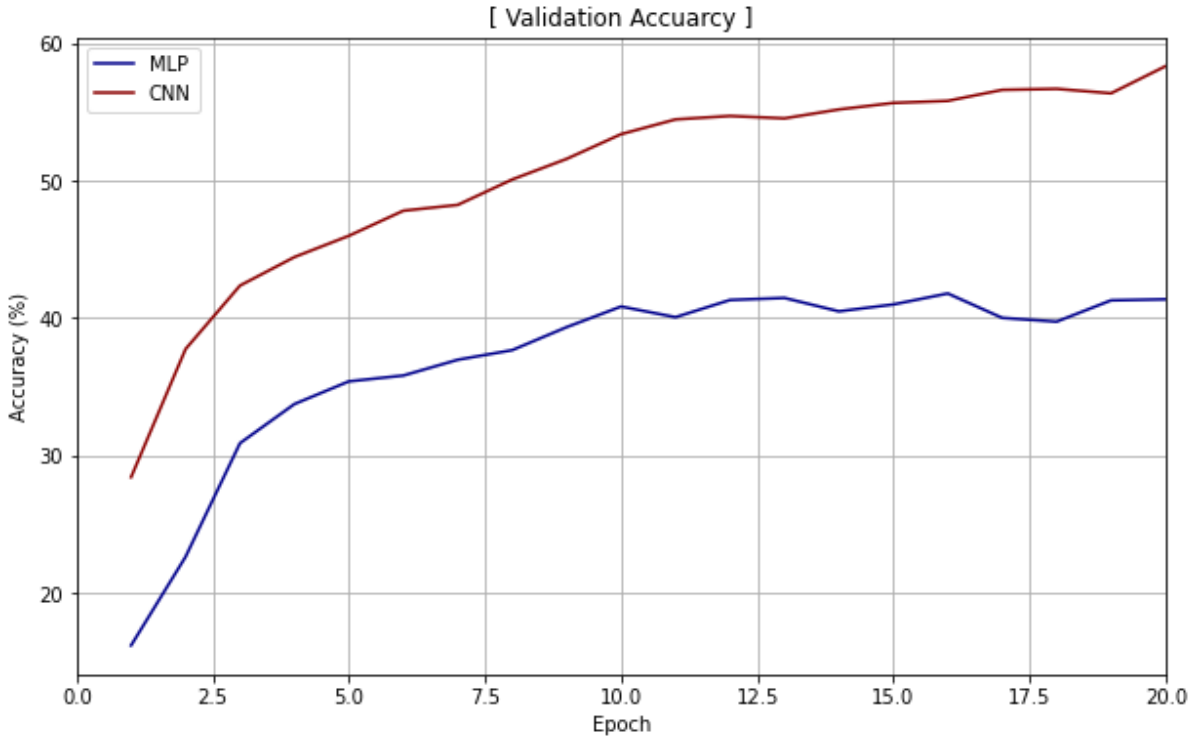
AI Assignment 04 보고서

```
AI Assignment04 보고서
학번 : 20171612
학과 : 컴퓨터공학과
이름 : 김성일
```

결과

```
----------- MLP Test Result ----------------
MLP Test Accuracy: 40.825

----------- CNN Test Result ----------------
CNN Test Accuracy: 57.75
```



결과를 통해 과제에서 요구한, **MLP에서의 40% 이상의 정확도와 CNN에서의 50% 이상의 정확도**를 가지고 있다는 것을 알 수 있습니다.

문제 1 - Flatten Function 구현

```
def flatten(x):
    data = x.shape[0]
    ###################### Write Your Code Here ##################
    flatten_x = x.view(data,-1)
    ##############################################################
    return flatten_x
```

flatten을 위한 함수를 구현하기 위해, 내장 함수인 view를 사용하였습니다.

문제 2 - MLP 구현

```
class MLP(nn.Module):
    def __init__(self,input_size,active_func,output_size):
        super(MLP,self).__init__()

        self.activation_func=active_func
        ######################### Write your Code Here ########################
        self.layers = [
            nn.Linear(input_size, 1024),
            self.activation_func,
            nn.Linear(1024, 512),
            self.activation_func,
            nn.Linear(512, 256),
            self.activation_func,
            nn.Linear(256, 128),
            self.activation_func,
            nn.Linear(128, 64),
            self.activation_func,
            nn.Linear(64, 32),
            self.activation_func,
            nn.Linear(32, output_size),
        ]

        self.seq = nn.Sequential(*self.layers)
        ######################################################################

    def forward(self,x):
        x = flatten(x)  # flatten layer: 위에서 정의한 flatten 함수를 먼저 작성해야 한다.
        x = self.seq(x) # 입력값이 하나인 것을 생각하고 network를 짜야한다.
        return x
```

문제에서의 요구대로 5개 이상 10개 이하의 MLP를 구현하기 위해 input_size → 1024 → 512 → 256 → 128 → 64 → 32 → 10 의 순서를 통해 계산 되는 7 layer의 MLP를 구현했

습니다.

각 Layer는 torch.nn의 Linear 함수를 통해 진행되었으며, Linear 함수가 끝나고 self.activation_func 을 실행하도록 하였습니다.

이 모든 일련의 작업은 nn.Sequential 함수를 통해 정의한 self.layers의 값을 대입함으로써 순차적으로 실행되게 하였습니다.

그렇게 나온 결과는 앞서 언급했던 것과 같이 **40.825 %** 의 결과를 보였습니다.

문제 3 - CNN 구현

```
class CNNModel(nn.Module):
    def __init__(self,input_channel,active_func):
        super(CNNModel, self).__init__()

        self.activation_func=active_func

        ######################### Write your Code Here ##########################
        self.seq = nn.Sequential(
            # input depth(input channel), output depth, kernel size
            nn.Conv2d(3,12,5, padding=1),
            self.activation_func,
            nn.Conv2d(12,12,5, padding=1),
            self.activation_func,
            nn.MaxPool2d(2),
            nn.Conv2d(12,24,5, padding=1),
            self.activation_func,
            nn.Conv2d(24,24,5, padding=1),
            self.activation_func,
            nn.MaxPool2d(2),
        )

        self.seq2 = nn.Sequential(
            nn.Linear(5*5*24, 128),
            self.activation_func,
            nn.Linear(128, 64),
            self.activation_func,
            nn.Linear(64, 10)
        )
        ###########################################################################

    def forward(self, output):
        ######################### Write your Code Here ##########################
        output = self.seq(output)
        output = flatten(output)
```

```
        output = self.seq2(output)
        return output
        ########################################################################
```

INPUT(32X32X3) --> CONV1(30X30X12) --> CONV2(28X28X12) --> POOL1(14X14X12) --> CONV3(12X12X24) --> CONV4(10X10X24) --> POOL2(5X5X24) --> FC(128) --> FC(64) --> FC (10)

문제에서 요구한 것과 같이 아래와 같은 단계를 거쳐 output을 만들어내게 됩니다.

Conv2d는 input channel, output channel, kernel, padding 그리고 다른 여러 인자를 사용하게 되는데

input Tensor는 (C_{in}, H_{in}, W_{in}) 의 shape을 가지며, output Tensor는 $(C_{out}, H_{out}, W_{out})$ 의 shape을 가집니다.

이것을 정리하면 아래와 같고,

Input Tensor

- C_in : input channel
- H_in : input tensor 의 height
- W_in : input tensor 의 width

Output Tensor

- C_out : output channel
- H_out : output tensor 의 height
- W_out : output tensor 의 width

위를 정리하여 공식으로 나타내면 아래와 같습니다.

$$H_{out} = \lfloor \frac{H_{in} + 2 \times padding - dilation \times (kernel\_size - 1) - 1}{stride} + 1 \rfloor$$
$$W_{out} = \lfloor \frac{W_{in} + 2 \times padding - dilation \times (kernel\_size - 1) - 1}{stride} + 1 \rfloor$$

dilation, stride 는 기본값을 1로 가지고 있으며, 저는 모든 Conv2d 함수에 padding을 1로 전달하였기 때문에 이것을 정리하여 나타내면 아래와 같습니다.

$$H_{out} = H_{in} + 3 - kernel\_size$$
$$W_{out} = W_{in} + 3 - kernel\_size$$

그렇기에 현재 이전 Tensor에서 다음 Tensor로 넘어갈 때, height와 width가 각 2씩 줄어들 고 있으므로 kernel_size는 모두 동일하게 5를 사용하면 됩니다.

그리고 Pool을 통해 height와 width를 각각 절반의 크기로 줄일 때는 pytorch의 nn.MaxPool2d(2) 함수를 사용하였습니다. 이를 통해 height와 width를 각각 절반의 크기로 줄일 수 있었습니다.

이 일련의 과정은 nn.Sequential를 통해 정의 된 self.seq 를 사용하였으며, 그 후 아래의 과정은 flatten layer를 통과시킨 후 아래와 같은 순서로 self.seq2를 통해 진행하였습니다.

FC(128) --> FC(64) --> FC (10)

MaxPool2d와 flatten layer를 지날 때를 제외하고는 모두 self.activation_func을 실행하도록 하였습니다.

그렇게 나온 결과는 앞서 언급했던 것과 같이 **57.75 %** 의 결과를 보였습니다.

문제 4 - activation function

```
active_function=choice_activation("relu")
```

저는 코드에서 나와있는 것과 같이 activation function으로 ReLU 함수를 사용하였습니다.

ReLU 함수의 정의는 아래와 같습니다.

$$f(x) = max(0, x)$$

ReLU 함수는 Vanishing Gradient 문제가 발생하지 않으며, 다른 기존의 활성화 함수에 비해 매우 빠릅니다.

실제로 시그모이드나 하이퍼볼릭 탄젠트 함수에 비해 6배 정도 빠른 결과를 보여줍니다.

그리고, Multi-layer에서 사용하게 될 때 단순히 Linear한 형태에서 더 나아가 Linear한 부분 부분의 결합의 합성함수를 만들 수 있게 되므로 최종적으로 Non Linear한 성질을 띄게 됩니다. 또한, 함수의 구현이 매우 간단하며 연산 비용이 적다는 특징이 있습니다.

또, 실제로 다른 활성화 함수를 선택하여 accuracy를 측정하였을 때, 어떤 활성화 함수는 MLP에서 좋은 성능을 보이지만 CNN에서는 좋지않은 성능을 보여주는 경우도 있었으며 반대로 MLP에서는 좋지않지만 CNN에서는 좋은 성능을 보여주는 활성화 함수도 있었습니다. 하지만 ReLU는 다른 함수들과 달리 MLP, CNN 두가지 경우 모두에서 적절한 성능을 보여주었고 과제에서 원하는 결과를 보여주었습니다.

따라서 결과적으로 위와 같은 이유로 ReLU를 선택하게 되었습니다.