Experiment 14 Implement a Pre Trained CNN Model as a feature extractor using transfer learning
27/10/25

Aim: To perform image classification on CIFAR-10 dataset using Transfer learning with a pretrained ResNet 18 model.

Objectives

To understand and implement transfer learning using a pretrained ResNet 18 model.

To train the modified model and visualize the loss curve

Pseudo code.

Begin

Import torch, torchvision, torch.nn

Set device

load pre-trained ResNet18 model:

Freeze all layers:

For each parameter in model
    set requires_grad = False.

Modify the final layer for CIFAR-10:

    trainset ← CIFAR10(train=True)

Define transformation:

    transform ← Resize(224×224)+ToTensor(),

Observation

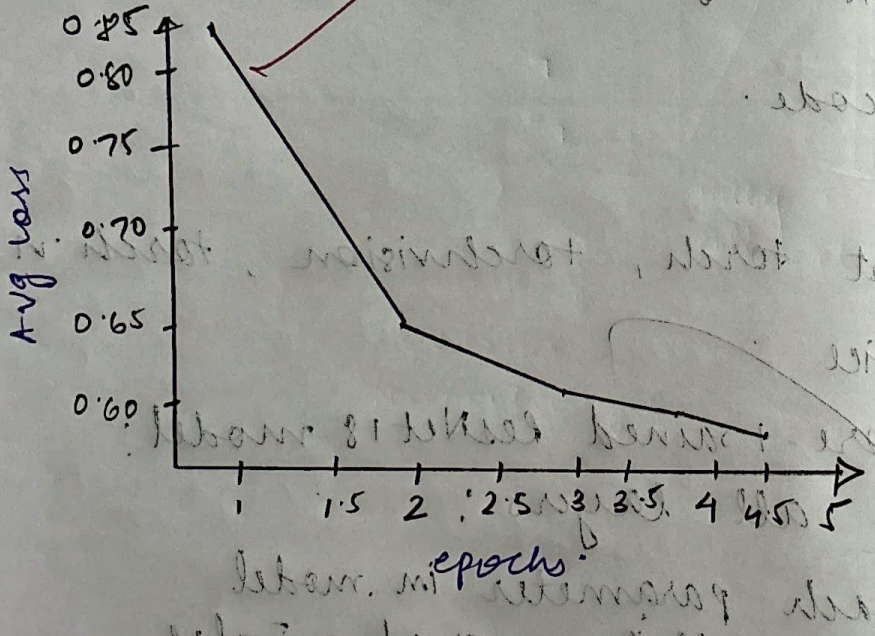
epoch 1: loss : 0.8348

epoch 2: loss : 0.6289

epoch 3: loss : 0.5937

epoch 4: loss : 0.5997

epoch 5: loss : 0.5699



Loss Curve

define loss function and optimizer:

criterion ← CrossEntropyLoss()

for each epoch in range(1 to 10):

avg loss ← running loss / total-batches

Append avg-loss to loss_values

Print epoch number and avg loss

print "Training Complete"

Plots loss curve:

X-axis ← epochs

Y-axis ← average loss

End.

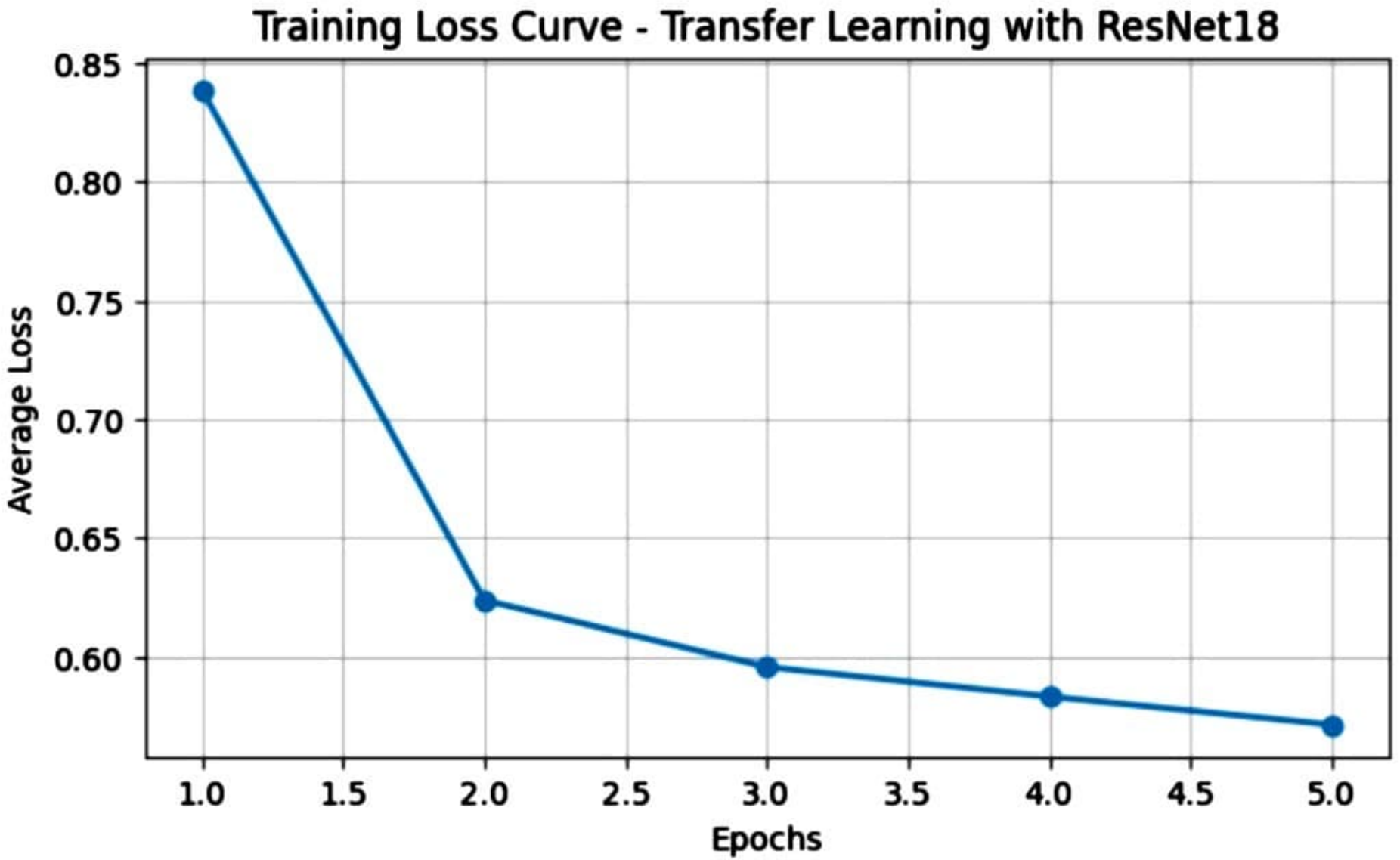
Result: Successfully implemented a Pretrained CNN model.

```
import matplotlib.pyplot as plt

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

# Load pre-trained ResNet18
model = models.resnet18(pretrained=True)

# Freeze all convolutional layers to use as feature extractor
for param in model.parameters():
    param.requires_grad = False

# Replace the final layer to match CIFAR-10 (10 classes)
model.fc = nn.Linear(model.fc.in_features, 10)
model = model.to(device)

# Data transformations
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor()
])

# Load dataset (CIFAR-10)
trainset = CIFAR10(root='./data', train=True, download=True, transform=transform)
trainloader = DataLoader(trainset, batch_size=64, shuffle=True)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.fc.parameters(), lr=0.001)

# Lists to store loss values for graph
loss_values = []

# Training for a few epochs
```

```
# Lists to store loss values for graph
loss_values = []

# Training for a few epochs
epochs = 5
for epoch in range(epochs):
    running_loss = 0.0
    for images, labels in trainloader:
        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()

    avg_loss = running_loss / len(trainloader)
    loss_values.append(avg_loss)
    print(f"Epoch [{epoch+1}/{epochs}], Loss: {avg_loss:.4f}")

print("\n✅ Training complete using Pre-trained ResNet18 as Feature Extractor.\n")

# Plot the training loss graph
plt.figure(figsize=(7, 4))
plt.plot(range(1, epochs + 1), loss_values, marker='o', linestyle='-', linewidth=2)
plt.title("Training Loss Curve - Transfer Learning with ResNet18")
plt.xlabel("Epochs")
plt.ylabel("Average Loss")
plt.grid(True)
plt.show()
```

```
Epoch [1/5], Loss: 0.8388
Epoch [2/5], Loss: 0.6238
Epoch [3/5], Loss: 0.5957
Epoch [4/5], Loss: 0.5832
Epoch [5/5], Loss: 0.5713
```

✅ Training complete using Pre-trained ResNet18 as Feature Extractor.



Experiment 15 — Implement a YOLO model to detect objects
27/10/25

Aim: Demonstrate how to perform object detection on sample using a pre-trained CNN model

Objective

→ Loading and understanding the YOLO model that was trained on COCO dataset

Pseudocode

→ Import necessary libraries YOLO from ultralytics

→ SETModel = load_yolo_model("yolov8n.pt")

→ Load the image as a URL form

→ Run prediction

SET confidence_threshold = 0.7

SET results_list = model(image_pil, conf = confidence_threshold)

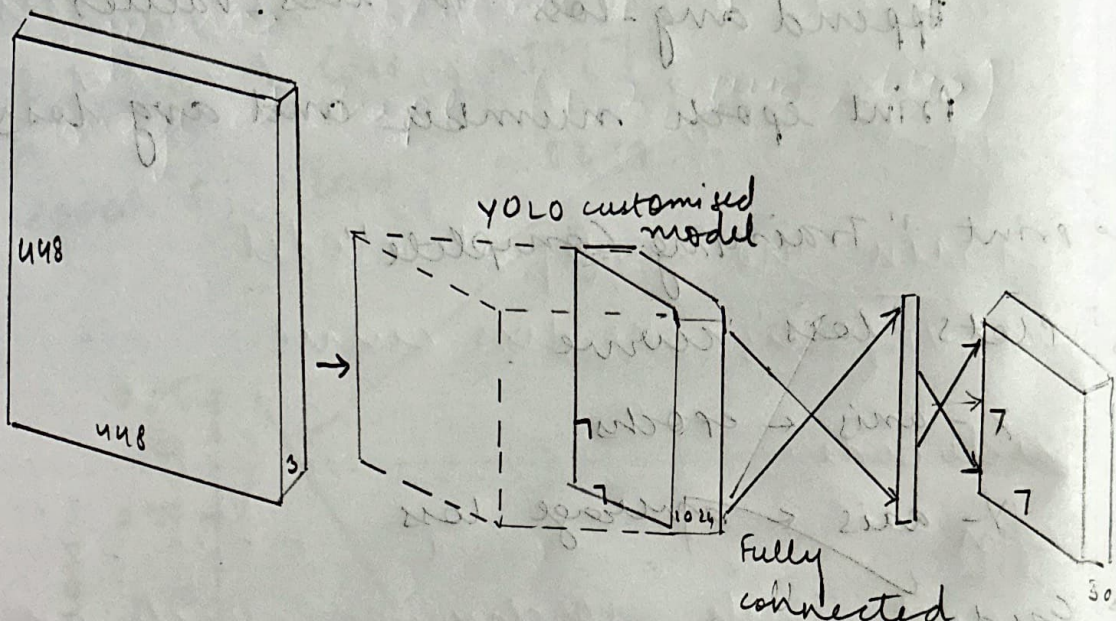
→ SET result = results_list[0]

→ Draw detection

// Use built-in plot() method to draw boxes and labels

// return a numpy array in BGR format

YOLO Model Architecture



consists of

Conv layer

Maxpool layer

output

448×640 1 car, 322.2ms
Speed: 34.0ms preprocess, 322.2ms inference, 32.6ms postprocess per image at shape (1, 3, 448, 640)

Convert from BGR to RGB from matplotlib

Initialize plot and display plot.

End.

Result: Successfully detected images using a YOLO model

Eg. 1

```python
# 1. Install dependencies
# We only need 'ultralytics' (which includes torch) and 'matplotlib'
!pip install ultralytics matplotlib --quiet

# 2. Import libraries
from ultralytics import YOLO
from PIL import Image
import matplotlib.pyplot as plt
import requests
import io # Needed for handling image from requests

# 3. Load pretrained YOLO model
# We'll use YOLOv8n (nano), which is fast and COCO-pretrained.
model = YOLO('yolov8n.pt')

# 4. Load a sample image from URL
url = "https://images.unsplash.com/photo-1503376780353-7e6692767b70" # has cars
img = None
try:
    # Get the image data
    response = requests.get(url)
    response.raise_for_status() # Raise an error for bad responses

    # Open the image from the in-memory bytes
    img = Image.open(io.BytesIO(response.content)).convert("RGB")

except Exception as e:
    print(f"Could not load image from URL. Error: {e}")
    # Create a dummy image if loading fails
    img = Image.new('RGB', (800, 600), color='gray')

# 5. Run Prediction
# YOLO handles all preprocessing (transforms, ToTensor) automatically.
# We set 'conf=0.7' to match the original code's confidence threshold.
```

```
Creating new Ultralytics Settings v0.0.6 file ✅
View Ultralytics Settings with 'yolo settings' or at '/root/.config/Ultralytics/settings.json'
Update Settings with 'yolo settings key=value', i.e. 'yolo settings runs_dir=path/to/dir'. For help see http
s://docs.ultralytics.com/quickstart/#ultralytics-settings.
Downloading https://github.com/ultralytics/assets/releases/download/v8.3.0/yolov8n.pt to 'yolov8n.pt': 100% ━
━━━━━━━━━━━━ 6.2MB 79.5MB/s 0.1s

0: 448x640 1 car, 322.2ms
Speed: 34.0ms preprocess, 322.2ms inference, 32.6ms postprocess per image at shape (1, 3, 448, 640)
```

YOLOv8 Object Detection (COCO Pretrained)