Konecta Internship Task 5 (Image Classification using Deep Learning)

Name: Ahmed Ayman Ahmed Alhofy
Track: Artificial Intelligence & Machine Learning
Repository Link: https://github.com/AhmedAyman4/konecta-internship/tree/main/Task-5

Load Image Data and Explore it

In [1]:
```python
# Core libraries
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# TensorFlow / Keras
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models
from tensorflow.keras.applications import ResNet50, MobileNetV2
from tensorflow.keras.preprocessing import image_dataset_from_directory
from tensorflow.keras import optimizers
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tqdm.keras import TqdmCallback

# Scikit-learn (for evaluation metrics)
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay

# Optional (for reproducibility)
import random

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

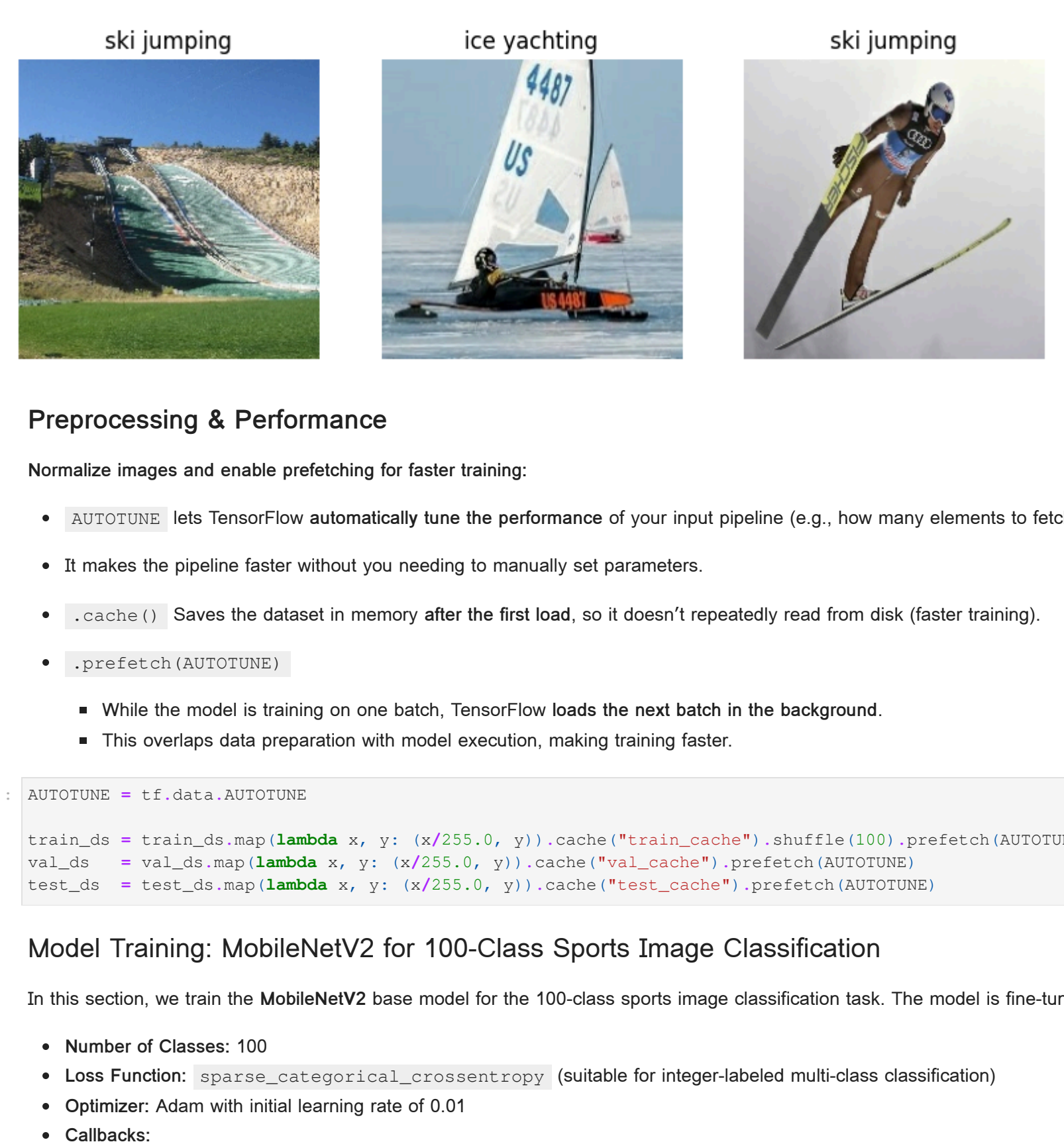
In [2]:
```python
import kagglehub

# Download latest version
path = kagglehub.dataset_download("gpiosenka/sports-classification")

print("Path to dataset files:", path)
```

```
Path to dataset files: /kaggle/input/sports-classification
```

In [3]:
```python
# Path from the KaggleHub log
base_dir = "/kaggle/input/sports-classification"

# Subdirectories
train_dir = f"{base_dir}/train"
val_dir   = f"{base_dir}/valid"
test_dir  = f"{base_dir}/test"

# Parameters
img_height, img_width = 224, 224   # dataset images are 224x224
batch_size = 16

# Load datasets
train_ds = image_dataset_from_directory(
    train_dir,
    image_size=(img_height, img_width),
    batch_size=batch_size,
    shuffle=True
)

val_ds = image_dataset_from_directory(
    val_dir,
    image_size=(img_height, img_width),
    batch_size=batch_size,
    shuffle=True
)

test_ds = image_dataset_from_directory(
    test_dir,
    image_size=(img_height, img_width),
    batch_size=batch_size,
    shuffle=False
)

# Class names
class_names = train_ds.class_names
print("Number of classes:", len(class_names))
print("Example classes:", class_names[:10])
```

```
Found 13492 files belonging to 100 classes.
Found 500 files belonging to 100 classes.
Found 500 files belonging to 100 classes.
Number of classes: 100
Example classes: ['air hockey', 'ampute football', 'archery', 'arm wrestling', 'axe throwing', 'balance beam', 'barell racing', 'baseball', 'basketball', 'baton twirling']
```

In [4]:
```python
# Preview some images from the training dataset
plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):   # take 1 batch
    for i in range(9):   # show first 9 images
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))  # convert tensor -> numpy -> image
        plt.title(class_names[labels[i]])  # show class name
        plt.axis("off")
```



Preprocessing & Performance

Normalize images and enable prefetching for faster training:

- `AUTOTUNE` lets TensorFlow **automatically tune the performance** of your input pipeline (e.g., how many elements to fetch in parallel).
- It makes the pipeline faster without you needing to manually set parameters.
- `.cache()` Saves the dataset in memory **after the first load**, so it doesn't repeatedly read from disk (faster training).
- `.prefetch(AUTOTUNE)`
  - While the model is training on one batch, TensorFlow **loads the next batch in the background**.
  - This overlaps data preparation with model execution, making training faster.

In [5]:
```python
AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.map(lambda x, y: (x/255.0, y)).cache("train_cache").shuffle(100).prefetch(AUTOTUNE)
val_ds   = val_ds.map(lambda x, y: (x/255.0, y)).cache("val_cache").prefetch(AUTOTUNE)
test_ds  = test_ds.map(lambda x, y: (x/255.0, y)).cache("test_cache").prefetch(AUTOTUNE)
```

Model Training: MobileNetV2 for 100-Class Sports Image Classification

In this section, we train the **MobileNetV2** base model for the 100-class sports image classification task. The model is fine-tuned using transfer learning with the following configuration:

- **Number of Classes:** 100
- **Loss Function:** `sparse_categorical_crossentropy` (suitable for integer-labeled multi-class classification)
- **Optimizer:** Adam with initial learning rate of 0.01
- **Callbacks:**
  - `EarlyStopping`: Monitors validation accuracy, stops training if no improvement for 5 epochs, and restores the best weights.
  - `ReduceLROnPlateau`: Reduces learning rate by a factor of 0.2 if validation loss does not improve for 2 epochs.
  - `TqdmCallback`: Provides a progress bar for training visualization.

The model is trained for up to **10 epochs**, with validation performance monitored at each epoch. Training may terminate early based on validation accuracy trends to prevent overfitting.

In [6]:
```python
def build_mobilenet(num_classes):
    base_model = MobileNetV2(weights="imagenet", include_top=False, input_shape=(224, 224, 3))
    base_model.trainable = False  # Freeze base model

    model = models.Sequential([
        base_model,
        layers.GlobalAveragePooling2D(),
        layers.Dropout(0.3),
        layers.Dense(num_classes, activation="softmax")
    ])
    return model
```

In [7]:
```python
num_classes = 100

# Define callbacks
early_stopping = EarlyStopping(
    monitor='val_accuracy',    # watch validation accuracy
    mode='max',                # higher is better
    patience=5,                # wait 5 epochs before stopping
    restore_best_weights=True # keep the best model
)


reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',  # watch validation loss
    factor=0.2,          # shrink learning rate to 20% of current
    patience=2,          # wait 2 bad epochs before reducing
    verbose=1
)


# MobileNetV2
mobilenet_model = build_mobilenet(num_classes)
mobilenet_model.compile(
    optimizer=optimizers.Adam(learning_rate=0.01),  # safer option would be the default 0.001
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"]
)

print("Training MobileNetV2...")
history_mobilenet = mobilenet_model.fit(
    train_ds,
    validation_data=val_ds,
    batch_size=16,
    epochs=6,  # higher max epochs, early stopping will stop earlier
    verbose=1,  # → one line per epoch (progress bar).
    callbacks=[TqdmCallback(verbose=1), early_stopping, reduce_lr]
)
```

```
Training MobileNetV2...
0epoch [00:00, ?epoch/s]
0batch [00:00, ?batch/s]
Epoch 1/6
844/844 ━━━━━━━━━━━━━━━━━━━━ 377s 424ms/step - accuracy: 0.5746 - loss: 2.5579 - val_accuracy: 0.8240 - val_loss: 0.9112 - learning_rate: 0.0100
Epoch 2/6
844/844 ━━━━━━━━━━━━━━━━━━━━ 376s 420ms/step - accuracy: 0.8055 - loss: 1.1838 - val_accuracy: 0.8640 - val_loss: 0.7968 - learning_rate: 0.0100
Epoch 3/6
844/844 ━━━━━━━━━━━━━━━━━━━━ 358s 410ms/step - accuracy: 0.8509 - loss: 0.9513 - val_accuracy: 0.8620 - val_loss: 1.0554 - learning_rate: 0.0100
Epoch 4/6
422/844 ━━━━━━━━━━━━━━━━━━━━ 5:34 792ms/step - accuracy: 0.8772 - loss: 0.8206
Epoch 4: ReduceLROnPlateau reducing learning rate to 0.0019999999552965165.
844/844 ━━━━━━━━━━━━━━━━━━━━ 359s 410ms/step - accuracy: 0.8747 - loss: 0.8459 - val_accuracy: 0.8680 - val_loss: 1.3052 - learning_rate: 0.0100
Epoch 5/6
844/844 ━━━━━━━━━━━━━━━━━━━━ 358s 412ms/step - accuracy: 0.9332 - loss: 0.3406 - val_accuracy: 0.8960 - val_loss: 0.7601 - learning_rate: 0.0020
Epoch 6/6
844/844 ━━━━━━━━━━━━━━━━━━━━ 359s 413ms/step - accuracy: 0.9465 - loss: 0.2441 - val_accuracy: 0.8940 - val_loss: 0.7669 - learning_rate: 0.0020
```

Model Evaluation

Evaluate the trained model using:

- **Accuracy**
- **Precision, Recall, F1-score** (per-class, macro & micro averages)

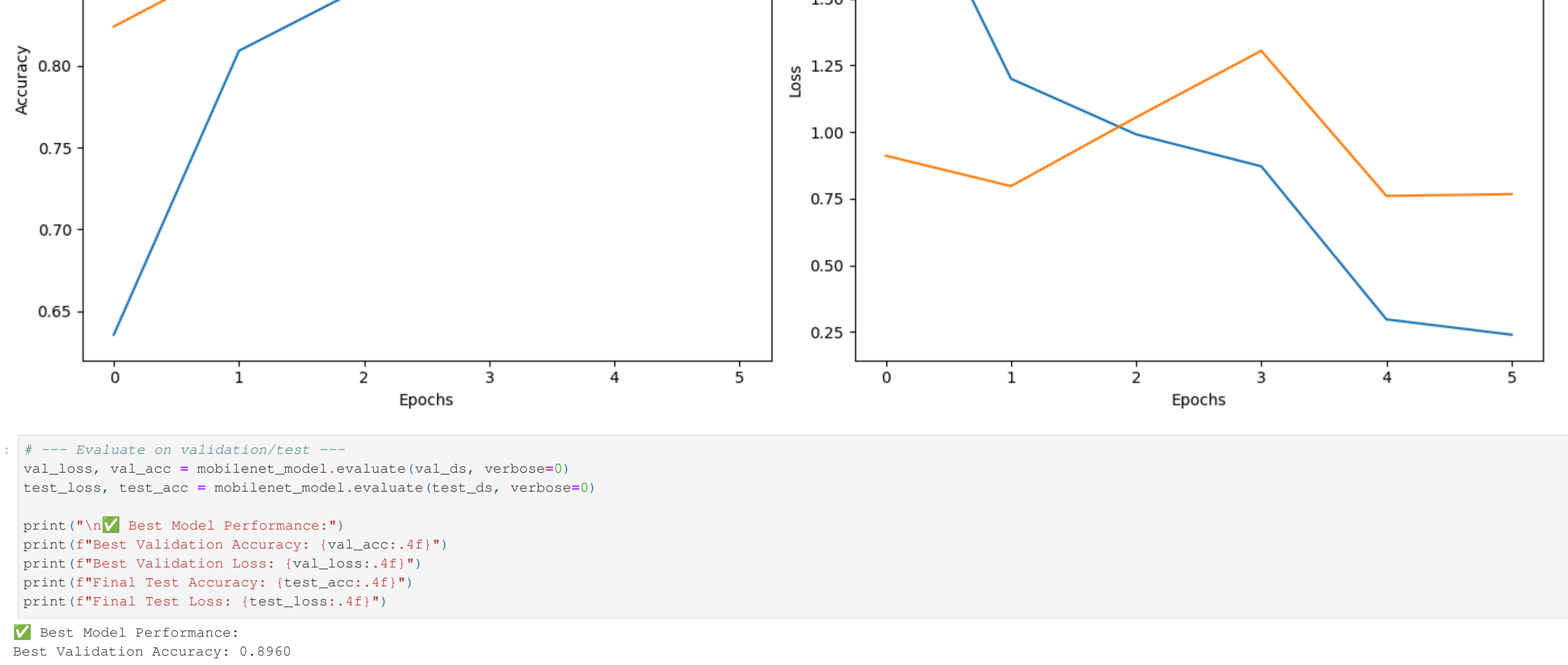
Plots to include:

- Training vs. Validation Accuracy over epochs
- Training vs. Validation Loss over epochs

In [9]:
```python
print("\nEvaluating model...")
val_loss, val_acc = mobilenet_model.evaluate(val_ds, verbose=1)
print(f"Validation Accuracy: {val_acc:.4f}")

# Collect true and predicted labels
y_true, y_pred = [], []
for images, labels in val_ds:
    preds = mobilenet_model.predict(images)
    y_true.extend(labels.numpy())    # true labels are already integer class indices
    y_pred.extend(np.argmax(preds, axis=1))            # predicted labels

y_true = np.array(y_true)
y_pred = np.array(y_pred)

# Classification report
print("\nClassification Report:")
print(classification_report(y_true, y_pred, digits=4))
```

```
Evaluating model...
32/32 ━━━━━━━━━━━━━━━━━━━━ 13s 390ms/step - accuracy: 0.9088 - loss: 0.7164
Validation Accuracy: 0.8960
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 788ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 772ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 770ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 774ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 770ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 743ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 983ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 778ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 818ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 765ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 791ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 756ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 754ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 759ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step

Classification Report:
              precision    recall  f1-score   support

           0     1.0000    1.0000    1.0000         5
           1     1.0000    1.0000    1.0000         5
           2     0.8333    1.0000    0.9091         5
           3     1.0000    1.0000    1.0000         5
           4     0.8000    0.8000    0.8000         5
           5     1.0000    0.6000    0.7500         5
           6     1.0000    0.8000    0.8889         5
           7     1.0000    0.8000    0.8889         5
           8     1.0000    0.8000    0.8889         5
           9     1.0000    1.0000    1.0000         5
          10     1.0000    1.0000    1.0000         5
          11     1.0000    1.0000    1.0000         5
          12     1.0000    1.0000    1.0000         5
          13     0.8000    0.8000    0.8000         5
          14     1.0000    1.0000    1.0000         5
          15     0.8333    1.0000    0.9091         5
          16     0.7143    1.0000    0.8333         5
          17     1.0000    1.0000    1.0000         5
          18     1.0000    1.0000    1.0000         5
          19     1.0000    1.0000    1.0000         5
          20     0.7143    1.0000    0.8333         5
          21     1.0000    1.0000    1.0000         5
          22     1.0000    1.0000    1.0000         5
          23     1.0000    1.0000    1.0000         5
          24     1.0000    0.6000    0.7500         5
          25     0.7143    1.0000    0.8333         5
          26     0.8333    1.0000    0.9091         5
          27     1.0000    0.8000    0.8889         5
          28     0.7143    1.0000    0.8333         5
          29     1.0000    1.0000    1.0000         5
          30     0.8333    1.0000    0.9091         5
          31     0.8333    1.0000    0.9091         5
          32     1.0000    1.0000    1.0000         5
          33     0.5000    0.8000    0.6154         5
          34     1.0000    1.0000    1.0000         5
          35     1.0000    0.8000    0.8889         5
          36     0.8333    1.0000    0.9091         5
          37     0.7500    0.6000    0.6667         5
          38     1.0000    1.0000    1.0000         5
          39     1.0000    0.8000    0.8889         5
          40     1.0000    0.8000    0.8889         5
          41     1.0000    1.0000    1.0000         5
          42     1.0000    1.0000    1.0000         5
          43     0.8333    1.0000    0.9091         5
          44     0.8000    0.8000    0.8000         5
          45     1.0000    1.0000    1.0000         5
          46     1.0000    0.6000    0.7500         5
          47     0.8333    1.0000    0.9091         5
          48     1.0000    1.0000    1.0000         5
          49     1.0000    1.0000    1.0000         5
          50     1.0000    0.6000    0.7500         5
          51     1.0000    0.8000    0.8889         5
          52     1.0000    0.4000    0.5714         5
          53     1.0000    0.8000    0.8889         5
          54     1.0000    1.0000    1.0000         5
          55     0.6667    0.8000    0.7273         5
          56     0.6250    1.0000    0.7692         5
          57     1.0000    1.0000    1.0000         5
          58     1.0000    1.0000    1.0000         5
          59     1.0000    1.0000    1.0000         5
          60     0.8333    1.0000    0.9091         5
          61     1.0000    1.0000    1.0000         5
          62     0.8333    1.0000    0.9091         5
          63     0.8000    0.8000    0.8000         5
          64     1.0000    1.0000    1.0000         5
          65     0.6667    0.8000    0.7273         5
          66     0.8333    1.0000    0.9091         5
          67     1.0000    1.0000    1.0000         5
          68     1.0000    0.8000    0.8889         5
          69     0.6667    0.8000    0.7273         5
          70     1.0000    1.0000    1.0000         5
          71     0.8333    1.0000    0.9091         5
          72     1.0000    1.0000    1.0000         5
          73     1.0000    0.6000    0.7500         5
          74     1.0000    0.8000    0.8889         5
          75     1.0000    0.8000    0.8889         5
          76     1.0000    0.6000    0.7500         5
          77     0.6667    0.8000    0.7273         5
          78     0.8333    1.0000    0.9091         5
          79     0.6000    0.6000    0.6000         5
          80     1.0000    0.8000    0.8889         5
          81     0.7500    0.6000    0.6667         5
          82     1.0000    0.6000    0.7500         5
          83     0.8333    1.0000    0.9091         5
          84     0.8333    1.0000    0.9091         5
          85     1.0000    1.0000    1.0000         5
          86     1.0000    0.8000    0.8889         5
          87     0.7143    1.0000    0.8333         5
          88     1.0000    0.8000    0.8889         5
          89     1.0000    1.0000    1.0000         5
          90     0.7143    1.0000    0.8333         5
          91     1.0000    0.4000    0.5714         5
          92     1.0000    0.8000    0.8889         5
          93     1.0000    0.8000    0.8889         5
          94     1.0000    1.0000    1.0000         5
          95     1.0000    1.0000    1.0000         5
          96     1.0000    1.0000    1.0000         5
          97     1.0000    1.0000    1.0000         5
          98     1.0000    1.0000    1.0000         5
          99     0.7143    1.0000    0.8333         5

    accuracy                         0.8960       500
   macro avg     0.9159    0.8960    0.8940       500
weighted avg     0.9159    0.8960    0.8940       500
```

In [11]:
```python
# Training Curves (Side by Side)
# ------------------------
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))

# Accuracy subplot
ax1.plot(history_mobilenet.history["accuracy"], label="Train Accuracy")
ax1.plot(history_mobilenet.history["val_accuracy"], label="Validation Accuracy")
ax1.set_xlabel("Epochs")
ax1.set_ylabel("Accuracy")
ax1.set_title("Training vs Validation Accuracy")
ax1.legend()

# Loss subplot
ax2.plot(history_mobilenet.history["loss"], label="Train Loss")
ax2.plot(history_mobilenet.history["val_loss"], label="Validation Loss")
ax2.set_xlabel("Epochs")
ax2.set_ylabel("Loss")
ax2.set_title("Training vs Validation Loss")
ax2.legend()

plt.tight_layout()
plt.show()
```



In [12]:
```python
# --- Evaluate on validation/test ---
val_loss, val_acc = mobilenet_model.evaluate(val_ds, verbose=0)
test_loss, test_acc = mobilenet_model.evaluate(test_ds, verbose=0)

print("\n✅ Best Model Performance:")
print(f"Best Validation Accuracy: {val_acc:.4f}")
print(f"Best Validation Loss: {val_loss:.4f}")
print(f"Final Test Accuracy: {test_acc:.4f}")
print(f"Final Test Loss: {test_loss:.4f}")
```

```
✅ Best Model Performance:
Best Validation Accuracy: 0.8960
Best Validation Loss: 0.7601
Final Test Accuracy: 0.9100
Final Test Loss: 0.5517
```

Sample Predictions on Test Data

Visualize 16 test images with model predictions vs. true labels:

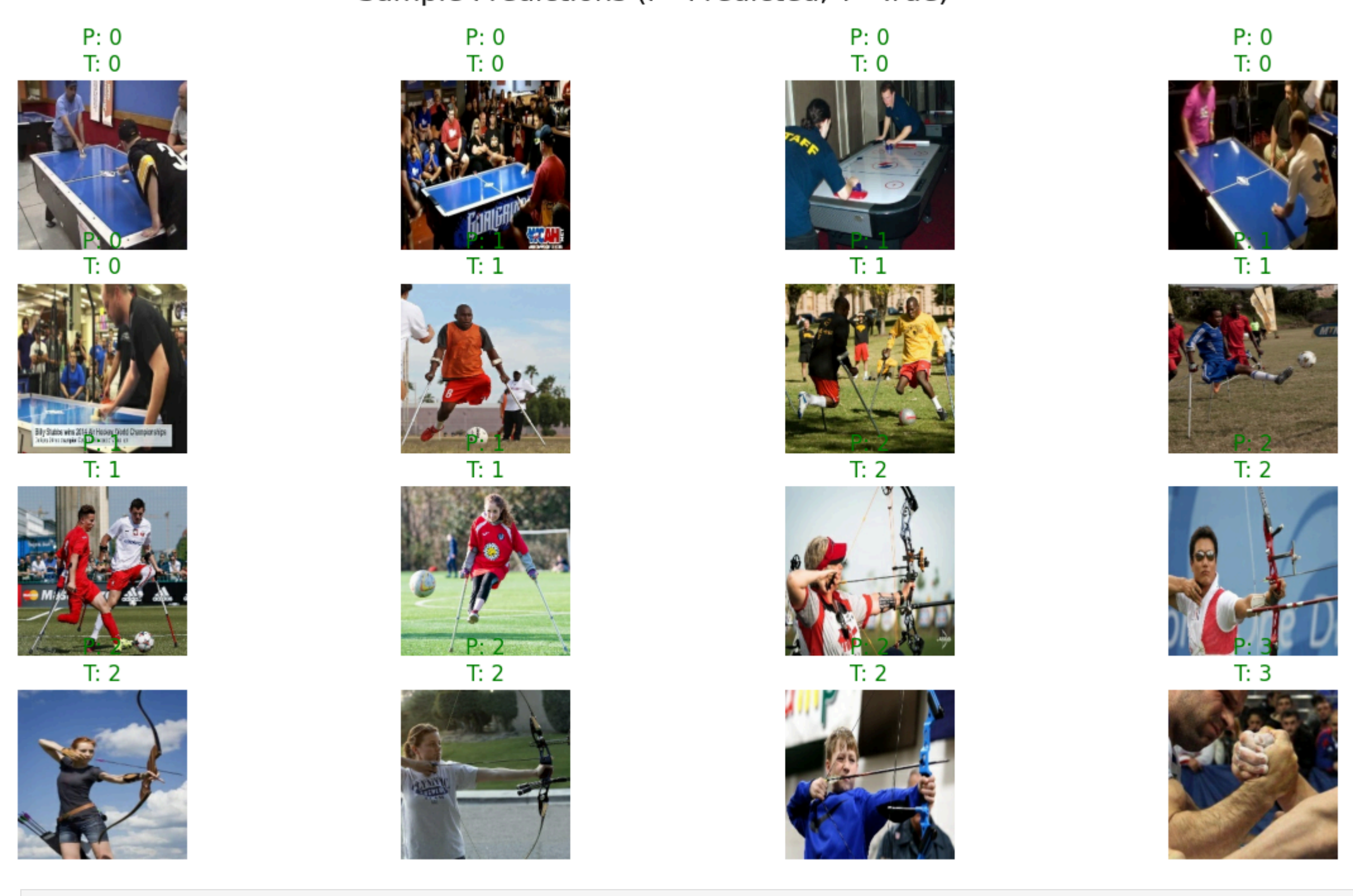
- Green title: Correct prediction
- Red title: Incorrect prediction

Uses batch from `test_ds` and converts class indices to names using `class_names`.
Prediction is done via `mobilenet_model.predict()`.

In [13]:
```python
# Get one batch of test data
images, labels = next(iter(test_ds.unbatch().batch(16)))  # 16 sample images

# Run predictions
pred_probs = mobilenet_model.predict(images)
pred_classes = np.argmax(pred_probs, axis=1)

# Plot results
plt.figure(figsize=(15, 8))
for i in range(16):
    plt.subplot(4, 4, i + 1)
    plt.imshow(images[i].numpy())
    plt.axis("off")

    true_label = labels[i].numpy()
    pred_label = pred_classes[i]

    # Show predicted and true labels
    color = "green" if pred_label == true_label else "red"
    plt.title(f"P: {pred_label}\nT: {true_label}", color=color)

plt.suptitle("Sample Predictions (P=Predicted, T=True)", fontsize=16)
plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 563ms/step
```

Sample Predictions (P=Predicted, T=True)



In [15]:
```python
# Get class names from dataset
# class_names = train_ds.class_names # This line caused the error, using the existing class_names variable instead

# Take a batch of 16 test images
images, labels = next(iter(test_ds.unbatch().batch(16)))

# Get predictions
pred_probs = mobilenet_model.predict(images)
pred_classes = np.argmax(pred_probs, axis=1)

# Plot images with predicted vs true labels
plt.figure(figsize=(15, 8))
for i in range(16):
    plt.subplot(4, 4, i + 1)
    plt.imshow(images[i].numpy())
    plt.axis("off")

    true_label = labels[i].numpy()
    pred_label = pred_classes[i]

    # Convert class index → class name
    true_name = class_names[true_label]
    pred_name = class_names[pred_label]

    # Show result
    color = "green" if pred_label == true_label else "red"
    plt.title(f"P: {pred_name}\nT: {true_name}", color=color, fontsize=9)

plt.suptitle("Sample Predictions (P=Predicted, T=True)", fontsize=16)
plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 522ms/step
```

Sample Predictions (P=Predicted, T=True)

In [16]:
```python
# --- Save the best model ---
mobilenet_model.save("mobilenet_best.h5")
print("\n💾 Model saved as 'mobilenet_best.h5'")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Ker
as format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
💾 Model saved as 'mobilenet_best.h5'
```