Project 2: MLP Implementation

In this project a network architecture resembling ResNet-50 was implemented. To fill out the blanks in the code, the table provided was followed, and additionally the equation for output size was used to find the correct padding for each layer.
For a layer with output dimension DIM_2 x DIM_2: $DIM_2 = \frac{DIM_1 - F + 2P}{S} + 1$

#1: Bottle Neck Building Block

When *downsample* is true, the first 1 x 1 convolutional layer in the residual block should have stride 2, as indicated in the given table. Otherwise, stride is 1 – like the other convolutional layers in the residual block.

In order to obtain the correct output size for each residual block padding 1 was added to the 3 x 3 convolutional layer, regardless of down-sampling. Other layers did not need padding.

#2: Class ResNet50 Layer 4

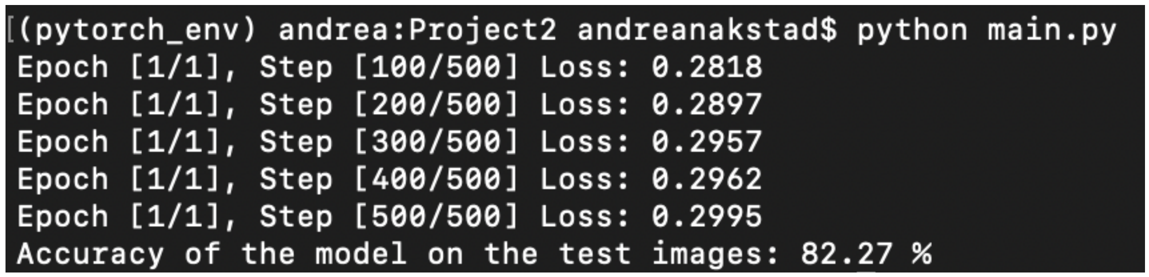
In layer 1 the input image is first halved. Therefore, the output dimension of the first convolutional layer should be 16 x 16. With stride 2 and kernel size 7, padding 3 is needed in order to obtain the correct dimension. The padding for the maxpool layer, padding 1, was also found in the same way.

As for layer 2, 3 and 4 - they were created using *ResidualBlock*. They were simply filled out using the table, and the input channel number was obtained from the previous layer or block. In layer 2 and 3 the last block of each layer also set *downsample* to true in order to reduce the activation map.

The output dimension of layer 4 is 2 x 2. Therefore, in the next layer – the avgpool layer – stride was set to 2 to obtain the correct dimension of 1 x 1.

Lastly, the fully connected layer's purpose is to classify an image based on the results of the previous convolutional layers. Therefore, the input feature number is the number of channels of the previous layer, and the output channel is the number of classes in the Cifar-10 dataset.

Code Result



```
[(pytorch_env) andrea:Project2 andreanakstad$ python main.py
 Epoch [1/1], Step [100/500] Loss: 0.2818
 Epoch [1/1], Step [200/500] Loss: 0.2897
 Epoch [1/1], Step [300/500] Loss: 0.2957
 Epoch [1/1], Step [400/500] Loss: 0.2962
 Epoch [1/1], Step [500/500] Loss: 0.2995
 Accuracy of the model on the test images: 82.27 %
```

Above is the result from running the code, and interestingly the loss increases during the epoch. However, according to *main.*py the test accuracy would be almost 80% for the trained checkpoint parameters after 285 epochs. Therefore, the code seems to be working correctly as the model's accuracy has increased after the epoch from almost 80% to 82.27%.