1.0 Define Hyper-parameters

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Hyper-parameters
batch_size = 4
max_pool_kernel = 2
learning_rate = 0.0001
num_epochs = 2

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

1.1 Load Data

```
train_data = torchvision.datasets.CIFAR10(root='./datasets',
                                        train=True,
                                        transform=transform,
                                        download=True)

test_data = torchvision.datasets.CIFAR10(root='./dataset',
                                       train=False,
                                       transform=transform,
                                       download=True)
```

1.2 Define Dataloader

```
train_loader = torch.utils.data.DataLoader(dataset=train_data,
                                           batch_size=batch_size,
                                           shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_data,
                                          batch_size=batch_size,
                                          shuffle=True)
```

1.3 Define Model

```
class ConvNet(nn.Module):
  def __init__(self, num_classes=10):
    super(ConvNet, self).__init__()
    self.layer1 = nn.Sequential(
        nn.Conv2d(3, 6, 5),
        nn.BatchNorm2d(6),
        nn.ReLU(),
        nn.MaxPool2d((2,2),stride=2)
    )
    self.layer2 = nn.Sequential(
        nn.Conv2d(6, 16, 5),
        nn.BatchNorm2d(16),
        nn.ReLU(),
        nn.MaxPool2d((2,2),stride=2)
    )
```

```python
    self.fc1 = nn.Linear(400, 120)
    self.fc2 = nn.Linear(120, 84)
    self.fc3 = nn.Linear(84,10)


  def forward(self, x):
    x = self.layer1(x)
    x = self.layer2(x)
    x = x.reshape(x.size(0),-1)
    x = F.relu(self.fc1(x))
    x = self.fc2(x)
    x = self.fc3(x)
    return x
```

Size 공식

Convolution layer의 output tensor size

- 각각 기호를 아래와 같이 정의
  - O: Size(width) of output image
  - I: Size(width) of input image
  - K: Size(width) of kernels used in the Conv layer
  - N: Number of kernels
  - S: Stride of the convolution operation
  - P: Padding size
- O(Size(width) of output image)는 다음과 같이 정의 됨

$$O = \frac{I - K + 2P}{S} + 1$$

- 출력 이미지의 채널 수는 커널의 갯수(N)와 같음

MaxPool layer의 output tensor size

- 각각 기호를 아래와 같이 정의
  - O: Size(width) of output image
  - I: Size(width) of input image
  - S: Stride of the convolution operation
  - P_s: Pooling size
- O(Size(width) of output image)는 다음과 같이 정의 됨

$$O = \frac{I - P_s}{s} + 1$$

- Convolution layer와는 다르게 출력의 채널 수는 입력의 개수와 동일
- Conv layer의 O 수식에서 커널 크기(K)를 P_s로 대체하고 $P = 0$으로 설정하면 동일한 식이 됨

Size 변화

32 x 32 x 3 -> conv2d -> 28 x 28 x 6 -> maxpooling -> 14 x 14 x 6 -> conv2d -> 10 x 10 x 16 -> maxpooling -> 5 x 5 x 16 -> FC -> 120 -> FC -> 84 -> FC -> 10

Model

```
ConvNet(
  (layer1): Sequential(
    (0): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
    (1): BatchNorm2d(6, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=(2, 2), stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (layer2): Sequential(
    (0): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
    (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=(2, 2), stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (fc1): Linear(in_features=400, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=10, bias=True)
)
```

1.4 Set Loss $ Optimizer

```
 criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr = learning_rate)
```

1.5 Train / Test

```
total_step = len(train_loader)
loss_list = []

# Train
for epoch in range(num_epochs):
  for i, (images, labels) in enumerate(train_loader,0):
    # Assign Tensors to Configured Device
    images = images.to(device)
    labels = labels.to(device)

    # Forward Propagation
    outputs = model(images)

    # Get Loss, Compute Gradient, Update Parameters
    loss = criterion(outputs, labels)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    # Append loss to plot graph
    loss_list.append(loss)

    # Print Loss for Tracking Training
    if (i+1) % 2000 == 0:
      acc = 0
      test_image, test_label = next(iter(test_loader))
      _, test_predicted = torch.max(model(test_image.to(device)).data, 
1)

      for (pred, ans) in zip(test_predicted, test_label):
        if pred == ans:
          acc += 1
      acc = acc / len(test_predicted)

      print('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}, Accuracy: {:.1f
}%'.format(epoch+1,num_epochs, i+1, total_step, loss.item(), acc*100))
      print('Testing data: [Predicted: {} / Real: {}]'.format(test_pred
icted, test_label))

  if epoch+1 == num_epochs:
    torch.save(model.state_dict(), 'model.pth')
  else:
    torch.save(model.state_dict(), 'model-
{:02d}_epochs.pth'.format(epoch+1))


test_model.eval()
```

```
with torch.no_grad():
  correct = 0

  for img, lab in test_loader:
    img = img.to(device)
    lab = lab.to(device)
    out = test_model(img)
    _, pred = torch.max(out.data, 1)
    correct += (pred == lab).sum().item()

  print("Accuracy of the network on the {} test images: {}%".format(len
(test_loader)*batch_size, 100 * correct / (len(test_loader) * batch_siz
e)))
```

1.6 Result

```
Accuracy of the network on the 10000 test images: 54.51%
```