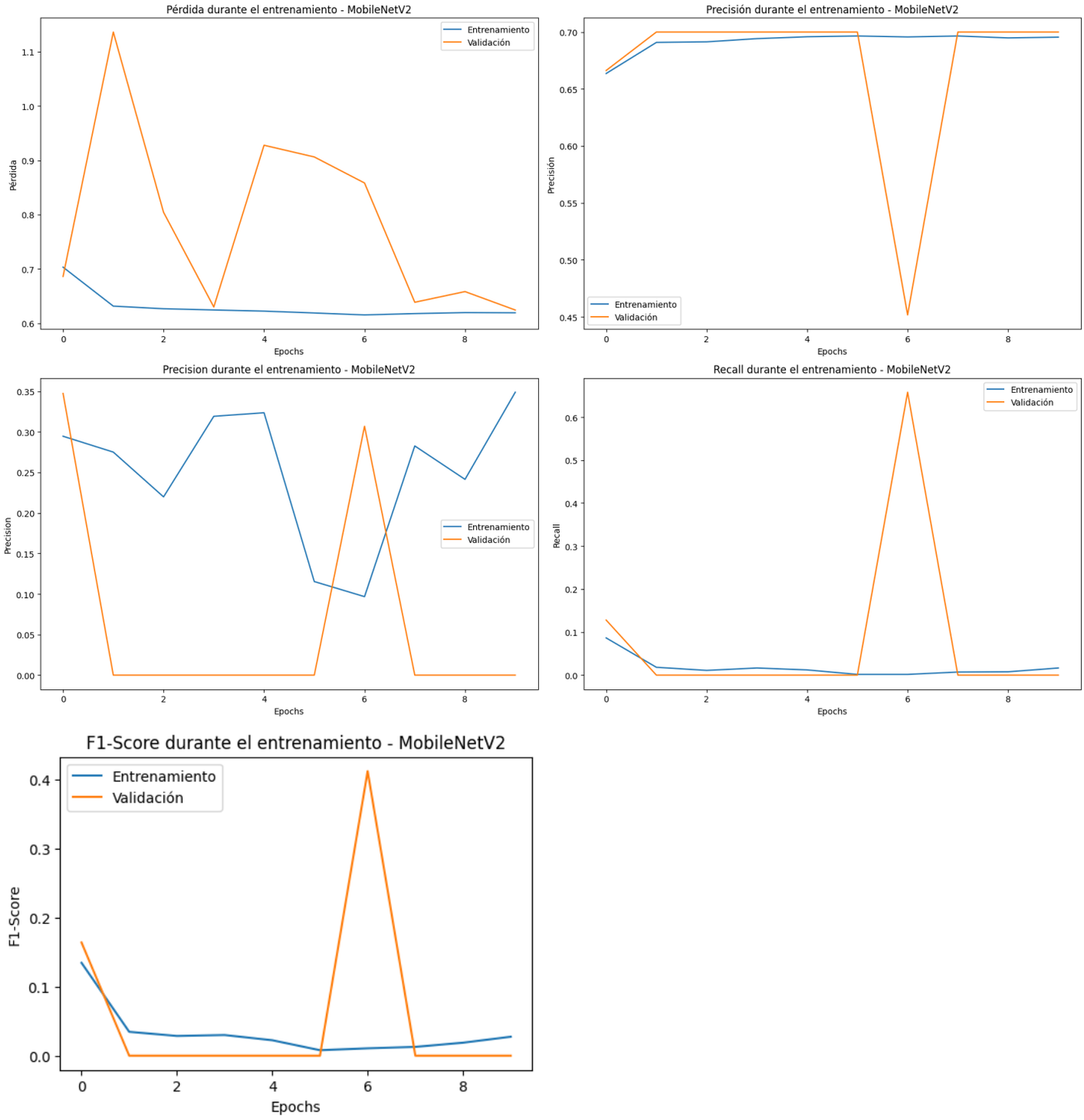
Compilamos el modelo MobileNetV2 utilizando el optimizador Adam y la función de pérdida binaria. Incluimos métricas adicionales como precisión, recall y el F1-score personalizado. Entrenamos el modelo durante 10 épocas.

In [ ]:
```
f1 = F1Score(name='f1_score', threshold=0.5)

mobilenet_model.compile(
    optimizer='Adam',
    loss='binary_crossentropy',
    metrics=[
        'accuracy',
        tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall'),
        f1
    ]
)

mobilenet_history = mobilenet_model.fit(
    train_data,
    validation_data=val_data,
    epochs=10
)
```

```
Epoch 1/10
C:\Users\Jm\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\LocalCache\local-packages\Python312\site-packages
\keras\src\trainers\data_adapters\py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)
` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`,
as they will be ignored.
  self._warn_if_super_not_called()
320/378 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6s 111ms/step - accuracy: 0.6182 - f1_score: 0.2203 - loss: 0.8352 - precision: 0.2935 - recal
l: 0.1818
C:\Users\Jm\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\LocalCache\local-packages\Python312\site-packages
\PIL\Image.py:981: UserWarning: Palette images with Transparency expressed in bytes should be converted to RGBA images
  warnings.warn(
378/378 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 63s 129ms/step - accuracy: 0.6253 - f1_score: 0.2080 - loss: 0.8152 - precision: 0.2933 - reca
ll: 0.1678 - val_accuracy: 0.6662 - val_f1_score: 0.1639 - val_loss: 0.6861 - val_precision: 0.3473 - val_recall: 0.1280
Epoch 2/10
378/378 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 40s 107ms/step - accuracy: 0.6866 - f1_score: 0.0414 - loss: 0.6311 - precision: 0.2306 - reca
ll: 0.0186 - val_accuracy: 0.7000 - val_f1_score: 0.0000e+00 - val_loss: 1.1361 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 3/10
378/378 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 40s 106ms/step - accuracy: 0.6994 - f1_score: 0.0230 - loss: 0.6218 - precision: 0.1938 - reca
ll: 0.0076 - val_accuracy: 0.7000 - val_f1_score: 0.0000e+00 - val_loss: 0.8042 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 4/10
378/378 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 40s 106ms/step - accuracy: 0.6958 - f1_score: 0.0335 - loss: 0.6220 - precision: 0.3267 - reca
ll: 0.0207 - val_accuracy: 0.7000 - val_f1_score: 0.0000e+00 - val_loss: 0.6298 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 5/10
378/378 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 40s 106ms/step - accuracy: 0.7029 - f1_score: 0.0186 - loss: 0.6161 - precision: 0.2728 - reca
ll: 0.0106 - val_accuracy: 0.7000 - val_f1_score: 0.0000e+00 - val_loss: 0.9277 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 6/10
378/378 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 40s 106ms/step - accuracy: 0.7025 - f1_score: 0.0066 - loss: 0.6153 - precision: 0.1023 - reca
ll: 0.0011 - val_accuracy: 0.7000 - val_f1_score: 0.0000e+00 - val_loss: 0.9061 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 7/10
378/378 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 41s 109ms/step - accuracy: 0.7035 - f1_score: 0.0115 - loss: 0.6084 - precision: 0.0886 - reca
ll: 0.0021 - val_accuracy: 0.4517 - val_f1_score: 0.4119 - val_loss: 0.8582 - val_precision: 0.3069 - val_recall: 0.6578
Epoch 8/10
378/378 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 40s 107ms/step - accuracy: 0.7030 - f1_score: 0.0141 - loss: 0.6136 - precision: 0.2053 - reca
ll: 0.0057 - val_accuracy: 0.7000 - val_f1_score: 0.0000e+00 - val_loss: 0.6386 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 9/10
378/378 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 41s 107ms/step - accuracy: 0.7054 - f1_score: 0.0187 - loss: 0.6071 - precision: 0.1829 - reca
ll: 0.0069 - val_accuracy: 0.7000 - val_f1_score: 0.0000e+00 - val_loss: 0.6582 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 10/10
378/378 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 41s 107ms/step - accuracy: 0.7032 - f1_score: 0.0244 - loss: 0.6141 - precision: 0.3032 - reca
ll: 0.0139 - val_accuracy: 0.7000 - val_f1_score: 0.0000e+00 - val_loss: 0.6242 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
```

Visualizamos las métricas de entrenamiento para analizar el comportamiento del modelo.

In [9]:
```
plot_training(mobilenet_history, 'MobileNetV2')
```




Análisis de resultados de MobileNetV2:

Observamos que la precisión en el conjunto de entrenamiento es del 70.32%, mientras que el F1-score es extremadamente bajo (0.0244). La precisión y el recall también son bajos, indicando que el modelo tiene dificultades para distinguir entre las clases. En el conjunto de validación, el F1-score es cero, lo que sugiere que el modelo no está generalizando bien.

3.3 Entrenamiento de EfficientNetB0

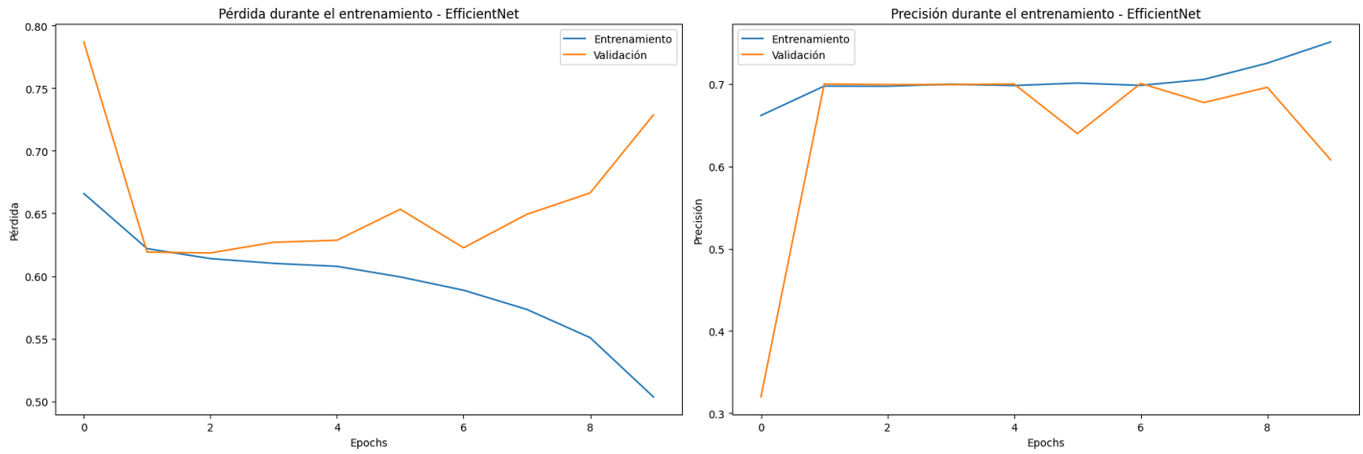
Procedemos de manera similar con EfficientNetB0, compilando y entrenando el modelo con las mismas configuraciones.

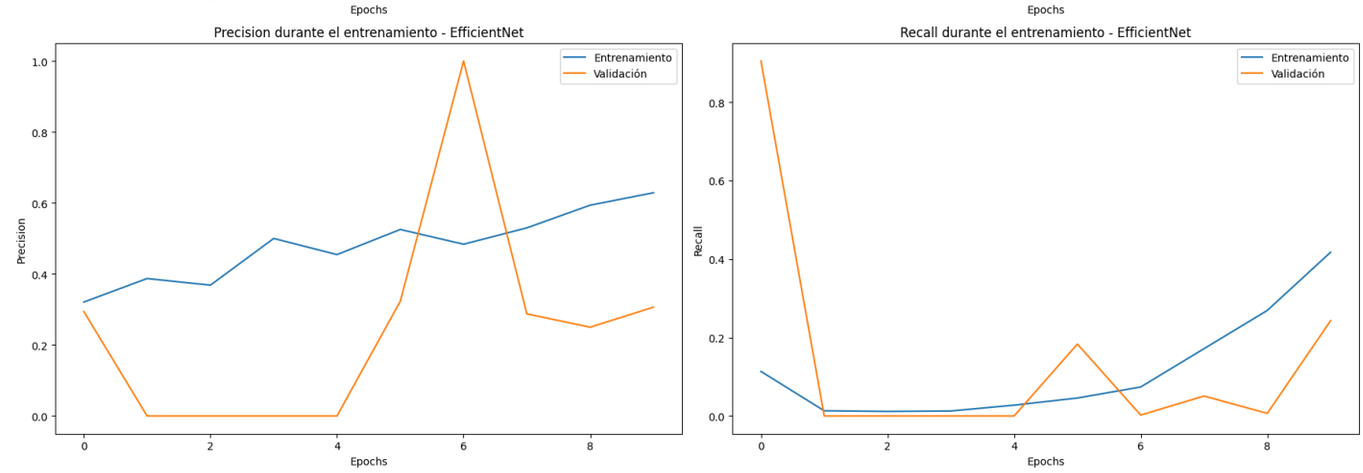
In [ ]:
```
f1 = F1Score(name='f1_score', threshold=0.5)

efficientnet_model.compile(
    optimizer='Adam',
    loss='binary_crossentropy',
    metrics=[
        'accuracy',
        tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall'),
        f1
    ]
)

efficientnet_history = efficientnet_model.fit(
    train_data,
    validation_data=val_data,
    epochs=10
)
```

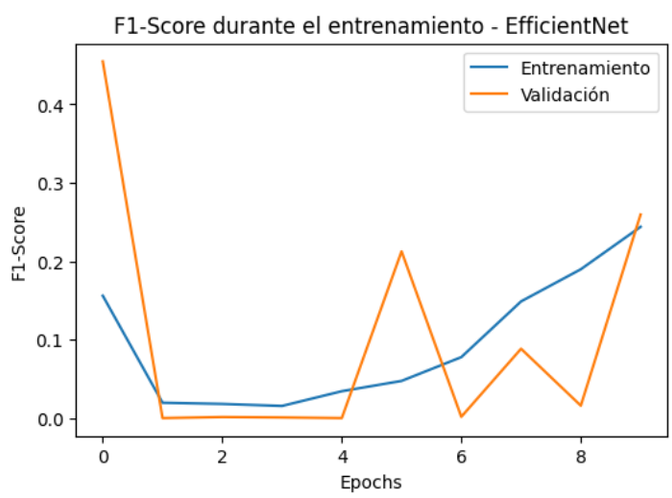
```
Epoch 1/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 88s 177ms/step - accuracy: 0.6402 - f1_score: 0.2133 - loss: 0.6897 - precision: 0.3112 - reca
ll: 0.1844 - val_accuracy: 0.3199 - val_f1_score: 0.4547 - val_loss: 0.7870 - val_precision: 0.2941 - val_recall: 0.9051
Epoch 2/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 63s 165ms/step - accuracy: 0.6941 - f1_score: 0.0441 - loss: 0.6241 - precision: 0.4090 - reca
ll: 0.0363 - val_accuracy: 0.7000 - val_f1_score: 0.0000e+00 - val_loss: 0.6192 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 3/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 63s 166ms/step - accuracy: 0.7040 - f1_score: 0.0168 - loss: 0.6055 - precision: 0.3707 - reca
ll: 0.0118 - val_accuracy: 0.6993 - val_f1_score: 0.0014 - val_loss: 0.6185 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 4/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 63s 166ms/step - accuracy: 0.6999 - f1_score: 0.0153 - loss: 0.6090 - precision: 0.4637 - reca
ll: 0.0136 - val_accuracy: 0.6993 - val_f1_score: 8.2827e-04 - val_loss: 0.6270 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 5/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 63s 166ms/step - accuracy: 0.6897 - f1_score: 0.0450 - loss: 0.6124 - precision: 0.4624 - reca
ll: 0.0367 - val_accuracy: 0.7000 - val_f1_score: 0.0000e+00 - val_loss: 0.6286 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 6/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 62s 164ms/step - accuracy: 0.7050 - f1_score: 0.0529 - loss: 0.5956 - precision: 0.5220 - reca
ll: 0.0552 - val_accuracy: 0.6397 - val_f1_score: 0.2125 - val_loss: 0.6534 - val_precision: 0.3230 - val_recall: 0.1832
Epoch 7/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 62s 164ms/step - accuracy: 0.7193 - f1_score: 0.0863 - loss: 0.5710 - precision: 0.5761 - reca
ll: 0.1065 - val_accuracy: 0.7007 - val_f1_score: 0.0017 - val_loss: 0.6226 - val_precision: 1.0000 - val_recall: 0.0022
Epoch 8/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 63s 166ms/step - accuracy: 0.7101 - f1_score: 0.1557 - loss: 0.5586 - precision: 0.5389 - reca
ll: 0.1924 - val_accuracy: 0.6775 - val_f1_score: 0.0884 - val_loss: 0.6493 - val_precision: 0.2875 - val_recall: 0.0508
Epoch 9/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 63s 167ms/step - accuracy: 0.7347 - f1_score: 0.2015 - loss: 0.5396 - precision: 0.6324 - reca
ll: 0.3019 - val_accuracy: 0.6960 - val_f1_score: 0.0159 - val_loss: 0.6664 - val_precision: 0.2500 - val_recall: 0.0066
Epoch 10/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 63s 168ms/step - accuracy: 0.7677 - f1_score: 0.2355 - loss: 0.4806 - precision: 0.6720 - reca
ll: 0.4293 - val_accuracy: 0.6079 - val_f1_score: 0.2594 - val_loss: 0.7288 - val_precision: 0.3064 - val_recall: 0.2428
```

Visualizamos las métricas de entrenamiento para EfficientNetB0.

In [11]:
```
plot_training(efficientnet_history, 'EfficientNet')
```




Análisis de resultados de EfficientNetB0:

EfficientNetB0 muestra un mejor rendimiento en comparación con MobileNetV2. La precisión en el conjunto de entrenamiento es del 76.77%, y el F1-score es significativamente mayor (0.2355). La precisión y el recall también son más altos, indicando una mejor capacidad del modelo para distinguir entre las clases.

En el conjunto de validación, aunque la precisión baja al 60.79%, el F1-score es de 0.2594, lo que sugiere que el modelo generaliza mejor que MobileNetV2. Esto indica que EfficientNetB0 es más adecuado para este problema.

### 3.4. Entrenamiento de ResNet50

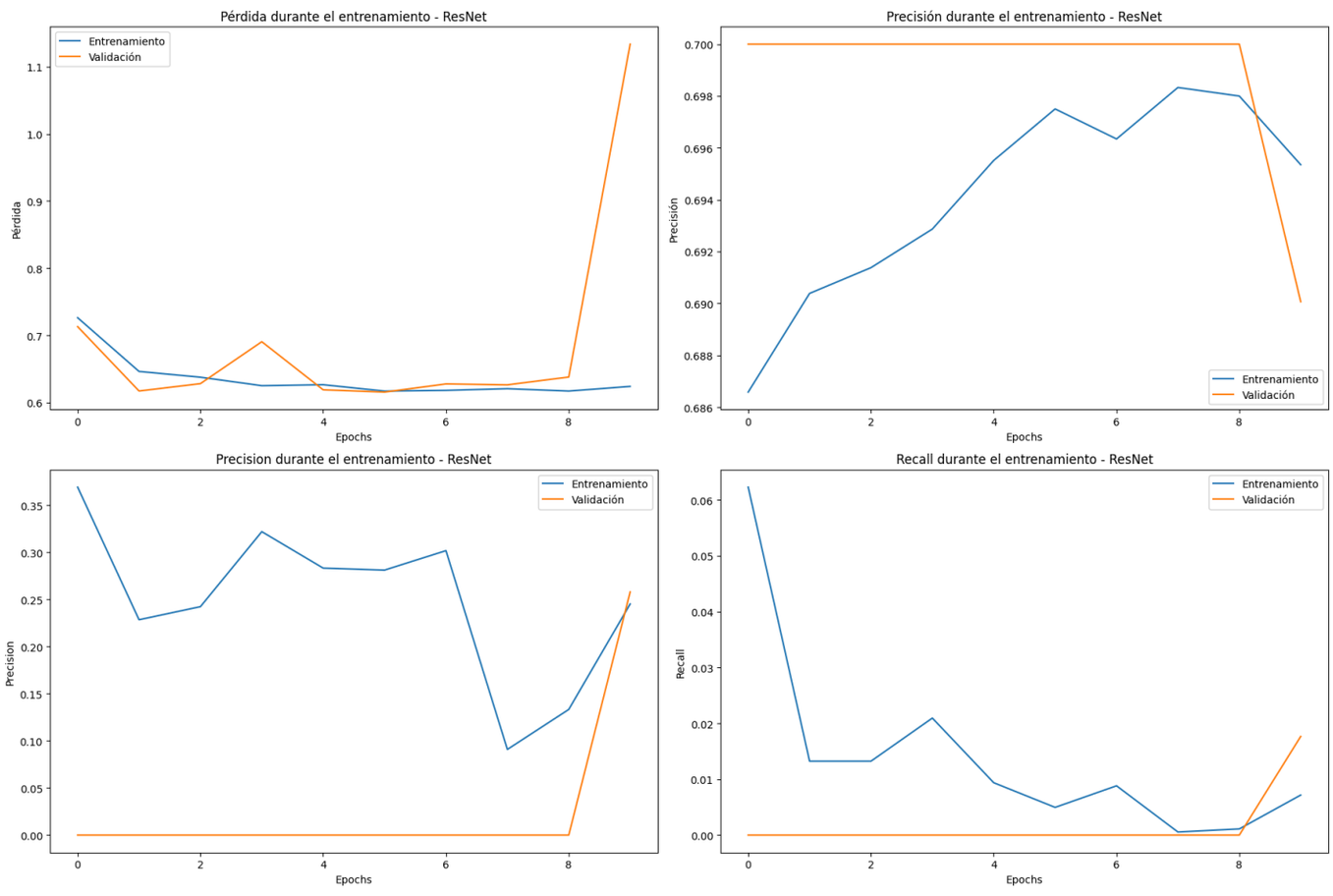
Entrenamos el modelo ResNet50 con las mismas configuraciones.

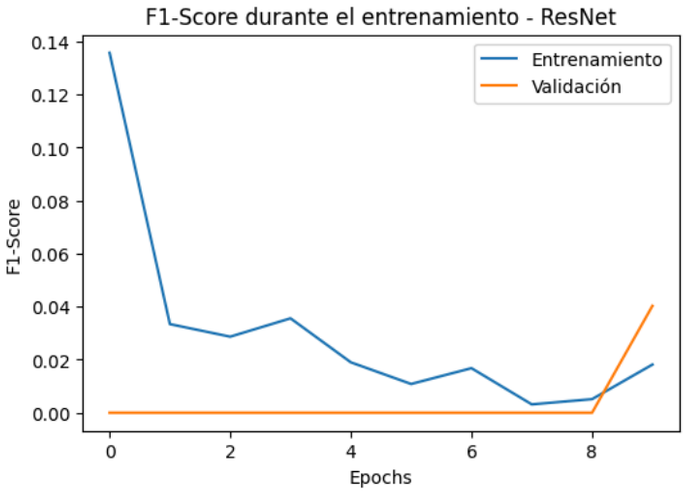
In [ ]:
```
resnet_model.compile(
    optimizer='Adam',
    loss='binary_crossentropy',
    metrics=[
        'accuracy',
        tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall'),
        f1
    ]
)

resnet_history = resnet_model.fit(
    train_data,
    validation_data=val_data,
    epochs=10
)
```

```
Epoch 1/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 130s 294ms/step - accuracy: 0.6614 - f1_score: 0.1876 - loss: 0.8377 - precision: 0.3670 - rec
all: 0.1154 - val_accuracy: 0.7000 - val_f1_score: 0.0000e+00 - val_loss: 0.7130 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 2/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 112s 295ms/step - accuracy: 0.6863 - f1_score: 0.0409 - loss: 0.6561 - precision: 0.1968 - rec
all: 0.0147 - val_accuracy: 0.7000 - val_f1_score: 0.0000e+00 - val_loss: 0.6171 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 3/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 112s 295ms/step - accuracy: 0.6785 - f1_score: 0.0317 - loss: 0.6484 - precision: 0.1809 - rec
all: 0.0107 - val_accuracy: 0.7000 - val_f1_score: 0.0000e+00 - val_loss: 0.6281 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 4/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 112s 295ms/step - accuracy: 0.6963 - f1_score: 0.0360 - loss: 0.6207 - precision: 0.3126 - rec
all: 0.0219 - val_accuracy: 0.7000 - val_f1_score: 0.0000e+00 - val_loss: 0.6906 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 5/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 111s 295ms/step - accuracy: 0.7025 - f1_score: 0.0136 - loss: 0.6157 - precision: 0.1671 - rec
all: 0.0048 - val_accuracy: 0.7000 - val_f1_score: 0.0000e+00 - val_loss: 0.6189 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 6/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 110s 291ms/step - accuracy: 0.6979 - f1_score: 0.0117 - loss: 0.6144 - precision: 0.2568 - rec
all: 0.0057 - val_accuracy: 0.7000 - val_f1_score: 0.0000e+00 - val_loss: 0.6155 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 7/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 111s 294ms/step - accuracy: 0.6975 - f1_score: 0.0206 - loss: 0.6144 - precision: 0.2175 - rec
all: 0.0086 - val_accuracy: 0.7000 - val_f1_score: 0.0000e+00 - val_loss: 0.6277 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 8/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 109s 288ms/step - accuracy: 0.6947 - f1_score: 0.0049 - loss: 0.6226 - precision: 0.0329 - rec
all: 2.4610e-04 - val_accuracy: 0.7000 - val_f1_score: 0.0000e+00 - val_loss: 0.6263 - val_precision: 0.0000e+00 - val_recall: 0.0000e+0
0
Epoch 9/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 109s 287ms/step - accuracy: 0.6992 - f1_score: 0.0056 - loss: 0.6136 - precision: 0.1357 - rec
all: 0.0013 - val_accuracy: 0.7000 - val_f1_score: 0.0000e+00 - val_loss: 0.6379 - val_precision: 0.0000e+00 - val_recall: 0.0000e+00
Epoch 10/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 109s 288ms/step - accuracy: 0.6931 - f1_score: 0.0069 - loss: 0.6217 - precision: 0.1079 - rec
all: 0.0029 - val_accuracy: 0.6901 - val_f1_score: 0.0403 - val_loss: 1.1337 - val_precision: 0.2581 - val_recall: 0.0177
```

Visualizamos las métricas de entrenamiento para ResNet50.

In [13]:
```
plot_training(resnet_history, 'ResNet')
```






Análisis de resultados de ResNet50:

ResNet50 muestra un rendimiento inferior en comparación con EfficientNetB0. La precisión en el conjunto de entrenamiento es del 69.31%, pero el F1-score es extremadamente bajo (0.0069), similar al de MobileNetV2. La precisión y el recall son muy bajos, lo que indica que el modelo no está capturando las características distintivas necesarias para clasificar correctamente las imágenes.

En el conjunto de validación, el F1-score es de 0.0403, lo que refuerza la conclusión de que ResNet50 no está funcionando bien para este conjunto de datos específico.

3.5. Análisis comparativo de los modelos

Comparación de resultados:

MobileNetV2:

Accuracy 70%, F1-score muy bajo. El modelo no está capturando las características necesarias y tiene un rendimiento pobre.

EfficientNetB0:

Accuracy 77%, F1-score razonable (0.2355). F1-score de 0.2594. Mejor equilibrio entre precisión y recall, generaliza mejor y muestra un rendimiento aceptable.

ResNet50:

Accuracy ~69%, F1-score muy bajo. F1-score de 0.0403. Similar a MobileNetV2, no es adecuado para este problema con la configuración actual.

Basándonos en los resultados, EfficientNetB0 es el modelo que ofrece el mejor rendimiento tanto en el conjunto de entrenamiento como en el de validación. Presenta un equilibrio razonable entre precisión y recall y un F1-score aceptable, lo que indica una mejor capacidad para generalizar y distinguir entre las clases.

## 4. Selección del mejor modelo y fine-tuning de capas

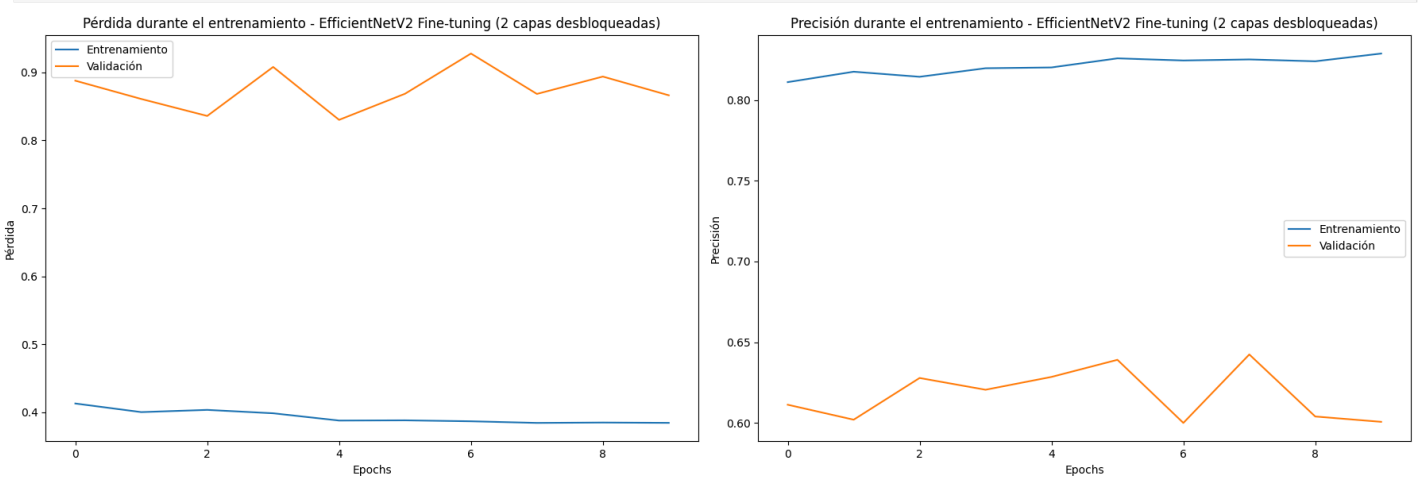
En esta sección, realizaremos un ajuste fino (fine-tuning) del modelo EfficientNetB0 desbloqueando diferentes cantidades de capas para mejorar su rendimiento.

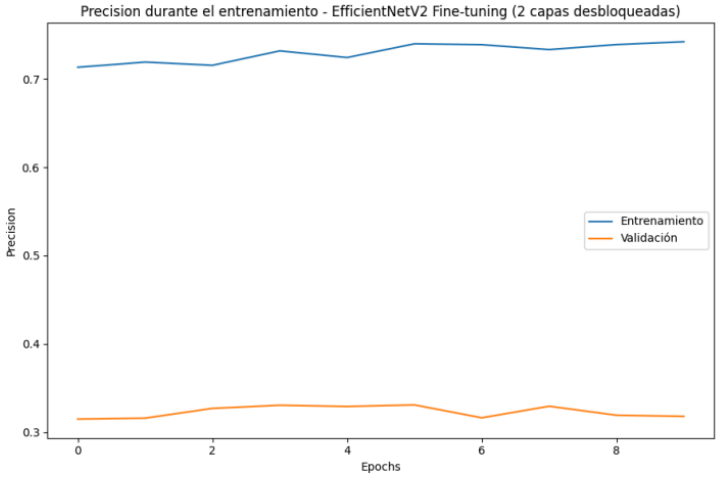
In [ ]:
```
base_model = efficientnet_model.layers[1]
base_model.trainable = True

for layer in base_model.layers[:-2]:
    layer.trainable = False

efficientnet_model.compile(
    optimizer='Adam',
    loss='binary_crossentropy',
    metrics=[
        'accuracy',
        tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall'),
        f1
    ]
)

efficientnet_fine_history_2 = efficientnet_model.fit(
    train_data,
    validation_data=val_data,
    epochs=10
)


for layer in base_model.layers[:-7]:
    layer.trainable = False

efficientnet_model.compile(
    optimizer='Adam',
    loss='binary_crossentropy',
    metrics=[
        'accuracy',
        tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall'),
        f1
    ]
)

efficientnet_fine_history_7 = efficientnet_model.fit(
    train_data,
    validation_data=val_data,
    epochs=10
)
```

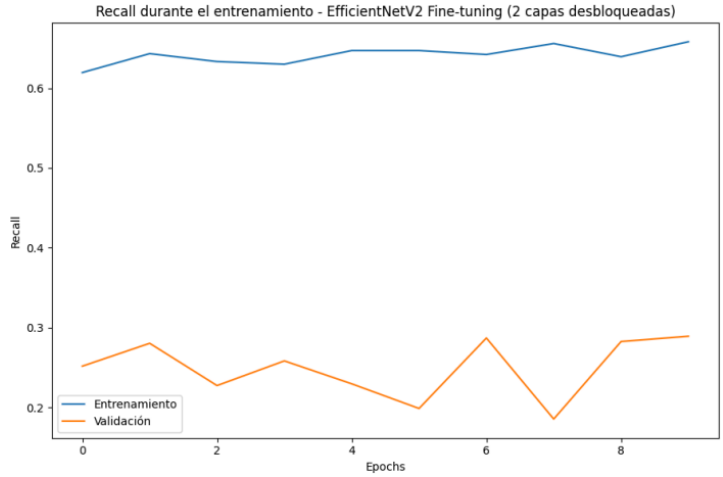
```
Epoch 1/10
  1/378 ━━━━━━━━━━━━━━━━━━━━ 32:40 5s/step - accuracy: 0.7500 - f1_score: 0.0418 - loss: 0.5120 - precision: 0.5000 - recal
l: 0.2500
C:\Users\Jm\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\LocalCache\local-packages\Python312\site-packages
\PIL\Image.py:981: UserWarning: Palette images with Transparency expressed in bytes should be converted to RGBA images
  warnings.warn(
378/378 ━━━━━━━━━━━━━━━━━━━━ 24s 51ms/step - accuracy: 0.8117 - f1_score: 0.2097 - loss: 0.4160 - precision: 0.7119 - recal
l: 0.5865 - val_accuracy: 0.6113 - val_f1_score: 0.2705 - val_loss: 0.8879 - val_precision: 0.3149 - val_recall: 0.2517
Epoch 2/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 18s 48ms/step - accuracy: 0.8096 - f1_score: 0.2978 - loss: 0.4150 - precision: 0.7110 - recal
l: 0.6237 - val_accuracy: 0.6020 - val_f1_score: 0.2885 - val_loss: 0.8610 - val_precision: 0.3159 - val_recall: 0.2804
Epoch 3/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 18s 47ms/step - accuracy: 0.8142 - f1_score: 0.3080 - loss: 0.4043 - precision: 0.7227 - recal
l: 0.6477 - val_accuracy: 0.6278 - val_f1_score: 0.2530 - val_loss: 0.8360 - val_precision: 0.3270 - val_recall: 0.2274
Epoch 4/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 18s 47ms/step - accuracy: 0.8173 - f1_score: 0.2873 - loss: 0.4069 - precision: 0.7362 - recal
l: 0.6229 - val_accuracy: 0.6205 - val_f1_score: 0.2584 - val_loss: 0.9081 - val_precision: 0.3305 - val_recall: 0.2583
Epoch 5/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 18s 47ms/step - accuracy: 0.8158 - f1_score: 0.3008 - loss: 0.3945 - precision: 0.7264 - recal
l: 0.6444 - val_accuracy: 0.6285 - val_f1_score: 0.2522 - val_loss: 0.8303 - val_precision: 0.3291 - val_recall: 0.2296
Epoch 6/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 18s 47ms/step - accuracy: 0.8287 - f1_score: 0.2938 - loss: 0.3868 - precision: 0.7453 - recal
l: 0.6459 - val_accuracy: 0.6391 - val_f1_score: 0.2311 - val_loss: 0.8687 - val_precision: 0.3309 - val_recall: 0.1987
Epoch 7/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 18s 47ms/step - accuracy: 0.8224 - f1_score: 0.2868 - loss: 0.3928 - precision: 0.7301 - recal
l: 0.6239 - val_accuracy: 0.6000 - val_f1_score: 0.2905 - val_loss: 0.9279 - val_precision: 0.3163 - val_recall: 0.2870
Epoch 8/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 18s 47ms/step - accuracy: 0.8276 - f1_score: 0.3036 - loss: 0.3792 - precision: 0.7390 - recal
l: 0.6629 - val_accuracy: 0.6424 - val_f1_score: 0.2133 - val_loss: 0.8684 - val_precision: 0.3294 - val_recall: 0.1854
Epoch 9/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 18s 47ms/step - accuracy: 0.8248 - f1_score: 0.2883 - loss: 0.3787 - precision: 0.7426 - recal
l: 0.6331 - val_accuracy: 0.6040 - val_f1_score: 0.2785 - val_loss: 0.8940 - val_precision: 0.3192 - val_recall: 0.2826
Epoch 10/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 18s 48ms/step - accuracy: 0.8284 - f1_score: 0.2992 - loss: 0.3849 - precision: 0.7290 - recal
l: 0.6557 - val_accuracy: 0.6007 - val_f1_score: 0.2854 - val_loss: 0.8664 - val_precision: 0.3180 - val_recall: 0.2892
Epoch 1/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 24s 51ms/step - accuracy: 0.8208 - f1_score: 0.2935 - loss: 0.3910 - precision: 0.7322 - recal
l: 0.6519 - val_accuracy: 0.5987 - val_f1_score: 0.2921 - val_loss: 0.8476 - val_precision: 0.3174 - val_recall: 0.2936
Epoch 2/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 18s 47ms/step - accuracy: 0.8320 - f1_score: 0.3072 - loss: 0.3804 - precision: 0.7561 - recal
l: 0.6698 - val_accuracy: 0.6040 - val_f1_score: 0.2789 - val_loss: 0.9194 - val_precision: 0.3183 - val_recall: 0.2804
Epoch 3/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 18s 48ms/step - accuracy: 0.8327 - f1_score: 0.3001 - loss: 0.3807 - precision: 0.7556 - recal
l: 0.6536 - val_accuracy: 0.5848 - val_f1_score: 0.3143 - val_loss: 0.9480 - val_precision: 0.3141 - val_recall: 0.3245
Epoch 4/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 18s 47ms/step - accuracy: 0.8114 - f1_score: 0.3073 - loss: 0.4026 - precision: 0.7092 - recal
l: 0.6484 - val_accuracy: 0.6364 - val_f1_score: 0.2525 - val_loss: 0.8980 - val_precision: 0.3471 - val_recall: 0.2406
Epoch 5/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 18s 47ms/step - accuracy: 0.8287 - f1_score: 0.2985 - loss: 0.3826 - precision: 0.7471 - recal
l: 0.6470 - val_accuracy: 0.6179 - val_f1_score: 0.2595 - val_loss: 0.8814 - val_precision: 0.3278 - val_recall: 0.2605
Epoch 6/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 18s 47ms/step - accuracy: 0.8336 - f1_score: 0.2999 - loss: 0.3747 - precision: 0.7587 - recal
l: 0.6618 - val_accuracy: 0.5881 - val_f1_score: 0.3127 - val_loss: 0.9482 - val_precision: 0.3175 - val_recall: 0.3245
Epoch 7/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 18s 47ms/step - accuracy: 0.8289 - f1_score: 0.3036 - loss: 0.3705 - precision: 0.7321 - recal
l: 0.6830 - val_accuracy: 0.6172 - val_f1_score: 0.2698 - val_loss: 0.9447 - val_precision: 0.3333 - val_recall: 0.2759
Epoch 8/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 18s 48ms/step - accuracy: 0.8281 - f1_score: 0.2955 - loss: 0.3811 - precision: 0.7392 - recal
l: 0.6551 - val_accuracy: 0.6252 - val_f1_score: 0.2644 - val_loss: 0.9229 - val_precision: 0.3390 - val_recall: 0.2627
Epoch 9/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 18s 48ms/step - accuracy: 0.8394 - f1_score: 0.3039 - loss: 0.3734 - precision: 0.7628 - recal
l: 0.6749 - val_accuracy: 0.6219 - val_f1_score: 0.2683 - val_loss: 0.8974 - val_precision: 0.3361 - val_recall: 0.2671
Epoch 10/10
378/378 ━━━━━━━━━━━━━━━━━━━━ 18s 47ms/step - accuracy: 0.8364 - f1_score: 0.2996 - loss: 0.3783 - precision: 0.7602 - recal
l: 0.6645 - val_accuracy: 0.6318 - val_f1_score: 0.2611 - val_loss: 0.9183 - val_precision: 0.3444 - val_recall: 0.2517
```

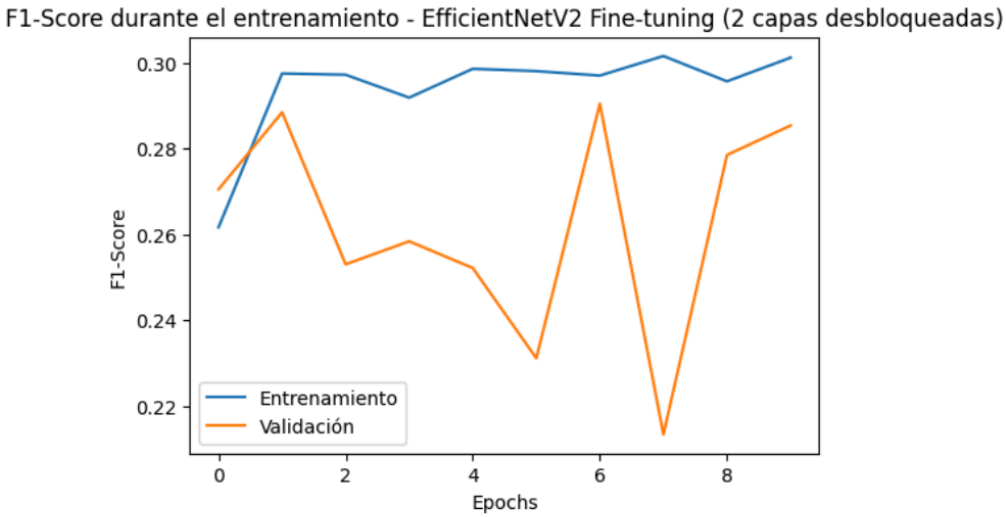
Visualizamos las métricas de entrenamiento para comparar el impacto de desbloquear diferentes cantidades de capas.

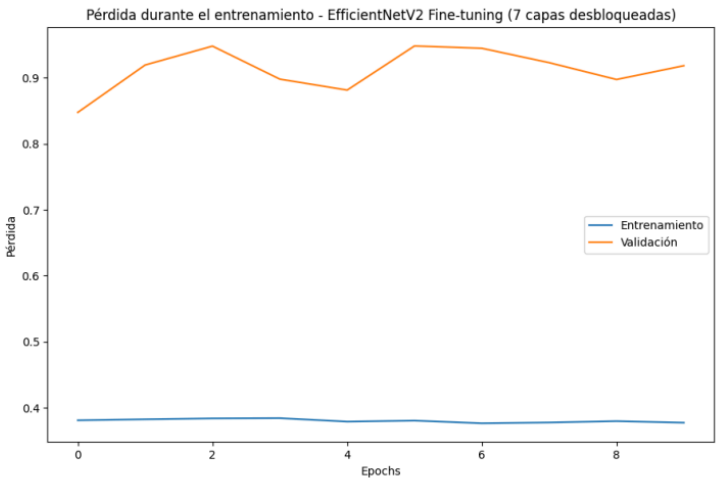
In [21]:
```
plot_training(efficientnet_fine_history_2, 'EfficientNetV2 Fine-tuning (2 capas desbloqueadas)')
plot_training(efficientnet_fine_history_7, 'EfficientNetV2 Fine-tuning (7 capas desbloqueadas)')
```

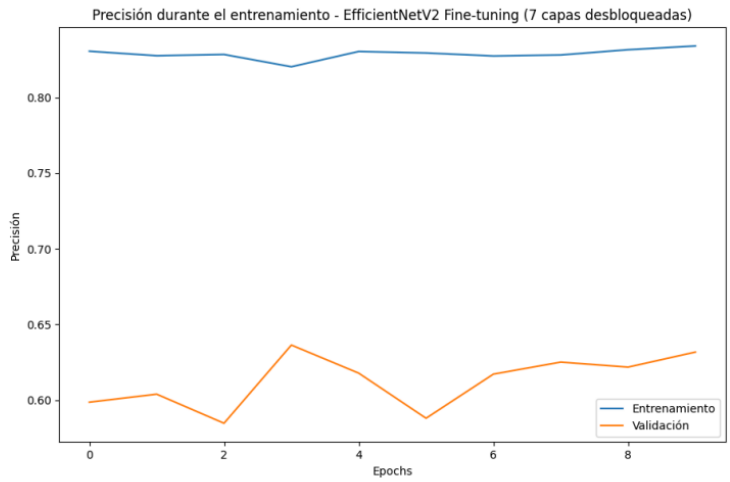


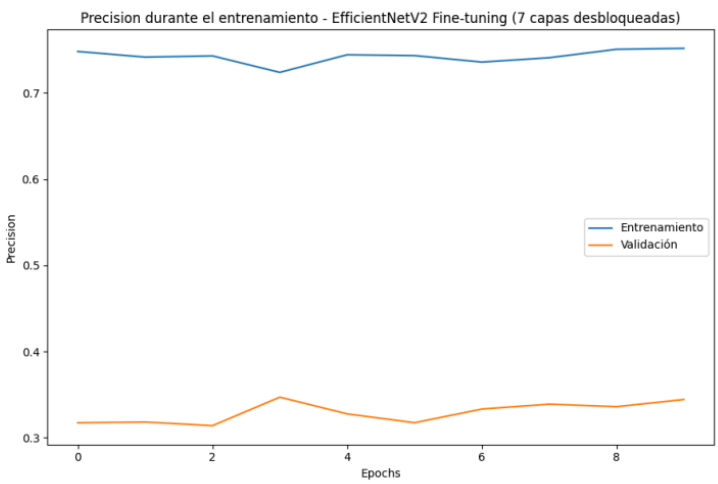

Precision durante el entrenamiento - EfficientNetV2 Fine-tuning (2 capas desbloqueadas)
Precision
Epochs
Entrenamiento
Validación

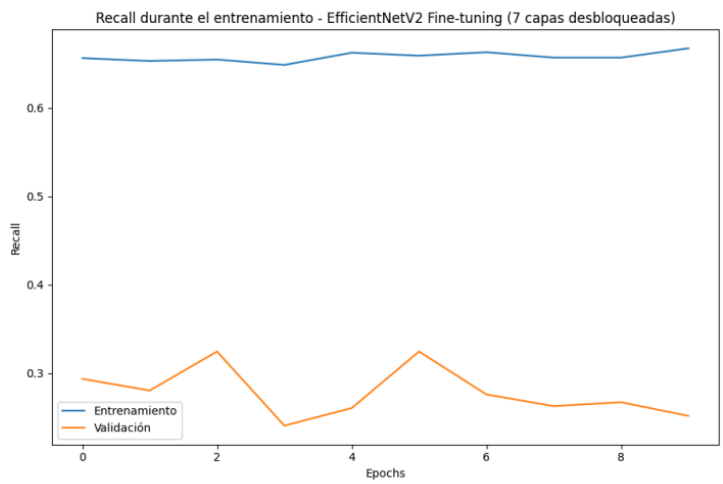

Recall durante el entrenamiento - EfficientNetV2 Fine-tuning (2 capas desbloqueadas)
Recall
Epochs
Entrenamiento
Validación

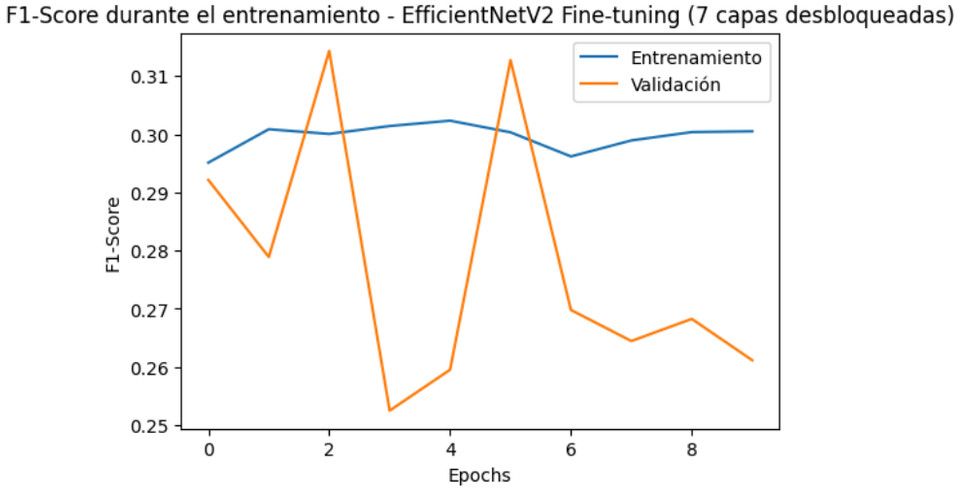

F1-Score durante el entrenamiento - EfficientNetV2 Fine-tuning (2 capas desbloqueadas)
F1-Score
Epochs
Entrenamiento
Validación


Pérdida durante el entrenamiento - EfficientNetV2 Fine-tuning (7 capas desbloqueadas)
Pérdida
Epochs
Entrenamiento
Validación


Precisión durante el entrenamiento - EfficientNetV2 Fine-tuning (7 capas desbloqueadas)
Precisión
Epochs
Entrenamiento
Validación


Precision durante el entrenamiento - EfficientNetV2 Fine-tuning (7 capas desbloqueadas)
Precision
Epochs
Entrenamiento
Validación


Recall durante el entrenamiento - EfficientNetV2 Fine-tuning (7 capas desbloqueadas)
Recall
Epochs
Entrenamiento
Validación




4.1. Análisis de resultados del fine-tuning

**Con 2 capas desbloqueadas:**

Entrenamiento:

- Accuracy: 82.84%
- F1-score: 0.2992
- Precisión: 72.90%
- Recall: 65.57%

Validación:

- Accuracy: 60.07%
- F1-score: 0.2854
- Precisión: 31.80%
- Recall: 28.92%

**Con 7 capas desbloqueadas:**

Entrenamiento:

- Accuracy: 83.64%
- F1-score: 0.2996
- Precisión: 76.02%
- Recall: 66.45%

Validación:

- Accuracy: 63.18%
- F1-score: 0.2611
- Precisión: 34.44%
- Recall: 25.17%

Al desbloquear más capas (7 en lugar de 2), observamos una ligera mejora en la precisión y el recall en el conjunto de entrenamiento, pero el F1-score se mantiene prácticamente igual. Sin embargo, en el conjunto de validación, el F1-score disminuye ligeramente cuando se desbloquean más capas, pasando de 0.2854 a 0.2611. Esto sugiere que desbloquear demasiadas capas puede llevar al modelo a sobreajustarse al conjunto de entrenamiento y no generalizar tan bien al conjunto de validación.

## 5. Evaluación de los modelos sobre conjunto de test y otro dataset

En esta sección, evaluaremos los modelos en un conjunto de test y en otro dataset para detección de mascarillas faciales. También probaremos el modelo en imágenes capturadas desde la webcam.

5.1. Preparación del nuevo conjunto de datos

Descomprimimos el nuevo conjunto de datos de detección de mascarillas faciales.

In [ ]:
```
import os
import zipfile
import shutil
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator


with zipfile.ZipFile('facemaskdataset2022.zip', 'r') as zip_ref:
    zip_ref.extractall('new_mask_dataset')
```

Verificamos las carpetas del nuevo conjunto de datos para asegurarnos de que está correctamente organizado.

In [ ]:
```
NEW_DATA_FOLDER = 'new_mask_dataset/Face_Mask_Dataset'


print(os.listdir(NEW_DATA_FOLDER))
```

```
['Without_Mask', 'With_Mask']
```

Preparamos el generador de datos para el nuevo conjunto de datos, aplicando el mismo preprocesamiento que antes.

In [ ]:
```
new_datagen = ImageDataGenerator(rescale=1./255)
```

```
new_data_generator = new_datagen.flow_from_directory(
    NEW_DATA_FOLDER,
    target_size=IMAGE_SIZE,
    batch_size=16,
    class_mode='binary',
    shuffle=False
)
```

```
Found 20347 images belonging to 2 classes.
```

5.2. Evaluación de los modelos en el nuevo conjunto de datos

5.2.1. Evaluación de MobileNetV2

Resultados de MobileNetV2 en el nuevo conjunto de datos:

In [ ]:
```
from sklearn.metrics import classification_report
import numpy as np

y_true = new_data_generator.classes

y_pred_prob = mobilenet_model.predict(new_data_generator)
y_pred = np.where(y_pred_prob > 0.26333, 1, 0)


class_indices = new_data_generator.class_indices
class_labels = {v: k for k, v in class_indices.items()}

print(classification_report(y_true, y_pred, target_names=class_labels.values()))
```

```
  3/1272 ━━━━━━━━━━━━━━━━━━━━ 1:02 49ms/step
C:\Users\Jm\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\LocalCache\local-packages\Python312\site-packages
\keras\src\trainers\data_adapters\py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)
` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`,
as they will be ignored.
  self._warn_if_super_not_called()
1272/1272 ━━━━━━━━━━━━━━━━━━━━ 90s 71ms/step
              precision    recall  f1-score   support

   With_Mask       0.43      0.12      0.18     10107
Without_Mask       0.49      0.85      0.62     10240

    accuracy                           0.48     20347
   macro avg       0.46      0.48      0.40     20347
weighted avg       0.46      0.48      0.40     20347
```

La precisión y el recall para la clase "With_Mask" son bajos, especialmente el recall (12%), indicando que el modelo está fallando en detectar correctamente las imágenes con mascarilla. Para la clase "Without_Mask", el recall es alto (85%), pero la precisión es moderada (49%). El modelo tiene un sesgo hacia predecir la clase "Without_Mask".

5.2.2. Evaluación de EfficientNetB0

Resultados de EfficientNetB0 en el nuevo conjunto de datos:

In [ ]:
```
y_true = new_data_generator.classes

y_pred_prob = efficientnet_model.predict(new_data_generator)
y_pred = np.where(y_pred_prob > 0.26333, 1, 0)


class_indices = new_data_generator.class_indices
class_labels = {v: k for k, v in class_indices.items()}


print(classification_report(y_true, y_pred, target_names=class_labels.values()))
```

```
1272/1272 ━━━━━━━━━━━━━━━━━━━━ 48s 37ms/step
              precision    recall  f1-score   support

   With_Mask       0.45      0.65      0.53     10107
Without_Mask       0.37      0.21      0.26     10240

    accuracy                           0.43     20347
   macro avg       0.41      0.43      0.40     20347
weighted avg       0.41      0.43      0.40     20347
```

El modelo muestra una precisión y recall más equilibrados para la clase "With_Mask". Sin embargo, el rendimiento para la clase "Without_Mask" es bajo, con un recall de solo 21%. El modelo parece estar sesgado hacia predecir la clase "With_Mask".

5.2.3. Evaluación de ResNet50

Resultados de ResNet50 en el nuevo conjunto de datos:

In [ ]:
```
y_true = new_data_generator.classes

y_pred_prob = resnet_model.predict(new_data_generator)
y_pred = np.where(y_pred_prob > 0.26333, 1, 0)

class_indices = new_data_generator.class_indices
class_labels = {v: k for k, v in class_indices.items()}

print(classification_report(y_true, y_pred, target_names=class_labels.values()))
```

```
1272/1272 ━━━━━━━━━━━━━━━━━━━━ 88s 69ms/step
              precision    recall  f1-score   support

   With_Mask       0.55      0.88      0.68     10107
Without_Mask       0.72      0.29      0.41     10240
```

```
    accuracy                           0.58     20347
   macro avg       0.63      0.59      0.55     20347
weighted avg       0.63      0.58      0.54     20347
```

ResNet50 muestra el mejor rendimiento general en este conjunto de datos externo. La precisión y el recall para "With_Mask" son altos, y aunque el recall para "Without_Mask" es bajo (29%), la precisión es alta (72%). Esto indica que el modelo es más confiable al predecir la clase "With_Mask" y tiene un mejor equilibrio en este conjunto de datos

5.4. Evaluación en imágenes de webcam

La aplicación puede funcionar de manera fluida gracias a la eficiencia computacional de EfficientNetB0. La precisión de la detección en tiempo real puede variar dependiendo de factores como la iluminación, el ángulo de la cámara y las características de la mascarilla. Este prototipo demuestra el potencial para desarrollar aplicaciones prácticas de detección de mascarillas en tiempo real.

In [15]:
```python
import cv2
import numpy as np

cap = cv2.VideoCapture(0)
while True:
    ret, frame = cap.read()
    if not ret:
        break
    img_resized = cv2.resize(frame, (128, 128))
    img_array = np.expand_dims(img_resized, axis=0) / 255.0
    prediction = efficientnet_model.predict(img_array)
    label = 'No Mask' if prediction < 0.2 else 'Mask'
    cv2.putText(frame, label, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
    cv2.imshow('Webcam', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
```

## 6. Bibliografía


- Documentación de Keras: https://keras.io/
- Transfer learning con Keras: https://keras.io/guides/transfer_learning/
- EfficientNet: Rethinking model scaling for convolutional neural networks: https://arxiv.org/abs/1905.11946
- MobileNetV2: Inverted residuals and linear bottlenecks: https://arxiv.org/abs/1801.04381
- Deep Residual Learning for image recognition (ResNet): https://arxiv.org/abs/1512.03385
- Scikit-learn classification report: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html
- Material de clase sobre aprendizaje profundo.