The University of Texas at Dallas

Handwritten Digit Recognition using Different Convolutional Neural Network Models in Machine Learning

CS 4375.005 – Introduction to Machine Learning – Spring 2022

1. Pham, Thanh Vu Thien | TVP200000 | tvp200000@utdallas.edu | Software Engineering
2. Pranay Mantramurti | PJM190001 | pjm190001@utdallas.edu | Software Engineering

5-11-2022

Handwritten Digit Recognition using ResNet-18 Convolutional Neural Network Model in Machine Learning

Thanh Pham & Pranay Mantramurti

The University of Texas at Dallas

{tvp200000, pjm190001}@utdallas.edu

Split Task (Thanh: 50% (Report); Pranay: 50% (Code))

Abstract

In the past decade, artificial intelligence in general or deep learning has made remarkable developments. This paper presents an artificial intelligence network model, recognizing handwritten digits using different convolutional neural networks (CNNs). Thereby clarifying the parameter concepts, assessing the importance of the hyperparameters in the model, and presenting the simulation results obtained when using a convolutional neural network to recognize handwritten digit images based on the data set - MNIST (Modified National Institute of Standards and Technology) [1] and applied the CNN network models to the problem of handwritten digit recognition. The performance of the models is evaluated through the accuracy of training, validation, and testing. Hyperparameter tuning is also applied on learning rate to examine the development of the models in training.

Keywords: machine learning, CNN, convolutional neural networks, image classification, MNIST, ResNet-18, MLP, LeNet.

1. Introduction

It can be said that, with the satisfaction of all three factors: large enough databases, strong supporting hardware and advanced algorithms, artificial intelligence (Artificial Intelligent - AI) has created a new technology movement in the current digital era. In particular, the collection of information from the huge image data system worldwide is an area of interest and research by many scientists worldwide. This is an opportunity and also a leading challenge, the application of artificial intelligence in general or deep learning techniques (DL) in particular is a competitive field. increase processing speed, ability to extract and collect information from the aforementioned data source for different uses. Image recognition is one of the subproblems in the field of Computer Vision, which helps computers understand the content of images. The manipulation of image recognition with humans is really simple, but with computers, it is really complicated. The computer will have to look at all the pixels, and from those pixels give the output, it is almost impossible to define predefined rules for effective recognition.

The MNIST database (abbreviated from Modified National Institute of Standards and Technology database) is a large database containing handwritten digits commonly used in training image processing systems. This database is also widely used for training and testing in the field of machine learning. The database was created by "blending" samples from the original NIST dataset. The database creators (LeCun and Cortes) felt that because the NIST training dataset was obtained from the US Census Bureau, while the test dataset was obtained from US high school students, hence it is not suitable for machine learning experiments. Furthermore, the black and white images from NIST were normalized to fit the 28×28 pixel bounding box and spatial anti-aliasing with the introduction of grayscale levels.

The report presents different Convolutional Neural Network (CNN) such as Multi-Layer Perceptron (MLP), ResNet-18, LeNet, and their applicability on the problem of object image recognition using convolutional neural networks. Within the scope of the report, we will try to clarify the CNN models and evaluate the function blocks, the impact of the parameters on the image recognition results. The specific object here is the MNIST handwritten digits database [1]. Finally, the paper also mentions the usability of building models for digit recognition.Y

1.1. Deep Learning Technique

Deep Learning (DL) is a sub-branch of Machine Learning or machine learning that allows computers to train themselves to perform learning from large amounts of data provided to solve specific problems.

The research and evaluation process will be carried out on the MNIST common handwritten digits dataset by the supervised learning method. That is, the input

data has been predefined, the learning will perform prediction and compare the output results with the input, then the network model will perform "learning" and adjust the network parameters. to fit the input data set.

In this paper, we choose the convolutional neural network model instead of the traditional multi-layer perceptron (MLP) network model because of the advantages of spatial partitioning analysis, which is an outstanding advantage when evaluating data sets. multidimensional data, in this case, image data.

1.2. Convolutional Neural Networks (CNNs)

1.2.1. ResNet-18 Convolutional Neural Network

In traditional MLP networks, each neuron in the front layer will be connected to all the neurons in the back layer, which causes the computational load in the network to increase sharply as the depth of the model increases (increasing the number of layers) for the model. The advent of CNN has helped to solve this problem by using local receptive regions, shared weights, and convolution methods to extract information instead of classical methods. Our team applied ResNet-18 CNN in PyTorch for this project.

He et al. 2015 coined the term ResNet, which stands for residual networks. The ResNet18 architecture has 72 layers and 18 deep layers. This network's architecture was designed to allow for the efficient operation of a large number of convolutional layers. However, adding multiple deep layers to a network frequently results in a degradation of output. This is known as the vanishing gradient problem, and it occurs when neural networks are trained using back propagation and rely on gradient descent to find the minimizing weights. Because of the presence of multiple layers, repeated multiplication causes the gradient to become smaller and smaller, eventually "vanishing," resulting in network saturation or degraded performance. [3]

ResNet has a 3×3 convolutional layer design as shown in Figure 1. The residual block has two 3×3 convolution stages with the same number of output channels. Each convolutional layer is followed by a batch normalized layer and a ReLU activation function. We feed the input through the residual block (Figure 2) and then add it to itself before the final ReLU activation function. This design requires that the output of the two convolutional stages be the same size as the input, in order to be added together. If it is desired to change the number of channels or strides in the residual block, an additional 1×1 convolution layer is needed to resize the input correspondingly in the outer branch. There are 4 convolutional layers in each module (excluding the 1×1 convolutional layer). Adding the first convolution layer and the last fully connected layer, the model has a total of 18 layers. Therefore, this model is often referred to as ResNet-18. The number of channels and residual blocks in the module can be changed to create different ResNet models, for example, ResNet-152 is 152-layer model. ResNet's structure is simple and easier to modify. All these factors led to the rapid and widespread popularity of ResNet.

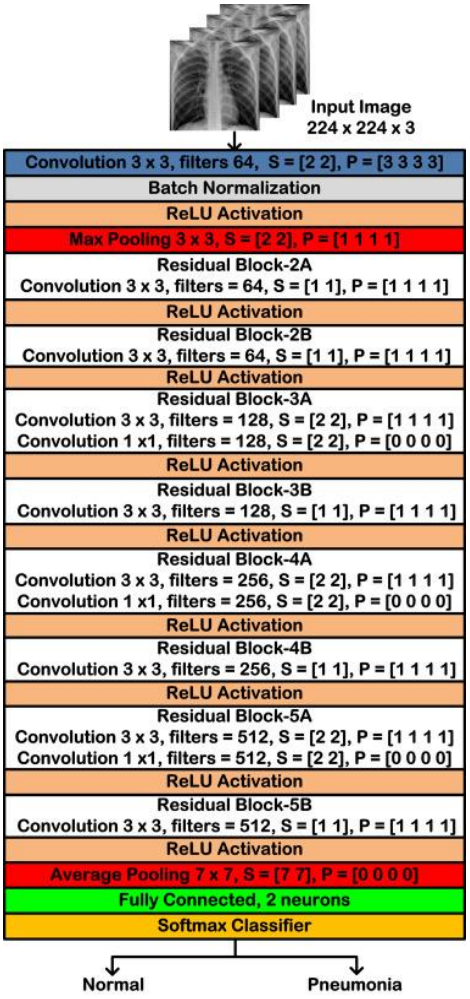

Figure 1. ResNet-18 Layer Architecture [3]

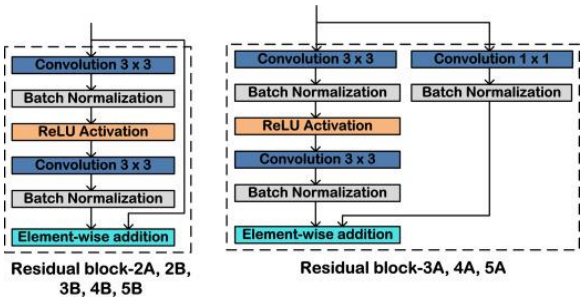

Figure 2. A typical residual block used in ResNet-18 CNN model [3]

1.2.2. Convolutional Layer

The convolution layer is a core component of the convolutional neural network, used to extract feature map information to support the learning process of the CNN. The method of operation of this layer is implemented through the process of sliding and convolution of the filter (filter/kernel) over the entire image. The output is the characteristic of the image corresponding to the filter used, with the more filters used, the more features of the corresponding image will be obtained. In the scope of this report, we will consider the parameters that affect the convolutional layer, including filter size, slip step (stride)

1.2.3. Pooling Layer

The pooling layer has the effect of reducing the size of the image data, thereby helping the network to learn more general information, and this process reduces the number of parameters in the network.

The commonly used pooling methods are Max Pooling and Average Pooling.

1.2.4. Dropout Layer

The dropout layer is a technique used to limit the phenomenon of overfitting (the phenomenon of a neural network that is too closely attached to the training data set and does not respond to new data sets), which is common in CNN and helps the model to calculate faster. Dropout uses the method of removing a random number of neurons in the network with a given probability by setting all the weights of those neurons to 0, meaning that the connections to that neuron are zero. valid, then the model will have to try to get the correct recognition while missing information from the discarded neurons. This will help increase the recognition rate of the model but is not too dependent on the training data.

1.2.5. Fully-connected Layer (FC)

The input to the fully connected layer is the output from the pooling layer or the final convolution layer, it is flattened and then fed into the fully connected layer for forwarding. The FC layer is responsible for synthesizing information to give the decision layer (output) to give the most accurate results.

2. Related Work

2.1. 2D CNN Model Using Keras with Tensorflow

In Sambit Mahapatra's CNN model [7], a simple 2D CNN using Keras with a Tensorflow backend for the MNIST digit recognition task was applied. The dataset includes 60,000 training images and 10,000 testing images, both of which are 28x28 in size. The architecture used here consists of two convolution layers, followed by a pooling layer, a fully connected layer, and a softmax layer as shown in Figure 3 below.

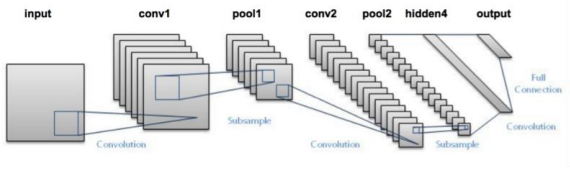

Figure 3. 2D CNN Architecture [7]

For different types of feature extraction, each convolution layer employs multiple filters. Test accuracy of 99 percent indicates that the model has been well trained for prediction. According to the author, if the entire training log is visualized, it can be seen that as the number of epochs increased, the model's loss and accuracy on training and testing data converged, resulting in a stable model. Figure 4 shows the graph of model accuracy and model loss comparing training data and test data.

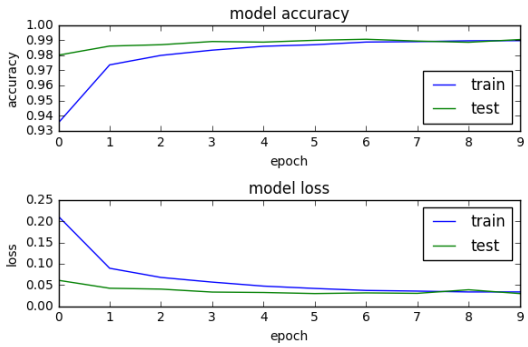

Figure 4. Model Accuracy and Model Loss in Sambit Mahapatra's model [7]

2.2. Kaggle Notebook: Digit Recognition using LeNet-5 Architecture

Yann LeCun proposed the LeNet-5 Architecture in 1998, which was used in the CNN model in Kaggle Notebook [8]. It is quite popular due to its simple structure and easy to train nature. The LeNet-5 architecture is well suited for object recognition and classification in low-resolution images, according to the author.

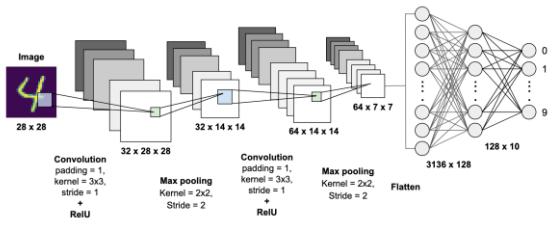

Figure 5. LeNet-5 Architecture [8]

In this project, the author implemented normalization. The purpose of normalization is to reduce the scale of the input values. The pixel value ranges from 0 to 255, indicating a gray gradient. The CNN will converge faster on values 0 to 1 than on values 0 to 255. To scale the data from [0..255] to [0..1], each value is divided by 255. If the data has larger scale, it helps the model learn features better by reducing computational complexities. As the previous related work, the accuracy on the result also reaches 99%.

2.3.3. MNIST Handwritten Digit Classification Using CNN by Milind Soorya [9]

Milind Soorya started with the task to convert the color images (example in Figure 6) in the dataset to grayscale (example in Figure 7), which is black and white yet the value ranges from [0-255]. 0 represents absolute black, while 255 represents absolute white.

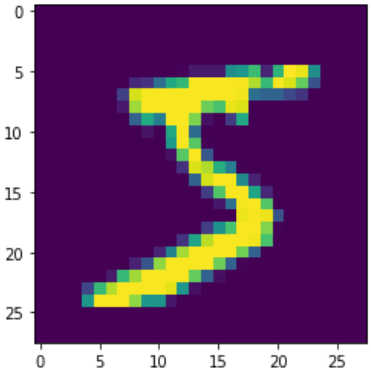

Figure 6. Original Color Image from MNIST dataset [9]

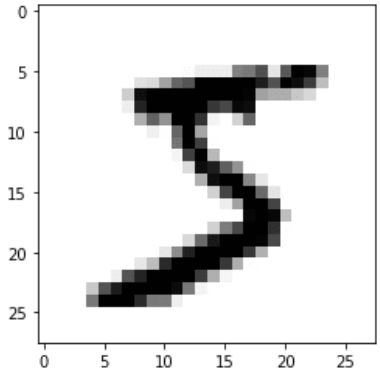

Figure 7. Grayscale Image After Converted [9]

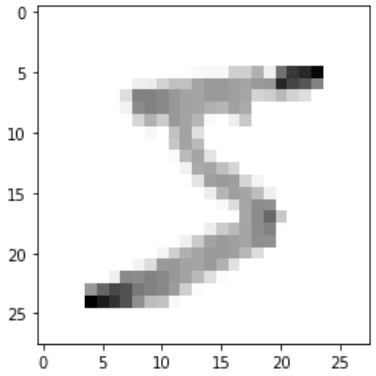

Figure 8. Image After Normalization [9]

Based on the nature of a CNN model, the number of filters, which increases with each layer, is determined by the task's complexity. For handwritten digits, the filter in the first layer is in charge of edge detection, then curves, and continues "with each layer detecting bigger features." [8]

The model after running got an accuracy rate of over ninety eight percent (98.05%) on 10,000 test samples from the MNIST dataset.

3. Proposed Approaches

The recognition problems being applied in practice today focus on pattern recognition, speech recognition, and handwriting recognition, etc. Handwriting recognition is a problem that has drawn great interest because it is one of the requirements in many practical applications. Applications of handwriting recognition can serve to read and process documents, invoices, notes, handwritten notes, etc. Currently, there are several handwriting recognition studies using models such as K-Nearest Neighbor (KNN), Support Vector Machine (SVM), Hidden Markov Model (HMM), ... However, these models give poor recognition results, and take a long time to

extract features in the image. Therefore, in this report, we will build a model that can automatically extract features in the image, and this model is also expected to give better results than the previously built models. From the successes of neural networks in the field of image processing, we will build an advanced machine learning model of convolutional neural networks such as ResNet-18, LeNet, Multi-Layer Perceptron (ML) to solve the handwritten digit recognition problem. Handwriting recognition is done through two forms: online recognition and offline recognition. Online recognition means that the computer will recognize the words written on the screen as soon as it is written. For this recognition system, the computer will save stroke information such as stroke order, direction, and speed of the stroke while it is being written. And offline recognition means that the recognition is done after the words have been written or printed on paper, then the input information is a text image or a character to be recognized. Within the scope of this report, we only consider offline identification for handwritten digits, using the MNIST dataset.

3.1. Model and Database

3.1.1. CNN Model

With the goal of assessing the role of the parameters affecting the output results, the team made an assessment based on three different CNN Models: ResNet-18, LeNet, and MLP and computing changes on the parameters and the model by inserting blocks according to the requirements of each specific survey. In the convolutional neural network model, the layers are linked together through the convolution mechanism. The next layer is the convolution result from the previous layer, so we have local connections. That is, each neuron in the next layer is generated from the filter applied to a local image area of the previous layer neuron. Each such layer is applied to different filters, usually a few hundred to several thousand such filters. Some other layers like the pooling layers are used to filter out more useful information (remove noise information).

During the training process, the CNN will automatically learn the parameters for the filters. For example, in the image classification task as illustrated in Figure 5, CNN will try to find the optimal parameters for the corresponding filters in order from raw pixel > edges > shapes > facial > high-level features. The last layer is used to classify the image.

CNN has location invariance and compositionality attributes. With the same object, if this object is projected from different angles (translation, rotation, scaling), the accuracy of the algorithm will be significantly affected. Pooling layer will give invariants to translation, rotation, and scaling. Local associativity gives us lower-to-higher and more abstract levels of information representation through convolution from filters. That is why CNN produces a model with quite high accuracy. The details of each layer in the model will be demonstrated as follows.

Convolutional Layer

This layer is where the original idea of CNN is expressed. Instead of connecting all the pixels, this layer will use a set of filters that are small compared to the image (3x3 in ResNet-18) applied to an area of the image and perform a convolution between the filters and the image pixel value in that local area. The filter will in turn be shifted by a stride value that runs along the image and scans the entire image.

Thus, with a 28x28 image and a 3x3 filter, we will have a new image of size 28x28 (provided that we have added padding to the original image to compute the convolution for the cases where the filter sweeps to the edges) is the convolution result of the filter and the image. With a certain number of filters in this layer, we will have as many corresponding images that this layer returns and is passed on to the next layer. The initial filter weights will be randomly initialized and will be gradually learned during the training of the model.

Rectified Linear Unit Layer (ReLU Layer)

This layer is usually installed right after the convolution layer. This layer uses the activation function $f(x) = \max(0,x)$. In simple terms, this layer is responsible for converting all negative values in the result taken from the convolution layer to zero. The meaning of this setting is to create nonlinearity for the model. Building on top of linear transformations would make building multilayer pointless. There are many ways to make the model nonlinear such as using activation functions like sigmoid, tanh, etc. but the function $f(x)=\max(0,x)$ is easy to set up, fast to calculate, and still effective. *Inplace* is set to true will replaces the input to output in the memory, which we want to use in terms of memory efficiency.

Pooling Layer

This layer uses a sliding window that scans through the entire data image, each time following a given slide. Unlike the convolution layer, the pooling layer does not compute convolution, but conducts sampling (subsampling). When the window slides over the image, only one value that is considered representative of the image information in that region (sample value) is retained. The common fetching methods in the pooling layer are MaxPooling (get the maximum value), MinPooling (get the minimum value), and

AveragePooling (get the average value). For example, consider an image of size 28x28 and the Pooling layer uses a 2x2 filter with a stride of 2, the method used is MaxPooling (see Figure 5). The filter will slide through the image in turn, with each slide only the largest of the 4 values within the 2x2 window area of the filter is retained and fed into the output matrix. Thus, after going through the Pooling layer, the image will be reduced in size to 14x14 (the size of each dimension is reduced by 2 times).

The pooling layer is responsible for reducing the data size. A large image through many pooling layers will be reduced but still retain the features needed for identification (through sampling). Reducing the data size will reduce the number of parameters, increase the computational efficiency and contribute to the control of overfitting.

Fully Connected Layer (FC Layer)

In this layer, the image values are fully bound to the neurons in the next layer. After the image is processed and features extracted from the previous layers, the image data will not be too large for identification.

3.1.2. MNIST Database

In the experimental part, we use the MNIST dataset (Yann LeCun, Corinna Cortes and Christopher, 1989) [1]. This is a commonly used data set to evaluate the effectiveness of handwritten digital character recognition models. The MNIST dataset is derived from the NIST set provided by the National Institute of Standards and Technology, then updated by LeCunn and divided into 2 separate sets: 60,000 images used for training and 10,000 images used for testing purposes.

The training dataset consisting of 60,000 28x28 images of handwritten digits was used to train the machine learning model. All images in the dataset are aligned and transformed into point data consisting of 60,000 elements (numerical characters) with 784 values ranging between 0 to 255 and 10 layers (values from 0 to 9).

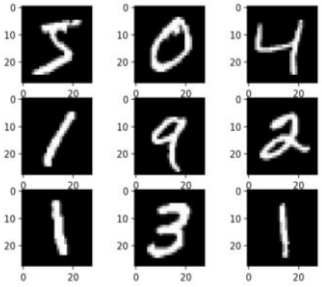

Figure 9. A Subset of Images from MNIST [5]

The test dataset consisting of 10,000 images of handwritten digits is used for testing. The images in the test dataset are also transformed and aligned to point data similar to the training dataset. Figure 9 is an example of some samples of the data set.

4. Implementation

4.1. t-SNE Exploratory Data Analysis

```
from torchvision.datasets import MNIST
from torchvision.transforms import ToTensor
import torch
import torchvision
import torchvision.transforms as transforms
from torchvision.models import resnet18, googlenet
from torch import nn
from torch.utils.data import DataLoader, random_split
import pytorch_lightning as pl
from tqdm.autonotebook import tqdm
from sklearn.metrics import classification_report
import os
import pandas as pd
import seaborn as sn
import numpy as np
from torch.nn import functional as F
from torchmetrics import Accuracy
from sklearn.manifold import TSNE
from torch.utils.tensorboard import SummaryWriter
import matplotlib.pyplot as plt
from random import randint
writer = SummaryWriter()

PATH_DATASETS = os.environ.get("PATH_DATASETS", ".")
BATCH_SIZE = 256 if torch.cuda.is_available() else 64

train_ds = MNIST("mnist", train=True, download=True, transform=ToTensor())
test_ds = MNIST("mnist", train=False, download=True, transform=ToTensor())

train_dl = DataLoader(train_ds, batch_size=64, shuffle=True)
test_dl = DataLoader(test_ds, batch_size=64)
```

Figure 10. Importing libraries

To ensure that the session runs properly, some preparation is essential before the session begins. To begin, the programmer set up the environment by importing the necessary libraries for future function usage. The data is saved in variables and reshaped to the necessary image dimensions (28x28). The data has been processed and normalized before being loaded into the data bunch.

$$x = np.reshape(x, (1000, 28 * 28))$$

This dataset can be readily transformed into an unsupervised format for simple visualization. We decomposed this dataset using t-Distribuited Stochastic Neighbour Embedding (t-SNE), allowing us to look at each image in a bi-dimensional space. The t-SNE transformation is defined, and the data are converted as follows:

$$tsne = TSNE(n\_components = 2)$$

$$X\_r3 = tsne.fit\_transform(x)$$

According to Christopher Olah [12], The goal of t-SNE is to keep the data topology intact. It creates an idea of which other points are its 'neighbors' for each

point, attempting to make all points have the same number of neighbors. Then it tries to embed them in such a way that each point has the same number of neighbors. t-SNE is like graph-based visualization in certain ways. Instead of having points be neighbors (if an edge exists) or not neighbors (if no edge exists), t-SNE has a continuous spectrum of having points be neighbors to varying degrees. In many cases, t-SNE is quite effective at revealing clusters and subclusters in data. Our visualization of the results is shown in Figure 11 below.

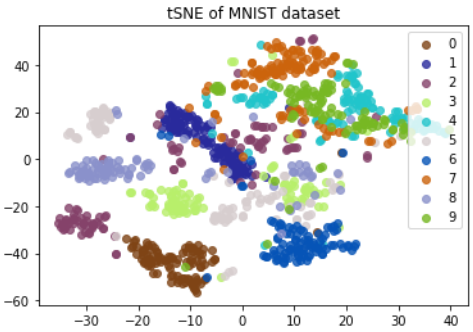

Figure 11. Visualizing MNIST with t-SNE on 1000 dataset

The above plot was performed on a 1000 dataset for demonstration purposes. The different clusters for each label may be seen in Figure 11. t-SNE groups data visually, so that the same color point in low-dimensional space forms one cluster. The points that are grouped together are closer together. [13] With observation, there are 10 clusters. We could see most of the numbers' data points are spread out except for "6", "1" and "0".

However, t-SNE produces good results with a large dataset. We tried the t-SNE on a 50,000 dataset. Below is the visualization of clusters. The expectation was true when data points of each label formed a better cluster and less spread out.

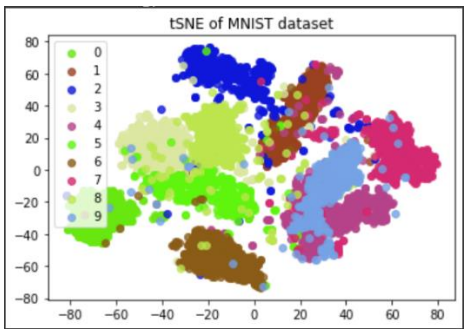

Figure 12. Visualizing MNIST with t-SNE on 5000 dataset

Our prediction was that if dataset size was increased more than 10,000, the clusters would form even closer together.

4.2. Training

4.2.1. ResNet-18

After the MNIST data has been downloaded to the computer, we train a CNN model on this dataset. As stated above, the implemented algorithm and architectures are CNN ResNet-18, MLP, and LeNet. Residual networks are deep networks that overcome the degradation problem by allowing a layer or group of layers to learn the residual of a mapping function instead of learning the entire mapping function. To put it another way, if the input is a simple variable *x*, the function that the layers are trying to approximate *F(x)*, and the output is the needed mapping *R(x)*, the learned function *R(x)* is

$$R(x) = F(x) - x \quad (1)$$

This is done by providing a direct shortcut connection from the input to the output which contains two mapping layers, each followed by a Rectified Linear Unit (ReLU).

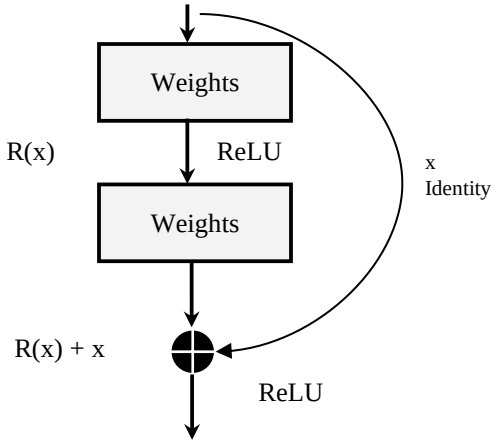

Figure 13. Working block diagram of ResNet [10]

This is done by providing a direct shortcut connection from the input to the output, as shown in Figure 13, which contains two mapping layers, each followed by a Rectified Linear Unit (ReLU).

ResNet uses the same 3x3 convolutional layer design. Two 3x3 convolutional layers with the same number of output channels comprise the residual block. The batch normalization layer and the ReLU activation function follow each convolutional layer. The input is then added before the final ReLU

activation function, skipping these two convolution procedures. The output of the two convolutional layers must have the same form as the input to be merged together in this configuration. If the number of channels is changed, an additional 1x1 convolutional layer must be added to turn the input into the correct shape for the addition operation.

The testing data is used to assess the learning model's performance. The model will score the likelihood of each conceivable class, and the photos will determine which class has the highest probability. The accuracy on the MNIST dataset with ResNet-18 was determined to be around 99 percent on Kaggle [8].

First, the model contains a convolution layer and a max pooling (`nn.Conv2d(1, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False`) with input dimension of `[1, 28, 28]`. With dense layer and softmax activation function, probability distribution over 10 subclasses is calculated. The classifier with the highest value will be the predicted digit. That's why we need 10 elements in the output layer. The loss function we use is cross-entropy (`nn.CrossEntropyLoss`). We define the parameters used to track the training process `['loss', 'lossVal', 'val_accuracy', 'train_accuracy']` which includes the accuracy and loss of both training and validation sets. The training set is used to train the model, and the test set is used to monitor the performance of the model after each epoch. If the model performs well on the training set but not well on the test set, there is a high chance that the model is overfitting.

```
   | Name           | Type             | Params
-------------------------------------------------
0 | model          | ResNet           | 11.2 M
1 | loss           | CrossEntropyLoss | 0
2 | val_accuracy   | Accuracy         | 0
3 | train_accuracy | Accuracy         | 0
-------------------------------------------------
11.2 M    Trainable params
0         Non-trainable params
11.2 M    Total params
44.701    Total estimated model params size (MB)
```

Figure 14. ResNet Parameters' Size

4.2.2. Multi-Layer Perceptron (MLP)

Multilayer Perceptron (MLP) is a supervised machine learning algorithm belonging to the class of Artificial Neural Networks, which is a collection of perceptrons divided into many groups, each group corresponds to a layer. The algorithm is basically trained on the data to learn a nonlinear function for classification or regression purposes, given a set of features and a target variable (e.g. labels). This report focuses on the classification case.

Logistic regression has only two layers, including input and output layers, and with MLP, in addition to Input and Output layers, the layers of neurons in the middle are collectively called Hidden layers. Because of this non-linearity, MLP can learn complex nonlinear functions and distinguish data that is not linearly separable. However, MLP also has some disadvantages, such as the decomposed function for the hidden layer, leading to the problem of non-convex optimization and the existence of local minima. Therefore, initializing different input volumes can lead to different outputs. In addition, MLP uses a number of hyper-parameters, including the number of hidden neurons or the information of the layers that need to be adjusted, analyzing these hyper-parameters easily leading to a waste of time and resources. In particular, MLP can be sensitive to feature expansion.

Each input 2D image is converted to a 1D vector with size `[1, 28, 28] = [1, 784]`. This results in 70 000 flat images (samples), each containing 784 pixels (28×28 = 784). Thus, the weight matrix of the input layer will have the shape `784 x number of neurons in the first hidden layer (nn.Linear(1*28*28, layers[0])`. The weight matrix of the output layer will have the shape `number of neurons in third hidden layer x number of classes (nn.Linear(layers[1], 10))`. [11] This step proceeds through the process of building the model, training and performing the classification.

```
   | Name           | Type             | Params
-------------------------------------------------
0 | layers         | Sequential       | 105 K
1 | loss           | CrossEntropyLoss | 0
2 | val_accuracy   | Accuracy         | 0
3 | train_accuracy | Accuracy         | 0
-------------------------------------------------
105 K     Trainable params
0         Non-trainable params
105 K     Total params
0.421     Total estimated model params size (MB)
```

Figure 15. MLP Parameters' Size

Our MLP architecture was built manually and applied the same approach that was suggested by Serafeim Loukas [11]. MLP classifier is a strong neural network model that allows complex data to learn non-linear functions. The loss is computed after the weights are built using forward propagation. The

weights are then updated using back propagation, which reduces the loss. As I mentioned in the introduction, this is done iteratively, with the number of iterations serving as an input hyperparameter. The number of neurons in each hidden layer, as well as the overall number of hidden layers, are also essential hyperparameters. These have to be tuned later on.

4.2.3. LeNet [8]

The convolutional neural nets (convnets or CNNs) (also known as LeNet) that brought some initial success to the handwritten digit recognition problem was started by Yann LeCun at AT&T Bell Labs (Yann LeCun is a student at AT&T Bell Labs). Hinton's graduate school at the University of Toronto in 1987-1988). LeNet network is a CNN with 5 layers, also known as LeNet-5 (1998). This model is widely used in systems for reading handwritten numbers on checks (bank checks) and zip codes in the United States. LeNet was the best algorithm at that time for handwritten digit image recognition problems. It is better than conventional MLP (with fully connected layers) because it has the ability to extract two-dimensional features of the image through two-dimensional filters. Moreover, these filters are small so the storage and calculation are also better than conventional MLP. [8]

Based on the approach in the book *Dive Into Deep Learning* [6], we implemented the model as below.

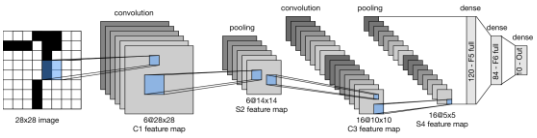

Figure 16. Data Flow in LeNet [6]

We can consider LeNet as consisting of two parts: (i) a block of convolutional layers; and (ii) a block of fully connected layers. “Input is a handwritten digit; output is a probability of 10 possible outcomes.” [6] A convolutional layer, a sigmoid activation function, and a subsequent average pooling operation are the basic units of each convolutional block. While ReLUs and max-pooling are more effective, they were not discovered until the 1990s. A kernel and a sigmoid activation function are used in each convolutional layer. These layers translate a number of two-dimensional feature maps to spatially arranged inputs, often increasing the number of channels. The output channels of the first convolutional layer are six, whereas the second has sixteen. Through spatial down sampling, each pooling operation (stride 2) reduces dimensionality by a factor of. The convolutional block produces a shape-based output (batch size, number of channels, height, width). We must flatten each example in the minibatch to pass output from the convolutional block to the dense block. In other words, we convert this four-dimensional input into the two-dimensional input that fully connected layers expect as a reminder, the two-dimensional representation we want uses the first dimension to index minibatch examples and the second to give each example a flat vector representation. The dense block from LeNet contains three fully connected layers, each with 120, 84, and 10 outputs. The 10-dimensional output layer corresponds to the amount of possible output classes because we are still performing categorization. [6]

```
  | Name           | Type             | Params
-------------------------------------------------
0 | model          | GoogLeNet        | 10.0 M
1 | loss           | CrossEntropyLoss | 0
2 | val_accuracy   | Accuracy         | 0
3 | train_accuracy | Accuracy         | 0
-------------------------------------------------
10.0 M    Trainable params
0         Non-trainable params
10.0 M    Total params
39.817    Total estimated model params size (MB)
```

Figure 17. LeNet Parameters’ Size

4.3. Empirical Result Comparison

Hyperparameter tuning is applied to the learning rate of each model. We decided to keep the other two constants: batch size (64) and number of epochs (10) to ensure that we knew what was causing the changes we would see. To expedite the tests, we tested the various learning rates on 10 epochs.

4.3.1. Results of MLP

Table 2 shows the result of a 64-batch experiment with 10 epochs and a variable learning rate. Increasing learning rate has no significant impact on testing, training, and validation accuracy. For most situations, accuracy on the MNIST dataset is greater than 97 percent.

Learning Rate	Training accuracy	Validation accuracy	Testing accuracy
0.02	97.0%	96.7%	97.1%
0.01	97.2%	97.1%	97.1%
0.0002	97.2%	97.3%	96.7%
0.0001	97.1%	97.3%	96.6%

Table 1. Learning Rate Experimentation of MLP

With the default learning rate (2e-4 = 0.0002), batch size 64, and 25 epochs, below figures are the visualization plots of cross entropy loss, train accuracy, and validation accuracy running MLP.

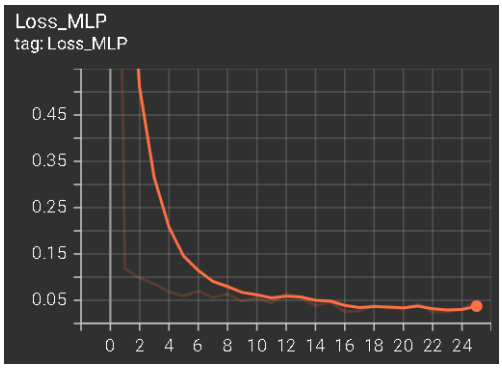

Figure 18. Loss Plot

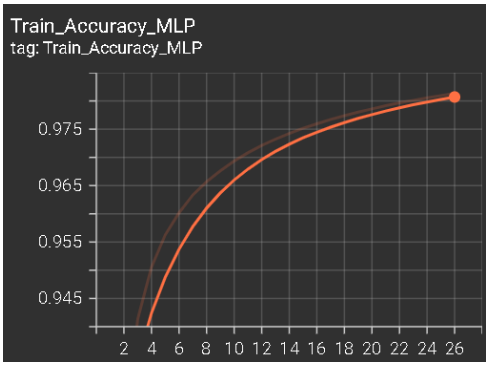

Figure 19. Train Accuracy Plot

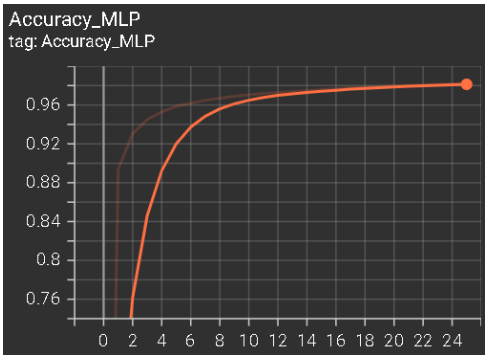

Figure 20. Validation Accuracy Plot

4.3.2. Results of LeNet

Table 2 shows us the learning rate experimentation when doing hyperparmeter tuning. There is also no significant impact on the training and testing accuracy. However, we noticed the validation accuracy of learning rate 0.01 and 0.0002 is significantly lower than the others, which we have not figured out the reason for it yet.

Learning Rate	Training accuracy	Validation accuracy	Testing accuracy
0.02	95.3%	96.9%	99.3%
0.01	95.3%	91.7%	99.2%
0.0002	96.3%	88.0%	98.1%
0.0001	96.2%	97.1%	99.2%

Table 2. Learning Rate Experimentation of LeNet

With the default learning rate (2e-4 = 0.0002), batch size 64, and 25 epochs, below figures are the visualization plots of cross entropy loss, train accuracy, and validation accuracy running LeNet.

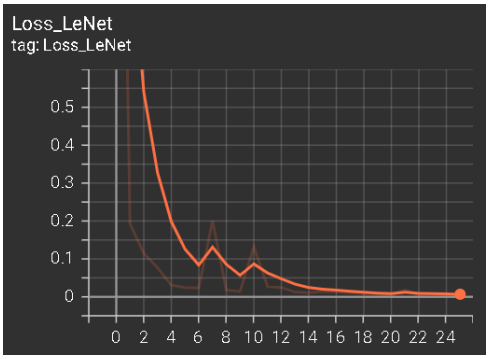

Figure 21. Loss Plot

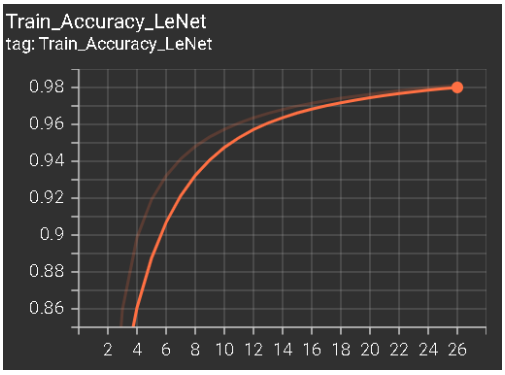

Figure 22. Train Accuracy Plot

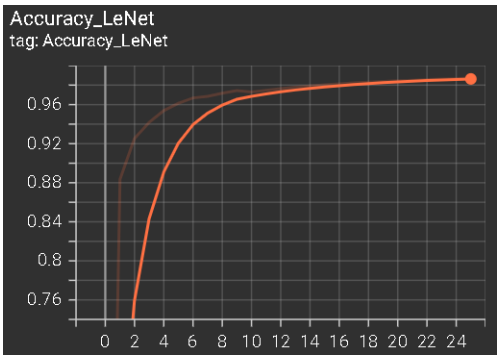

Figure 23. Validation Accuracy Plot

4.3.3. Results of ResNet-18

The best training and testing accuracy for the MNIST dataset were 98.2 percent and 98.3 percent, respectively with the default learning rate as 0.0002. The table below also shows the result of a 64-batch experiment with 10 epochs with other experiment learning rates. Like MLP and LeNet, there is no significant impact on the accuracy rates by increasing learning rate in ResNet-18 model. For most situations, accuracy on the MNIST dataset is greater than 98 percent, which is 1% higher than the MLP.

Learning Rate	Training accuracy	Validation accuracy	Testing accuracy
0.02	98.2%	97.8%	99.1%
0.01	98.0%	98.2%	99.2%
0.0002	98.2%	98.3%	99.2%
0.0001	98.1%	97.2%	99.4%

Table 3. Learning Rate Experimentation of ResNet-18

With the default learning rate (2e-4 = 0.0002), batch size 64, and 25 epochs, below figures are the visualization plots of cross entropy loss, train accuracy, and validation accuracy running ResNet-18.

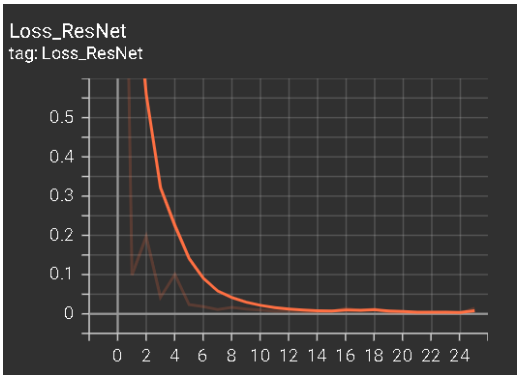

Figure 24. Loss Plot

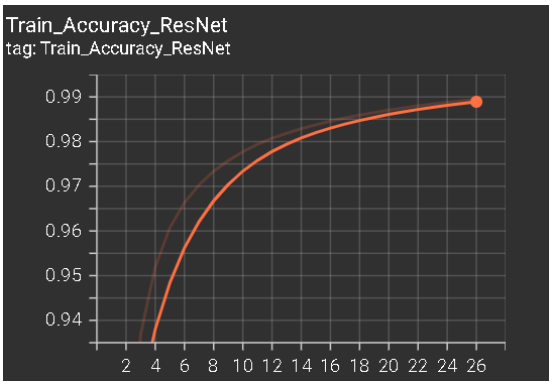

Figure 25. Train Accuracy Plot

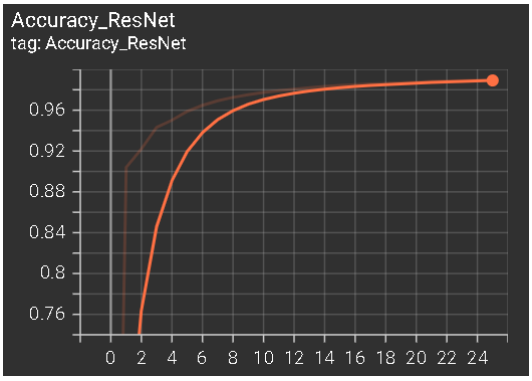

Figure 26. Validation Accuracy Plot

4.3.4. Result Comparisons

The outcomes of proposed CNN architectures are compared in this subsection. The collected results are also compared to the literature. The results show that CNNs can solve the MNIST dataset.

Model	Training accuracy	Validation accuracy	Testing accuracy	Training Time	Cross Entropy Loss
ResNet-18	98.9%	99.0%	99.1%	502s	1.4%
LeNet	98.1%	98.7%	99.2%	908s	0.6%
MLP	98.4%	98.6%	97.3%	235s	4.2%

Table 4. Comparison of Extracted Results

All architectures produce results that are close to or better than 97 percent. ResNet-18 provides the highest level of precision in general while MLP provides the fastest training time. The training time required for proposed architectures to solve the MNIST dataset is shown in Figure 17. MLP has fastest training time yet the most loss. With more convolutional layers, training time increases, which is what happened to ResNet-18.

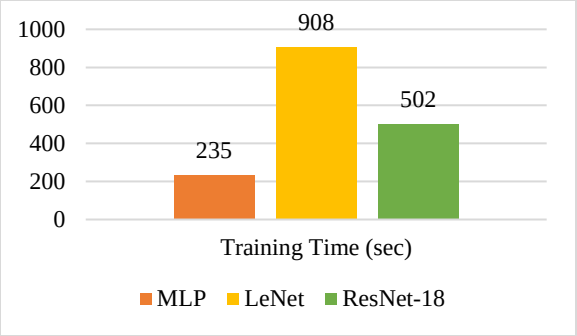

Figure 27. Training Time Comparison

5. Conclusions

The report made an introduction to different convolutional neural networks and examined the parameters as well as function blocks, assessing their role and impact on the output image recognition rate. The project implemented different CNN network models (ResNet-18, MLP, and LeNet) to solve the handwritten digit recognition problem using the MNIST dataset. Any CNN model gives over 97 percent accuracy for the MNIST dataset, according to the results. With only one input layer and two fully connected layers, MLP takes less time to train the algorithm. With more convolutional layers, training time increases. With a 3x3 filter size, training time increases significantly as in ResNet-18.

6. References

[1] "The MNIST database," *MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges*. [Online]. Available: http://yann.lecun.com/exdb/mnist/. [Accessed: 10-May-2022].

[2] A. P, "Handwritten digit recognition using pytorch," *Medium*, 21-May-2018. [Online]. Available: https://medium.com/@athul929/hand-written-digit-classifier-in-pytorch-42a53e92b63e. [Accessed: 10-May-2022].

[3] Y. Chandola, J. Virmani, H. S. Bhadauria, and P. Kumar, "End-to-end pre-trained CNN-based computer-aided classification system design for chest radiographs," *Deep Learning for Chest Radiographs*, 23-Jul-2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780323901840000114. [Accessed: 10-May-2022].

[4] "Using ResNet for MNIST in pytorch," *Marcin Zabłocki blog*, 06-Jan-2019. [Online]. Available: https://zablo.net/blog/post/using-resnet-for-mnist-in-pytorch-tutorial/. [Accessed: 10-May-2022].

[5] J. Brownlee, "How to develop a CNN for MNIST handwritten digit classification," *Machine Learning Mastery*, 14-Nov-2021. [Online]. Available: https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/. [Accessed: 10-May-2022].

[6] *Dive into Deep Learning 0.17.5 documentation*. [Online]. Available: https://www.d2l.ai/chapter_convolutional-modern/resnet.html. [Accessed: 10-May-2022].

[7] S. Mahapatra, "A simple 2D CNN for MNIST digit recognition," *Medium*, 22-May-2018. [Online]. Available: https://towardsdatascience.com/a-simple-2d-cnn-for-mnist-digit-recognition-a998dbc1e79a. [Accessed: 10-May-2022].

[8] arunrk7, "Digit recognition using CNN (99% accuracy)," *Kaggle*, 13-May-2020. [Online]. Available: https://www.kaggle.com/code/arunrk7/digit-recognition-using-cnn-99-accuracy/notebook. [Accessed: 10-May-2022].

[9] "MNIST handwritten digit classification using CNN," *Milind Soorya*, 17-Oct-2021. [Online]. Available: https://www.milindsoorya.com/blog/handwritten-digit-classification-using-cnn. [Accessed: 10-May-2022].

[10] G. Abosamra and H. Oqaibi, "An optimized deep residual network with a depth concatenated block for handwritten characters classification," *Tech Science*. [Online]. Available: https://www.techscience.com/cmc/v68n1/41811/html. [Accessed: 10-May-2022].

[11] S. Loukas, "Classifying Handwritten Digits Using A Multilayer Perceptron Classifier (MLP)," *Medium*, Dec. 01, 2021. https://towardsdatascience.com/classifying-handwritten-digits-using-a-multilayer-perceptron-classifier-mlp-bc8453655880 (accessed May 11, 2022).

[12] "Visualizing MNIST: An Exploration of Dimensionality Reduction - colah's blog," *colah.github.io*. https://colah.github.io/posts/2014-10-Visualizing-MNIST/ (accessed May 11, 2022).

[13] R. singh, "T-SNE visualization of high dimension MNIST dataset," *Medium*, Sep. 05, 2019. https://ranasinghiitkgp.medium.com/t-sne-visualization-of-high-dimension-mnist-dataset-48fb23d1bafd (accessed May 11, 2022)