Exp 13:

Understanding the Architecture of a Pre-trained Model

Aim : To study & understand the architecture of a pre-trained CNN model. ~~such as VGG~~

Pseudo code:

Step 1: Import necessary libraries.
import torchvision.models as models.

Step 2: Load a pre-trained model (eg: ResNet18)
model = models.resnet18(pretrained = ~~False~~ True

Step 3:
Display model architecture
print (model)

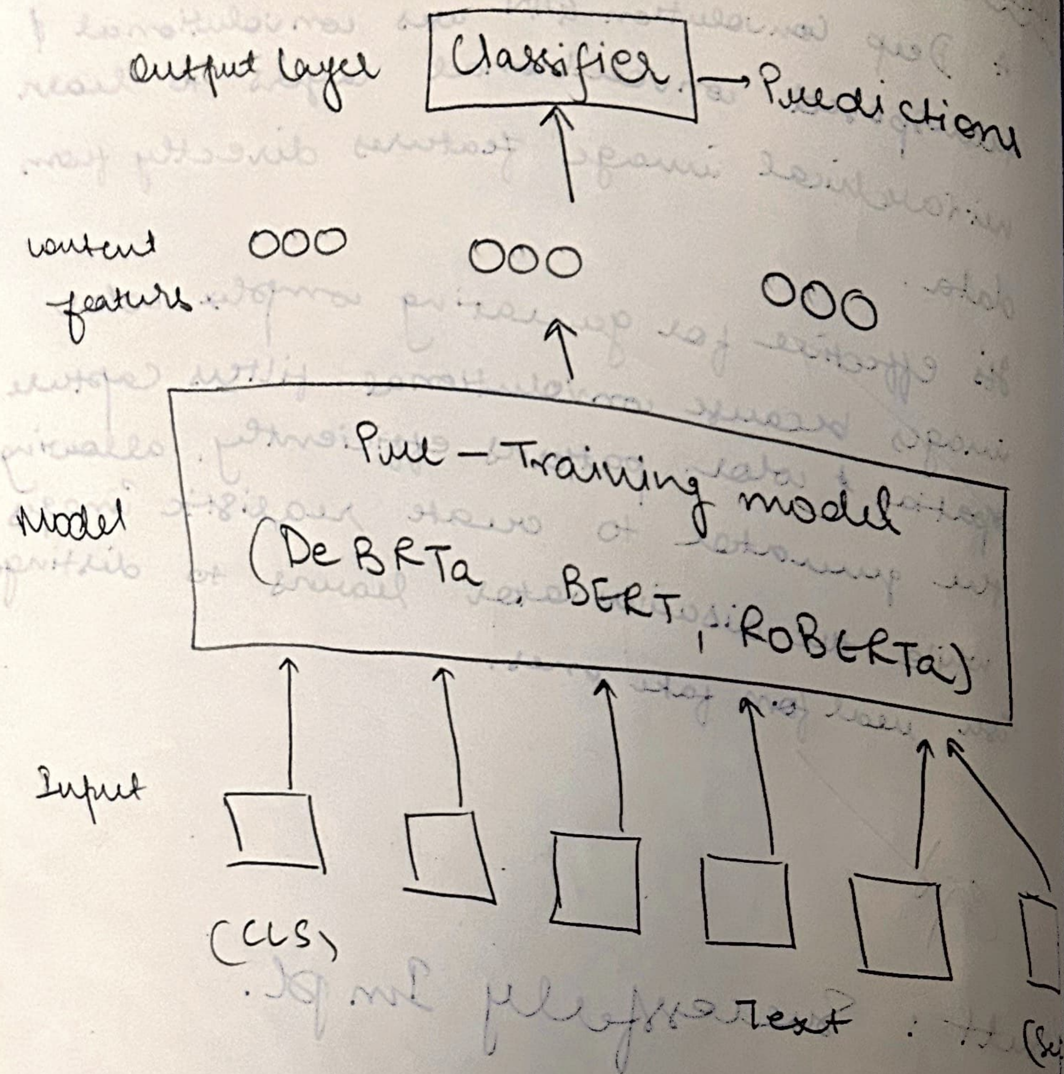
Step 4: Freeze all model parameters to prevent training.

for param in model.parameters():
    param.requires_grad = False.

Step 5: Examine features extraction layers
~~print (model. layer 1)~~
print (model. layer 4)

Step 6: Observe how the input passes through the network

# forward pass example (optional)
output = model (input_image)


Output layer
Classifier
Predictions
content features
Pre-Training model
(DeBRTa, BERT, RoBERTa)
Model
Input
(CLS)
Text

Justification:

→ Pre trained models are neural networks trained on large datasets

→ These models have already learned rich & general image features like edges, textures & shapes.

Result: Program implemented successfully.

	Top 1 Accuracy	Top 5 Acarg	Para
VGG16	79.0%	94.5%	2

Training	Validation	Testing
86.62%	91.95%	89.97%

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

RAM Disk

[1]

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.metrics import accuracy_score, f1_score, recall_score
import matplotlib.pyplot as plt
import numpy as np
class SimplePretrainedModel(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(SimplePretrainedModel, self).__init__()
        self.feature_extractor = nn.Sequential(
            nn.Linear(input_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU()
        )
        self.classifier = nn.Linear(hidden_dim, output_dim)
    def forward(self, x):
        features = self.feature_extractor(x)
        output = self.classifier(features)
        return output
np.random.seed(0)
X_train = np.random.rand(100, 10).astype(np.float32)
y_train = np.random.randint(0, 2, 100)
X_test = np.random.rand(30, 10).astype(np.float32)
y_test = np.random.randint(0, 2, 30)
train_data = TensorDataset(torch.from_numpy(X_train), torch.from_numpy(y_train))
test_data = TensorDataset(torch.from_numpy(X_test), torch.from_numpy(y_test))
train_loader = DataLoader(train_data, batch_size=16, shuffle=True)
test_loader = DataLoader(test_data, batch_size=10, shuffle=False)
```

Variables Terminal ✓ 7:43 PM Python 3

```python
    def forward(self, x):
        features = self.feature_extractor(x)
        output = self.classifier(features)
        return output
np.random.seed(0)
X_train = np.random.rand(100, 10).astype(np.float32)
y_train = np.random.randint(0, 2, 100)
X_test = np.random.rand(30, 10).astype(np.float32)
y_test = np.random.randint(0, 2, 30)
train_data = TensorDataset(torch.from_numpy(X_train), torch.from_numpy(y_train))
test_data = TensorDataset(torch.from_numpy(X_test), torch.from_numpy(y_test))
train_loader = DataLoader(train_data, batch_size=16, shuffle=True)
test_loader = DataLoader(test_data, batch_size=10, shuffle=False)
model = SimplePretrainedModel(input_dim=10, hidden_dim=16, output_dim=2)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)
num_epochs = 20
train_losses = []
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    for inputs, labels in train_loader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item() * inputs.size(0)
    epoch_loss = running_loss / len(train_loader.dataset)
    train_losses.append(epoch_loss)
    print(f'Epoch {epoch+1}/{num_epochs}, Loss: {epoch_loss:.4f}')
model.eval()
```

Untitled8.ipynb Saving failed since 7:38 PM

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

RAM Disk

[1]

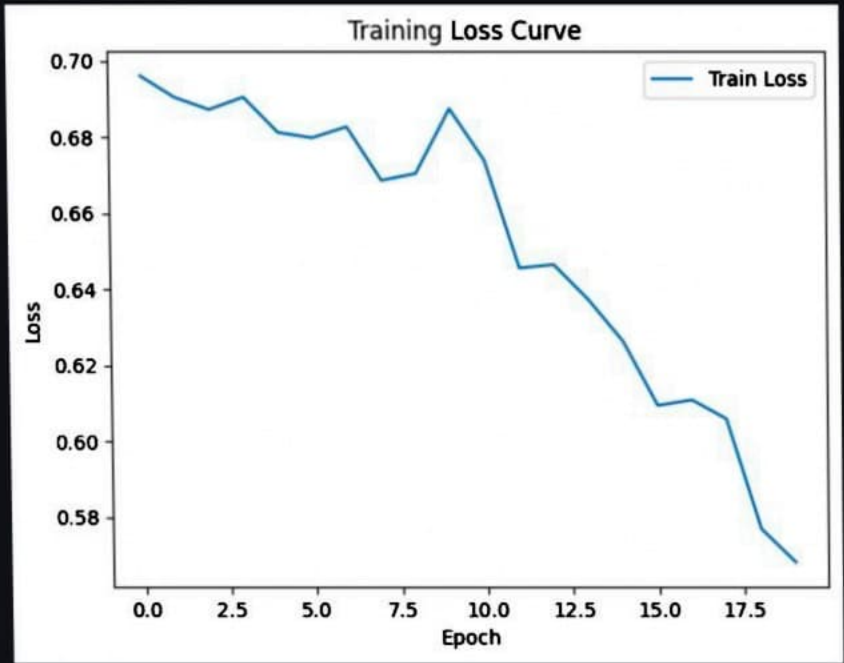
```python
            running_loss += loss.item() * inputs.size(0)
        epoch_loss = running_loss / len(train_loader.dataset)
        train_losses.append(epoch_loss)
        print(f'Epoch {epoch+1}/{num_epochs}, Loss: {epoch_loss:.4f}')
    model.eval()
    all_preds = []
    all_labels = []
    with torch.no_grad():
        for inputs, labels in test_loader:
            outputs = model(inputs)
            _, preds = torch.max(outputs, 1)
            all_preds.extend(preds.numpy())
            all_labels.extend(labels.numpy())
    accuracy = accuracy_score(all_labels, all_preds)
    f1 = f1_score(all_labels, all_preds)
    recall = recall_score(all_labels, all_preds)
    print(f'Accuracy: {accuracy:.4f}')
    print(f'F1 Score: {f1:.4f}')
    print(f'Recall: {recall:.4f}')
    plt.plot(train_losses, label='Train Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Training Loss Curve')
    plt.legend()
    plt.show()
```

```
Epoch 1/20, Loss: 0.6961
Epoch 2/20, Loss: 0.6904
Epoch 3/20, Loss: 0.6872
Epoch 4/20, Loss: 0.6905
Epoch 5/20, Loss: 0.6812
Epoch 6/20, Loss: 0.6798
```

Variables Terminal 7:43

```
Epoch 1/20, Loss: 0.6961
Epoch 2/20, Loss: 0.6904
Epoch 3/20, Loss: 0.6872
Epoch 4/20, Loss: 0.6905
Epoch 5/20, Loss: 0.6812
Epoch 6/20, Loss: 0.6798
Epoch 7/20, Loss: 0.6827
Epoch 8/20, Loss: 0.6686
Epoch 9/20, Loss: 0.6704
Epoch 10/20, Loss: 0.6874
Epoch 11/20, Loss: 0.6740
Epoch 12/20, Loss: 0.6456
Epoch 13/20, Loss: 0.6465
Epoch 14/20, Loss: 0.6373
Epoch 15/20, Loss: 0.6264
Epoch 16/20, Loss: 0.6094
Epoch 17/20, Loss: 0.6109
Epoch 18/20, Loss: 0.6059
Epoch 19/20, Loss: 0.5769
Epoch 20/20, Loss: 0.5681
Accuracy: 0.5333
F1 Score: 0.5333
Recall: 0.6154
```

```
Epoch 18/20, Loss: 0.6059
Epoch 19/20, Loss: 0.5769
Epoch 20/20, Loss: 0.5681
Accuracy: 0.5333
F1 Score: 0.5333
Recall: 0.6154
```




```
import torch
```