Here's a **complete image classification script** using your **own image folders** (e.g., `cats/`, `dogs/`, etc.) using **TensorFlow and Keras**.

□ What this does:

- Loads images from your own folder structure like:

```
dataset/
├── cats/
│   ├── cat1.jpg
│   ├── cat2.jpg
│   └── ...
├── dogs/
│   ├── dog1.jpg
│   ├── dog2.jpg
│   └── ...
```

- Automatically labels based on folder names (e.g., `"cats"`, `"dogs"`).
- Trains a CNN to classify images.
- Evaluates and visualizes the results.

Full python script image-classification-image-folder.py :-

```python
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.preprocessing import image_dataset_from_directory

# Step 1: Set up directory paths
data_dir = 'dataset'  # change this to your actual dataset path
img_height, img_width = 180, 180
batch_size = 32

# Step 2: Load the dataset
train_ds = image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
```

```
    subset="training",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size
)

val_ds = image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size
)

class_names = train_ds.class_names
print("Class Names:", class_names)

# Step 3: Prefetch for performance
AUTOTUNE = tf.data.AUTOTUNE
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)

# Step 4: Build the CNN model
model = models.Sequential([
    layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),

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

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

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

    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(len(class_names), activation='softmax')
])

# Step 5: Compile the model
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
```

```
    metrics=['accuracy']
)

# Step 6: Train the model
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=10
)

# Step 7: Evaluate the model
loss, acc = model.evaluate(val_ds)
print(f"\nValidation Accuracy: {acc:.2f}")

# Step 8: Plot accuracy and loss curves
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.legend()
plt.title('Accuracy')

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.legend()
plt.title('Loss')

plt.show()

# Step 9: Make predictions on validation samples
for images, labels in val_ds.take(1):
    preds = model.predict(images)
    pred_labels = np.argmax(preds, axis=1)

    plt.figure(figsize=(12, 6))
    for i in range(6):
        plt.subplot(2, 3, i+1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(f"Actual: {class_names[labels[i]]}\nPredicted:
{class_names[pred_labels[i]]}")
        plt.axis('off')
    plt.tight_layout()
    plt.show()
```

Let's go line-by-line and explain **each part of the image classification script** that uses your **own image folders** (like `cats/`, `dogs/`, etc.) with TensorFlow and Keras.

✅ Full Code with Detailed Explanation

✅ Imports and Configuration

```
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.preprocessing import image_dataset_from_directory
```

- `os` – for working with paths (not strictly used in the current code, but useful)
- `numpy` – for numerical operations like array manipulation
- `matplotlib.pyplot` – for plotting images, accuracy/loss graphs
- `tensorflow` – the main deep learning framework used here
- `layers, models` – convenient access to Keras model layers
- `image_dataset_from_directory` – helper function to load images from folders and auto-label based on folder names

---

✅ Set Up Your Dataset Path and Parameters

```
data_dir = 'dataset'  # change this to your actual dataset path
img_height, img_width = 180, 180
batch_size = 32
```

- `data_dir`: Folder where your images are stored, structured by subfolders for each class (e.g., `cats/`, `dogs/`)
- `img_height`, `img_width`: Images will be resized to this shape when loaded
- `batch_size`: Number of images loaded in one training step

✅ Load Training and Validation Data

```
train_ds = image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=(img_height, img_width),
```

```
    batch_size=batch_size
)
```

- Loads 80% of images (because `validation_split=0.2`) for training.
- Automatically assigns labels based on folder names.
- `seed=123` ensures repeatability (same data is split each time)
- `image_size=(180, 180)` resizes all images
- `batch_size=32` loads 32 images per batch

```
val_ds = image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size
)
```

- Loads the remaining 20% of data for validation (to monitor overfitting)
- Same parameters to ensure consistency

✅ Print Class Names

```
class_names = train_ds.class_names
print("Class Names:", class_names)
```

- `class_names` contains the subfolder names (e.g., `['cats', 'dogs']`)
- Automatically assigned as labels

✅ Improve Performance with Caching and Prefetching

```
AUTOTUNE = tf.data.AUTOTUNE
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

- `cache()` stores the data in memory after first read for faster reuse
- `shuffle(1000)` randomly shuffles 1000 samples (for training only)
- `prefetch()` overlaps data preprocessing with model training

🧠 Build the CNN Model

```
model = models.Sequential([
    layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
```

- `Rescaling(1./255)` normalizes pixel values from `[0, 255]` to `[0, 1]`
- `input_shape=(180, 180, 3)` expects 180x180 RGB images

```
    layers.Conv2D(32, 3, activation='relu'),
    layers.MaxPooling2D(),
```

- `Conv2D(32, 3, ...)`: 32 filters of size 3x3, detects features
- `ReLU`: adds non-linearity
- `MaxPooling2D()`: reduces image dimensions (downsampling)

```
    layers.Conv2D(64, 3, activation='relu'),
    layers.MaxPooling2D(),
```

- More filters = detect more complex features
- Pool again to reduce size

```
    layers.Conv2D(128, 3, activation='relu'),
    layers.MaxPooling2D(),
```

- Even more filters for deeper feature extraction

```
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(len(class_names), activation='softmax')
])
```

- `Flatten()`: Converts 2D image features into 1D
- `Dense(128)`: Fully connected layer with 128 neurons
- `Dense(len(class_names))`: Final layer, one output per class
- `softmax`: Converts outputs into probabilities for each class

⚙️ Compile the Model

```
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

- `adam`: Adaptive learning optimizer
- `sparse_categorical_crossentropy`: Used for multi-class classification with integer labels
- `accuracy`: Tracks classification performance

🏋️ Train the Model

```
history = model.fit(
    train_ds,
    validation_data=val_ds,
    epochs=10
)
```

- Trains for 10 epochs (you can increase this)
- Validates on `val_ds` during training
- `history` stores accuracy and loss over time

Evaluate the Model on Validation Data

```
loss, acc = model.evaluate(val_ds)
print(f"\nValidation Accuracy: {acc:.2f}")
```

- Prints accuracy of the model on validation set

Plot Accuracy and Loss Graphs

```
plt.figure(figsize=(12, 4))
```

Creates a wide figure with 2 plots side by side.

```
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.legend()
plt.title('Accuracy')
```

- Plots training and validation accuracy per epoch

```
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.legend()
plt.title('Loss')
```

- Plots training and validation loss per epoch

```
plt.show()
```

- Displays both plots

Make Predictions and Show Sample Results

```
for images, labels in val_ds.take(1):
    preds = model.predict(images)
    pred_labels = np.argmax(preds, axis=1)
```

- `val_ds.take(1)` takes 1 batch of images (32 images)
- `model.predict()` outputs probabilities for each class
- `argmax()` picks the class with the highest probability

```
    plt.figure(figsize=(12, 6))
    for i in range(6):
        plt.subplot(2, 3, i+1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(f"Actual: {class_names[labels[i]]}\nPredicted:
{class_names[pred_labels[i]]}")
        plt.axis('off')
    plt.tight_layout()
    plt.show()
```

- Displays the first 6 images with their predicted and actual labels
- Converts tensors back to image format using `.numpy().astype("uint8")`

□ Folder Structure Recap

```
dataset/
├── cats/
│   ├── cat1.jpg
├── dogs/
│   ├── dog1.jpg
```

Each subfolder name becomes a class label.