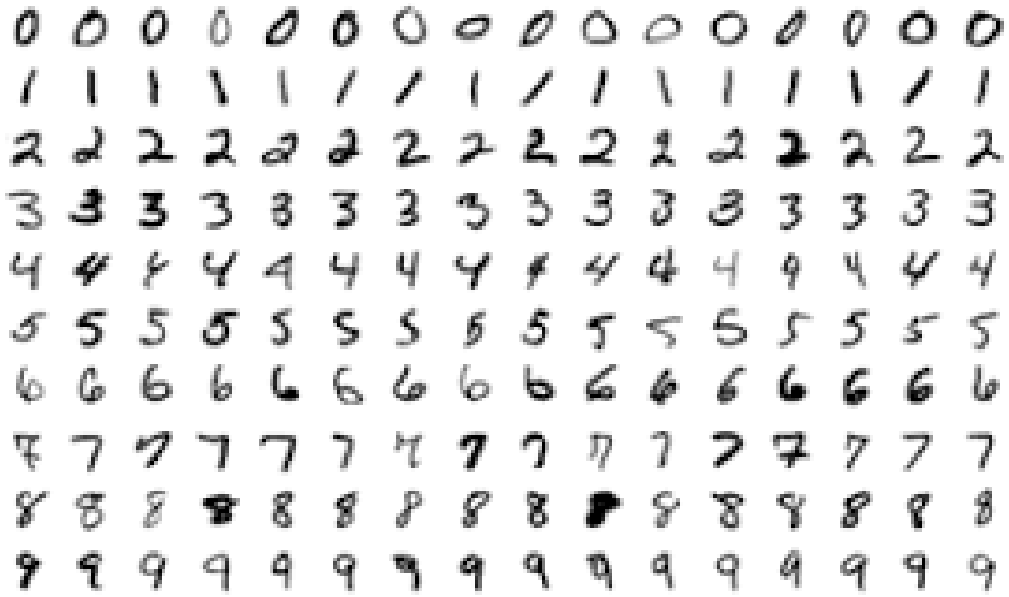
Handwritten Digit Recognition on MNIST Data Using PyTorch

- Pravallika Reddy
(pkethaha@asu.edu)

Dataset Information

- The **MNIST database** is a large database of handwritten digits that is commonly used for training various image processing systems.
- It contains grayscale images of single handwritten digits between 0 and 9. Each of these images is **28 by 28** pixels in size and the goal is to identify what the number is in these images.
- The set consists of a total of **70,000 images**, the training set having 60,000 and the test set has 10,000.
- This means that there are 10 classes of digits, which includes the labels for the numbers 0 to 9.
- To access this dataset the TorchVision package, which comes along with PyTorch, is used.



Problem Statement

- The MNIST Image Classification Problem is a *Multiclass Classification* problem.
- The *training data* is fed to the neural network. The network will then learn to associate images and labels.
- Finally, the network will produce predictions for *testing data* and the degree of accuracy achieved.
- The goal is to correctly identify digits from a dataset of tens of thousands of handwritten images.

Procedure

1. Loading MNIST Dataset from TorchVision
2. Building the Network
3. Training and Testing the Model
4. Evaluating the Network

Loading MNIST Dataset from TorchVision

- Training and Test data sets are downloaded, shuffled and transformed using the DataLoader object in Pytorch.

- The following parameters are specified:
  - **Batch_size**: denotes the number of samples contained in each generated batch
  - **Shuffle**: True, so that the data sequence is randomized.

- A transform is specified to convert the input data set to a **PyTorch tensor**.

- A PyTorch tensor is a specific data type used in PyTorch which is a multi-dimensional matrix.

- PyTorch requires the data set to be transformed into a tensor so it can be consumed in the training and testing of the network.

- Neural networks train better when input data is normalized so that the data ranges from -1 to 1 or 0 to 1.

- **PyTorch Normalize** transform is used and we need to supply the mean and standard deviation of the MNIST dataset, which in this case is 0.1307 and 0.3081 respectively.

Building the Network

Architecture 1: Shallow Neural Network

PyTorch's "torch.nn" module is used to build the network.

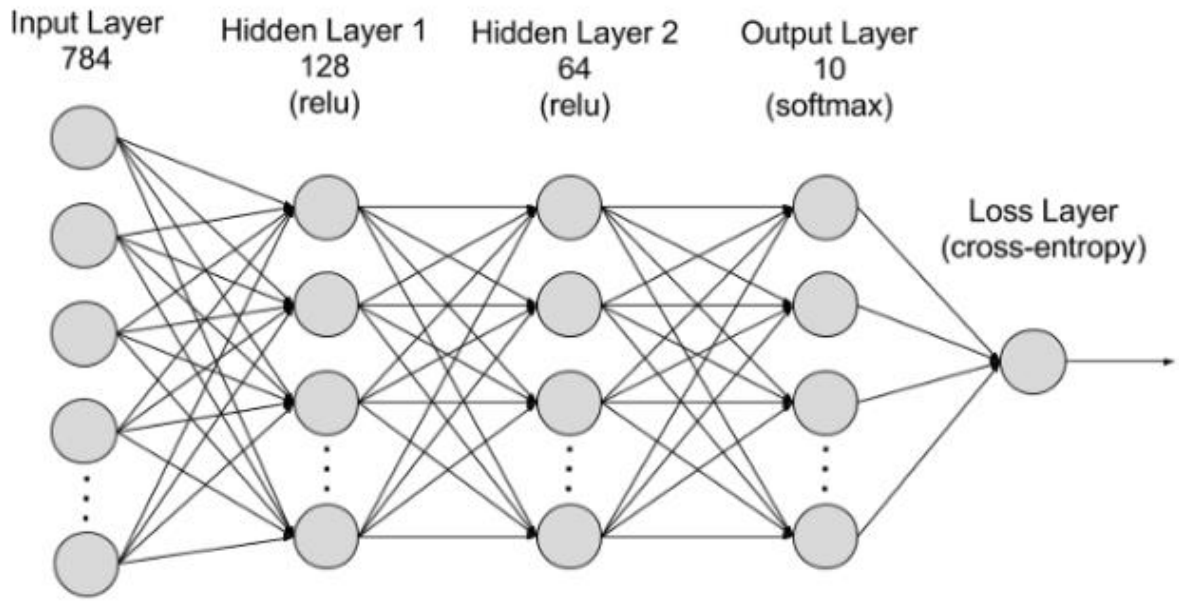
- The neural network contains an input layer (28*28 flattened image), an output layer of 10 neurons and two hidden layers with 128 and 64 neurons in between.

- There are 2 **linear layers** with **ReLU activation** (a simple function which allows positive values to pass through, whereas negative values are modified to zero).

- The output layer is a linear layer with **LogSoftmax** activation because this is a multiclass classification problem.

- The **negative log-likelihood loss** is used to train a classification problem with C classes.

- Together the LogSoftmax() and NLLLoss() acts as the **Cross-entropy loss.**

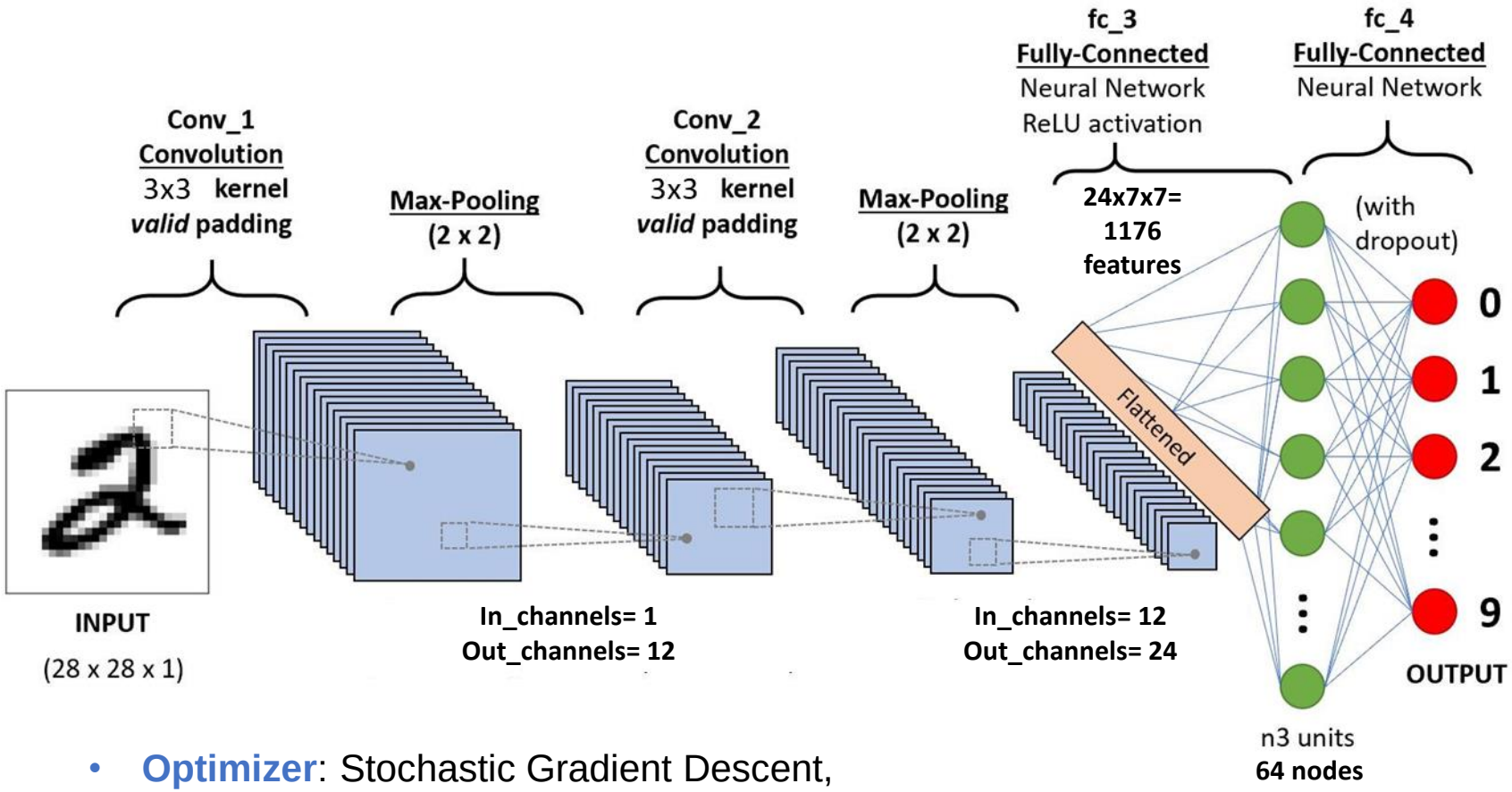


- **Optimizer**: Stochastic Gradient Descent, learning rate: 0.03
- Batch size: 64
- Epochs: 15

Building the Network

Architecture 2: Convolutional Neural Network

- There are two 2d Convolution layers with convolution filter size (3x3).
- Two 2d MaxPool layers with filter size (2x2).
- **ReLu** activation function is used all over and **Log SoftMax** activation for the output.
- Two Fully Connected layers with a dropout probability of 0.2 per cent.
- **Loss**: Cross Entropy Loss

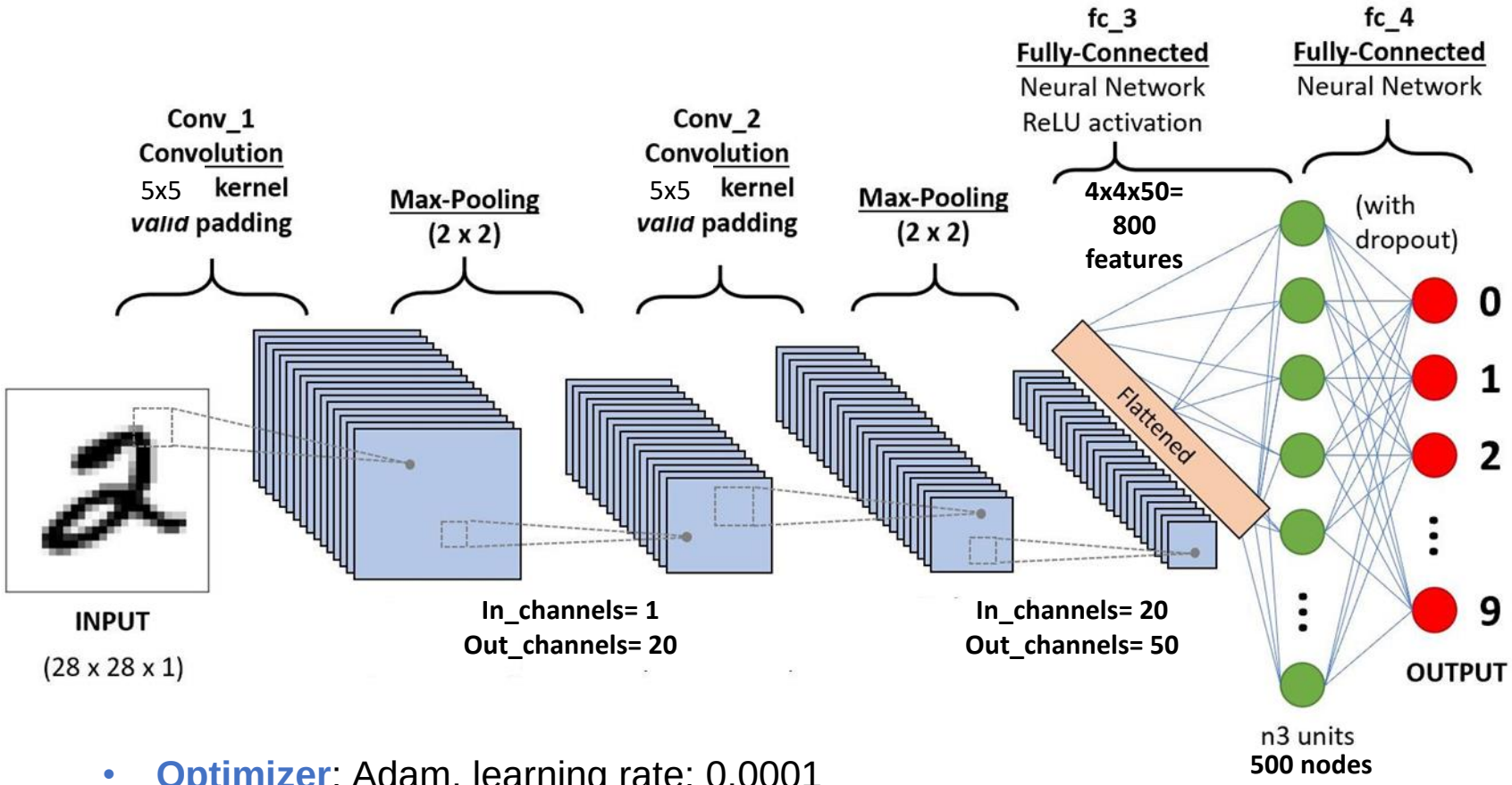


- **Optimizer**: Stochastic Gradient Descent, learning rate: 0.01
- Batch size: 64
- Epochs: 25

Building the Network

Architecture 3: Convolutional Neural Network

- There are two 2d Convolution layers with convolution filter size (5x5).
- Two 2d MaxPool layers with filter size (2x2).
- **ReLu** activation function is used all over and **Log SoftMax** activation for the output.
- Two Fully Connected layers with a dropout probability of 0.5 per cent.
- **Loss**: Cross Entropy Loss



- **Optimizer**: Adam, learning rate: 0.0001
- Batch size: 100
- Epochs: 15

Building the Network

- **Loss function** is used to evaluate how well the network models the dataset.
  - Huge difference between the true values and predictions => then loss function will be high
  - Predictions are pretty good => loss function will be lower.
  - **Cross-entropy** is used as the most common loss function.
  - This function is a measure of the difference between two probability distributions when given a random set of events (the dataset).

- **Optimizer** is used to apply a gradient to the network and to make the network learn.
  - A good optimizer trains the model fast while preventing the model from getting stuck in a local minimum.
  - It is important to have a good learning rate, which is the parameter in an optimization function that determines the step size for each iteration while moving toward a minimum of a loss function.

- **Epoch** refers to one cycle through the full training set.
  - It is essential to find the right epoch size, too small of an epoch can risk the model not learning enough, and too big of an epoch can lead to overfitting.

Convolutional Neural Networks

Convolutional networks are a specialized type of neural networks that use convolution in at least one of their layers

- Fully connected feedforward neural networks are impractical for larger inputs such as high-resolution images. It would require a very high number of neurons due to the large input size of images.

- CNNs perform a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights.

Convolution Layer

- Extracts high-level features such as edges, from the input image.
- Each filter is convolved across the width and height of the input volume, producing stacks of 2D activation maps or feature maps.

Pooling Layer

- Decreases computational power required to process the data through dimensionality reduction.
- Useful for extracting dominant features which are rotational and positional invariant.

Fully Connected Layers

- Connect every neuron in one layer to every neuron in another layer.
- The flattened matrix goes through a fully connected layer to classify the images.

Dropout: The most common method to reduce overfitting is dropout, where we randomly drop input units.

Training and Testing the Model

- The **training** pass consists of four different steps on the training set,

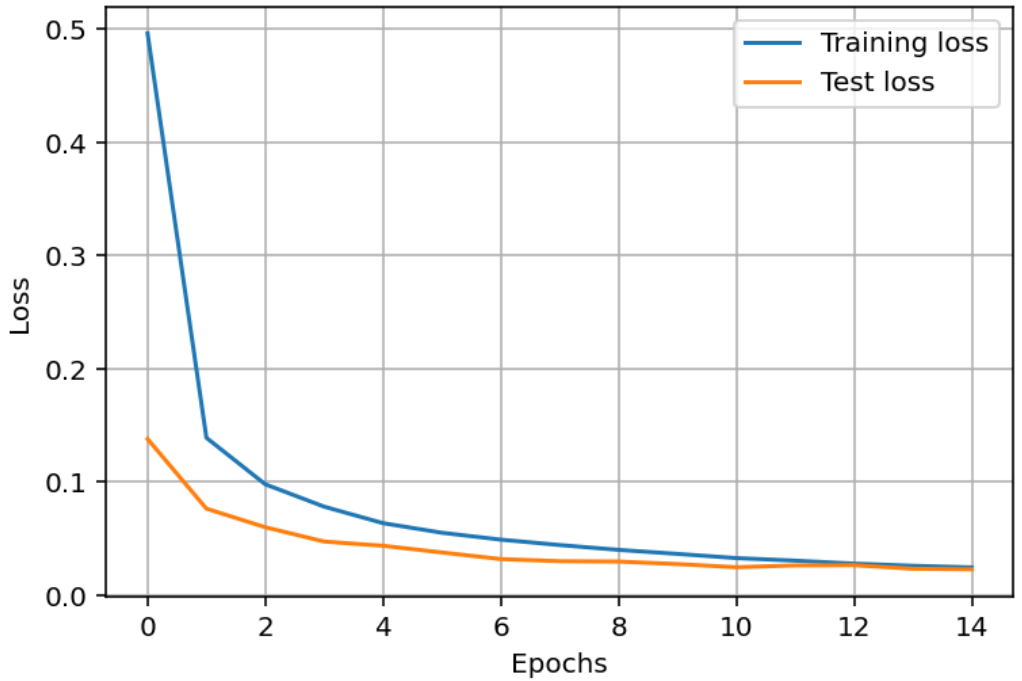
  - Make a forward pass through the network
  - Use the network output to calculate the loss
  - Perform a backward pass through the network with to calculate the gradients
  - Take a step with the optimizer to update the weights

- The model is **tested** to see how it's performing after each epoch, on the test set.

  - Gradients are turned off for validation as it is not needed and saves a lot of memory and computation.
  - To determine the model prediction, for each sample in the batch the maximum value over the 10 output nodes is found. The output node with the highest value will be the prediction of the model.
  - The predictions are compared with the true labels and the number of correct predictions are determined. The accuracy is calculated to determine the performance of the model.

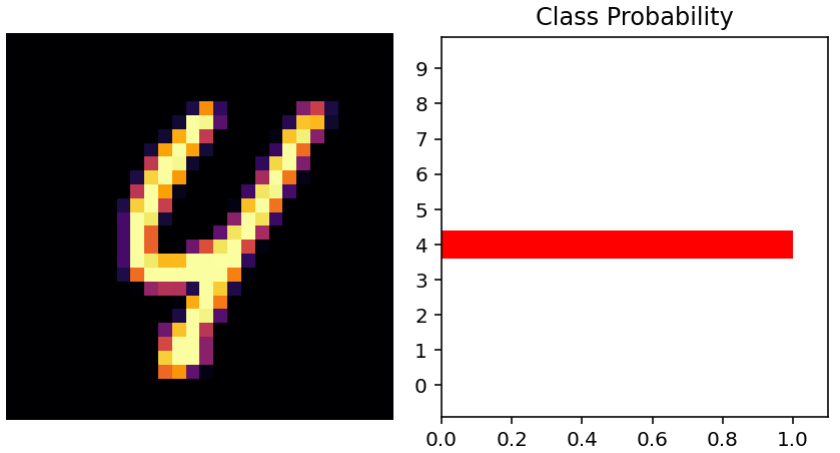
Results

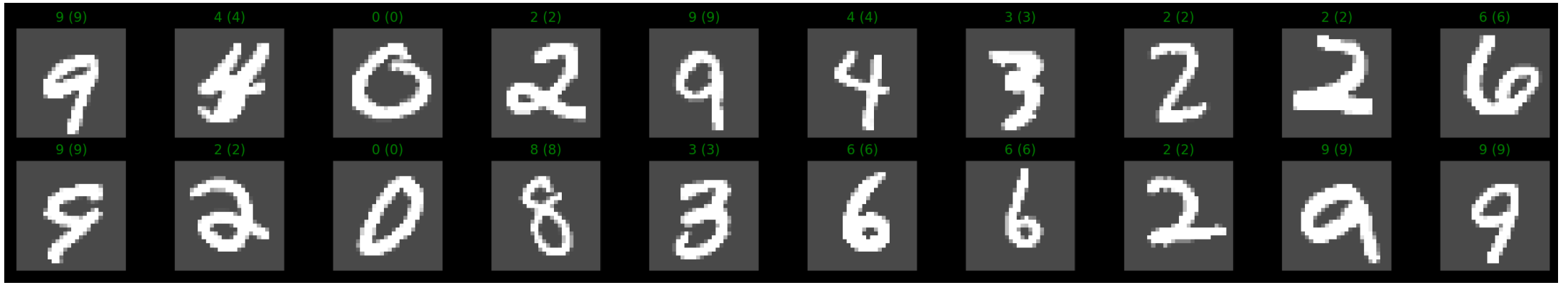
- Three different architectures with different parameters were trained and tested. The obtained accuracies were very high indicating that the model will perform well on classifying new images.
  - Architecture 1: 97.4%
  - Architecture 2: 98.7%
  - Architecture 3: 99.2%
- Visualization
  - The stored training loss and test loss are plotted.
  - The losses decrease with the increase in number of epochs.



Evaluating the Network

- Now that the model is trained, we can simply use the already pre-trained weights to make some new predictions by turning off the gradients.
- The model returns logits which are passed through a SoftMax function and probabilities values are returned as output.
- We can see the input image 4 matches the probability with the highest value in the probability class figure.





The Google Colab Notebooks of the implementations of the three different architectures are given below.

Architecture 1:
https://colab.research.google.com/drive/1Lim2tcdO9mWVZ0aD0WgNM20TzPx7zcra?usp=sharing

Architecture 2:
https://colab.research.google.com/drive/1ZNxVCEJNPPBCICr7nYPwGwevdFEXxdXO?usp=sharing

Architecture 3:
https://colab.research.google.com/drive/1m4gV6T788m2x0xCMM4qM9xsr3S0DR07u?usp=sharing

THANK YOU!