

Computer Vision

Deep Convolutional Neural Networks

NGUYỄN HẢI TRIỀU¹

¹ Bộ môn Kỹ thuật phần mềm,
Khoa Công nghệ thông tin, Trường ĐH Nha Trang

NhaTrang, September 2024

- 1 Convolutional Neural Networks
- 2 Training Convolutional Neural Networks
- 3 Transfer Learning

So sánh CNN và MLP

Qua tìm hiểu phần trước, **CNN có ưu điểm là giảm số lượng tham số và tính toán hiệu quả hơn so với MLP** do tính chất:

- **Sparse-connectivity**: mỗi neuron trong feature map *chỉ kết nối với một nhóm nhỏ các pixel đầu vào thông qua tích chập với Kernel* giúp CNN **học các đặc trưng cục bộ và giảm mạnh số lượng tham số**. Ví dụ:
 - ▶ **MLP**: Một neuron kết nối với toàn bộ 784 pixel của ảnh đầu vào.
 - ▶ **CNN**: Một neuron chỉ kết nối với một cửa sổ 3×3 (tổng cộng 9 pixel), sau đó cửa sổ này trượt trên ảnh để phát hiện đặc trưng.

So sánh CNN và MLP

Qua tìm hiểu phần trước, CNN có ưu điểm là giảm số lượng tham số và tính toán hiệu quả hơn so với MLP do tính chất:

- Cơ chế Parameter-sharing: trong CNN các trọng số ở một Kernel được sử dụng chung cho nhiều vùng ảnh khác nhau thay vì mỗi pixel sẽ có trọng số riêng biệt như MLP. Nhờ tính chất chia sẻ tham số, CNN có thể học cùng một đặc trưng ở các vị trí khác nhau.

So sánh CNN và MLP

Qua tìm hiểu phần trước, CNN có ưu điểm là giảm số lượng tham số và tính toán hiệu quả hơn so với MLP do tính chất:

- Cơ chế Many layers: Do sử dụng nhiều lớp tích chập chồng lên nhau nên các đặc trưng cục bộ (local patterns) được kết hợp để tạo thành đặc trưng tổng quát hơn (global patterns).

Ví dụ bài toán nhận diện bệnh cá:

- ▶ Lớp đầu tiên (phát hiện các cạnh): Nhận diện các cạnh của vây cá.
- ▶ Lớp thứ hai (phát hiện hình dạng đơn giản): Nhận diện vùng mắt, vây, và các cấu trúc nhỏ.
- ▶ Lớp thứ ba (nhận diện cấu trúc phức tạp hơn): Nhận diện toàn bộ hình dạng cá ...
- ▶ Lớp cuối cùng (xác định toàn bộ đối tượng): Phân loại con cá có bệnh hay không.

So sánh CNN và MLP I

Code minh hoạ

- Cơ chế Many layers: Do sử dụng nhiều lớp tích chập chồng lên nhau nên các đặc trưng cục bộ (local patterns) được kết hợp để tạo thành đặc trưng tổng quát hơn (global patterns). Ví dụ bài toán nhận diện bệnh cá (Chi tiết code xem ở file “*Chuong_6.1_CNN_local_2_global_pattern.pdf*”)

```
1 import torch
2 import torch.nn as nn
3 import torch.nn.functional as F
4 import matplotlib.pyplot as plt
5 from PIL import Image
6 import cv2
7
8 # load ảnh bằng openCV và chuyển sang tensor
```

So sánh CNN và MLP II

```

9  def load_image(image_path):
10     img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
11     img = cv2.resize(img, (64, 64))
12     img = img / 255.0 # Chuẩn hóa về [0,1]
13     img_tensor = torch.tensor(img,
14                               dtype=torch.float32).unsqueeze(0).unsqueeze(0)
15     return img, img_tensor
16
17 image_path = "fish.jpg"
18 original_img, img_tensor = load_image(image_path)
19 print("Original image shape:", original_img.shape)
20 print("Image tensor shape:", img_tensor.shape)
21
22 class ML_CNN(nn.Module):
23     def __init__(self):
24         super(ML_CNN, self).__init__()
25         self.conv1 = nn.Conv2d(in_channels=1, out_channels=4, kernel_size=3,
26                                stride=1, padding=1) # Lớp 1: Phát hiện cạnh
27         self.conv2 = nn.Conv2d(in_channels=4, out_channels=8, kernel_size=3,
28                                stride=1, padding=1) # Lớp 2: Kết hợp cạnh

```

So sánh CNN và MLP III

```

26         self.conv3 = nn.Conv2d(in_channels=8, out_channels=16, kernel_size=3,
27                                 stride=1, padding=1) # Lớp 3: Học đặc trưng tổng thể
28     def forward(self, x):
29         x1 = F.relu(self.conv1(x))
30         x2 = F.relu(self.conv2(x1))
31         x3 = F.relu(self.conv3(x2))
32         return x1, x2, x3
33
34     # Khởi tạo mạng CNN
35     model = ML_CNN()
36
37     # Dự đoán qua từng lớp
38     with torch.no_grad():
39         feature_maps = model(img_tensor)
40
41     # Vẽ đặc trưng của từng lớp
42     def plot_feature_maps(feature_maps, title):
43         num_filters = feature_maps.shape[1]
44         fig, axes = plt.subplots(1, num_filters, figsize=(15, 5))

```


So sánh CNN và MLP IV

```

45
46     for i in range(num_filters):
47         axes[i].imshow(feature_maps[0, i].cpu().numpy(), cmap='gray')
48         axes[i].axis("off")
49
50     plt.suptitle(title, fontsize=14)
51     plt.show()
52
53     # Hiển thị ảnh gốc
54     plt.imshow(original_img, cmap='gray')
55     plt.title("Ảnh Gốc")
56     plt.axis("off")
57     plt.show()
58
59     # Hiển thị đầu ra của từng lớp CNN
60     plot_feature_maps(feature_maps[0], "Lớp 1 - Phát hiện cạnh")
61     plot_feature_maps(feature_maps[1], "Lớp 2 - Học hình dạng")
62     plot_feature_maps(feature_maps[2], "Lớp 3 - Học đặc trưng tổng thể")

```

So sánh CNN và MLP

```

5 class ML_CNN(nn.Module):
6     def __init__(self):
7         super(ML_CNN, self).__init__()
8         self.conv1 = nn.Conv2d(in_channels=1, out_channels=4, kernel_size=3, stride=1,
9 padding=1) 3*3*4*1+4
10        self.conv2 = nn.Conv2d(in_channels=4, out_channels=8, kernel_size=3, stride=1,
11 padding=1) 3*3*8*4+8
12        self.conv3 = nn.Conv2d(in_channels=8, out_channels=16, kernel_size=3, stride=1,
13 padding=1) 3*3*16*8+16
14        self.fc_layer = torch.nn.Sequential(
15            nn.Linear(16*28*28, 100), nn.ReLU(), nn.Linear(100, 10),)
16        def forward(self, x): 16*28*28*100+100 100*10+10
17            x1 = F.relu(self.conv1(x))
18            x2 = F.relu(self.conv2(x1))
19            x3 = F.relu(self.conv3(x2))
20            x = torch.flatten(x3, 1)
21            logits = self.fc_layer(x)
22            return logits
23
24 # Khởi tạo mạng CNN
25 model = ML_CNN()
26 # Tính tổng tham số của mô hình
27 number_of_parameter = 0
28 storage_memory = 0.0
29 for name, param in model.named_parameters():
30     number_of_parameter += param.numel()
31     storage_memory += (param.numel() * param.element_size()) / (1024 ** 3)
32 print("Số lượng tham số của mô hình:", number_of_parameter)
33 print(f"Dung lượng lưu trữ của mô hình: {storage_memory:f} GB")

```

✓ 0.0s Python

Số lượng tham số của mô hình: 1257014
Dung lượng lưu trữ của mô hình: 0.004683 GB

1,257,014 parameters!!!

```

5 class MLP(nn.Module):
6     def __init__(self):
7         super(MLP, self).__init__()
8         self.Linear1 = nn.Linear(3*224*224, 10000) 3*224*224*10000+10000
9         self.Linear2 = nn.Linear(10000, 1000) 10000*1000+1000
10        self.Linear3 = nn.Linear(1000, 10) 1000*100+100
11        self.output = nn.Linear(100, 10) 100*10+10
12
13    def forward(self, x):
14        x1 = F.relu(self.Linear1(x))
15        x2 = F.relu(self.Linear2(x1))
16        x3 = F.relu(self.Linear3(x2))
17        x = torch.flatten(x3, 1)
18        logits = self.output(x)
19        return logits
20
21 # Khởi tạo mạng MLP
22 model = MLP()
23 # Tính tổng tham số của mô hình
24 number_of_parameter = 0
25 storage_memory = 0.0
26 for name, param in model.named_parameters():
27     number_of_parameter += param.numel()
28     storage_memory += (param.numel() * param.element_size()) / (1024 ** 3)
29 print("Số lượng tham số của mô hình:", number_of_parameter)
30 print(f"Dung lượng lưu trữ của mô hình: {storage_memory:f} GB")
31
32
33

```

✓ 5.0s

Số lượng tham số của mô hình: 1515392110
Dung lượng lưu trữ của mô hình: 5.645276 GB

1,515,392,110 parameters!!!

Hình 1: Như đã đề cập, CNN có ưu điểm giảm mạnh số lượng tham số vì vậy nó cho kết quả tính toán hiệu quả hơn so với MLP. Đặc biệt tiết kiệm dung lượng lưu trữ của mô hình. CNN phù hợp các bài toán Xử lý ảnh

- 1 Convolutional Neural Networks
- 2 Training Convolutional Neural Networks
- 3 Transfer Learning

Huấn luyện mạng CNN trên tập MNIST I

Trong phần này, chúng ta sẽ tìm hiểu cách huấn luyện mạng CNN từ ban đầu trên tập dữ liệu nhận diện chữ số viết tay MNIST.

Định nghĩa mạng CNN cho bộ dữ liệu MNIST

```
1 class PyTorchCNN(torch.nn.Module):
2     def __init__(self, num_classes):
3         super().__init__()
4
5         self.cnn_layers = torch.nn.Sequential(
6             #block 01
7             torch.nn.Conv2d(1, 3, kernel_size=5),
8             torch.nn.BatchNorm2d(3),
9             torch.nn.ReLU(),
10            torch.nn.MaxPool2d(kernel_size=2),
11            #block 02
12            torch.nn.Conv2d(3, 16, kernel_size=3),
13            torch.nn.BatchNorm2d(16),
```

Huấn luyện mạng CNN trên tập MNIST II

```
14     torch.nn.ReLU(),
15     torch.nn.MaxPool2d(kernel_size=2),
16     #block03
17     torch.nn.Conv2d(16, 32, kernel_size=3),
18     torch.nn.BatchNorm2d(32),
19     torch.nn.ReLU(),
20     torch.nn.MaxPool2d(kernel_size=2),
21 )
22
23 self.fc_layers = torch.nn.Sequential(
24     # hidden layer
25     torch.nn.Linear(32, 20),
26     torch.nn.BatchNorm1d(20),
27     torch.nn.ReLU(),
28
29     # output layer
30     torch.nn.Linear(20, num_classes)
31 )
32
33 def forward(self, x):
```

Huấn luyện mạng CNN trên tập MNIST III

```

34     x = self.cnn_layers(x)
35     # print(x.shape)
36     x = torch.flatten(x, start_dim=1)
37     logits = self.fc_layers(x)
38     return logits

```

Định nghĩa bộ dữ liệu MNIST

```

1  class MnistDataModule(L.LightningDataModule):
2      def __init__(self, data_path="./", batch_size=64, num_workers=0):
3          super().__init__()
4          self.batch_size = batch_size
5          self.data_path = data_path
6          self.num_workers = num_workers
7
8      def prepare_data(self):
9          datasets.MNIST(root=self.data_path, download=True)
10         return
11

```

Huấn luyện mạng CNN trên tập MNIST IV

```
12 def setup(self, stage=None):
13     # Note transforms.ToTensor() scales input images
14     # to 0-1 range
15     train = datasets.MNIST(
16         root=self.data_path,
17         train=True,
18         transform=transforms.ToTensor(),
19         download=False,
20     )
21
22     self.test = datasets.MNIST(
23         root=self.data_path,
24         train=False,
25         transform=transforms.ToTensor(),
26         download=False,
27     )
28
29     self.train, self.valid = random_split(train, lengths=[55000, 5000],
30         generator=torch.Generator().manual_seed(42))
```

Huấn luyện mạng CNN trên tập MNIST V

```
31 def train_dataloader(self):
32     train_loader = DataLoader(
33         dataset=self.train,
34         batch_size=self.batch_size,
35         drop_last=True,
36         shuffle=True,
37         num_workers=self.num_workers,
38     )
39     return train_loader
40
41 def val_dataloader(self):
42     valid_loader = DataLoader(
43         dataset=self.valid,
44         batch_size=self.batch_size,
45         drop_last=False,
46         shuffle=False,
47         num_workers=self.num_workers,
48     )
49     return valid_loader
50
```


Huấn luyện mạng CNN trên tập MNIST VI

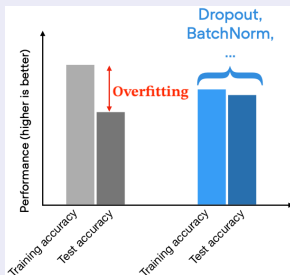
```
51 def test_dataloader(self):  
52     test_loader = DataLoader(  
53         dataset=self.test,  
54         batch_size=self.batch_size,  
55         drop_last=False,  
56         shuffle=False,  
57         num_workers=self.num_workers,  
58     )  
59     return test_loader
```

(Chi tiết code xem ở file

“[Chuong_6.1_SimpleTrainingCNN_MNIST.pdf](#)”)

Tăng cường dữ liệu (Data Augmentation) I

Trong quá trình huấn luyện thực tế ta sẽ gặp hiện tượng **Overfitting**.



Hình 2: Ta có thể sử dụng kỹ thuật như *BatchNorm* hoặc *Dropout* để cải thiện độ chính xác như các chương trước.

Ngoài ra, giải pháp đơn giản nhất là sử dụng kỹ thuật tăng cường dữ liệu (Data Augmentation).

Tăng cường dữ liệu (Data Augmentation) II

Random data augmentation is a way to artificially **increase the dataset size**.

Lưu ý: chúng ta **không áp dụng tăng cường dữ liệu cho tập test và tập validation**.

Để tăng cường dữ liệu, chúng ta chỉ bổ sung thêm “*Data Transform*”

Tăng cường dữ liệu (Data Augmentation) III

```

1  from torchvision import transforms
2
3  train_transform = transforms.Compose(
4      [
5          transforms.ToPILImage(), # Chuyển ảnh định dạng PNG hoặc JPEG sang ảnh PIL
                                   chuẩn của python
6          transforms.Resize((150, 150)), # Resize ảnh về kích thước 150x150
7          transforms.RandomCrop((128, 128)), # Cắt ngẫu nhiên ảnh về kích thước
                                   128x128
8          transforms.RandomHorizontalFlip(p=0.2), # Lật ngang ảnh với xác suất 0.2
9          transforms.ColorJitter(brightness=0.1, contrast=0.1, saturation=0.1,
                                   hue=0.1), # Thay đổi màu sắc
10         transforms.ToTensor(), # Chuyển ảnh về tensor và thực hiện chuẩn hoá pixel
                                   từ [0,255] -> [0,1]
11         transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)), # chuẩn hoá ảnh từ
                                   [0, 1] -> [-1, 1] thông qua mean và std
12     ]
13 )
14

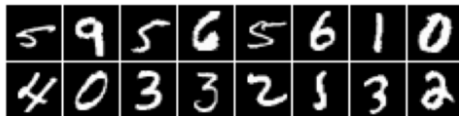
```

Tăng cường dữ liệu (Data Augmentation) IV

```
15 test_transform = transforms.Compose(  
16     [  
17         transforms.ToPILImage(),  
18         transforms.Resize((150, 150)),  
19         transforms.CenterCrop((128, 128)),  
20         transforms.ToTensor(),  
21         transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),  
22     ]  
23 )
```

Tăng cường dữ liệu (Data Augmentation) V

Original training images



Randomly cropped training images



Randomly rotated training images



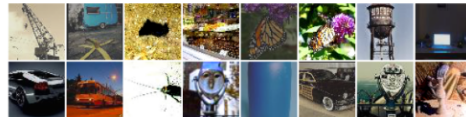
Hình 3: Ví dụ một số kỹ thuật tăng cường dữ liệu phổ biến

Tăng cường dữ liệu (Data Augmentation) VI

Original training images



Randomly modified brightness

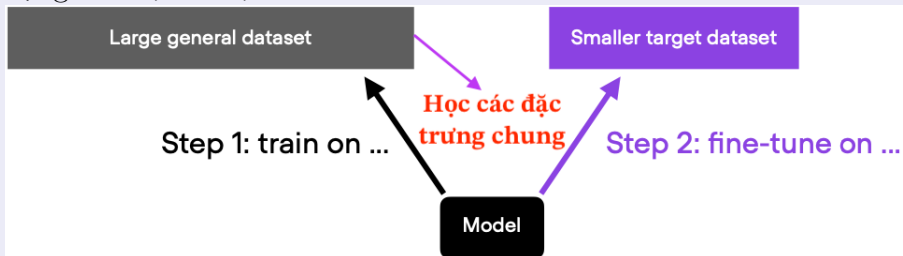


Hình 4: Ví dụ một số kỹ thuật tăng cường dữ liệu phổ biến

- 1 Convolutional Neural Networks
- 2 Training Convolutional Neural Networks
- 3 Transfer Learning

Transfer Learning

Transfer Learning-Học chuyển giao: là một quy trình gồm 2 bước để giúp chúng ta có thể tận dụng kiến thức chung (**học các *general feature extraction layers***) từ các bộ dữ liệu lớn hơn và áp dụng cho bộ dữ liệu nhỏ hơn.



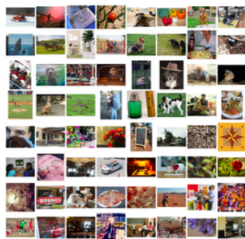
Transfer Learning

Ví dụ chúng ta có thể tận dụng mô hình đã được huấn luyện trên tập dữ liệu lớn như ImageNet để giúp huấn luyện mô hình trên tập dữ liệu nhỏ hơn như bộ goldfinch.

Large general dataset

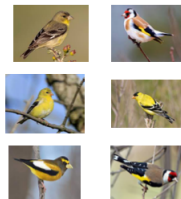
Smaller target dataset

For example, ImageNet



Chúng ta có thể
dùng lại các lớp
trích xuất đặc trưng
(pre-trained) trên tập
dữ liệu lớn ImageNet
và “chỉ huấn luyện lại
lớp fully connected”
cuối cùng cho
bài toán phân loại
goldfinch

For example, different
goldfinch species



Các loại Transfer learning

Quan điểm Transfer learning dựa trên tập dữ liệu gồm 2 loại chính sau:

1

Step 1: train on ...

Large general dataset

Step 2: fine-tune on ...

Smaller target dataset

2

Step 1: train on ...

Large general dataset

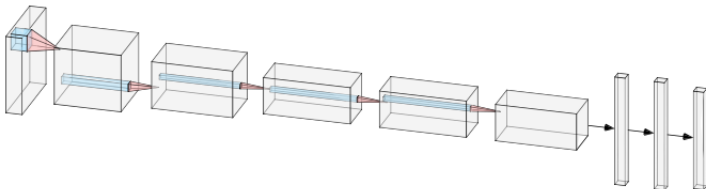
+

Smaller target dataset

Biến thể loại 1

1 Fine-tune last layer

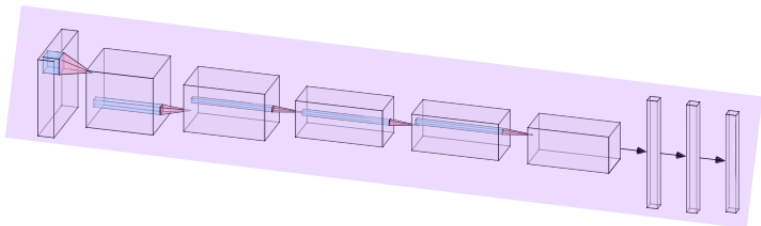
Step 1: train whole model on large dataset



Biến thể loại 1

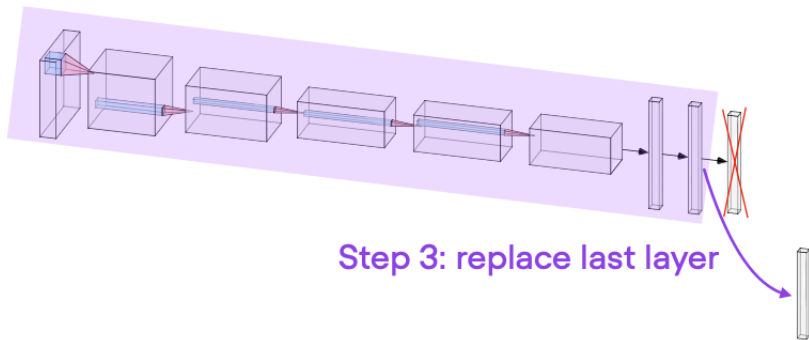
1 Fine-tune last layer

Step 2: freeze weights



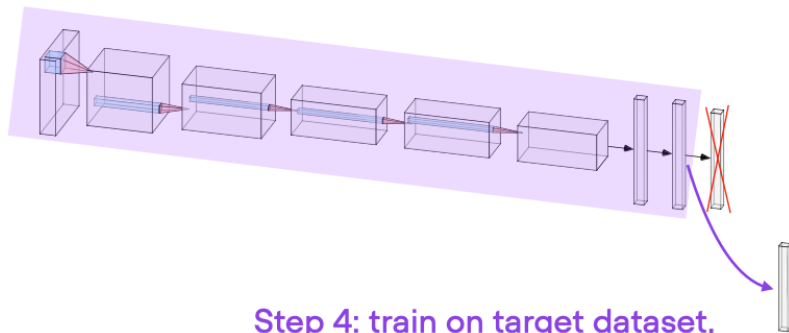
Biến thể loại 1

1 Fine-tune last layer



Biến thể loại 1

1 Fine-tune last layer



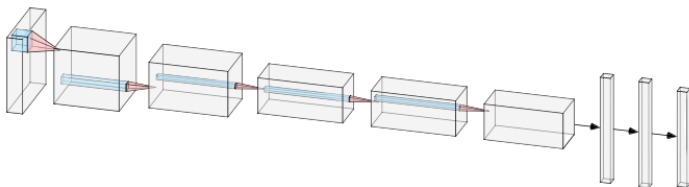
Step 4: train on target dataset,
but only update new output layer

Biến thể loại 2

Thay vì chỉ **cập nhật trọng số cho một lớp cuối cùng** như biến thể 1, chúng ta **cập nhật trọng số cho các lớp FC cuối cùng** trong biến thể loại 2.

2 Fine-tune last layers

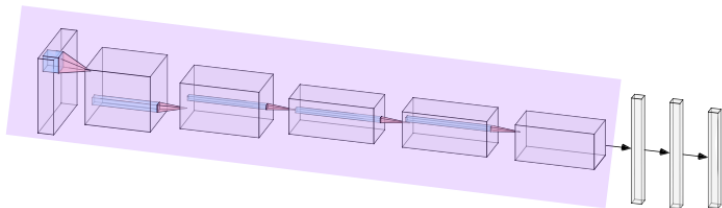
Step 1: train whole model on large dataset



Biến thể loại 2

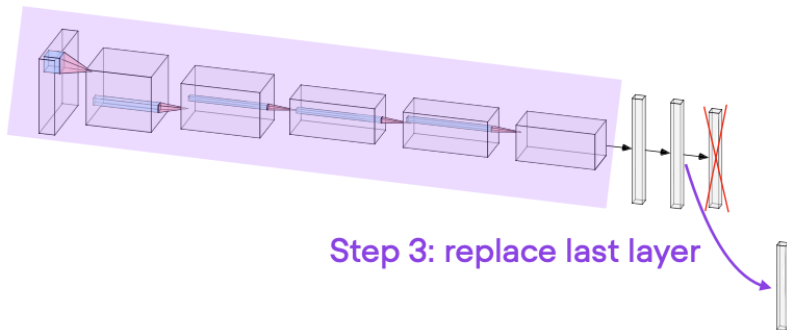
2 Fine-tune last layers

Step 2: freeze weights



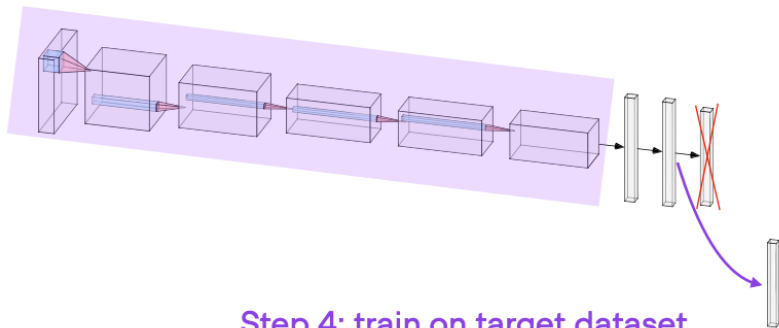
Biến thể loại 2

2 Fine-tune last layers



Biến thể loại 2

2 Fine-tune last layers

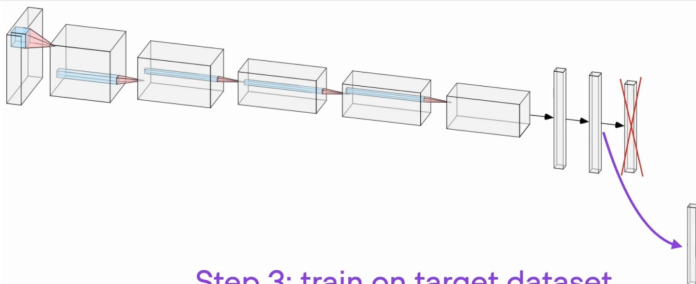


Step 4: train on target dataset,
but only update the last output layers

Biến thể loại 3

Ngoài biến thể loại 1,2, chúng ta còn có biến thể loại 3:

Fine-tuning all layers. Nghĩa là chúng ta sẽ **cập nhật trọng số** cho **toàn bộ các lớp** trong mô hình đã được huấn luyện trước đó. Cách này sẽ tốn nhiều chi phí nhưng cho kết quả tốt nhất.



Step 3: train on target dataset,
update all layers

Sử dụng Transfer Learning cho bộ dữ liệu Cifar10 với mạng ResNet với trọng số từ ImageNet

- Chúng ta sẽ sử dụng mạng ResNet đã được huấn luyện trên tập dữ liệu lớn ImageNet để giúp huấn luyện mô hình trên tập dữ liệu nhỏ hơn Cifar10 có 10 lớp.
- Chúng ta sẽ sử dụng biến thể 1: **Fine-tuning the last layer.**

Liệt kê các mô hình ResNet từ PyTorch Hub

```
1 entrypoints = torch.hub.list('pytorch/vision:v0.13.0', force_reload=True)
2 for e in entrypoints:
3     if "resnet" in e:
4         print(e)
5     #deeplabv3_resnet50
6     #deeplabv3_resnet101
7     #fcn_resnet101
8     #fcn_resnet50
9     #resnet101
```

```
10 #resnet152
11 #resnet18
12 #resnet34
13 #resnet50
14 #wide_resnet101_2
15 #wide_resnet50_2
```

Tải mô hình/trọng số Resnet18 từ PyTorch Hub. *Lưu ý: trọng số được huấn luyện trên bộ ImageNet, weights='IMAGENET1K_V1'*

```
1 pytorch_model = torch.hub.load('pytorch/vision:v0.13.0', 'resnet18',
    weights='IMAGENET1K_V1')
```

Nên in ra kiến trúc mô hình để fine-tune

```

1 print(pytorch_model)
2 # Kiến trúc mô hình ResNet18
3 ResNet(
4     (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
5         bias=False)
6     (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
7         track_running_stats=True)
8     (relu): ReLU(inplace=True)
9     (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
10         ceil_mode=False)
11     (layer1): Sequential(
12         (0): BasicBlock(
13             (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
14                 1), bias=False)
15             (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
16                 track_running_stats=True)
17             (relu): ReLU(inplace=True)
18             (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
19                 1), bias=False)
20             (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
21                 track_running_stats=True)
22         )
23     )
24     (layer2): Sequential(
25         (0): BasicBlock(
26             (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
27                 1), bias=False)
28             (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
29                 track_running_stats=True)
30             (relu): ReLU(inplace=True)
31             (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
32                 1), bias=False)
33             (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
34                 track_running_stats=True)
35         )
36     )
37     (layer3): Sequential(
38         (0): BasicBlock(
39             (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
40                 1), bias=False)
41             (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
42                 track_running_stats=True)
43             (relu): ReLU(inplace=True)
44             (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
45                 1), bias=False)
46             (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
47                 track_running_stats=True)
48         )
49     )
50     (layer4): Sequential(
51         (0): BasicBlock(
52             (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
53                 1), bias=False)
54             (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
55                 track_running_stats=True)
56             (relu): ReLU(inplace=True)
57             (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
58                 1), bias=False)
59             (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
60                 track_running_stats=True)
61         )
62     )
63     (fc): Linear(512, 1000, bias=True)
64 )

```

```

17     (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
18         1), bias=False)
19     (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
20         track_running_stats=True)
21     (relu): ReLU(inplace=True)
22     (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
23         1), bias=False)
24     (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
25         track_running_stats=True)
26 )
27 )
28 (layer2): Sequential(
29   (0): BasicBlock(
30     (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1,
31         1), bias=False)
32     (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
33         track_running_stats=True)
34     (relu): ReLU(inplace=True)
35     (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
36         1), bias=False)
37     (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
38         track_running_stats=True)
39     (downsample): Sequential(
40       (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)

```



```

33         (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
34             track_running_stats=True)
35     )
36     (1): BasicBlock(
37         (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
38             1), bias=False)
39         (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
40             track_running_stats=True)
41         (relu): ReLU(inplace=True)
42         (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
43             1), bias=False)
44         (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
45             track_running_stats=True)
46     )
47 )
48 (layer3): Sequential(
49     (0): BasicBlock(
50         (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
51             1), bias=False)
52         (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
53             track_running_stats=True)
54         (relu): ReLU(inplace=True)

```

```

49     (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
50         1), bias=False)
51     (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
52         track_running_stats=True)
53     (downsample): Sequential(
54         (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
55         (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
56             track_running_stats=True)
57     )
58     (1): BasicBlock(
59         (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
60             1), bias=False)
61         (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
62             track_running_stats=True)
63         (relu): ReLU(inplace=True)
64         (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
65             1), bias=False)
66         (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
67             track_running_stats=True)
68     )
69 )
70 (layer4): Sequential(
71     (0): BasicBlock(

```

```

66     (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1,
        1), bias=False)
67     (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
68     (relu): ReLU(inplace=True)
69     (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
        1), bias=False)
70     (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
71     (downsample): Sequential(
72         (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
73         (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
            track_running_stats=True)
74     )
75 )
76 (1): BasicBlock(
77     (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
        1), bias=False)
78     (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
79     (relu): ReLU(inplace=True)
80     (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
        1), bias=False)

```

```

81         (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
82             track_running_stats=True)
83     )
84     (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
85     (fc): Linear(in_features=512, out_features=1000, bias=True) # Chỉ cần cập
        nhật trọng số lớp cuối này, đóng băng tất cả các lớp phía trước
86 )

```

Fine-tune lớp cuối cùng và đóng băng các lớp trước bằng lệnh:

```
param.requires_grad = False
```

```

1  for param in pytorch_model.parameters():
2      param.requires_grad = False
3  pytorch_model.fc = torch.nn.Linear(512, 10) # Thay đổi số lớp đầu ra từ 1000
        -> 10 cho bộ dữ liệu Cifar10

```

Chú ý

Huấn luyện dữ liệu trên tập Cifar10 cũng phải sử dụng lại đúng các *transform* đã được sử dụng để huấn luyện mô hình ResNet18 trên tập ImageNet.

Xem lại các transform đã sử dụng cho tập ImageNet

```

1  from torchvision.models import resnet18, ResNet18_Weights
2
3  weights = ResNet18_Weights.IMAGENET1K_V1
4  preprocess_transform = weights.transforms()
5  preprocess_transform
6  #ImageClassification(
7  #   crop_size=[224]
8  #   resize_size=[256]
9  #   mean=[0.485, 0.456, 0.406]
10 #   std=[0.229, 0.224, 0.225]
11 #   interpolation=InterpolationMode.BILINEAR
12 #)

```

Áp dụng thông số *preprocess_transform* cho Cifar10

```

1  L.pytorch.seed_everything(123)
2  dm = Cifar10DataModule(batch_size=64, num_workers=4,
3      train_transform=preprocess_transform,
4      test_transform=preprocess_transform)
5  lightning_model = LightningModel(model=pytorch_model, learning_rate=0.1)
6  trainer = L.Trainer(
7      max_epochs=50,
8      accelerator="auto",
9      devices="auto",
10     logger=CSVLogger(save_dir="logs/", name="my-model"),
11     deterministic=True,
12 )

```

(Chi tiết code xem ở file
 “*Chuong_6.1_TransferLearning_Cifar10_LastLayer.pdf*”)

Trường hợp Transfer Learning sử dụng biến thể số 3: *Fine-tuning all layers* thì chúng ta sẽ **không** đóng băng trọng số nào. Cập nhật toàn bộ trọng số trong mô hình.


Fine-tune lớp cuối cùng và **không** đóng băng các lớp trước

```
1 #for param in pytorch_model.parameters():
2 # param.requires_grad = False
3 pytorch_model.fc = torch.nn.Linear(512, 10) # Thay đổi số lớp đầu ra từ 1000
    -> 10 cho bộ dữ liệu Cifar10
```

Tài liệu tham khảo

 Rafael C. Gonzalez, Richard E. Woods

Digital Image Processing (2018), Fourth Edition, Global Edition, Pearson.

 Sebastian Raschka, Yuxi (Hayden) Liu, Vahid Mirjalili
Machine Learning with PyTorch and Scikit-Learn: Develop machine learning and deep learning models with Python (2022). Published by Packt Publishing Ltd, ISBN 978-1-80181-931-2.

 LightningAI

LightningAI: PyTorch Lightning (2024) .