Title of Project

COEN 6331 Project – Submitted to Dr. Kash Khorasani as partial fulfilment of the requirements of COEN 6331, Winter 2023.

Mohsen Asghari
ID: 40257411

Jean-Baptiste Waring
ID: 40054925

***Abstract*—Nowadays, Machine Learning approaches have received large amounts of attention. Not only in the software aspect, in the area of robotics and edge devices, there is a trend of research for light weight approaches. Among the current researches, Hyper-dimensional Computing (HDC) has been considered as a lightweight alternative to the Neural Networks. This approach mathematically proved that it solves classification problems using the advantage of Hyper-Dimensional Vectors (HV) working on high dimensions around 10000. The most problematic part of HDC is the Encoder definition which is responsible for generating quasi-orthogonal HVs representing inputs. In this project, for an object detection problem (image classification), we want to use a/some frozen pre-trained convolutional layer/layers considered as the Encoder of HDC. We will use a pre-trained Convolutional Neural Network (CNN) such as Lenet5 or Resnet-12; then, we cut the first filtering layers and use them as the HDC Encoder.**

***Index Terms*—TO BE ADDED**

I. Introduction

–Classification problem

– whats HDC

In this work, we aim to implement and improve upon the approach presented in the paper [1] for encoding high-dimensional data in the context of a Hyperdimensional Computing (HDC) Classifier. The authors have successfully employed a pre-trained Convolution Neural Network (CNN) as the encoder for an HDC classifier. Our goal is to further optimize this approach by enhancing the performance of a modified LeNet-5 encoder and refining the integration of the HDC classifier. The primary motivation behind using CNNs, specifically the LeNet-5 architecture, is their proven capability in handling spatial data and their effectiveness in extracting meaningful features from raw input.

–main contributions

II. Dataset

The MNIST dataset [2] is a widely-used and well-established benchmark for evaluating the performance of machine learning algorithms, particularly in the field of image classification. Using MNIST as a dataset for evaluating our modified LeNet-5 encoder [3] and HDC classifier offers several advantages:

1) **Standard Benchmark:** The MNIST dataset has become a standard benchmark for evaluating image classification algorithms, providing a reliable and consistent basis for comparison with other methods. By using MNIST, our results can be directly compared with other state-of-the-art approaches and provide a clear understanding of our method's performance.
2) **Simplicity:** The MNIST dataset consists of grayscale images of handwritten digits with a fixed size of 28×28 pixels. The simplicity of the dataset allows for faster training and evaluation of the proposed method, enabling us to focus on optimizing the architecture and hyperparameters.
3) **Wide Applicability:** Despite its simplicity, the MNIST dataset effectively represents a wide range of real-world image classification tasks, such as character recognition and object detection. By achieving good performance on MNIST, our method demonstrates its potential for generalization to other more complex image classification problems.
4) **Availability and Preprocessing:** The MNIST dataset is widely available and has been extensively preprocessed, which facilitates easy access and use. The dataset is already split into training and testing sets, allowing for a straightforward evaluation of our method's performance.
5) **Compatibility with LeNet-5:** The original LeNet-5 architecture was designed specifically for recognizing handwritten digits, making it well-suited for the MNIST dataset. By using the MNIST dataset, we can directly apply the modified LeNet-5 architecture and focus on improving its performance for HDC classification.

By choosing the MNIST dataset, we can efficiently evaluate the performance of our modified LeNet-5 encoder and HDC classifier while ensuring a fair comparison with other methods. The dataset's simplicity and compatibility with the LeNet-5 architecture allow us to focus on optimizing our approach and demonstrating its potential applicability to more complex image classification tasks. Figure 1 displays the first digit of each class (0-9) from the MNIST dataset.

III. CNN Encoder

In this work, we have implemented a modified LeNet-5 architecture as an encoder for the HDC classifier. This section presents the details of the implemented Convolutional Neural Network (CNN) architecture in the form of a `ConvNeuralNet` class using the PyTorch library. Detailed code can be found in Appendix A. The general architecture of LeNet-5 is shown Figure 2.

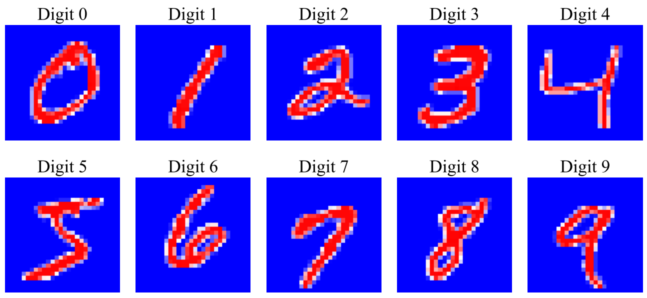

Fig. 1. The first digit of each class (0-9) from the MNIST dataset.

A. Training the CNN

The training loop plays a critical role in the learning process, as it iteratively updates the model parameters to minimize the loss. In this study, we opted for the cross-entropy loss as the cost function and the Adam optimizer to update the model parameters. Below, we explain the rationale behind these choices:

1) **Cross-Entropy Loss**: The cross-entropy loss is a widely used cost function for classification tasks, especially when dealing with multiple classes. It measures the difference between the predicted probability distribution and the true distribution of the target classes. The cross-entropy loss is well-suited for our task because it is sensitive to the differences between the predicted and true probabilities, allowing the model to learn more effectively from misclassified samples. Moreover, this loss function has desirable properties, such as being differentiable, which is essential for gradient-based optimization methods. It is defined as:

$$H(p,q) = -\sum_i p_i \log(q_i) \tag{1}$$

Where $H(p,q)$ represents the cross-entropy loss between the true probability distribution p and the predicted probability distribution q. The sum is taken over all classes i, where p_i is the true probability for class i and q_i is the predicted probability for class i.

2) **Adam Optimizer**: The choice of the optimizer is crucial in determining how the model parameters are updated during the training process. We chose the Adam (Adaptive Moment Estimation) optimizer for this task due to its advantages over other optimization algorithms, such as stochastic gradient descent (SGD). Adam is an adaptive learning rate optimizer that computes individual learning rates for each parameter, which can help speed up convergence and improve model performance. Additionally, Adam is computationally efficient and requires minimal memory, making it well-suited for large-scale problems and deep learning models like our modified LeNet-5.

In summary, the choice of the cross-entropy loss function and the Adam optimizer in the training loop is motivated by their effectiveness and suitability for the classification task at hand. These choices enable the model to learn more efficiently, ultimately resulting in improved performance on the MNIST dataset.

B. Utilizing a Pre-trained CNN as the Encoder

After successfully pre-training a Convolutional Neural Network (CNN) that exhibits high accuracy, we modify its forward function to obtain the output from the penultimate hidden layer. It is hypothesized that this output serves as an effective query vector, given its capability to discern features of the input classes. The architecture of this network is shown Figure 3 Figure 4 demonstrates an example of the input-output relationship. Here, the input has dimensions of 32×32, while the output is 1×400. The output is shown as a 20×20 image.

IV. CNN-HDC

V. Results

VI. Conclusion

References

[1] M. Hersche, G. Karunaratne, G. Cherubini, L. Benini, A. Sebastian, and A. Rahimi, "Constrained few-shot class-incremental learning," 2022.

[2] Y. LeCun, C. Cortes, and C. J. Burges, "Mnist handwritten digit database," *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, vol. 2, 2010.

[3] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

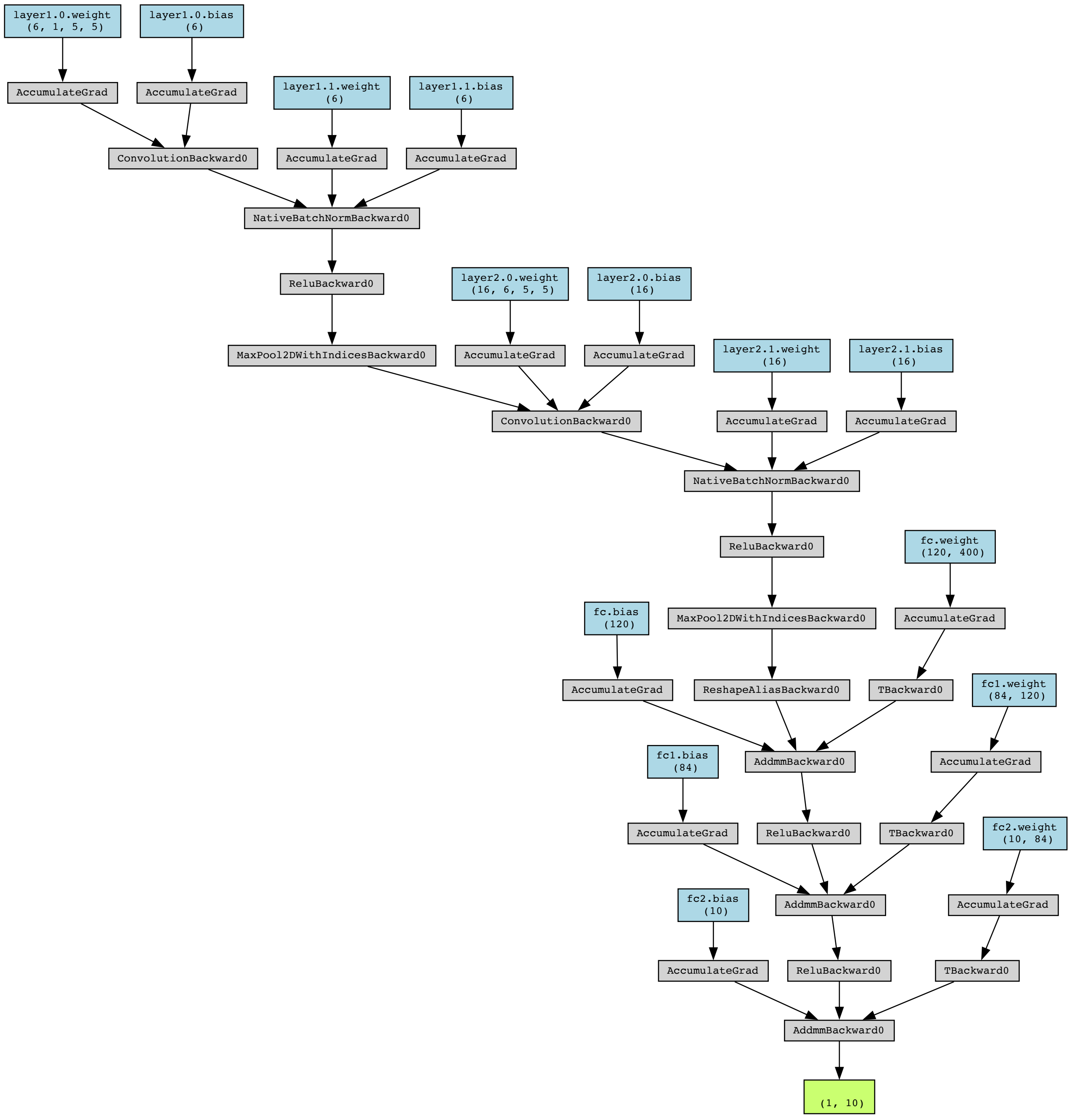

Fig. 2. Architecture of the LeNet5 CNN

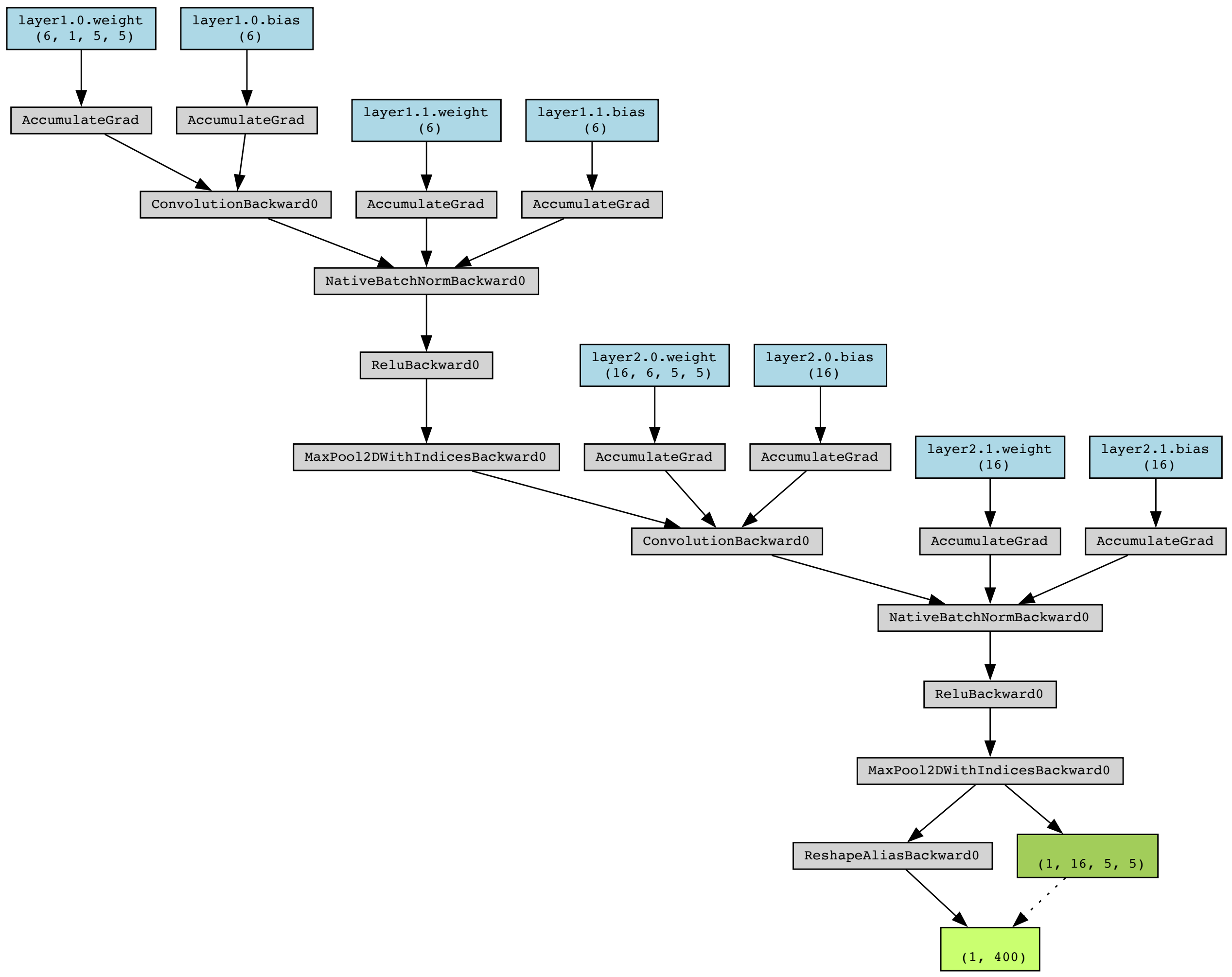

Fig. 3. Architecture of the LeNet5 CNN used as an HDC Encoder.

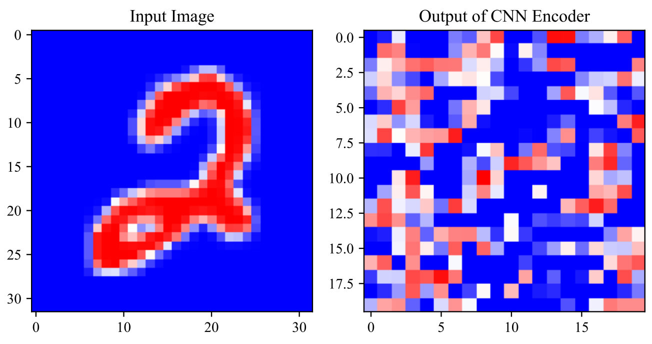

Fig. 4. Sample output from the CNN Encoder for an input of class ”2”.

Appendix A
CNN Encoder Implementation

The CustomLenet5 class is defined as follows:

1) **Initialization:** The class constructor takes three arguments: num_classes, path, and use_big_hidden_layer. The num_classes parameter specifies the number of output classes for the final linear layer. The path parameter, if provided, loads a pre-trained model from the specified path. The use_big_hidden_layer parameter, if set to True, modifies the fully connected layer size to accommodate a larger hidden layer.
2) **Layers:** The architecture consists of two sequential convolutional layers (layer1 and layer2), followed by fully connected layers (fc, fc1, and fc2). Each convolutional layer is followed by Batch Normalization, a ReLU activation function, and a max-pooling layer. The fully connected layers also include ReLU activation functions.
3) **HDC Mode:** The HDCMode attribute is used to control whether the HDC classifier is enabled or disabled. The enableHDC() and disableHDC() methods are provided to set this attribute.
4) **Forward Pass:** The forward() method defines the forward pass of the network. If the HDC mode is enabled, the method returns the output of the last fully connected layer, excluding the final output layer. Otherwise, it returns the output of the final layer.
5) **Forward Pass without Last Layer:** The forwardWithoutLastLayer() method returns the output of the last fully connected layer, excluding the final output layer, regardless of the HDC mode.
6) **Save and Load:** The save() and load() methods are provided for saving and loading the model's state dictionary to/from a specified file path.

The implemented modified LeNet-5 architecture serves as the encoder for our HDC classifier. By utilizing this encoder, we can efficiently extract robust and informative features from the input data, thereby improving the performance of the HDC classifier. The HDC mode enables us to selectively return the output from the last fully connected layer, which is particularly useful when integrating with the HDC classifier.

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


class CustomLenet5(nn.Module):
    def __init__(self, num_classes, path=None, use_big_hidden_layer=False):
        super(CustomLenet5, self).__init__()
        self.USE_BIG_HIDDEN_LAYER = use_big_hidden_layer
        self.layer1 = nn.Sequential(
            nn.Conv2d(1, 6, kernel_size=5, stride=1, padding=0),
            nn.BatchNorm2d(6),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.layer2 = nn.Sequential(
            nn.Conv2d(6, 16, kernel_size=5, stride=1, padding=0),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        if (self.USE_BIG_HIDDEN_LAYER):
            self.fc = nn.Linear(400, 2048)
            self.relu = nn.ReLU()
            self.fc1 = nn.Linear(2048, 84)
        else:
            self.fc = nn.Linear(400, 120)
            self.relu = nn.ReLU()
            self.fc1 = nn.Linear(120, 84)
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(84, num_classes)
        if (path):
            try:
                self.load(path)
            except:
                print("Error loading model from path")
        self.HDCMode = False

    def enableHDC(self):
        self.HDCMode = True

    def disableHDC(self):
        self.HDCMode = False

    def forward(self, x):
        if (self.HDCMode is True):
            return self.forwardWithoutLastLayer(x)
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.reshape(out.size(0), -1)
        out = self.fc(out)
        out = self.relu(out)
        out = self.fc1(out)
        out = self.relu1(out)
        out = self.fc2(out)
        return out

    def forwardWithoutLastLayer(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        if (self.USE_BIG_HIDDEN_LAYER):
            out = out.reshape(out.size(0), -1)
            out = self.fc(out)
            out = torch.sign(out)
            out = (out + 1) / 2  # map -1 to 0 and 1 to 1

        out = out.reshape(out.size(0), -1)
        return out

    def save(self, path):
        torch.save(self.state_dict(), path)

    def load(self, path):
        self.load_state_dict(torch.load(path))
```

Originality Statement

This form sets out the requirements for originality for work submitted by students in the Faculty of Engineering and Computer Science. Submissions such as assignments, lab reports, project reports, computer programs and take-home exams must conform to the requirements stated on this form and to the Academic Code of Conduct. The course outline may stipulate additional requirements for the course.

1) Your submissions must be your own original work. Group submissions must be the original work of the students in the group.
2) Direct quotations must not exceed 5% of the content of a report, must be enclosed in quotation marks, and must be attributed to the source by a numerical reference citation1. Note that engineering reports rarely contain direct quotations.
3) Material paraphrased or taken from a source must be attributed to the source by a numerical reference citation.
4) Text that is inserted from a web site must be enclosed in quotation marks and attributed to the web site by numerical reference citation.
5) Drawings, diagrams, photos, maps or other visual material taken from a source must be attributed to that source by a numerical reference citation.
6) No part of any assignment, lab report or project report submitted for this course can be submitted for any other course.
7) In preparing your submissions, the work of other past or present students cannot be consulted, used, copied, paraphrased or relied upon in any manner whatsoever.
8) Your submissions must consist entirely of your own or your group's ideas, observations, calculations, information and conclusions, except for statements attributed to sources by numerical citation.
9) Your submissions cannot be edited or revised by any other student.
10) For lab reports, the data must be obtained from your own or your lab group's experimental work.
11) For software, the code must be composed by you or by the group submitting the work, except for code that is attributed to its sources by numerical reference.

We hereby certify on our honour that this submission complies with the University's Code of Conduct and Originality Policies.

Mohsen Asghari Jean-Baptiste Waring