East West University

Final Lab Report

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

Course Information,

Course Code:CSE366 Artificial Intelligence
Section-02

Submitted By

Disha Sikder Puja
2021-2-60-159

Submitted to

Dr Raihan Ul Islam(DRUI)
Associate Professor Department of Computer Science and Engineering
East-West University

Date of Submission: 31st january,2025

Objective

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

Introduction

This project's main goal is to use the Caltech-101 dataset to construct, train, and assess a Convolutional Neural Network (CNN) for picture categorization. Utilizing the ResNet50 architecture, optimizing it for the dataset, and evaluating its effectiveness using training and validation metrics are all part of this endeavor. This study focuses on using Explainable AI (XAI) tools, notably Grad-CAM, to display and comprehend the trained CNN's decision-making process in addition to model evaluation. The report describes the approach, findings, and lessons learned from this project, as well as the difficulties faced and how they were overcome.

Model Training & Performance

1.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.

2.Implementation Overview

- Google Collab was used for training with GPU acceleration.
- Images were resized, converted to tensors, and normalized.
- The dataset was split into training (80%), validation (10%), and testing (10%).

3.Validation & Testing Performance

- Validation accuracy 93.49%
- Test accuracy 92.75%

4.Training and Validation Accuracy/Loss

- The model achieved consistent performance improvements over epochs.
- Training loss decreased steadily, indicating effective learning.
- Validation accuracy remained stable, suggesting minimal overfitting

Key Code Snippets:-

Step 1: Setup Google Colab Environment

```
pip install --upgrade pip setuptools wheel
!pip install grad-cam
```

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

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms, models
from torch.utils.data import DataLoader, random_split
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix, classification_report
from pytorch_grad_cam import GradCAM
from pytorch_grad_cam.utils.image import show_cam_on_image
```

Step 2: Check GPU Availability

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")
```

Step 3: Download and Load Caltech-101 Dataset

```
from torchvision import transforms

# Define transformations
transform = transforms.Compose([
    transforms.Resize((224, 224)),  # Resize images to 224x224 (standard for pre-trained models)
    transforms.ToTensor(),          # Convert images to PyTorch tensors
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])  # Normalize pixel values
])
```

```
# Load the Caltech101 dataset
dataset = datasets.Caltech101(root="./data", download=True, transform=transform)
```

Step 4: Split Dataset into Training, Validation, and Test

```
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])

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)
```

Step 5: Load Pretrained ResNet50 Model

```
model = models.resnet50(pretrained=True)
model.fc = nn.Linear(2048, 101)  # Adjust final layer for 101 classes
model = model.to(device)
```

Step 6: Define Loss Function and Optimizer

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

Step 7: Train the Model

```
transform = transforms.Compose([
    transforms.Grayscale(num_output_channels=3),  # Convert grayscale to 3-channel RGB
    transforms.Resize((224, 224)),  # Resize images to 224x224
    transforms.ToTensor(),  # Convert images to PyTorch tensors
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])  # Normalize pixel values
])

dataset = datasets.Caltech101(root="./data", download=True, transform=transform)
train_loader = DataLoader(dataset, batch_size=32, shuffle=True)

epochs = 10
```

```
for epoch in range(epochs):
    model.train()
    running_loss = 0
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()

    print(f"Epoch {epoch+1}/{epochs}, Loss: {running_loss/len(train_loader):.4f}")
```

Step 8: Validate the Model

```
transform = transforms.Compose([
    transforms.Grayscale(num_output_channels=3),  # Convert grayscale to 3-channel RGB
    transforms.Resize((224, 224)),  # Resize images to 224x224
    transforms.ToTensor(),  # Convert images to PyTorch tensors
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])  # Normalize pixel values
])

dataset = datasets.Caltech101(root="./data", download=True, transform=transform)

# Split into train and validation sets
train_size = int(0.8 * len(dataset))
val_size = len(dataset) - train_size
train_dataset, val_dataset = random_split(dataset, [train_size, val_size])

# Create DataLoaders
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)

model.eval()
correct, total = 0, 0
with torch.no_grad():
    for images, labels in val_loader:
        images, labels = images.to(device), labels.to(device)
```

```
        outputs = model(images)
        _, preds = torch.max(outputs, 1)
        correct += (preds == labels).sum().item()
        total += labels.size(0)

print(f"Validation Accuracy: {100 * correct / total:.2f}%")
```

Validation Accuracy: 93.49%

Step 9: Evaluate on Test Data

```
transform = transforms.Compose([
    transforms.Grayscale(num_output_channels=3),  # Convert grayscale to 3-channel RGB
    transforms.Resize((224, 224)),  # Resize images to 224x224
    transforms.ToTensor(),  # Convert images to PyTorch tensors
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])  # Normalize pixel values
])

test_dataset = datasets.Caltech101(root="./data", download=True, transform=transform)

# Create the DataLoader for the test dataset
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

Step 10: Apply Grad-CAM for Explainability

```
import matplotlib.pyplot as plt
from pytorch_grad_cam import GradCAM
from pytorch_grad_cam.utils.image import show_cam_on_image
import numpy as np

# Adjust based on the model architecture (e.g., ResNet)
target_layer = model.layer4[-1]  # Choose the appropriate layer

# Initialize Grad-CAM with the target layer as a list
cam = GradCAM(model=model, target_layers=[target_layer])

# Iterate over the first 5 images in the test dataset
for i, (images, labels) in enumerate(test_loader):
    if i >= 5:  # Stop after 5 images
        break
```

```
    # Get one test image
    image = images[0].unsqueeze(0).to(device)

    # Generate Grad-CAM visualization
    grayscale_cam = cam(input_tensor=image)

    # Normalize and convert the mask to uint8
    grayscale_cam = np.uint8(255 * grayscale_cam[0])  # Scale the values between 0 and 255
    grayscale_cam = np.squeeze(grayscale_cam)  # Remove the extra dimension

    # Convert the image from tensor to numpy and prepare for visualization
    image_np = image.cpu().squeeze().permute(1, 2, 0).numpy()
    cam_image = show_cam_on_image(image_np, grayscale_cam)

    # Display the image with Grad-CAM overlay
    plt.imshow(cam_image)
    plt.title(f"Grad-CAM Visualization {i+1}")
    plt.axis("off")
    plt.show()

print("Grad-CAM visualizations for 5 test images displayed.")
```

Step 11: Save and Export the Model

```
from google.colab import drive
drive.mount('/content/drive')
```

Confusion Matrix

```
Confusion Matrix:
[[434   1   0 ...   0   0   0]
 [  6 427   0 ...   0   0   0]
 [  0   0 197 ...   0   0   0]
 ...
 [  0   0   0 ...  55   0   0]
 [  0   0   0 ...   0  35   1]
 [  0   0   0 ...   0   0  58]]
```

Classification report

86	0.96	0.94	0.95	86
87	0.97	0.95	0.96	59
88	1.00	0.97	0.98	64
89	0.94	0.97	0.96	35
90	1.00	0.98	0.99	85
91	0.98	0.98	0.98	49
92	0.95	1.00	0.97	86
93	0.98	0.85	0.91	75
94	0.99	1.00	0.99	239
95	0.70	1.00	0.82	37
96	0.73	0.98	0.84	59
97	0.80	0.94	0.86	34
98	1.00	0.98	0.99	56
99	1.00	0.90	0.95	39
100	0.98	0.97	0.97	60
accuracy			0.94	8677
macro avg	0.93	0.91	0.91	8677
weighted avg	0.95	0.94	0.94	8677

Explainability with Grad-CAM

1.Overview of Grad-CAM

Grad-CAM was applied to generate heat maps that highlight important regions influencing the model's predictions. This helps in understanding the model's decision-making process.

2.Visualization & Insights

- Grad-CAM highlighted salient image region**s** critical to predictions.
- Correctly classified images showed heatmaps focused on relevant object features.
- Misclassified images often had heatmaps centered on irrelevant background details, indicating model biases or dataset limitations.

Challenges

Class Imbalance : Some categories had significantly fewer images
Misclassifications: Certain objects with similar shapes and color were confused.
Computational Constraints: Training deep networks require high computational power.

Conclusion

This model works well with achieving high accuracy 92.75%.Grad-CAM effectively provides interpretability,highlighting key image regions.Some misclassifications suggest the need for further fine-tuning and augmentation.