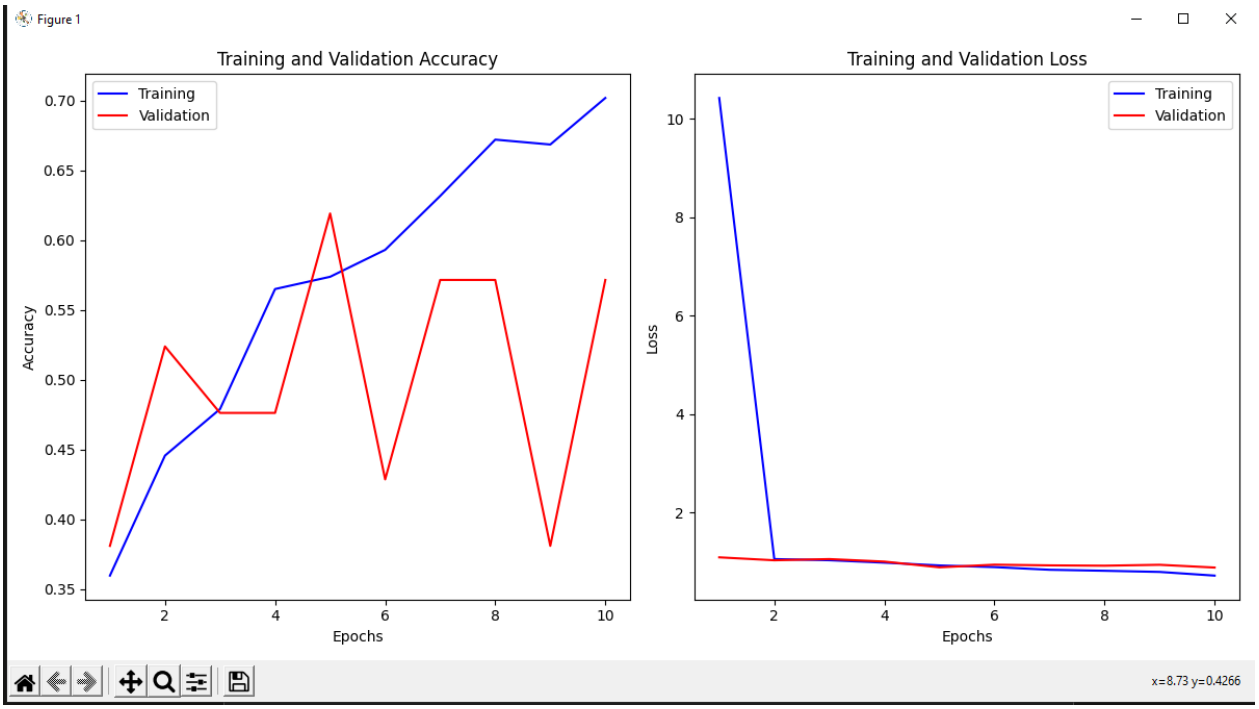
README

Primeros tests

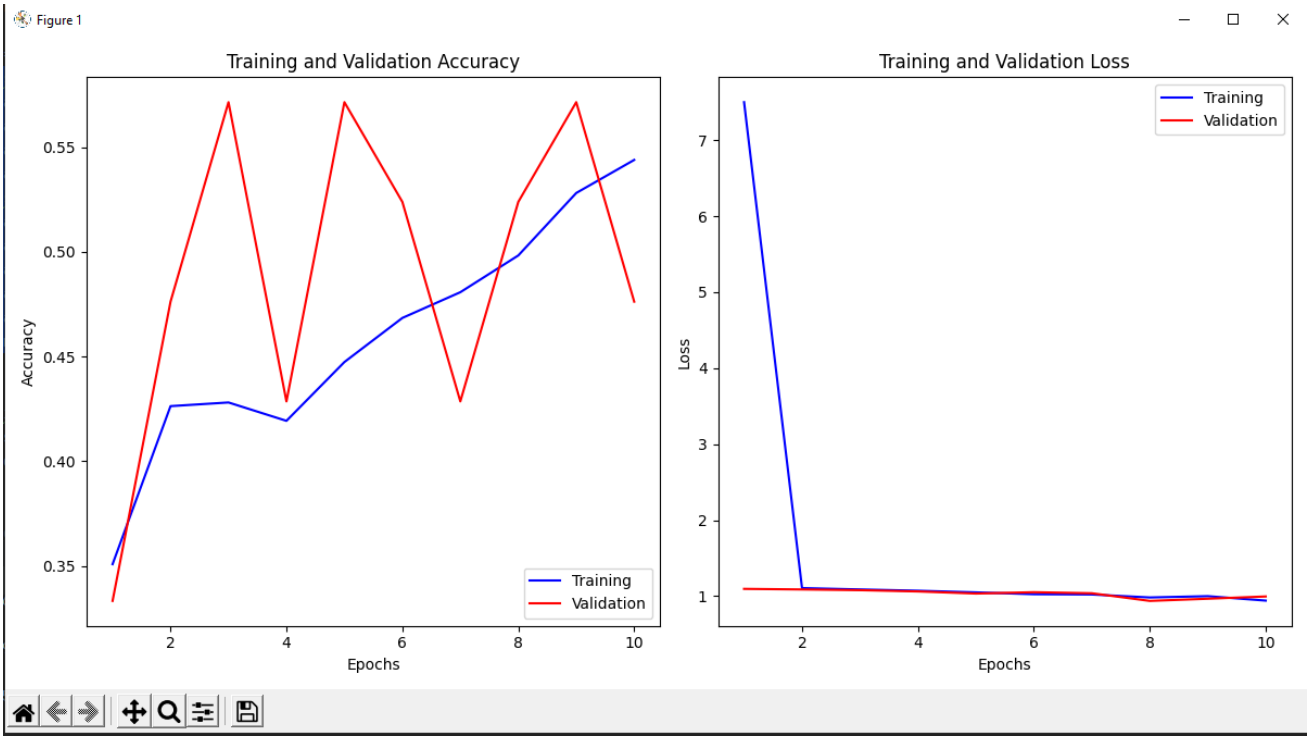
Para estos tests solo utilizamos las imágenes que nos daban en el dataset, eran alrededor de 750. Para este problema eran pocas imágenes y por tanto había mucho sobreajuste.

```
# Capas convolucionales
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(240, 240, 3)))  # 3 canales de color
model.add(Dropout(0.25))  # Dropout después de la capa de conv2D
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())  # Aplanar la salida de la capa convolucional
# Capas fully connected (clasificador)
model.add(Dense(328, activation='relu'))
model.add(Dropout(0.5))  # Dropout antes de la capa de salida
model.add(Dense(3, activation='softmax'))
print(model.summary())
```



Añadimos una capa de clasificacion

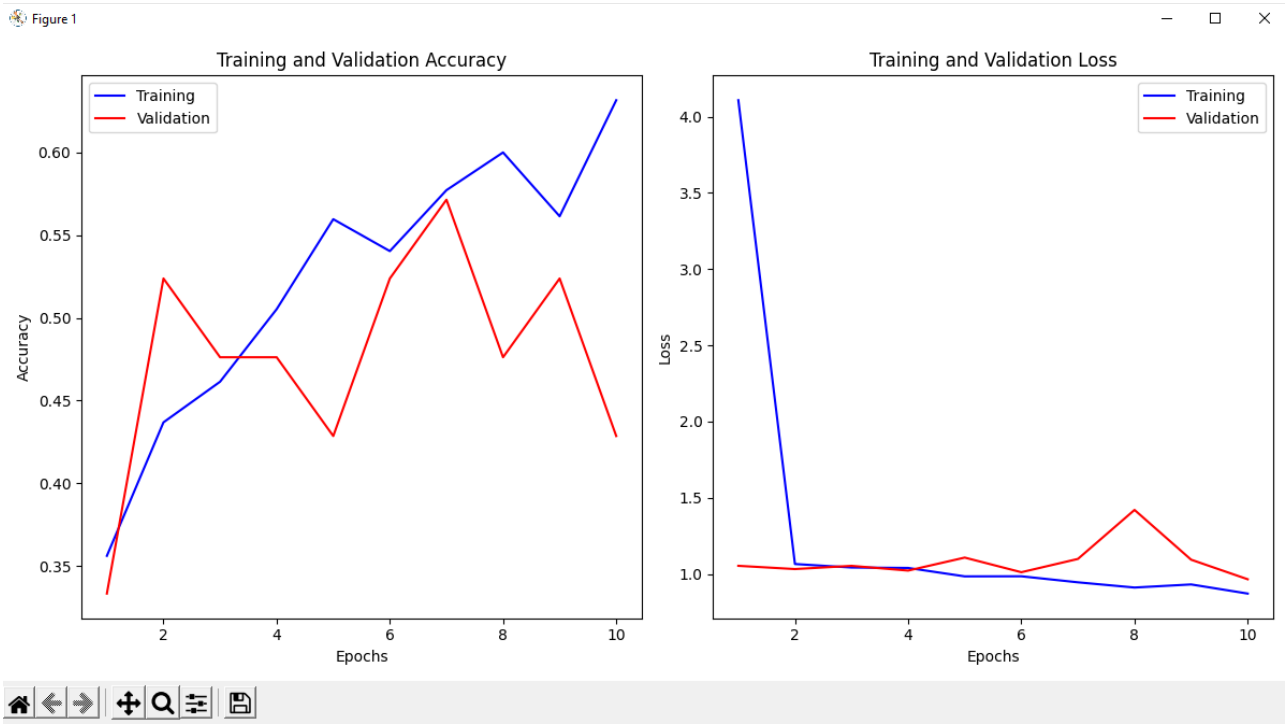
```
# Capas convolucionales
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(240,
240, 3)))  # 3 canales de color
model.add(Dropout(0.25))  # Dropout después de la capa de conv2D
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())  # Aplanar la salida de la capa convolucional
# Capas fully connected (clasificador)
model.add(Dense(328, activation='relu'))
model.add(Dropout(0.5))  # Dropout antes de la capa de salida
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))  # Dropout antes de la capa de salida
model.add(Dense(3, activation='softmax'))
print(model.summary())
```



Probamos cambiando las capas de clasificacion

(ahora hay una de 68 en vez de 2 antes de la final)

```
# Capas convolucionales
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(240, 240, 3)))  # 3 canales de color
model.add(Dropout(0.25))  # Dropout después de la capa de conv2D
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())  # Aplanar la salida de la capa convolucional
# Capas fully connected (clasificador)
model.add(Dense(68, activation='relu'))
model.add(Dropout(0.5))  # Dropout antes de la capa de salida
model.add(Dense(3, activation='softmax'))
print(model.summary())
```

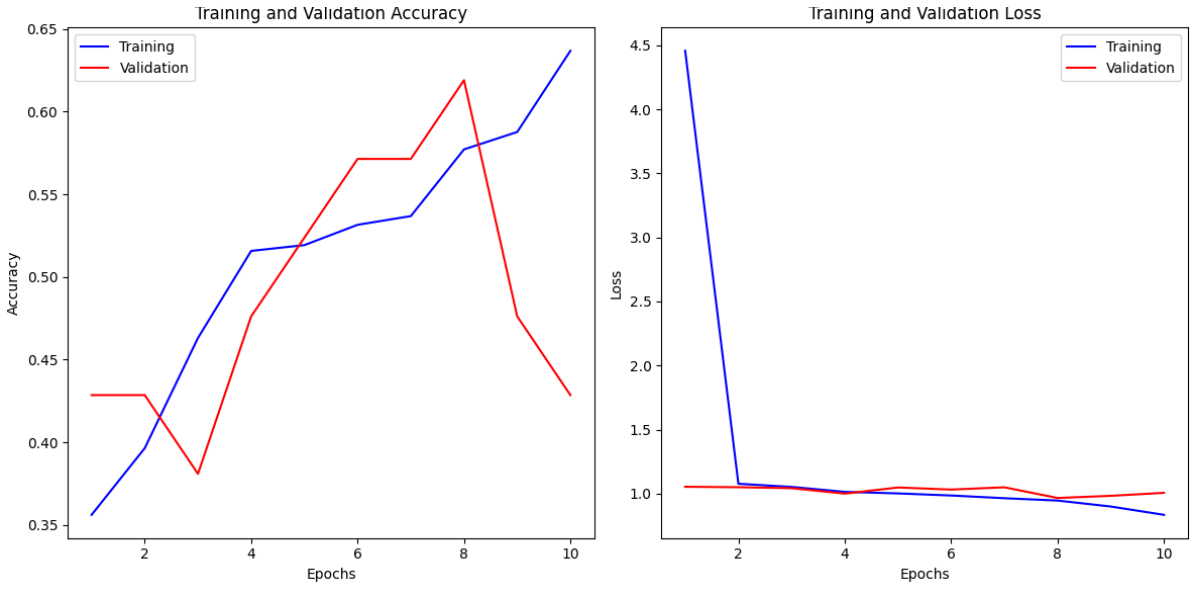


Probamos otra configuracion distinta para las capas del clasificador

(ahora hay una de 100)

```
model = Sequential()
# Capas convolucionales
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(240, 240, 3)))
# 3 canales de color
model.add(Dropout(0.25))  # Dropout después de la capa de conv2D
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())  # Aplanar la salida de la capa convolucional
# Capas fully connected (clasificador)
model.add(Dense(100, activation='relu'))
model.add(Dropout(0.5))  # Dropout antes de la capa de salida
model.add(Dense(3, activation='softmax'))
print(model.summary())
```

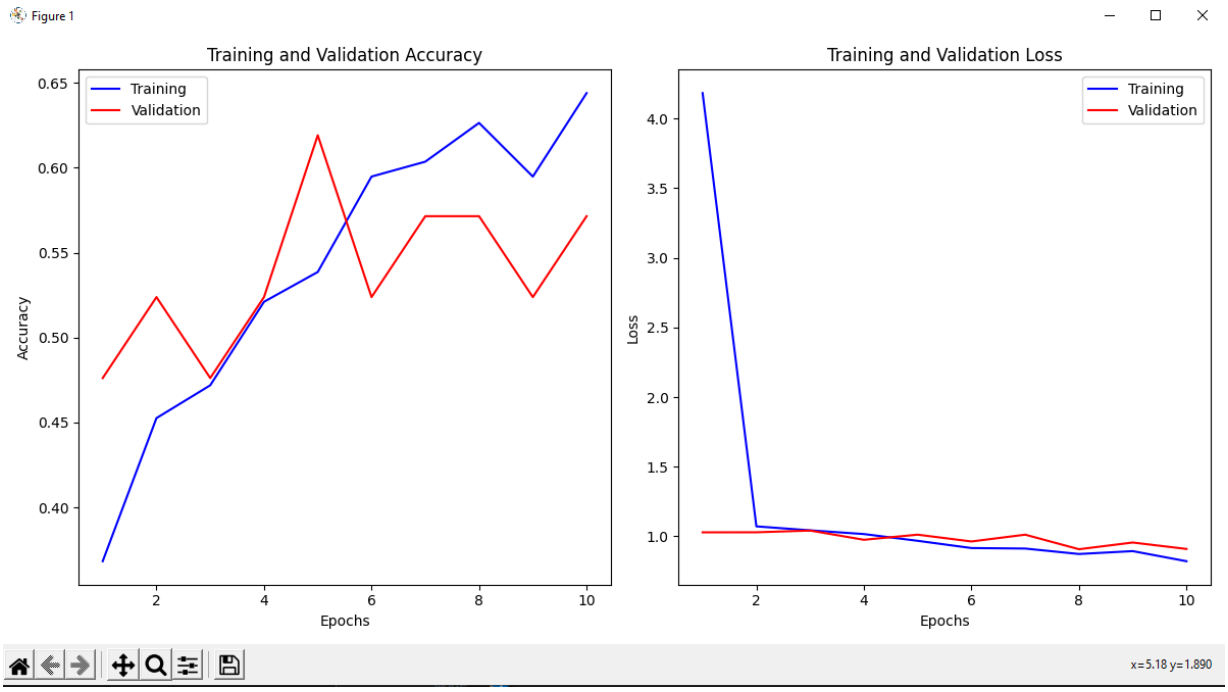
```
# Configurar Early Stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5)  # 'patience' es el número
de épocas sin mejora después de las cuales el entrenamiento se detendrá
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
epochs = 10
```



Probamos otra configuracion distinta para las capas del clasificador

(ahora hay una de 108)

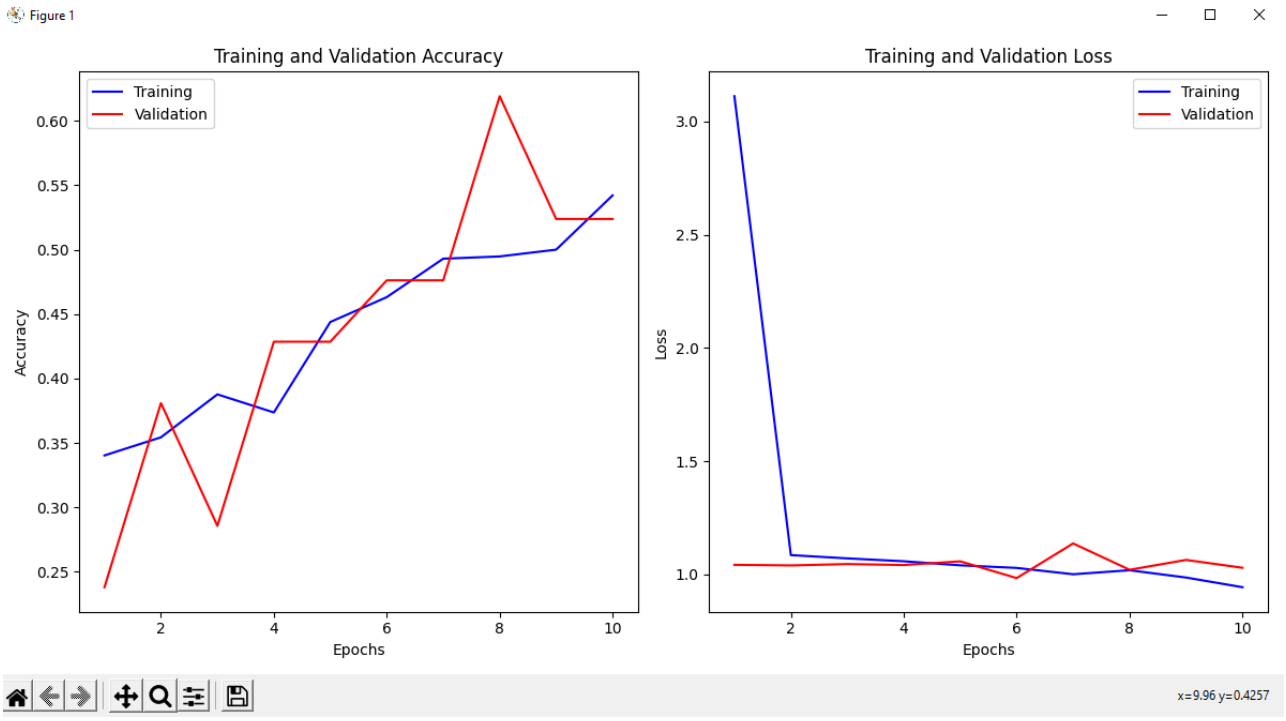
```
# Capas convolucionales
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(240,
240, 3)))  # 3 canales de color
model.add(Dropout(0.25))  # Dropout después de la capa de conv2D
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())  # Aplanar la salida de la capa convolucional
# Capas fully connected (clasificador)
model.add(Dense(108, activation='relu'))
model.add(Dropout(0.5))  # Dropout antes de la capa de salida
model.add(Dense(3, activation='softmax'))
print(model.summary())
```



Probamos otra configuracion distinta para las capas del clasificador

(ahora hay una de 50)

```
# Capas convolucionales
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(240, 240, 3)))  # 3 canales de color
model.add(Dropout(0.25))  # Dropout después de la capa de conv2D
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())  # Aplanar la salida de la capa convolucional
# Capas fully connected (clasificador)
model.add(Dense(50, activation='relu'))
model.add(Dropout(0.5))  # Dropout antes de la capa de salida
model.add(Dense(3, activation='softmax'))
print(model.summary())
```



Segunda parte de los test:

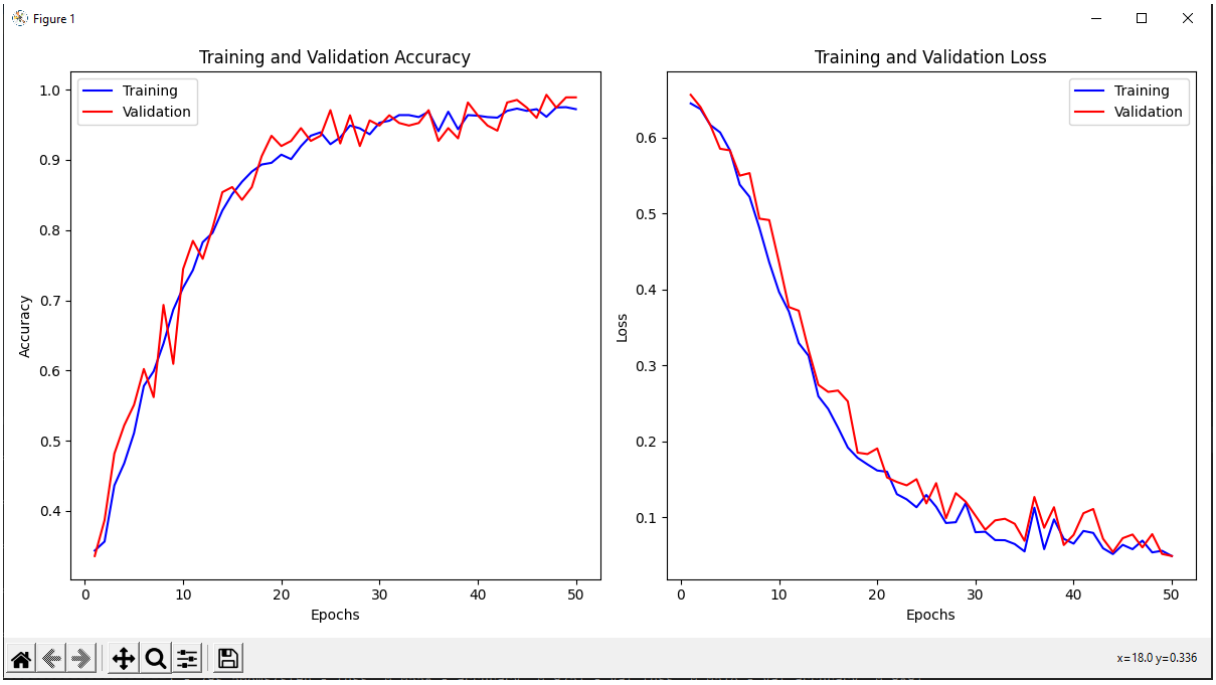
A partir de aquí comenzamos a utilizar muchas más imágenes, alrededor de 2700 imágenes utilizando generación de imágenes. Entonces, se evita la mayor parte del sobreajuste.

```
image_size = 240
batch_size = 36
rescale_factor = 1. / 255

# Crear un generador para aumentar datos
train_datagen = ImageDataGenerator(
    rescale=rescale_factor,  # Normalizar los valores de los píxeles
    shear_range=0.2,  # Rango para las transformaciones aleatorias
    zoom_range=0.2,  # Rango para el zoom aleatorio
    horizontal_flip=True,  # Activar el giro horizontal aleatorio
    validation_split=0.2)  # Establecer el porcentaje de imágenes para el
conjunto de validación

# Cargar imágenes de entrenamiento
train_generator = train_datagen.flow_from_directory(
    "C:/Users/Usuario/Desktop/practicas ap/pythonProject/archive/train",  #
Directorio con datos
    target_size=(image_size, image_size),  # Cambiar el tamaño de las
imágenes a 50x50
    batch_size=batch_size,
    class_mode='categorical',  # 'binary' para clasificación binaria,
'categorical' para multiclase
    subset='training')  # Seleccionar solo el conjunto de entrenamiento

# Cargar imágenes de validación
validation_generator = train_datagen.flow_from_directory(
    "C:/Users/Usuario/Desktop/practicas ap/pythonProject/archive/test",
    target_size=(image_size, image_size),
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation')  # Seleccionar solo el conjunto de validación
```

Utilizamos ahora dos capas convolucionales una de 64 y otra de 3, también hemos cambiado como utilizamos los clasificadores ya que nos estuvimos fijando como venía bien que se estructuraran sus capas.

```
model = Sequential()
# Capas convolucionales
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(240,
240, 3)))  # 3 canales de color
model.add(Dropout(0.25))  # Dropout después de la capa de conv2D
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(Dropout(0.25))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())  # Aplanar la salida de la capa convolucional
# Capas fully connected (clasificador)
model.add(Dense(256, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))  # Dropout antes de la capa de salida
model.add(Dense(3, activation='softmax'))
print(model.summary())
```

No lo apuntamos, pero llega alrededor de 0.94 de precisión y 0.06 de perdida aproximadamente

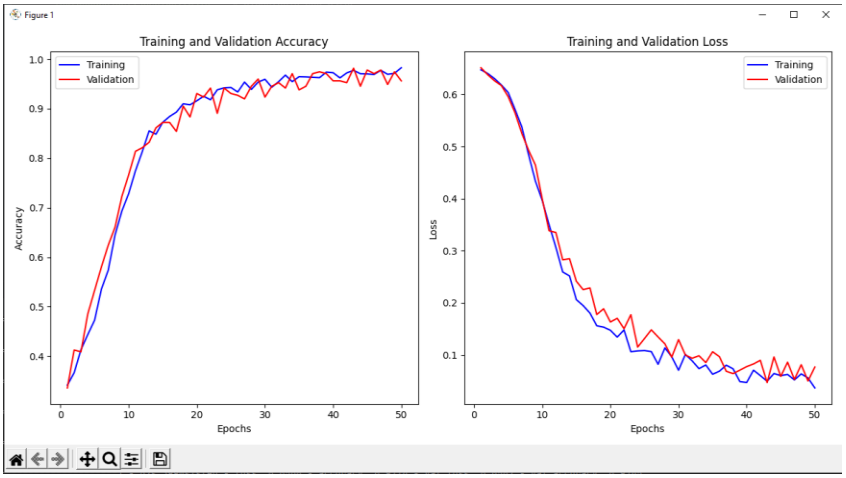


Cambiando las mayor parte de los hyperparametros:

Tambien hemos cambiado la primera capa convolucional a 64, además de que ahora tenemos 3 capas convolucionales (antes eran 2 una de 32 otra de 64), y hemos cambiado el pooling a 4x4.

En la capa de clasificacion ahora la primera solo tiene 128 (antes era de 256) y la segunda 64 (anterior era 128).

En la gráfica: Aumenta la precision, mejora la continuidad de la gráfica de precisión y disminuye la perdida.



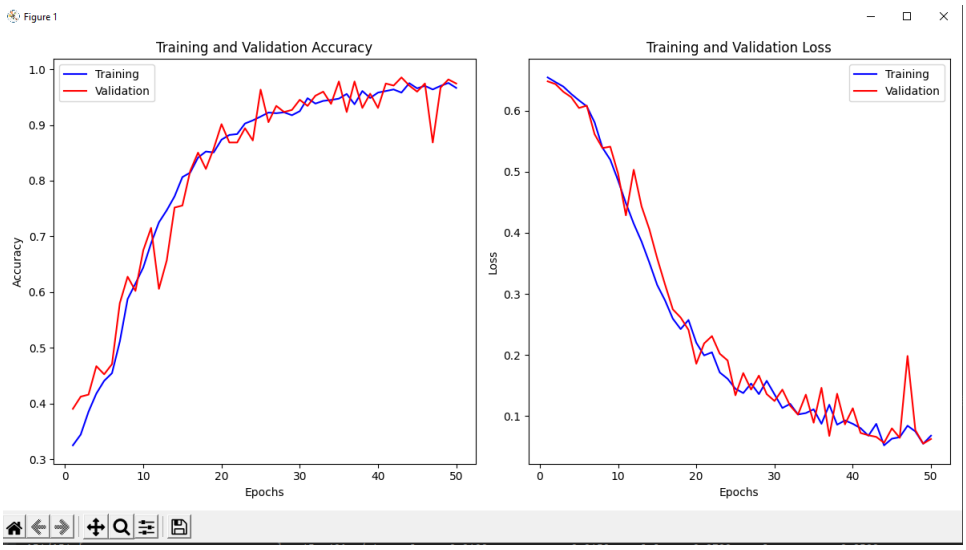
62s 399ms/step - loss: 0.0368 - accuracy: 0.9827 - val_loss: 0.0770 - val_accuracy: 0.9562

```
model = Sequential()
# Capas convolucionales
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu', input_shape=(240,
240, 3)))  # 3 canales de color
model.add(Dropout(0.25))  # Dropout después de la capa de conv2D
model.add(MaxPooling2D(pool_size=(4, 4)))
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(Dropout(0.25))
model.add(MaxPooling2D(pool_size=(4, 4)))
model.add(Conv2D(256, kernel_size=(3, 3), activation='relu'))
model.add(Dropout(0.25))
model.add(MaxPooling2D(pool_size=(4, 4)))


model.add(Flatten())  # Aplanar la salida de la capa convolucional
# Capas fully connected (clasificador)
model.add(Dense(128, activation='relu'))

#model.add(Dense(128, activation='relu'))
```

```
model.add(Dense(64, activation='relu'))

#model.add(Dense(32, activation='relu'))

model.add(Dropout(0.5))

# Dropout antes de la capa de salida
model.add(Dense(3, activation='softmax'))
```

Aumentando otra capa en el clasificador: (a 3)

Tenemos más picos en la gráfica de precisión en la parte de validación (no nos quedamos con este cambio)

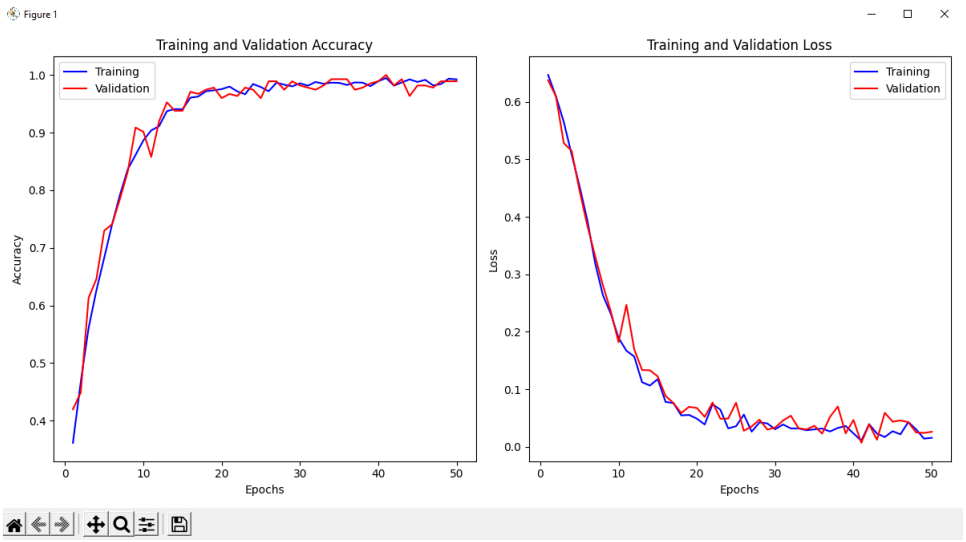


loss: 0.0679 - accuracy: 0.9666 - val_loss: 0.0624 - val_accuracy: 0.9745

```
# Capas convolucionales
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu', input_shape=(224,
224, 3)))  # 3 canales de color
model.add(Dropout(0.25))  # Dropout después de la capa de conv2D
model.add(MaxPooling2D(pool_size=(4, 4)))
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(Dropout(0.25))
model.add(MaxPooling2D(pool_size=(4, 4)))
model.add(Conv2D(256, kernel_size=(3, 3), activation='relu'))
model.add(Dropout(0.25))
model.add(MaxPooling2D(pool_size=(4, 4)))


model.add(Flatten())  # Aplanar la salida de la capa convolucional
# Capas fully connected (clasificador)
model.add(Dense(128, activation='relu'))

#model.add(Dense(128, activation='relu'))

model.add(Dense(64, activation='relu'))

model.add(Dense(32, activation='relu'))

model.add(Dropout(0.5))

# Dropout antes de la capa de salida
model.add(Dense(3, activation='softmax'))
```

Reducimos las capas de convolucion: (a 2)

Aumenta la precisión bastante y reducimos mucho la perdida también las gráficas representan menos picos y están mejor alineadas validación y entrenamiento.



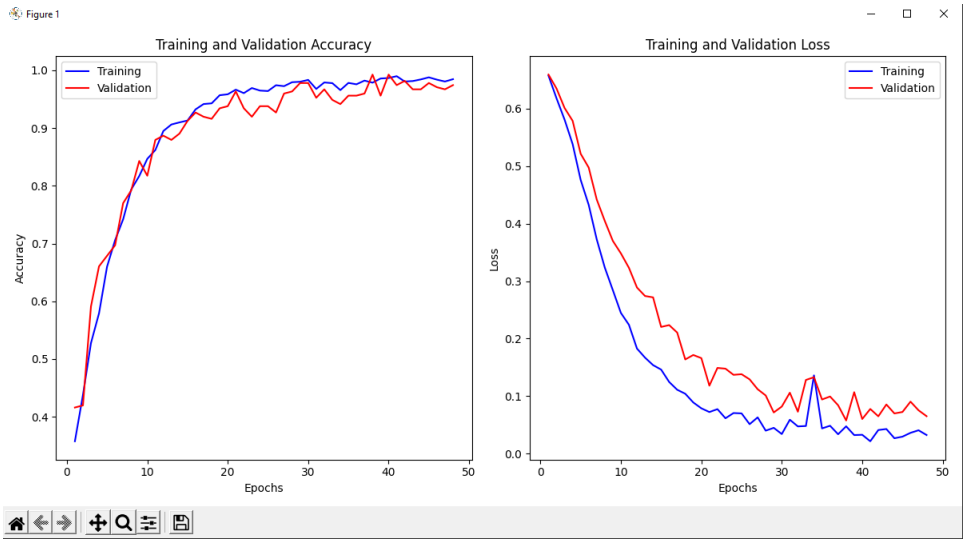
64s 407ms/step - loss: 0.0157 - accuracy: 0.9924 - val_loss: 0.0263 - val_accuracy: 0.9891

```
# Capas convolucionales
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu', input_shape=(224, 224, 3)))  # 3 canales de color
model.add(Dropout(0.25))  # Dropout después de la capa de conv2D
model.add(MaxPooling2D(pool_size=(4, 4)))
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(Dropout(0.25))
model.add(MaxPooling2D(pool_size=(4, 4)))
#model.add(Conv2D(256, kernel_size=(3, 3), activation='relu'))
#model.add(Dropout(0.25))
#model.add(MaxPooling2D(pool_size=(4, 4)))


model.add(Flatten())  # Aplanar la salida de la capa convolucional
# Capas fully connected (clasificador)
model.add(Dense(128, activation='relu'))

#model.add(Dense(128, activation='relu'))

model.add(Dense(64, activation='relu'))

#model.add(Dense(32, activation='relu'))

model.add(Dropout(0.5))
```

```
# Dropout antes de la capa de salida
model.add(Dense(3, activation='softmax'))
```

Aumentar el dropout al 0.5 en las capas de convulcion

La grafica de perdida es mucho peor asi que no nos quedamos con este cambio, haremos pruebas utilizando menor diferencia en el dropout ya que parece afectar mucho.



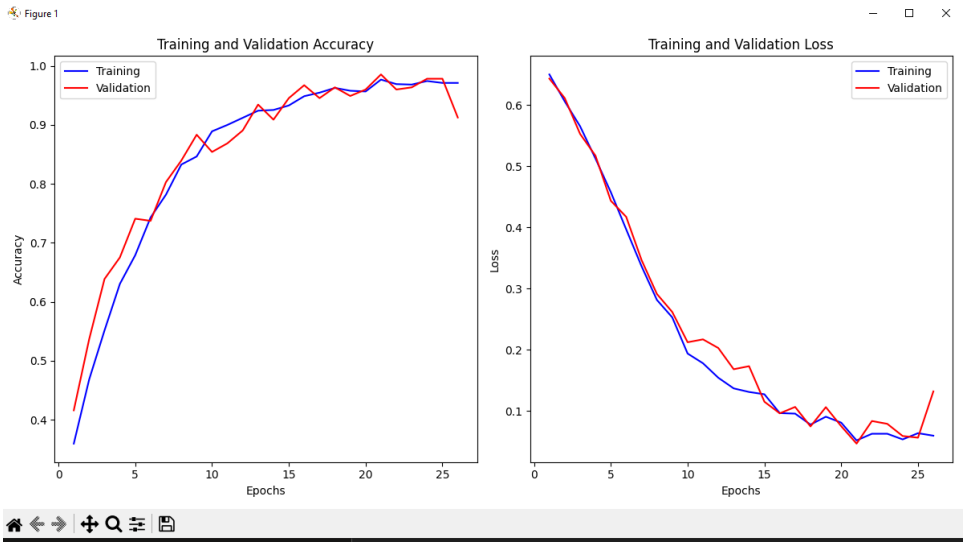
loss: 0.0322 - accuracy: 0.9847 - val_loss: 0.0649 - val_accuracy: 0.9745

```
model = Sequential()
# Capas convolucionales
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu', input_shape=(224, 224, 3)))  # 3 canales de color
model.add(Dropout(0.5))  # Dropout después de la capa de conv2D
model.add(MaxPooling2D(pool_size=(4, 4)))
model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(Dropout(0.5))
model.add(MaxPooling2D(pool_size=(4, 4)))
#model.add(Conv2D(256, kernel_size=(3, 3), activation='relu'))
#model.add(Dropout(0.5))
#model.add(MaxPooling2D(pool_size=(4, 4)))
```

```python
model.add(Flatten())  # Aplanar la salida de la capa convolucional
# Capas fully connected (clasificador)
model.add(Dense(128, activation='relu'))

model.add(Dense(64, activation='relu'))

#model.add(Dense(32, activation='relu'))

model.add(Dropout(0.5))

# Dropout antes de la capa de salida
model.add(Dense(3, activation='softmax'))
```

Aumentar el dropout al 0.3 en las capas de convulcion

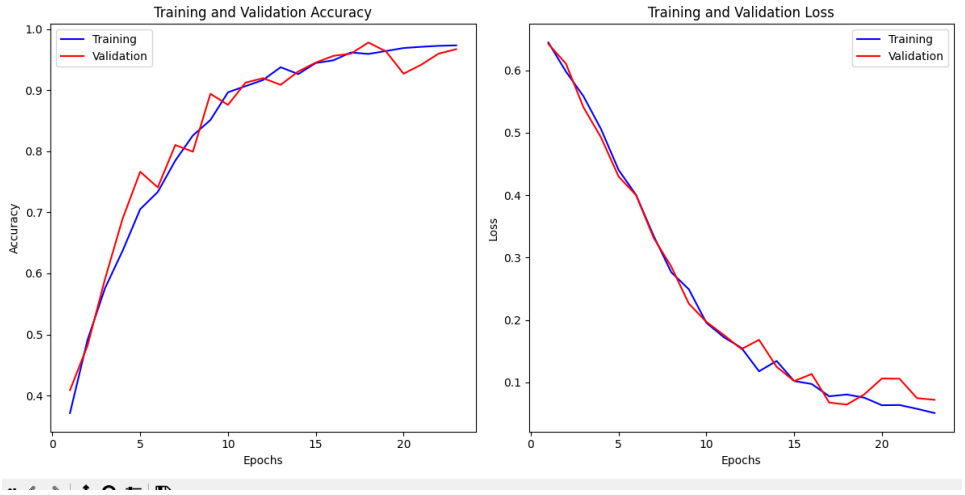
También empeora con respecto al dropout de 0.25




```
Epoch 24/150
156/156 [==============================] - 55s 353ms/step - loss: 0.0534 - accuracy: 0.9743 - val_loss: 0.0589 - val_accuracy: 0.9781
Epoch 25/150
156/156 [==============================] - 57s 364ms/step - loss: 0.0635 - accuracy: 0.9710 - val_loss: 0.0562 - val_accuracy: 0.9781
Epoch 26/150
156/156 [==============================] - 56s 358ms/step - loss: 0.0595 - accuracy: 0.9710 - val_loss: 0.1318 - val_accuracy: 0.9124
```

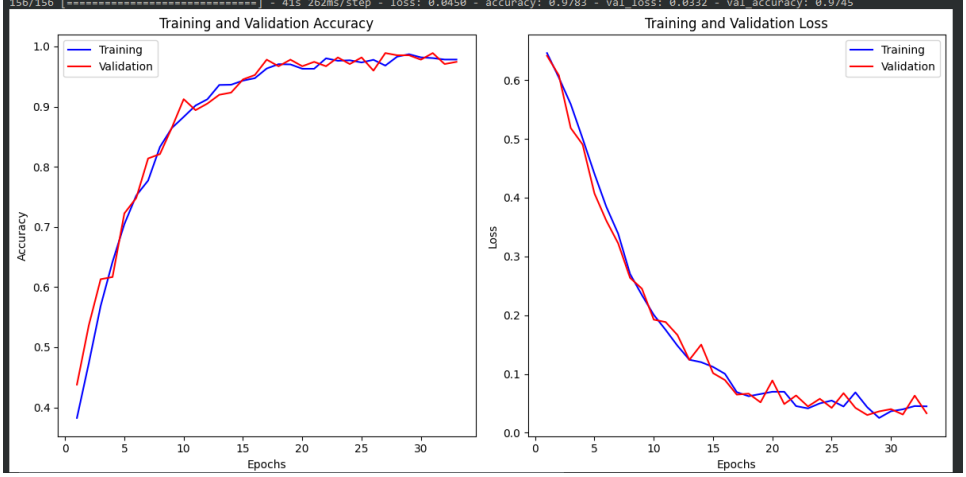
Disminuirlo a 0.2 el dropout.

Perdemos perdida y también precisión con respecto al original 0.25



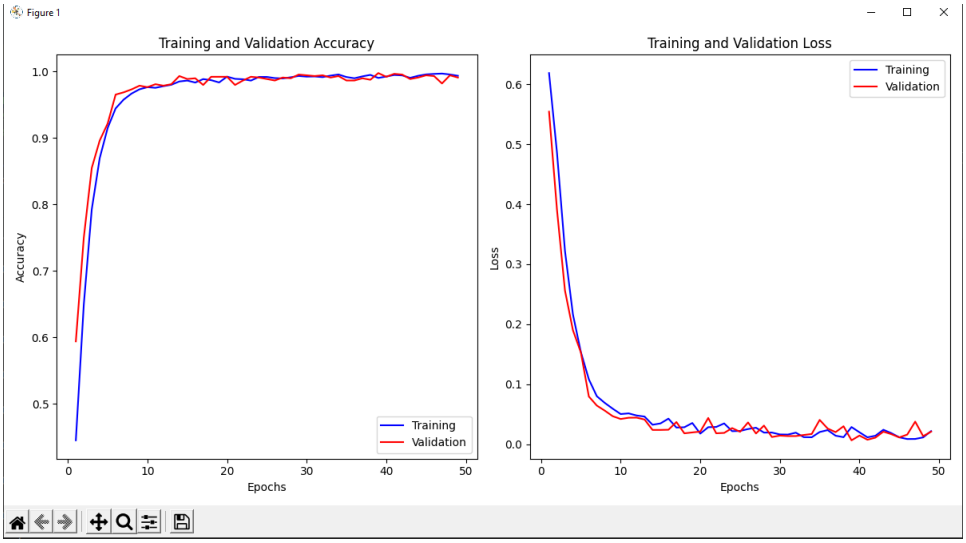

```
156/156 [==============================] - 55s 352ms/step - loss: 0.0637 - accuracy: 0.9710 - val_loss: 0.1058 - val_accuracy: 0.9416
Epoch 22/150
156/156 [==============================] - 55s 352ms/step - loss: 0.0576 - accuracy: 0.9726 - val_loss: 0.0746 - val_accuracy: 0.9599
Epoch 23/150
156/156 [==============================] - 55s 351ms/step - loss: 0.0508 - accuracy: 0.9735 - val_loss: 0.0720 - val_accuracy: 0.9672
```

```
Epoch 31/150
156/156 [==============================] - 43s 278ms/step - loss: 0.0399 - accuracy: 0.9807 - val_loss: 0.0311 - val_accuracy: 0.9891
Epoch 32/150
156/156 [==============================] - 44s 280ms/step - loss: 0.0455 - accuracy: 0.9783 - val_loss: 0.0632 - val_accuracy: 0.9708
Epoch 33/150
156/156 [==============================] - 41s 262ms/step - loss: 0.0450 - accuracy: 0.9783 - val_loss: 0.0332 - val_accuracy: 0.9745
```



Tercera prueba

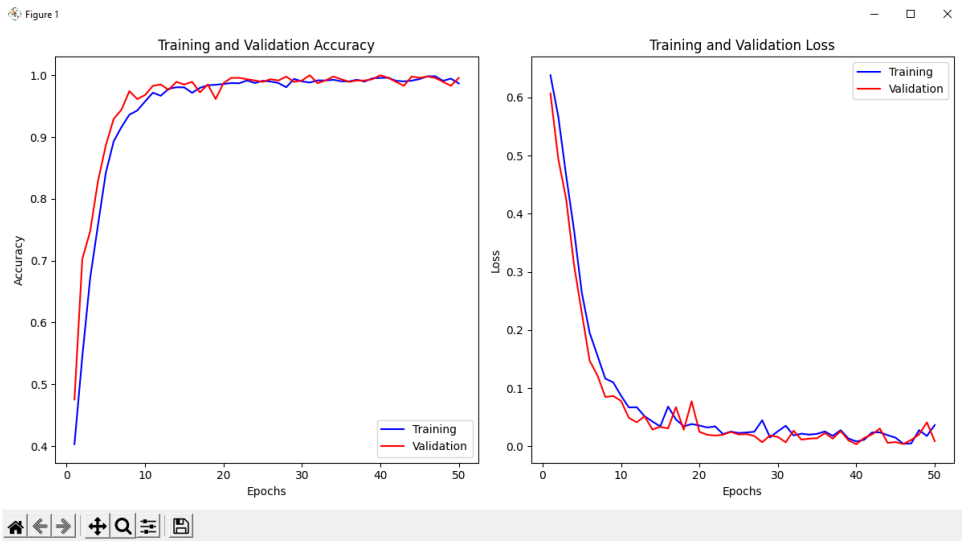
Aumentamos el número de imágenes al triple que teníamos antes (8800).




```
506/506 [==============================] - 256s 506ms/step - loss: 0.0181 -
accuracy: 0.9946 - val_loss: 0.0144 - val_accuracy: 0.9911

Epoch 47/50

506/506 [==============================] - 268s 530ms/step - loss: 0.0097 -
accuracy: 0.9970 - val_loss: 0.0065 - val_accuracy: 0.9978

Epoch 48/50

506/506 [==============================] - 281s 556ms/step - loss: 0.0097 -
accuracy: 0.9965 - val_loss: 0.0046 - val_accuracy: 0.9978

Epoch 49/50

506/506 [==============================] - 275s 543ms/step - loss: 0.0092 -
accuracy: 0.9964 - val_loss: 0.0043 - val_accuracy: 0.9978

Epoch 50/50
```

```
506/506 [==============================] - 269s 532ms/step - loss: 0.0115 -
accuracy: 0.9964 - val_loss: 0.0068 - val_accuracy: 0.9967
```

Cuarta prueba



Epoch 43/50

264/264 [==============================] - 134s 509ms/step - loss: 0.0241 - accuracy: 0.9900 - val_loss: 0.0305 - val_accuracy: 0.9829

Epoch 44/50

264/264 [==============================] - 137s 517ms/step - loss: 0.0191 - accuracy: 0.9912 - val_loss: 0.0058 - val_accuracy: 0.9979

Epoch 45/50

264/264 [==============================] - 130s 492ms/step - loss: 0.0148 - accuracy: 0.9938 - val_loss: 0.0072 - val_accuracy: 0.9957

Epoch 46/50

264/264 [==============================] - 130s 493ms/step - loss: 0.0043 - accuracy: 0.9983 - val_loss: 0.0040 - val_accuracy: 0.9979

Epoch 47/50

264/264 [==============================] - 131s 494ms/step - loss: 0.0049 - accuracy: 0.9983 - val_loss: 0.0109 - val_accuracy: 0.9957

Epoch 48/50

264/264 [==============================] - 132s 501ms/step - loss: 0.0278 - accuracy: 0.9910 - val_loss: 0.0204 - val_accuracy: 0.9893

Epoch 49/50

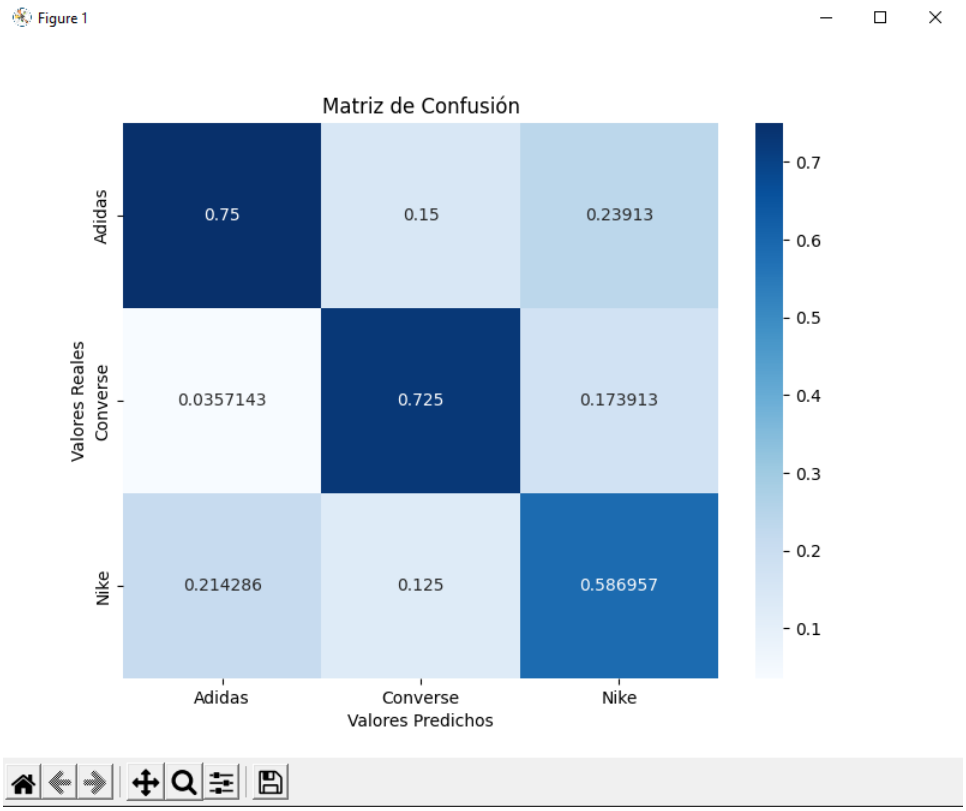
264/264 [==============================] - 137s 519ms/step - loss: 0.0178 - accuracy: 0.9945 - val_loss: 0.0412 - val_accuracy: 0.9829

Epoch 50/50

264/264 [==============================] - 131s 497ms/step - loss: 0.0365 - accuracy: 0.9867 - val_loss: 0.0087 - val_accuracy: 0.9957

```
Precisión: 1.0, Sensibilidad: 1.0
Test Loss: 0.004224284552037716
Test Accuracy: 1.0
```

Matriz de confusión



Explicación de la matriz de confusión:

Podemos ver que nuestro modelo parece tener un ajuste correcto aunque, tiene problemas para diferenciar las adidas y las nike entre ellas puesto a que suelen tener diseños parecidos, o que hay más modelos de estas dos marcas en donde el modelo es blanco por lo que la marca para distinguir las muestras se distingue poco.

Explicación

Hemos podido observar que al añadirle más imágenes la precisión aumenta y la configuración más favorable que teníamos también mejora considerablemente acorde a ese aumento de imágenes, además, al probar el modelo da una proporción de perdida considerable y es la mejor proporción que hemos llegado comparando con los otros modelos más favorables que teníamos en la segunda parte.