Middlesex University London

CST3170 - Artificial Intelligence

HANDWRITTEN DIGITS RECOGNITION

Name: Francesco Arrabito
Student ID: M00696513

Tutor: Chris Huyck

Introduction

The purpose of this project is to create a machine learning model capable of categorising and predicting handwritten digits.

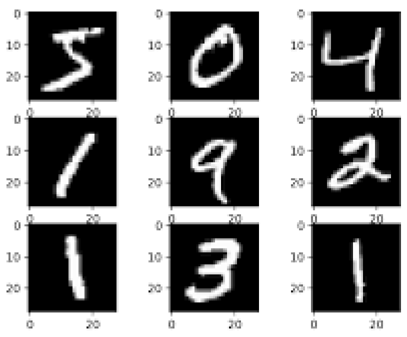
The technique used in this study is the building of a specific MLP called the Convolutional Neural Network (CNN), which has attracted substantial interest in its understanding and implementation from scratch owing to its shown performance in the area of Computer Vision.

Additionally, this architecture is implemented with an optimizer called ADAM, which stabilises and enhances the gradient descent approach during back propagation, hence enhancing training and generating better results.

- **Approach: CNN (Convolutional Neural Network)**
- **Learning strategy: GDMB (Gradient descent with Mini-Batch)**
- **Optimizer: ADAM (Adaptive Moment Estimation)**

Problem Definition

Implement an architecture capable of training on one of the two datasets supplied and then correctly classifying and predicting images of such handwritten digits from the second dataset.



Proposed Solution

Choosing which network type to use might be difficult for a first deep learning approach. There are many network types to choose from, and new approaches are introduced regularly.
Worse, most neural networks are flexible enough to operate (make a prediction) even when given the wrong input or task.

Deep learning is the use of artificial neural networks on modern hardware, which allows for far larger (more layers) neural networks to be built, trained, and used than previously thought.

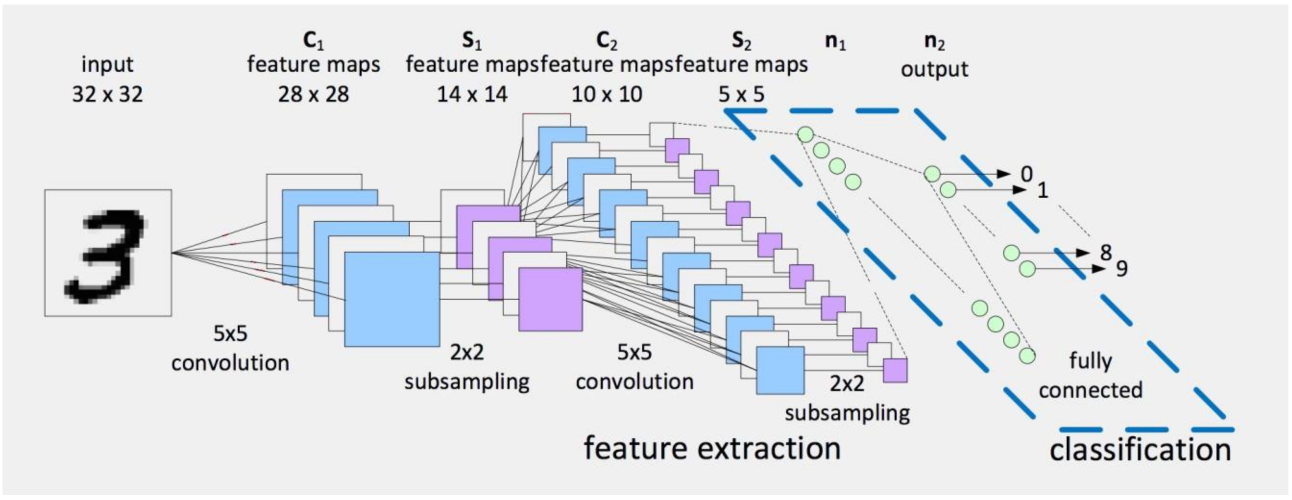
Researchers have proposed hundreds of different neural network modifications to existing models. Rarely are completely new procedures employed.

The most well-known artificial neural networks are:

- **Multy Layer perceptrons (MLPs)**
- **Convolutional Neural Networks (CNNs)**
- **Recurrent Neural Networks (RNNs)**

These three networks have proven successful and reliable in a wide range of applications over decades. Specifically, CNN was designed for images classification thus particularly efficient in Computer vision.

This characteristic determines this architecture to be the approach chosen to tackle the accomplishment of this project.



Selection justification

CNN were created to translate picture data to an output variable (convolutional neural networks), which precisely match this projet needs.

They are so effective that they have established a standard for any prediction problem when using image data. Infact, the benefit of CNNs is that they can internalise a two-dimensional image.
Through the use of appropriate filters, a ConvNet is capable of effectively capturing the spatial and temporal relationships in a picture. Due to the reduced number of parameters and reusability of weights, the architecture achieves a better fit to the images dataset. In other words, the network can be taught to recognise the image's complexity.

CNNs are often used for image data, classification, and regression prediction.
Its input is typically two-dimensional, but it may also be one-dimensional, allowing it to build an internal representation of a one-dimensional sequence. As a result, the CNN may be used on more data types.

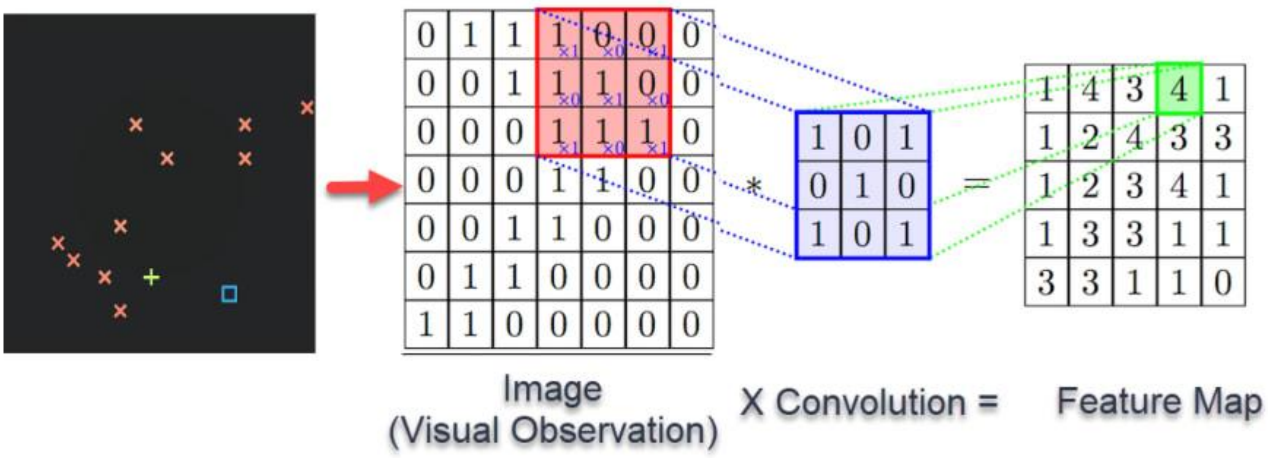
Accordigly, while CNNs were not designed to analyse data other than images, they achieve state-of-the-art performance in various tasks other than computer vision.

CNN

CNN is a neural network with at least one convolutional layer that categorises and classifies pictures processed based on the characteristics extracted through filter application.

Convolutional operations are defined by their ability to compress an image by running it through a Feature detector/Filter/Kernel and converting it to a Feature Map, Convolved Feature Map, or Activation Map.
Additionally, it assists in removing superfluous information from the picture.

Technically, convolution entails multiplying the convolutional filter element by element by the slice of an input matrix and eventually summarising all values in the output matrix.

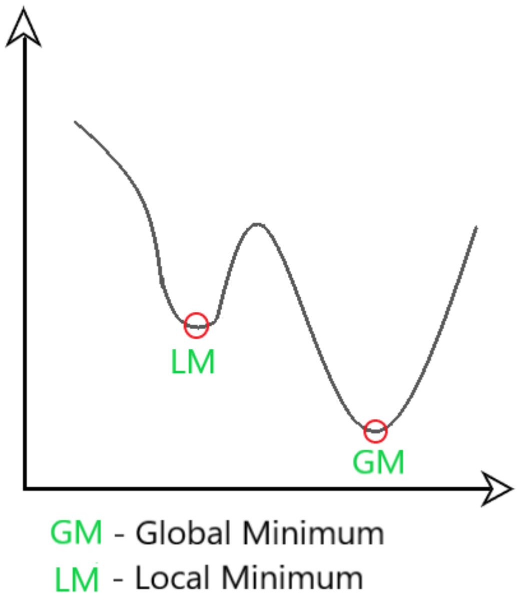


ADAM Optimizer

Adaptive Moment Estimation is a method for gradient descent optimization. The approach is very efficient when dealing with complex problems with a large number of variables or data. It is memory-efficient and consumes less memory. On the surface, it seems to be a mix of the 'gradient descent with momentum' and the 'RMSP' algorithms.

Adam optimizer employs a hybrid of two gradient descent algorithms, Momentum and RMSP.

Momentum: This approach accelerates the gradient descent process by taking into account the gradients' exponentially weighted average.



Using averages accelerates the algorithm's convergence to the minima.

Root Mean Square Propagation (RMSP): RMSprop is an adaptive learning technique that attempts to enhance AdaGrad. Rather of calculating the cumulative sum of squared gradients like AdaGrad does, it computes an 'exponential moving average'.

Adam Optimizer takes the advantages or positive characteristics of the preceding two techniques and improves on them to provide a more optimal gradient descent.

MODEL

Apart from the input and output layers, the proposed model consists of three hidden layers, two of which are convolutional and one of which is fully linked.

The first convolutional layer was created using 32 filters (nodes), a kernel by 2X2, and a Leaky RELU activation function.

The first convolutional layer was constructed using 64 filters (nodes), the same kernel, and the same activation function as described earlier.

The fully connected layer, also known as DenseNet, utilises 128 nodes and the same activation function as the previous layers.

The last layer, namely the output layer, has as many nodes as categories need, which is 10, and this time, by using an activation function called SoftMax, which is often employed as the final layer's activation function.

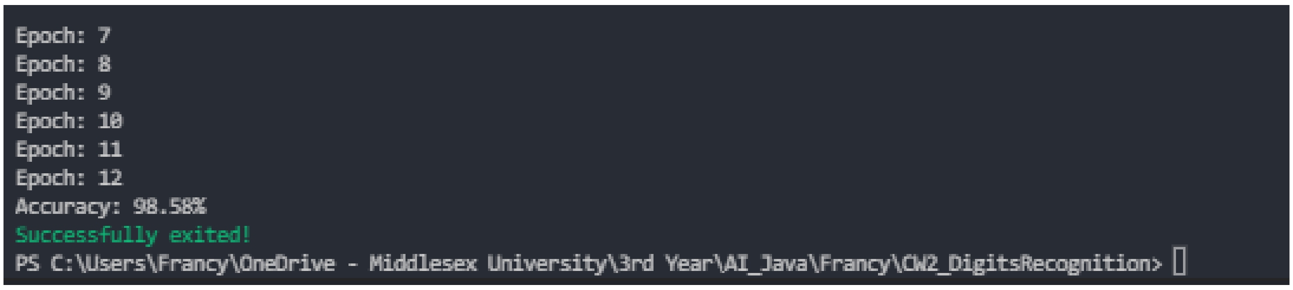
A Categorical-Cross-Entropy approach was used to calculate the error or loss function.

To conclude, the Learning rate was set with a value of 0.001, the mini-batch was equals to 4 and epochs used by 50.

TESTS

Accuracy: 98.58%

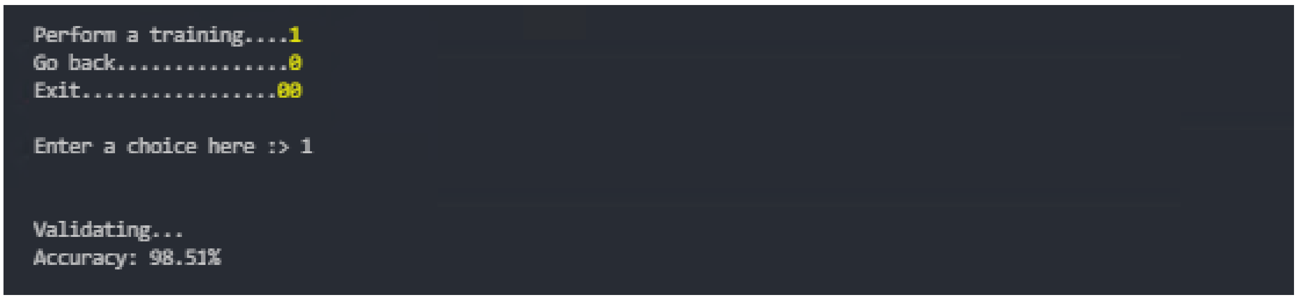
Training with the Dataset 1 for 12 Epochs and then performing the validation on Dataset 2



```
Epoch: 7
Epoch: 8
Epoch: 9
Epoch: 10
Epoch: 11
Epoch: 12
Accuracy: 98.58%
Successfully exited!
PS C:\Users\Francy\OneDrive - Middlesex University\3rd Year\AI_Java\Francy\CW2_DigitsRecognition> 
```

Accuracy: 98.51%

Training with the Dataset 1 for 50 Epochs and then performing the validation on Dataset 2



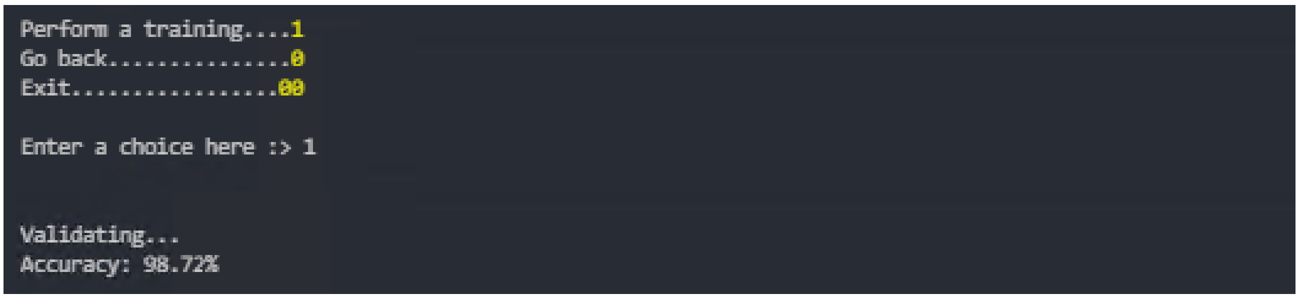
```
Perform a training....1
Go back...............0
Exit.................00

Enter a choice here :> 1


Validating...
Accuracy: 98.51%
```

Accuracy: 98.72%

Training with the Dataset 2 for 50 Epochs and then performing the validation on Dataset 1



```
Perform a training....1
Go back...............0
Exit.................00

Enter a choice here :> 1


Validating...
Accuracy: 98.72%
```