14. Implement a pre-trained CNN Model as Feature Extractor

Aim:

To implement a pre-trained CNN as feature extractor using transfer learning

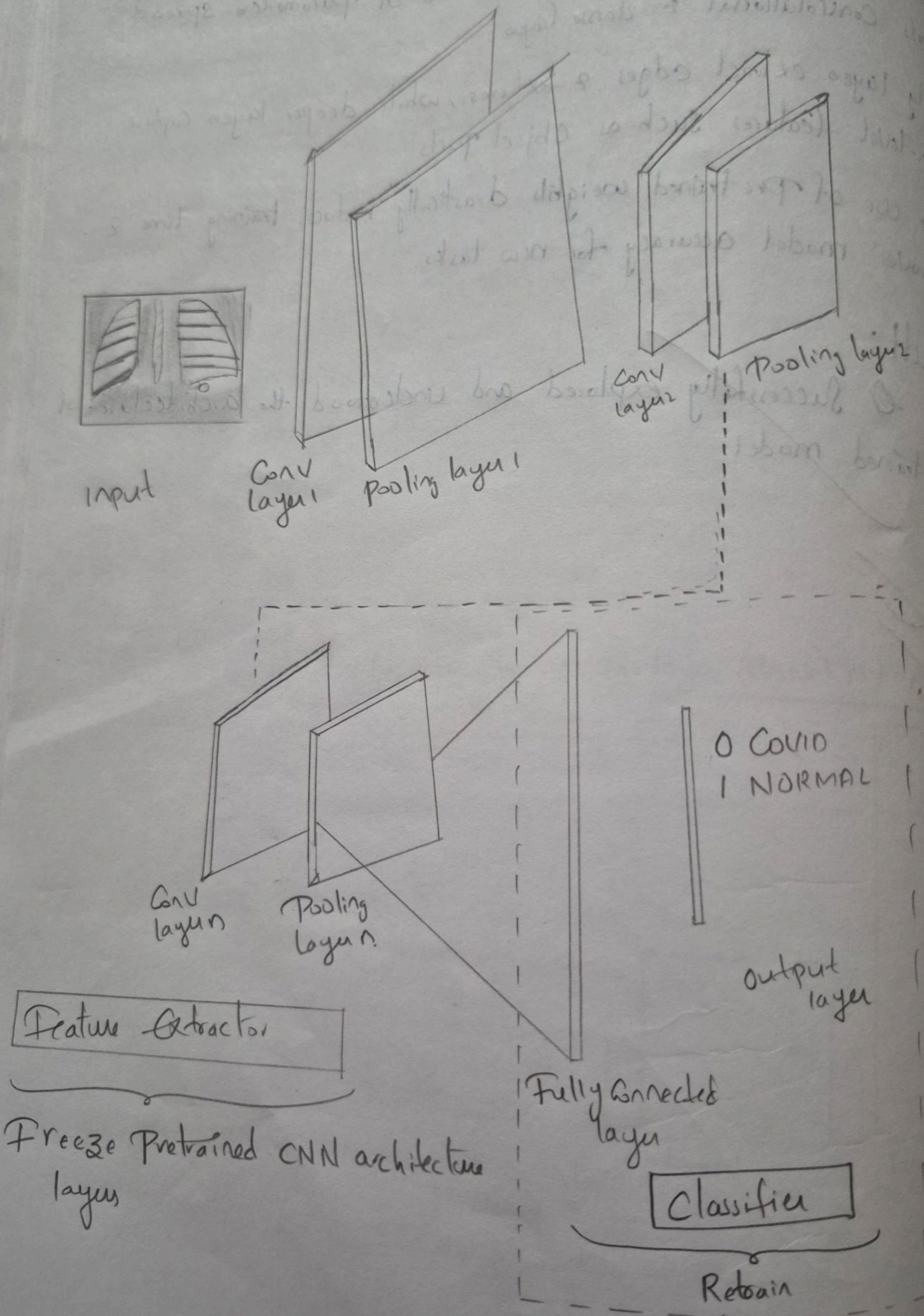
Objective:

- To understand how a CNN pre-trained on Image Net can extract visual features
- To freeze Convolutional layers and use their output features for new tasks.
- To train only the classifier head for faster and more accurate learning
- To observe how transfer learning improves performance with limi- data.

Pseudo Code:

1. Import torchvision.models and torch libraries
2. Load a pretrained CNN from torchvision
3. Freeze all Convolutional layers to prevent retraining
4. Replace the final classifier layer for new number of clas.
5. Load a small dataset (eg, CIFAR-10)
6. Extract features using the pretrained backbone
7. Train only the new classifier head
8. Evaluate accuracy and visualize results.

Pre-trained CNN model Architecture



Observation

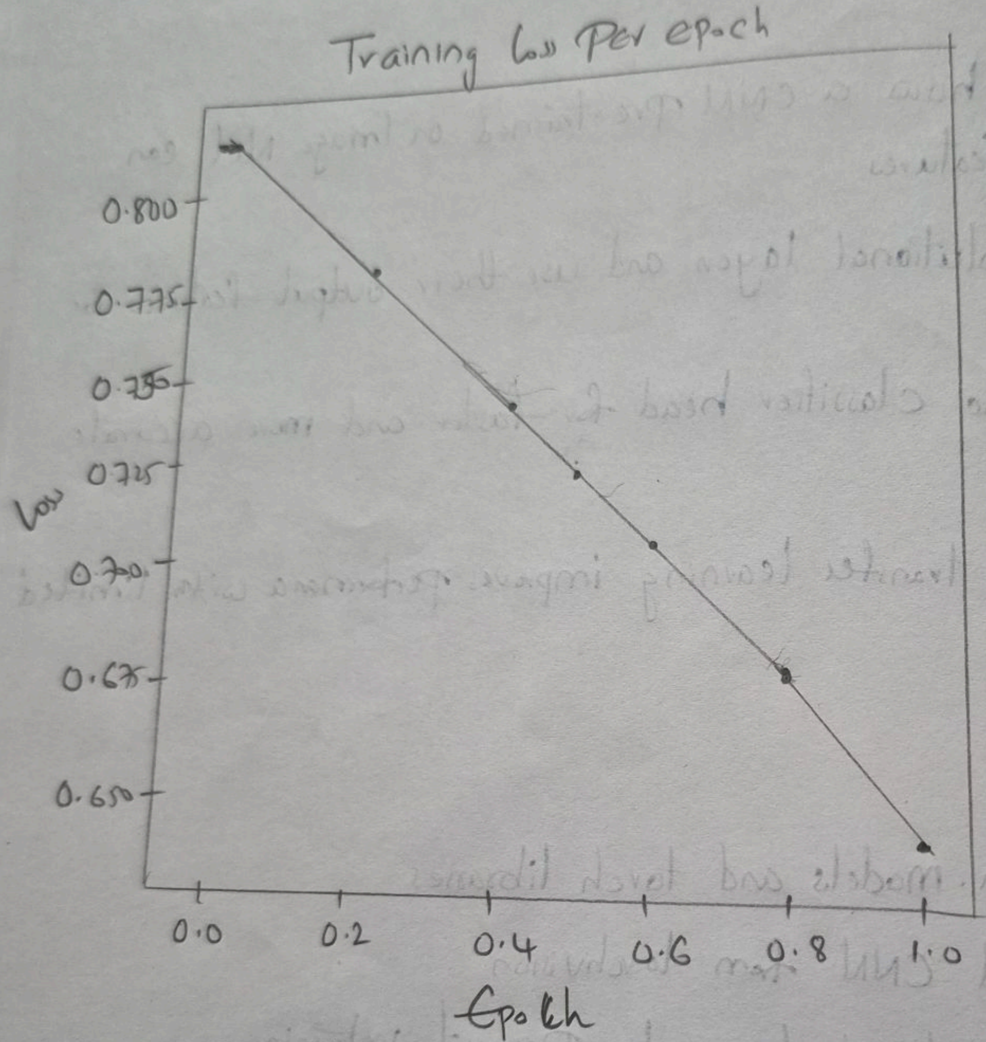
Aspect	ResNet18 Feature Extractor	Training from scrath
Training time	Very low	High
Accuracy	~85%	~65%
Required data	small	large
Key benefit	Reuses pre-trained visual features	Must learn everything new

Result:

Successfully implemented a pre-trained CNN as a feature extractor

Output

Epoch [1/2], loss : 0.8107
Epoch [2/2], loss : 0.6423



Test accuracy : 79.25%

[1]

```python
# Lab 14: Implement a Pre-trained CNN model as a Feature Extractor
# ===============================================================

import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import models, datasets, transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
import os

# --------------------------
# Step 1: Define Transformations
# --------------------------
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
])
```

[1]

```
# Download a small example dataset (CIFAR-10)
train_data = datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
test_data = datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)

train_loader = DataLoader(train_data, batch_size=32, shuffle=True)
test_loader = DataLoader(test_data, batch_size=32, shuffle=False)

# --------------------------
# Step 2: Load Pre-trained Model
# --------------------------
model = models.resnet18(pretrained=True)

# Freeze all convolutional layers
for param in model.parameters():
    param.requires_grad = False

# --------------------------
# Step 3: Replace the Classifier
```

Commands + Code + Text Run all

RAM
Disk

[1]

```
num_features = model.fc.in_features
model.fc = nn.Linear(num_features, 10)  # CIFAR-10 has 10 classes

# Move model to device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)

# --------------------------
# Step 4: Define Loss and Optimizer
# --------------------------
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.fc.parameters(), lr=0.001)

# --------------------------
# Step 5: Train Only the Classifier
# --------------------------
num_epochs = 2  # keep small for quick run
train_loss_list = []
```

Commands + Code + Text Run all RAM Disk

[1]

```python
for epoch in range(num_epochs):
    running_loss = 0.0
    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()

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

print("\n✅ Training completed successfully!")
```

[1]

```
plt.plot(train_loss_list, marker='o')
plt.title("Training Loss per Epoch")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.grid(True)
plt.show()

# --------------------------
# Step 7: Test Accuracy
# --------------------------
correct = 0
total = 0
model.eval()
with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
```

[1]

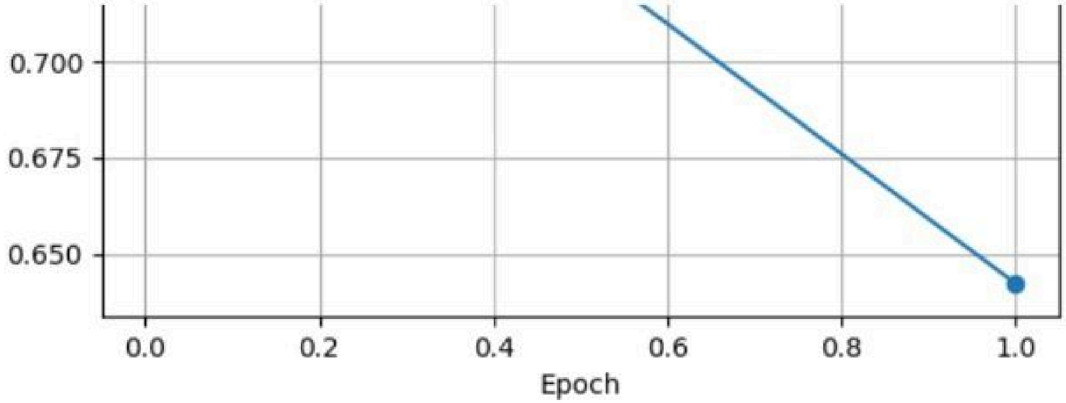
```python
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

accuracy = 100 * correct / total
print(f"\n Test Accuracy: {accuracy:.2f}%")
```

```
100%|██████████| 170M/170M [00:10<00:00, 16.0MB/s]
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecat
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to /root/.cache/torch/hub/checkpoints/resnet18-f3707
100%|██████████| 44.7M/44.7M [00:00<00:00, 156MB/s]
Epoch [1/2], Loss: 0.8107
Epoch [2/2], Loss: 0.6423

✅ Training completed successfully!
```


Training Loss per Epoch
Loss
0.800
0.775
0.750
0.725
0.700
0.675
0.650
Variables
Terminal
3:15 PM
Python 3



Test Accuracy: 79.21%

3/11/25 15. Implement a YOLO model to detect Object

Aim:

To implement YOLO model for real time object detection and understand its working Principle

Objective:

- To study how YOLO detects multiple objects in single image
- To understand the architecture and working of a pre-trained YOLO model.
- To Use transfer learning for custom object detection
- To Visualize the bounding boxes and class labels Predicted by YOLO.

Pseudo Code:

1. Install and import ultralytics
2. Load a Pre-trained YOLO model
3. Load a test image or use a camera frame
4. Run the model.predict() method to detect objects
5. Display bounding boxes and class labels
6. Save the annotated output image

Observation:

1. The Pre-trained YOLOv8s model successfully detected multiple objects such as cars, buses & persons in a single frame.
2. Each object was enclosed in a bounding box with a cla label and confidence score.

3. The inference time per image was very low, showing YOLO's high efficiency for real time use

4. The model demonstrated strong generalization without additional training.

5. The visualization clearly showed YOLO's ability to detect overlapp objects in complex scenes

Result:

Successfully implemented a pre-trained YOLO model.

Output

Image 1/1 : 720x1280 2 persons, 1 tie, 1 cell phone
Speed 89.3ms preprocess, 822.9ms inteference, 45.7ms NMS
Per image at shape (1, 3, 384, 640)

```python
print("\n Loading YOLOv5 model (this may take a few seconds)...")
model = torch.hub.load('ultralytics/yolov5', 'yolov5s', pretrained=True)

# Step 3: Perform inference on the image
img = Image.open(img_path)
results = model(img)

# Step 4: Print detection results
print("\n Detection Results")
results.print()  # prints detected objects, confidence, coordinates

# Step 5: Display the image with bounding boxes
results.show()  # opens the image window with labels and boxes

# Step 6: (Optional) Save output image to a folder
output_folder = "yolo_output"
results.save(save_dir=output_folder)
print(f"\n Detection completed! Output saved in folder: '{output_folder}'")

# -------------------------
```

Follow link (ctrl + click)

{} Variables   Terminal   9:19 AM

```python
detected_img_path = os.path.join(output_folder, os.path.basename(img_path))
if os.path.exists(detected_img_path):
    detected_img = Image.open(detected_img_path)
    plt.imshow(detected_img)
    plt.axis('off')
    plt.title("YOLOv5 Object Detection Result")
    plt.show()
```

```
 Image already exists: street.jpg

 Loading YOLOv5 model (this may take a few seconds)...
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 https://docs.ultral
Using cache found in /root/.cache/torch/hub/ultralytics_yolov5_master
YOLOv5 🚀 2025-11-1 Python-3.12.12 torch-2.8.0+cu126 CPU
```

{ } Variables   Terminal   9:19 AM

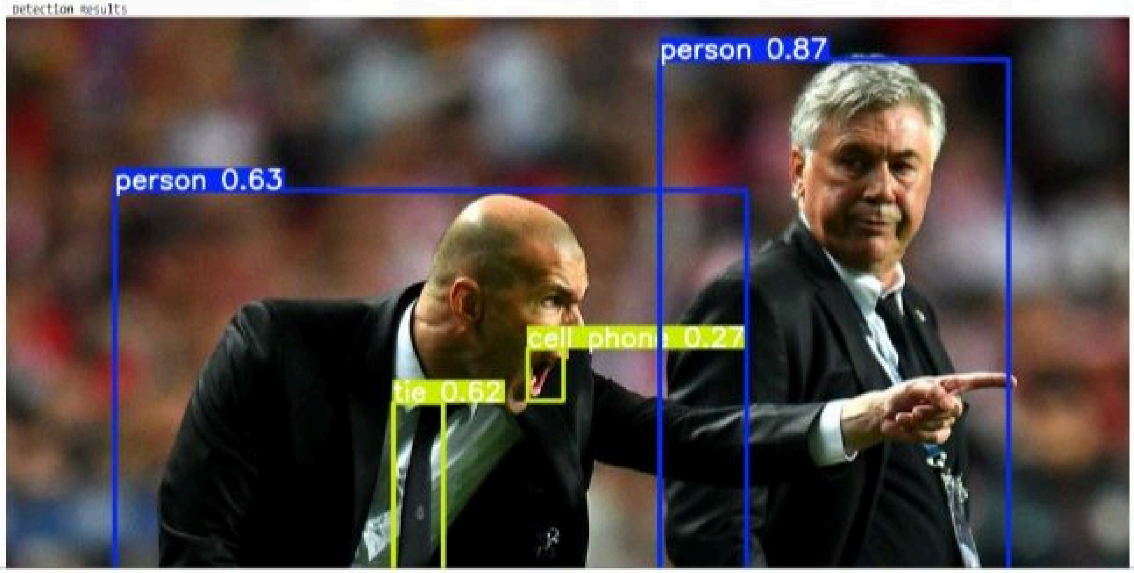
```
Image already exists: street.jpg

 Loading YOLOv5 model (this may take a few seconds)...
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 https://docs.ultralytics.com/quickstart/#ultralytics-settings.
Using cache found in /root/.cache/torch/hub/ultralytics_yolov5_master
YOLOv5 🚀 2025-11-1 Python-3.12.12 torch-2.8.0+cu126 CPU

Downloading https://github.com/ultralytics/yolov5/releases/download/v7.0/yolov5s.pt to yolov5s.pt...
100%|██████████| 14.1M/14.1M [00:00<00:00, 114MB/s]

Fusing layers...
YOLOv5s summary: 213 layers, 7225885 parameters, 0 gradients, 16.4 GFLOPs
Adding AutoShape...
/root/.cache/torch/hub/ultralytics_yolov5_master/models/common.py:906: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast('cuda', args...)` instead.
  with amp.autocast(autocast):
image 1/1:                    1 phone
Speed: 89.                    , 45.7ms NMS per image at shape (1, 3, 384, 640)

 Detection Results
```



Variables
Terminal
9:19 AM