```
import os
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, optimizers
from sklearn.metrics import classification_report, confusion_matrix

DATA_DIR = "caltech-101-img"
WEIGHTS_FILE = "vgg16_weights_tf_dim_ordering_tf_kernels_notop (1).h5"
IMAGE_SIZE = (224, 224)        # VGG default
BATCH_SIZE = 32
EPOCHS = 10
LR = 0.001

train_ds = keras.preprocessing.image_dataset_from_directory(
    DATA_DIR,
    labels="inferred",
    label_mode="int",
    image_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    validation_split=0.2,
    subset="training",
    seed=42
)
val_ds = keras.preprocessing.image_dataset_from_directory(
    DATA_DIR,
    labels="inferred",
    label_mode="int",
    image_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE,
    validation_split=0.2,
    subset="validation",
    seed=42
)

class_names = train_ds.class_names
num_classes = len(class_names)
print("Detected classes:", num_classes, class_names)

preprocess = keras.applications.vgg16.preprocess_input

def preprocess_ds(ds):
    return ds.map(lambda x, y: (preprocess(tf.cast(x, tf.float32)), y),
                  num_parallel_calls=tf.data.AUTOTUNE)

train_ds = preprocess_ds(train_ds).cache().prefetch(tf.data.AUTOTUNE)
val_ds = preprocess_ds(val_ds).cache().prefetch(tf.data.AUTOTUNE)
```

```
from tensorflow.keras.applications import VGG16

base_model = VGG16(include_top=False, weights=None, input_shape=(IMAGE_SIZE[0],
IMAGE_SIZE[1], 3))
base_model.load_weights(WEIGHTS_FILE)
base_model.trainable = False  # freeze base

inputs = keras.Input(shape=(IMAGE_SIZE[0], IMAGE_SIZE[1], 3))
x = base_model(inputs, training=False)
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dense(256, activation="relu")(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(num_classes, activation="softmax")(x)
model = keras.Model(inputs, outputs, name="vgg16_transfer")

model.summary()

optimizer = optimizers.SGD(learning_rate=LR, momentum=0.9)
model.compile(optimizer=optimizer, loss="sparse_categorical_crossentropy",
metrics=["accuracy"])

history = model.fit(train_ds, epochs=EPOCHS, validation_data=val_ds, verbose=2)

y_true = []
y_pred = []
images_for_display = []
labels_for_display = []

# gather all validation examples (may be moderate size)
for batch_imgs, batch_labels in val_ds.unbatch().batch(BATCH_SIZE):
    preds = model.predict(batch_imgs, verbose=0)
    y_true.extend(batch_labels.numpy().tolist())
    y_pred.extend(np.argmax(preds, axis=1).tolist())
    # collect up to 16 images for display AS NUMPY ARRAYS (not lists)
    if len(images_for_display) < 16:
        need = 16 - len(images_for_display)
        imgs_np = batch_imgs.numpy()[:need]          # this is a numpy array
        labs_np = batch_labels.numpy()[:need]
        # ensure we append numpy arrays (not python lists)
        for im_arr, lab in zip(imgs_np, labs_np):
            images_for_display.append(im_arr.copy())
            labels_for_display.append(int(lab))

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

print("\nValidation classification report:")
```

```
print(classification_report(y_true, y_pred, target_names=class_names, zero_division=0))
cm = confusion_matrix(y_true, y_pred)
print("Confusion matrix shape:", cm.shape)

plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
plt.plot(history.history["loss"], label="train loss")
plt.plot(history.history["val_loss"], label="val loss")
plt.title("Loss"); plt.xlabel("Epoch"); plt.legend()
plt.subplot(1,2,2)
plt.plot(history.history["accuracy"], label="train acc")
plt.plot(history.history["val_accuracy"], label="val acc")
plt.title("Accuracy"); plt.xlabel("Epoch"); plt.legend()
plt.tight_layout()
plt.show()

def deprocess_vgg(x):
    # x is a preprocessed array in float32 (single image)
    y = x.copy()
    # add VGG mean (BGR order)
    y[..., 0] += 103.939
    y[..., 1] += 116.779
    y[..., 2] += 123.68
    # BGR -> RGB
    y = y[..., ::-1]
    y = np.clip(y, 0, 255).astype("uint8")
    return y

plt.figure(figsize=(12,12))
for i, img in enumerate(images_for_display[:16]):
    plt.subplot(4,4,i+1)
    img_np = deprocess_vgg(img)
    preds = model.predict(np.expand_dims(img, axis=0), verbose=0)
    pred_label = class_names[int(np.argmax(preds))]
    true_label = class_names[int(labels_for_display[i])]
    plt.imshow(img_np)
    plt.title(f"P:{pred_label}\nT:{true_label}")
    plt.axis("off")
plt.tight_layout()
plt.show()
```

🔹 1. Deep Learning and CNN

- **Deep Learning** is a subset of Machine Learning that uses neural networks with multiple layers to automatically learn features from data.

- **Convolutional Neural Networks (CNNs)** are used for image-related tasks such as classification, detection, and segmentation.
  CNNs use convolutional layers to extract spatial features (edges, textures, shapes) from images.

🔹 2. Transfer Learning

- **Transfer Learning** is a method where a pre-trained model (trained on a large dataset like ImageNet) is reused for a new but related task.

- Instead of training a model from scratch, we use the **feature extraction layers** from a pre-trained network and only train the **final layers** on our dataset.

- This helps in:

    - Faster training

    - Better accuracy with limited data

    - Avoiding overfitting

🔹 3. VGG16 Architecture

- **VGG16** is a popular CNN model developed by Oxford's Visual Geometry Group (VGG).

- It consists of:

    - 13 **convolutional layers**

    - 5 **max pooling layers**

    - 3 **fully connected layers**

- It uses **3×3 convolution filters** and **ReLU activation**.

- The model is trained on **ImageNet** dataset (over 1 million images, 1000 classes).

🔹 4. Dataset – Caltech-101

- The **Caltech-101 dataset** contains **101 object categories** (e.g., animals, vehicles, instruments, etc.).
- Each class has **40–800 images**.
- The images are of varying sizes and are resized to **224×224 pixels** for VGG16 input.

🔹 5. Model Workflow

1. **Dataset Preparation:**
    - Images are loaded from folders using `image_dataset_from_directory()`.
    - Split into training (80%) and validation (20%).
    - Labels are automatically inferred from folder names.
    - Preprocessing is done using `vgg16.preprocess_input()` to match VGG16's input format.
2. **Model Building:**
    - Import VGG16 with `include_top=False` to exclude its final classification layer.
    - Load **pre-trained ImageNet weights**.
    - **Freeze base layers** to keep pre-learned features.
    - Add new layers:
        - `GlobalAveragePooling2D` – reduces spatial dimensions.
        - `Dense(256, ReLU)` – fully connected layer.
        - `Dropout(0.5)` – prevents overfitting.
        - `Dense(num_classes, Softmax)` – output layer for classification.

3. **Compilation:**
    - Optimizer: **SGD** (Stochastic Gradient Descent) with momentum = 0.9.
    - Loss: **Sparse Categorical Crossentropy**.
    - Metric: **Accuracy**.
4. **Training:**
    - Train for 10 epochs on training data.
    - Validate on validation data.
5. **Evaluation:**
    - Compute **classification report** and **confusion matrix**.
    - Plot **loss** and **accuracy** curves.
6. **Visualization:**
    - Display sample images with predicted and true labels.

🔹 6. Important Concepts

Concept	Description
Feature Extraction	Using pre-trained CNN layers to extract meaningful features from images.
Freezing Layers	Prevents updating weights of pre-trained layers during training.
Fine-Tuning	Optionally unfreezing top layers of the base model for slight adjustment.
Global Average Pooling	Converts feature maps into a single vector per feature channel.
Dropout	Randomly drops neurons during training to reduce overfitting.

🔹 7. Advantages of Transfer Learning

- Reduces training time.

- Requires less data.
- Improves accuracy for smaller datasets.
- Leverages previously learned general features.

🧮 Result:

The VGG16 Transfer Learning model successfully classifies images from the Caltech-101 dataset with good accuracy.
Training and validation graphs show the model's performance improvement over epochs.

📊 Conclusion:

Transfer Learning using VGG16 effectively performs image classification with limited data by reusing learned features from ImageNet. The model achieves high validation accuracy and demonstrates the efficiency of pre-trained CNN architectures.