Convolution Neural Network with MNIST

Mohit Vijay Agarwal
School of Computer Science
University of Windsor
Windsor, Canada
SID: 110065664

Prem Shanker Mohan
School of Computer Science
University of Windsor
Windsor, Canada
SID: 110036738

Dhara Kishorbhai Hirpara
School of Computer Science
University of Windsor
Windsor, Canada
SID: 110071938

***Abstract*—This project focuses on the classification of the MNIST datasets using a convolutional neural network. The MNIST dataset is a huge dataset with a collection of handwritten digits. It has 10 classes, namely, 0,1,2,3,4,5,6,7,8,9. We study the effects of different optimizers (Stochastic Gradient, Adam and RMS Prop) and loss functions (Categorical Entropy, Mean Squared Error, and Sparse Categorical Cross Entropy)) have on the accuracy of our CNN model. We also compare our model with the Lenet model. After analysis and choosing the best loss and optimization function, our model produced an accuracy of 99.00%, whereas, Lenet model produced a much lower accuracy.**

***Index Terms*—MNIST, Artificial Intelligence, Machine Learning, Neural Networks, Handwritten Digits, Classification**

I. Introduction

Artificial Intelligence has become an integral part of our lives. Each industry is becoming heavily dependent on intelligent machines. Deep learning & Machine learning have advanced exponentially in the past few years. Moreover, many people like to dig their hands into these technologies. To learn Python, you first know how to print "Hello world". To learn deep learning, you learn to classify the famous benchmark dataset "MNIST". The MNIST dataset is the "Hello World" of the machine learning discipline. The MNIST (Modified National Institute of Standards and Technology) [1] dataset is a large dataset of handwritten digits which is used to train different Image processing models. Most of the time, it's used in training & testing in the Machine Learning field. The dataset includes about 70,000 images for training and testing. It's a subset of a larger dataset available from NIST. However, the dataset is working most accurately, it is utilized as the base of practicing and learning how to develop, evaluate and use convolutional deep learning neural networks for image classification from scratch.

The set of images in the MNIST dataset was created in 1998 as the combination of two of NIST's datasets: Special dataset 1 & Special dataset 3. Special dataset 1 contains digits written by high school students, whereas Special dataset 3 contains digits written by employees of the United States Census Bureau. [2] Furthermore, the black and white images from NIST were size-normalized and centered to fit into a 28x28 pixel bounding box and anti-aliased, which presented grayscale levels.

MNIST is one of the benchmark datasets for image classification. Top-performing models use convolutional neural systems that accomplish a classification precision of over 99%, with a blunder rate between 0.6% and 0.2% on the hold-out test dataset.

The task is to classify a given set of handwritten images into one of 10 classes, namely, 0,1,2,3,4,5,6,7,8,9. There is a wide range of applications for this technology, such as postcode, banknotes, and accounting. Before CNN's it was a challenging and labor-intensive process to extract appropriate features from the image dataset.

In this project, we are comparing our model to Lenet-5. Lenet-5 is one of the earliest pre-trained models proposed by Yann LeCun and his colleagues in the year 1998 in the research paper Gradient-Based Learning Applied to Document Recognition [5]. This architecture was used to recognize the machine-printed and handwritten characters. The best aspect of this architecture is that it's straightforward. The model features multiple layers of convolutional neural networks for image classification.

The workflow of the project is as follows. We first use pandas to extract the features and the respective class labels available in the dataset. The next step is to process the data and normalize it. Then we reshape the image so that the CNN model has an easier time taking in the input information and then creating a model around it. We then use the ImageDataGenerator function to generate more data that will help our model to improve its accuracy. Finally, we will use loss functions and optimization functions to train the model.

II. Problem Statement

A. Problem Description

Nowadays, Digital documents take the place of Physical papers. Be that as it may, we still see a parcel of paper documentation in our way of life. Machines can't understand what's written on those physical papers. Converting written by hand characters to advanced characters has been a difficult task within the past and proceeds to be. We cannot effectively process those physical records with computers unless we convert them to digital documents.

The purpose of this research is to create a model having an innate capacity to recognize the handwritten digits from the images by utilizing the concepts of Convolution Neural Network. Though the goal is to make a model which can identify the digits and amplify to letters of an individual's handwriting. The major objective of the proposed framework is to understand Convolutional Neural Network and apply it to handwritten digits.

B. Motivation

In the growing world of Artificial Intelligence & Machine learning, it's simple for humans to recognize any digit. However, it is not the same in the case of machines. Many techniques or methods have developed to make the machines work as a human. Apart from all, The advanced technologies that came up, there's still significant research that needs to be done. For instance, online digits recognition compared to recognition of handwritten digits is more accurate and advanced. Hence, there should be research done in this area to improve the result of handwritten digit recognition which in turn will help in the recognition of handwritten letters and sentences.

C. Justification

Hand-written digit recognition may be a key step toward artificial intelligence and the computer vision field. To test the classification of model, pattern recognition and machine learning communities utilize the issue of manually written digit recognition as a model using the MNIST dataset.

III. Literature Review

"Extraction Method of Handwritten Digit Recognition Tested on the MNIST Database" [3] displays the process of Optical Character Recognition (OCR) using the neural network with an accuracy of over 80%. The descent of the gradient algorithm has been used along with modification of the weights between the neurons to emphasize learning. For the recognition, MLP multilayer perceptron with backpropagation has been used. This neural network is made up of three layers of neurons. The first layer corresponds to the input. To get a more accurate output, the image has been preprocessed to reduce the noise. The second is for extraction, where the image is divided into five characteristic zones for processing namely the North Zone, South Zone, East Zone, West Zone, and the Central Zone, which are identified by dilation of the image. Finally, the last layer is the output layer which consists of 10 neurons for the classification of digits into 10 classes namely, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

"CNN-Based Recognition of Handwritten Digits in MNIST Database" [4] uses a LeNet-5 convolutional network along with the Stochastic Gradient Descent (SGD) training algorithm on the MNIST database to achieve an accuracy of more than 90%. The LeNet-5 model which uses the backpropagation algorithm can recognize patterns directly from pixel images and can also identify extreme changing patterns. The neural network deployed here consists of 3 layers, the input layer with 28x28 neurons, the hidden layer with 100 neurons, utilizing the Sigmoid activation function, and the output layer with 10 neurons representing the classification. To minimize the loss, SGD is used to manipulate the weight of the connection between the neurons. It is said that the accuracy can be improved even more if the training set is larger.

IV. Components

A. Dataset

In this paper, we applied our model to the MNIST (Modified National Institute of Standards and Technology) database. The MNIST database obtained from the NIST's training and the testing database contains 60,000 training images and 10,000 testing images of handwritten digits, of size 28x28 pixels with 256 gray levels. All of the digits have already been size-normalized and preprocessed and formatted (LeCun et al., 1998) [1]

B. Model Configuration

To create the model with the highest accuracy, we had to tune various hyperparameters which helped up fine-tune the model. The hyperparameters are as follows

- **Batch size**: It refers to the number of samples to work through before updating the internal model parameter. In this project, the batch size that gave the most optimum result was 128 in our project
- **Loss function**: This function helps our model to assess the error and optimize the model during the training process. The loss function used in this project was "Categorical Crossentropy Loss Function"
- **Number of classes & epoch**: This project has 10 classes, and we used 20 epochs to achieve desired result
- **Optimizer**: It is the function you use to update the weight in our neural network. In our project, we use Adam optimizer
- **Validation Split**: The training data will be divided into training and test data in the ratio 90:10, meaning, 90% of our data will be used for training the model, and 10% of the data will be used for testing our model

C. Convolutional Neural Network

Convolutional Neural Networks interpret things in the same way that a human brain does, and that is why they are considered the best when it comes to Image Classification. When the network encounters an image, it divides it into small parts and processes each part separately (which is called extraction). CNN extracts features which it learned during the training phase from these subparts.

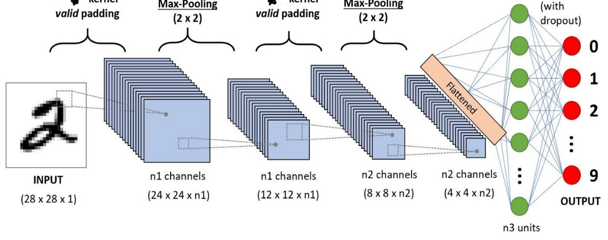


Fig. 1. Convolutional Neural Network

A CNN usually contains various layers, and each layer may contain numerous neurons. Generally, there are three parts the input layer, the hidden layer that will consist of the convolution and pooling layers, and the output layer that will consist of the dense layer. For this project, an image is supplied to the input layer, the convolutional layer then applies a filter of size 3x3. This filter contains weights that are applied over the pixel values of the image following a vertical or a horizontal approach.

During the training process, the convolution layer is applied to all the training images which then generate different pixel values. These values are compared to the pixels of the original image and then the weights are adjusted accordingly so that our output is closer to the original image.

D. Pooling

The main functionality of the convolution layer is to extract features from the image by modifying and updating the weights. However, a major concern over utilizing this feature extraction is that the convolution layers extract features based upon the precise locations of the feature in the image. So, on the off chance that the position of the feature changes indeed a small bit at that point, a diverse image will be created by the convolution layer.

The resolution for this issue is to utilize a pooling layer. This layer is combined after the convolution layer. The pooling layer is similar to the convolution layer as it applies a filter over the original image, which is generally 2x2 pixels with a stride of 2 pixels. A pooling layer will always reduce the measure of the picture depending upon the size of the filter the developer uses. Generally, two sorts of pooling layers are used:

- **Average Pooling**: It Calculates the average of all the pixel values covered by the filter.
- **Max Pooling**: It returns the maximum pixel value from all the pixels covered by the filter.

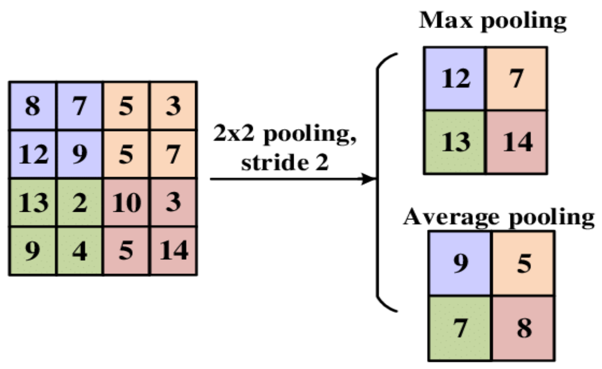


Fig. 2. Pooling layer(Source: [6])

E. Optimizers

Optimizers are algorithms that help minimize the losses by letting us know how to change weights and what is the learning rate of the neural network [7]. The optimizers we have used are mentioned below:

- **Adam Optimizer**: Adam(Adaptive Moment Estimation) is one of the most popular gradient descent optimization algorithms. It is a method that computes adaptive learning rates for each parameter. Adam is described as a combination of the AdaGrad and RMSProp algorithms and can handle sparse gradients on noisy problems [7].
- **SGD Optimizer**: SGD (Stochastic Gradient Descent) updates the model parameters one by one and is a variation of the Gradient Descent algorithm. SGD requires less memory and allows the updated dataset of large datasets. However, it does not always guarantee good convergence [7].
- **RMSProp Optimizer**: RMSProp (Root Mean Square Propagation) is quite similar to SGD and is considered a special variation of AdaGrad. Here, the learning rate is an exponential average of the gradients instead of the cumulative sum of squared gradients [7].

F. Evaluation metrics

Accuracy: Accuracy is the ratio of accurate predictions to total predictions. The accuracy metric is used when one needs to measure the algorithm's performance in an accountable way. It is usually determined after the model parameters and is calculated in the form of a percentage. It is the measure of how accurate the model's prediction is compared to the true data. Below you have the formula for the accuracy.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{1}$$

Loss: A loss function is used when one wants to optimize the algorithm. The loss is calculated on training and validation, which determines how well the model weights updates to create the most accurate model. Loss value indicates how poorly or well the model behaves after each iteration of the optimization process. The different loss functions used in this project are mentioned below:

- **Mean Squared Error**: This loss function is used for regression. Here, the loss is calculated by taking the mean of squared differences between the target and predicted values.

- **Categorical Crossentropy**: This loss function is used when we have the same number of outputs nodes (neurons) as the classes. It is generally used for multi-class classification, but the constraint here is that the final output should be passed through a softmax activation function so that the probability at the output nodes remains between 0-1.

- **Huber Loss**: The Huber loss function comes in between the Mean Squared Error and the Mean Absolute error. It combines both these functions to provide the best result. We can define it using the following equation:

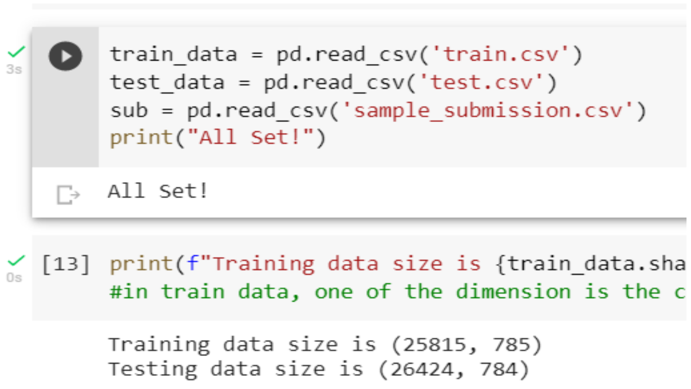
$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \le \delta, \\ \delta\,|y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

As we can see, huber loss is quadratic MSE when the error is small, otherwise it is MAE.

As the final part of our evaluation, the model was examined against the test set. The data required for this was created, and the model's 'evaluate' method in TensorFlow was used to compute the metric values for the trained model. This method returns the loss value and metrics values for the model in test mode.

G. Implementation Details

In this project, we first import the data from the CSV file using Pandas. Our objective in this project is to find the best CNN model which classifies the MNIST dataset with the highest accuracy. The training dataset has 25815 data points and the test dataset has 26424 data points.

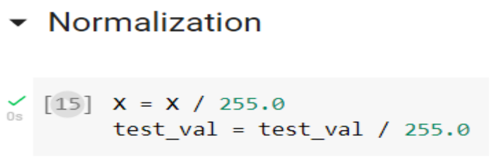


```
train_data = pd.read_csv('train.csv')
test_data = pd.read_csv('test.csv')
sub = pd.read_csv('sample_submission.csv')
print("All Set!")

All Set!
```

```
[13] print(f"Training data size is {train_data.sha
     #in train data, one of the dimension is the c

Training data size is (25815, 785)
Testing data size is (26424, 784)
```

Fig. 3. Importing Data

We then proceed to normalize and reshape the data. The reason we do both of those things is because in the process of training our CNN network, we're going to be multiplying (weights) and adding to (biases) these initial inputs in order to cause activations that we then backpropogate with the gradients to train the model. We'd like in this process for each feature to have a similar range (0-1) so that our gradients don't go out of control (and that we only need one global learning rate multiplier).

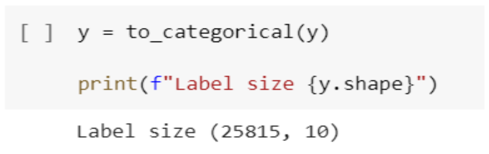


Normalization

```
[15] x = x / 255.0
     test_val = test_val / 255.0
```

Fig. 4. Normalization

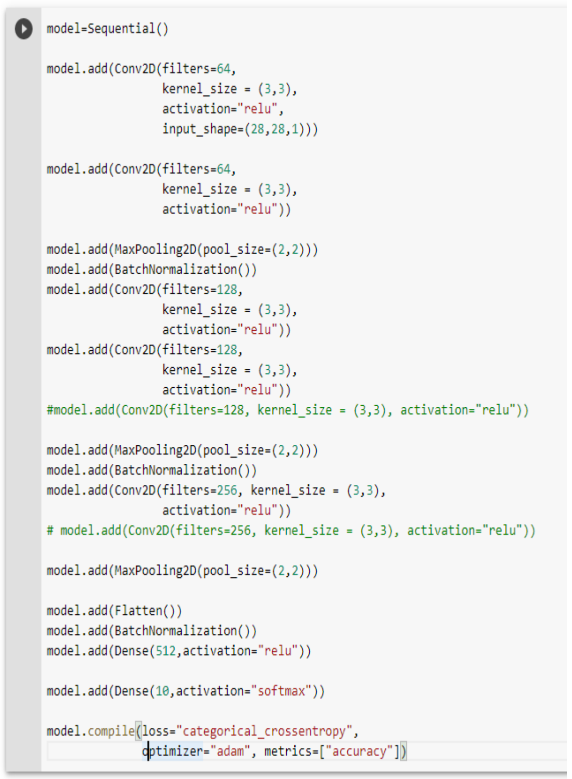
We also on hot encode out label data. The reason for this is for categorical variables where no such ordinal relationship exists. The integer encoding is not enough if one uses this style of encoding. The model will assume there is a natural hierarchy which not only leads to inaccurate results but also poor performance. Therefore, we one hot encode our class labels.



```
[ ] y = to_categorical(y)
    print(f"Label size {y.shape}")

    Label size (25815, 10)
```

Fig. 5. One Hot Encoding

Let's now understand our model configuration. We used the Keras Sequential API, where you add one layer at a time. We start with the input followed by our first convolutional(Conv2D) layer. We chose to set 64 filters for the two-second layers and 128 filters for two-third layers and 256 for the last Conv2D layer. Our CNN model is designed to find and isolate all the useful features from transforming images.



```
model=Sequential()

model.add(Conv2D(filters=64,
                 kernel_size = (3,3),
                 activation="relu",
                 input_shape=(28,28,1)))

model.add(Conv2D(filters=64,
                 kernel_size = (3,3),
                 activation="relu"))

model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Conv2D(filters=128,
                 kernel_size = (3,3),
                 activation="relu"))
model.add(Conv2D(filters=128,
                 kernel_size = (3,3),
                 activation="relu"))
#model.add(Conv2D(filters=128, kernel_size = (3,3), activation="relu"))

model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Conv2D(filters=256, kernel_size = (3,3),
                 activation="relu"))
# model.add(Conv2D(filters=256, kernel_size = (3,3), activation="relu"))

model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())
model.add(BatchNormalization())
model.add(Dense(512,activation="relu"))

model.add(Dense(10,activation="softmax"))

model.compile(loss="categorical_crossentropy",
              optimizer="adam", metrics=["accuracy"])
```

Fig. 6. CNN Model

The next significant layer in the CNN is the pooling (MaxPool2D) layer. This layer acts as a filter that downsamples our images. It observes the 2 neighboring pixels and chooses the maximal value. These kinds of layers are primarily used to reduce the computational cost, and to some extent also reduce overfitting. Combining convolutional and pooling layers, our CNN is able to combine local features and understand more about the global features of our MNIST image. 'relu' is the rectifier (activation function max(0,x)). This rectifier activation function is used to add non-linearity to the network which helps the model to fit better.

The Flatten layer is used to convert the features from final feature maps into one single 1D vector. This flattening step is because you can make use of fully connected layers after some convolutional/max pool layers. It combines all the found local features of the previous convolutional layers. Finally, We used the features in two fully connected (Dense) layers which is Artificial Neural Networks (ANN) classifier. In the last layer(Dense(10,activation="softmax")) we use a softmax function which nets the output distribution of probability of each class.

H. Results and Analysis

Compared to Lenet model which had an accuracy of 93.45%, our model scored an accuracy of 98.50% on the validation dataset after 4 epochs. The image comparing their accuracy is shown below.

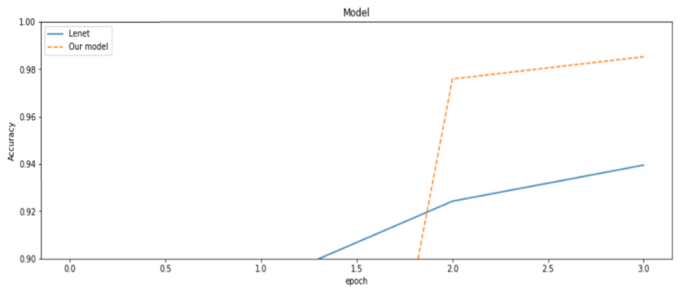


Fig. 7. CNN Model

In our model, we compared and contrasted different loss functions namely categorical crossentropy loss, mean squared loss, and Huber loss. As you can see in Fig. 9, categorical crossentropy loss gave us the highest accuracy among all the losses.

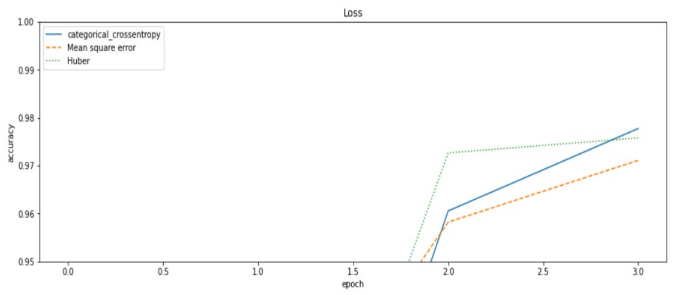


Fig. 8. Graph for Loss Functions

We also compared different optimization functions like Adam, SGD, and RMSprop. Out of all 3, Adam optimizer gave us the highest accuracy.

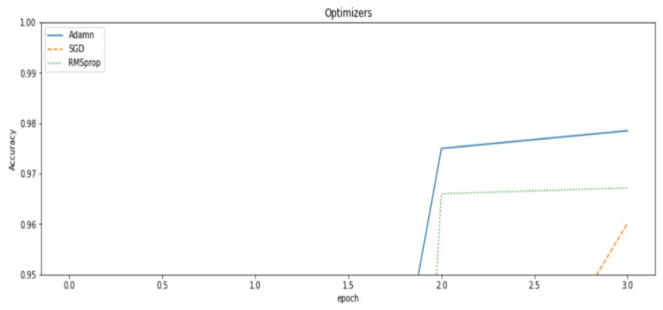


Fig. 9. Graph for Optimizers

Therefore, we used Adam optimizer and categorical crossentropy loss in our model and ran it for 20 epochs. Finally, the model gave us an accuracy of 99.00%. Also, the loss percentage after training was 1.51%

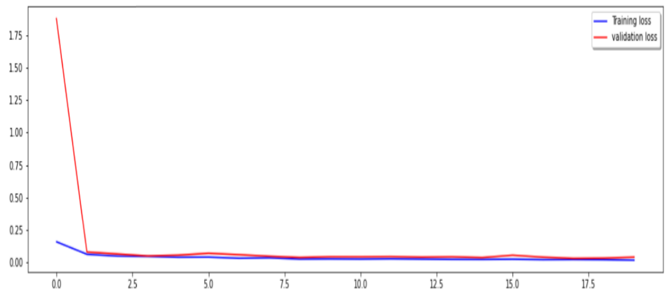


Fig. 10. Loss Curve

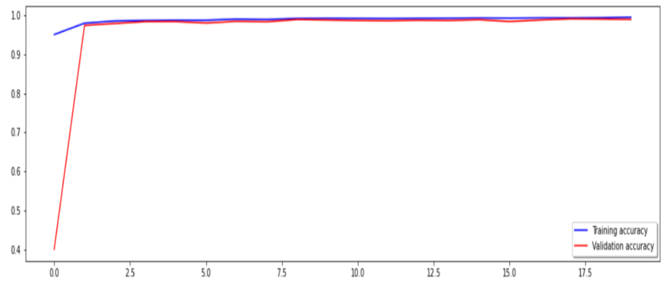


Fig. 11. Accuracy Curve

V. Conclusion

A. Summary

In this research, we discovered how to develop a Convolutional neural layer for Handwritten digits optimization from scratch and correlated various models to get the best result. Specifically, we learned the things defined below:

1) Develop a test model, robust evaluation of a model, and establish a baseline performance for a classification task.
2) Explore extensions to a baseline model to improve learning and result of the model.

3) Develop a finalized model, evaluate the performance, and use it to get an accurate result of images.

The best loss function and optimization function was categorical crossentropy loss and adam respectively. This resulted in our model haveing an accuracy of 99.00% in classifying the MNIST dataset.

Acknowledgement

We're privileged that Dr. Robin Gras gave us a chance to study Convolution Neural Network with MNIST in this tough time and taught us various related topics to this. Without his guidance and proper knowledge about research, it would be more difficult to complete the study.

The project code and final results are accessible on GitHub: shorturl.at/aeqJP

References

[1] "THE MNIST DATABASE of handwritten digits". Yann LeCun, Courant Institute, NYU Corinna Cortes, Google Labs, New York Christopher J.C. Burges, Microsoft Research, Redmond.

[2] LeCun, Yann; Cortez, Corinna; Burges, Christopher C.J. "The MNIST Handwritten Digit Database". Yann LeCun's Website yann.lecun.com. Retrieved 30 April 2020.

[3] El Kessab, B., Daoui, C., Bouikhalene, B., Fakir, M., & Moro, K. (2013). Extraction method of handwritten digit recognition tested on the mnist database. International Journal of Advanced Science & Technology, 50, 99-110.

[4] "CNN-Based Recognition of Handwritten Digits in MNIST Database". Huimin Wu, Research School of Computer Science, The Australia National University (2018).

[5] Yann LeCun, Leon Bottou, Yoshua Bengio, Patrick Haffner "Gradient based Learning Applied to Document Recognition" http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

[6] Deep Neural Networks on Chip - A Survey - Scientific Figure on ResearchGate, Available from: https://www.researchgate.net/figure/Pooling-layer-operation-oproache-1-Pooling-layers-For-the-function-of-decreasing-the_fig4_340812216 [accessed 22 Oct, 2021]

[7] Musstafa. "Optimizers in Deep Learning", Available from: https://medium.com/mlearning-ai/optimizers-in-deep-learning-7bf81fed78a0

[8] George Seif. "Understanding the 3 most common loss functions for Machine Learning Regression", Available from: https://towardsdatascience.com/understanding-the-3-most-common-loss-functions-for-machine-learning-regression-23e0ef3e14d3