Monthly Project2

A-3 김수, 성현규, 최갑주

P1. 데이터셋 다운로드 및 분석하기

- 다음의 코드를 통해 다운로드 가능

```
# 깃허브에서 데이터셋 다운로드하기
!git clone https://github.com/ndb796/Scene-Classification-Dataset-Split
# 폴더 안으로 이동
%cd Scene-Classification-Dataset-Split
```

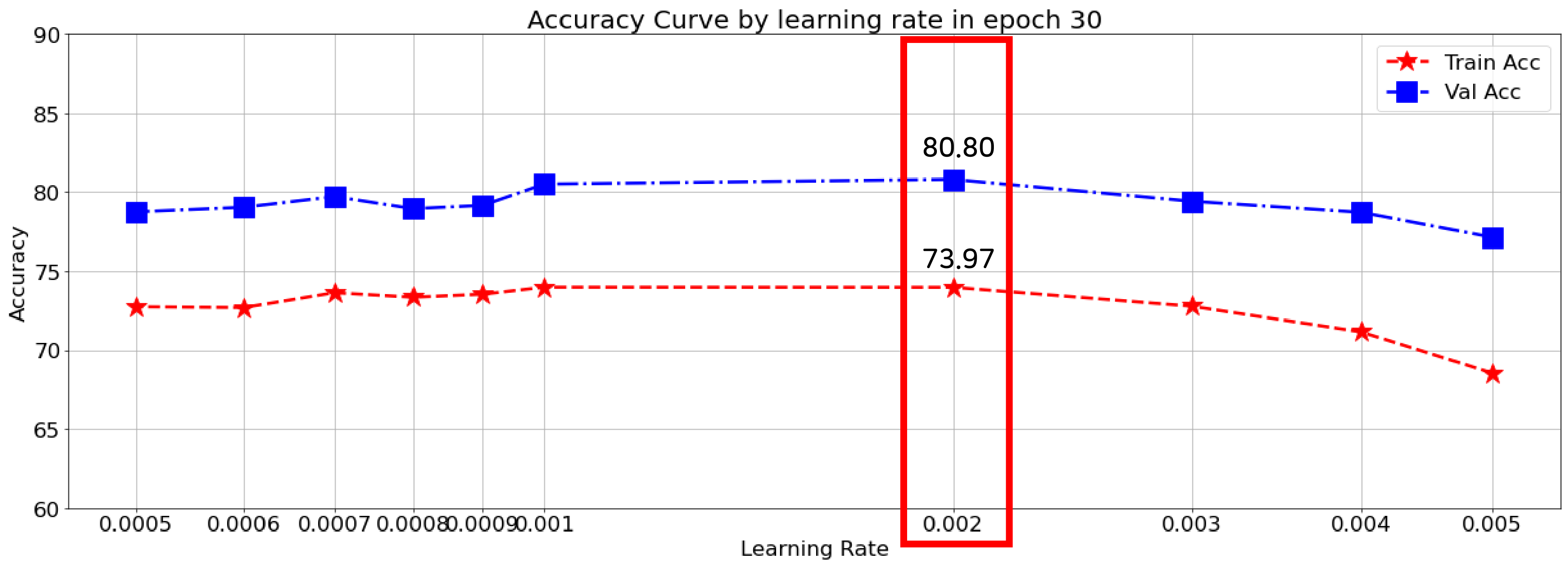
출력 차원 계산 코드 및 출력

```
def Output_Dim(height, width, filter_height, filter_width, stride, padding):
    output_height = (height + 2 * padding - filter_height) // stride + 1
    output_width = (width + 2 * padding - filter_width) // stride + 1
    return output_height, output_width

print("출력 높이 : %d, 출력 너비 : %d"%(Output_Dim(32,32,5,5,2,2)))
print("출력 높이 : %d, 출력 너비 : %d"%(Output_Dim(64,64,3,3,1,1)))
print("출력 높이 : %d, 출력 너비 : %d"%(Output_Dim(16,16,4,4,2,1)))
print("출력 높이 : %d, 출력 너비 : %d"%(Output_Dim(60,45,8,5,3,1)))
```

```
출력 높이 : 16, 출력 너비 : 16
출력 높이 : 64, 출력 너비 : 64
출력 높이 : 8, 출력 너비 : 8
출력 높이 : 19, 출력 너비 : 15
```

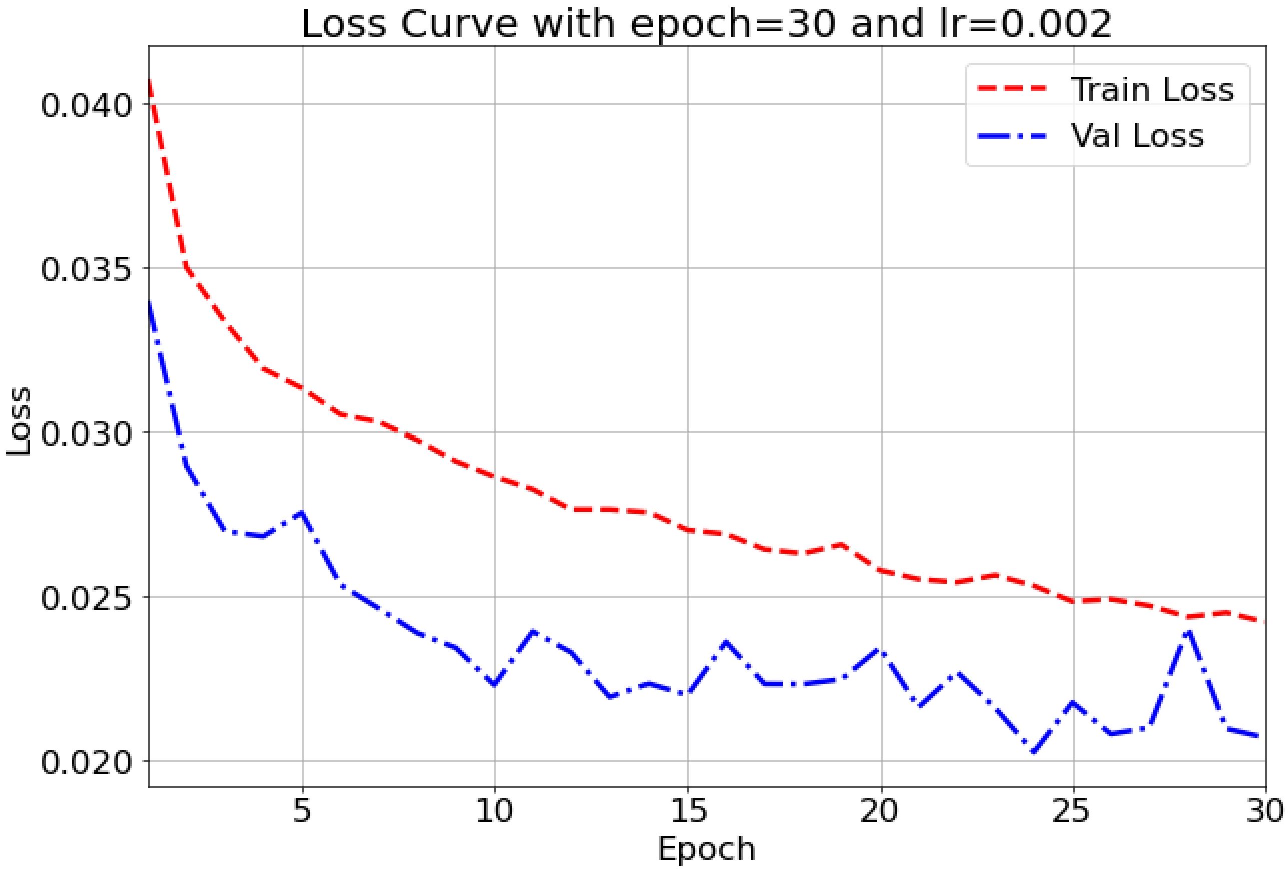
P2. Convolution 연산 이해하기

- LeNet 학습
  - 0.007이상의 Lr의 경우 손실 값 NaN



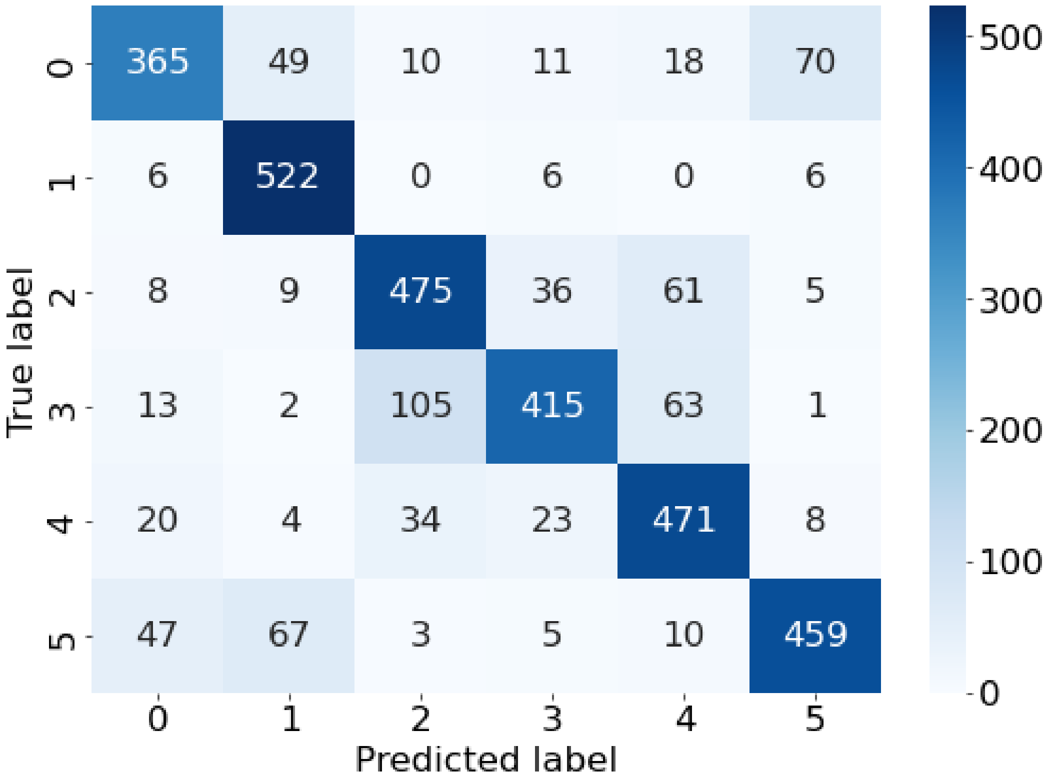
P2. Convolution 연산 이해하기

Loss Curve



P2. Convolution 연산 이해하기

- Confusion Matrix(LeNet)
  - 전체 평균 정확도 : 0.7945



각 클래스에 따른 정확도	
0	0.6979
1	0.9667
2	0.7997
3	0.6928
4	0.8411
5	0.7766

CustomLeNet 구조

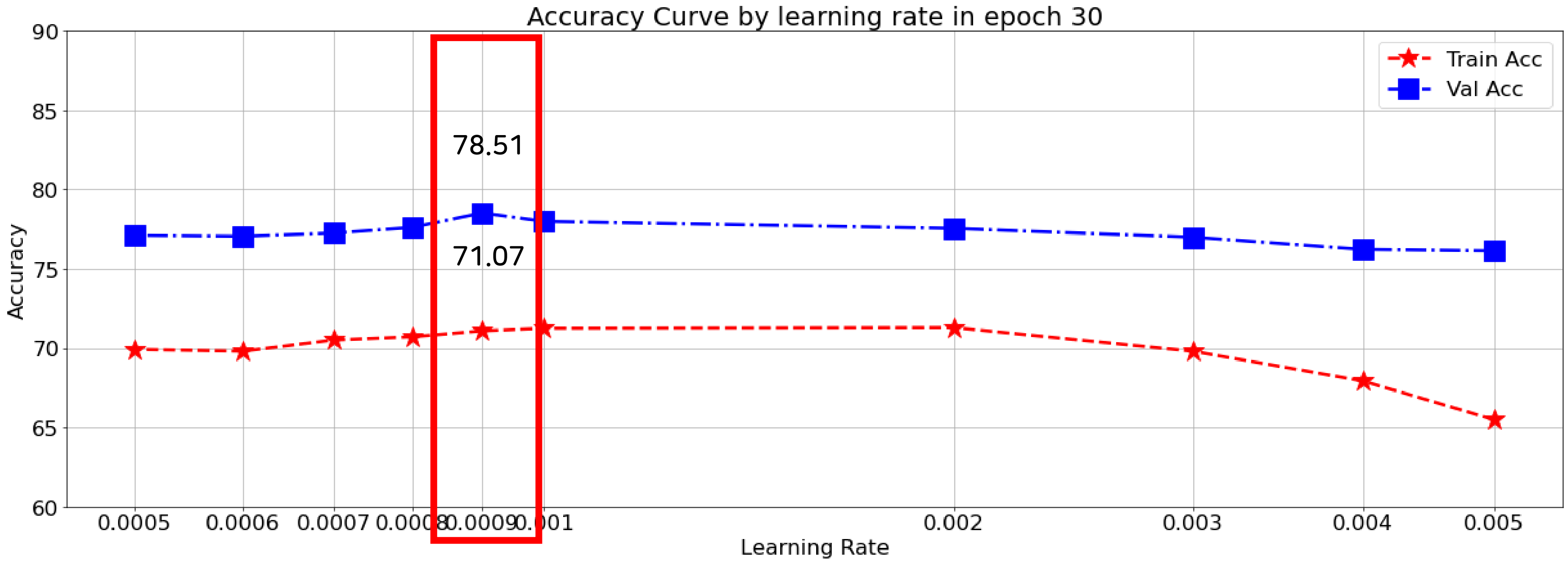
Layer	Type	Input dimension	Specification
1	Input	-	image size: 3 X 64 X 64
2	Convolution	3 X 64 X 64	# of kernel: 128, kernel size: 8 X 8, stride: 1, zero padding: 0
3	Pooling	128 x 57 x 57	max pooling, kernel size: 2 X 2, stride: 2
4	Convolution	128 x 28 x 28	# of kernel: 256, kernel size: 8 X 8, stride: 1, zero padding: 0
5	Pooling	256 x 21 x 21	max pooling, kernel size: 2 X 2, stride: 2
6	Convolution	256 x 10 x 10	# of kernel: 512, kernel size: 4 X 4, stride: 1, zero padding: 0
7	Pooling	512 x 7 x 7	max pooling, kernel size: 2 X 2, stride: 2
8	Fully Connected	512 x 3 x 3	# of neuron: 4096
9	Activation	4096	ReLU
10	Fully Connected	4096	# of neuron: 6
11	Softmax	6	6 classes

P3. CustomLeNet 아키텍처 작성하기

CustomLeNet Code

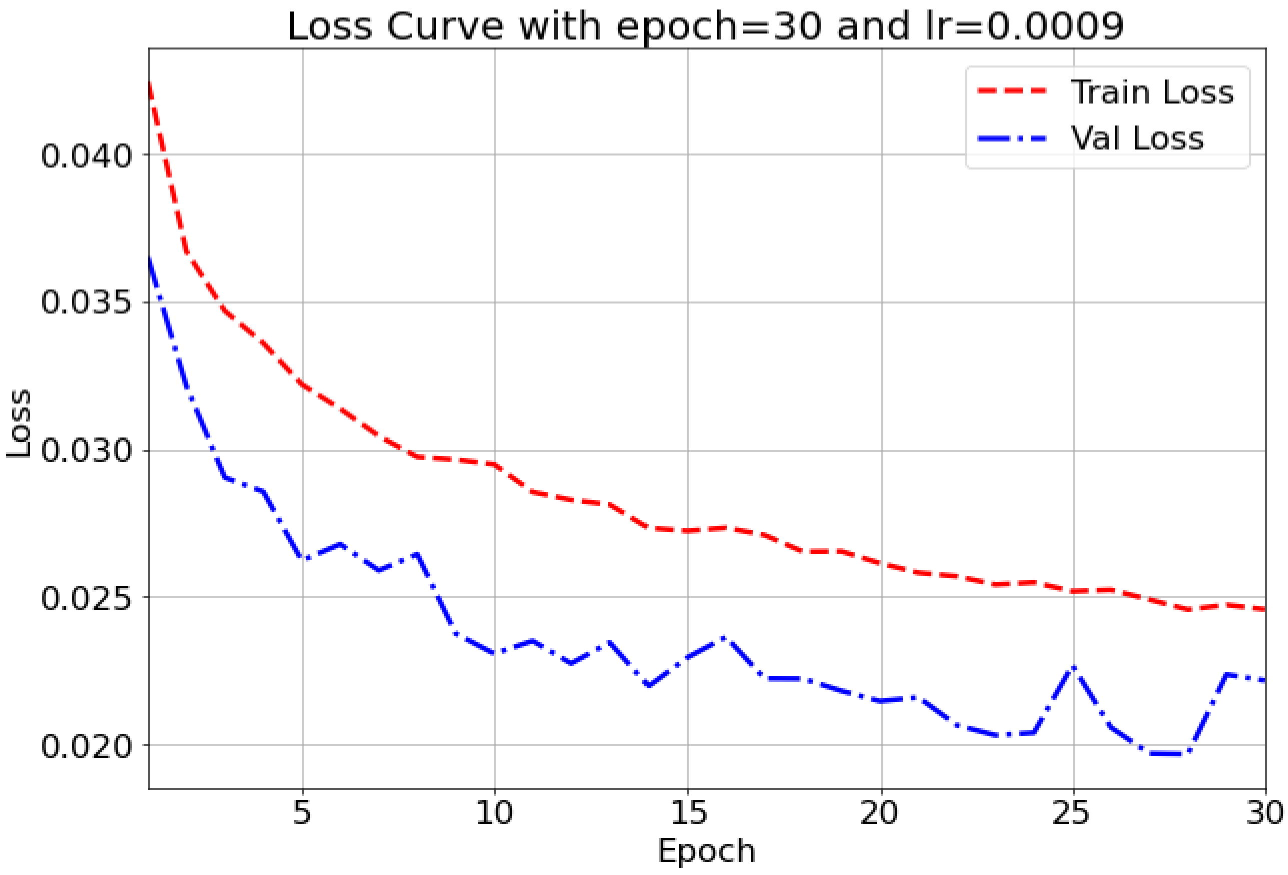
```
class CustomLeNet(nn.Module):
    def __init__(self):
        super(CustomLeNet, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=128, kernel_size=8, stride=1, padding=0)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(in_channels=128, out_channels=256, kernel_size=8, stride=1, padding=0)
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv3 = nn.Conv2d(in_channels=256, out_channels=512, kernel_size=4, stride=1, padding=0)
        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.fc1 = nn.Linear(512 * 3 * 3, 4096)
        self.fc2 = nn.Linear(4096, 6)

    def forward(self, x):
        x = self.pool1(self.conv1(x))
        x = self.pool2(self.conv2(x))
        x = self.pool3(self.conv3(x))
        x = torch.flatten(x, 1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

CustomLeNet Accuracy



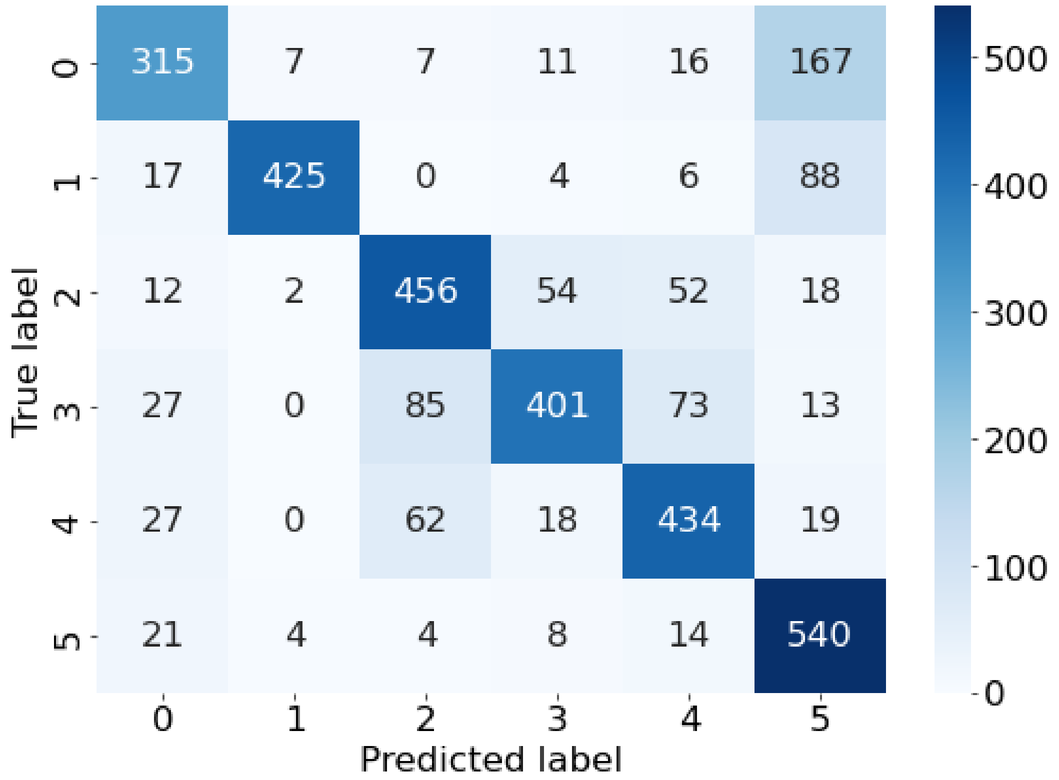
P3. CustomLeNet 아키텍처 작성하기

CustomLeNet Loss



P3. CustomLeNet 아키텍처 작성하기

- Confusion Matrix(CustomLeNet)
  - 전체 평균 정확도 : 0.7546



각 클래스에 따른 정확도	
0	0.6023
1	0.7870
2	0.7677
3	0.6694
4	0.7750
5	0.9137

→ 기존 LeNet(0.7945)보다 전체 평균 정확도가 0.0399 감소함

P4. AlexNet 아키텍처 작성하기

AlexNet 구조

Layer	Type	Input dimension	Specification
1	Input	-	image size: 3 X 64 X 64
2	Convolution	3 x 64 x 64	# of kernel: 96, kernel size: 5 X 5, stride: 1, zero padding: 2
3	Activation	96 x 64 x 64	ReLU
4	Normalization	96 x 64 x 64	LRN (Local Response Normalization), size: 5
5	Pooling	96 x 64 x 64	max pooling, kernel size: 3 X 3, stride: 2
6	Convolution	96 x 31 x 31	# of kernel: 256, kernel size: 5 X 5, stride: 1, zero padding: 2
7	Activation	256 x 31 x 31	ReLU
8	Normalization	256 x 31 x 31	LRN (Local Response Normalization), size: 5
9	Pooling	256 x 31 x 31	max pooling, kernel size: 3 X 3, stride: 2
10	Convolution	256 x 15 x 15	# of kernel: 384, kernel size: 3 X 3, stride: 1, zero padding: 1
11	Activation	384 x 15 x 15	ReLU

AlexNet 구조

Layer	Type	Input dimension	Specification
12	Convolution	384 x 15 x 15	# of kernel: 384, kernel size: 3 X 3, stride: 1, zero padding: 1
13	Activation	384 x 15 x 15	ReLU
14	Convolution	384 x 15 x 15	# of kernel: 256, kernel size: 3 X 3, stride: 1, zero padding: 1
15	Activation	256 x 15 x 15	ReLU
16	Pooling	256 x 15 x 15	max pooling, kernel size: 3 X 3, stride: 2
17	Fully Connected	256 x 7 x 7	# of neuron: 4096
18	Activation	4096	ReLU
19	Dropout	4096	Probability: 0.5
20	Fully Connected	4096	# of neuron: 6
21	Dropout	6	Probability: 0.5
22	Softmax	6	6 classes

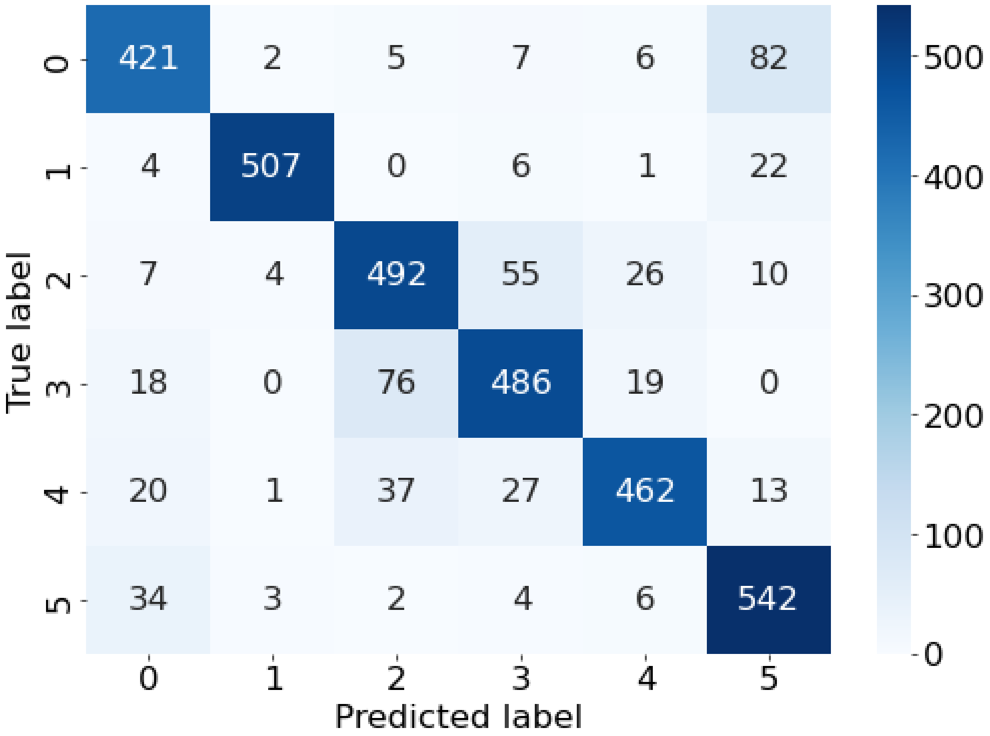
P4. AlexNet 아키텍처 작성하기

AlexNet Code

```
class AlexNet(nn.Module):
    def __init__(self):
        super(AlexNet, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=96, kernel_size=5, stride=1, padding=2, padding_mode='zeros')
        self.LRN1 = nn.LocalResponseNorm(size=5)
        self.pool1 = nn.MaxPool2d(kernel_size=3, stride=2)
        self.conv2 = nn.Conv2d(in_channels=96, out_channels=256, kernel_size=5, stride=1, padding=2, padding_mode='zeros')
        self.LRN2 = nn.LocalResponseNorm(size=5)
        self.pool2 = nn.MaxPool2d(kernel_size=3, stride=2)
        self.conv3 = nn.Conv2d(in_channels=256, out_channels=384, kernel_size=3, stride=1, padding=1, padding_mode='zeros')
        self.conv4 = nn.Conv2d(in_channels=384, out_channels=384, kernel_size=3, stride=1, padding=1, padding_mode='zeros')
        self.conv5 = nn.Conv2d(in_channels=384, out_channels=256, kernel_size=3, stride=1, padding=1, padding_mode='zeros')
        self.pool3 = nn.MaxPool2d(kernel_size=3, stride=2)
        self.fc1 = nn.Linear(256 * 7 * 7, 4096)
        self.Drop1 = nn.Dropout(p=0.5)
        self.fc2 = nn.Linear(4096, 6)
        self.Drop2 = nn.Dropout(p=0.5)

    def forward(self, x):
        x = self.pool1(self.LRN1(F.relu(self.conv1(x),inplace=True)))
        x = self.pool2(self.LRN2(F.relu(self.conv2(x),inplace=True)))
        x = F.relu(self.conv3(x),inplace=True)
        x = F.relu(self.conv4(x),inplace=True)
        x = self.pool3(F.relu(self.conv5(x),inplace=True))
        x = torch.flatten(x, 1)
        x = F.relu(self.fc1(x),inplace=True)
        x = self.Drop1(x)
        x = self.fc2(x)
        x = self.Drop2(x)

        return x
```

P4. AlexNet 아키텍처 작성하기

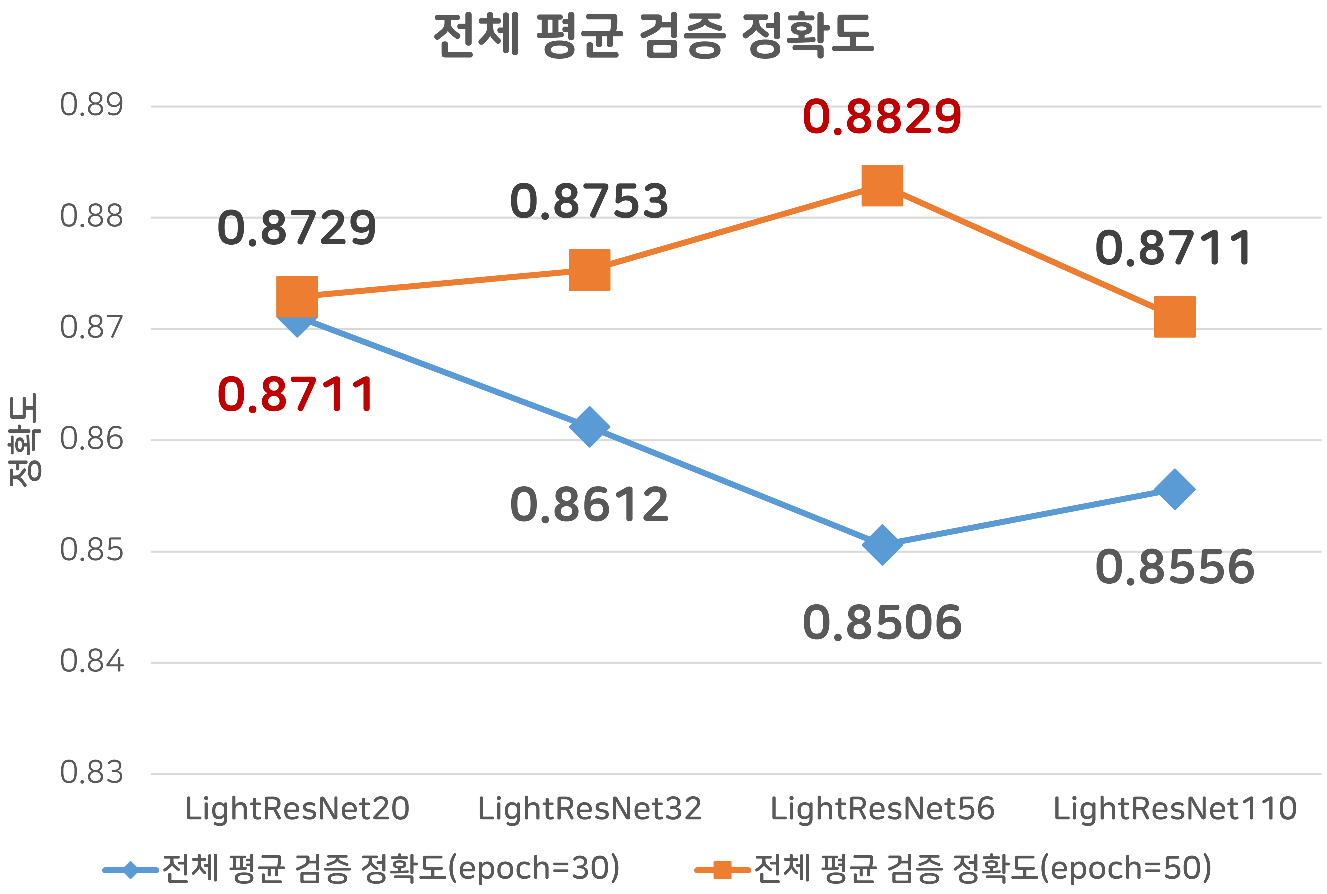
- Confusion Matrix(AlexNet)
  - 전체 평균 정확도 : 0.8541



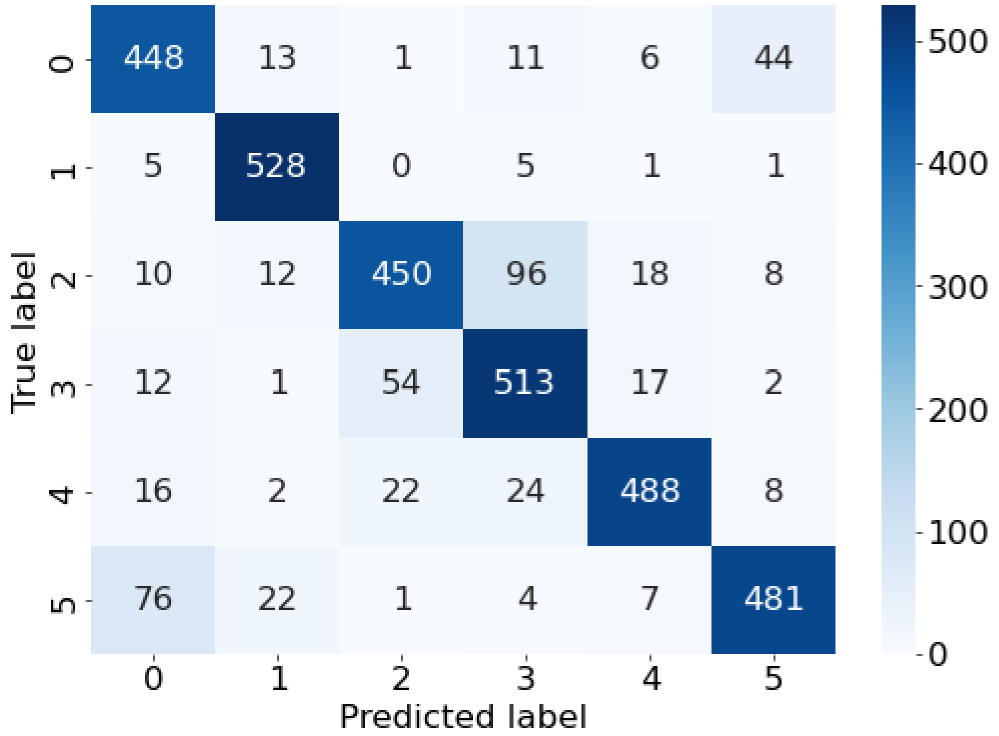
각 클래스에 따른 정확도	
0	0.8050
1	0.9389
2	0.8283
3	0.8114
4	0.8250
5	0.9171

P5. 다양한 레이어 깊이에 따라 LightResNet 학습하기

- Layer가 깊어짐에 따라 성능이 좋지 않음
  - Layer가 **깊을 수록** 학습을 더 진행해야 함



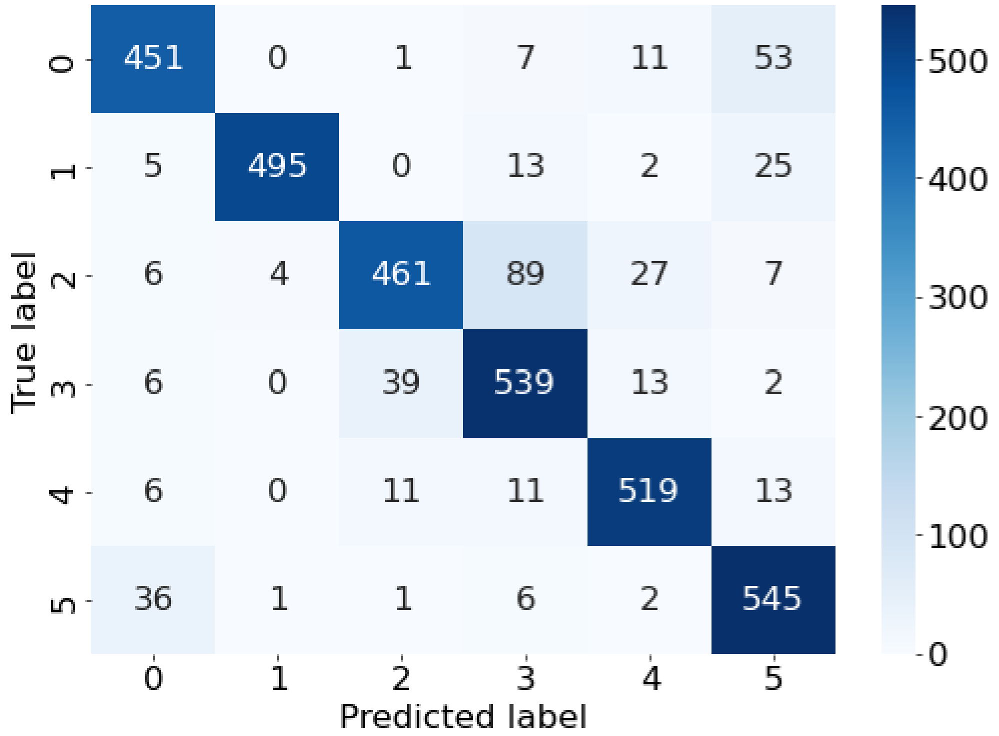
- Confusion Matrix(ResNet18)
  - 전체 평균 정확도 : 0.8576



각 클래스에 따른 정확도	
0	0.9598
1	0.9481
2	0.8754
3	0.7212
4	0.9518
5	0.7157

P6. Mixup 결과 분석하기

- Confusion Matrix(ResNet18+Mixup)
  - 전체 평균 정확도 : 0.8835

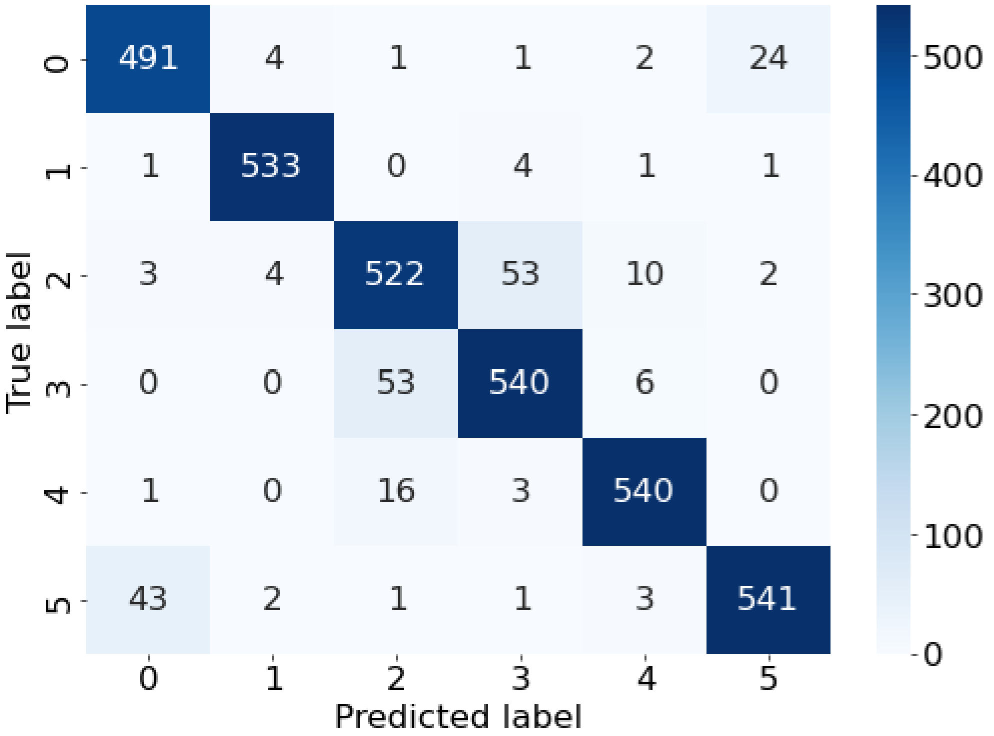


각 클래스에 따른 정확도	
0	0.8623
1	0.9167
2	0.7761
3	0.8998
4	0.9268
5	0.9222

→ 기존 ResNet(0.8576)보다 전체 평균 정확도가 0.0259 증가함

P6. Mixup 결과 분석하기

- Confusion Matrix(ResNet18+Transffered)
  - 전체 평균 정확도 : 0.9296



각 클래스에 따른 정확도	
0	0.9388
1	0.9870
2	0.8788
3	0.9015
4	0.9643
5	0.9154

→ 기존 ResNet(0.8576)보다 전체 평균 정확도가 0.072 증가함

P7. State of the Art 성능 만들기

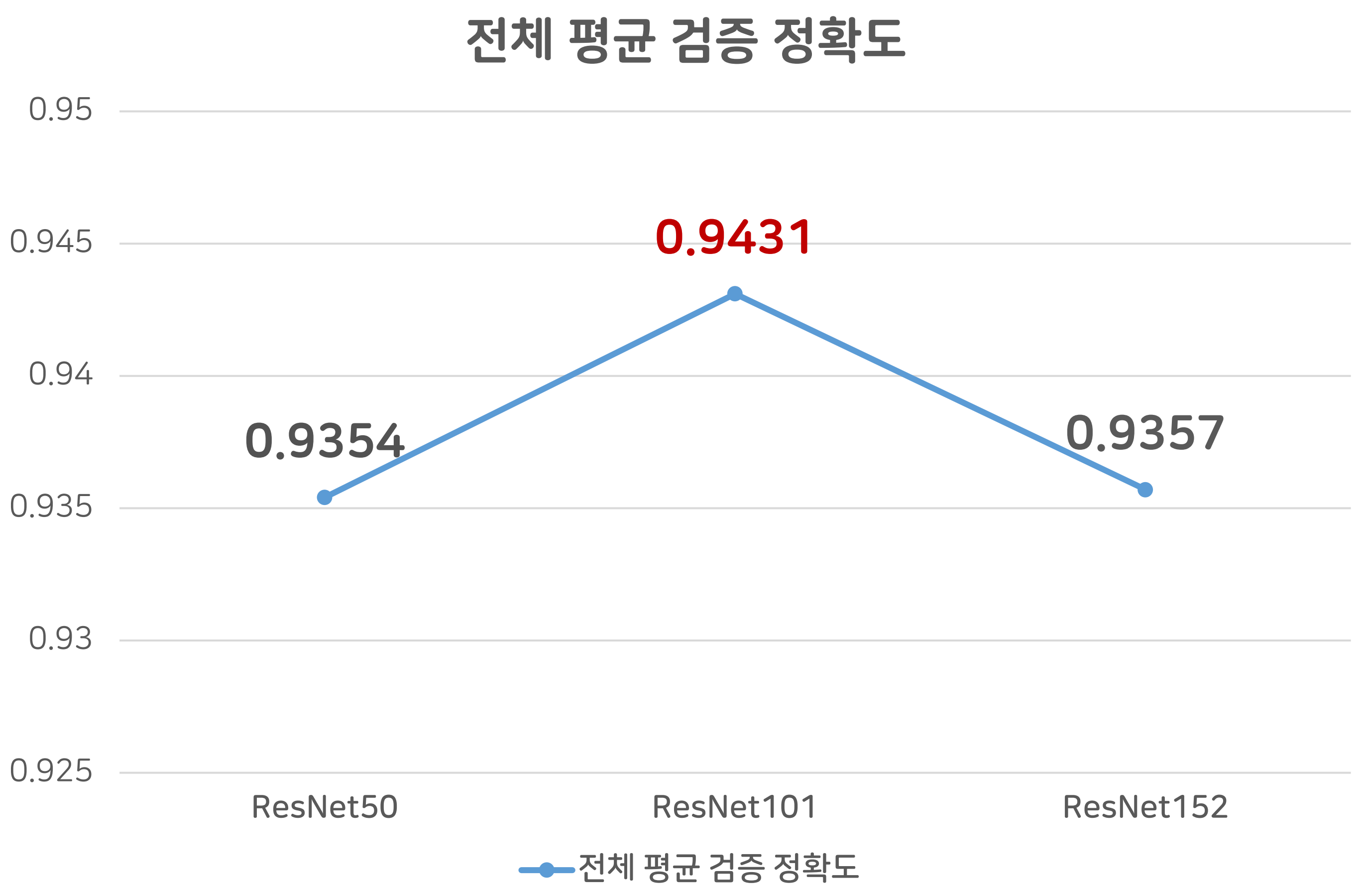
- Model
  - Transffered + Mixup
  - 사용 모델
    - ResNet50
    - ResNet101
    - ResNet152

- 모델 조건
  - Epoch : 50
  - Learning rate : 0.001
  - Optimizer : SGD
  - Scheduler
    - MultiStepLR 사용

$$lr_t = \begin{cases} lr_{t-1} & \text{if } t \leq 10 \\ lr_{t-1} \times 0.5 & \text{elseif } t \leq 20 \\ lr_{t-1} \times 0.25 & \text{elseif } t \leq 30 \\ lr_{t-1} \times 0.125 & \text{elseif } t \leq 40 \\ lr_{t-1} \times 0.0625 & \text{otherwise.} \end{cases}$$

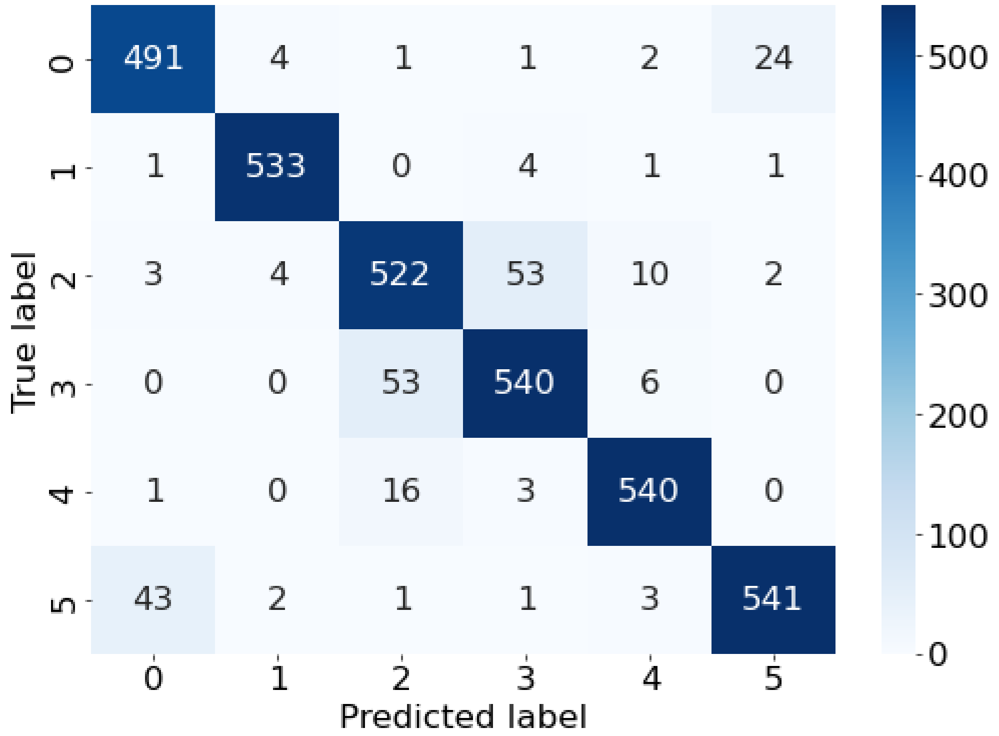
P7. State of the Art 성능 만들기

전체 평균 검증 정확도



P7. State of the Art 성능 만들기

- Confusion Matrix(ResNet101)
  - 전체 평균 정확도 : 0.9431



각 클래스에 따른 정확도	
0	0.9273
1	0.9852
2	0.8704
3	0.9232
4	0.9911
5	0.9662

→ 최종 목표 94% 넘기는 것을 성공함

질문

- Scheduler 설정 방법
  - 좋은 성능을 끌어내기 위하여 고르는 기준
- Hyperparameter
  - 일반적으로 Tuning하는 순서
- Alexnet
  - 원래 (4096,4096)이 들어갔는데 이 모델에서 빠진 이유