Assignment 4: ResNet

[Objective]

Your model should classifiy of the images into 10 classes.

[Requirements]

1. Implement ResNet32 model with Pytorch or Tensorflow.

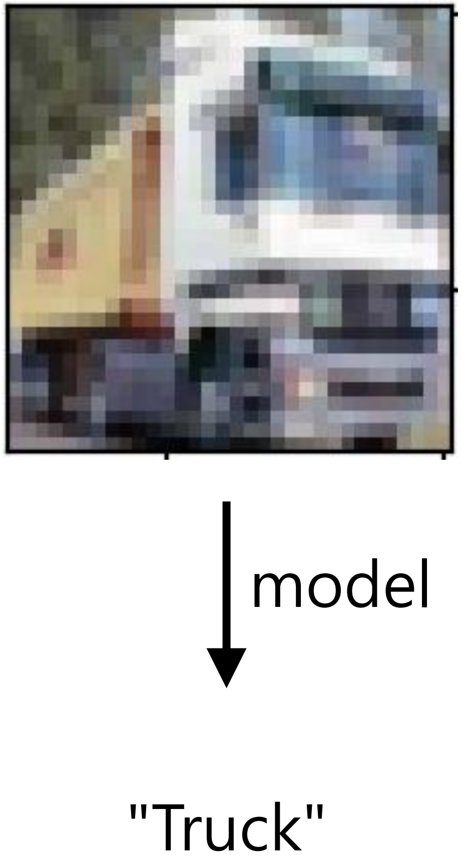
(Basic code is provided)

2. You should experiment with settings stated in the evaluation report, and report the result of each settings.

3. You should attach the plot of the validation dataset accuracy plot.

4. You should report the experimental results.

(all kinds of additional experiments are recommended)



model

"Truck"

Code review

[Objective]

Your model should classifiy of the images into 10 classes.

[Classes]

classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

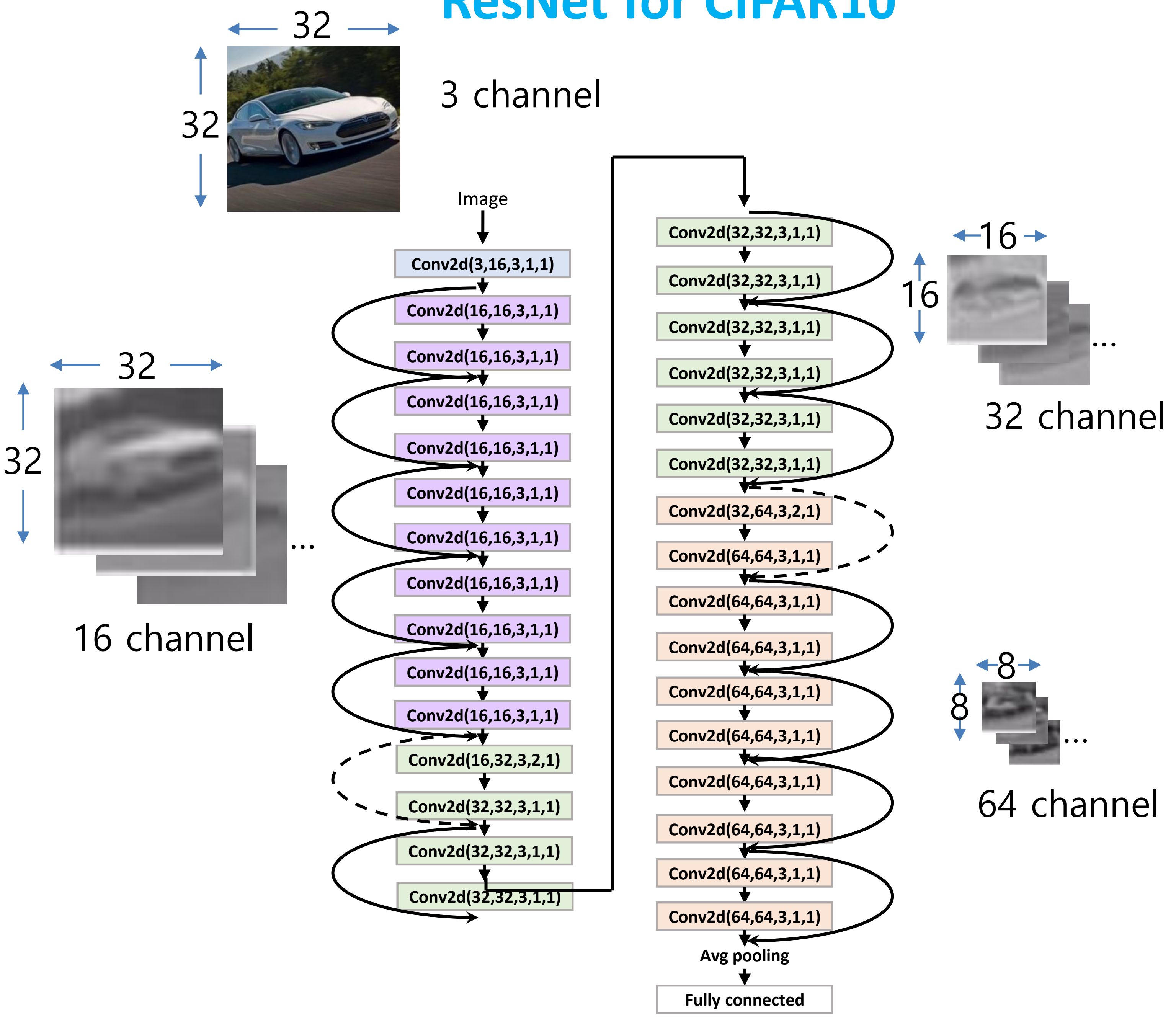
[PyTorch Code structure]

- ResNet_model.py
- ResNet_train.py
- ResNet_evaluation.py
- ResNet_infer.py

[TensorFLow Code structure]

- resnet.py
- resnet_train.py
- resnet_eval.py
- resnet_infer.py
- data_helpers.py

ResNet for CIFAR10



3 channel

16 channel

32 channel

64 channel

Image

Conv2d(3,16,3,1,1)

Conv2d(16,16,3,1,1)

Conv2d(16,16,3,1,1)

Conv2d(16,16,3,1,1)

Conv2d(16,16,3,1,1)

Conv2d(16,16,3,1,1)

Conv2d(16,16,3,1,1)

Conv2d(16,16,3,1,1)

Conv2d(16,16,3,1,1)

Conv2d(16,16,3,1,1)

Conv2d(16,16,3,1,1)

Conv2d(16,32,3,2,1)

Conv2d(32,32,3,1,1)

Conv2d(32,32,3,1,1)

Conv2d(32,32,3,1,1)

Conv2d(32,32,3,1,1)

Conv2d(32,32,3,1,1)

Conv2d(32,32,3,1,1)

Conv2d(32,32,3,1,1)

Conv2d(32,32,3,1,1)

Conv2d(32,32,3,1,1)

Conv2d(32,64,3,2,1)

Conv2d(64,64,3,1,1)

Conv2d(64,64,3,1,1)

Conv2d(64,64,3,1,1)

Conv2d(64,64,3,1,1)

Conv2d(64,64,3,1,1)

Conv2d(64,64,3,1,1)

Conv2d(64,64,3,1,1)

Conv2d(64,64,3,1,1)

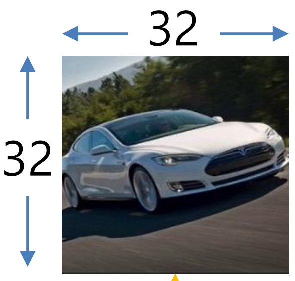
Conv2d(64,64,3,1,1)

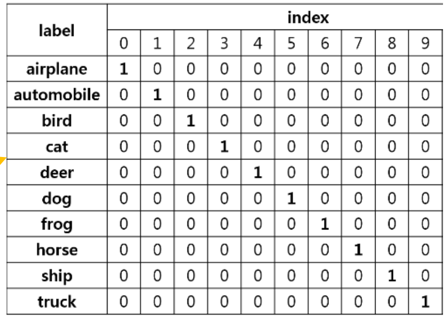
Avg pooling

Fully connected

ResNet input - TensorFlow

```
import tensorflow as tf
import numpy as np
from tensorflow.python.training import moving_averages

class ResNet:
    def __init__(self, config):

        self._num_residual_units = config.num_residual_units
        self._batch_size = config.batch_size
        self._relu_leakiness = config.relu_leakiness
        self._num_classes = config.num_classes
        self._l2_reg_lambda = config.l2_reg_lambda

        self.X = tf.placeholder(tf.float32, [None, 32, 32, 3], name="X")
        self.Y = tf.placeholder(tf.float32, [None, self._num_classes], name="Y")
        self.extra_train_ops = []
```





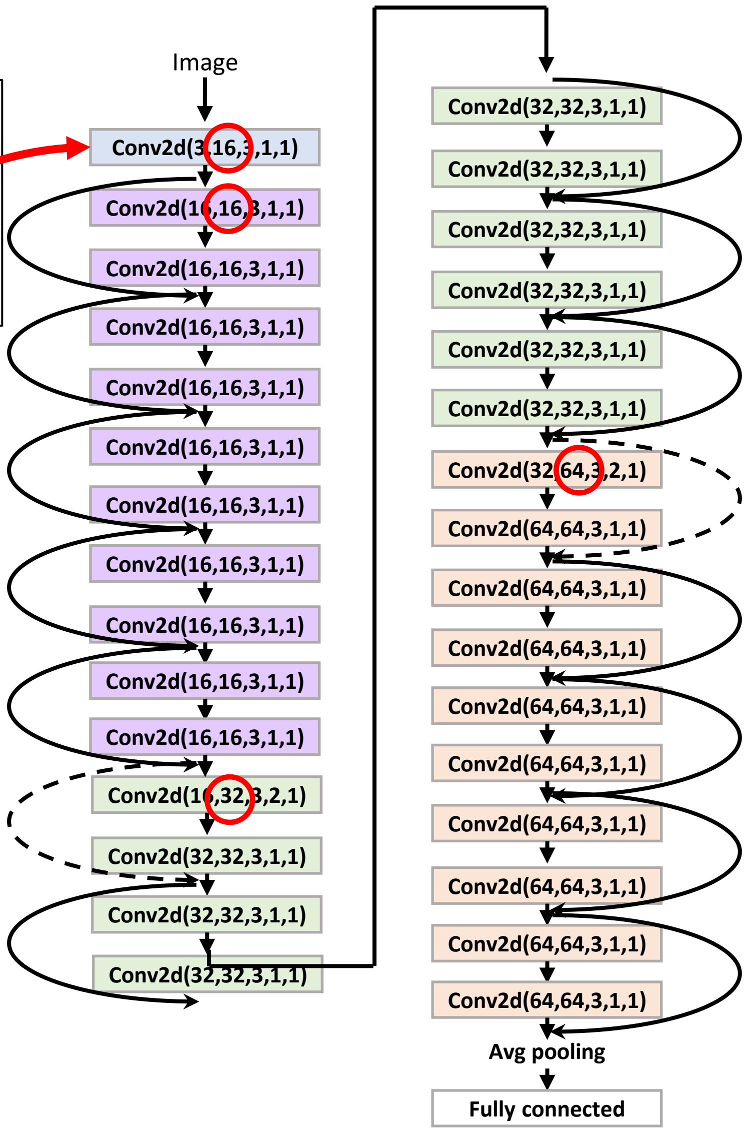
label	index									
	0	1	2	3	4	5	6	7	8	9
airplane	**1**	0	0	0	0	0	0	0	0	0
automobile	0	**1**	0	0	0	0	0	0	0	0
bird	0	0	**1**	0	0	0	0	0	0	0
cat	0	0	0	**1**	0	0	0	0	0	0
deer	0	0	0	0	**1**	0	0	0	0	0
dog	0	0	0	0	0	**1**	0	0	0	0
frog	0	0	0	0	0	0	**1**	0	0	0
horse	0	0	0	0	0	0	0	**1**	0	0
ship	0	0	0	0	0	0	0	0	**1**	0
truck	0	0	0	0	0	0	0	0	0	**1**

ResNet input - TensorFlow

```
filters = [16, ?, ?, 64]
activate_before_residual = [True, False, False]

with tf.variable_scope('init'):
  x = self._conv('init_conv', self.X, ?, ?, ?, strides=?)
```

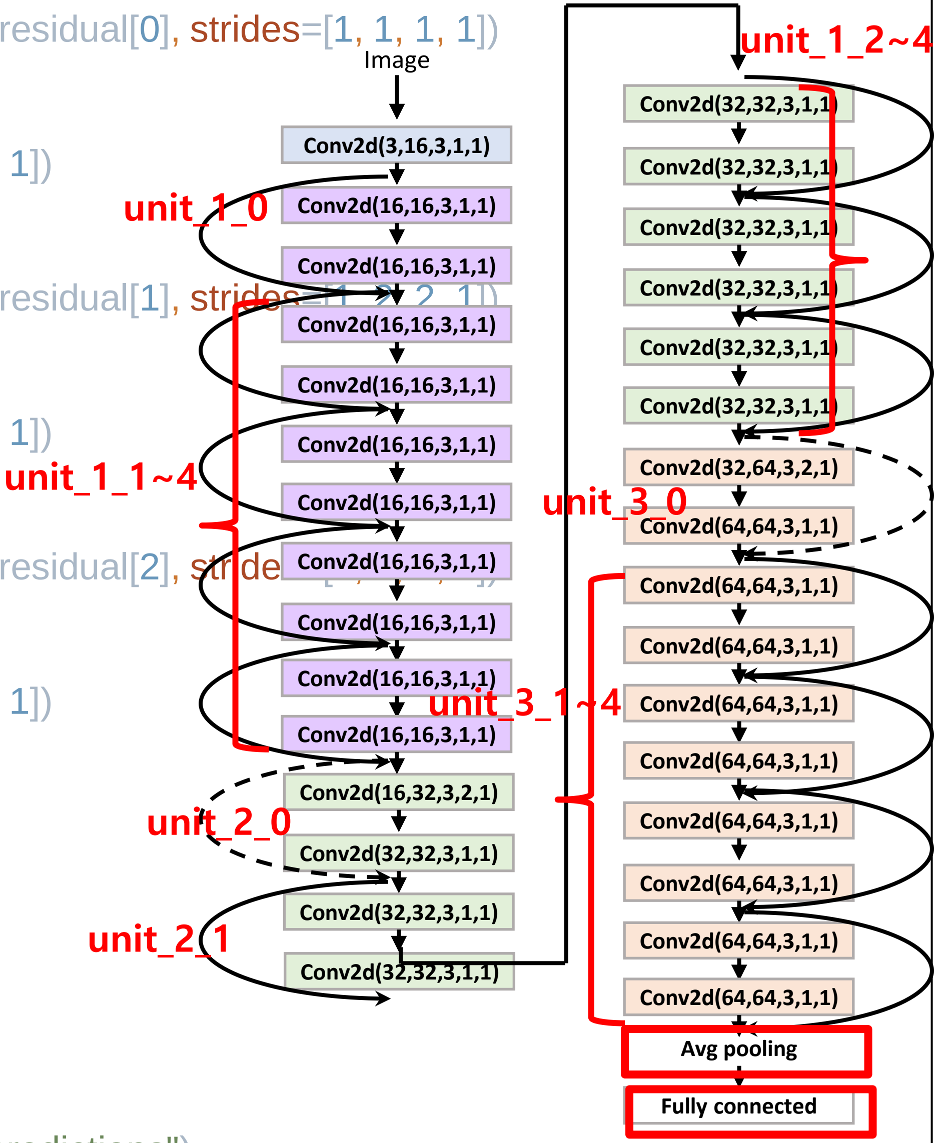
층별 filter 개수 (output channel)

최초 convolutional layer



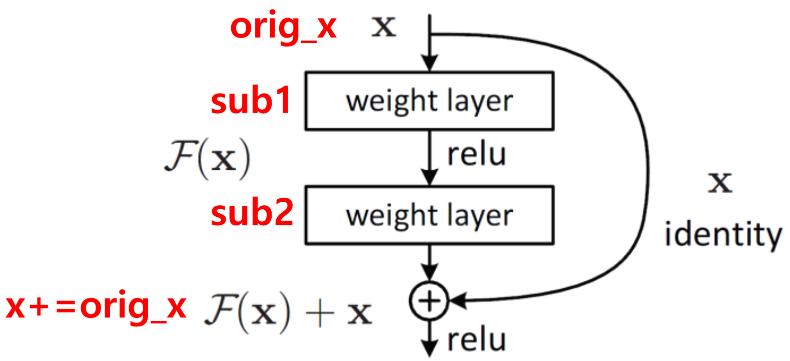
ResNet - TensorFlow

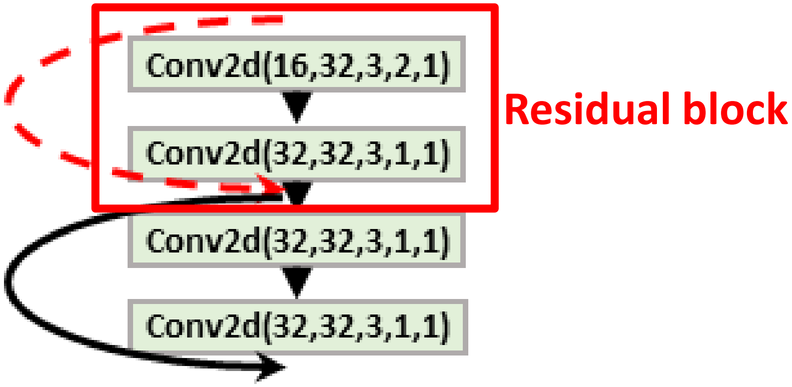
```
with tf.variable_scope('unit_1_0'):
  x = self._residual(x, filters[0], filters[1], activate_before_residual[0], strides=[1, 1, 1, 1])
for i in range(1, self._num_residual_units):
  with tf.variable_scope('unit_1_%d' % i):
    x = self._residual(x, filters[1], filters[1], strides=[1, 1, 1, 1])

with tf.variable_scope('unit_2_0'):
  x = self._residual(x, filters[1], filters[2], activate_before_residual[1], strides=[1, 2, 2, 1])
for i in range(1, self._num_residual_units):
  with tf.variable_scope('unit_2_%d' % i):
    x = self._residual(x, filters[2], filters[2], strides=[1, 1, 1, 1])

with tf.variable_scope('unit_3_0'):
  x = self._residual(x, filters[2], filters[3], activate_before_residual[2], strides=[1, 2, 2, 1])
for i in range(1, self._num_residual_units):
  with tf.variable_scope('unit_3_%d' % i):
    x = self._residual(x, filters[3], filters[3], strides=[1, 1, 1, 1])

with tf.variable_scope('unit_last'):
  x = self._batch_norm('final_bn', x)
  x = self._relu(x, self._relu_leakiness)
  x = self._global_avg_pool(x)

with tf.variable_scope('logit'):
  logits = self._fully_connected(x, self._num_classes)
  self.predictions = tf.nn.softmax(logits)
  self.predictions = tf.argmax(self.predictions, 1, name="predictions")
```

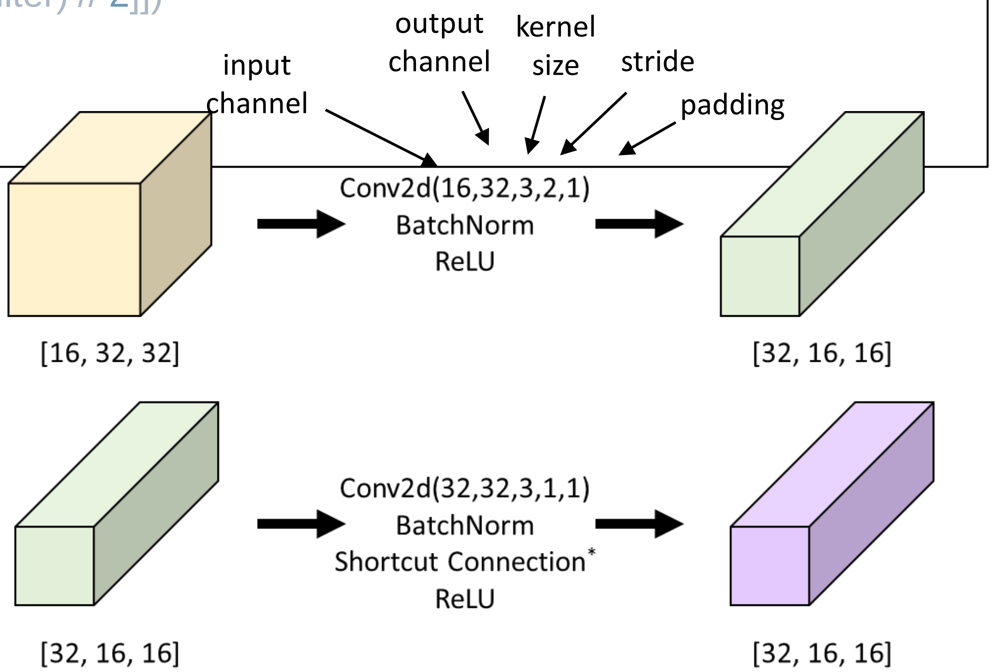


Residual block - TensorFlow

```
…
with tf.variable_scope('sub1'):
  x = self._conv('conv1', x, ?, in_filter, out_filter, ?)
with tf.variable_scope('sub2'):
  x = self._batch_norm('bn2', x)
  x = self._relu(x, self._relu_leakiness)
  x = self._conv('conv2', x, ?, out_filter, out_filter, ?)

with tf.variable_scope('sub_add'):
  if in_filter != out_filter:
          orig_x = tf.nn.avg_pool(orig_x, strides, strides, 'VALID') # pooling으로
          orig_x = tf.pad(
        orig_x, [[0, 0], [0, 0], [0, 0],
               [(out_filter - in_filter) // 2, (out_filter - in_filter) // 2]])
  x += orig_x # skip connection
tf.logging.debug('image after unit %s', x.get_shape())
return x
```

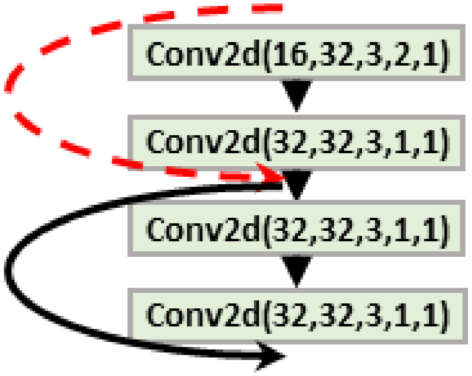


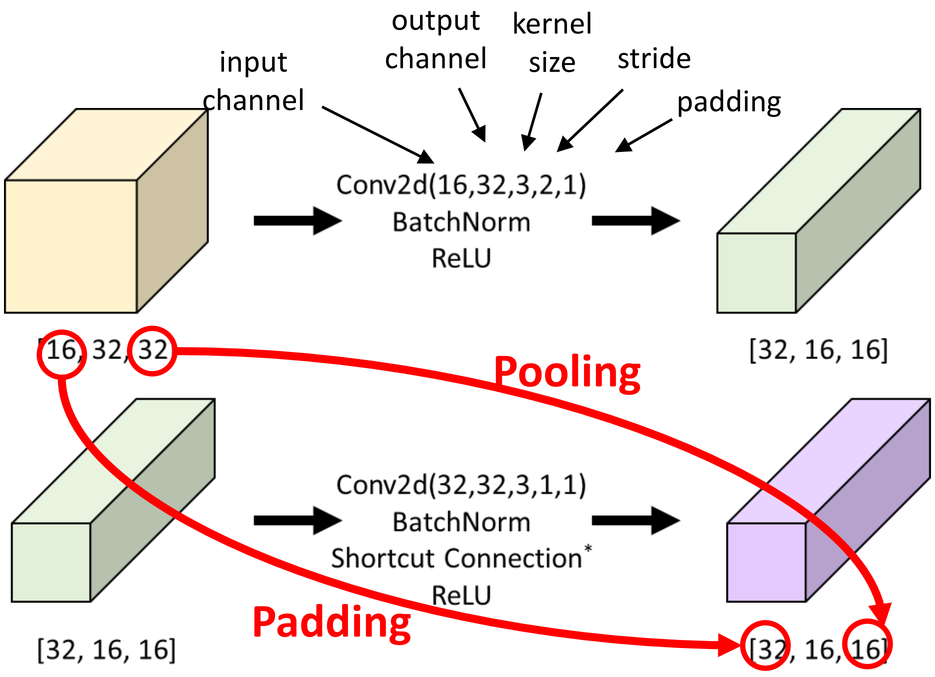




Indentity - TensorFlow

```
with tf.variable_scope('sub_add'):
    if in_filter != out_filter:
              orig_x = tf.nn.avg_pool(orig_x, strides, strides, 'VALID') # pooling으로
              orig_x = tf.pad(
            orig_x, [[0, 0], [0, 0], [0, 0],
                     [(out_filter - in_filter) // 2, (out_filter - in_filter) // 2]])
    x += orig_x # skip connection
tf.logging.debug('image after unit %s', x.get_shape())
return x
```





input channel

output channel

kernel size

stride

padding

Conv2d(16,32,3,2,1)
BatchNorm
ReLU

[16, 32, 32]

[32, 16, 16]

Pooling

Conv2d(32,32,3,1,1)
BatchNorm
Shortcut Connection*
ReLU

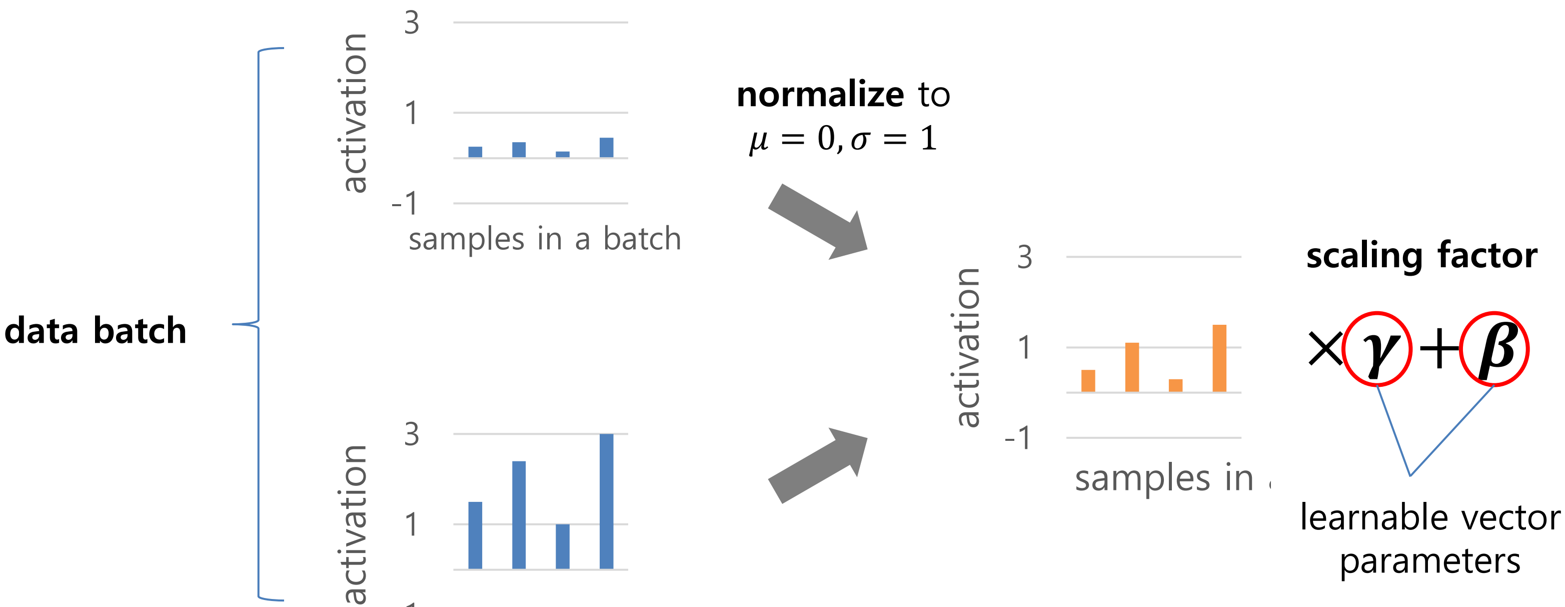
Padding

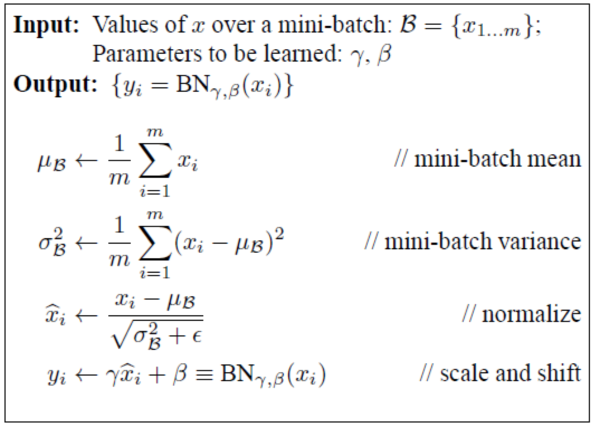
[32, 16, 16]

[32, 16, 16]

Batch normalization

- Training 과정을 안정화하여 Gradient Vanishing/Exploding 문제를 완화
- NN의 각 층마다 input의 distribution이 달라지는 문제를 해결
- 학습 안정화로 learning rate를 크게 잡을 수 있고 이로 인해 학습 속도 향상
- 자체적인 regularization 효과로 기존 regularization (dropout 등) 기법 생략 가능





Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

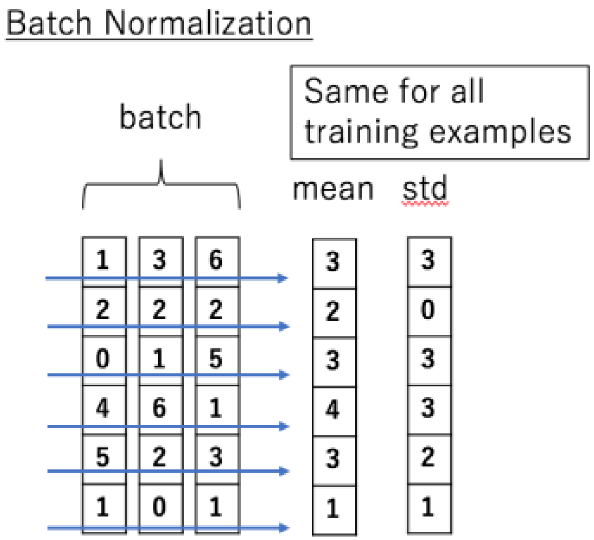
$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Batch normalization vs Layer normalization

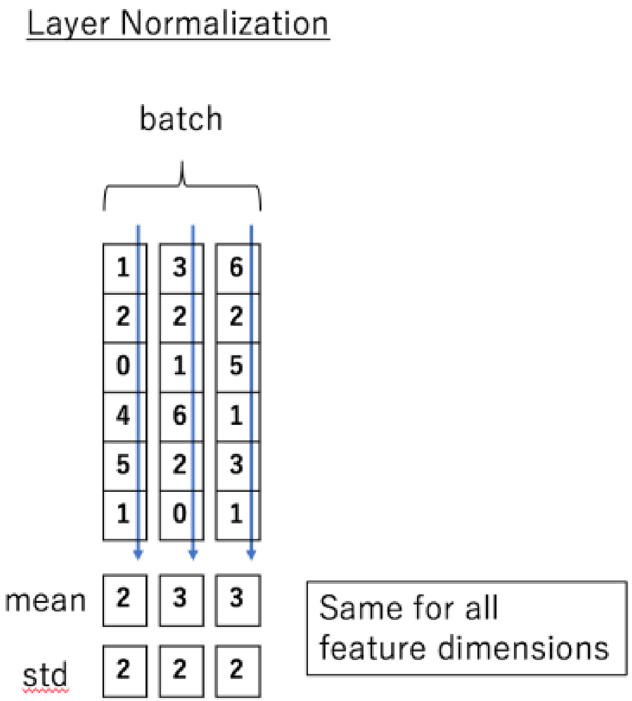


Batch Normalization

batch

Same for all training examples

			mean	std
1	3	6	3	3
2	2	2	2	0
0	1	5	3	3
4	6	1	4	3
5	2	3	3	2
1	0	1	1	1

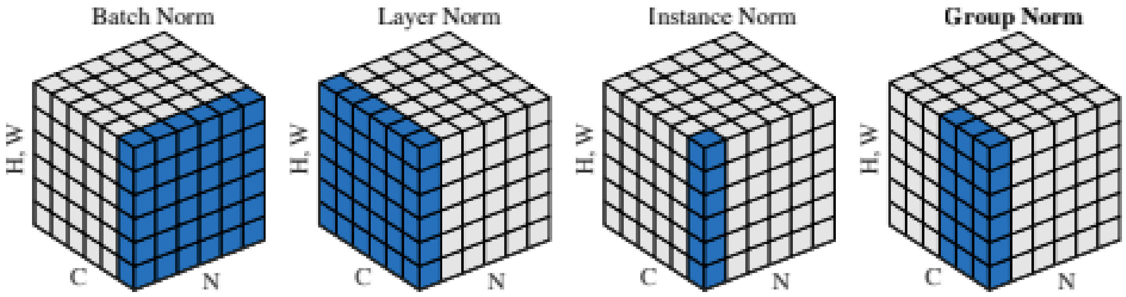


Layer Normalization

batch

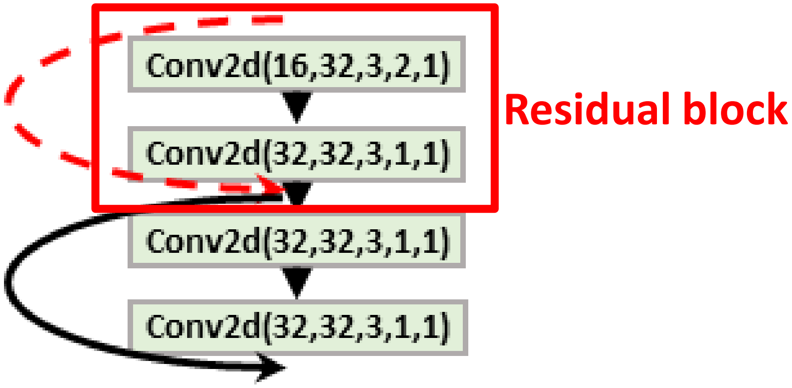
	1	3	6
	2	2	2
	0	1	5
	4	6	1
	5	2	3
	1	0	1
mean	2	3	3
std	2	2	2

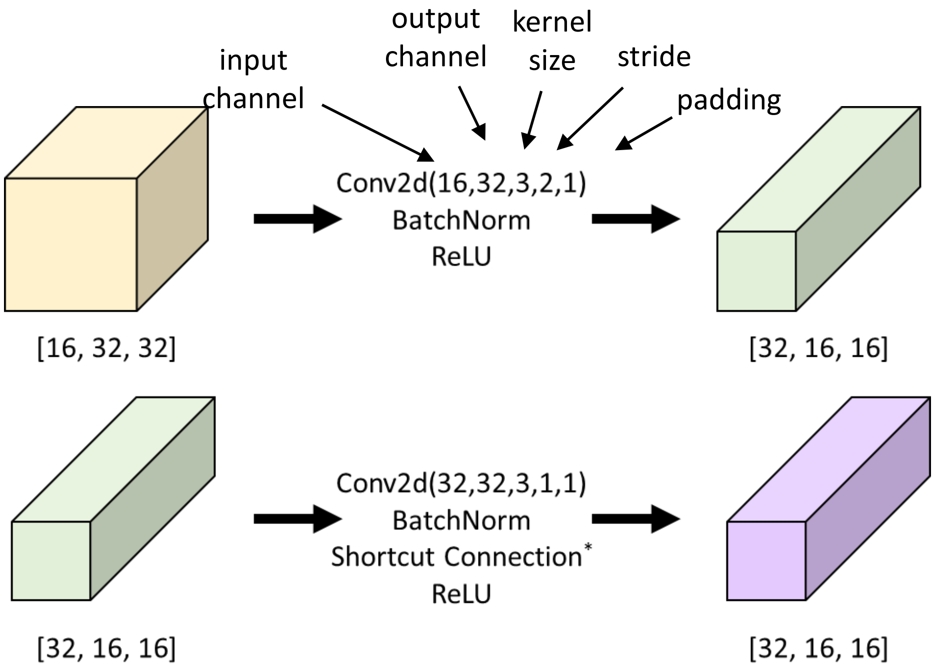
Same for all feature dimensions



Residual block - Pytorch

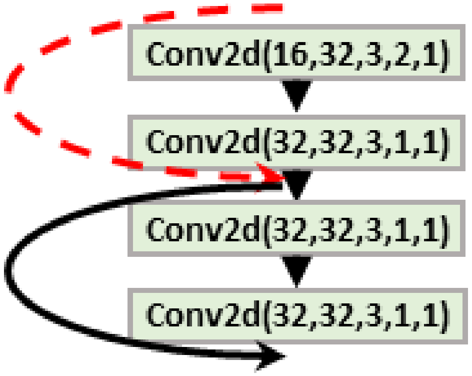
```
self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=?, stride=?, padding=?, bias=False)
self.bn1 = nn.BatchNorm2d(out_channels)
self.relu = nn.ReLU(inplace=True)

self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=?, stride=?, padding=?, bias=False)
self.bn2 = nn.BatchNorm2d(out_channels)
self.stride = stride
if down_sample:
    self.down_sample = IdentityPadding(in_channels, out_channels, stride)
else:
    self.down_sample = None
```

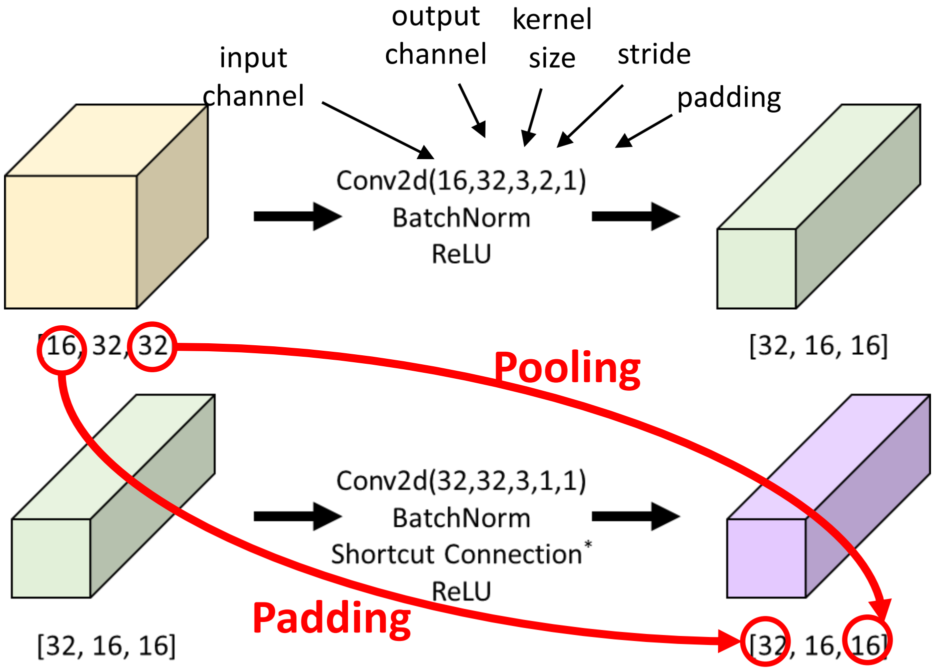




Identity - PyTorch

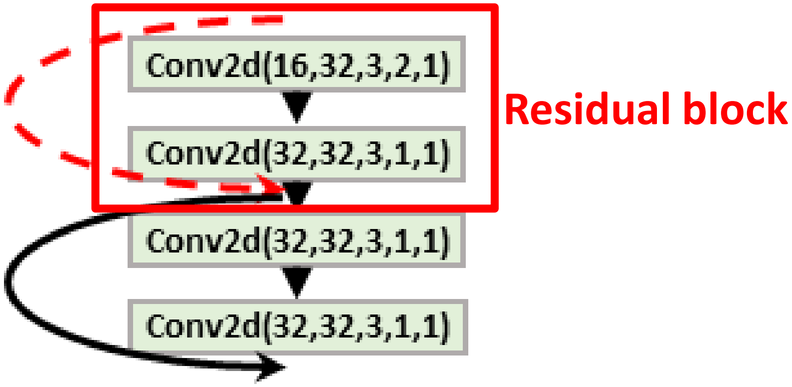
```
def __init__(self, in_channels, out_channels, stride):
    super().__init__()
    self.pooling = nn.MaxPool2d(kernel_size=1, stride=stride)
    self.add_channels = out_channels - in_channels

def forward(self, x):
    # 패딩전 x: torch.Size([200, 32, 16, 16])
    x = F.pad(x, [0, 0, 0, 0, 0, self.add_channels])
    # 패딩후 x: torch.Size([200, 64, 16, 16])
    x = self.pooling(x)
    # 풀링후 x: torch.Size([200, 64, 8, 8])
    return x
```

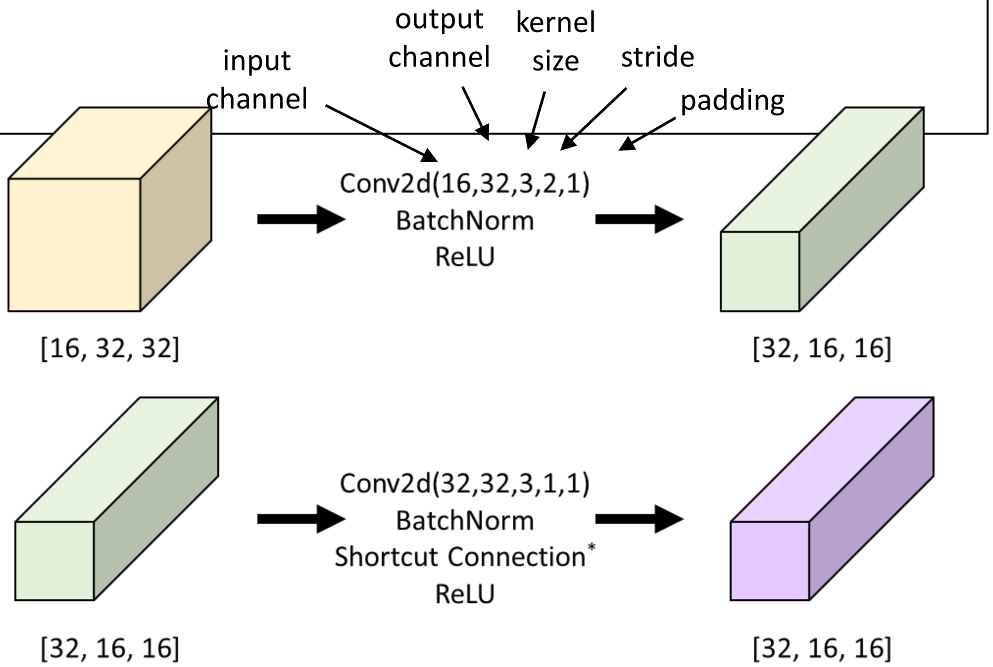




Residual block - PyTorch

```
def forward(self, x):
    shortcut = x
    x = self.conv1(x)
    x = self.bn1(x)
    x = self.relu(x)

    x = self.conv2(x)
    x = self.bn2(x)

    if self.down_sample is not None:
        shortcut = self.down_sample(shortcut)

    x += shortcut
    x = self.relu(x)
    return x
```

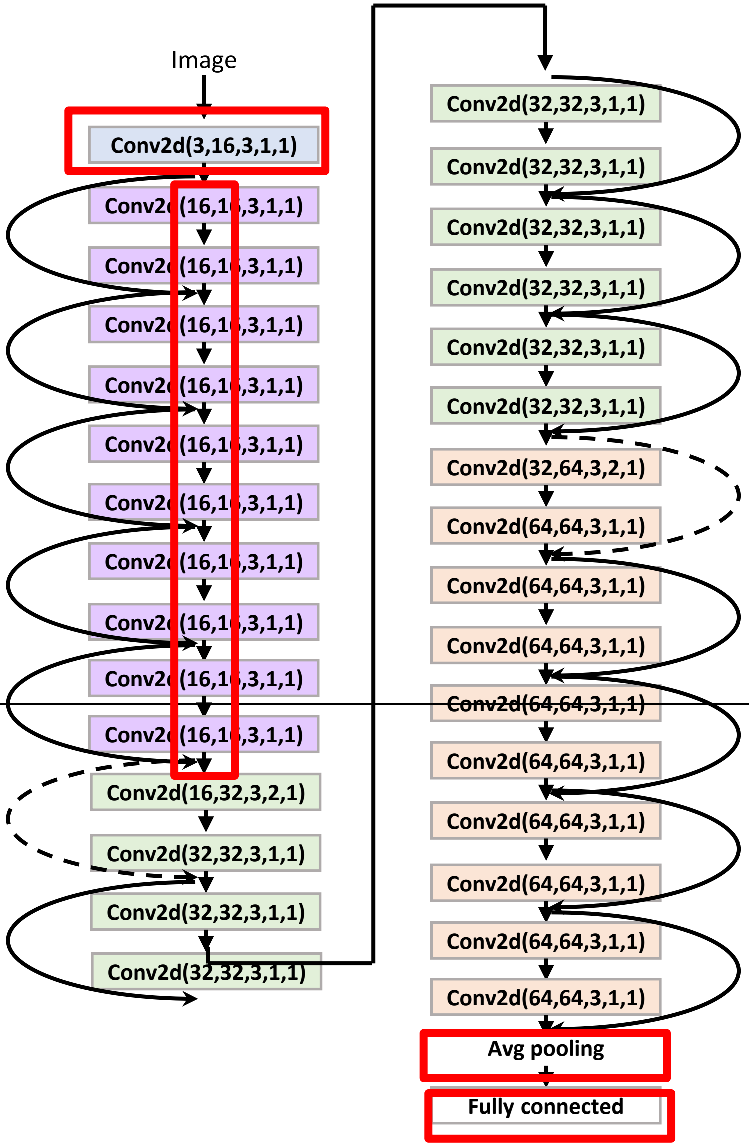




ResNet - PyTorch

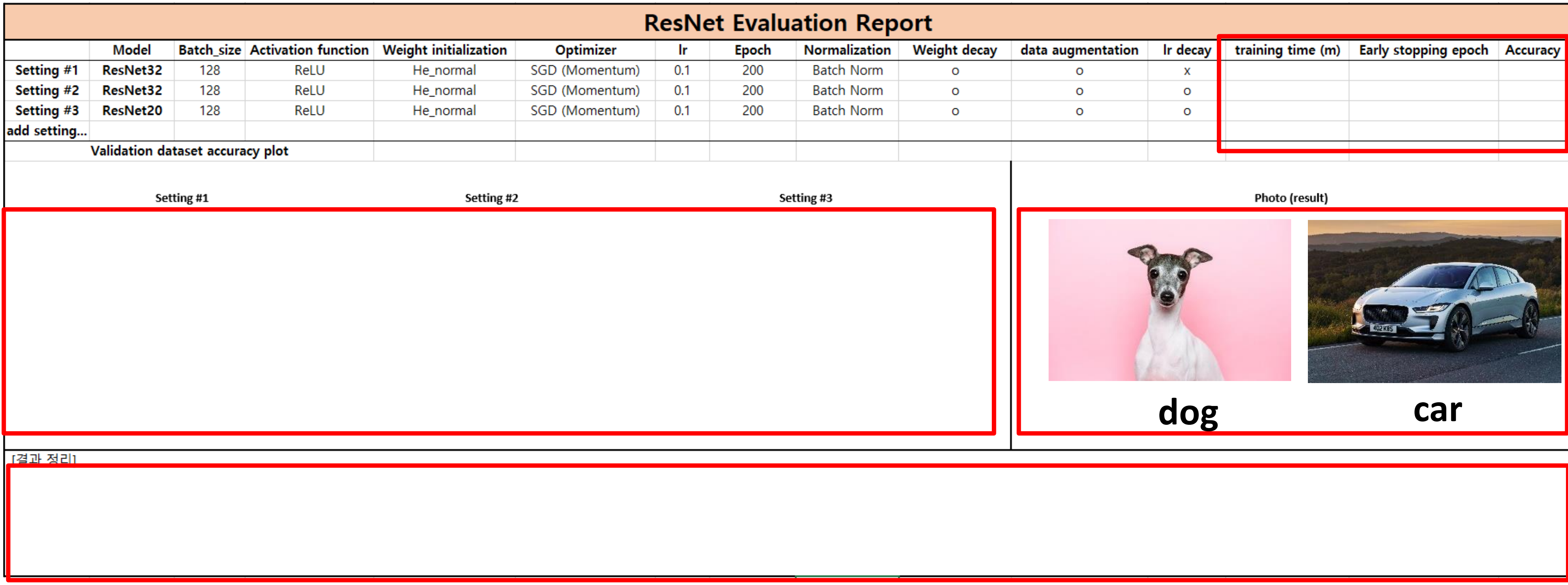
```
self.num_layers = num_layers
self.conv1 = nn.Conv2d(in_channels=?, out_channels=?,
                       kernel_size=?, padding=?, bias=False)
self.bn1 = nn.BatchNorm2d(16)
self.relu = nn.ReLU(inplace=True)

# feature map size = [16,32,32]
self.layers_2n = self.get_layers(block, 16, 16, stride=1)
# feature map size = [32,16,16]
self.layers_4n = self.get_layers(block, 16, 32, stride=2)
# feature map size = [64,8,8]
self.layers_6n = self.get_layers(block, 32, 64, stride=2)

# output layers
self.pool = nn.AvgPool2d(8, stride=1)
self.fc_out = nn.Linear(64, num_classes)
```

num_layers = 5



Assignment 4: ResNet

[Evaluation report]



ResNet Evaluation Report

	Model	Batch_size	Activation function	Weight initialization	Optimizer	lr	Epoch	Normalization	Weight decay	data augmentation	lr decay	training time (m)	Early stopping epoch	Accuracy
Setting #1	**ResNet32**	128	ReLU	He_normal	SGD (Momentum)	0.1	200	Batch Norm	o	o	x			
Setting #2	**ResNet32**	128	ReLU	He_normal	SGD (Momentum)	0.1	200	Batch Norm	o	o	o			
Setting #3	**ResNet20**	128	ReLU	He_normal	SGD (Momentum)	0.1	200	Batch Norm	o	o	o			
add setting...														

Validation dataset accuracy plot

Setting #1 **Setting #2** **Setting #3**

Photo (result)

[결과 정리]

Assignment 4: ResNet

- **Evaluation Criteria**

Simplicity	How concisely did you write the code? - 배점 7점
Performance	How well did the results of the code perform? - 배점 4점 - acc 89%이상 달성: 3점 - 개인 사진 추론: 1점
Brevity and Clarity	How concisely and clearly did you explain the results? - 배점 4점

Assignment 4: ResNet

- Due to : **~ 10.04(Sun)**

- Submission : Online submission on blackboard

- Your submission should contain
  1) The whole code of your implementation
  2) The evaluation report

- You must implement the components yourself!
- File name : StudentID_Name.zip