Assignment 3

Introduction

In this assignment, we are going to build *Convolutional Neural Network (CNN)*, and it applies to beginners who like to know how to start building a CNN with *Pytorch*. In this guide we will explain the steps to write code for basic CNN, with link to relevant to topics. The reader sould have basic knowledge of Deep Learning for him to use it.

Requirements

- Python 3
- Pytorch

```
pip install torch torchvision
```

The code for this project is available as a zip archive and includes the following files. In this project, you will implement the forward pass for several basic neural network layers using operators that are similar to those in NumPy. Since we are using PyTorch, the backward pass (back-propagation) will be handled automatically. First, you will implement these layers necessary for building a convolutional neural network (CNN). Then, you will use your implemented layers to train a simple CNN for digit recognition. Finally, you will answer the questions in text.

As in previous projects, this project incudes an autograder for you to grade your solutions on your machine. This can be run on all questions with the command:

```
python autograder.py -q all
```

It can be run for one particular question, such as q2, by:

```
python autograder.py -q q2
```

The code for this project contains the following files:

Files you'll edit:	
layers.py	Some basic neural network layers
Files you might want to look at:	
train.py	Training script for a simple CNN
Supporting files you can ignore:	
autograder.py	

Files to Edit and Submit: You will fill in portions of `layers.py` during the assignment. Once you have completed the assignment, you will submit this file to **HKU Moodle** by uploading a zip file named with your uid, i.e., `A3_{uid}.zip` that archives a `layers.py` and `a pdf file` for the non–coding questions.

Evaluation: Your code will be autograded for technical correctness. Please do not change the names of any provided functions or classes within the code, or you will wreak havoc on the autograder. The objective is to provide a thorough understanding of network layers, focusing on both efficiency and numerical stability. Therefore, only basic operations are allowed, including creation, indexing, slicing, and arithmetic operations, which are very similar to NumPy. To facilitate your assignment, we also allow a few high–level functions: `nn.functional.unfold` and `torch.einsum`. You can find the specific requirements in the `layers.py`. Using any other dependencies beyond this scope will be discounted.

Academic Dishonesty: We will be checking your code against other submissions in the class for logical redundancy. If you copy someone else's code and submit it with minor changes, we will know. If you use GEN AI during the assignment, You must declare with ONE of the following statements if you have used any GenAI tools:

- I did not use any AI technologies in preparing or writing up this assignment.
- I acknowledge the use of <insert AI system(s) and link> to generate initial ideas for background research in the drafting of this assignment.
- I acknowledge the use of <insert AI system(s) and link> to generate materials that were included within my final assignment in its modified form.

If you have used GenAI tool(s), you must (i) name the tool(s), (ii) describe how it/they were used,

AND (iii) reference the tool(s). You are only allowed to ask some background information instead of directly generating code. Please incode the declare in the `layers.py`.

Convolutional Neural Network (80 pts)

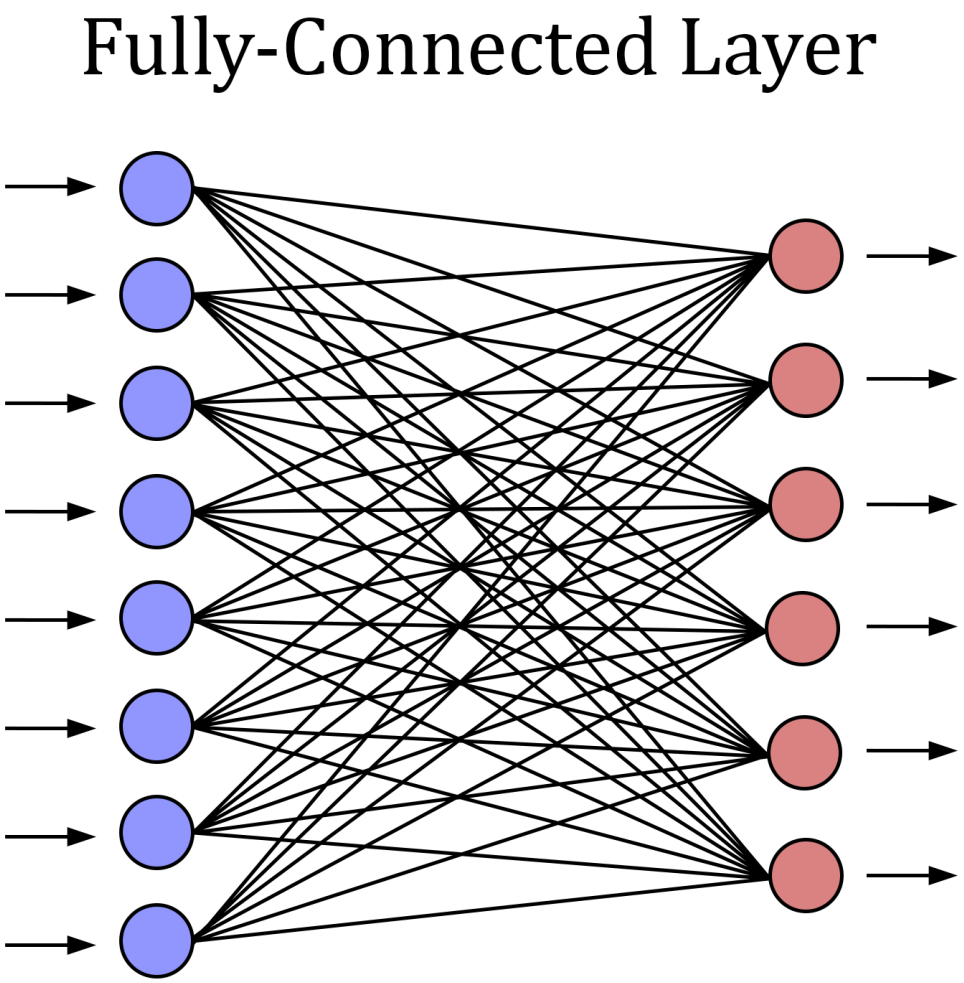
In this section we will briefly go through CNN properties and components. *CNN* is type of Feed–Forward Network which learns to perform tasks like classification, the CNN does it through feature (parameters) optimization. with a given input we will perform a *Forward–pass* (Forward

propagation), calculation and storage of intermediate variables, in forward–pass each layer performs its actions according to its type and the relevant inner variables are stored. the loss will of the network will be calculated according to the criterion, chosen *Loss function*. Then we will perform *Backpropagation*, a process designed to correct the parameters of each layer, using *Grdient Descent Algorithm* in order to find the minimum (local) of the *Loss function*.

Layers

In *Convolutional Neural Network* there are several types of layers, we will discuss the types that are relevant to our SimpleCNN model.

Qeustion 1 (20 points): Fully–Connected Layer



The Fully Connected (FC) layer consists of the weights and biases. Every member of the output layer is connected through the weights to every member of the input layer.

(In the image you can see the input layer in blue, the output layer in red and the arcs that connects them are the weights)

In this manner, every member of the output layer is affected by every member of the input layer according to the corresponding weight.
On top of the linear operation, an activation function will be applied, a non–linear function. In this layer we would like to optimize the weights and the biases.
The formula below shows how to calculate the j–th output:

$$y_j = h\left(\sum_{i=1}^{n} w_{ji}x_i + w_{j0}\right)$$

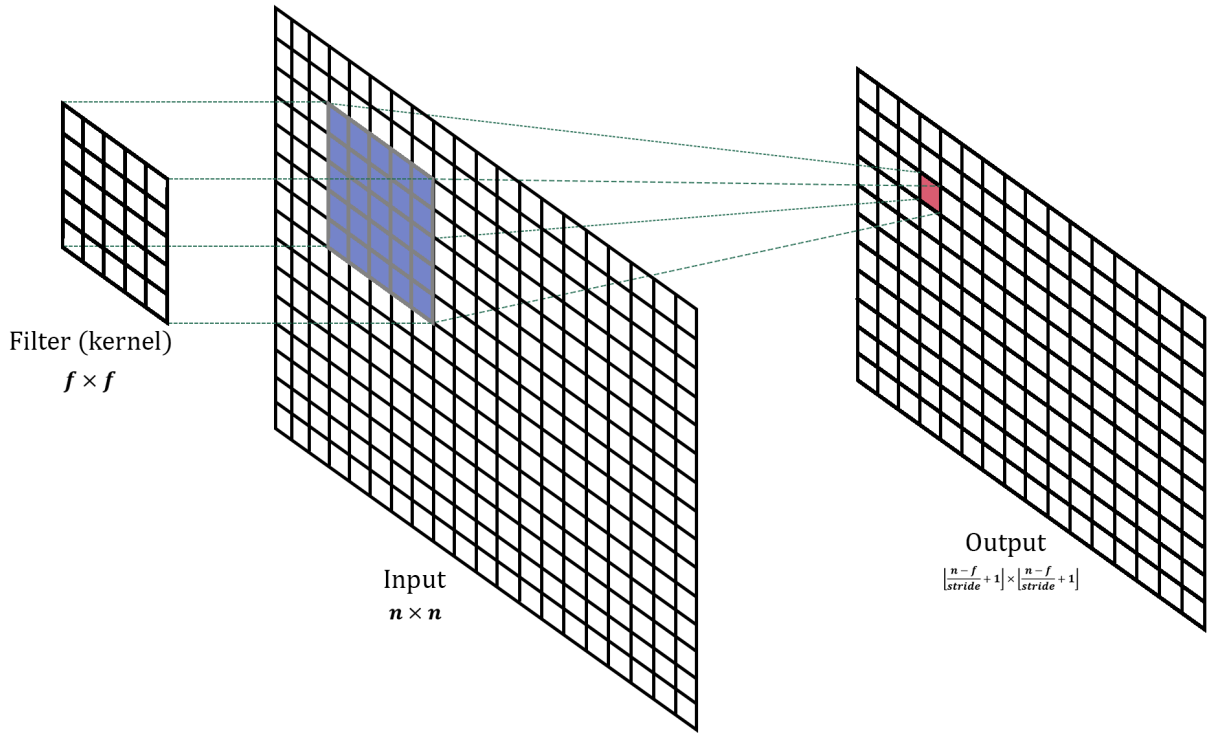
A network consisting solely of *Fully–Connected Layers* is called *Multilayer perceptron*.

You should Implement the `CustomLinear` class in the `layers.py`. To test your implementation, run the autograder:

```
python autograder -q q1
```

Question 2 (20 points): Convolutional Layer

Convolutional Layer



The convolutional layer is considered an essential block of the *CNN*. The convolutional layer performs a dot product between two matrices, where one matrix is the set of learnable parameters otherwise known as a kernel, and the other matrix is portion of the layer's input. The parameters to optimize are the kernels (filters).

In a forward–pass, the filter go through the input in the form of scanning according to its specifications (stride, padding etc.), and for every filter stop a convolution (cross–correlation) operation is performed on the corresponding portion of the input to build one output value (As you can see in the image on the side). In the case of multi–channel input, for every filter all channel goes through the process and combined in the end.

The size of the output can be calculated as follows:

$$H_{out} = \left\lfloor \frac{H_{in} - h_{kernel} + 2 \cdot padding}{stride} \right\rfloor + 1 \; ; \; W_{out} = \left\lfloor \frac{W_{in} - w_{kernel} + 2 \cdot padding}{stride} \right\rfloor + 1$$

The number of output channels is the number of filters in the layer.

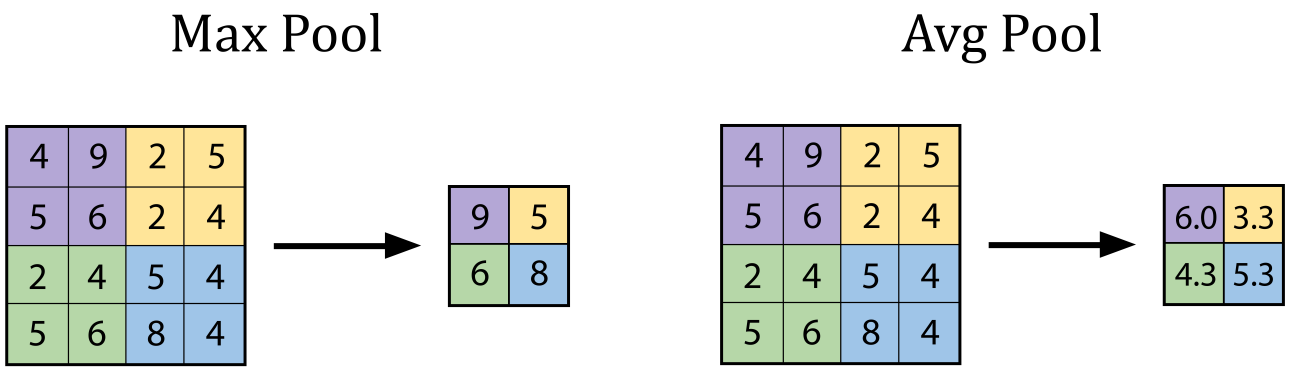
You should Implement the `CustomConv2D` class in the `layers.py`. To test your implementation, run the autograder:

```
python autograder -q q2
```

Question 3 (20 points): Pooling Layer

A pooling layer is used in Convolutional Neural Networks (CNNs) to reduce the spatial dimensions (height and width) of feature maps while preserving the most important information. The pooling layer technique use a kernel that goes though the input and extract one value to the output according to type of the pooling layer (In the figure below you can see example of Max–Pooling and Avg–Pooling). The output size, for every dimension, determined by the input size, kernel size, stride (step size) and padding (if applied):

$$H_{out} = \left\lfloor \frac{H_{in} - h_{kernel} + 2 \cdot padding}{stride} \right\rfloor + 1 \; ; \; W_{out} = \left\lfloor \frac{W_{in} - w_{kernel} + 2 \cdot padding}{stride} \right\rfloor + 1$$



You should Implement the `CustomMaxPool2D` class in the `layers.py`. To test your implementation, run the autograder:

```
python autograder -q q3
```

Question 4 (20 points): Loss & Loss function

The Loss represent the difference between the output of the network and the desired output according to established criterion. In mathematical optimization and decision theory, a *Loss function* is a function that maps an event or values of one or more variables onto a real number intuitively representing some "cost" associated with the even.

Common examples of loss functions:

- Mean Squared Error (MSE) – $'MSE = \frac{1}{N}\sum_{i=0}(y_i - t_i)^2'$
- Mean Absolute Error (L1 Loss) – $'MAE = \frac{\sum_{i=0}|y_i - t_i|}{N}'$
- Mean Bias Error – $'MBE = \frac{\sum_{i=0}(y_i - t_i)}{N}'$
- Hinge (SVM) – $'H_i = \sum_{j \neq y_i} max(0, s_j - s_{y_j} + 1)'$
- Cross–Entropy – $'CE = -\frac{1}{N}\sum_{i=0} y_i * \log t_i'$

You should Implement the `CustomCrossEntropyLoss` class in the `layers.py`. To test your implementation, run the autograder:

```
python autograder -q q4
```

Training Simple CNN (20 pts)

Now is the time to use your layers to build a simple Convolutional Neural Network (CNN) and train a classifier! While this part is not compulsory, it is highly encouraged, as it will help you gain a deeper understanding of neural network training and help you finish the remaining questions.

To get started, run the following command:

Bash

```
python train.py
```

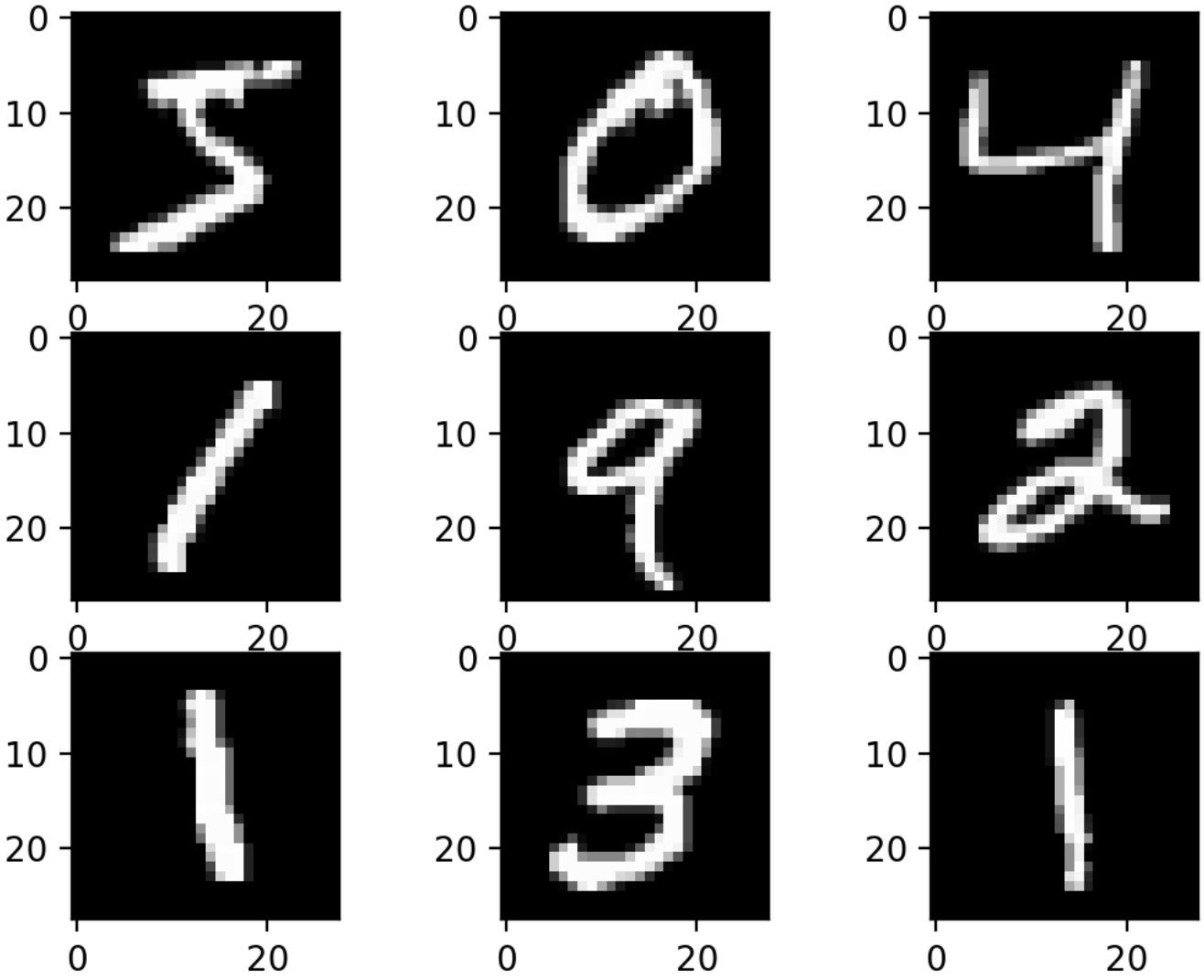
This script will automatically download a dataset and train a simple CNN.

The Database

The network in this guide is a 6 layers network contains: 2 convolution layers, 2 pooling layers and 2 fully–connected layers. The network also applies dropout and batch–normalization methods. For reference the network will be called "Simple CNN".

MNIST Database

This network is trained on MNIST database, a simple gray-scale images of a writen one-digit numbers (0-9), such that the network gets an image and it's target to classify it as the correct number (class).



The MNIST database has 70,000 images, such that the training dataset is 60,000 images and the test dataset is 10,000 images. For more imformation on MNIST Dataset

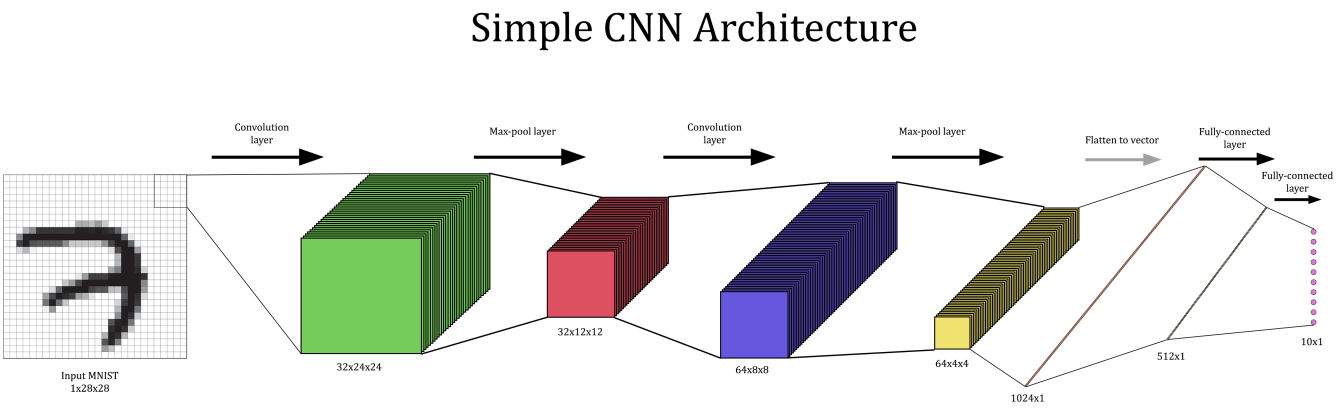
The Model

Simple CNN

Our Network is consist of 6 layers:

1. Convolution Layer with a kernel size of 5x5, and ReLU activation function.
2. Max-pool Layer with a kernel size of 2x2.

3. Convolution Layer with a kernel size of 5x5and ReLU activation function..
4. Max–pool Layer with a kernel size of 2x2.
5. Fully–connected Layer with input layer of 1024 and output layer of 512 and ReLU activation function.
6. Fully–connected Layer with input layer of 512 and output layer of 10 (classes) and Softmax activation function.



The Simple CNN also use methods to accelerate and stablize the convergence of the network training, and avoid overfitting.
After the second layer and fourth layer (Max–pool) the Simple CNN applies *Dropout*, and after the first layer and the third layer (Convolution) it applies *Batch–Normalization*, before the activation.

The Model with pytorch

The Simple CNN is implemented with *pytorch*. In order to implement the network layers and methods pytorch module *torch.nn* is being used. Every Layer/method apart of the fully connected gets an input of 4–dimentions *(N,C,H,W)*, were *N* is the batch size, *C* is the number of the channels and *H,W* are height and width respectively, the resolution of the images.
There are multiple kinds of layers, methods and function that can be used from this module, and for the *Simple CNN* network we used:

- **CustomConv2d –** Applies a 2D convolution over an input signal composed of several input planes.
- **CustomMaxPool2d –** Applies a 2D max pooling over an input signal composed of several input planes.
- **CustomLinear –** Applies a linear transformation to the layer's input, $`y = xA^T + b`$. In that case the input is 2–dimentions, *(N,H)* with the same notations above.

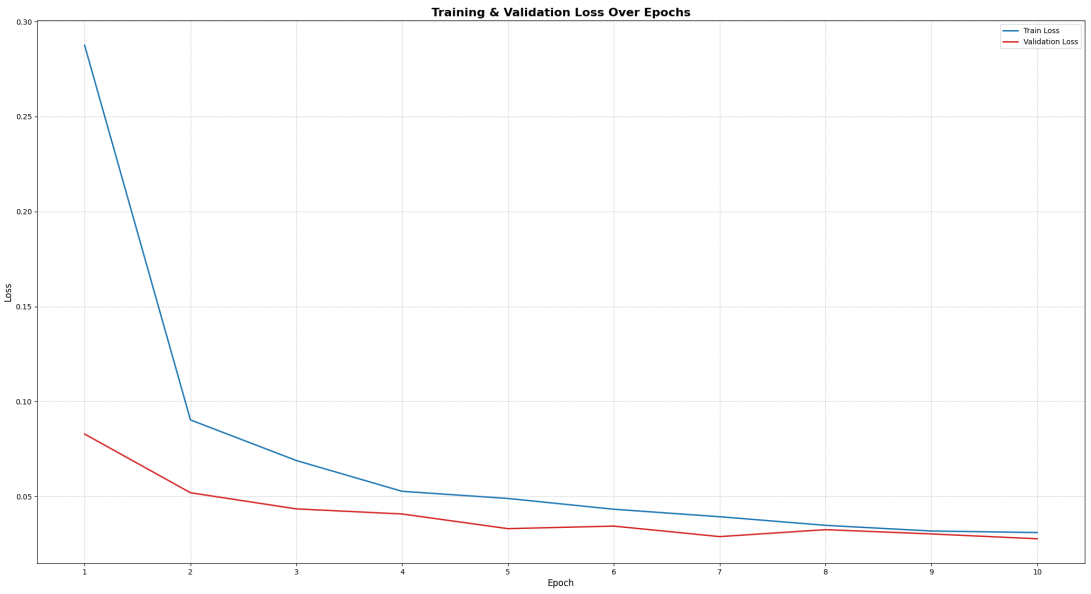
- **Dropout** – During training, randomly zeroes some of the elements of the input tensor with a given probability *p* using samples from a Bernoulli distribution. Each channel will be zeroed out independently on every forward call.
- **BatchNorm2d** – Applies Batch Normalization over a 4D input, sclicing through *N* and computing statistics on *(N,H,W)* slices.

Loss & Optimization

- **Cross Enthropy Loss** – This criterion computes the cross entropy loss between input logits and target. Loss function is a function that maps an event or values of one or more variables onto a real number intuitively representing some "loss" associated with the event. The Cross Enthropy Loss function is commonly used in classification tasks both in traditional ML and deep learning, and it also has its advantages. For more information on Loss function and Cross Enthropy Loss function.
- **Adam optimizer** – The Adam optimization algorithm is an extension to stochastic gradient descent (SGD). Unlike SGD, The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. For more information on Stochastic gradient descent, extensions and variants.

Evaluation

Train & Validation Loss



Typical Run

```
Using cpu

Epoch 1: Train Loss: 0.2876, Train Accuracy: 91.20% | Validation Loss: 0.0828, Validation Accuracy: 97.47%
Epoch 2: Train Loss: 0.0902, Train Accuracy: 97.16% | Validation Loss: 0.0519, Validation Accuracy: 98.42%
Epoch 3: Train Loss: 0.0689, Train Accuracy: 97.81% | Validation Loss: 0.0434, Validation Accuracy: 98.69%
Epoch 4: Train Loss: 0.0526, Train Accuracy: 98.39% | Validation Loss: 0.0407, Validation Accuracy: 98.80%
Epoch 5: Train Loss: 0.0489, Train Accuracy: 98.41% | Validation Loss: 0.0330, Validation Accuracy: 98.98%
Epoch 6: Train Loss: 0.0432, Train Accuracy: 98.62% | Validation Loss: 0.0343, Validation Accuracy: 99.01%
Epoch 7: Train Loss: 0.0393, Train Accuracy: 98.71% | Validation Loss: 0.0288, Validation Accuracy: 99.12%
Epoch 8: Train Loss: 0.0347, Train Accuracy: 98.84% | Validation Loss: 0.0324, Validation Accuracy: 99.06%
Epoch 9: Train Loss: 0.0317, Train Accuracy: 98.99% | Validation Loss: 0.0302, Validation Accuracy: 99.12%
Epoch 10: Train Loss: 0.0309, Train Accuracy: 98.99% | Validation Loss: 0.0276, Validation Accuracy: 99.22%

Test Accuracy: 99.42%

Process finished with exit code 0
```

Question 5 (20 pts)

In this part, answer the following questions:

1. Why is a non–linear activation function (e.g., ReLU) necessary after a fully–connected layer in a CNN? What would happen if we omitted it?
2. What is the primary purpose of a max–pooling layer in a CNN? How does it differ from an average–pooling layer?
3. Why is cross–entropy loss preferred over mean squared error (MSE) for classification tasks?
4. Explain why combining `log(softmax(logits))` directly might lead to numerical instability. How is this addressed in practice?
5. During training, if the training loss decreases but the validation loss increases, what might be happening and how can it be addressed?

References

Simple CNN Guide

The Back Propagation Method for CNN

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

Improving neural networks by preventing co–adaptation of feature detectors