

as a feature Extractor using Transfer learning

## EXP NO: 14:- Implement a pre-Trained CNN model

Aim:-

To implement a pre-trained CNN model as a feature extractor for image classification using transfer learning.

Objectives:-

1. To use a pre-trained model trained on Imagenet as a fixed feature extractor.
2. To extract deep visual features from images for classification tasks.
3. To attach and train a new classifier on top of the extracted features.
4. To reduce training time & improve accuracy using transfer learning.

Algorithm:-

1. Import dataset (CIFAR-10 or custom dataset).
2. Load a pre-trained model (VGG16 or ResNet50) without top (fully connected) layers.
3. Freeze pre-trained model weights to prevent retraining.
4. Add custom layers (Dense + softmax) for classification.
5. Compile & train the new model using extracted features.
6. Evaluate accuracy on test images.

## Pseudo Code:-

```
Load dataset (train, test)  
preprocess images to target size  
Load pre-trained CNN (include_top=False,  
weights='imagenet')  
Freeze pre-trained layers  
Add custom layers:  
    flatten → Dense (256, ReLU) → Dense(num-  
    classes, softmax)  
Compile model (Adam, categorical_crossentropy)  
Train model on dataset  
Evaluate accuracy
```

## Observations

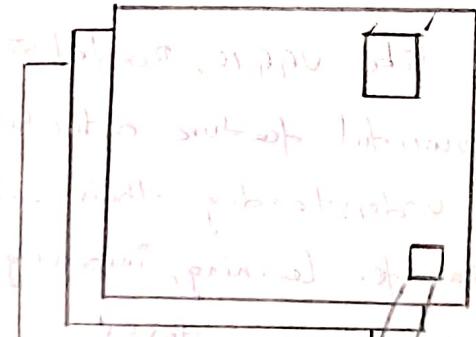
- The pre-trained VGG16 network extracts robust visual features from images.
- Training time is greatly reduced compared to training from scratch
- The model achieves high accuracy with limited epochs.

## Conclusion:-

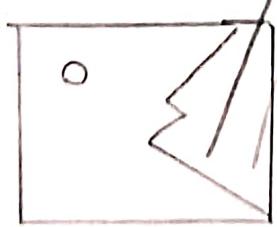
Using a pre-trained CNN as a feature extractor provides efficient & accurate image classification. Transfer learning leverages previously learned features, significantly improving performance & saving computation time.

# TRANSFER LEARNING

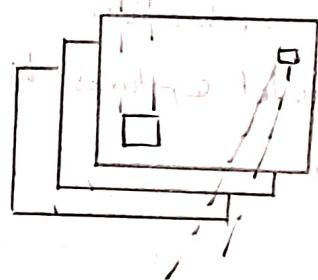
Convolutional  
layer



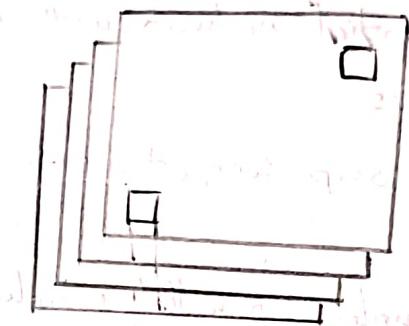
Pre-trained model



Pooling  
layer



Convolutional  
layer



fully connected

Output  
predictions



Deep learning  
NN

Machine learning  
algorithms

Image classification task

Output:-

Epoch [1/5], Loss : 0.8181

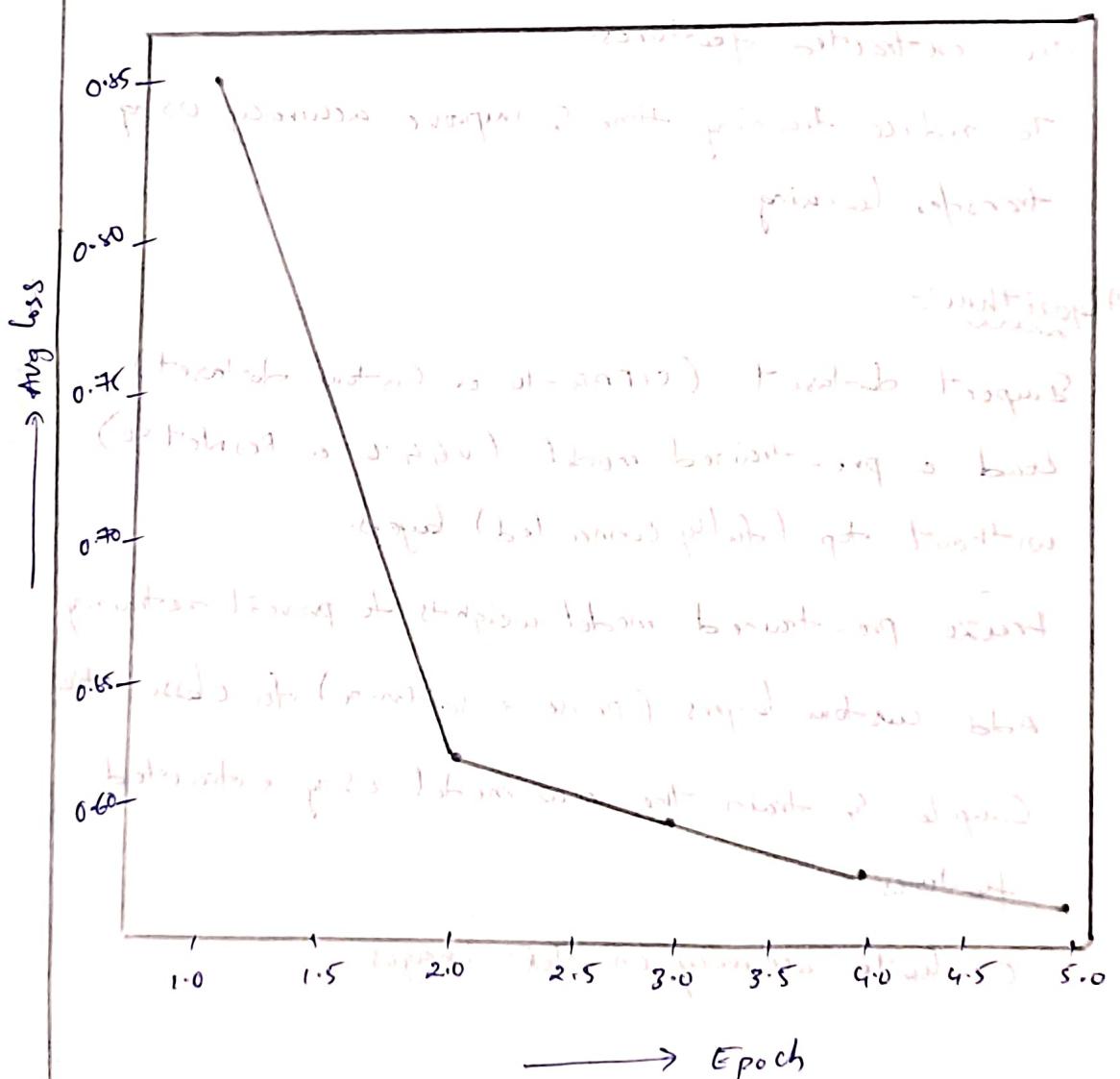
Epoch [2/5], Loss : 0.6266

Epoch [3/5], Loss : 0.5937

Epoch [4/5], Loss : 0.5797

Epoch [5/5], Loss : 0.5677

Training completed using pre-trained ResNet18 as feature extractor and soft loss function.





dltlab14.ipynb



File Edit View Insert Runtime Tools Help

Commands + Code + Text | ▶ Run all ▾



```
[ ] # lab14_transfer_learning_feature_extractor_with_graph.py
import torch
import torch.nn as nn
import torchvision.models as models
import torchvision.transforms as transforms
from torchvision.datasets import CIFAR10
from torch.utils.data import DataLoader
import torch.optim as optim
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
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()
```

```
Using device: cuda
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to /root/.cache/torch/hub/checkpoints/resnet18-f37072fd.pth
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be removed in the future, please use 'weights' instead.
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or 'None' for 'weights' are deprecated since 0.13 and may be removed in the future. The current value is 'ResNet18_Weights.IMAGENET1K_V1'.
  warnings.warn(msg)
100% [██████████] 44.7M/44.7M [00:00<00:00, 199MB/s]
100% [██████████] 170M/170M [00:03<00:00, 43.5MB/s]
Epoch [1/5], Loss: 0.8481
Epoch [2/5], Loss: 0.6266
Epoch [3/5], Loss: 0.5937
Epoch [4/5], Loss: 0.5797
Epoch [5/5], Loss: 0.5699
 Training complete using Pre-trained ResNet18 as Feature Extractor.
```

