Results:  
[Skip to Main](http://localhost:8888/notebooks/Abi%20Karimireddy/Downloads/deep%20learning/Training/trainingsec.ipynb#first-cell)

**trainingsec**

Last Checkpoint: 5 hours ago

import tensorflow as tf

from tensorflow.keras import models ,layers

import matplotlib.pyplot as plt

len(temp\_test\_ds)

val\_size = 0.1

len(dataset) \* val\_size

val\_ds = temp\_test\_ds.take(64)

len(val\_ds)

test\_ds = temp\_test\_ds.skip(64)

len(test\_ds)

def get\_dataset\_partitions\_tf(ds, train\_split=0.8, val\_split=0.1, test\_split=0.1, shuffle=True, shufflesize=10000):

ds\_size = len(ds)

if shuffle:

ds = ds.shuffle(shufflesize, seed=12) # Fixed the variable name here

train\_size = int(train\_split \* ds\_size)

val\_size = int(val\_split \* ds\_size)

train\_ds = ds.take(train\_size)

val\_ds = ds.skip(train\_size).take(val\_size)

test\_ds = ds.skip(train\_size + val\_size) # Adjusted to skip both train and validation data

return train\_ds, val\_ds, test\_ds

train\_ds, val\_ds, test\_ds = get\_dataset\_partitions\_tf(dataset)

train\_ds = train\_ds.cache().shuffle(1000).prefetch(buffer\_size=tf.data.AUTOTUNE)

val\_ds = val\_ds.cache().shuffle(1000).prefetch(buffer\_size=tf.data.AUTOTUNE)

test\_ds = test\_ds.cache().shuffle(1000).prefetch(buffer\_size=tf.data.AUTOTUNE)

resize\_and\_rescale = tf.keras.Sequential([

layers.Resizing(IMAGE\_SIZE, IMAGE\_SIZE),

layers.Rescaling(1.0 / 255)

])

data\_augmentation = tf.keras.Sequential([

layers.RandomFlip("horizontal\_and\_vertical"),

layers.RandomRotation(0.2),

])

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense

n\_classes = 15

# Define the image size and number of channels

IMAGE\_SIZE = 256

CHANNELS = 3

# Define the rescaling factor

rescale\_factor = 1.0 / 255.0

# Define the model

model = Sequential([

Input(shape=(IMAGE\_SIZE, IMAGE\_SIZE, CHANNELS)), # Input layer with specified shape

Conv2D(32, (3, 3), activation='relu'),

MaxPooling2D((2, 2)),

Conv2D(64, kernel\_size=(3, 3), activation='relu'),

MaxPooling2D((2, 2)),

Conv2D(64, kernel\_size=(3, 3), activation='relu'),

MaxPooling2D((2, 2)),

Conv2D(64, (3, 3), activation='relu'),

MaxPooling2D((2, 2)),

Conv2D(64, (3, 3), activation='relu'),

MaxPooling2D((2, 2)),

Conv2D(64, (3, 3), activation='relu'),

MaxPooling2D((2, 2)),

Flatten(),

Dense(64, activation='relu'),

Dense(n\_classes, activation='softmax')

])

# Compile the model

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

# Print model summary

model.summary()

# Train the model

history = model.fit(

train\_ds.cache().shuffle(1000).prefetch(buffer\_size=tf.data.AUTOTUNE),

epochs=EPOCHS,

batch\_size=BATCH\_SIZE,

verbose=1,

validation\_data=val\_ds.cache().shuffle(1000).prefetch(buffer\_size=tf.data.AUTOTUNE)

)

# Evaluate the model

score = model.evaluate(test\_ds)

Found 20638 files belonging to 15 classes.

**Model: "sequential\_2"**

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓

┃ **Layer (type)** ┃ **Output Shape** ┃ **Param #** ┃

┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩

│ conv2d (Conv2D) │ (None, 254, 254, 32) │ 896 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ max\_pooling2d (MaxPooling2D) │ (None, 127, 127, 32) │ 0 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ conv2d\_1 (Conv2D) │ (None, 125, 125, 64) │ 18,496 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ max\_pooling2d\_1 (MaxPooling2D) │ (None, 62, 62, 64) │ 0 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ conv2d\_2 (Conv2D) │ (None, 60, 60, 64) │ 36,928 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ max\_pooling2d\_2 (MaxPooling2D) │ (None, 30, 30, 64) │ 0 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ conv2d\_3 (Conv2D) │ (None, 28, 28, 64) │ 36,928 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ max\_pooling2d\_3 (MaxPooling2D) │ (None, 14, 14, 64) │ 0 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ conv2d\_4 (Conv2D) │ (None, 12, 12, 64) │ 36,928 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ max\_pooling2d\_4 (MaxPooling2D) │ (None, 6, 6, 64) │ 0 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ conv2d\_5 (Conv2D) │ (None, 4, 4, 64) │ 36,928 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ max\_pooling2d\_5 (MaxPooling2D) │ (None, 2, 2, 64) │ 0 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ flatten (Flatten) │ (None, 256) │ 0 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ dense (Dense) │ (None, 64) │ 16,448 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ dense\_1 (Dense) │ (None, 15) │ 975 │

└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘

**Total params:** 184,527 (720.81 KB)

**Trainable params:** 184,527 (720.81 KB)

**Non-trainable params:** 0 (0.00 B)

Epoch 1/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 1757s 3s/step - accuracy: 0.3552 - loss: 2.4108 - val\_accuracy: 0.7056 - val\_loss: 0.9259

Epoch 2/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 1581s 3s/step - accuracy: 0.7369 - loss: 0.7935 - val\_accuracy: 0.7671 - val\_loss: 0.7147

Epoch 3/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 1713s 3s/step - accuracy: 0.8035 - loss: 0.5839 - val\_accuracy: 0.8604 - val\_loss: 0.4145

Epoch 4/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 1337s 3s/step - accuracy: 0.8563 - loss: 0.4245 - val\_accuracy: 0.8843 - val\_loss: 0.3416

Epoch 5/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 1382s 3s/step - accuracy: 0.8827 - loss: 0.3456 - val\_accuracy: 0.8809 - val\_loss: 0.3505

Epoch 6/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 1198s 2s/step - accuracy: 0.9033 - loss: 0.2829 - val\_accuracy: 0.8906 - val\_loss: 0.3299

Epoch 7/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 1090s 2s/step - accuracy: 0.9144 - loss: 0.2427 - val\_accuracy: 0.9165 - val\_loss: 0.2637

Epoch 8/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 1055s 2s/step - accuracy: 0.9212 - loss: 0.2308 - val\_accuracy: 0.9292 - val\_loss: 0.2194

Epoch 9/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 1281s 2s/step - accuracy: 0.9351 - loss: 0.1859 - val\_accuracy: 0.9028 - val\_loss: 0.3089

Epoch 10/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 1611s 3s/step - accuracy: 0.9394 - loss: 0.1858 - val\_accuracy: 0.9424 - val\_loss: 0.1879

Epoch 11/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 1610s 3s/step - accuracy: 0.9456 - loss: 0.1570 - val\_accuracy: 0.8315 - val\_loss: 0.5245

Epoch 12/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 1551s 3s/step - accuracy: 0.9418 - loss: 0.1718 - val\_accuracy: 0.9565 - val\_loss: 0.1409

Epoch 13/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 1569s 3s/step - accuracy: 0.9581 - loss: 0.1296 - val\_accuracy: 0.9165 - val\_loss: 0.2513

Epoch 14/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 1010s 2s/step - accuracy: 0.9418 - loss: 0.1703 - val\_accuracy: 0.9136 - val\_loss: 0.2892

Epoch 15/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 528s 1s/step - accuracy: 0.9572 - loss: 0.1295 - val\_accuracy: 0.9453 - val\_loss: 0.1764

Epoch 16/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 512s 992ms/step - accuracy: 0.9630 - loss: 0.1188 - val\_accuracy: 0.9478 - val\_loss: 0.1719

Epoch 17/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 543s 1s/step - accuracy: 0.9524 - loss: 0.1490 - val\_accuracy: 0.8867 - val\_loss: 0.3785

Epoch 18/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 609s 1s/step - accuracy: 0.9510 - loss: 0.1383 - val\_accuracy: 0.9443 - val\_loss: 0.2036

Epoch 19/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 596s 1s/step - accuracy: 0.9648 - loss: 0.1109 - val\_accuracy: 0.9604 - val\_loss: 0.1467

Epoch 20/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 600s 1s/step - accuracy: 0.9642 - loss: 0.1154 - val\_accuracy: 0.9438 - val\_loss: 0.2039

Epoch 21/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 605s 1s/step - accuracy: 0.9541 - loss: 0.1360 - val\_accuracy: 0.9512 - val\_loss: 0.1564

Epoch 22/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 601s 1s/step - accuracy: 0.9739 - loss: 0.0802 - val\_accuracy: 0.9209 - val\_loss: 0.2565

Epoch 23/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 589s 1s/step - accuracy: 0.9627 - loss: 0.1092 - val\_accuracy: 0.9590 - val\_loss: 0.1505

Epoch 24/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 569s 1s/step - accuracy: 0.9725 - loss: 0.0833 - val\_accuracy: 0.9658 - val\_loss: 0.1195

Epoch 25/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 615s 1s/step - accuracy: 0.9719 - loss: 0.0854 - val\_accuracy: 0.9478 - val\_loss: 0.2058

Epoch 26/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 600s 1s/step - accuracy: 0.9603 - loss: 0.1260 - val\_accuracy: 0.9580 - val\_loss: 0.1585

Epoch 27/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 601s 1s/step - accuracy: 0.9707 - loss: 0.0961 - val\_accuracy: 0.9507 - val\_loss: 0.1863

Epoch 28/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 598s 1s/step - accuracy: 0.9626 - loss: 0.1229 - val\_accuracy: 0.9160 - val\_loss: 0.3265

Epoch 29/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 1557s 3s/step - accuracy: 0.9750 - loss: 0.0731 - val\_accuracy: 0.9390 - val\_loss: 0.2419

Epoch 30/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 1428s 3s/step - accuracy: 0.9688 - loss: 0.1013 - val\_accuracy: 0.9453 - val\_loss: 0.1791

Epoch 31/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 502s 973ms/step - accuracy: 0.9709 - loss: 0.0973 - val\_accuracy: 0.9419 - val\_loss: 0.2245

Epoch 32/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 833s 2s/step - accuracy: 0.9707 - loss: 0.0923 - val\_accuracy: 0.9727 - val\_loss: 0.1257

Epoch 33/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 513s 993ms/step - accuracy: 0.9744 - loss: 0.0872 - val\_accuracy: 0.9741 - val\_loss: 0.1273

Epoch 34/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 514s 995ms/step - accuracy: 0.9774 - loss: 0.0732 - val\_accuracy: 0.9590 - val\_loss: 0.1457

Epoch 35/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 498s 965ms/step - accuracy: 0.9748 - loss: 0.0796 - val\_accuracy: 0.9644 - val\_loss: 0.1289

Epoch 36/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 511s 990ms/step - accuracy: 0.9694 - loss: 0.0938 - val\_accuracy: 0.9639 - val\_loss: 0.1373

Epoch 37/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 501s 970ms/step - accuracy: 0.9745 - loss: 0.0726 - val\_accuracy: 0.9424 - val\_loss: 0.2032

Epoch 38/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 514s 995ms/step - accuracy: 0.9639 - loss: 0.1165 - val\_accuracy: 0.9580 - val\_loss: 0.1557

Epoch 39/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 507s 982ms/step - accuracy: 0.9740 - loss: 0.0857 - val\_accuracy: 0.9717 - val\_loss: 0.1023

Epoch 40/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 505s 980ms/step - accuracy: 0.9778 - loss: 0.0664 - val\_accuracy: 0.9551 - val\_loss: 0.1797

65/65 ━━━━━━━━━━━━━━━━━━━━ 210s 920ms/step - accuracy: 0.9731 - loss: 0.0893

history

<keras.src.callbacks.history.History at 0x24f9f1b3d90>

history.params

{'verbose': 1, 'epochs': 40, 'steps': 516}

history.history.keys()

dict\_keys(['accuracy', 'loss', 'val\_accuracy', 'val\_loss'])

acc = history.history['accuracy']

val\_acc = history.history['accuracy']

loss = history.history['loss']

val\_loss = history.history['val\_loss']

len(acc)

40

Selection deleted

import matplotlib.pyplot as plt

plt.figure(figsize=(8, 4))

plt.plot(range(EPOCHS), acc, label='Training Accuracy', color='blue')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.title('Training Accuracy')

plt.legend()

plt.grid(True)

plt.show()

plt.figure(figsize=(8, 4))

plt.plot(range(EPOCHS), val\_acc, label='Validation Accuracy', color='red')

plt.xlabel('Epochs')

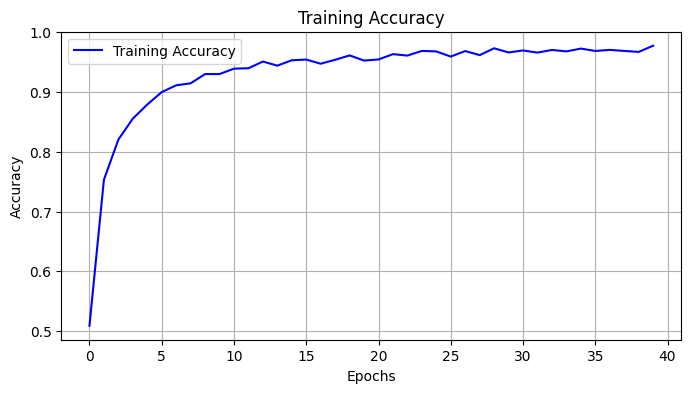
plt.ylabel('Accuracy')

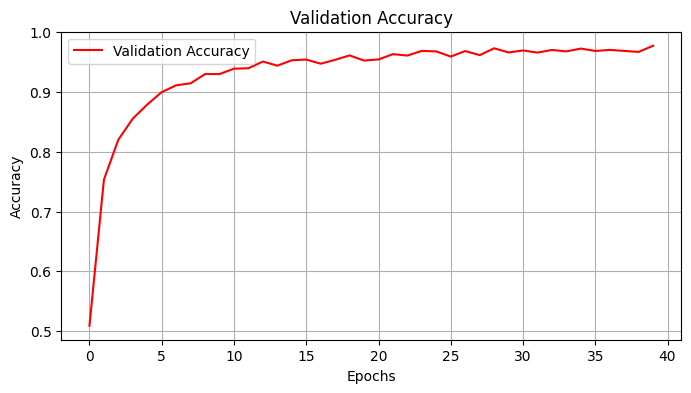
plt.title('Validation Accuracy')

plt.legend()

plt.grid(True)

plt.show()





import matplotlib.pyplot as plt

plt.figure(figsize=(8, 4))

# Plot training loss

plt.plot(range(EPOCHS), loss, label='Training Loss', color='blue')

# Plot validation loss

plt.plot(range(EPOCHS), val\_loss, label='Validation Loss', color='red')

plt.xlabel('Epochs')

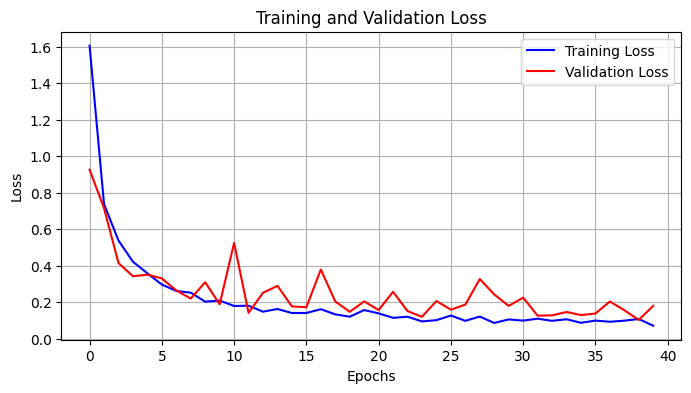
plt.ylabel('Loss')

plt.title('Training and Validation Loss')

plt.legend()

plt.grid(True)

plt.show()



import numpy as np

import matplotlib.pyplot as plt

image, label = next(iter(test\_ds))

plt.imshow(image[0].numpy().astype('uint8'))

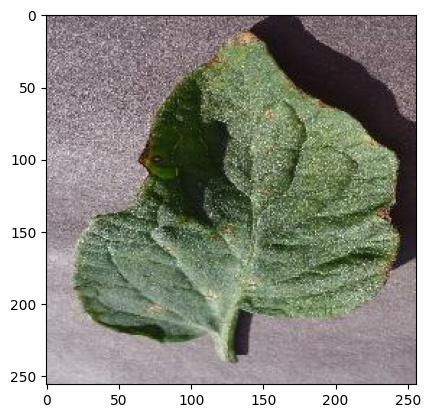
plt.show()

print("Actual label:", class\_names[label[0].numpy()])

prediction = model.predict(image)

predicted\_probabilities = prediction[0]

print("Predicted probabilities:", predicted\_probabilities)



Actual label: Tomato\_\_Target\_Spot

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

Predicted probabilities: [1.7452953e-07 4.2517889e-11 4.2178453e-12 2.3849415e-11 4.7031552e-13

7.6958059e-08 1.3967436e-05 1.0625347e-07 4.7693813e-09 2.3179207e-05

4.6357567e-08 9.9996138e-01 1.9930698e-11 9.5604948e-13 1.0525345e-06]

import numpy as np

for images\_batch, labels\_batch in test\_ds.take(1):

first\_image = images\_batch[0].numpy().astype('uint8')

first\_label = labels\_batch[0].numpy()

plt.imshow(first\_image)

plt.show()

print("Actual label:", class\_names[first\_label])

batch\_prediction = model.predict(images\_batch)

print("Predicted label:", class\_names[np.argmax(batch\_prediction[0])])

…>

import numpy as np

import matplotlib.pyplot as plt

from sklearn.metrics import f1\_score

# Collect all predicted labels for the test set

predicted\_labels = []

true\_labels = []

for images\_batch, labels\_batch in test\_ds:

predictions = model.predict(images\_batch)

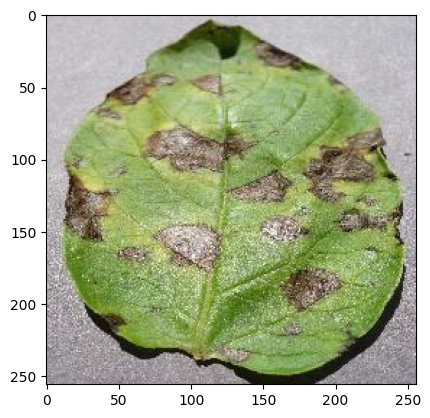
predicted\_labels.extend(np.argmax(predictions, axis=1))

true\_labels.extend(labels\_batch.numpy())

# Calculate F1 score

f1 = f1\_score(true\_labels, predicted\_labels, average='weighted')

print("F1 score:", f1)



Actual label: Potato\_\_\_Early\_blight

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 808ms/step

Predicted label: Potato\_\_\_Early\_blight

import numpy as np

import matplotlib.pyplot as plt

from sklearn.metrics import f1\_score

# Collect all predicted labels for the test set

predicted\_labels = []

true\_labels = []

for images\_batch, labels\_batch in test\_ds:

predictions = model.predict(images\_batch)

predicted\_labels.extend(np.argmax(predictions, axis=1))

true\_labels.extend(labels\_batch.numpy())

# Calculate F1 score

f1 = f1\_score(true\_labels, predicted\_labels, average='weighted')

print("F1 score:", f1)

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 806ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 663ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 777ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 814ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 867ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 791ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 823ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 766ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 806ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 712ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 928ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 785ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 815ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 813ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 789ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 767ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 871ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 846ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 776ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 776ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 738ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 780ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 883ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 772ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 728ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 776ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 802ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 789ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 783ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 726ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 716ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 927ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 818ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 777ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 804ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 713ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 840ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 833ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 881ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 803ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 820ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 788ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 730ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 910ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 802ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 857ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 775ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 814ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 525ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 506ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 920ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 901ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 969ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 944ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 891ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 952ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 868ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 920ms/step

F1 score: 0.9721214022040059

import numpy as np

# Initialize variables to count TP, FP, TN, FN

TP = 0

FP = 0

TN = 0

FN = 0

# Iterate over the test dataset to make predictions

for images\_batch, labels\_batch in test\_ds:

predictions\_batch = model.predict(images\_batch)

predicted\_labels\_batch = np.argmax(predictions\_batch, axis=1)

# Update TP, FP, TN, FN counts

for predicted\_label, true\_label in zip(predicted\_labels\_batch, labels\_batch):

if predicted\_label == true\_label:

if predicted\_label == 1: # Positive class

TP += 1

else: # Negative class

TN += 1

else:

if predicted\_label == 1: # Positive class

FP += 1

else: # Negative class

FN += 1

# Calculate metrics

accuracy = (TP + TN) / (TP + FN + TN + FP)

precision = TP / (TP + FP)

recall = TP / (TP + FN)

f1\_score = 2 \* (precision \* recall) / (precision + recall)

sensitivity = recall

specificity = TN / (FP + TN)

false\_positive\_rate = 1 - specificity

# Print the calculated metrics

print("Accuracy:", accuracy)

print("Precision:", precision)

print("Recall (Sensitivity):", recall)

print("F1 Score:", f1\_score)

print("Specificity:", specificity)

print("False Positive Rate:", false\_positive\_rate)

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 833ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 769ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 778ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 873ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 911ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 914ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 831ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 722ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 832ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 787ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 723ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 771ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 802ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 769ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 897ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 818ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 661ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 835ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 834ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 378ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 887ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 791ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 789ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 866ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 981ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 996ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 532ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 987ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 954ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 859ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 872ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 485ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 416ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 456ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 500ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 447ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 536ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 749ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 801ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 935ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 791ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 860ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 997ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 973ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 880ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 887ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 729ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 983ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 871ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 980ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 894ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 845ms/step

Accuracy: 0.9721153846153846

Precision: 0.9863013698630136

Recall (Sensitivity): 0.72

F1 Score: 0.8323699421965317

Specificity: 0.9989361702127659

False Positive Rate: 0.0010638297872340718

plt.figure(figsize=(20,20))

for images, labels in test\_ds.take(1):

for i in range(6):

ax = plt.subplot(3, 2, i+1)

plt.imshow(images[i].numpy().astype("uint8"))

predictions = model.predict(np.expand\_dims(images[i], axis=0))

predicted\_class = class\_names[np.argmax(predictions)]

confidence = np.max(predictions) \* 100

actual\_class = class\_names[labels[i]]

plt.title(f"Actual: {actual\_class},\nPredicted: {predicted\_class}.\nConfidence: {confidence:.2f}%")

plt.axis("off")

plt.show()

**trainingsec**

import tensorflow as tf

from tensorflow.keras import models ,layers

import matplotlib.pyplot as plt

len(temp\_test\_ds)

val\_size = 0.1

len(dataset) \* val\_size

val\_ds = temp\_test\_ds.take(64)

len(val\_ds)

test\_ds = temp\_test\_ds.skip(64)

len(test\_ds)

def get\_dataset\_partitions\_tf(ds, train\_split=0.8, val\_split=0.1, test\_split=0.1, shuffle=True, shufflesize=10000):

ds\_size = len(ds)

if shuffle:

ds = ds.shuffle(shufflesize, seed=12) # Fixed the variable name here

train\_size = int(train\_split \* ds\_size)

val\_size = int(val\_split \* ds\_size)

train\_ds = ds.take(train\_size)

val\_ds = ds.skip(train\_size).take(val\_size)

test\_ds = ds.skip(train\_size + val\_size) # Adjusted to skip both train and validation data

return train\_ds, val\_ds, test\_ds

train\_ds, val\_ds, test\_ds = get\_dataset\_partitions\_tf(dataset)

train\_ds = train\_ds.cache().shuffle(1000).prefetch(buffer\_size=tf.data.AUTOTUNE)

val\_ds = val\_ds.cache().shuffle(1000).prefetch(buffer\_size=tf.data.AUTOTUNE)

test\_ds = test\_ds.cache().shuffle(1000).prefetch(buffer\_size=tf.data.AUTOTUNE)

resize\_and\_rescale = tf.keras.Sequential([

layers.Resizing(IMAGE\_SIZE, IMAGE\_SIZE),

layers.Rescaling(1.0 / 255)

])

data\_augmentation = tf.keras.Sequential([

layers.RandomFlip("horizontal\_and\_vertical"),

layers.RandomRotation(0.2),

])

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense

n\_classes = 15

# Define the image size and number of channels

IMAGE\_SIZE = 256

CHANNELS = 3

# Define the rescaling factor

rescale\_factor = 1.0 / 255.0

# Define the model

model = Sequential([

Input(shape=(IMAGE\_SIZE, IMAGE\_SIZE, CHANNELS)), # Input layer with specified shape

Conv2D(32, (3, 3), activation='relu'),

MaxPooling2D((2, 2)),

Conv2D(64, kernel\_size=(3, 3), activation='relu'),

MaxPooling2D((2, 2)),

Conv2D(64, kernel\_size=(3, 3), activation='relu'),

MaxPooling2D((2, 2)),

Conv2D(64, (3, 3), activation='relu'),

MaxPooling2D((2, 2)),

Conv2D(64, (3, 3), activation='relu'),

MaxPooling2D((2, 2)),

Conv2D(64, (3, 3), activation='relu'),

MaxPooling2D((2, 2)),

Flatten(),

Dense(64, activation='relu'),

Dense(n\_classes, activation='softmax')

])

# Compile the model

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

# Print model summary

model.summary()

# Train the model

history = model.fit(

train\_ds.cache().shuffle(1000).prefetch(buffer\_size=tf.data.AUTOTUNE),

epochs=EPOCHS,

batch\_size=BATCH\_SIZE,

verbose=1,

validation\_data=val\_ds.cache().shuffle(1000).prefetch(buffer\_size=tf.data.AUTOTUNE)

)

# Evaluate the model

score = model.evaluate(test\_ds)

Found 20638 files belonging to 15 classes.

**Model: "sequential\_2"**

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓

┃ **Layer (type)** ┃ **Output Shape** ┃ **Param #** ┃

┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩

│ conv2d (Conv2D) │ (None, 254, 254, 32) │ 896 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ max\_pooling2d (MaxPooling2D) │ (None, 127, 127, 32) │ 0 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ conv2d\_1 (Conv2D) │ (None, 125, 125, 64) │ 18,496 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ max\_pooling2d\_1 (MaxPooling2D) │ (None, 62, 62, 64) │ 0 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ conv2d\_2 (Conv2D) │ (None, 60, 60, 64) │ 36,928 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ max\_pooling2d\_2 (MaxPooling2D) │ (None, 30, 30, 64) │ 0 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ conv2d\_3 (Conv2D) │ (None, 28, 28, 64) │ 36,928 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ max\_pooling2d\_3 (MaxPooling2D) │ (None, 14, 14, 64) │ 0 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ conv2d\_4 (Conv2D) │ (None, 12, 12, 64) │ 36,928 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ max\_pooling2d\_4 (MaxPooling2D) │ (None, 6, 6, 64) │ 0 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ conv2d\_5 (Conv2D) │ (None, 4, 4, 64) │ 36,928 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ max\_pooling2d\_5 (MaxPooling2D) │ (None, 2, 2, 64) │ 0 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ flatten (Flatten) │ (None, 256) │ 0 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ dense (Dense) │ (None, 64) │ 16,448 │

├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤

│ dense\_1 (Dense) │ (None, 15) │ 975 │

└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘

**Total params:** 184,527 (720.81 KB)

**Trainable params:** 184,527 (720.81 KB)

**Non-trainable params:** 0 (0.00 B)

Epoch 1/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 1757s 3s/step - accuracy: 0.3552 - loss: 2.4108 - val\_accuracy: 0.7056 - val\_loss: 0.9259

Epoch 2/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 1581s 3s/step - accuracy: 0.7369 - loss: 0.7935 - val\_accuracy: 0.7671 - val\_loss: 0.7147

Epoch 3/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 1713s 3s/step - accuracy: 0.8035 - loss: 0.5839 - val\_accuracy: 0.8604 - val\_loss: 0.4145

Epoch 4/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 1337s 3s/step - accuracy: 0.8563 - loss: 0.4245 - val\_accuracy: 0.8843 - val\_loss: 0.3416

Epoch 5/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 1382s 3s/step - accuracy: 0.8827 - loss: 0.3456 - val\_accuracy: 0.8809 - val\_loss: 0.3505

Epoch 6/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 1198s 2s/step - accuracy: 0.9033 - loss: 0.2829 - val\_accuracy: 0.8906 - val\_loss: 0.3299

Epoch 7/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 1090s 2s/step - accuracy: 0.9144 - loss: 0.2427 - val\_accuracy: 0.9165 - val\_loss: 0.2637

Epoch 8/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 1055s 2s/step - accuracy: 0.9212 - loss: 0.2308 - val\_accuracy: 0.9292 - val\_loss: 0.2194

Epoch 9/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 1281s 2s/step - accuracy: 0.9351 - loss: 0.1859 - val\_accuracy: 0.9028 - val\_loss: 0.3089

Epoch 10/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 1611s 3s/step - accuracy: 0.9394 - loss: 0.1858 - val\_accuracy: 0.9424 - val\_loss: 0.1879

Epoch 11/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 1610s 3s/step - accuracy: 0.9456 - loss: 0.1570 - val\_accuracy: 0.8315 - val\_loss: 0.5245

Epoch 12/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 1551s 3s/step - accuracy: 0.9418 - loss: 0.1718 - val\_accuracy: 0.9565 - val\_loss: 0.1409

Epoch 13/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 1569s 3s/step - accuracy: 0.9581 - loss: 0.1296 - val\_accuracy: 0.9165 - val\_loss: 0.2513

Epoch 14/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 1010s 2s/step - accuracy: 0.9418 - loss: 0.1703 - val\_accuracy: 0.9136 - val\_loss: 0.2892

Epoch 15/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 528s 1s/step - accuracy: 0.9572 - loss: 0.1295 - val\_accuracy: 0.9453 - val\_loss: 0.1764

Epoch 16/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 512s 992ms/step - accuracy: 0.9630 - loss: 0.1188 - val\_accuracy: 0.9478 - val\_loss: 0.1719

Epoch 17/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 543s 1s/step - accuracy: 0.9524 - loss: 0.1490 - val\_accuracy: 0.8867 - val\_loss: 0.3785

Epoch 18/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 609s 1s/step - accuracy: 0.9510 - loss: 0.1383 - val\_accuracy: 0.9443 - val\_loss: 0.2036

Epoch 19/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 596s 1s/step - accuracy: 0.9648 - loss: 0.1109 - val\_accuracy: 0.9604 - val\_loss: 0.1467

Epoch 20/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 600s 1s/step - accuracy: 0.9642 - loss: 0.1154 - val\_accuracy: 0.9438 - val\_loss: 0.2039

Epoch 21/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 605s 1s/step - accuracy: 0.9541 - loss: 0.1360 - val\_accuracy: 0.9512 - val\_loss: 0.1564

Epoch 22/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 601s 1s/step - accuracy: 0.9739 - loss: 0.0802 - val\_accuracy: 0.9209 - val\_loss: 0.2565

Epoch 23/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 589s 1s/step - accuracy: 0.9627 - loss: 0.1092 - val\_accuracy: 0.9590 - val\_loss: 0.1505

Epoch 24/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 569s 1s/step - accuracy: 0.9725 - loss: 0.0833 - val\_accuracy: 0.9658 - val\_loss: 0.1195

Epoch 25/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 615s 1s/step - accuracy: 0.9719 - loss: 0.0854 - val\_accuracy: 0.9478 - val\_loss: 0.2058

Epoch 26/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 600s 1s/step - accuracy: 0.9603 - loss: 0.1260 - val\_accuracy: 0.9580 - val\_loss: 0.1585

Epoch 27/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 601s 1s/step - accuracy: 0.9707 - loss: 0.0961 - val\_accuracy: 0.9507 - val\_loss: 0.1863

Epoch 28/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 598s 1s/step - accuracy: 0.9626 - loss: 0.1229 - val\_accuracy: 0.9160 - val\_loss: 0.3265

Epoch 29/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 1557s 3s/step - accuracy: 0.9750 - loss: 0.0731 - val\_accuracy: 0.9390 - val\_loss: 0.2419

Epoch 30/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 1428s 3s/step - accuracy: 0.9688 - loss: 0.1013 - val\_accuracy: 0.9453 - val\_loss: 0.1791

Epoch 31/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 502s 973ms/step - accuracy: 0.9709 - loss: 0.0973 - val\_accuracy: 0.9419 - val\_loss: 0.2245

Epoch 32/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 833s 2s/step - accuracy: 0.9707 - loss: 0.0923 - val\_accuracy: 0.9727 - val\_loss: 0.1257

Epoch 33/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 513s 993ms/step - accuracy: 0.9744 - loss: 0.0872 - val\_accuracy: 0.9741 - val\_loss: 0.1273

Epoch 34/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 514s 995ms/step - accuracy: 0.9774 - loss: 0.0732 - val\_accuracy: 0.9590 - val\_loss: 0.1457

Epoch 35/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 498s 965ms/step - accuracy: 0.9748 - loss: 0.0796 - val\_accuracy: 0.9644 - val\_loss: 0.1289

Epoch 36/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 511s 990ms/step - accuracy: 0.9694 - loss: 0.0938 - val\_accuracy: 0.9639 - val\_loss: 0.1373

Epoch 37/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 501s 970ms/step - accuracy: 0.9745 - loss: 0.0726 - val\_accuracy: 0.9424 - val\_loss: 0.2032

Epoch 38/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 514s 995ms/step - accuracy: 0.9639 - loss: 0.1165 - val\_accuracy: 0.9580 - val\_loss: 0.1557

Epoch 39/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 507s 982ms/step - accuracy: 0.9740 - loss: 0.0857 - val\_accuracy: 0.9717 - val\_loss: 0.1023

Epoch 40/40

516/516 ━━━━━━━━━━━━━━━━━━━━ 505s 980ms/step - accuracy: 0.9778 - loss: 0.0664 - val\_accuracy: 0.9551 - val\_loss: 0.1797

65/65 ━━━━━━━━━━━━━━━━━━━━ 210s 920ms/step - accuracy: 0.9731 - loss: 0.0893

history

<keras.src.callbacks.history.History at 0x24f9f1b3d90>

history.params

{'verbose': 1, 'epochs': 40, 'steps': 516}

history.history.keys()

dict\_keys(['accuracy', 'loss', 'val\_accuracy', 'val\_loss'])

acc = history.history['accuracy']

val\_acc = history.history['accuracy']

loss = history.history['loss']

val\_loss = history.history['val\_loss']

len(acc)

40

Selection deleted

import matplotlib.pyplot as plt

plt.figure(figsize=(8, 4))

plt.plot(range(EPOCHS), acc, label='Training Accuracy', color='blue')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.title('Training Accuracy')

plt.legend()

plt.grid(True)

plt.show()

plt.figure(figsize=(8, 4))

plt.plot(range(EPOCHS), val\_acc, label='Validation Accuracy', color='red')

plt.xlabel('Epochs')

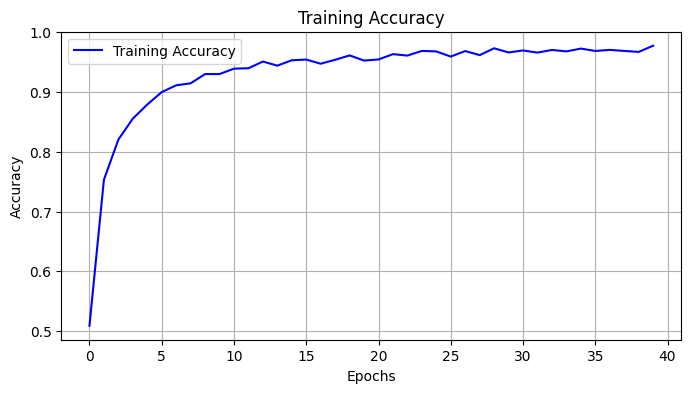
plt.ylabel('Accuracy')

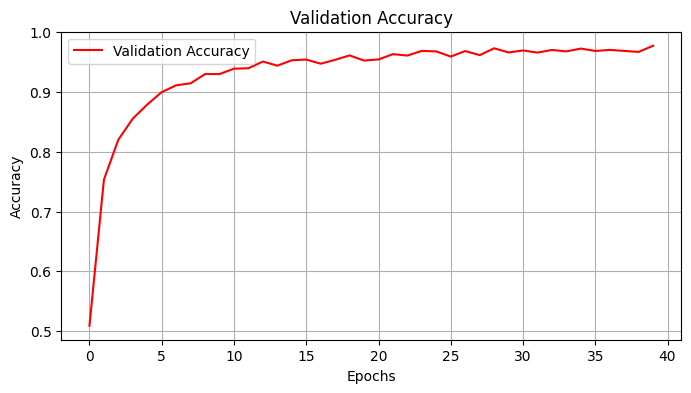
plt.title('Validation Accuracy')

plt.legend()

plt.grid(True)

plt.show()





import matplotlib.pyplot as plt

plt.figure(figsize=(8, 4))

# Plot training loss

plt.plot(range(EPOCHS), loss, label='Training Loss', color='blue')

# Plot validation loss

plt.plot(range(EPOCHS), val\_loss, label='Validation Loss', color='red')

plt.xlabel('Epochs')

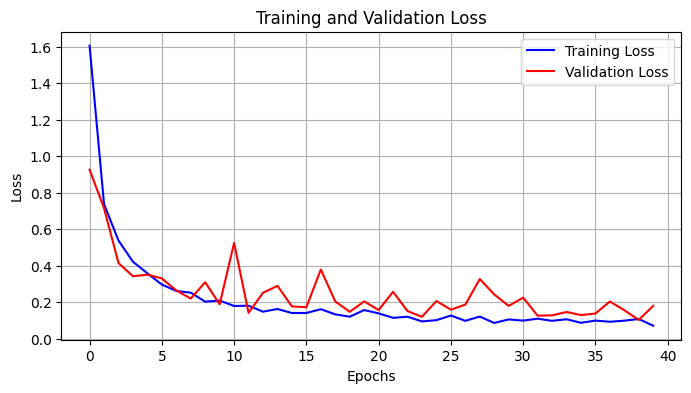
plt.ylabel('Loss')

plt.title('Training and Validation Loss')

plt.legend()

plt.grid(True)

plt.show()



import numpy as np

import matplotlib.pyplot as plt

image, label = next(iter(test\_ds))

plt.imshow(image[0].numpy().astype('uint8'))

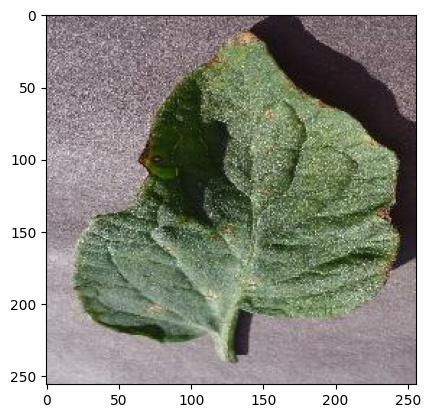
plt.show()

print("Actual label:", class\_names[label[0].numpy()])

prediction = model.predict(image)

predicted\_probabilities = prediction[0]

print("Predicted probabilities:", predicted\_probabilities)



Actual label: Tomato\_\_Target\_Spot

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

Predicted probabilities: [1.7452953e-07 4.2517889e-11 4.2178453e-12 2.3849415e-11 4.7031552e-13

7.6958059e-08 1.3967436e-05 1.0625347e-07 4.7693813e-09 2.3179207e-05

4.6357567e-08 9.9996138e-01 1.9930698e-11 9.5604948e-13 1.0525345e-06]

import numpy as np

for images\_batch, labels\_batch in test\_ds.take(1):

first\_image = images\_batch[0].numpy().astype('uint8')

first\_label = labels\_batch[0].numpy()

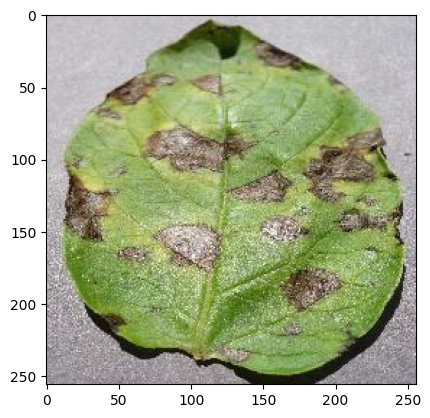
plt.imshow(first\_image)

plt.show()

print("Actual label:", class\_names[first\_label])

batch\_prediction = model.predict(images\_batch)

print("Predicted label:", class\_names[np.argmax(batch\_prediction[0])])



Actual label: Potato\_\_\_Early\_blight

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 808ms/step

Predicted label: Potato\_\_\_Early\_blight

import numpy as np

import matplotlib.pyplot as plt

from sklearn.metrics import f1\_score

# Collect all predicted labels for the test set

predicted\_labels = []

true\_labels = []

for images\_batch, labels\_batch in test\_ds:

predictions = model.predict(images\_batch)

predicted\_labels.extend(np.argmax(predictions, axis=1))

true\_labels.extend(labels\_batch.numpy())

# Calculate F1 score

f1 = f1\_score(true\_labels, predicted\_labels, average='weighted')

print("F1 score:", f1)

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 806ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 663ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 777ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 814ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 867ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 791ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 823ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 766ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 806ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 712ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 928ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 785ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 815ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 813ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 789ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 767ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 871ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 846ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 776ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 776ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 738ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 780ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 883ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 772ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 728ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 776ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 802ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 789ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 783ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 726ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 716ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 927ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 818ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 777ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 804ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 713ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 840ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 833ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 881ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 803ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 820ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 788ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 730ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 910ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 802ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 857ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 775ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 814ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 525ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 506ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 920ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 901ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 969ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 944ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 891ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 952ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 868ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 920ms/step

F1 score: 0.9721214022040059

import numpy as np

# Initialize variables to count TP, FP, TN, FN

TP = 0

FP = 0

TN = 0

FN = 0

# Iterate over the test dataset to make predictions

for images\_batch, labels\_batch in test\_ds:

predictions\_batch = model.predict(images\_batch)

predicted\_labels\_batch = np.argmax(predictions\_batch, axis=1)

# Update TP, FP, TN, FN counts

for predicted\_label, true\_label in zip(predicted\_labels\_batch, labels\_batch):

if predicted\_label == true\_label:

if predicted\_label == 1: # Positive class

TP += 1

else: # Negative class

TN += 1

else:

if predicted\_label == 1: # Positive class

FP += 1

else: # Negative class

FN += 1

# Calculate metrics

accuracy = (TP + TN) / (TP + FN + TN + FP)

precision = TP / (TP + FP)

recall = TP / (TP + FN)

f1\_score = 2 \* (precision \* recall) / (precision + recall)

sensitivity = recall

specificity = TN / (FP + TN)

false\_positive\_rate = 1 - specificity

# Print the calculated metrics

print("Accuracy:", accuracy)

print("Precision:", precision)

print("Recall (Sensitivity):", recall)

print("F1 Score:", f1\_score)

print("Specificity:", specificity)

print("False Positive Rate:", false\_positive\_rate)

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 833ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 769ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 778ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 873ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 911ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 914ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 831ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 722ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 832ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 787ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 723ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 771ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 802ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 769ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 897ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 818ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 661ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 835ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 834ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 378ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 887ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 791ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 789ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 866ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 981ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 996ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 532ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 987ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 954ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 859ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 872ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 485ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 416ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 456ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 500ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 447ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 536ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 749ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 801ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 935ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 791ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 860ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 997ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 973ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 880ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 887ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 729ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 983ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 871ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 980ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 894ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 845ms/step

Accuracy: 0.9721153846153846

Precision: 0.9863013698630136

Recall (Sensitivity): 0.72

F1 Score: 0.8323699421965317

Specificity: 0.9989361702127659

False Positive Rate: 0.0010638297872340718

plt.figure(figsize=(20,20))

for images, labels in test\_ds.take(1):

for i in range(6):

ax = plt.subplot(3, 2, i+1)

plt.imshow(images[i].numpy().astype("uint8"))

predictions = model.predict(np.expand\_dims(images[i], axis=0))

predicted\_class = class\_names[np.argmax(predictions)]

confidence = np.max(predictions) \* 100

actual\_class = class\_names[labels[i]]

plt.title(f"Actual: {actual\_class},\nPredicted: {predicted\_class}.\nConfidence: {confidence:.2f}%")

plt.axis("off")

plt.show()

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 306ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 79ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 107ms/step

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 80ms/step

