BINARY CLASSIFICATION USING SIMPLE CNN

```
import tensorflow as tf
import os

# Path to your dataset
# The folder structure should look like:
# dataset/
#   class1/
#       img1.jpg
#       img2.jpg
#   class2/
#       img3.jpg
#       img4.jpg
dataset_dir = "/content/drive/MyDrive/MANGODATASET"

# Parameters
img_size = (224, 224)   # resize images (you can change)
batch_size = 32
seed = 123  # Setting a seed fixes the randomness, so you get reproducible results across
                #runs.
# --- Step 1: Load training dataset (70%) ---
train_ds = tf.keras.utils.image_dataset_from_directory(
    dataset_dir,
    validation_split=0.30,   # 30% reserved for val+test
    subset="training",
    seed=seed,
    image_size=img_size,
    batch_size=batch_size
)
```

```
Found 600 files belonging to 2 classes.
Using 420 files for training.
```

```
# --- Step 2: Load temp dataset (30%) to split into val & test ---
temp_ds = tf.keras.utils.image_dataset_from_directory(
    dataset_dir,
    validation_split=0.30,
    subset="validation",
    seed=seed,
    image_size=img_size,
    batch_size=batch_size
)
```

```
Found 600 files belonging to 2 classes.
Using 180 files for validation.
```

```
# --- Step 3: Split temp_ds into validation (15%) and test (15%) ---
val_size = 0.5  # 50% of temp_ds → validation, 50% → test
```

```
val_batches = int(len(temp_ds) * val_size)

val_ds = temp_ds.take(val_batches)
test_ds = temp_ds.skip(val_batches)

# --- Check dataset sizes ---
print("Training batches:", len(train_ds))
print("Validation batches:", len(val_ds))
print("Testing batches:", len(test_ds))
```

```
Training batches: 14
Validation batches: 3
Testing batches: 3
```

```
import tensorflow as tf
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt

# ======================
# Define CNN Model
# ======================
model = models.Sequential([
    layers.Rescaling(1./255, input_shape=img_size + (3,)),  # normalize

    layers.Conv2D(32, (3, 3), activation="relu"),
    layers.MaxPooling2D(2, 2),

    layers.Conv2D(64, (3, 3), activation="relu"),
    layers.MaxPooling2D(2, 2),

    layers.Conv2D(128, (3, 3), activation="relu"),
    layers.MaxPooling2D(2, 2),

    layers.Flatten(),
    layers.Dense(128, activation="relu"),
    layers.Dropout(0.5),
    layers.Dense(1, activation="sigmoid")  # binary classification
])

# ======================
# Compile Model
# ======================
model.compile(
    optimizer="adam",
    loss="binary_crossentropy",
    metrics=["accuracy",
             tf.keras.metrics.Precision(name="precision"),
             tf.keras.metrics.Recall(name="recall"),
             tf.keras.metrics.AUC(name="auc")]
```

```
)

# ======================
# Train Model
# ======================
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=20
)

# ======================
# Evaluate on Test Set
# ======================
test_loss, test_acc, test_prec, test_rec, test_auc =
model.evaluate(test_ds)
print("\n=== Test Results ===")
print(f"Accuracy : {test_acc:.4f}")
print(f"Precision: {test_prec:.4f}")
print(f"Recall   : {test_rec:.4f}")
print(f"AUC      : {test_auc:.4f}")

# ======================
# Plot Training Curves
# ======================
plt.figure(figsize=(12, 5))

# Accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history["accuracy"], label="train_acc")
plt.plot(history.history["val_accuracy"], label="val_acc")
plt.title("Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()

# Loss
plt.subplot(1, 2, 2)
plt.plot(history.history["loss"], label="train_loss")
plt.plot(history.history["val_loss"], label="val_loss")
plt.title("Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()

plt.show()
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import numpy as np
```

```
# ======================
# Confusion Matrix on Test Set
# ======================

# Get true labels from test_ds
y_true = np.concatenate([y for x, y in test_ds], axis=0)

# Get predicted probabilities
y_pred_probs = model.predict(test_ds)
y_pred = (y_pred_probs > 0.5).astype("int32").flatten()  # threshold at
0.5

# Compute confusion matrix
cm = confusion_matrix(y_true, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=train_ds.class_names)

# Plot confusion matrix
disp.plot(cmap="Blues")
plt.title("Confusion Matrix")
plt.show()
```