IT5005 Artificial Intelligence

Sirigina Rajendra Prasad
AY2025/2026: Semester 1

Tutorial: CNN

1. **ConvNets**

(a) You are given an image $\mathbf{x} \in \mathbb{R}^{4\times4}$ and a filter $\mathbf{W} \in \mathbb{R}^{3\times3}$. **No** extra padding is added.

0.5	0.2	0.1	0.7
0.1	0.6	0.9	0.5
0.0	0.8	0.2	0.7
0.2	0.4	0.0	0.4

0.1	0.2	0.6
0.4	0.3	0.5
0.9	0.8	0.7

Figure 1: (left) The image $\mathbf{x}$. (right) The filter $\mathbf{W}$.

Get the output feature map from this convolution and write down its output dimensions if the kernel moved with a stride of 1×1. No Padding or Pooling operation required. Note that while mathematically, there is a need to flip the kernel matrix, in practice when using CNNs we do not flip the kernel. This saves on the cost of flipping while making little practical difference. For this question, we compute the output feature map with no flipping.

$$O = \left\lfloor \frac{I + 2P - K}{S} \right\rfloor + 1$$

where

O is the output dimension (either height or width).

I is the output dimension (either height or width).

$P = padding$

$K = kernel\_size$

$S = stride$

Usually, the convolution layer is designed to retain the height and width of input, i.e., $O = I$

Q1a: Solution

$$O = \left\lfloor \frac{I + 2P - K}{S} \right\rfloor + 1$$

- Output Dimension:
  - 2×2

$$P = 0$$
$$S = 0$$
$$K = 3$$
$$\mathrm{I} = 4$$
$$\mathrm{O} = 2$$

- Output Feature Map:

$$\begin{bmatrix} 1.6 & 2.59 \\ 1.51 & 1.91 \end{bmatrix}$$

Q1

(b) Now, let's say you have access to a very deep, large CNN model. We feed a single image to the network. Each image has $3(C)$ channels (RGB) with a height and width of 224×224 ($H = 224, W = 224$). Here our input is a 3-dimensional tensor ($H \times W \times C$).

The first layer of the big CNN is a Convolutional Layer with 96 (C_1) kernels which has a height and width of 11, and each kernel has the same number of channels as the input channel. The stride is 4×4 and no padding is used. Calculate the output size $H_1 \times W_1 \times C_1$ after the first Convolutional Layer.

Q1b

$$P = 0$$
$$S = 4$$
$$K = 11$$
$$\mathrm{I} = \mathrm{H} = \mathrm{W} = 224$$
$$\mathrm{O} = \mathrm{H} = \mathrm{W} = 54$$

$$\mathrm{O} = \left\lfloor \frac{I + 2P - K}{S} \right\rfloor + 1$$

Output size: $54 \times 54 \times 96$

LeNet Architecture

 # LeNet 5 Architecture

MNIST Image (28×28) → C1 (28×28×6) → S2 (14×14×6) → C3 (14×14×16) → S4 (7×7×16) → C5 (120) → F6 (84) → Output (10)

1. Input: 28x28 grayscale image
2. C1: Convolutional layer (6 feature maps, 5x5 kernels)
3. S2: Average pooling layer (2x2)
4. C3: Convolutional layer (16 feature maps, 5x5 kernels)
5. S4: Average pooling layer (2x2)
6. C5: Fully connected layer (120 units)
7. F6: Fully connected layer (84 units)
8. Output: Fully connected layer (10 units)

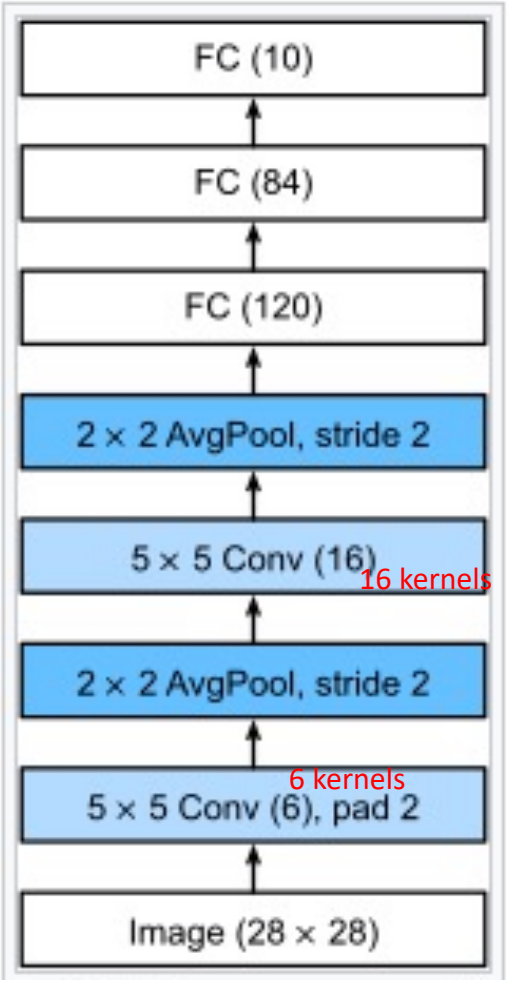
Build LeNet-5 Architecture. Note that the activation function is Sigmoid.

The code provides the parameter count for LeNet-5. Justify the number of parameters through manual calculation of number of parameters.

Modify the code to use ReLU activation instead of Sigmoid. Which activation function is better: ReLU or Sigmoid? Explain.

What changes would you make to this architecture to handle CIFAR-10 images?

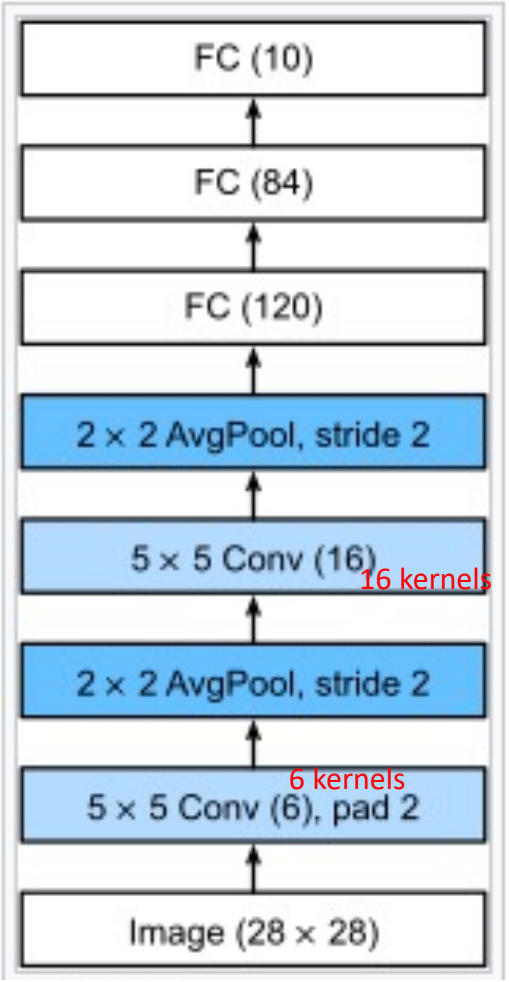
LetNet5 Architecture:




```
nn.Linear(in_features =__, out_features=__)

nn.Linear(in_features=__, out_features=__)

nn.Linear(in_features=__, out_features=__)

nn.AvgPool2d(kernel_size=__, stride=__)

nn.Conv2d(in_channels=__, out_channels=__, kernel_size=__, stride=1, padding=2)

nn.AvgPool2d(kernel_size=__, stride=__)

nn.Conv2d(in_channels=__, out_channels=__, kernel_size=__, stride=1, padding=2)
```

Image taken from: https://en.wikipedia.org/wiki/LeNet

LetNet5 Architecture




```
nn.Linear(in_features=84, out_features=10)

nn.Linear(in_features =120, out_features =84)

nn.Linear(in_features =784, out_features =120)

nn.AvgPool2d(kernel_size=2, stride=2)

nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5, stride=1, padding=2)

nn.AvgPool2d(kernel_size=2, stride=2)

nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5, stride=1, padding=2)
```

Image taken from: https://en.wikipedia.org/wiki/LeNet

Skip Connection in CNN

```python
class SkipConnectionCNN(nn.Module):
    def __init__(self):
        super(SkipConnectionCNN, self).__init__()

        # Layer 1: Conv2d(in_channels = 3, out_channels = 16, kernel_size=3, stride=1, padding=1)
        self.conv1 = nn.Conv2d(in_channels = 3, out_channels = 16, kernel_size=3, stride=1, padding=1)
        # MaxPool2d(2, 2)
        self.pool1 = nn.MaxPool2d(2, 2)

        # Layer 2: Conv2d(16, 32, kernel_size=3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(in_channels = 16, out_channels = 32, kernel_size = 3, stride=1, padding=1)

        # MaxPool2d(2, 2)
        self.pool2 = nn.MaxPool2d(2, 2)

        # Layer 3: Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.conv3 = nn.Conv2d(in_channels = 32, out_channels = 64, kernel_size = 3, stride=1, padding=1)

        # Activation function
        self.relu = nn.ReLU()

        # Define skip connection from the output of Layer 1 (a1) to output of Layer 3 (a3)
        # Hint: a1 and a3 needs to be added and the shapes of a1 and a3 must be compatible for addition
```

Skip Connection in CNN

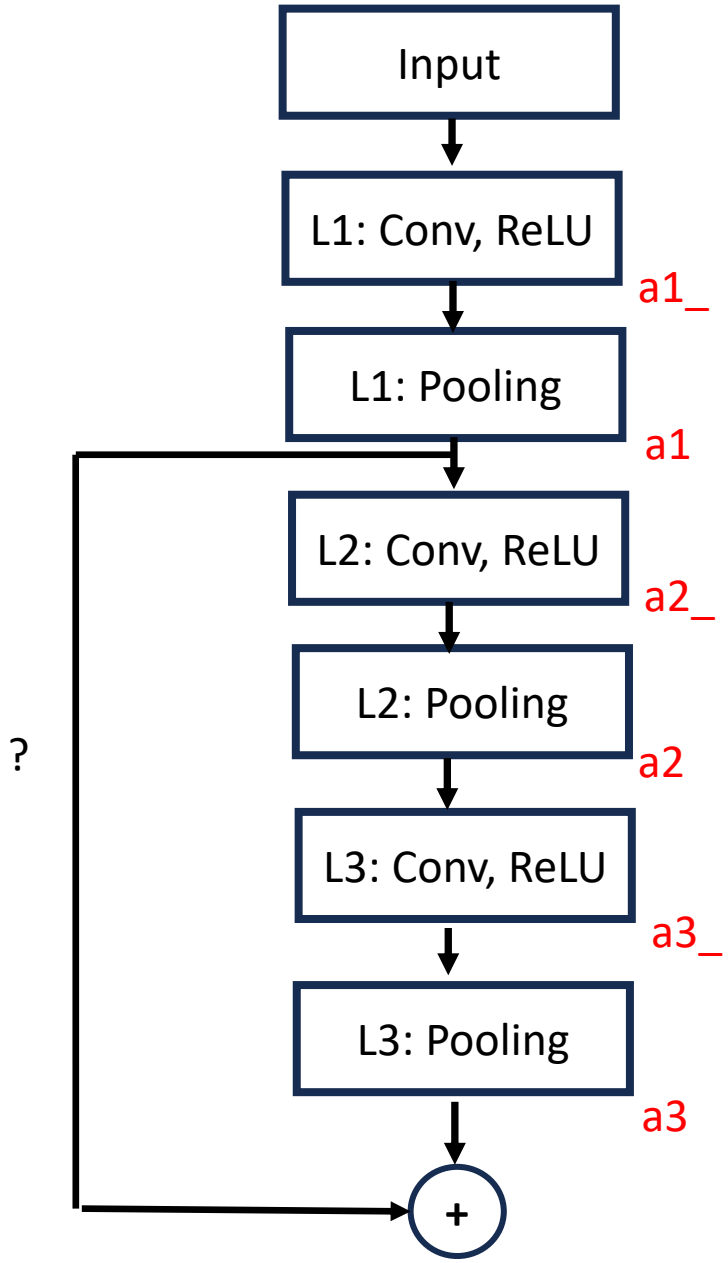
```python
def forward(self, x):

    # Main path
    #Input to Layer 1
    a1_ = self.relu(self.conv1(x))
    a1 = self.pool1(a1_)
    #Layer 1 to Layer 2
    a2_ = self.relu(self.conv2(a1))
    a2 = self.pool2(a2_)
    #Layer 2 to Layer 3
    a3_ = self.relu(self.conv3(a2))
    a3 = self.pool2(a3_)

    # Implement skip connection from Layer 1 to Layer 3
    # Combine a3 with a1
    # a3skip = a3 + a1
    # Hint: Check the shapes a1 and a3; they must be compatible for addition

    return a3skip
```

Variable	Shape
Input	32x32x3
a1_	32x32x16
a1	16x16x16
a2_	16x16x32
a2	8x8x32
a3_	8x8x64
a3	4x4x64

Skip Connection in CNN

Variable	Shape
Input	32x32x3
a1_	32x32x16
a1	16x16x16
a2_	16x16x32
a2	8x8x32
a3_	8x8x64
a3	4x4x64

Dimension mismatch!
How to overcome?

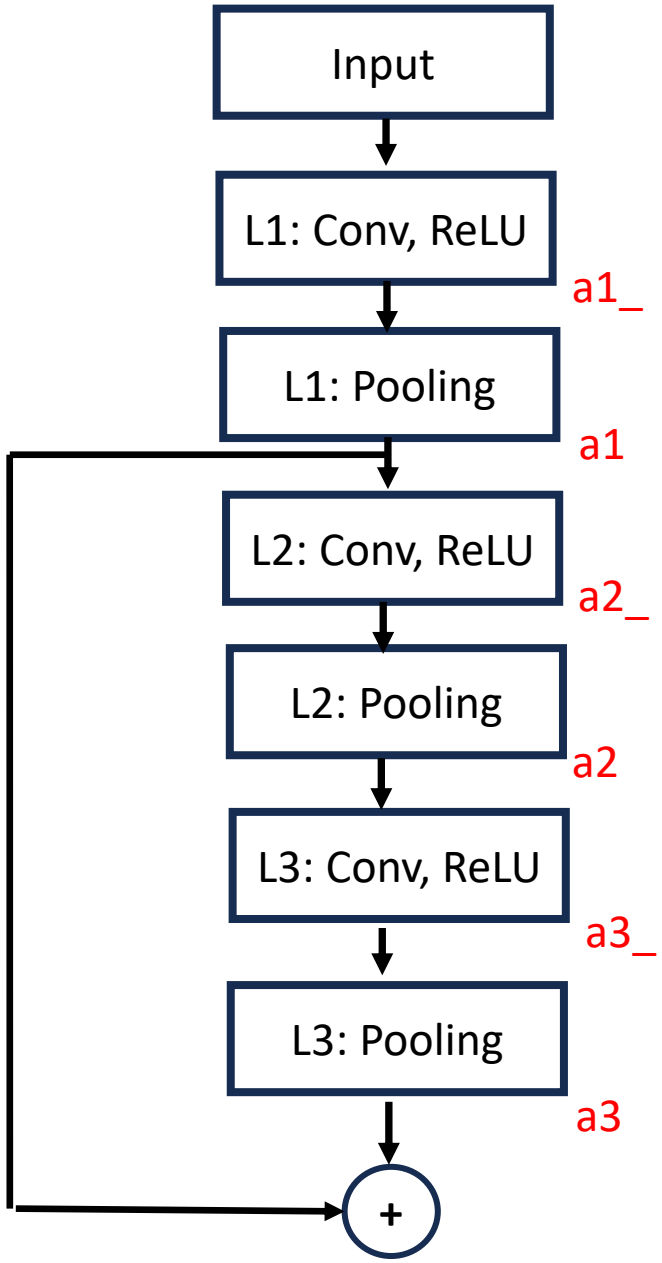



```
self.SkipConv = nn.Conv2d(in_channels = 16, out_channels = 64, kernel_size = 1)
```

Skip Connection in CNN

Variable	Shape
Input	32x32x3
a1_	32x32x16
a1	16x16x16
a2_	16x16x32
a2	8x8x32
a3_	8x8x64
a3	4x4x64

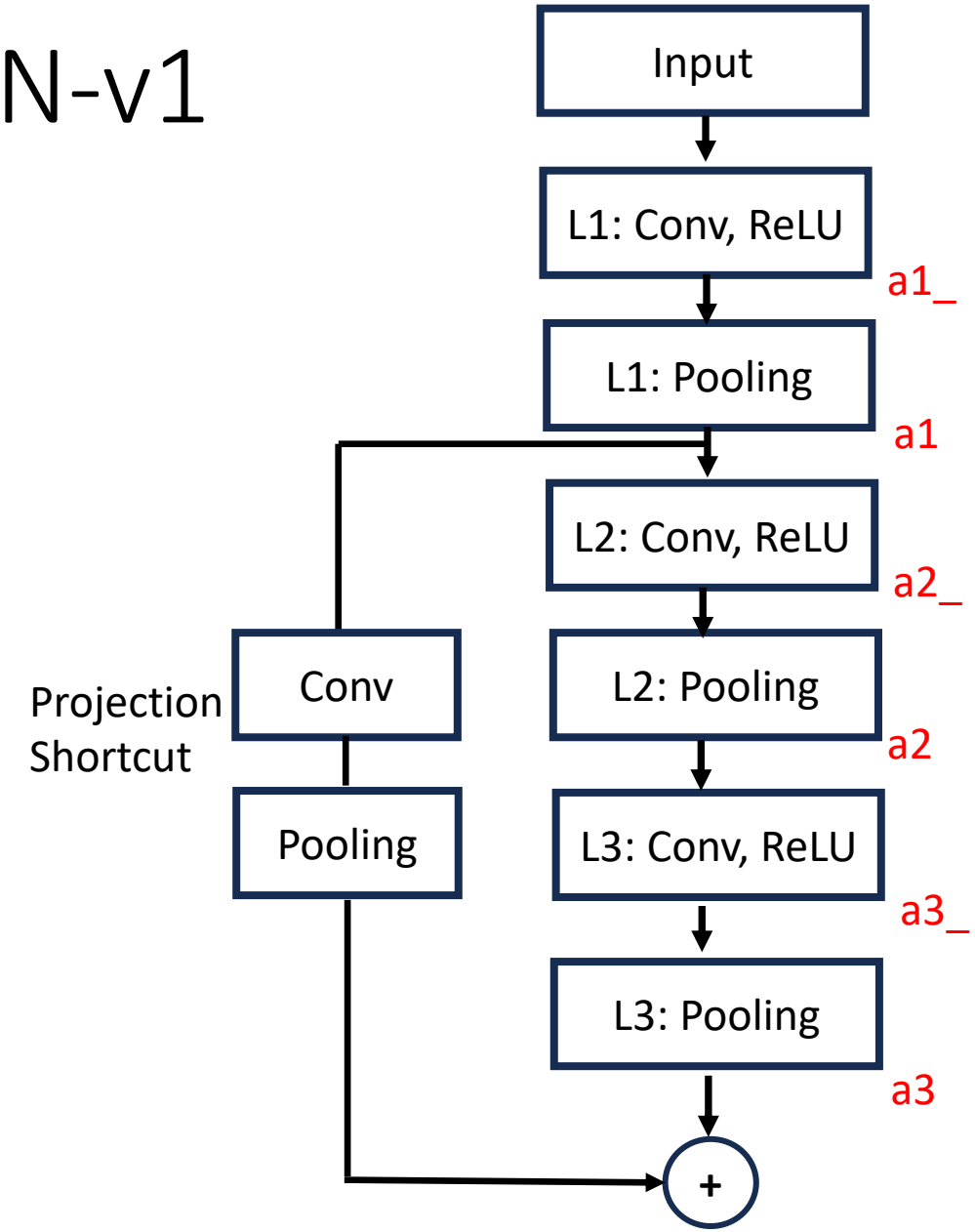
Dimension mismatch!
How to overcome?
Projection Shortcut




```
self.SkipConv = nn.Conv2d(in_channels = 16, out_channels = 64, kernel_size = 1)
```

Skip Connection in CNN-v1

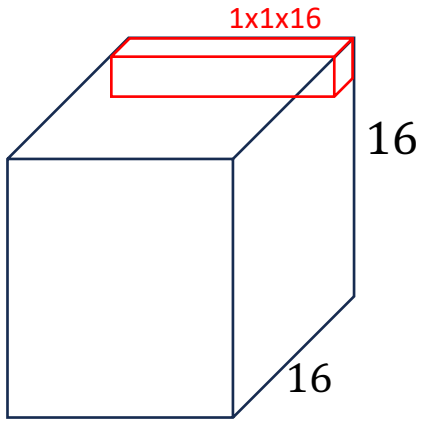
Variable	Shape
Input	32x32x3
a1_	32x32x16
a1	16x16x16
a2_	16x16x32
a2	8x8x32
a3_	8x8x64
a3	4x4x64

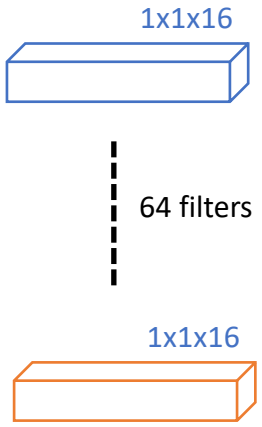


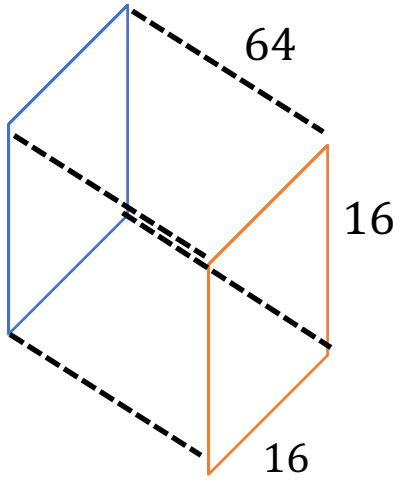

```
self.SkipConv = nn.Conv2d(in_channels = 16, out_channels = 64, kernel_size = 1)
self.SkipPool = nn.MaxPool2d(4,4)
```

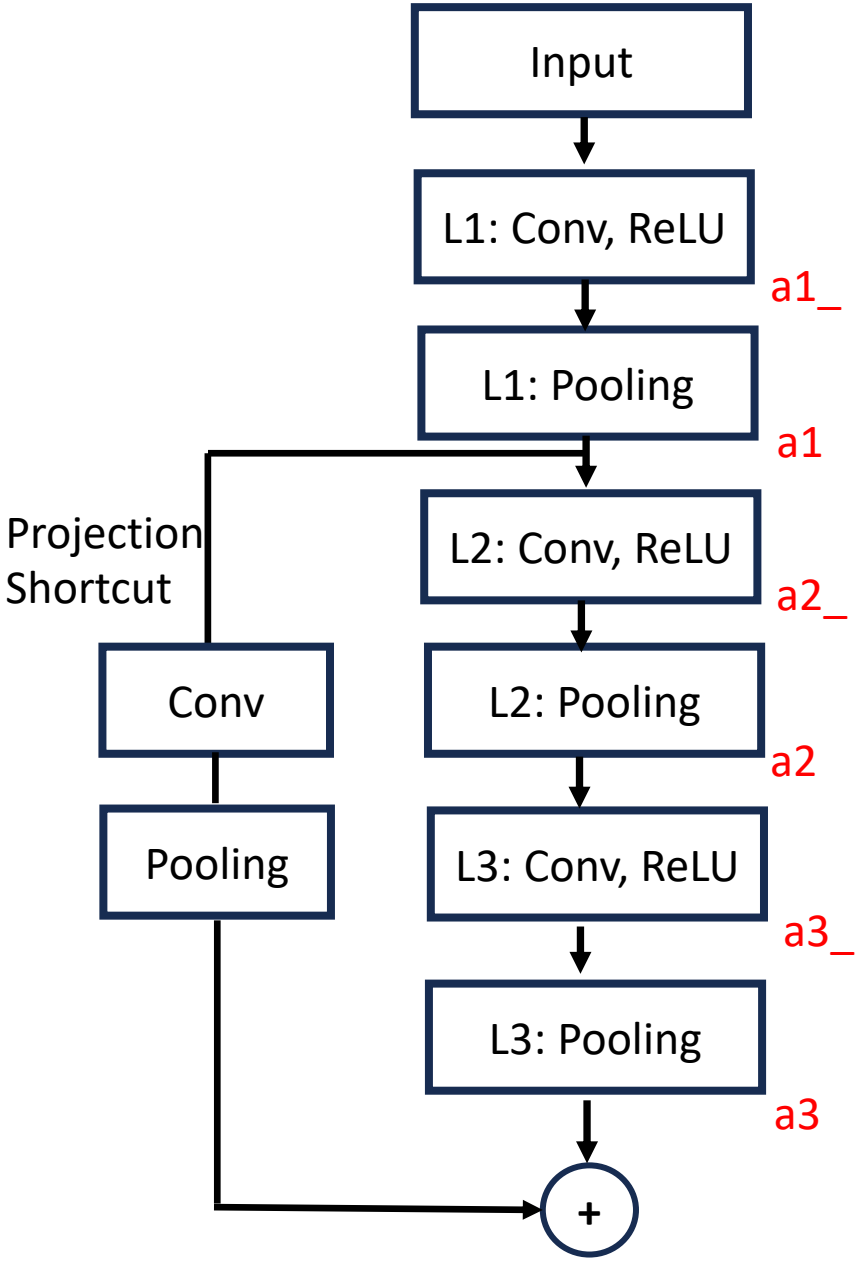
Skip Connection in CNN-v1

Variable	Shape
a1	16x16x16
a3	4x4x64





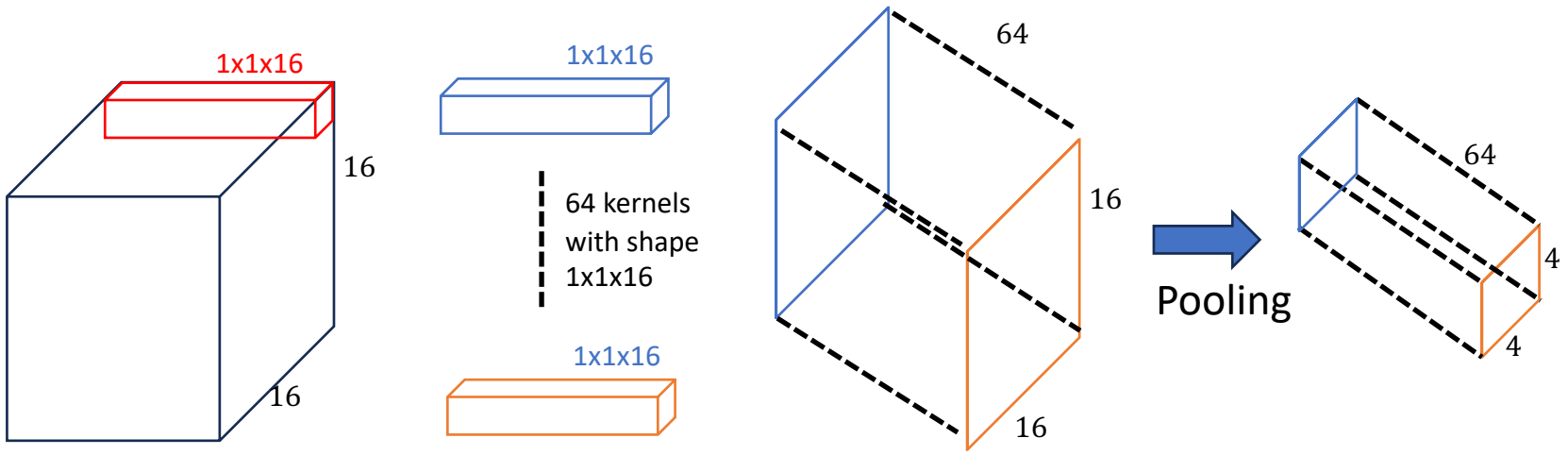


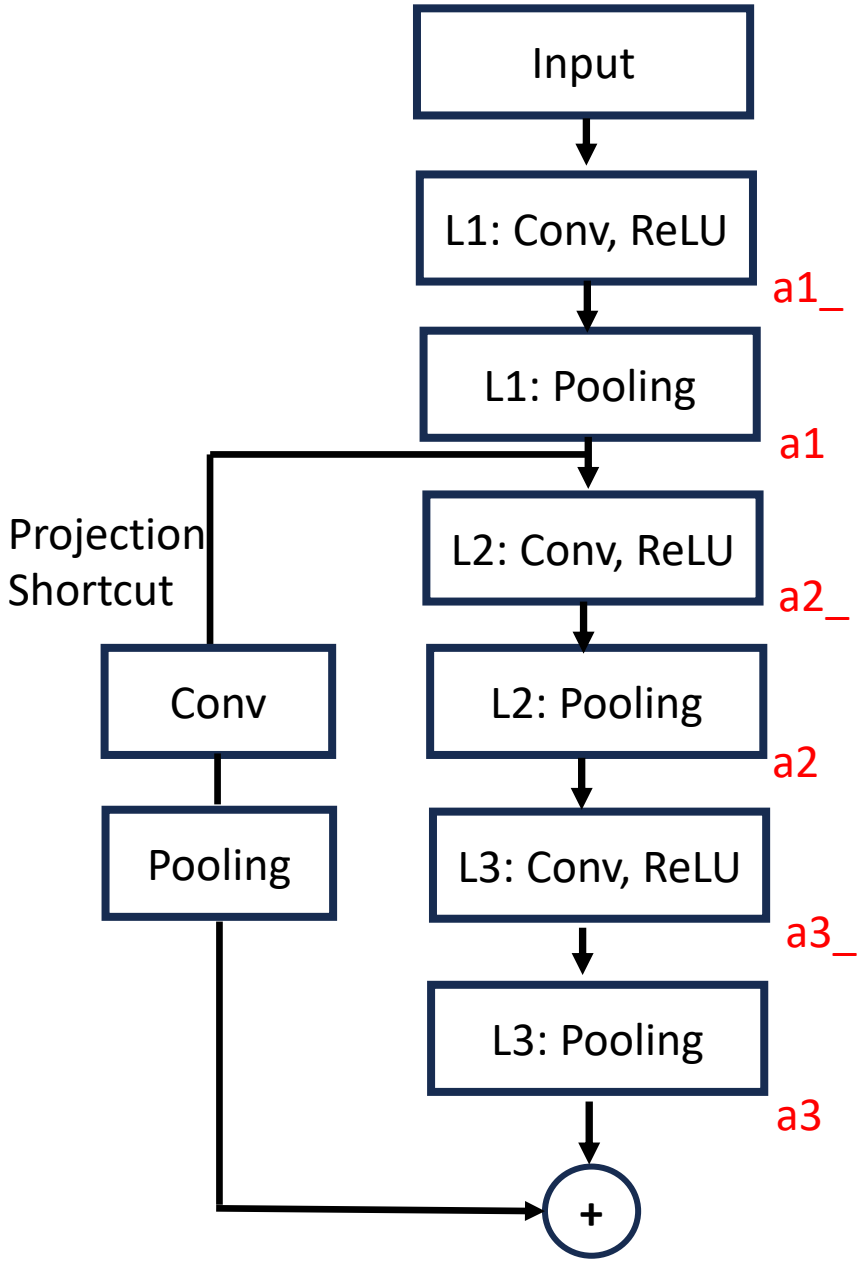



```
self.SkipConv = nn.Conv2d(in_channels = 16, out_channels = 64, kernel_size = 1)
```

Skip Connection in CNN-v1

Variable	Shape
a1	16x16x16
a3	4x4x64

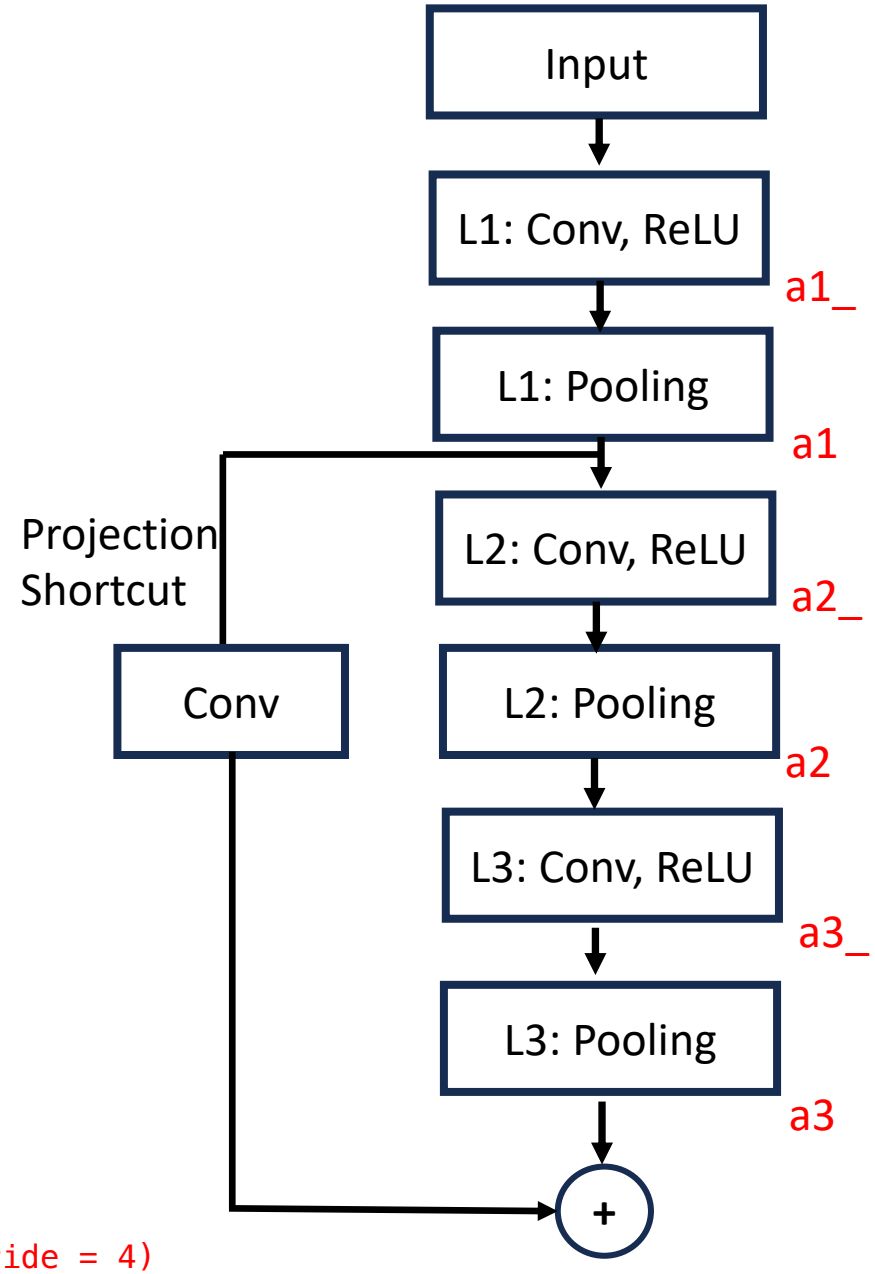




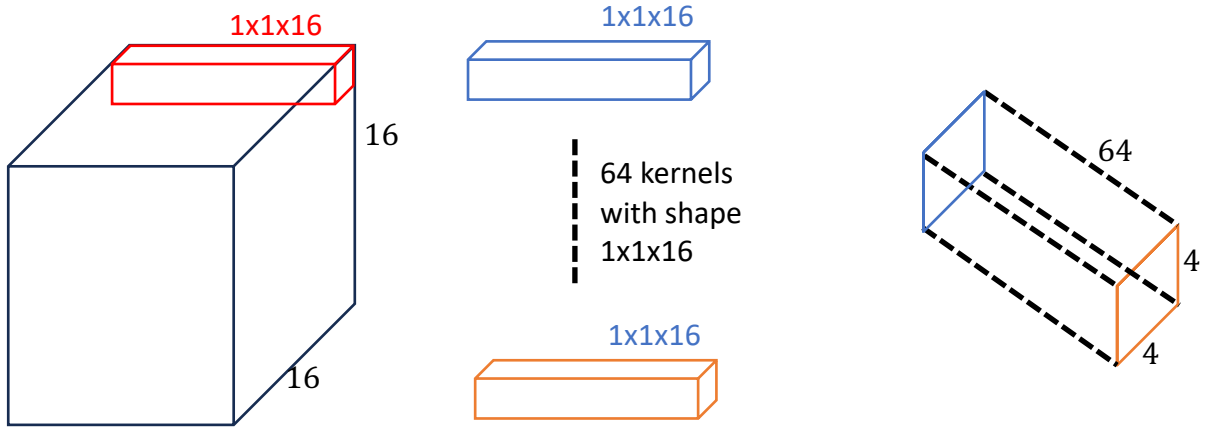

```
self.SkipConv = nn.Conv2d(in_channels = 16, out_channels = 64, kernel_size = 1)

self.SkipPool = nn.MaxPool2d(4,4)
```

Skip Connection in CNN-v2

Variable	Shape
a1	16x16x16
a3	4x4x64



Strided Convolution




```
self.SkipConv = nn.Conv2d(in_channels = 16, out_channels = 64, kernel_size = 1, stride = 4)
```

Skip Connections

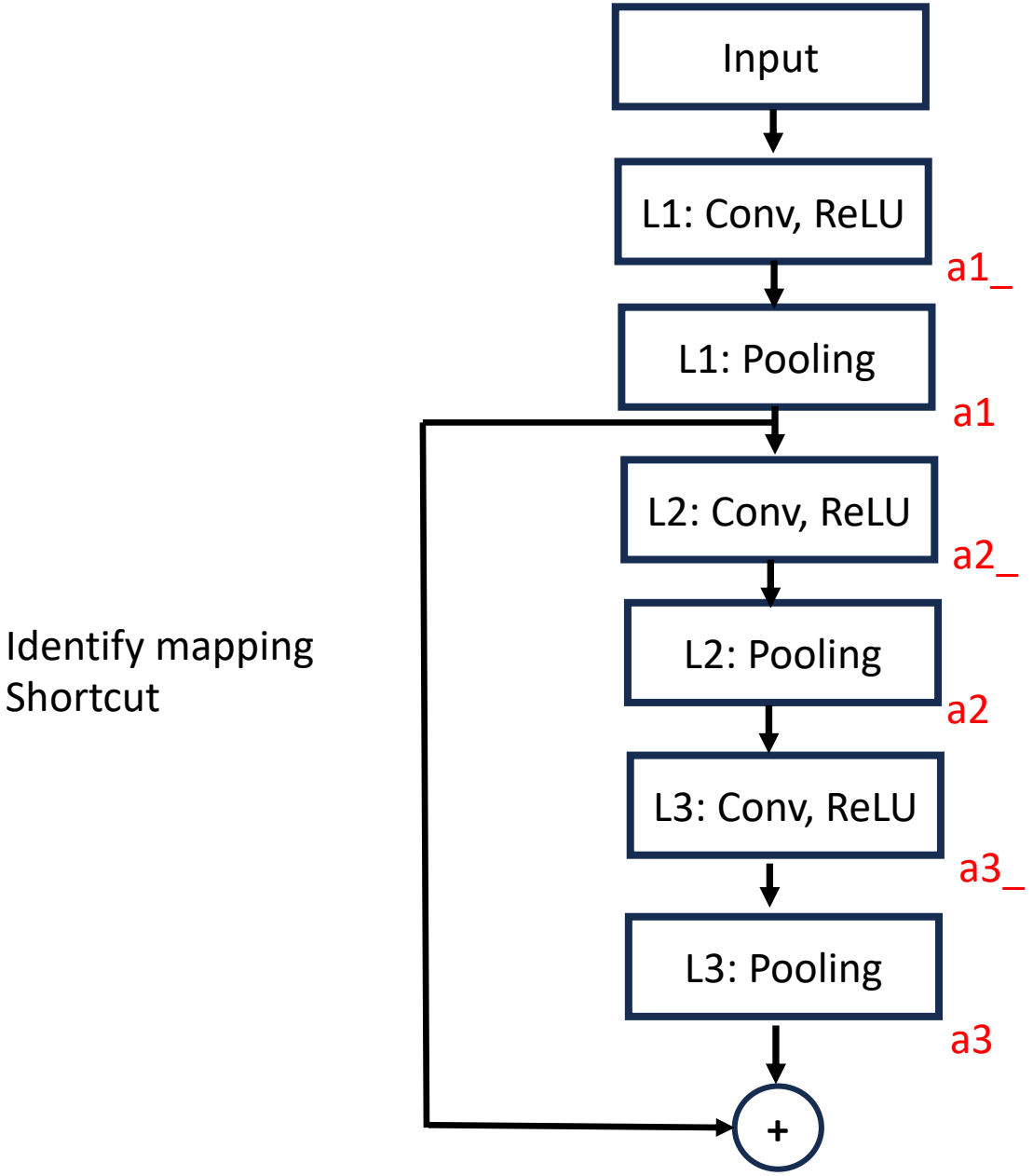
- Why a kernel size of 1?
  - Preserve spatial information in skip connections
  - Parameter-efficient
    - Minimal number of weights

Variable	Shape
a1	16x16x16
a3	4x4x64

Kernel_Size	Parameter Count
1	16x64x1x1
3	16x64x3x3

Skip Connections in CNN

- What if dimensions match?
  - Identity shortcut



Bottleneck Design

- Aiken wants to perform convolution on an input with 256 channels. The objective is to get an output with 256 channels with same height and width as input. To this end, he proposed a convolution layer with 3x3 kernel with 256 channels. The stride and padding are selected to maintain the same width and height as input. However, Dueet argued that this architecture is parameter-inefficient and instead he proposed an alternative architecture named bottleneck:
  1. A 1x1 convolution that reduces the channels to 64
  2. A 3x3 convolution that maintains 64 channels
  3. A 1x1 convolution that expands the channels to 256.

A network is called parameter-efficient if it has a smaller number of parameters. Prove/disprove that Dueet's bottleneck architecture is efficient.

- # of Parameters = $(kernel\_size * kernel\_size * in\_channels * out\_channels) + \#\ of\ bias\ terms$
- **Aiken's Design**:

  Number of Parameters = 3*3*256*256+256 = 590080
- **Dueet's design**:
  - # of Parameters in first 1x1 convolution layer = 1*1*256*64+64=16448
  - # of Parameters in first 3x3 convolution layer = 3*3*64*64+64=36928
  - # of Parameters in second 1x1 convolution layer = 1*1*64*256+256=16640
  - Number of parameters in Dueet's design = 16448+36928+16640 = 70016

- Dueet's bottle neck is parameter-efficient