User Manual for Image Classification using pre-trained VGG16 CNN

Objective

To understand and implement image classification using a pre-trained Convolutional Neural Network (CNN), specifically the VGG16 model.

1. Introduction

This Lab manual provides a step-by-step guide on using a pre-trained VGG16 convolutional neural network (CNN) for image classification. The VGG16 model, developed by Oxford's Visual Geometry Group, is a deep learning architecture widely used for image recognition tasks.

2. System Requirements

Before proceeding, ensure your system meets the following requirements:

Hardware:

- A computer with at least **8GB RAM** (16GB recommended)
- A **compatible GPU** (NVIDIA recommended) for faster processing (optional but preferred)

Software:

- Python (>=3.6)
- TensorFlow (>=2.0) or Keras (>=2.3)
- NumPy
- OpenCV (for image preprocessing, optional)
- Matplotlib (for visualization, optional)

3. Step-by-Step Guide

Step 1: Install Required Packages

To begin, install the necessary Python libraries by running the following command:

```
pip install tensorflow keras numpy opencv-python matplotlib
```

Step 2: Import Required Libraries

Once installed, import the essential libraries in your Python script:

```
import tensorflow as tf
```

```
from tensorflow.keras.applications import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input, decode_predictions
from tensorflow.keras.preprocessing import image
import numpy as np
import matplotlib.pyplot as plt
```

Step 3: Load the Pre-trained VGG16 Model

Load the pre-trained VGG16 model with ImageNet weights:

```
model = VGG16(weights='imagenet')
```

This loads the VGG16 model, pre-trained on the ImageNet dataset, making it ready for image classification tasks.

Step 4: Preprocess an Image

VGG16 requires input images to be **224x224 pixels** and properly preprocessed.

4.1 Load and Preprocess the Image

```
def load_and_preprocess_image(img_path):
    img = image.load_img(img_path, target_size=(224, 224))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = preprocess_input(img_array)
    return img, img_array
```

This function:

- Loads an image from the specified path.
- Resizes it to **224x224 pixels**.
- Converts it into a format compatible with the VGG16 model.
- Normalizes pixel values as required by VGG16.

Step 5: Perform Image Classification

Now, use the model to classify the image.

5.1 Define the Classification Function

```
def classify_image(img_path):
    img, img_array = load_and_preprocess_image(img_path)
    preds = model.predict(img_array)
    decoded_preds = decode_predictions(preds, top=3)[0]
    plt.imshow(img)
    plt.axis('off')
    plt.show()
    for i, (imagenet_id, label, score) in enumerate(decoded_preds):
        print(f"{i+1}: {label} ({score:.2f})")
```

This function:

- Loads and preprocesses the image.
- Feeds the image into the VGG16 model for prediction.
- Decodes and displays the top 3 predicted classes with confidence scores.
- Displays the image for reference.

5.2 Run the Classification Function

```
img_path = 'path_to_your_image.jpg'  # Change this to the path of your image
classify_image(img_path)
```

This line executes the classification function on the given image.

Step 6: Interpret Results

After running the script, you will see:

- The image displayed.
- The **top 3 predicted classes** with confidence scores.

Example output:

1: Labrador Retriever (0.85)

2: Golden Retriever (0.07)

3: Cocker Spaniel (0.03)

4. Troubleshooting and Additional Considerations

Troubleshooting

- **ModuleNotFoundError:** Ensure all required libraries are installed using pip install <library-name>.
- **GPU Issues:** If TensorFlow doesn't detect your GPU, verify that CUDA and cuDNN are installed properly.
- **Invalid Image Path:** Ensure the file path and format of your image are correct.

Additional Considerations

- **Batch Processing:** Modify the script to process multiple images at once.
- **Fine-Tuning:** For domain-specific tasks, fine-tune VGG16 using transfer learning.
- **Alternative Models:** Explore other pre-trained models like ResNet, Inception, or MobileNet for different use cases.

5. Conclusion

This step-by-step guide provides a straightforward approach to using a pre-trained VGG16 model for image classification. By following these instructions, users can efficiently classify images into meaningful categories.

For further customization, consider fine-tuning VGG16 on your dataset or exploring other pre-trained models.

What is VGG16: A Deep Learning Convolutional Neural Network

1. Introduction

VGG16 is a **deep convolutional neural network (CNN)** designed by the Visual Geometry Group (VGG) at the University of Oxford. It was introduced in the **2014 ImageNet Large Scale Visual Recognition Challenge (ILSVRC-2014)** and became one of the most influential models in deep learning for image classification.

---

2. Architecture of VGG16

VGG16 follows a **simple and uniform architecture** with **16 layers**, primarily composed of:

- **13 Convolutional Layers (Conv Layers)**
- **3 Fully Connected (FC) Layers**
- **Max Pooling Layers**
- **Softmax Layer for Classification**

Key Features:

- Uses **3×3 convolutional filters** throughout the network.
- Has **ReLU (Rectified Linear Unit) activation** in all layers.
- Uses **2×2 max pooling layers** to reduce spatial dimensions.
- Ends with **three fully connected layers** and a **softmax classifier**.

VGG16 Layer-wise Breakdown:

Layer Type	Number of Layers	Kernel Size	Activation
Convolutional	13	3×3	ReLU
Max Pooling	5	2×2	-
Fully Connected	3	-	ReLU
Output	1	-	Softmax

---

3. Why is VGG16 Important?

- **Deep but Simple:** Unlike other complex architectures, VGG16 maintains a uniform structure using **only 3×3 convolution filters**.

- **Pre-trained on ImageNet:** Trained on **millions of images** across **1,000 categories**, making it highly effective for **feature extraction and transfer learning**.
- **Widely Used in Computer Vision:** Its robustness and accuracy make it a **popular choice** in medical imaging, autonomous driving, and other AI applications.

---

4. Advantages of VGG16

✅ **High Accuracy:** Performs well on large-scale image classification tasks.
✅ **Transfer Learning:** Can be used as a feature extractor for custom datasets.
✅ **Well-Structured:** Uses small filter sizes, making it easier to analyze.

5. Disadvantages of VGG16

❌ **Large Model Size:** Requires **over 500MB of storage**.
❌ **Slow Training Speed:** Due to its **138 million parameters**, training is computationally expensive.
❌ **High Memory Usage:** Requires a **powerful GPU** for efficient processing.

---

6. Applications of VGG16

- **Image Classification** (e.g., detecting objects like cats, cars, planes)
- **Medical Image Analysis** (e.g., tumor detection in MRI scans)
- **Autonomous Vehicles** (e.g., identifying pedestrians, traffic signs)
- **Security and Surveillance** (e.g., face recognition)

---

7. How to Use Pre-trained VGG16 in Python

You can load and use the VGG16 model in **TensorFlow/Keras** as follows:

```
from tensorflow.keras.applications import VGG16

# Load pre-trained VGG16 model with ImageNet weights

model = VGG16(weights='imagenet')
```

For image classification:

```
from tensorflow.keras.preprocessing import image

from tensorflow.keras.applications.vgg16 import preprocess_input, decode_predictions
```

```
import numpy as np

# Load and preprocess an image
img_path = 'image.jpg'  # Replace with your image path
img = image.load_img(img_path, target_size=(224, 224))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array = preprocess_input(img_array)

# Predict using VGG16
predictions = model.predict(img_array)
decoded_preds = decode_predictions(predictions, top=3)[0]

# Display predictions
for i, (imagenet_id, label, score) in enumerate(decoded_preds):
    print(f"{i+1}: {label} ({score:.2f})")
```
