Implement a convolutional neural network (CNN) in PyTorch that processes 32×32 RGB images and defines a custom loss function. Your implementation must use the exact variable and function names given below so that the autograder can verify your solution automatically.¶

In [ ]:

```
      +---------------------+
      |  Input: 32×32×3     |
      +----------+----------+
                 |
                 v
      +---------------------+
      | Conv1: 3×3, s=1,    |
      | no padding, 32 ch   |
      | Output: 30×30×32    |
      +----------+----------+
                 |
                 v
      +---------------------+
      | MaxPool1: 2×2, s=2  |
      | Output: 15×15×32    |
      +----------+----------+
                 |
                 v
      +---------------------+
      | Conv2: 3×3, s=1,    |
      | no padding, 64 ch   |
      | Output: 13×13×64    |
      +----------+----------+
                 |
                 v
      +---------------------+
      | MaxPool2: 2×2, s=2  |
      | Output: 6×6×64      |
      +----------+----------+
                 |
                 v
      +---------------------+
      | Flatten             |
      | (6×6×64 = 2304)     |
      +----------+----------+
                 |
                 v
      +---------------------+
      | Fully Connected     |
      | 2304 → 10           |
      +---------------------+
```

1. Network Architecture¶ Implement a CNN with the following specifications:

Input:¶ Images of size 32×32 with 3 channels (RGB).

Block 1:¶ Convolutional Layer (conv1): kernel_size: 3×3 stride: 1 padding: 0 (i.e. no padding) in_channels: 3 out_channels: 32 Max-Pooling Layer (pool): kernel_size: 2 stride: 2

Block 2:¶ Convolutional Layer (conv2): kernel_size: 3×3 stride: 1 padding: 0 Output channels: 64 Max-Pooling Layer (pool): kernel_size: 2 stride: 2

Fully Connected Layer (fc):¶ After the two convolution+pooling blocks, flatten the output and add a fully connected layer that maps the flattened features to 10 output classes.

forward Method Your forward method should perform the following steps exactly:

Apply ReLU activation after each convolution. Use the pooling layer after each convolution block. Flatten the output before passing it to the fully connected layer. Finally, produce the logits via the fully connected layer. Variable Name for the Model:¶ The final model must be stored in a variable called myModel.

In [4]:

```
import torch
import torch.nn as nn
```

In [8]:

```
class myCNN(nn.Module):
    def __init__(self,num_classes=10,input_size=32):
        """
        Constructs a CNN with two convolution+maxpool blocks followed by a fully connected layer.

        Specifications:
          - Input: images of size 32x32 with 3 channels.
          - Block 1:
              * Conv layer with kernel_size 3x3, stride 1, no padding, output channels = 32.
              * MaxPool layer with kernel size 2, stride 2.
          - Block 2:
              * Conv layer with kernel_size 3x3, stride 1, no padding, output channels = 64.
              * MaxPool layer with kernel size 2, stride 2.
          - Flatten the features and apply a fully connected layer mapping to output size 10.
        """
        super(myCNN,self).__init__()

        self.conv1 = torch.nn.Conv2d(in_channels=3,out_channels=32,kernel_size=3,stride=1,padding=False)
        self.conv2 = torch.nn.Conv2d(in_channels=32,out_channels=64,kernel_size=3,stride=1,padding=False)
        self.pool = torch.nn.MaxPool2d(stride=2,kernel_size=2)
        self.fc = torch.nn.Linear(6*6*64,10)

    def forward(self,x):

        x = torch.nn.Functional.relu(self.conv1(x))
        x = self.pool(x)
        x = torch.nn.Functional.relu(self.conv2(x))
        x = self.pool(x)
        x = torch.flatten(x,start_dim=1)
        return self.fc(x)
```

In [9]:

```
myModel = myCNN(num_classes=10,input_size=32)
```

2. Loss Function¶ Define a custom loss function that meets the following criteria:

Function Name:

The function must be named LF. Function Arguments:

The function should take three parameters: the model's output (logits), the target labels, and the model itself. Function Behavior:

Compute the standard cross-entropy loss between the output and the target. Add an L2 regularization term calculated as the sum of squared L2 norms of all the model's parameters. The regularization term must be scaled by a factor of 0.1. The function must return a scalar tensor representing the combined loss. Variable Name for the Loss Function:

The loss function must be stored in a variable called LF (i.e. the function name is LF)

In [10]:
```python
def LF(output,target,model):
    """
    Return:
        Tensor: Scalar loss value.
    """
    cross_entropy = torch.nn.Functional.cross_entropy(output,target)
    l2 = 0
    for param in model.parameters():
        l2+=torch.sum(param**2)

    total_loss = cross_entropy + 0.1*l2
    return total_loss
```

Initialize the total_params value with total number of trainable parameters in the notebook in the network¶

In [13]:
```python
myModel
```

Out[13]:
```
myCNN(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1))
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc): Linear(in_features=2304, out_features=10, bias=True)
)
```

In [21]:
```python
import numpy as np

total_params = 0
for params in myModel.parameters():
    total_params+=np.prod(params.shape)
    print(params.shape)
print("Total Trainable parameters: ",total_params)
```

```
torch.Size([32, 3, 3, 3])
torch.Size([32])
torch.Size([64, 32, 3, 3])
torch.Size([64])
torch.Size([10, 2304])
torch.Size([10])
Total Trainable parameters:  42442
```