AI VIETNAM
All-in-One Course

Exploitation of Pretrained Models

Quang-Vinh Dinh
Ph.D. in Computer Science

Year 2023

Deedy @debarghya_das · 13h

How much does it cost to train a state-of-the-art foundational LLM?

$4M.

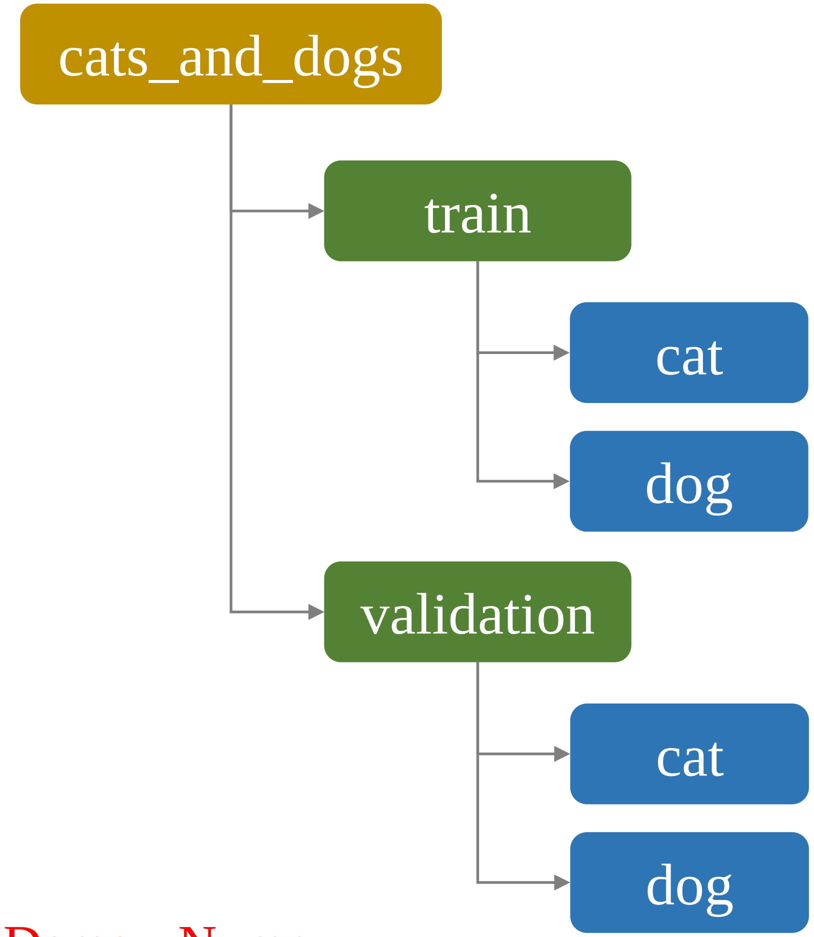
Facebook's 65B LLaMA trained for 21 days on 2048 Nvidia A100 GPUs. At $3.93/hr on GCP, that's a total of ~$4M.

Outline

- Data Processing
- Network Manipulation
- Reuse a Pre-trained Model
- Case Studies

Data Processing

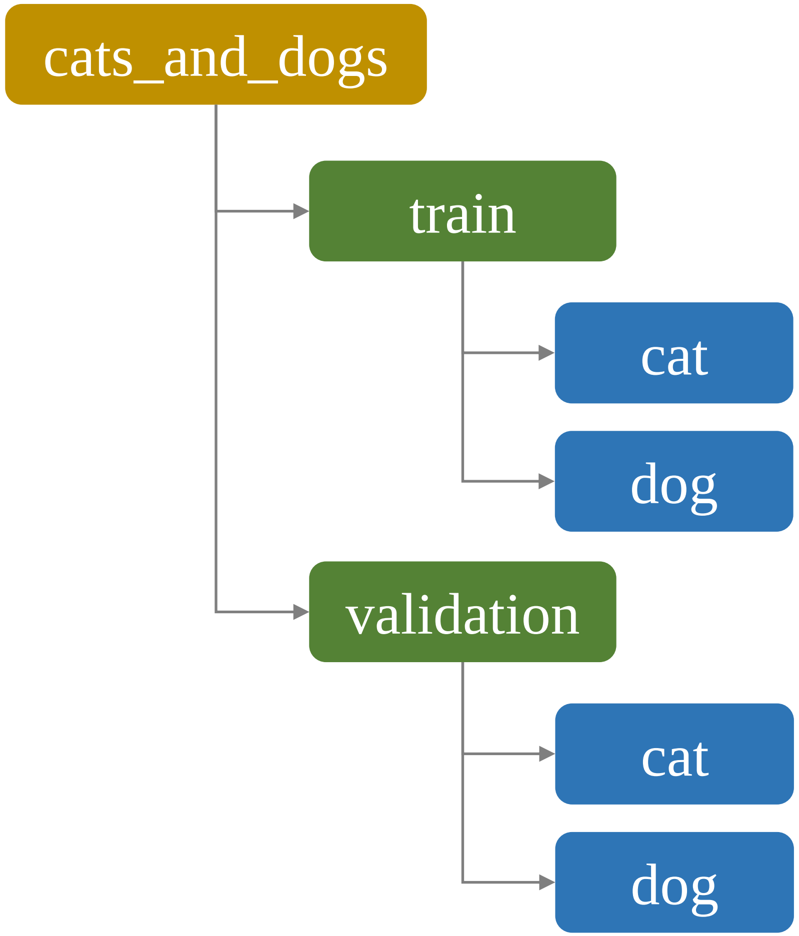
❖ Cat-Dog dataset



Demo - Numpy

Data Processing

❖ In PyTorch

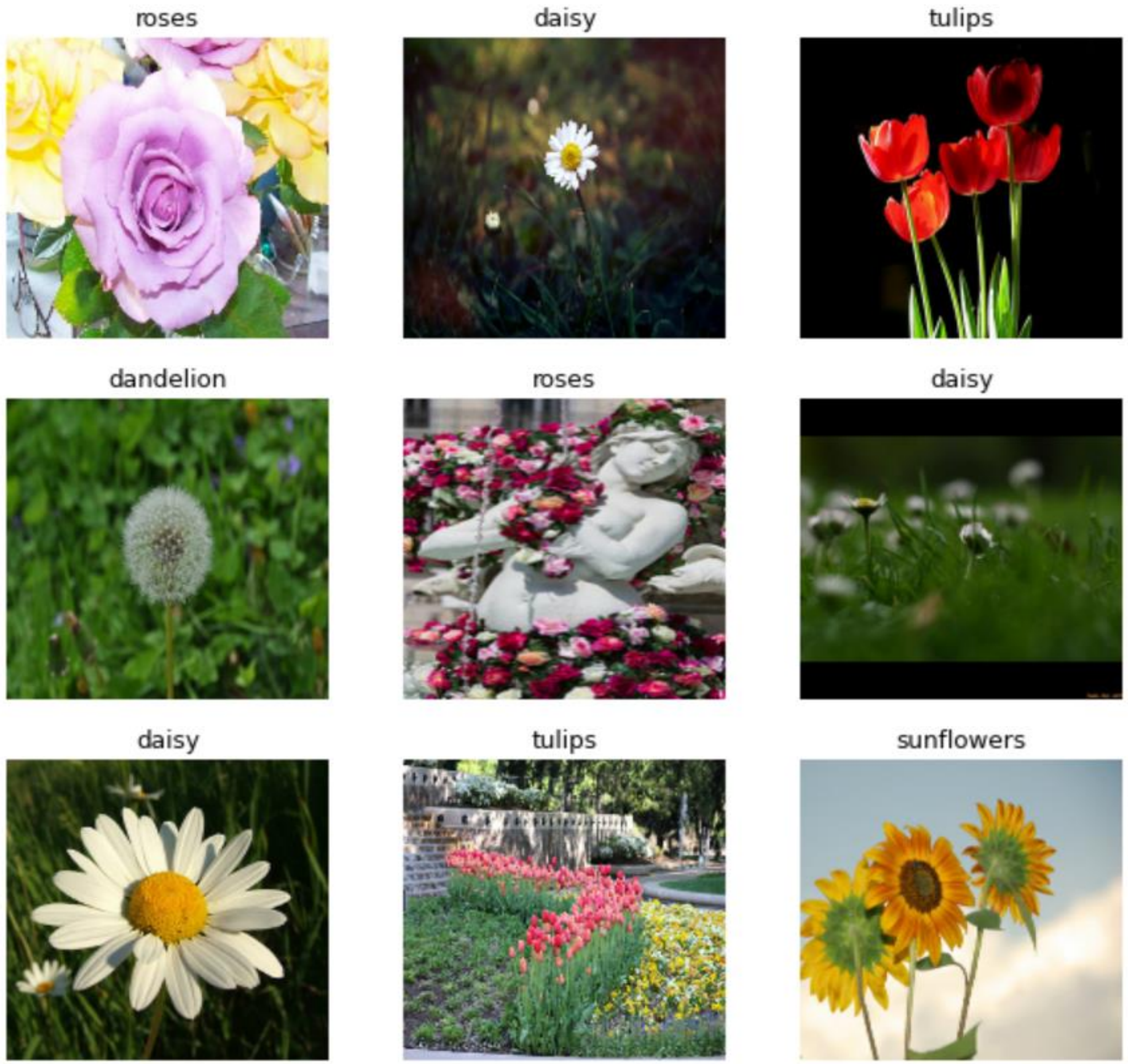



```python
from torchvision import datasets, transforms

transform = transforms.Compose(
    [
        transforms.Resize((224, 224)),
        transforms.ToTensor(),

    ])

# Load datasets
train_dataset = datasets.ImageFolder('data/train',
                                     transform=transform)
test_dataset = datasets.ImageFolder('data/validation',
                                    transform=transform)

# Create data loaders
train_loader = DataLoader(train_dataset,
                          batch_size=32,
                          shuffle=True)
test_loader = DataLoader(test_dataset,
                         batch_size=32,
                         shuffle=False)
```

Data Processing

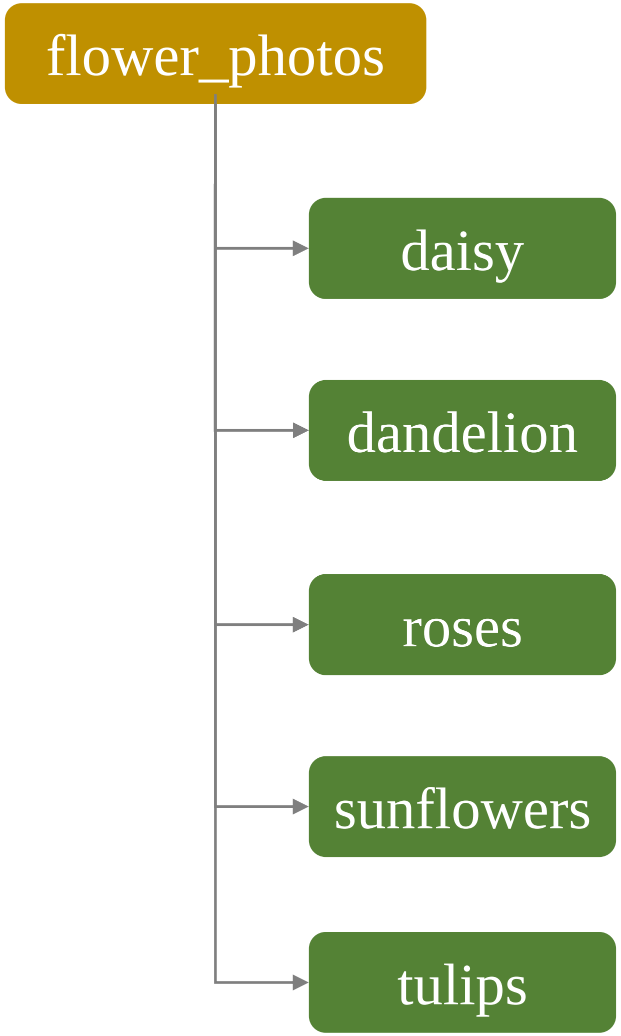
❖ In PyTorch

- flower_photos
  - daisy
  - dandelion
  - roses
  - sunflowers
  - tulips



Data Processing

❖ In PyTorch

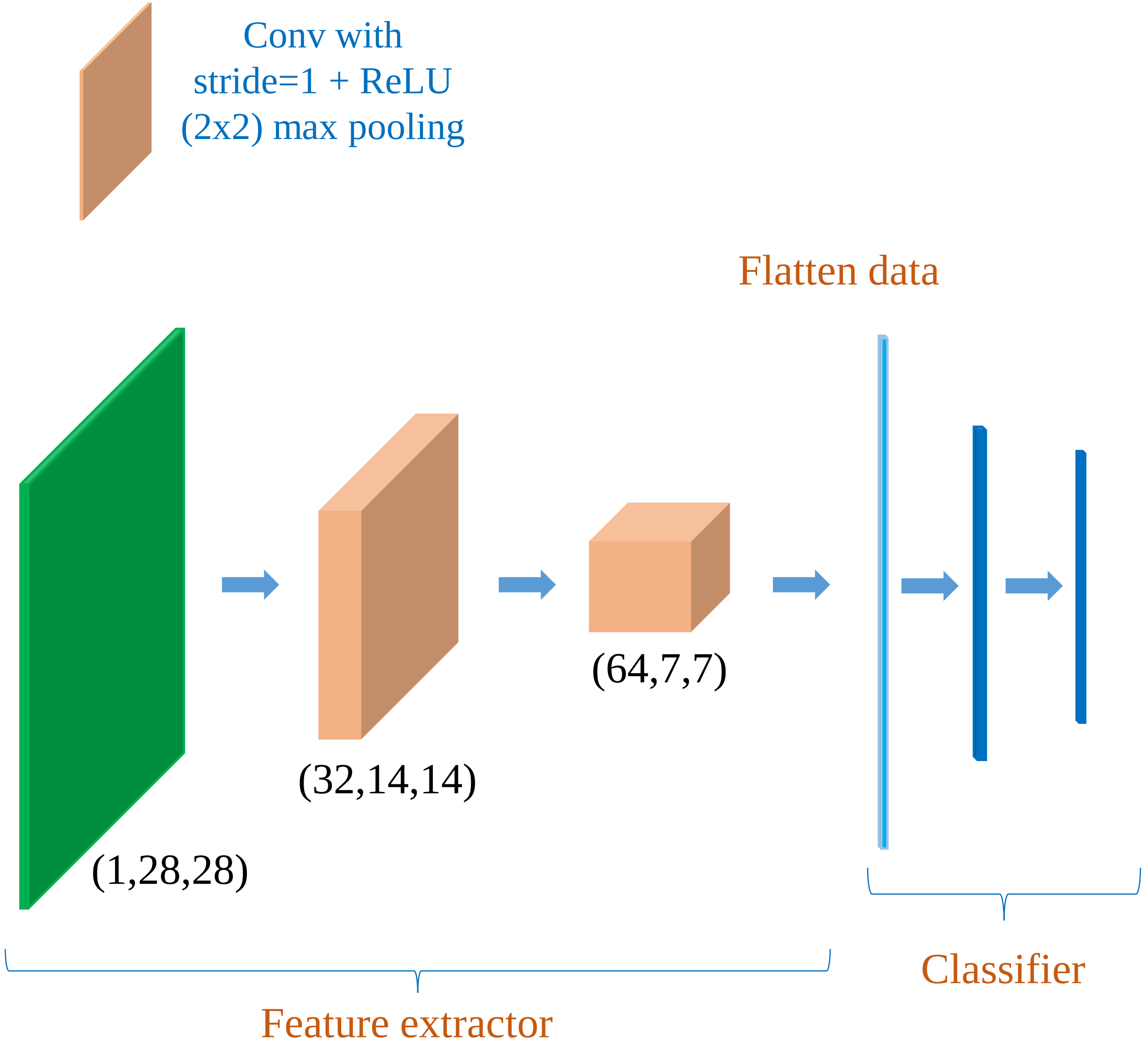



```python
from torch.utils.data import random_split

transform = transforms.Compose([
                        transforms.Resize((224, 224)),
                        transforms.ToTensor()])

# Load the dataset
dataset = ImageFolder(root='data',
                      transform=transform)

# Split the dataset
train_size = int(0.8 * len(dataset))  # 80% for training
test_size = len(dataset) - train_size  # 20% for testing
train_dataset, test_dataset = random_split(dataset,
                                           [train_size,
                                            test_size])

# Create data loaders
train_loader = DataLoader(train_dataset,
                          batch_size=32,
                          shuffle=True)
test_loader = DataLoader(test_dataset,
                         batch_size=32,
                         shuffle=False)
```

Outline

- Data Processing
- Network Manipulation
- Reuse a Pre-trained Model
- Case Studies

Network Manipulation




```python
class SimpleCNN(nn.Module):
    def __init__(self, num_classes=10):
        super(SimpleCNN, self).__init__()
        # Convolutional layers
        self.features = nn.Sequential(
            # First block
            nn.Conv2d(1, 32, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),

            # Second block
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
        )

        # Fully connected layers
        self.classifier = nn.Sequential(
            nn.Linear(64 * 7 * 7, 128),
            nn.ReLU(inplace=True),
            nn.Linear(128, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
        x = torch.flatten(x, 1)s
        x = self.classifier(x)
        return x
```

Network Manipulation

```python
from torchsummary import summary
summary(model, (1, 28, 28))
```

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
├─Sequential: 1-1                        [-1, 64, 7, 7]            --
|    └─Conv2d: 2-1                       [-1, 32, 28, 28]          320
|    └─ReLU: 2-2                         [-1, 32, 28, 28]          --
|    └─MaxPool2d: 2-3                    [-1, 32, 14, 14]          --
|    └─Conv2d: 2-4                       [-1, 64, 14, 14]          18,496
|    └─ReLU: 2-5                         [-1, 64, 14, 14]          --
|    └─MaxPool2d: 2-6                    [-1, 64, 7, 7]            --
├─Sequential: 1-2                        [-1, 10]                  --
|    └─Linear: 2-7                       [-1, 128]                 401,536
|    └─ReLU: 2-8                         [-1, 128]                 --
|    └─Linear: 2-9                       [-1, 10]                  1,290
==========================================================================================
Total params: 421,642
Trainable params: 421,642
Non-trainable params: 0
Total mult-adds (M): 4.66
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.29
Params size (MB): 1.61
Estimated Total Size (MB): 1.90
==========================================================================================
```

Network Manipulation

Check if a layer is trainable

```
for name, module in model.named_modules():
    if hasattr(module, 'parameters'):
        is_trainable = any(param.requires_grad
                           for param in module.parameters())
        print(f"{name}: {'Trainable' if is_trainable
                          else 'Not trainable'}")
```

```
: Trainable
features: Trainable
features.0: Trainable
features.1: Not trainable
features.2: Not trainable
features.3: Trainable
features.4: Not trainable
features.5: Not trainable
classifier: Trainable
classifier.0: Trainable
classifier.1: Not trainable
classifier.2: Trainable
```

```
class SimpleCNN(nn.Module):
    def __init__(self, num_classes=10):
        super(SimpleCNN, self).__init__()
        # Convolutional layers
        self.features = nn.Sequential(
            # First block
            nn.Conv2d(1, 32, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),

            # Second block
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
        )

        # Fully connected layers
        self.classifier = nn.Sequential(
            nn.Linear(64 * 7 * 7, 128),
            nn.ReLU(inplace=True),
            nn.Linear(128, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
        x = torch.flatten(x, 1)s
        x = self.classifier(x)
        return x
```

Network Manipulation

Check if a layer is trainable

```
for name, module in model.features.named_modules():
    if hasattr(module, 'parameters'):
        is_trainable = any(param.requires_grad
                           for param in module.parameters())
        print(f"{name}: {'Trainable' if is_trainable
                                     else 'Not trainable'}")
```

```
: Trainable
0: Trainable
1: Not trainable
2: Not trainable
3: Trainable
4: Not trainable
5: Not trainable
```

```
for name, module in model.classifier.named_modules():
    if hasattr(module, 'parameters'):
        is_trainable = any(param.requires_grad
                           for param in module.parameters())
        print(f"{name}: {'Trainable' if is_trainable
                                     else 'Not trainable'}")
```

```
: Trainable
0: Trainable
1: Not trainable
2: Trainable
```

```
class SimpleCNN(nn.Module):
    def __init__(self, num_classes=10):
        super(SimpleCNN, self).__init__()
        # Convolutional layers
        self.features = nn.Sequential(
            # First block
            nn.Conv2d(1, 32, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),

            # Second block
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
        )

        # Fully connected layers
        self.classifier = nn.Sequential(
            nn.Linear(64 * 7 * 7, 128),
            nn.ReLU(inplace=True),
            nn.Linear(128, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
        x = torch.flatten(x, 1)s
        x = self.classifier(x)
        return x
```

Network Manipulation

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

```
from torchsummary import summary
summary(model, (1, 28, 28))
```

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
├─Sequential: 1-1                        [-1, 64, 7, 7]            --
|    └─Conv2d: 2-1                       [-1, 32, 28, 28]          (320)
|    └─ReLU: 2-2                         [-1, 32, 28, 28]          --
|    └─MaxPool2d: 2-3                    [-1, 32, 14, 14]          --
|    └─Conv2d: 2-4                       [-1, 64, 14, 14]          (18,496)
|    └─ReLU: 2-5                         [-1, 64, 14, 14]          --
|    └─MaxPool2d: 2-6                    [-1, 64, 7, 7]            --
├─Sequential: 1-2                        [-1, 10]                  --
|    └─Linear: 2-7                       [-1, 128]                 (401,536)
|    └─ReLU: 2-8                         [-1, 128]                 --
|    └─Linear: 2-9                       [-1, 10]                  (1,290)
==========================================================================================
Total params: 421,642
Trainable params: 0
Non-trainable params: 421,642
Total mult-adds (M): 4.66
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.29
Params size (MB): 1.61
Estimated Total Size (MB): 1.90
==========================================================================================
```

Network Manipulation

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

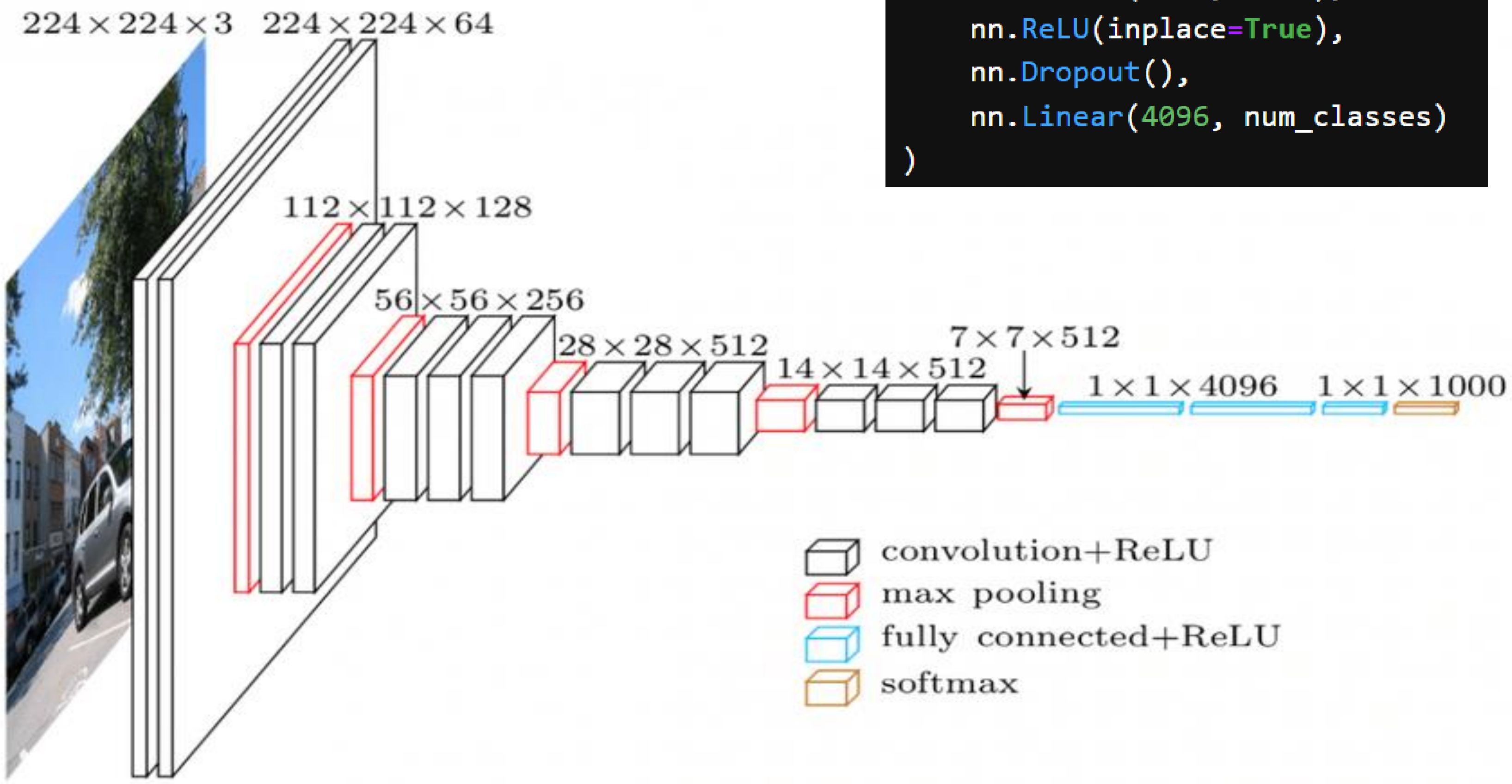
```python
from torchsummary import summary
summary(model, (1, 28, 28))
```

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
├─Sequential: 1-1                        [-1, 64, 7, 7]            --
|    └─Conv2d: 2-1                       [-1, 32, 28, 28]          (320)
|    └─ReLU: 2-2                         [-1, 32, 28, 28]          --
|    └─MaxPool2d: 2-3                    [-1, 32, 14, 14]          --
|    └─Conv2d: 2-4                       [-1, 64, 14, 14]          (18,496)
|    └─ReLU: 2-5                         [-1, 64, 14, 14]          --
|    └─MaxPool2d: 2-6                    [-1, 64, 7, 7]            --
├─Sequential: 1-2                        [-1, 10]                  --
|    └─Linear: 2-7                       [-1, 128]                 401,536
|    └─ReLU: 2-8                         [-1, 128]                 --
|    └─Linear: 2-9                       [-1, 10]                  1,290
==========================================================================================
Total params: 421,642
Trainable params: 402,826
Non-trainable params: 18,816
Total mult-adds (M): 4.66
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.29
Params size (MB): 1.61
Estimated Total Size (MB): 1.90
==========================================================================================
```

Network Manipulation

❖ VGG16 Model

```
# Fully connected layers
self.classifier = nn.Sequential(
    nn.Linear(512 * 7 * 7, 4096),
    nn.ReLU(inplace=True),
    nn.Dropout(),
    nn.Linear(4096, 4096),
    nn.ReLU(inplace=True),
    nn.Dropout(),
    nn.Linear(4096, num_classes)
)
```




```
self.features = nn.Sequential(
    # First block
    nn.Conv2d(3, 64, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),
    nn.Conv2d(64, 64, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=2, stride=2),
    # Second block
    nn.Conv2d(64, 128, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),
    nn.Conv2d(128, 128, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=2, stride=2),
    # Third block
    nn.Conv2d(128, 256, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),
    nn.Conv2d(256, 256, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),
    nn.Conv2d(256, 256, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=2, stride=2),
    # Fourth block
    nn.Conv2d(256, 512, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),
    nn.Conv2d(512, 512, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),
    nn.Conv2d(512, 512, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=2, stride=2),
    # Fifth block
    nn.Conv2d(512, 512, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),
    nn.Conv2d(512, 512, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),
    nn.Conv2d(512, 512, kernel_size=3, padding=1),
    nn.ReLU(inplace=True),
    nn.MaxPool2d(kernel_size=2, stride=2))
```

```
==========================================================
Layer (type:depth-idx)                   Output Shape
==========================================================
├─Conv2d: 1-1                            [-1, 64, 224, 224]
├─ReLU: 1-2                              [-1, 64, 224, 224]
├─Conv2d: 1-3                            [-1, 64, 224, 224]
├─ReLU: 1-4                              [-1, 64, 224, 224]
├─MaxPool2d: 1-5                         [-1, 64, 112, 112]
├─Conv2d: 1-6                            [-1, 128, 112, 112]
├─ReLU: 1-7                              [-1, 128, 112, 112]
├─Conv2d: 1-8                            [-1, 128, 112, 112]
├─ReLU: 1-9                              [-1, 128, 112, 112]
├─MaxPool2d: 1-10                        [-1, 128, 56, 56]
├─Conv2d: 1-11                           [-1, 256, 56, 56]
├─ReLU: 1-12                             [-1, 256, 56, 56]
├─Conv2d: 1-13                           [-1, 256, 56, 56]
├─ReLU: 1-14                             [-1, 256, 56, 56]
├─Conv2d: 1-15                           [-1, 256, 56, 56]
├─ReLU: 1-16                             [-1, 256, 56, 56]
├─MaxPool2d: 1-17                        [-1, 256, 28, 28]
├─Conv2d: 1-18                           [-1, 512, 28, 28]
├─ReLU: 1-19                             [-1, 512, 28, 28]
├─Conv2d: 1-20                           [-1, 512, 28, 28]
├─ReLU: 1-21                             [-1, 512, 28, 28]
├─Conv2d: 1-22                           [-1, 512, 28, 28]
├─ReLU: 1-23                             [-1, 512, 28, 28]
├─MaxPool2d: 1-24                        [-1, 512, 14, 14]
├─Conv2d: 1-25                           [-1, 512, 14, 14]
├─ReLU: 1-26                             [-1, 512, 14, 14]
├─Conv2d: 1-27                           [-1, 512, 14, 14]
├─ReLU: 1-28                             [-1, 512, 14, 14]
├─Conv2d: 1-29                           [-1, 512, 14, 14]
├─ReLU: 1-30                             [-1, 512, 14, 14]
├─MaxPool2d: 1-31                        [-1, 512, 7, 7]
```

Network Manipulation

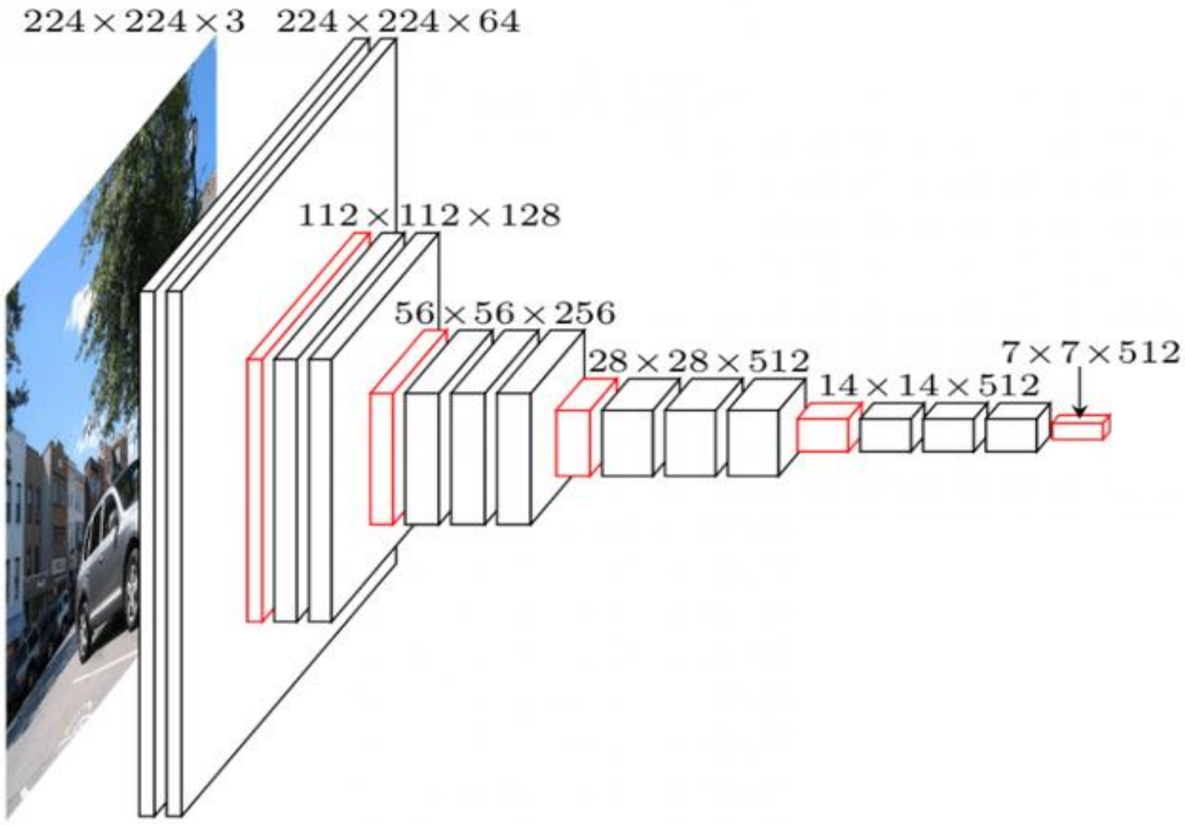
❖ VGG16 Model

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

vgg16 = models.vgg16()
summary(vgg16, (3, 224, 224))
```

```
├─AdaptiveAvgPool2d: 1-2                 [-1, 512, 7, 7]
├─Sequential: 1-3                        [-1, 1000]
|    └─Linear: 2-32                      [-1, 4096]
|    └─ReLU: 2-33                        [-1, 4096]
|    └─Dropout: 2-34                     [-1, 4096]
|    └─Linear: 2-35                      [-1, 4096]
|    └─ReLU: 2-36                        [-1, 4096]
|    └─Dropout: 2-37                     [-1, 4096]
|    └─Linear: 2-38                      [-1, 1000]
==========================================================
```

Network Manipulation

❖ VGG16 Model: Feature extractor



Demo

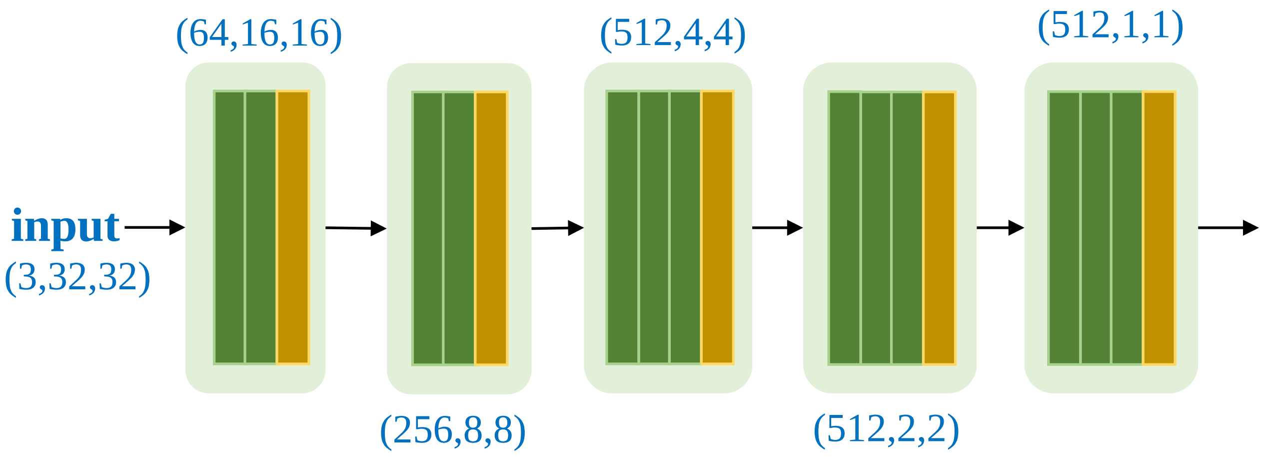
```
summary(vgg16.features, (3, 224, 224))
```

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
├─Conv2d: 1-1                            [-1, 64, 224, 224]        1,792
├─ReLU: 1-2                              [-1, 64, 224, 224]        --
├─Conv2d: 1-3                            [-1, 64, 224, 224]        36,928
├─ReLU: 1-4                              [-1, 64, 224, 224]        --
├─MaxPool2d: 1-5                         [-1, 64, 112, 112]        --
├─Conv2d: 1-6                            [-1, 128, 112, 112]       73,856
├─ReLU: 1-7                              [-1, 128, 112, 112]       --
├─Conv2d: 1-8                            [-1, 128, 112, 112]       147,584
├─ReLU: 1-9                              [-1, 128, 112, 112]       --
├─MaxPool2d: 1-10                        [-1, 128, 56, 56]         --
├─Conv2d: 1-11                           [-1, 256, 56, 56]         295,168
├─ReLU: 1-12                             [-1, 256, 56, 56]         --
├─Conv2d: 1-13                           [-1, 256, 56, 56]         590,080
├─ReLU: 1-14                             [-1, 256, 56, 56]         --
├─Conv2d: 1-15                           [-1, 256, 56, 56]         590,080
├─ReLU: 1-16                             [-1, 256, 56, 56]         --
├─MaxPool2d: 1-17                        [-1, 256, 28, 28]         --
├─Conv2d: 1-18                           [-1, 512, 28, 28]         1,180,160
├─ReLU: 1-19                             [-1, 512, 28, 28]         --
├─Conv2d: 1-20                           [-1, 512, 28, 28]         2,359,808
├─ReLU: 1-21                             [-1, 512, 28, 28]         --
├─Conv2d: 1-22                           [-1, 512, 28, 28]         2,359,808
├─ReLU: 1-23                             [-1, 512, 28, 28]         --
├─MaxPool2d: 1-24                        [-1, 512, 14, 14]         --
├─Conv2d: 1-25                           [-1, 512, 14, 14]         2,359,808
├─ReLU: 1-26                             [-1, 512, 14, 14]         --
├─Conv2d: 1-27                           [-1, 512, 14, 14]         2,359,808
├─ReLU: 1-28                             [-1, 512, 14, 14]         --
├─Conv2d: 1-29                           [-1, 512, 14, 14]         2,359,808
├─ReLU: 1-30                             [-1, 512, 14, 14]         --
├─MaxPool2d: 1-31                        [-1, 512, 7, 7]           --
==========================================================================================
```

Network Manipulation

❖ VGG16 Model: Feature extractor

```python
import torch.nn as nn
import torchvision.models as models

vgg16 = models.vgg16()
f_extractor = vgg16.features
```




```python
summary(vgg16.features, (3, 32, 32))
```

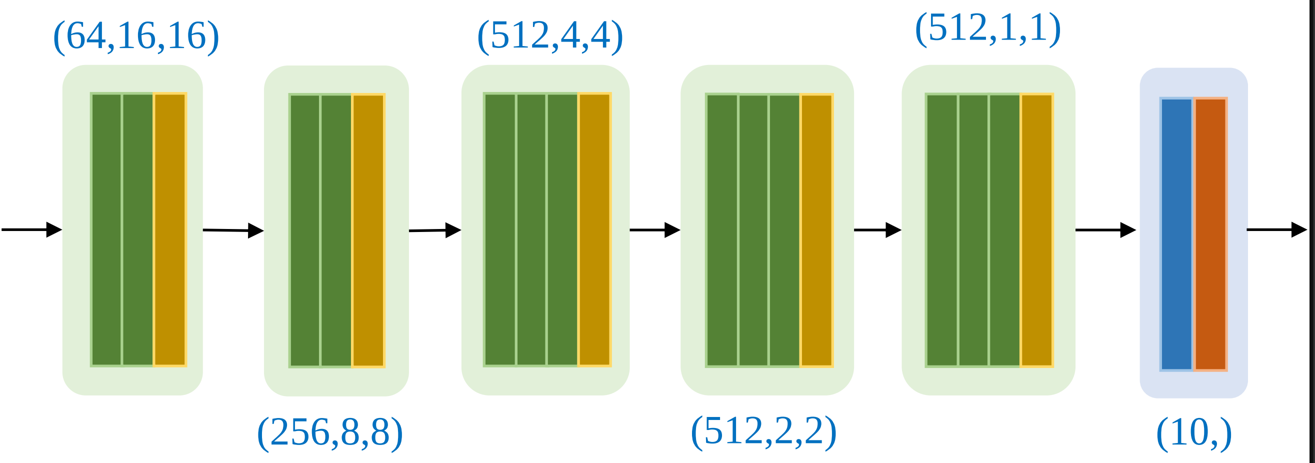
```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Para
==========================================================================================
├─Conv2d: 1-1                            [-1, 64, 32, 32]          1,79
├─ReLU: 1-2                              [-1, 64, 32, 32]          --
├─Conv2d: 1-3                            [-1, 64, 32, 32]          36,9
├─ReLU: 1-4                              [-1, 64, 32, 32]          --
├─MaxPool2d: 1-5                         [-1, 64, 16, 16]          --
├─Conv2d: 1-6                            [-1, 128, 16, 16]         73,8
├─ReLU: 1-7                              [-1, 128, 16, 16]         --
├─Conv2d: 1-8                            [-1, 128, 16, 16]         147,
├─ReLU: 1-9                              [-1, 128, 16, 16]         --
├─MaxPool2d: 1-10                        [-1, 128, 8, 8]           --
├─Conv2d: 1-11                           [-1, 256, 8, 8]           295,
├─ReLU: 1-12                             [-1, 256, 8, 8]           --
├─Conv2d: 1-13                           [-1, 256, 8, 8]           590,
├─ReLU: 1-14                             [-1, 256, 8, 8]           --
├─Conv2d: 1-15                           [-1, 256, 8, 8]           590,
├─ReLU: 1-16                             [-1, 256, 8, 8]           --
├─MaxPool2d: 1-17                        [-1, 256, 4, 4]           --
├─Conv2d: 1-18                           [-1, 512, 4, 4]           1,18
├─ReLU: 1-19                             [-1, 512, 4, 4]           --
├─Conv2d: 1-20                           [-1, 512, 4, 4]           2,35
├─ReLU: 1-21                             [-1, 512, 4, 4]           --
├─Conv2d: 1-22                           [-1, 512, 4, 4]           2,35
├─ReLU: 1-23                             [-1, 512, 4, 4]           --
├─MaxPool2d: 1-24                        [-1, 512, 2, 2]           --
├─Conv2d: 1-25                           [-1, 512, 2, 2]           2,35
├─ReLU: 1-26                             [-1, 512, 2, 2]           --
├─Conv2d: 1-27                           [-1, 512, 2, 2]           2,35
├─ReLU: 1-28                             [-1, 512, 2, 2]           --
├─Conv2d: 1-29                           [-1, 512, 2, 2]           2,35
├─ReLU: 1-30                             [-1, 512, 2, 2]           --
├─MaxPool2d: 1-31                        [-1, 512, 1, 1]           --
==========================================================================================
```

Network Manipulation

❖ **Create a new model from the VGG16 feature extractor**

```python
import torch.nn as nn
import torchvision.models as models

vgg16 = models.vgg16()
f_extractor = vgg16.features

model = nn.Sequential(f_extractor,
                      nn.Flatten(),
                      nn.Linear(512*1*1, 10))
summary(model, (3, 32, 32))
```

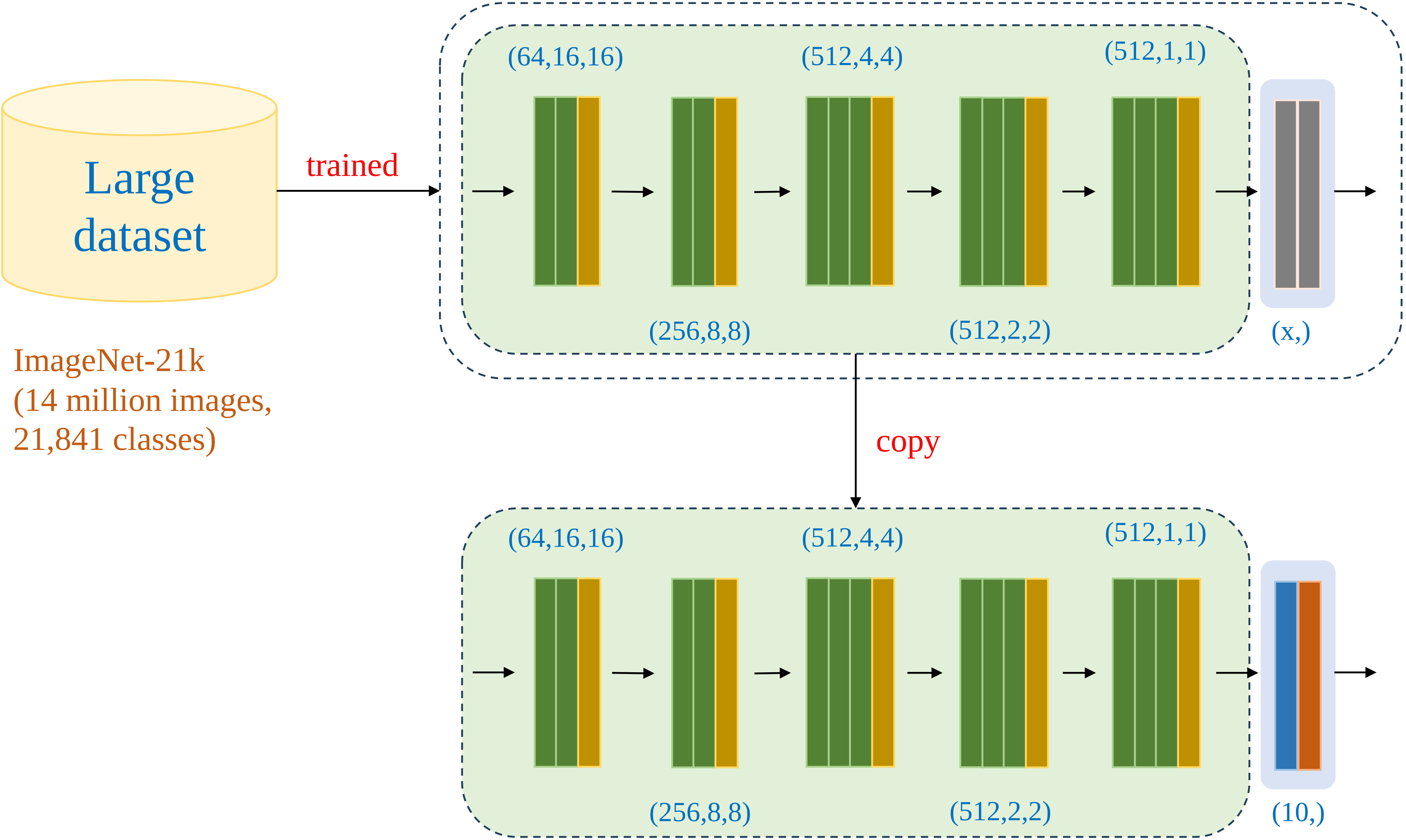
why doing so?




```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
├─Sequential: 1-1                        [-1, 512, 1, 1]           --
|    └─Conv2d: 2-1                       [-1, 64, 32, 32]          1,792
|    └─ReLU: 2-2                         [-1, 64, 32, 32]          --
|    └─Conv2d: 2-3                       [-1, 64, 32, 32]          36,928
|    └─ReLU: 2-4                         [-1, 64, 32, 32]          --
|    └─MaxPool2d: 2-5                    [-1, 64, 16, 16]          --
|    └─Conv2d: 2-6                       [-1, 128, 16, 16]         73,856
|    └─ReLU: 2-7                         [-1, 128, 16, 16]         --
|    └─Conv2d: 2-8                       [-1, 128, 16, 16]         147,584
|    └─ReLU: 2-9                         [-1, 128, 16, 16]         --
|    └─MaxPool2d: 2-10                   [-1, 128, 8, 8]           --
|    └─Conv2d: 2-11                      [-1, 256, 8, 8]           295,168
|    └─ReLU: 2-12                        [-1, 256, 8, 8]           --
|    └─Conv2d: 2-13                      [-1, 256, 8, 8]           590,080
|    └─ReLU: 2-14                        [-1, 256, 8, 8]           --
|    └─Conv2d: 2-15                      [-1, 256, 8, 8]           590,080
|    └─ReLU: 2-16                        [-1, 256, 8, 8]           --
|    └─MaxPool2d: 2-17                   [-1, 256, 4, 4]           --
|    └─Conv2d: 2-18                      [-1, 512, 4, 4]           1,180,160
|    └─ReLU: 2-19                        [-1, 512, 4, 4]           --
|    └─Conv2d: 2-20                      [-1, 512, 4, 4]           2,359,808
|    └─ReLU: 2-21                        [-1, 512, 4, 4]           --
|    └─Conv2d: 2-22                      [-1, 512, 4, 4]           2,359,808
|    └─ReLU: 2-23                        [-1, 512, 4, 4]           --
|    └─MaxPool2d: 2-24                   [-1, 512, 2, 2]           --
|    └─Conv2d: 2-25                      [-1, 512, 2, 2]           2,359,808
|    └─ReLU: 2-26                        [-1, 512, 2, 2]           --
|    └─Conv2d: 2-27                      [-1, 512, 2, 2]           2,359,808
|    └─ReLU: 2-28                        [-1, 512, 2, 2]           --
|    └─Conv2d: 2-29                      [-1, 512, 2, 2]           2,359,808
|    └─ReLU: 2-30                        [-1, 512, 2, 2]           --
|    └─MaxPool2d: 2-31                   [-1, 512, 1, 1]           --
├─Flatten: 1-2                           [-1, 512]                 --
├─Linear: 1-3                            [-1, 10]                  5,130
==========================================================================================
```

Network Manipulation

- Why?



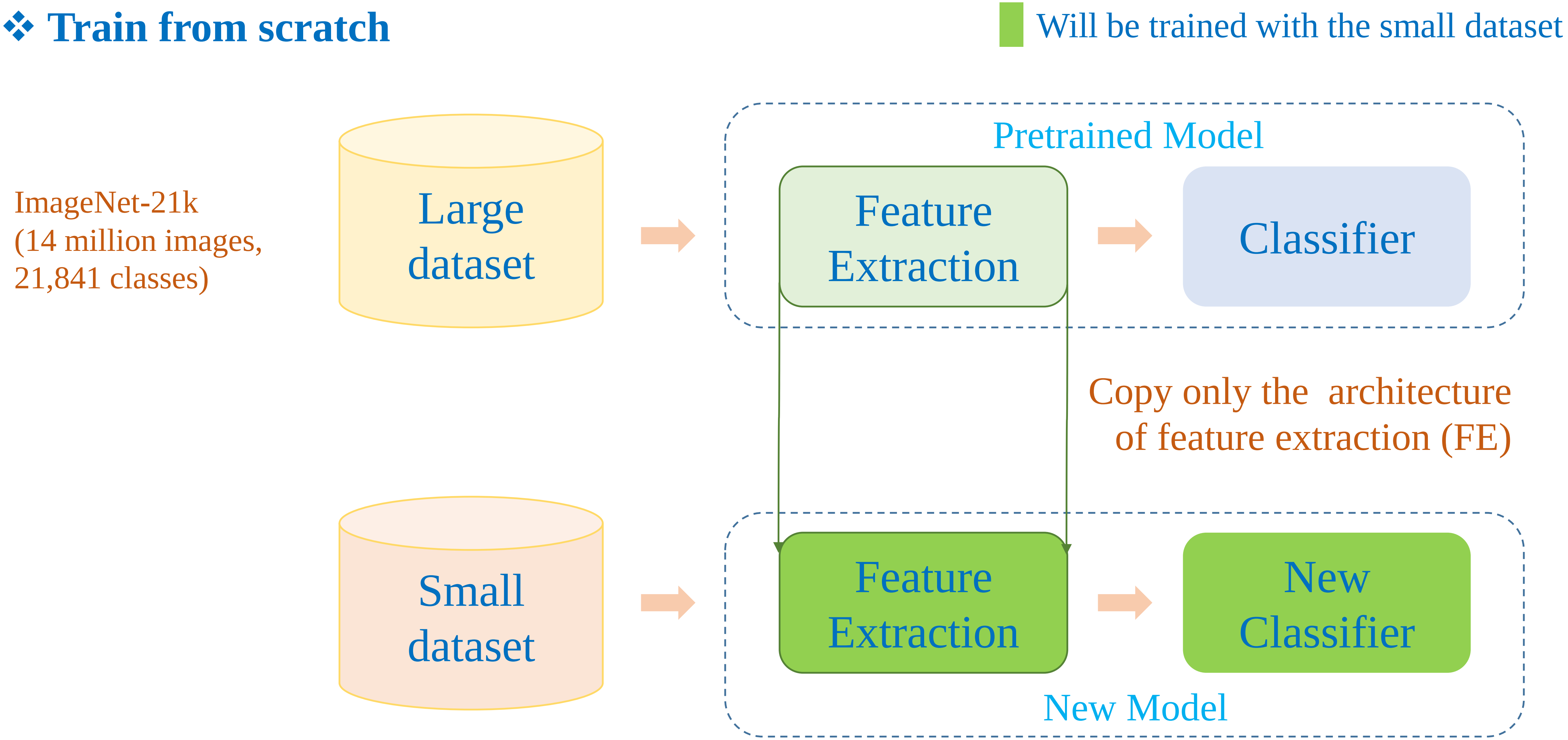
ImageNet-21k
(14 million images,
21,841 classes)

Outline

- Data Processing
- Network Manipulation
- Reuse a Pre-trained Model
- Case Studies

Exploitation of Pretrained Models

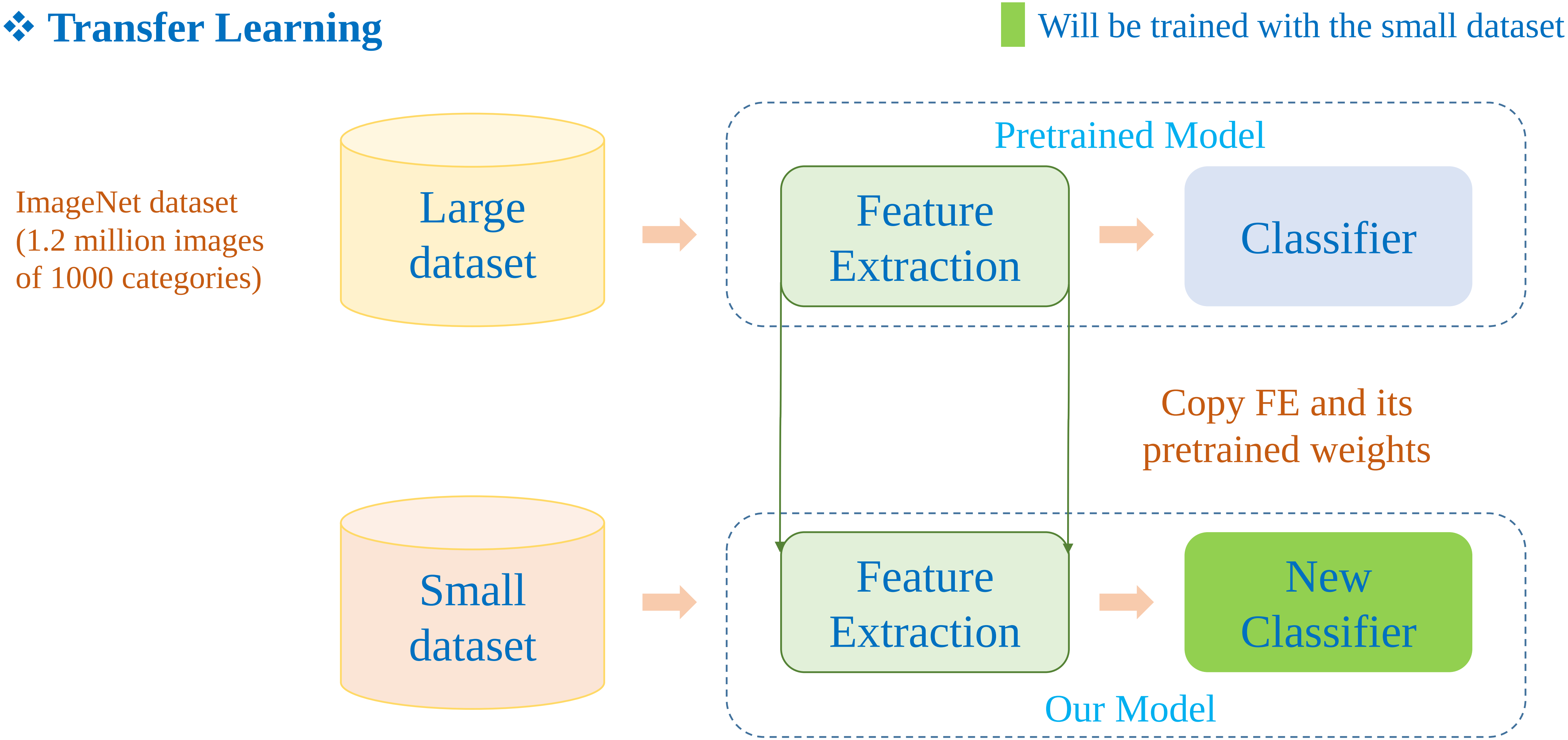
- **Train from scratch**



Initialize and train the whole new model using a small dataset

Exploitation of Pretrained Models

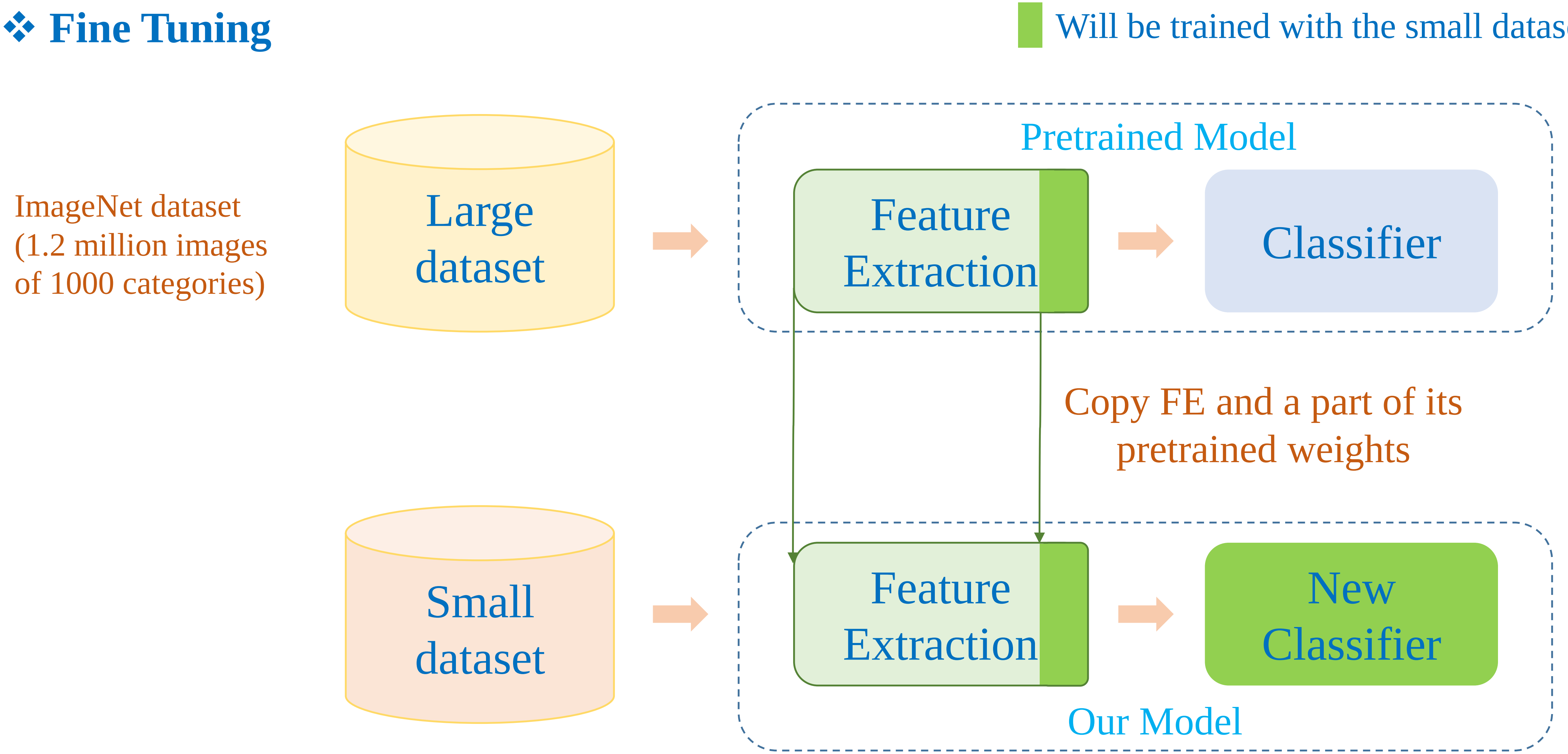
❖ Transfer Learning



Freeze FE and train only new classifier

Exploitation of Pretrained Models

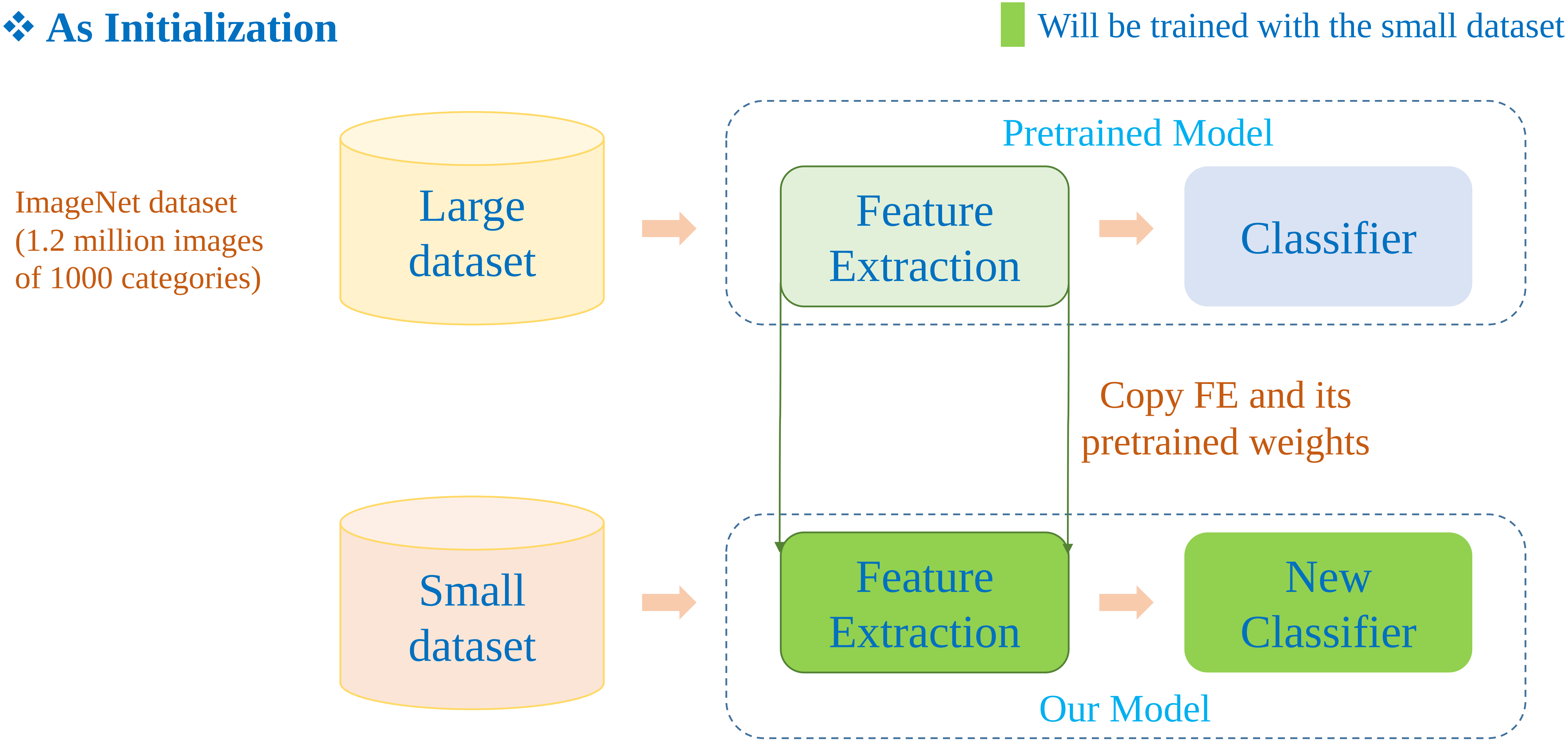
❖ **Fine Tuning**



Freeze a part of FE and train the rest and new classifier

Exploitation of Pretrained Models

❖ As Initialization



Train the whole new model using a small dataset

Outline

- Data Processing
- Network Manipulation
- Reuse a Pre-trained Model
- Case Studies

1

Case Study 1

- ❖ **Cat-Dog dataset**
  - ❖ **Train from scratch**

- cats_and_dogs
  - train
    - cat
    - dog
  - validation
    - cat
    - dog

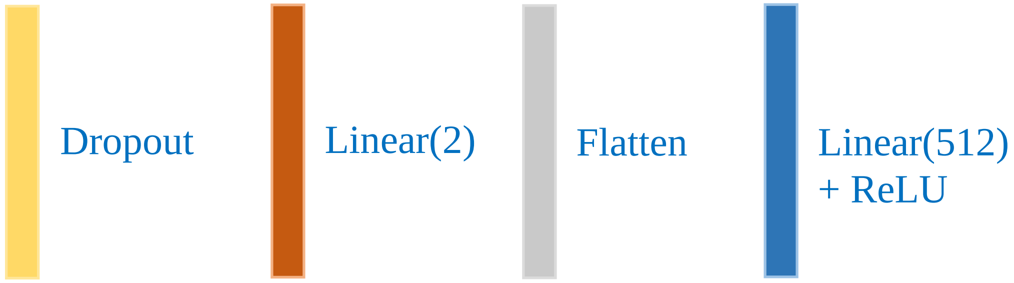

```
train_transform = transforms.Compose(
    [
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                             std=[0.229, 0.224, 0.225]),
        transforms.RandomErasing(p=0.75,
                                 scale=(0.01, 0.3),
                                 ratio=(1.0, 1.0),
                                 value=0,
                                 inplace =True)
    ])
test_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])
])

# Load datasets
train_dataset = datasets.ImageFolder('data1000/train',
                                     transform=train_transform)
test_dataset = datasets.ImageFolder('data1000/validation',
                                    transform=test_transform)

# Create data loaders
train_loader = DataLoader(train_dataset,
                          batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset,
                         batch_size=32, shuffle=False)
```

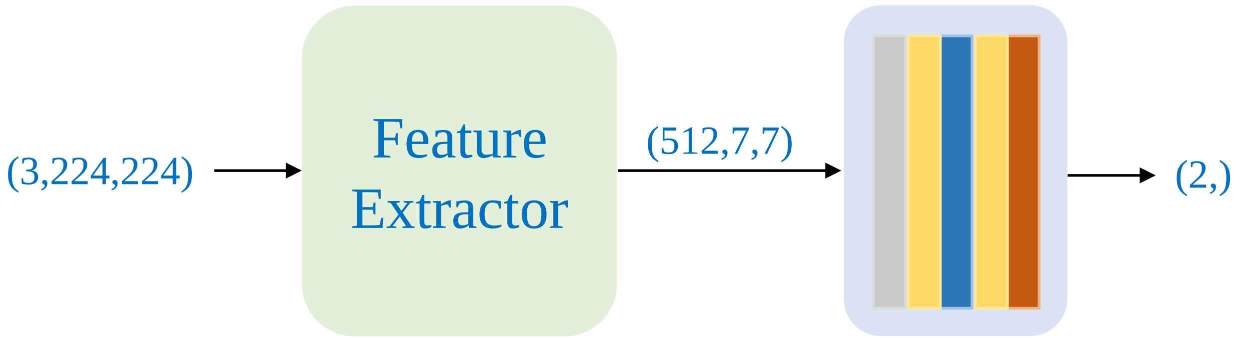
Case Study 1

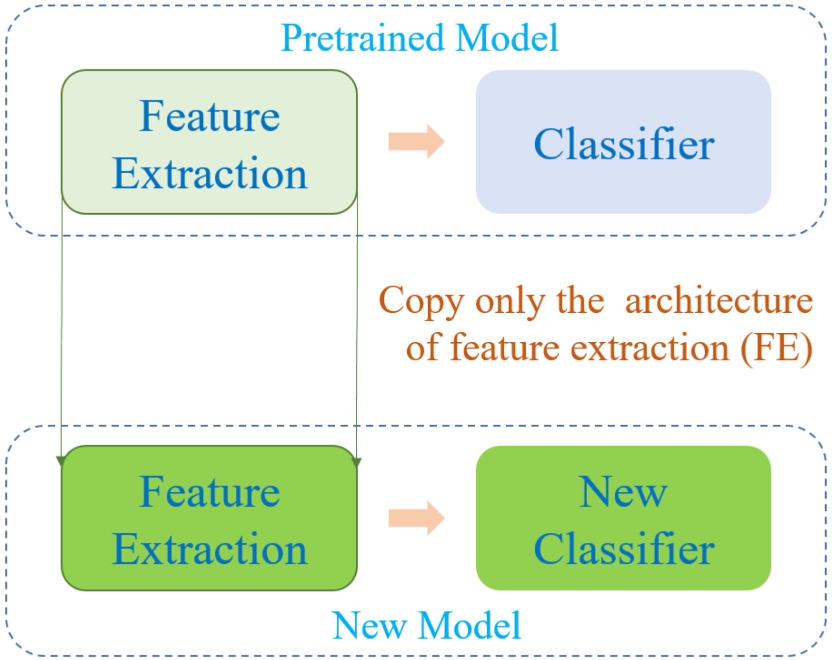
❖ Cat-Dog dataset

❖ Train from scratch

```
# Load the VGG16 model
vgg16 = models.vgg16()
f_extractor = vgg16.features

model = nn.Sequential(f_extractor,
                      nn.Flatten(),
                      nn.Dropout(0.3),
                      nn.Linear(512*7*7, 512),
                      nn.ReLU(),
                      nn.Dropout(0.3),
                      nn.Linear(512, 2))
```



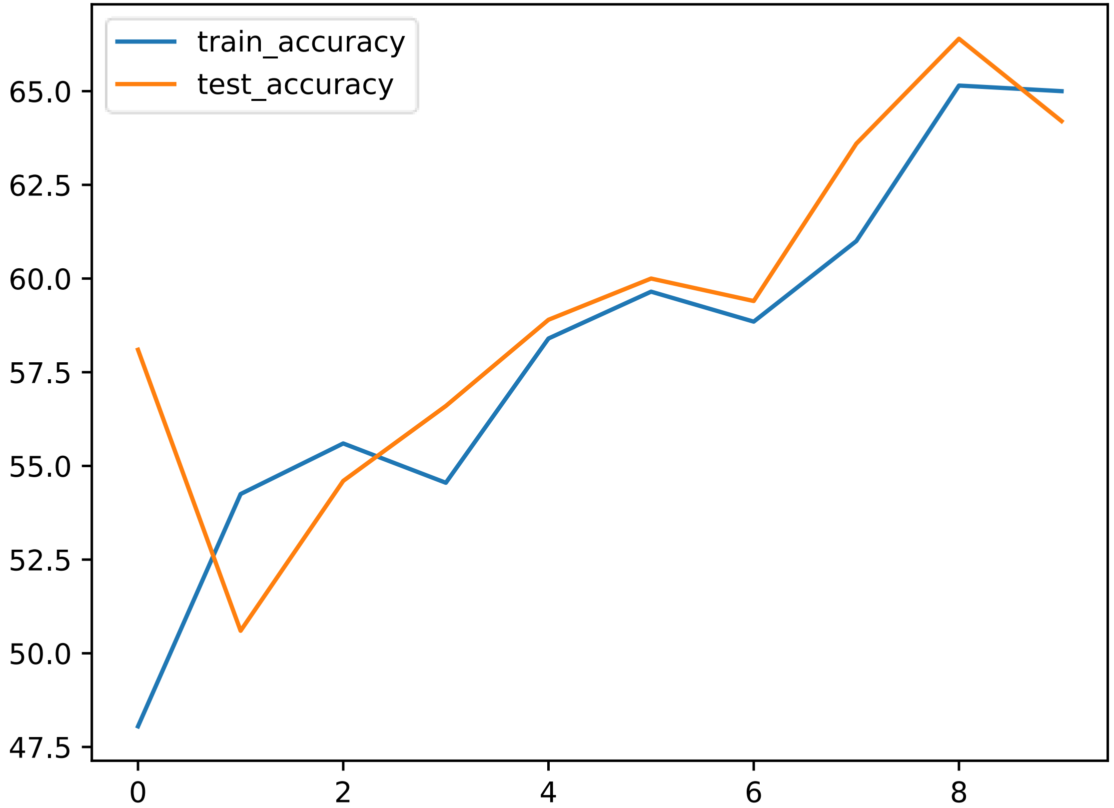




Case Study 1

❖ Cat-Dog dataset

❖ Train from scratch



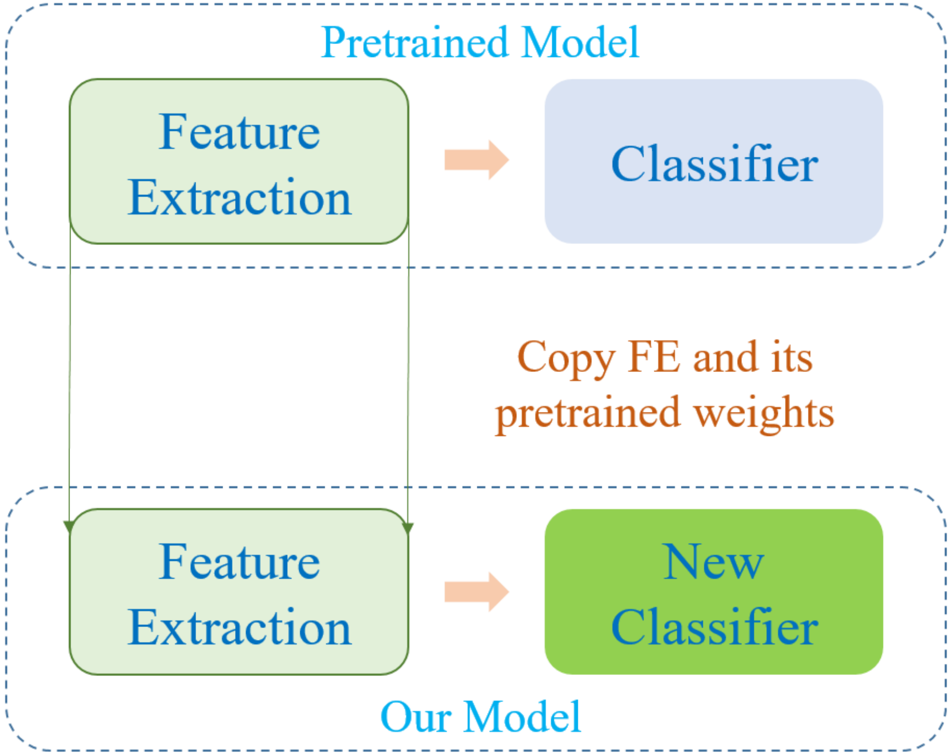
Training accuracy: 65%   Test accuracy: 66%

```python
# train
for epoch in range(max_epoch):
    model.train()

    for i, (inputs, labels) in enumerate(train_loader, 0):
        # Move inputs and labels to the device
        inputs, labels = inputs.to(device), labels.to(device)

        # Zero the parameter gradients
        optimizer.zero_grad()

        # Forward pass
        outputs = model(inputs)
        loss = criterion(outputs, labels)

        # Backward pass and optimization
        loss.backward()
        optimizer.step()

    # evaluate
    test_loss, test_accuracy = evaluate(model,
                                        test_loader,
                                        criterion)
```

Case Study 1

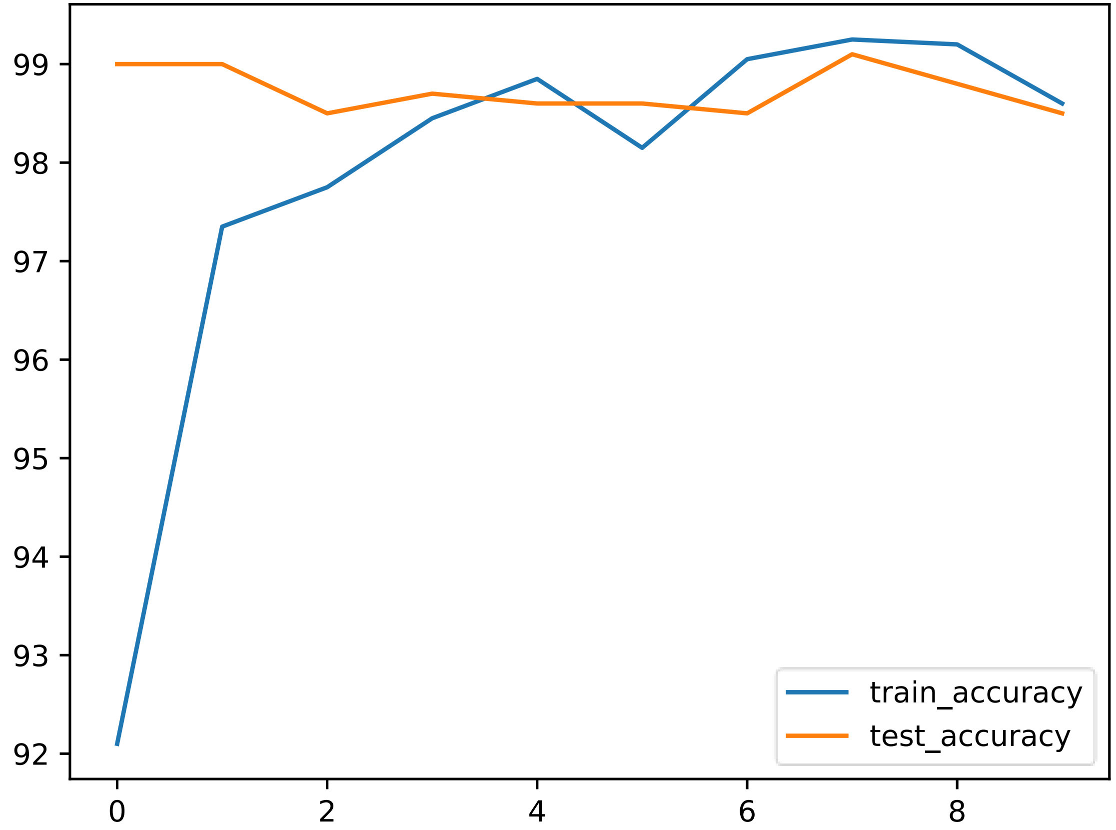
❖ Transfer learning

```python
# Load the pretrained VGG16 model
vgg16 = models.vgg16(weights=models.VGG16_Weights.DEFAULT)
f_extractor = vgg16.features

# Freeze the feature extraction part
for param in f_extractor.parameters():
    param.requires_grad = False

model = nn.Sequential(f_extractor,
                      nn.Flatten(),
                      nn.Dropout(0.3),
                      nn.Linear(512*7*7, 512),
                      nn.ReLU(),
                      nn.Dropout(0.3),
                      nn.Linear(512, 2))
```



Case Study 1

❖ Transfer learning



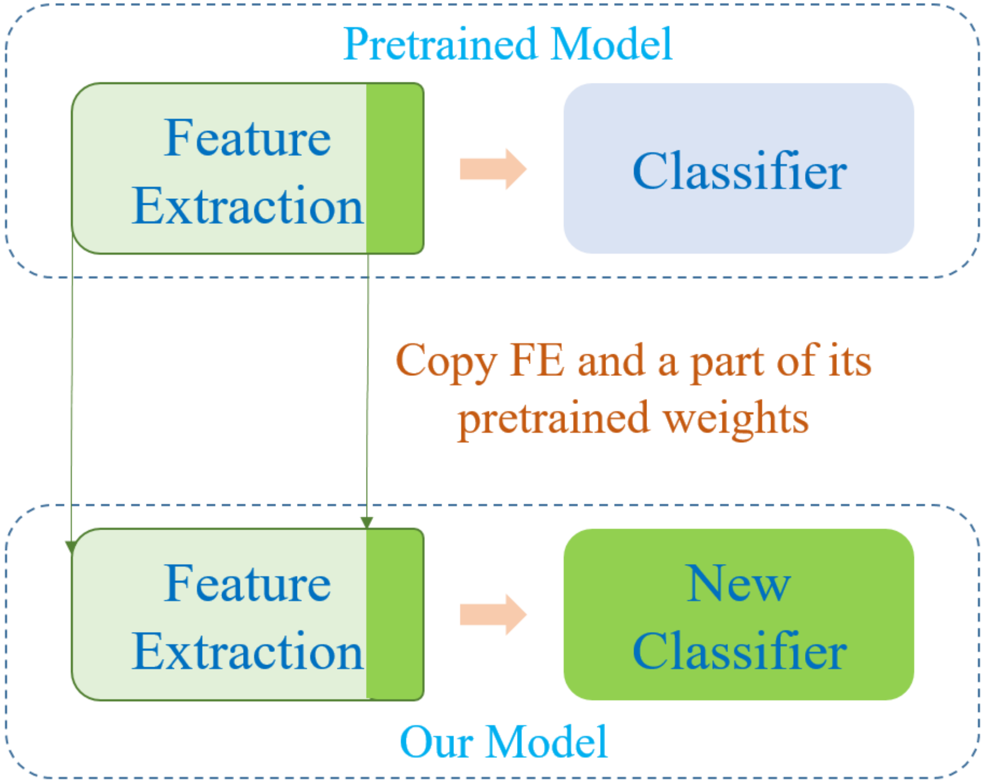
Training accuracy: 99%

Test accuracy: 99%

```
# train
for epoch in range(max_epoch):
    model.train()

    for i, (inputs, labels) in enumerate(train_loader, 0):
        # Move inputs and labels to the device
        inputs, labels = inputs.to(device), labels.to(device)

        # Zero the parameter gradients
        optimizer.zero_grad()

        # Forward pass
        outputs = model(inputs)
        loss = criterion(outputs, labels)

        # Backward pass and optimization
        loss.backward()
        optimizer.step()

    # evaluate
    test_loss, test_accuracy = evaluate(model,
                                        test_loader,
                                        criterion)
```

Case Study 1

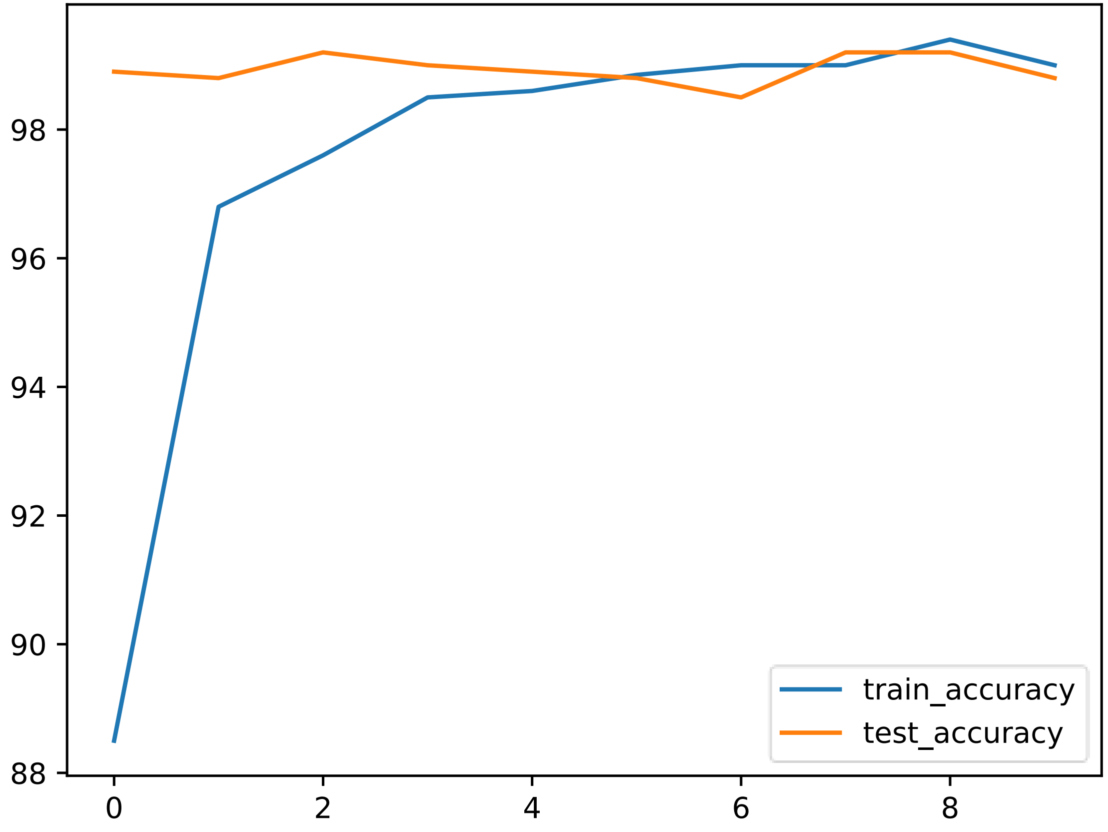
❖ Fine tuning




```python
# Load the pretrained VGG16 model
vgg16 = models.vgg16(weights=models.VGG16_Weights.DEFAULT)
f_extractor = vgg16.features

# Freeze the first 10 layers
layer_count = 0
for child in f_extractor.children():
    if layer_count < 10:
        for param in child.parameters():
            param.requires_grad = False
    layer_count += 1

model = nn.Sequential(f_extractor,
                      nn.Flatten(),
                      nn.Dropout(0.3),
                      nn.Linear(512*7*7, 512),
                      nn.ReLU(),
                      nn.Dropout(0.3),
                      nn.Linear(512, 2))
```

Case Study 1

❖ Fine tuning



Training accuracy: 99%

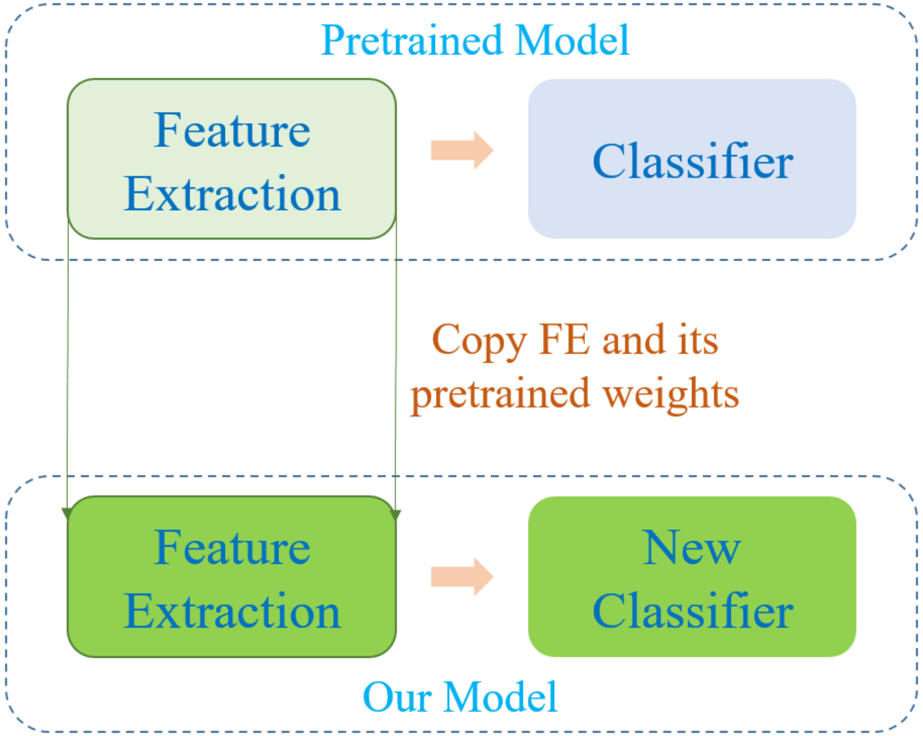
Test accuracy: 99%

```python
# train
for epoch in range(max_epoch):
    model.train()

    for i, (inputs, labels) in enumerate(train_loader, 0):
        # Move inputs and labels to the device
        inputs, labels = inputs.to(device), labels.to(device)

        # Zero the parameter gradients
        optimizer.zero_grad()

        # Forward pass
        outputs = model(inputs)
        loss = criterion(outputs, labels)

        # Backward pass and optimization
        loss.backward()
        optimizer.step()

    # evaluate
    test_loss, test_accuracy = evaluate(model,
                                        test_loader,
                                        criterion)
```

Case Study 1

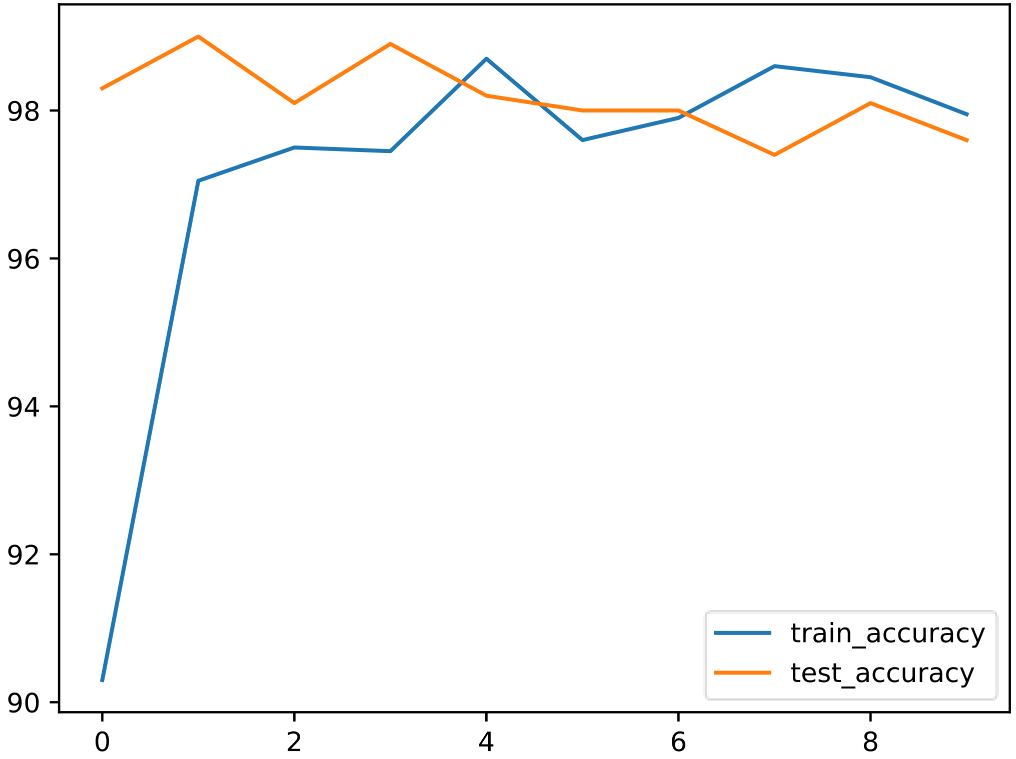
❖ Use the pretrained weights as an initialization

```python
# Load the pretrained VGG16 model
vgg16 = models.vgg16(weights=models.VGG16_Weights.DEFAULT)
f_extractor = vgg16.features

model = nn.Sequential(f_extractor,
                      nn.Flatten(),
                      nn.Dropout(0.3),
                      nn.Linear(512*7*7, 512),
                      nn.ReLU(),
                      nn.Dropout(0.3),
                      nn.Linear(512, 2))
```



Case Study 1

❖ **Use the pretrained weights as an initialization**



Training accuracy: 97.5%

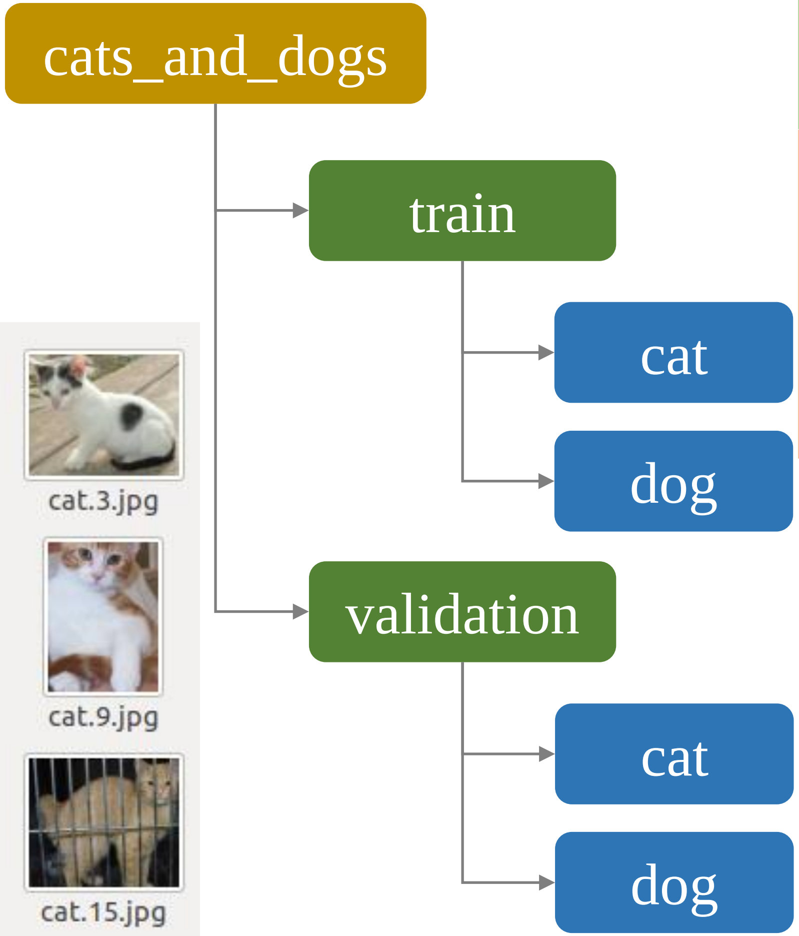
Test accuracy: 99%

```
# train
for epoch in range(max_epoch):
    model.train()

    for i, (inputs, labels) in enumerate(train_loader, 0):
        # Move inputs and labels to the device
        inputs, labels = inputs.to(device), labels.to(device)

        # Zero the parameter gradients
        optimizer.zero_grad()

        # Forward pass
        outputs = model(inputs)
        loss = criterion(outputs, labels)

        # Backward pass and optimization
        loss.backward()
        optimizer.step()

    # evaluate
    test_loss, test_accuracy = evaluate(model,
                                        test_loader,
                                        criterion)
```

1

Case Study 2

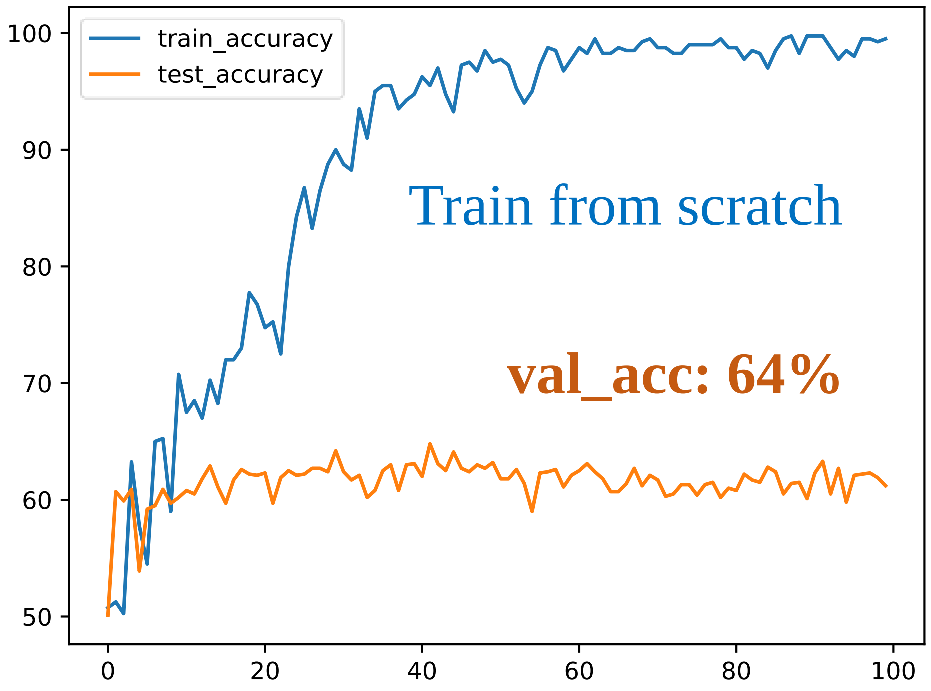
❖ Cat-Dog dataset

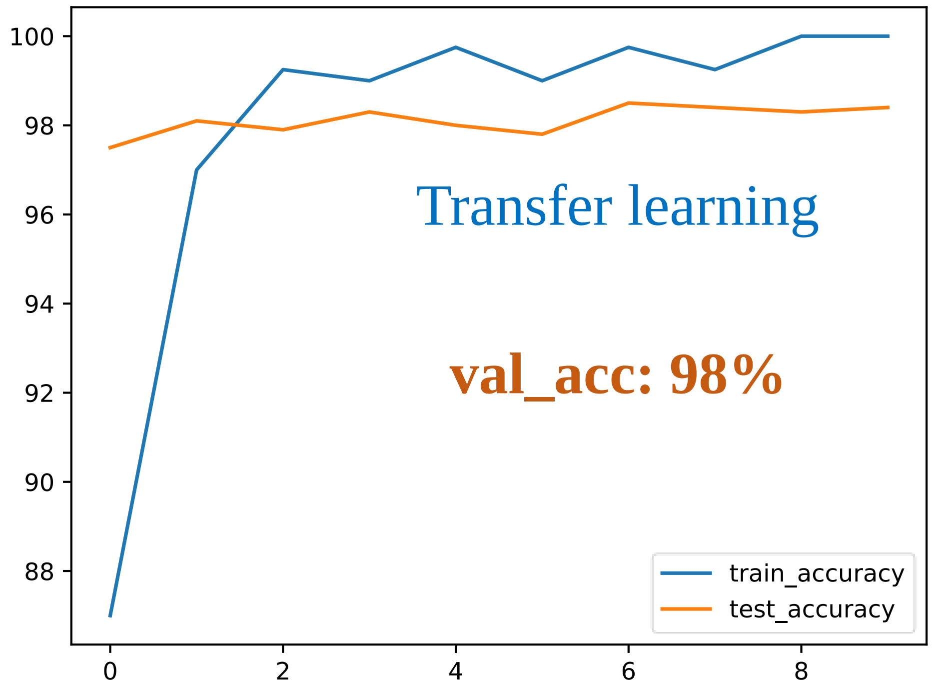


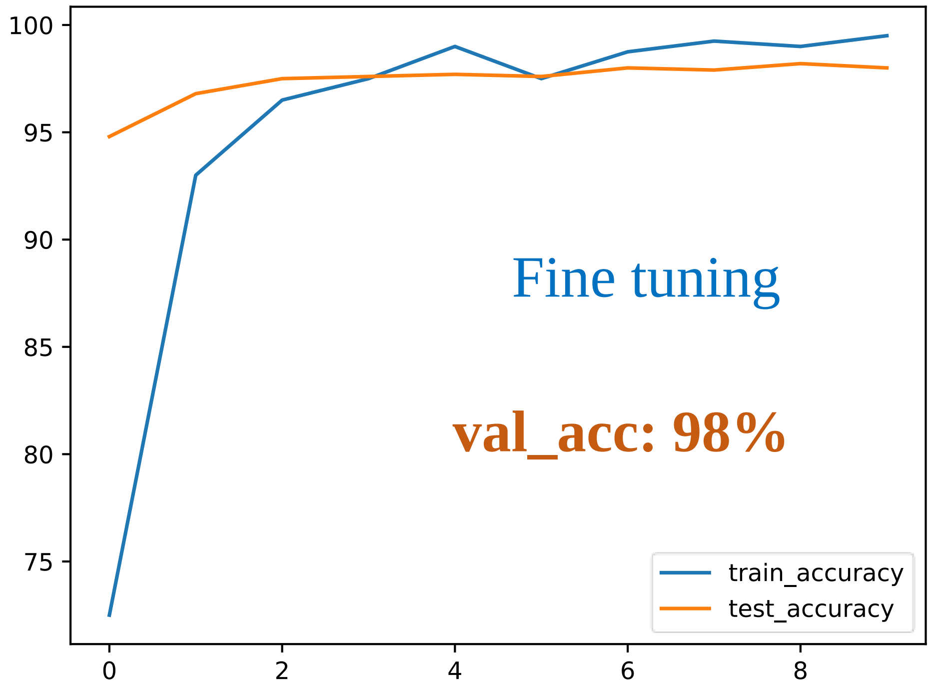

```python
train_transform = transforms.Compose(
    [
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                             std=[0.229, 0.224, 0.225]),
        transforms.RandomErasing(p=0.75, scale=(0.01, 0.3),
                                 ratio=(1.0, 1.0),
                                 value=0, inplace =True)
    ])
test_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])
])

# Load datasets
train_dataset = datasets.ImageFolder('data200/train',
                                     transform=train_transform)
test_dataset = datasets.ImageFolder('data200/validation',
                                    transform=test_transform)

# Create data loaders
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

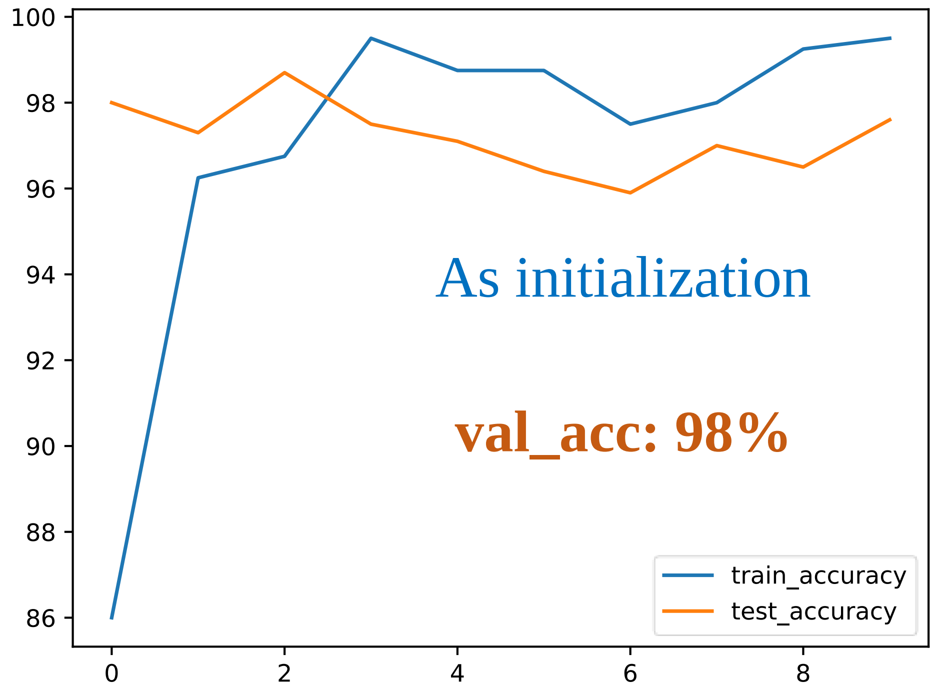
2

Case Study 2





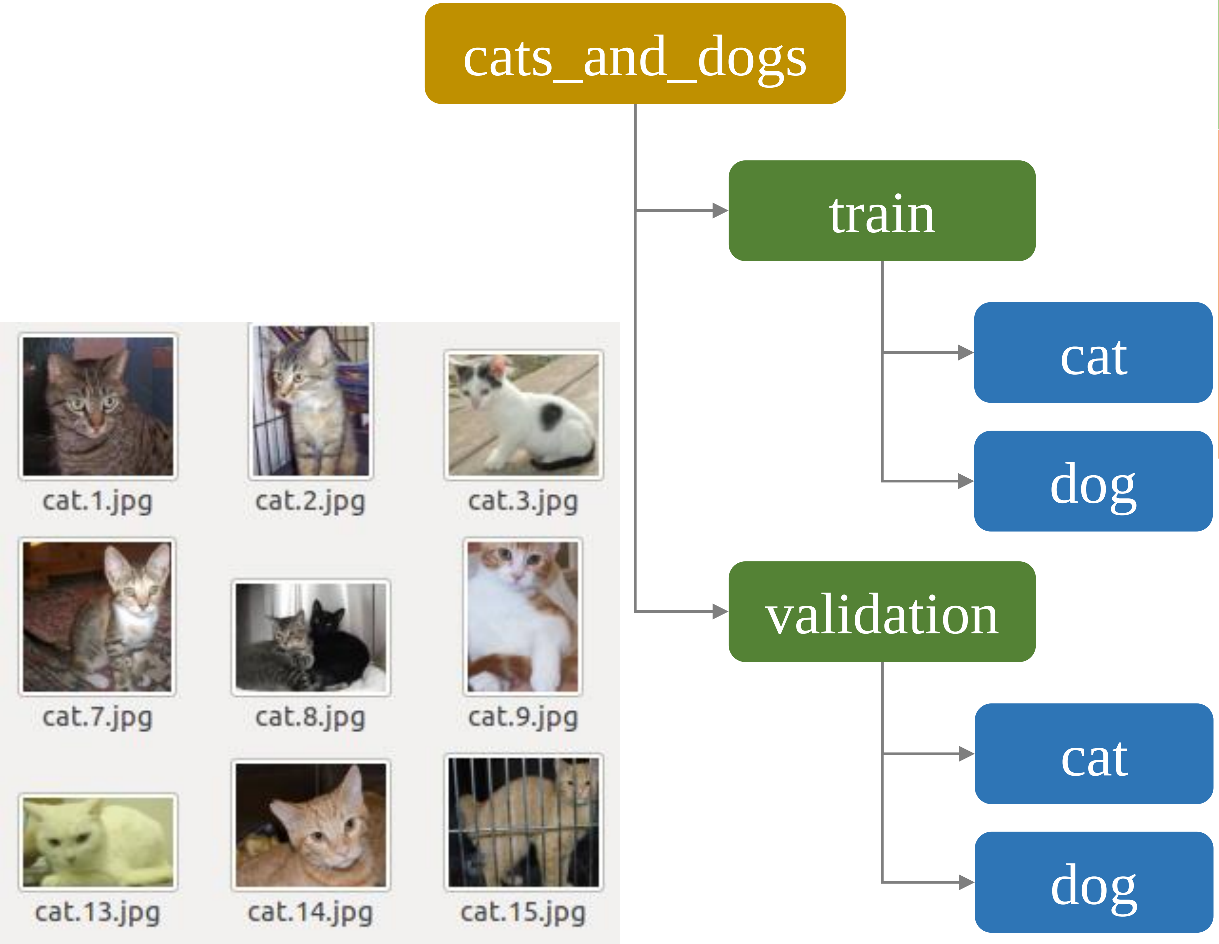




1

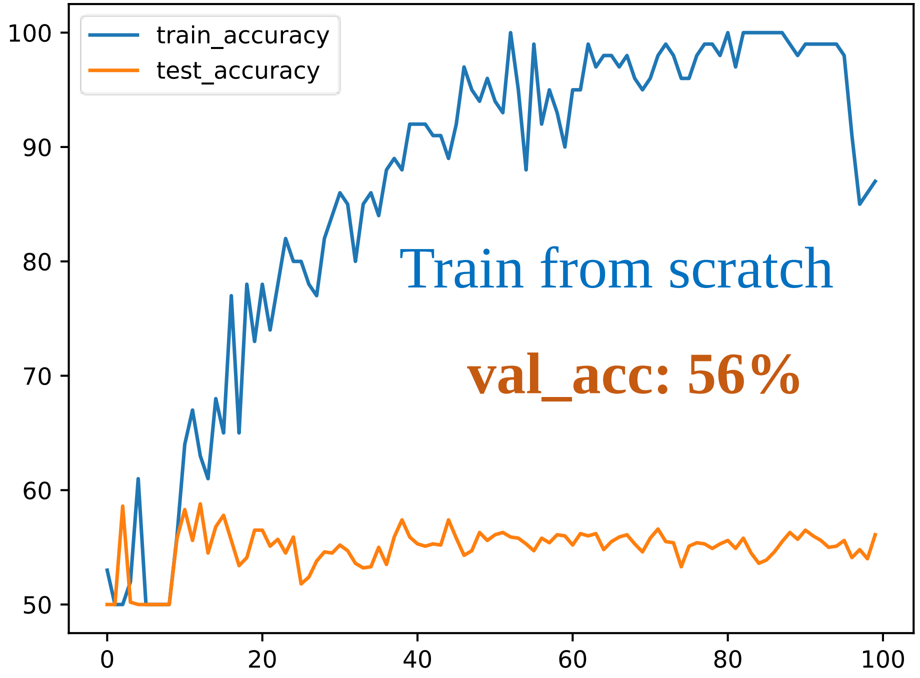
Case Study 3

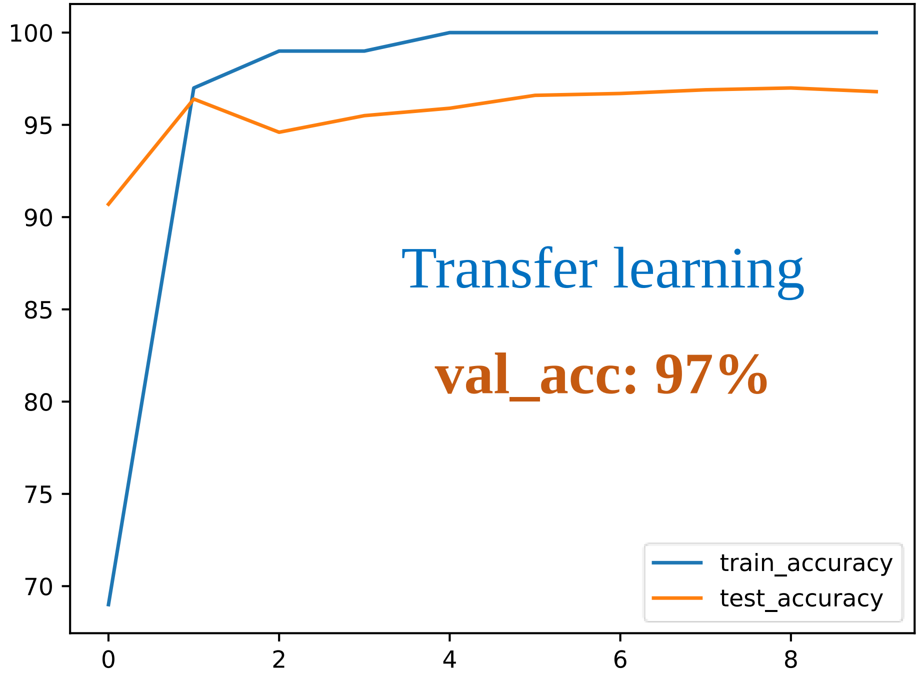
❖ **Cat-Dog dataset**

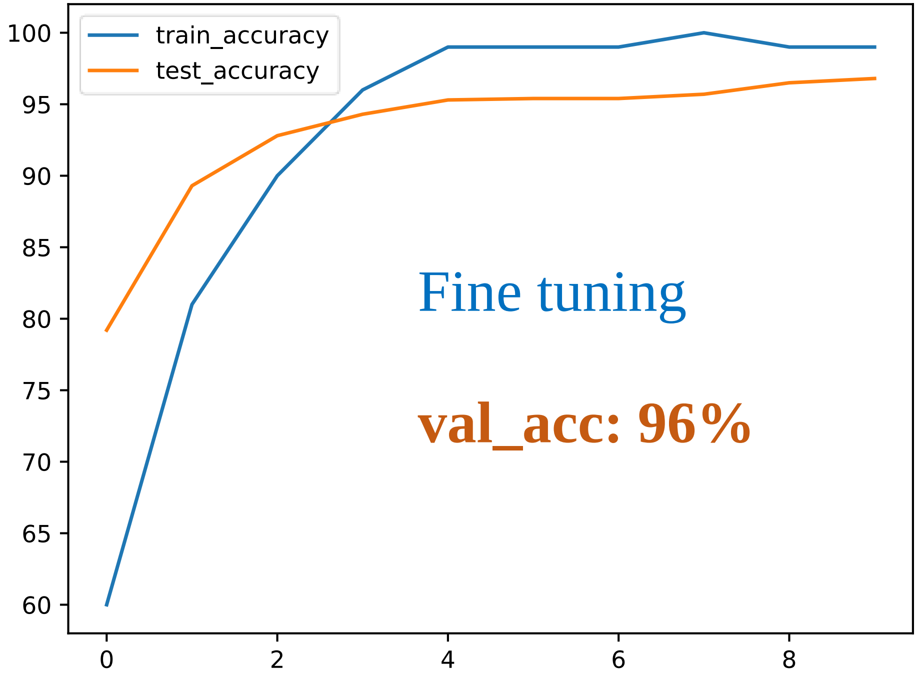


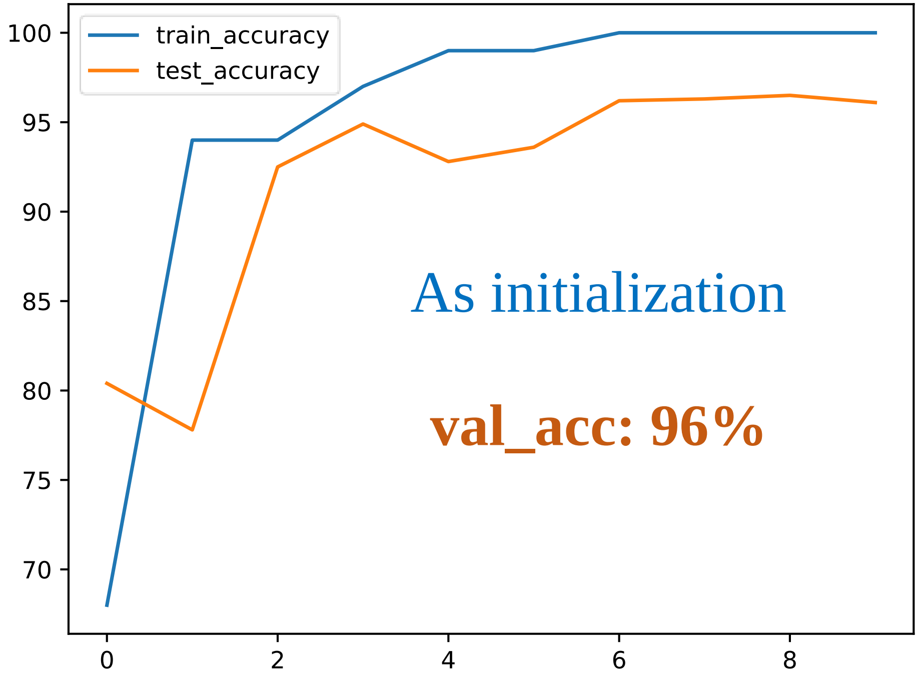

```python
train_transform = transforms.Compose(
    [
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                             std=[0.229, 0.224, 0.225]),
        transforms.RandomErasing(p=0.75, scale=(0.01, 0.3),
                                 ratio=(1.0, 1.0),
                                 value=0, inplace =True)
    ])
test_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])
])

# Load datasets
train_dataset = datasets.ImageFolder('data50/train',
                                     transform=train_transform)
test_dataset = datasets.ImageFolder('data50/validation',
                                    transform=test_transform)

# Create data loaders
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

Case Study 3


train_accuracy
test_accuracy
Train from scratch
val_acc: 56%


Transfer learning
val_acc: 97%
train_accuracy
test_accuracy


train_accuracy
test_accuracy
Fine tuning
val_acc: 96%


train_accuracy
test_accuracy
As initialization
val_acc: 96%

1

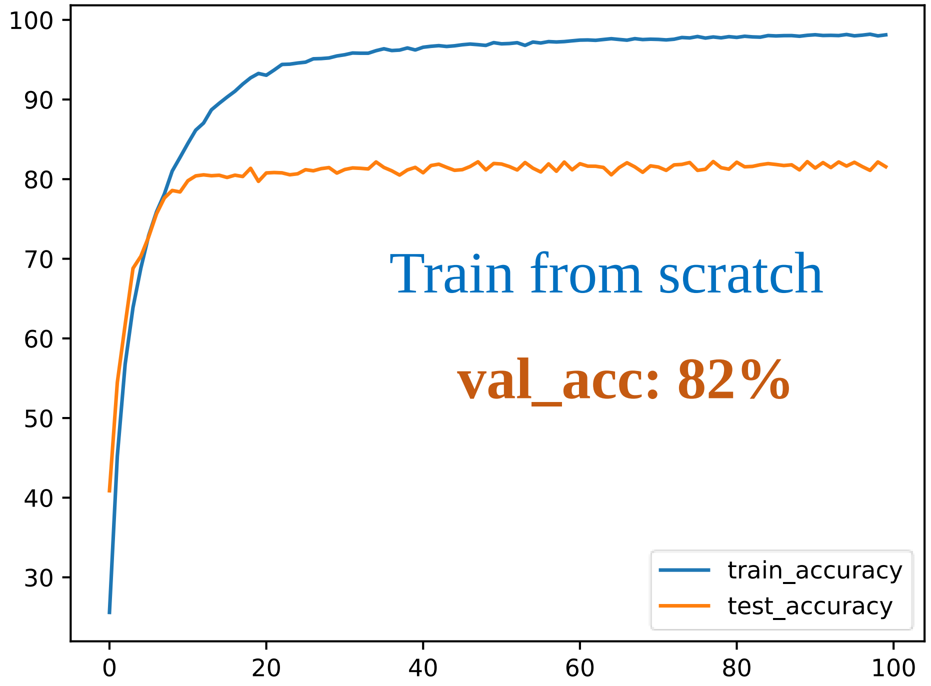
Case Study 4

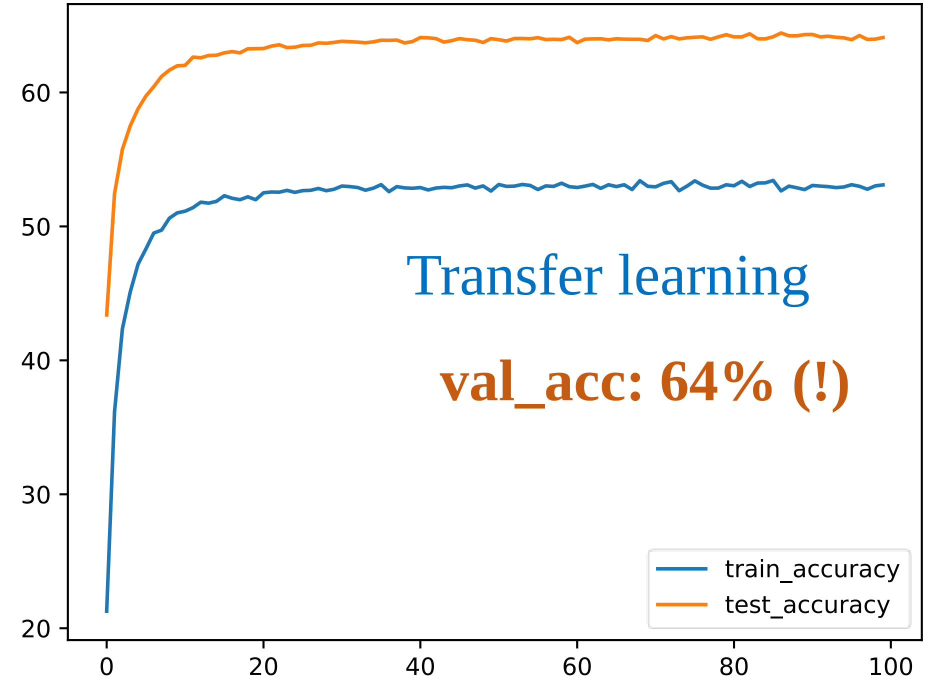
❖ **Cifar10 dataset**

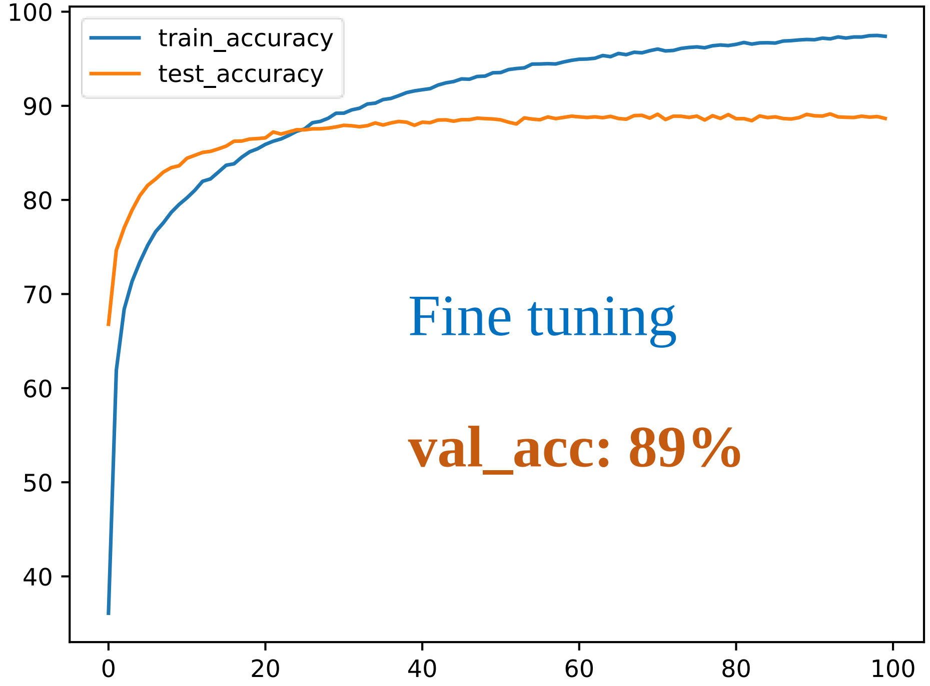
```python
train_transform = transforms.Compose(
    [
        transforms.ToTensor(),
        transforms.Normalize([0.4914, 0.4822, 0.4465],
                             [0.2470, 0.2435, 0.2616]),
        transforms.RandomErasing(p=0.75, scale=(0.01, 0.3),
                                 ratio=(1.0, 1.0),
                                 value=0, inplace =True)
    ])
val_transform = transforms.Compose(
    [
        transforms.ToTensor(),
        transforms.Normalize([0.4914, 0.4822, 0.4465],
                             [0.2470, 0.2435, 0.2616])
    ])

train_set = CIFAR10(root='./data', train=True,
                    download=True, transform=train_transform)
val_set = CIFAR10(root='./data', train=False,
                  download=True, transform=val_transform)

trainloader = DataLoader(train_set, batch_size=batch_size,
                         shuffle=True, num_workers=3)
testloader = DataLoader(val_set, batch_size=batch_size,
                        shuffle=False, num_workers=3)
```

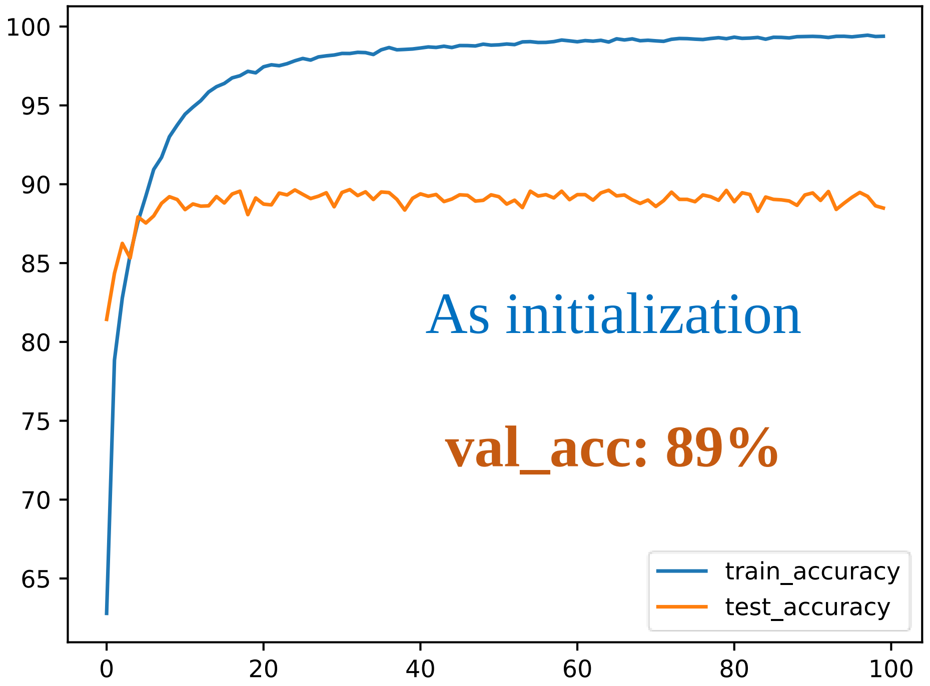
2

Case Study 4

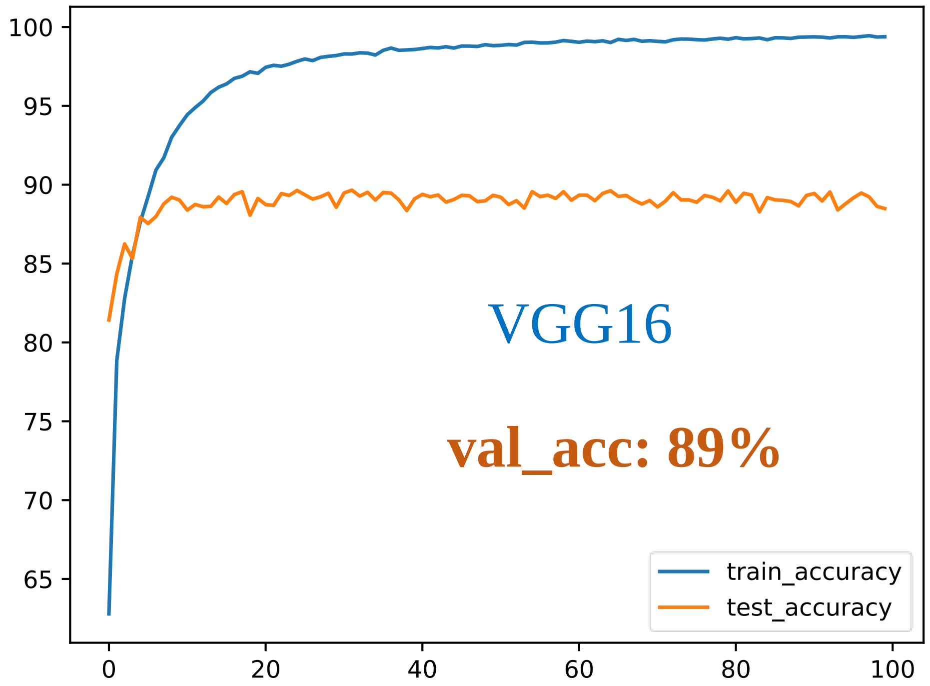


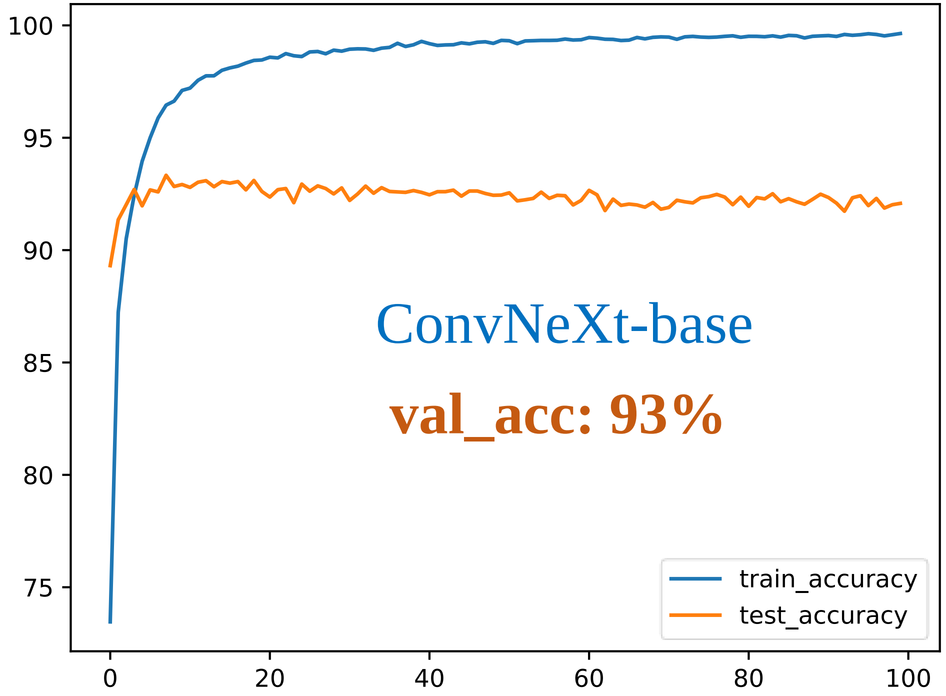


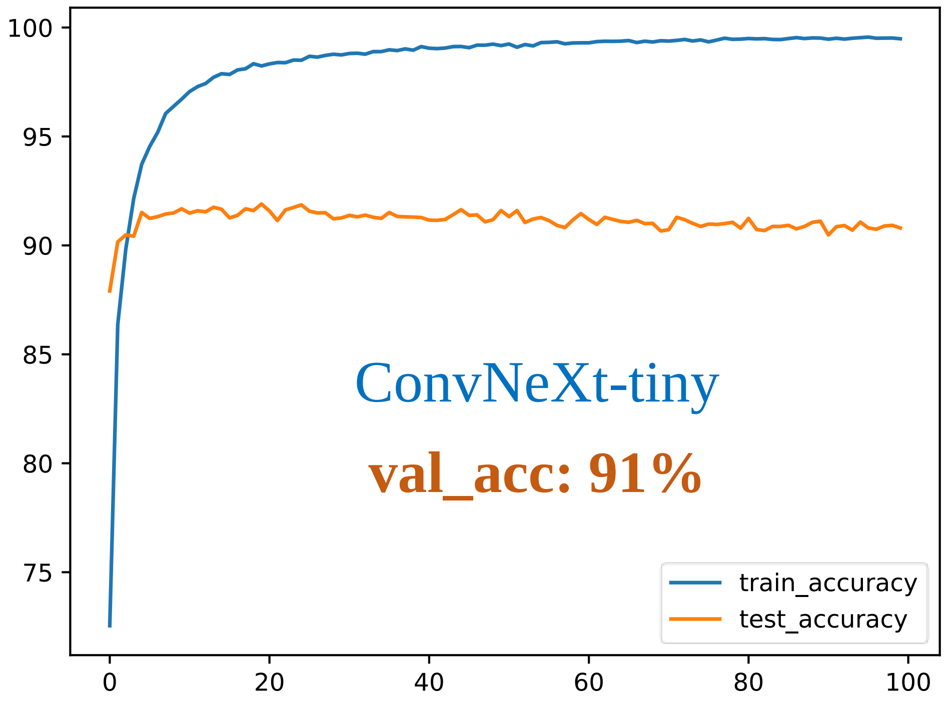


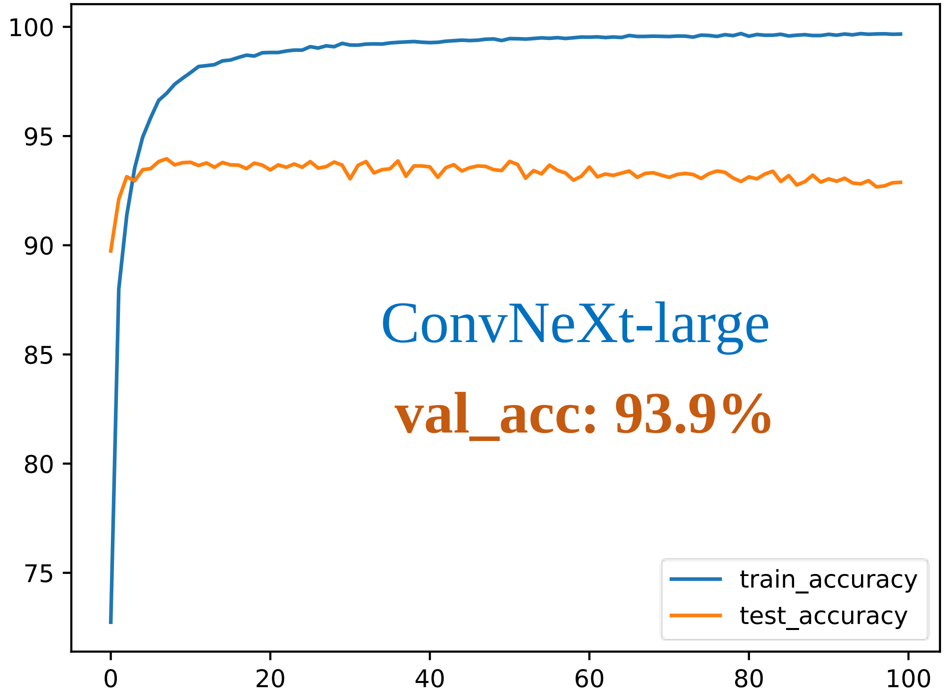


Case Study 4


VGG16
val_acc: 89%
train_accuracy
test_accuracy
100
95
90
85
80
75
70
65
0
20
40
60
80
100


ConvNeXt-base
val_acc: 93%
train_accuracy
test_accuracy
100
95
90
85
80
75
0
20
40
60
80
100

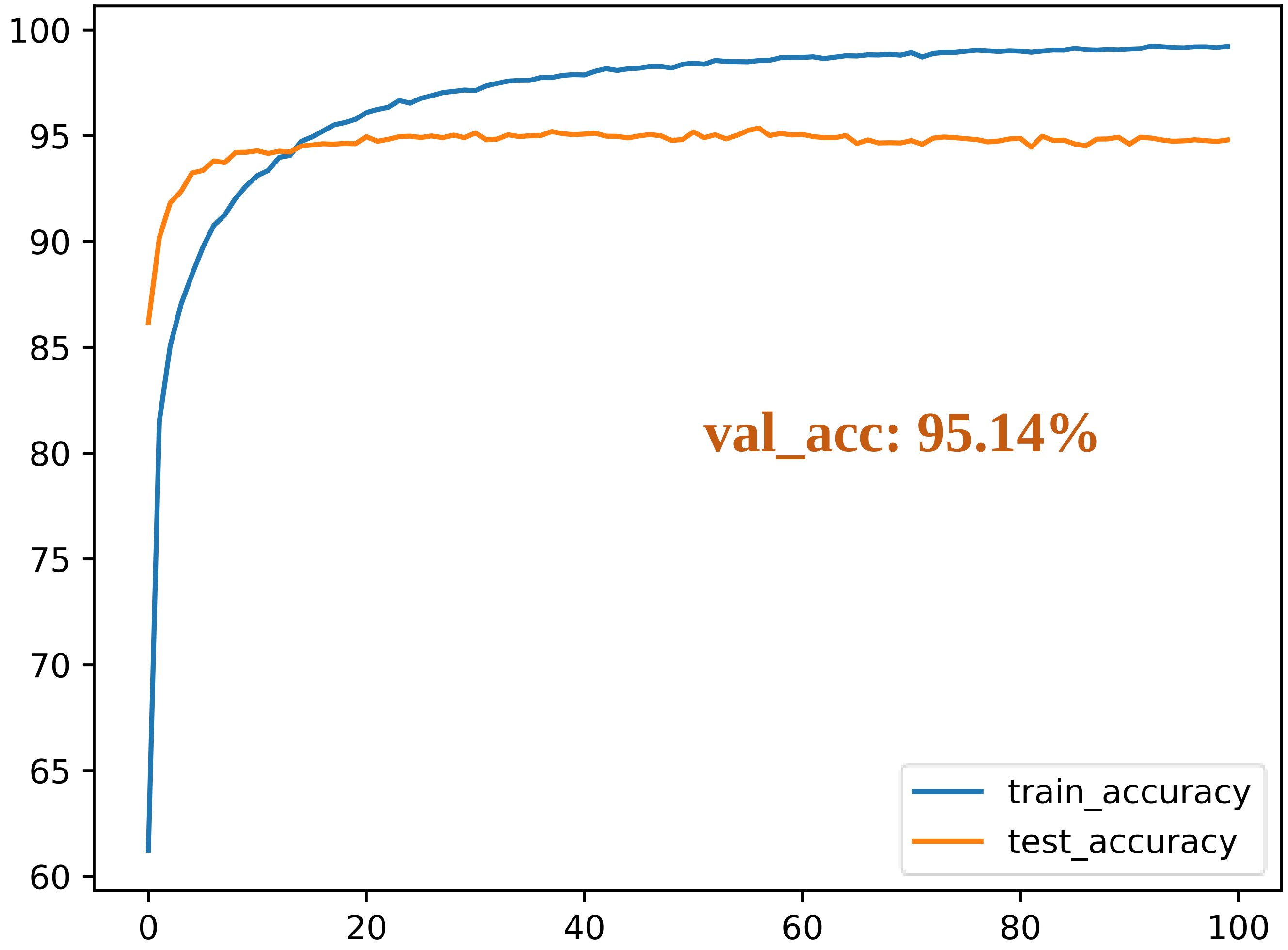

ConvNeXt-tiny
val_acc: 91%
train_accuracy
test_accuracy
100
95
90
85
80
75
0
20
40
60
80
100


ConvNeXt-large
val_acc: 93.9%
train_accuracy
test_accuracy
100
95
90
85
80
75
0
20
40
60
80
100

Case Study 4

❖ Add more augmentations

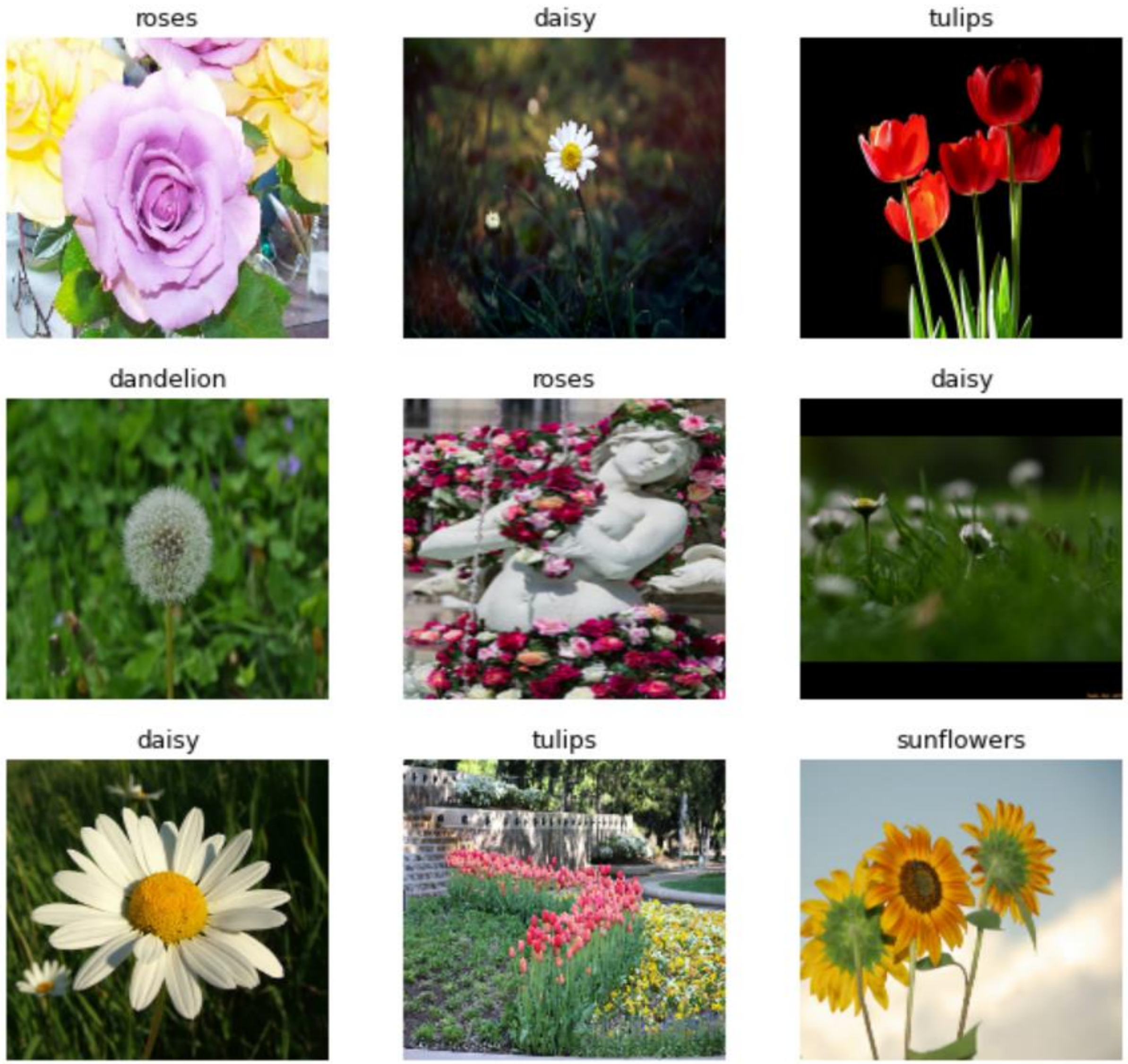
```
train_transform = transforms.Compose(
    [
        transforms.RandomCrop(32, padding=2),
        transforms.RandomHorizontalFlip(p=0.5),
        transforms.RandomRotation(5),
        transforms.ToTensor(),
        transforms.Normalize([0.4914, 0.4822, 0.4465],
                             [0.2470, 0.2435, 0.2616]),
        transforms.RandomErasing(p=0.75,
                                 scale=(0.01, 0.3),
                                 ratio=(1.0, 1.0),
                                 value=0,
                                 inplace =True)
    ])
```

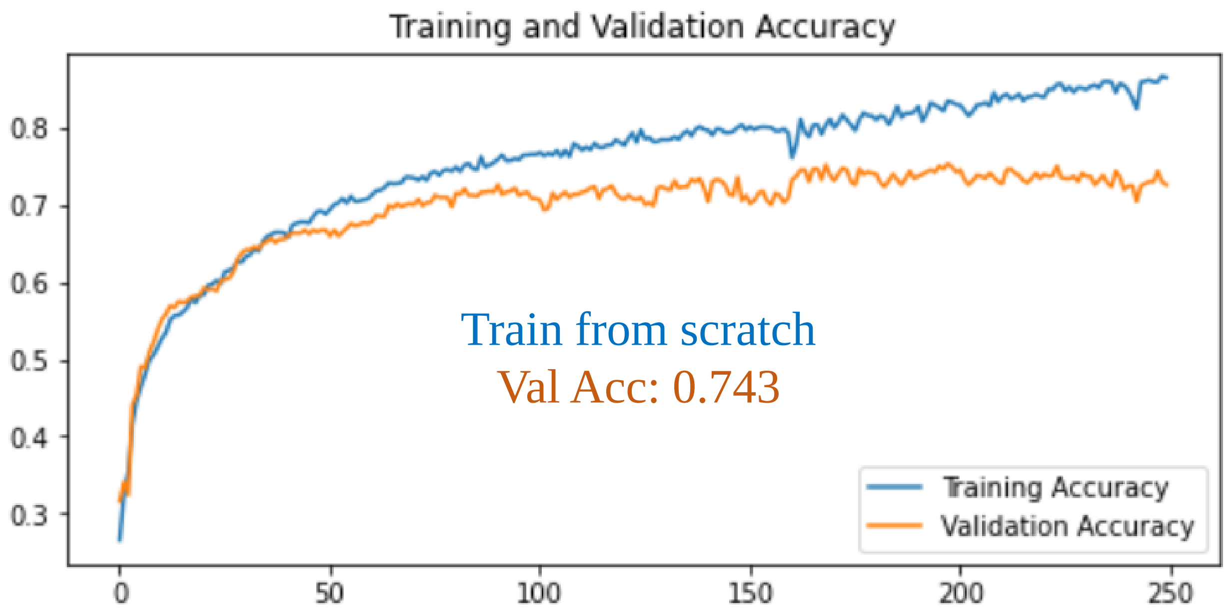


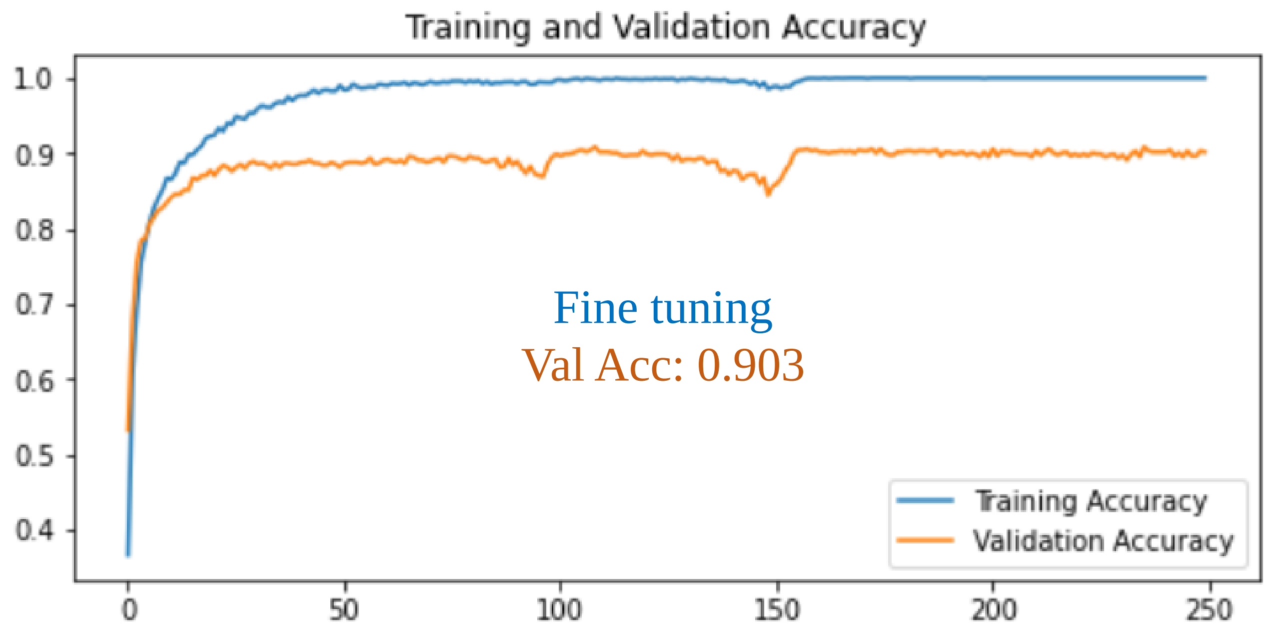
Case Study 5

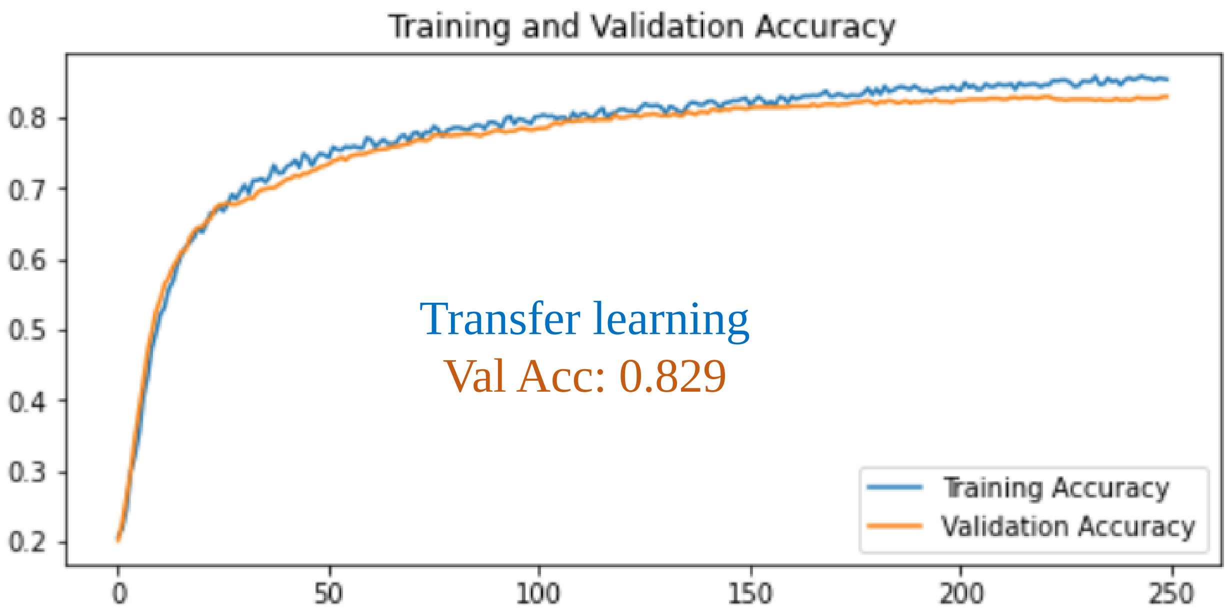
❖ New dataset

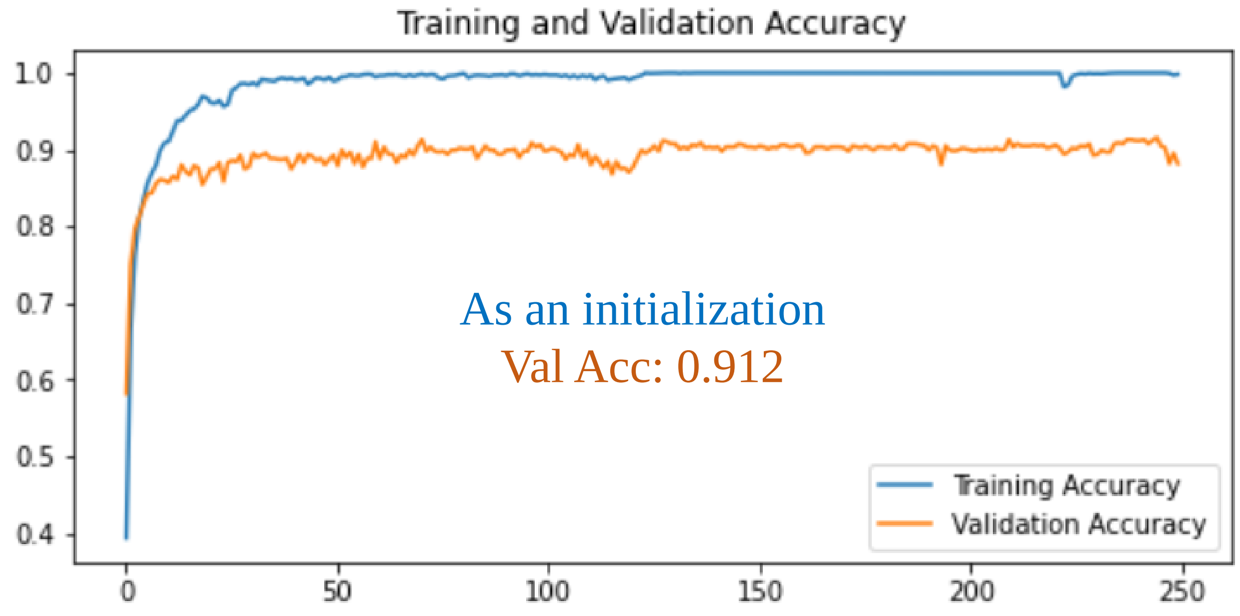
- flower_photos
  - daisy
  - dandelion
  - roses
  - sunflowers
  - tulips




Training and Validation Accuracy
0.8
0.7
0.6
0.5
0.4
0.3
0
50
100
150
200
250
Training Accuracy
Validation Accuracy
Train from scratch
Val Acc: 0.743


Training and Validation Accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0
50
100
150
200
250
Training Accuracy
Validation Accuracy
Fine tuning
Val Acc: 0.903


Training and Validation Accuracy
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
50
100
150
200
250
Training Accuracy
Validation Accuracy
Transfer learning
Val Acc: 0.829


Training and Validation Accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0
50
100
150
200
250
Training Accuracy
Validation Accuracy
As an initialization
Val Acc: 0.912