Dong-A Univ. (ISPL)

Main Modules for CNN Design

동아대학교 DONG-A UNIVERSITY

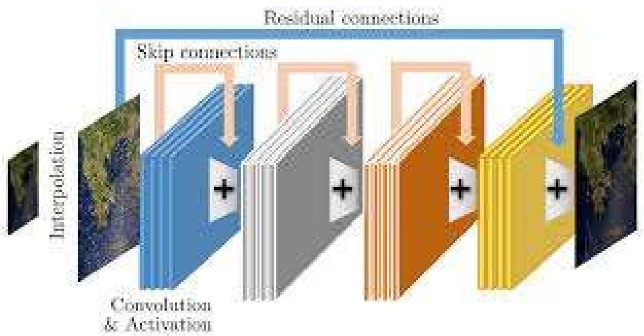
컴퓨터AI공학부 AI학과

2024년 1학기 인공지능

딥러닝 주요모델 구성요소

- **Skip connection (ResNet, 2015)**
- **Dense connection (DenseNet, 2017)**
- **Channel attention (SENet, 2018)**

❖ Skip connection



❖ Dense connection

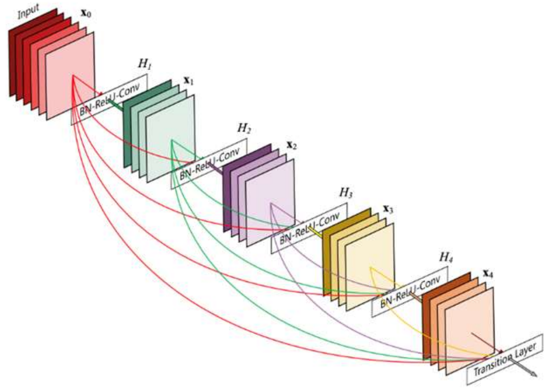
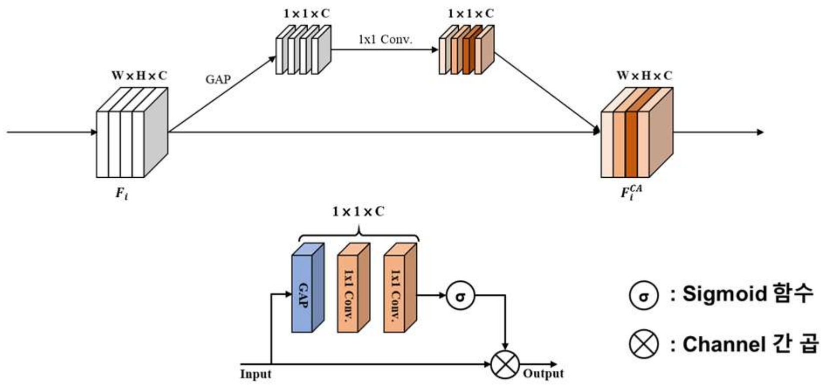


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

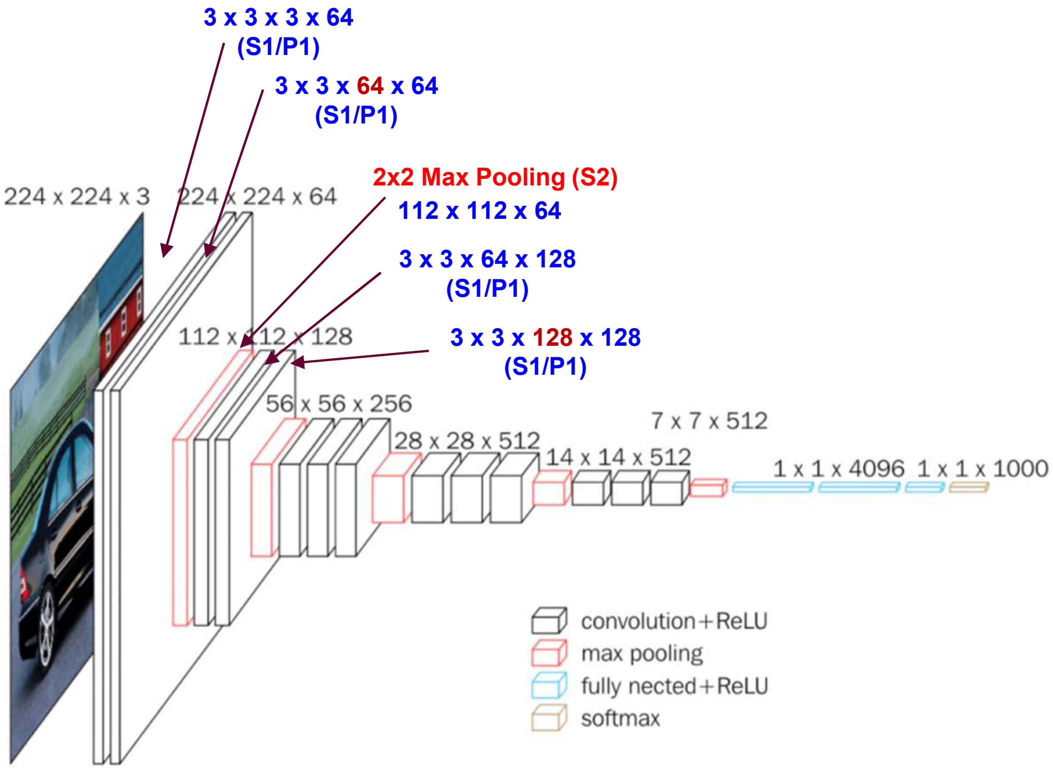
❖ Channel attention



동아대학교 DONG-A UNIVERSITY

Orig. Network - VGGNet(VGG-16)

- 기존 VGGNet을 사용하여 실습 시 많은 시간 소요 ➔ 금일 실습 시 간소화된 모델 사용



<VGG-16 구조>

Wrap-up

- **torch.nn.Conv2d() 함수를 이용한 합성곱 계층 구현**

CONV2D

CLASS torch.nn.Conv2d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *padding_mode='zeros'*, *device=None*, *dtype=None*) [SOURCE]

① in_channels: 입력 특징맵의 채널 개수

② out_channels: 출력 특징맵의 채널 개수

③ kernel_size: 커널 크기

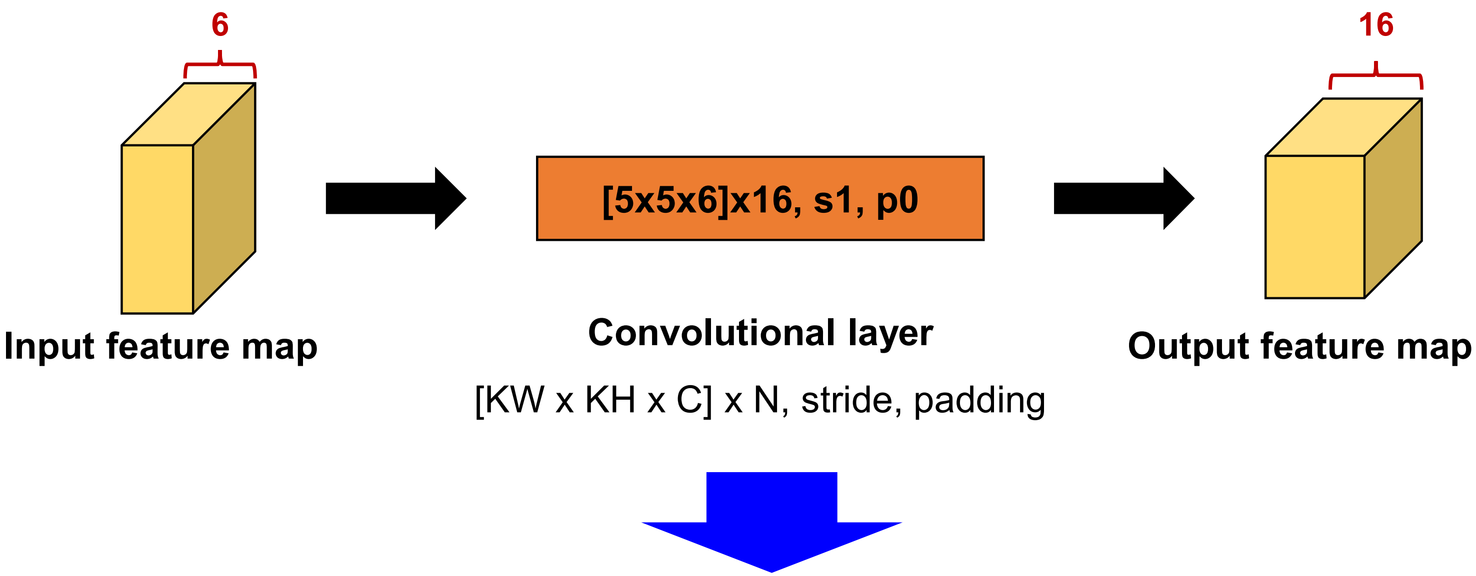
④ stride: stride 크기

⑤ padding: padding 크기

Ref.: https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html?highlight=conv2d#torch.nn.Conv2d

Wrap-up

- **torch.nn.Conv2d() 함수를 이용한 합성곱 계층 구현**



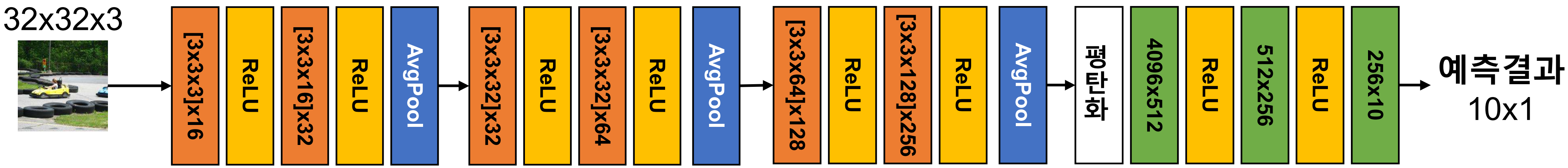
self.conv = nn.Conv2d (in_channels = 6, out_channels = 16, kernel_size = 5, stride = 1, padding = 0)

Ref.: https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html?highlight=conv2d#torch.nn.Conv2d

Modified Network - VGGNet(VGG-16)

- 기존 VGG16을 CNN layer를 6개로 간소화
- CIFAR-10 데이터셋 분류를 위해 네트워크 구조 변경 (32x32x3, 10 classes)
- 실제 VGG 네트워크는 Max Pooling을 사용

: Convolution layer
: Activation function
: Pooling layer
: Fully connected layer



CIFAR10 분류 실습 - CNN을 이용한 분류(VGGNet)

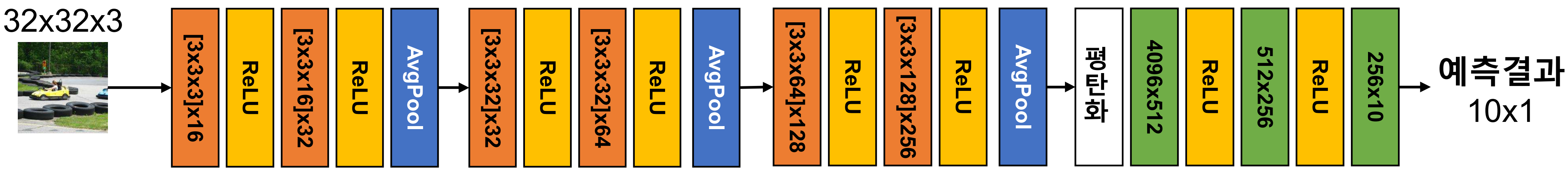
- **VGG 간소화 모델 코드 공유**
  - LMS 12주차 VGG base code 다운로드
  - 실습 시 **[3] Model 구조 선언** 부분만 수정

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

        self.conv1_1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, padding=1)      # Convolution: [3x3x3]x16, s1, p1
        self.conv1_2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, padding=1)     # Convolution: [3x3x16]x32, s1, p1

        self.conv2_1 = nn.Conv2d(in_channels=32, out_channels=32, kernel_size=3, padding=1)     # Convolution: [3x3x32]x32, s1, p1
        self.conv2_2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding=1)     # Convolution: [3x3x32]x64, s1, p1

        self.conv3_1 = nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, padding=1)    # Convolution: [3x3x64]x128, s1, p1
        self.conv3_2 = nn.Conv2d(in_channels=128, out_channels=256, kernel_size=3, padding=1)   # Convolution: [3x3x128]x256, s1, p1

        self.fc1 = nn.Linear(in_features=4096, out_features=512)    # Fully connected: 4096x512
        self.fc2 = nn.Linear(in_features=512, out_features=256)     # Fully connected: 512x256
        self.fc3 = nn.Linear(in_features=256, out_features=10)      # Fully connected: 256x10

        # 파라미터를 가지지 않은 layer는 한번만 선언해도 문제 없음
        self.relu = nn.ReLU()
        self.avgPool2d = nn.AvgPool2d(kernel_size=2, stride=2)

    def forward(self, x):

        # convolutional layers
        out = self.relu(self.conv1_1(x))
        out = self.relu(self.conv1_2(out))
        out = self.avgPool2d(out)

        out = self.relu(self.conv2_1(out))
        out = self.relu(self.conv2_2(out))
        out = self.avgPool2d(out)

        out = self.relu(self.conv3_1(out))
        out = self.relu(self.conv3_2(out))
        out = self.avgPool2d(out)

        out = torch.reshape(out, (-1, 4096)) # feature map 평탄화

        # fully connected layers
        out = self.relu(self.fc1(out))
        out = self.relu(self.fc2(out))
        out = self.fc3(out)

        return out
```

CIFAR10 분류 실습 - CNN을 이용한 분류(VGGNet)

- **CIFAR-10 데이터셋 분류를 위해 네트워크 구조 변경 (32x32x3, 10 classes)**

: Convolution layer
: Activation function
: Pooling layer
: Fully connected layer



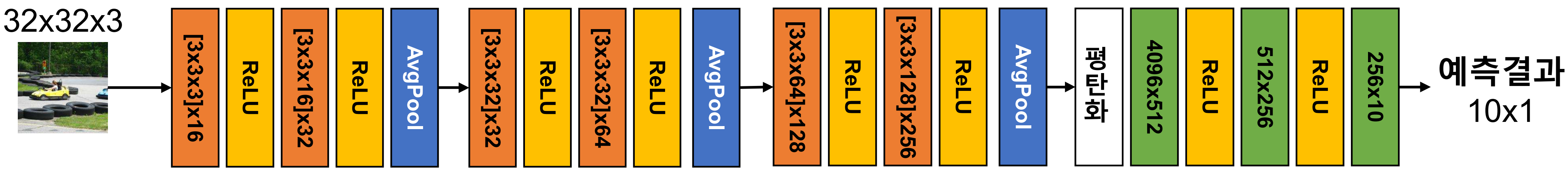
실습 Network base 구조 (Stride와 Padding size는 1로 고정)

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

        self.conv1_1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, padding=1)      # Convolution: [3x3x3]x16, s1, p1
        self.conv1_2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, padding=1)     # Convolution: [3x3x16]x32, s1, p1

        self.conv2_1 = nn.Conv2d(in_channels=32, out_channels=32, kernel_size=3, padding=1)     # Convolution: [3x3x32]x32, s1, p1
        self.conv2_2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding=1)     # Convolution: [3x3x32]x64, s1, p1

        self.conv3_1 = nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, padding=1)    # Convolution: [3x3x64]x128, s1, p1
        self.conv3_2 = nn.Conv2d(in_channels=128, out_channels=256, kernel_size=3, padding=1)   # Convolution: [3x3x128]x256, s1, p1

        self.fc1 = nn.Linear(in_features=4096, out_features=512)    # Fully connected: 4096x512
        self.fc2 = nn.Linear(in_features=512, out_features=256)     # Fully connected: 512x256
        self.fc3 = nn.Linear(in_features=256, out_features=10)      # Fully connected: 256x10

        # 파라미터를 가지지 않은 layer는 한번만 선언해도 문제 없음
        self.relu = nn.ReLU()
        self.avgPool2d = nn.AvgPool2d(kernel_size=2, stride=2)
```

CIFAR10 분류 실습 - CNN을 이용한 분류(VGGNet)

- **CIFAR-10 데이터셋 분류를 위해 네트워크 구조 변경 (32x32x3, 10 classes)**

: Convolution layer
: Activation function
: Pooling layer
: Fully connected layer



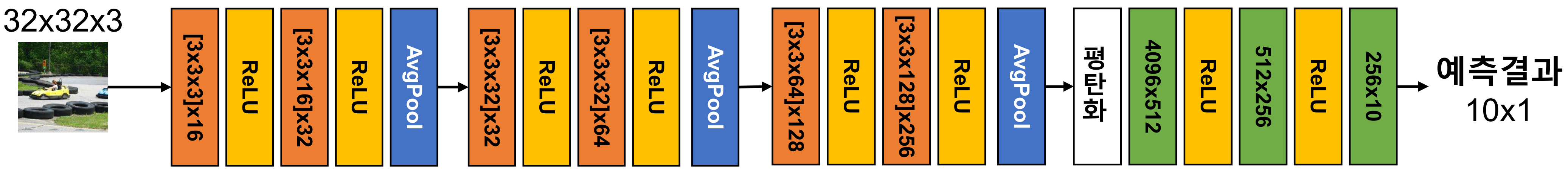
실습 Network base 구조 (Stride와 Padding size는 1로 고정)

```
    def forward(self, x):

        # convolutional layers
        out = self.relu(self.conv1_1(x))
        out = self.relu(self.conv1_2(out))
        out = self.avgPool2d(out)

        out = self.relu(self.conv2_1(out))
        out = self.relu(self.conv2_2(out))
        out = self.avgPool2d(out)

        out = self.relu(self.conv3_1(out))
        out = self.relu(self.conv3_2(out))
        out = self.avgPool2d(out)

        out = torch.reshape(out, (-1, 4096)) # feature map 평탄화

        # fully connected layers
        out = self.relu(self.fc1(out))
        out = self.relu(self.fc2(out))
        out = self.fc3(out)

        return out
```

CIFAR10 분류 실습 - CNN을 이용한 분류(VGGNet)

- **CIFAR-10 데이터셋 분류를 위해 네트워크 구조 변경 (32x32x3, 10 classes)**

: Convolution layer
: Activation function
: Pooling layer
: Fully connected layer



실습 Network base 구조 (Stride와 Padding size는 1로 고정)

- **하이퍼 파라미터**
  - Training epoch: 20
  - Batch size: 100
  - Learning rate: 0.1
  - Loss function: Cross Entropy Loss
  - Optimizer: SGD

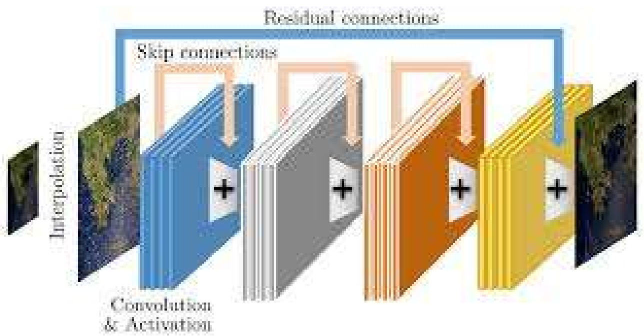
```
network.eval()
network = network.to('cpu')
img_test = torch.tensor(np.transpose(cifar10_test.data , (0, 3, 1, 2))) /255.
label_test = torch.tensor(cifar10_test.targets)

with torch.no_grad(): # test에서는 기울기 계산 제외
    prediction = network(img_test) # 전체 test data를 한번에 계산

correct_prediction = torch.argmax(prediction, 1) == label_test
accuracy = correct_prediction.float().mean()
print('Accuracy:', accuracy.item())
```

```
Accuracy: 0.6011000275611877
```

딥러닝 주요모델 구성요소

- **Skip connection (ResNet, 2015)**
- Dense connection (DenseNet, 2017)
- Channel attention (SENet, 2018)

❖ Skip connection



❖ Dense connection

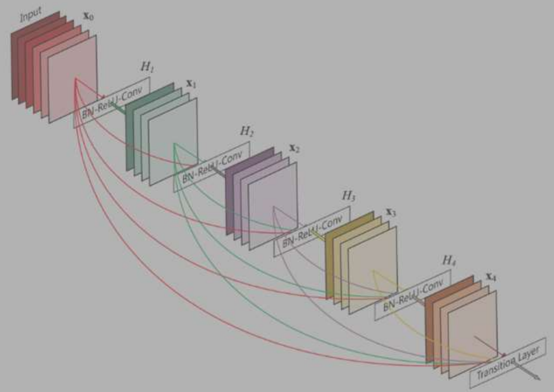
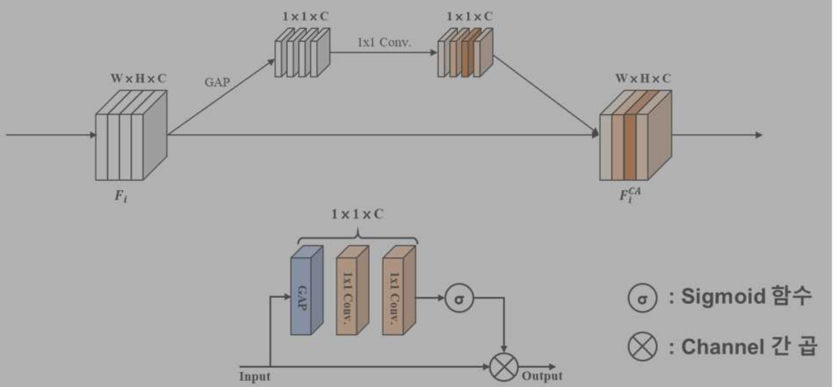


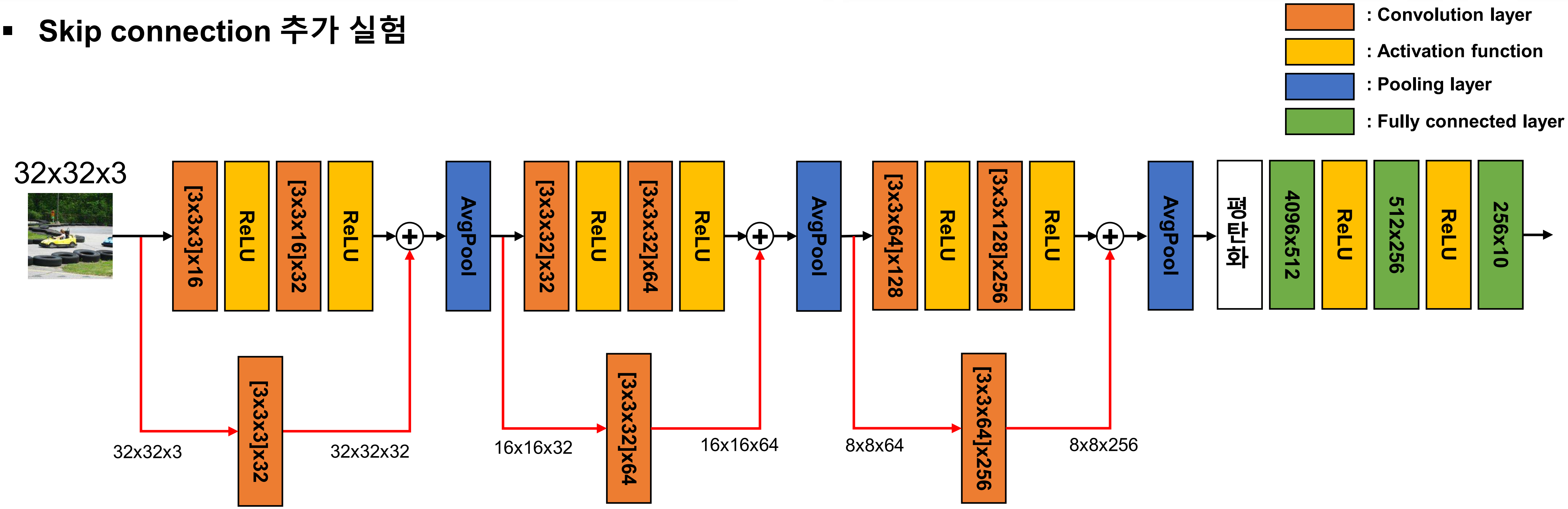
Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

❖ Channel attention



CIFAR10 분류 실습 - CNN을 이용한 분류(VGGNet)

- **Skip connection 추가 실험**

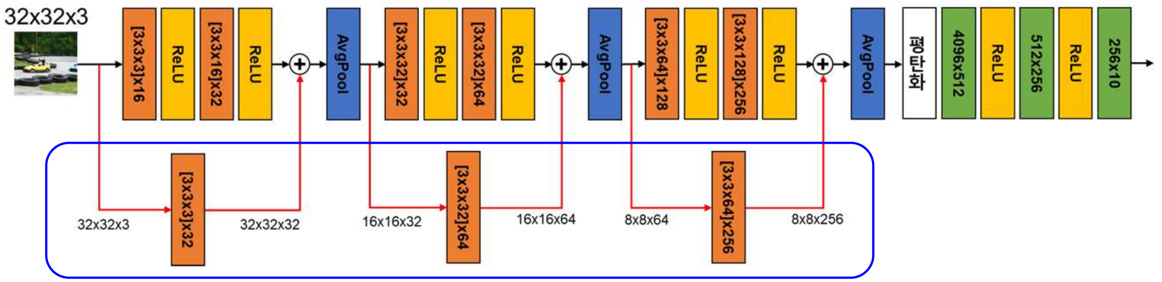


❖ 주의사항: Skip connection은 Width, Height, Channel이 모두 같아야 사용 가능

CIFAR10 분류 실습 - CNN을 이용한 분류(VGGNet)

- **Skip connection 추가 실험**
  - Skip connection을 위한 convolution layer 선언

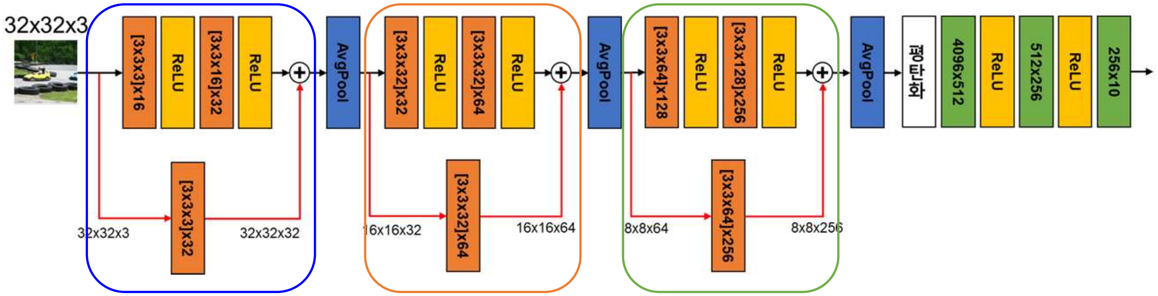
```
class VGG_SKIP (nn.Module):
    def __init__(self): # 신경망 구성요소 정의
        super(VGG_SKIP, self).__init__()
        self.conv1_1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, padding=1)
        self.conv1_2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, padding=1)

        self.conv2_1 = nn.Conv2d(in_channels=32, out_channels=32, kernel_size=3, padding=1)
        self.conv2_2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding=1)

        self.conv3_1 = nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, padding=1)
        self.conv3_2 = nn.Conv2d(in_channels=128, out_channels=256, kernel_size=3, padding=1)

        # Skip Connection을 위한 Conv. layer
        self.conv_skip1 = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, padding=1)
        self.conv_skip2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding=1)
        self.conv_skip3 = nn.Conv2d(in_channels=64, out_channels=256, kernel_size=3, padding=1)

        self.fc1 = nn.Linear(in_features=4096, out_features=512)
        self.fc2 = nn.Linear(in_features=512, out_features=256)
        self.fc3 = nn.Linear(in_features=256, out_features=10)

        self.relu = nn.ReLU()
        self.avgPool2d = nn.AvgPool2d(kernel_size=2, stride=2)
```



CIFAR10 분류 실습 - CNN을 이용한 분류(VGGNet)

- **Skip connection 추가 실험**
  - Skip connection 적용

```
def forward(self,x):

  input_feature1 = x #Skip 입력을 위한 Input 저장
  out = self.relu(self.conv1_1(x))
  out = self.relu(self.conv1_2(out))
  input_skip1 = self.relu(self.conv_skip1(input_feature1)) #Skip 입력을 위한 Conv layer 적용
  out = torch.add(out, input_skip1) #Skip connection 적용
  out = self.avgPool2d(out)

  input_feature2 = out #Skip 입력을 위한 Input 저장
  out = self.relu(self.conv2_1(out))
  out = self.relu(self.conv2_2(out))
  input_skip2 = self.relu(self.conv_skip2(input_feature2)) #Skip 입력을 위한 Conv layer 적용
  out = torch.add(out, input_skip2) #Skip connection 적용
  out = self.avgPool2d(out)

  input_feature3 = out #Skip 입력을 위한 Input 저장
  out = self.relu(self.conv3_1(out))
  out = self.relu(self.conv3_2(out))
  input_skip3 = self.relu(self.conv_skip3(input_feature3)) #Skip 입력을 위한 Conv layer 적용
  out = torch.add(out, input_skip3) #Skip connection 적용
  out = self.avgPool2d(out)

  out = out.view(-1, 4096) # feature map 평탄화

  out = self.relu(self.fc1(out))
  out = self.relu(self.fc2(out))
  out = self.fc3(out)
  return out
```

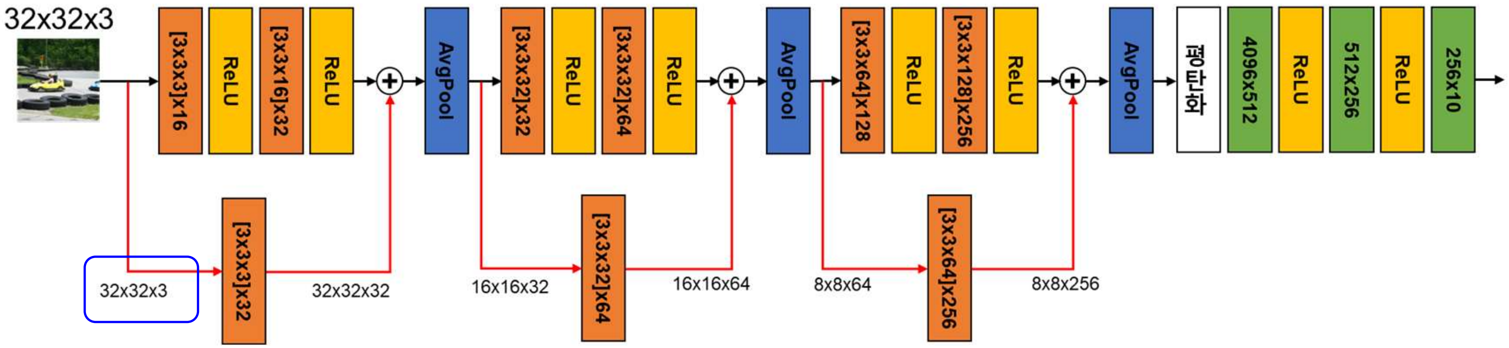


CIFAR10 분류 실습 - CNN을 이용한 분류(VGGNet)

- **Skip connection 추가 실험**

```
def forward(self,x):

   input_feature1 = x #Skip 입력을 위한 Input 저장
   out = self.relu(self.conv1_1(x))
   out = self.relu(self.conv1_2(out))
   input_skip1 = self.relu(self.conv_skip1(input_feature1)) #Skip 입력을 위한 Conv layer 적용
   out = torch.add(out, input_skip1) #Skip connection 적용
   out = self.avgPool2d(out)
```

Skip connection적용을 위해 Conv. 입력 저장

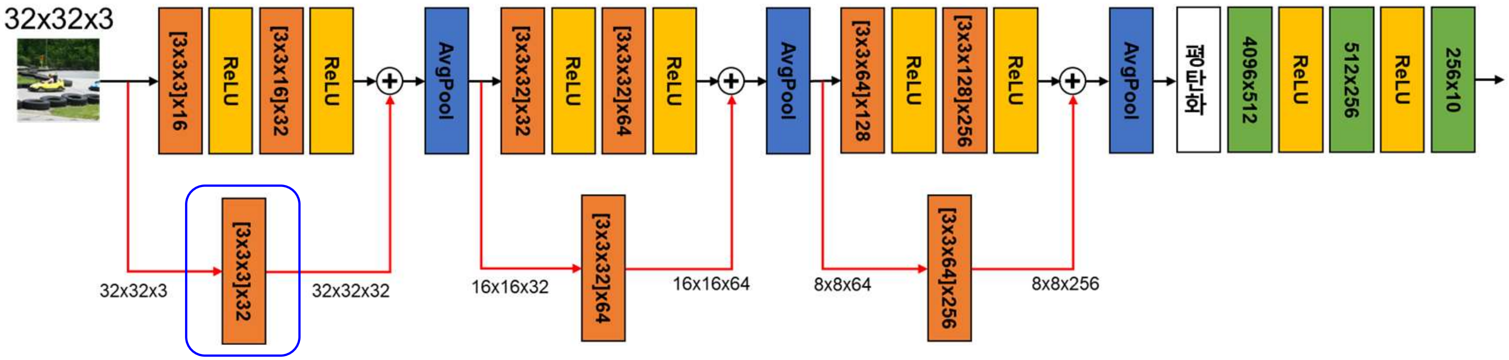


CIFAR10 분류 실습 - CNN을 이용한 분류(VGGNet)

- **Skip connection 추가 실험**

```
def forward(self,x):

    input_feature1 = x #Skip 입력을 위한 Input 저장
    out = self.relu(self.conv1_1(x))
    out = self.relu(self.conv1_2(out))
    input_skip1 = self.relu(self.conv_skip1(input_feature1)) #Skip 입력을 위한 Conv layer 적용
    out = torch.add(out, input_skip1) #Skip connection 적용
    out = self.avgPool2d(out)
```

Width, Height, Channel을 맞춰 주기 위한 Conv. 적용

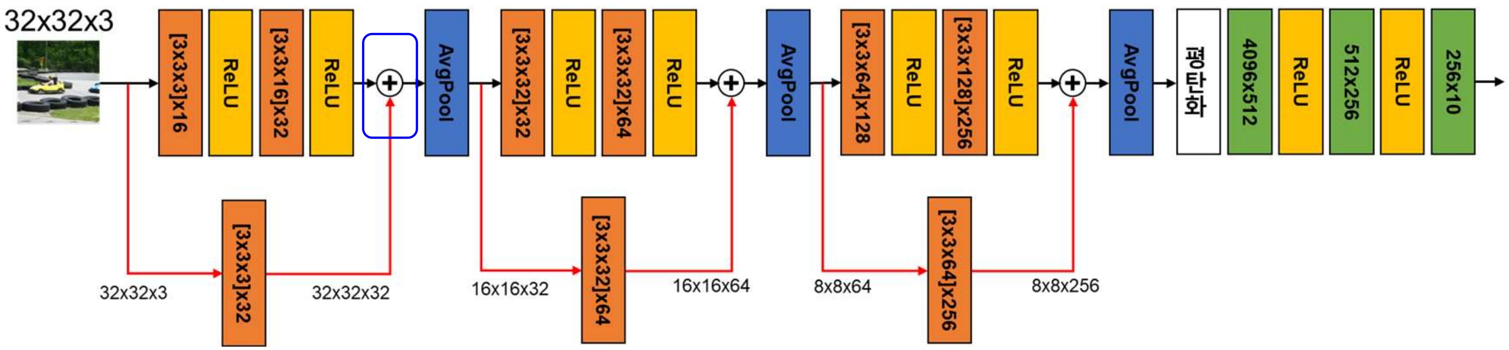


CIFAR10 분류 실습 - CNN을 이용한 분류(VGGNet)

- **Skip connection 추가 실험**

```
def forward(self,x):

    input_feature1 = x #Skip 입력을 위한 Input 저장
    out = self.relu(self.conv1_1(x))
    out = self.relu(self.conv1_2(out))
    input_skip1 = self.relu(self.conv_skip1(input_feature1)) #Skip 입력을 위한 Conv layer 적용
    out = torch.add(out, input_skip1) #Skip connection 적용
    out = self.avgPool2d(out)
```

→ Skip connection 적용 코드



CIFAR10 분류 실습 - CNN을 이용한 분류(VGGNet)

- **Skip connection 추가 실험 결과 확인**

```
Epoch: 1 Loss = 2.303002
Epoch: 2 Loss = 2.302858
Epoch: 3 Loss = 2.302659
Epoch: 4 Loss = 2.246866
Epoch: 5 Loss = 1.997299
Epoch: 6 Loss = 1.824729
Epoch: 7 Loss = 1.672605
Epoch: 8 Loss = 1.496609
Epoch: 9 Loss = 1.346635
Epoch: 10 Loss = 1.229228
Epoch: 11 Loss = 1.127741
Epoch: 12 Loss = 1.025967
Epoch: 13 Loss = 0.922246
Epoch: 14 Loss = 0.813664
Epoch: 15 Loss = 0.702598
Epoch: 16 Loss = 0.583456
Epoch: 17 Loss = 0.467354
Epoch: 18 Loss = 0.360702
Epoch: 19 Loss = 0.284199
Epoch: 20 Loss = 0.228450
Learning finished
```

```
Epoch: 1 Loss = 2.133430
Epoch: 2 Loss = 1.784824
Epoch: 3 Loss = 1.573649
Epoch: 4 Loss = 1.431847
Epoch: 5 Loss = 1.312706
Epoch: 6 Loss = 1.211934
Epoch: 7 Loss = 1.106290
Epoch: 8 Loss = 1.014058
Epoch: 9 Loss = 0.923362
Epoch: 10 Loss = 0.828185
Epoch: 11 Loss = 0.733846
Epoch: 12 Loss = 0.639242
Epoch: 13 Loss = 0.537734
Epoch: 14 Loss = 0.442022
Epoch: 15 Loss = 0.353637
Epoch: 16 Loss = 0.271488
Epoch: 17 Loss = 0.220454
Epoch: 18 Loss = 0.166728
Epoch: 19 Loss = 0.135304
Epoch: 20 Loss = 0.108017
Learning finished
```

Training 결과

```
network.eval()
network = network.to('cpu')
img_test = torch.tensor(np.transpose(cifar10_test.data , (0, 3, 1, 2))) /255.
label_test = torch.tensor(cifar10_test.targets)

with torch.no_grad(): # test에서는 기울기 계산 제외
    prediction = network(img_test) # 전체 test data를 한번에 계산

correct_prediction = torch.argmax(prediction, 1) == label_test
accuracy = correct_prediction.float().mean()
print('Accuracy:', accuracy.item())
```

```
Accuracy: 0.6011000275611877
```

```
network.eval()
network = network.to('cpu')
img_test = torch.tensor(np.transpose(cifar10_test.data , (0, 3, 1, 2))) /255.
label_test = torch.tensor(cifar10_test.targets)

with torch.no_grad(): # test에서는 기울기 계산 제외
    prediction = network(img_test) # 전체 test data를 한번에 계산

correct_prediction = torch.argmax(prediction, 1) == label_test
accuracy = correct_prediction.float().mean()
print('Accuracy:', accuracy.item())
```

```
Accuracy: 0.6686999797821045
```

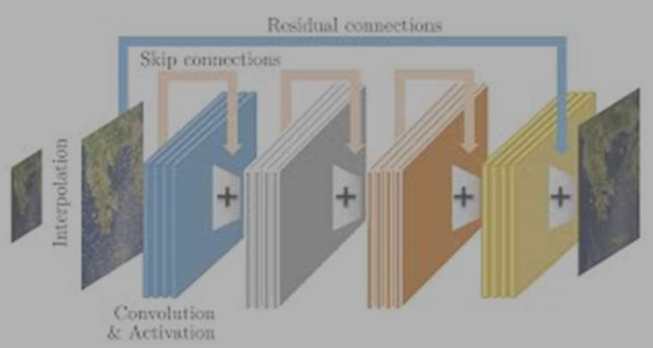
Test 결과

동아대학교 DONG-A UNIVERSITY

딥러닝 주요모델 구성요소

- Skip connection (ResNet, 2015)
- **Dense connection (DenseNet, 2017)**
- Channel attention (SENet, 2018)

❖ Skip connection



❖ **Dense connection**

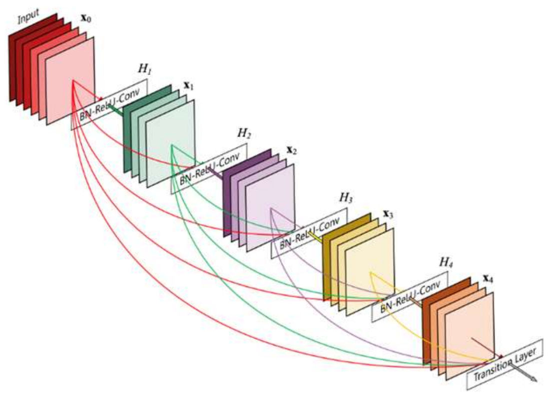
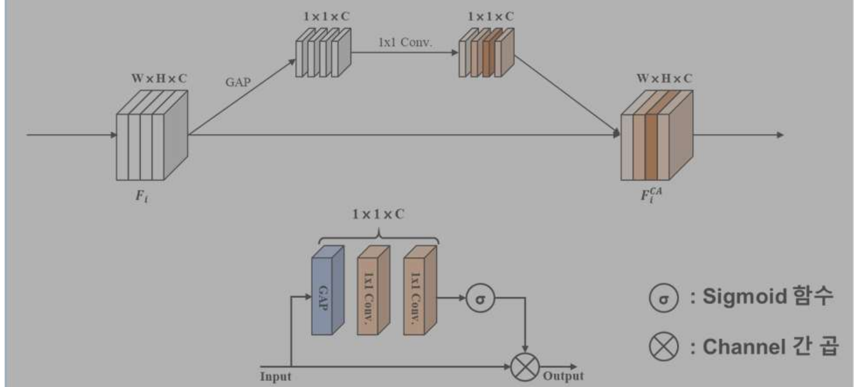


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

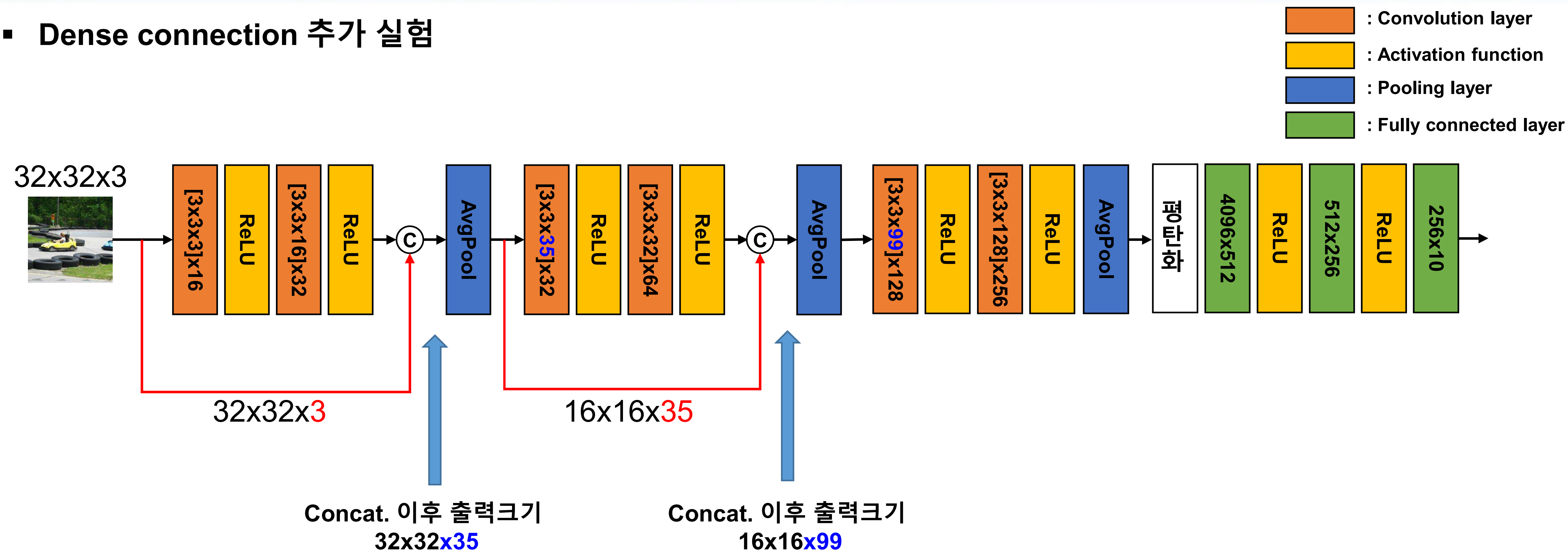
❖ Channel attention



동아대학교 DONG-A UNIVERSITY

CIFAR10 분류 실습 - CNN을 이용한 분류(VGGNet)

- **Dense connection 추가 실험**



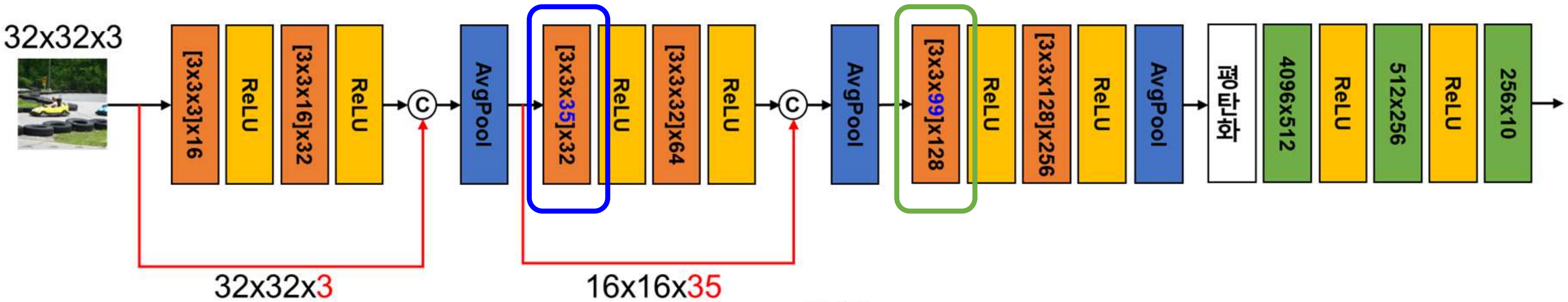
❖ **주의사항: Dense connection (torch.cat)은 width, height이 동일해야 적용 가능**

동아대학교 DONG-A UNIVERSITY

CIFAR10 분류 실습 - CNN을 이용한 분류(VGGNet)

- **Dense connection 추가 실험**
  - Dense 추가로 인한 Input channels 변경

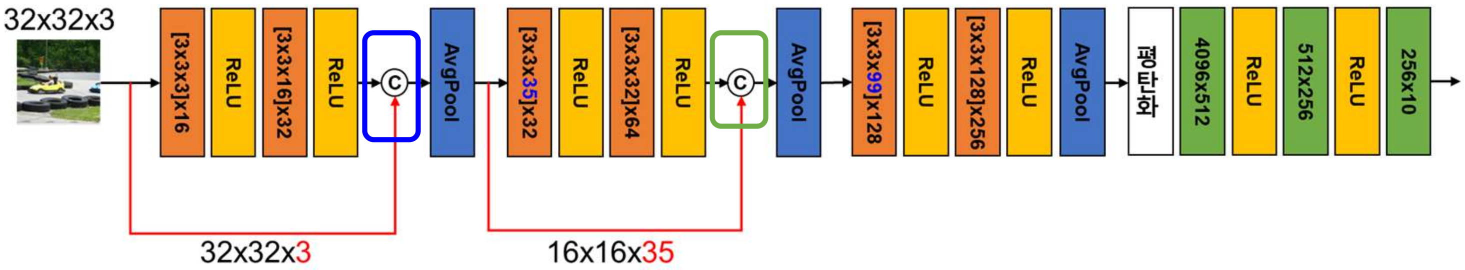
```
class VGG_DENSE (nn.Module):
    def __init__(self): # 신경망 구성요소 정의
        super(VGG_DENSE, self).__init__()
        self.conv1_1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, padding=1)
        self.conv1_2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, padding=1)

        self.conv2_1 = nn.Conv2d(in_channels=35, out_channels=32, kernel_size=3, padding=1)
        self.conv2_2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding=1)

        self.conv3_1 = nn.Conv2d(in_channels=99, out_channels=128, kernel_size=3, padding=1)
        self.conv3_2 = nn.Conv2d(in_channels=128, out_channels=256, kernel_size=3, padding=1)

        self.fc1 = nn.Linear(in_features=4096, out_features=512)
        self.fc2 = nn.Linear(in_features=512, out_features=256)
        self.fc3 = nn.Linear(in_features=256, out_features=10)

        self.relu = nn.ReLU()
        self.avgPool2d = nn.AvgPool2d(kernel_size=2, stride=2)
```



CIFAR10 분류 실습 - CNN을 이용한 분류(VGGNet)

- **Dense connection 추가 실험**
  - Dense를 위한 Concat. 코드 추가

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

    out1 = self.relu(self.conv1_1(x))
    out1 = self.relu(self.conv1_2(out1))
    out1 = torch.cat([x, out1], dim=1)
    out1 = self.avgPool2d(out1)

    out2 = self.relu(self.conv2_1(out1))
    out2 = self.relu(self.conv2_2(out2))
    out2 = torch.cat([out1, out2], dim=1)
    out2 = self.avgPool2d(out2)

    out3 = self.relu(self.conv3_1(out2))
    out3 = self.relu(self.conv3_2(out3))
    #out3 = torch.cat([out2, out3], dim=1)
    out = self.avgPool2d(out3)

    out = out.view(-1, 4096) # feature map 평탄화

    out = self.relu(self.fc1(out))
    out = self.relu(self.fc2(out))
    out = self.fc3(out)
    return out
```



out1 = torch.cat([x, out1], dim=1)

Feature map 형상: (Batch_size, Channel, Width, Height)

dim: 0 1 2 3

동아대학교 DONG-A UNIVERSITY

CIFAR10 분류 실습 - CNN을 이용한 분류(VGGNet)

- **Dense connection 추가 실험 결과 확인**

```
Epoch: 1 Loss = 2.303002
Epoch: 2 Loss = 2.302858
Epoch: 3 Loss = 2.302659
Epoch: 4 Loss = 2.246866
Epoch: 5 Loss = 1.997299
Epoch: 6 Loss = 1.824729
Epoch: 7 Loss = 1.672605
Epoch: 8 Loss = 1.496609
Epoch: 9 Loss = 1.346635
Epoch: 10 Loss = 1.229228
Epoch: 11 Loss = 1.127741
Epoch: 12 Loss = 1.025967
Epoch: 13 Loss = 0.922246
Epoch: 14 Loss = 0.813664
Epoch: 15 Loss = 0.702598
Epoch: 16 Loss = 0.583456
Epoch: 17 Loss = 0.467354
Epoch: 18 Loss = 0.360702
Epoch: 19 Loss = 0.284199
Epoch: 20 Loss = 0.228450
Learning finished
```

```
Epoch: 1 Loss = 2.254756
Epoch: 2 Loss = 1.989823
Epoch: 3 Loss = 1.779417
Epoch: 4 Loss = 1.609736
Epoch: 5 Loss = 1.490473
Epoch: 6 Loss = 1.384320
Epoch: 7 Loss = 1.268125
Epoch: 8 Loss = 1.179736
Epoch: 9 Loss = 1.089010
Epoch: 10 Loss = 0.999267
Epoch: 11 Loss = 0.914411
Epoch: 12 Loss = 0.825907
Epoch: 13 Loss = 0.740529
Epoch: 14 Loss = 0.647511
Epoch: 15 Loss = 0.560547
Epoch: 16 Loss = 0.462542
Epoch: 17 Loss = 0.378811
Epoch: 18 Loss = 0.292290
Epoch: 19 Loss = 0.225068
Epoch: 20 Loss = 0.166460
Learning finished
```

Training 결과

```
network.eval()
network = network.to('cpu')
img_test = torch.tensor(np.transpose(cifar10_test.data , (0, 3, 1, 2))) /255.
label_test = torch.tensor(cifar10_test.targets)

with torch.no_grad(): # test에서는 기울기 계산 제외
    prediction = network(img_test) # 전체 test data를 한번에 계산

correct_prediction = torch.argmax(prediction, 1) == label_test
accuracy = correct_prediction.float().mean()
print('Accuracy:', accuracy.item())
```

```
Accuracy: 0.6011000275611877
```

```
network.eval()
network = network.to('cpu')
img_test = torch.tensor(np.transpose(cifar10_test.data , (0, 3, 1, 2))) /255.
label_test = torch.tensor(cifar10_test.targets)

with torch.no_grad(): # test에서는 기울기 계산 제외
    prediction = network(img_test) # 전체 test data를 한번에 계산

correct_prediction = torch.argmax(prediction, 1) == label_test
accuracy = correct_prediction.float().mean()
print('Accuracy:', accuracy.item())
```

```
Accuracy: 0.6967999935150146
```

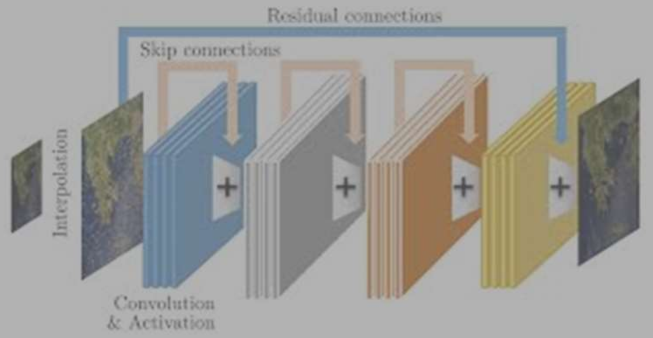
Test 결과

동아대학교 DONG-A UNIVERSITY

딥러닝 주요모델 구성요소

- Skip connection (ResNet, 2015)
- Dense connection (DenseNet, 2017)
- **Channel attention (SENet, 2018)**

❖ Skip connection



❖ Dense connection

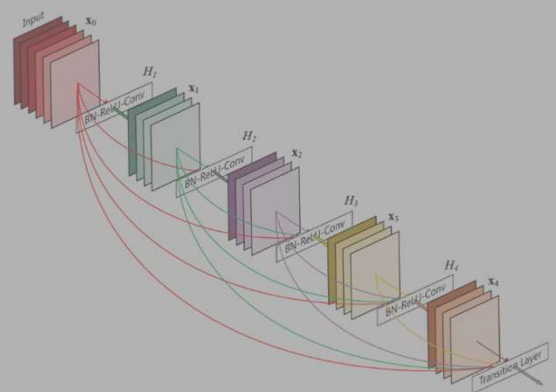
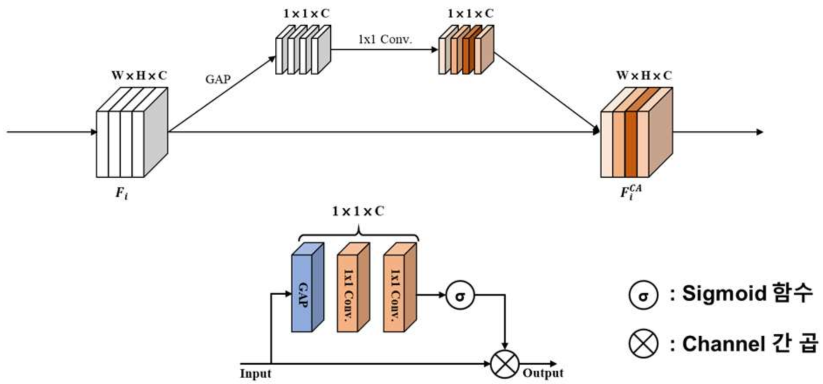


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

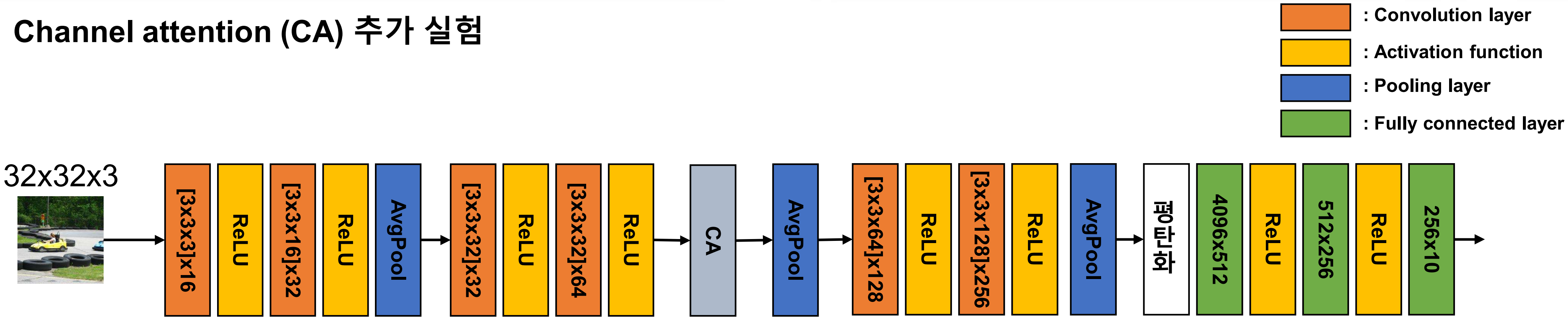
❖ Channel attention

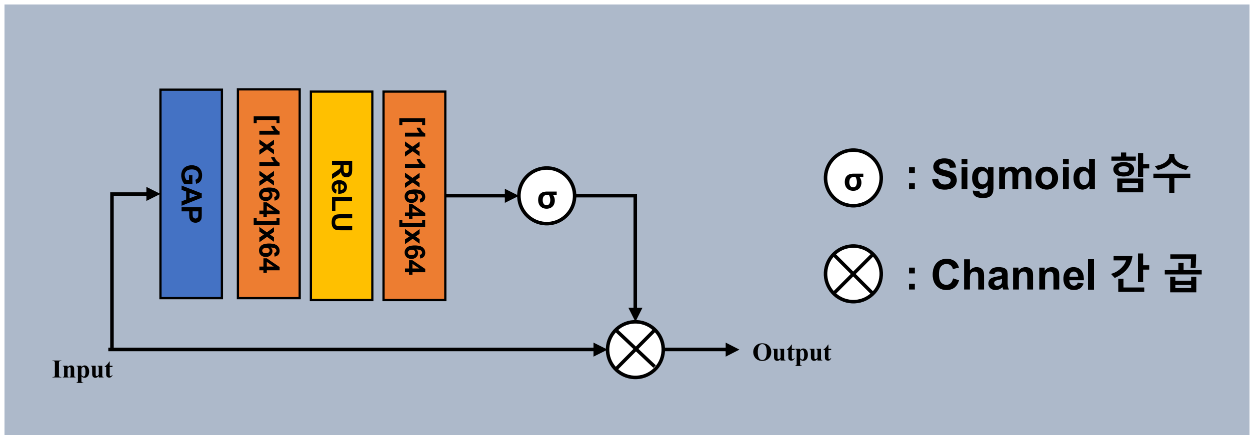


동아대학교 DONG-A UNIVERSITY

CIFAR10 분류 실습 - CNN을 이용한 분류(VGGNet)

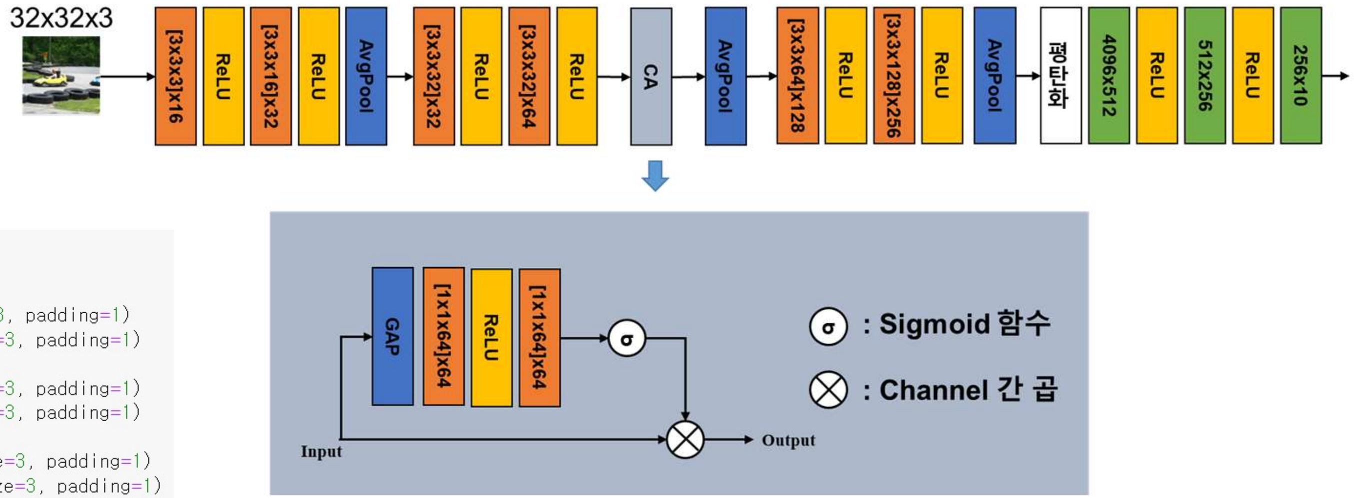
- **Channel attention (CA) 추가 실험**





CIFAR10 분류 실습 - CNN을 이용한 분류(VGGNet)

- **Channel attention (CA) 추가 실험**
  - CA 구성요소 정의

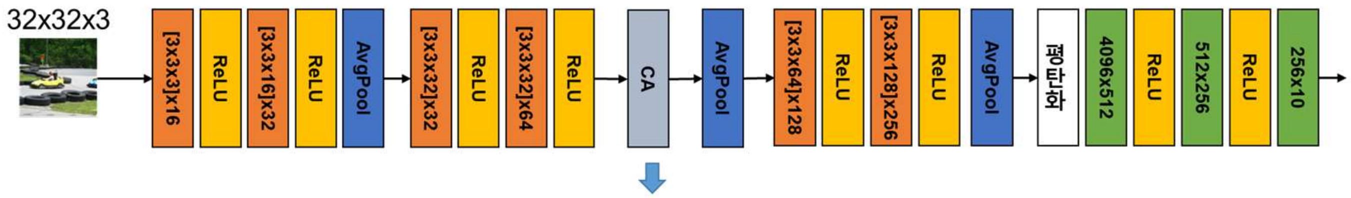


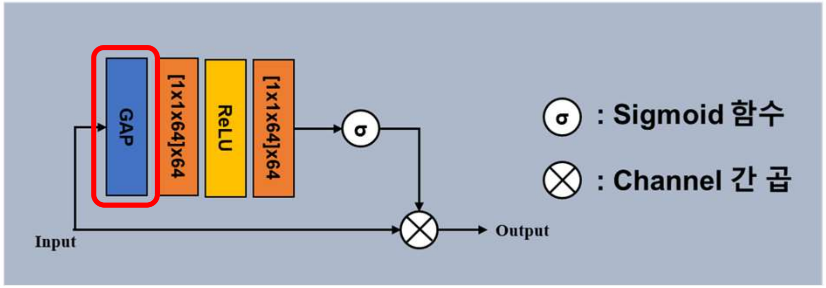

```python
class VGG_CA (nn.Module):
    def __init__(self): # 신경망 구성요소 정의
        super(VGG_CA, self).__init__()
        self.conv1_1 = nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, padding=1)
        self.conv1_2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, padding=1)

        self.conv2_1 = nn.Conv2d(in_channels=32, out_channels=32, kernel_size=3, padding=1)
        self.conv2_2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding=1)

        self.conv3_1 = nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, padding=1)
        self.conv3_2 = nn.Conv2d(in_channels=128, out_channels=256, kernel_size=3, padding=1)

        self.fc1 = nn.Linear(in_features=4096, out_features=512)
        self.fc2 = nn.Linear(in_features=512, out_features=256)
        self.fc3 = nn.Linear(in_features=256, out_features=10)


        # Channel Attention
        self.adaptiveAvgPool2d = nn.AdaptiveAvgPool2d((1, 1)) # Global average pooling
        self.caconv1 = nn.Conv2d(in_channels=64, out_channels=64, kernel_size=1)
        self.caconv2 = nn.Conv2d(in_channels=64, out_channels=64, kernel_size=1)
        self.sigmoid = nn.Sigmoid()

        self.relu = nn.ReLU()
        self.avgPool2d = nn.AvgPool2d(kernel_size=2, stride=2)
```

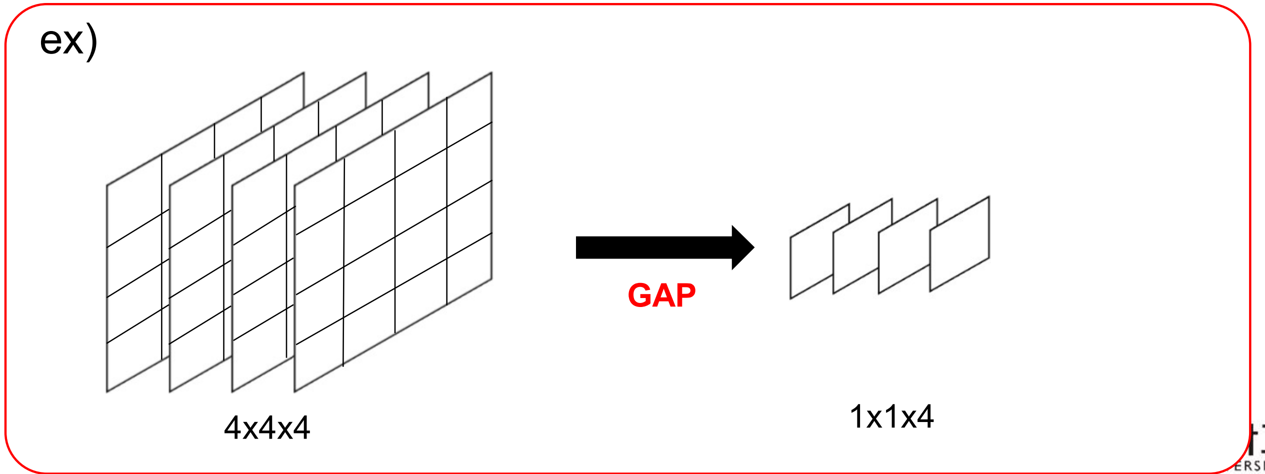
CIFAR10 분류 실습 - CNN을 이용한 분류(VGGNet)

- **Channel attention (CA) 추가 실험**
  - CA 동작 코드 작성

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

    out = self.relu(self.conv1_1(x))
    out = self.relu(self.conv1_2(out))
    out = self.avgPool2d(out)

    out = self.relu(self.conv2_1(out))
    out = self.relu(self.conv2_2(out))

    ## Channel attention 적용
    caout = self.adaptiveAvgPool2d(out)
    caout = self.relu(self.caconv1(caout))
    caout = self.sigmoid(self.caconv2(caout))
    CA_map = caout.expand_as(out)
    out = out * CA_map

    out = self.avgPool2d(out)

    out = self.relu(self.conv3_1(out))
    out = self.relu(self.conv3_2(out))
    out = self.avgPool2d(out)

    out = out.view(-1, 4096) # feature map 평탄화

    out = self.relu(self.fc1(out))
    out = self.relu(self.fc2(out))
    out = self.fc3(out)
    return out
```



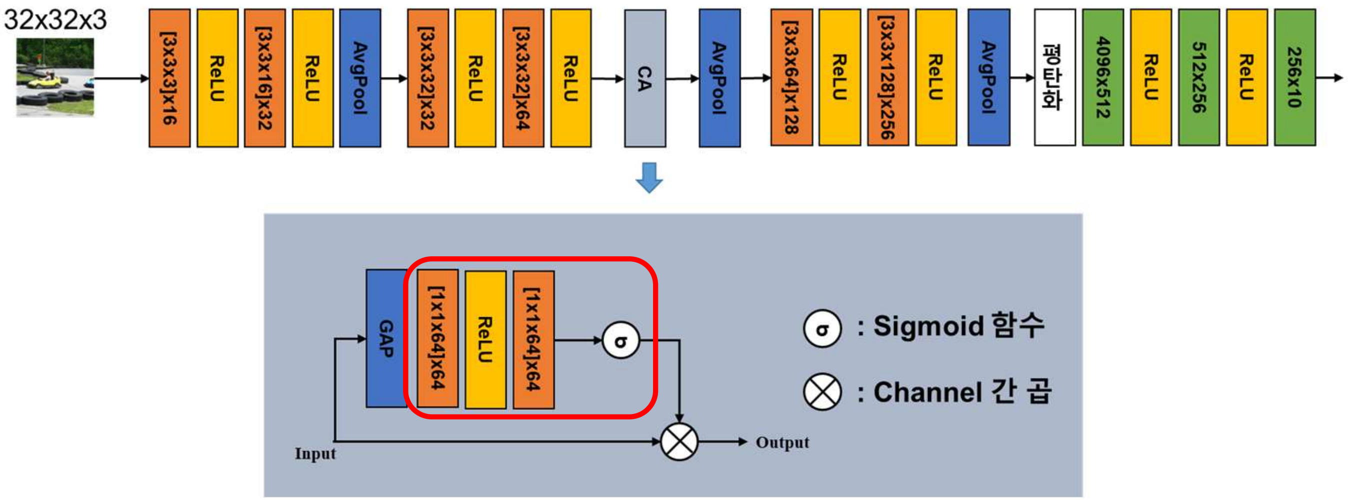




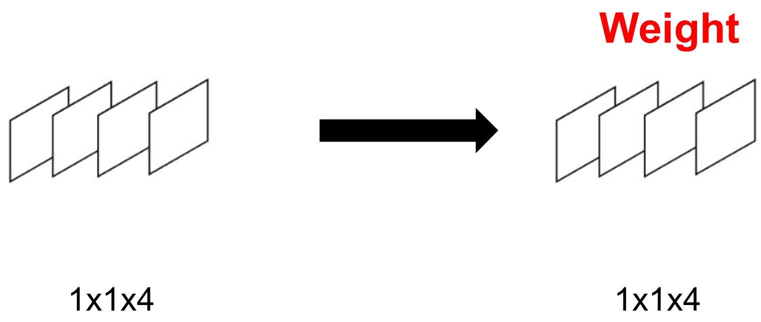
CIFAR10 분류 실습 - CNN을 이용한 분류(VGGNet)

- **Channel attention (CA) 추가 실험**
  - CA 동작 코드 작성

```
def forward(self,x):

    out = self.relu(self.conv1_1(x))
    out = self.relu(self.conv1_2(out))
    out = self.avgPool2d(out)

    out = self.relu(self.conv2_1(out))
    out = self.relu(self.conv2_2(out))

    ## Channel attention 적용
    caout = self.adaptiveAvgPool2d(out)
    caout = self.relu(self.caconv1(caout))
    caout = self.sigmoid(self.caconv2(caout))
    CA_map = caout.expand_as(out)
    out = out * CA_map

    out = self.avgPool2d(out)

    out = self.relu(self.conv3_1(out))
    out = self.relu(self.conv3_2(out))
    out = self.avgPool2d(out)

    out = out.view(-1, 4096) # feature map 평탄화

    out = self.relu(self.fc1(out))
    out = self.relu(self.fc2(out))
    out = self.fc3(out)
    return out
```



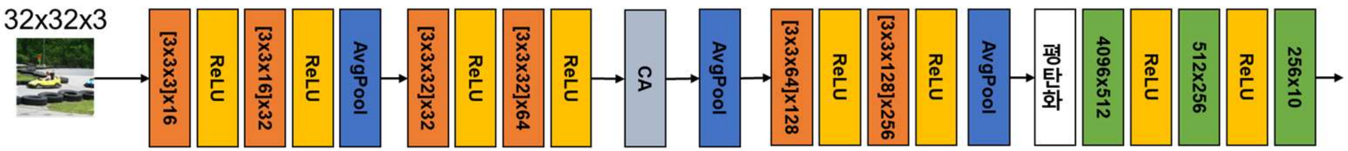
ex)

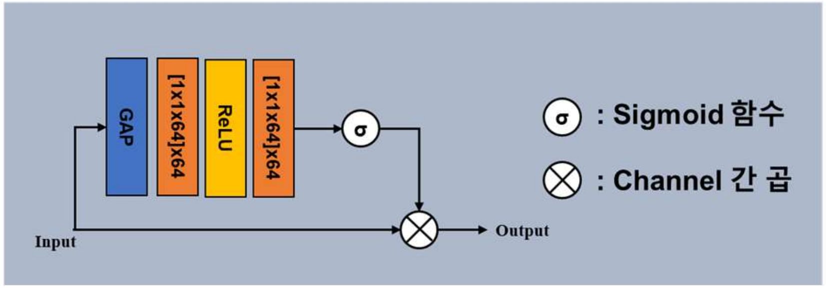


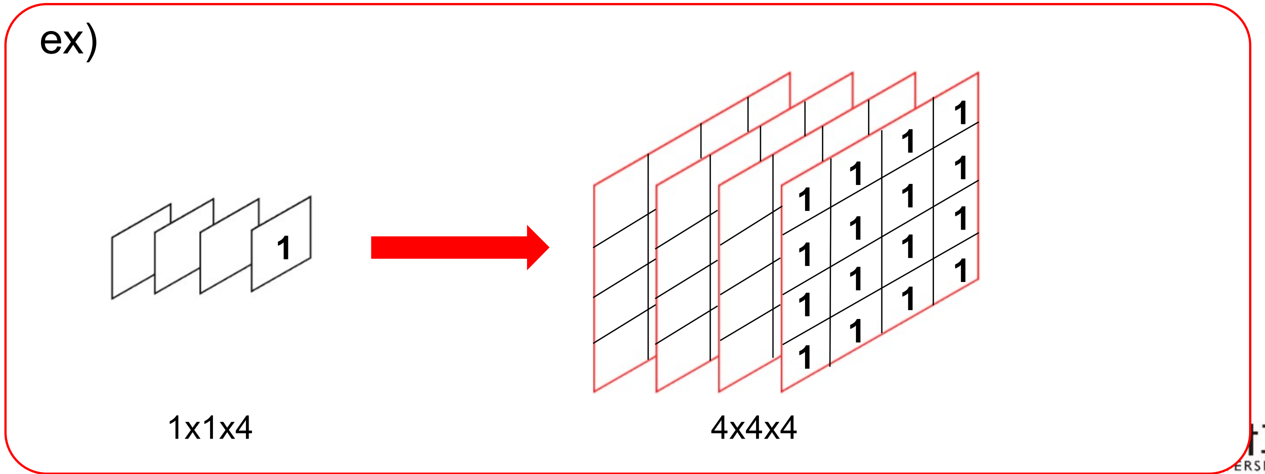
CIFAR10 분류 실습 - CNN을 이용한 분류(VGGNet)

- **Channel attention (CA) 추가 실험**
  - CA 동작 코드 작성

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

    out = self.relu(self.conv1_1(x))
    out = self.relu(self.conv1_2(out))
    out = self.avgPool2d(out)

    out = self.relu(self.conv2_1(out))
    out = self.relu(self.conv2_2(out))

    ## Channel attention 적용
    caout = self.adaptiveAvgPool2d(out)
    caout = self.relu(self.caconv1(caout))
    caout = self.sigmoid(self.caconv2(caout))
    CA_map = caout.expand_as(out)
    out = out * CA_map

    out = self.avgPool2d(out)

    out = self.relu(self.conv3_1(out))
    out = self.relu(self.conv3_2(out))
    out = self.avgPool2d(out)

    out = out.view(-1, 4096) # feature map 평탄화

    out = self.relu(self.fc1(out))
    out = self.relu(self.fc2(out))
    out = self.fc3(out)
    return out
```



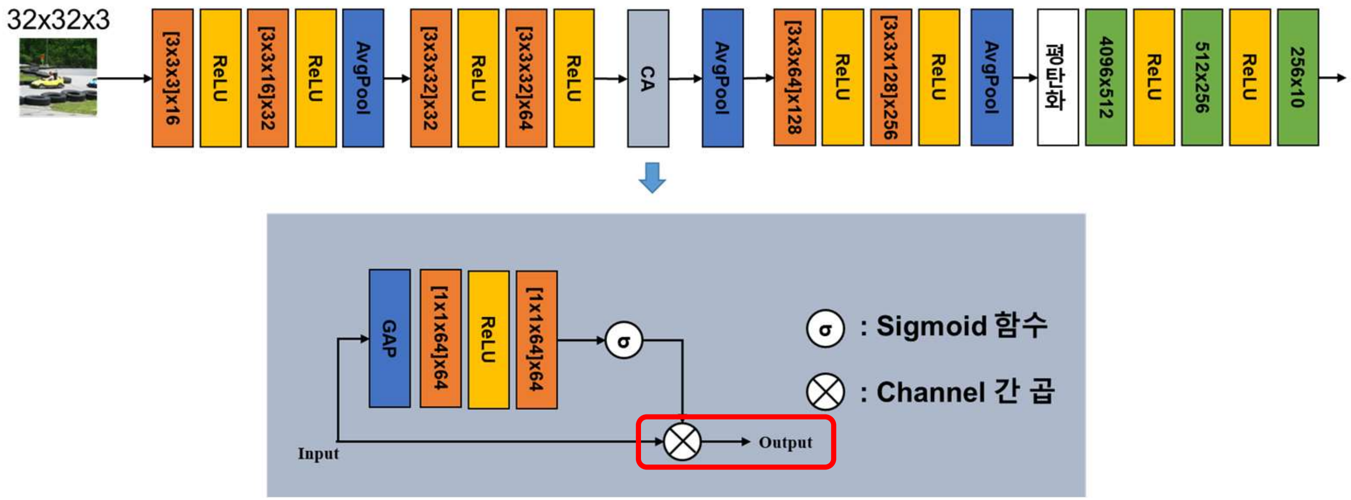


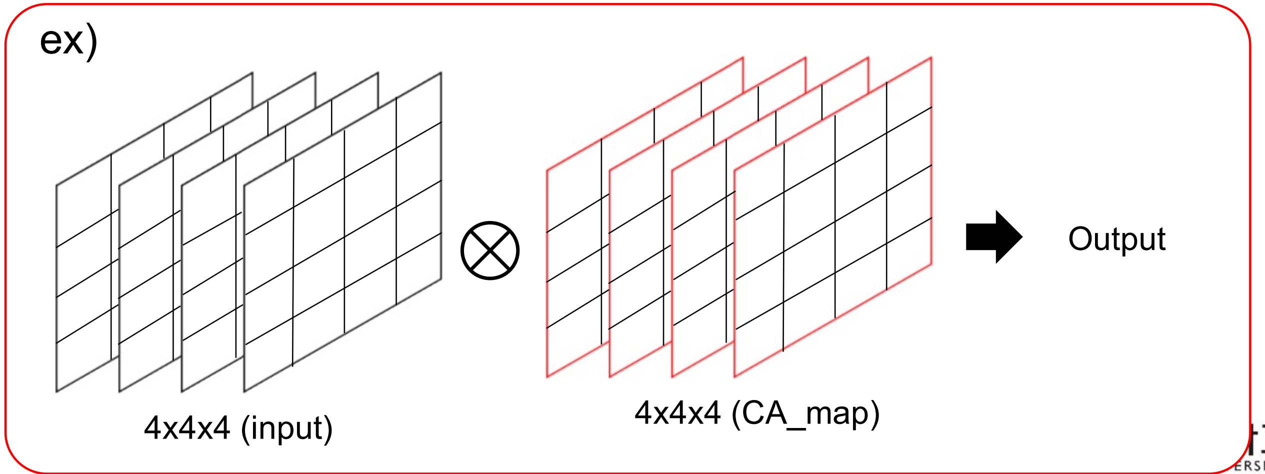


CIFAR10 분류 실습 - CNN을 이용한 분류(VGGNet)

- **Channel attention (CA) 추가 실험**
  - CA 동작 코드 작성

```
def forward(self,x):

    out = self.relu(self.conv1_1(x))
    out = self.relu(self.conv1_2(out))
    out = self.avgPool2d(out)

    out = self.relu(self.conv2_1(out))
    out = self.relu(self.conv2_2(out))

    ## Channel attention 적용
    caout = self.adaptiveAvgPool2d(out)
    caout = self.relu(self.caconv1(caout))
    caout = self.sigmoid(self.caconv2(caout))
    CA_map = caout.expand_as(out)
    out = out * CA_map

    out = self.avgPool2d(out)

    out = self.relu(self.conv3_1(out))
    out = self.relu(self.conv3_2(out))
    out = self.avgPool2d(out)

    out = out.view(-1, 4096) # feature map 평탄화

    out = self.relu(self.fc1(out))
    out = self.relu(self.fc2(out))
    out = self.fc3(out)
    return out
```





CIFAR10 분류 실습 - CNN을 이용한 분류(VGGNet)

- **Channel attention 추가 실험 결과 확인**

```
Epoch: 1 Loss = 2.303002
Epoch: 2 Loss = 2.302858
Epoch: 3 Loss = 2.302659
Epoch: 4 Loss = 2.246866
Epoch: 5 Loss = 1.997299
Epoch: 6 Loss = 1.824729
Epoch: 7 Loss = 1.672605
Epoch: 8 Loss = 1.496609
Epoch: 9 Loss = 1.346635
Epoch: 10 Loss = 1.229228
Epoch: 11 Loss = 1.127741
Epoch: 12 Loss = 1.025967
Epoch: 13 Loss = 0.922246
Epoch: 14 Loss = 0.813664
Epoch: 15 Loss = 0.702598
Epoch: 16 Loss = 0.583456
Epoch: 17 Loss = 0.467354
Epoch: 18 Loss = 0.360702
Epoch: 19 Loss = 0.284199
Epoch: 20 Loss = 0.228450
Learning finished
```

```
Epoch: 1 Loss = 2.302945
Epoch: 2 Loss = 2.302926
Epoch: 3 Loss = 2.302883
Epoch: 4 Loss = 2.302598
Epoch: 5 Loss = 2.238352
Epoch: 6 Loss = 1.993443
Epoch: 7 Loss = 1.834756
Epoch: 8 Loss = 1.717770
Epoch: 9 Loss = 1.580547
Epoch: 10 Loss = 1.429867
Epoch: 11 Loss = 1.297576
Epoch: 12 Loss = 1.185068
Epoch: 13 Loss = 1.087217
Epoch: 14 Loss = 0.992931
Epoch: 15 Loss = 0.889833
Epoch: 16 Loss = 0.785588
Epoch: 17 Loss = 0.673162
Epoch: 18 Loss = 0.555354
Epoch: 19 Loss = 0.444460
Epoch: 20 Loss = 0.362085
Learning finished
```

Training 결과

```
network.eval()
network = network.to('cpu')
img_test = torch.tensor(np.transpose(cifar10_test.data , (0, 3, 1, 2))) /255.
label_test = torch.tensor(cifar10_test.targets)

with torch.no_grad(): # test에서는 기울기 계산 제외
    prediction = network(img_test) # 전체 test data를 한번에 계산

correct_prediction = torch.argmax(prediction, 1) == label_test
accuracy = correct_prediction.float().mean()
print('Accuracy:', accuracy.item())
```

```
Accuracy: 0.6011000275611877
```

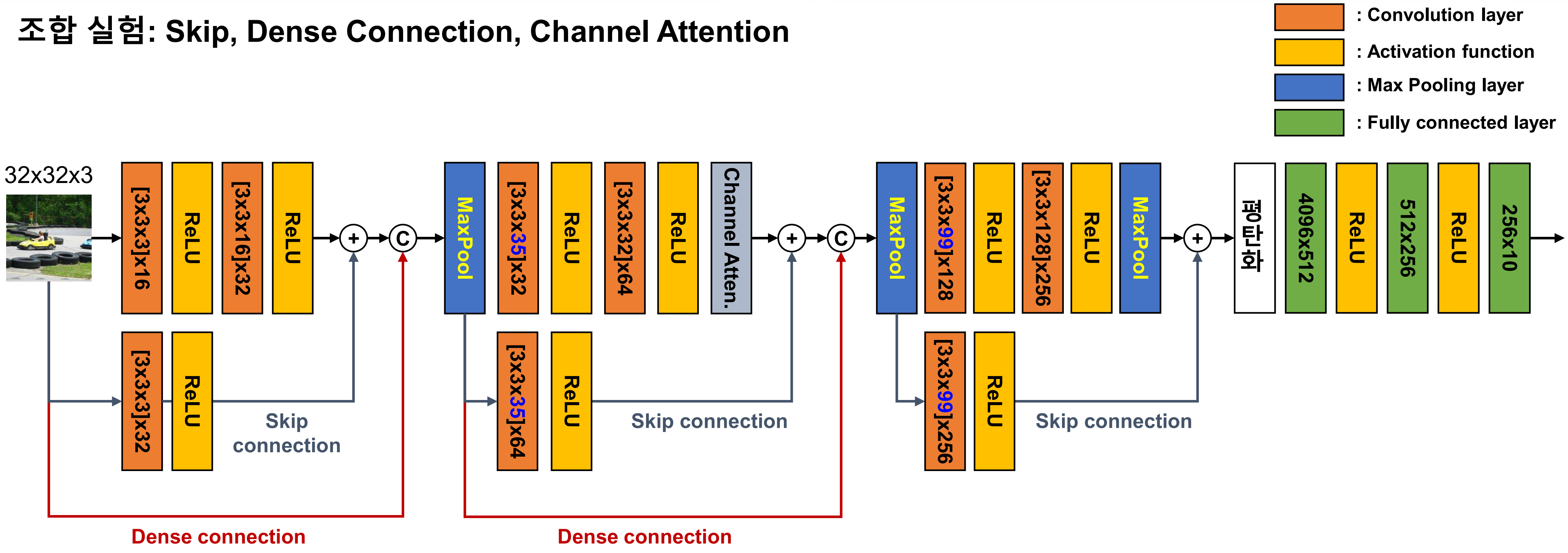
```
network.eval()
network = network.to('cuda:0')
img_test = torch.tensor(np.transpose(cifar10_test.data , (0, 3, 1, 2))) /255.
label_test = torch.tensor(cifar10_test.targets)

img_test = img_test.to('cuda:0')
label_test = label_test.to('cuda:0')

with torch.no_grad(): # test에서는 기울기 계산 제외
    prediction = network(img_test) # 전체 test data를 한번에 계산

correct_prediction = torch.argmax(prediction, 1) == label_test
accuracy = correct_prediction.float().mean()
print('Accuracy:', accuracy.item())
```

```
Accuracy: 0.6122999787330627
```

Test 결과

Appendix – CNN 구성요소 조합 실험

- 조합 실험: Skip, Dense Connection, Channel Attention



❖ 주의사항(1): Skip connection은 Width, Height, Channel이 모두 같아야 사용 가능

❖ 주의사항(2): Dense connection (torch.cat)은 width, height이 동일해야 적용 가능

Questions & Answers

Dongsan Jun (dsjun@dau.ac.kr)
Image Signal Processing Laboratory (www.donga-ispl.kr)
Dept. of Computer Engineering
Dong-A University, Busan, Rep. of Korea