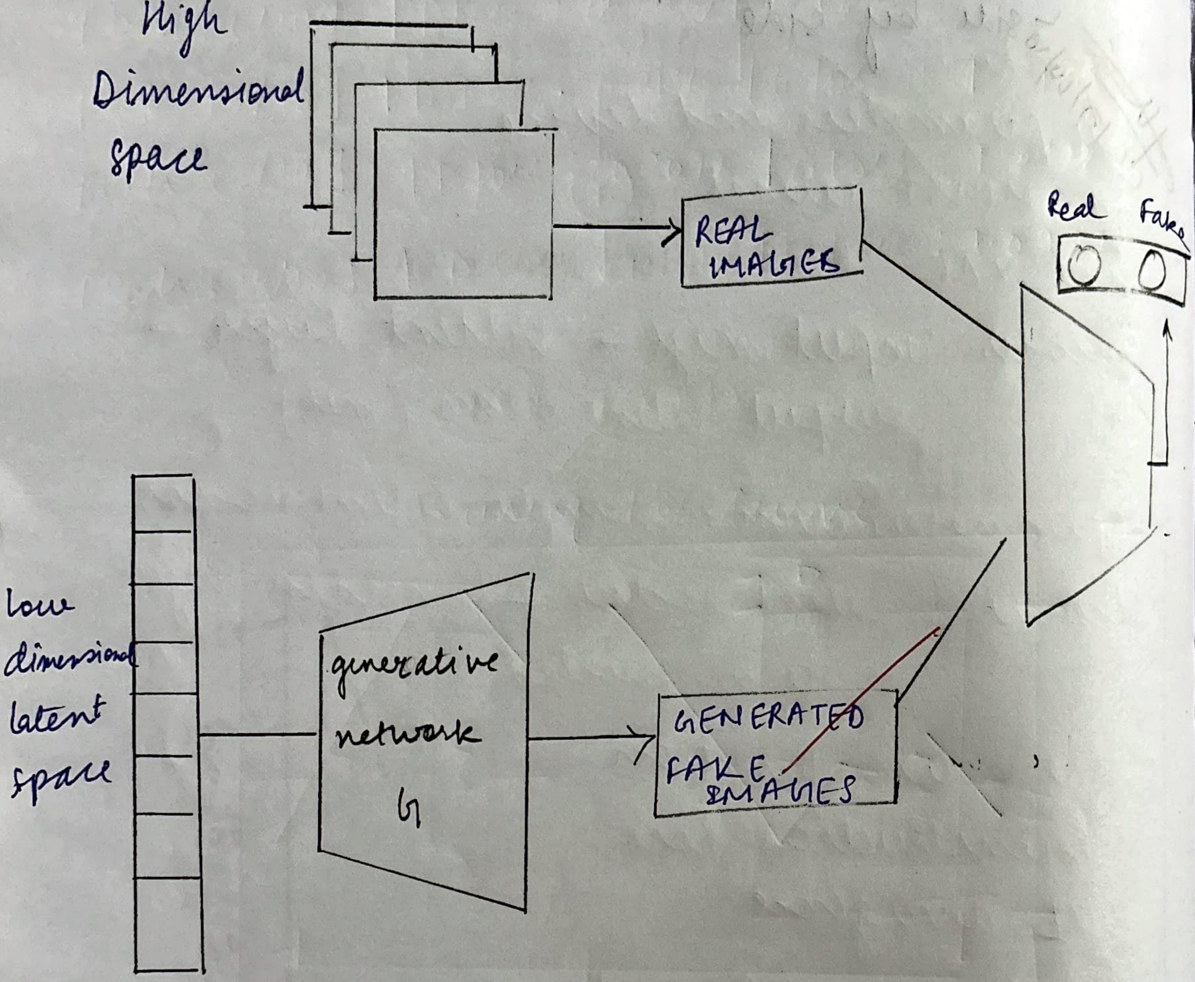
experiment 12
27/10/2025

Implement GAN to generate images

Aim: To design and implement a Deep convolutional Generative Adversarial Network using PyTorch for generating realistic handwritten digit images

Objectives:
- Build a generator network using transposed convolutional layers to generate synthetic MNIST like images from random noise and also colorful images
- build Discriminator network
- Train both network adversarially
- optimize both networks using BCE loss and Adam optimizer

Pseudocode
- Import Libraries
- Define generator Network
  - layers:
    - ConvTranspose2d → BatchNorm → ReLU
    - ConvTranspose2d → BatchNorm → ReLU
    - ConvTranspose2d → Tanh
- Define Discriminator
  - Layers:
    - Conv2d → LeakyReLU
    - Conv2d → BatchNorm → LeakyReLU
    - Flatten → Linear → sigmoid


High Dimensional space
REAL IMAGES
Real
Fake
low dimensional latent space
generative network
G
GENERATED FAKE IMAGES

Initialize Components
Load Dataset
Train generator
optimizer_G. zero_grad()
Sample random noise (z)
generate fake_images = G(z)
compute g-loss = BCE(D(fake_images), real labels)
Backpropagate and update generator

Train Discriminator
optimizer_D. zero_grad()
Compute real_loss = BCE(D(real_images), real_labels)

Compute fake_loss = BCE(fake_images.detach()), fake_labels

Backpropagate and update discriminator

Visualize Results.

Results: Successfully implemented GAN and generated images (Handwritten images and colorful images) epoch [1/10] | D loss : 0.2675 | G loss : 1.3297

epoch [2/10] | D loss : 0.2787 | G loss : 1.8825

epoch [3/10] | D loss : 0.2546 | G loss : 2.0490

:

:

epoch [6/10] | D loss : 0.3462 | G loss : 1.1048

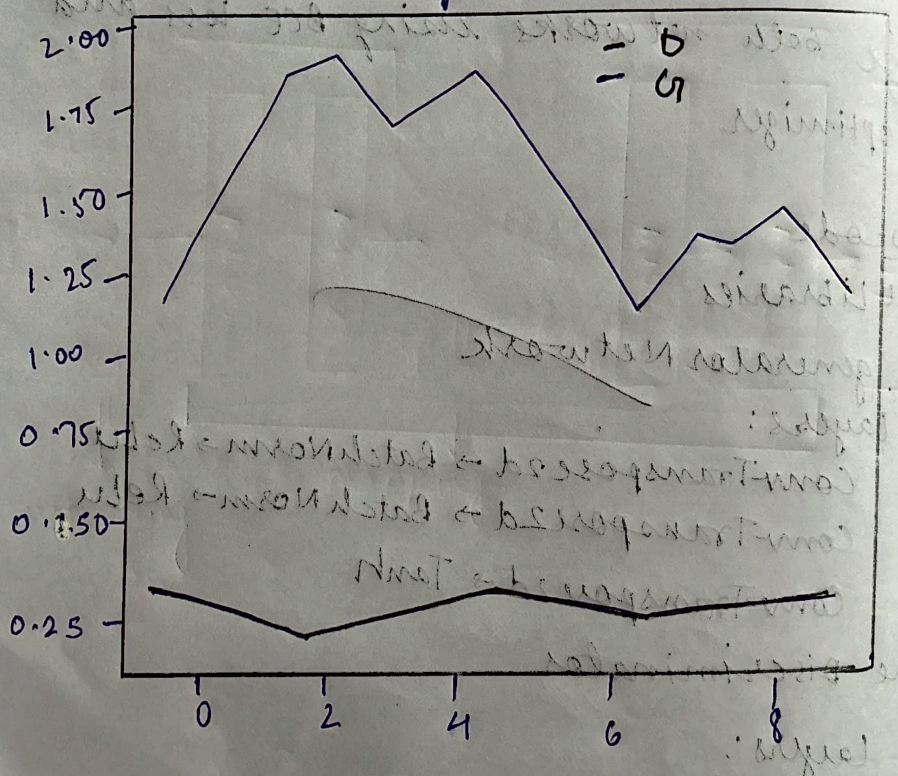
epoch [7/10] | D Loss : 0.3175 | G loss : 1.3139

epoch [8/10] | D loss : 0.3072 | G Loss : 1.2928

.

epoch [10/10] | D Loss : 0.3195 | G Loss : 1.0922

Training Loss Curve



```python
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import torchvision.utils as vutils
import matplotlib.pyplot as plt
import numpy as np

# 1. Hyperparameters
batch_size = 128
latent_dim = 100
epochs = 10
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# 2. Data Loading (CIFAR-10 color images)
transform = transforms.Compose([
    transforms.Resize(64),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
dataset = datasets.CIFAR10(root='./data', download=True, transform=transform)
dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True)

# 3. Define Generator
class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        self.model = nn.Sequential(
            nn.ConvTranspose2d(100, 512, 4, 1, 0, bias=False),
            nn.BatchNorm2d(512), nn.ReLU(True),
            nn.ConvTranspose2d(512, 256, 4, 2, 1, bias=False),
            nn.BatchNorm2d(256), nn.ReLU(True),
            nn.ConvTranspose2d(256, 128, 4, 2, 1, bias=False),
```

3:30 PM

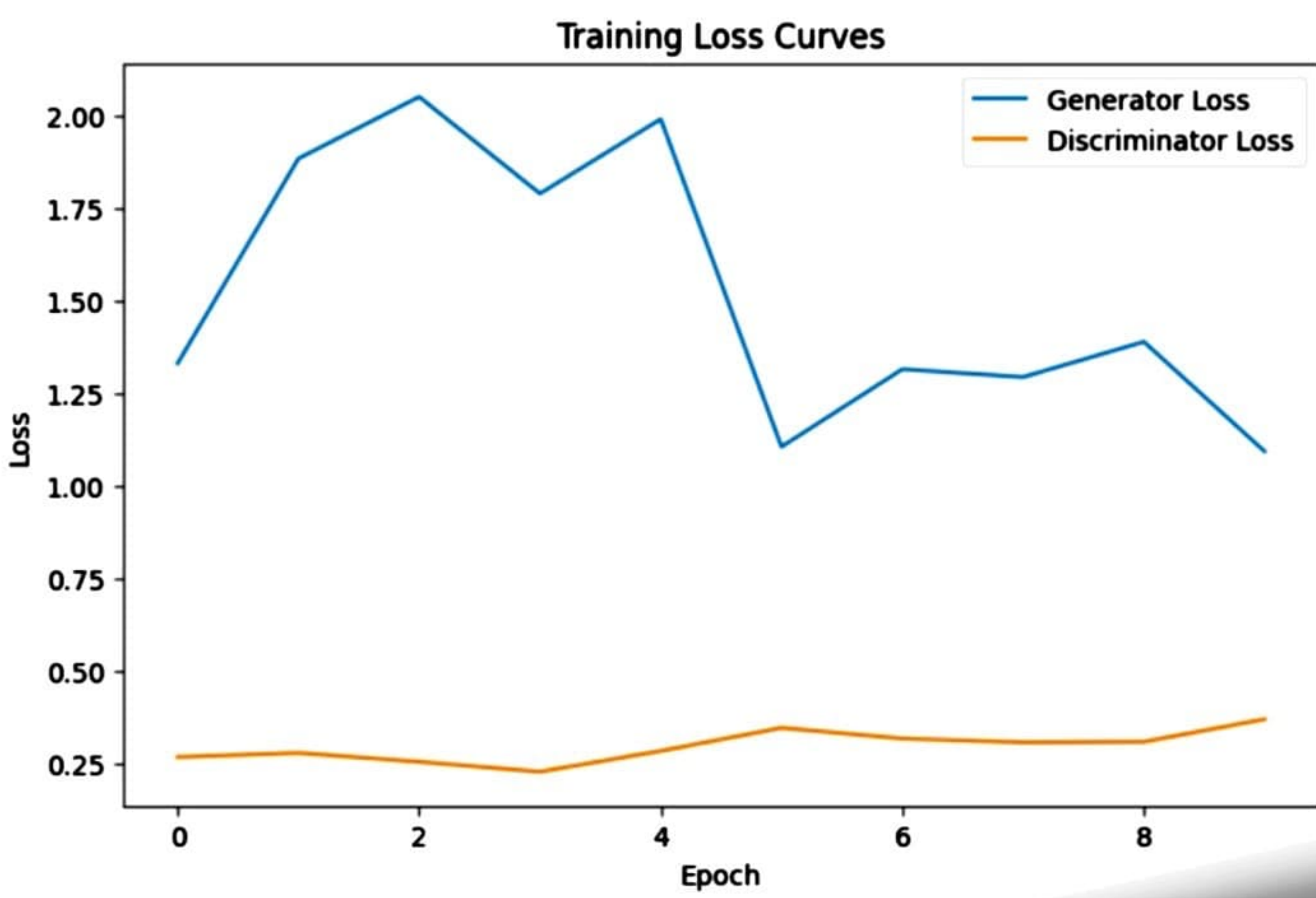
```
Epoch [1/10] D Loss: 0.6843, G Loss: 4.9264
Epoch [2/10] D Loss: 0.3498, G Loss: 2.2724
Epoch [3/10] D Loss: 0.3579, G Loss: 3.2501
Epoch [4/10] D Loss: 2.5385, G Loss: 1.3741
Epoch [5/10] D Loss: 0.6166, G Loss: 2.7679
Epoch [6/10] D Loss: 0.2046, G Loss: 1.8963
Epoch [7/10] D Loss: 0.0389, G Loss: 3.4507
Epoch [8/10] D Loss: 0.6460, G Loss: 5.6845
Epoch [9/10] D Loss: 0.0319, G Loss: 4.3721
Epoch [10/10] D Loss: 0.0202, G Loss: 5.0011
```

Generated Images

3:31 PM

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
from tqdm import tqdm

# ---------------------------------------------------------
# Step 3: Define Generator and Discriminator (Conv-based)
# ---------------------------------------------------------
class Generator(nn.Module):
    def __init__(self, latent_dim=100):
        super(Generator, self).__init__()
        self.model = nn.Sequential(
            nn.ConvTranspose2d(latent_dim, 128, 7, 1, 0, bias=False),
            nn.BatchNorm2d(128),
            nn.ReLU(True),

            nn.ConvTranspose2d(128, 64, 4, 2, 1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(True),

            nn.ConvTranspose2d(64, 1, 4, 2, 1, bias=False),
            nn.Tanh()
        )

    def forward(self, z):
        return self.model(z)


class Discriminator(nn.Module):
    def __init__(self):
```

3:31 PM



3:31 PM

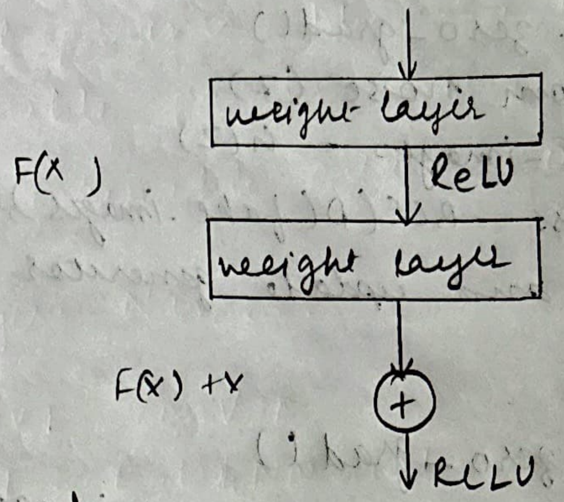
Experiment 13
27/10/2025

Understanding a Pre Trained Model

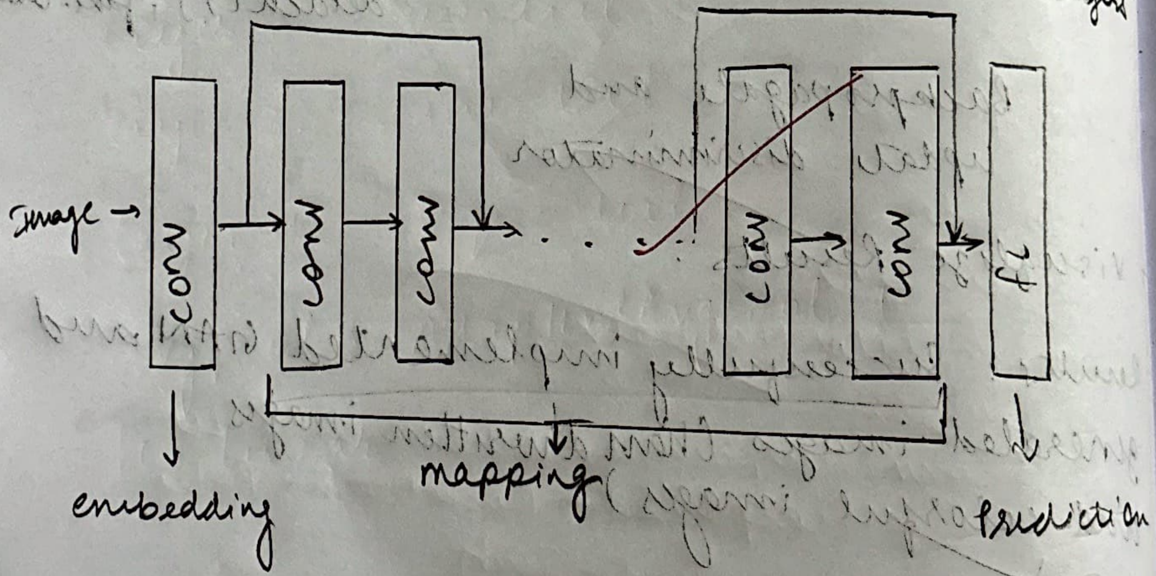
Aim: The primary aim of this experiment is to load and inspect the complete architecture of a pre-trained ResNet-18 model using PyTorch.

Objectives:
- → To import the libraries and understand its use.
- → To fetch the ResNet-18 model architecture
- → Print the summary and print the architecture.

Pseudocode
- → Import the necessary libraries like torch, torchvision and summary from torchsummary
- → Load the ResNet-18 model from the models library
- → Include the weight pre-trained on ImageNet
- → Set newly loaded mode to evaluation mode
- → After evaluation print the model architecture and the Model summary.



desired mapping ← $H(x) = F(x) + x$ → residual function learned by stacked layers



Output

Total params : 11,689,512

Trainable params: 11,689,512

Non Trainable params : 0

Layers Names :

conv1 : Conv2D
bn1 : BatchNorm2d
relu : ReLU
maxpool : MaxPool2d
layer1 : Sequential
layer2 : Sequential
layer3 : Sequential
layer4 : Sequential
avgpool : AdaptiveAvgPool2d
fc : Linear

The summary is calculated based on sample input size of 3 channels, 224 pixels height and 224 pixels width.

Result: Successfully understood the structure of a pretrained model.

89.1

```python
import torch
import torchvision.models as models
from torchsummary import summary

model = models.resnet18(pretrained=True)

# Set model to evaluation mode
model.eval()

print("\n🔍 Model Architecture:\n")
print(model)


print("\n🔍 Model Summary:\n")
summary(model, (3, 224, 224))

print("\n🔍 Layer Names:\n")
for name, layer in model.named_children():
    print(f"{name}: {layer.__class__.__name__}")
```

3:35 PM ✓

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
```

3:35 PM ✓

```
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 112, 112]           9,408
       BatchNorm2d-2         [-1, 64, 112, 112]             128
              ReLU-3         [-1, 64, 112, 112]               0
         MaxPool2d-4           [-1, 64, 56, 56]               0
            Conv2d-5           [-1, 64, 56, 56]          36,864
       BatchNorm2d-6           [-1, 64, 56, 56]             128
              ReLU-7           [-1, 64, 56, 56]               0
            Conv2d-8           [-1, 64, 56, 56]          36,864
       BatchNorm2d-9           [-1, 64, 56, 56]             128
             ReLU-10           [-1, 64, 56, 56]               0
       BasicBlock-11           [-1, 64, 56, 56]               0
           Conv2d-12           [-1, 64, 56, 56]          36,864
      BatchNorm2d-13           [-1, 64, 56, 56]             128
             ReLU-14           [-1, 64, 56, 56]               0
           Conv2d-15           [-1, 64, 56, 56]          36,864
      BatchNorm2d-16           [-1, 64, 56, 56]             128
             ReLU-17           [-1, 64, 56, 56]               0
       BasicBlock-18           [-1, 64, 56, 56]               0
           Conv2d-19          [-1, 128, 28, 28]          73,728
      BatchNorm2d-20          [-1, 128, 28, 28]             256
             ReLU-21          [-1, 128, 28, 28]               0
           Conv2d-22          [-1, 128, 28, 28]         147,456
      BatchNorm2d-23          [-1, 128, 28, 28]             256
           Conv2d-24          [-1, 128, 28, 28]           8,192
      BatchNorm2d-25          [-1, 128, 28, 28]             256
             ReLU-26          [-1, 128, 28, 28]               0
       BasicBlock-27          [-1, 128, 28, 28]               0
           Conv2d-28          [-1, 128, 28, 28]         147,456
      BatchNorm2d-29          [-1, 128, 28, 28]             256
             ReLU-30          [-1, 128, 28, 28]               0
           Conv2d-31          [-1, 128, 28, 28]         147,456
      BatchNorm2d-32          [-1, 128, 28, 28]             256
             ReLU-33          [-1, 128, 28, 28]               0
       BasicBlock-34          [-1, 128, 28, 28]               0
```

3:35 PM ✓

```
           Conv2d-60            [-1, 512, 7, 7]       2,359,296
      BatchNorm2d-61            [-1, 512, 7, 7]           1,024
             ReLU-62            [-1, 512, 7, 7]               0
           Conv2d-63            [-1, 512, 7, 7]       2,359,296
      BatchNorm2d-64            [-1, 512, 7, 7]           1,024
             ReLU-65            [-1, 512, 7, 7]               0
       BasicBlock-66            [-1, 512, 7, 7]               0
AdaptiveAvgPool2d-67            [-1, 512, 1, 1]               0
           Linear-68                 [-1, 1000]         513,000
================================================================
Total params: 11,689,512
Trainable params: 11,689,512
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 62.79
Params size (MB): 44.59
Estimated Total Size (MB): 107.96
----------------------------------------------------------------

🔍 Layer Names:

conv1: Conv2d
bn1: BatchNorm2d
relu: ReLU
maxpool: MaxPool2d
layer1: Sequential
layer2: Sequential
layer3: Sequential
layer4: Sequential
avgpool: AdaptiveAvgPool2d
fc: Linear
```

3:36 PM ✓