

AI VIETNAM

Khóa học tất cả trong một

Mạng lưới thần kinh chuyển đổi

Động lực và
Giới thiệu về CNN

Quang-Vinh Dinh
Ph.D. in Computer Science

Đề cương

Hạn chế của MLP

Từ MLP đến CNN

Lấy mẫu xuống bản đồ tính năng

Một số ví dụ

Ứng dụng cho Cifar10

Tập dữ liệu thời trang-MNIST

Hình ảnh thang độ xám

Độ phân giải=28x28

Bộ huấn luyện: 60000 mẫu

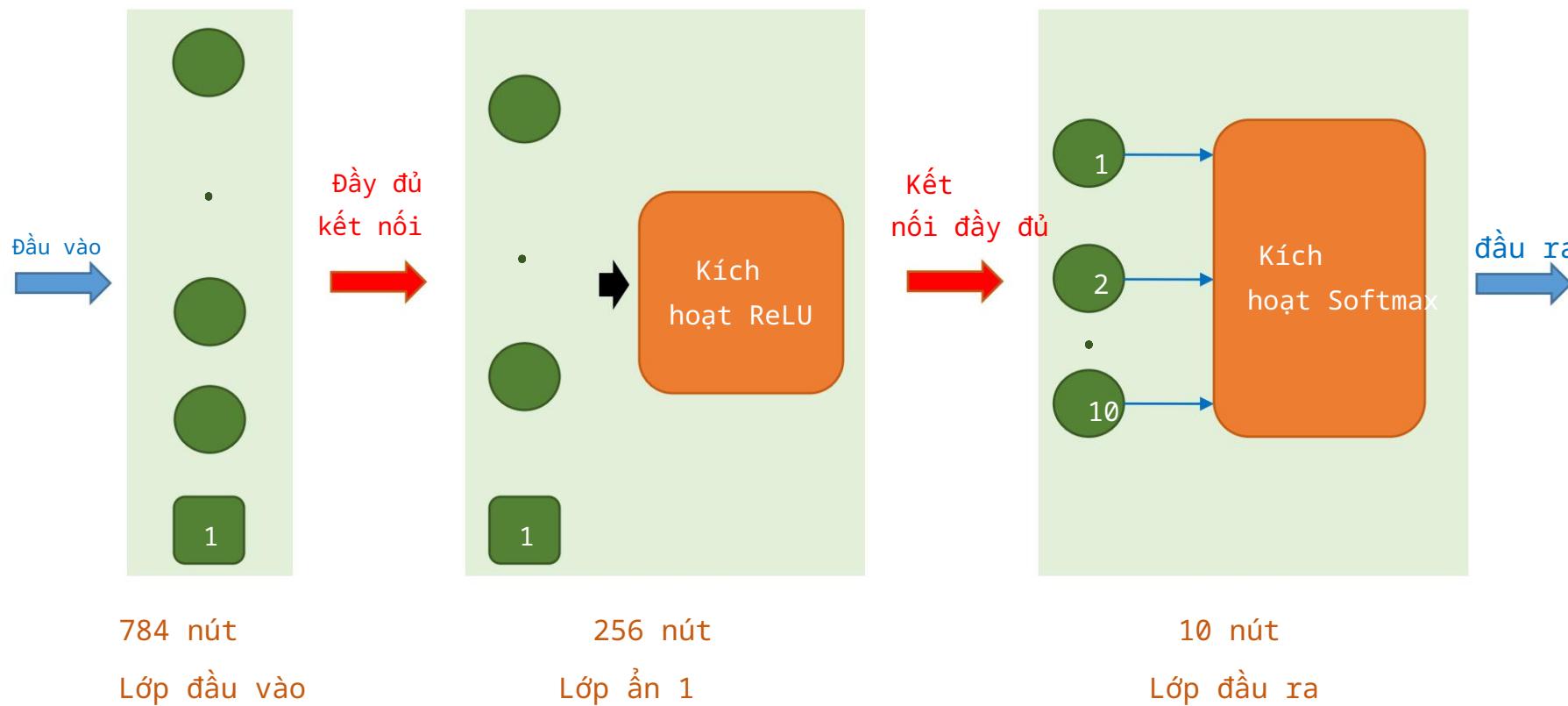
Bộ thử nghiệm: 10000 mẫu



MLP cho Thời trang-MNIST

Trường hợp 1

ReLU, Anh và Adam



MLP cho Thời trang-MNIST

Trường hợp 1

ReLU, Anh và Adam

```
# model
model = nn.Sequential(
    nn.Flatten(),
    nn.Linear(784, 256),
    nn.ReLU(),
    nn.Linear(256, 10)
)
```

```
# Initialize the weights
for layer in model:
    if isinstance(layer, nn.Linear):
        init.kaiming_uniform_(layer.weight,
                              nonlinearity='relu')
    if layer.bias is not None:
        layer.bias.data.fill_(0)
```

```
# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(),
                      lr=0.001)
```

```
# Load CFashionMNIST dataset
transform = Compose([transforms.ToTensor(),
                     transforms.Normalize((0.5, ),
                                         (0.5,))])

trainset = FashionMNIST(root='data',
                        train=True,
                        download=True,
                        transform=transform)

trainloader = DataLoader(trainset,
                        batch_size=1024,
                        num_workers=10,
                        shuffle=True,
                        drop_last=True)

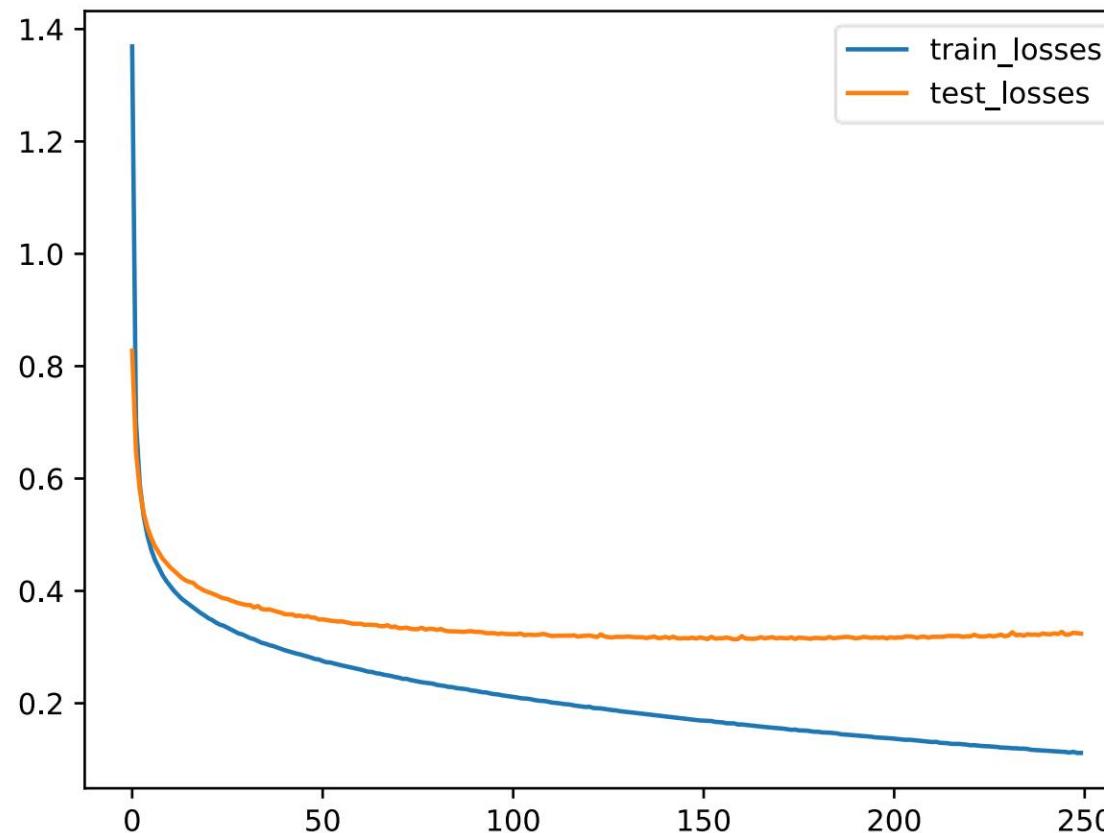
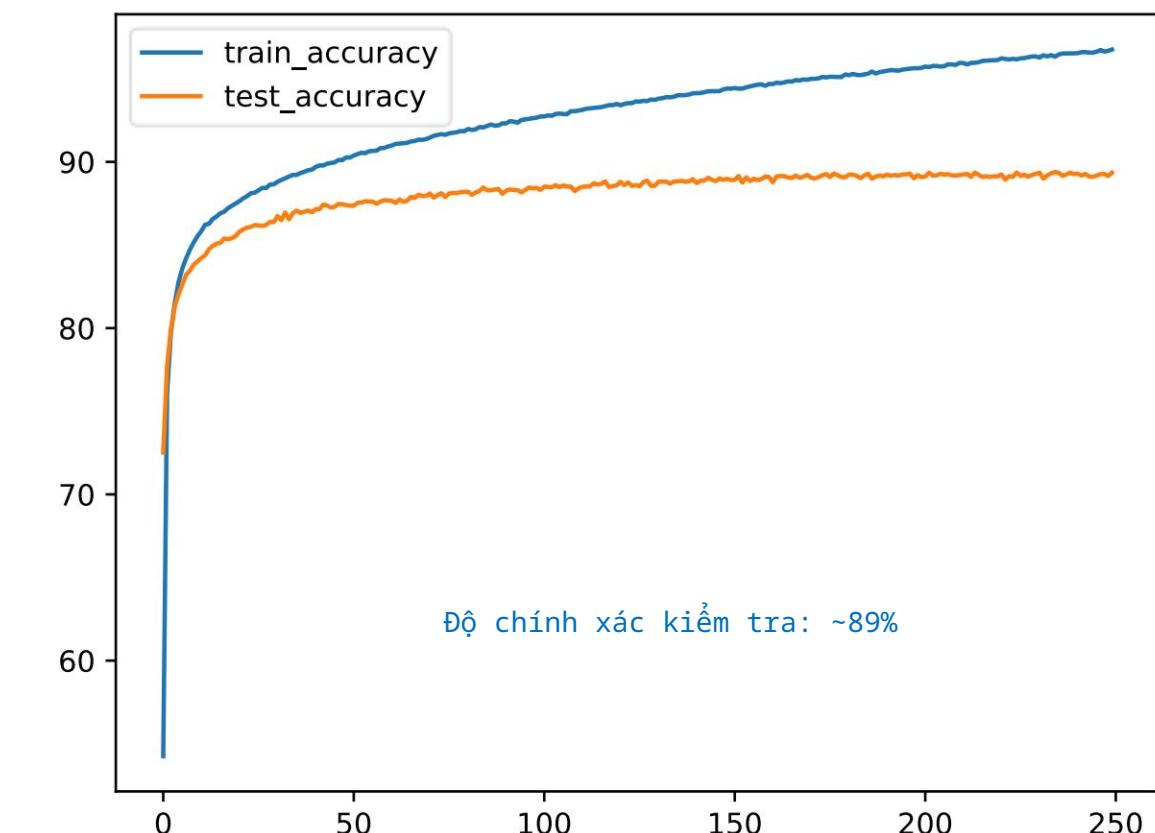
testset = FashionMNIST(root='data',
                       train=False,
                       download=True,
                       transform=transform)

testloader = DataLoader(testset,
                        batch_size=1024,
                        num_workers=10,
                        shuffle=False)
```

MLP cho Thời trang-MNIST

Trường hợp 1

ReLU, Anh và Adam

Adam với tốc độ học tập là $1e-4$ 

Độ chính xác kiểm tra: ~89%

Thực hiện hợp lý

Bộ dữ liệu Cifar-10

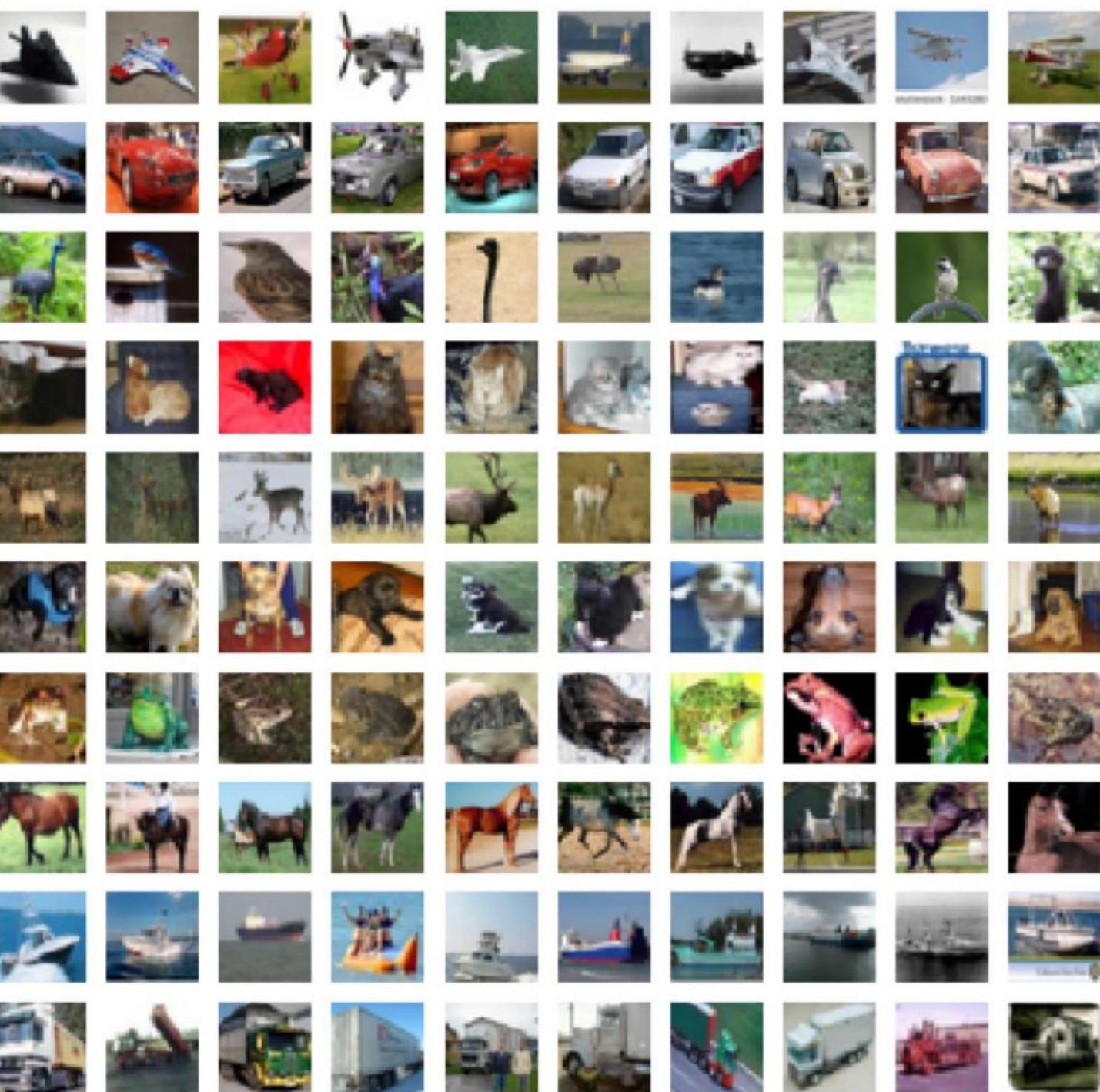
Hình ảnh màu

Độ phân giải=32x32

Bộ huấn luyện: 50000 mẫu

Bộ thử nghiệm: 10000 mẫu

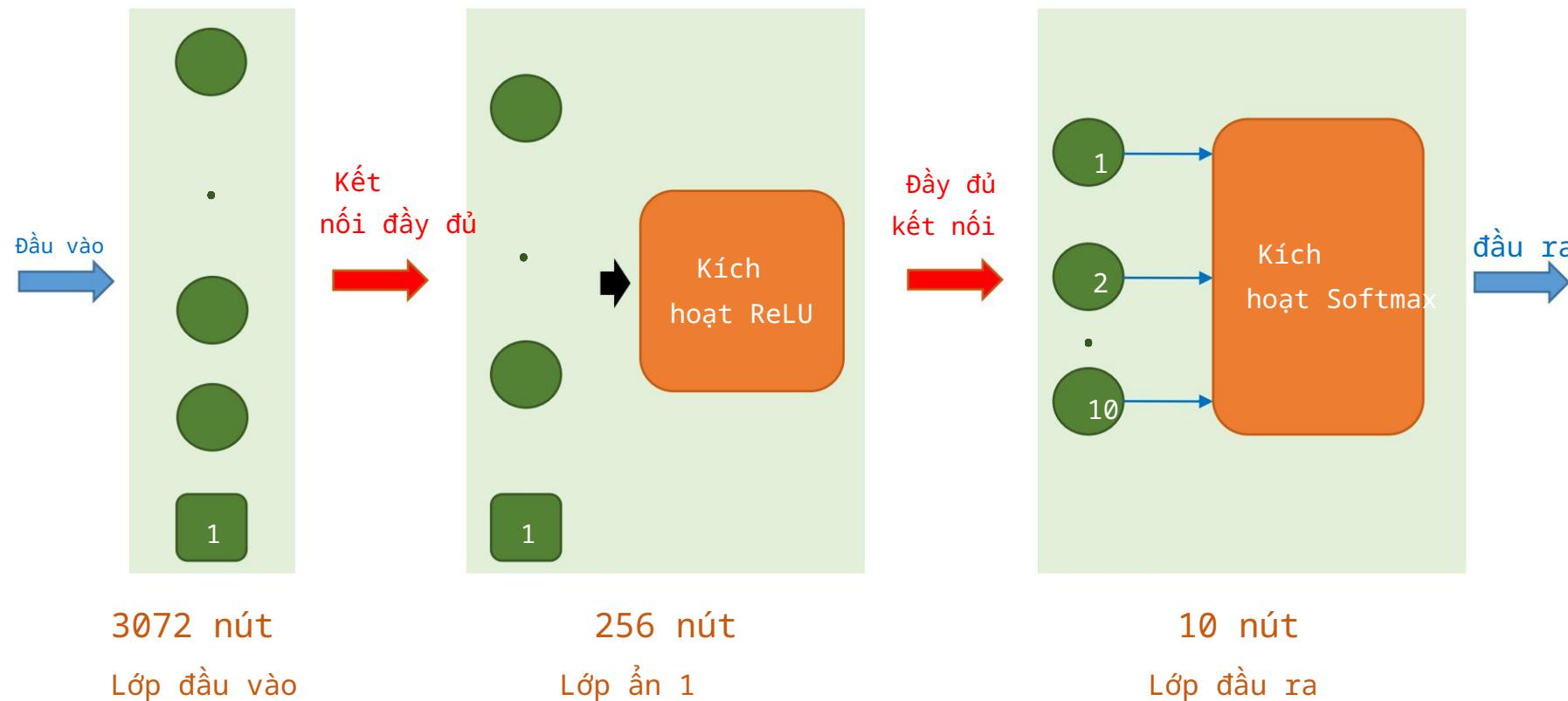
Máy bay



MLP cho Cifar-10

Trường hợp 2

ReLU, Anh và Adam



MLP cho Cifar-10

Trường hợp 2

ReLU, Anh và Adam

```
# model
model = nn.Sequential(
    nn.Flatten(),
    nn.Linear(32*32*3, 256),
    nn.ReLU(),
    nn.Linear(256, 10)
)

# Initialize the weights
for layer in model:
    if isinstance(layer, nn.Linear):
        init.kaiming_uniform_(layer.weight,
                              nonlinearity='relu')
        if layer.bias is not None:
            layer.bias.data.fill_(0)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(),
                      lr=0.001)
```

```
# Load CIFAR10 dataset
transform = Compose([ToTensor(),
                     Normalize((0.5,0.5, 0.5),
                               (0.5,0.5, 0.5))])

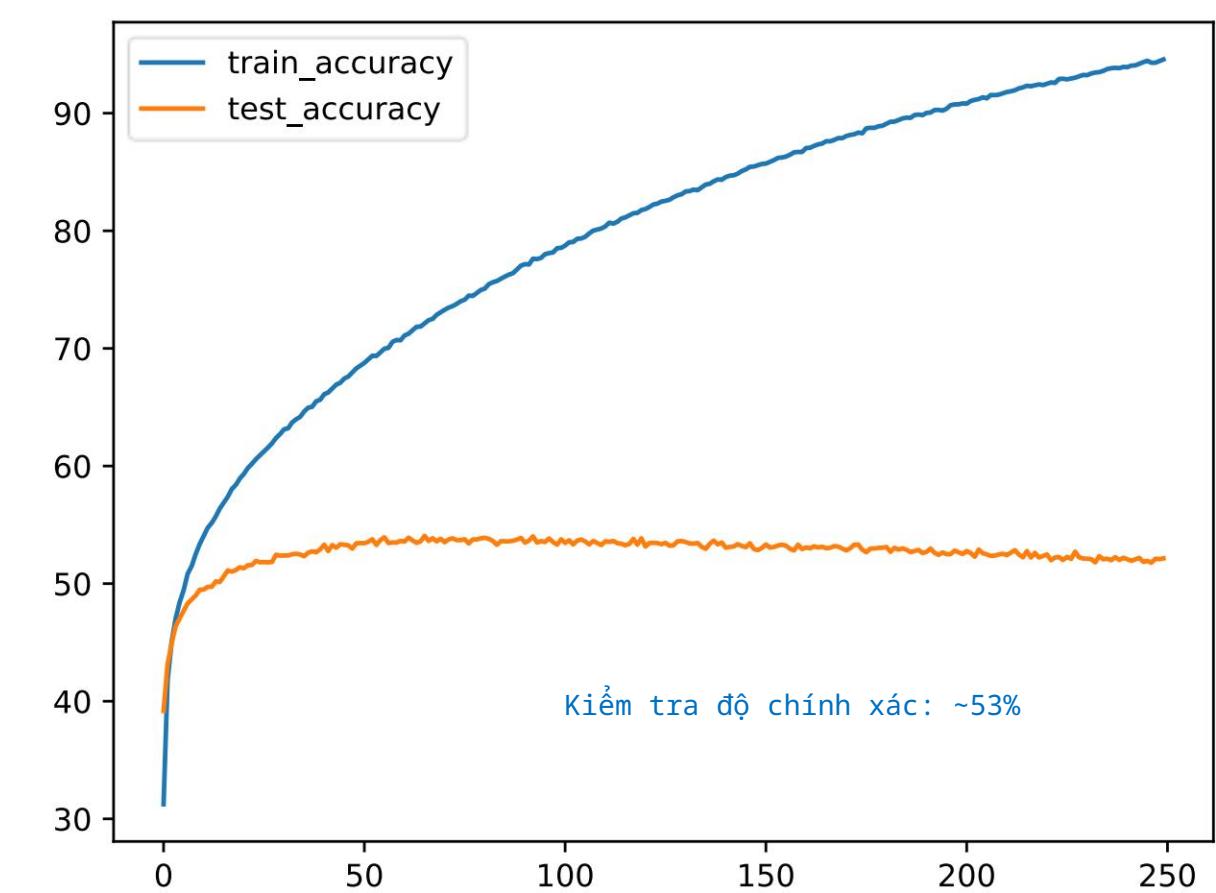
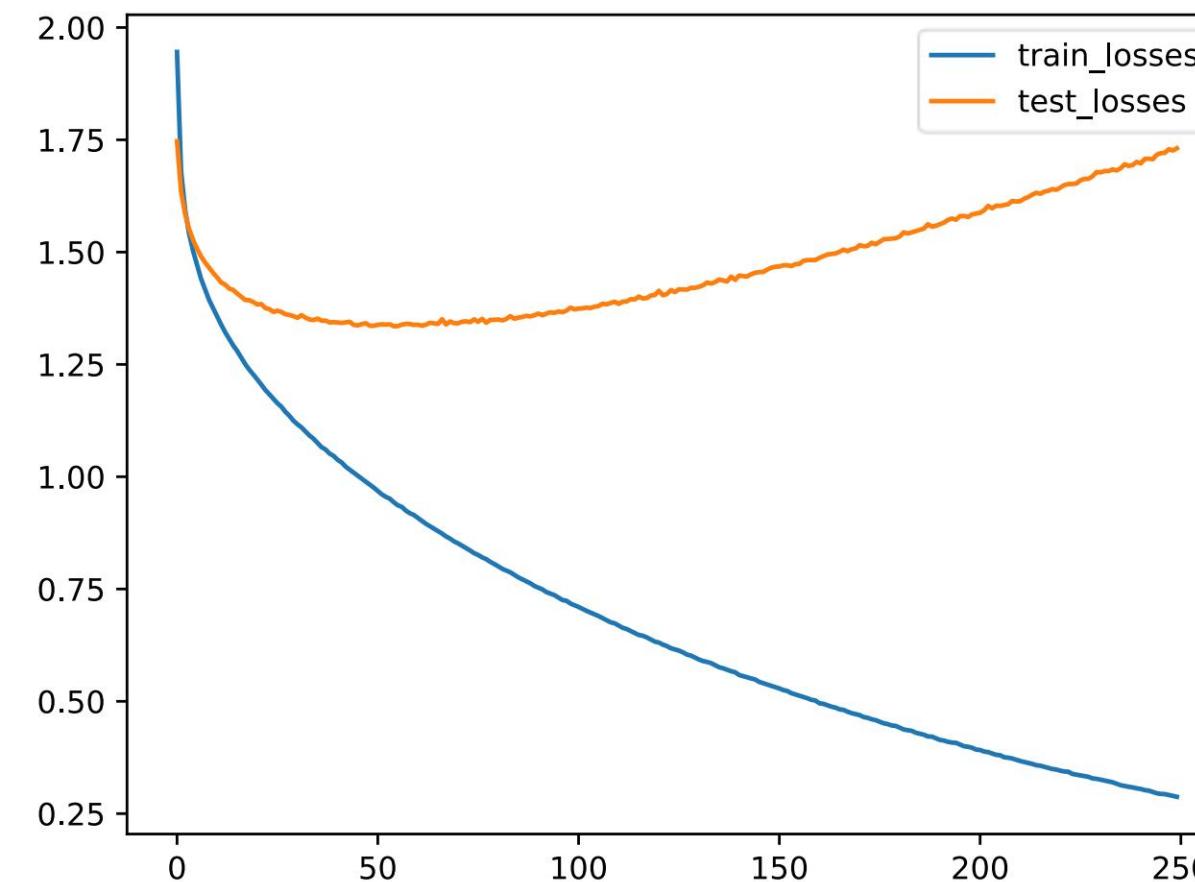
trainset = CIFAR10(root='data',
                    train=True,
                    download=True,
                    transform=transform)
trainloader = DataLoader(trainset,
                        batch_size=1024,
                        num_workers=10,
                        shuffle=True,
                        drop_last=True)

testset = CIFAR10(root='data',
                  train=False,
                  download=True,
                  transform=transform)
testloader = DataLoader(testset,
                        batch_size=1024,
                        num_workers=10,
                        shuffle=False)
```

MLP cho Cifar-10

Trường hợp 2

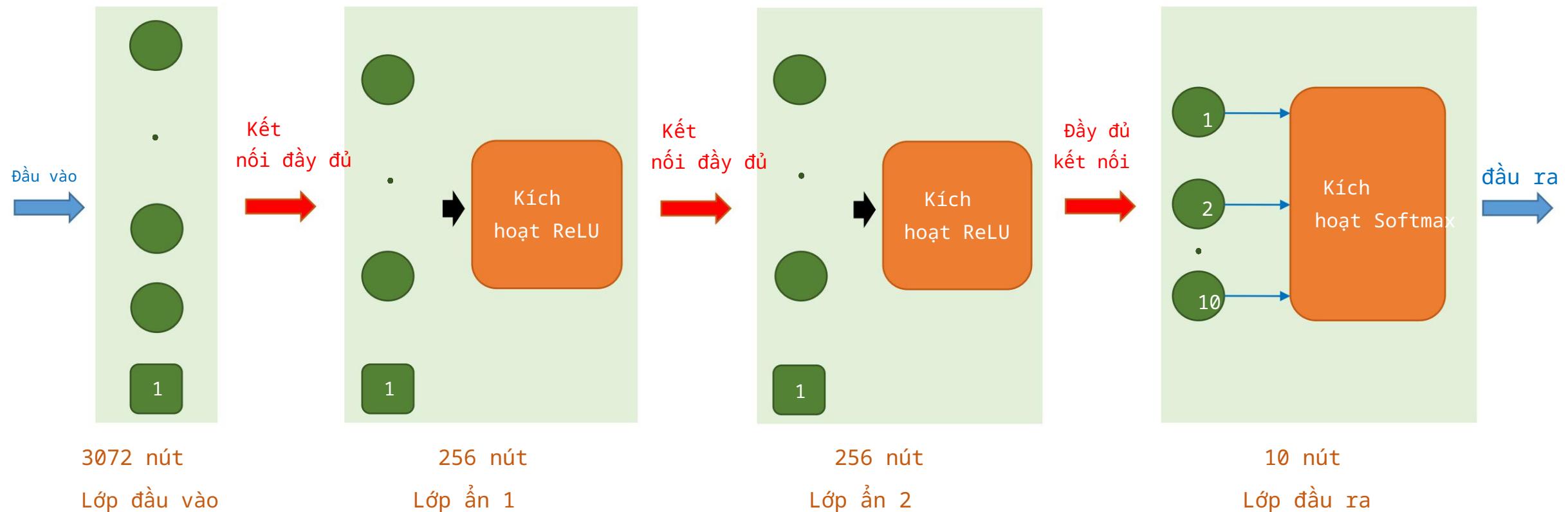
ReLU, Anh và Adam



MLP cho Cifar-10

Trường hợp 3

ReLU, He và Adam: thêm nhiều lớp hơn n



MLP cho Cifar-10

Trường hợp 3

ReLU, Anh và Adam

```
# model
model = nn.Sequential(
    nn.Flatten(),
    nn.Linear(32*32*3, 256),
    nn.ReLU(),
    nn.Linear(256, 256),
    nn.ReLU(),
    nn.Linear(256, 10)
)

# Initialize the weights
for layer in model:
    if isinstance(layer, nn.Linear):
        init.kaiming_uniform_(layer.weight,
                              nonlinearity='relu')
        if layer.bias is not None:
            layer.bias.data.fill_(0)

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

```
# Load CIFAR10 dataset
transform = Compose([ToTensor(),
                     Normalize((0.5,0.5, 0.5),
                               (0.5,0.5, 0.5))])

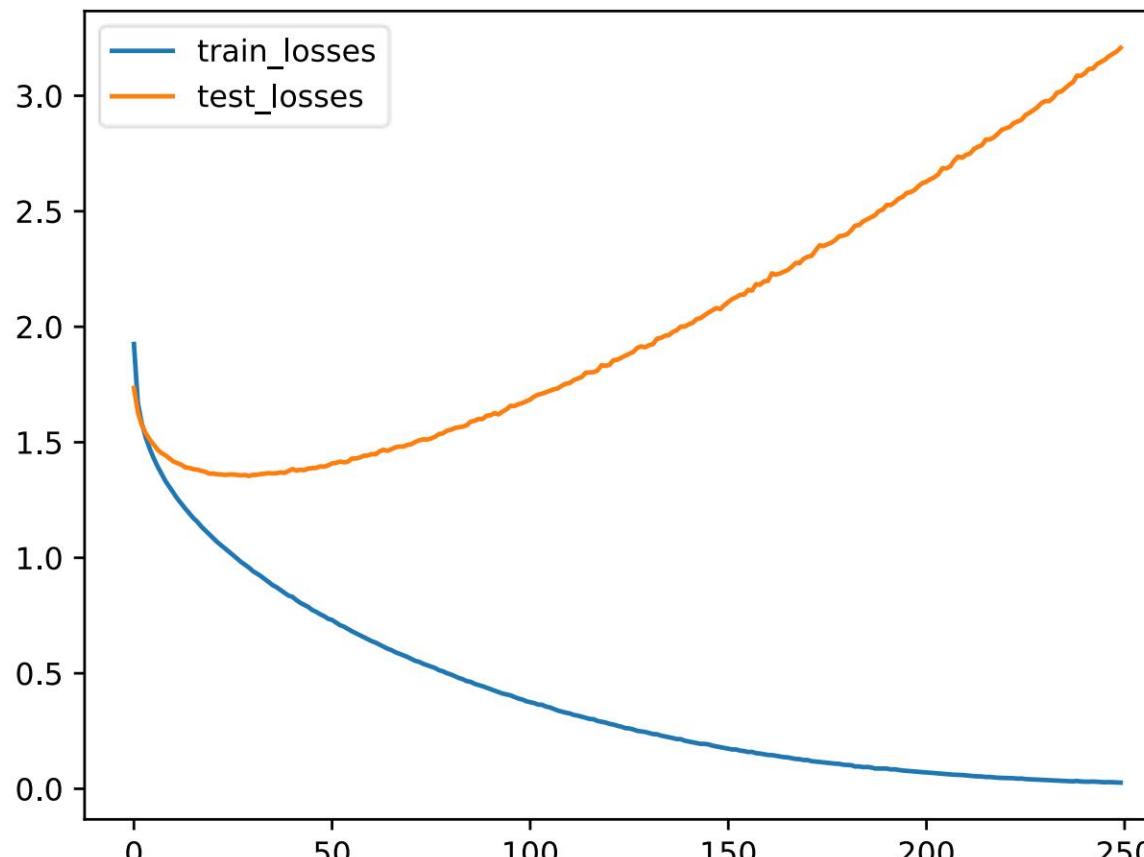
trainset = CIFAR10(root='data',
                    train=True,
                    download=True,
                    transform=transform)
trainloader = DataLoader(trainset,
                        batch_size=1024,
                        num_workers=10,
                        shuffle=True,
                        drop_last=True)

testset = CIFAR10(root='data',
                  train=False,
                  download=True,
                  transform=transform)
testloader = DataLoader(testset,
                        batch_size=1024,
                        num_workers=10,
                        shuffle=False)
```

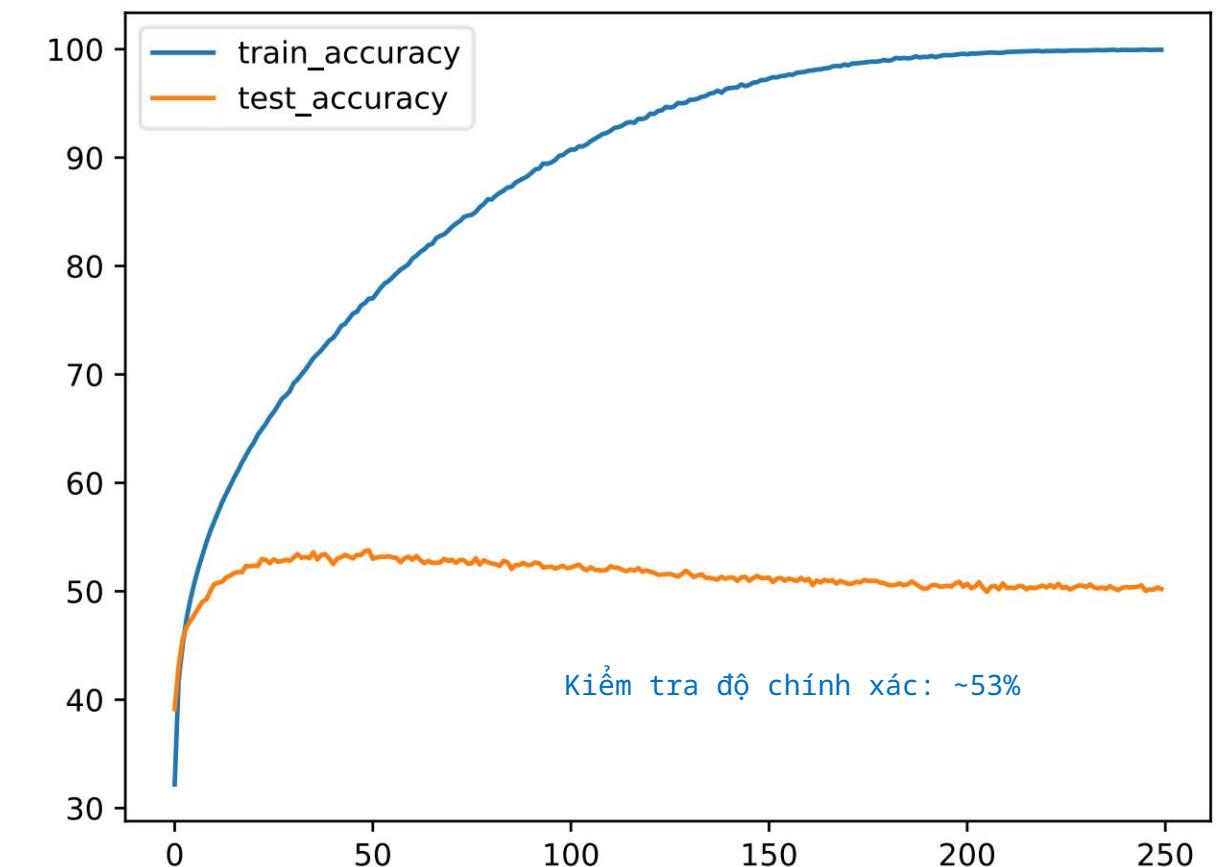
MLP cho Cifar-10

Trường hợp 3

ReLU, Anh và Adam



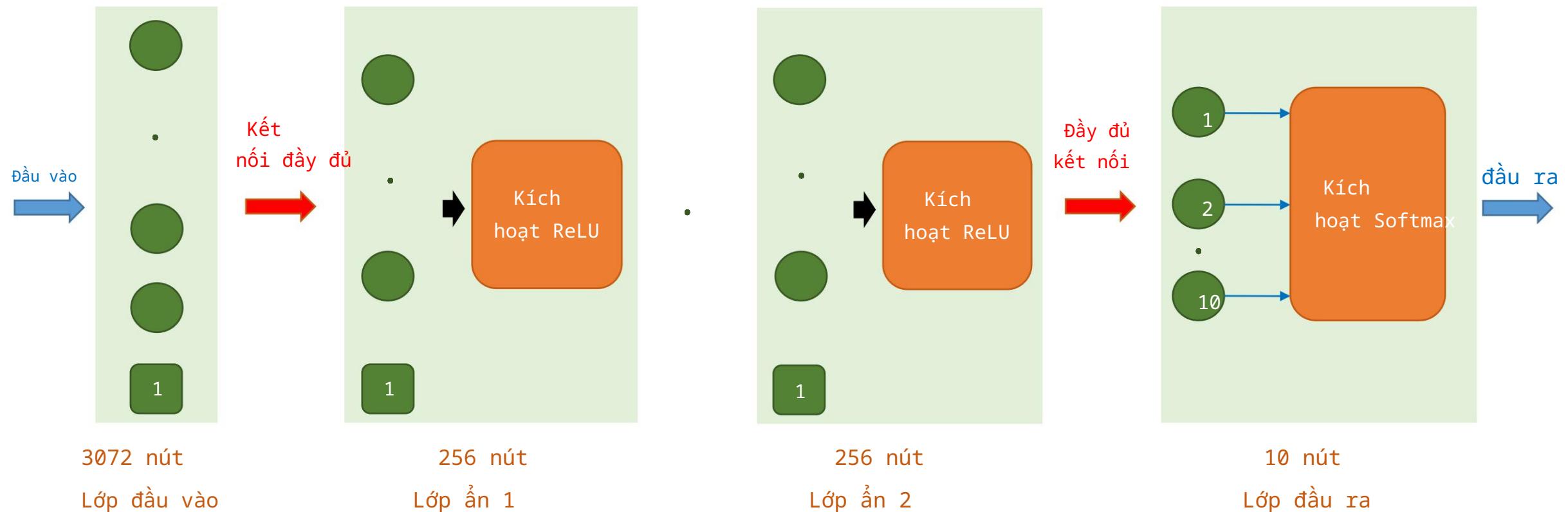
Văn hoạt động kém



MLP cho Cifar-10

Trường hợp 4

ReLU, Anh và Adam



MLP cho Cifar-10

Trường hợp 4

ReLU, Anh và Adam

```
# model
model = nn.Sequential(
    nn.Flatten(), nn.Linear(32*32*3, 256),
    nn.ReLU(), nn.Linear(256, 256),
    nn.ReLU(), nn.Linear(256, 256),
    nn.ReLU(), nn.Linear(256, 10)
)

# Initialize the weights
for layer in model:
    if isinstance(layer, nn.Linear):
        init.kaiming_uniform_(layer.weight,
                              nonlinearity='relu')
        if layer.bias is not None:
            layer.bias.data.fill_(0)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(),
                      lr=0.001)
```

```
# Load CIFAR10 dataset
transform = Compose([ToTensor(),
                     Normalize((0.5,0.5, 0.5),
                               (0.5,0.5, 0.5))])

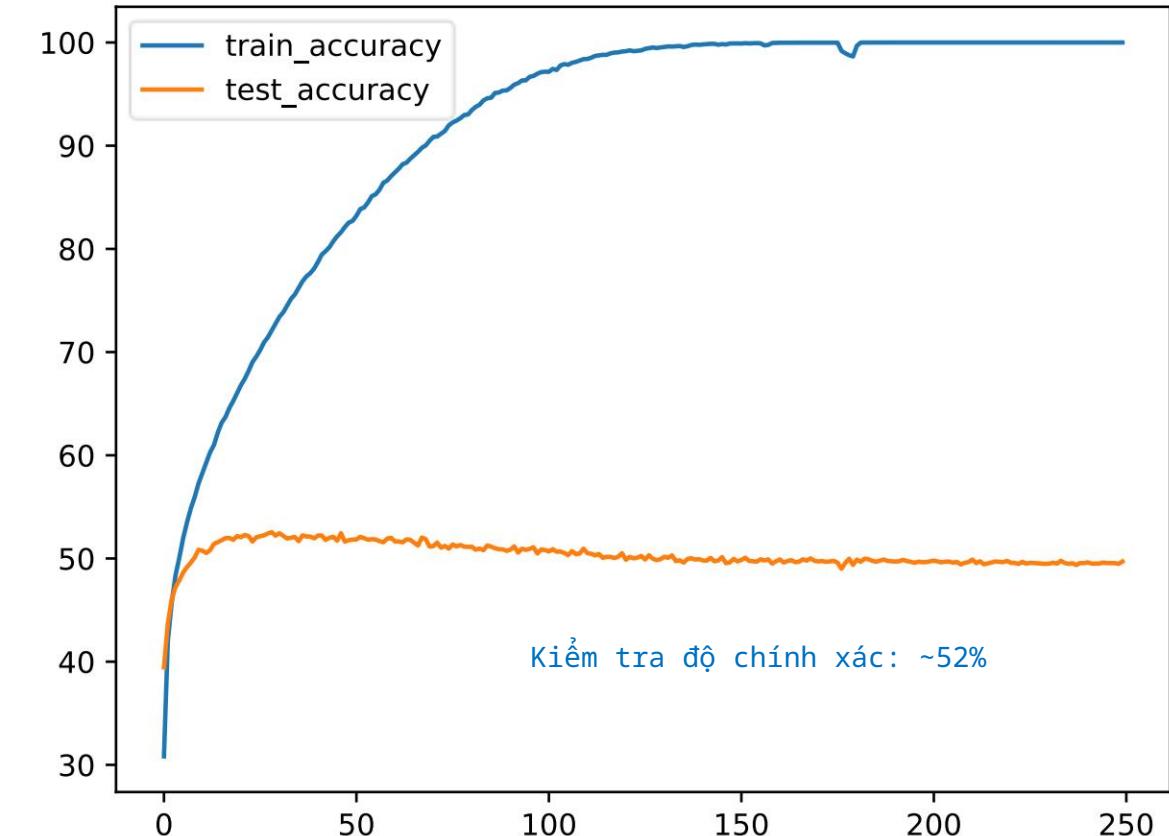
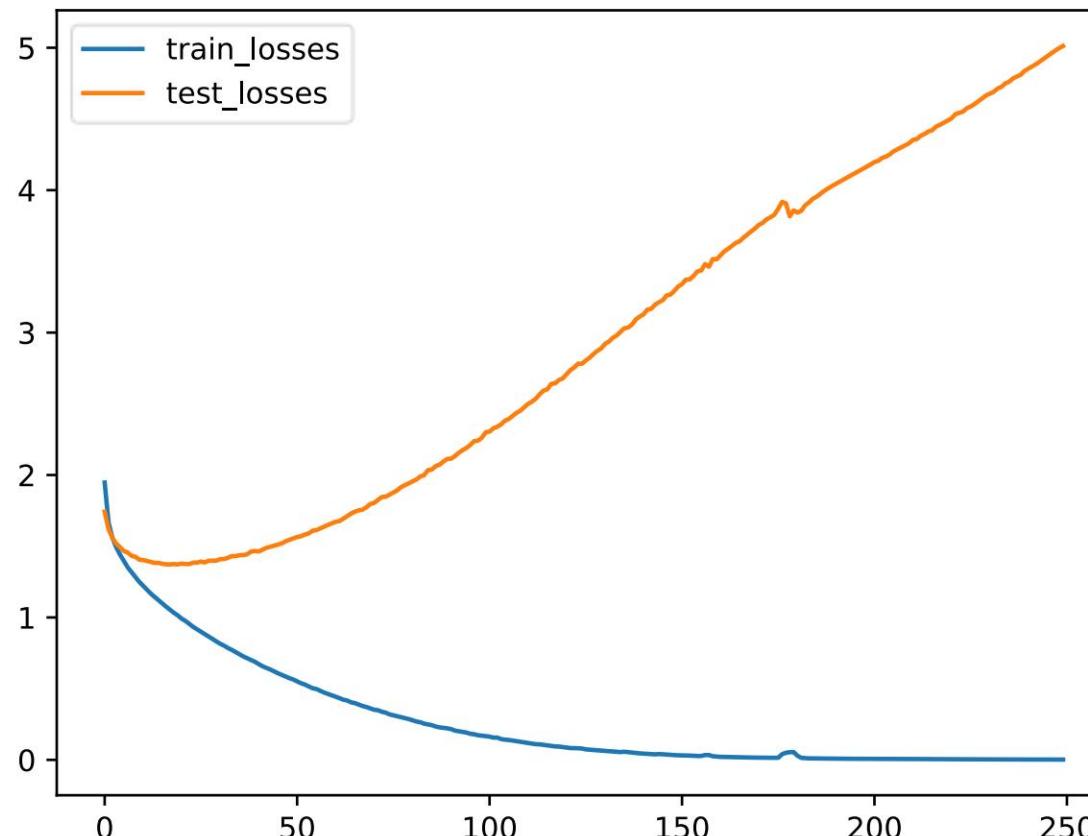
trainset = CIFAR10(root='data',
                    train=True,
                    download=True,
                    transform=transform)
trainloader = DataLoader(trainset,
                        batch_size=1024,
                        num_workers=10,
                        shuffle=True,
                        drop_last=True)

testset = CIFAR10(root='data',
                  train=False,
                  download=True,
                  transform=transform)
testloader = DataLoader(testset,
                        batch_size=1024,
                        num_workers=10,
                        shuffle=False)
```

MLP cho Cifar-10

Trường hợp 4

ReLU, He và Adam: Sử dụng 3 lớp ẩn



Đề cương

Hạn chế của MLP

Từ MLP đến CNN

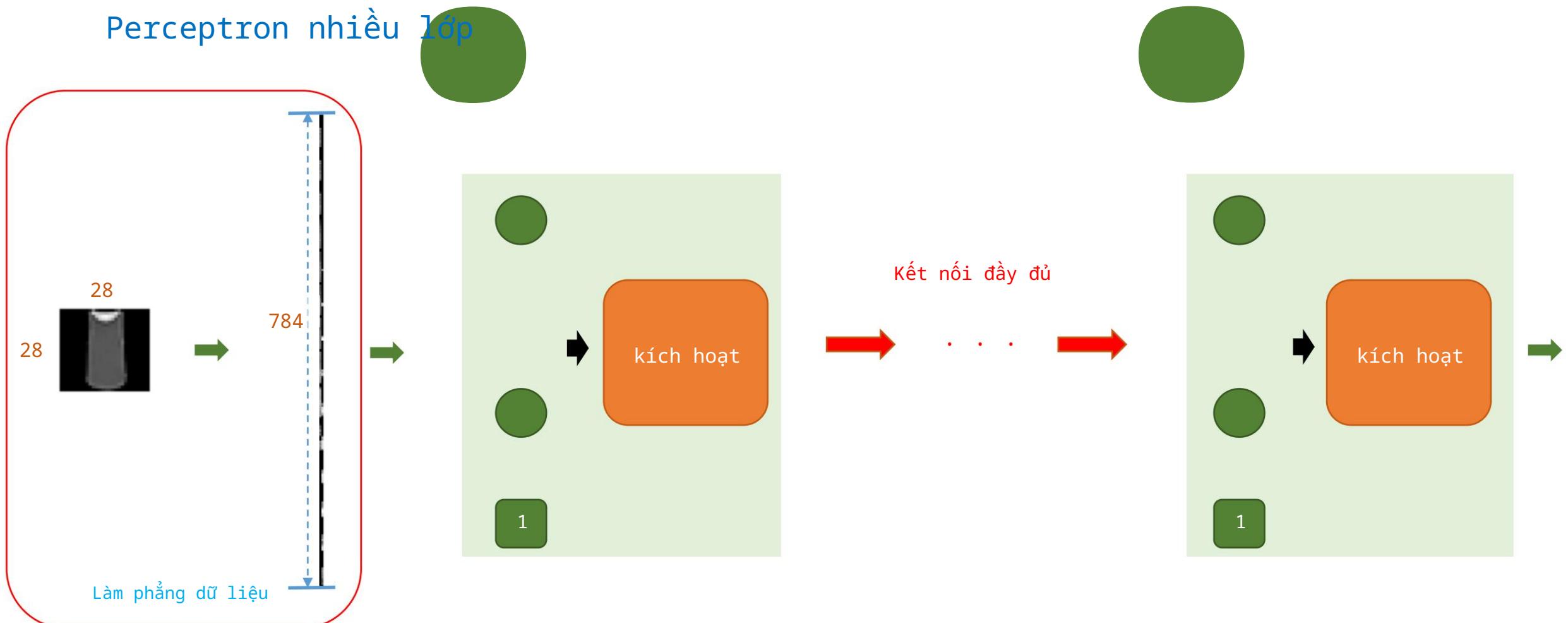
Lấy mẫu xuống bản đồ tính năng

Một số ví dụ

Ứng dụng cho Cifar10

Từ MLP tới CNN

Perceptron nhiều lớp

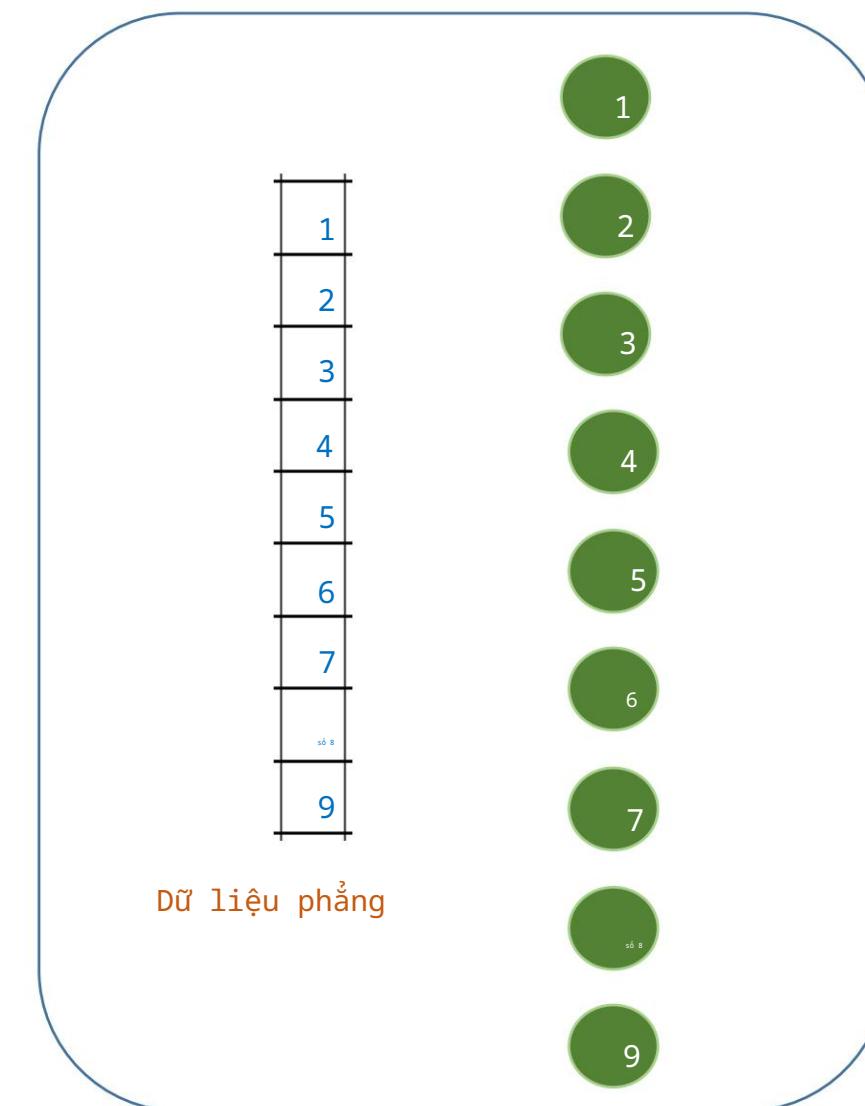
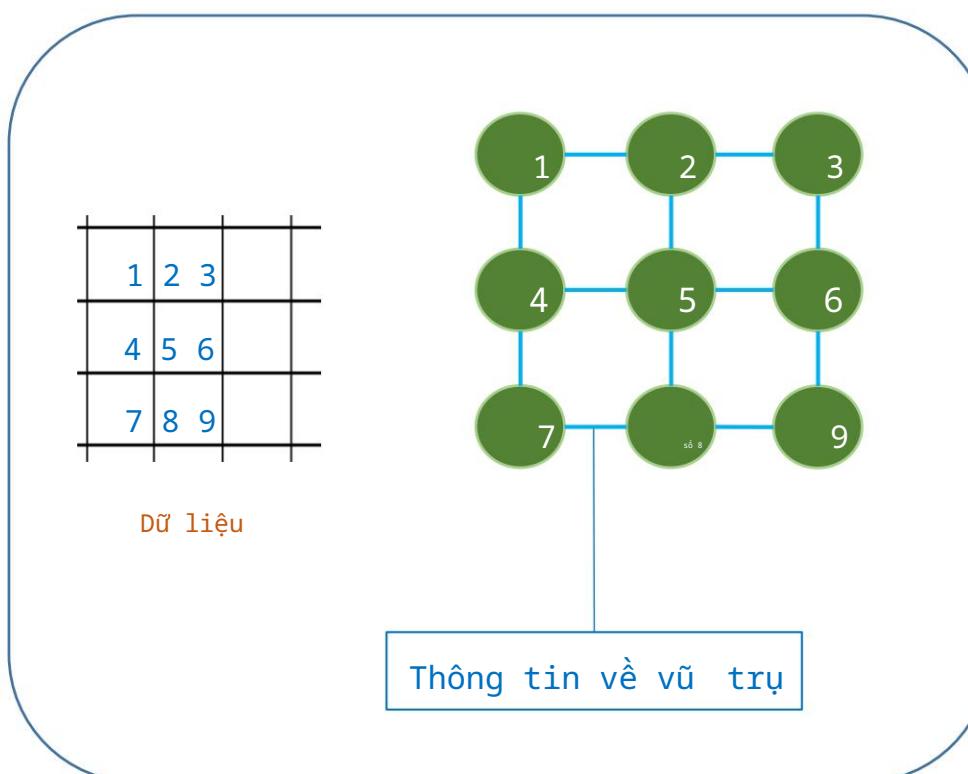


Vấn đề: Xóa thông tin không gian của dữ liệu

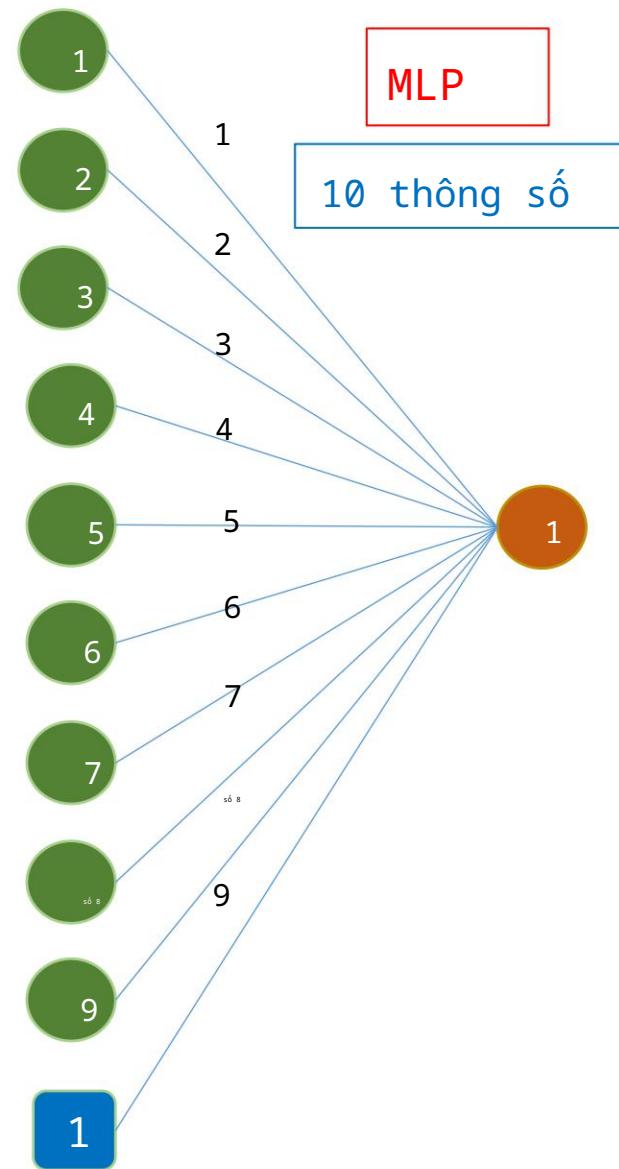
Không hiệu quả có một lượng lớn các tham số

Từ MLP tới CNN

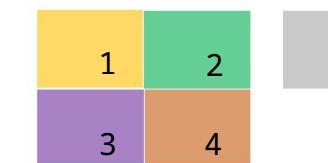
Vấn đề làm phẳng dữ liệu



Từ MLP tới CNN

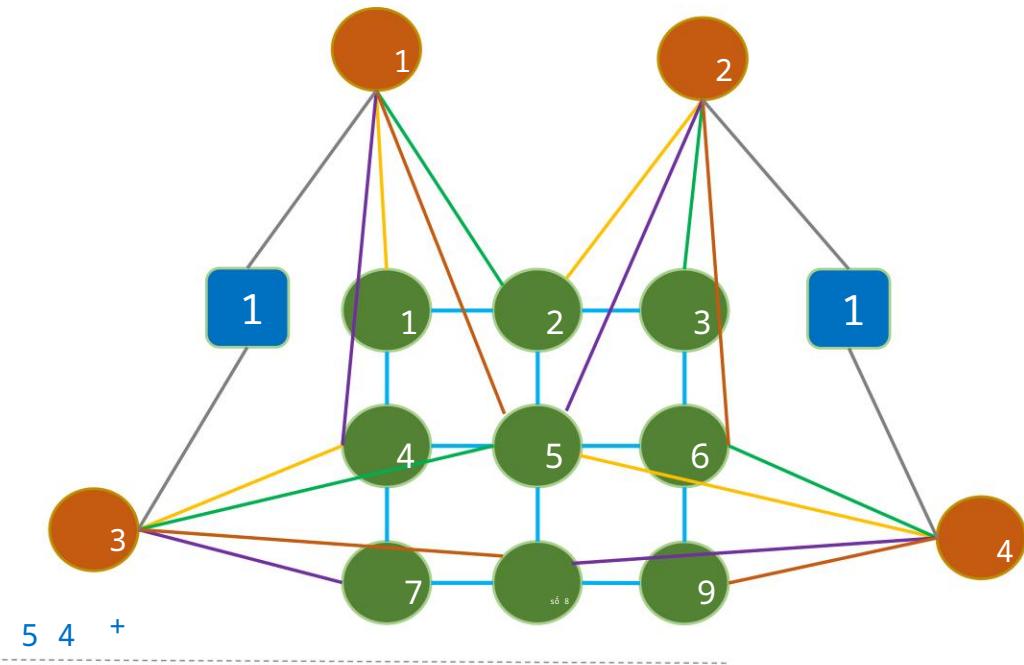
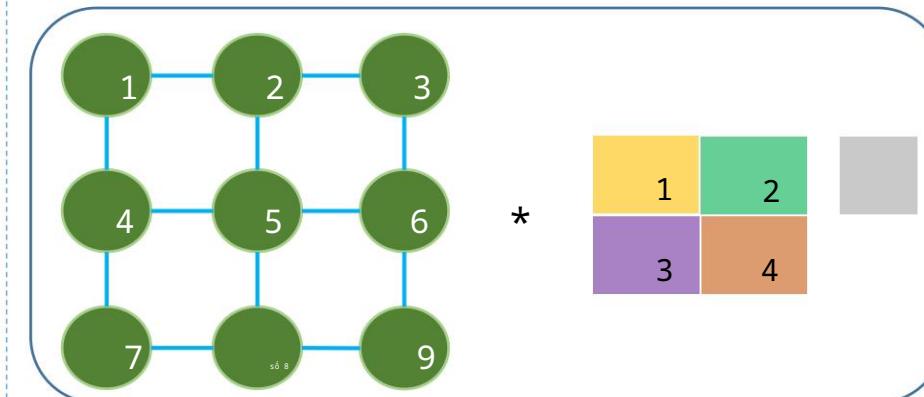


Thông số CNN 5



Hạt nhân của tham số

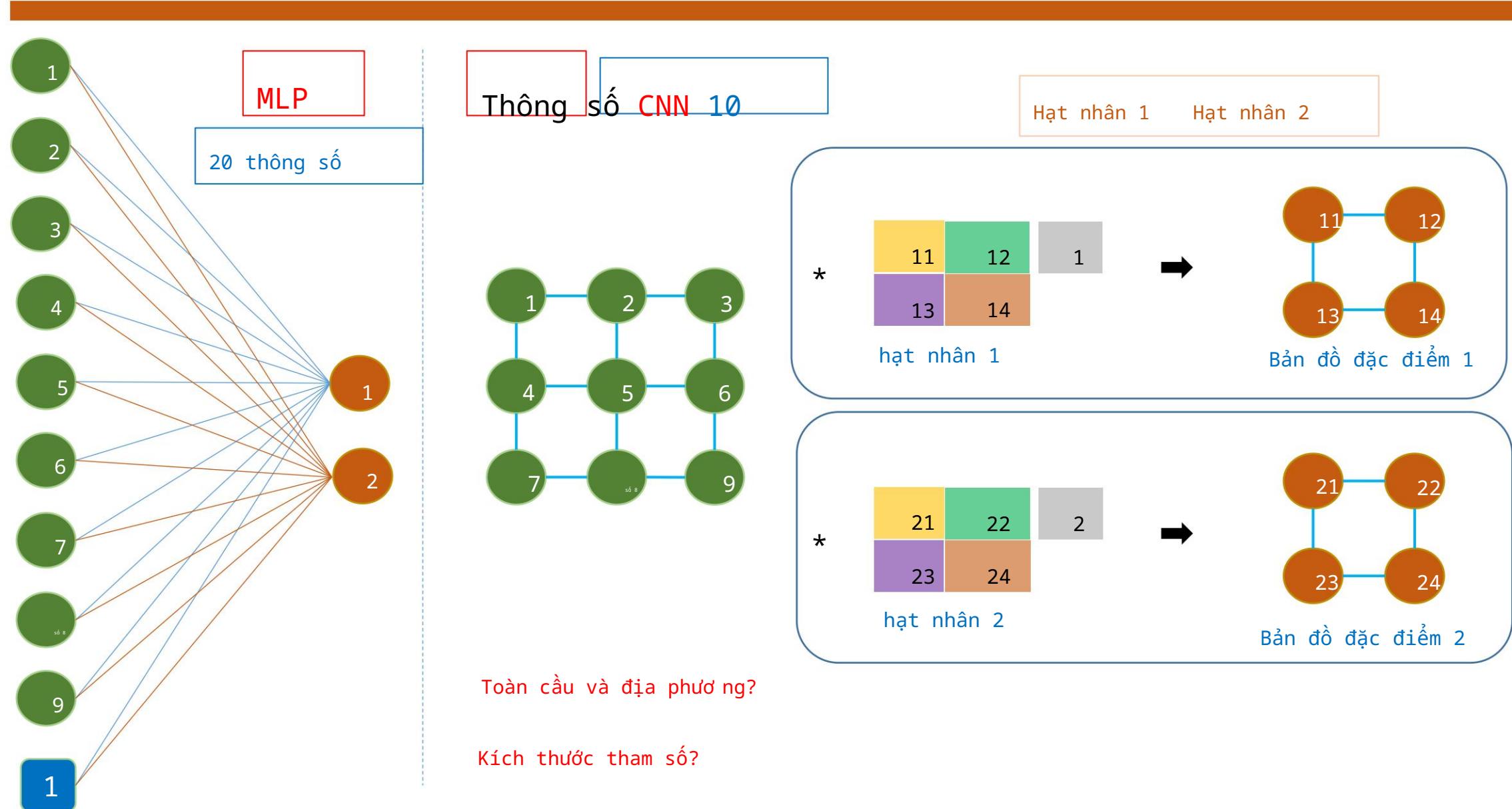
$$1 = 1 \ 1 + 2 \ 2 + 4 \ 3 + 5 \ 4 +$$



Bản đồ đặc điểm

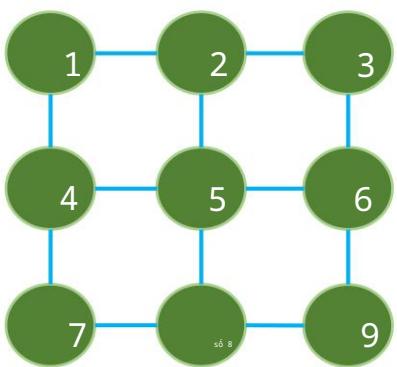
Tích chập (=Tương quan)

Từ MLP tới CNN



Từ MLP tới CNN

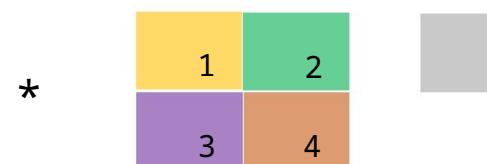
Hiệu tích chập



Hình dạng=(1,3,3)

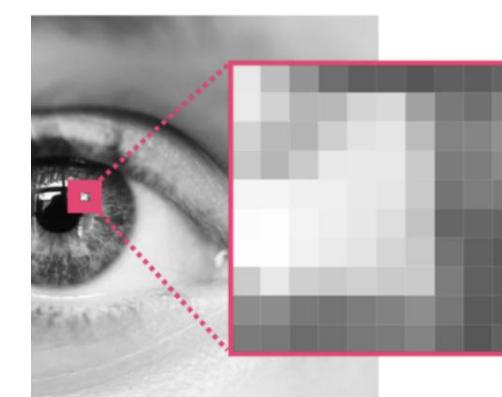
(Kênh=1, Chiều cao=3, Chiều rộng=3)

#kênh dữ liệu = #kênh của kernel phải

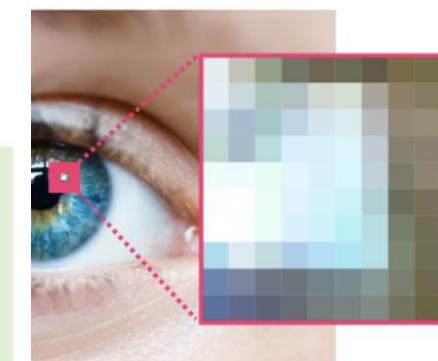


Hình dạng=(1,2,2)

#tham số (+độ lệch) = 5



230	194	147	108	90	98	84	96	91	101
237	206	188	195	207	213	163	123	116	128
210	183	180	205	224	234	188	122	134	147
198	189	201	227	229	232	200	125	127	135
249	241	237	244	232	226	202	116	125	126
251	254	241	239	230	217	196	102	103	99
243	255	240	231	227	214	203	116	95	91
204	231	208	200	207	201	200	121	95	95
144	140	120	115	125	127	143	118	92	91
121	121	108	109	122	121	134	106	86	97



233	188	137	96	90	95	63	73	73	82
237	202	159	120	105	110	88	107	112	121
226	191	147	110	101	112	98	123	110	119
221	191	176	182	203	214	169	144	133	145
185	160	161	184	205	223	186	137	147	161
181	174	189	207	206	215	194	136	142	151
246	237	237	231	208	206	192	122	143	144
254	254	241	224	199	192	181	99	122	117
239	248	232	207	187	182	184	110	114	110
193	215	193	167	158	164	181	114	112	111
113	119	110	111	113	123	135	120	108	106
93	97	91	103	107	111	122	112	104	114

tích chập

Có bao nhiêu trường hợp?

0	0	1	2	2	
1	2	2	1	2	
0	2	0	2	1	
0	1	1	1	0	
1	0	0	0	1	

Dữ liệu D

0,0	0,1	-0,1		
-0,2	0,0	0,1		
0,0	0,0	0,1		

hạt nhân K

0	0	1	2	2	
1	2	2	1	2	
0	2	0	2	1	
0	1	1	1	0	
1	0	0	0	1	
0	0	1	2	2	
1	2	2	1	2	
0	2	0	2	1	
0	1	1	1	0	
1	0	0	0	1	
0	0	1	2	2	
1	2	2	1	2	
0	2	0	2	1	
0	1	1	1	0	
1	0	0	0	1	

Độ lệch b = 0,0

tích chập

Ví dụ

0	0	1	2	2
1	2	2	1	2
0	2	0	2	1
0	1	1	1	0
1	0	0	0	1

Dữ liệu D

Độ lệch b = 0,0

0,0	0,1	-0,1
-0,2	0,0	0,1
0,0	0,0	0,1

hạt nhân K

1		

đầu ra

Kích thước dữ liệu = 5x5

$$= 0 \times 0,0 + 0 \times 0,1 + 1 \times 0,1 + 1$$

Kích thước hạt nhân = 3x3

$$1 \times 0,2 + 2 \times 0,0 + 2 \times 0,1 +$$

Sải bước = 1

$$0 \times 0,0 + 2 \times 0,0 + 0 \times 0,1$$



$$1 = 0,1$$

tích chập

Ví dụ

0	0	1	2	2
1	2	2	1	2
0	2	0	2	1
0	1	1	1	0
1	0	0	0	1

Dữ liệu D

Độ lệch b = 0,0

0,0	0,1	-0,1
-0,2	0,0	0,1
0,0	0,0	0,1

hạt nhân K

-0,1	-0,1	-0,2
0,3	-0,2	0,1
0,3	-0,3	0,1

đầu ra

Kích thước dữ liệu = 5×5

Kích thước hạt nhân = 3×3

Sải bước = 1

$$= \left[\quad \right] + 1$$

tích chập

Ví dụ

0	0	1	2	2
1	2	2	1	2
0	2	0	2	1
0	1	1	1	0
1	0	0	0	1

Dữ liệu D

Độ lệch b = 0,0

0,0	0,1	-0,1
-0,2	0,0	0,1
0,0	0,0	0,1

hạt nhân K

1	
---	--

đầu ra

Kích thước dữ liệu = 5×5

$$= 0 \times 0,0 + 0 \times 0,1 + 1 \times 0,1 + 1$$

Kích thước hạt nhân = 3×3

$$1 \times 0,2 + 2 \times 0,0 + 2 \times 0,1 +$$

Sải bước = 2

$$0 \times 0,0 + 2 \times 0,0 + 0 \times 0,1$$



$$1 = 0,1$$

tích chập

Ví dụ

0	0	1	2	2	
1	2	2	1	2	
0	2	0	2	1	
0	1	1	1	0	
1	0	0	0	1	

Dữ liệu D

Độ lệch b = 0,0

0,0	0,1	-0,1
-0,2	0,0	0,1
0,0	0,0	0,1

hạt nhân K

-0,1	-0,2
0,3	0,1

đầu ra

Kích thước dữ liệu = 5×5

Kích thước hạt nhân = 3×3

Sải bước = 2

Mạng lưới thần kinh chuyển đổi

Hiệu tích chập



Kết hợp với 1
kernel $(3, 5, 5)$

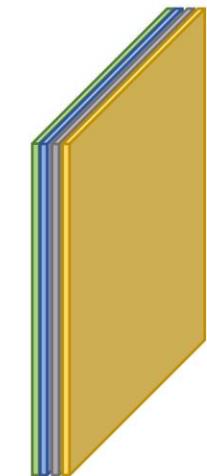


Dữ liệu đầu
vào $(3, 32, 32)$

Bản đồ đặc
điểm $(1, 28, 28)$



Kết hợp với 4
hạt nhân $(3, 5, 5)$

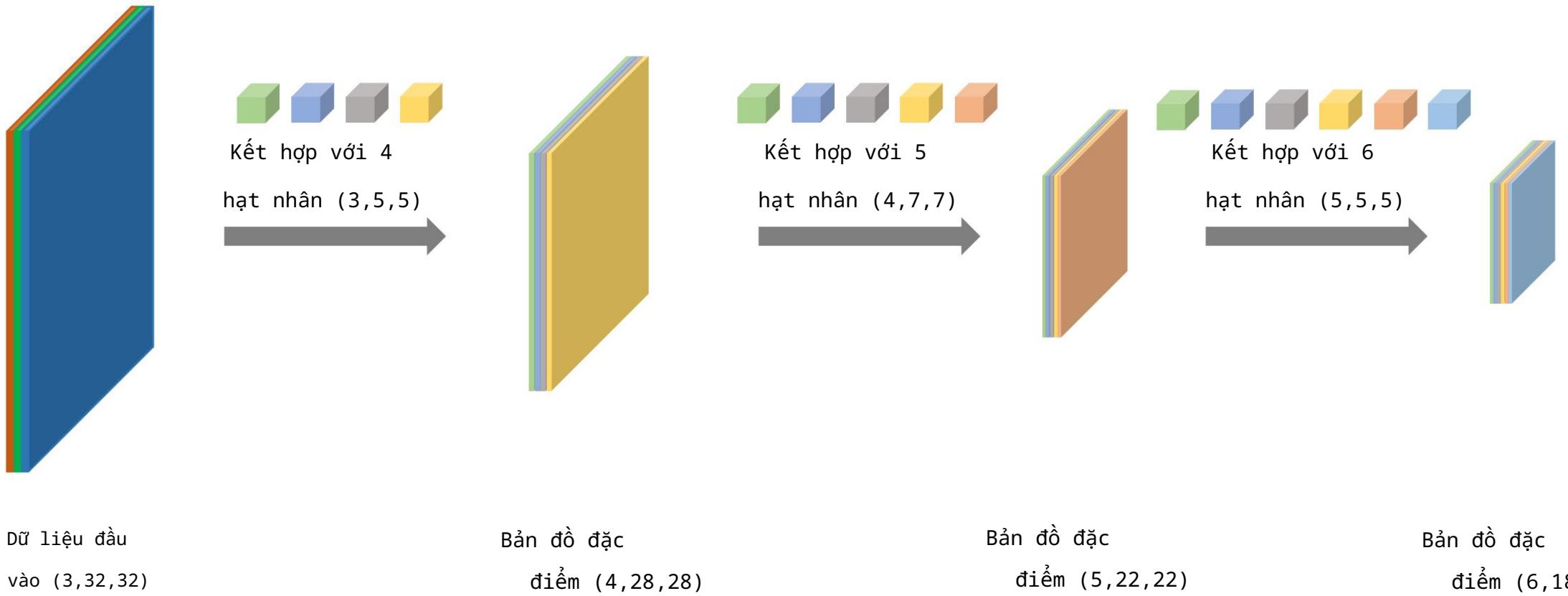


Dữ liệu đầu
vào $(3, 32, 32)$

Bản đồ đặc điểm
 $(4, 28, 28)$

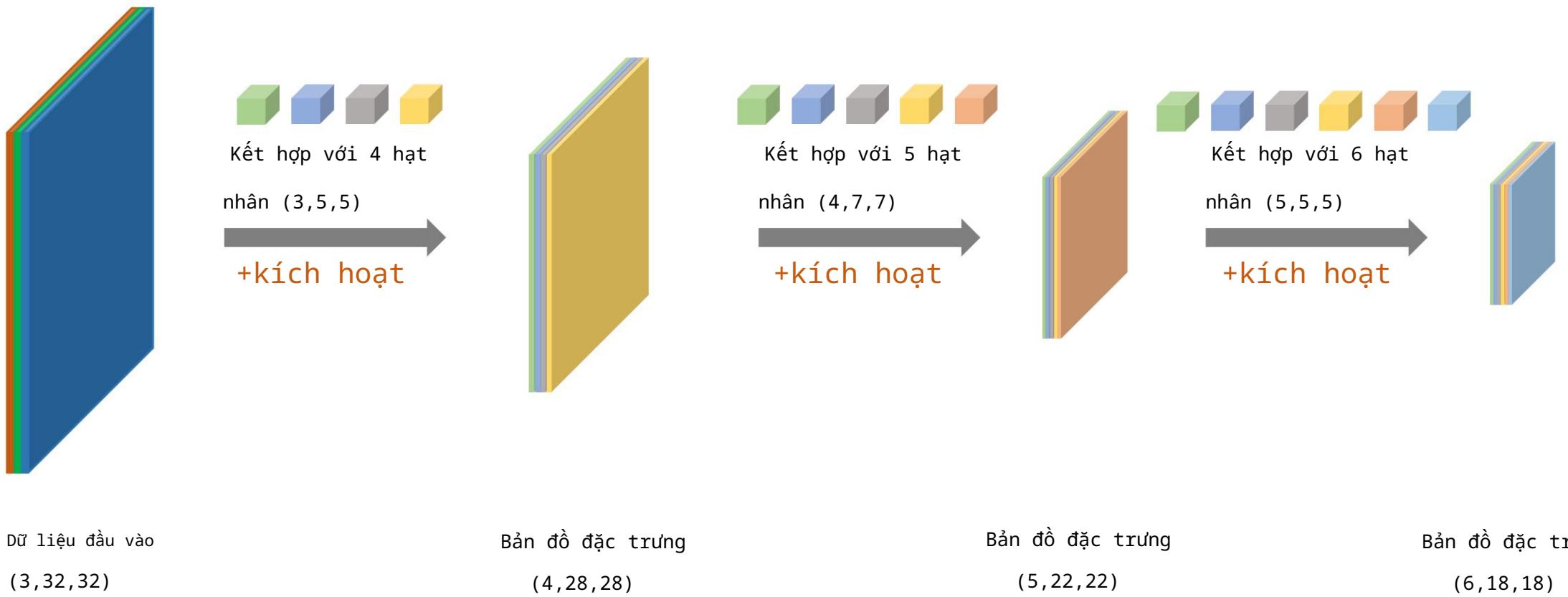
Mạng lưới thần kinh chuyển đổi

Một chồng các phép chập



Mạng lưới thần kinh chuyển đổi

Một chồng các cặp tích chập+kích hoạt



Mạng lưới thần kinh chuyển đổi

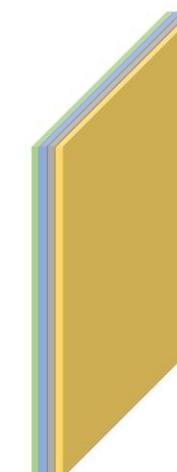
Lớp tích chập trong PyTorch

`nn.Conv2d(in_channels, out_channels, kernel_size)`



Dữ liệu đầu
vào $(1, 32, 32)$

Kết hợp với 4
hạt nhân $(1, 5, 5)$

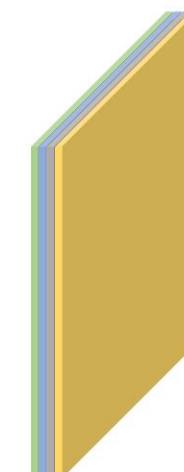


Bản đồ đặc
điểm $(4, 28, 28)$



Dữ liệu đầu
vào $(3, 32, 32)$

Kết hợp với 4
hạt nhân $(3, 5, 5)$



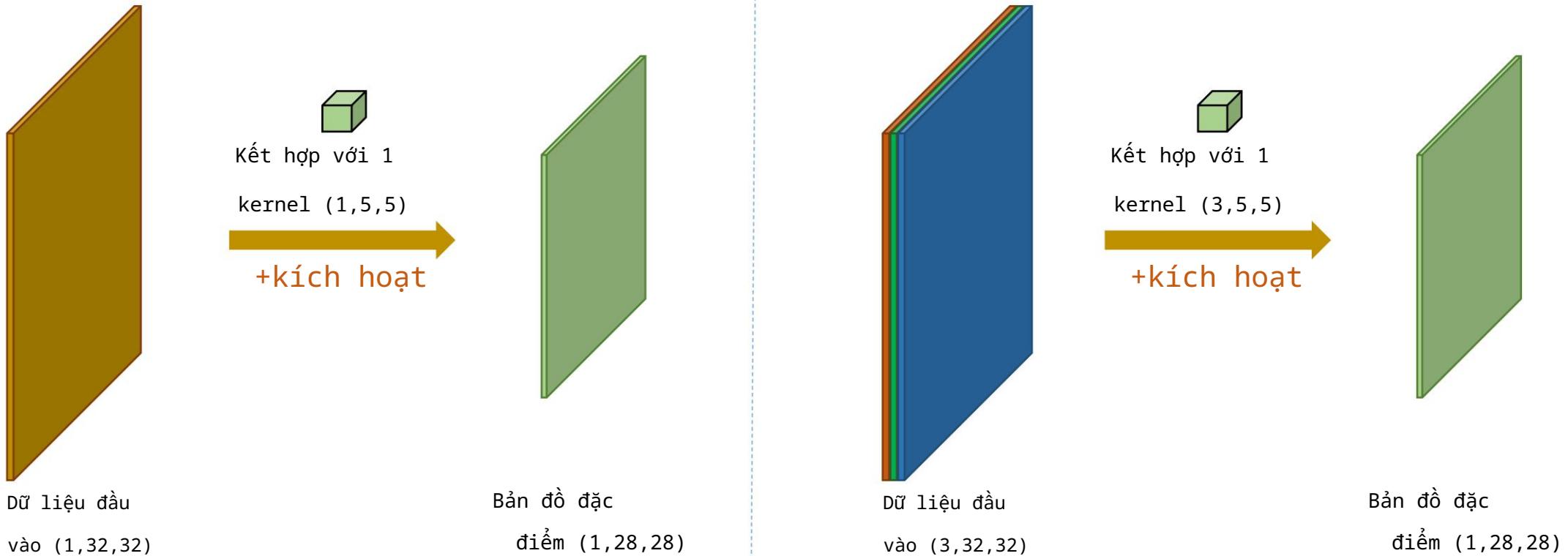
Bản đồ đặc
điểm $(4, 28, 28)$

Mạng lưới thần kinh chuyển đổi

Lớp tích chập trong PyTorch

thử nghiệm

```
nn.Conv2d(in_channels, out_channels, kernel_size)  
nn.ReLU()
```



tích chập

Mạng lưới thần kinh

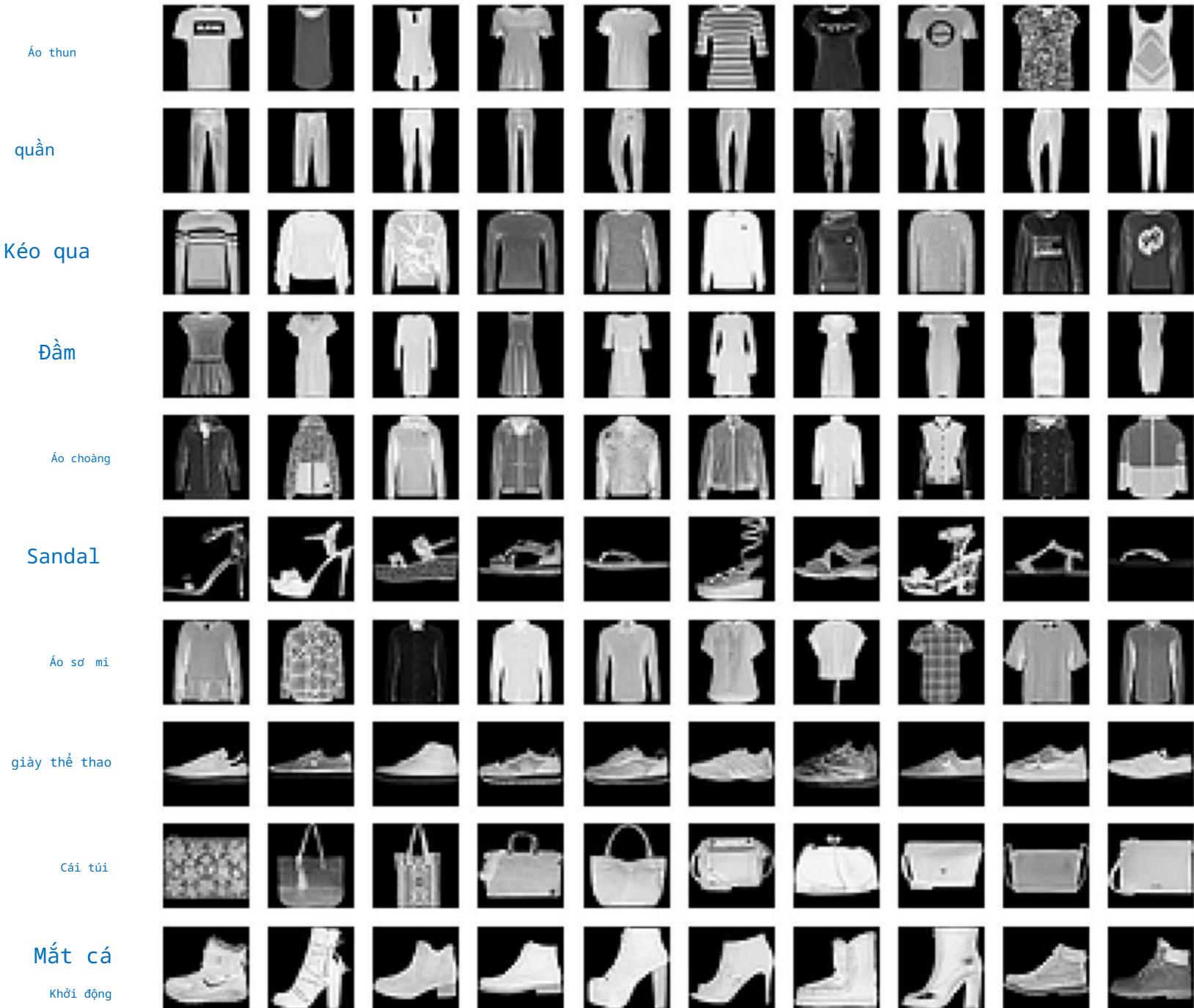
Tập dữ liệu thời trang-MNIST

Hình ảnh thang độ xám

Độ phân giải=28x28

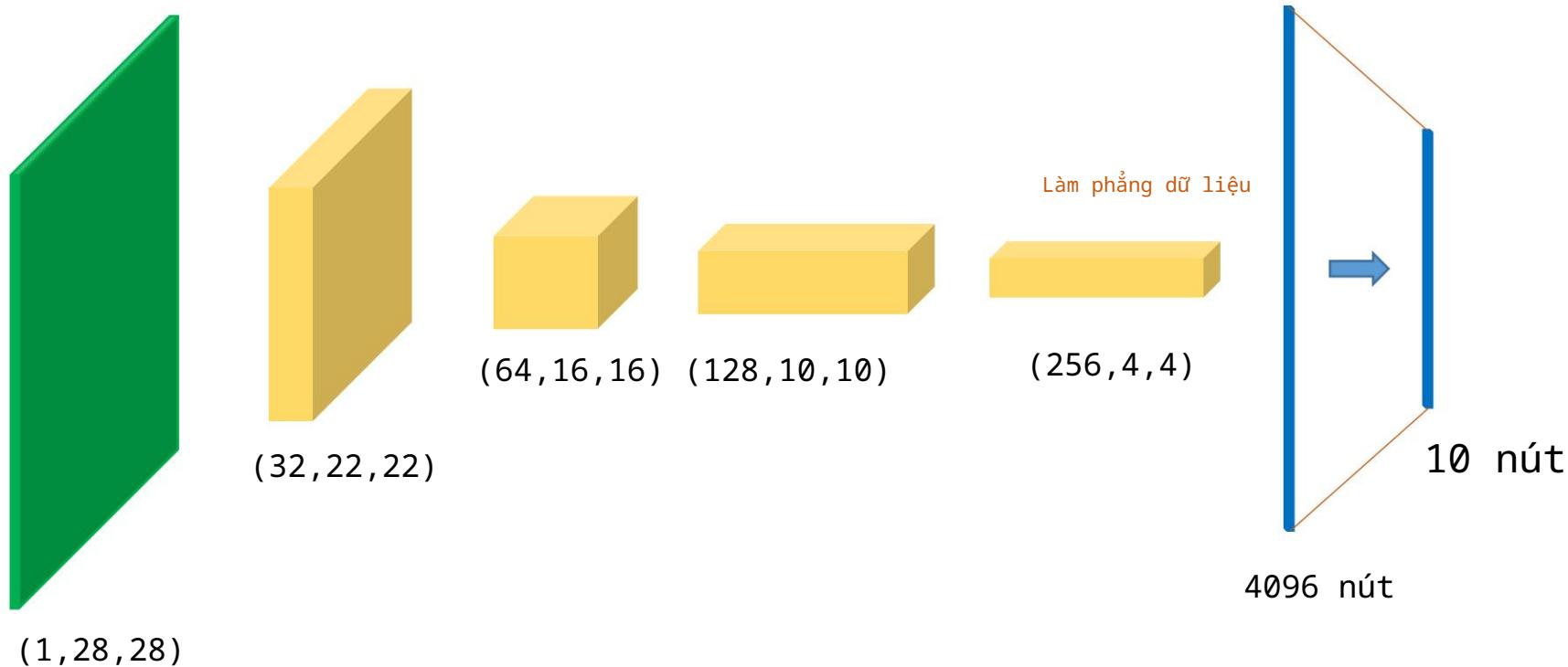
Bộ huấn luyện: 60000 mẫu

Bộ thử nghiệm: 10000 mẫu



Mạng lưới thần kinh chuyển đổi

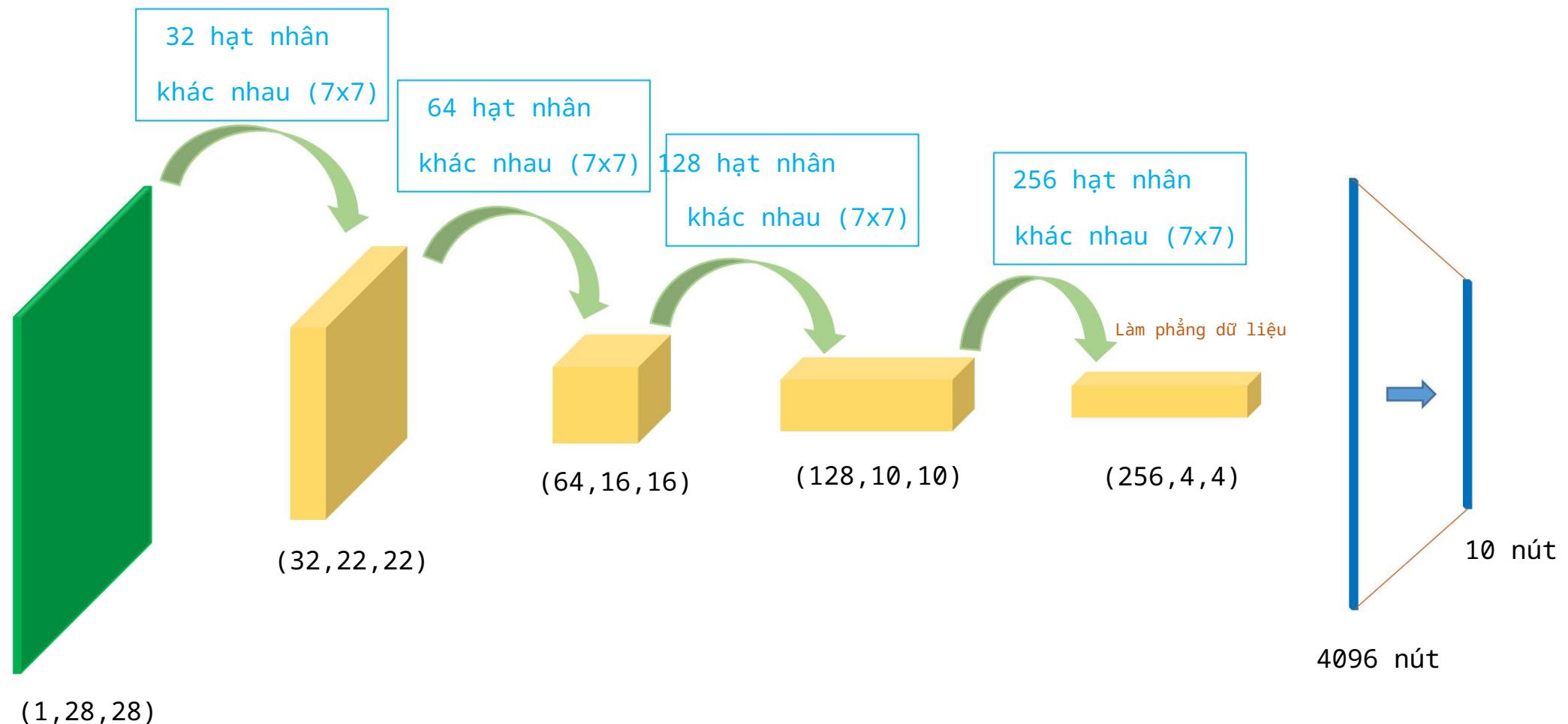
Áp dụng cho bộ dữ liệu Fashion-MNIST



Mạng lưới thần kinh chuyển đổi

Áp dụng cho bộ dữ liệu Fashion-MNIST

thử nghiệm



Mạng nơ -ron tích chập đơn giản

```

class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=7)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=7)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=7)
        self.conv4 = nn.Conv2d(128, 256, kernel_size=7)
        self.flatten = nn.Flatten()
        self.dense = nn.Linear(4*4*256, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.relu(self.conv2(x))
        x = self.relu(self.conv3(x))
        x = self.relu(self.conv4(x))
        x = self.flatten(x)
        x = self.dense(x)
        return x

model = CustomModel()

```

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 32, 22, 22]	1,600
ReLU-2	[-1, 32, 22, 22]	0
Conv2d-3	[-1, 64, 16, 16]	100,416
ReLU-4	[-1, 64, 16, 16]	0
Conv2d-5	[-1, 128, 10, 10]	401,536
ReLU-6	[-1, 128, 10, 10]	0
Conv2d-7	[-1, 256, 4, 4]	1,605,888
ReLU-8	[-1, 256, 4, 4]	0
Flatten-9	[-1, 4096]	0
Linear-10	[-1, 128]	524,416
ReLU-11	[-1, 128]	0
Linear-12	[-1, 10]	1,290

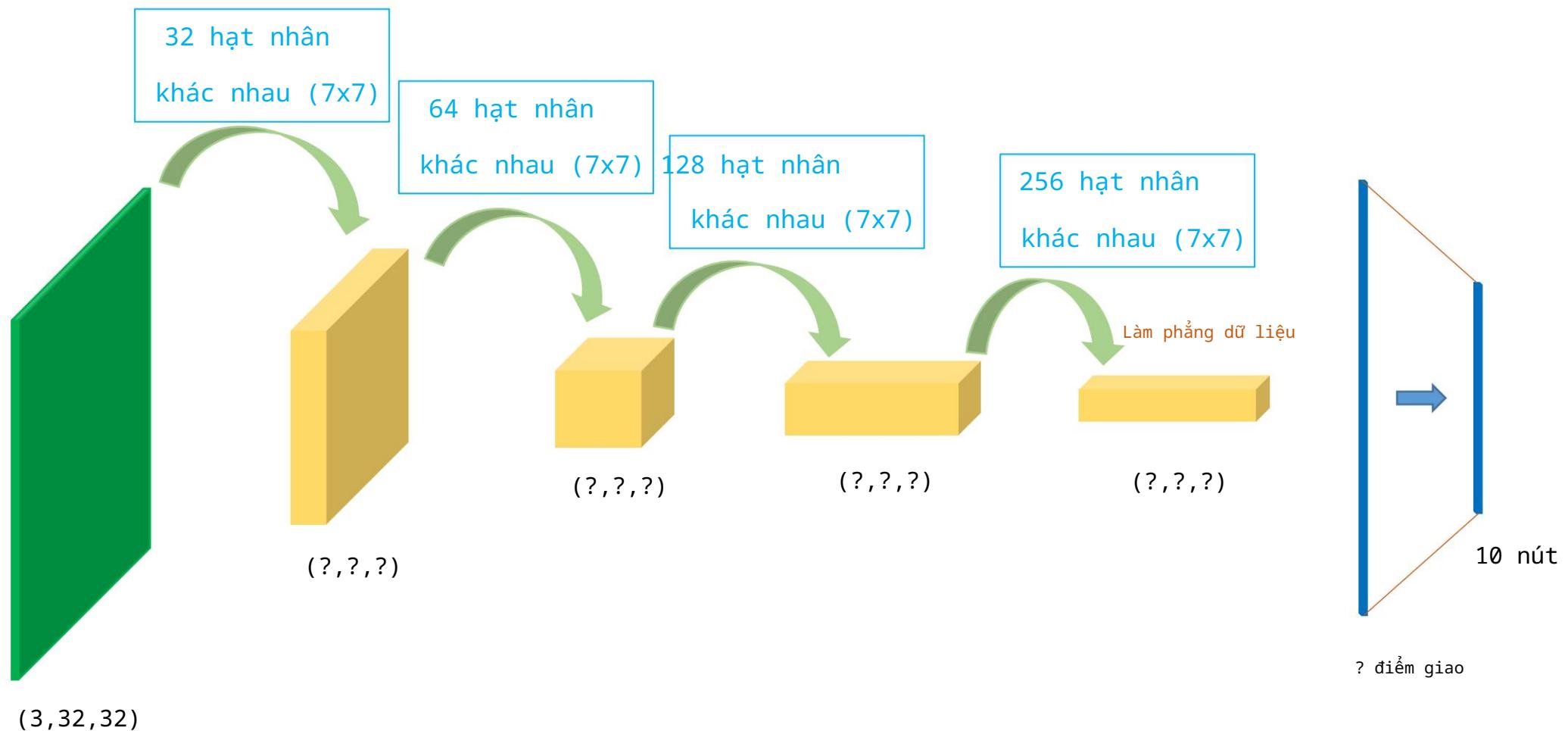
Total params: 2,635,146
Trainable params: 2,635,146
Non-trainable params: 0

Input size (MB): 0.00
Forward/backward pass size (MB): 0.78
Params size (MB): 10.05
Estimated Total Size (MB): 10.83

Mạng lưới thần kinh chuyển đổi

Áp dụng cho bộ dữ liệu Cifar-10

thử nghiệm



Đề cương

Hạn chế của MLP

Từ MLP đến CNN

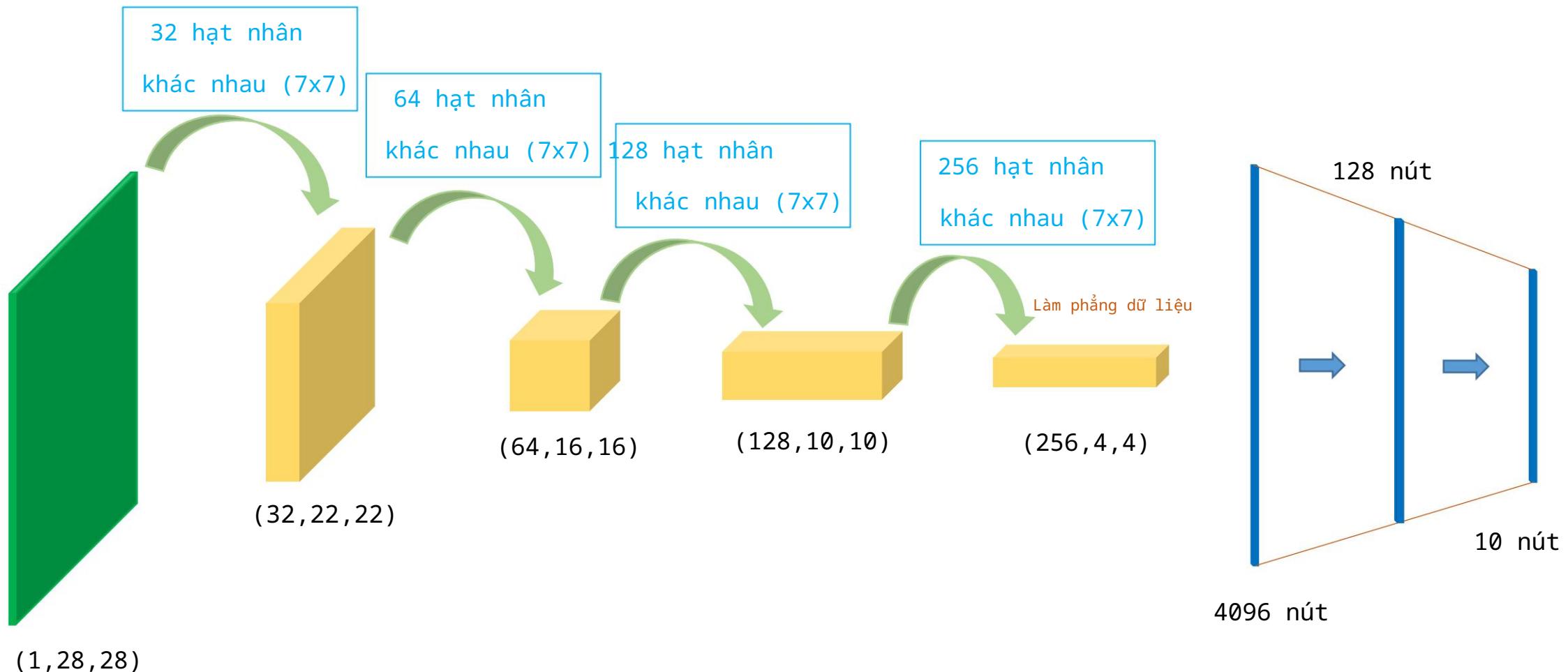
Lấy mẫu xuống bản đồ tính năng

Một số ví dụ

Ứng dụng cho Cifar10

Mạng lưới thần kinh chuyển đổi

Áp dụng cho bộ dữ liệu Fashion-MNIST: trường hợp 1



```

class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=7)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=7)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=7)
        self.conv4 = nn.Conv2d(128, 256, kernel_size=7)
        self.flatten = nn.Flatten()
        self.dense1 = nn.Linear(4*4*256, 128)
        self.dense2 = nn.Linear(128, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.relu(self.conv2(x))
        x = self.relu(self.conv3(x))
        x = self.relu(self.conv4(x))
        x = self.flatten(x)
        x = self.relu(self.dense1(x))
        x = self.dense2(x)
        return x

model = CustomModel()
model = model.to(device)

```

```

# Load FashionMNIST dataset
transform = Compose([ToTensor(),
                     Normalize((0.5,),
                               (0.5,))])

trainset = FashionMNIST(root='data',
                        train=True,
                        download=True,
                        transform=transform)
trainloader = DataLoader(trainset,
                        batch_size=1024,
                        num_workers=10,
                        shuffle=True,
                        drop_last=True)

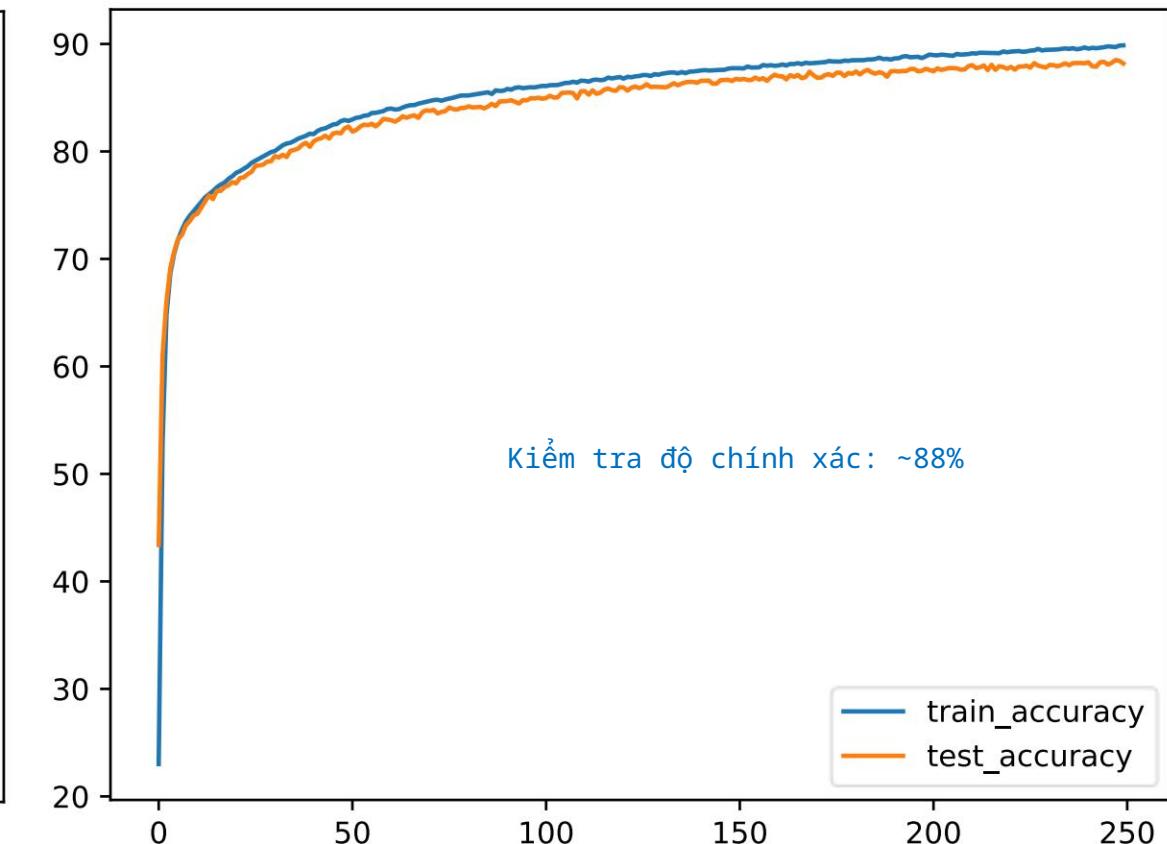
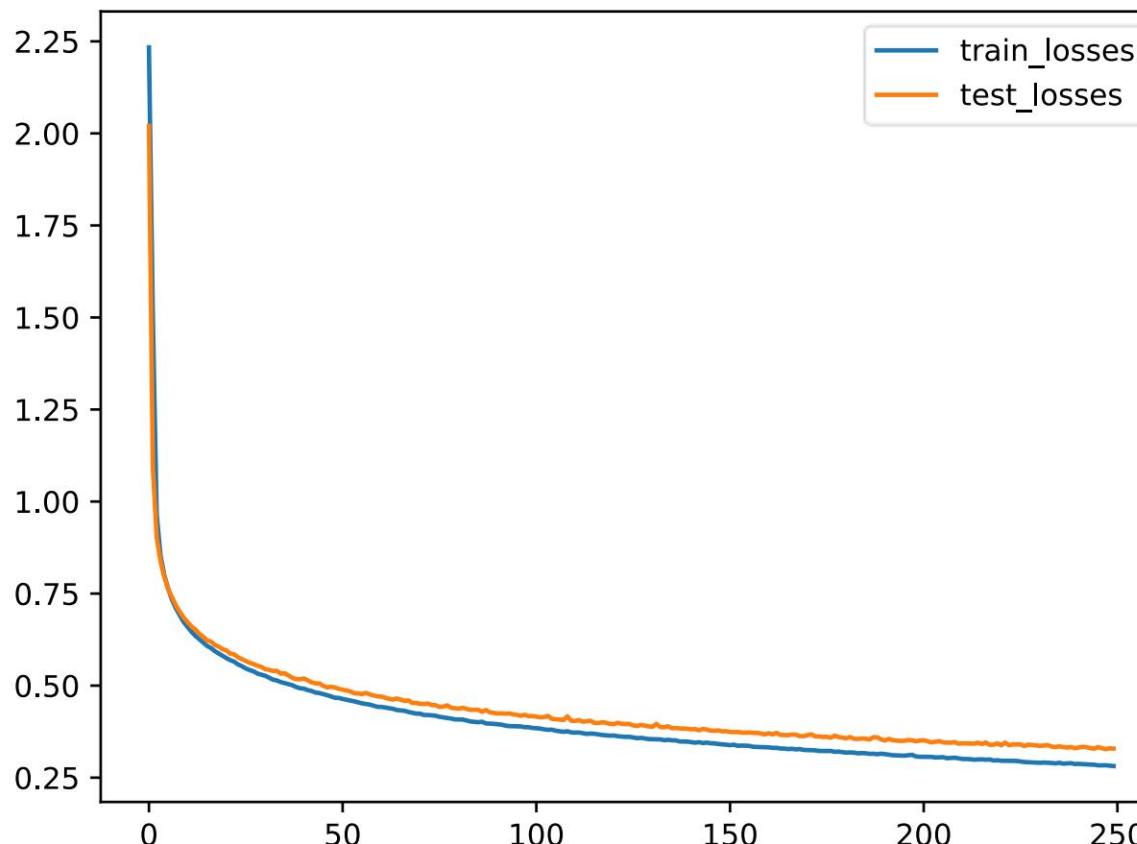
testset = FashionMNIST(root='data',
                       train=False,
                       download=True,
                       transform=transform)
testloader = DataLoader(testset,
                        batch_size=1024,
                        num_workers=10,
                        shuffle=False)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-5)

```

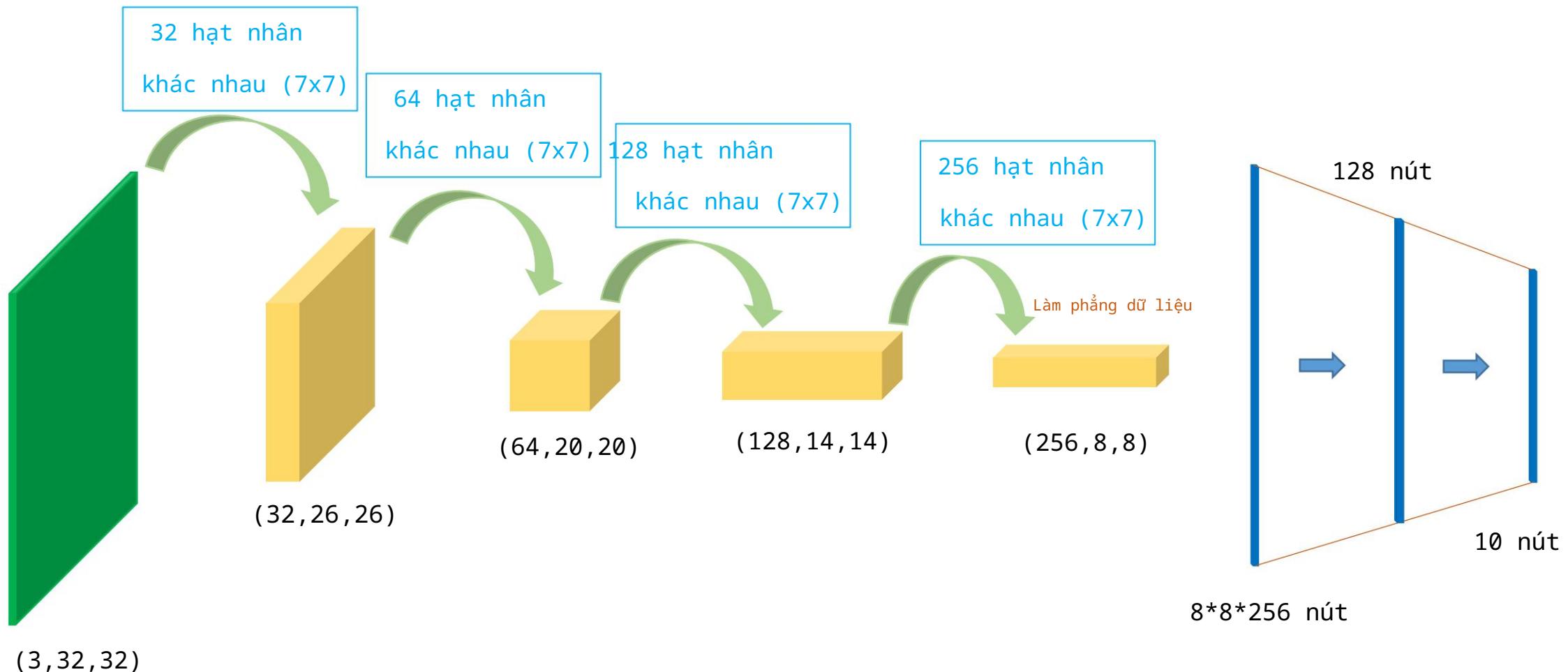
Mạng lưới thần kinh chuyển đổi

Áp dụng cho bộ dữ liệu Fashion-MNIST: trường hợp 1



Mạng lưới thần kinh chuyển đổi

Áp dụng cho tập dữ liệu Cifar-10: trường hợp 2



```

class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=7)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=7)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=7)
        self.conv4 = nn.Conv2d(128, 256, kernel_size=7)
        self.flatten = nn.Flatten()
        self.dense1 = nn.Linear(8*8*256, 128)
        self.dense2 = nn.Linear(128, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.relu(self.conv2(x))
        x = self.relu(self.conv3(x))
        x = self.relu(self.conv4(x))
        x = self.flatten(x)
        x = self.relu(self.dense1(x))
        x = self.dense2(x)
        return x

model = CustomModel()
model = model.to(device)

```

```

# Load CIFAR10 dataset
transform = Compose([ToTensor(),
                     Normalize((0.5, 0.5, 0.5),
                               (0.5, 0.5, 0.5))])

trainset = CIFAR10(root='data',
                   train=True,
                   download=True,
                   transform=transform)
trainloader = DataLoader(trainset,
                        batch_size=1024,
                        num_workers=10,
                        shuffle=True,
                        drop_last=True)

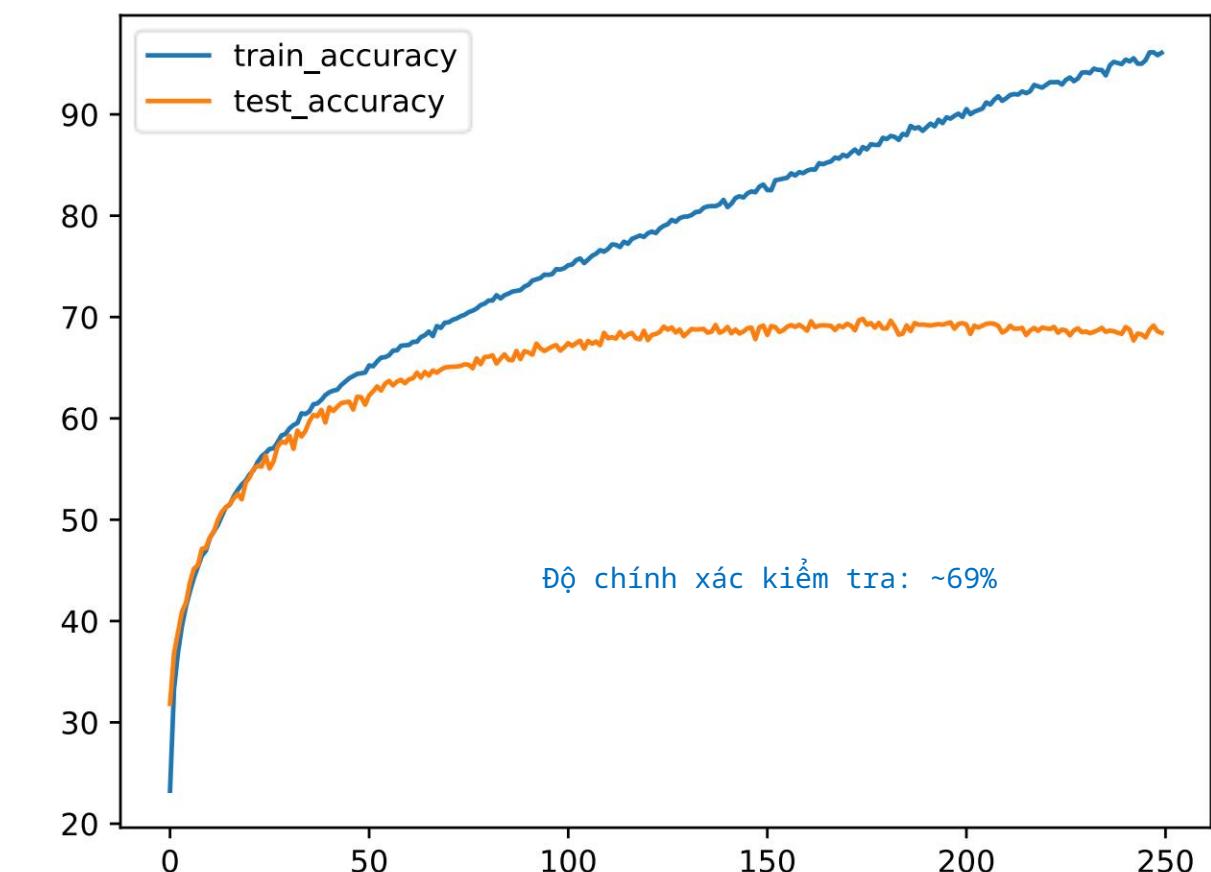
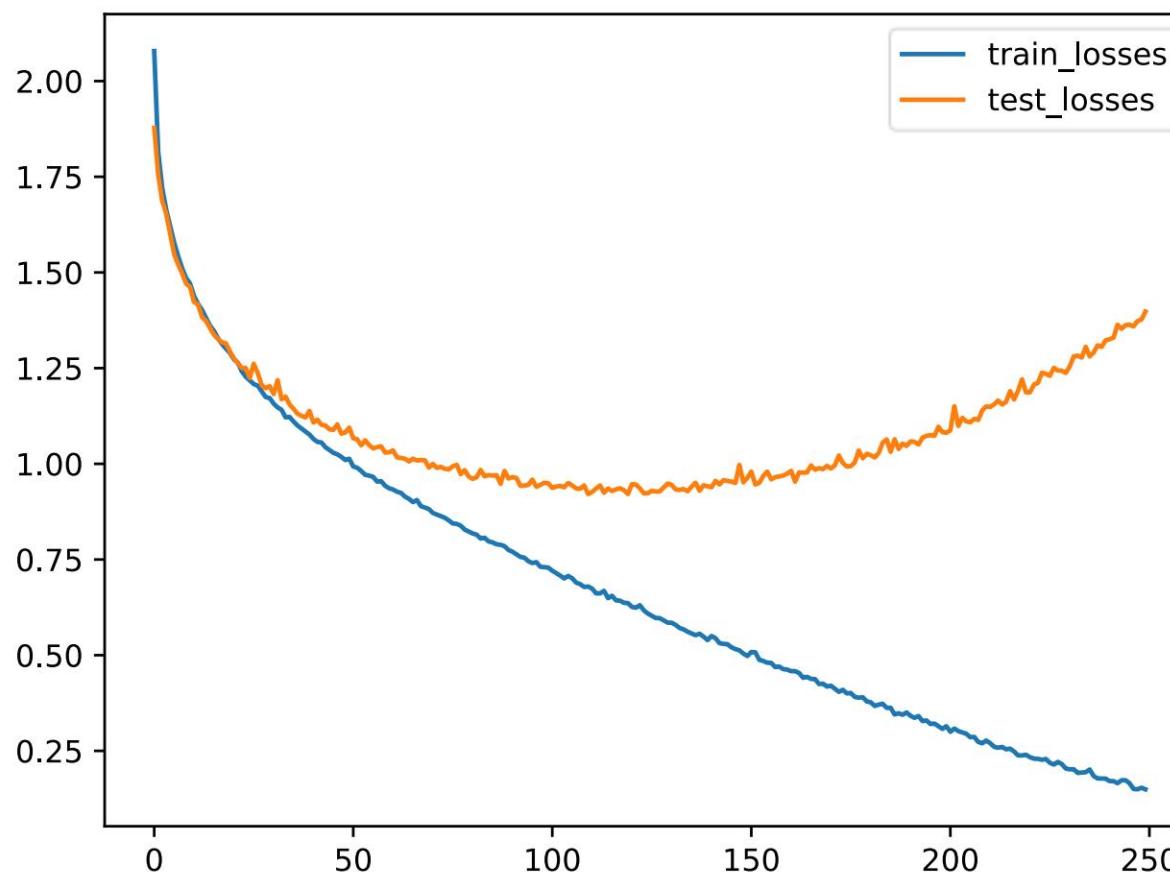
testset = CIFAR10(root='data',
                  train=False,
                  download=True,
                  transform=transform)
testloader = DataLoader(testset,
                        batch_size=1024,
                        num_workers=10,
                        shuffle=False)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-5)

```

Mạng lưới thần kinh chuyển đổi

Áp dụng cho tập dữ liệu Cifar-10: trường hợp 2



Kiểm tra độ chính xác từ MLP: ~53%

Đọc thêm

Đọc

<https://cs231n.github.io/convolutional-networks/>

<https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>

<https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks>



AI VIETNAM

Khóa học tất cả trong một

Dây thần kinh chuyển đổi Mạng

Xây dựng CNN tiêu chuẩn

Quang-Vinh Dinh

Ph.D. in Computer Science

Đề cương

Fashion-MNIST/Cifar10 Chỉ sử dụng Conv2d

Pooling

Padding

1x1 Convolution

LeNet và VGG Models

Mạng lưới thần kinh chuyển đổi

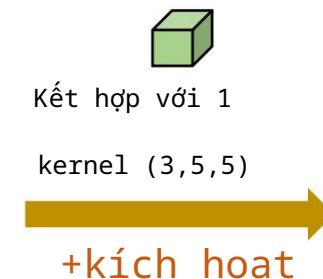
Lớp tích chập trong PyTorch

```
nn.Conv2d(in_channels, out_channels, kernel_size)  
nn.ReLU()
```



Dữ liệu đầu
vào (1, 32, 32)

Bản đồ đặc điểm
(1, 28, 28)



Dữ liệu đầu
vào (3, 32, 32)

Bản đồ đặc điểm
(1, 28, 28)

tích chập

Mạng lưới thần kinh

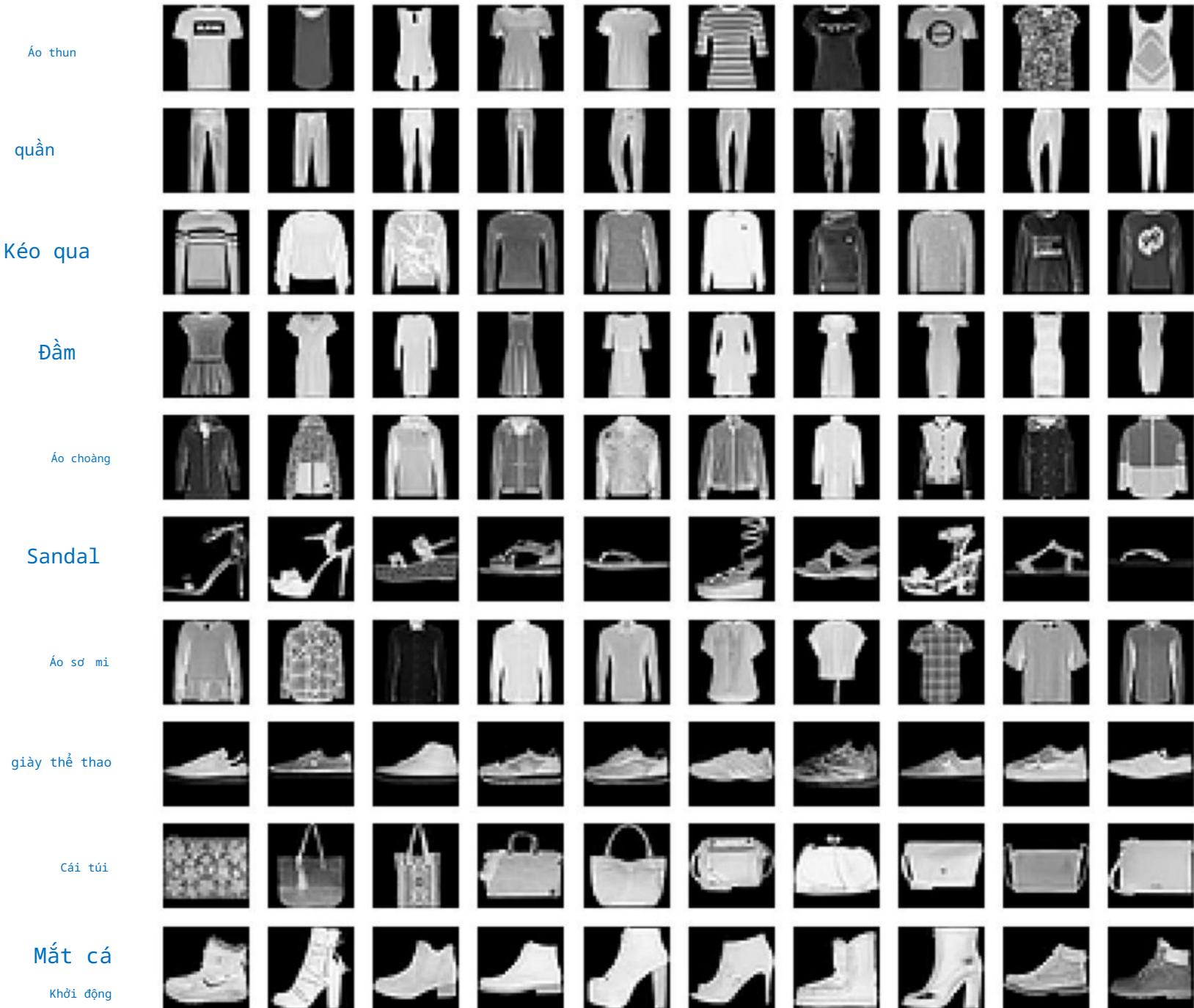
Tập dữ liệu thời trang-MNIST

Hình ảnh thang độ xám

Độ phân giải=28x28

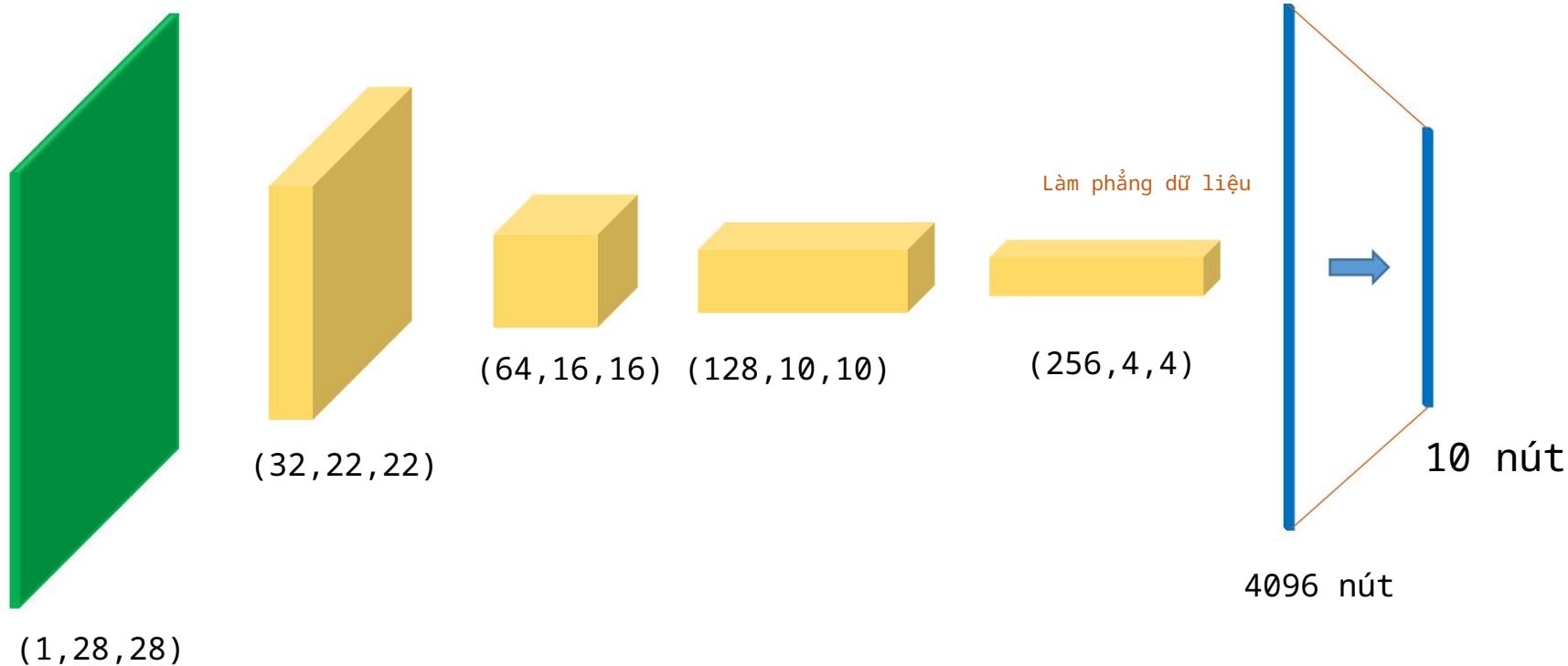
Bộ huấn luyện: 60000 mẫu

Bộ thử nghiệm: 10000 mẫu



Mạng lưới thần kinh chuyển đổi

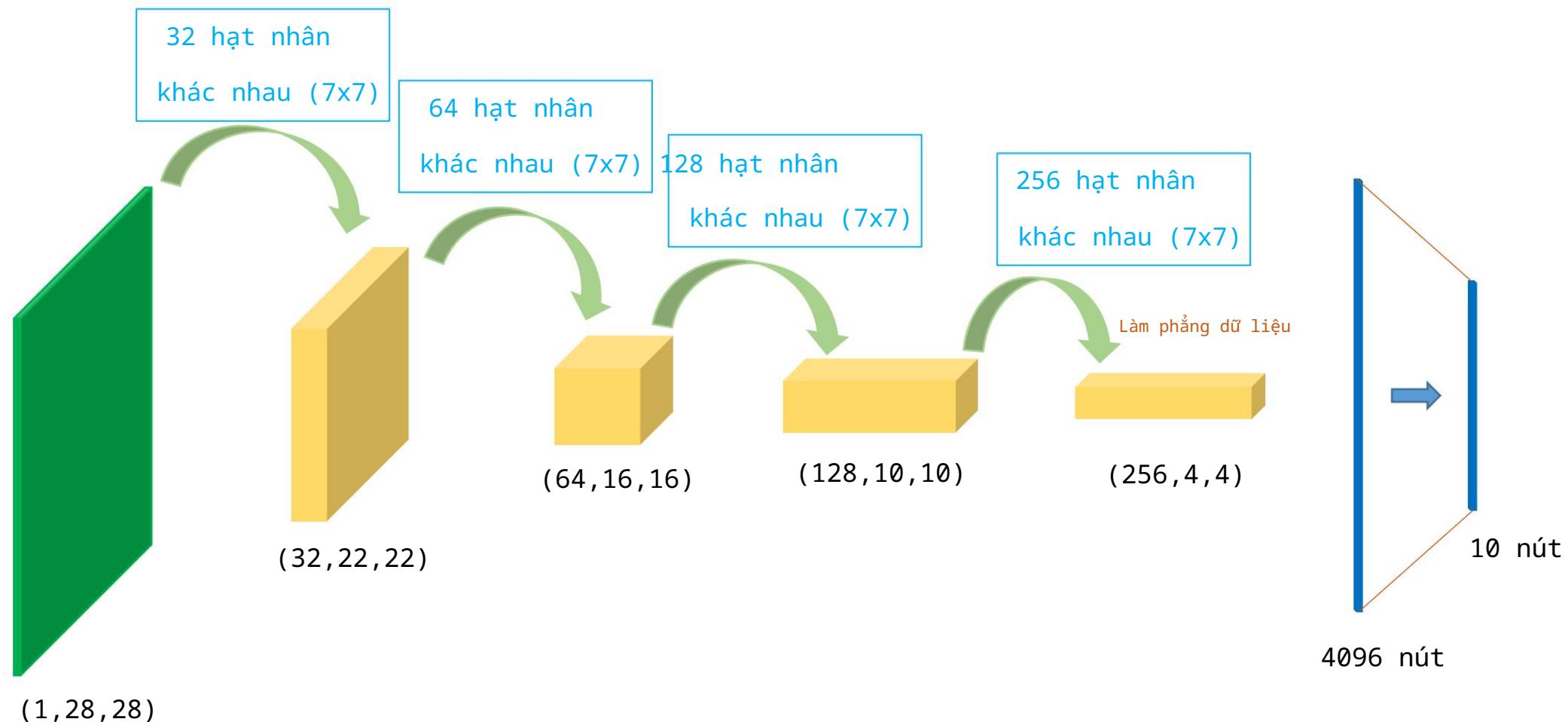
Thiết kế mô hình cho bộ dữ liệu Fashion-MNIST



Mạng lưới thần kinh chuyển đổi

Thiết kế mô hình cho tập dữ liệu Fashion-MNIST

thử nghiệm



Mạng nơ -ron tích chập đơn giản

```

class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=7)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=7)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=7)
        self.conv4 = nn.Conv2d(128, 256, kernel_size=7)
        self.flatten = nn.Flatten()
        self.dense = nn.Linear(4*4*256, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.relu(self.conv2(x))
        x = self.relu(self.conv3(x))
        x = self.relu(self.conv4(x))
        x = self.flatten(x)
        x = self.dense(x)
        return x

model = CustomModel()

```

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 32, 22, 22]	1,600
ReLU-2	[-1, 32, 22, 22]	0
Conv2d-3	[-1, 64, 16, 16]	100,416
ReLU-4	[-1, 64, 16, 16]	0
Conv2d-5	[-1, 128, 10, 10]	401,536
ReLU-6	[-1, 128, 10, 10]	0
Conv2d-7	[-1, 256, 4, 4]	1,605,888
ReLU-8	[-1, 256, 4, 4]	0
Flatten-9	[-1, 4096]	0
Linear-10	[-1, 128]	524,416
ReLU-11	[-1, 128]	0
Linear-12	[-1, 10]	1,290

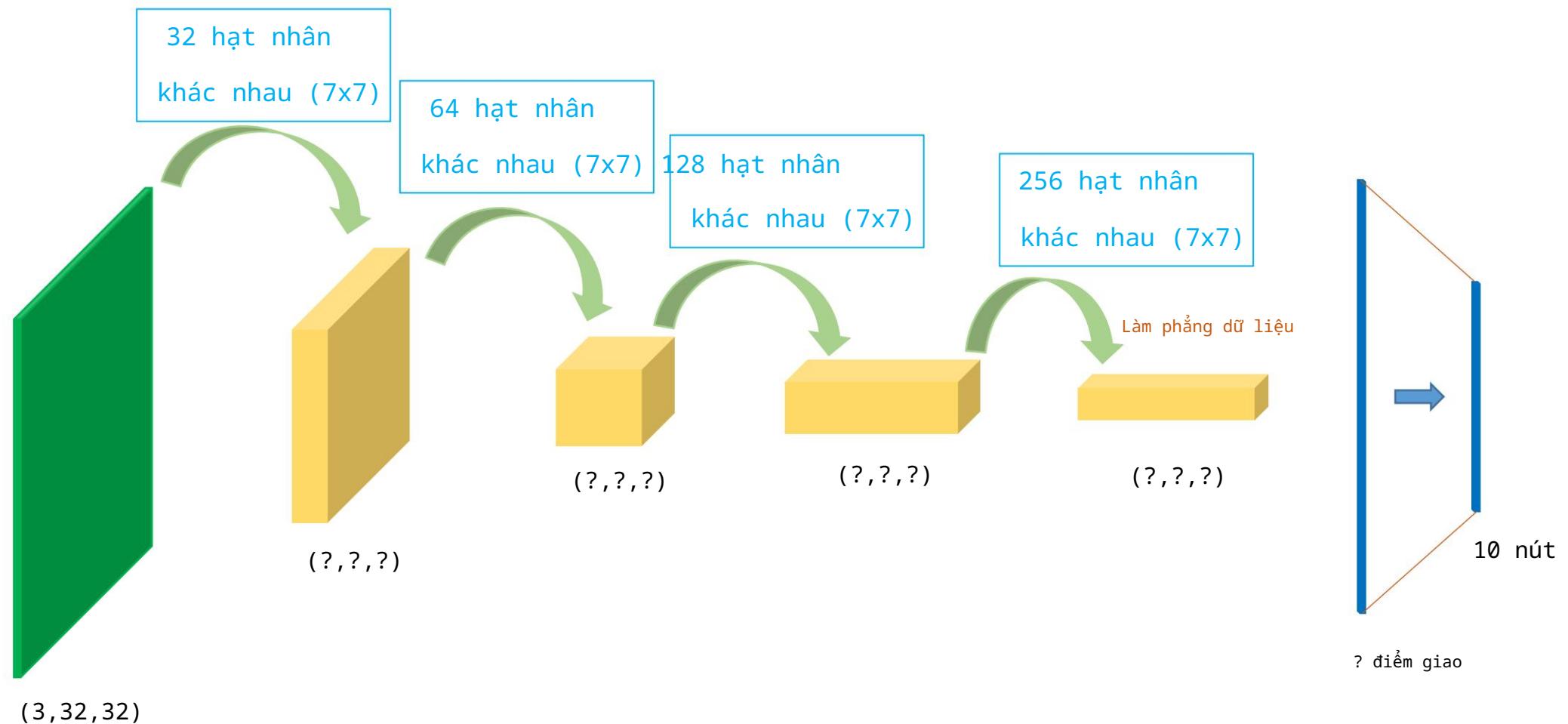
Total params: 2,635,146
Trainable params: 2,635,146
Non-trainable params: 0

Input size (MB): 0.00
Forward/backward pass size (MB): 0.78
Params size (MB): 10.05
Estimated Total Size (MB): 10.83

Mạng lưới thần kinh chuyển đổi

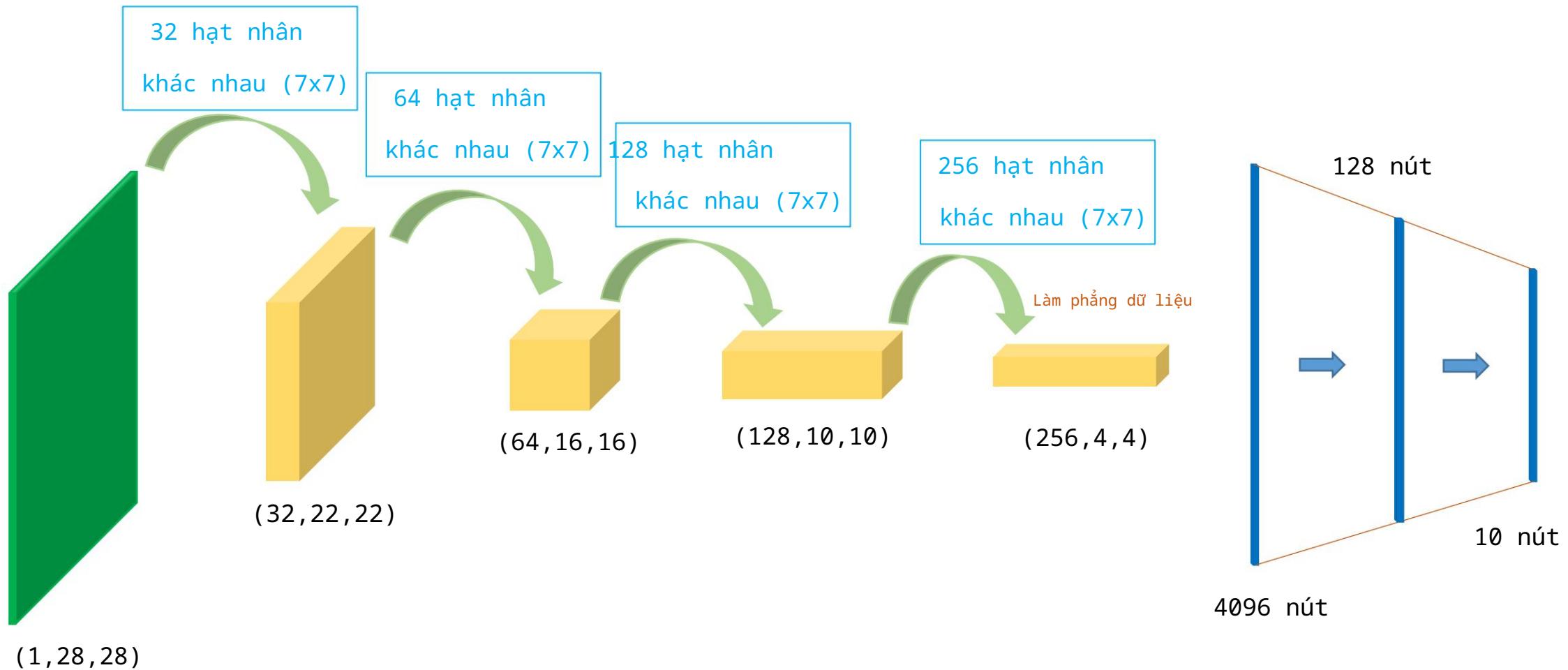
Thiết kế mô hình cho tập dữ liệu Cifar-10

thử nghiệm



Mạng lưới thần kinh chuyển đổi

Bộ dữ liệu Fashion-MNIST: trường hợp 1



```

class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=7)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=7)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=7)
        self.conv4 = nn.Conv2d(128, 256, kernel_size=7)
        self.flatten = nn.Flatten()
        self.dense1 = nn.Linear(4*4*256, 128)
        self.dense2 = nn.Linear(128, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.relu(self.conv2(x))
        x = self.relu(self.conv3(x))
        x = self.relu(self.conv4(x))
        x = self.flatten(x)
        x = self.relu(self.dense1(x))
        x = self.dense2(x)
        return x

model = CustomModel()
model = model.to(device)

```

```

# Load FashionMNIST dataset
transform = Compose([ToTensor(),
                     Normalize((0.5,),
                               (0.5,))])

trainset = FashionMNIST(root='data',
                        train=True,
                        download=True,
                        transform=transform)
trainloader = DataLoader(trainset,
                        batch_size=1024,
                        num_workers=10,
                        shuffle=True,
                        drop_last=True)

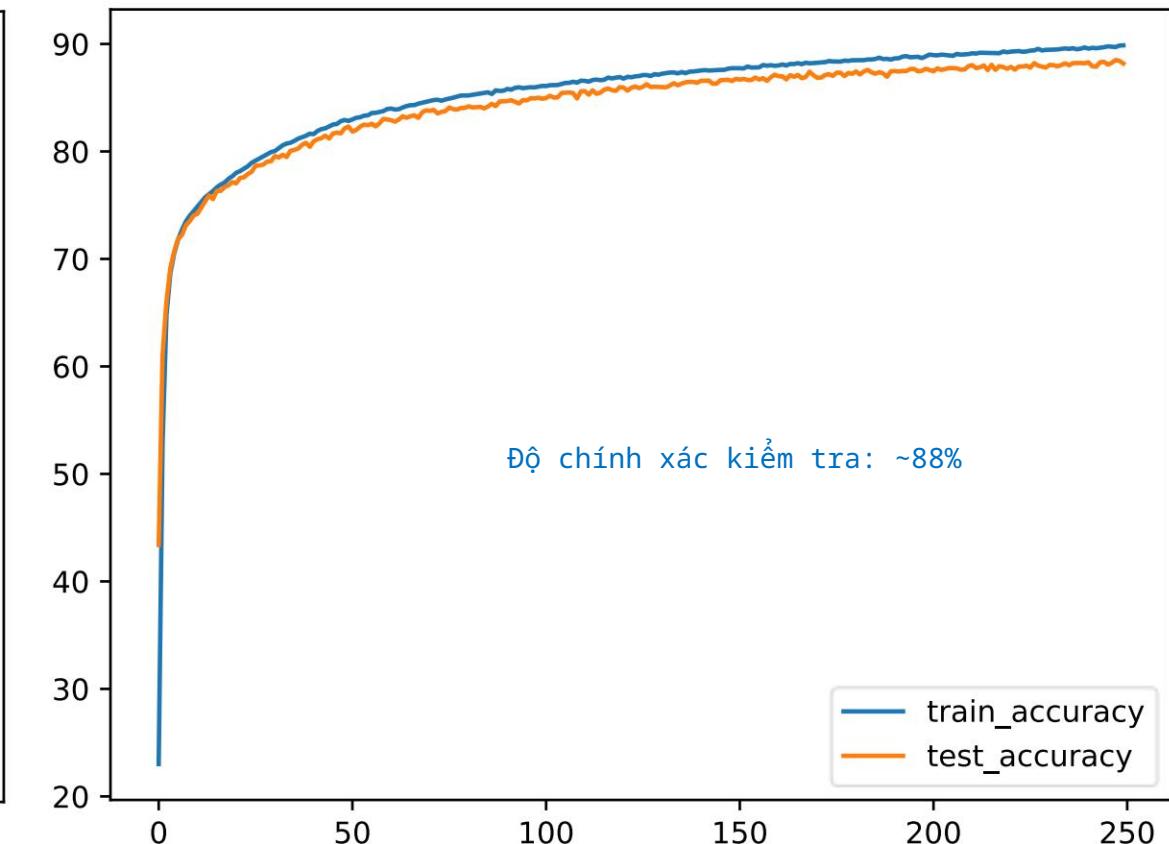
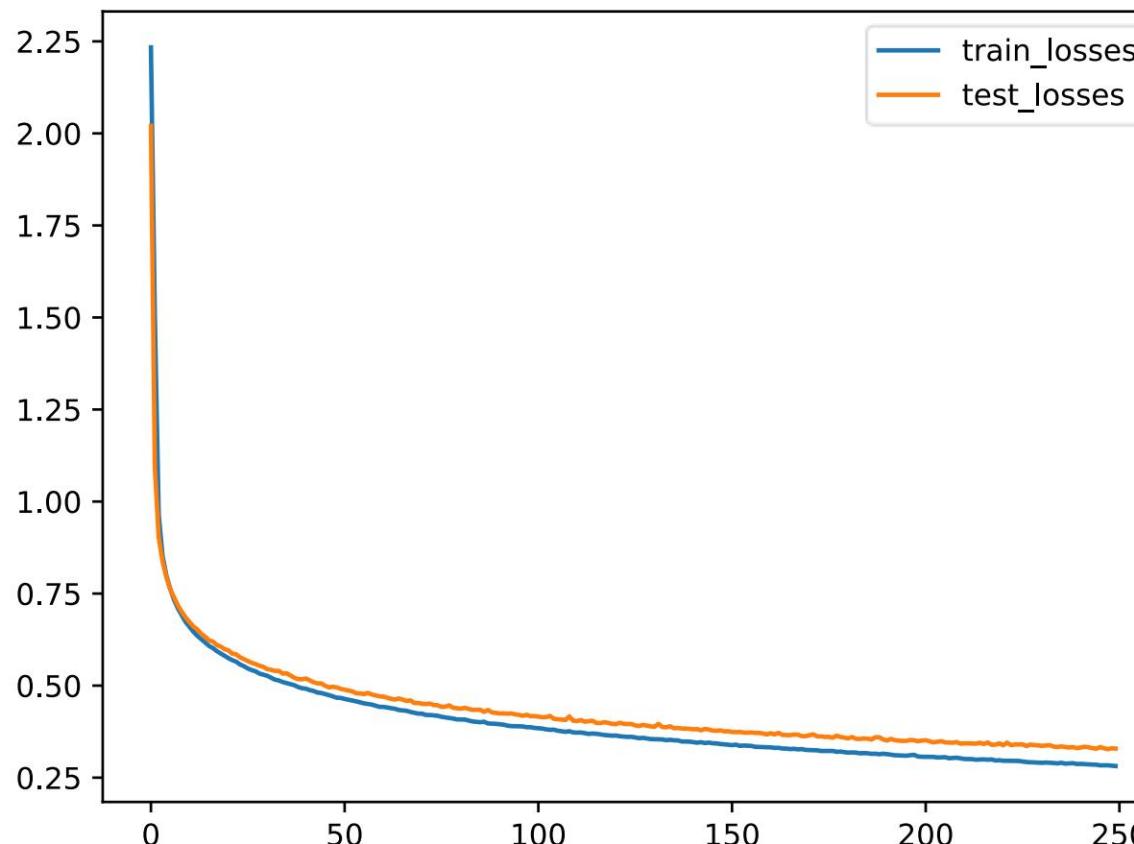
testset = FashionMNIST(root='data',
                       train=False,
                       download=True,
                       transform=transform)
testloader = DataLoader(testset,
                        batch_size=1024,
                        num_workers=10,
                        shuffle=False)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-5)

```

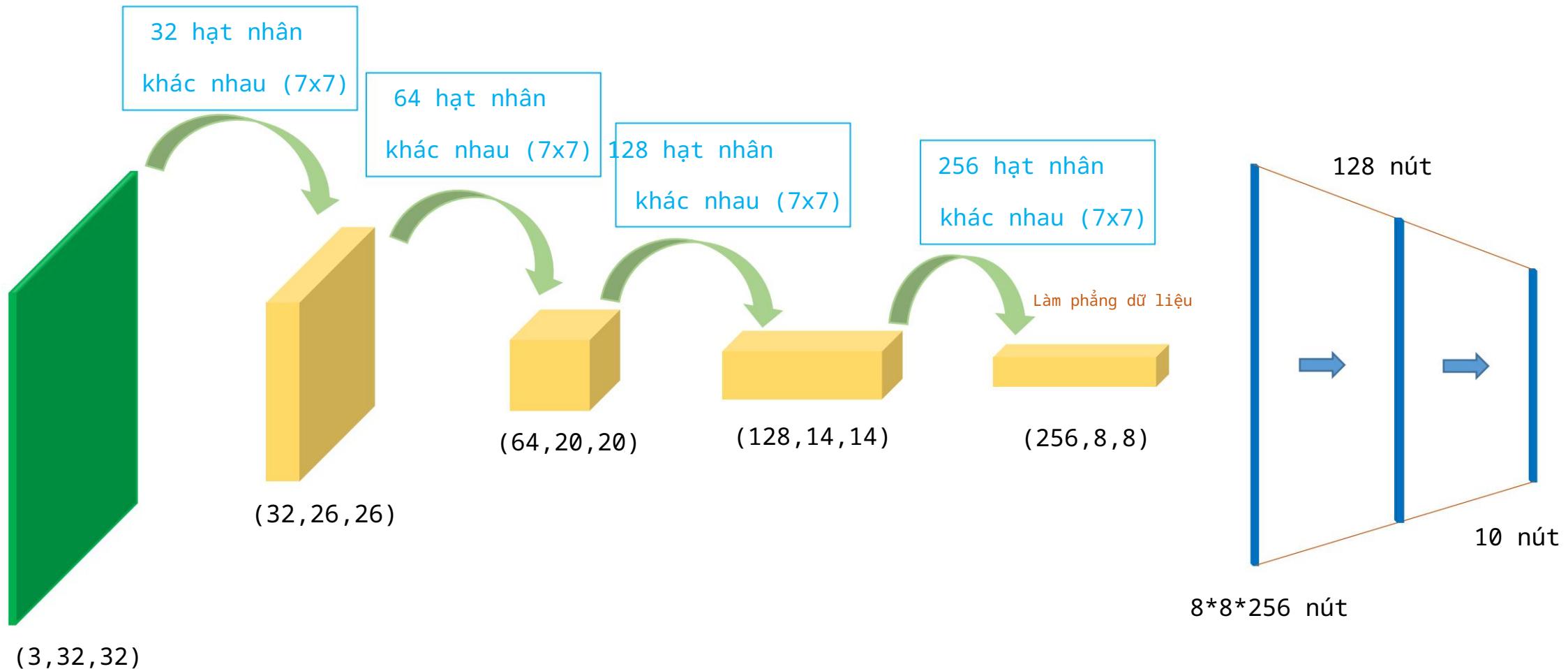
Mạng lưới thần kinh chuyển đổi

Bộ dữ liệu Fashion-MNIST: trường hợp 1



Mạng lưới thần kinh chuyển đổi

Bộ dữ liệu Cifar-10: trường hợp 2



```

class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=7)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=7)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=7)
        self.conv4 = nn.Conv2d(128, 256, kernel_size=7)
        self.flatten = nn.Flatten()
        self.dense1 = nn.Linear(8*8*256, 128)
        self.dense2 = nn.Linear(128, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.relu(self.conv2(x))
        x = self.relu(self.conv3(x))
        x = self.relu(self.conv4(x))
        x = self.flatten(x)
        x = self.relu(self.dense1(x))
        x = self.dense2(x)
        return x

model = CustomModel()
model = model.to(device)

```

```

# Load CIFAR10 dataset
transform = Compose([ToTensor(),
                     Normalize((0.5, 0.5, 0.5),
                               (0.5, 0.5, 0.5))])

trainset = CIFAR10(root='data',
                   train=True,
                   download=True,
                   transform=transform)
trainloader = DataLoader(trainset,
                        batch_size=1024,
                        num_workers=10,
                        shuffle=True,
                        drop_last=True)

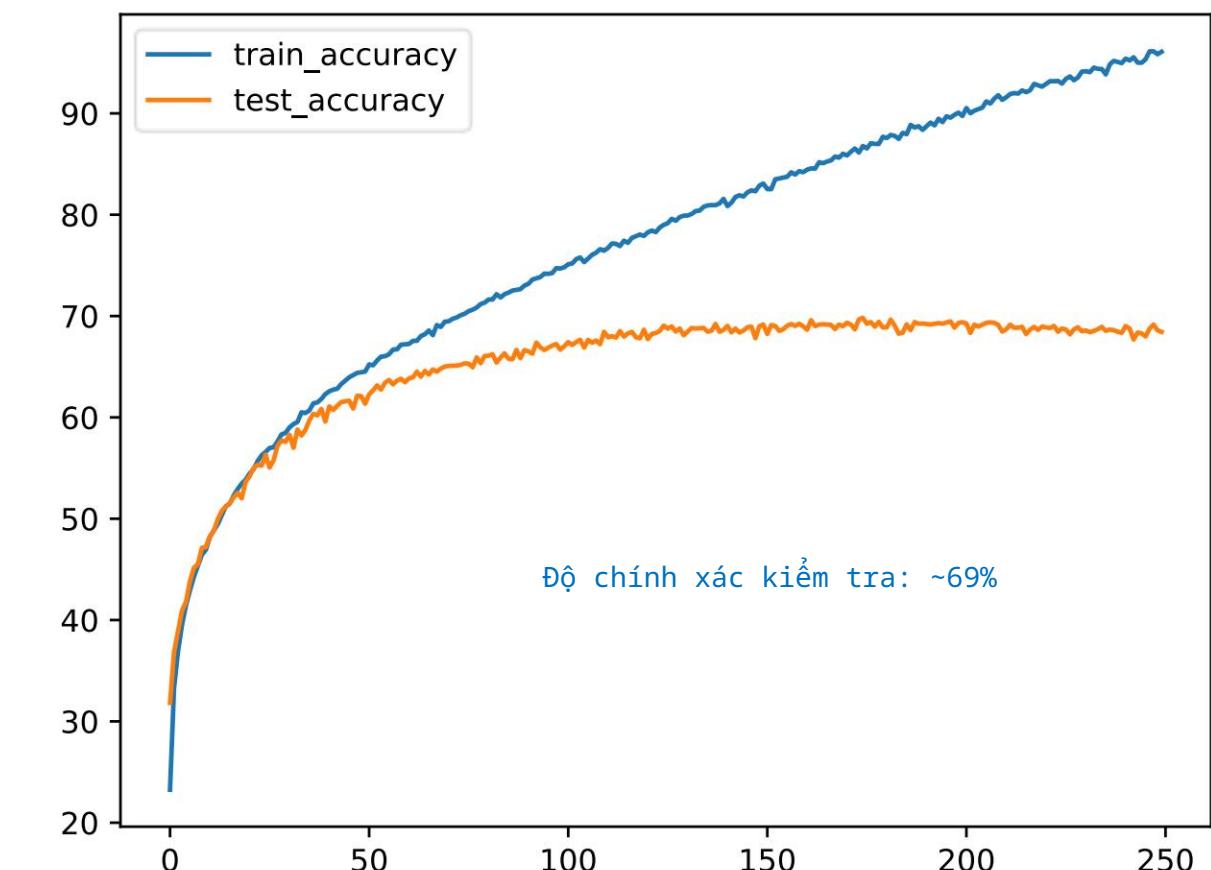
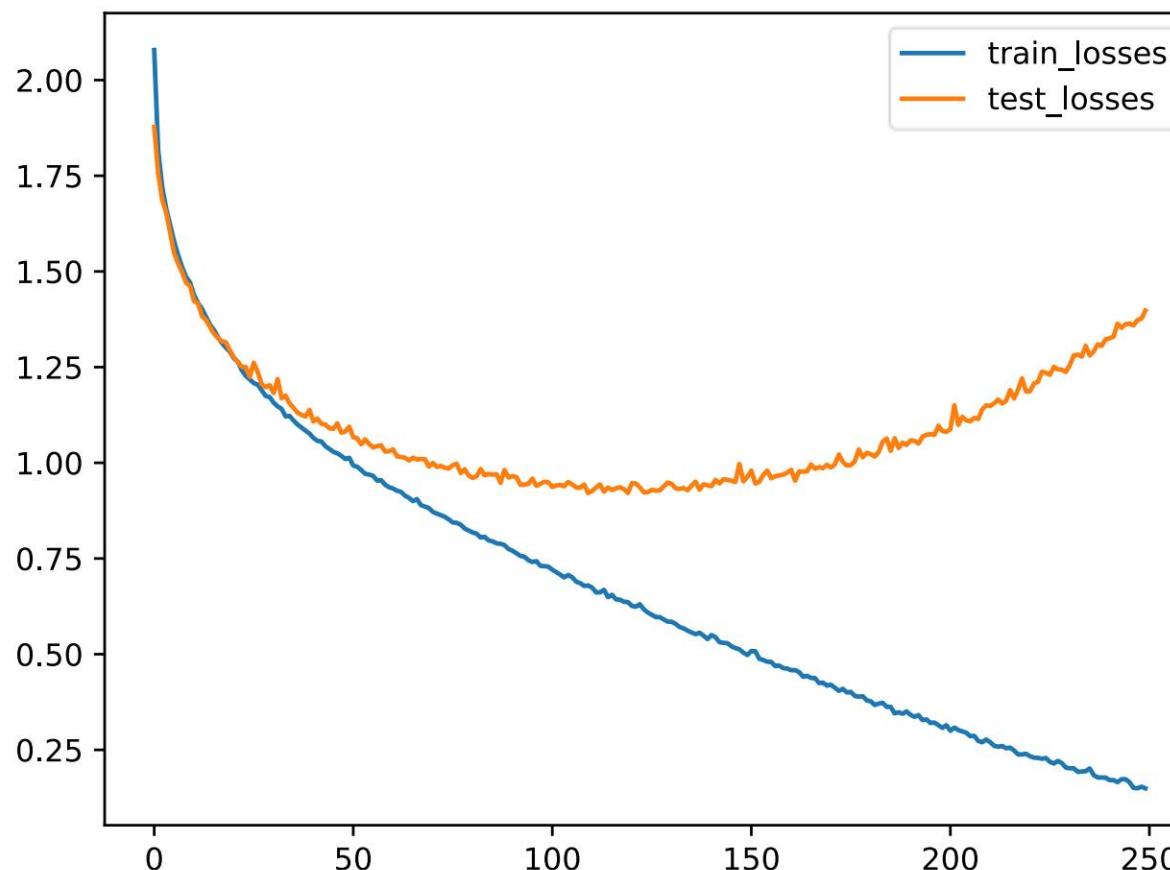
testset = CIFAR10(root='data',
                  train=False,
                  download=True,
                  transform=transform)
testloader = DataLoader(testset,
                        batch_size=1024,
                        num_workers=10,
                        shuffle=False)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-5)

```

Mạng lưới thần kinh chuyển đổi

Bộ dữ liệu Cifar-10: trường hợp 2



Kiểm tra độ chính xác từ MLP: ~53%

Đề cương

Fashion-MNIST/Cifar10 Chỉ sử dụng Conv2d

Pooling

Padding

1x1 Convolution

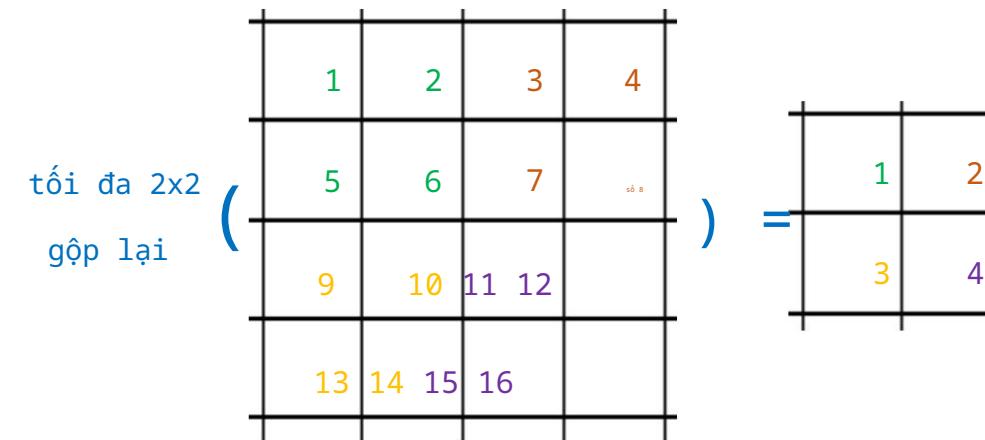
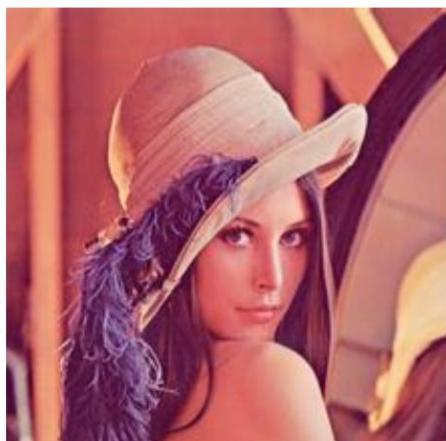
LeNet và VGG Models

Bản đồ tính năng mẫu xuống

Tổng hợp tối đa: Các tính năng được giữ nguyên

1	2	3	4
5	6	7	số 8
9	10	11	12
13	14	15	16

Dữ liệu



1 = tối đa(1, 2 , 5 , 6)
2 = tối đa(3, 4 , 7 , 8)
3 = tối đa (, 10 , 13 , 14)
4 = tối đa(11, 12 , 15 , 16)

`nn.MaxPool2d(2, 2)`



Bản đồ tính năng (220x220)

tổng hợp tối đa
(2x2)

Bản đồ đặc trưng
(110x110)

tổng hợp tối đa
(2x2)

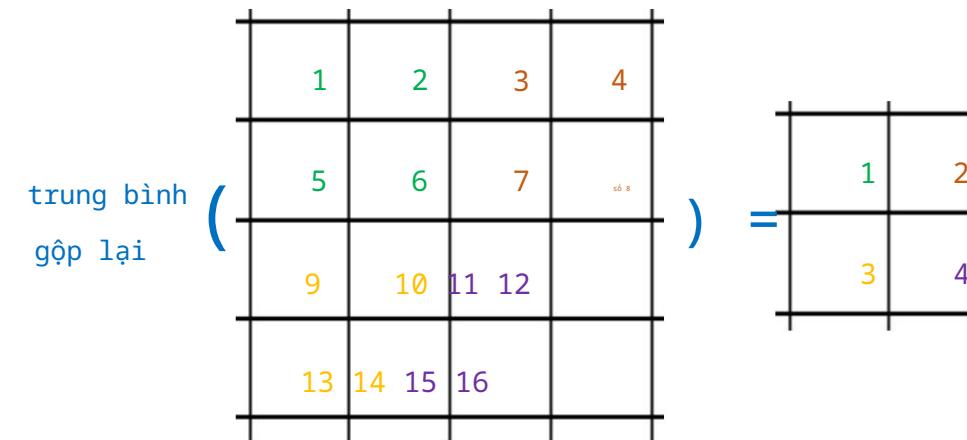
Bản đồ đặc trưng
(55x55)

Bản đồ tính năng mẫu xuống

Tổng hợp trung bình: Các tính năng được giữ nguyên

1	2	3	4
5	6	7	số 8
9	10	11	12
13	14	15	16

Dữ liệu



1 = trung bình (1, 2, 5, 6)
2 = trung bình (3, 4, 7, 8)
3 = trung bình (9, 10, 13, 14)
4 = trung bình (11, 12, 15, 16)

`nn.AvgPool2d(2, 2)`



Trung bình
Tổng hợp (2x2)



Bản đồ đặc trưng
(110x110)

Trung bình
Tổng hợp (2x2)



Bản đồ đặc trưng
(55x55)

Bản đồ tính năng mẫu xuống

Tổng hợp tối đa so với tổng hợp trung bình

`nn.MaxPool2d(2, 2)`



Bản đồ tính năng (220x220)

tổng hợp tối
đa (2x2)



tổng hợp tối
đa (2x2)



Bản đồ đặc
điểm (55x55)

`nn.AvgPool2d(2, 2)`



Bản đồ tính
năng (220x220)

Trung bình
Tổng hợp (2x2)



Trung bình
Tổng hợp (2x2)



Bản đồ tính
năng (110x110)

Bản đồ đặc
điểm (55x55)

```

def forward(self, x):
    x = self.relu(self.conv1(x))
    x = self.relu(self.conv2(x))
    x = self.pool(x)

    x = self.relu(self.conv3(x))
    x = self.relu(self.conv4(x))
    x = self.pool(x)

    x = self.relu(self.conv5(x))
    x = self.flatten(x)
    x = self.dense(x)

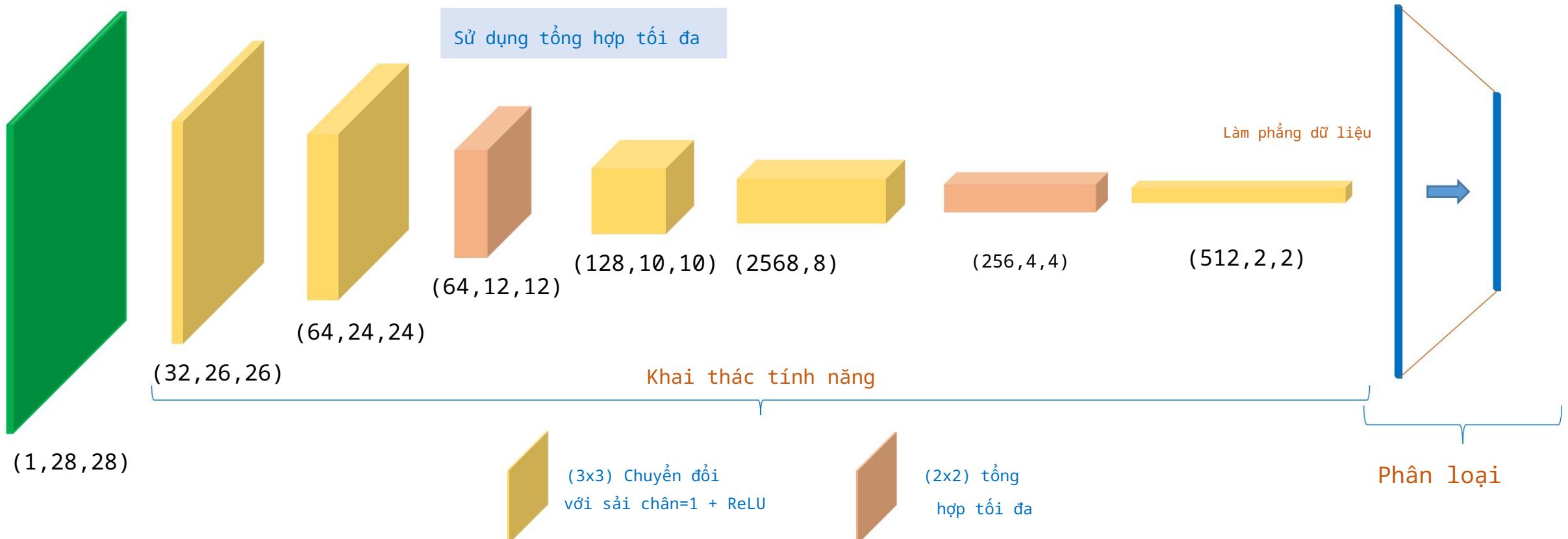
    return x

```

```

class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3)
        self.conv4 = nn.Conv2d(128, 256, kernel_size=3)
        self.conv5 = nn.Conv2d(256, 512, kernel_size=3)
        self.relu = nn.ReLU()
        self.pool = nn.MaxPool2d(2, 2) # 2x2 Max pooling
        self.flatten = nn.Flatten()
        self.dense = nn.Linear(2*2*512, 10)

```



Mẫu xuống

Bản đồ đặc điểm

Kết hợp với bước tiến

1	2	3	4
5	6	7	số 8
9	10	11	12
13	14	15	16

Dữ liệu D

1	2
3	4

Hạt nhân của tham số

Kết hợp D với
sải chân=1

1	2	3
4	5	6
7	số 8	9

đầu ra

1	2	3	4
5	6	7	số 8
9	10	11	12
13	14	15	16

1	2	3	4
5	6	7	số 8
9	10	11	12
13	14	15	16

1	2	3	4
5	6	7	số 8
9	10	11	12
13	14	15	16

Mẫu xuống

Bản đồ đặc điểm

Kết hợp với bước tiến

1	2	3	4
5	6	7	số 8
9	10	11	12
13	14	15	16

Dữ liệu D

1	2
3	4

Hạt nhân của tham số

Kết hợp D
với sải chân=2

1	2
3	4

đầu ra

1	2	3	4
5	6	7	
9	10	11	12
13	14	15	16

1	2	3	4
5	6	7	số 8
9	10	11	12
13	14	15	16

1	2	3	4
5	6	7	
9	10	11	12
13	14	15	16

1	2	3	4
5	6	7	số 8
9	10	11	12
13	14	15	16

`nn.Conv2d(in_channels, out_channels, kernel_size, sải chân=1)`

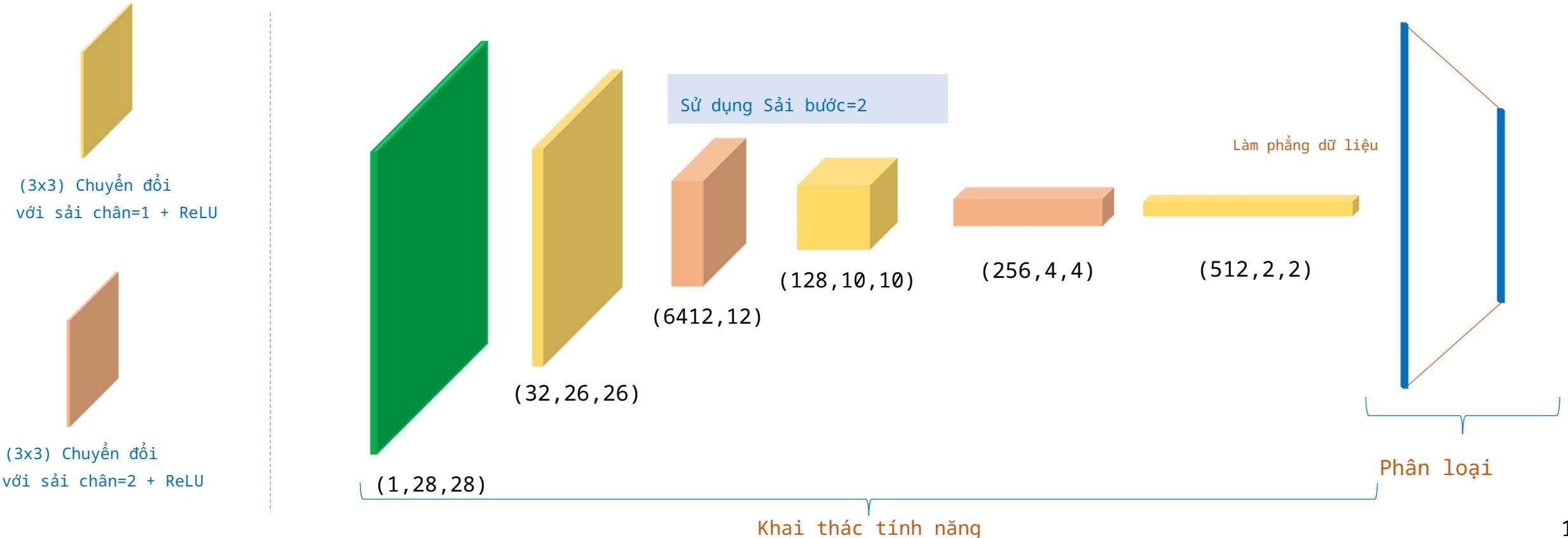
`nn.Conv2d(in_channels, out_channels, kernel_size, sải chân=2)`

```
class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=2)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3)
        self.conv4 = nn.Conv2d(128, 256, kernel_size=3, stride=2)
        self.conv5 = nn.Conv2d(256, 512, kernel_size=3)
        self.relu = nn.ReLU()
        self.flatten = nn.Flatten()
        self.dense = nn.Linear(2*2*512, 10)
```

```
def forward(self, x):
    x = self.relu(self.conv1(x))
    x = self.relu(self.conv2(x))

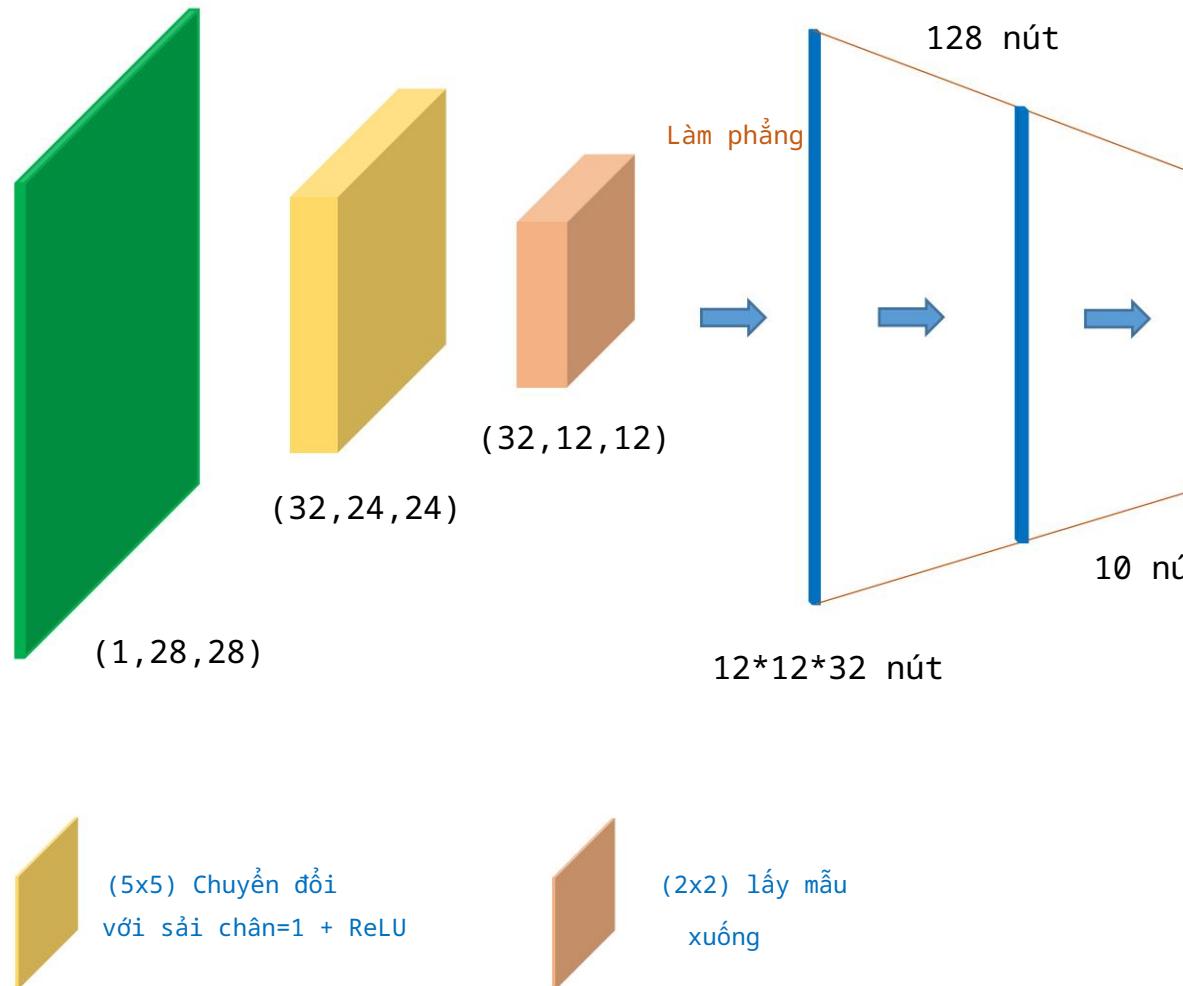
    x = self.relu(self.conv3(x))
    x = self.relu(self.conv4(x))

    x = self.relu(self.conv5(x))
    x = self.flatten(x)
    x = self.dense(x)
    return x
```



So sánh các mẫu xuống

Thiết kế mẫu



Thực hiện 1

```

class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.conv = nn.Conv2d(1, 32, kernel_size=5)
        self.pool = nn.MaxPool2d(2, 2)
        self.flatten = nn.Flatten()
        self.dense1 = nn.Linear(12*12*32, 128)
        self.dense2 = nn.Linear(128, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.conv(x)
        x = self.relu(x)
        x = self.pool(x)
        x = self.flatten(x)
        x = self.relu(self.dense1(x))
        x = self.dense2(x)
        return x
    
```

tổng hợp tối đa

```

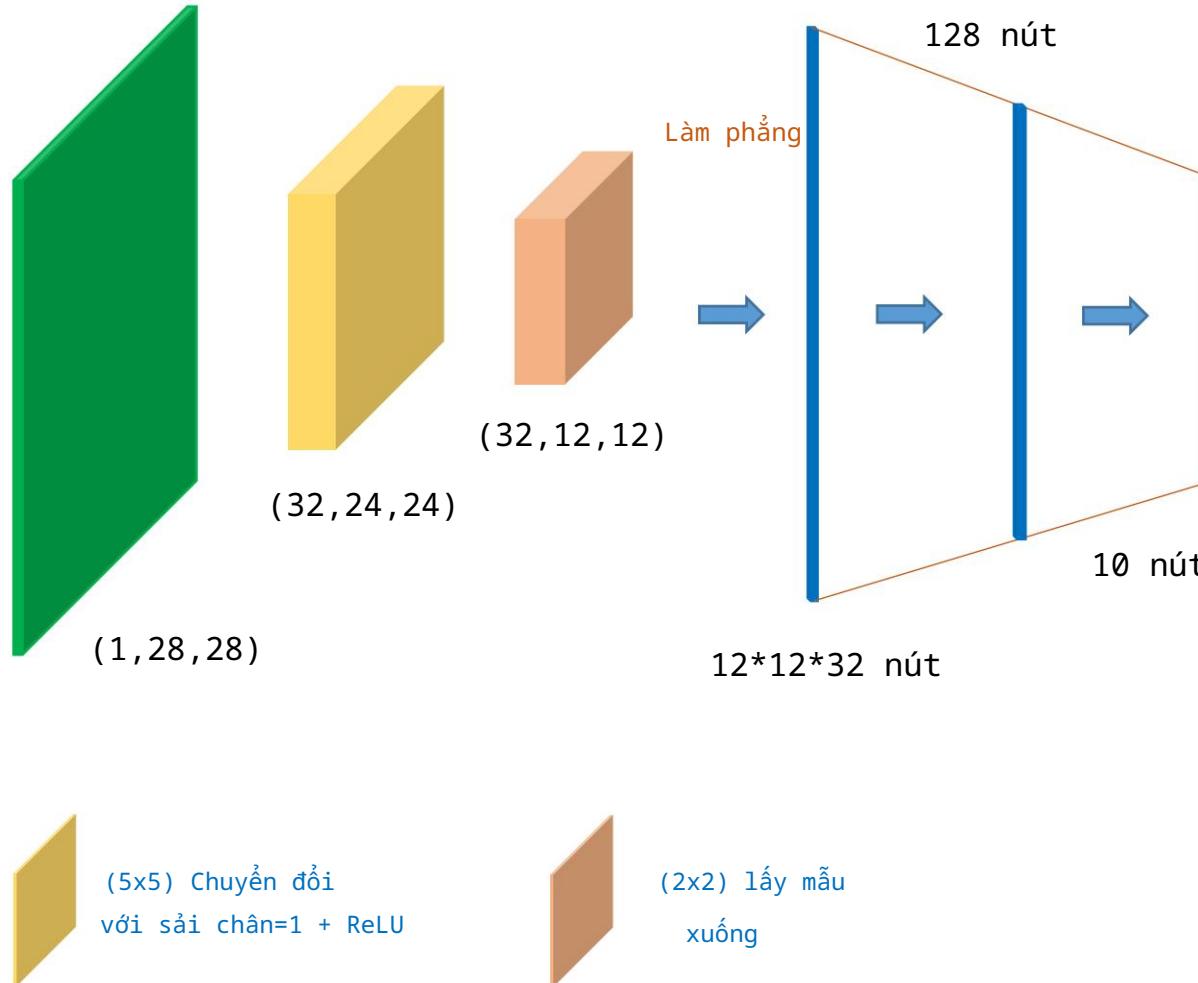
# create model
model = CustomModel()
model = model.to(device)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-5)

```

So sánh các mẫu xuống

Thiết kế mẫu



Thực hiện 2

```

class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.conv = nn.Conv2d(1, 32, kernel_size=5)
        self.pool = nn.AvgPool2d(2, 2)
        self.flatten = nn.Flatten()
        self.dense1 = nn.Linear(12*12*32, 128)
        self.dense2 = nn.Linear(128, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.conv(x)
        x = self.relu(x)
        x = self.pool(x)
        x = self.flatten(x)
        x = self.relu(self.dense1(x))
        x = self.dense2(x)
        return x
    
```

tổng hợp trung bình

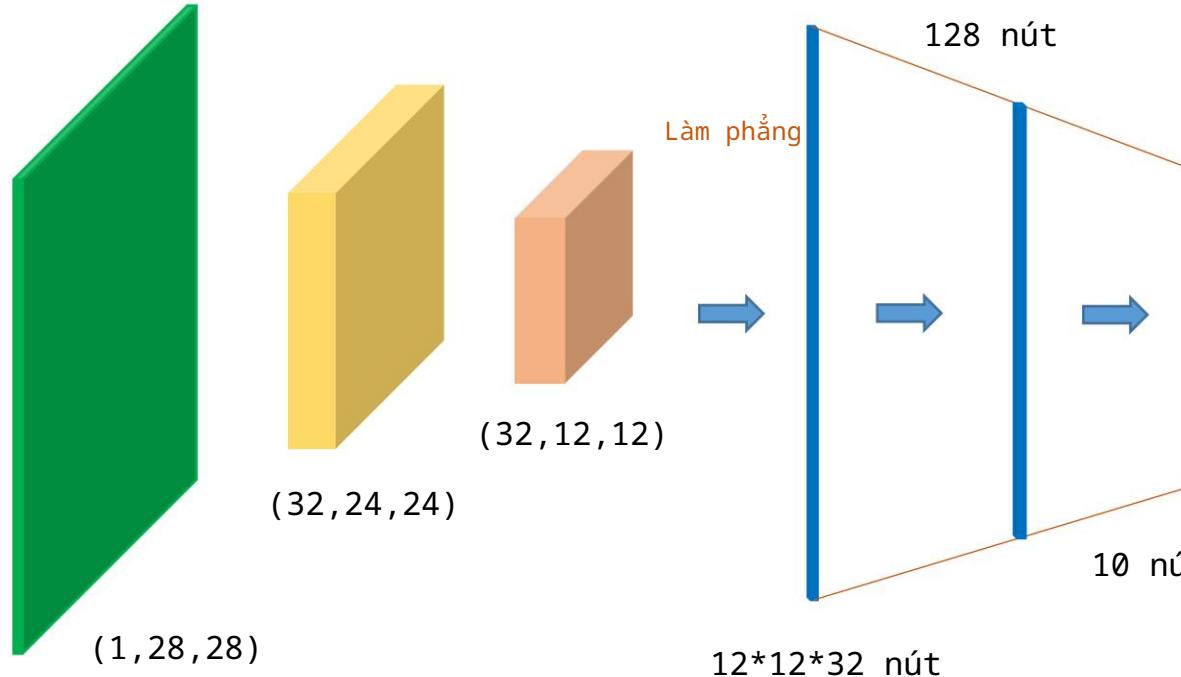
```

# create model
model = CustomModel()
model = model.to(device)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-5)
    
```

So sánh các mẫu xuống

Thiết kế mẫu



(5x5) Chuyển đổi
với sải chân=1 + ReLU

(2x2) lấy
mẫu xuống

```

class ResizeLayer(nn.Module):
    def __init__(self, scale_factor, mode='bilinear',
                 align_corners=False):
        super(ResizeLayer, self).__init__()
        self.scale_factor = scale_factor
        self.mode = mode
        self.align_corners = align_corners

    def forward(self, x):
        return F.interpolate(x, scale_factor=self.scale_factor,
                           mode=self.mode,
                           align_corners=self.align_corners)

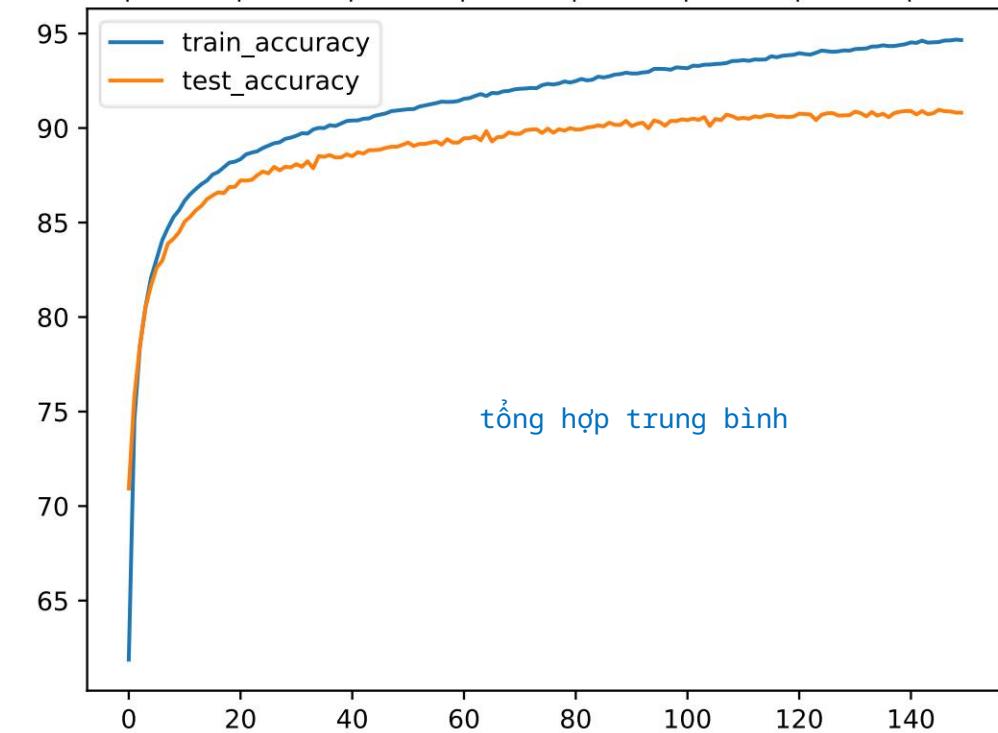
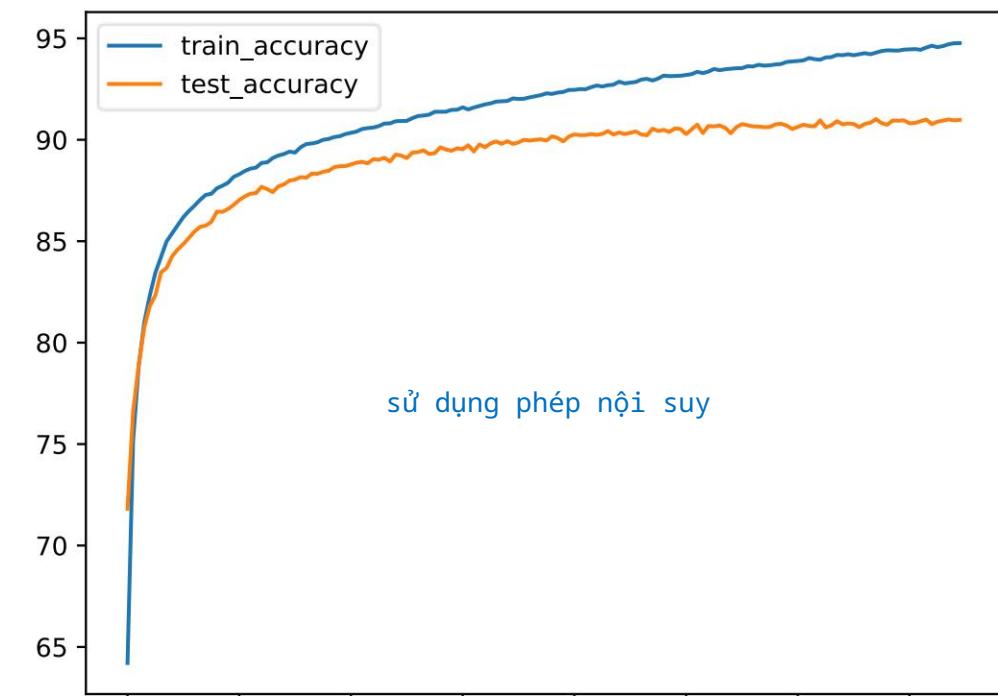
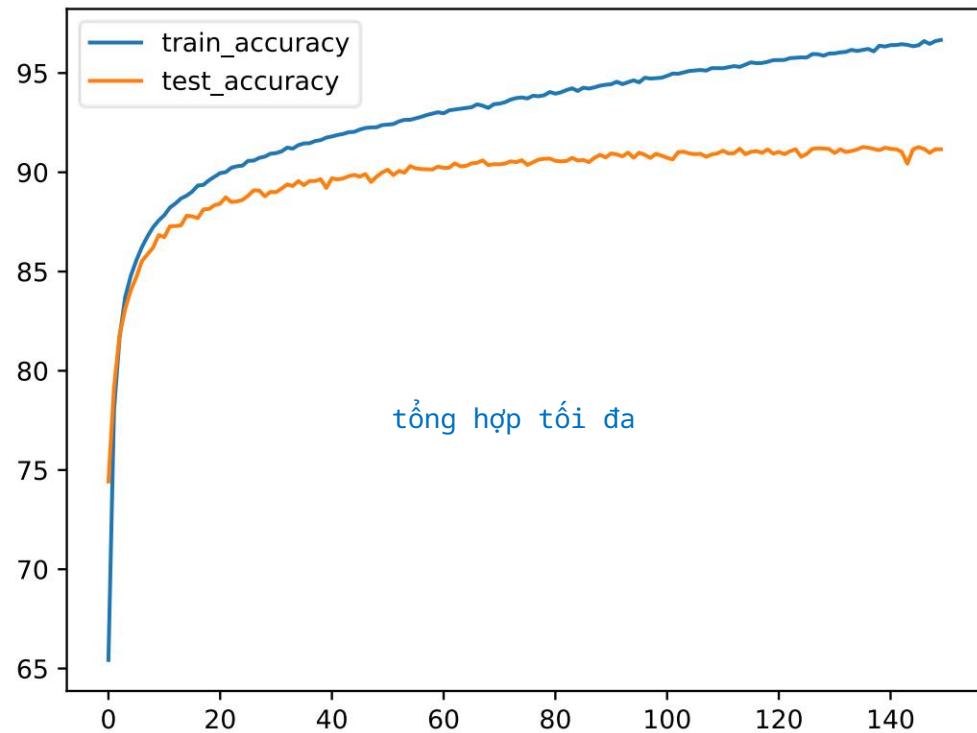
class CustomModel(nn.Module):
    def __init__(self):
        super(CustomModel, self).__init__()
        self.conv = nn.Conv2d(1, 32, kernel_size=5)
        self.resize = ResizeLayer(0.5)
        self.flatten = nn.Flatten()
        self.dense1 = nn.Linear(12*12*32, 128)
        self.dense2 = nn.Linear(128, 10)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.conv(x))
        x = self.resize(x)
        x = self.flatten(x)
        x = self.relu(self.dense1(x))
        x = self.dense2(x)
        return x

```

phép nội suy

So sánh các mẫu xuống



Khóa học tất cả trong một

Phân đệm

$$= \left[\begin{array}{c} + 2 \\ \hline \end{array} \right] + 1$$

Giữ độ phân giải
của bản đồ tính năng

1	2	3	4
5	6	7	số 8
9	10	11	12
13	14	15	16

Dữ
liệu (4x4)

1	2	3
4	5	6
7	số 8	9

Hạt nhân của tham số

Không sử dụng phân đệm
hoặc phân đệm = 0

Cây dây leo ()
với sải chân=1



đầu ra
(2x2)

1	2	3	4
5	6	7	số 8
9	10	11	12
13	14	15	16

Dữ liệu

Cây dây leo ()
với sải chân=1

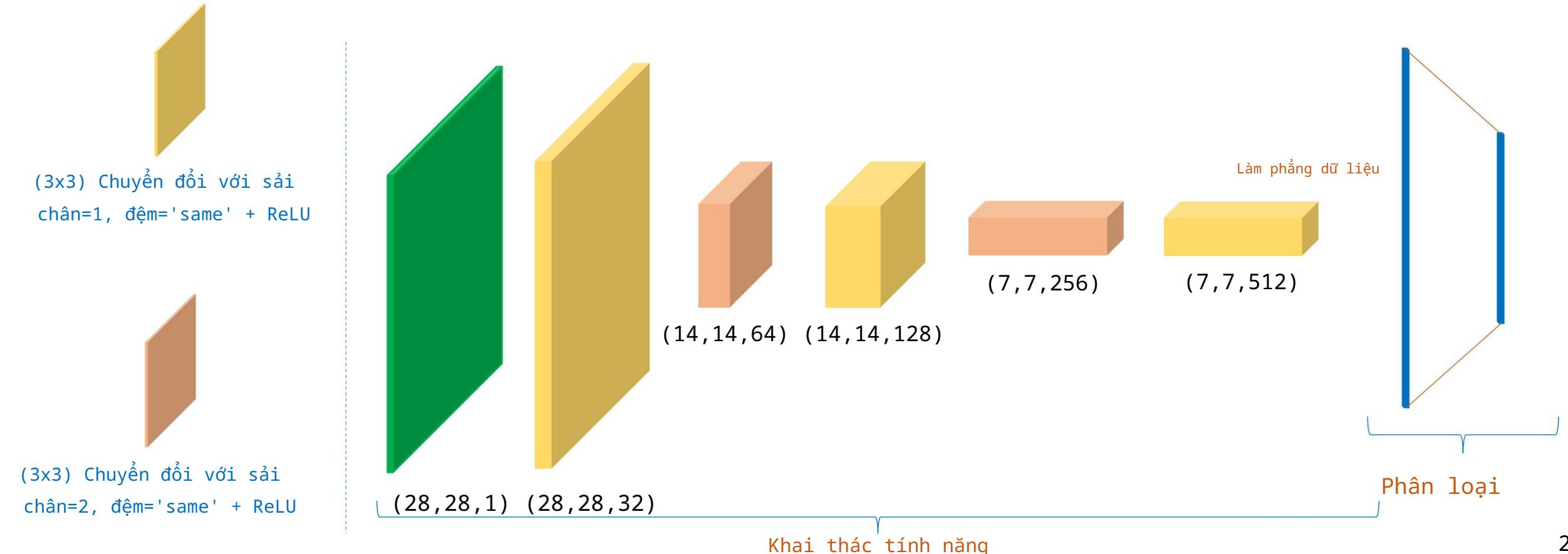
1	2	3	4
5	6	7	số 8
9	10	11	12
13	14	15	16

đầu ra
(4x4)

Phân đệm

Ví dụ

```
model = nn.Sequential(
    nn.Conv2d(1, 32, kernel_size=3, padding='same', stride=1), nn.ReLU(),
    nn.Conv2d(32, 64, kernel_size=3, padding=1, stride=2), nn.ReLU(),
    nn.Conv2d(64, 128, kernel_size=3, padding=1, stride=1), nn.ReLU(),
    nn.Conv2d(128, 256, kernel_size=3, padding=1, stride=2), nn.ReLU(),
    nn.Conv2d(256, 512, kernel_size=3, padding=1, stride=1), nn.ReLU(),
    nn.Flatten(), nn.Linear(7*7*512, 10)
)
```



Đề cương

Fashion-MNIST/Cifar10 Chỉ sử dụng Conv2d

Pooling

Padding

1x1 Convolution

LeNet và VGG Models

Trình trich xuất tính năng kỹ thuật

Phát hiện cạnh



Độ dốc



Phát hiện cạnh thông qua ước tính cường độ cạnh
Sử dụng lý luận mờ và ngưỡng tối ưu
Lựa chọn sử dụng tối ưu hóa nhóm hạt



Fuzzy-BFA

Proposed (Th = 0.18)

Proposed (Th = 0.35)

Proposed (optimal)

Đạo hàm và Ứng dụng

Tính đạo hàm trung bình theo hướng x

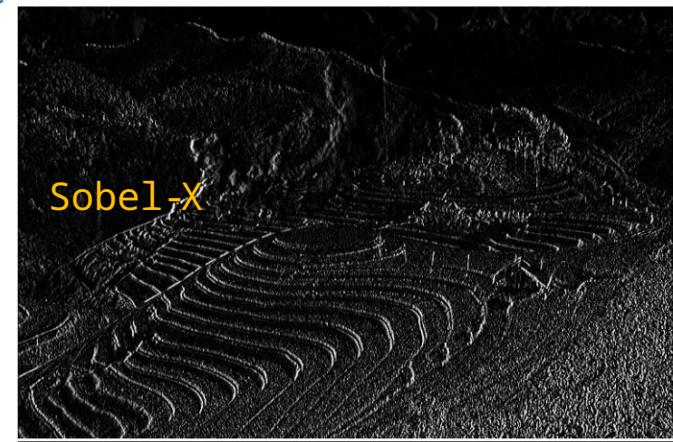
$$\begin{matrix} 1 \\ 2 \\ 1 \end{matrix} * \begin{matrix} -1 & 0 & 1 \end{matrix} = \begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix}$$

weighted average x-derivative Sobel for x direction

Tính đạo hàm trung bình theo hướng y

$$\begin{matrix} 1 \\ 0 \\ -1 \end{matrix} * \begin{matrix} 1 & 2 & 1 \end{matrix} = \begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix}$$

y-derivative weighted average Sobel for y direction

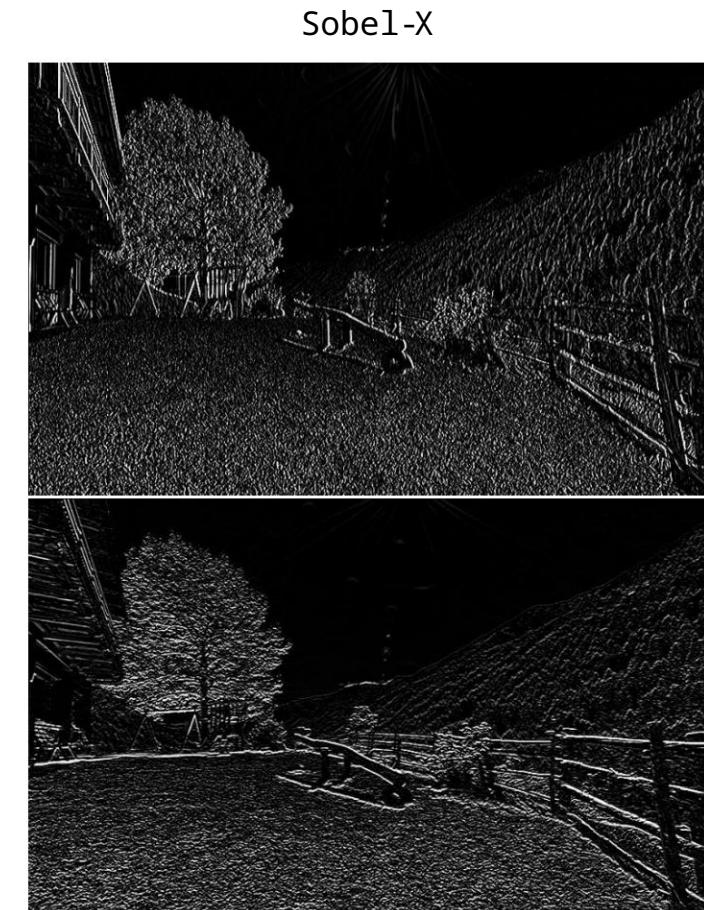


Đạo hàm và Ứng dụng

Phát hiện cạnh



Phát
hiện cạnh



$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * I$$

Trình trich xuất tính năng kỹ thuật

Phát hiện cạnh

Tính đạo hàm trung bình theo hướng x

$$\begin{array}{|c|} \hline 1 \\ \hline 2 \\ \hline 1 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array}$$

weighted average x-derivative Sobel for x direction

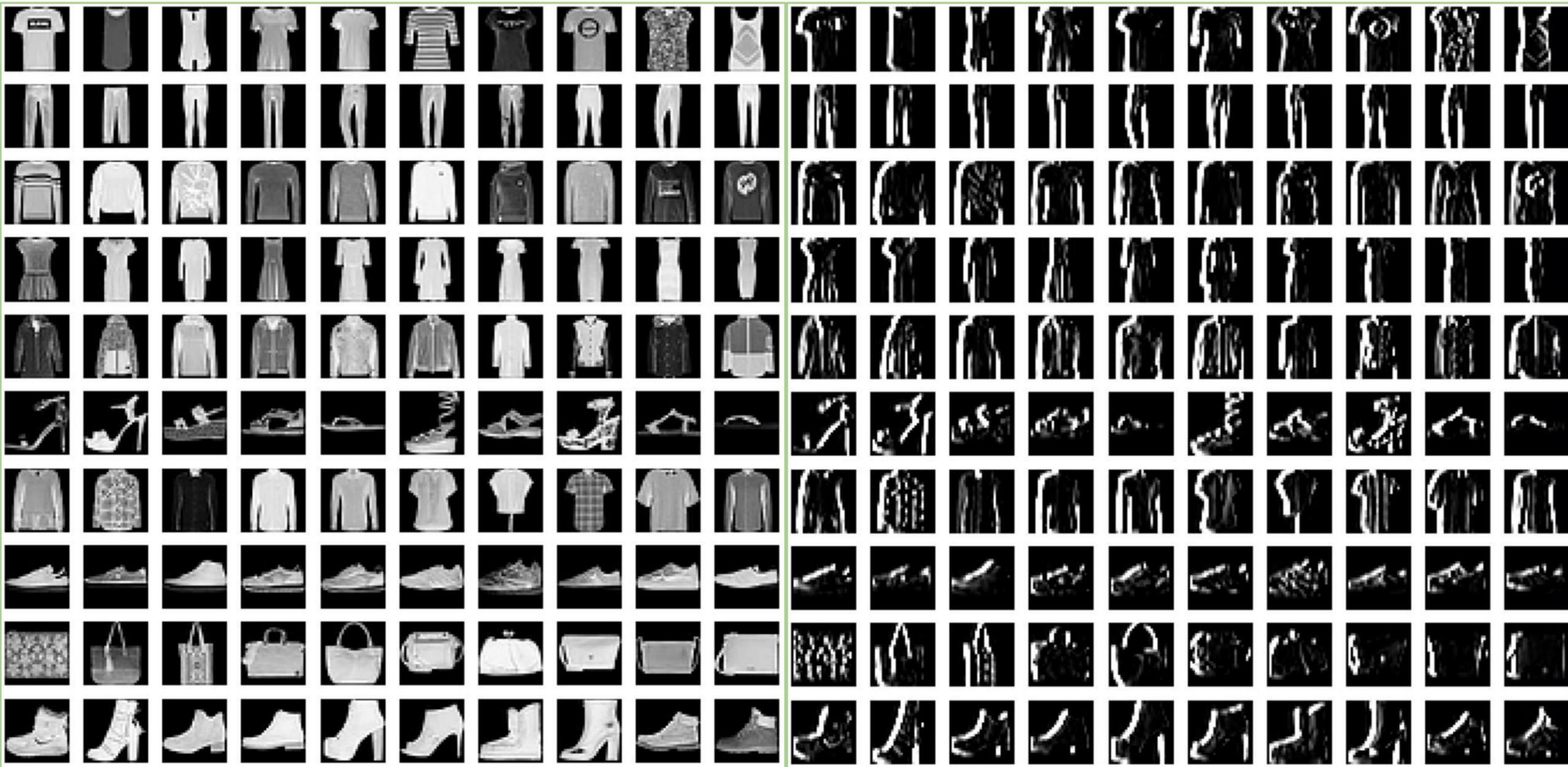
```
@staticmethod
def _sobel(image, ksize=3):
    sobel_x = cv2.Sobel(image, cv2.CV_32F, 1, 0, ksize=ksize)
    sobel_y = cv2.Sobel(image, cv2.CV_32F, 0, 1, ksize=ksize)
    sobel_x = torch.from_numpy(sobel_x)
    sobel_y = torch.from_numpy(sobel_y)
    sobel_magnitude = torch.hypot(sobel_x, sobel_y)
    return sobel_magnitude
```

Tính đạo hàm trung bình theo hướng y

$$\begin{array}{|c|} \hline 1 \\ \hline 0 \\ \hline -1 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array}$$

y-derivative weighted average Sobel for y direction

Sobel_X



Sobel_Y



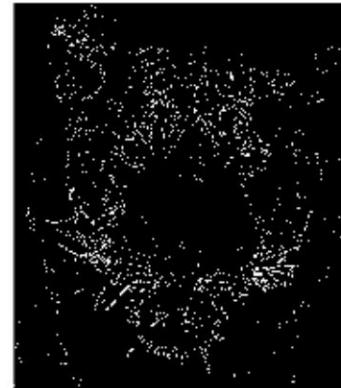
Trình trich xuất tính năng kỹ thuật

Phương pháp truyền thống

Laplacian



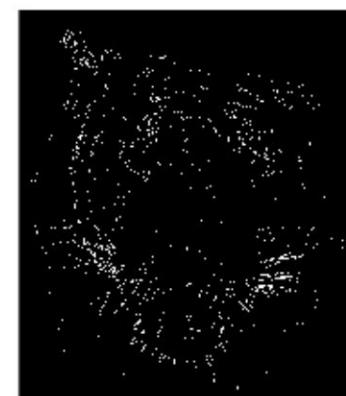
Prewitt



Sobel



Roberts

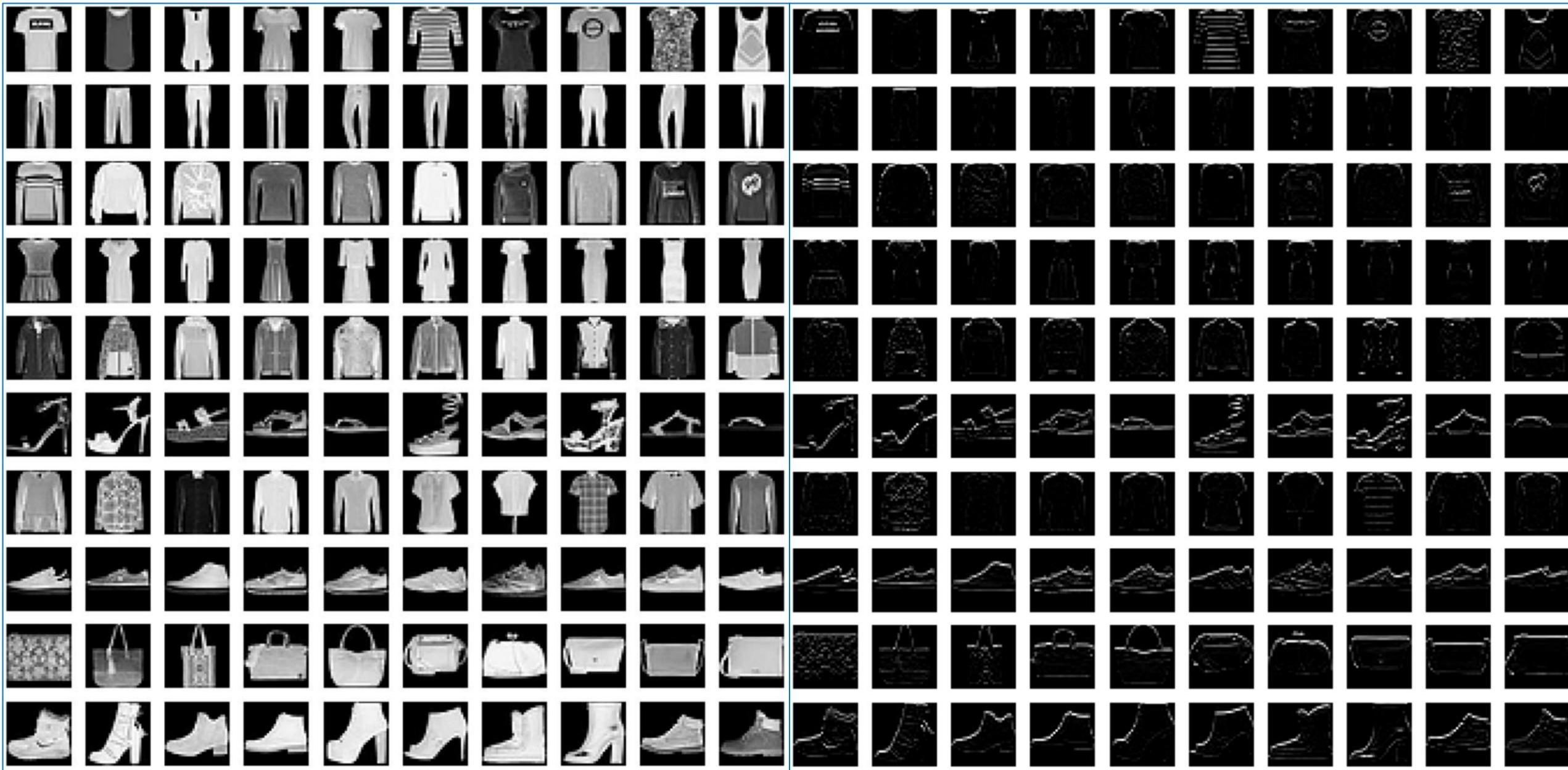


```
@staticmethod
def _scharr(image):
    scharr_x = cv2.Scharr(image, cv2.CV_32F, 1, 0)
    scharr_y = cv2.Scharr(image, cv2.CV_32F, 0, 1)
    scharr_x = torch.from_numpy(scharr_x)
    scharr_y = torch.from_numpy(scharr_y)
    scharr_magnitude = torch.hypot(scharr_x, scharr_y)
    return scharr_magnitude
```

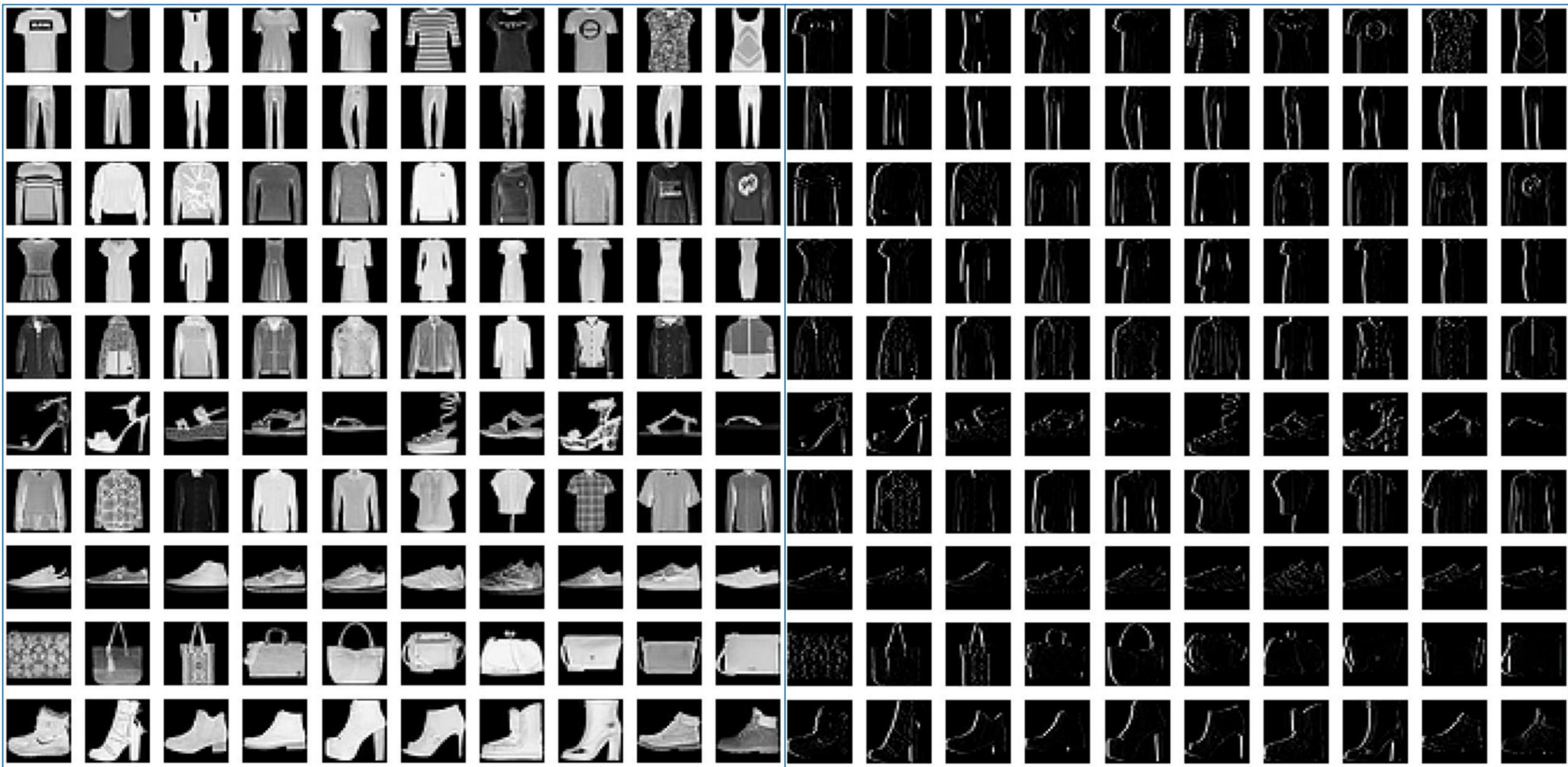
```
@staticmethod
```

```
def _laplacian(image):
    laplacian_img = cv2.Laplacian(image, cv2.CV_32F)
    laplacian_img = torch.from_numpy(laplacian_img)
    return laplacian_img
```

Độ dốc_X

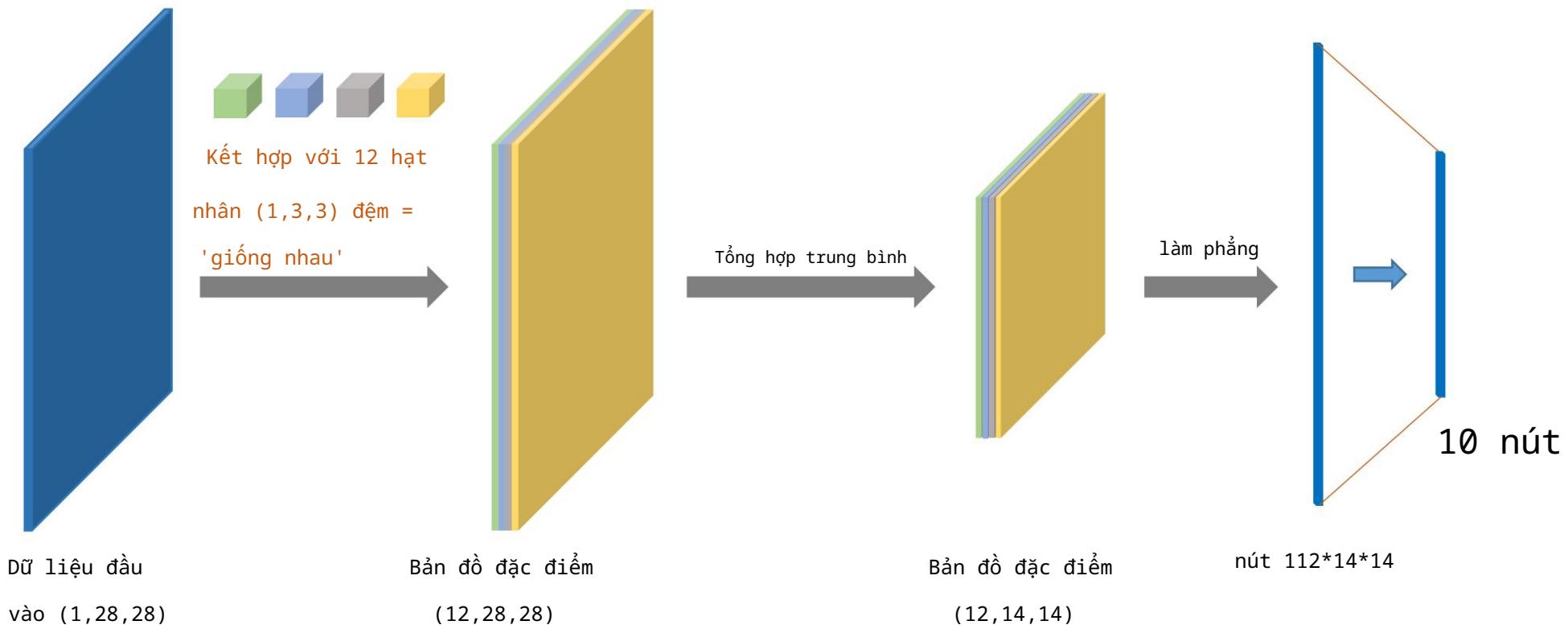


Độ dốc_Y



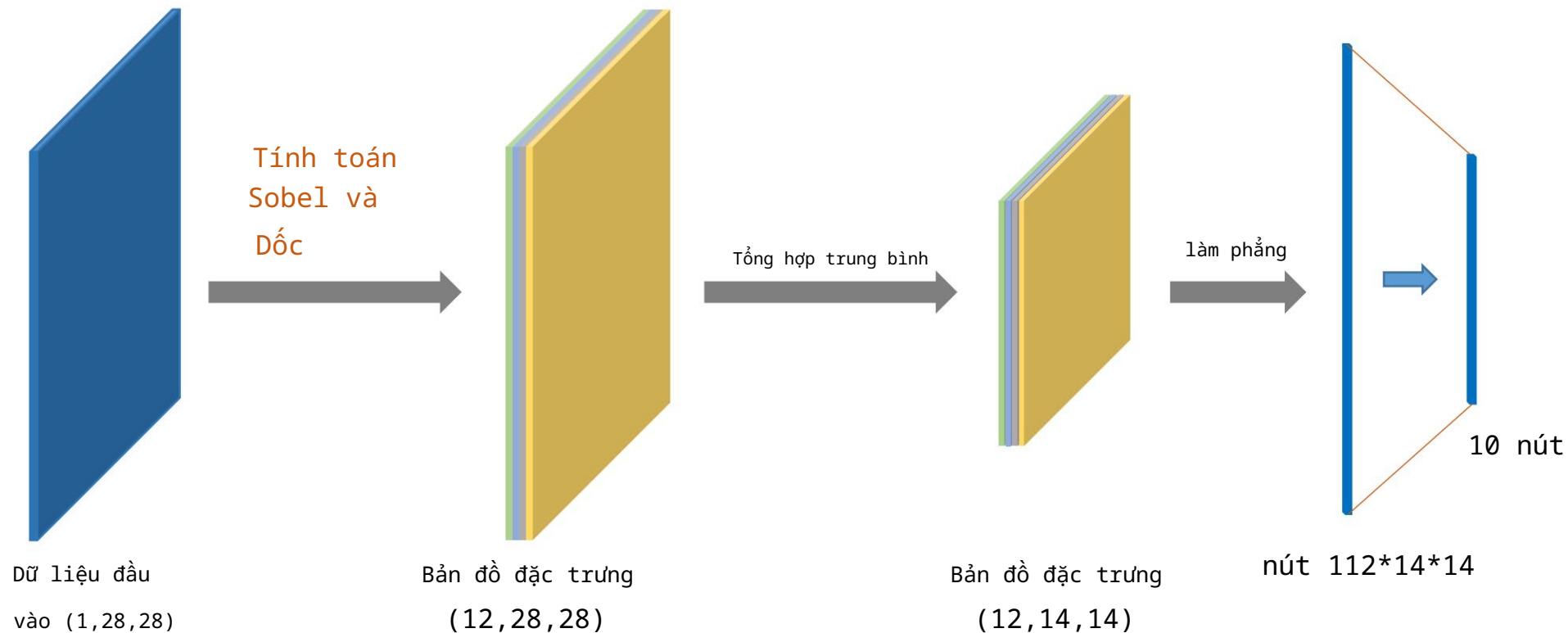
So sánh

Tính năng kỹ thuật so với tính năng CNN

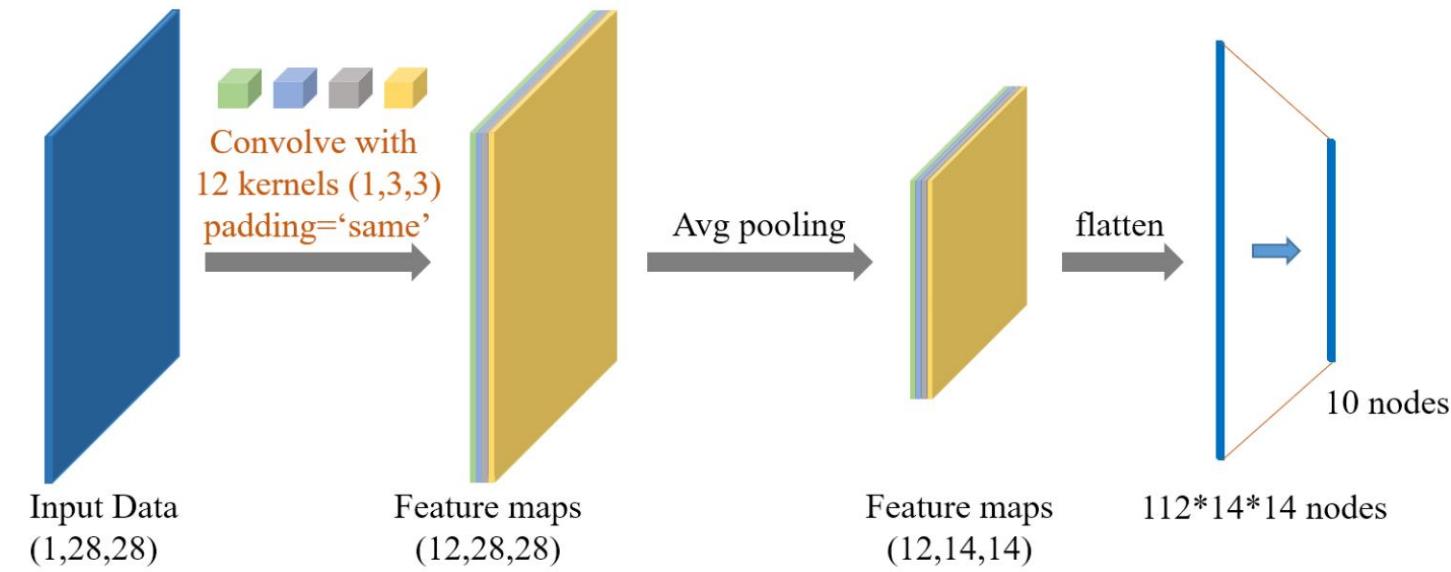
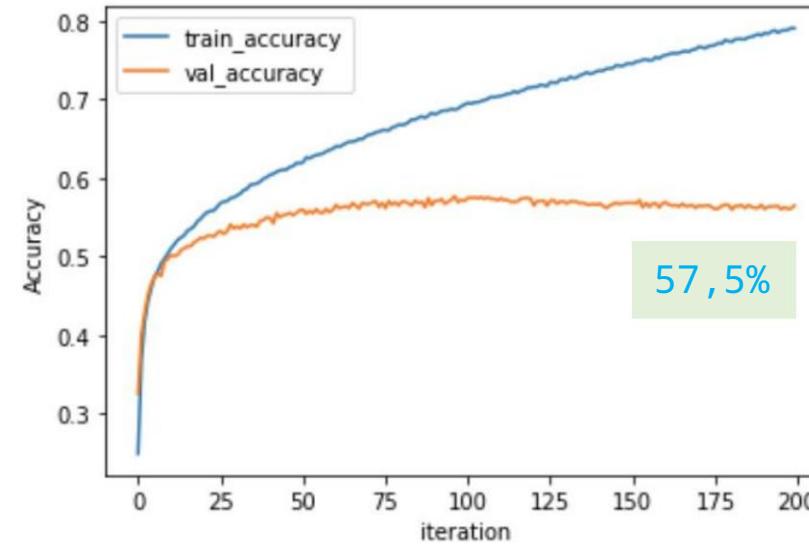
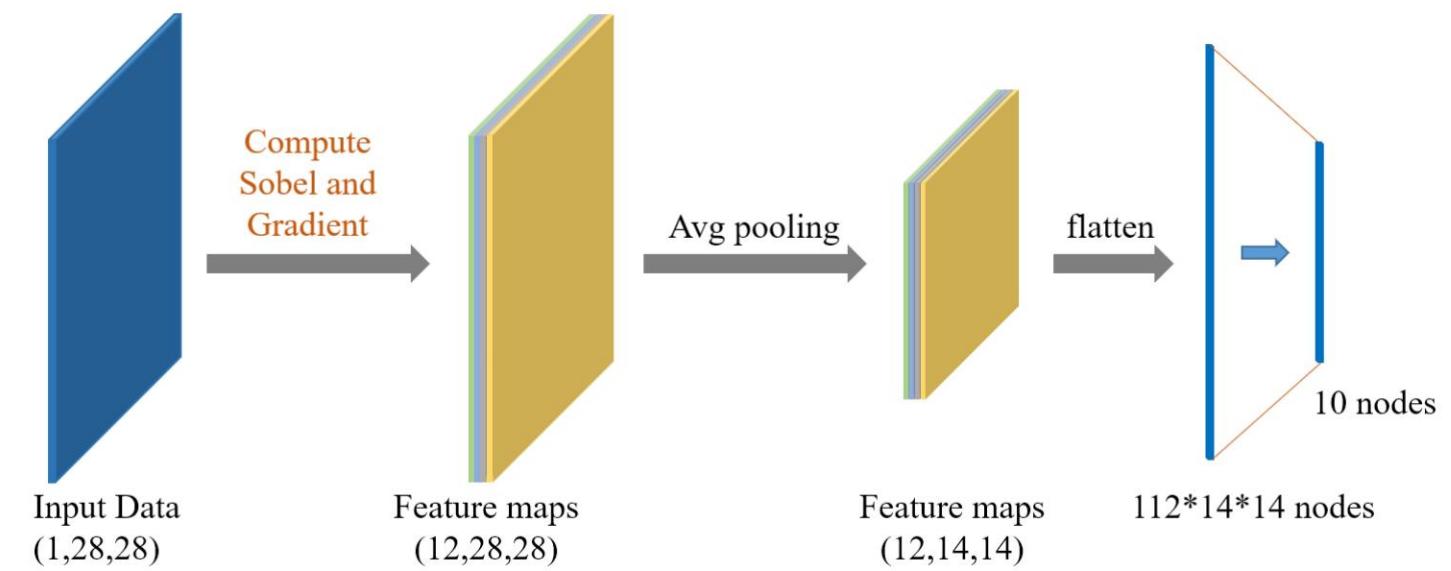
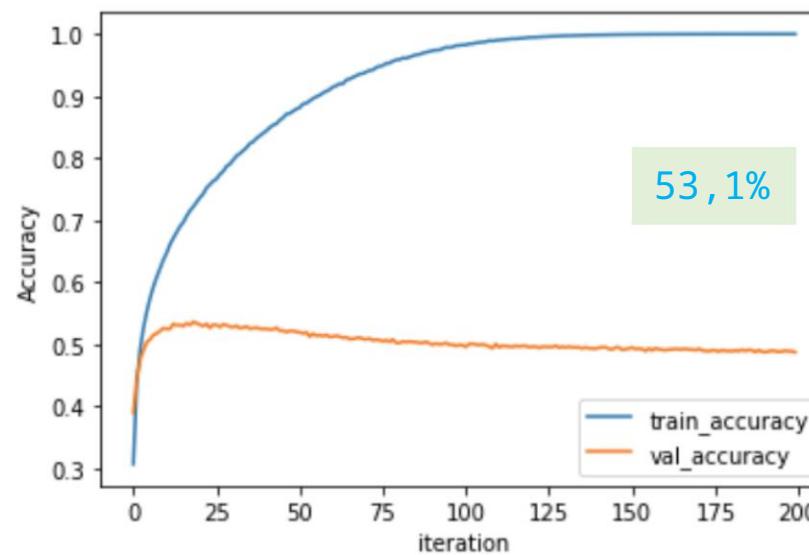


So sánh

Tính năng kỹ thuật so với tính năng CNN



Tính năng kỹ thuật so với tính năng CNN sử dụng Cifar10



Đề cương

Fashion-MNIST/Cifar10 Chỉ sử dụng Conv2d

Pooling

Padding

1x1 Convolution

LeNet và VGG Models

Tích chập 1x1

Tại sao nên sử dụng Convolution 1x1

Kích thước đầu vào linh hoạt



Yann LeCun

April 7, 2015 ·

...

In Convolutional Nets, there is no such thing as "fully-connected layers".

There are only convolution layers with 1x1 convolution kernels and a full connection table.

It's a too-rarely-understood fact that ConvNets don't need to have a fixed-size input. You can train them on inputs that happen to produce a single output vector (with no spatial extent), and then apply them to larger images. Instead of a single output vector, you then get a spatial map of output vectors. Each vector sees input windows at different locations on the input.

In that scenario, the "fully connected layers" really act as 1x1 convolutions.

Yann LeCun

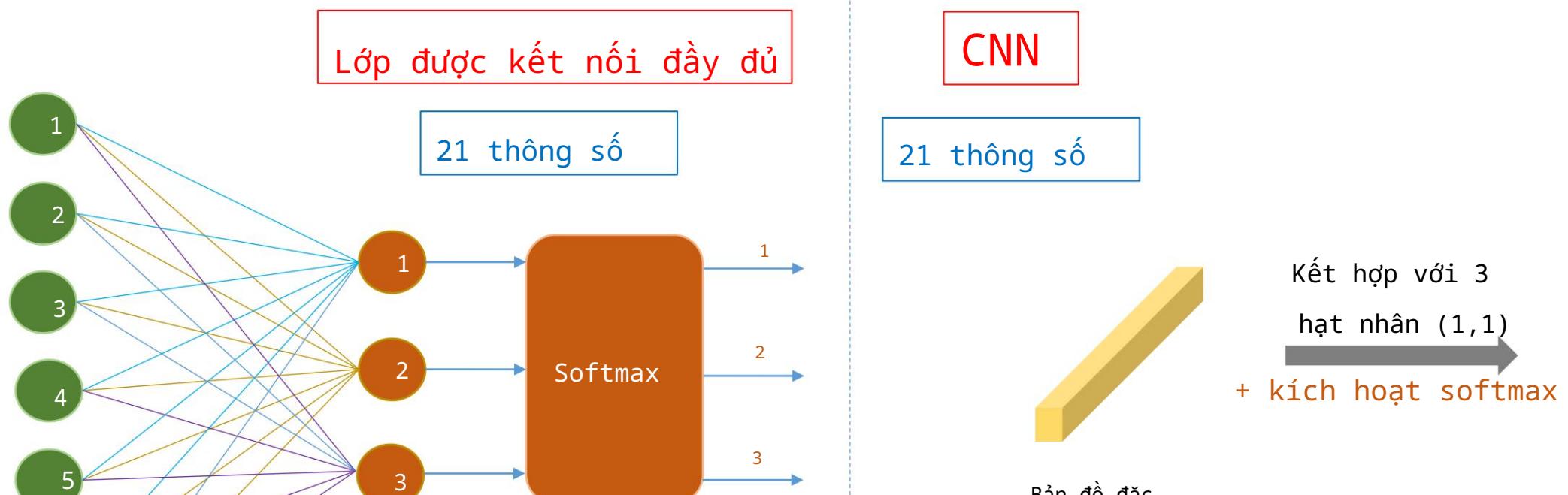


Yann LeCun in 2018

Born	July 8, 1960 (age 60) Soisy-sous-Montmorency, France
Alma mater	ESIEE Paris (MSc) Pierre and Marie Curie University (PhD)
Known for	Deep learning
Awards	Turing Award (2018) AAAI Fellow (2019) Legion of Honour (2020)
	Scientific career
Institutions	Bell Labs (1988-1996) New York University Facebook
Thesis	<i>Modèles connexionnistes de l'apprentissage (connectionist learning models)</i> (1987a)
Doctoral advisor	Maurice Milgram
Website	yann.lecun.com

Tích chập 1×1

So sánh



Tích chập 1×1

Thay thế FC bằng Chuyển đổi 1×1



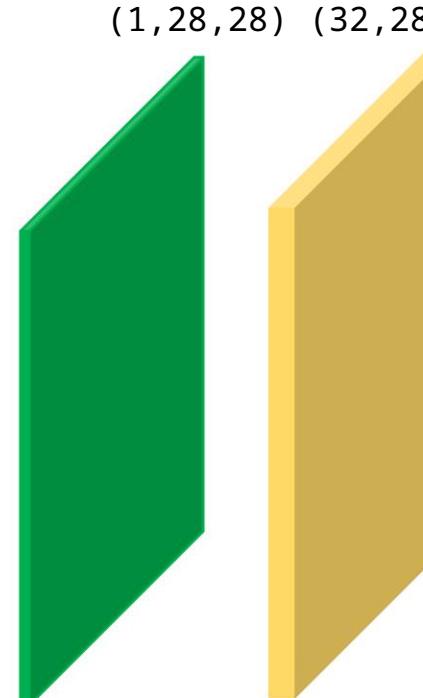
(3x3) Chuyển đổi với sải
chân=1, đệm='same' + ReLU



(3x3) Chuyển đổi với sải
chân=2, đệm='same' + ReLU

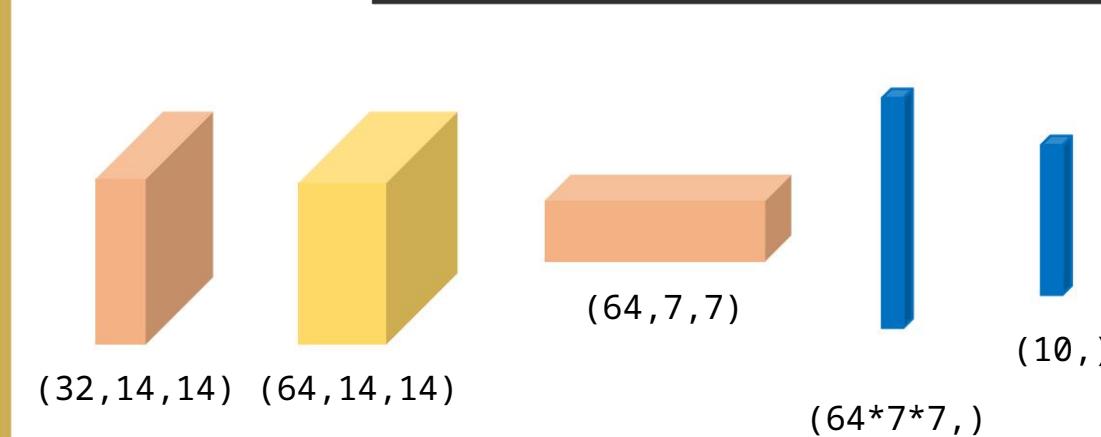


(7x7) Chuyển đổi + ReLU



```
LeNet = nn.Sequential(
    nn.Conv2d(1, 32, kernel_size=3, padding=1), nn.ReLU(), nn.MaxPool2d(2, 2),
    nn.Conv2d(32, 64, kernel_size=3, padding=1), nn.ReLU(), nn.MaxPool2d(2, 2),
    nn.Flatten(),
    nn.Linear(7*7*64, 128),
    nn.ReLU(),
    nn.Linear(128, 10)
)
```

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 32, 28, 28]	320
ReLU-2	[-1, 32, 28, 28]	0
MaxPool2d-3	[-1, 32, 14, 14]	0
Conv2d-4	[-1, 64, 14, 14]	18,496
ReLU-5	[-1, 64, 14, 14]	0
MaxPool2d-6	[-1, 64, 7, 7]	0
Flatten-7	[-1, 3136]	0
Linear-8	[-1, 128]	401,536
ReLU-9	[-1, 128]	0
Linear-10	[-1, 10]	1,290

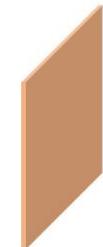


Tích chập 1×1

Thay thế FC bằng Chuyển đổi 1×1



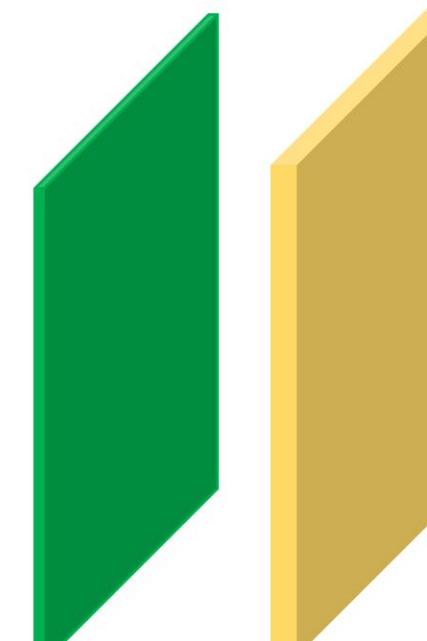
(3x3) Chuyển đổi với sải
chân=1, đậm='same' + ReLU



(3x3) Chuyển đổi với sải
chân=2, đậm='same' + ReLU



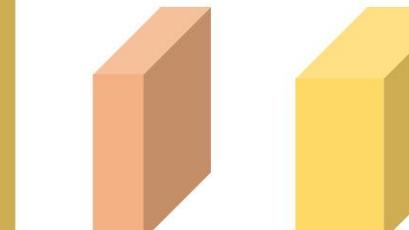
(7x7) Chuyển đổi + ReLU



(1, 28, 28)

Khai thác tính năng

(32, 14, 14) (64, 14, 14)



(64, 7, 7)



(128, 1, 1)

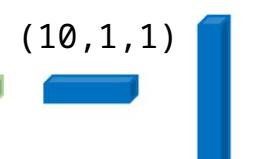


(10,)

Phân loại

```
LeNet = nn.Sequential(
    nn.Conv2d(1, 32, kernel_size=3, padding=1), nn.ReLU(), nn.MaxPool2d(2, 2),
    nn.Conv2d(32, 64, kernel_size=3, padding=1), nn.ReLU(), nn.MaxPool2d(2, 2),
    nn.Conv2d(64, 128, kernel_size=7), nn.ReLU(),
    nn.Conv2d(128, 10, kernel_size=1),
    nn.Flatten()
)
```

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 32, 28, 28]	320
ReLU-2	[-1, 32, 28, 28]	0
MaxPool2d-3	[-1, 32, 14, 14]	0
Conv2d-4	[-1, 64, 14, 14]	18,496
ReLU-5	[-1, 64, 14, 14]	0
MaxPool2d-6	[-1, 64, 7, 7]	0
Conv2d-7	[-1, 128, 1, 1]	401,536
ReLU-8	[-1, 128, 1, 1]	0
Conv2d-9	[-1, 10, 1, 1]	1,290
Flatten-10	[-1, 10]	0



Đề cương

Fashion-MNIST/Cifar10 Chỉ sử dụng Conv2d

Pooling

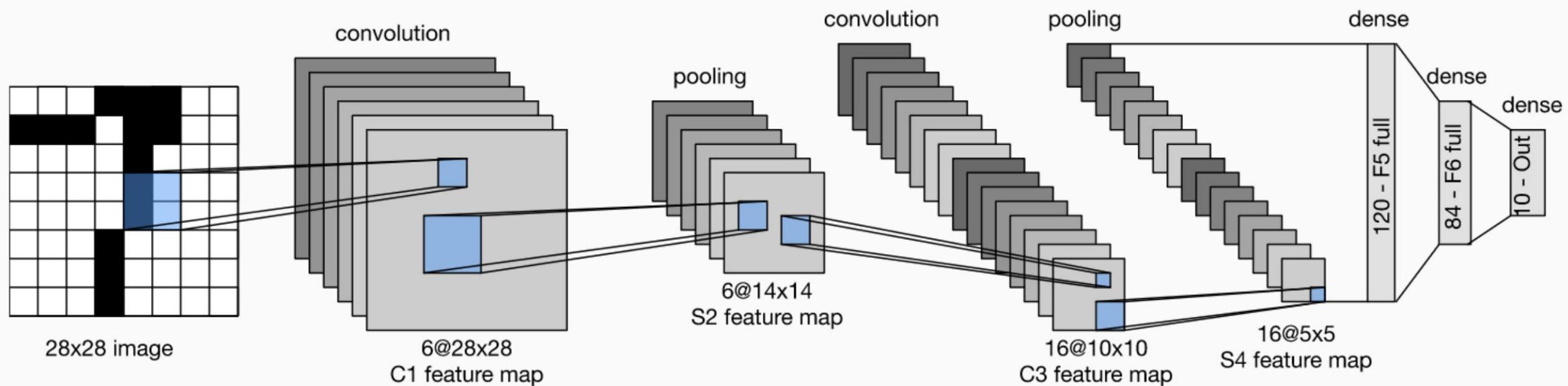
Padding

1x1 Convolution

LeNet và VGG Models

Kiến trúc LeNet

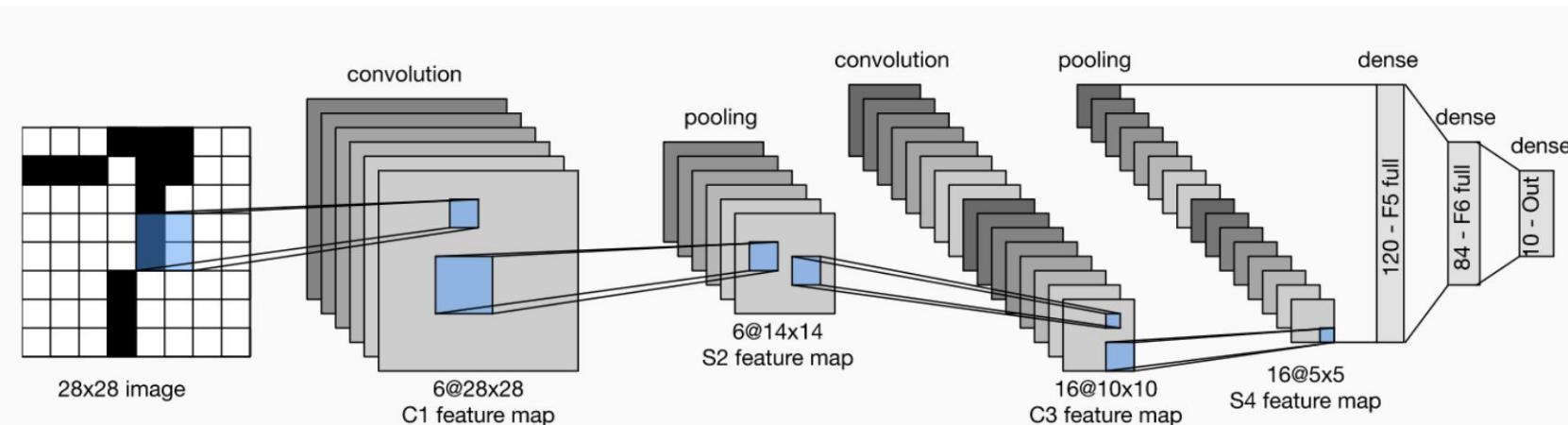
Xây dựng mô hình



LeNet

Ngành kiến trúc

Xây dựng mô hình



```
LeNet = nn.Sequential(
    nn.Conv2d(1, 6, 5, padding=2),
    nn.ReLU(),
    nn.MaxPool2d(2, 2),

    nn.Conv2d(6, 16, 5),
    nn.ReLU(),
    nn.MaxPool2d(2, 2),

    nn.Flatten(),
    nn.Linear(16*5*5, 120),
    nn.ReLU(),

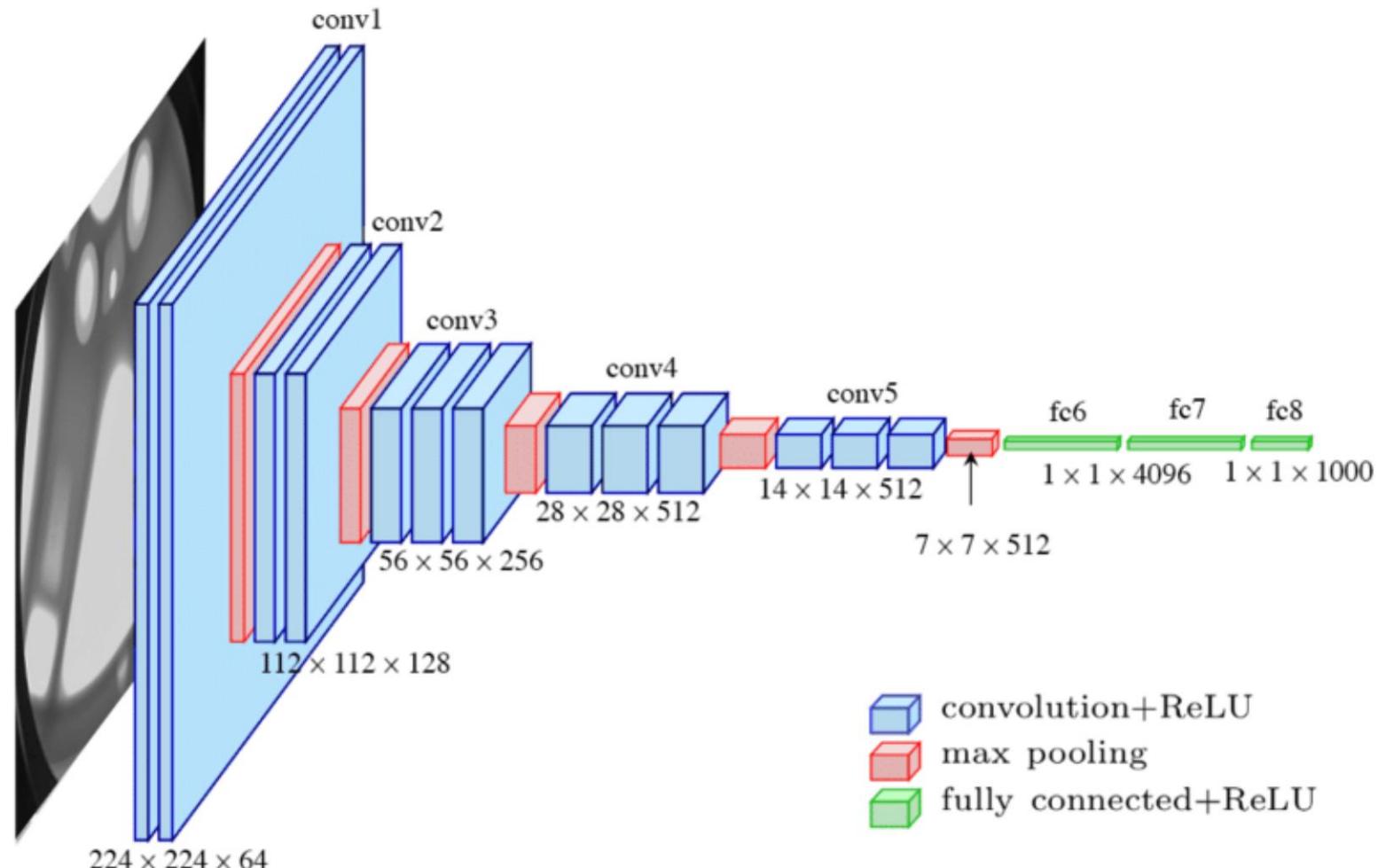
    nn.Linear(120, 84),
    nn.ReLU(),
    nn.Linear(84, 10)
)
```

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 6, 28, 28]	156
ReLU-2	[-1, 6, 28, 28]	0
MaxPool2d-3	[-1, 6, 14, 14]	0
Conv2d-4	[-1, 16, 10, 10]	2,416
ReLU-5	[-1, 16, 10, 10]	0
MaxPool2d-6	[-1, 16, 5, 5]	0
Flatten-7	[-1, 400]	0
Linear-8	[-1, 120]	48,120
ReLU-9	[-1, 120]	0
Linear-10	[-1, 84]	10,164
ReLU-11	[-1, 84]	0
Linear-12	[-1, 10]	850

Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0

Kiến trúc VGG16

Xây dựng mô hình



```

self.relu = nn.ReLU()
self.features = nn.Sequential(
    nn.Conv2d(3, 64, kernel_size=3, padding=1), self.relu,
    nn.Conv2d(64, 64, kernel_size=3, padding=1), self.relu,
    nn.MaxPool2d(kernel_size=2, stride=2),

    nn.Conv2d(64, 128, kernel_size=3, padding=1), self.relu,
    nn.Conv2d(128, 128, kernel_size=3, padding=1), self.relu,
    nn.MaxPool2d(kernel_size=2, stride=2),

    nn.Conv2d(128, 256, kernel_size=3, padding=1), self.relu,
    nn.Conv2d(256, 256, kernel_size=3, padding=1), self.relu,
    nn.Conv2d(256, 256, kernel_size=3, padding=1), self.relu,
    nn.MaxPool2d(kernel_size=2, stride=2),

    nn.Conv2d(256, 512, kernel_size=3, padding=1), self.relu,
    nn.Conv2d(512, 512, kernel_size=3, padding=1), self.relu,
    nn.Conv2d(512, 512, kernel_size=3, padding=1), self.relu,
    nn.MaxPool2d(kernel_size=2, stride=2),

    nn.Conv2d(512, 512, kernel_size=3, padding=1), self.relu,
    nn.Conv2d(512, 512, kernel_size=3, padding=1), self.relu,
    nn.Conv2d(512, 512, kernel_size=3, padding=1), self.relu,
    nn.MaxPool2d(kernel_size=2, stride=2)
)

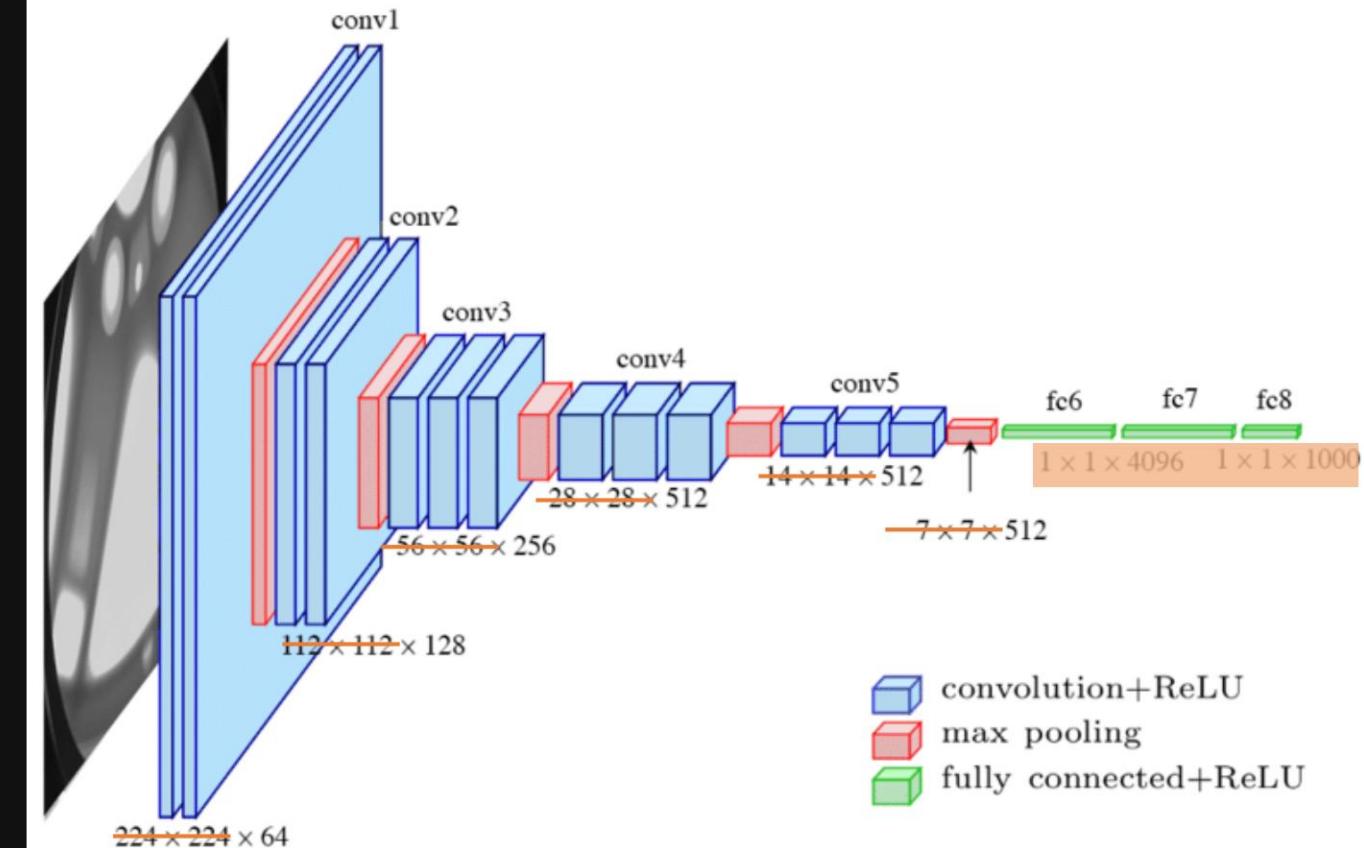
self.classifier = nn.Sequential(
    nn.Flatten(), nn.Linear(512 * 1 * 1, 128), self.relu,
    nn.Linear(128, 128), self.relu, nn.Linear(128, 10)
)

```

VGG16

Ngành kiến trúc

Xây dựng mô hình



Đọc thêm

Đọc

<https://cs231n.github.io/convolutional-networks/>

<https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>

<https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks>



AI VIETNAM

Khóa học tất cả trong một

Đào tạo CNN

Làm thế nào để tăng độ chính xác đào tạo?

Quang-Vinh Dinh

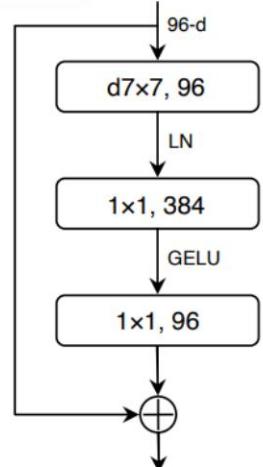
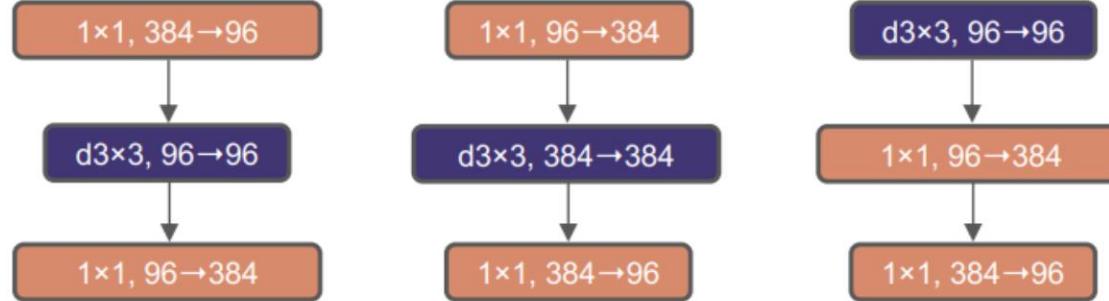
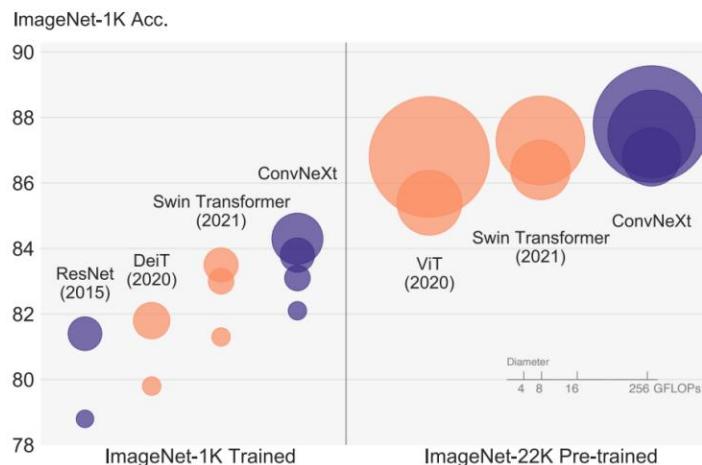
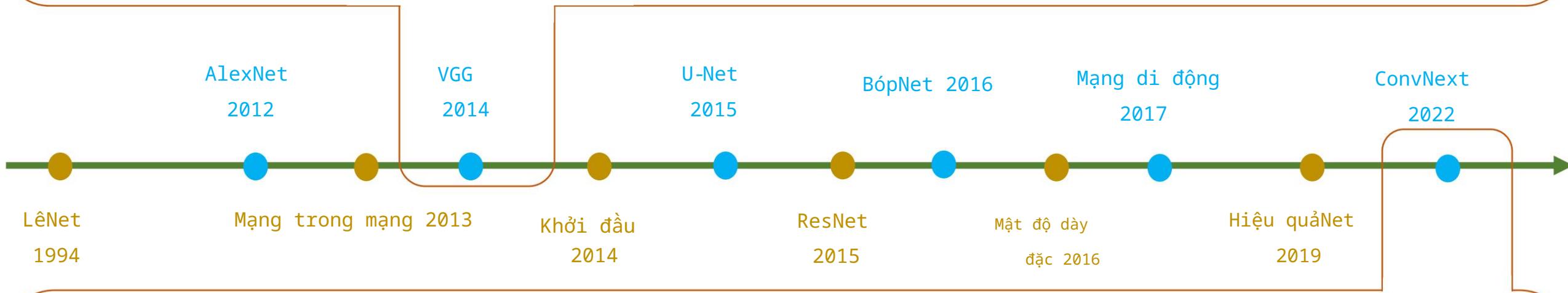
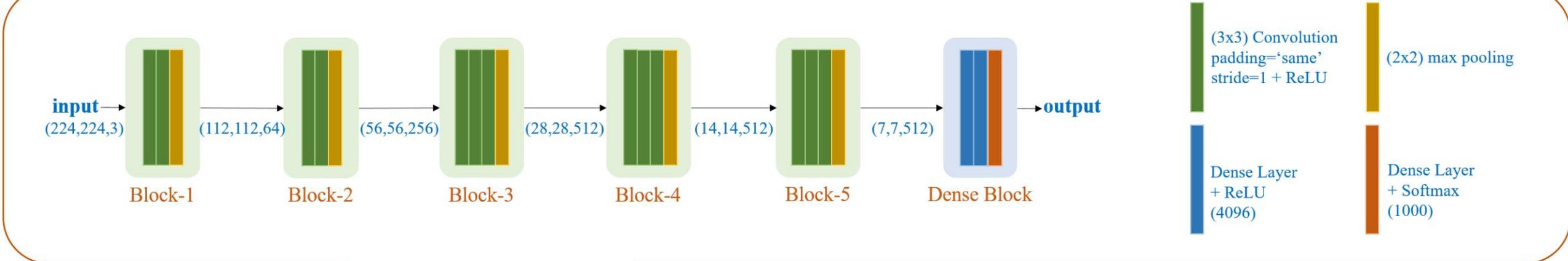
Ph.D. in Computer Science

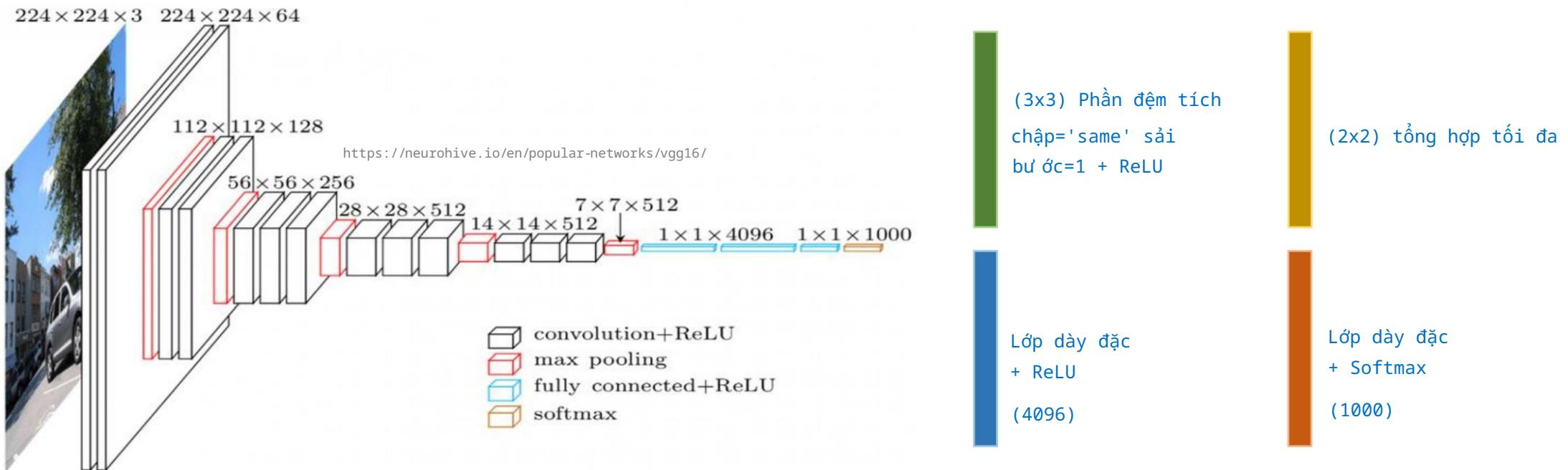
Đề cương

Kiến trúc mạng Đào

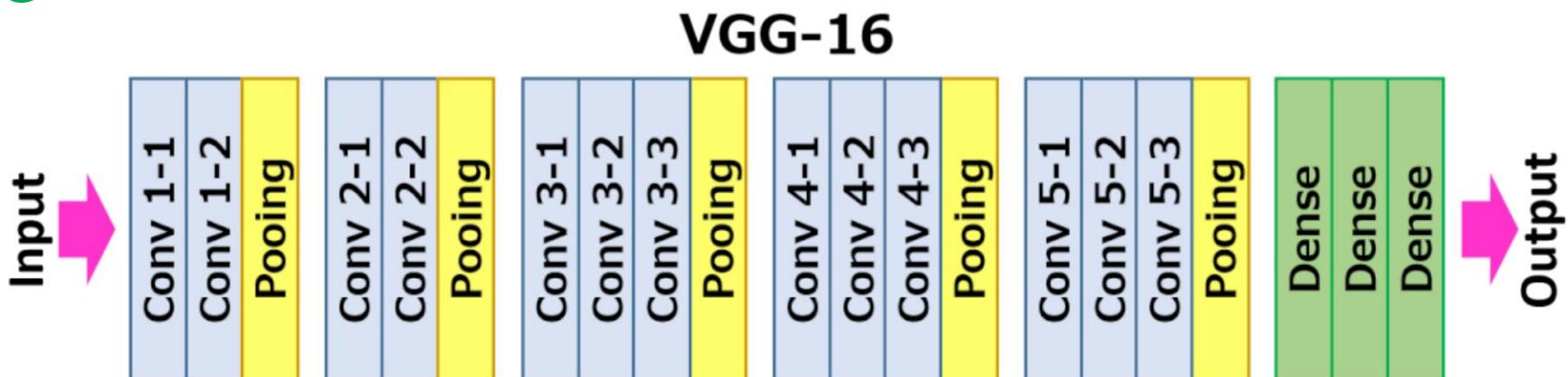
tạo mạng Nghiên
cứu điển hình

Phương pháp giải quyết vấn đề



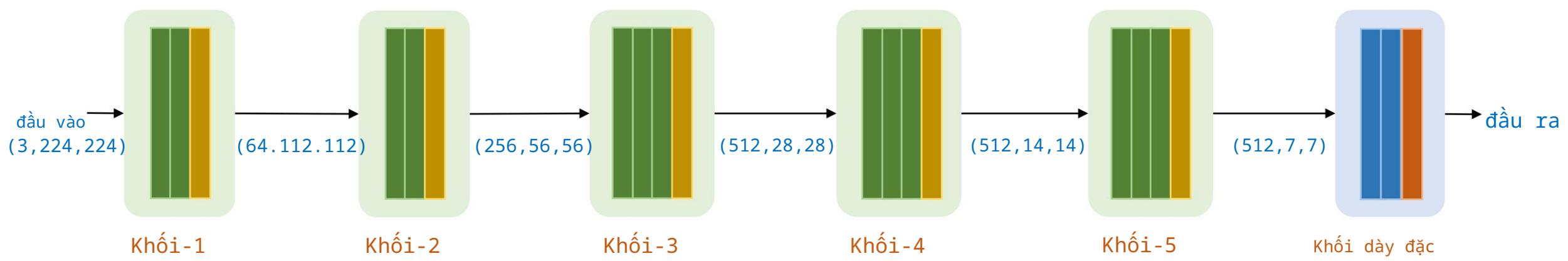


VGG16



Kiến trúc CNN

VGG16 cho ImageNet



(3x3) Tích chập
đệm='giống nhau'
sải bư ớc=1 + ReLU

(2x2) tổng hợp tối đa

Lớp dày đặc
+ ReLU
(4096)

Lớp dày đặc
+ Softmax
(1000)

```
# Define the blocks
block1 = nn.Sequential(
    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),
)
block2 = nn.Sequential(
    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),
)
block3 = nn.Sequential(
    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),
)
block4 = nn.Sequential(
    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),
)
block5 = nn.Sequential(
    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),
)
```

```
# Classifier
classifier = nn.Sequential(
    nn.Flatten(),
    nn.Linear(512*7*7, 4096), nn.ReLU(inplace=True),
    nn.Linear(4096, 4096), nn.ReLU(inplace=True),
    nn.Linear(4096, 1000),
)

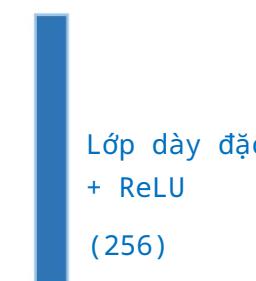
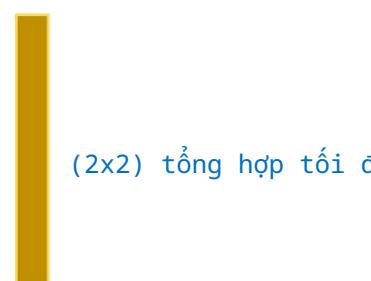
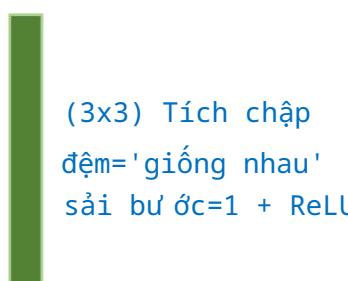
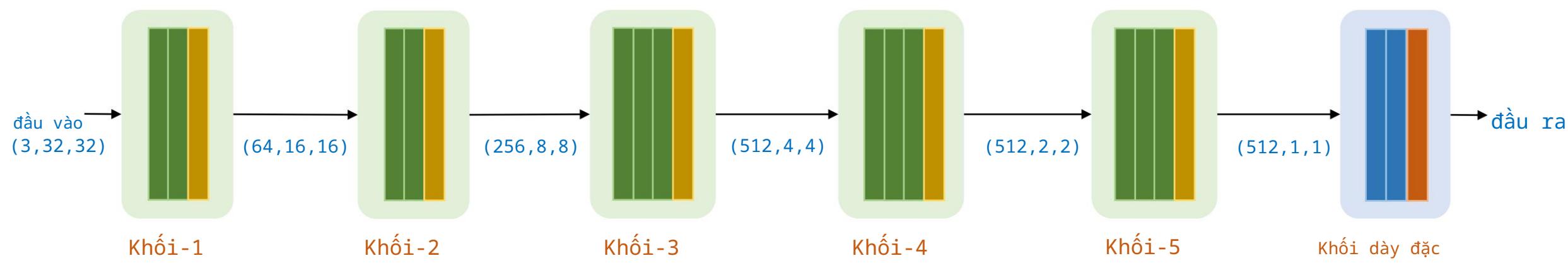
# Combine all blocks into one model
class VGG16(nn.Module):
    def __init__(self):
        super(VGG16, self).__init__()
        self.block1 = block1
        self.block2 = block2
        self.block3 = block3
        self.block4 = block4
        self.block5 = block5
        self.classifier = classifier

    def forward(self, x):
        x = self.block1(x)
        x = self.block2(x)
        x = self.block3(x)
        x = self.block4(x)
        x = self.block5(x)
        x = self.classifier(x)
        return x

# Instantiate the model
model = VGG16()
```

Kiến trúc CNN

Tư ơng tự VGG16 cho Cifar-10



Machine Translated by Google

```
# Define the blocks
block1 = nn.Sequential(
    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),
)
block2 = nn.Sequential(
    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),
)
block3 = nn.Sequential(
    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),
)
block4 = nn.Sequential(
    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),
)
block5 = nn.Sequential(
    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),
)
```

```
# Classifier
classifier = nn.Sequential(
    nn.Flatten(),
    nn.Linear(512*1*1, 256), nn.ReLU(inplace=True),
    nn.Linear(256, 256), nn.ReLU(inplace=True),
    nn.Linear(256, 10),
)

# Combine all blocks into one model
class VGG16(nn.Module):
    def __init__(self):
        super(VGG16, self).__init__()
        self.block1 = block1
        self.block2 = block2
        self.block3 = block3
        self.block4 = block4
        self.block5 = block5
        self.classifier = classifier

    def forward(self, x):
        x = self.block1(x)
        x = self.block2(x)
        x = self.block3(x)
        x = self.block4(x)
        x = self.block5(x)
        x = self.classifier(x)
        return x

# Instantiate the model
model = VGG16()
```

Đề cương

Kiến trúc mạng

Đào

tạo mạng

Nghiên

cứu điển hình

Phương pháp giải quyết vấn đề

Dữ liệu hình ảnh

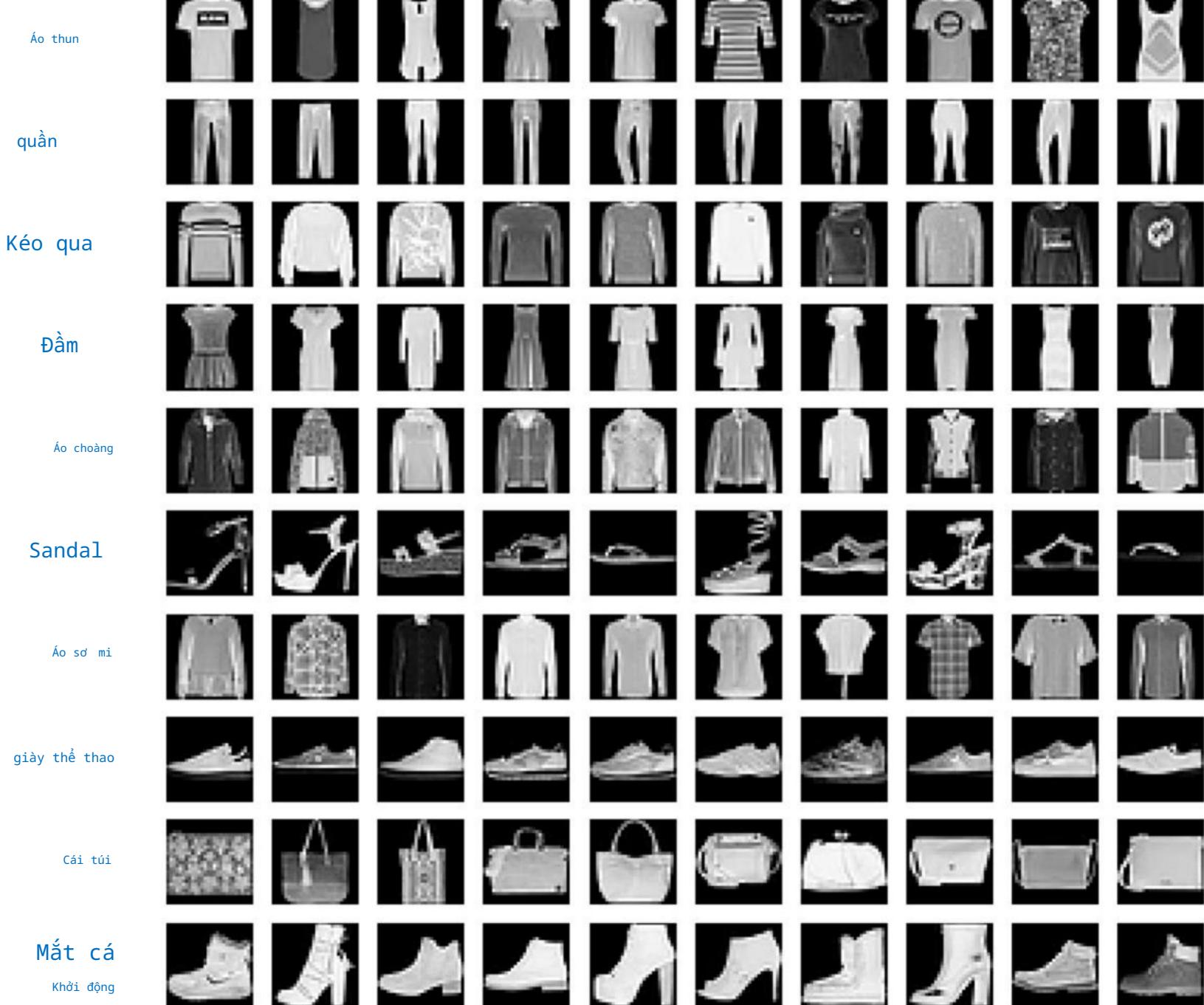
Tập dữ liệu thời trang-MNIST

Hình ảnh thang độ xám

Độ phân giải=28x28

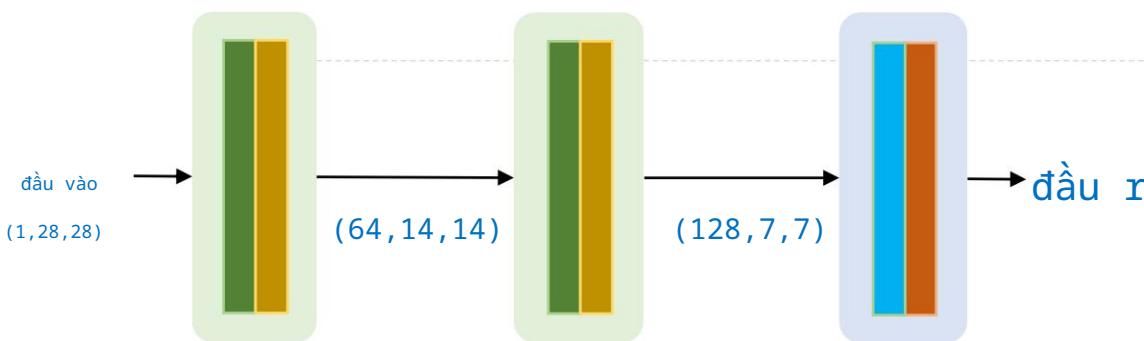
Bộ huấn luyện: 60000 mẫu

Bộ thử nghiệm: 10000 mẫu



Đào tạo mạng

Bộ dữ liệu Fashion-MNIST

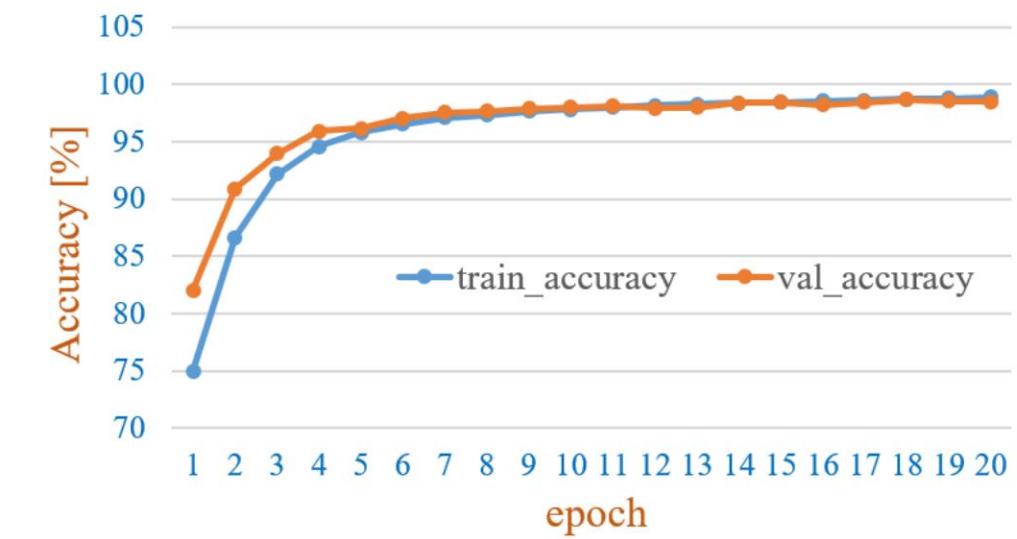
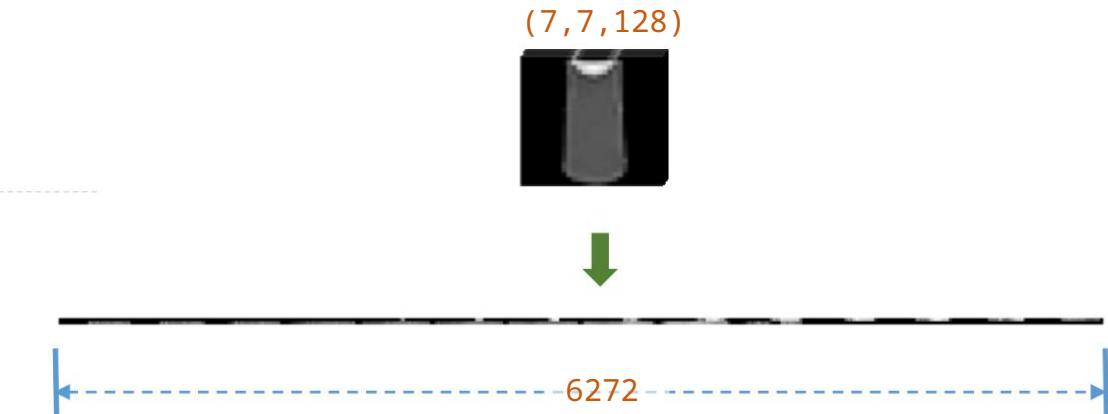


(3x3) Phản đậm tích
chập='same' sải
chân=1 + Sigmoid

(2x2) tổng hợp tối đa

Làm phẳng

Lớp dày đặc-10
+ Softmax



Đào tạo mạng

Bộ dữ liệu Fashion-MNIST

Định dạng dữ liệu X

(lô, kênh, chiều cao, chiều rộng)

Chuẩn hóa dữ liệu [0,1]

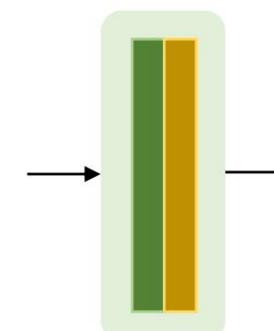
(3x3) Tích chập với 64 bộ lọc, sải
chân=1, đệm='same'

+ Kích hoạt sigmoid +
khởi tạo glorot_uniform

Trình tối ưu hóa Adam và mất entropy chéo

(3x3) Phản đệm tích
chập='same' sải
chân=1 + Sigmoid

(2x2) tổng hợp tối đa



```
# Data
transform = Compose([transforms.ToTensor()])
train_set = FashionMNIST(root='data',
                           train=True,
                           download=True,
                           transform=transform)
trainloader = DataLoader(train_set,
                        batch_size=256,
                        shuffle=True,
                        num_workers=4)
```

```
import torch.nn as nn
import torch.nn.init as init

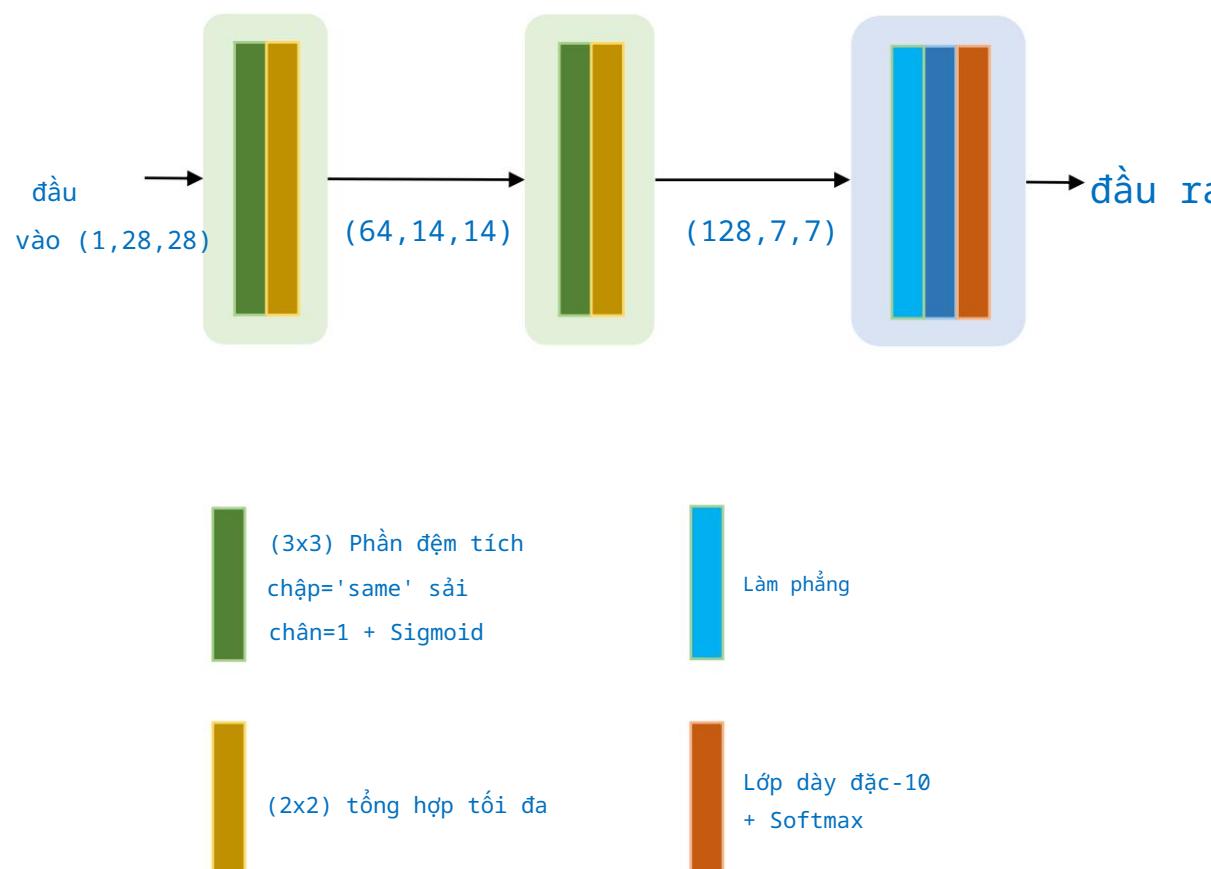
block = nn.Sequential(nn.Conv2d(1, 64, 3,
                               stride=1,
                               padding='same'),
                      nn.Sigmoid(),
                      nn.MaxPool2d(2, 2))

for m in block:
    if isinstance(m, nn.Conv2d):
        init.xavier_uniform_(m.weight)
        if m.bias is not None:
            init.zeros_(m.bias)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = Adam(model.parameters(), lr=1e-3)
```

Đào tạo mạng

Bộ dữ liệu Fashion-MNIST



```
# Declare layers
conv_layer1 = nn.Sequential(
    nn.Conv2d(1, 64, 3, stride=1, padding='same'),
    nn.Sigmoid(),
    nn.MaxPool2d(2, 2)
)
conv_layer2 = nn.Sequential(
    nn.Conv2d(64, 128, 3, stride=1, padding='same'),
    nn.Sigmoid(),
    nn.MaxPool2d(2, 2)
)

flatten = nn.Flatten()
fc_layer1 = nn.Sequential(
    nn.Linear(128*7*7, 512),
    nn.Sigmoid()
)
fc_layer2 = nn.Linear(512, 10)

# Given data x
x = conv_layer1(x)
x = conv_layer2(x)
x = flatten(x)
x = fc_layer1(x)
x = fc_layer2(x)
```

Đào tạo mạng

Tập dữ liệu Cifar-10
(tập dữ liệu phức tạp hơn)

Hình ảnh màu

Độ phân giải=32x32

Bộ huấn luyện: 50000 mẫu

Bộ thử nghiệm: 10000 mẫu

Máy bay



ô tô



chim



con mèo



con nai



chó



con éch



ngựa



tàu thủy

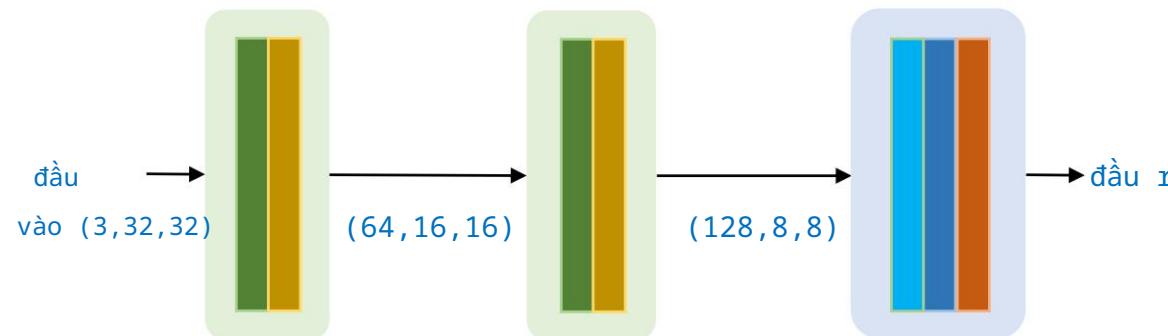


xe tải



Đào tạo mạng

Bộ dữ liệu Cifar-10



(3x3) Phản đệm
tích chập='same'
sải chân=1 + Sigmoid

(2x2) tổng hợp tối đa

Làm phẳng

Lớp dày đặc-10
+ Softmax

Chuẩn hóa dữ liệu [0,1]

Khởi tạo đồng phục Glorot

Trình tối ưu hóa Adam với lr=1e-3

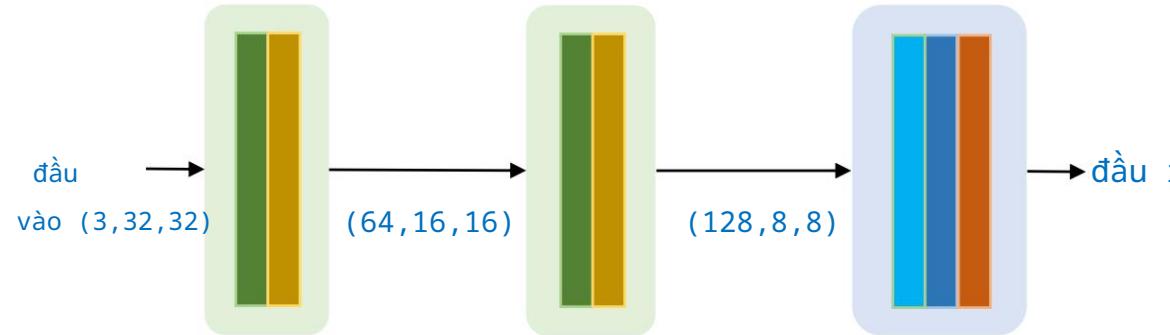
```
conv_layer1 = nn.Sequential(
    nn.Conv2d(1, 64, 3, stride=1, padding='same'),
    nn.Sigmoid(),
    nn.MaxPool2d(2, 2)
)
conv_layer2 = nn.Sequential(
    nn.Conv2d(64, 128, 3, stride=1, padding='same'),
    nn.Sigmoid(),
    nn.MaxPool2d(2, 2)
)

flatten = nn.Flatten()
fc_layer1 = nn.Sequential(
    nn.Linear(128*8*8, 512),
    nn.Sigmoid()
)
fc_layer2 = nn.Linear(512, 10)

# Given data x
x = conv_layer1(x)
x = conv_layer2(x)
x = flatten(x)
x = fc_layer1(x)
x = fc_layer2(x)
```

Đào tạo mạng

Bộ dữ liệu Cifar-10



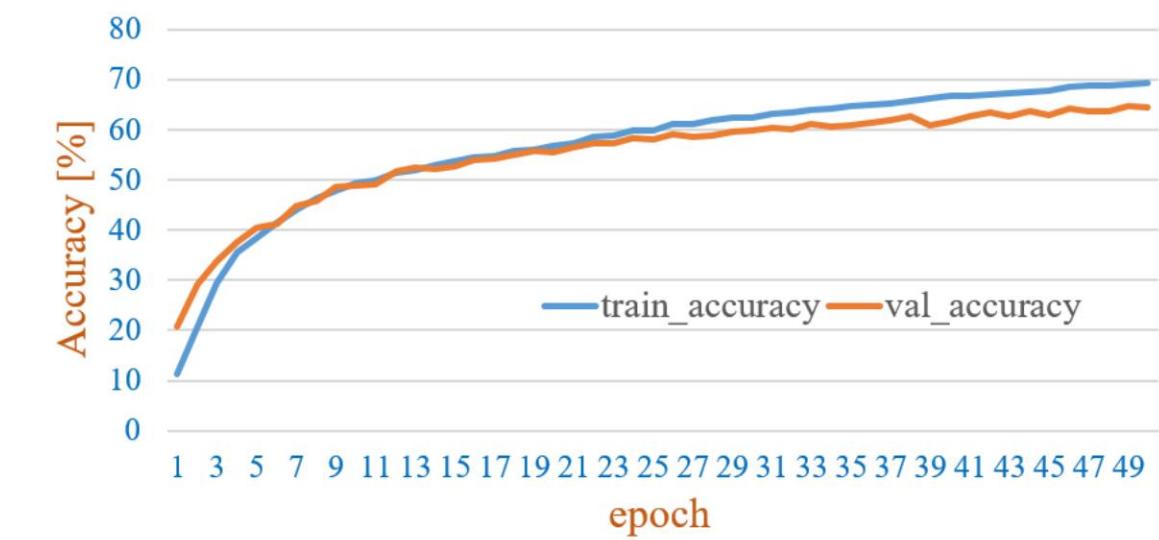
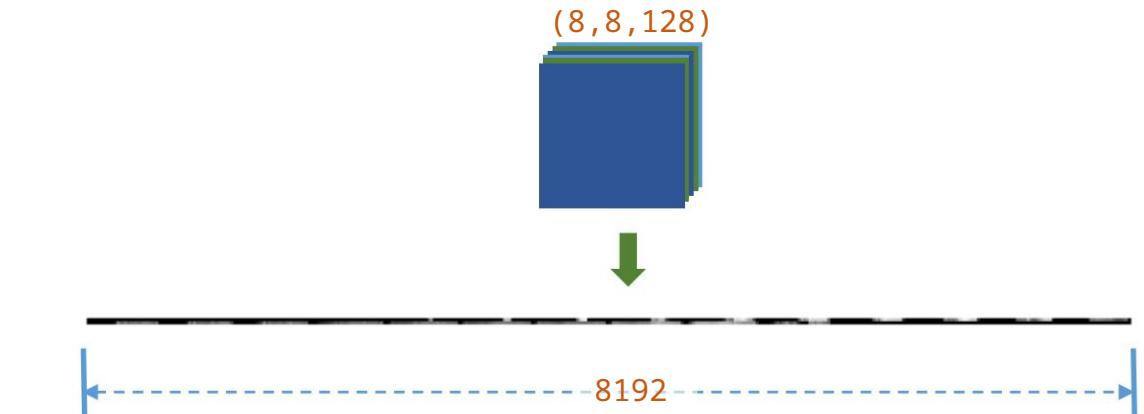
(3x3) Phần đậm tích
chập='same' sải
chân=1 + Sigmoid

(2x2) tổng hợp tối đa

Làm phẳng

Lớp dày đặc-10
+ Softmax

Độ chính xác: 69,3% - Val_accuracy: 64,5%



Đào tạo mạng

Bộ dữ liệu Cifar-10:

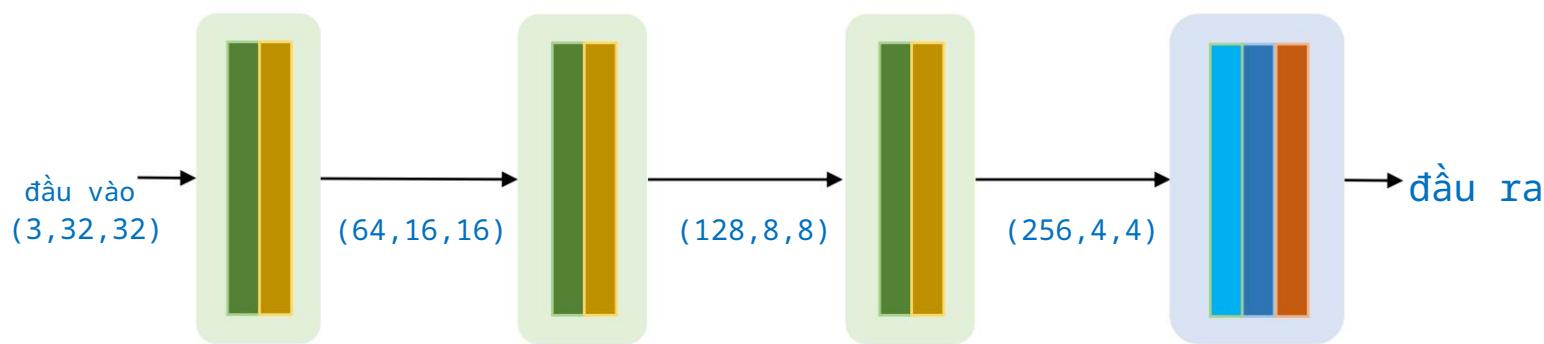
Thêm nhiều lớp hơn n



Chuẩn hóa dữ liệu [0,1]

Khởi tạo đồng phục Glorot

Trình tối ưu hóa Adam với lr=1e-3



```

conv_layer1 = nn.Sequential(
    nn.Conv2d(3, 64, 3, stride=1, padding='same'),
    nn.Sigmoid(),
    nn.MaxPool2d(2, 2)
)
conv_layer2 = nn.Sequential(
    nn.Conv2d(64, 128, 3, stride=1, padding='same'),
    nn.Sigmoid(),
    nn.MaxPool2d(2, 2)
)
conv_layer3 = nn.Sequential(
    nn.Conv2d(128, 256, 3, stride=1, padding='same'),
    nn.Sigmoid(),
    nn.MaxPool2d(2, 2)
)

flatten = nn.Flatten()
fc_layer1 = nn.Sequential(
    nn.Linear(256*4*4, 512),
    nn.Sigmoid()
)
fc_layer2 = nn.Linear(512, n_classes)

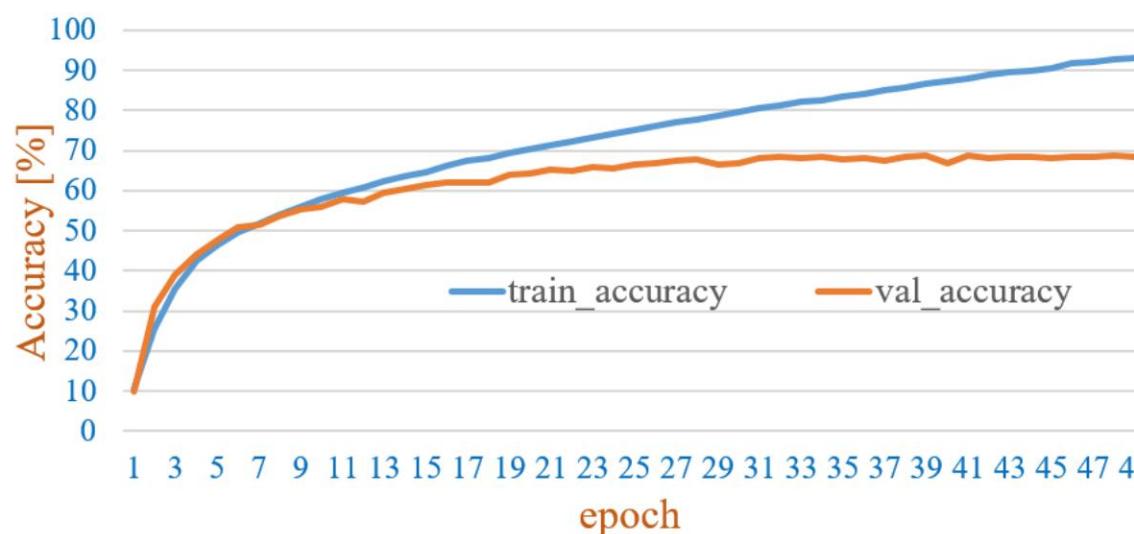
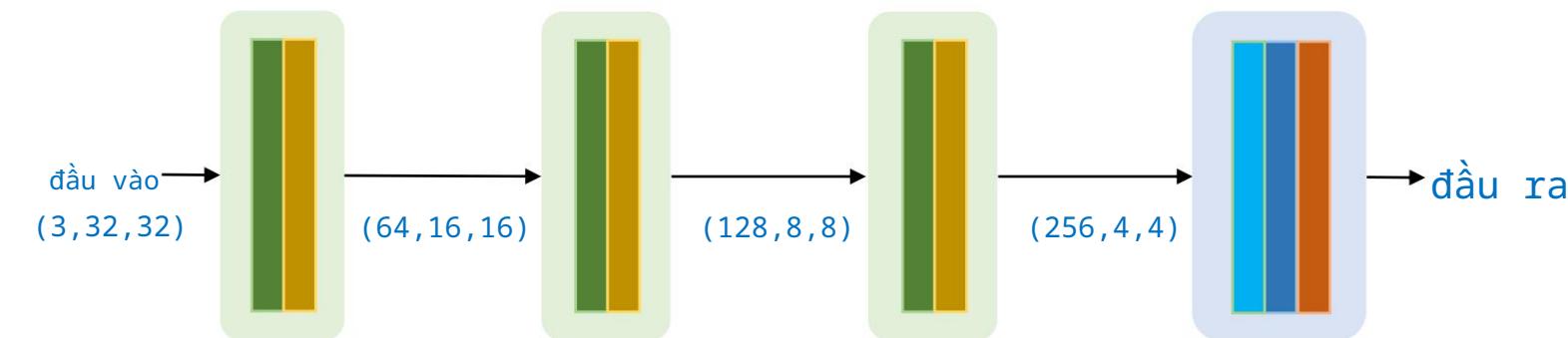
```

Đào tạo mạng

Bộ dữ liệu Cifar-10:

Thêm nhiều lớp hơn

Tin tốt: Độ chính xác của mạng tăng
khoảng 25%

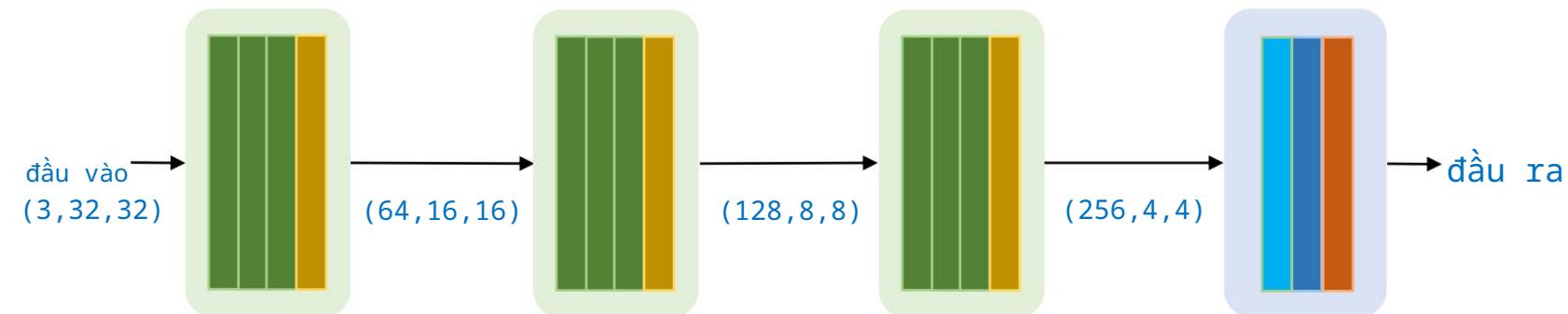


Độ chính xác: 93,8% - Val_accuracy: 68,7%

Mạng

Đào tạo

Tập dữ liệu Cifar-10:



Tiếp tục bổ sung thêm nhiều lớp hơn n

(3x3) Tích chập
đệm='giống nhau'
sài bù $\sigma_c=1$ + Sigmoid



Làm phẳng

(2x2) tổng hợp tối đa



Lớp dày đặc-10
+ Softmax

Chuẩn hóa dữ liệu [0,1]

Khởi tạo đồng phục Glorot

Trình tối ưu hóa Adam với lr=1e-3

```

conv_layer1 = nn.Sequential(nn.Conv2d(3, 64, 3, stride=1, padding='same'), nn.Sigmoid())
conv_layer2 = nn.Sequential(nn.Conv2d(64, 64, 3, stride=1, padding='same'), nn.Sigmoid())
conv_layer3 = nn.Sequential(nn.Conv2d(64, 64, 3, stride=1, padding='same'), nn.Sigmoid(),
                           nn.MaxPool2d(2, 2))

conv_layer4 = nn.Sequential(nn.Conv2d(64, 128, 3, stride=1, padding='same'), nn.Sigmoid())
conv_layer5 = nn.Sequential(nn.Conv2d(128, 128, 3, stride=1, padding='same'), nn.Sigmoid())
conv_layer6 = nn.Sequential(nn.Conv2d(128, 128, 3, stride=1, padding='same'), nn.Sigmoid(),
                           nn.MaxPool2d(2, 2))

conv_layer7 = nn.Sequential(nn.Conv2d(128, 256, 3, stride=1, padding='same'), nn.Sigmoid())
conv_layer8 = nn.Sequential(nn.Conv2d(256, 256, 3, stride=1, padding='same'), nn.Sigmoid())
conv_layer9 = nn.Sequential(nn.Conv2d(256, 256, 3, stride=1, padding='same'), nn.Sigmoid(),
                           nn.MaxPool2d(2, 2))

flatten = nn.Flatten()

fc_layer1 = nn.Sequential(nn.Linear(256*4*4, 512), nn.Sigmoid())
fc_layer2 = nn.Linear(512, 10)

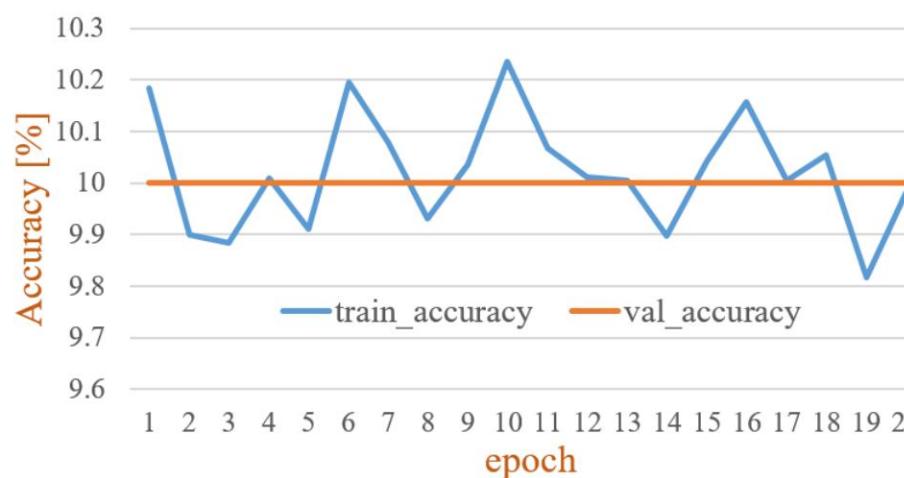
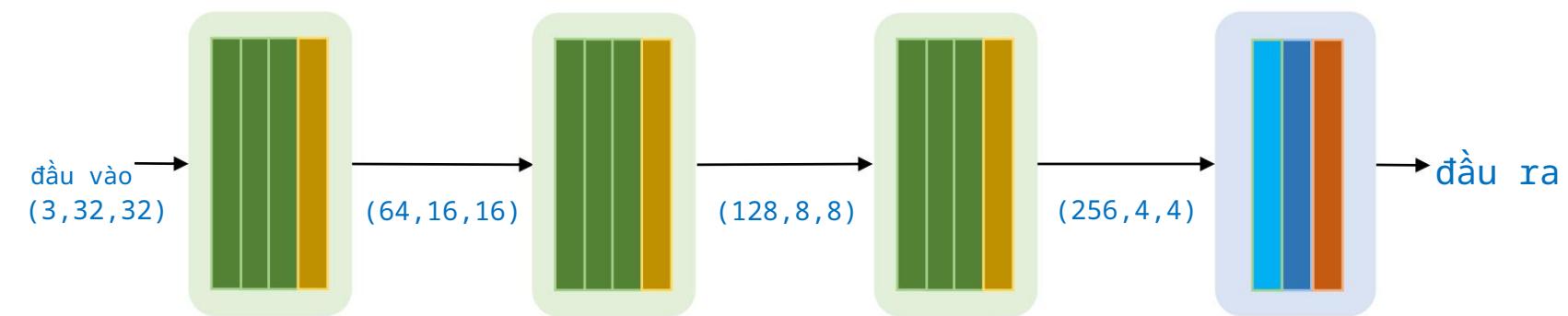
```

Đào tạo mạng

Bộ dữ liệu Cifar-10:

Tiếp tục bổ sung thêm nhiều lớp

Mạng không học



(3x3) Tích chập
đệm='giống nhau'
sải bù ớc=1 + Sigmoid

(2x2) tổng hợp tối đa

Làm phẳng

Lớp dày đặc-10
+ Softmax

Lớp dày đặc-512
+ Sigmoid

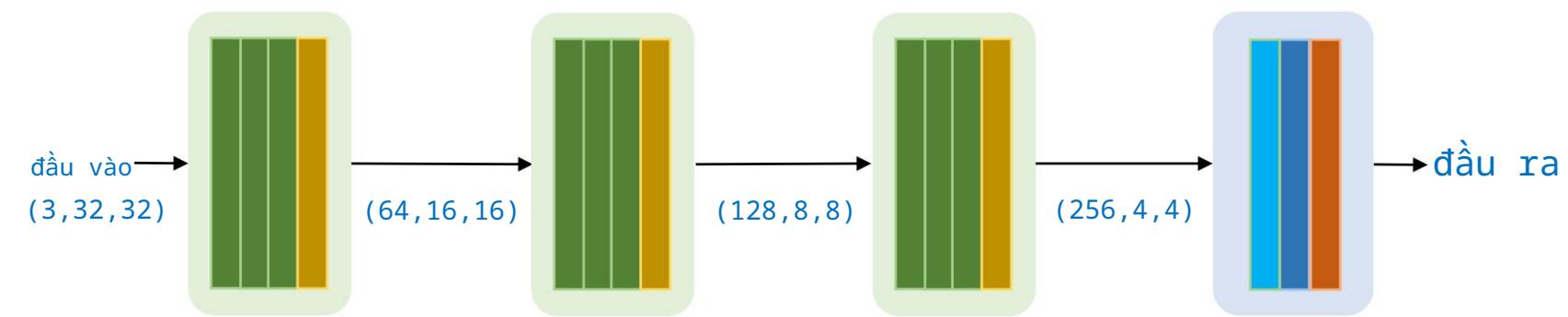
Đào tạo mạng

Bộ dữ liệu Cifar-10:

Tiếp tục bổ sung thêm nhiều lớp

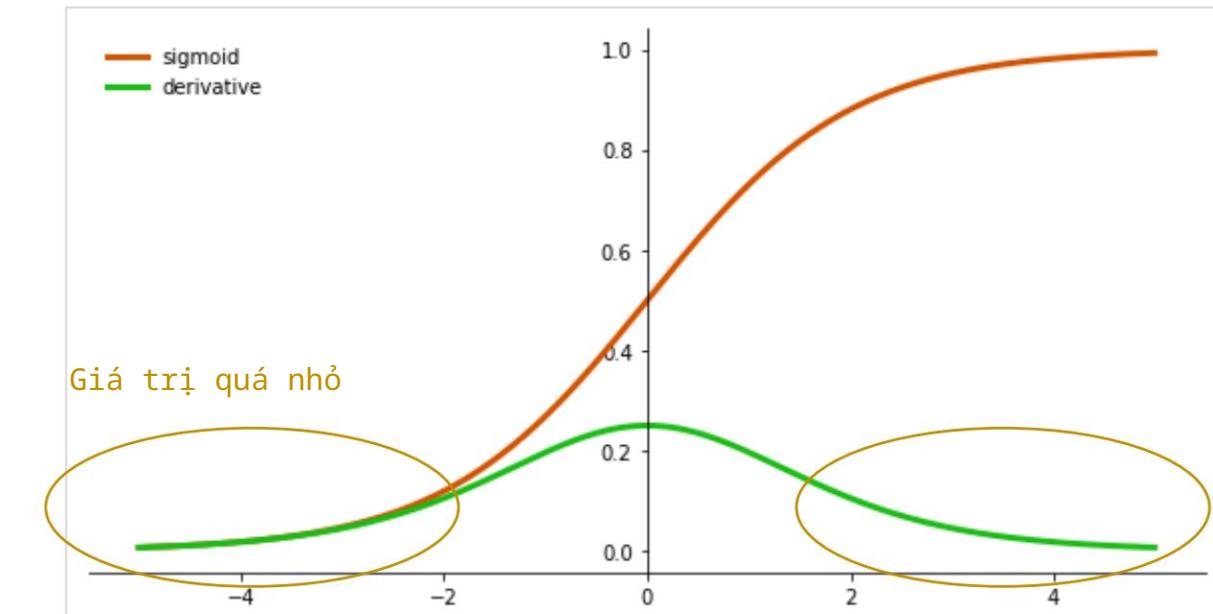
(3x3) Tích chập
đệm='giống nhau'
sài bù orts=1 + Sigmoid

Lớp dày đặc-512
+ Sigmoid



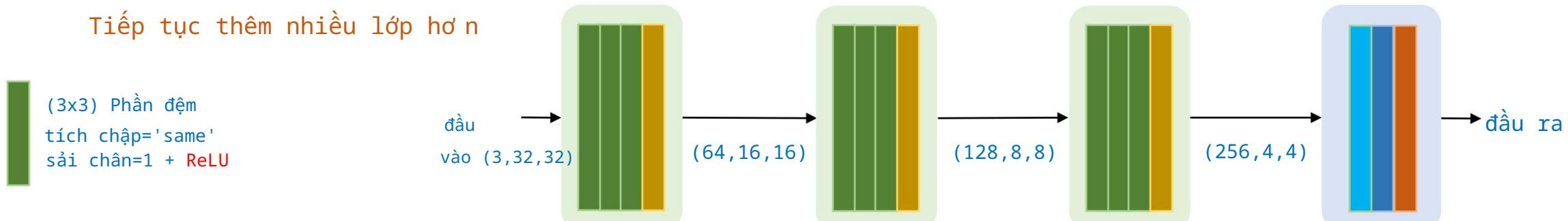
$$\text{sigmoid} \Rightarrow \frac{1}{1 +}$$

Vấn đề biến mất



Đào tạo mạng

Bộ dữ liệu Cifar-10:

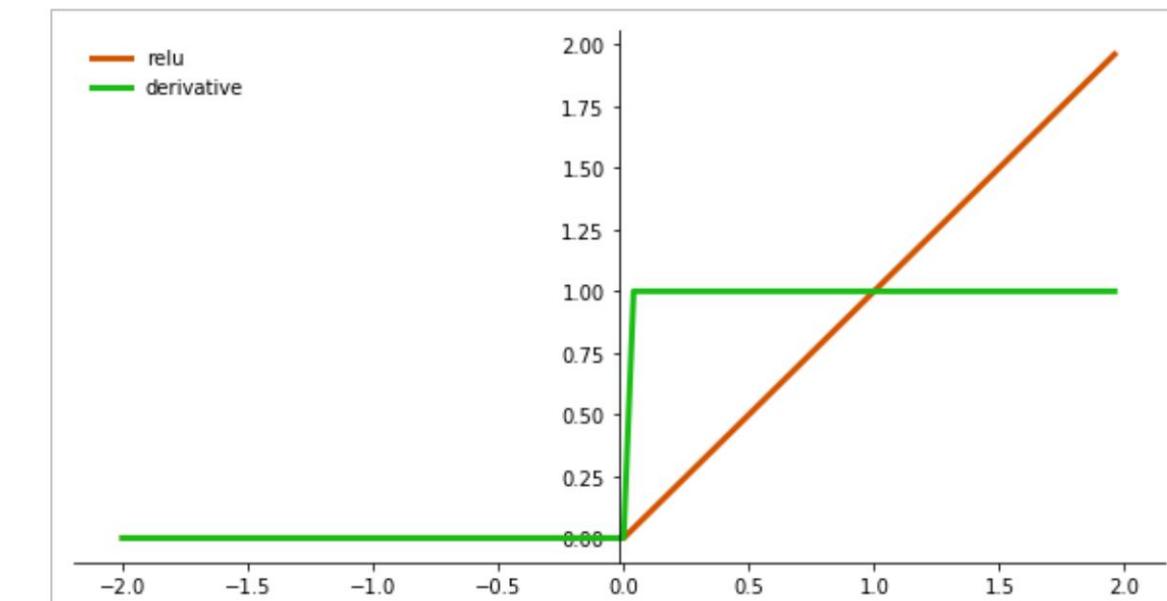


Lớp dày đặc-512
+ ReLU

$$\text{ReLU}(\theta) = \begin{cases} \theta & \text{if } \theta \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

nn.Conv2D(...), nn.Sigmoid()

nn.Conv2D(...), nn.ReLU()

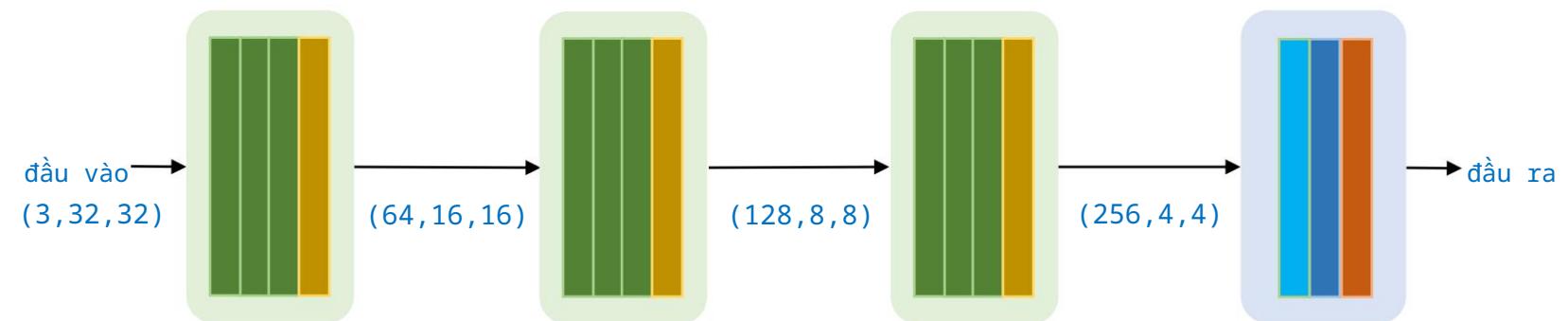


Mạng

Đào tạo

Bộ dữ liệu Cifar-10:

Sử dụng ReLU



```

conv_layer1 = nn.Sequential(nn.Conv2d(3, 64, 3, stride=1, padding='same'), nn.ReLU())
conv_layer2 = nn.Sequential(nn.Conv2d(64, 64, 3, stride=1, padding='same'), nn.ReLU())
conv_layer3 = nn.Sequential(nn.Conv2d(64, 64, 3, stride=1, padding='same'), nn.ReLU(),
                           nn.MaxPool2d(2, 2))

conv_layer4 = nn.Sequential(nn.Conv2d(64, 128, 3, stride=1, padding='same'), nn.ReLU())
conv_layer5 = nn.Sequential(nn.Conv2d(128, 128, 3, stride=1, padding='same'), nn.ReLU())
conv_layer6 = nn.Sequential(nn.Conv2d(128, 128, 3, stride=1, padding='same'), nn.ReLU(),
                           nn.MaxPool2d(2, 2))

conv_layer7 = nn.Sequential(nn.Conv2d(128, 256, 3, stride=1, padding='same'), nn.ReLU())
conv_layer8 = nn.Sequential(nn.Conv2d(256, 256, 3, stride=1, padding='same'), nn.ReLU())
conv_layer9 = nn.Sequential(nn.Conv2d(256, 256, 3, stride=1, padding='same'), nn.ReLU(),
                           nn.MaxPool2d(2, 2))

flatten = nn.Flatten()

fc_layer1 = nn.Sequential(nn.Linear(256*4*4, 512), nn.ReLU())
fc_layer2 = nn.Linear(512, 10)

```

(3x3) Tích chập
đệm='giống nhau'
sải bù ớc=1 + ReLU

Làm phẳng

(2x2) tổng hợp tối đa

Lớp dày đặc-10
+ Softmax

Lớp dày đặc-512
+ ReLU

Chuẩn hóa dữ liệu [0,1]

Khởi tạo đồng phục Glorot

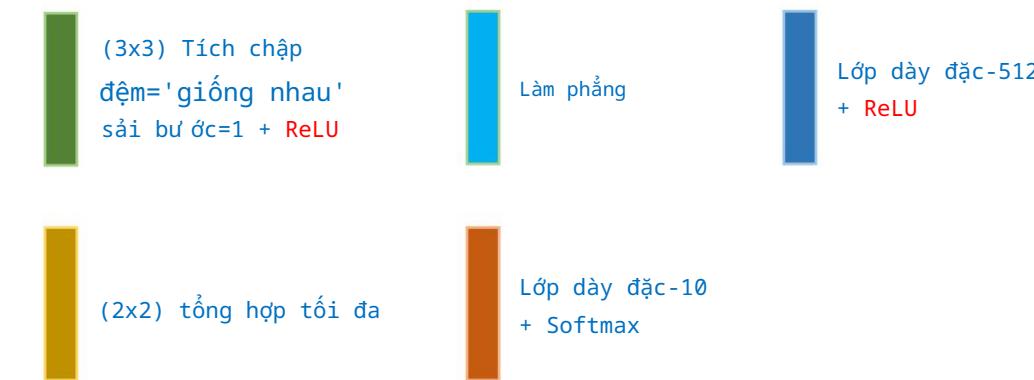
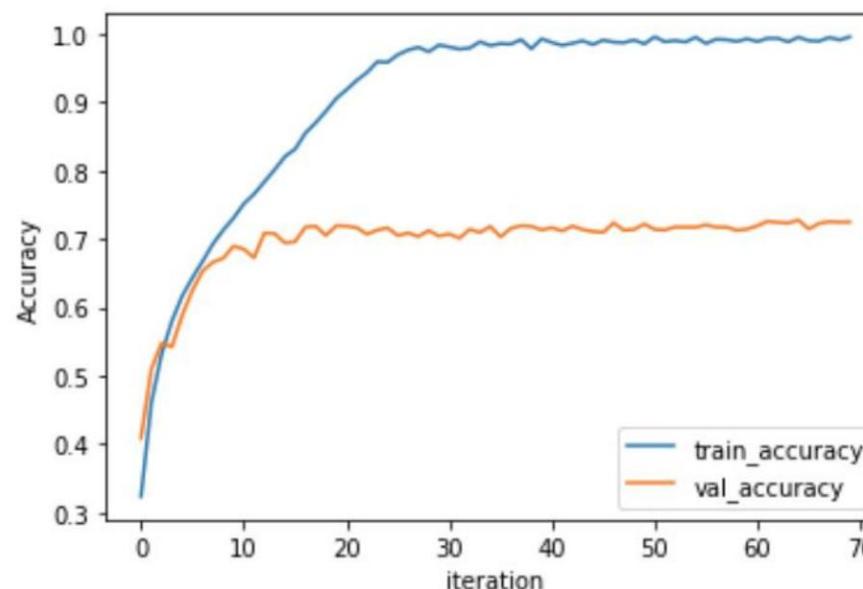
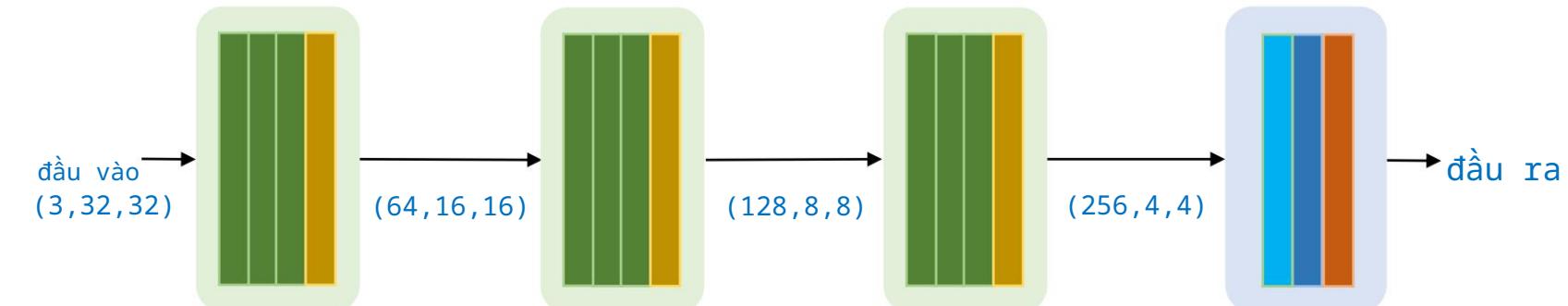
Trình tối ưu hóa Adam với lr=1e-3

Đào tạo mạng

Bộ dữ liệu Cifar-10:

Sử dụng ReLU

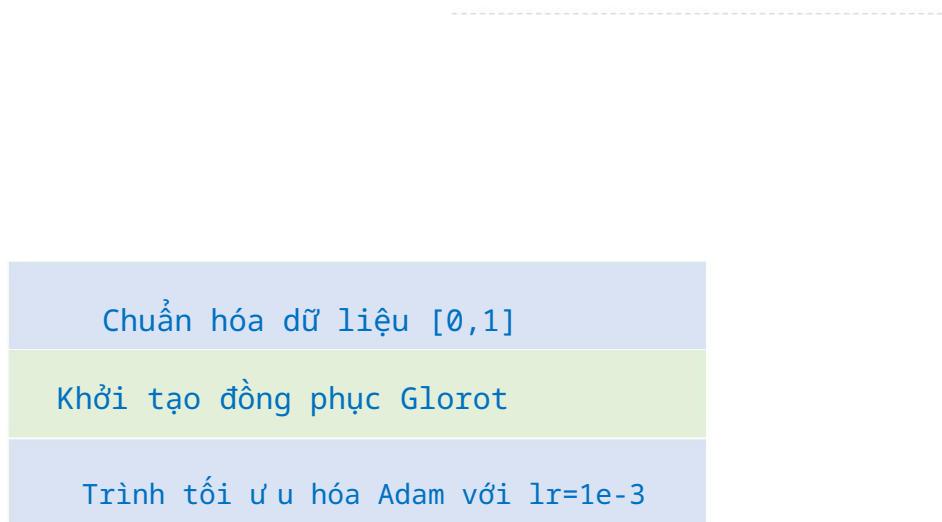
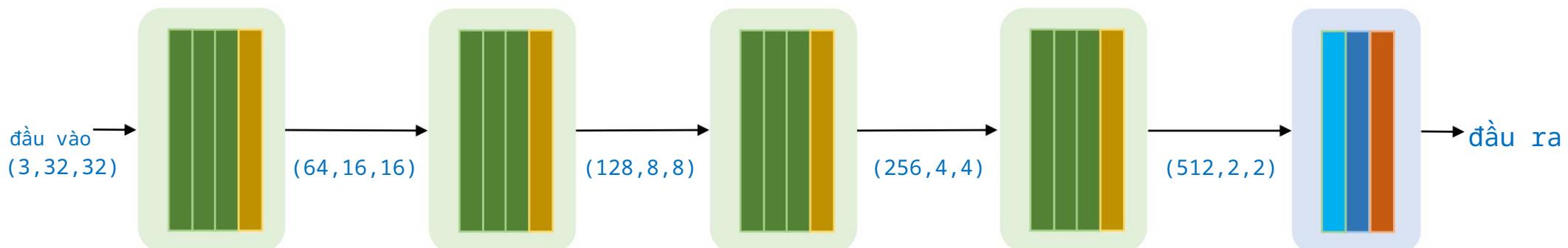
Độ chính xác đào tạo
đạt tới 99%



Thêm nhiều lớp hơn; Hi vọng đạt 100%

Đào tạo mạng

Sử dụng ReLU và thêm nhiều lớp hơn



(3x3) Tích chập
đệm='giống nhau'
sải bù 0c=1 + ReLU

(2x2) tổng hợp tối đa

Làm phẳng

Lớp dày đặc-10
+ Softmax

Lớp dày đặc-512
+ ReLU

Thực hiện

```

conv_layer1 = nn.Sequential(nn.Conv2d(3, 64, 3, stride=1, padding='same'), nn.ReLU())
conv_layer2 = nn.Sequential(nn.Conv2d(64, 64, 3, stride=1, padding='same'), nn.ReLU())
conv_layer3 = nn.Sequential(nn.Conv2d(64, 64, 3, stride=1, padding='same'), nn.ReLU(),
                           nn.MaxPool2d(2, 2))

conv_layer4 = nn.Sequential(nn.Conv2d(64, 128, 3, stride=1, padding='same'), nn.ReLU())
conv_layer5 = nn.Sequential(nn.Conv2d(128, 128, 3, stride=1, padding='same'), nn.ReLU(),)
conv_layer6 = nn.Sequential(nn.Conv2d(128, 128, 3, stride=1, padding='same'), nn.ReLU(),
                           nn.MaxPool2d(2, 2))

conv_layer7 = nn.Sequential(nn.Conv2d(128, 256, 3, stride=1, padding='same'), nn.ReLU())
conv_layer8 = nn.Sequential(nn.Conv2d(256, 256, 3, stride=1, padding='same'), nn.ReLU())
conv_layer9 = nn.Sequential(nn.Conv2d(256, 256, 3, stride=1, padding='same'), nn.ReLU(),
                           nn.MaxPool2d(2, 2))

conv_layer10 = nn.Sequential(nn.Conv2d(256, 512, 3, stride=1, padding='same'), nn.ReLU())
conv_layer11 = nn.Sequential(nn.Conv2d(512, 512, 3, stride=1, padding='same'), nn.ReLU())
conv_layer12 = nn.Sequential(nn.Conv2d(512, 512, 3, stride=1, padding='same'), nn.ReLU(),
                           nn.MaxPool2d(2, 2))

flatten = nn.Flatten()

fc_layer1 = nn.Sequential(nn.Linear(512 * 2 * 2, 512), nn.ReLU())
fc_layer2 = nn.Linear(512, 10)

```

```

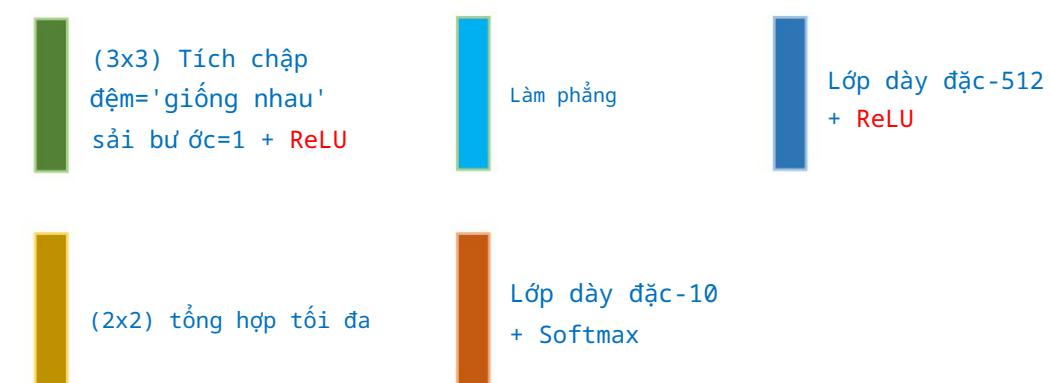
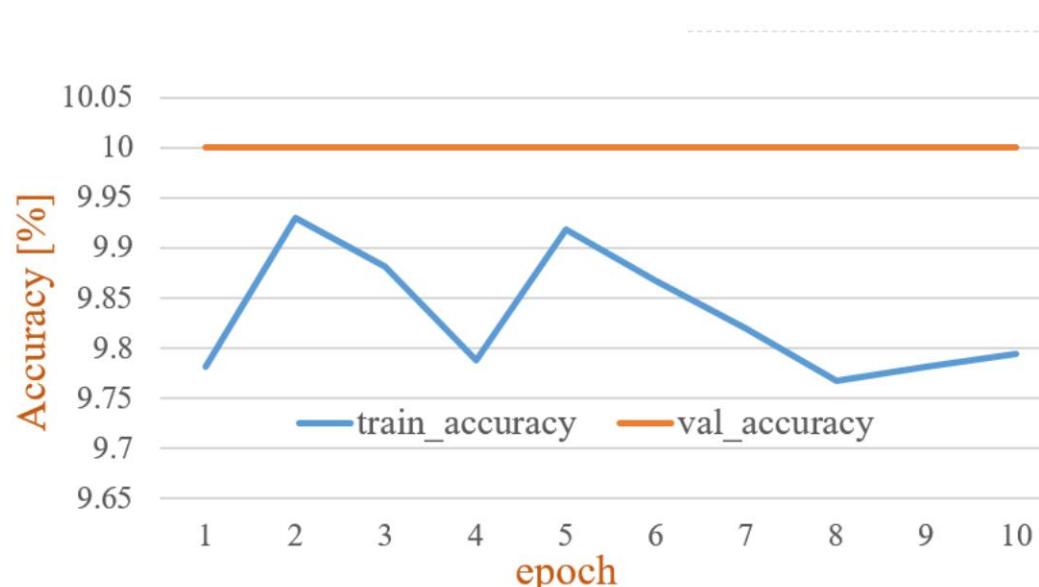
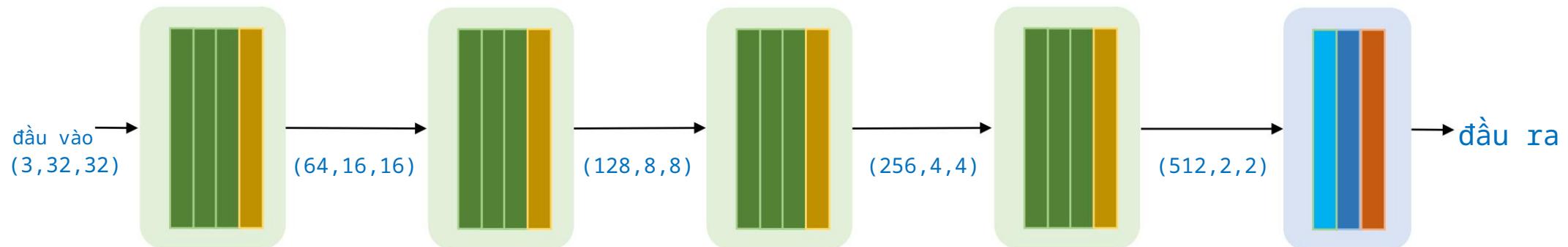
def initialize_weights(self):
    for m in self.modules():
        if isinstance(m, nn.Conv2d):
            init.xavier_uniform_(m.weight)
            if m.bias is not None:
                init.zeros_(m.bias)
        elif isinstance(m, nn.Linear):
            init.xavier_uniform_(m.weight)
            if m.bias is not None:
                init.zeros_(m.bias)

def forward(self, x):
    x = self.conv_layer1(x)
    x = self.conv_layer2(x)
    x = self.conv_layer3(x)
    x = self.conv_layer4(x)
    x = self.conv_layer5(x)
    x = self.conv_layer6(x)
    x = self.conv_layer7(x)
    x = self.conv_layer8(x)
    x = self.conv_layer9(x)
    x = self.conv_layer10(x)
    x = self.conv_layer11(x)
    x = self.conv_layer12(x)
    x = self.flatten(x)
    x = self.fc_layer1(x)
    out = self.fc_layer2(x)
    return out

```

Đào tạo mạng

Sử dụng ReLU và thêm nhiều lớp hơn n



Đào tạo mạng

Tóm tắt mạng lư ới hiện tại

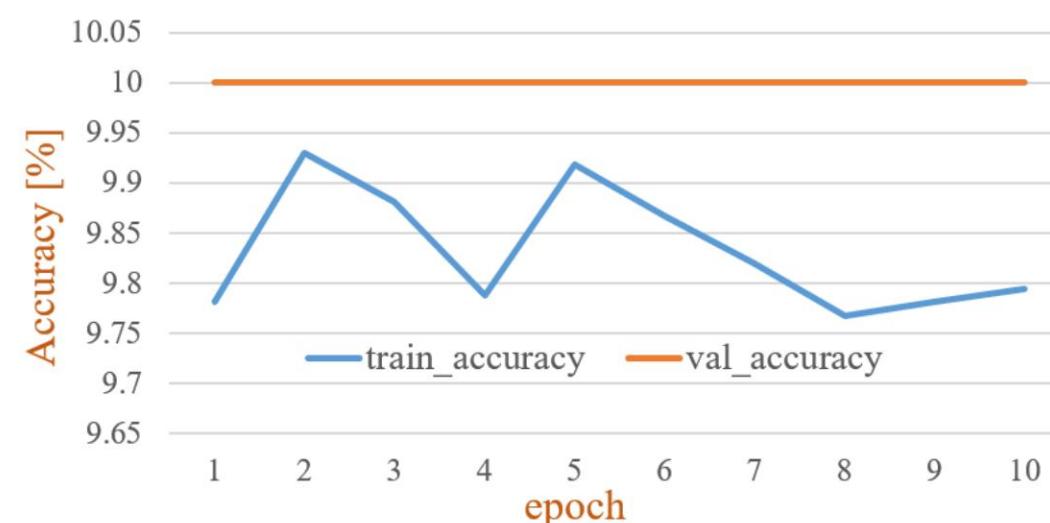


Chuẩn hóa dữ liệu (tỷ lệ thành $[0,1]$)

Xây dựng mạng lư ới
(Chuyển đổi, ReLU,
gộp tối đa, Lớp dày đặc)

Tham số
Khởi tạo
(Đồng phục Glorot)

Mạng không học

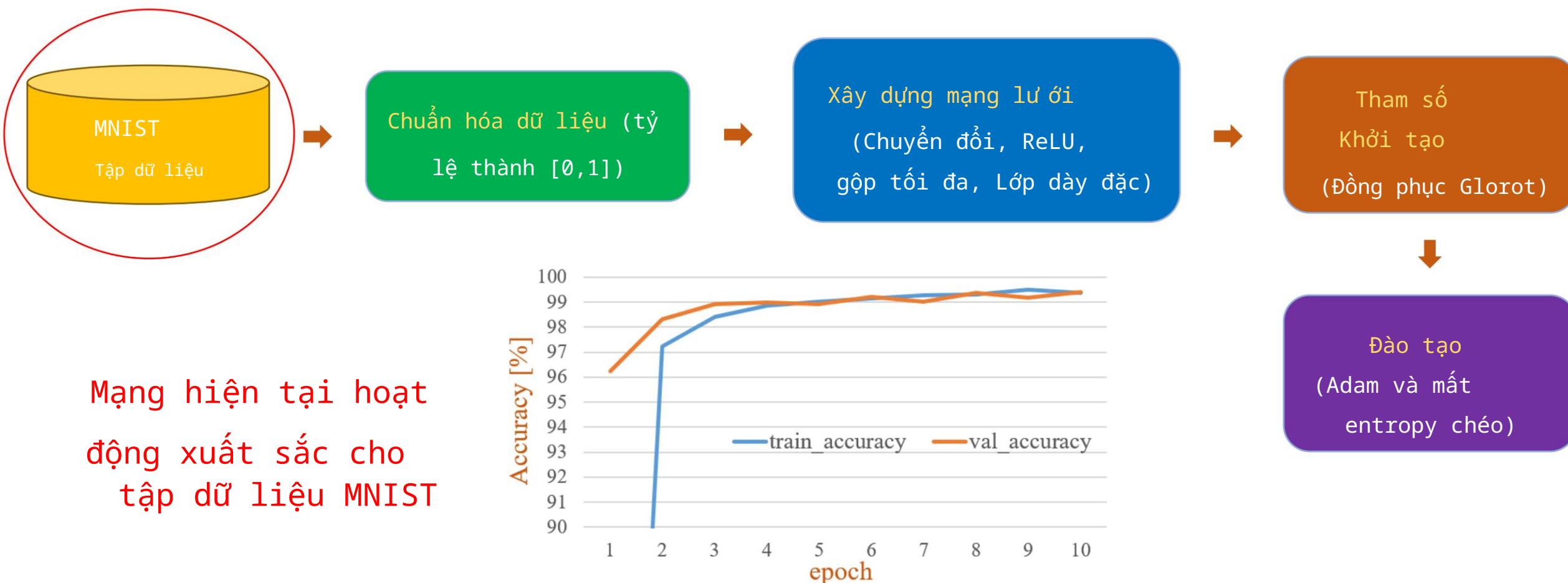


Đào tạo
(Adam và mất
entropy chéo)



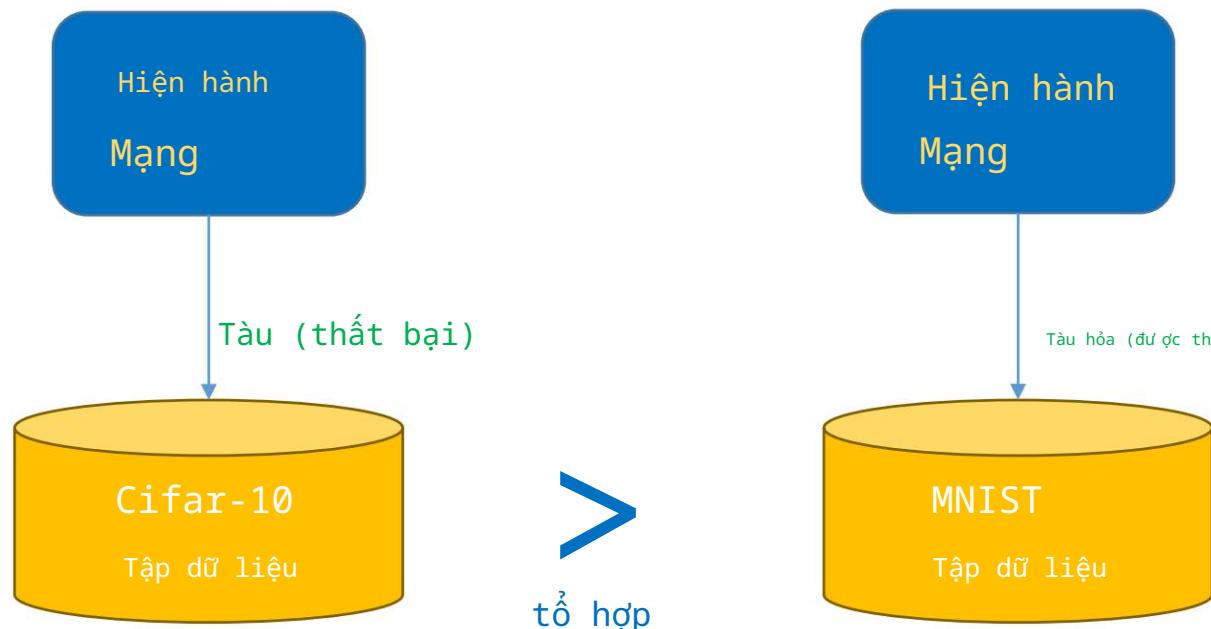
Đào tạo mạng

Giải pháp 1: Quan sát



Đào tạo mạng

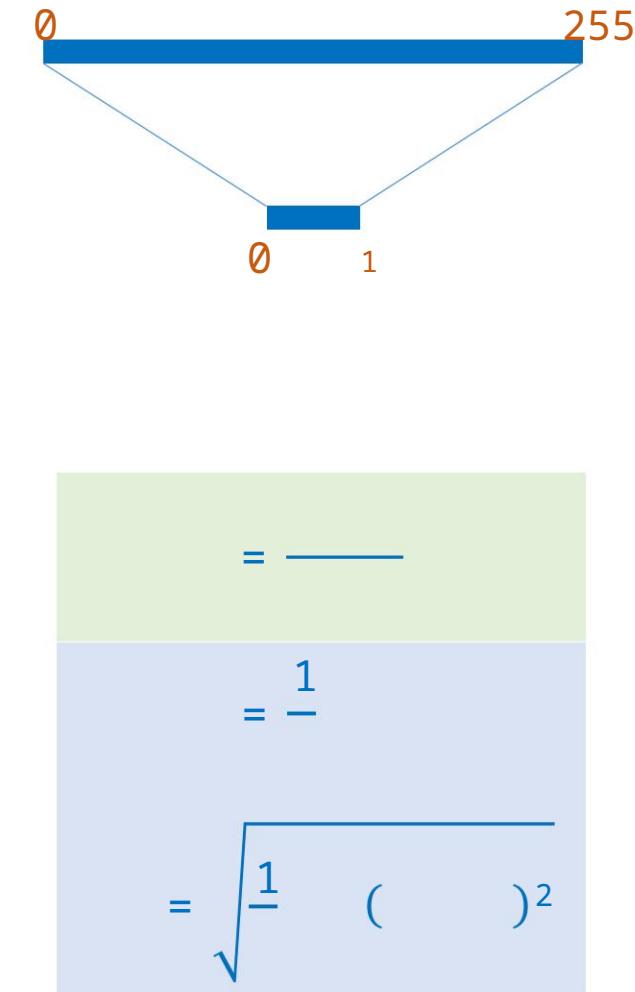
Giải pháp 1: Ý tư ởng



Cách giảm độ phức tạp của bộ dữ liệu Cifar-10

Chuẩn hóa dữ liệu
(tỷ lệ thành $[0,1]$)

Chuẩn hóa dữ liệu
(chuyển đổi thành giá trị
trung bình 0 và độ lệch 1)



Đào tạo mạng

Giải pháp 1: Ý tư ởng

$$\begin{aligned} &= \underline{\quad} \\ &= \frac{1}{\underline{\quad}} \\ &= \sqrt{\frac{1}{\underline{\quad}} (\quad)^2} \end{aligned}$$

Việc chuẩn hóa này giúp
mạng bắt biến với phép
biến đổi tuyến tính

$$\begin{aligned} &= + \\ &= \underline{\quad} = - \end{aligned}$$



$$\begin{aligned} &= + \\ &= \underline{\quad} = \frac{(\sigma + \frac{1}{P}) (+)}{\sqrt{\frac{1}{P} \left((\sigma + \frac{1}{P}) (+) \right)^2}} \\ &= \frac{\frac{1}{P}}{\sqrt{\frac{1}{P} \left(\frac{1}{P} \right)^2}} \\ &= \frac{\frac{1}{P}}{\sqrt{\frac{1}{P} \left(\frac{1}{P} \right)^2}} = \frac{\sqrt{\frac{1}{P} (\quad)^2}}{\underline{\quad}} = \end{aligned}$$

Đào tạo mạng

Giải pháp 1: Chuẩn hóa giá trị trung bình 0 và độ lệch đơn vị

Chuẩn hóa dữ liệu

(chuyển đổi thành giá trị
trung bình 0 và độ lệch 1)

= _____

là giá trị trung bình của tập dữ liệu

là độ lệch cho toàn bộ tập dữ liệu

```
# Load dataset with only the ToTensor transform
compute_transform = transforms.Compose([transforms.ToTensor()])
dataset = torchvision.datasets.CIFAR10(root='data', train=True,
                                         transform=compute_transform,
                                         download=True)
loader = torch.utils.data.DataLoader(dataset, batch_size=1024,
                                         shuffle=False, num_workers=4)

mean = 0.0
for images, _ in loader:
    batch_samples = images.size(0) # Batch size
    images = images.view(batch_samples, images.size(1), -1)
    mean += images.mean(2).sum(0)
mean = mean / len(loader.dataset)

variance = 0.0
for images, _ in loader:
    batch_samples = images.size(0)
    images = images.view(batch_samples, images.size(1), -1)
    variance += ((images - mean.unsqueeze(1))**2).sum([0,2])
std = torch.sqrt(variance / (len(loader.dataset)*32*32))

# Data
transform = Compose([ToTensor(),
                     Normalize(mean, std)])
train_set = CIFAR10(root='data', train=True,
                     download=True, transform=transform)
trainloader = DataLoader(train_set, batch_size=256,
                        shuffle=True, num_workers=4)
```

Đào tạo mạng

Giải pháp 1: Chuẩn hóa trung bình 0 và độ lệch đơn vị

Chuẩn hóa dữ liệu (chuyển đổi thành giá trị trung bình 0 và độ lệch 1)

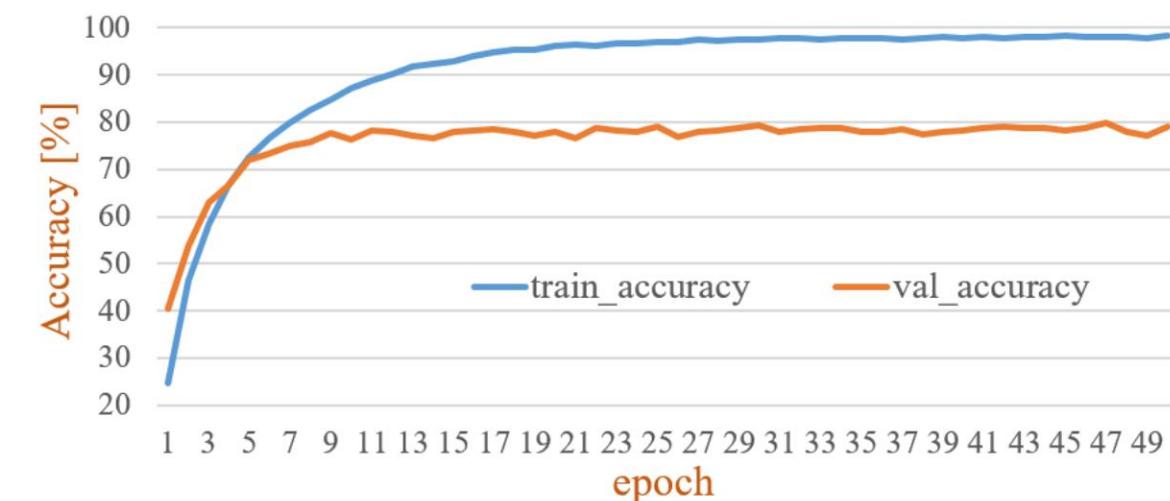
Chuẩn hóa từng kênh riêng biệt

```
transform = Compose([ToTensor(),
                     Normalize([0.4914, 0.4822, 0.4465],
                               [0.2470, 0.2435, 0.2616])])
train_set = CIFAR10(root='data', train=True,
                     download=True, transform=transform)
```

= _____

là giá trị trung bình của tập dữ liệu

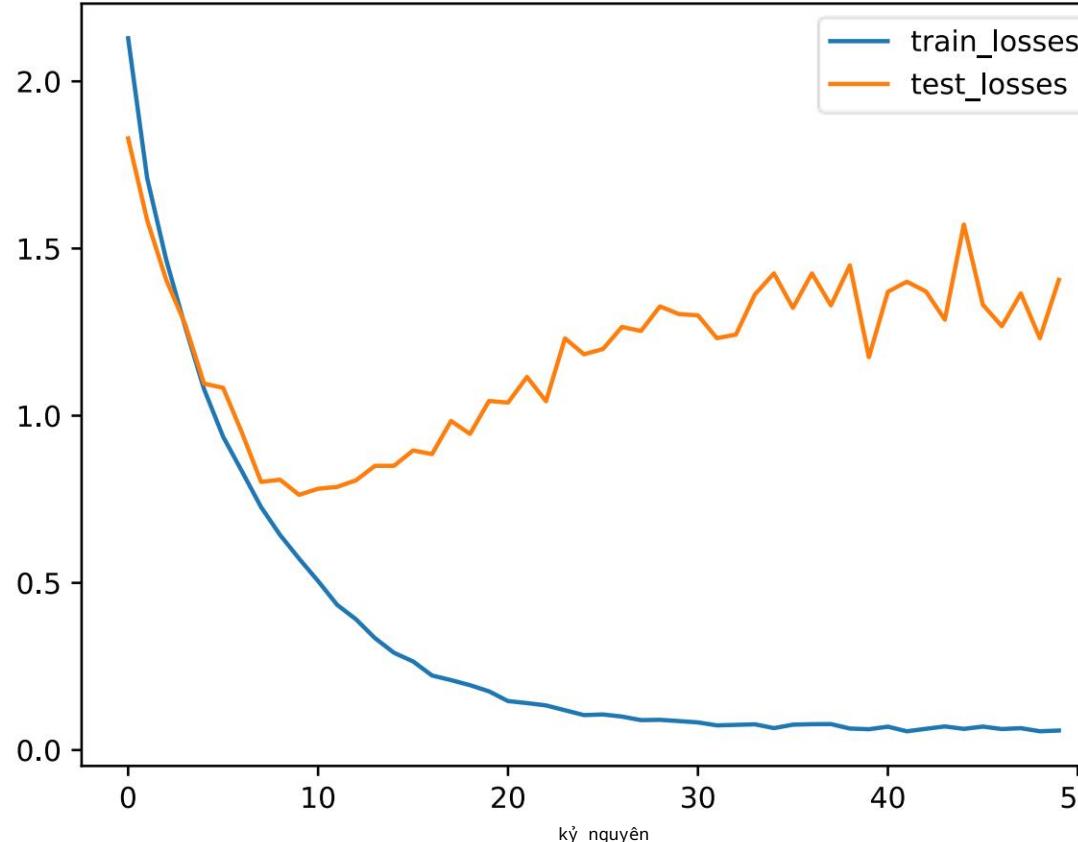
là độ lệch cho toàn bộ tập dữ liệu



Đào tạo mạng

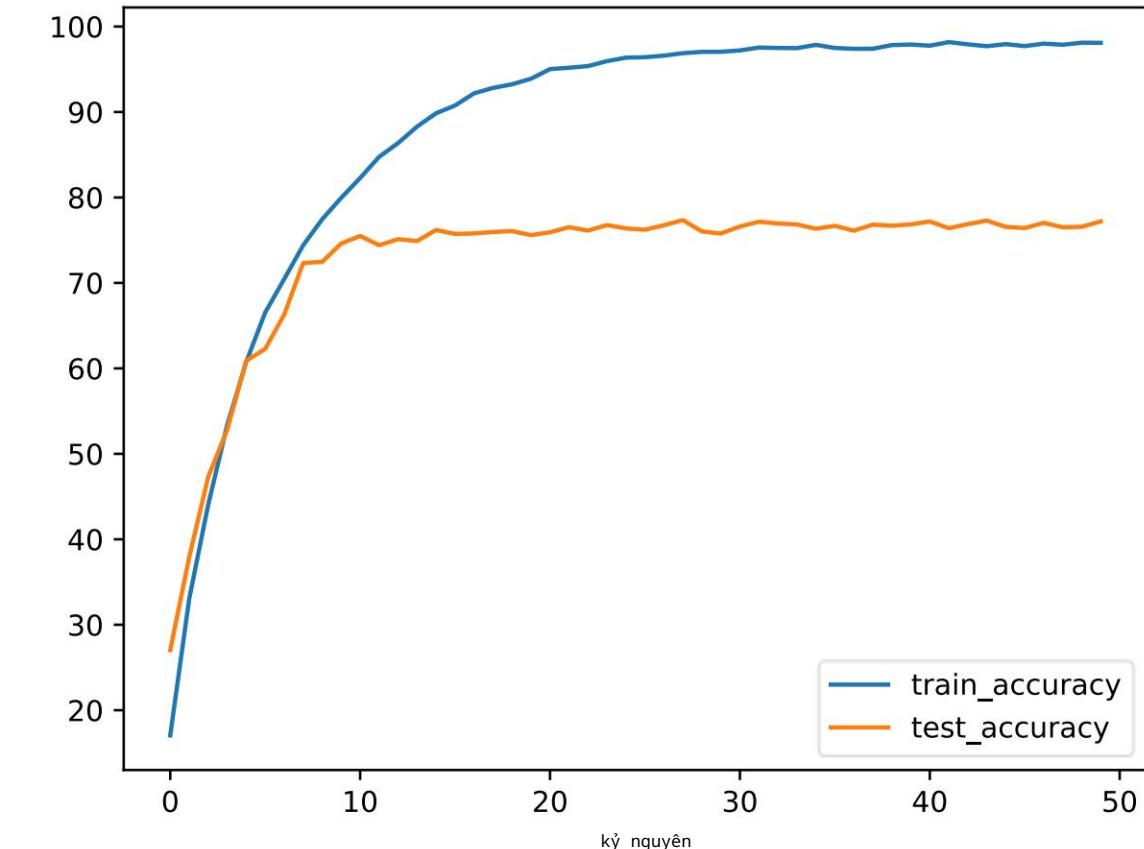
Giải pháp 1 (mở rộng):

Chuẩn hóa thành [-1, 1]



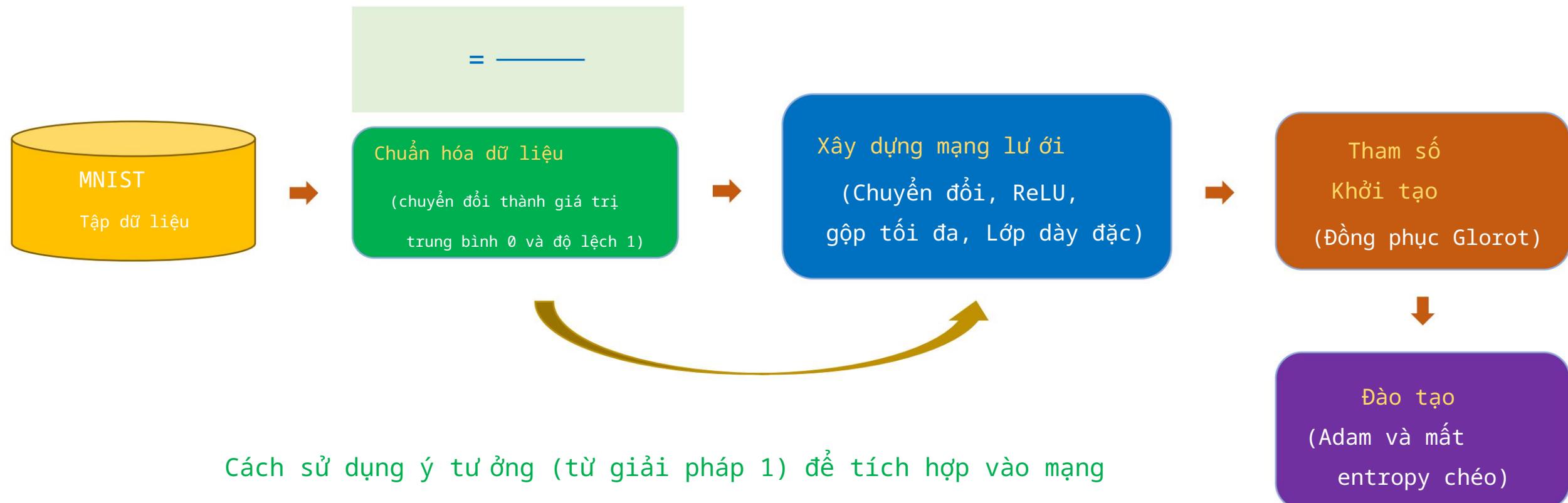
Chuẩn hóa từng kênh riêng biệt

```
transform = Compose([ToTensor(),
                    Normalize((0.5, 0.5, 0.5),
                              (0.5, 0.5, 0.5))])
train_set = CIFAR10(root='data', train=True,
                     download=True, transform=transform)
```



Đào tạo mạng

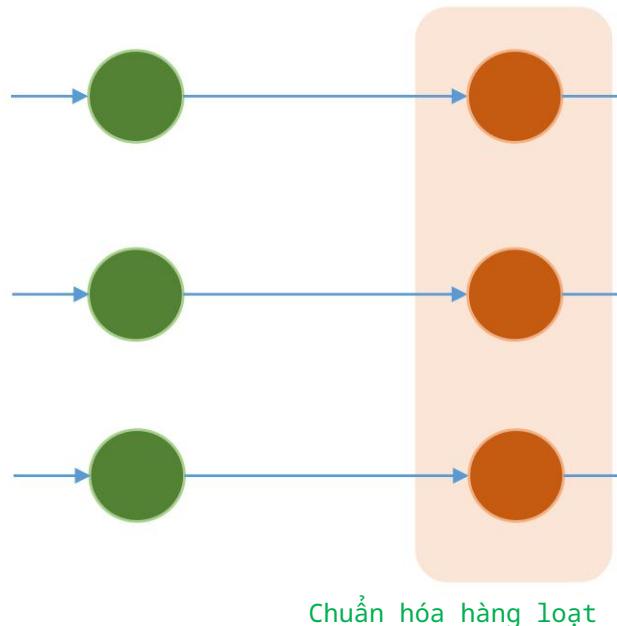
Giải pháp 2



Chuẩn hóa hàng loạt

Đào tạo mạng

Giải pháp 2: Chuẩn hóa hàng loạt



Không cần sai lệch khi sử dụng BN*

và được cập nhật trong chuyển tiếp
và β được cập nhật theo hướng ngược lại

Nhập dữ liệu cho một nút trong lớp chuẩn hóa hàng loạt

$$= \{ \quad, \quad \}$$

là kích thước lô nhỏ

Tính giá trị trung bình và phuơng sai

$$= \frac{1}{2} = \frac{1}{2} ()^2 = 1 = 1$$

Bình thường hóa

$$= \frac{1}{\sqrt{2 + }}$$

là một giá trị rất nhỏ

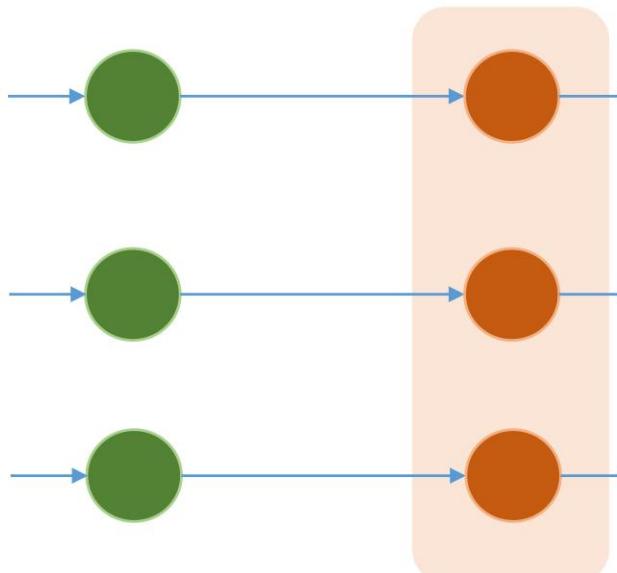
Quy mô và sự thay đổi

$$= + b$$

và β là hai tham số học tập

Đào tạo mạng

Giải pháp 2: Chuẩn hóa hàng loạt



Chuyện gì xảy ra nếu

$$\sigma = \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2} \text{ và } \beta =$$

Nhập dữ liệu cho một nút trong lớp chuẩn hóa hàng loạt

$$= \{x_1, \dots, x_n\}$$

là kích thước lô nhỏ

Tính giá trị trung bình và phuơng sai

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i = \frac{1}{n} (x_1 + x_2 + \dots + x_n)$$

Bình thường hóa

$$\hat{x} = \frac{x - \bar{x}}{\sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2}}$$

là một giá trị rất nhỏ

Quy mô và sự thay đổi

$$\hat{x} = \gamma \hat{x} + \beta \quad \text{và } \beta \text{ là hai tham số học tập}$$

Đào tạo mạng

= 10 5

$$\begin{aligned}
 &= 5,0 \\
 &= 2,64 \\
 &= 1,0 \\
 \beta = 0,0 &= \begin{bmatrix} 1,51 \\ 0,75 \\ 1,51 \\ 0,37 \\ 0,37 \\ 0,75 \end{bmatrix}
 \end{aligned}$$

The diagram illustrates a vector space with a horizontal axis and a vertical axis. A shaded orange region represents a subspace. A point labeled β is shown on the boundary of this subspace. The vertical axis has tick marks at 2, 5, 6, 7, 4, and 6. The horizontal axis has tick marks at 0, 61, 0, 61, 1, 22, 0, 0, 0, 61, and 1, 83.

$\beta = 0, 0$

$= 1, 63$

$= 1, 0$

$= 5, 0$

$= 0, 61$

$= 0, 61$

$= 1, 22$

$= 0, 0$

$= 0, 61$

$= 1, 83$

Giải pháp 2: Chuẩn hóa hàng loạt

Nhập dữ liệu cho một nút trong lớp chuẩn hóa hàng loạt

$$= \{$$

là kích thư ớc lô nhở

Tính giá trị trung bình và phu ơng sai

$$= \frac{1}{-1} \quad 2 = \frac{1}{-1} \quad ()^2$$

Bình thư ờng hò

$$= \frac{1}{\sqrt{2}}$$

là một quả tri rất nhỏ

Quy mô và sự thay đổi

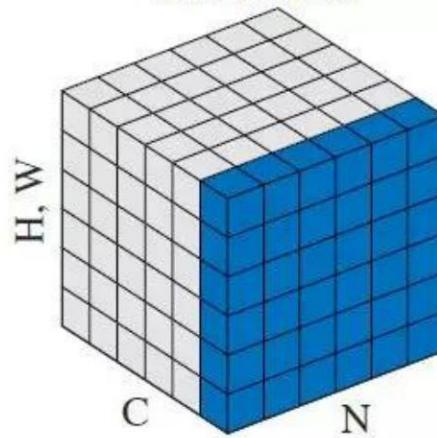
= + |

và β là hai tham số học tập

và β được cập nhật trong quá trình huấn luyện

Chuẩn hóa hàng loạt

Batch Norm



[https://arxiv.org/
pdf/ 1803.08494.pdf](https://arxiv.org/pdf/1803.08494.pdf)

$$= 10 \quad 5$$

$$= \frac{1}{\sqrt{\frac{1}{3} \times \frac{1}{3}}} = \frac{1}{\sqrt{\frac{1}{3} + \frac{1}{3} + \frac{1}{3}}} = \sqrt{\frac{1}{3} + \frac{1}{3} + \frac{1}{3}}$$

$$\sigma^2 = 2,5$$

$$\sigma^2 = 6,58$$

$$\beta = 1,0$$

$$\gamma = 0,0$$

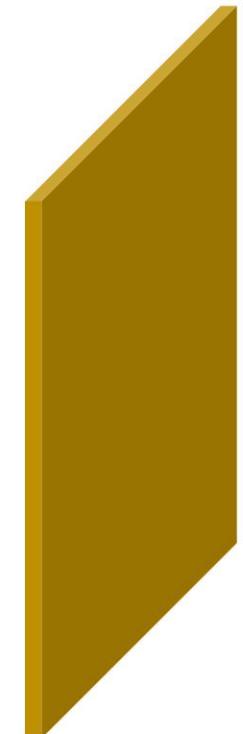
mẫu 1 mẫu 2 mẫu 3

$$= \left\{ \begin{bmatrix} 7 \\ 5 \end{bmatrix}, \begin{bmatrix} 0 \\ 7 \end{bmatrix}, \begin{bmatrix} 2 \\ 0 \end{bmatrix} \right\}$$



cỡ lô = 3

input_shape = (BS=3, C=1, H=2, W=2)



$$= \left\{ \begin{bmatrix} 1,75 & 0,97 \\ 0,97 & 0,58 \end{bmatrix}, \begin{bmatrix} 0,97 & 1,75 \\ 0,19 & 0,58 \end{bmatrix}, \begin{bmatrix} 0,19 & 0,97 \\ 0,97 & 0,58 \end{bmatrix} \right\}$$



$$= \left\{ \begin{bmatrix} 1,75 & 0,97 & 0,58 \\ 0,97 & 1,75 & 0,58 \\ 0,19 & 0,97 & 0,58 \end{bmatrix} \right\}$$

Đào tạo mạng

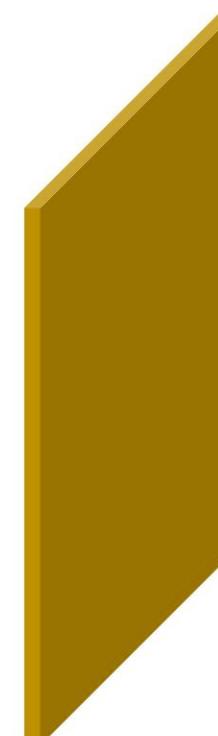
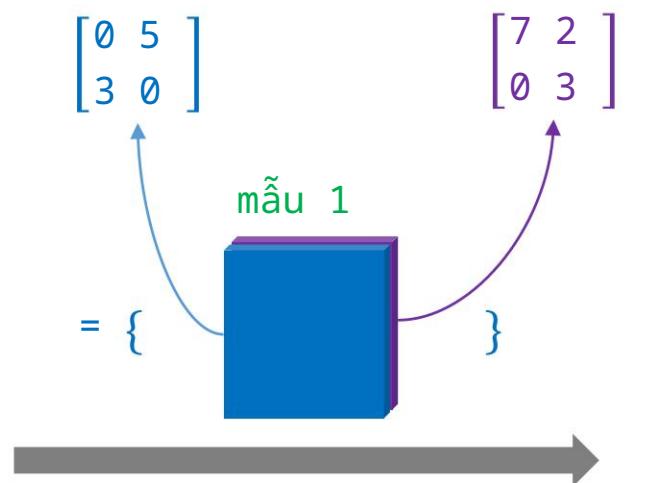
= 10 5

= [2.0, 3.0]

$^2 = [6,0, 8,67]$

= 1,0

$\beta = 0,0$



$$= \left\{ \begin{bmatrix} 0,94 & 1,41 \\ 0,47 & -0,94 \\ 1,56 & -0,39 \\ 1,17 & 0 \end{bmatrix} \right\}$$

$$= \left\{ \begin{bmatrix} 0,94 & 1,41 \\ 0,47 & -0,94 \\ 1,56 & -0,39 \\ 1,17 & 0 \end{bmatrix} \right\}$$

cỡ lô = 1

sample_shape = (BS=1, C=2, H=2, W=2)

Đào tạo mạng

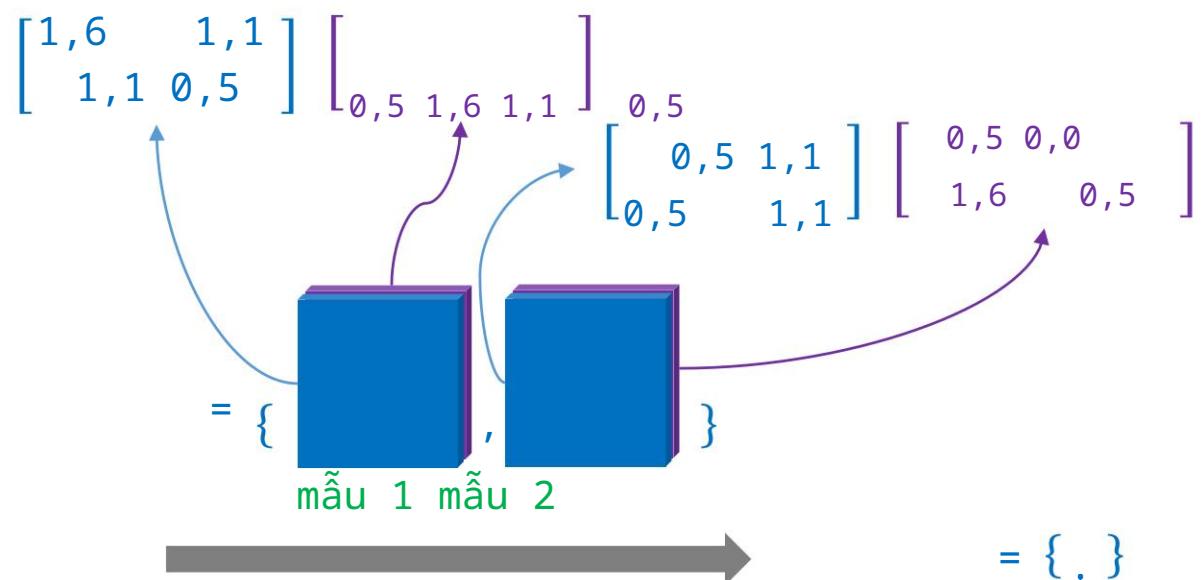
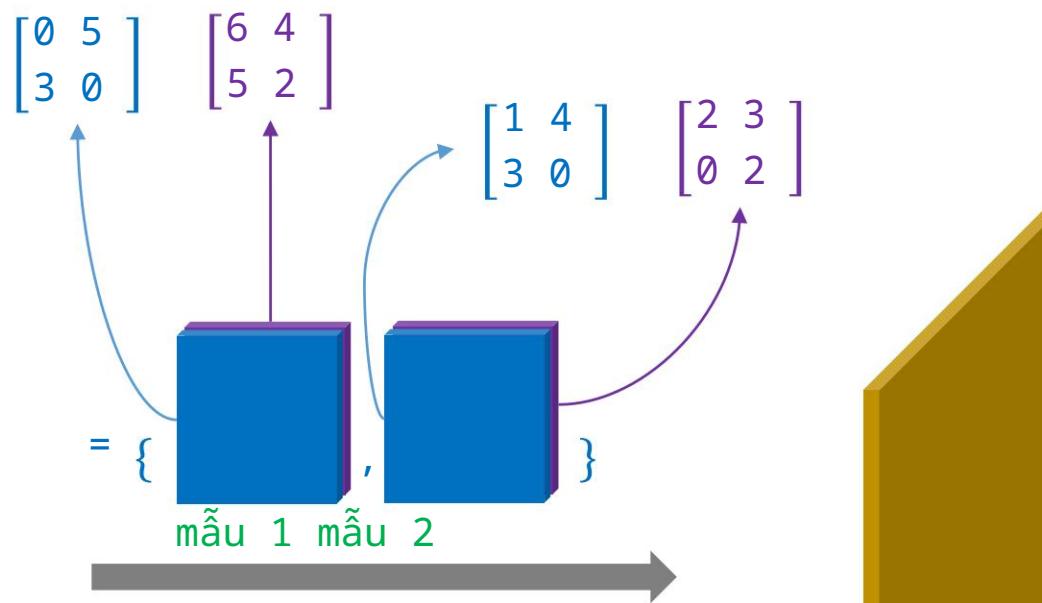
$$= 10 \quad 5$$

$$= [2.0, \quad 3.0]$$

$$^2 = [4.0, \quad 3.7]$$

$$= 1,0$$

$$\beta = 0,0$$



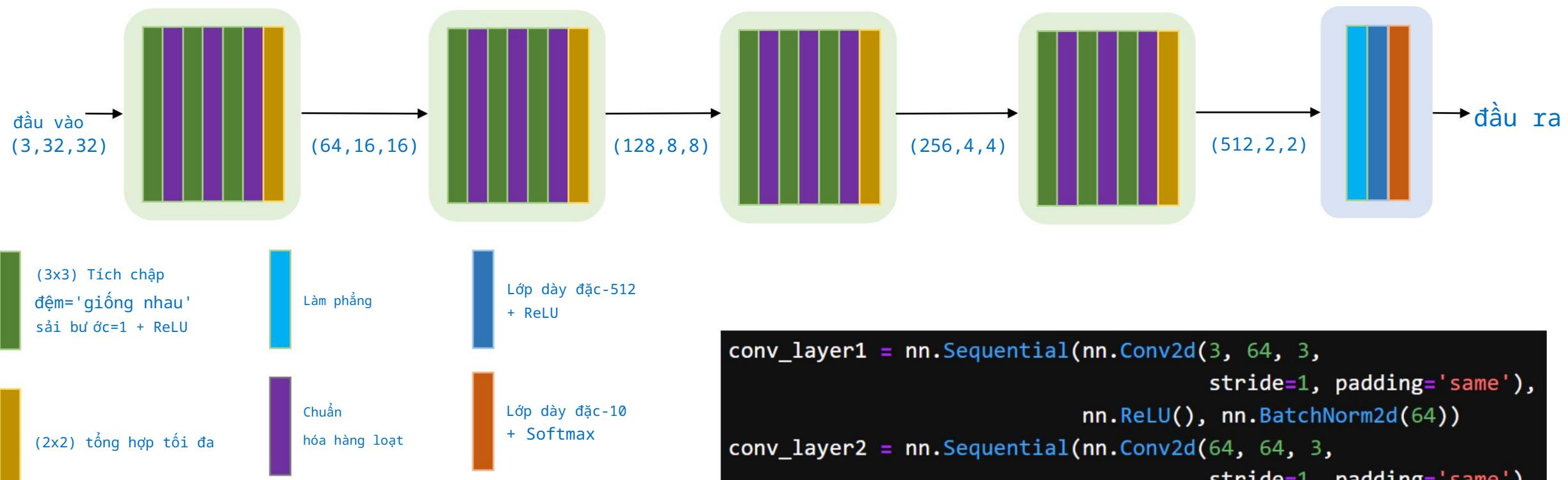
Lớp định mức hàng loạt

cố lô = 2

sample_shape = (BS=2, C=2, H=2, W=2)

Đào tạo mạng

Giải pháp 2: Chuẩn hóa hàng loạt

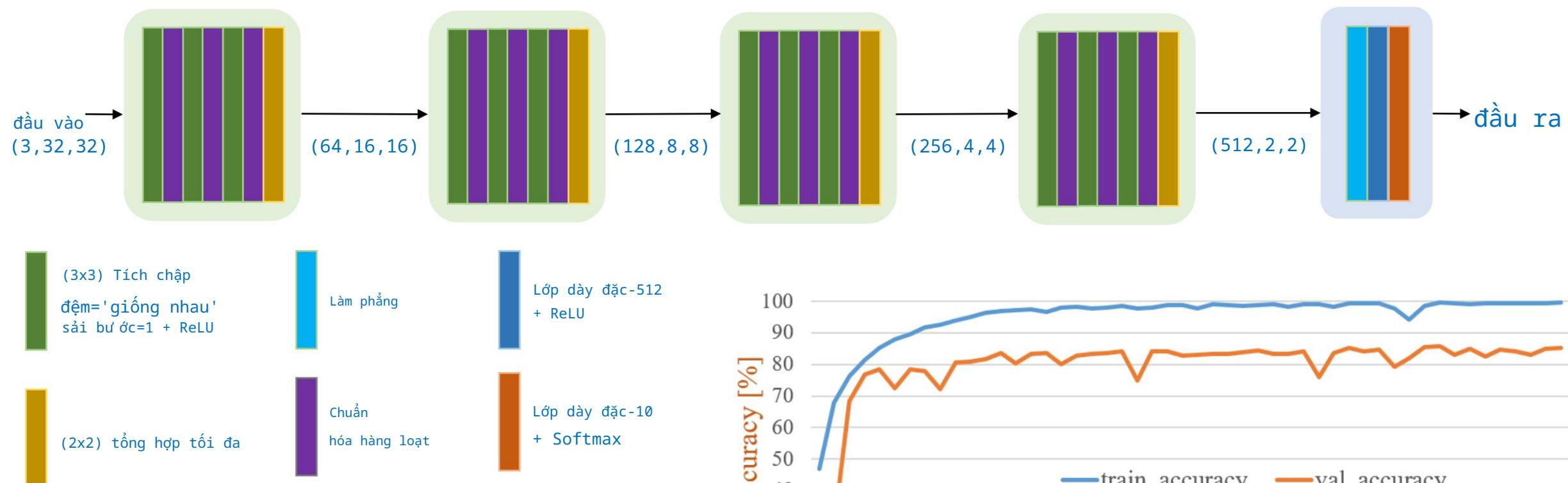


```
torch.nn.BatchNorm2d(num_features)
num_features (int): C từ kích thư ớc đầu vào dự
kiến (N, C, H, W)
```

```
conv_layer1 = nn.Sequential(nn.Conv2d(3, 64, 3,
                                         stride=1, padding='same'),
                            nn.ReLU(), nn.BatchNorm2d(64))
conv_layer2 = nn.Sequential(nn.Conv2d(64, 64, 3,
                                         stride=1, padding='same'),
                            nn.ReLU(), nn.BatchNorm2d(64))
conv_layer3 = nn.Sequential(nn.Conv2d(64, 64, 3,
                                         stride=1, padding='same'),
                            nn.ReLU(), nn.BatchNorm2d(64),
                            nn.MaxPool2d(2, 2))
```

Đào tạo mạng

Giải pháp 2: Chuẩn hóa hàng loạt



```
conv = nn.Sequential(nn.Conv2d(3, 64, 3),
                     nn.ReLU(),
                     nn.BatchNorm2d(64))
```

Đào tạo mạng

Giải pháp 2: Chuẩn hóa hàng loạt

Tăng tốc đào tạo

Giảm sự phụ thuộc vào trọng lượng ban đầu

Khái quát hóa mô hình

Nhập dữ liệu cho một nút trong lớp chuẩn hóa hàng loạt

$$= \{ \quad _1, \dots, \quad \}$$

là kích thước lô nhỏ

Tính giá trị trung bình và phuơng sai

$$= \frac{1}{n} \sum_{i=1}^n x_i = \bar{x}$$

$$= \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2} = s$$

Bình thường hóa

$$= \frac{x - \bar{x}}{s}$$

là một giá trị rất nhỏ

Quy mô và sự thay đổi

$$= \alpha x + \beta$$

và β là hai tham số học tập

Đào tạo mạng

Giải pháp 3: Sử dụng khả năng khởi tạo mạnh mẽ hơn



Anh khởi tạo (2015)

Đi sâu vào bộ chỉnh lưu: Vượt qua hiệu suất ở cấp độ con người trên Phân loại ImageNet <https://arxiv.org/pdf/1502.01852.pdf>

Đào tạo mạng

Giải pháp 3: Khởi tạo

Khởi tạo Glorot (2010)

$$1 \sim \begin{pmatrix} 0, & - \end{pmatrix}$$

n_j là #đầu vào trong lớp j

Giả sử các hàm kích hoạt là tuyến tính

Anh khởi tạo (2015)

Tính đến chức năng kích hoạt

Thích ứng với kích hoạt ReLU

$$2 \sim \begin{pmatrix} 0, & - \end{pmatrix}$$

```
def initialize_weights(self):
    for m in self.modules():
        if isinstance(m, nn.Conv2d):
            init.kaiming_normal_(m.weight,
                                  nonlinearity='relu')
            if m.bias is not None:
                init.zeros_(m.bias)
        elif isinstance(m, nn.Linear):
            init.kaiming_normal_(m.weight,
                                  nonlinearity='relu')
            if m.bias is not None:
                init.zeros_(m.bias)
```

Chuẩn hóa dữ liệu [0,1]

Anh ấy khởi tạo bình thường

Trình tối ưu hóa Adam với lr=1e-3

Đào tạo mạng

Giải pháp 3: Khởi tạo

Khởi tạo Glorot (2010)

$$\mathbf{1} \sim \left(0, \frac{1}{\sqrt{n_j}}\right)$$

n_j là #đầu vào trong lớp j

Giả sử các hàm kích hoạt là tuyến tính

Anh khởi tạo (2015)

Tính đến chức năng kích hoạt

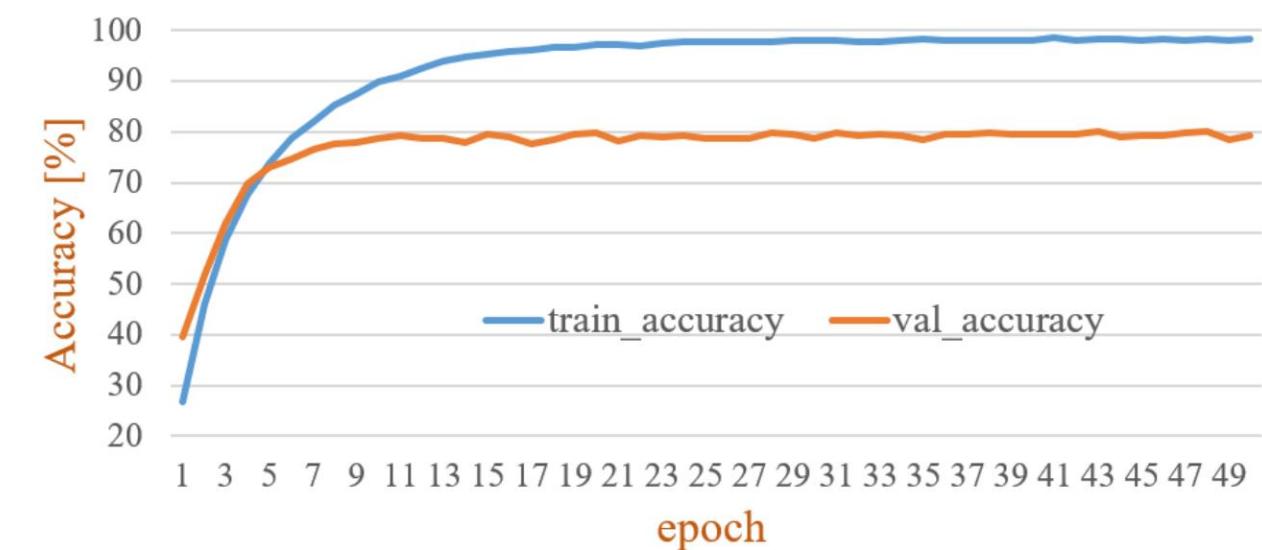
Thích ứng với kích hoạt ReLU

$$\mathbf{2} \sim \left(0, \frac{1}{\sqrt{n_j}}\right)$$

Chuẩn hóa dữ liệu [0,1]

Anh ấy khởi tạo bình thường

Trình tối ưu hóa Adam với lr=1e-3

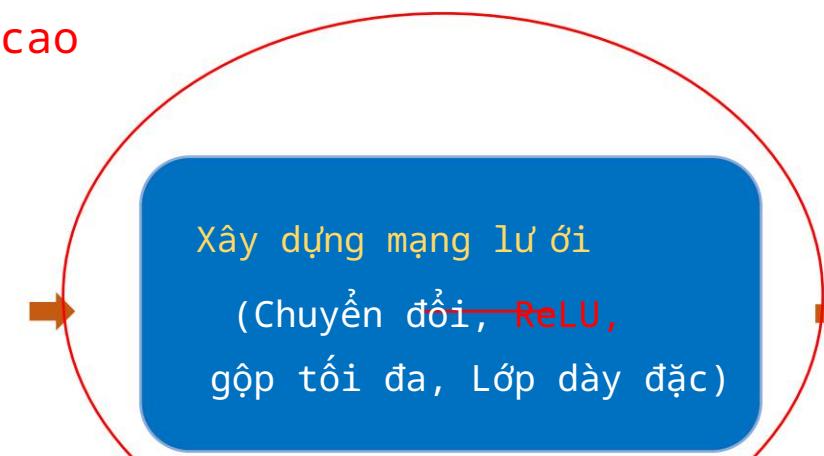


Đào tạo mạng

Giải pháp 4: Sử dụng kích hoạt nâng cao



Chuẩn hóa dữ liệu (chuyển đổi thành giá trị trung bình 0 và độ lệch 1)



Tham số
Khởi tạo
(Đồng phục Glorot)



Đào tạo
(Adam và mất entropy chéo)

2017

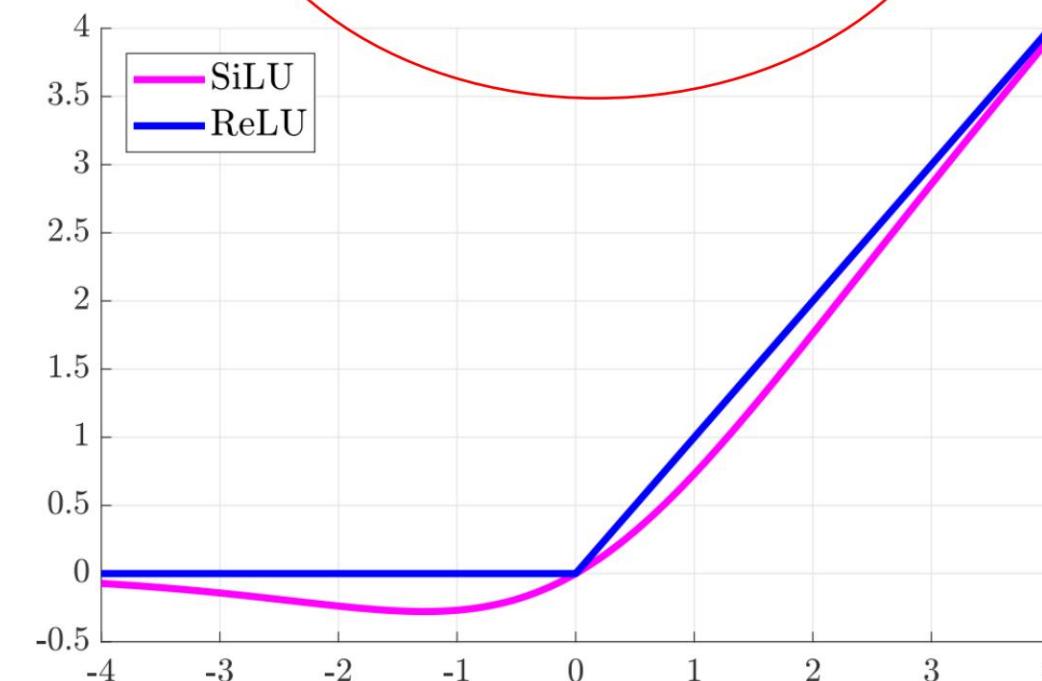
Đơn vị tuyến tính sigmoid (SiLU)

$$\text{lắc lư}(x) = \frac{1}{1 + e^{-x}}$$

2010

 $\text{ReLU}(x) = 0 \text{ nếu } x \geq 0$

```
conv_layer1 = nn.Sequential(
    nn.Conv2d(3, 64, 3, stride=1, padding=1),
    nn.SiLU()
)
```



<https://arxiv.org/pdf/1702.03118.pdf>

Đào tạo mạng

Giải pháp 4:

Sử dụng kích hoạt nâng cao

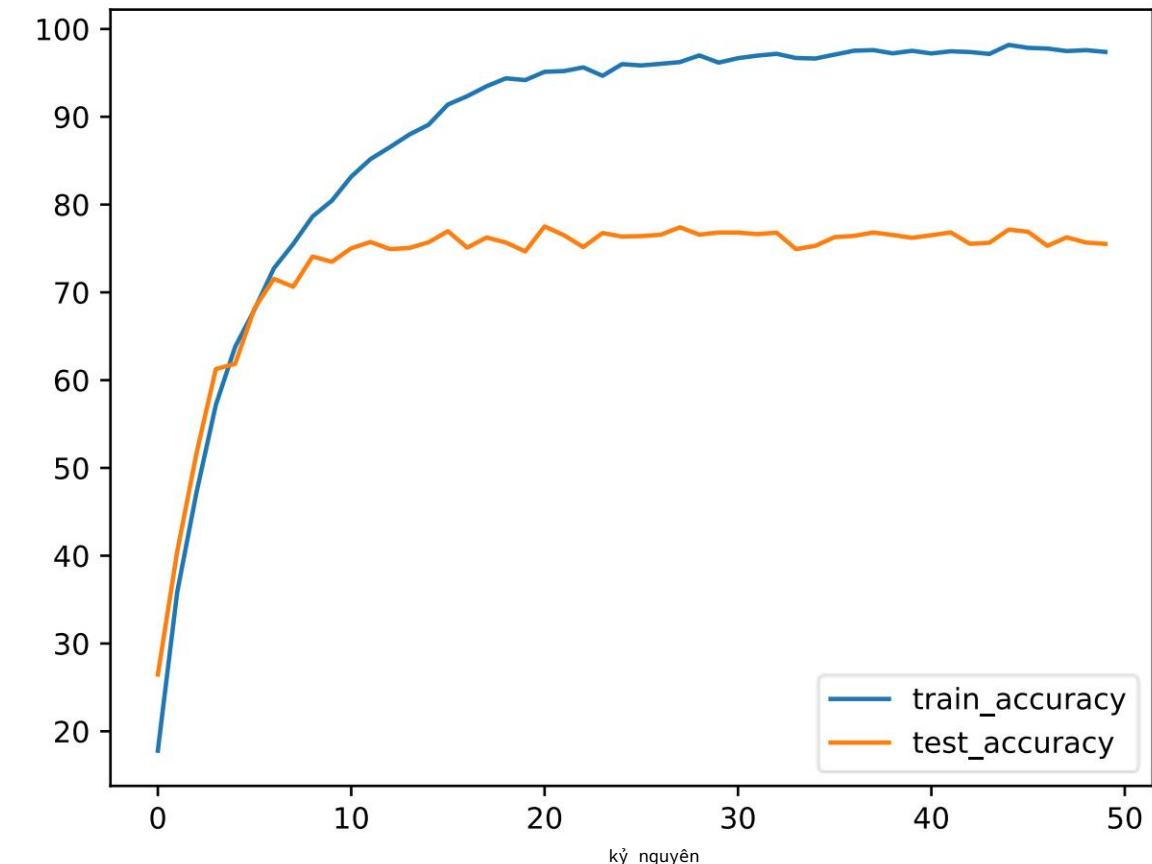
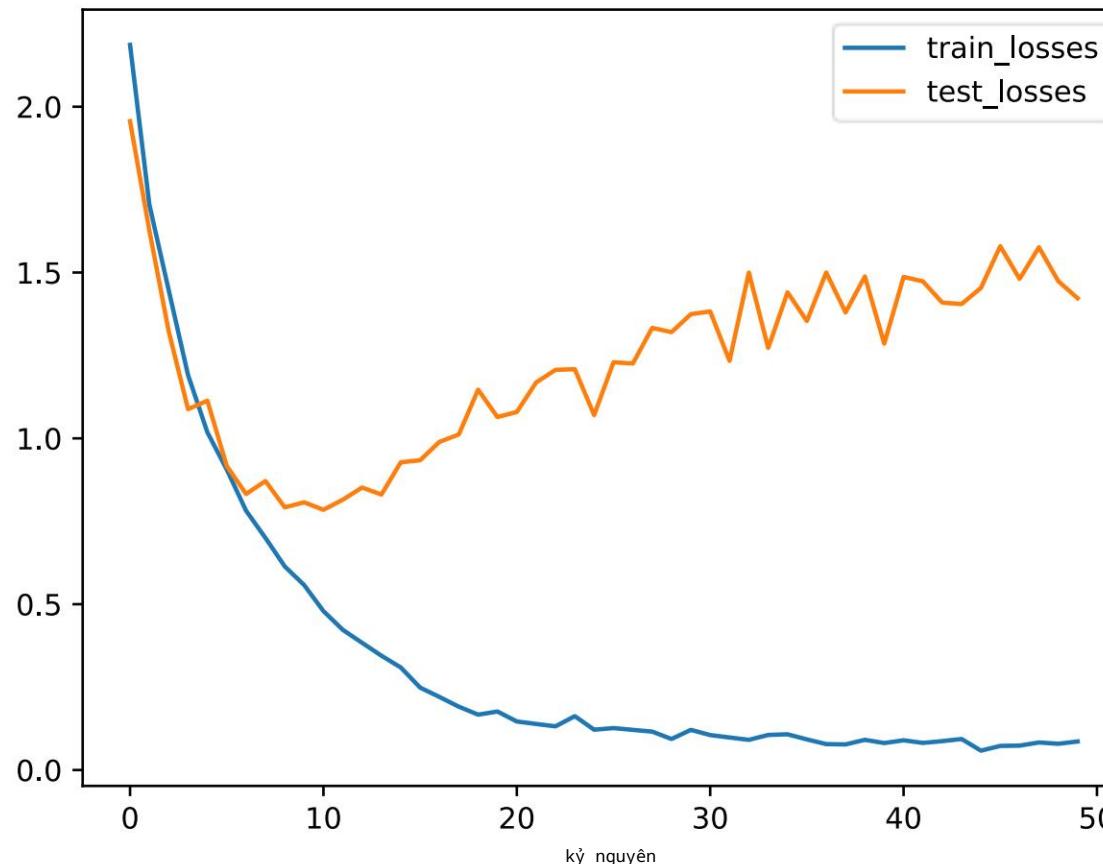
2017

Đơn vị tuyến tính sigmoid (SiLU)

$$\text{vung}(x) = \frac{1}{1 + }$$

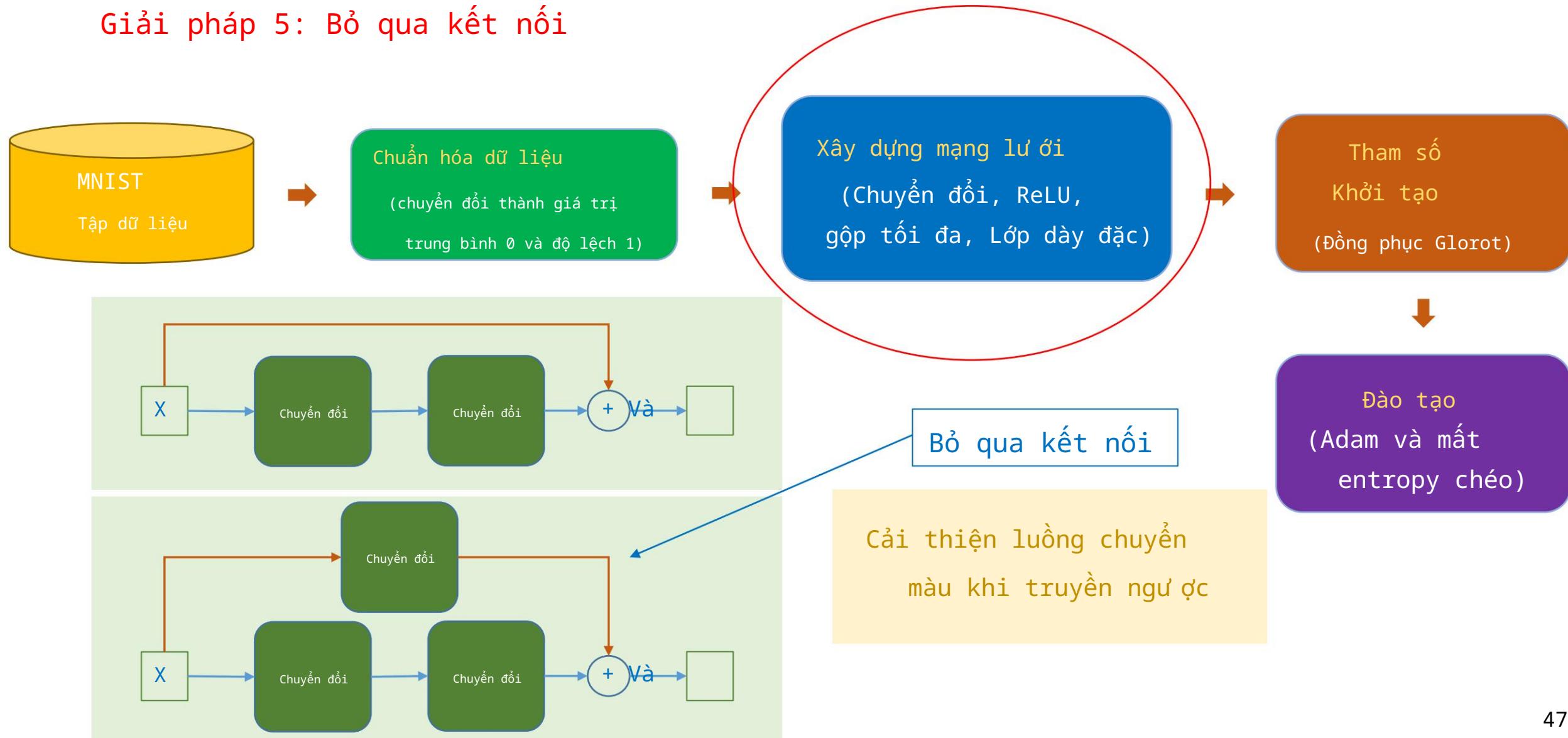
```
conv_layer1 = nn.Sequential(
    nn.Conv2d(3, 64, 3, stride=1, padding=1),
    nn.SiLU()
)
```

<https://arxiv.org/pdf/1702.03118.pdf>



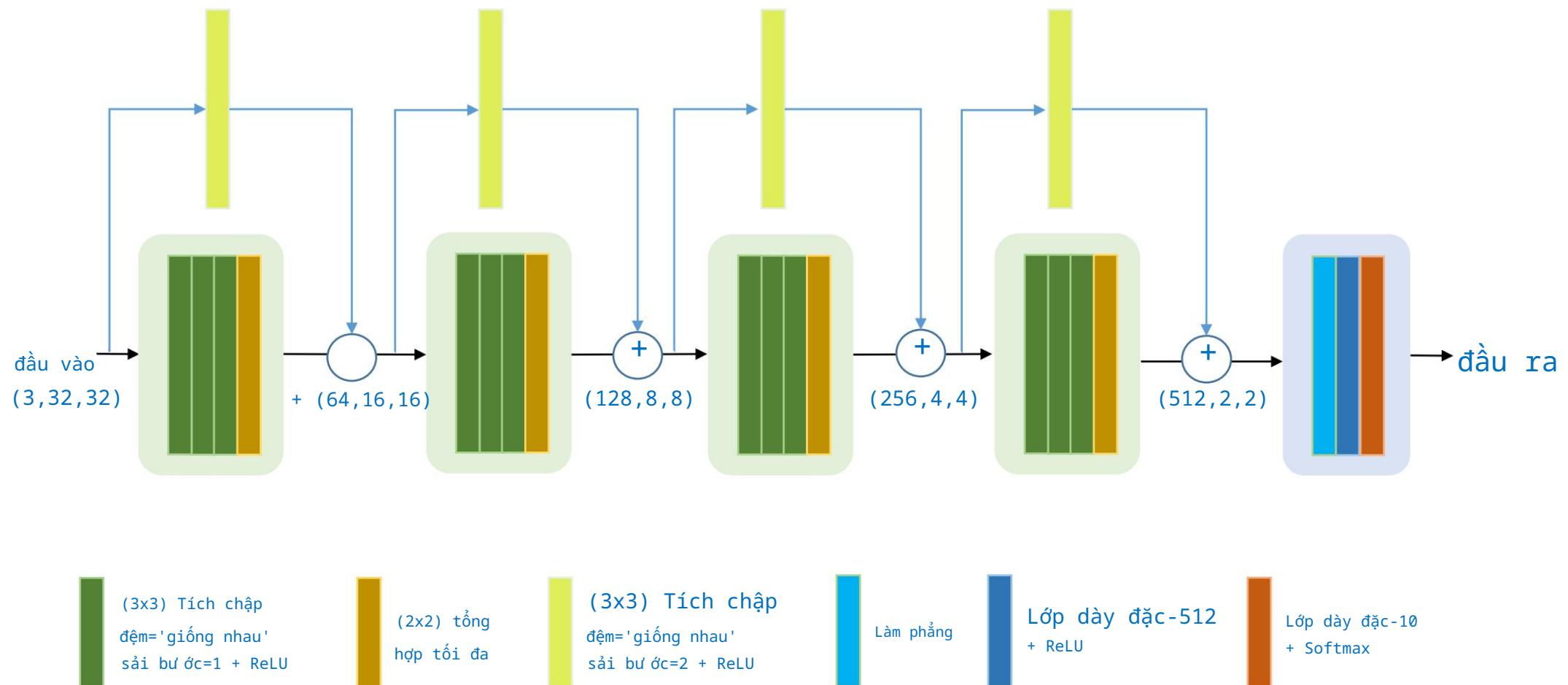
Đào tạo mạng

Giải pháp 5: Bỏ qua kết nối



Đào tạo mạng

Giải pháp 5: Bỏ qua kết nối



Mạng

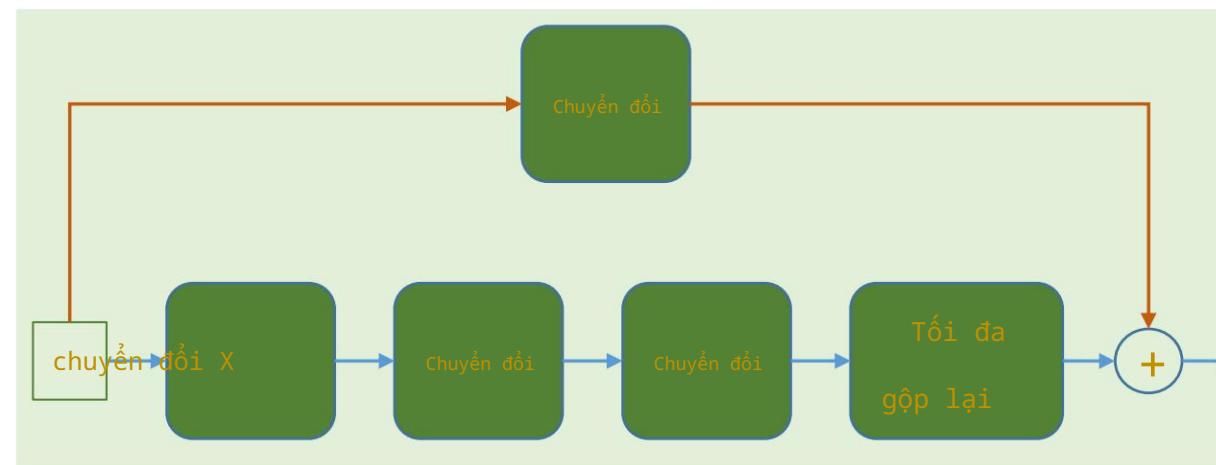
Đào tạo

Giải pháp 5:

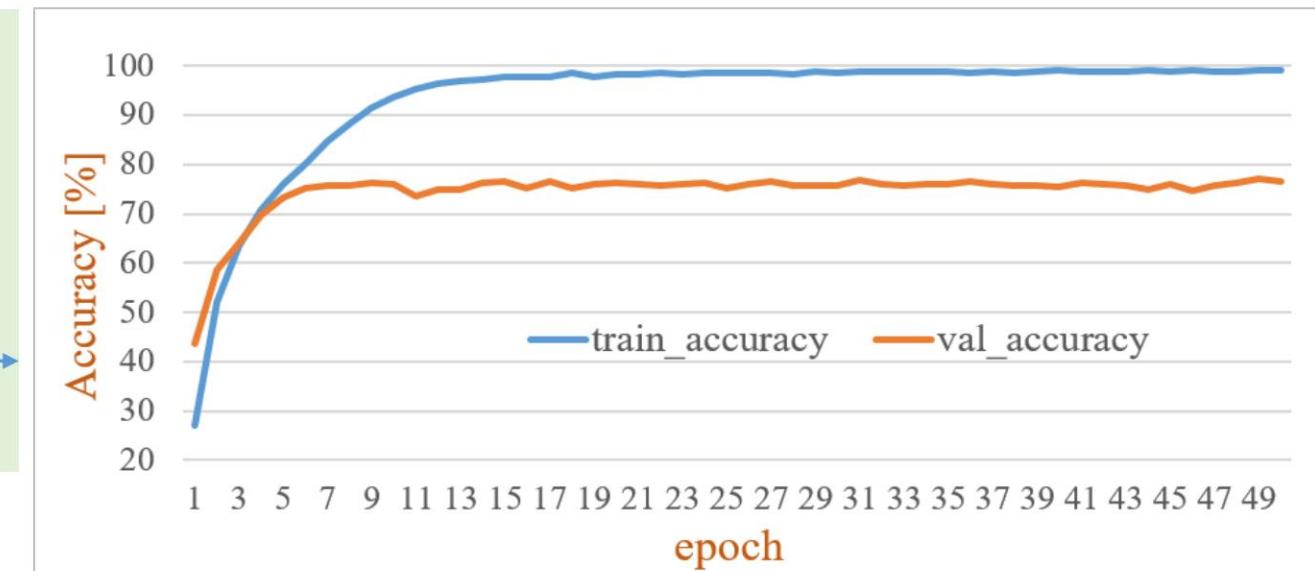
Bỏ qua kết nối

```
conv_layer1 = nn.Sequential(nn.Conv2d(3, 64, 3, stride=1, padding='same'), nn.ReLU())
conv_layer2 = nn.Sequential(nn.Conv2d(64, 64, 3, stride=1, padding='same'), nn.ReLU())
conv_layer3 = nn.Sequential(nn.Conv2d(64, 64, 3, stride=1, padding='same'), nn.ReLU(),
                            nn.MaxPool2d(2, 2))
res_layer1 = nn.Sequential(nn.Conv2d(3, 64, 3, stride=2, padding=1), nn.ReLU())

# Given x
previous_input_x = x
x = self.conv_layer1(x)
x = self.conv_layer2(x)
x = self.conv_layer3(x)
res = self.res_layer1(previous_input_x)
x = x + res
```

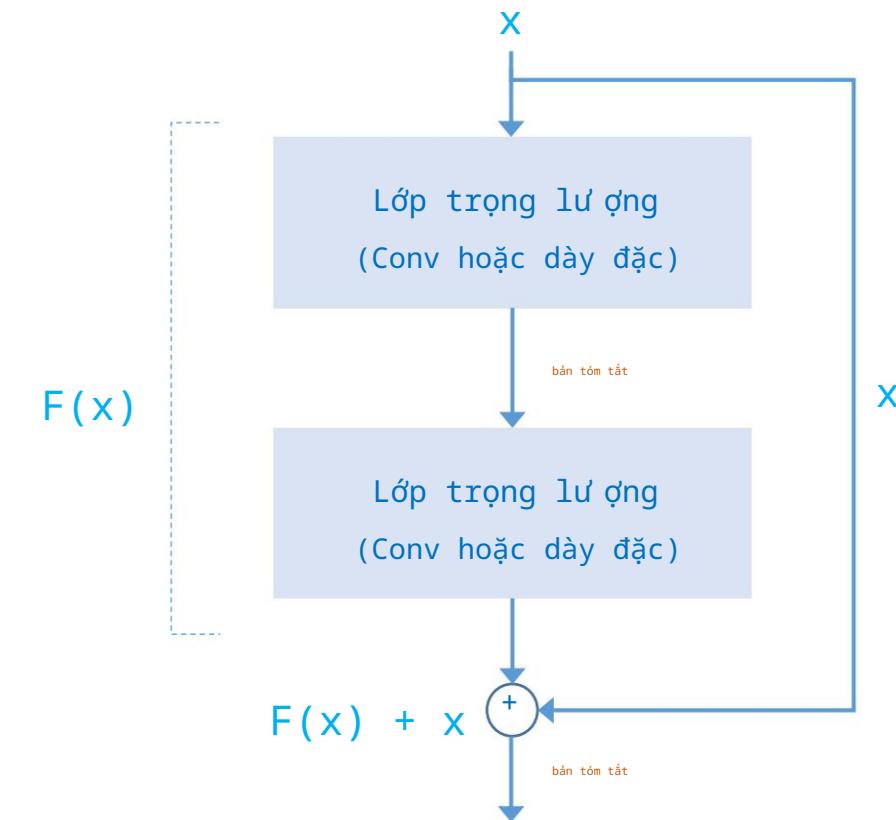
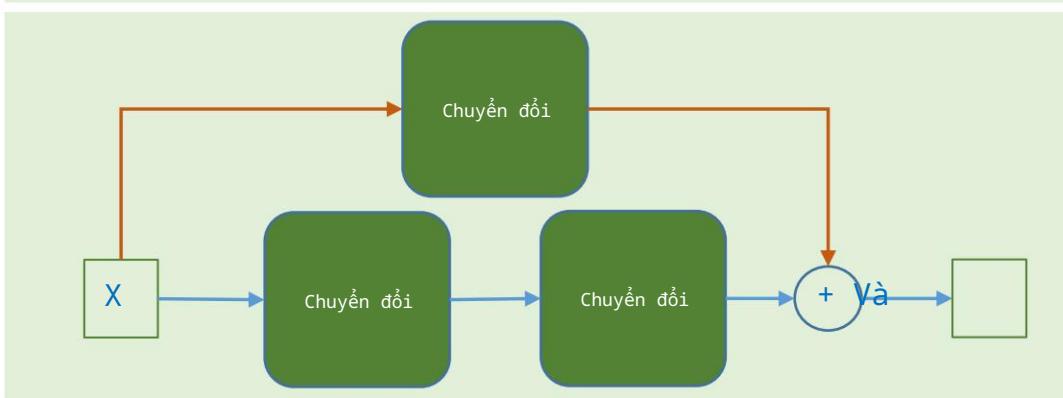
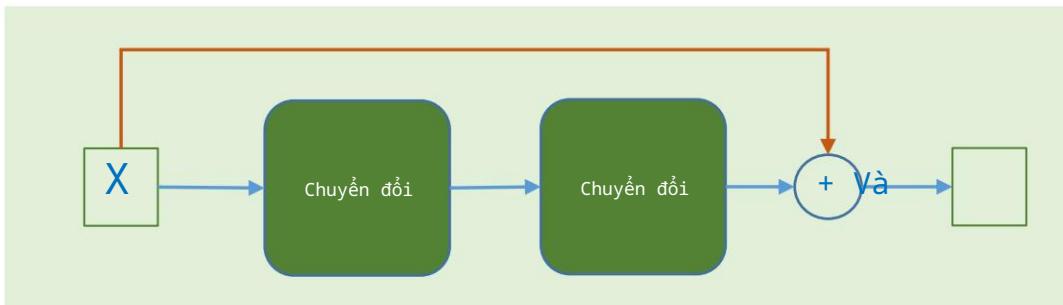


Có một số biến thể sử dụng kết nối bỏ qua hoàn toàn, ghép nối, kết nối bỏ qua dài



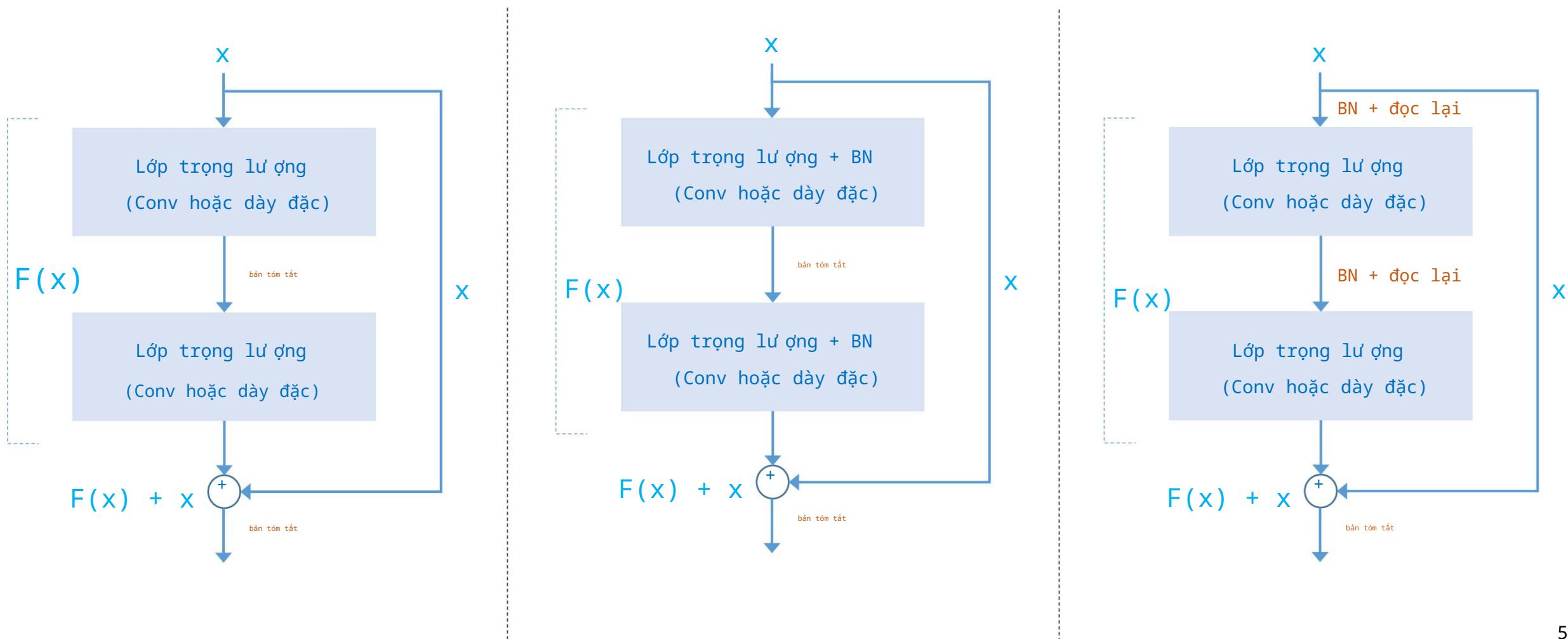
Đào tạo mạng

Giải pháp 5: Bỏ qua kết nối



Đào tạo mạng

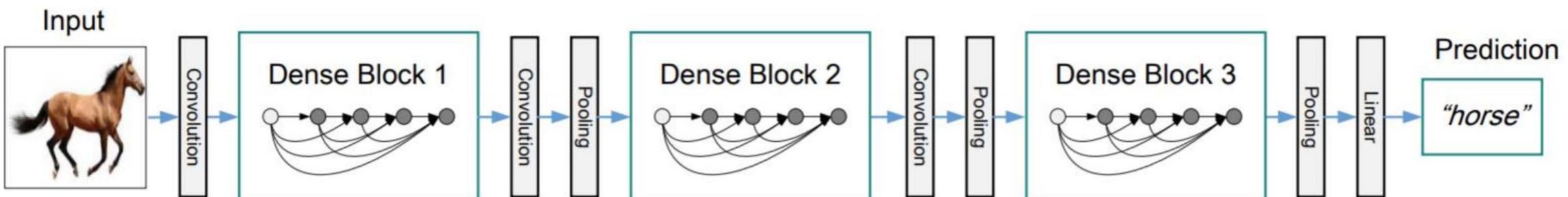
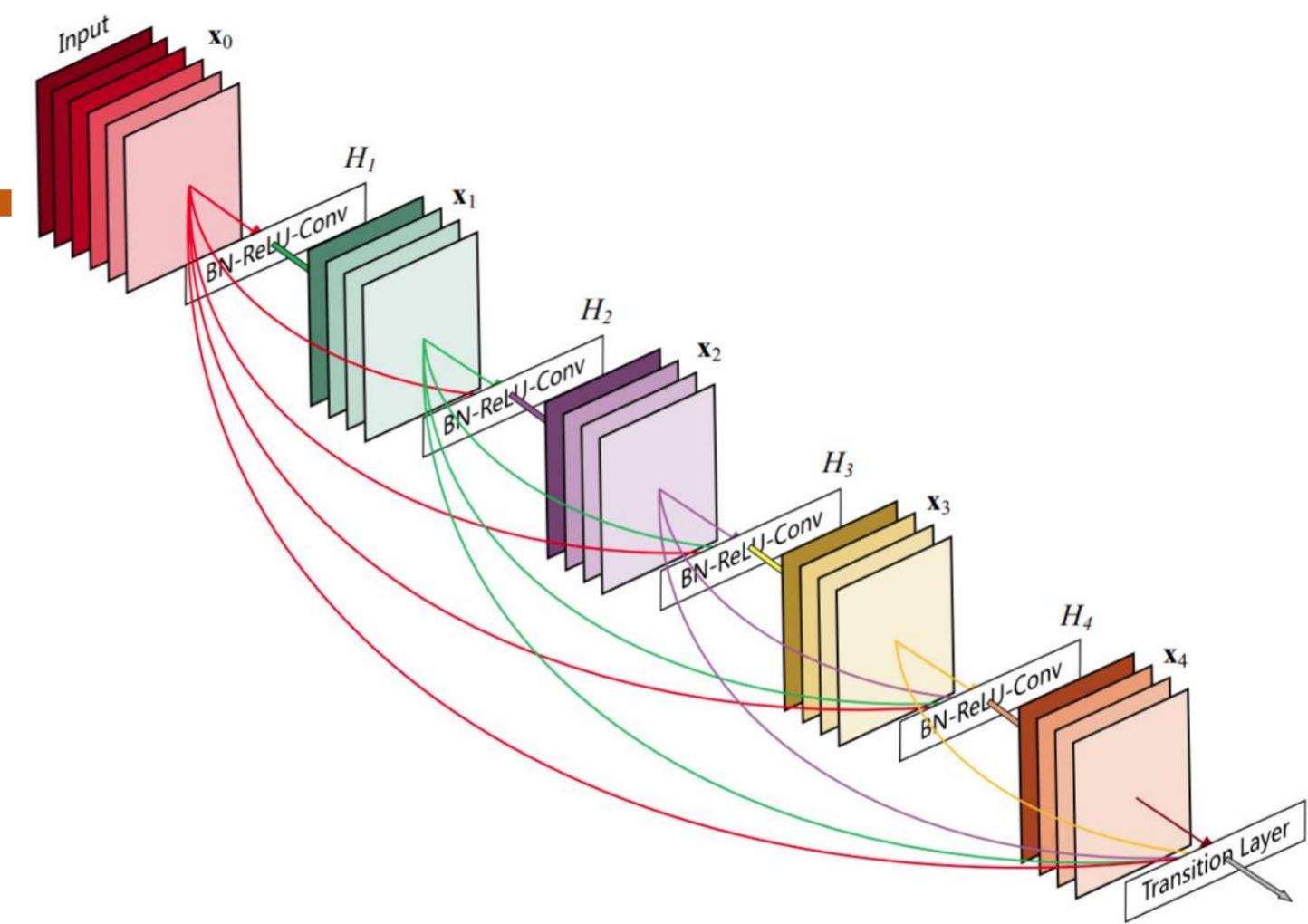
Giải pháp 5: Bỏ qua kết nối



Đào tạo mạng

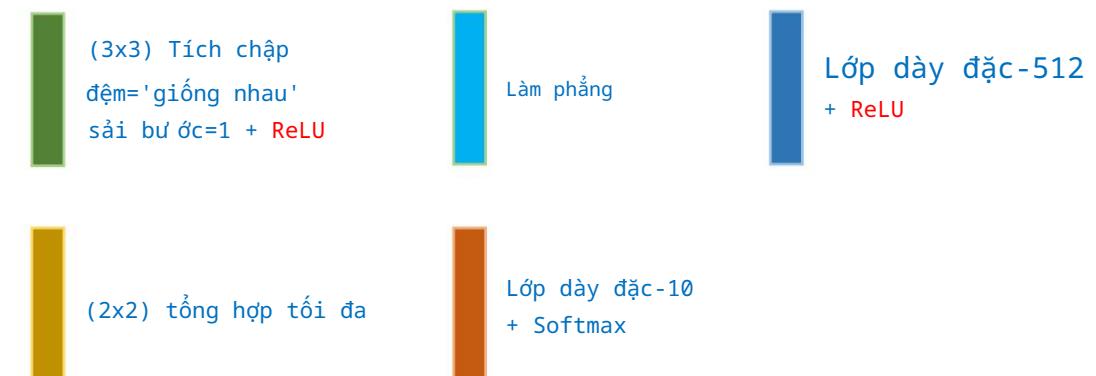
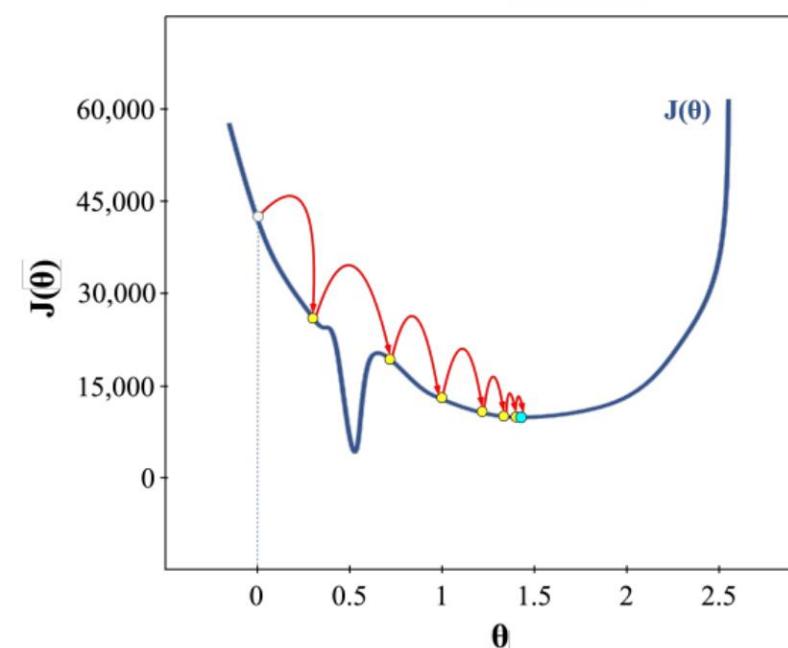
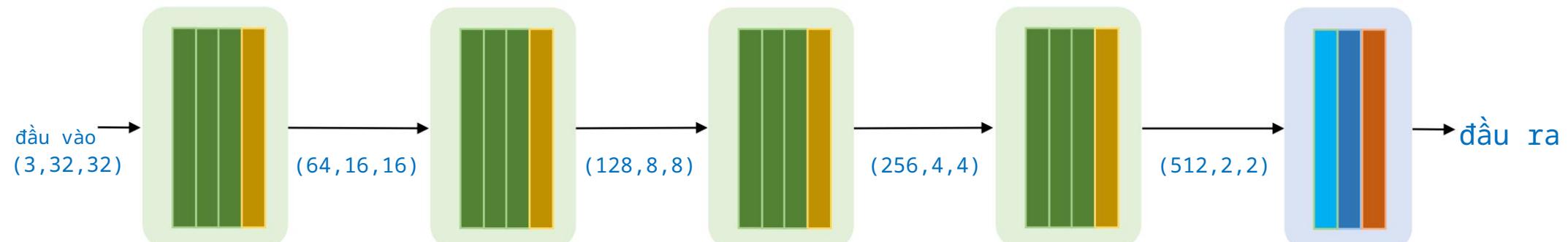
Giải pháp 5: Bỏ qua kết nối

<https://arxiv.org/pdf/1608.06993v5.pdf>



Đào tạo mạng

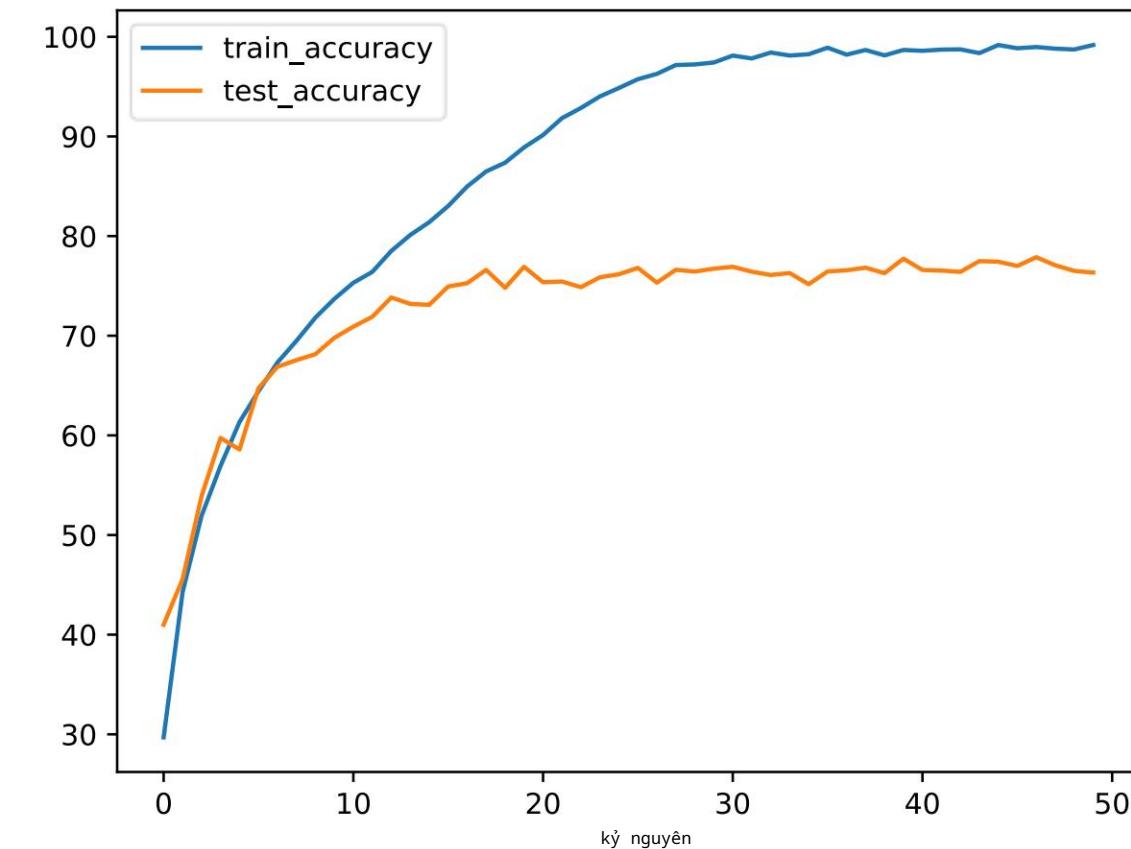
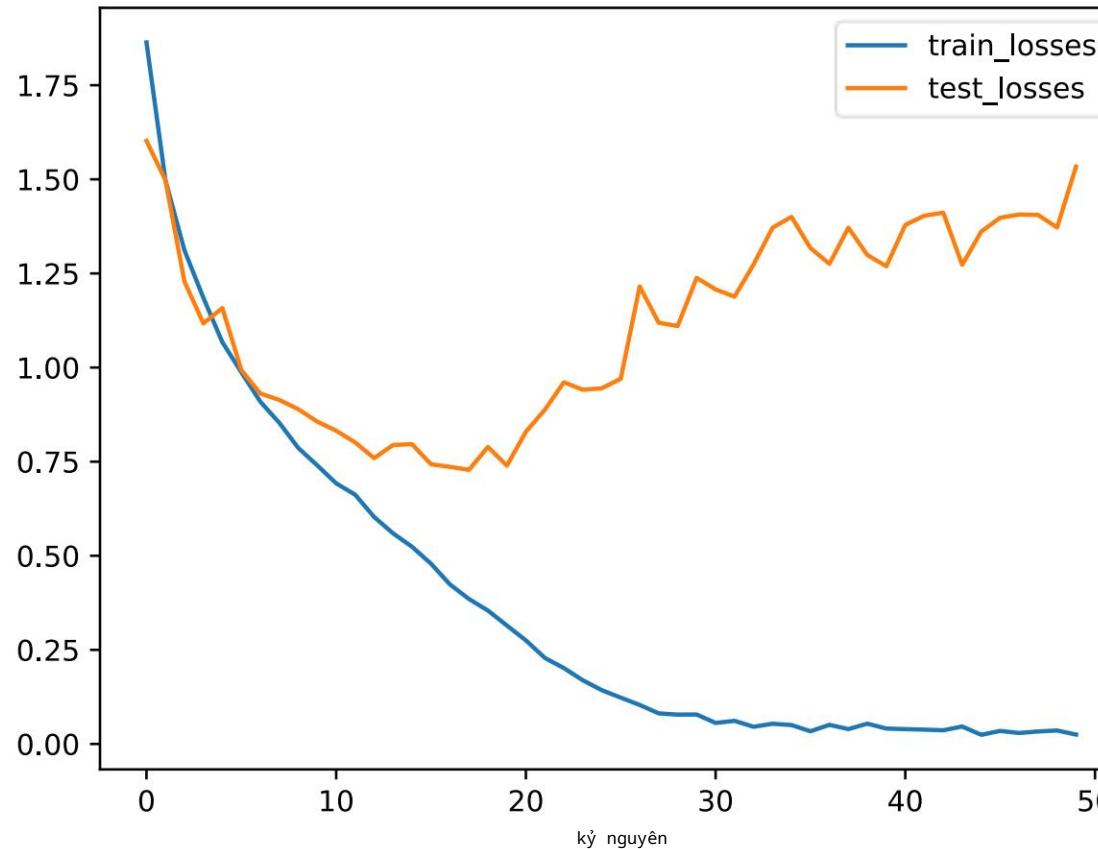
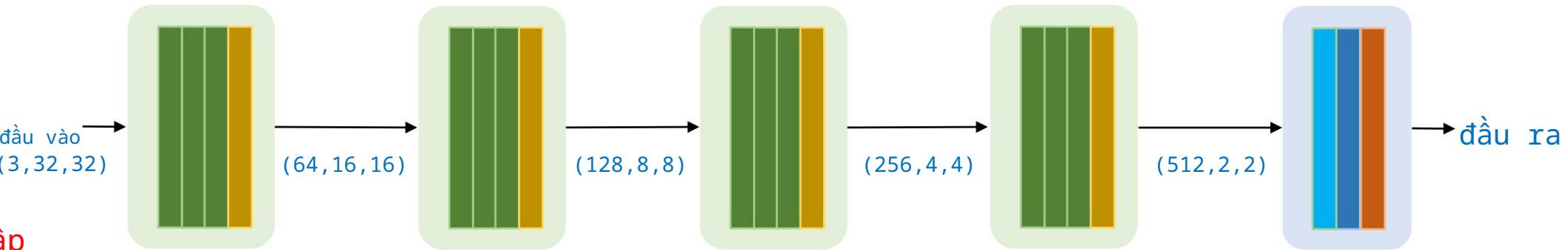
Giải pháp 6: Giảm tỷ lệ học tập



```
# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = Adam(model.parameters(), lr=1e-4)
```

Mạng Đào tạo

Giảm tỷ lệ học tập



Đọc thêm

Bỏ qua kết nối

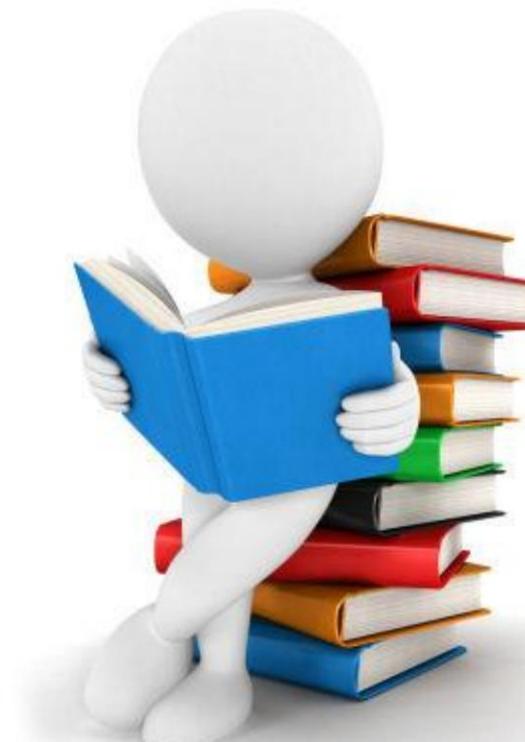
<https://theaisummer.com/skip-connections/>

Đang cố gắng điều chỉnh quá mức Dữ liệu

<http://karpathy.github.io/2019/04/25/recipe/>

Mạng dày đặc

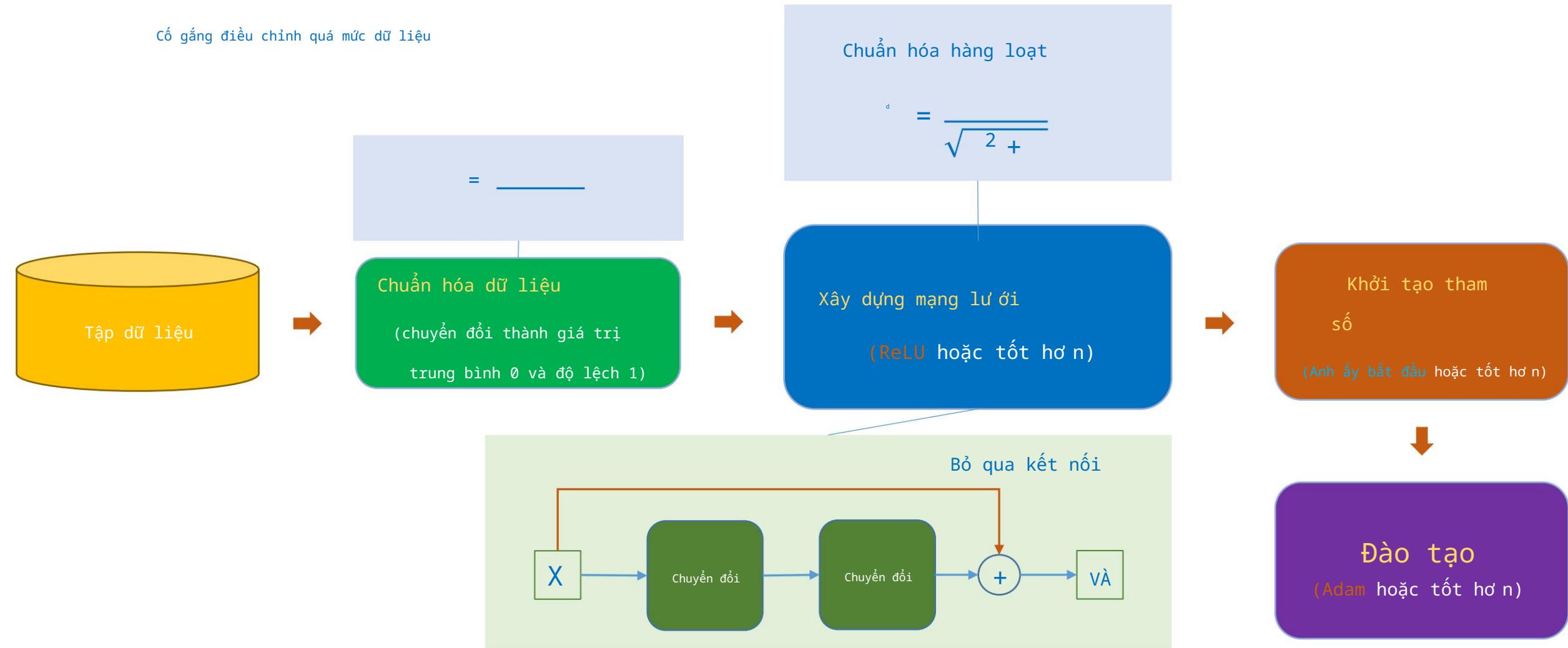
<https://arxiv.org/pdf/1608.06993v5.pdf>



Bản tóm tắt

Đào tạo mô hình CNN

Có găng điều chỉnh quá mức dữ liệu





AI VIETNAM

Khóa học tất cả trong một

Khái quát hóa mô hình

Quang-Vinh Dinh

Ph.D. in Computer Science

Đào tạo mạng

Tập dữ liệu Cifar-10
(tập dữ liệu phức tạp)

Hình ảnh màu

Độ phân giải=32x32

Bộ huấn luyện: 50000 mẫu

Bộ thử nghiệm: 10000 mẫu

Máy bay



ô tô



chim



con mèo



con nai



chó



con éch



ngựa



tàu thủy



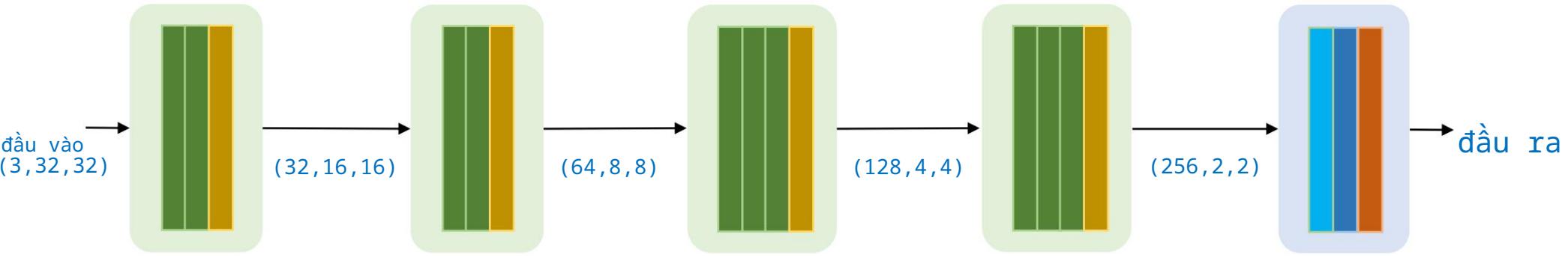
xe tải



Khái quát hóa mô hình

Cifar-10

Tập dữ liệu



Chuẩn hóa dữ liệu

(chuyển đổi thành giá trị

trung bình 0 và độ lệch 1)

$$\begin{aligned} &= \text{---} \\ &= \frac{1}{\sqrt{\dots}} \\ &= \sqrt{\frac{1}{\dots} (\dots)^2} \end{aligned}$$

(3x3) Tích chập
đệm='giống nhau'
sải bù 0c=1 + ReLU

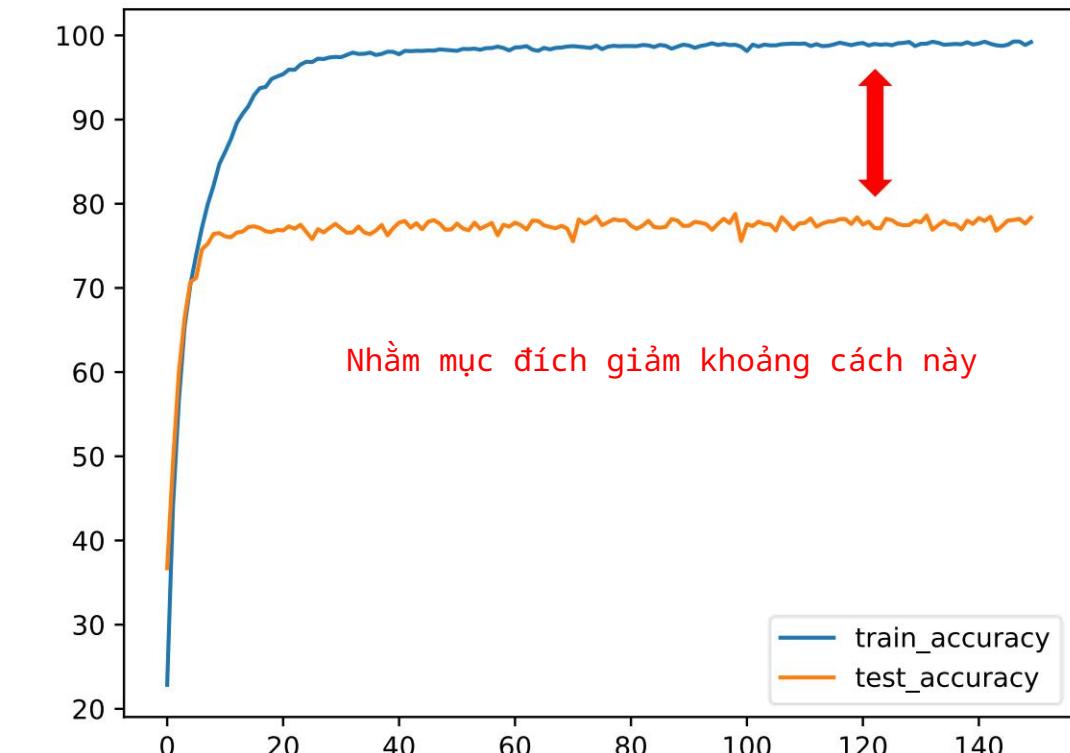
(2x2) tổng hợp tối đa

Làm phẳng

Lớp dày đặc-10
+ Softmax

Lớp dày đặc-512
+ ReLU

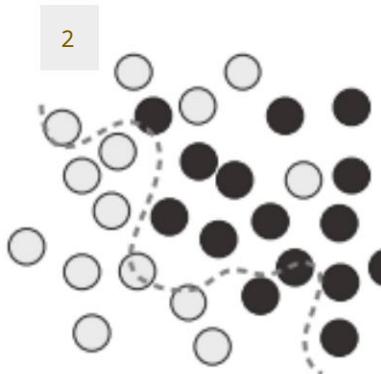
Adam lr=1e-3 ; Anh ấy bắt đầu



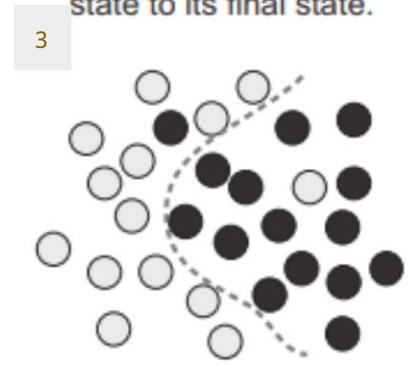
Before training:
the model starts
with a random initial state.



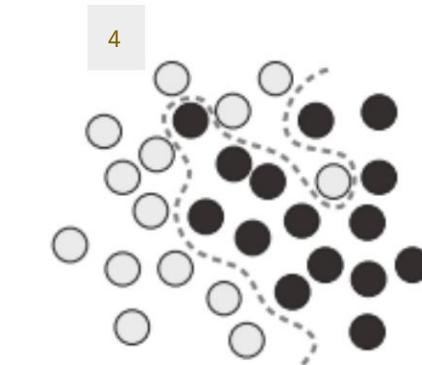
Beginning of training:
the model gradually
moves toward a better fit.



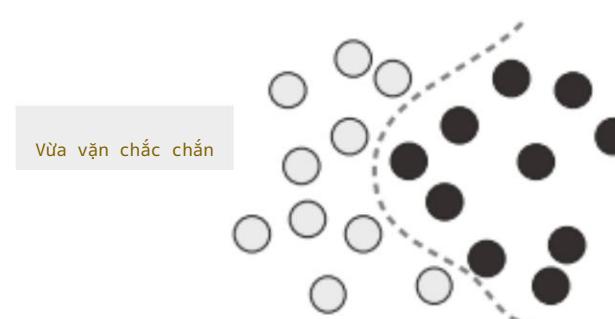
Further training: a robust
fit is achieved, transitively,
in the process of morphing
the model from its initial
state to its final state.



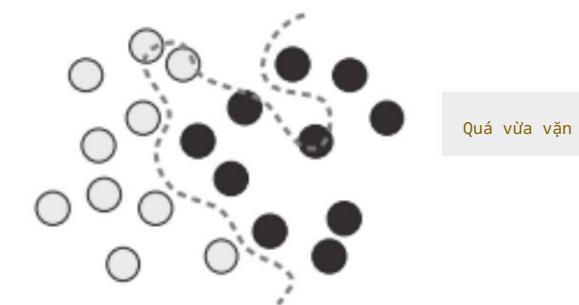
Final state: the model
overfits the training data,
reaching perfect training loss.



Test time: performance
of robustly fit model
on new data points



Test time: performance
of overfit model
on new data points



Khái quát hóa mô hình

Bí quyết 1: 'Học chăm chỉ' - thêm ngẫu nhiên nhiều vào dữ liệu huấn luyện



Khái quát hóa mô hình

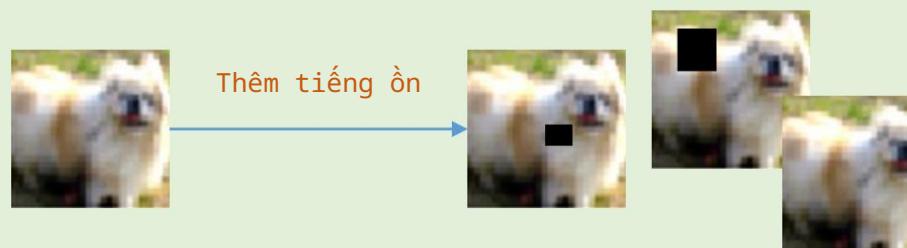
Bí quyết 1: 'Học chăm chỉ' - thêm ngẫu nhiên nhiều vào dữ liệu huấn luyện



Phát hiện biển
báo giới hạn tốc độ

Khái quát hóa mô hình

Bí quyết 1: 'Học chăm chỉ' - thêm ngẫu nhiên nhiễu vào dữ liệu huấn luyện



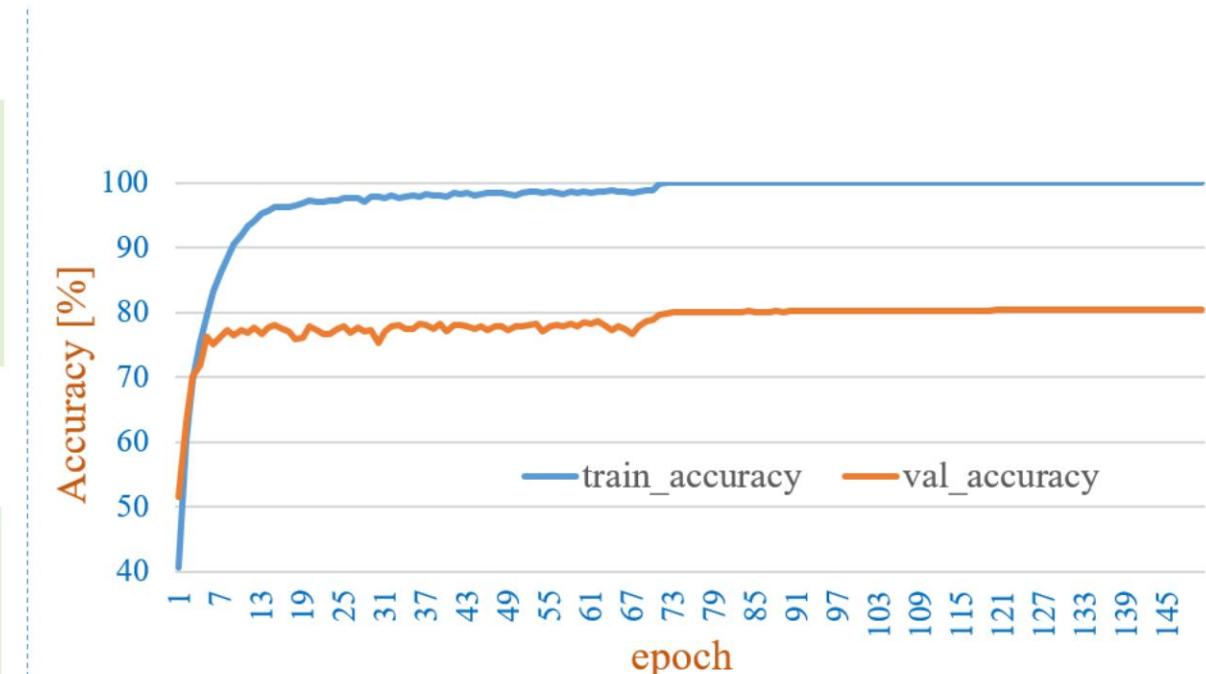
Trong PyTorch

```
RandomErasing(p=0.75, scale=(0.01, 0.3),  
               ratio=(1.0, 1.0), value=0,  
               inplace=True)
```

```
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)  
    ])  
train_set = CIFAR10(root='./data', train=True,  
                    download=True,  
                    transform=train_transform)  
trainloader = DataLoader(train_set,  
                        batch_size=256,  
                        shuffle=True,  
                        num_workers=10 )
```

Khái quát hóa mô hình

Bí quyết 1: 'Học chăm chỉ' - thêm ngẫu nhiên nhiều vào dữ liệu huấn luyện



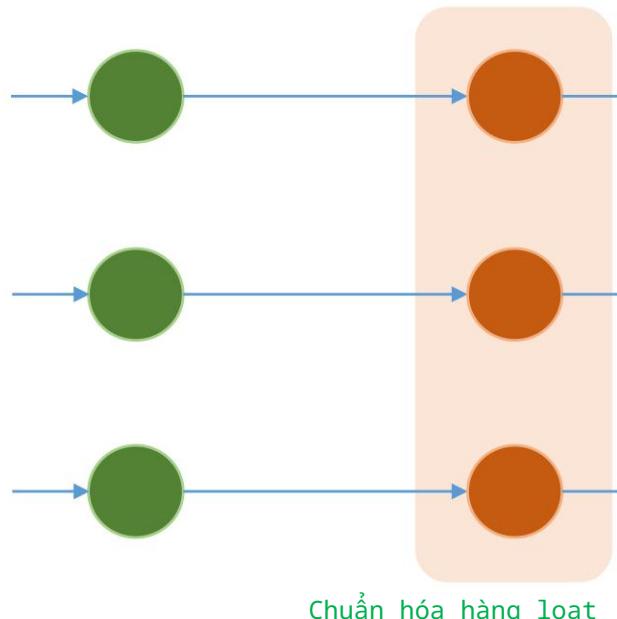
Trong PyTorch

```
RandomErasing(p=0.75, scale=(0.01, 0.3),  
               ratio=(1.0, 1.0), value=0,  
               inplace=True)
```

val_accuracy tăng
từ ~78% lên ~80%

Đào tạo mạng

Giải pháp 2: Chuẩn hóa hàng loạt



Không cần thiên vị khi sử dụng BN

và được cập nhật trong chuyển tiếp
và β được cập nhật theo hướng ngược lại

Nhập dữ liệu cho một nút trong lớp chuẩn hóa hàng loạt

$$= \{ \quad , \quad \}$$

là kích thước lô nhỏ

Tính giá trị trung bình và phuơng sai

$$= \frac{1}{n} \sum_{i=1}^n x_i = \bar{x}$$

$$\text{Phuơng sai} = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2 = s^2$$

Bình thường hóa

$$= \frac{x - \bar{x}}{s}$$

là một giá trị rất nhỏ

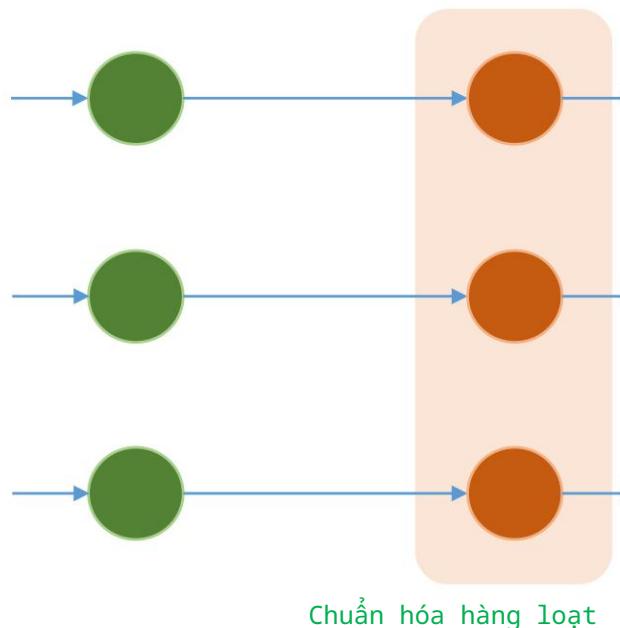
Quy mô và sự thay đổi

$$= \gamma x + b$$

và β là hai tham số học tập

Đào tạo mạng

Thủ thuật 2: Chuẩn hóa hàng loạt



Chuyện gì xảy ra nếu

$$\sigma = \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2} \text{ và } \beta = \frac{1}{n} \sum_{i=1}^n x_i$$

Nhập dữ liệu cho một nút trong lớp chuẩn hóa hàng loạt

$$= \{x_1, \dots, x_n\}$$

là kích thước lô nhỏ

Tính giá trị trung bình và phuơng sai

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i = \frac{1}{n} (x_1 + x_2 + \dots + x_n)$$

Bình thường hóa

$$\hat{x} = \frac{x - \bar{x}}{\sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2}}$$

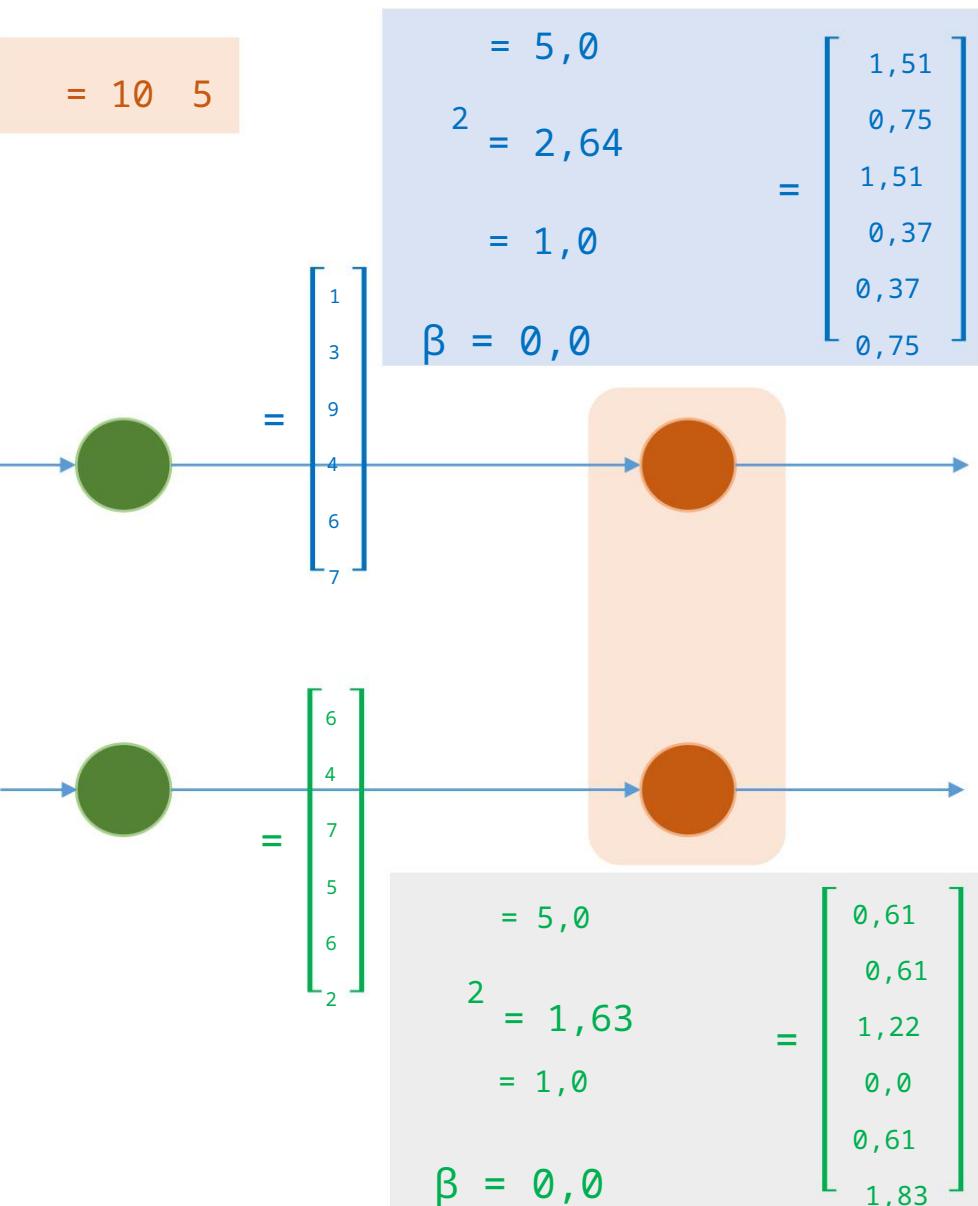
là một giá trị rất nhỏ

Quy mô và sự thay đổi

$$\hat{x} = \frac{x - \bar{x}}{\sigma} + \beta$$

và β là hai tham số học tập

Đào tạo mạng



Thủ thuật 2: Chuẩn hóa hàng loạt

Nhập dữ liệu cho một nút trong lớp chuẩn hóa hàng loạt

$$= \{ \quad _1, \dots, \quad \}$$

là kích thước lô nhỏ

Tính giá trị trung bình và phuơng sai

$$= \frac{1}{n} \quad 2 = \frac{1}{n} \quad (\quad)^2$$

$= 1 \quad = 1$

Bình thường hóa

$$= \frac{\overline{x}}{\sqrt{s^2}}$$

là một giá trị rất nhỏ

Quy mô và sự thay đổi

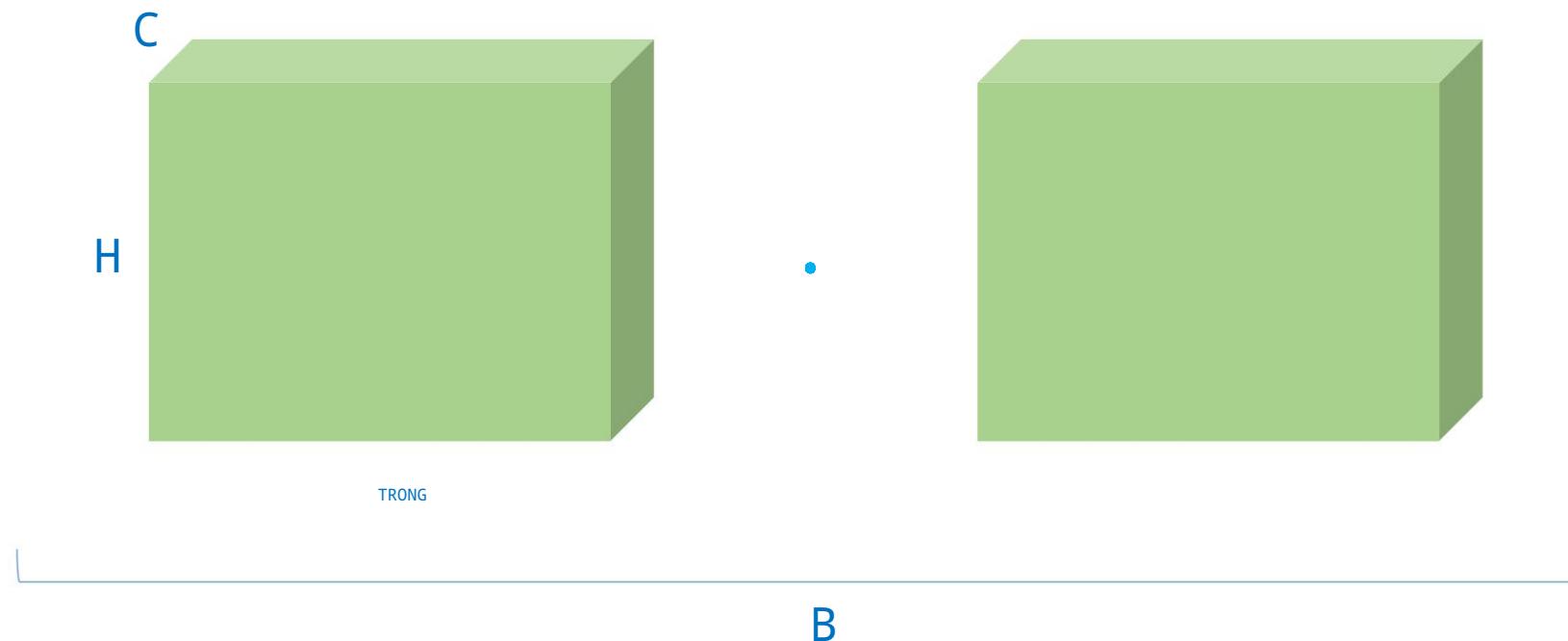
$$= \quad + b$$

và β là hai tham số học tập

và β được cập nhật trong quá trình huấn luyện

Đào tạo mạng

Thủ thuật 2: Chuẩn hóa hàng loạt cho dữ liệu 2D



Tính C có nghĩa là giá trị $H \times W \times B$

Tính toán phuơng sai C của giá trị $H \times W \times B$

Đào tạo mạng

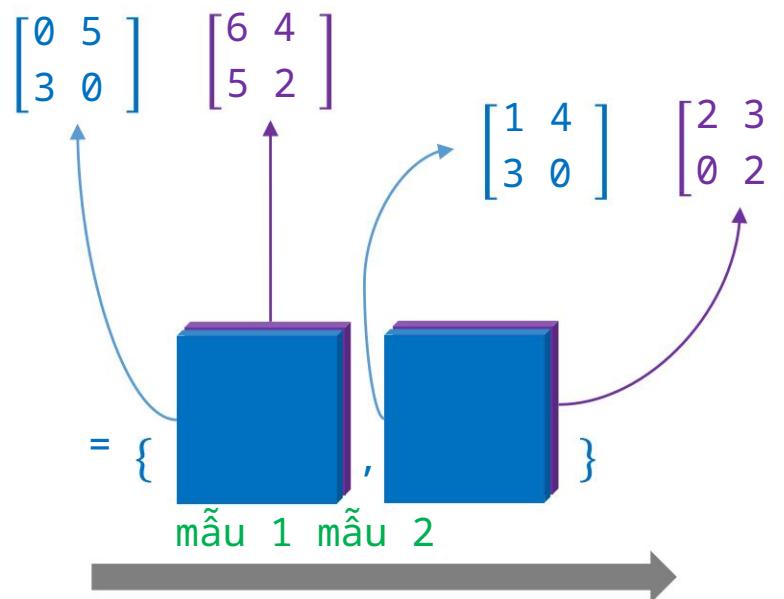
= 10 5

= [2.0, 3.0]

$^2 = [4.0, 3.7]$

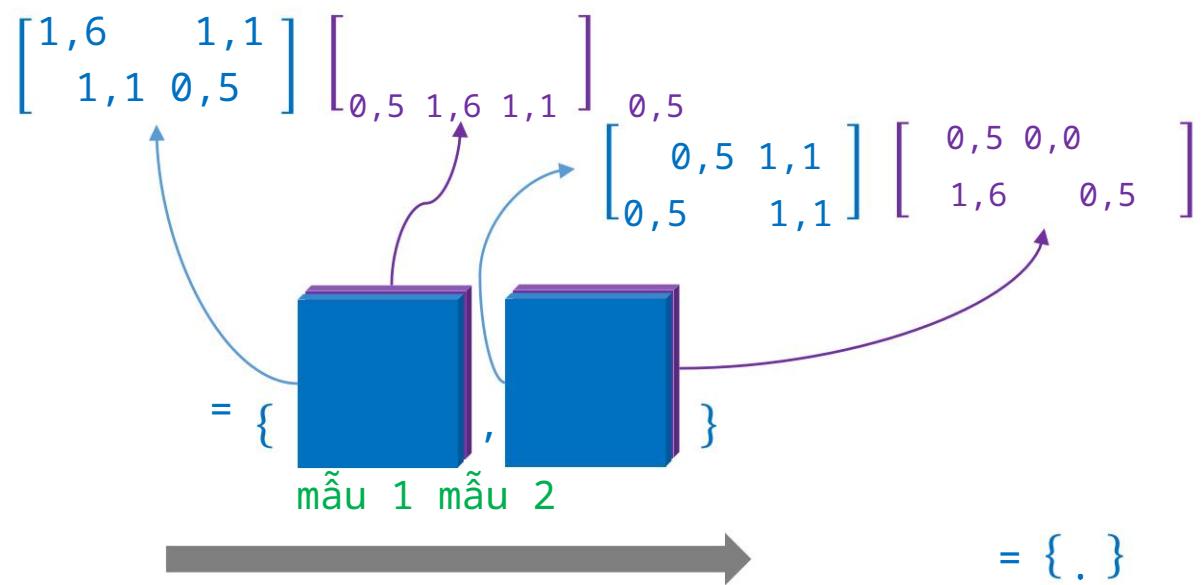
= 1,0

$\beta = 0,0$



cỡ lô = 2

sample_shape = (2, 2, 2)



= { . }

Đào tạo mạng

Thủ thuật 2: Chuẩn hóa hàng loạt

Nhập dữ liệu cho một nút trong lớp chuẩn hóa hàng loạt

$$= \{ \dots \}$$

là kích thước lô nhỏ

Tính giá trị trung bình và phu ứng sai

$$= \frac{1}{n} \quad \bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$

Bình thường hóa

$$= \frac{x - \bar{x}}{\sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2}}$$

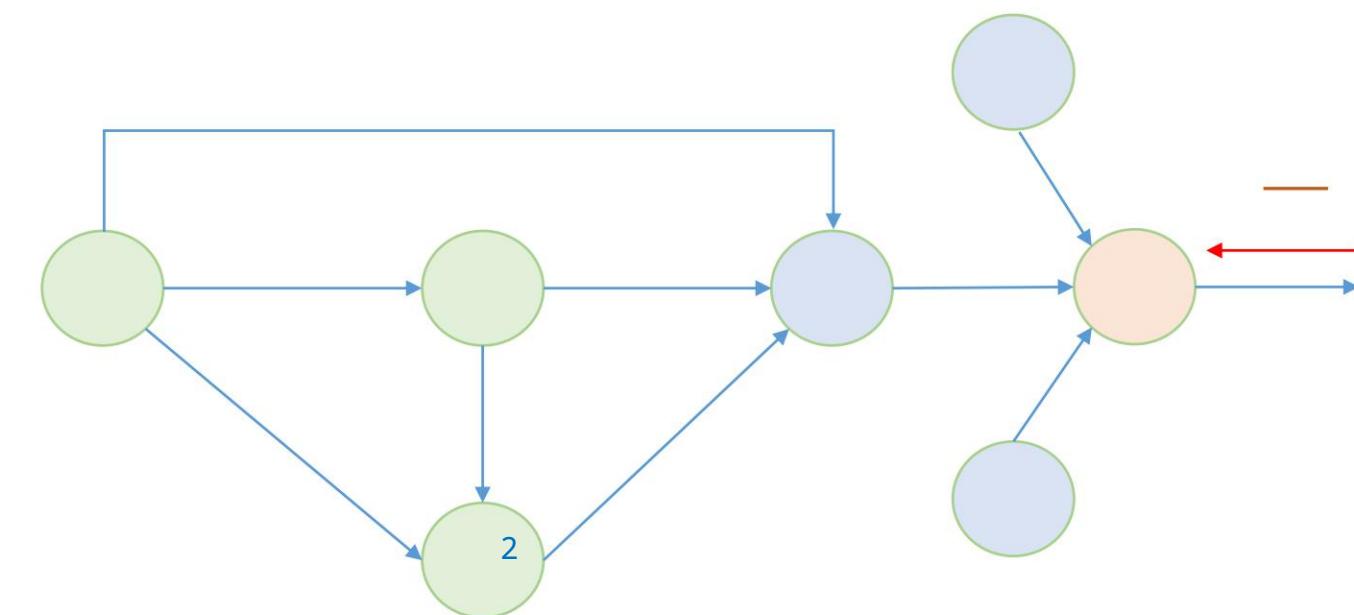
là một giá trị rất nhỏ

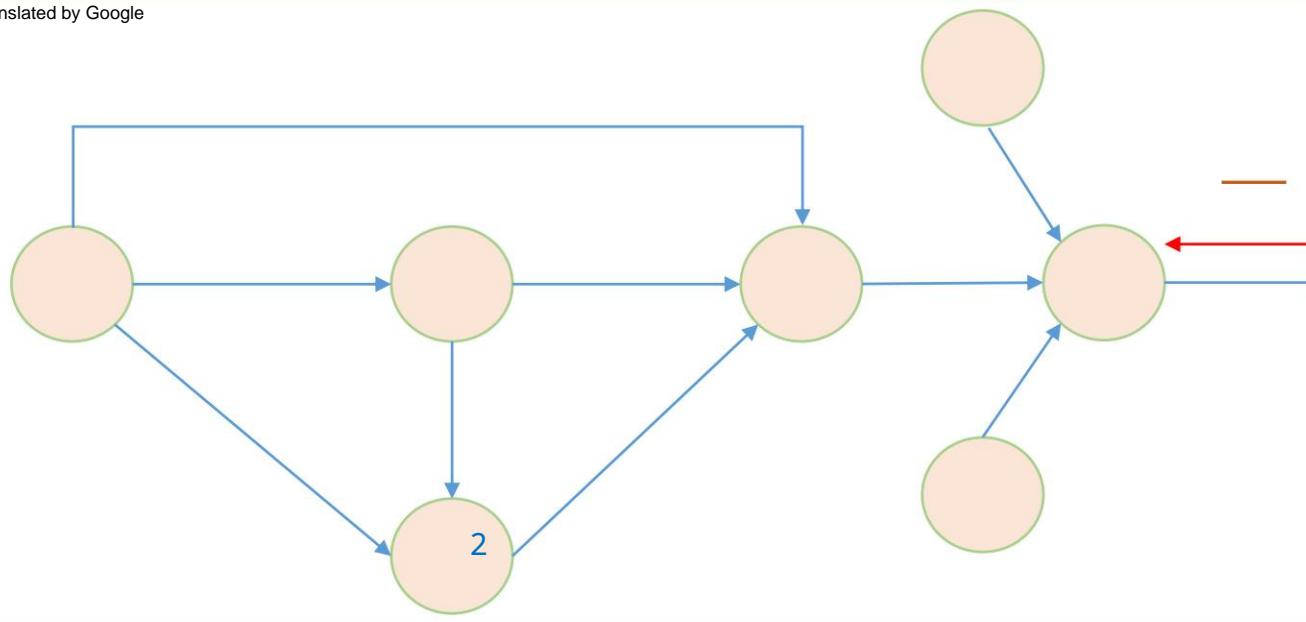
Quy mô và sự thay đổi

$$= \alpha x + b$$

và β là hai tham số học tập

Phía sau





$$= \frac{1}{\sqrt{2}}$$

= 1

$$2 = \frac{1}{\sqrt{2}} ()^2$$

= 1

$$= \frac{1}{\sqrt{2} + }$$

+ b

$$= \frac{1}{\sqrt{2} + }$$

$$b = \frac{1}{\sqrt{2} + }$$

$$= \frac{1}{\sqrt{2} + }$$

$$= \frac{1}{\sqrt{2} + } + \frac{1}{\sqrt{2} + } + \frac{1}{\sqrt{2} + }$$

$$\frac{1}{\sqrt{2}} = \frac{1}{\sqrt{2} + } = \frac{1}{\sqrt{2} + } - \frac{1}{\sqrt{2} + } = \frac{1}{\sqrt{2} + } \left(\frac{\sqrt{2} - \sqrt{2}}{\sqrt{2} - \sqrt{2}} \right) = \frac{1}{\sqrt{2} + } \left(1 - \frac{1}{\sqrt{2} + } \right)$$

$$= \frac{1}{\sqrt{2} + }$$

$$= \frac{2}{\sqrt{2} + }$$

$$= \frac{1}{\sqrt{2} + }$$

$$= \frac{1}{\sqrt{2} + } - \frac{1}{\sqrt{2} + } = \frac{1}{\sqrt{2} + } \left(1 - \frac{1}{\sqrt{2} + } \right)$$

Khái quát hóa mô hình

Thủ thuật 2: Chuẩn hóa hàng loạt



đợt nhỏ 1



đợt nhỏ thứ 2

$$(1, 1) \quad (2, 2)$$

rất

rất có thể



Thêm nhiều vào đầu ra của lớp BN

Nhập dữ liệu cho một nút trong lớp chuẩn hóa hàng loạt

$$= \{1, \dots\}$$

là kích thước lô nhỏ

Tính giá trị trung bình và phuơng sai

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i = \frac{1}{n} (x_1 + x_2 + \dots + x_n)$$

Bình thường hóa

$$z = \frac{x - \bar{x}}{\sigma} = \frac{x - \bar{x}}{\sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2}}$$

là một giá trị rất nhỏ

Quy mô và sự thay đổi

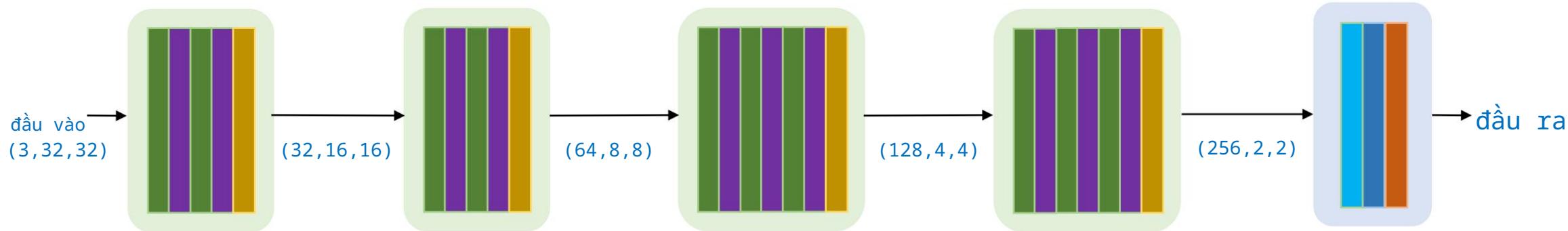
$$z = \frac{x - \bar{x}}{\sigma} + b$$

và b là hai tham số học tập

Khái quát hóa mô hình

```
conv_layer = nn.Sequential(nn.Conv2d(3, 32, 3,
padding=1),
nn.ReLU(),
nn.BatchNorm2d(32))
```

Thủ thuật 2: Chuẩn hóa hàng loạt



(3x3) Tích chập
đệm='giống nhau'
sải bù 0c=1 + ReLU

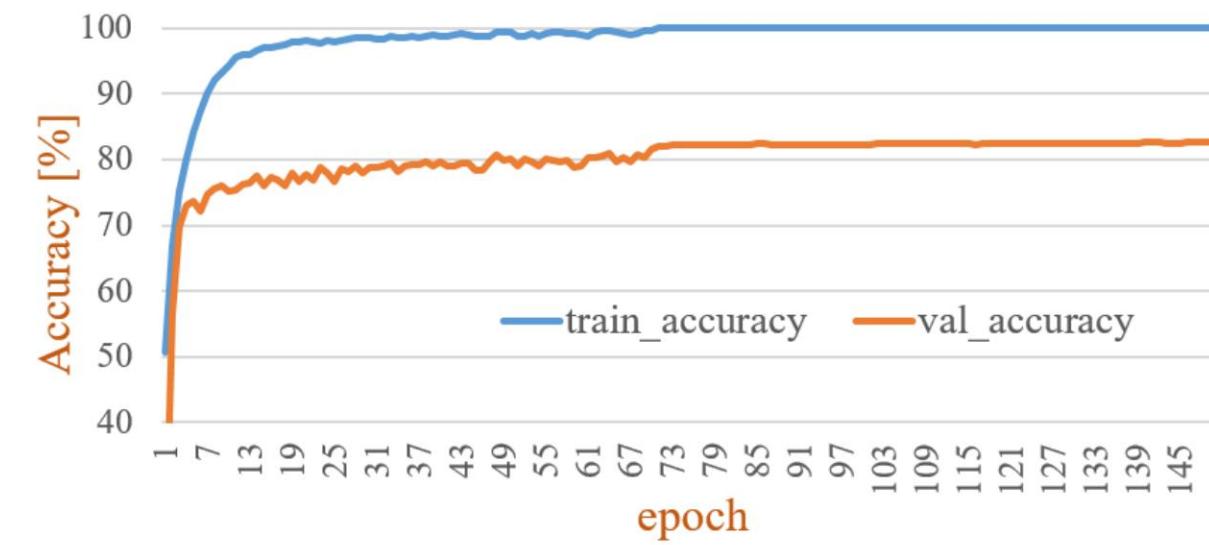
Làm phẳng

(2x2) tổng hợp tối đa

Lớp dày đặc-512
+ ReLU

Chuẩn
hóa hàng loạt

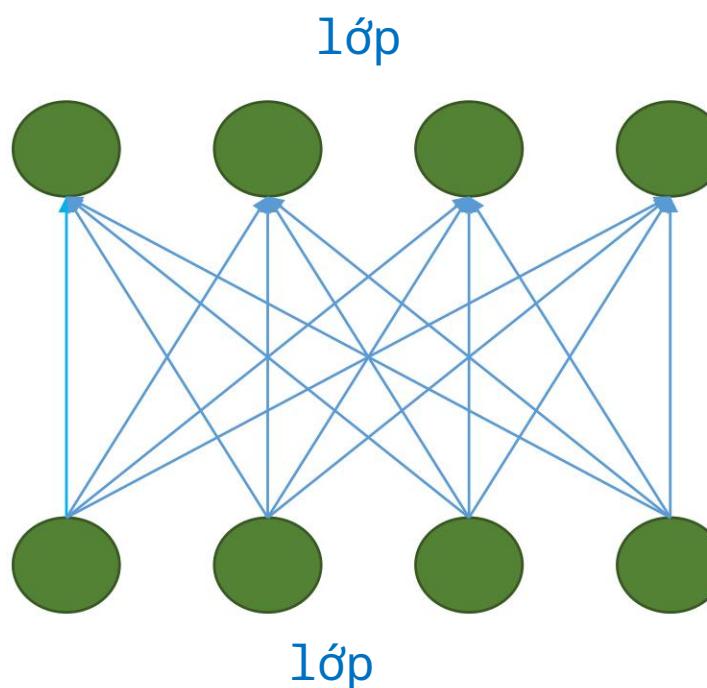
Lớp dày đặc-10
+ Softmax



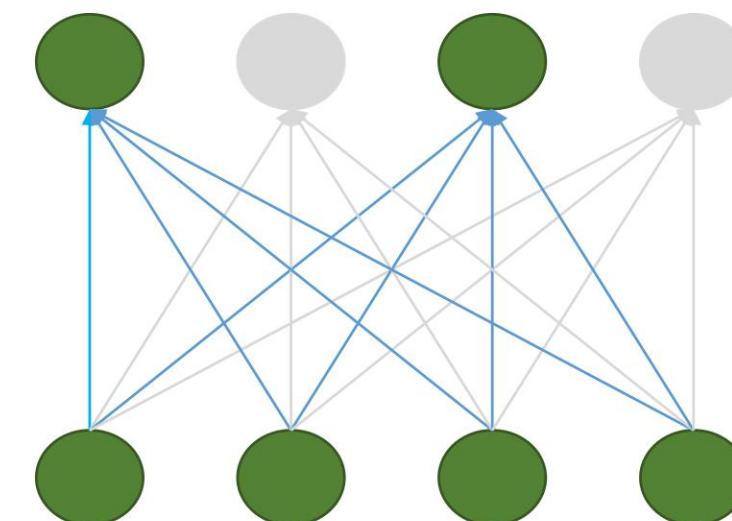
val_accuracy tăng từ ~80,9% lên ~84%

Khái quát hóa mô hình

Bí quyết 3: Bỏ học



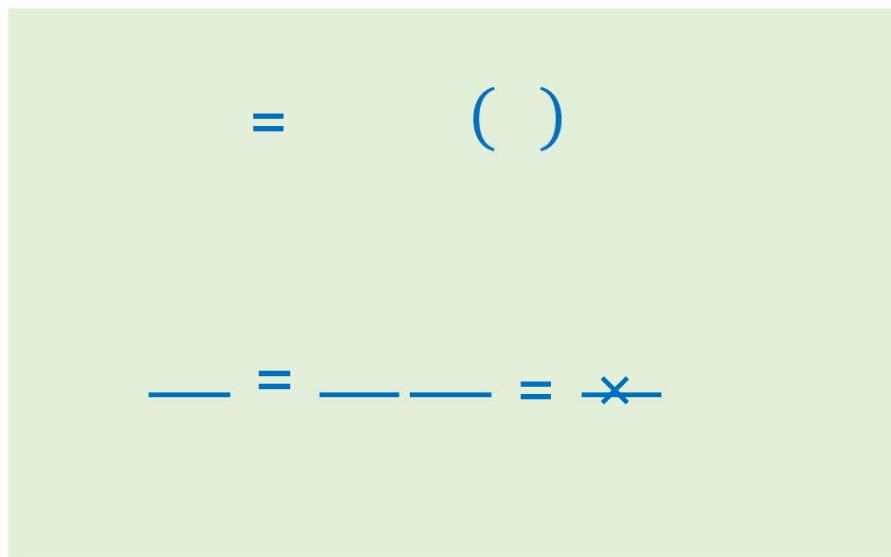
Áp dụng mức bỏ học 50% cho lớp



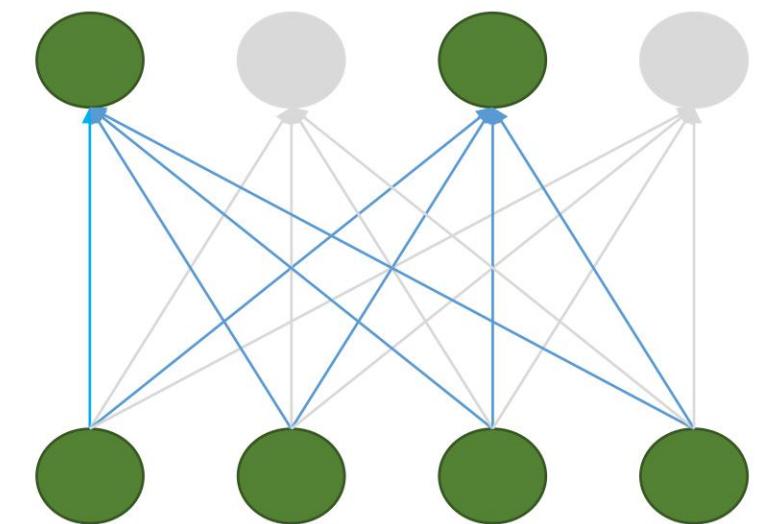
~50% nút đư ợc chọn ngẫu nhiên trong lớp đư ợc
đặt thành 0 (loại thêm nhiễu)

Khái quát hóa mô hình

Bí quyết 3: Bỏ học



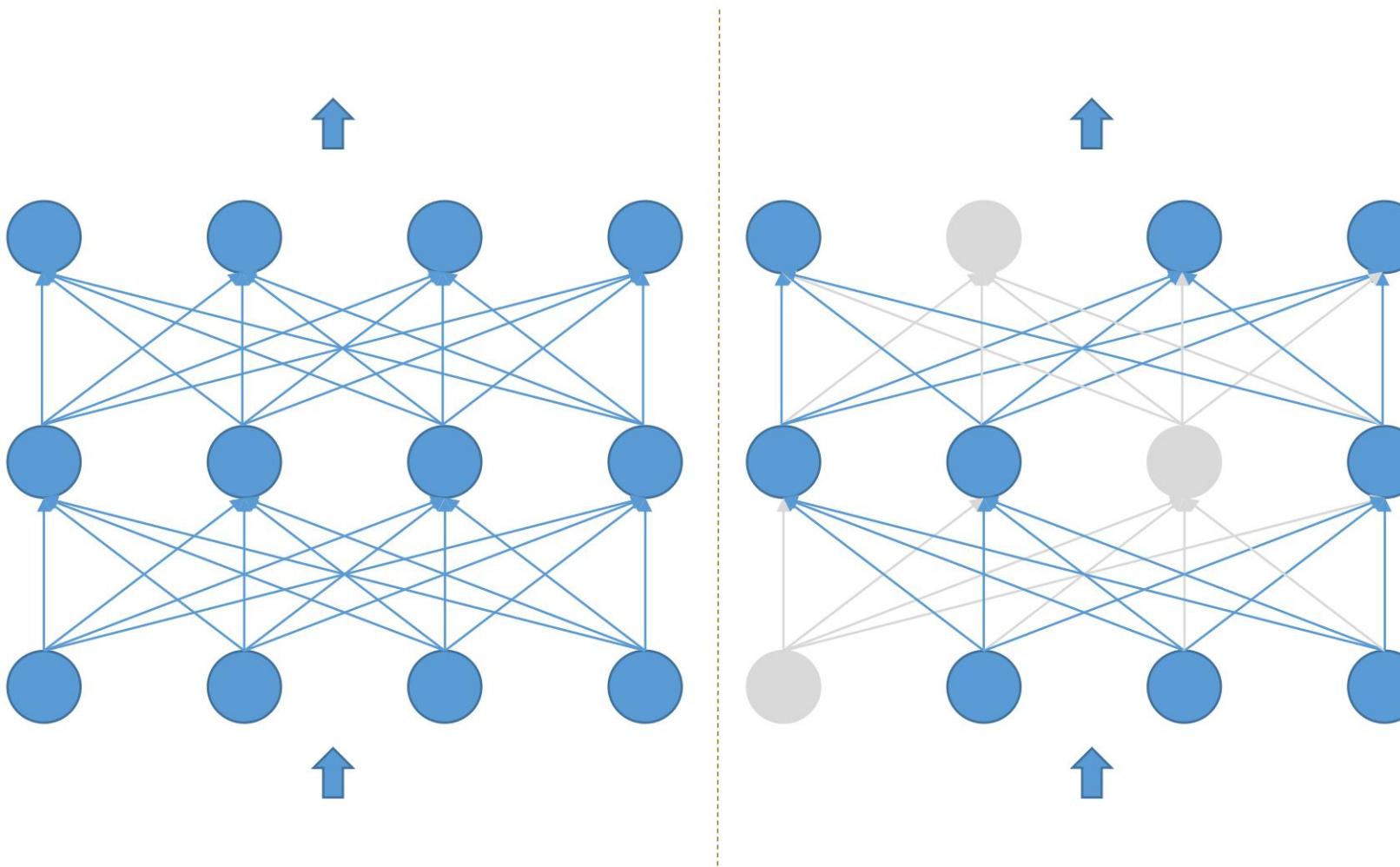
Áp dụng mức bỏ học 50% cho lớp



~50% nút đư ợc chọn ngẫu nhiên trong
các h lớp đư ợc đặt thành số không

Trang bị quá mức

Rơi ra ngoài



Với tốc độ giảm r

Đặt ngẫu nhiên các đơn vị đầu vào
về 0 với tần số r

Chỉ áp dụng trong chế độ đào tạo

```
nn.Sequential(nn.Conv2d(32, 32, 3,  
padding=1),  
nn.ReLU(),  
nn.BatchNorm2d(32),  
nn.MaxPool2d(2, 2),  
nn.Dropout(dropout_rate))
```

$$= \frac{1}{1}$$

Khái quát hóa mô hình

Bí quyết 3: Bỏ học

```
class Dropout():

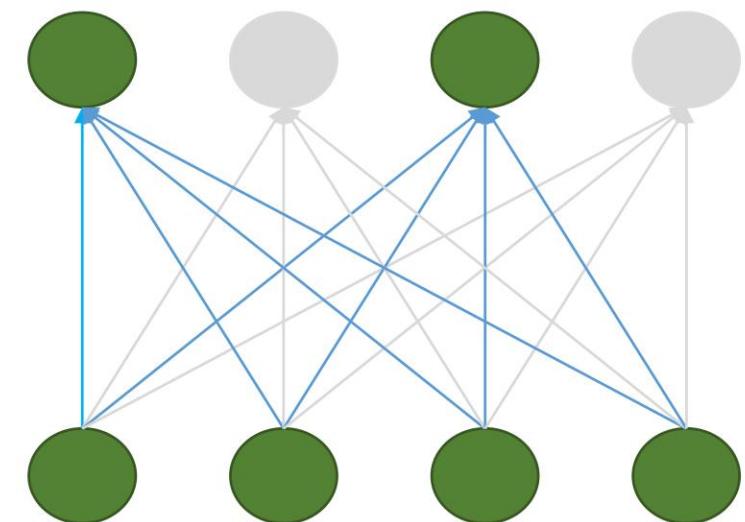
    def __init__(self, prob=0.5):
        self.prob = prob
        self.params = []

    def forward(self, X):
        self.mask = np.random.binomial(1, self.prob, size=X.shape) / self.prob
        out = X * self.mask
        return out.reshape(X.shape)

    def backward(self, dout):
        dX = dout * self.mask
        return dX, []
```

<https://deepnotes.io/dropout>

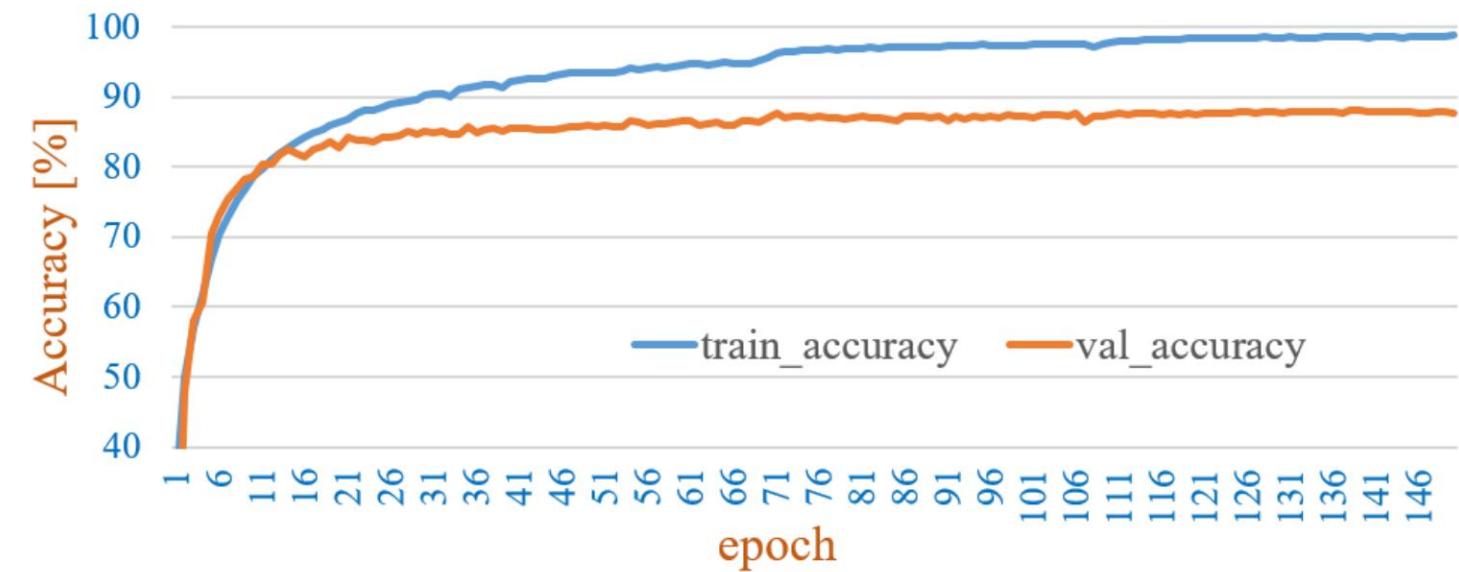
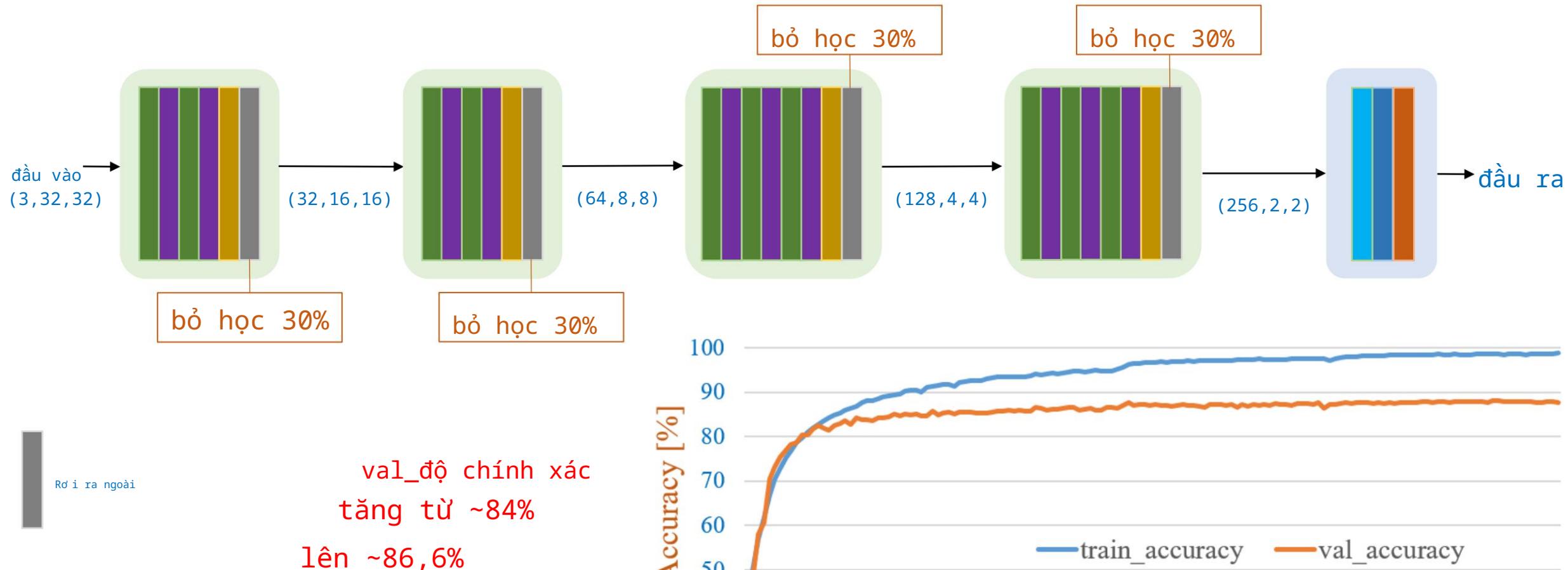
Áp dụng mức bỏ học 50% cho lớp



~50% nút đư ợc chọn ngẫu nhiên trong lớp
các h đư ợc đặt thành số không

Khái quát hóa mô hình

Bí quyết 3: Bỏ học



Khái quát hóa mô hình

Thủ thuật 4: Chính quy hóa kernel

 $=$ $+ \parallel \parallel^2$ 

2chính quy hóa

Ngăn mạng tập trung
vào các tính năng cụ thể

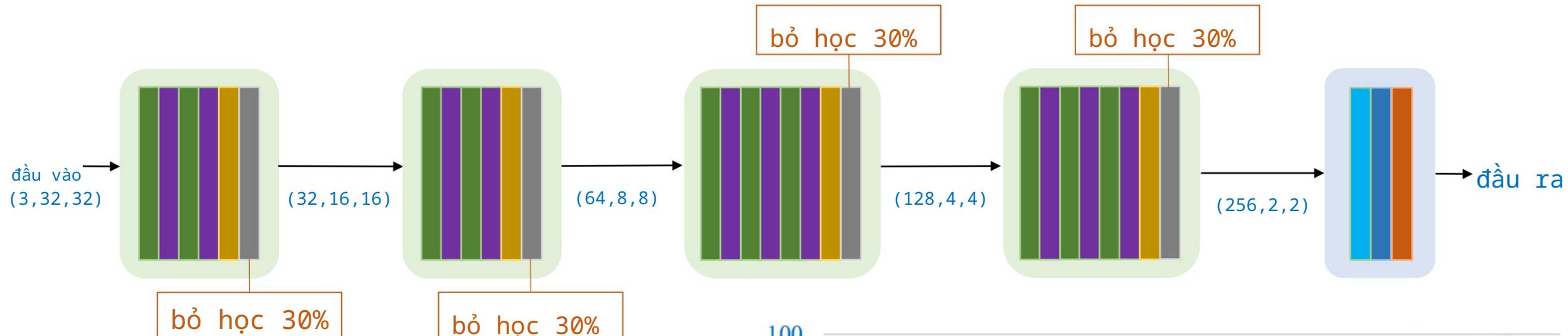
Trọng lượng nhỏ hơn
mô hình đơn giản hơn

Trong PyTorch

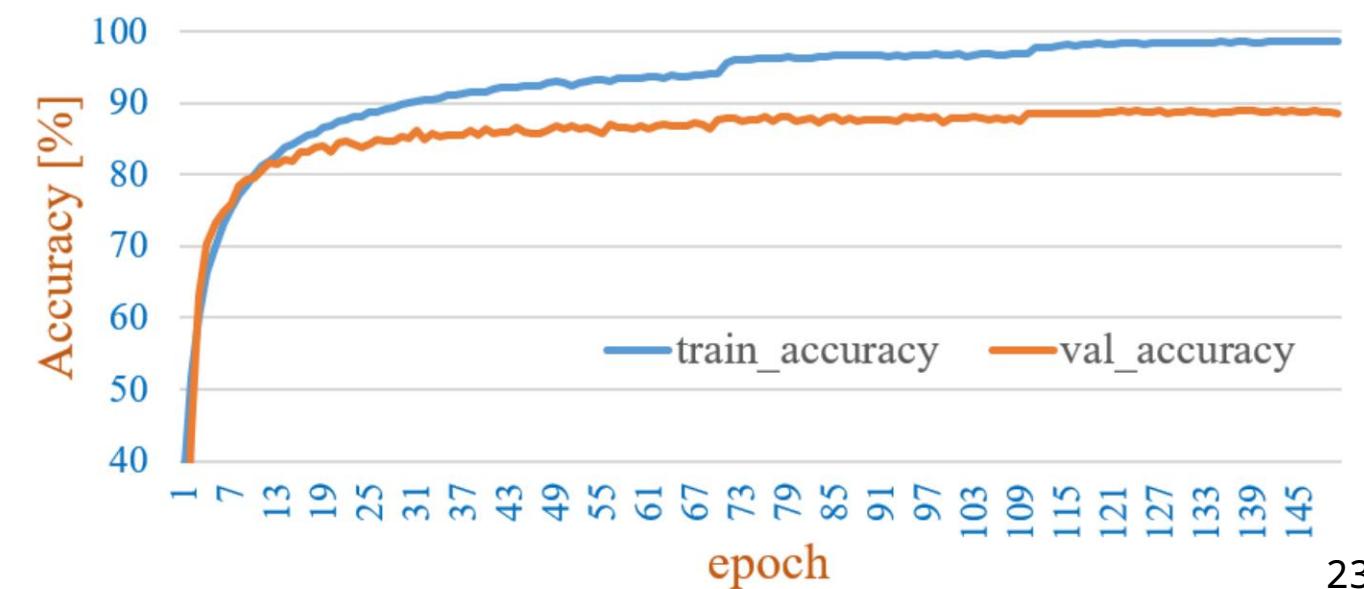
```
optimizer = Adam(model.parameters(),  
                 lr=1e-3,  
                 weight_decay=5e-4)
```

Khái quát hóa mô hình

Thủ thuật 4: Trình chỉnh sửa kernel



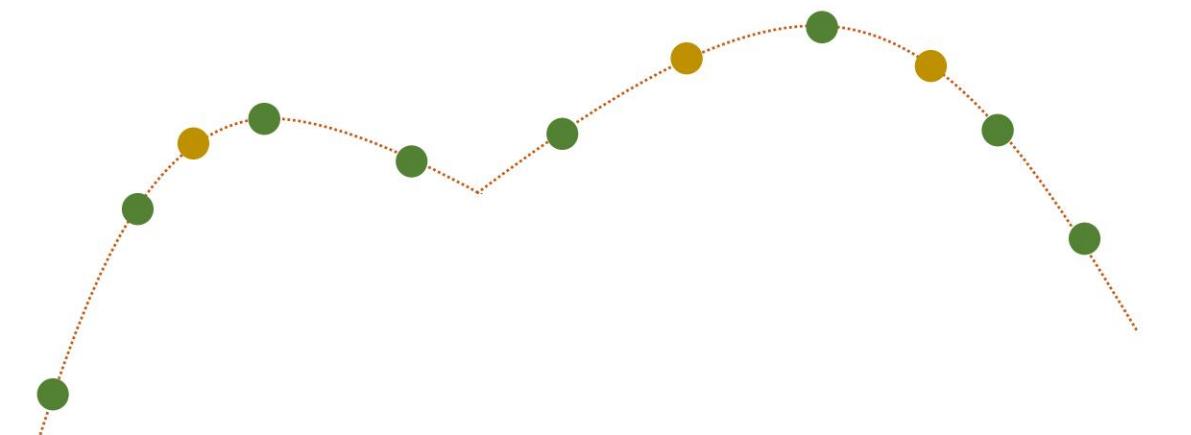
val_độ chính xác
tăng từ ~86,6%
lên ~87,5%



Khái quát hóa mô hình

Thủ thuật 5: Tăng cường dữ liệu

Một trung hợp bình thường



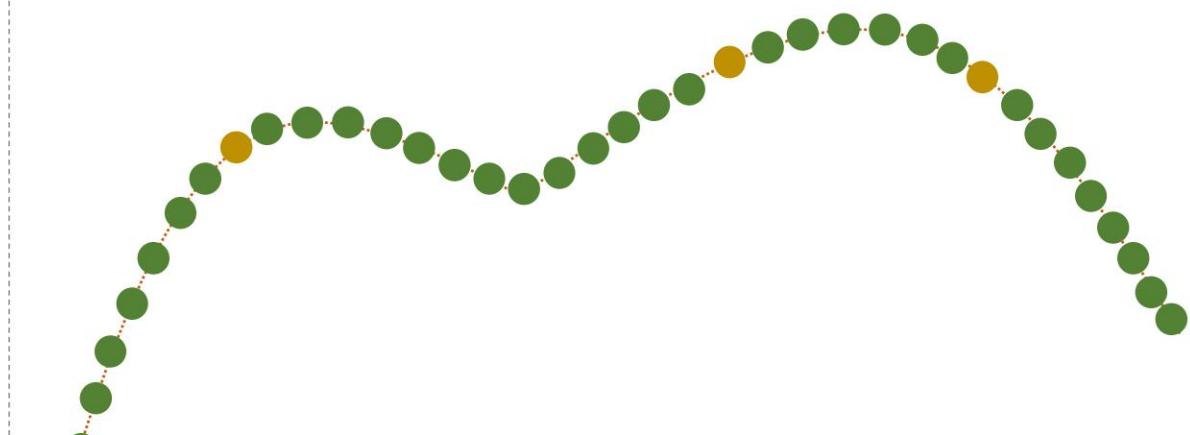
Hình ảnh

..... Phân phối dữ liệu

..... Dữ liệu thử nghiệm

● Dữ liệu đào tạo

Một trung hợp hoàn hảo: Được đào tạo không giới hạn



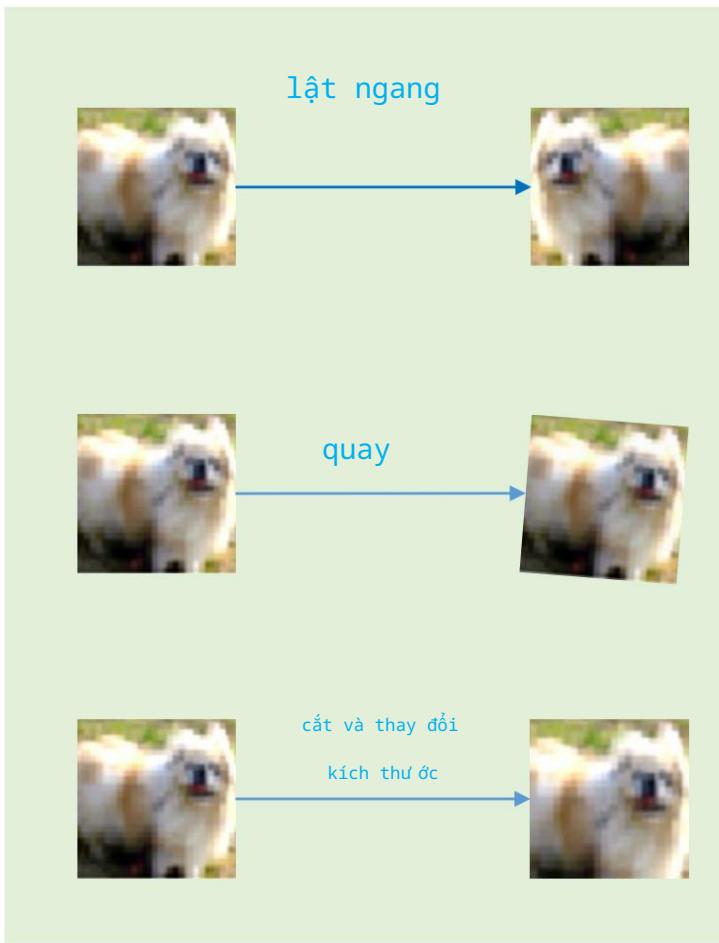
Hình ảnh

Dữ liệu đào tạo bao gồm toàn bộ phân phối

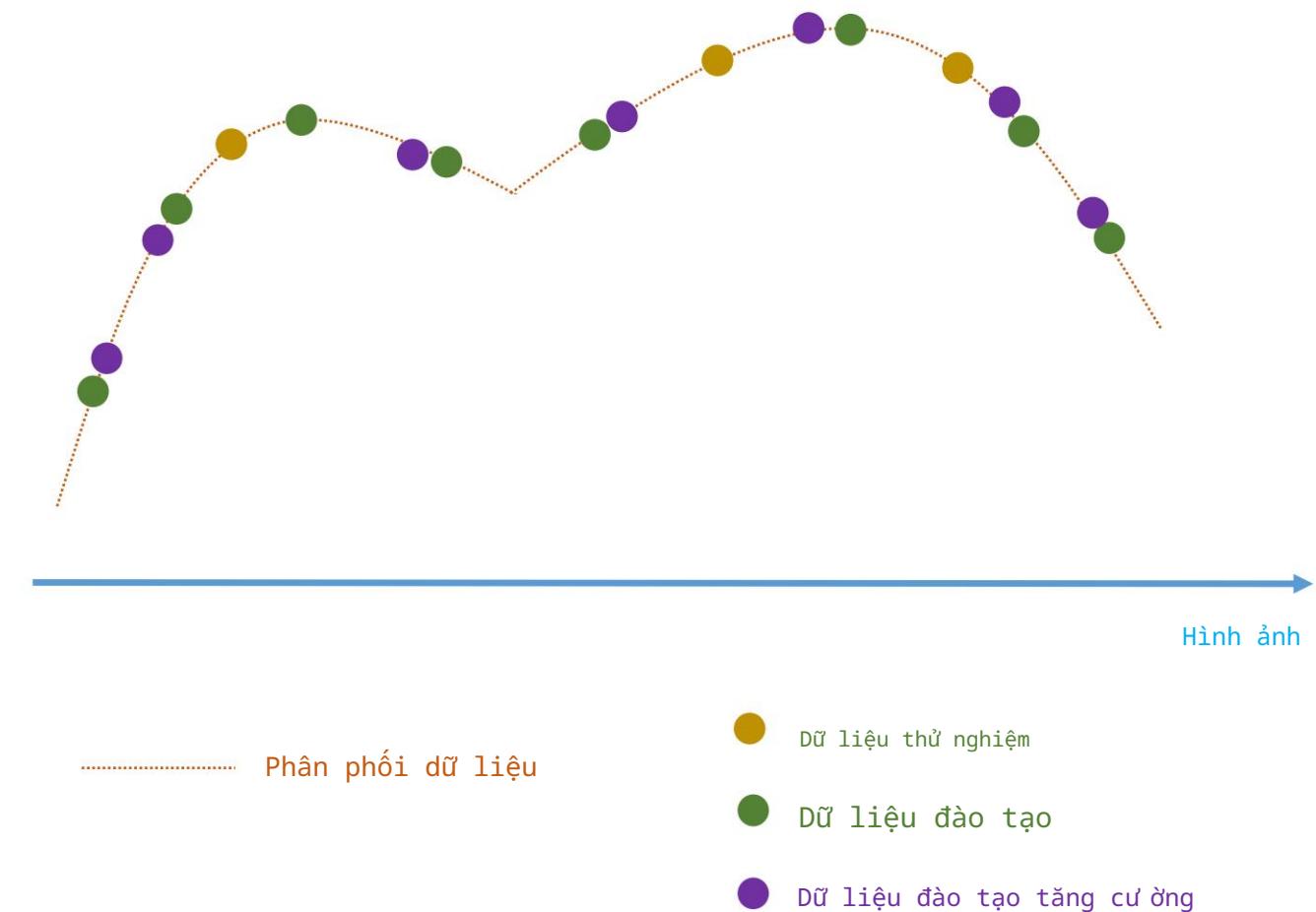
Như ng, không thực tế!!!

Khái quát hóa mô hình

Thủ thuật 5: Tăng cường dữ liệu



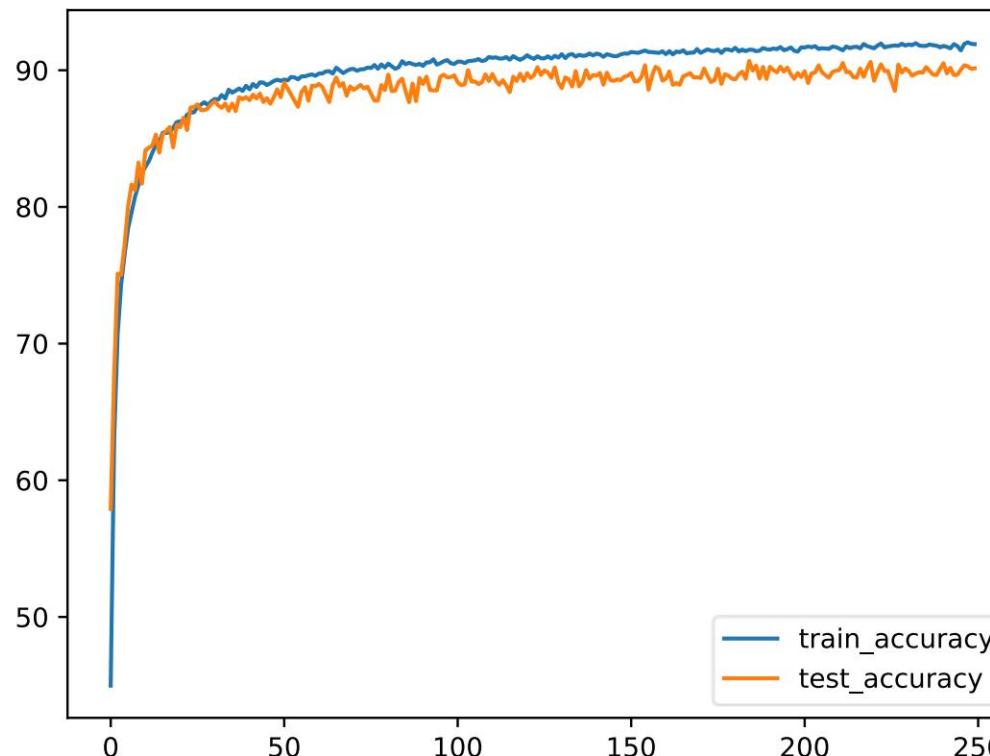
Tăng dữ liệu bằng cách thay đổi dữ liệu đào tạo



Khái quát hóa mô hình

Thủ thuật 5: Tăng cường dữ liệu

Lật ngang + cắt và thay đổi kích thước



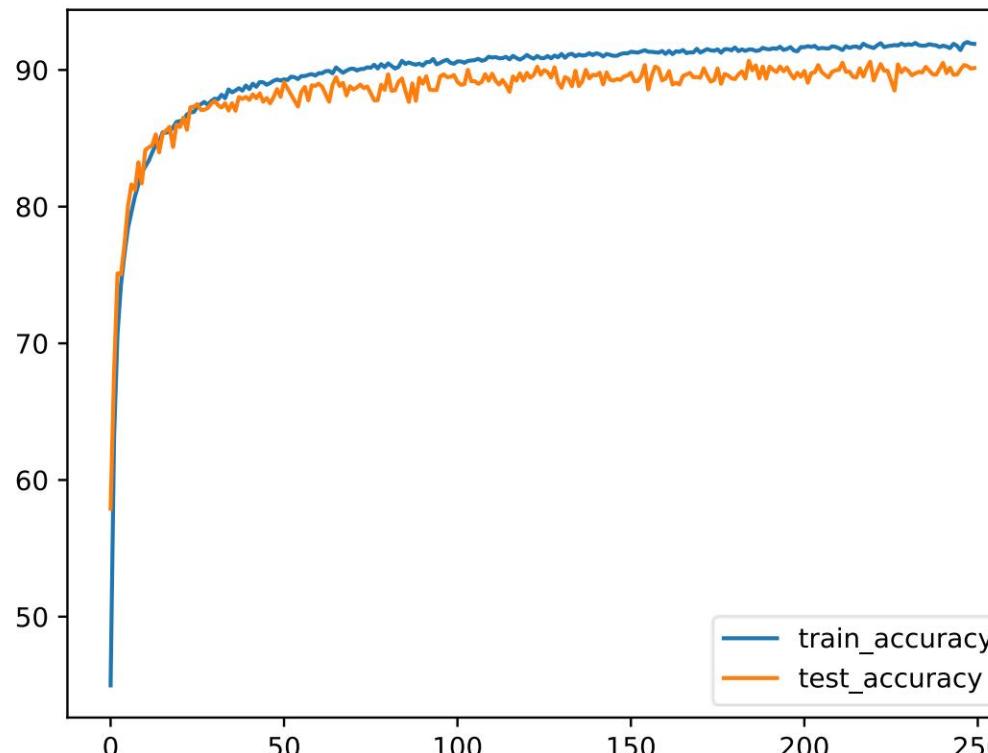
```
train_transform = transforms.Compose([
    transforms.RandomCrop(32, padding=2),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.4914, 0.4821, 0.4465],
                        std=[0.2471, 0.2435, 0.2616])
])
```

val_accuracy đạt tới ~90,6%

Khái quát hóa mô hình

Những gì chúng tôi có

Lật ngang + cắt và thay đổi kích thước



val_accuracy đạt ~90,6%

train_accuracy đạt ~92%

Chuẩn hóa hàng loạt

Rời ra ngoài

Chính quy hóa hạt nhân

Tăng cường dữ liệu

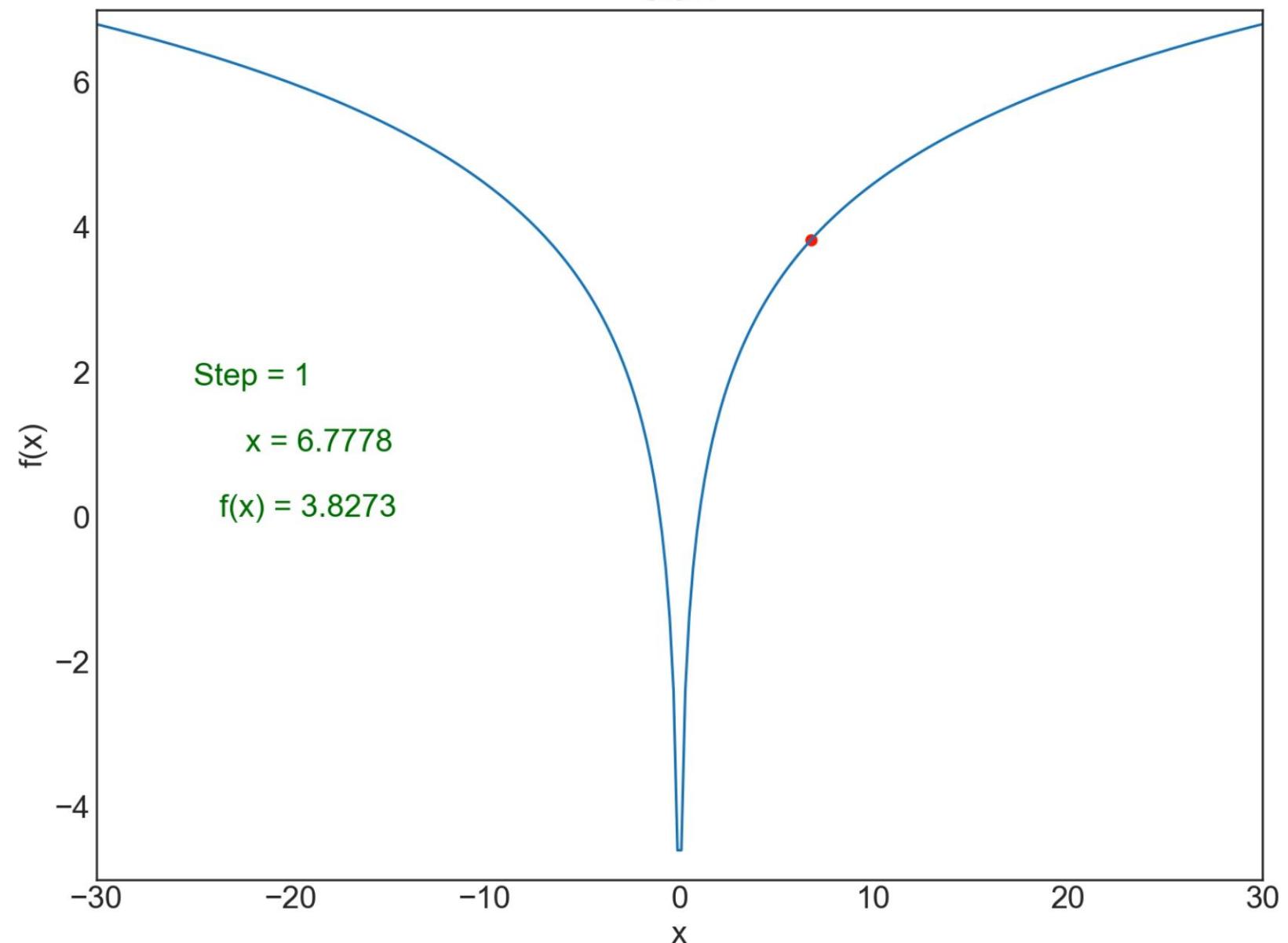
Ý tưởng: cố gắng tăng train_accuracy, mong đợi
val_accuracy cũng tăng

→ Tăng công suất mô hình

Tối ưu hóa

Tỷ lệ học tập

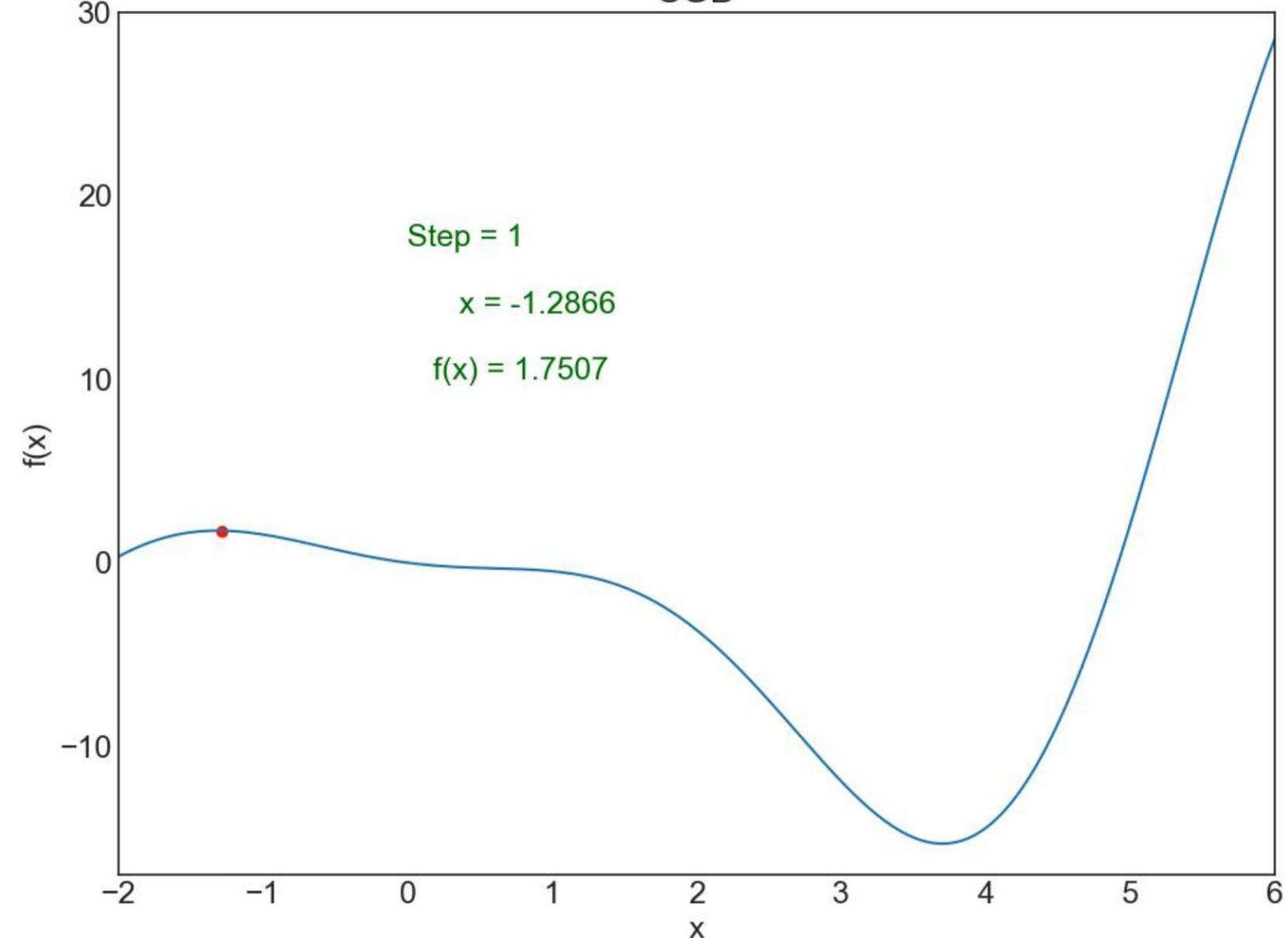
SGD



Tối ưu hóa

Tỷ lệ học tập

SGD



Khái quát hóa mô hình

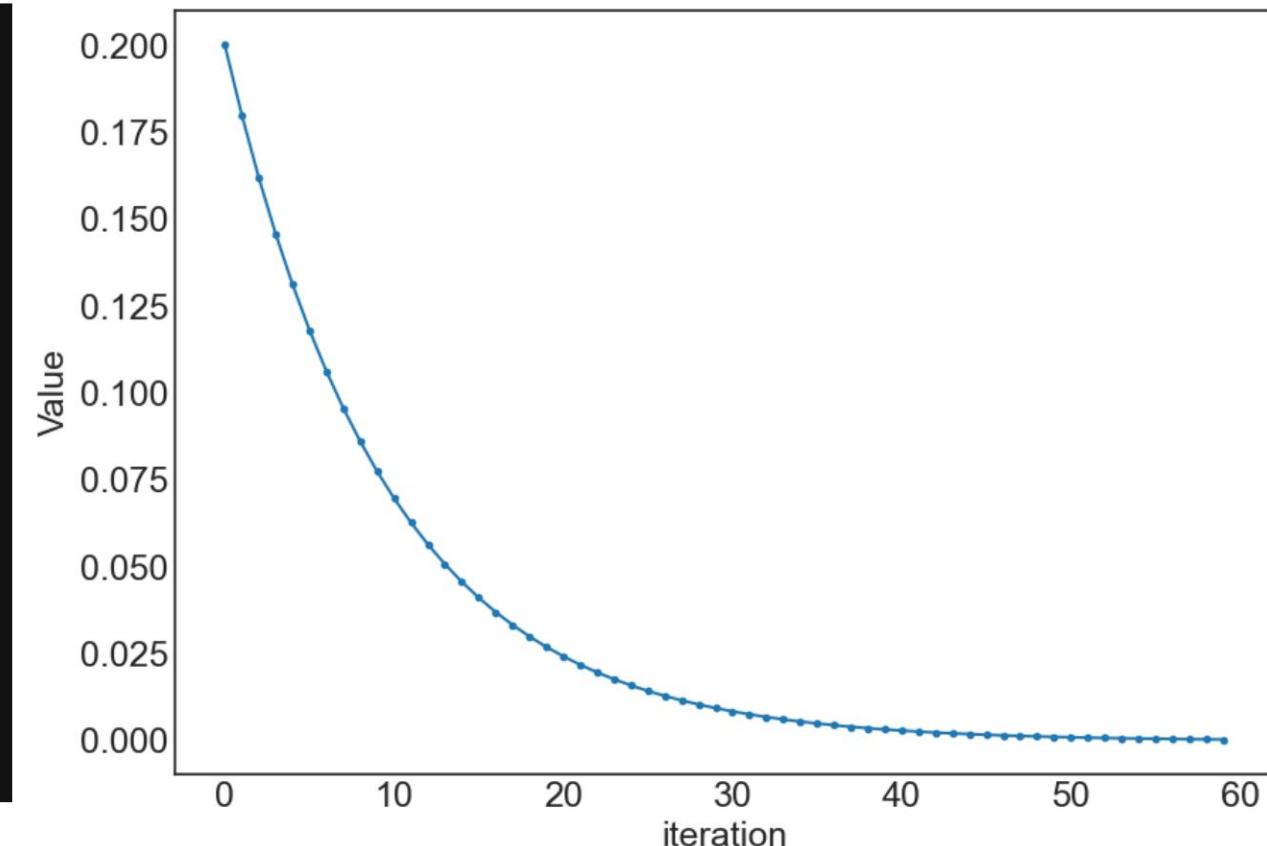
Bí quyết 6: Giảm tốc độ học

$$= \emptyset \times h$$

```
lr_scheduler.ExponentialLR(optimizer=optimizer,
                            gamma=0.96)
```

```
# train
for epoch in range(max_epoch):
    # ...
    for i, (inputs, labels) in enumerate(trainloader):
        # ...

    #...
    # update Learning rate
    lr_scheduler.step()
```

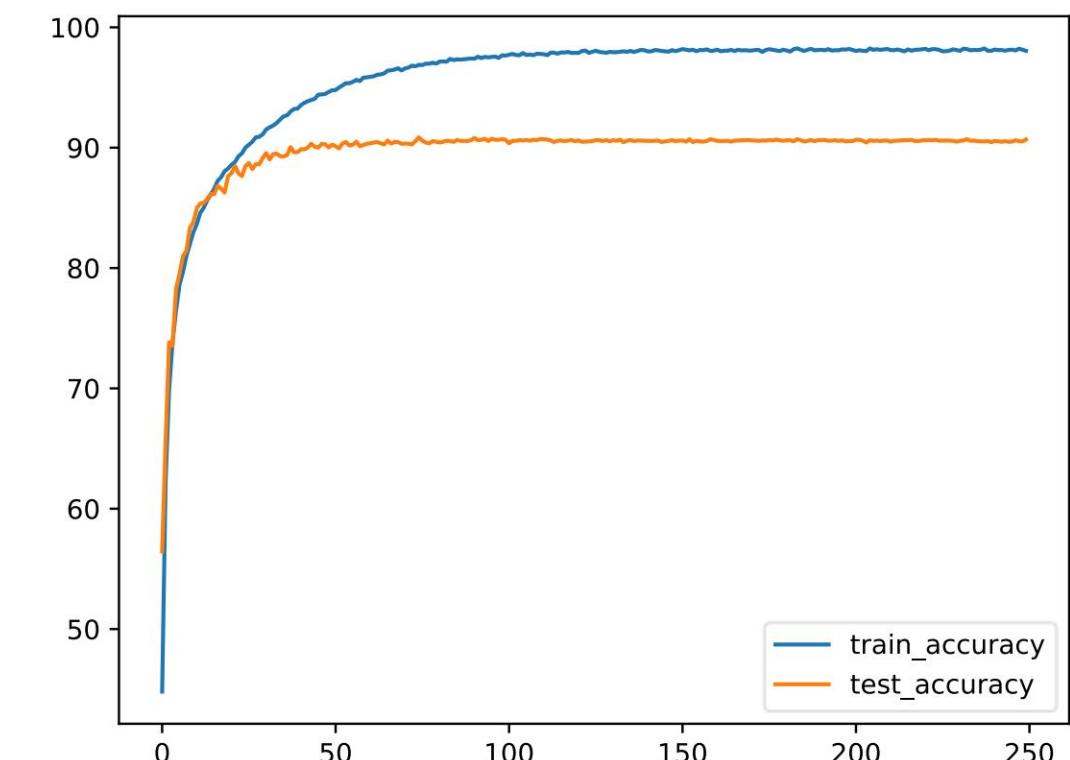
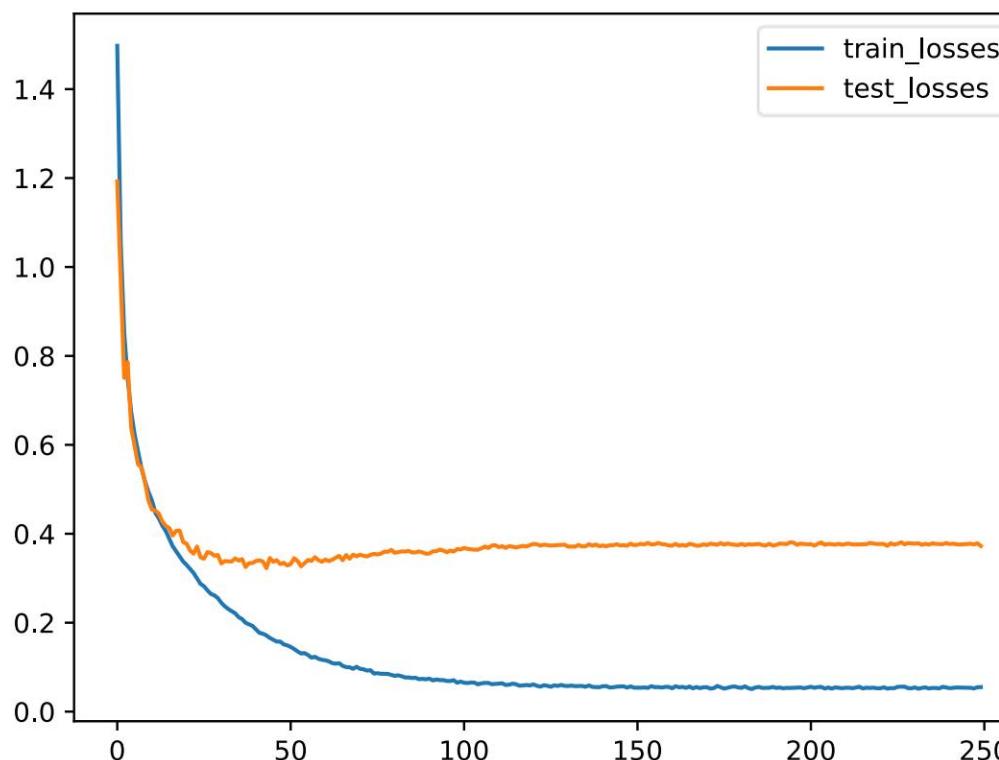


Khái quát hóa mô hình

Bí quyết 6: Giảm tốc độ học

val_accuracy đạt ~90,6%

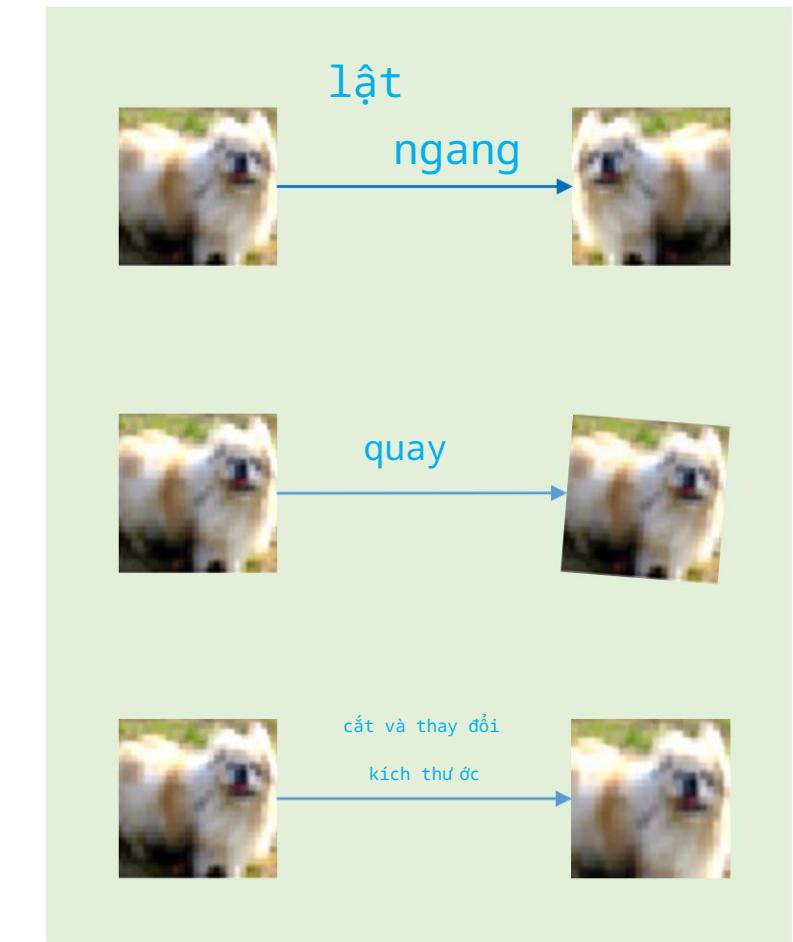
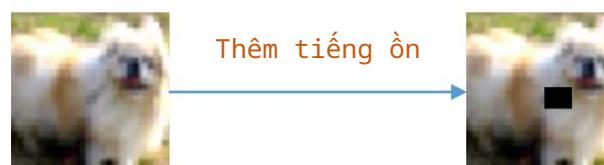
train_accuracy đạt ~98%



Khái quát hóa mô hình

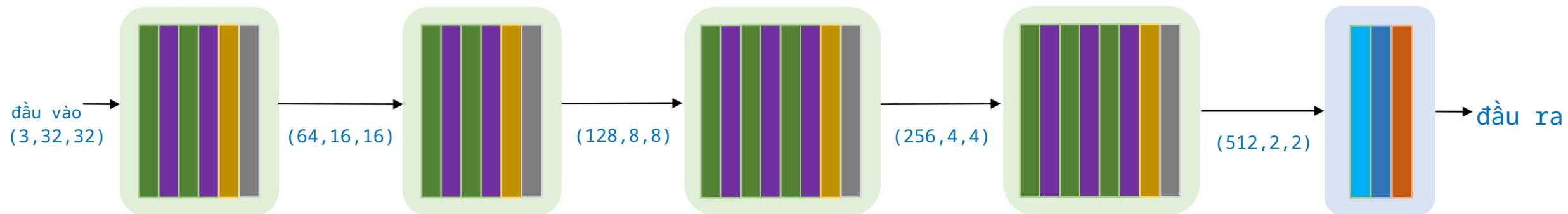
Thảo luận: Dự đoán độ chính xác của việc huấn luyện và kiểm tra khi sử dụng nhiều dữ liệu tăng cường hơn

```
train_transform = transforms.Compose([
    transforms.RandomCrop(32, padding=2),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(5),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.4914, 0.4821, 0.4465],
                        std=[0.2471, 0.2435, 0.2616]),
    transforms.RandomErasing(p=0.75,
                            scale=(0.01, 0.3),
                            ratio=(1.0, 1.0),
                            value=0,
                            inplace =True)
])
```



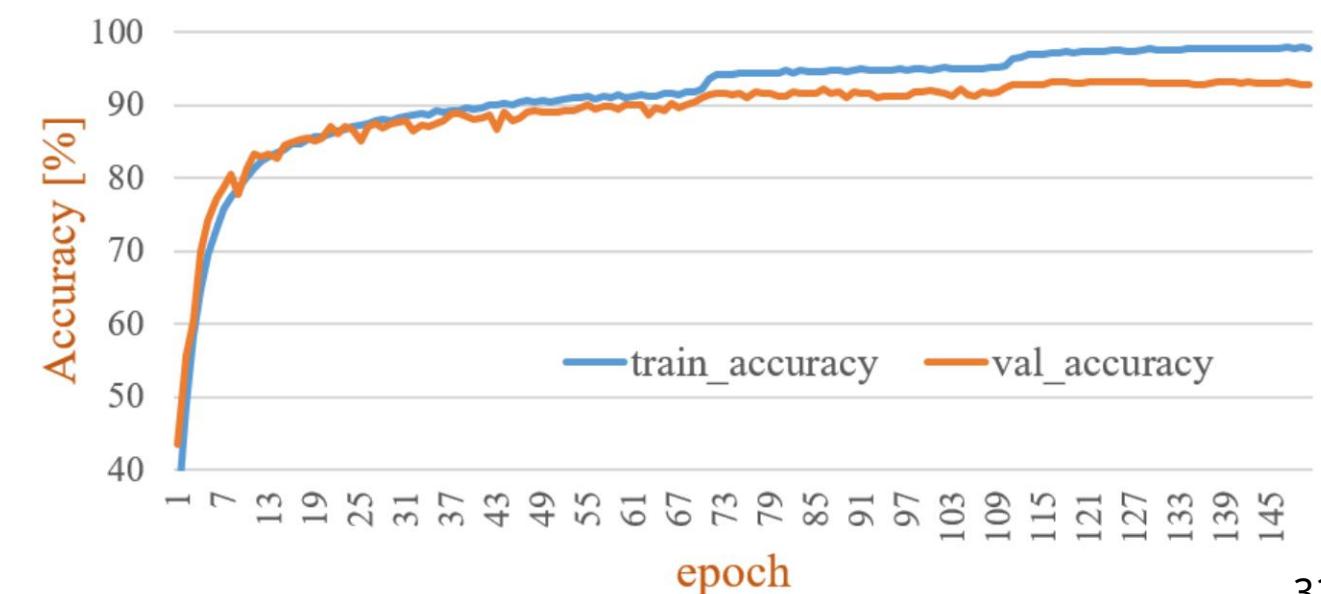
Khái quát hóa mô hình

Thủ thuật 7: Tăng dung lư ợng mô hình (và sử dụng nhiều dữ liệu hơn)



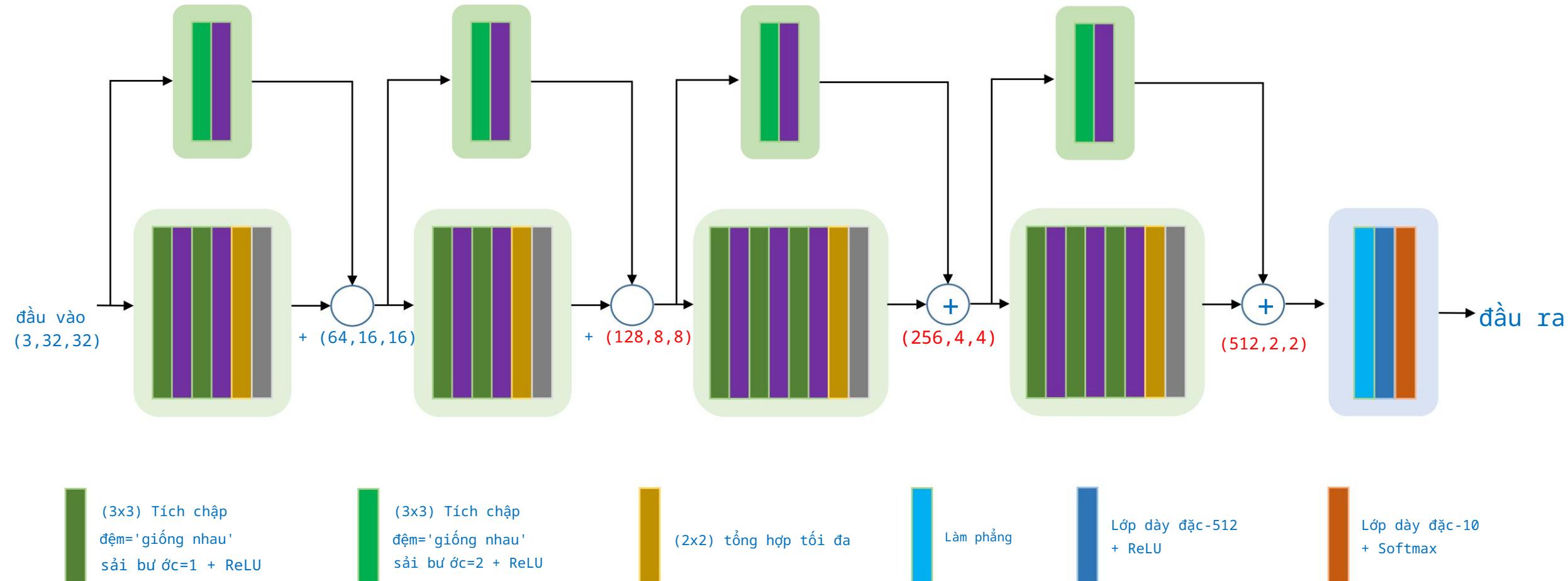
val_accuracy đạt tới ~93%

train_accuracy đạt tới ~96%



Khái quát hóa mô hình

Thủ thuật 8: Sử dụng bỏ qua kết nối

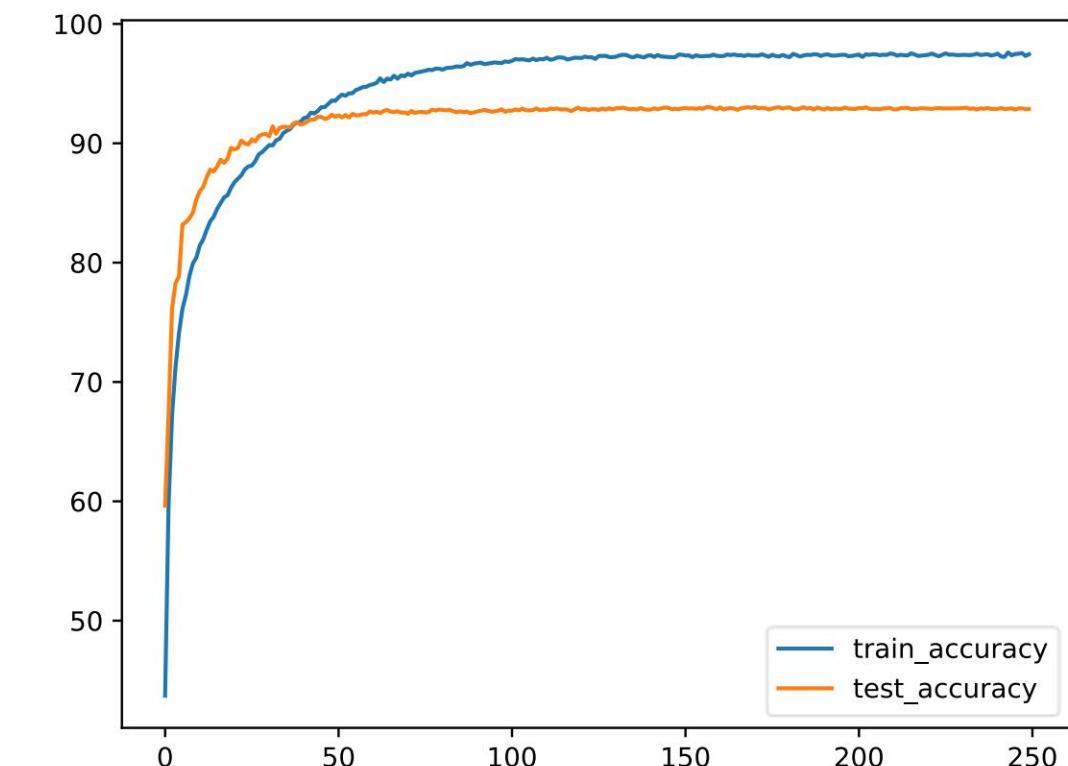
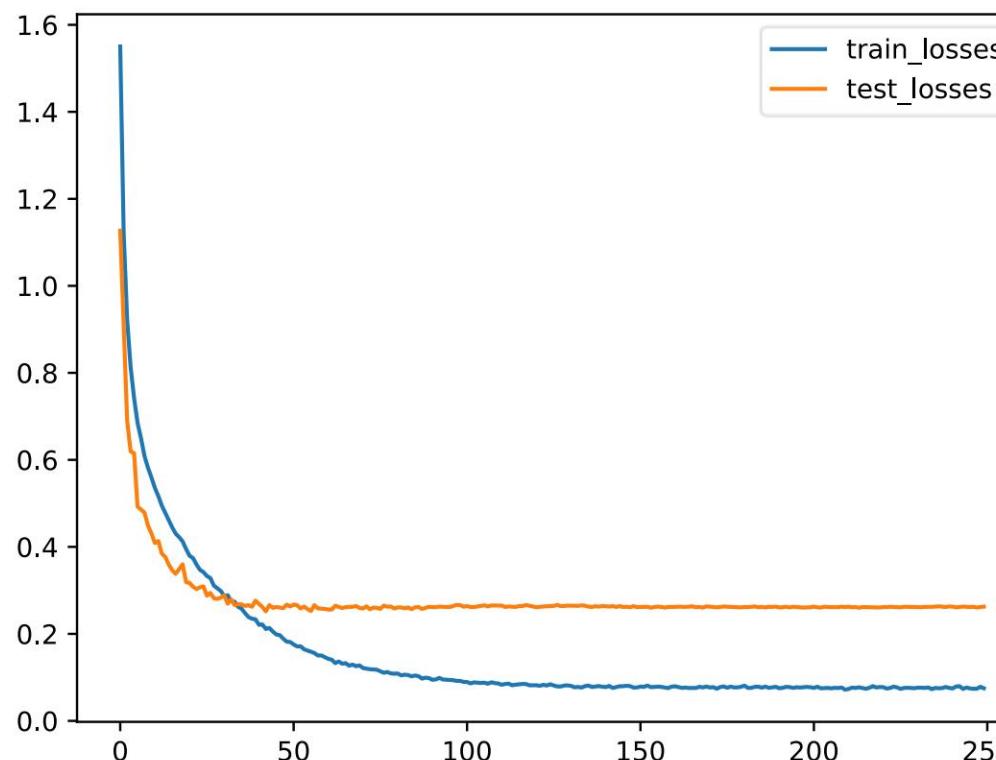


Khái quát hóa mô hình

Thủ thuật 8: Sử dụng bỏ qua kết nối

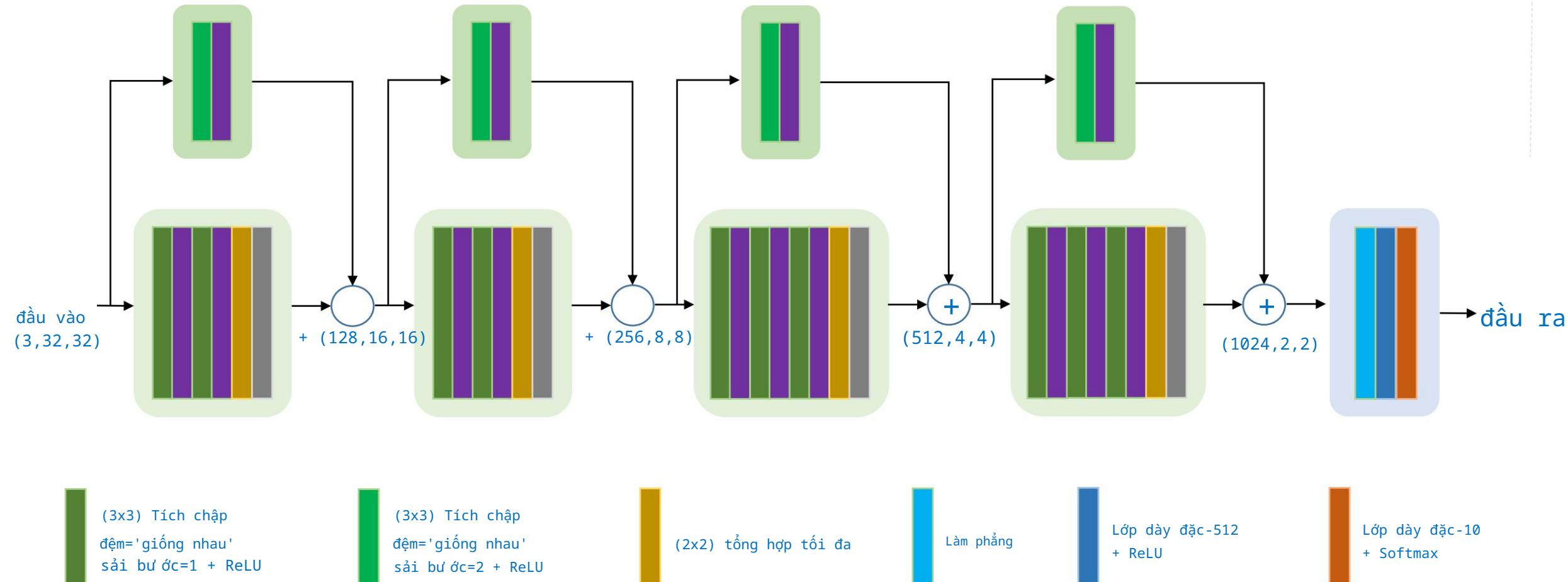
val_accuracy đạt ~93%

train_accuracy đạt ~97%



Khái quát hóa mô hình

Tăng công suất mô hình một lần nữa

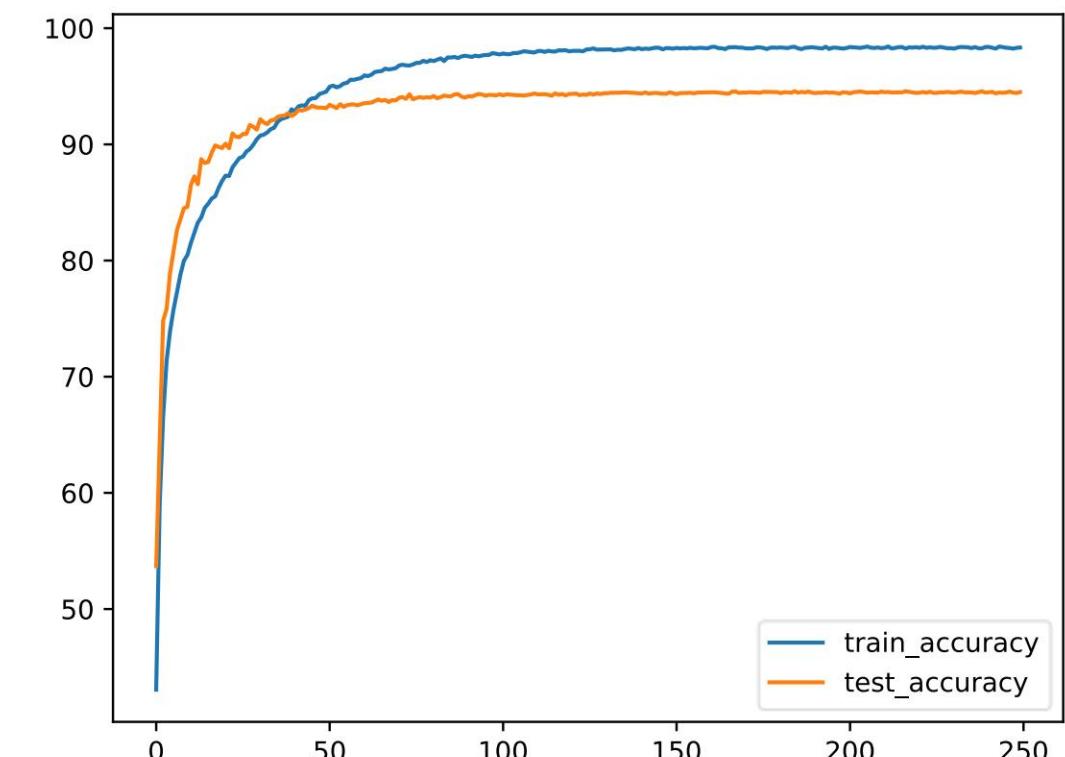
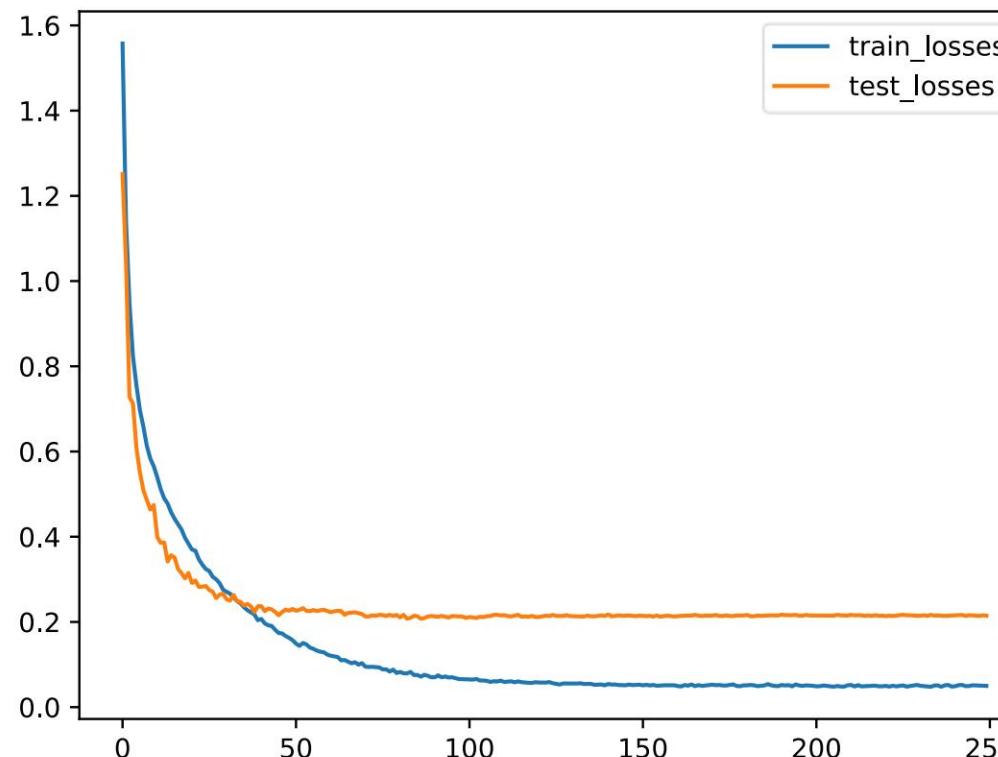


Khái quát hóa mô hình

Tăng công suất mô hình một lần nữa

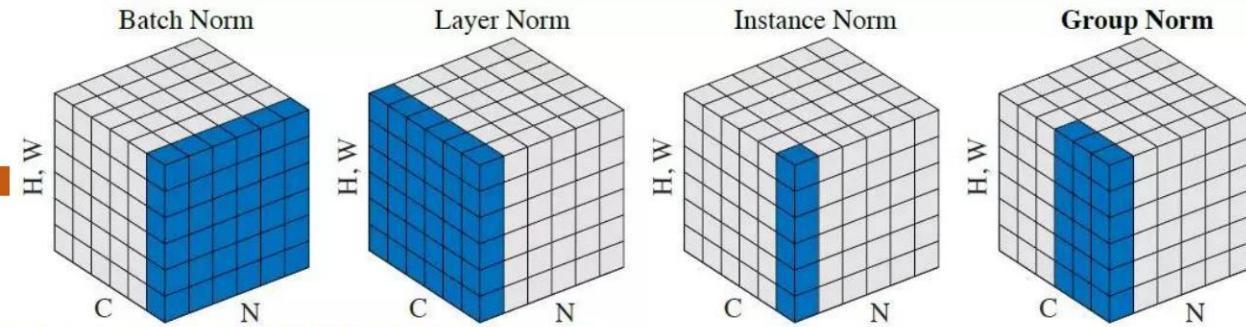
val_accuracy đạt tới ~94,5%

train_accuracy đạt tới ~98,3%

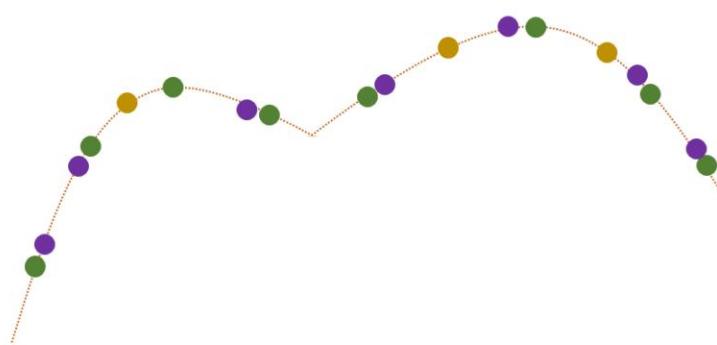


Bản tóm tắt

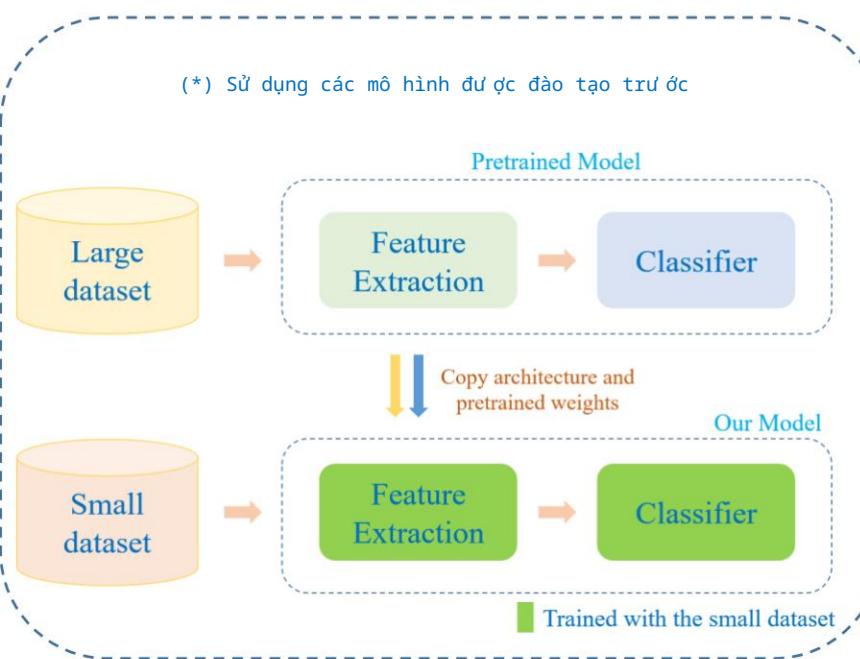
Cách tăng độ chính xác khi xác thực



Increase data by altering the training data



Image



Thủ thuật 5: Tăng cường dữ liệu

$$= + \parallel \parallel^2$$

2chỉnh quy hóa

Thủ thuật 4: Chính quy hóa kernel

Thủ thuật 2:
Sử dụng chuẩn hóa hàng loạt

$$= \sqrt{2} +$$

Thủ thuật 3: Sử dụng Dropout

