Completed

D. Khaja Nawaz

RA2311047010037

Deep Learning Techniques (LAB)

Date	Title	Sign
24/07/2025	1. Exploring the deep learning platform	
31/07/25	2. Implement a classifier using open-source dataset	7/8/25
31/07/25	3. Study of the classifiers with respect to statistical parameters	14/8/25
14/08/25	4. Build a simple feed forward neural network to recognize handwritten character	14/8/25
22/08/25	5. Study of Activation functions and their role	9/9
09/09/25	6. Implement gradient descent and backpropagation in deep neural network	
16/09/25	7. Build a CNN model to classify cat and Dog image	23/9/25
30/09/2025	8. Experiment of LSTM	10/10/25
30/09/2025	9. Build a Recurrent Neural Network	
9/10/2025	10. perform compression on MNIST dataset using autoencoder	
9/10/2025	11. Experiments using variational autoen	
03/11/25	12. Implement a Deep convolutional GAN to generate complex color image	
03/11/25	13. Understanding the architecture of pre-trained model	
03/11/25	14. Implement a pre-trained CNN model as feature Extractor using	
03/11/25	15. Implement a yolo model to detect objects	

03/11/25 LAB: 13

Understanding the architecture of pre-trained model.

*Aim :-

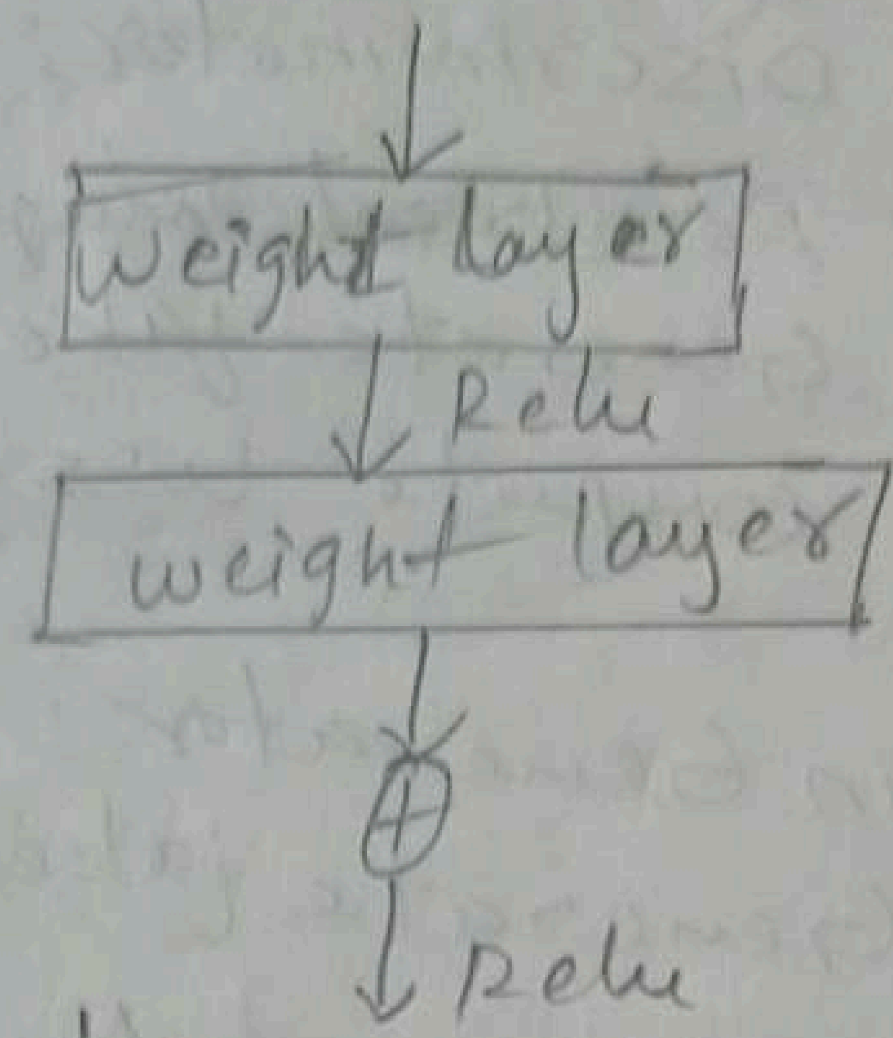
To analyze and understand the architecture and working principle of a pre-trained deep learning model (such as VGG16, ResNet, or inception), used for image classification and feature extraction

*Objective :-

1. To study the layer structure (convolutional, pooling, fully connected) of a pre-trained CNN model.
2. To understand transfer learning - how pre-trained model can be reused for new tasks.
3. To visualize and summarize the model architecture
4. To identify how pre-trained weights improve training efficiency and accuracy.

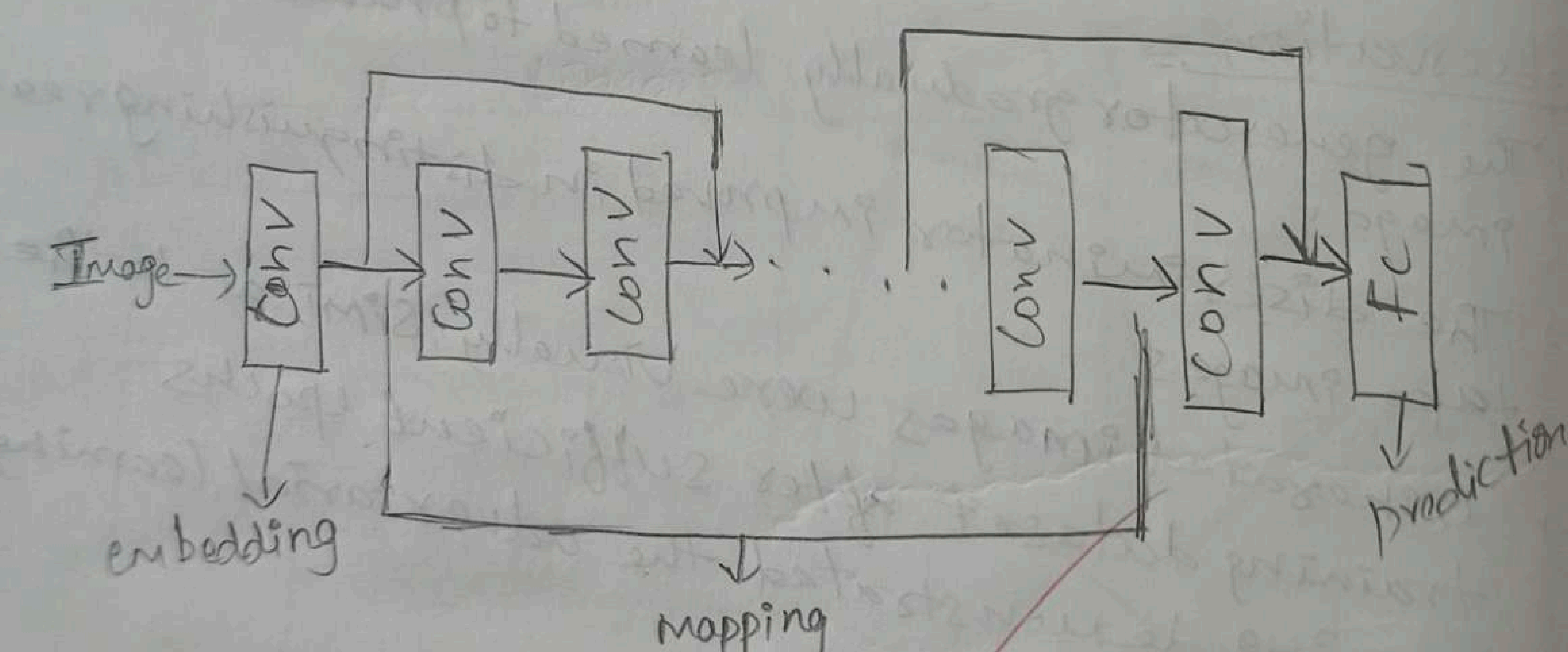
*Pseudocode :-

Begin
1. Import ~~deep learning~~ library (e.g., Tensorflow or pytorch)
2. Load a pre-trained model (e.g., VGG16, ResNet50)
   -> include weights = 'imagenet'
   -> exclude top layer if using for feature extraction
3. Display model summary
   -> print layer names, types, output shapes and parameter counts



desired mapping $H(x) = F(x) + x$

required Function



Output :

Total params : 11,689,512
Trainable params : 11,689,512
Non Trainable params : 0

layers Names :

conv1 : Conv 2d
bn1 : Batch Norm 2d
Relu : Relu
maxpool : Max pool 2d
layer1 : Sequential
layer2 : Sequential
layer3 : Sequential
layer4 : Sequential
avgpool : Adaptive Avgpool2d
Fc : Linear

4. For each layer in model:
   → Identity type (conv, pool, Dense, etc)
   → note activation function and number of filteres

5. Visualize architecture diagram
   → show flow from input image to output class

6. optional: Test with a sample image
   → preprocess image to required input size
   → pass through model → get prediction

*observation:-

→ The pre-trained model has convolutional, pooling and fully connected layers

→ convolutional layers extract features, pooling layers reduce size

→ pre-trained weights allow faster training and better accuracy.

→ sample images are correctly classified or have meaningful features extracted.

* Result:-

Successfully implemented understanding the archeticture of pre-trained model.

File Edit View Insert Runtime Tools Help

mmands + Code + Text ▷ Run all

```python
# lab13_pretrained_model_architecture.py
import torch
import torchvision.models as models

# Load a pretrained model (VGG16)
model = models.vgg16(pretrained=True)

# Print model architecture
print("===== VGG16 Pre-trained Model Architecture =====")
print(model)

# Print only feature extractor part
print("\n===== Feature Extractor Layers =====")
print(model.features)

# Print classifier part
print("\n===== Classifier Layers =====")
print(model.classifier)

# Print total trainable parameters
total_params = sum(p.numel() for p in model.parameters() if p.requires_grad)
print(f"\nTotal Trainable Parameters: {total_params}")


14


import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms, models
from torch.utils.data import DataLoader

# 1. Data loading
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
train_data = datasets.CIFAR10(root='./data', train=True, transform=transform, download=True)
test_data = datasets.CIFAR10(root='./data', train=False, transform=transform, download=True)
train_loader = DataLoader(train_data, batch_size=64, shuffle=True)
test_loader = DataLoader(test_data, batch_size=64, shuffle=False)

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

```python
from torchvision import datasets, transforms, models
from torch.utils.data import DataLoader

# 1. Data loading
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
train_data = datasets.CIFAR10(root='./data', train=True, transform=transform, download=Tru
test_data = datasets.CIFAR10(root='./data', train=False, transform=transform, download=Tru
train_loader = DataLoader(train_data, batch_size=64, shuffle=True)
test_loader = DataLoader(test_data, batch_size=64, shuffle=False)

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

# 3. Freeze feature extraction layers
for param in model.parameters():
    param.requires_grad = False

# 4. Replace final FC layer for 10 CIFAR classes
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, 10)

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

# 6. Train the model
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

for epoch in range(5):
    model.train()
    running_loss = 0.0
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        loss = criterion(outputs, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    print(f"Epoch [{epoch+1}/5] Loss: {running_loss/len(train_loader):.4f}")

# 7. Evaluate accuracy
model.eval()
```

ommands + Code + Text ▷ Run all

```
model = models.resnet18(pretrained=True)

# 3. Freeze feature extraction layers
for param in model.parameters():
    param.requires_grad = False

# 4. Replace final FC layer for 10 CIFAR classes
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, 10)

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

# 6. Train the model
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

for epoch in range(5):
    model.train()
    running_loss = 0.0
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        loss = criterion(outputs, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    print(f"Epoch [{epoch+1}/5] Loss: {running_loss/len(train_loader):.4f}")

# 7. Evaluate accuracy
model.eval()
correct, total = 0, 0
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)
        correct += (predicted == labels).sum().item()
print(f"Test Accuracy: {100 * correct / total:.2f}%")
```

```
        correct += (predicted == labels).sum().item()
print(f"Test Accuracy: {100 * correct / total:.2f}%")
```

```
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments oth
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/vgg16-397923af.pth" to /root/.cache/torch/hub/chec
100%|██████████| 528M/528M [00:07<00:00, 78.0MB/s]
===== VGG16 Pre-trained Model Architecture =====
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
```

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

```
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)

===== Feature Extractor Layers =====
Sequential(
  (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): ReLU(inplace=True)
  (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (3): ReLU(inplace=True)
  (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (6): ReLU(inplace=True)
  (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (8): ReLU(inplace=True)
  (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (11): ReLU(inplace=True)
  (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (13): ReLU(inplace=True)
  (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (15): ReLU(inplace=True)
  (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (18): ReLU(inplace=True)
  (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (20): ReLU(inplace=True)
  (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (22): ReLU(inplace=True)
  (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (25): ReLU(inplace=True)
  (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (27): ReLU(inplace=True)
  (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (29): ReLU(inplace=True)
  (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)

===== Classifier Layers =====
Sequential(
  (0): Linear(in_features=25088, out_features=4096, bias=True)
  (1): ReLU(inplace=True)
```