Middlesex University London

CST3170 Artificial Intelligence Coursework Two

Digit Recognition

Edoardo Fratantonio
M00702510

Introduction

This project focuses on the Digit Recognition Problem, a well known machine learning problem, which is about the computer recognizing handwritten digits from images. The following algorithms were used in this project:

- CNN. (Convolutional Neural Network) :

  - Linear Transformation.

  - Non-linear Activation Functions:Leaky ReLu and Softmax.

  - Gradient Descent and Mini-Batch. (Optimisation)

Proposed Method

As previously stated, the CNN (Convolutional Neural Network) is my proposed method since it is one of the most widely used algorithms for computer vision and is extremely powerful. It differs from the well-known Neural Network in that it is implemented with Convolutional Layers that are not fully connected layers, while the Neural Network uses a fully connected structure, which is a part of the CNN indeed.

Convolutional Neural Network(CNN)

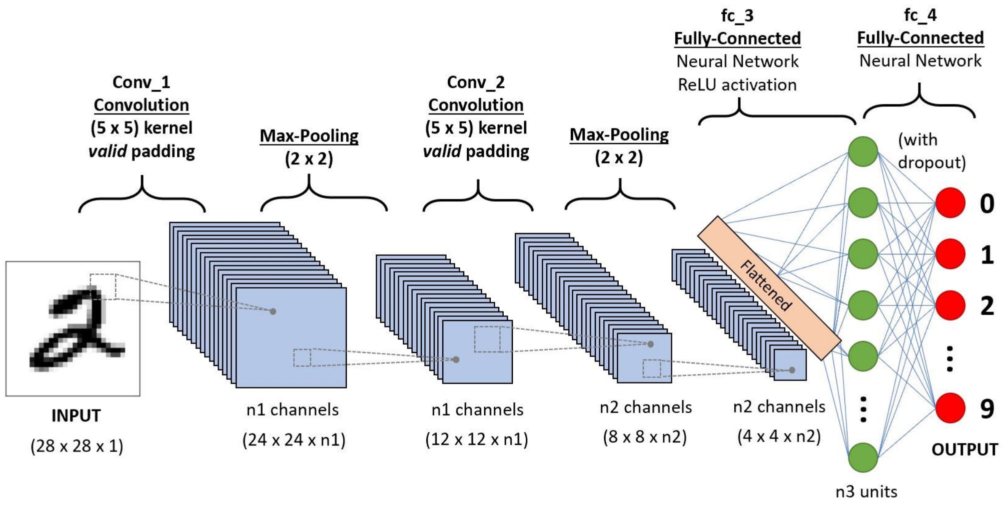
The Convolutional Neural Network[1.0] is a very powerful algorithm. Essentially it is composed of two parts, **Forward Propagation** and **Back Propagation.**

Forward Propagation

A picture is fed into the model as an input, after which it proceeds to the Convolution Layer, in order to extract features from the image, where the picture is multiplied to a matrix of weights, and then added to the bias, which in this instance will be one for each output from the sum of the input and the kernel matrix.

The kernel[1.1] is initialised based on the Activation function that will be used; in this case, the Activation function will be the Leaky ReLu, which is an updated version of the well-known ReLu. The ReLu can have a vanishing gradient problem, which means that the weights can become so small that they misguide the model to make the correct predictions during the training process. In this case, the Leaky ReLu comes into play,since it was designed in order to tackle the vanishing gradient issue and, as a consequence, may provide significantly superior results.

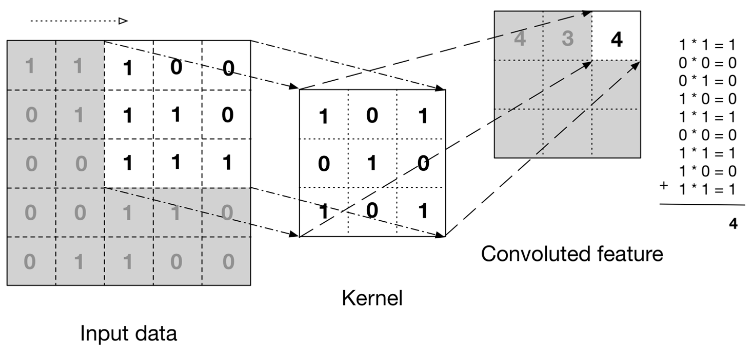
[Fig 1.0 CNN Structure]



Following the Convolutional Layer, also known as linear transformation, and the activation function, also known as non-linear transformation, the structure will output a N matrix of weights, which will then need to be passed to the flatten layer,which will transform the data from 2-Dimension to 1-Dimension, as the fully connected layer only accepts one dimension data. Afterwards, the data will then be fed to the Neural Network; these final layers are referred to as Dense layers or fully connected layers. In the final layer, there will be as many outputs as there are labels, in this case 10 outputs will be required. These outputs will be passed through Softmax, which is a widely used algorithm, to ensure that the sum of the total outputs equals to 1, which will be used to calculate a percentage out of the results obtained.

Once the outputs are calculated from the Softmax, the loss function is executed in order to understand how the model performed; in my case, because I am using a Leaky ReLu as the activation function, the loss function is the Cross Entropy[1.2], which will calculate how far the model prediction was from the actual label. From this loss, the Back Propagation process begins to learn more about the model's performance.

[Fig 1.1 Kernel Structure]



[Fig 1.2 Cross Entropy Formula]

$$L_{example} = -\log(\hat{y}) \cdot y$$

Back Propagation

Back Propagation is essentially the core of the CNN as this is where the model actually trains itself in order to adjust its parameters, weights and biases, in order to tune the final prediction and decrease the loss function, hence increasing the accuracy of the final prediction.

It starts at the loss, the last layer, and works its way back to the first layer, applying the chain rule to calculate the gradient descent, which is required by the model in order for it to understand whether it needs to lower the parameters or raise them, essentially in which direction it needs to move so that it can minimise the loss value as much as possible and try to overcome the local minima and get closer to the global minima.

[Fig 1.3 Chain Rule]

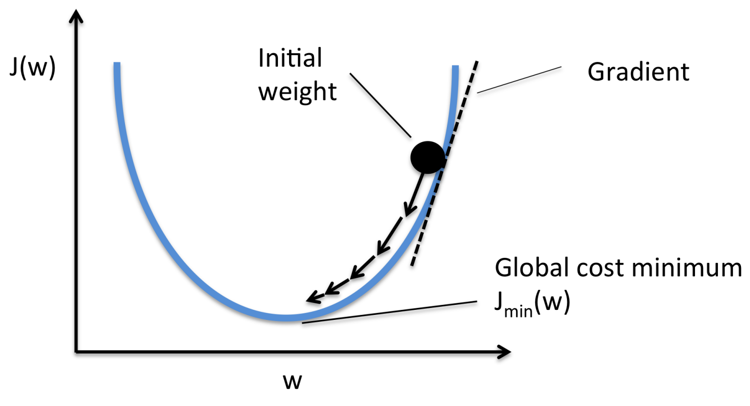
$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$$

$\frac{dy}{dx}$ = derivative of y with respect to x

$\frac{dy}{du}$ = derivative of y with respect to u

$\frac{du}{dx}$ = derivative of u with respect to x

[Fig 1.4 Gradient Descent]



Program Structure

Main Class

This is the main class, where it will ask the user if they would like to train the model or exit the software.

CNN Class

This class is where the user can structure the model how they wish, adding layers, deleting layers, modifying parameters, and datasets.

Dataset Class

This class is where the data will be read in and stored in and the Sample object will be created.

Sample Class

This class contains all the Sample information for making in order to store the label of the sample and the image in 1-Dimension and 2-Dimensions.

Layers Class

This class will initialise the layers, from Convolutional to the fully connected layers, it will keep all the utilities needed for the Layers.

Layers_init Class

This class will initialise the very first layer, whether it is a new Convolutional Layer or Dense Layer, in order to prepare the layer with all the right array sizes.

Model_Settings Class

This class will keep the training, the forward propagation, the validation steps, and call them in order to pass the parameters to the Layers class in order to perform the Forward or Back Propagation, or the update of the parameters.

Neuron Class

This class will keep all the essentials for a Neuron, which will need few parameters inside it, such as the Kernel matrix, the gradients, the number of outputs and the bias.

Node Class

This class is the node inside the neuron, which would be the parameters, with all their connections, the forward propagation linear sum, the back propagation linear sum, the chain rule, the partial derivative, and the derivative, all attributes which will be needed to perform the back propagation.

Utils Class

This class is where all the utilities are kept, created by myself,, which are software related and not CNN related, which can come in handy in order to perform certain operations.

Activations Class

This class is where all the activation functions used for the software have been kept, in order to decide which one would perform better.

Loss Class

This class is where all the Loss functions used are kept, in order to decide which one would perform the best with the software.

Model Structure

My model starts with two Convolutional Layers, where the input image will be given to 32 neurons, with 32 different kernels of 3 x 3, which will be multiplied up to the input image, added to the bias and then passed through the activation function, Leaky ReLu.

The second layer will be given as an input, all the 32 outputs from the previous layer for each neuron, where now are 64 neurons, whereas as many kernels as the previous layers are given as input, therefore with 32 different kernels of 3x3 for each neuron which will be then all added up to each other, to the respective index, making one single kernel and then passed to the activation function.

1st Hidden Layer ——-->	**Layers.Conv2d(32, 3, 3, Leaky ReLu)**
2nd Hidden Layer —--->	**Layers.Conv2d(64, 3, 3, Leaky ReLu)**

Afterwards, the last Convolutional Layer, the Dense Layer starts, which is the fully connected layer, with 128 neurons, waiting for inputs from the last layer, which in this case will be as many as the outputs from the previous layer, 64 in this case, for each neuron and then will be multiplied up to the respective weight of the fully connected layer, and then added them all up together, to give one output, and then passed to activation function, which still is Leaky ReLu.

In the last layer, the neurons will be as many as the labels, 10 in this case, which will be fed with the outputs from the last layer, multiplied to the respective weight and then summed up to be passed to the softmax, which will be used in order to make sure that all the outputs will add up to 1 and therefore make good sense of the data.

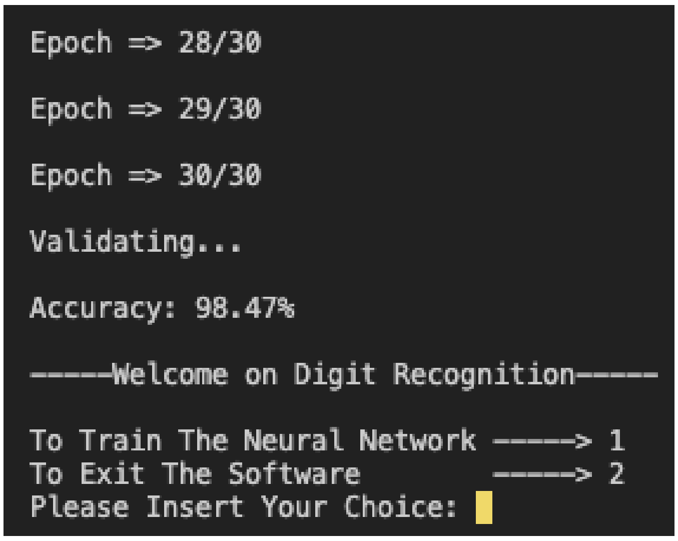
1st Fully Connected Layer ——-->	**Layers.Dense(128, Leaky ReLu)**
2nd Fully Connected Layer —---->	**Layers.Dense(10, Softmax)**

Results

A 2k-folding test has been executed, and the following results have been achieved with the following parameters:

Learning rate = 0.03;
Mini Batch size = 8;
Epochs = 50;

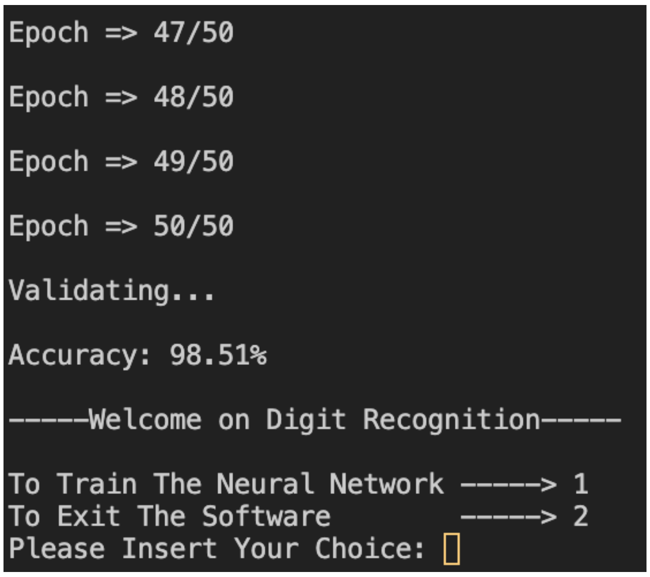
Training Test 1



```
Epoch => 28/30

Epoch => 29/30

Epoch => 30/30

Validating...

Accuracy: 98.47%

-----Welcome on Digit Recognition-----

To Train The Neural Network -----> 1
To Exit The Software        -----> 2
Please Insert Your Choice: 
```

This is the result that has been achieved as a Training dataset the "cw2DataSet1.csv" and for Validation "cw2DataSet2.csv", 98.47% Accuracy with 30 epochs even though has been achieved with 50 Epochs as well, which the result turn out to be more stable than 30 epochs.

Training Test 2



```
Epoch => 47/50

Epoch => 48/50

Epoch => 49/50

Epoch => 50/50

Validating...

Accuracy: 98.51%

-----Welcome on Digit Recognition-----

To Train The Neural Network -----> 1
To Exit The Software        -----> 2
Please Insert Your Choice: 
```

This is the result that has been achieved as a Training dataset the "cw2DataSet2.csv" and for Validation "cw2DataSet1.csv", 98.51% Accuracy.

Bibliography

[Fig 1.0 Cnn Structure] lSaha, S. (2018, December 17). *A Comprehensive Guide To Convolutional Neural Networks—the ELI5 Way.* Medium. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[Fig 1.1 Kernel Structure]Convolutional Neural Networks for Text Classification. (2018, March 31). Convolutional Neural Networks for Text Classification. https://www.davidsbatista.net/blog/2018/03/31/SentenceClassificationConvNets/.

[Fig 1.2 Cross Entropy Formula] Perrotta, P. (2021, March 15). *Grokking the Cross Entropy Loss. This Post Explains the Intuitive… | By Paolo Perrotta | Level Up Coding.* Medium. https://levelup.gitconnected.com/grokking-the-cross-entropy-loss-cda6eb9ec307.

[Fig 1.3 Chain Rule] Chain Rule - Google Search. (n.d.). chain rule - Google Search. https://www.google.com/search?q=chain+rule&oq=chain+rule&aqs=chrome.0.69i59l3j0i433i512l2j69i61l3.3683j0j4&sourceid=chrome&ie=UTF-8.

[Fig 1.4 Gradient Descent] Raschka, S. (n.d.). *Gradient Descent And Stochastic Gradient Descent - Mlxtend.* Gradient Descent and Stochastic Gradient Descent - mlxtend. http://rasbt.github.io/mlxtend/user_guide/general_concepts/gradient-optimization/.