



CSE: Faculty of Computer Science and Engineering

Thuyloi University

Mạng nơ ron tích chập hiện đại

TS. Nguyễn Thị Kim Ngân

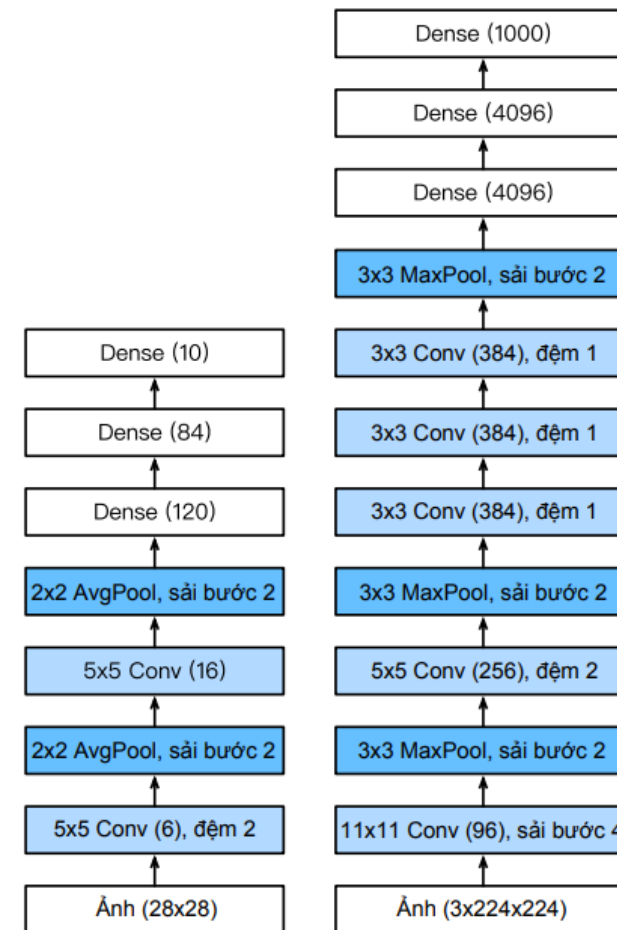


Nội dung

- Mạng nơ-ron tích chập sâu (AlexNet)
- Mạng sử dụng khối (VGG)
- Mạng trong mạng (NiN)
- Mạng nối song song (GoogLeNet)
- Mạng phần dư (ResNet)
- Mạng tích chập kết nối dày đặc (DenseNet)

Mạng nơ-ron tích chập sâu (AlexNet)

- AlexNet sâu hơn nhiều so với LeNet5.
AlexNet có tám tầng gồm:
 - 5 tầng tích chập,
 - 2 tầng ẩn kết nối đầy đủ
 - 1 tầng đầu ra kết nối đầy đủ
- AlexNet sử dụng hàm kích hoạt ReLU thay vì sigmoid
- AlexNet kiểm soát năng lực của tầng kết nối đầy đủ bằng cách áp dụng dropout (LeNet chỉ sử dụng suy giảm trọng số)

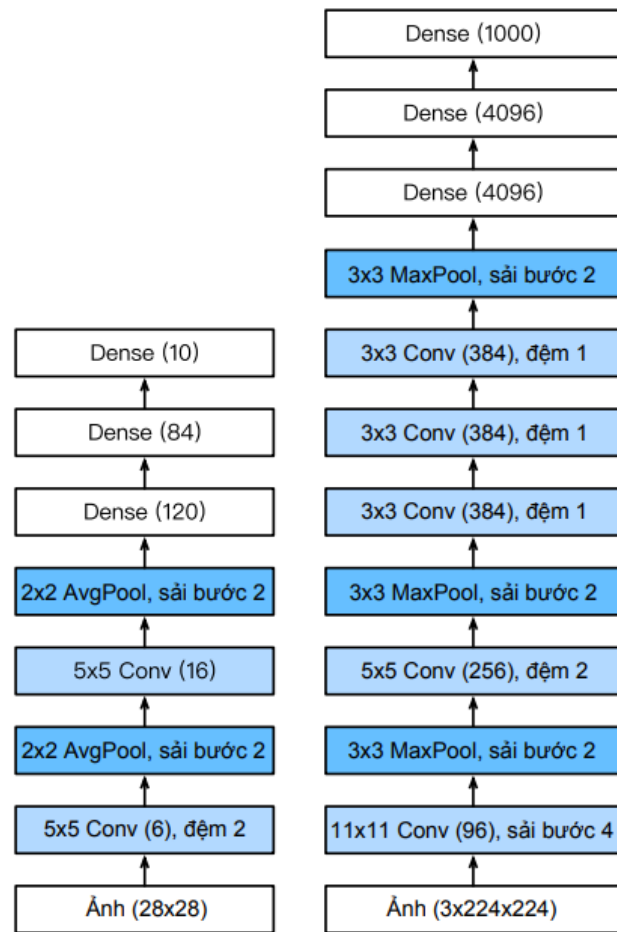


LeNet (trái) và AlexNet (phải)

Mạng nơ-ron tích chập sâu (AlexNet)

```
from d2l import mxnet as d2l
from mxnet import np, npx
from mxnet.gluon import nn
npx.set_np()

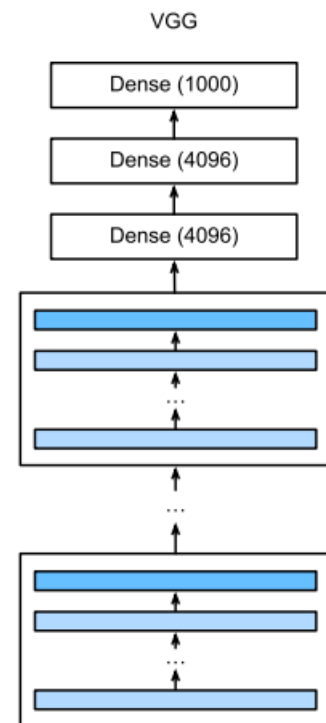
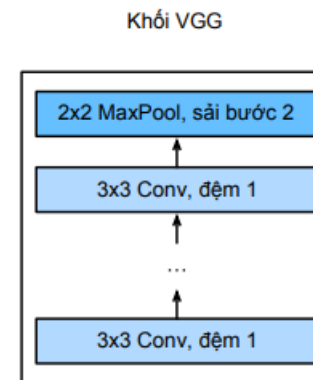
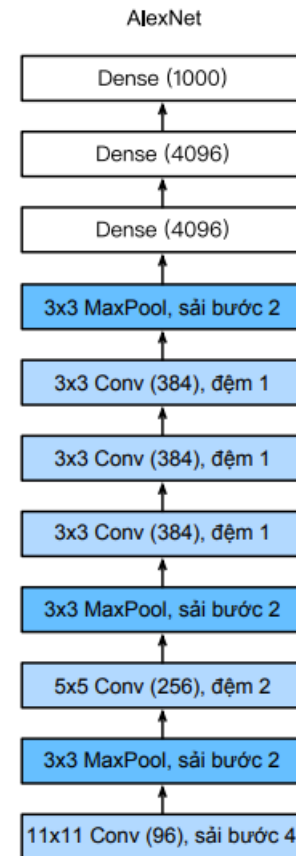
net = nn.Sequential()
# Here, we use a larger 11 x 11 window to capture objects. At the same time,
# we use a stride of 4 to greatly reduce the height and width of the output.
# Here, the number of output channels is much larger than that in LeNet
net.add(nn.Conv2D(96, kernel_size=11, strides=4, activation='relu'),
        nn.MaxPool2D(pool_size=3, strides=2),
        # Make the convolution window smaller, set padding to 2 for consistent
        # height and width across the input and output, and increase the
        # number of output channels
        nn.Conv2D(256, kernel_size=5, padding=2, activation='relu'),
        nn.MaxPool2D(pool_size=3, strides=2),
        # Use three successive convolutional layers and a smaller convolution
        # window. Except for the final convolutional layer, the number of
        # output channels is further increased. Pooling layers are not used to
        # reduce the height and width of input after the first two
        # convolutional layers
        nn.Conv2D(384, kernel_size=3, padding=1, activation='relu'),
        nn.Conv2D(384, kernel_size=3, padding=1, activation='relu'),
        nn.Conv2D(256, kernel_size=3, padding=1, activation='relu'),
        nn.MaxPool2D(pool_size=3, strides=2),
        # Here, the number of outputs of the fully connected layer is several
        # times larger than that in LeNet. Use the dropout layer to mitigate
        # overfitting
        nn.Dense(4096, activation="relu"), nn.Dropout(0.5),
        nn.Dense(4096, activation="relu"), nn.Dropout(0.5),
        # Output layer. Since we are using Fashion-MNIST, the number of
        # classes is 10, instead of 1000 as in the paper
        nn.Dense(10))
```



LeNet (trái) và AlexNet (phải)

Mạng sử dụng Khối (VGG)

- Một khối VGG gồm một chuỗi các tầng tích chập, tiếp nối bởi một tầng gộp cực đại để giảm chiều không gian.
- Bài báo gốc của VGG (Simonyan & Zisserman, 2014) sử dụng tích chập với các hạt nhân 3×3 và tầng gộp cực đại 2×2 với sải bước bằng 2 (giảm một nửa độ phân giải sau mỗi khối)



Thiết kế mạng từ các khối cơ bản



Mạng sử dụng Khối (VGG)

- Một khối VGG gồm một chuỗi các tầng tích chập, tiếp nối bởi một tầng gộp cực đại để giảm chiều không gian.
- Bài báo gốc của VGG (Simonyan & Zisserman, 2014) sử dụng tích chập với các hạt nhân 3×3 và tầng gộp cực đại 2×2 với sải bước bằng 2 (giảm một nửa độ phân giải sau mỗi khối)

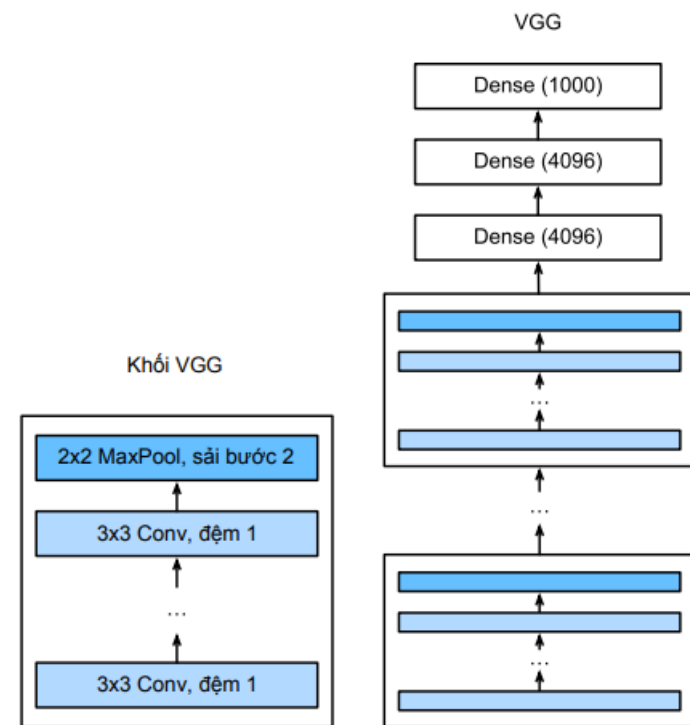
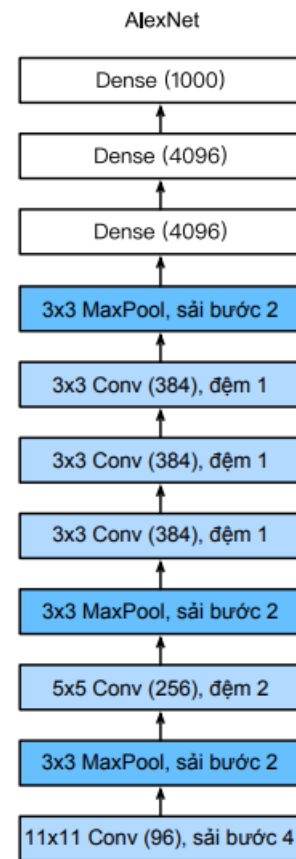
```
from d2l import mxnet as d2l
from mxnet import np, npx
from mxnet.gluon import nn
npx.set_np()

def vgg_block(num_convs, num_channels):
    blk = nn.Sequential()
    for _ in range(num_convs):
        blk.add(nn.Conv2D(num_channels, kernel_size=3,
                           padding=1, activation='relu'))
    blk.add(nn.MaxPool2D(pool_size=2, strides=2))
    return blk
```

Mạng sử dụng Khối (VGG)

- Mạng VGG gốc có 5 khối tích chập
 - 2 khối đầu tiên: gồm một tầng tích chập ở mỗi khối
 - 3 khối còn lại chứa hai tầng tích chập ở mỗi khối
 - Khối đầu tiên có 64 kênh đầu ra, mỗi khối tiếp theo nhân đôi số kênh đầu ra cho tới khi đạt giá trị 512
 - Mạng VGG sử dụng 8 tầng tích chập, 3 tầng kết nối đầy nên nó thường được gọi là VGG-11

```
conv_arch = ((1, 64), (1, 128), (2, 256), (2, 512), (2, 512))
```



Thiết kế mạng từ các khối cơ bản



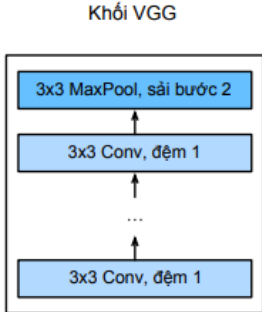
Mạng sử dụng Khối (VGG)

- Mạng VGG gốc có 5 khối tích chập
 - 2 khối đầu tiên: gồm một tầng tích chập ở mỗi khối
 - 3 khối còn lại chứa hai tầng tích chập ở mỗi khối
 - Khối đầu tiên có 64 kênh đầu ra, mỗi khối tiếp theo nhân đôi số kênh đầu ra cho tới khi đạt giá trị 512
 - Mạng VGG sử dụng 8 tầng tích chập, 3 tầng kết nối đầy nên nó thường được gọi là VGG-11

```
conv_arch = ((1, 64), (1, 128), (2, 256), (2, 512), (2, 512))
```

```
def vgg(conv_arch):  
    net = nn.Sequential()  
    # The convolutional layer part  
    for (num_convs, num_channels) in conv_arch:  
        net.add(vgg_block(num_convs, num_channels))  
    # The fully connected layer part  
    net.add(nn.Dense(4096, activation='relu'), nn.Dropout(0.5),  
            nn.Dense(4096, activation='relu'), nn.Dropout(0.5),  
            nn.Dense(10))  
    return net
```


- 1

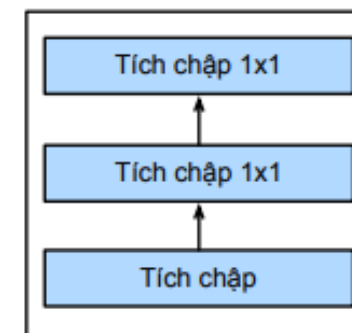


Mạng trong mạng (Network in Network - NiN)

Khối NiN bao gồm: 1 tầng tích chập theo sau bởi hai tầng tích chập 1×1 hoạt động như các tầng kết nối đầy đủ trên điểm ảnh và sau đó là hàm kích hoạt ReLU

- Kích thước cửa sổ tích chập của tầng thứ nhất thường được định nghĩa bởi người dùng
- Kích thước cửa sổ tích chập ở các tầng tiếp theo được cố định bằng 1×1

Khối NiN

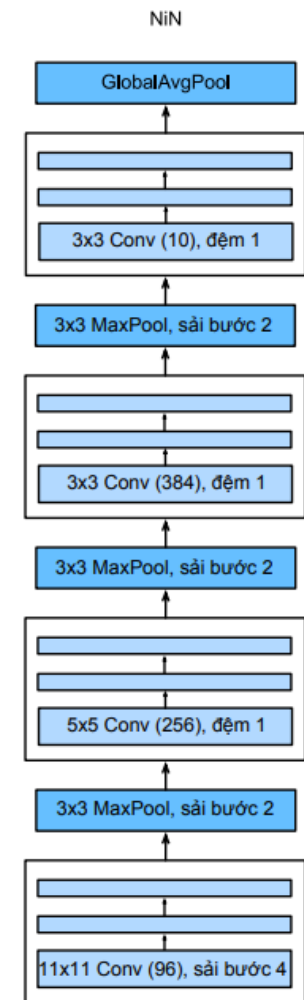


```
from d2l import mxnet as d2l
from mxnet import np, npx
from mxnet.gluon import nn
npx.set_np()

def nin_block(num_channels, kernel_size, strides, padding):
    blk = nn.Sequential()
    blk.add(nn.Conv2D(num_channels, kernel_size, strides, padding,
                      activation='relu'),
            nn.Conv2D(num_channels, kernel_size=1, activation='relu'),
            nn.Conv2D(num_channels, kernel_size=1, activation='relu'))
    return blk
```

Mạng trong mạng (Network in Network - NiN)

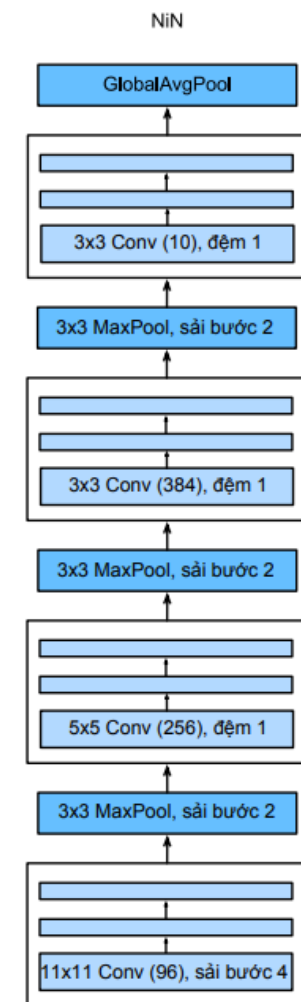
- NiN sử dụng các tầng tích chập có kích thước cửa sổ 11×11 , 5×5 , 3×3 , và số lượng các kênh đầu ra tương ứng giống với AlexNet
- Mỗi khối NiN theo sau bởi một tầng gộp cực đại với sai bước 2 và kích thước cửa sổ 3×3
- Các khối NiN có số kênh đầu ra bằng với số lớp nhân, theo sau bởi một tầng gộp trung bình toàn cục, tạo ra một vector logit
- Ưu điểm: Giảm được các tham số cần thiết của mô hình một cách đáng kể
- Nhược điểm: Đôi lúc đòi hỏi tăng thời gian huấn luyện mô hình



Mạng trong mạng (Network in Network - NiN)

- Xây dựng mô hình

```
net = nn.Sequential()
net.add(nin_block(96, kernel_size=11, strides=4, padding=0),
        nn.MaxPool2D(pool_size=3, strides=2),
        nin_block(256, kernel_size=5, strides=1, padding=2),
        nn.MaxPool2D(pool_size=3, strides=2),
        nin_block(384, kernel_size=3, strides=1, padding=1),
        nn.MaxPool2D(pool_size=3, strides=2),
        nn.Dropout(0.5),
        # There are 10 label classes
        nin_block(10, kernel_size=3, strides=1, padding=1),
        # The global average pooling layer automatically sets the window shape
        # to the height and width of the input
        nn.GlobalAvgPool2D(),
        # Transform the four-dimensional output into two-dimensional output
        # with a shape of (batch size, 10)
        nn.Flatten())
```





Mạng nối song song (GoogLeNet)

- Kích thước nào của bộ lọc tích chập là tốt nhất?

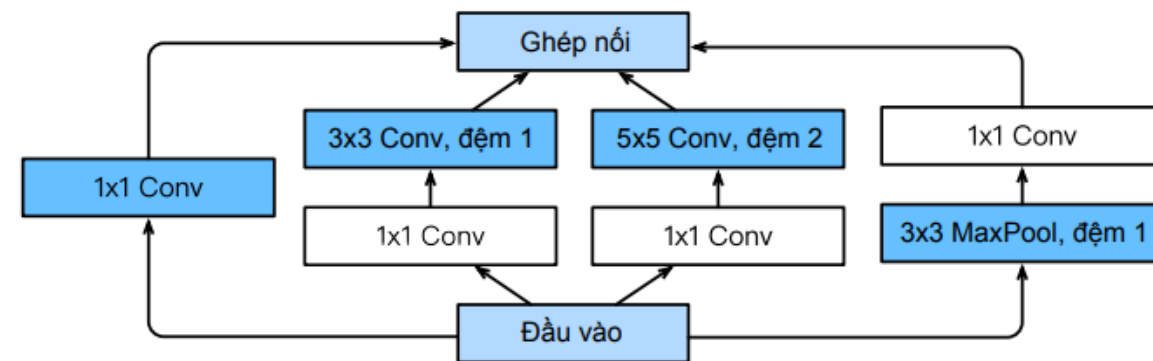
=> Đôi khi việc kết hợp các bộ lọc có kích thước khác nhau có thể sẽ hiệu quả

Mạng nối song song (GoogLeNet)

Khối Interception

Gồm 4 nhánh song song với nhau:

- Ba nhánh đầu sử dụng các tầng tích chập với kích thước cửa sổ trượt lần lượt là 1×1 , 3×3 , và 5×5 để trích xuất thông tin từ các vùng không gian có kích thước khác nhau
- Hai nhánh giữa thực hiện phép tích chập 1×1 trên dữ liệu đầu vào để giảm số kênh đầu vào
- Nhánh thứ tư sử dụng một tầng gộp cực đại kích thước 3×3 , theo sau là một tầng tích chập 1×1 để thay đổi số lượng kênh
- Cả bốn nhánh sử dụng phần đệm phù hợp để đầu vào và đầu ra của khối có cùng chiều cao, chiều rộng. Các đầu ra của mỗi nhánh được nối lại theo chiều kênh để tạo thành đầu ra của cả khối
- Các tham số thường được tinh chỉnh của khối Inception là số lượng kênh đầu ra mỗi tầng



Cấu trúc của khối Inception

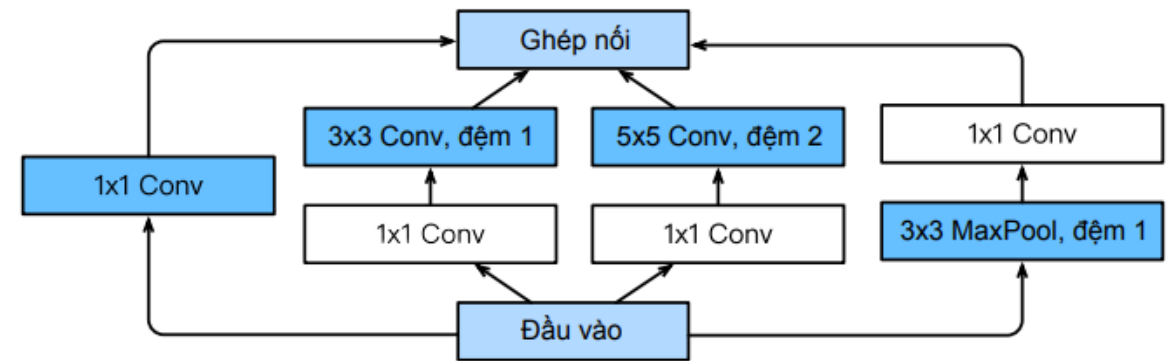
Mạng nối song song (GoogLeNet)

Khối Interception

```
from d2l import mxnet as d2l
from mxnet import np, npx
from mxnet.gluon import nn
npx.set_np()

class Inception(nn.Block):
    # c1 - c4 are the number of output channels for each layer in the path
    def __init__(self, c1, c2, c3, c4, **kwargs):
        super(Inception, self).__init__(**kwargs)
        # Path 1 is a single 1 x 1 convolutional layer
        self.p1_1 = nn.Conv2D(c1, kernel_size=1, activation='relu')
        # Path 2 is a 1 x 1 convolutional layer followed by a 3 x 3
        # convolutional layer
        self.p2_1 = nn.Conv2D(c2[0], kernel_size=1, activation='relu')
        self.p2_2 = nn.Conv2D(c2[1], kernel_size=3, padding=1,
                               activation='relu')
        # Path 3 is a 1 x 1 convolutional layer followed by a 5 x 5
        # convolutional layer
        self.p3_1 = nn.Conv2D(c3[0], kernel_size=1, activation='relu')
        self.p3_2 = nn.Conv2D(c3[1], kernel_size=5, padding=2,
                               activation='relu')
        # Path 4 is a 3 x 3 maximum pooling layer followed by a 1 x 1
        # convolutional layer
        self.p4_1 = nn.MaxPool2D(pool_size=3, strides=1, padding=1)
        self.p4_2 = nn.Conv2D(c4, kernel_size=1, activation='relu')

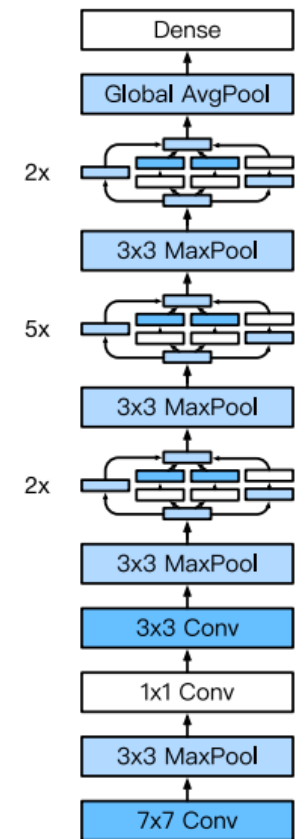
    def forward(self, x):
        p1 = self.p1_1(x)
        p2 = self.p2_2(self.p2_1(x))
        p3 = self.p3_2(self.p3_1(x))
        p4 = self.p4_2(self.p4_1(x))
        # Concatenate the outputs on the channel dimension
        return np.concatenate((p1, p2, p3, p4), axis=1)
```



Cấu trúc của khối Inception

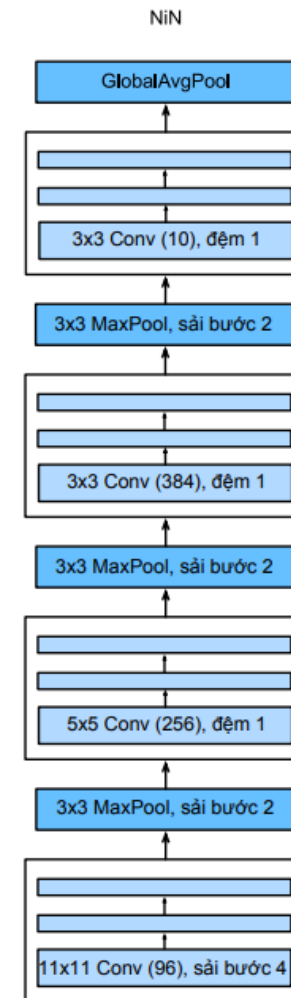
Mạng nối song song (GoogLeNet)

- Mô hình GoogLeNet sử dụng 9 khối inception và tầng gộp trung bình toàn cục xếp chồng lên nhau
- Phép gộp cực đại giữa các khối inception có tác dụng làm giảm kích thước chiều



Chuẩn hóa theo batch

- Khi huấn luyện các mạng Perceptron đa tầng hay CNN, các giá trị kích hoạt ở các tầng trung gian có thể nhận các giá trị với mức độ biến thiên lớn- dọc theo các tầng từ đầu vào đến đầu ra,
- Sự thay đổi trong phân phối của những giá trị kích hoạt có thể cản trở sự hội tụ của mạng. Nếu một tầng có các giá trị kích hoạt lớn gấp 100 lần so với các tầng khác, thì cần phải có các điều chỉnh bổ trợ trong tốc độ học





Chuẩn hóa theo batch


- Chuẩn hoá theo batch (Batch Normalization - BN)) được áp dụng cho từng tầng riêng lẻ (hoặc có thể cho tất cả các tầng) và hoạt động như sau:
 - Trong mỗi vòng lặp huấn luyện, tại mỗi tầng, đầu tiên tính giá trị kích hoạt như thường lệ.
 - Sau đó, chuẩn hóa những giá trị kích hoạt của mỗi nút bằng việc trừ đi giá trị trung bình và chia cho độ lệch chuẩn. Cả hai đại lượng này được ước tính dựa trên số liệu thống kê của minibatch hiện tại
- BN chuyển đổi những giá trị kích hoạt tại tầng x nhất định theo công thức sau:

$$\text{BN}(\mathbf{x}) = \gamma \odot \frac{\mathbf{x} - \hat{\mu}}{\hat{\sigma}} + \beta$$

- Ký hiệu một minibatch là \mathcal{B} , chúng ta tính $\hat{\mu}_{\mathcal{B}}$ và $\hat{\sigma}_{\mathcal{B}}$ theo công thức sau:

$$\hat{\mu}_{\mathcal{B}} \leftarrow \frac{1}{|\mathcal{B}|} \sum_{\mathbf{x} \in \mathcal{B}} \mathbf{x} \text{ và } \hat{\sigma}_{\mathcal{B}}^2 \leftarrow \frac{1}{|\mathcal{B}|} \sum_{\mathbf{x} \in \mathcal{B}} (\mathbf{x} - \mu_{\mathcal{B}})^2 + \epsilon$$

- Hằng số rất nhỏ $\epsilon > 0$ vào biểu thức tính phương sai để đảm bảo tránh phép chia cho 0 khi chuẩn hóa




Chuẩn hóa theo batch

Chuẩn hoá theo batch cho tầng kết nối đầy đủ

- Chèn BN sau bước biến đổi affine và trước hàm kích hoạt phi tuyến
- Kí hiệu đầu vào của tầng là \mathbf{x} , hàm biến đổi tuyến tính là $f_\theta(\cdot)$ (với trọng số là θ), hàm kích hoạt là $\phi(\cdot)$, và phép tính BN là $\text{BN}_{\alpha,\beta}$ với tham số β và γ , việc tính toán tầng kết nối đầy đủ h khi chèn lớp BN vào như sau:

$$\mathbf{h} = \phi(\text{BN}_{\beta,\gamma}(f_\theta(\mathbf{x})))$$



Chuẩn hóa theo batch

Chuẩn hoá theo batch cho tầng tích chập:

- Áp dụng BN sau phép tích chập và trước hàm kích hoạt phi tuyến
- Khi áp dụng phép tích chập cho đầu ra nhiều kênh, chúng ta cần thực hiện chuẩn hóa theo batch cho mỗi đầu ra của những kênh này, và mỗi kênh sẽ có riêng cho nó các tham số tỉ lệ và độ chệch, cả hai đều là các số vô hướng



Chuẩn hóa theo batch

Chuẩn hoá theo batch trong quá trình dự đoán

- Sau khi huấn luyện, chúng ta sử dụng toàn bộ tập dữ liệu để tính toán các con số thống kê của các giá trị kích hoạt và sau đó cố định chúng tại thời điểm dự đoán.
- Do đó, BN hoạt động khác nhau trong quá trình huấn luyện và kiểm tra



Chuẩn hóa theo batch

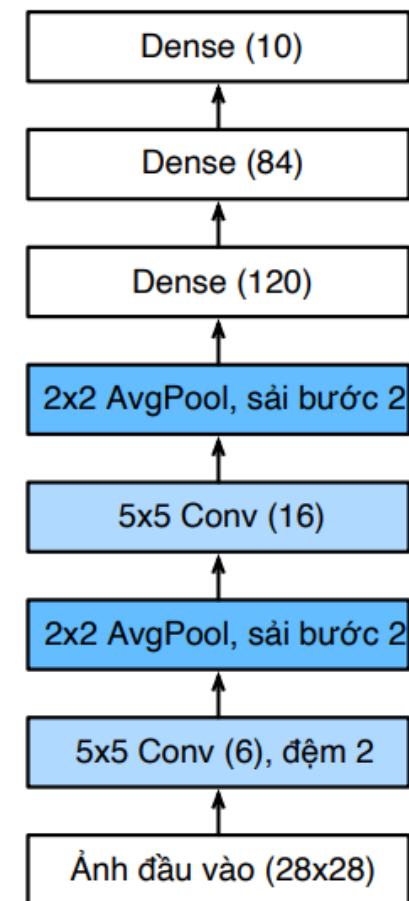
```
class BatchNorm(nn.Block):
    def __init__(self, num_features, num_dims, **kwargs):
        super(BatchNorm, self).__init__(**kwargs)
        if num_dims == 2:
            shape = (1, num_features)
        else:
            shape = (1, num_features, 1, 1)
        # The scale parameter and the shift parameter involved in gradient
        # finding and iteration are initialized to 0 and 1 respectively
        self.gamma = self.params.get('gamma', shape=shape, init=init.One())
        self.beta = self.params.get('beta', shape=shape, init=init.Zero())
        # All the variables not involved in gradient finding and iteration are
        # initialized to 0 on the CPU
        self.moving_mean = np.zeros(shape)
        self.moving_var = np.zeros(shape)

    def forward(self, X):
        # If X is not on the CPU, copy moving_mean and moving_var to the
        # device where X is located
        if self.moving_mean.ctx != X.ctx:
            self.moving_mean = self.moving_mean.copyto(X.ctx)
            self.moving_var = self.moving_var.copyto(X.ctx)
        # Save the updated moving_mean and moving_var
        Y, self.moving_mean, self.moving_var = batch_norm(
            X, self.gamma.data(), self.beta.data(), self.moving_mean,
            self.moving_var, eps=1e-12, momentum=0.9)
        return Y
```

Sử dụng LeNet với chuẩn hóa theo batch

- BN thường được sử dụng sau tầng tích chập và tầng kết nối đầy đủ và trước hàm kích hoạt tương ứng

```
net = nn.Sequential()  
net.add(nn.Conv2D(6, kernel_size=5),  
        BatchNorm(6, num_dims=4),  
        nn.Activation('sigmoid'),  
        nn.MaxPool2D(pool_size=2, strides=2),  
        nn.Conv2D(16, kernel_size=5),  
        BatchNorm(16, num_dims=4),  
        nn.Activation('sigmoid'),  
        nn.MaxPool2D(pool_size=2, strides=2),  
        nn.Dense(120),  
        BatchNorm(120, num_dims=2),  
        nn.Activation('sigmoid'),  
        nn.Dense(84),  
        BatchNorm(84, num_dims=2),  
        nn.Activation('sigmoid'),  
        nn.Dense(10))
```



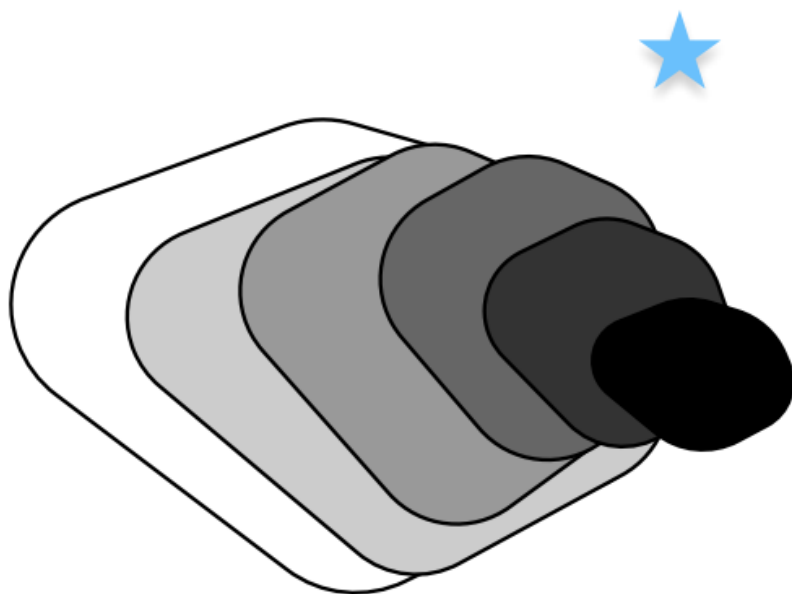


Mạng phân dư (ResNet)

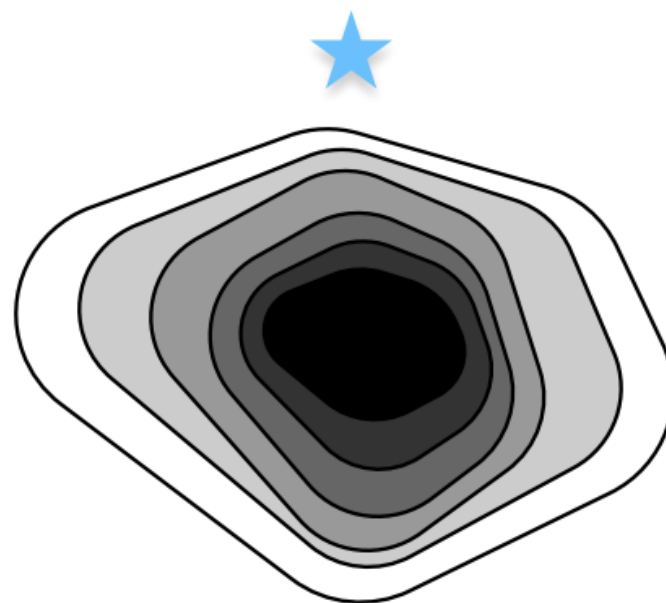
- Coi F là một lớp các hàm mà một kiến trúc mạng cụ thể (cùng với tốc độ học và các siêu tham số khác) có thể đạt được. Nói cách khác, với mọi hàm số $f \in F$, luôn tồn tại một số tập tham số W có thể tìm được bằng việc huấn luyện trên một tập dữ liệu phù hợp
- Giả sử f^* là hàm cần tìm. Sẽ rất thuận lợi nếu hàm này thuộc tập F , nhưng thường không may mắn như vậy.
- Thay vào đó, ta sẽ cố gắng tìm các hàm số f_F^* tốt nhất có thể trong tập F
- Tìm f_F^* bằng cách giải bài toán tối ưu sau:

$$f_F^* := \underset{f}{\operatorname{argmin}} L(X, Y, f) \text{ đối tượng thoả mãn } f \in F.$$

Mạng phần dư (ResNet)



các lớp hàm số tổng quát

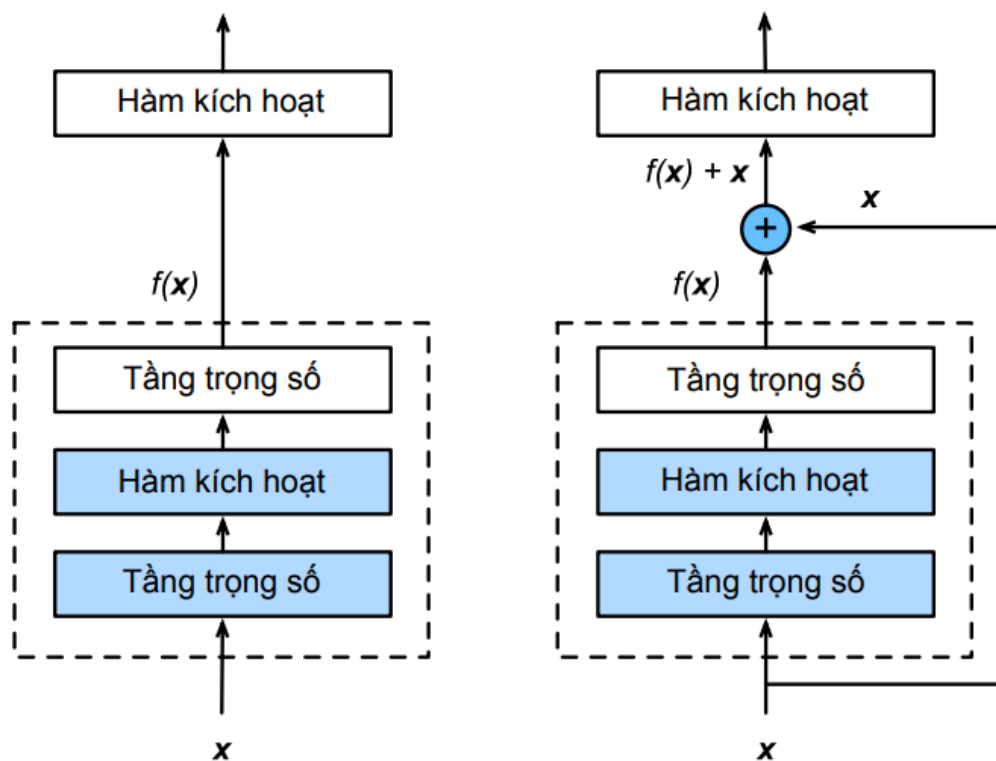


các lớp hàm số lồng nhau

Hình trái: Các lớp hàm số tổng quát. Khoảng cách đến hàm cần tìm f^* (ngôi sao), trên thực tế có thể tăng khi độ phức tạp tăng lên. Hình phải: với các lớp hàm số lồng nhau, điều này không xảy ra.

Mạng phần dư (ResNet)

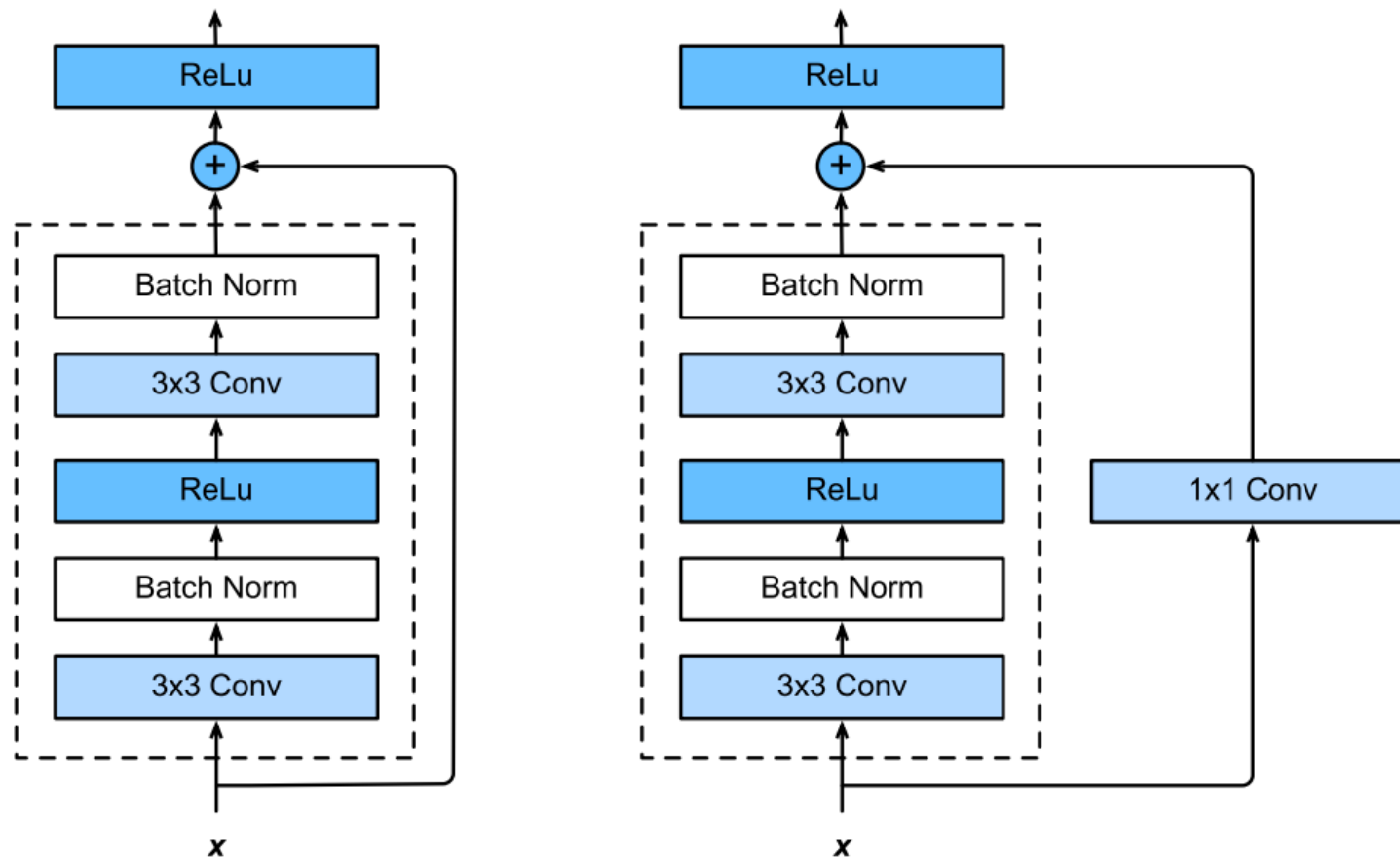
- **Khôi phần dư**



Sự khác biệt giữa một khối thông thường (trái) và một khối phần dư (phải). Trong khối phần dư, ta có thể nối tắt các tích chập.

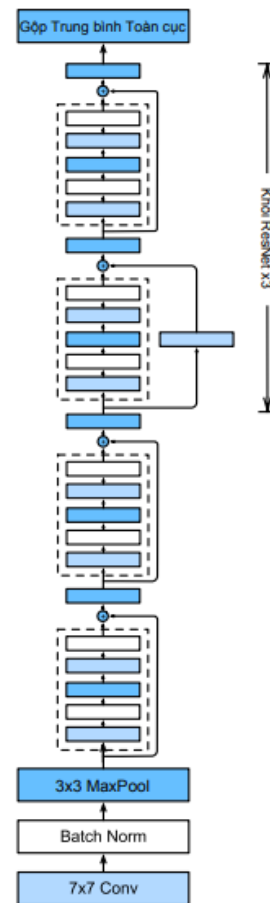
Mạng phân dư (ResNet)

- **Khối ResNet**



Trái: khối ResNet thông thường; Phải: Khối ResNet với tầng tích chập 1x1

Mạng phân dư (ResNet)



ResNet-18