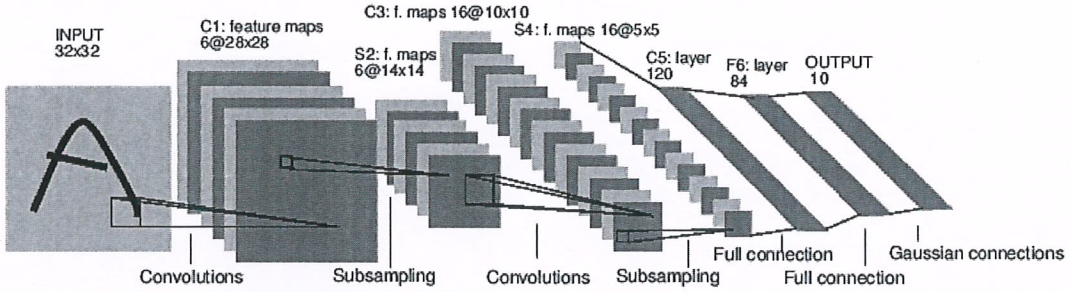
1. [신경망] 다음 코드를 무엇을 의미하는지 이해하고 실행하여 결과를 확인하세요. (14점)

(코드의 해석과 결과의 의미를 작성하세요.)

```
# torch.nn 패키지를 사용하여 신경망을 생성함.
# nn 패키지는 모델을 정의할 때, autograd를 통해 자동미분 기능을 제공함
# nn.Module은 층과 전방전파forward propagation (입력→출력)을 쉽게 구현함
#  참고로 nn 패키지는 mini-batch만 지원함, 예로 nn.Conv2d는 4차 Tensor를 받음 (nSamples * nChannels * height * width)
#  아래 AlexNet (이미 수업에서 학습함) 예시는 숫자를 분류하는 간단한 컨볼루션 신경망의 예임
```




```
# 예시는 사진 입력을 받고, 몇 개의 층에 전방 전파하면서 분류를 출력함
# 출력을 위해서 모델은 다음과 같은 학습을 수행함
# - 신경망은 학습가능한 매개변수들 (가중치들)을 가짐
# - 사진 데이터를 반복적으로 입력함
# - 신경망을 통해 입력을 처리함 (전방 전파)
# - 손실 (오차)를 계산함 (실제 출력과 예측 출력을 비교하여 학습의 올바름을 판단함)
# - 오차로부터 그레이디언트 (경사, 방향)을 신경망의 각 매개변수에게 역전파함 (오류 역전파)
# - 신경망의 매개변수들을 갱신함 ( (미래)가중치 = (현재)가중치 – 학습률 * 그레이디언트 )


# 위의 컨볼루션 신경망의 부분들을 torch를 통해서 손쉽게 구현할 수 있음
#  단지forward함수만정의하면,autograd를이용해해당연산그래프의그레이디언트를구하는backward자동적으로정의됨
# forward 함수는 Tensor를 이용할 수 있는 다양한 연산들 (합, 곱 등등) 사용하여 정의 가능함
# torch.Tensor: 자동 미분 기능을 지원하는 다차원 배열, 각 Tensor에 해당하는 그레이디언트를 가짐
# nn.Module: 신경망 모듈이며 매개변수의 캡슐화, GPU 연산 등 작업을 쉽게 가능하게 함
# nn.Parameter: 모듈이 지정되면 매개변수들을 자동으로 관리하는 Tensor의 하나임

import torch
import torch.nn as nn
import torch.nn.functional as F

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

```
        # 1 input image channel, 6 output channels, 5x5 square convolution
        # kernel
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square you can only specify a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

    def num_flat_features(self, x):
        size = x.size()[1:]  # all dimensions except the batch dimension
        num_features = 1
        for s in size:
            num_features *= s
        return num_features

net = Net()
print(net)
# (1) 화면 출력 확인 및 의미를 서술

# (2) 정의된 컨볼루션 신경망의 구조 설명 (위의 AlexNet 그림 참고)

# net.parameters()를 사용하여 정의된 신경망의 학습가능한 매개변수들을 확인할 수 있음
params = list(net.parameters())
print(len(params))
print(params[0].size())  # conv1's .weight
# (3) 화면 출력 확인
```

```
# 다음의 임의의 32*32 입력을 가정함
# 참고로 크기가 다른 입력을 받을 때는 입력의 크기를 재조정하거나 신경망 수정함
input = torch.randn(1, 1, 32, 32)
out = net(input)
print(out)
# (4) 화면 출력 확인


# 오류역전파를 통해 그레이디언트를 구하기 전에 모든 가중치의 그레이디언트 버퍼들을 초기화
net.zero_grad()
out.backward(torch.randn(1, 10))

# 손실 함수 정의 및 임의의 값들에 대해서 오차 결과 확인
# nn 패키지는 많이 사용되는 손실함수들을 제공하며, 해당 예제는 단순한 MSE를 사용
output = net(input)
target = torch.randn(10)  # a dummy target, for example
target = target.view(1, -1)  # make it the same shape as output
criterion = nn.MSELoss()

loss = criterion(output, target)
print(loss)
# (5) 화면 출력 확인


# 앞에 코드에서 언급한 것과 같이 오류 역전파하기 전, 그레이디언트를 초기화해야 함
# backward() 수행 후 어떤 변화가 있는지 확인하고, 초기화의 필요성을 확인함
net.zero_grad()       # zeroes the gradient buffers of all parameters

print('conv1.bias.grad before backward')
print(net.conv1.bias.grad)
# (6) 화면 출력 확인


loss.backward()

print('conv1.bias.grad after backward')
print(net.conv1.bias.grad)
# (7) 화면 출력 확인
```

```
# 스토캐스틱경사하강법((미래)가중치=(현재)가중치-학습률*그레이디언트)을이용하여가중치갱신하는코드는다음과같음
learning_rate = 0.01
for f in net.parameters():
    f.data.sub_(f.grad.data * learning_rate)

# 하지만 위 구현 코드보다 실제, torch.optim에서 구현되는 SDG, Adam, RMSProp 등을 사용함
# 오류 역전파에서 최적화하는 방법을 보인 예제 코드
import torch.optim as optim

# create your optimizer
optimizer = optim.SGD(net.parameters(), lr=0.01)

# in your training loop:
optimizer.zero_grad()   # zero the gradient buffers
output = net(input)
loss = criterion(output, target)
loss.backward()
optimizer.step()    # Does the update
```

2. [분류기 학습] 다음 코드를 무엇을 의미하는지 이해하고 실행하여 결과를 확인하세요. (14점)

(코드의 해석과 결과의 의미를 작성하세요.)

```
# 1번의 기초적인 신경망을 통해서 사진 분류기를 학습함
# 데이터집합은 CIFAR-10이며, 아래의 예와 같이 10가지의 3 (R, G, B)채널의 32*32 크기의 사진으로 구성됨
```


```
# CIFAR-10과 같이 많이 사용되는 데이터집합은 torchvision 패키지에서 제공함
# 분류기 학습은 다음과 같은 과정을 가짐
# 1. 정규화된 CIFAR-10 훈련집합과 테스트집합을 torchvision을 이용하여 적재함
```

```
# 2. 컨볼루션 신경망을 정의함
# 3. 손실함수 정의
# 4. 훈련집합을 이용하여 신경망을 학습시킴
# 5. 테스트집합을 이용하여 신경망 성능 확인

### 1. 정규화된 CIFAR-10 훈련집합과 테스트집합을 torchvision을 이용하여 적재함
import torch
import torchvision
import torchvision.transforms as transforms

transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4, shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=4, shuffle=False, num_workers=2)

classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
# (1) 화면 출력 확인

# 훈련집합의 일부 사진들 확인
import matplotlib.pyplot as plt
import numpy as np

# functions to show an image
def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))

# get some random training images
dataiter = iter(trainloader)
images, labels = dataiter.next()

# show images
imshow(torchvision.utils.make_grid(images))
# print labels
print(' '.join('%5s' % classes[labels[j]] for j in range(4)))
```

 # (2) 화면 출력 확인

```
### 2. 컨볼루션 신경망을 정의함
# 3채널 32*32 크기의 사진을 입력받고, 신경망을 통과해 10 부류를 수행
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

net = Net()

### 3. 손실함수 정의, 교차 엔트로피와 SGD+momentum
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

### 4. 훈련집합을 이용하여 신경망을 학습시킴
for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
```

```
    for i, data in enumerate(trainloader, 0):
        # get the inputs
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 1000 == 999:      # print every 1000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 1000))
            running_loss = 0.0

print('Finished Training')
# (3) 화면 출력 확인 및 학습이 되고 있는지 서술


### 5. 테스트집합을 이용하여 신경망 성능 확인
dataiter = iter(testloader)
images, labels = dataiter.next()

# print images
imshow(torchvision.utils.make_grid(images))
print('GroundTruth: ', ' '.join('%5s' % classes[labels[j]] for j in range(4)))
# (4) 화면 출력 확인

outputs = net(images)
_, predicted = torch.max(outputs, 1)
print('Predicted: ', ' '.join('%5s' % classes[predicted[j]] for j in range(4)))
# (5) 화면 출력 확인

# performance on the whole test dataset
```

```
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 10000 test images: %d %%' % (100 * correct / total))
# (6) 화면 출력 확인 및 일반화 성능 서술


# performance on each class
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(4):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1

for i in range(10):
    print('Accuracy of %5s : %2d %%' % (
        classes[i], 100 * class_correct[i] / class_total[i]))
# (7) 화면 출력 확인 및 부류별 분류기의 성능 서술
```

3. 다음 조건을 만족하는 컨볼루션 신경망을 구현하고, 2번의 (3), (6), (7)의 결과를 확인하고 비교하세요.

(1) INPUT-CONV(32 3*3)-CONV(32 3*3)-RELU-POOL-CONV(32 3*3)- CONV(32 3*3)-RELU-POOL-FC-OUTPUT (15점)

(2) 2번 문제의 신경망에 Adam 최적화 (강의자료의 기본 하이퍼-매개변수 사용) 적용 (3점)

(3) 데이터 확대 방법들 중 하나를 적용한 후, 2번 문제의 신경망 학습 (Hint: transforms) (3점)

(4) 2번 문제의 신경망에 CONV 층마다 배치 정규화를 적용 (Hint: nn.BatchNorm) (3점)

(5) 2번 문제의 신경망에 로그우도 손실함수를 적용 (3점)

(6) 2번 문제의 신경망에 L2 놈 적용 (3점)

4. 신경망의 출력이 $(0.4, 2.0, 0.001, 0.32)^{\mathrm{T}}$일 때 소프트맥스 함수를 적용한 결과를 쓰시오. (6점)

5. 소프트맥스 함수를 적용한 후 출력이 $(0.001, 0.9, 0.001, 0.098)^{\mathrm{T}}$이고 레이블 정보가 $(0, 0, 0, 1)^{\mathrm{T}}$일 때, 세 가지 목적함수, 평균제곱 오차, 교차 엔트로피, 로그우도를 계산하시오. (6점)