MoonRiver: Deep Neural Network in C++

Chung-Yi Weng

System

Motivation

- Deep neural network is a black box
- The packages of deep neural work (like Torch, pyTorch, Tensorflow, Caffe, MXNet) are another black boxes
- Understand what happened in these black boxes

Goal

- Implement deep neural network in C++ from scratch, including training and testing, which has the following properties
  - **Independence**: MoonRiver shouldn't have any dependence on any third-party libraries. It should be easily compiled just using standard C++ compilers.
  - **Portability**: MoonRiver should be easily ported on any OSes, including Windows, Linux, and MacOS.
  - **Convenience**: MoonRiver should make users easily build any neural networks they want.
  - **Scalability**: MoonRiver should be easily scaled to build large neural network in minimum effort.

What We Support

Layer

- Convolutional Layer
- Fully Connected Layer
- Max Pooling Layer
- Flatten Layer
- Softmax Layer

Activation

- Linear
- ReLu
- Tanh
- Sigmoid

Optimizer

- SGD
- Momentum
- RMSprop
- Adam

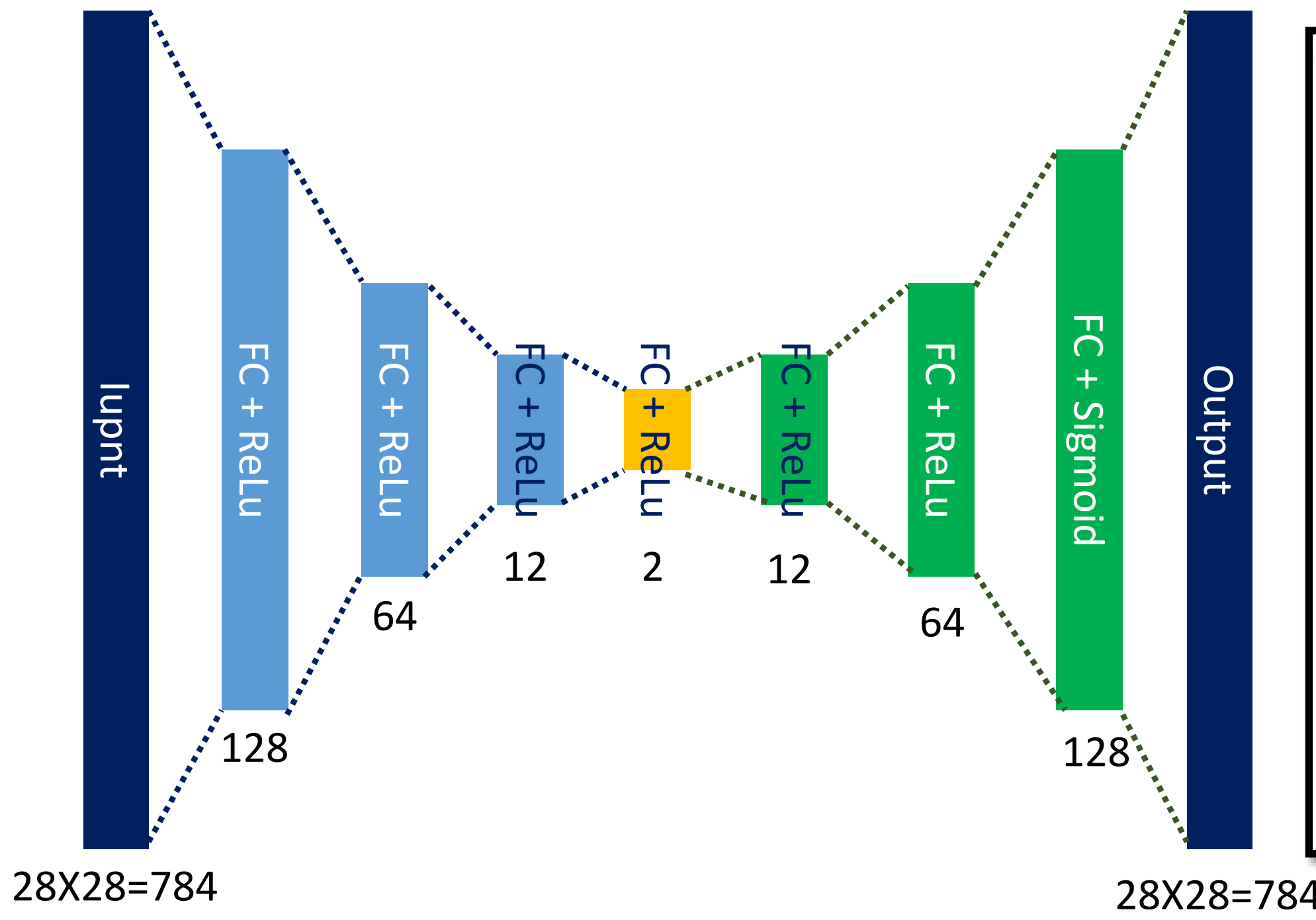
Cost Function

- Mean Square
- Negative Log Likelihood

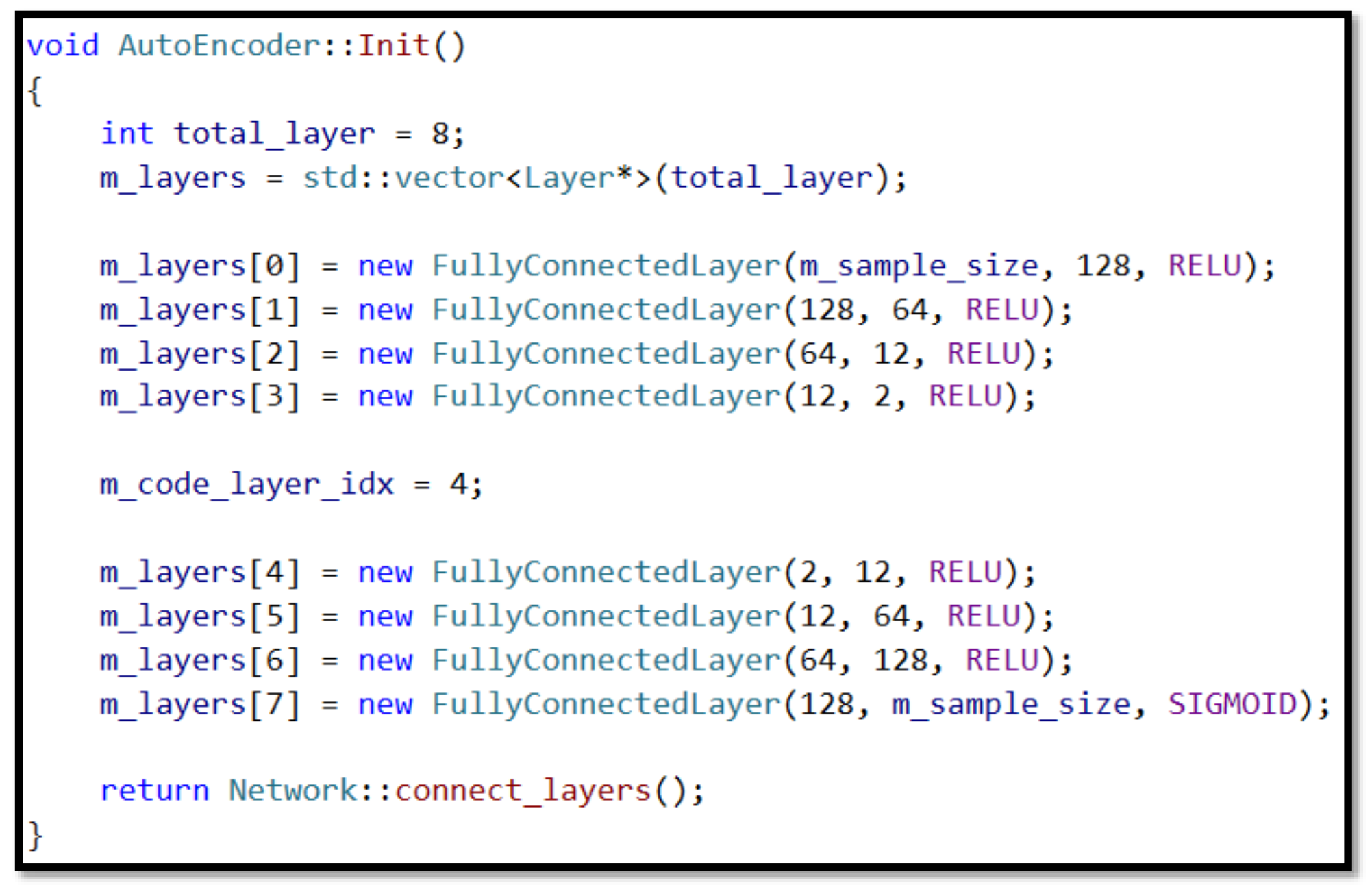
Misc

- MNIST Data Loader
- Mini-batch Random Sampler
- Network Saving / Loading

Auto Encoder

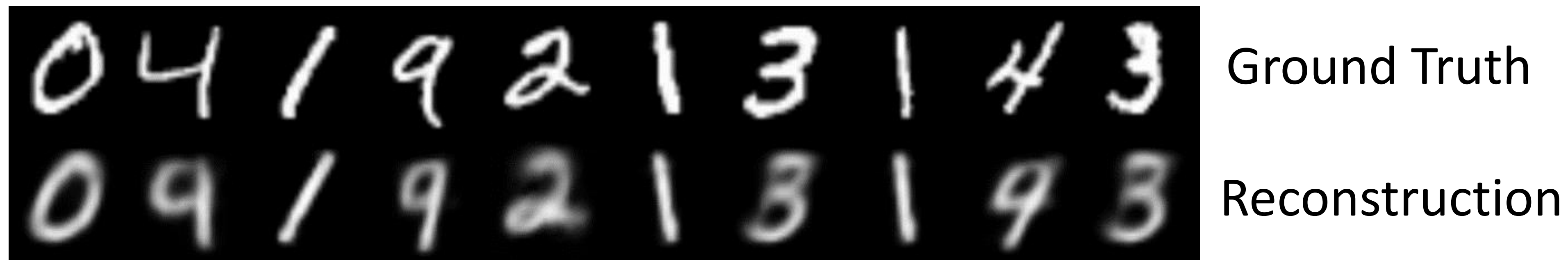


Network Architecture



```
void AutoEncoder::Init()
{
    int total_layer = 8;
    m_layers = std::vector<Layer*>(total_layer);

    m_layers[0] = new FullyConnectedLayer(m_sample_size, 128, RELU);
    m_layers[1] = new FullyConnectedLayer(128, 64, RELU);
    m_layers[2] = new FullyConnectedLayer(64, 12, RELU);
    m_layers[3] = new FullyConnectedLayer(12, 2, RELU);

    m_code_layer_idx = 4;

    m_layers[4] = new FullyConnectedLayer(2, 12, RELU);
    m_layers[5] = new FullyConnectedLayer(12, 64, RELU);
    m_layers[6] = new FullyConnectedLayer(64, 128, RELU);
    m_layers[7] = new FullyConnectedLayer(128, m_sample_size, SIGMOID);

    return Network::connect_layers();
}
```

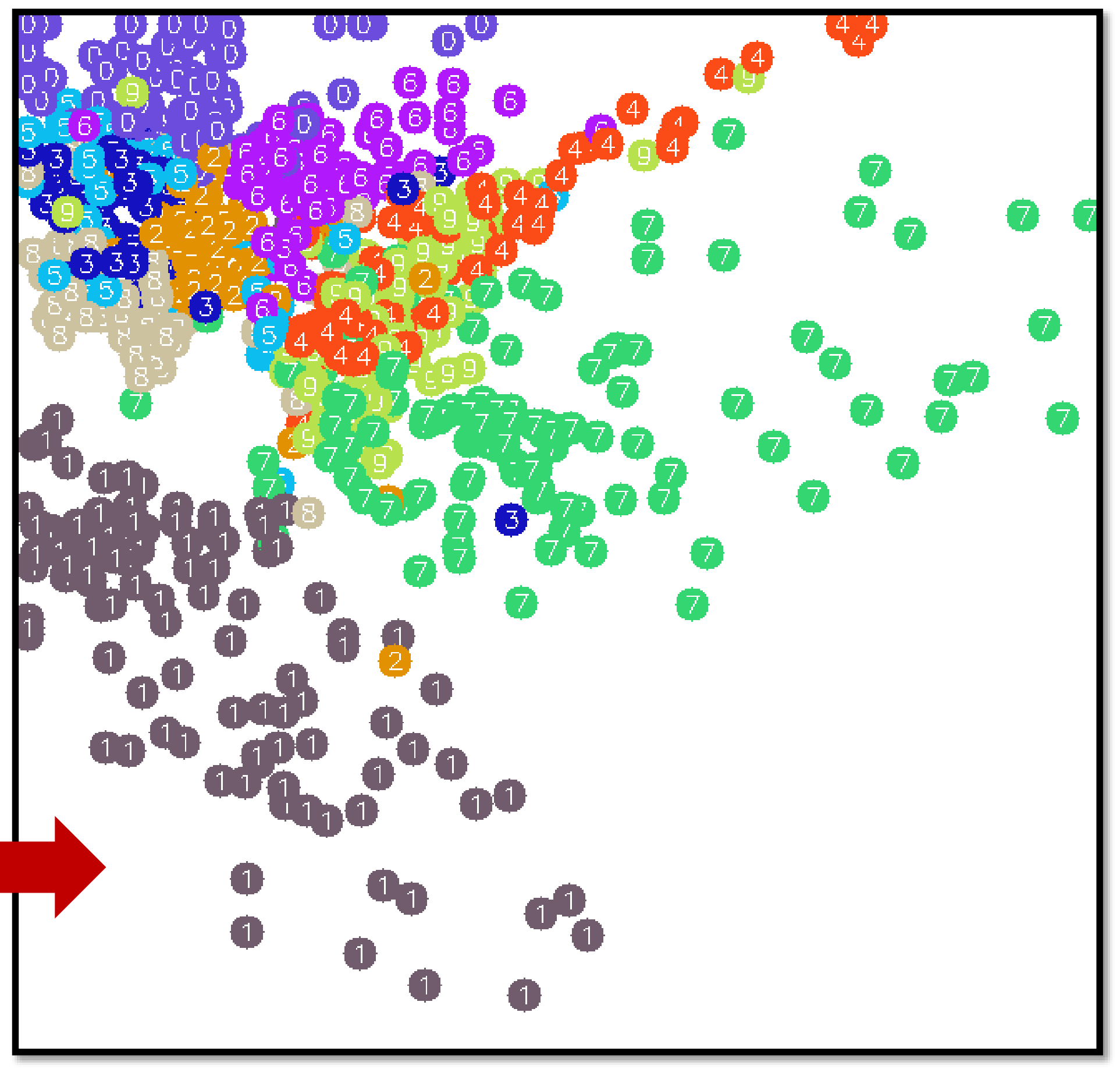
MoonRiver Implementation

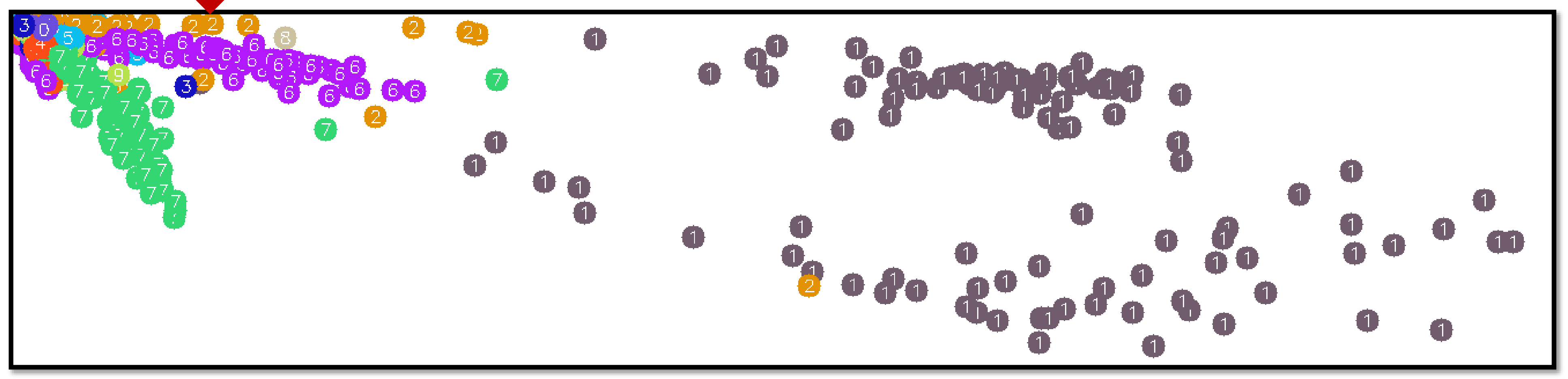


Sample Results

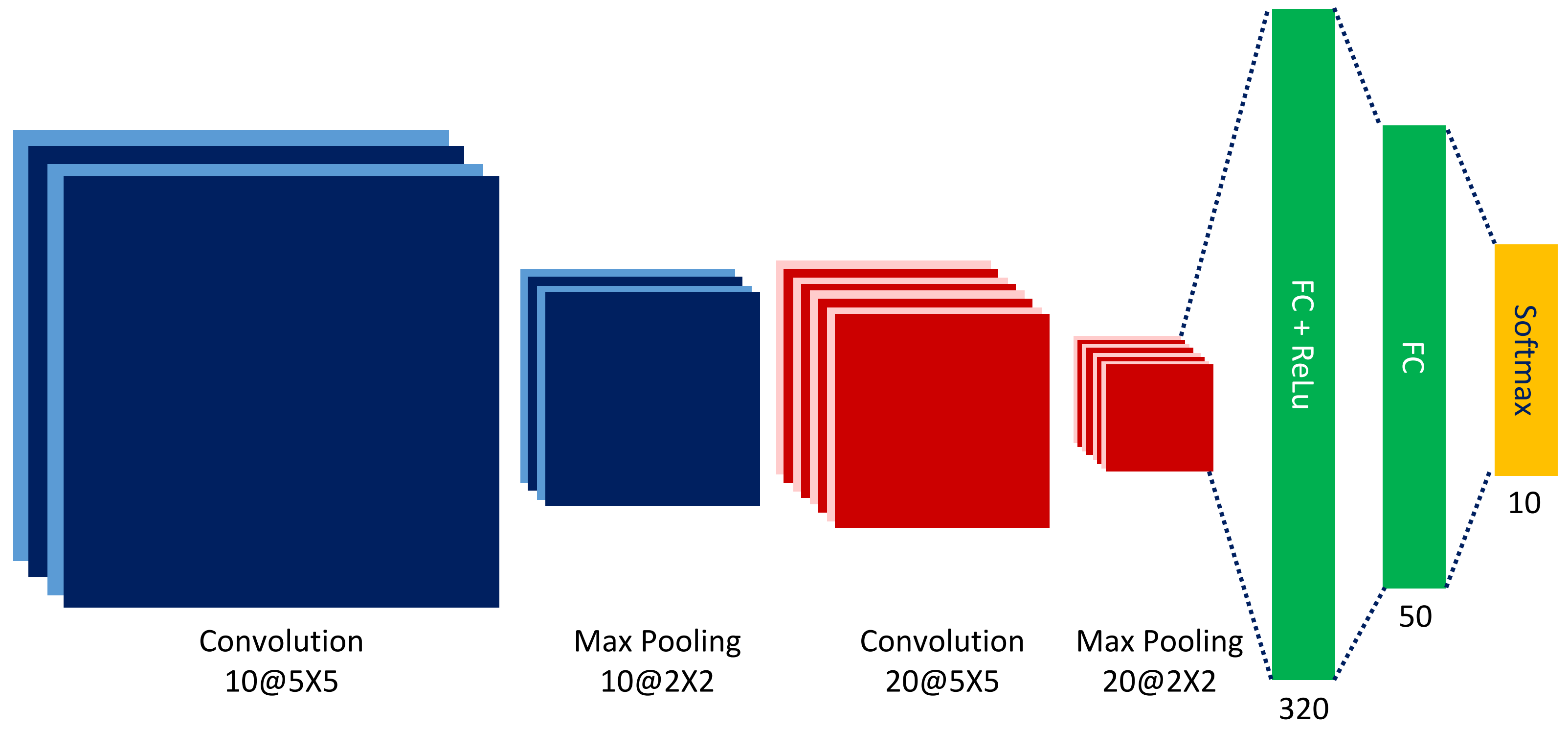
Training Setting

- Training Size
  - 60,000 MNIST training images
- Mini-batch Size
  - 64
- Total Epochs
  - 10
- Optimizer
  - Adam
- Cost Function
  - MSE Loss + Code L2 regularization
  - MSE Loss





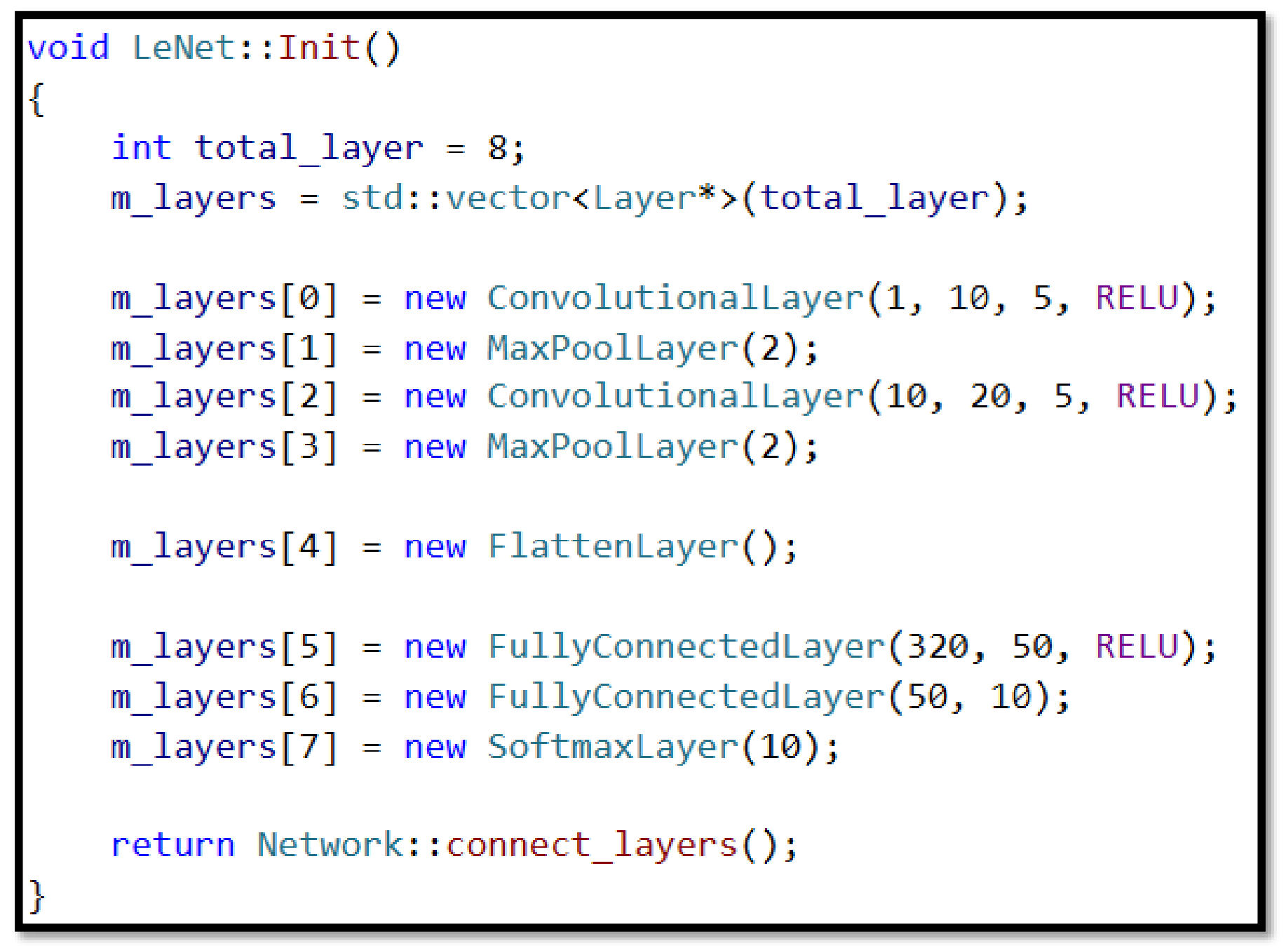
LeNet



Network Architecture

Training Setting

- Training Size
  - 60,000 MNIST images
- Mini-batch Size: 64
- Total Epochs: 10
- Optimizer
  - Momentum
- Cost Function
  - Negative Log Likelihood



```
void LeNet::Init()
{
    int total_layer = 8;
    m_layers = std::vector<Layer*>(total_layer);

    m_layers[0] = new ConvolutionalLayer(1, 10, 5, RELU);
    m_layers[1] = new MaxPoolLayer(2);
    m_layers[2] = new ConvolutionalLayer(10, 20, 5, RELU);
    m_layers[3] = new MaxPoolLayer(2);

    m_layers[4] = new FlattenLayer();

    m_layers[5] = new FullyConnectedLayer(320, 50, RELU);
    m_layers[6] = new FullyConnectedLayer(50, 10);
    m_layers[7] = new SoftmaxLayer(10);

    return Network::connect_layers();
}
```

MoonRiver Implementation

Testing Result

- Testing Size
  - 10,000 MNIST images

Error	Accuracy
0.038	98.97% (9,897/10,000)

Future Work

- Support GPU acceleration
- Support Recurrent Neural Network, like LSTM
- Support GAN
- Convert existed trained network, like AlexNet, VGG-Net, or ResNet, into MoonRiver accepted network format