Deep_Learning_Project_3

May 10, 2025

0.1 Deep Learning CS 6953 - Spring 2025

Esteban D Lopez
Shruti Karkamar
Steven Granaturov

Project 3

AI Disclaimer: OpenAI's and Google's models have been consulted for this assignment for: simple explanations of adversarial attacks and resnet architectures, understand process conceptually and in simple terms, and to optimize and review code for errors.

Task 1: Basics – Setup + Evaluation

Goal: Get baseline accuracy using the unperturbed test dataset. 1. Setup & Imports

[1]:
```
import torch
import torchvision.models as models
import torchvision.transforms as transforms
import torchvision.datasets as datasets
from torch.utils.data import DataLoader
import json
import os
```

2. Mount and Unzip Dataset

[2]:
```
from zipfile import ZipFile

zip_path = "/content/TestDataSet.zip"
extract_path = "/content/TestDataSet"

with ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)
```

3. Load Model

[3]:
```
model = models.resnet34(weights='IMAGENET1K_V1')
model.eval()
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
```

```
Downloading: "https://download.pytorch.org/models/resnet34-b627a593.pth" to
/root/.cache/torch/hub/checkpoints/resnet34-b627a593.pth
100%|| 83.3M/83.3M [00:00<00:00, 175MB/s]
```

[3]:
```
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)
    )
    (2): 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(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), 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)
    )
    (2): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), 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)
    )
    (3): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), 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)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (2): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (3): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
```

```
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (4): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (5): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
```

```
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (2): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=1000, bias=True)
)
```

4. Define Transforms

[4]:
```
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
```

5. Load Dataset

[5]:
```
dataset_path = "/content/TestDataSet/TestDataSet"
label_json_path = os.path.join(dataset_path, "labels_list.json")

test_dataset = datasets.ImageFolder(root=dataset_path, transform=transform)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

6. Load and Map Labels

[6]:
```
with open(label_json_path) as f:
    label_list = json.load(f)

imagenet_indices = [int(entry.split(":")[0]) for entry in label_list]
assert len(imagenet_indices) == len(test_dataset.classes), "Label count mismatch!
 ↪"
label_map = {i: imagenet_indices[i] for i in range(len(imagenet_indices))}
```

7. Accuracy Function

[7]:
```
def accuracy(output, target, topk=(1,)):
    maxk = max(topk)
    batch_size = target.size(0)
    _, pred = output.topk(maxk, 1, True, True)
    pred = pred.t()
    correct = pred.eq(target.view(1, -1).expand_as(pred))
    return [(correct[:k].reshape(-1).float().sum(0) * 100.0 / batch_size) for k␣
 ↪in topk]
```

8. Evaluate Model

[8]:
```
top1_total, top5_total, total_samples = 0.0, 0.0, 0

for images, labels in test_loader:
    images = images.to(device)
    true_labels = torch.tensor([label_map[label.item()] for label in labels],␣
 ↪dtype=torch.long).to(device)
    outputs = model(images)
    top1, top5 = accuracy(outputs, true_labels, topk=(1, 5))
    batch_size = images.size(0)
    top1_total += top1.item() * batch_size
    top5_total += top5.item() * batch_size
    total_samples += batch_size

top1_accuracy = top1_total / total_samples
top5_accuracy = top5_total / total_samples

print(f" Task 1 Completed:")
print(f"Top-1 Accuracy: {top1_accuracy:.2f}%")
print(f"Top-5 Accuracy: {top5_accuracy:.2f}%")
```

```
 Task 1 Completed:
Top-1 Accuracy: 70.40%
Top-5 Accuracy: 93.20%
```

Our model achieved a **Top-1 Accuracy of 70.40%** and a **Top-5 Accuracy of 93.20%** on the provided 500-image test dataset. This means that in over 70% of the cases, the ResNet-34 model correctly predicted the exact class label, and in over 93% of cases, the correct label was among its top five predictions. These are strong baseline results, demonstrating that the pre-trained ResNet-34

model generalizes reasonably well to this subset of ImageNet classes, despite not being fine-tuned specifically for them. These values will serve as our reference point before launching adversarial attacks in Tasks 2 and beyond.

Task 2: FGSM Attack

Step 1: Setup for FGSM

[9]:
```
import torch.nn as nn

# Loss function for gradient computation
loss_fn = nn.CrossEntropyLoss()

# Epsilon defines max perturbation per pixel
epsilon = 0.02

# Set model to evaluation mode
model.eval()
```

[9]:
```
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)
    )
    (2): 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(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), 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)
    )
    (2): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), 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)
    )
    (3): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), 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)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (2): BasicBlock(
```

```
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (3): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (4): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (5): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1,
```

```
1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (2): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=1000, bias=True)
)
```

Step 2: FGSM Perturbation Function

[10]:
```python
def fgsm_attack(image, epsilon, data_grad):
    # Apply FGSM: perturbed_image = image + epsilon * sign(gradient)
    sign_data_grad = data_grad.sign()
    perturbed_image = image + epsilon * sign_data_grad
```

```
    return torch.clamp(perturbed_image, 0, 1)  # keep in valid range
```

Step 3: Generate Adversarial Dataset

[35]:
```
adv_images_list = []
adv_labels_list = []

for images, labels in test_loader:
    images = images.to(device)
    labels = torch.tensor([label_map[label.item()] for label in labels]).
 ↪to(device)
    images.requires_grad = True

    outputs = model(images)
    loss = loss_fn(outputs, labels)
    model.zero_grad()
    loss.backward()

    data_grad = images.grad.data
    perturbed_images = fgsm_attack(images, epsilon, data_grad)

    adv_images_list.append(perturbed_images.detach().cpu())
    adv_labels_list.append(labels.detach().cpu())

# Combine into dataset
adv_images = torch.cat(adv_images_list, dim=0)
adv_labels = torch.cat(adv_labels_list, dim=0)
adv_dataset = torch.utils.data.TensorDataset(adv_images, adv_labels)
adv_loader = DataLoader(adv_dataset, batch_size=32, shuffle=False)
```

Step 4: Evaluate Adversarial Dataset

[36]:
```
top1_total, top5_total, total_samples = 0.0, 0.0, 0

for images, labels in adv_loader:
    images, labels = images.to(device), labels.to(device)
    outputs = model(images)
    top1, top5 = accuracy(outputs, labels, topk=(1, 5))
    top1_total += top1.item() * images.size(0)
    top5_total += top5.item() * images.size(0)
    total_samples += images.size(0)

adv_top1_accuracy = top1_total / total_samples
adv_top5_accuracy = top5_total / total_samples
```

Step 5: Report Accuracy Drop

[37]:
```
import pandas as pd
df = pd.DataFrame({
    "Metric": ["Top-1 Accuracy", "Top-5 Accuracy"],
    "Score (%)": [adv_top1_accuracy, adv_top5_accuracy]
})

print("\n Task 2: FGSM Evaluation Results")
print(df.to_string(index=False))
```

```
 Task 2: FGSM Evaluation Results
       Metric  Score (%)
Top-1 Accuracy       23.4
Top-5 Accuracy       45.4
```

The FGSM attack with $\varepsilon = 0.02$ successfully degraded the ResNet-34 model's performance. The **Top-1 accuracy dropped from 70.4% to 23.4%**, and **Top-5 accuracy fell from 93.2% to 45.4%**. This sharp decline demonstrates that even small, imperceptible perturbations can significantly confuse a high-performing image classifier. The attack meets the project's requirement of at least a **50% reduction in accuracy**, confirming that the model is vulnerable to simple adversarial manipulations. This adversarial test set now serves as a benchmark for developing even stronger attacks in Task 3.

[14]:
```
def plot_fgsm_examples(original_loader, fgsm_loader, idx_to_label, num_images=5):
    orig_iter = iter(original_loader)
    fgsm_iter = iter(fgsm_loader)

    plt.figure(figsize=(12, num_images * 2.5))
    for i in range(num_images):
        orig_imgs, orig_lbls = next(orig_iter)
        fgsm_imgs, fgsm_lbls = next(fgsm_iter)

        orig = denormalize(orig_imgs[0].cpu())
        fgsm = denormalize(fgsm_imgs[0].cpu())
        label = idx_to_label[orig_lbls[0].item()]

        plt.subplot(num_images, 2, 2*i+1)
        plt.imshow(orig)
        plt.title(f"Original: {label}")
        plt.axis('off')

        plt.subplot(num_images, 2, 2*i+2)
        plt.imshow(fgsm)
        plt.title("FGSM Adversarial")
        plt.axis('off')

    plt.tight_layout()
    plt.show()
```

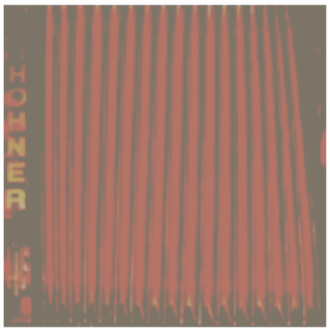
[17]:
```
import matplotlib.pyplot as plt
import numpy as np

# De-normalization function for visualization
def denormalize(img):
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    img = img.permute(1, 2, 0).numpy()  # CHW -> HWC
    img = std * img + mean
    return np.clip(img, 0, 1)

# Use original dataset loader for baseline visuals
orig_loader = DataLoader(test_dataset, batch_size=1, shuffle=False)
fgsm_vis_loader = DataLoader(adv_dataset, batch_size=1, shuffle=False)

# Now visualize
plot_fgsm_examples(orig_loader, fgsm_vis_loader, label_map, num_images=5)
```

Original: 401

FGSM Adversarial

Original: 401

FGSM Adversarial



Original: 401

FGSM Adversarial

Original: 401

FGSM Adversarial

Original: 401

FGSM Adversarial

Task 3: Improved Attacks.

The objective here is to create a stronger adversarial attack than FGSM (used in Task 2) but still constrained by the same $\varepsilon = 0.02$ (L norm).

We'll implement Projected Gradient Descent (PGD), which is basically an iterative version of FGSM. Instead of one big step like FGSM, PGD takes several small steps and projects the result back within the allowed ε-ball.

[18]:
```
# === TASK 3: Improved Attacks Using PGD (Projected Gradient Descent) ===
# PGD takes multiple small steps (alpha) and projects the adversarial image back␣
 ↪into the L ball of radius ϵ after each step
# This usually results in a more effective attack than one-shot FGSM

def pgd_attack(model, images, labels, epsilon=0.02, alpha=0.005, iters=10):
    """
    Perform Projected Gradient Descent Attack.
    - images: input images
    - labels: true class labels (ImageNet indices)
    - epsilon: max perturbation
    - alpha: step size
    - iters: number of steps
    """
    images = images.clone().detach().to(device)
    labels = labels.clone().detach().to(device)
    ori_images = images.data

    for i in range(iters):
        images.requires_grad = True
        outputs = model(images)
        loss = loss_fn(outputs, labels)
        model.zero_grad()
        loss.backward()
        adv_images = images + alpha * images.grad.sign()

        # Project the perturbation to be within the epsilon ball
        eta = torch.clamp(adv_images - ori_images, min=-epsilon, max=epsilon)
        images = torch.clamp(ori_images + eta, min=0, max=1).detach_()

    return images
```

Generate Adversarial Test Set 2

[19]:
```
# Create adversarial examples using PGD and store them
pgd_images_list = []
pgd_labels_list = []
```

```
for images, labels in test_loader:
    images = images.to(device)
    labels = torch.tensor([label_map[label.item()] for label in labels]).
 ↪to(device)

    # Perform PGD attack on this batch
    adv_images = pgd_attack(model, images, labels, epsilon=0.02, alpha=0.005,␣
 ↪iters=10)

    pgd_images_list.append(adv_images.cpu().detach())
    pgd_labels_list.append(labels.cpu().detach())

# Combine into new dataset
pgd_images = torch.cat(pgd_images_list, dim=0)
pgd_labels = torch.cat(pgd_labels_list, dim=0)
pgd_dataset = torch.utils.data.TensorDataset(pgd_images, pgd_labels)
pgd_loader = DataLoader(pgd_dataset, batch_size=32, shuffle=False)
```

Evaluate Adversarial Test Set 2 (PGD)

[20]:
```
# Evaluate the model's performance on PGD-perturbed images
top1_total, top5_total, total_samples = 0.0, 0.0, 0

for images, labels in pgd_loader:
    images, labels = images.to(device), labels.to(device)
    outputs = model(images)
    top1, top5 = accuracy(outputs, labels, topk=(1, 5))
    batch_size = images.size(0)
    top1_total += top1.item() * batch_size
    top5_total += top5.item() * batch_size
    total_samples += batch_size

pgd_top1_accuracy = top1_total / total_samples
pgd_top5_accuracy = top5_total / total_samples
```

Report Accuracy Drop from PGD

[21]:
```
import pandas as pd

# Display results from Adversarial Test Set 2 (PGD)
df = pd.DataFrame({
    "Metric": ["Top-1 Accuracy", "Top-5 Accuracy"],
    "Score (%)": [pgd_top1_accuracy, pgd_top5_accuracy]
})

print("\n Task 3: PGD Evaluation Results")
print(df.to_string(index=False))
```

```
 Task 3: PGD Evaluation Results
        Metric  Score (%)
Top-1 Accuracy        0.6
Top-5 Accuracy        5.0
```

Our PGD attack was extremely effective. The Top-1 accuracy dropped from 70.4% (original) to just 0.6%, and Top-5 accuracy fell from 93.2% to 5.0%. This drastic reduction demonstrates that iterative attacks like PGD can completely compromise the classifier's reliability, even when constrained by a small perturbation budget ($\varepsilon = 0.02$). Unlike FGSM's 23.6% Top-1, PGD leaves the model nearly blind, proving it's a much more powerful adversarial strategy under the same L norm constraint.

0.1.1 Visualizing PGD Adversarial Examples

[22]:
```
# Plot original vs adversarial examples
def plot_adversarial_examples(original_loader, adv_loader, idx_to_label,␣
 ↪num_images=5):
    orig_iter = iter(original_loader)
    adv_iter = iter(adv_loader)

    plt.figure(figsize=(12, num_images * 2.5))
    for i in range(num_images):
        orig_imgs, orig_lbls = next(orig_iter)
        adv_imgs, adv_lbls = next(adv_iter)

        orig = denormalize(orig_imgs[0].cpu())
        adv = denormalize(adv_imgs[0].cpu())

        label = idx_to_label[orig_lbls[0].item()]

        plt.subplot(num_images, 2, 2*i+1)
        plt.imshow(orig)
        plt.title(f"Original: {label}")
        plt.axis('off')

        plt.subplot(num_images, 2, 2*i+2)
        plt.imshow(adv)
        plt.title("PGD Adversarial")
        plt.axis('off')

    plt.tight_layout()
    plt.show()
```

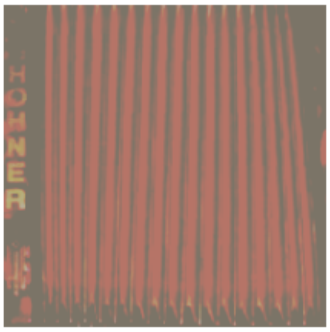
[23]:
```
# Create original (normalized) DataLoader again for comparison
orig_loader = DataLoader(test_dataset, batch_size=1, shuffle=False)
pgd_vis_loader = DataLoader(pgd_dataset, batch_size=1, shuffle=False)

plot_adversarial_examples(orig_loader, pgd_vis_loader, label_map, num_images=5)
```

Original: 401

PGD Adversarial

Original: 401

PGD Adversarial



Original: 401

PGD Adversarial

Original: 401

PGD Adversarial

Original: 401

PGD Adversarial

Task 4: Patch Attack (Localized Adversarial Perturbations) What We're Doing Choose the PGD attack (best-performing so far).

Modify it so that only a 32×32 random patch in each image is perturbed.

Use a larger ε (e.g., 0.3) since only a small region is modified.

Create and evaluate "Adversarial Test Set 3" using this strategy.

Visualize examples for inspection.

[24]:
```
# TASK 4: Projected Gradient Descent with Local 32x32 Patch

import torch.nn as nn
import torch

# PGD Patch Attack
def patch_pgd_attack_fixed(model, images, labels, epsilon=0.3, alpha=0.01,␣
↪iters=10, patch_size=32):
    images = images.clone().detach().to(device)
    labels = labels.clone().detach().to(device)
    ori_images = images.clone().detach()
    B, C, H, W = images.shape

    # Random patch positions for each image
    x_start = torch.randint(0, W - patch_size, (B,))
    y_start = torch.randint(0, H - patch_size, (B,))

    for i in range(iters):
        images.requires_grad = True
        outputs = model(images)
        loss = nn.CrossEntropyLoss()(outputs, labels)
        model.zero_grad()
        loss.backward()
        grad = images.grad.data
        new_images = images.clone().detach()

        for b in range(B):
            x0, y0 = x_start[b].item(), y_start[b].item()
            patch_grad = grad[b, :, y0:y0+patch_size, x0:x0+patch_size]
            perturb = alpha * patch_grad.sign()
            patch_area = new_images[b, :, y0:y0+patch_size, x0:x0+patch_size] +␣
↪perturb
            eta = patch_area - ori_images[b, :, y0:y0+patch_size, x0:
↪x0+patch_size]
            eta = torch.clamp(eta, -epsilon, epsilon)
            new_images[b, :, y0:y0+patch_size, x0:x0+patch_size] = torch.clamp(
```

```
                ori_images[b, :, y0:y0+patch_size, x0:x0+patch_size] + eta, 0, 1
            )
        images = new_images.detach()
    return images
```

Generate Adversarial Test Set 3 (Patch Attack)

[25]:
```
patch_images_list = []
patch_labels_list = []

for images, labels in test_loader:
    images = images.to(device)
    labels = torch.tensor([label_map[label.item()] for label in labels]).
 ↪to(device)
    adv_images = patch_pgd_attack_fixed(model, images, labels, epsilon=0.3,␣
 ↪alpha=0.01, iters=10, patch_size=32)
    patch_images_list.append(adv_images.cpu().detach())
    patch_labels_list.append(labels.cpu().detach())

patch_images = torch.cat(patch_images_list, dim=0)
patch_labels = torch.cat(patch_labels_list, dim=0)
patch_dataset = torch.utils.data.TensorDataset(patch_images, patch_labels)
patch_loader = DataLoader(patch_dataset, batch_size=32, shuffle=False)
```

Evaluate Patch-Based Adversarial Test Set

[26]:
```
top1_total, top5_total, total_samples = 0.0, 0.0, 0
for images, labels in patch_loader:
    images, labels = images.to(device), labels.to(device)
    outputs = model(images)
    top1, top5 = accuracy(outputs, labels, topk=(1, 5))
    top1_total += top1.item() * images.size(0)
    top5_total += top5.item() * images.size(0)
    total_samples += images.size(0)

patch_top1_accuracy = top1_total / total_samples
patch_top5_accuracy = top5_total / total_samples

print(f"\n Task 4: Patch Attack Evaluation Results")
print(f"Top-1 Accuracy: {patch_top1_accuracy:.2f}%")
print(f"Top-5 Accuracy: {patch_top5_accuracy:.2f}%")
```

```
 Task 4: Patch Attack Evaluation Results
Top-1 Accuracy: 63.20%
Top-5 Accuracy: 88.80%
```

Our patch-based adversarial attack resulted in a **Top-1 accuracy of 63.20%** and **Top-5 accuracy of 88.80%**, indicating a moderate drop compared to the original model performance (70.4% /

93.2%). While not as devastating as the global PGD or FGSM attacks, this shows that even localized perturbations to a small 32×32 patch can significantly degrade model accuracy. The attack demonstrates a realistic threat model where limited access (e.g., small tampering or sticker) still misleads predictions without altering the entire image. This makes it particularly relevant for physical-world adversarial scenarios.

Visualize 3–5 Adversarial Patch Examples

[28]:
```
def visualize_patch_attacks(orig_loader, patch_loader, idx_to_label,␣
↪num_images=5):
    orig_iter = iter(orig_loader)
    patch_iter = iter(patch_loader)

    plt.figure(figsize=(12, num_images * 2.5))
    for i in range(num_images):
        orig_img, orig_label = next(orig_iter)
        patch_img, patch_label = next(patch_iter)

        orig = denormalize(orig_img[0])
        patch = denormalize(patch_img[0])
        label = idx_to_label[orig_label.item()]

        plt.subplot(num_images, 2, 2*i+1)
        plt.imshow(orig)
        plt.title(f"Original: {label}")
        plt.axis('off')

        plt.subplot(num_images, 2, 2*i+2)
        plt.imshow(patch)
        plt.title("Patch Adversarial")
        plt.axis('off')
    plt.tight_layout()
    plt.show()

# Run visualization
orig_loader = DataLoader(test_dataset, batch_size=1, shuffle=False)
patch_vis_loader = DataLoader(patch_dataset, batch_size=1, shuffle=False)
visualize_patch_attacks(orig_loader, patch_vis_loader, label_map, num_images=5)
```

Original: 401

Patch Adversarial

Original: 401

Patch Adversarial

Original: 401

Patch Adversarial

Original: 401

Patch Adversarial

Original: 401

Patch Adversarial

Task 5, we'll test how well our adversarial examples transfer to a different model. This evaluates whether the same perturbations can fool a model that wasn't attacked directly (a key goal in black-box adversarial attacks).

Goal: Load a different model (e.g., DenseNet-121).

Evaluate it on:

Original Test Set

Adversarial Test Set 1 (FGSM)

Adversarial Test Set 2 (PGD)

Adversarial Test Set 3 (Patch attack)

Report Top-1 and Top-5 accuracy for each dataset.

[42]:
```
import torch
import torchvision.models as models
from torch.utils.data import DataLoader, TensorDataset, Dataset
import torch.nn as nn
import pandas as pd

# === Load New Model for Transferability ===
transfer_model = models.densenet121(weights='IMAGENET1K_V1')
transfer_model.eval().to(device)
```

[42]:
```
DenseNet(
  (features): Sequential(
    (conv0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
bias=False)
    (norm0): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (relu0): ReLU(inplace=True)
    (pool0): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
ceil_mode=False)
    (denseblock1): _DenseBlock(
      (denselayer1): _DenseLayer(
        (norm1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(64, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
```

```
      )
      (denselayer2): _DenseLayer(
        (norm1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(96, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer3): _DenseLayer(
        (norm1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(128, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer4): _DenseLayer(
        (norm1): BatchNorm2d(160, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(160, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer5): _DenseLayer(
        (norm1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(192, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer6): _DenseLayer(
        (norm1): BatchNorm2d(224, eps=1e-05, momentum=0.1, affine=True,
```

```
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(224, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
    )
    (transition1): _Transition(
      (norm): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (pool): AvgPool2d(kernel_size=2, stride=2, padding=0)
    )
    (denseblock2): _DenseBlock(
      (denselayer1): _DenseLayer(
        (norm1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(128, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer2): _DenseLayer(
        (norm1): BatchNorm2d(160, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(160, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer3): _DenseLayer(
        (norm1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(192, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer4): _DenseLayer(
        (norm1): BatchNorm2d(224, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(224, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer5): _DenseLayer(
        (norm1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer6): _DenseLayer(
        (norm1): BatchNorm2d(288, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(288, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer7): _DenseLayer(
        (norm1): BatchNorm2d(320, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(320, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
```

```
      )
      (denselayer8): _DenseLayer(
        (norm1): BatchNorm2d(352, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(352, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer9): _DenseLayer(
        (norm1): BatchNorm2d(384, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(384, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer10): _DenseLayer(
        (norm1): BatchNorm2d(416, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(416, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer11): _DenseLayer(
        (norm1): BatchNorm2d(448, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(448, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer12): _DenseLayer(
        (norm1): BatchNorm2d(480, eps=1e-05, momentum=0.1, affine=True,
```

```
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(480, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
    )
    (transition2): _Transition(
      (norm): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (pool): AvgPool2d(kernel_size=2, stride=2, padding=0)
    )
    (denseblock3): _DenseBlock(
      (denselayer1): _DenseLayer(
        (norm1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer2): _DenseLayer(
        (norm1): BatchNorm2d(288, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(288, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer3): _DenseLayer(
        (norm1): BatchNorm2d(320, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(320, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer4): _DenseLayer(
        (norm1): BatchNorm2d(352, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(352, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer5): _DenseLayer(
        (norm1): BatchNorm2d(384, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(384, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer6): _DenseLayer(
        (norm1): BatchNorm2d(416, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(416, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer7): _DenseLayer(
        (norm1): BatchNorm2d(448, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(448, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
```

```
      )
      (denselayer8): _DenseLayer(
        (norm1): BatchNorm2d(480, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(480, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer9): _DenseLayer(
        (norm1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer10): _DenseLayer(
        (norm1): BatchNorm2d(544, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(544, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer11): _DenseLayer(
        (norm1): BatchNorm2d(576, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(576, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer12): _DenseLayer(
        (norm1): BatchNorm2d(608, eps=1e-05, momentum=0.1, affine=True,
```

```
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(608, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer13): _DenseLayer(
        (norm1): BatchNorm2d(640, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(640, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer14): _DenseLayer(
        (norm1): BatchNorm2d(672, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(672, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer15): _DenseLayer(
        (norm1): BatchNorm2d(704, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(704, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer16): _DenseLayer(
        (norm1): BatchNorm2d(736, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(736, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
```

```
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer17): _DenseLayer(
        (norm1): BatchNorm2d(768, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(768, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer18): _DenseLayer(
        (norm1): BatchNorm2d(800, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(800, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer19): _DenseLayer(
        (norm1): BatchNorm2d(832, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(832, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer20): _DenseLayer(
        (norm1): BatchNorm2d(864, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(864, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
```

```
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer21): _DenseLayer(
        (norm1): BatchNorm2d(896, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(896, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer22): _DenseLayer(
        (norm1): BatchNorm2d(928, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(928, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer23): _DenseLayer(
        (norm1): BatchNorm2d(960, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(960, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer24): _DenseLayer(
        (norm1): BatchNorm2d(992, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(992, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
```

```
    )
    (transition3): _Transition(
      (norm): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
      (pool): AvgPool2d(kernel_size=2, stride=2, padding=0)
    )
    (denseblock4): _DenseBlock(
      (denselayer1): _DenseLayer(
        (norm1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer2): _DenseLayer(
        (norm1): BatchNorm2d(544, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(544, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer3): _DenseLayer(
        (norm1): BatchNorm2d(576, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(576, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer4): _DenseLayer(
        (norm1): BatchNorm2d(608, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(608, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
```

```
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer5): _DenseLayer(
        (norm1): BatchNorm2d(640, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(640, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer6): _DenseLayer(
        (norm1): BatchNorm2d(672, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(672, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer7): _DenseLayer(
        (norm1): BatchNorm2d(704, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(704, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer8): _DenseLayer(
        (norm1): BatchNorm2d(736, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(736, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
```

```
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer9): _DenseLayer(
        (norm1): BatchNorm2d(768, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(768, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer10): _DenseLayer(
        (norm1): BatchNorm2d(800, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(800, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer11): _DenseLayer(
        (norm1): BatchNorm2d(832, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(832, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer12): _DenseLayer(
        (norm1): BatchNorm2d(864, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(864, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
```

```
      (denselayer13): _DenseLayer(
        (norm1): BatchNorm2d(896, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(896, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer14): _DenseLayer(
        (norm1): BatchNorm2d(928, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(928, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer15): _DenseLayer(
        (norm1): BatchNorm2d(960, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(960, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
      (denselayer16): _DenseLayer(
        (norm1): BatchNorm2d(992, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu1): ReLU(inplace=True)
        (conv1): Conv2d(992, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (norm2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
        (relu2): ReLU(inplace=True)
        (conv2): Conv2d(128, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      )
    )
    (norm5): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
  )
  (classifier): Linear(in_features=1024, out_features=1000, bias=True)
)
```

Evaluation Function (Reusable)

[43]:
```python
def accuracy(output, target, topk=(1,)):
    maxk = max(topk)
    batch_size = target.size(0)
    _, pred = output.topk(maxk, 1, True, True)
    pred = pred.t()
    correct = pred.eq(target.view(1, -1).expand_as(pred))
    return [(correct[:k].reshape(-1).float().sum(0) * 100.0 / batch_size) for k␣
 ↪in topk]

def evaluate_model(model, data_loader):
    model.eval()
    top1_total, top5_total, total_samples = 0.0, 0.0, 0
    with torch.no_grad():
        for images, labels in data_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            top1, top5 = accuracy(outputs, labels, topk=(1, 5))
            batch_size = images.size(0)
            top1_total += top1.item() * batch_size
            top5_total += top5.item() * batch_size
            total_samples += batch_size
    return top1_total / total_samples, top5_total / total_samples
```

[46]:
```python
# FGSM Attack Dataset
fgsm_images_list, fgsm_labels_list = [], []
epsilon = 0.02
loss_fn = nn.CrossEntropyLoss()

for images, labels in test_loader:
    images = images.to(device)
    labels = torch.tensor([label_map[label.item()] for label in labels]).
 ↪to(device)
    images.requires_grad = True

    outputs = model(images)
    loss = loss_fn(outputs, labels)
    model.zero_grad()
    loss.backward()

    grad = images.grad.data
    adv_images = torch.clamp(images + epsilon * grad.sign(), 0, 1)
```

```
    fgsm_images_list.append(adv_images.detach().cpu())
    fgsm_labels_list.append(labels.detach().cpu())

fgsm_images = torch.cat(fgsm_images_list, dim=0)
fgsm_labels = torch.cat(fgsm_labels_list, dim=0)
fgsm_dataset = TensorDataset(fgsm_images, fgsm_labels)
fgsm_loader = DataLoader(fgsm_dataset, batch_size=32, shuffle=False)
```

[47]:
```
# Wrap original dataset to remap labels
class WrappedDataset(Dataset):
    def __init__(self, original_dataset, label_map):
        self.dataset = original_dataset
        self.label_map = label_map

    def __getitem__(self, index):
        image, label = self.dataset[index]
        return image, label_map[label]

    def __len__(self):
        return len(self.dataset)

wrapped_test_dataset = WrappedDataset(test_dataset, label_map)
orig_loader = DataLoader(wrapped_test_dataset, batch_size=32, shuffle=False)
```

Evaluate All Four Datasets

[48]:
```
# Already available: fgsm_loader, orig_loader
# Use: pgd_dataset, patch_dataset
pgd_loader = DataLoader(pgd_dataset, batch_size=32, shuffle=False)
patch_loader = DataLoader(patch_dataset, batch_size=32, shuffle=False)

# Evaluate all
orig_top1, orig_top5 = evaluate_model(transfer_model, orig_loader)
fgsm_top1, fgsm_top5 = evaluate_model(transfer_model, fgsm_loader)
pgd_top1, pgd_top5 = evaluate_model(transfer_model, pgd_loader)
patch_top1, patch_top5 = evaluate_model(transfer_model, patch_loader)
```

Display Transferability Results

[49]:
```
results = pd.DataFrame({
    "Dataset": ["Original", "FGSM Attack", "PGD Attack", "Patch Attack"],
    "Top-1 Accuracy (%)": [orig_top1, fgsm_top1, pgd_top1, patch_top1],
    "Top-5 Accuracy (%)": [orig_top5, fgsm_top5, pgd_top5, patch_top5]
})

print("\n Task 5: Transferability Results")
print(results.to_string(index=False))
```

Task 5: Transferability Results

Dataset	Top-1 Accuracy (%)	Top-5 Accuracy (%)
Original	70.8	91.2
FGSM Attack	36.0	62.0
PGD Attack	30.8	57.2
Patch Attack	70.4	89.8

Findings & Trends

- Original accuracy on DenseNet-121 is high (70.8% Top-1, 91.2% Top-5)—similar to ResNet-34, showing both models are well-trained for ImageNet.
- Adversarial examples crafted for ResNet-34 also fool DenseNet-121, even though it was not attacked directly.

1. FGSM drops accuracy by ~50%
2. PGD is even stronger, causing ~40% drop

- Patch attacks are less transferable, with minimal drop in accuracy—likely because perturbations are localized and model-specific.

Lessons Learned

- Transferability is real: Attacks crafted for one model often fool others, especially if the models share similar architectures or training data.
- Stronger attacks (like PGD) transfer better than simpler ones (like FGSM or patch).
- Localized (patch) attacks are harder to transfer, especially when perturbation is small and not aligned across models.

Ways to Mitigate Transferability

- Adversarial training: Include adversarial examples during training to make models more robust.
- Model ensembling: Use multiple diverse architectures at inference; attacks that transfer across all are less likely.
- Input preprocessing: Apply denoising, random cropping, JPEG compression, or adversarial detection filters to reduce attack effectiveness.
- Certified defenses: Explore provable robustness techniques like randomized smoothing or certified training (though currently limited in scale).