EX 5 COMPARATIVE ANALYSIS OF VGG, RESNET, AND GOOGLENET ON IMAGE CLASSIFICATION

DATE: 18/09/2025

Problem Statement:

Implement and compare the performance of three popular CNN architectures: VGG, ResNet, and GoogLeNet for image classification. Use a labeled dataset to train each model and evaluate their convergence and accuracy.

Suggested Dataset: Dogs vs. Cats dataset

Objectives:

1. Understand the architectural differences between VGG, ResNet, and GoogLeNet.
2. Train all three models on the same dataset using transfer learning.
3. Analyze and compare performance using validation accuracy.
4. Apply the trained models to predict classes for custom images.

Scope:

This experiment demonstrates the power of transfer learning using pre-trained CNN models. Students explore how architectural changes affect accuracy and generalization. Comparing models under identical training settings aids in choosing the right model for real-world applications.

Tools and Libraries Used:

1. Python 3.x
2. PyTorch
3. torchvision
4. Matplotlib
5. PIL (Python Imaging Library)

Implementation Steps:

Step 1: Import Necessary Libraries

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, models, transforms
from torch.utils.data import DataLoader, random_split, Subset
import matplotlib.pyplot as plt
from PIL import Image
import os
```

Step 2: Configure Device and Define Labels

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

class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
        'dog', 'frog', 'horse', 'ship', 'truck']
```

Step 3: Preprocess Data and Load CIFAR-10

```
transform = transforms.Compose([
  transforms.Resize((224, 224)),
  transforms.ToTensor(),
  transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

dataset = datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
test_dataset = datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)

dataset = Subset(dataset, range(500))
train_size = int(0.8 * len(dataset))
val_size = len(dataset) - train_size
train_dataset, val_dataset = random_split(dataset, [train_size, val_size])

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=64, shuffle=False)
```

Step 4: Define Training and Evaluation Function

```
def train_and_evaluate(model, name, num_epochs=5):
  model = model.to(device)
  criterion = nn.CrossEntropyLoss()
  optimizer = optim.Adam(model.parameters(), lr=1e-4)

  train_accs, val_accs = [], []

  for epoch in range(num_epochs):
    model.train()
    correct, total = 0, 0
    for inputs, labels in train_loader:
      inputs, labels = inputs.to(device), labels.to(device)

      optimizer.zero_grad()
      outputs = model(inputs)
      loss = criterion(outputs, labels)
      loss.backward()
      optimizer.step()
```

```
            _, predicted = torch.max(outputs, 1)
            correct += (predicted == labels).sum().item()
            total += labels.size(0)
        train_accs.append(100 * correct / total)

        model.eval()
        correct, total = 0, 0
        with torch.no_grad():
            for inputs, labels in val_loader:
                inputs, labels = inputs.to(device), labels.to(device)
                outputs = model(inputs)
                _, predicted = outputs.max(1)
                correct += (predicted == labels).sum().item()
                total += labels.size(0)
        val_accs.append(100 * correct / total)

        print(f"{name} Epoch {epoch+1}/{num_epochs} - Train Acc: {train_accs[-1]:.2f}%, Val
Acc: {val_accs[-1]:.2f}%")

    return model, train_accs, val_accs
```

Step 5: Select Pretrained Models and Replace Final Layers

```
def get_model(name):
    if name == "vgg":
        model = models.vgg16(pretrained=True)
        model.classifier[6] = nn.Linear(4096, 10)
    elif name == "resnet":
        model = models.resnet18(pretrained=True)
        model.fc = nn.Linear(model.fc.in_features, 10)
    elif name == "googlenet":
        model = models.googlenet(pretrained=True, aux_logits=True)
        model.fc = nn.Linear(model.fc.in_features, 10)
    else:
        raise ValueError("Unknown model")
    return model
```

Step 6: Train All Models and Collect Results

```
results = {}
trained_models = {}

for model_name in ["vgg", "resnet", "googlenet"]:
    print(f"\n◆ Training {model_name.upper()} on CIFAR-10...")
    model = get_model(model_name)
    trained_model, train_acc, val_acc = train_and_evaluate(model, model_name.upper(),
num_epochs=5)
    results[model_name] = (train_acc, val_acc)
    trained_models[model_name] = trained_model
```

Step 7: Plot Accuracy Comparison

```
plt.figure(figsize=(10, 6))
for name, (train_acc, val_acc) in results.items():
    plt.plot(val_acc, label=f'{name.upper()} Val Acc')
plt.title('Validation Accuracy Comparison on CIFAR-10')
plt.xlabel('Epoch')
plt.ylabel('Accuracy (%)')
plt.legend()
plt.grid(True)
plt.show()
```

Step 8: Predict on Custom Image

```
def predict_image(image_path, models_dict):
    image = Image.open(image_path).convert('RGB')
    image = transform(image).unsqueeze(0).to(device)

    print(f"\n🖼 Prediction results for image: {image_path}")
    for model_name, model in models_dict.items():
        model.eval()
        with torch.no_grad():
            outputs = model(image)
            _, predicted = outputs.max(1)
            pred_class = class_names[predicted.item()]
            print(f"{model_name.upper():<10} => {pred_class}")

custom_image_path = "download.jpeg"
if os.path.exists(custom_image_path):
    predict_image(custom_image_path, trained_models)
else:
    print(f"\n! Image not found: {custom_image_path}. Please add an image to test.")
```

Output: