CIFAR10 이미지 분류 모델 최종 발표

E조
강한을 김성현 박다영

차례

1. CIFAR-10 dataset
2. Code
   (1) Data Augmentation
   (2) CNN : E-Net
   (3) Epoch 설정
3. Result
4. Conclusion
5. Q&A

```python
transforms_cifar10 = transforms.Compose([
    transforms.Resize((32, 32)),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
                                ])

transforms_cifar10_training = transforms.Compose([
    transforms.Resize((32, 32)),
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomVerticalFlip(p=0.5),
    transforms.RandomRotation(degrees=15),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.2),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
                                ])

# Train dataset
trainset = datasets.CIFAR10(root='./data', train=True, download=True, transform=transforms_cifar10_training)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=32, shuffle=True, num_workers=2)

# Test dataset
testset = datasets.CIFAR10(root='./data', train=False, download=True, transform=transforms_cifar10)
testloader = torch.utils.data.DataLoader(testset, batch_size=32, shuffle=False, num_workers=2)

# Classes of CIFAR-10 dataset
classes = ("plane", "car", "bird", "cat", "deer", "dog", "frog", "horse", "ship", "truck")
```

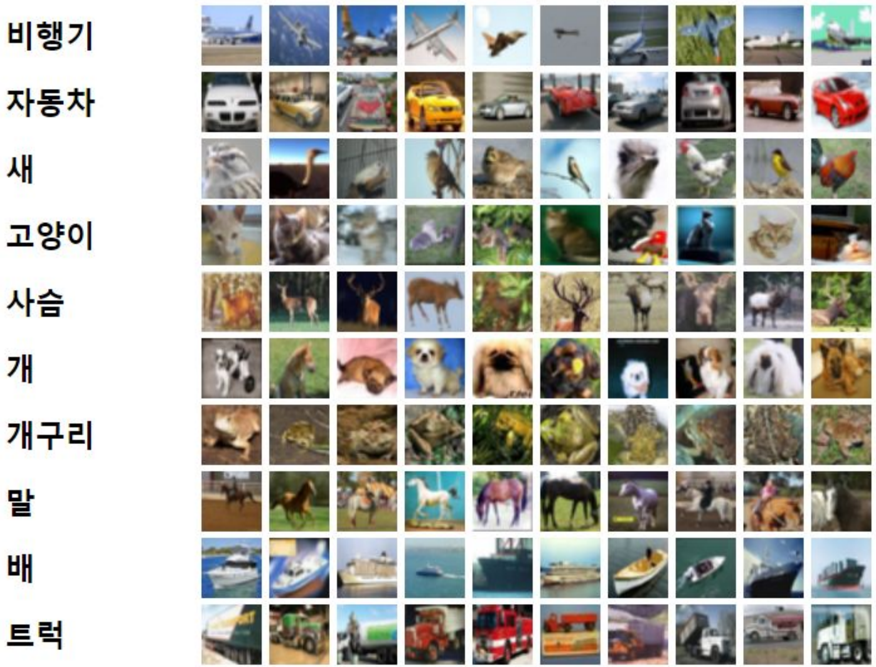
```python
transforms_cifar10 = transforms.Compose([
    transforms.Resize((32, 32)),
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomVerticalFlip(p=0.5),
    transforms.RandomRotation(degrees=15),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.2),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
                                ])

# Train dataset
trainset = datasets.CIFAR10(root='./data', train=True, download=True, transform=transforms_cifar10)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=32, shuffle=True, num_workers=2)

# Test dataset
testset = datasets.CIFAR10(root='./data', train=False, download=True, transform=transforms_cifar10)
testloader = torch.utils.data.DataLoader(testset, batch_size=32, shuffle=False, num_workers=2)

# Classes of CIFAR-10 dataset
classes = ("plane", "car", "bird", "cat", "deer", "dog", "frog", "horse", "ship", "truck")
```

1. CIFAR-10 Dataset



10개의 클래스

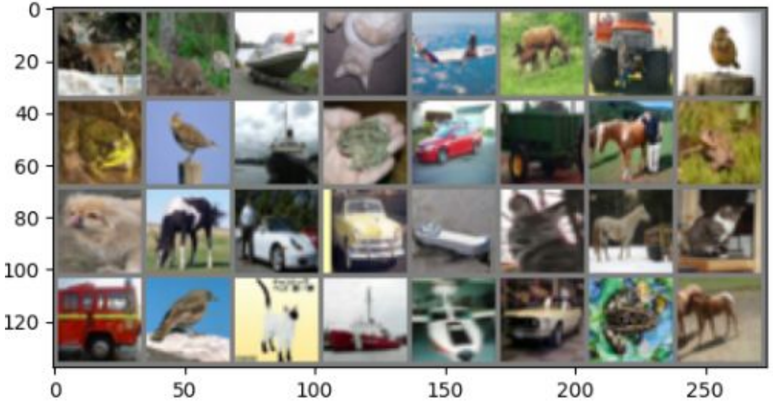
32X32 픽셀, RGB 이미지

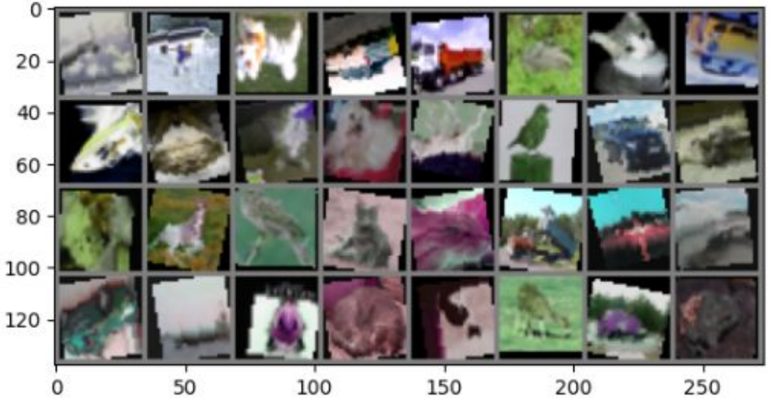
데이터 = 5만 개(train)+ 1만 개(test)

2. Code (1) Data Augmentation

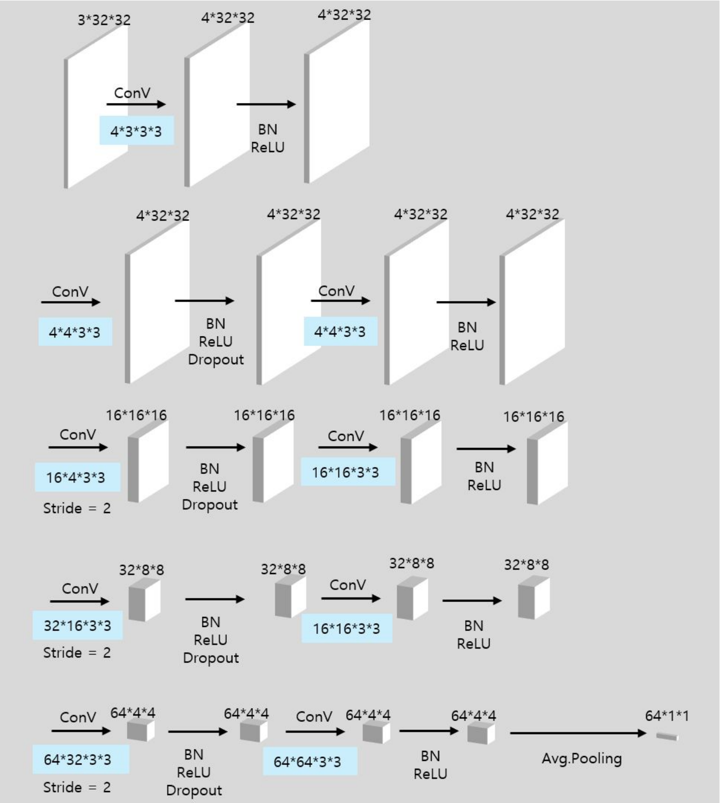
```
transforms_cifar10 = transforms.Compose([
    transforms.Resize((32, 32)),
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomVerticalFlip(p=0.5),
    transforms.RandomRotation(degrees=15),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.2),
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
                                  ])
```

2. Code (1) Data Augmentation





2. Code (2) E-net



ResNet을 베이스로

Convolutional layers

Batch Normalization

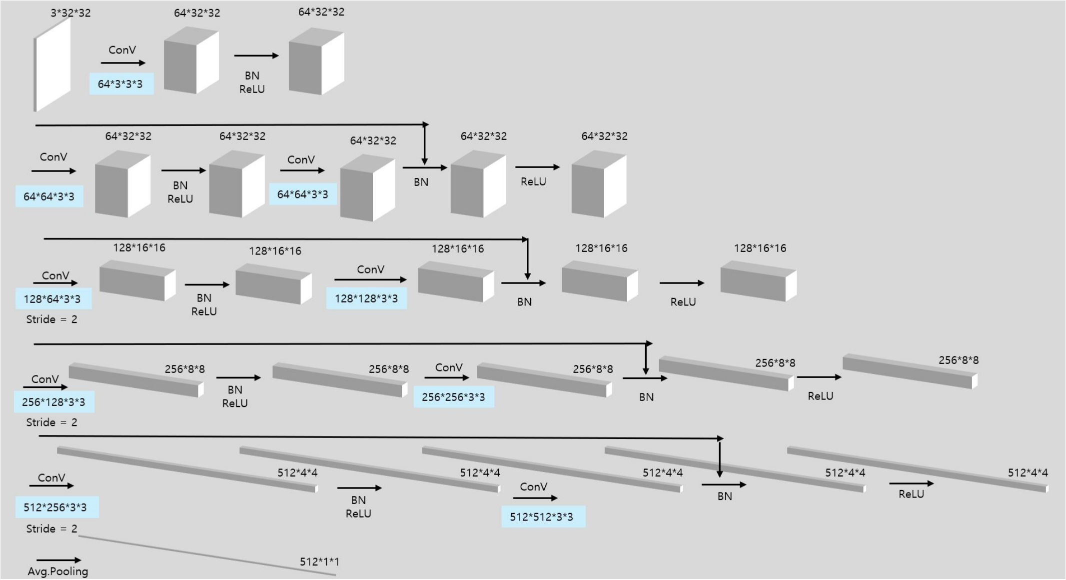
ReLU

Average Pooling

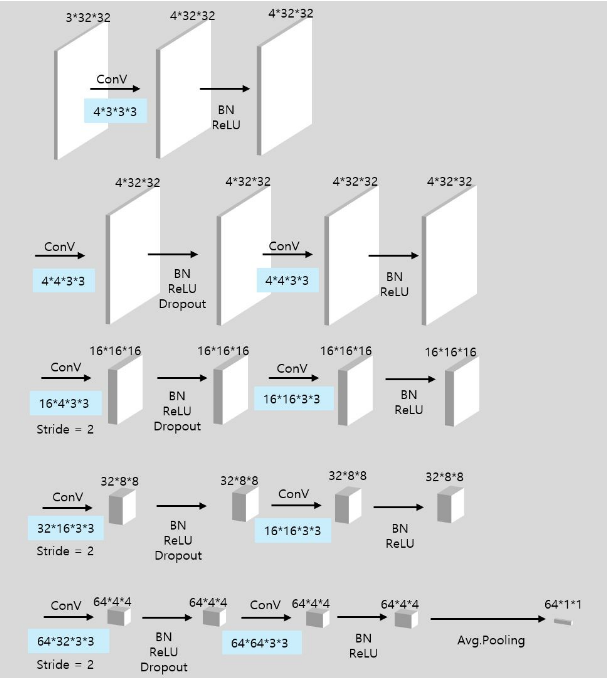
Dropout(0.4)

2. Code (2) E-net

Resnet 흐름



E-net 흐름



2. Code (2) E-net

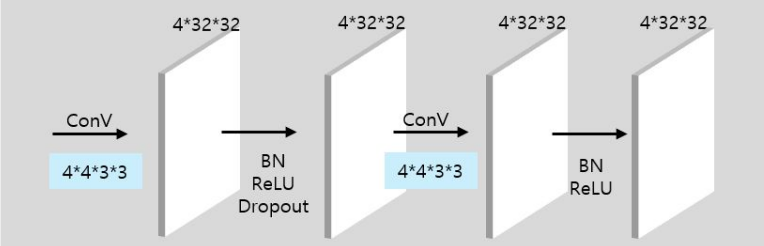
```python
#E조 코드
class BasicBlock(nn.Module):
    def __init__(self, input_channel, output_channel, stride=1):
        super(BasicBlock, self).__init__()

        self.conv1 = nn.Conv2d(input_channel, output_channel, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(output_channel)

        self.conv2 = nn.Conv2d(output_channel, output_channel, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(output_channel)

        self.dropout = nn.Dropout(p=0.4)

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.dropout(out)
        out = F.relu(self.bn2(self.conv2(out)))
        return out
```



2. Code (2) E-net

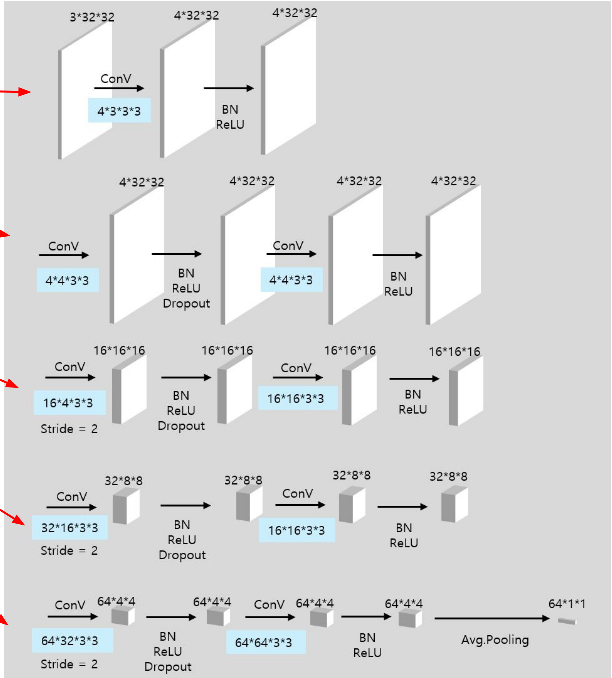
```python
# E_Net 모델 정의
class E_Net(nn.Module):
    def __init__(self):
        super(E_Net, self).__init__()

        self.conv1 = nn.Conv2d(3, 4, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(4)

        self.layer1 = BasicBlock(4, 4, stride=1)
        self.layer2 = BasicBlock(4, 16, stride=2)
        self.layer3 = BasicBlock(16, 32, stride=2)
        self.layer4 = BasicBlock(32, 64, stride=2)
        self.linear = nn.Linear(64, 10)

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = F.avg_pool2d(out, 4)
        out = out.view(out.size(0), -1)
        out = self.linear(out)
        return out

# 모델 초기화
net = E_Net().to(device)
```



2. Code (3) Epoch 코드

기존 코드

```python
# Train the model
epochs = 50

for epoch in range(epochs):

    loss_tmp = 0.0
    epoch_loss = 0.0
    for i, data in enumerate(trainloader, start=0):
        # Load the data
        inputs, labels = data
        inputs = inputs.to(device)
        labels = labels.to(device)

        # Estimate the output using the network
        outputs = net(inputs)

        # Calculate the loss between the output of the network and label
        loss = criterion(outputs, labels)

        # Optimize the network
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        loss_tmp += loss.data
        epoch_loss += loss.data

        if i % 5000 == 4999:    # Print loss every 5000 mini-batches
            print('[Epoch - %d, Iteration - %5d] Loss: %.3f' %
                  (epoch + 1, i + 1, loss_tmp / (i+1)))
            loss_tmp = 0.0

    # Update the learning rate according to the learnig rate scheduler
    scheduler.step()

    # Print the epoch loss
    print('[Epoch - %d] Loss: %.3f' %(epoch + 1, epoch_loss / (i+1)))

print('Finished Training')
```

변경 코드

```python
# 각 학습률에 해당하는 epochs. epoch을 20, 20, 10으로 나누어서 돌린다.
epochs = [20, 20, 10]

for num_epochs in epochs:
    for epoch in range(num_epochs):
        epoch_loss = 0.0  # 각 epoch의 손실 초기화

        for i, data in enumerate(trainloader, start=1):
            # Load the data
            inputs, labels = data
            inputs = inputs.to(device)
            labels = labels.to(device)

            # Estimate the output using the network
            outputs = net(inputs)

            # Calculate the loss between the output of the network and label
            loss = criterion(outputs, labels)

            # Optimize the network
            optimizer.zero_grad()
            loss.backward()  # backpropagation
            optimizer.step()

            epoch_loss += loss.item()  # mini-batch 손실을 누적

            if i % 5000 == 0:  # Print loss every 5000 mini-batches
                print('[Epoch - %d, Iteration - %5d] Loss: %.3f' %
                      (epoch + 1, i, epoch_loss / i))

        # 에포크가 끝날 때마다 해당 에포크의 평균 손실 출력
        print('[Epoch - %d] Loss: %.3f' % (epoch + 1, epoch_loss / len(trainloader)))

        # Update the learning rate according to the learning rate scheduler
        scheduler.step()

    print('Finished Training for current learning rate')

print('Finished Training')
```

3. Result

(1) Batch size

Batch size	16	32	64
정확도(%)	67	87	204

(2) channel size

channel size	4-8-12-16	4-8-12-32	4-16-32-64	16-32-64-128
정확도(%)	75	80	96	132

3. Result

(3) Epoch

Epoch	20회, 20회, 10회 따로	50회 한번에
정확도(%)	92	86

(4) Dropout

Dropout	0.3	0.4	0.5
정확도(%)	105	96	72

4. Result

(6) 최종 정확도

```
Accuracy of the network on the 10,000 test images: 92 %
```

5. Conclusion

(1) Data augmentation training(x) + test(x)

```
Accuracy of the network on the 10,000 test images: 134 %
```

(2) Data augmentation training(o) + Data augmentation test(o)

```
Accuracy of the network on the 10,000 test images: 92 %
```

(3) Data augmentation training(o) + test(x) -> 모델의 성능 일관되게 평가, 모델간 공정한 비교

```
Accuracy of the network on the 10,000 test images: 80 %
```

Q&A