Lab Instructions: Image Classification with Caltech-101 Dataset and Explainability using Grad-CAM

Objective

To build, train, and evaluate a Convolutional Neural Network (CNN) for image classification using the Caltech-101 dataset. Students will also learn to apply Explainable AI (XAI) techniques, specifically Grad-CAM, to visualize model decision-making.

Prerequisites

- Basic knowledge of Python programming.
- Familiarity with machine learning and deep learning concepts.
- Understanding of Convolutional Neural Networks (CNNs).
- Basic knowledge of Explainable AI (XAI).

Part 1: Introduction to Caltech-101 Dataset

Dataset Overview

- Caltech-101 contains images from 101 object categories and 1 background category.
- Images per category range from 40 to 800, with a total of approximately 9,146 images.
- Images are diverse in size and can be resized for consistency.

Task

- Classify images into one of the 101 categories using a CNN.

- Visualize model decision-making using Grad-CAM.

Part 2: Dataset Loading and Preprocessing

Download Dataset

- Download the Caltech-101 dataset from Caltech-101 Dataset.

Load Dataset

Use PyTorch to load and preprocess the dataset. Example in PyTorch:

python

```
from torchvision import datasets, transforms

transform = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    transforms.RandomRotation(30),
    transforms.ColorJitter(brightness=0.4, contrast=0.4,
saturation=0.4, hue=0.1),
    transforms.RandomAffine(degrees=15, translate=(0.1, 0.1),
scale=(0.9, 1.1)),
    transforms.RandomGrayscale(p=0.2),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229,
0.224, 0.225])
])

dataset = datasets.ImageFolder(root='path_to_caltech101',
transform=transform)
```

Split Dataset

Split into training, validation, and test sets.

python

```
from torch.utils.data import random_split

train_size = int(0.8 * len(dataset))
val_size = int(0.1 * len(dataset))
test_size = len(dataset) - train_size - val_size

train_data, val_data, test_data = random_split(dataset, [train_size, 
val_size, test_size])
```

Data Loaders

Create data loaders for efficient batching.

python

```
from torch.utils.data import DataLoader

train_loader = DataLoader(train_data, batch_size=32, shuffle=True)
val_loader = DataLoader(val_data, batch_size=32)
test_loader = DataLoader(test_data, batch_size=32)
```

Part 3: Using Pre-trained Models

VGG19 Model

python

```
from torchvision.models import vgg19

model = vgg19(pretrained=True)
model.classifier[6] = nn.Linear(4096, 101)  # Adjust the final layer 
for 101 classes
```

ResNet50 Model

python

```
from torchvision.models import resnet50

model = resnet50(pretrained=True)
```

```
model.fc = nn.Linear(2048, 101)  # Adjust the final layer for 101
classes
```

EfficientNet (EfficientNet-B0)

python

```
from torchvision.models import efficientnet_b0

model = efficientnet_b0(pretrained=True)
model.classifier[1] = nn.Linear(1280, 101)  # Adjust the final layer
for 101 classes
```

Part 4: Training the Model

Define Loss and Optimizer

python

```
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

Train the Model

python

```
for epoch in range(10):
    model.train()
    for images, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

Validate the Model

python

```
model.eval()
with torch.no_grad():
    for images, labels in val_loader:
        outputs = model(images)
        # Compute validation metrics
```

Parameter Tuning with Grid Search

python

```
from sklearn.model_selection import ParameterGrid

param_grid = {
    'lr': [0.1, 0.01, 0.001],
    'batch_size': [16, 32, 64]
}

best_params = None
best_accuracy = 0

for params in ParameterGrid(param_grid):
    optimizer = optim.Adam(model.parameters(), lr=params['lr'])
    train_loader = DataLoader(train_data,
batch_size=params['batch_size'], shuffle=True)

    # Perform training and validation here
    # Compare and store the best parameters based on validation
accuracy
print(f"Best Params: {best_params}")
```

Part 5: Evaluating the Model

Evaluate on Test Data

python

```
model.eval()
```

```
with torch.no_grad():
    for images, labels in test_loader:
        outputs = model(images)
        # Compute test metrics
```

Confusion Matrix

python

```
from sklearn.metrics import confusion_matrix

y_pred = []
y_true = []

with torch.no_grad():
    for images, labels in test_loader:
        outputs = model(images)
        _, preds = torch.max(outputs, 1)
        y_pred.extend(preds.numpy())
        y_true.extend(labels.numpy())

cm = confusion_matrix(y_true, y_pred)
print(cm)
```

Classification Report

python

```
from sklearn.metrics import classification_report

print(classification_report(y_true, y_pred, target_names=class_names))
```

Top-k Accuracy

python

```
def top_k_accuracy(output, target, k=5):
    with torch.no_grad():
        max_k_preds = torch.topk(output, k, dim=1).indices
        correct = max_k_preds.eq(target.view(-1,
1).expand_as(max_k_preds))
        return correct.any(dim=1).float().mean().item()
```

Per-Class Accuracy

python

```python
per_class_accuracy = cm.diagonal() / cm.sum(axis=1)
for i, acc in enumerate(per_class_accuracy):
    print(f"Class {class_names[i]} Accuracy: {acc:.2f}")
```

t-SNE Visualization

python

```python
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

features = []
labels_list = []
model.eval()
with torch.no_grad():
    for images, labels in test_loader:
        output = model(images)
        features.append(output)
        labels_list.append(labels)

features = torch.cat(features).numpy()
labels_list = torch.cat(labels_list).numpy()

tsne = TSNE(n_components=2, random_state=42)
reduced_features = tsne.fit_transform(features)

plt.scatter(reduced_features[:, 0], reduced_features[:, 1],
c=labels_list, cmap='tab10')
plt.colorbar()
plt.show()
```

Part 6: Explainable AI (XAI) with Grad-CAM

Install Grad-CAM Library

bash

```
pip install grad-cam
```

Apply Grad-CAM

Visualize the regions of the image that contribute most to the predictions.

python

```
from pytorch_grad_cam import GradCAM
from pytorch_grad_cam.utils.image import show_cam_on_image

target_layer = model.layer4[-1]  # Adjust based on the model's
architecture
cam = GradCAM(model=model, target_layer=target_layer)

for images, labels in test_loader:
    grayscale_cam = cam(input_tensor=images,
target_category=labels[0])
    cam_image = show_cam_on_image(images[0].permute(1, 2, 0).numpy(),
grayscale_cam)
    plt.imshow(cam_image)
    plt.show()
```

The codes might not work directly, there will be some errors, which you need to fix it.

Submission Requirements

- Submit your Notebook using github.
- Include:
  - Final trained model.
  - Grad-CAM visualizations for at least 5 test images.
  - A brief report discussing:
    - Training and validation accuracy/loss.
    - Insights gained from Grad-CAM visualizations.
    - Challenges faced and solutions implemented.

Assessment Criteria

- Model Architecture: 30%
- Implementation and Code Quality: 30%
- Performance (Accuracy): 20%
- Explainability (Grad-CAM Visualizations): 20%

Good Luck!