**Handwritten Digit Recognizer**

Benoît MANGEARD, Jixiong LIU, Hong YU

Student at ESIGELEC engineering school

Big Data department

Rouen, FRANCE

**Abstract:**

Thinking about the fact that a machine could understand, in a way, the world as we do, by reproducing our senses into codes, isn’t it exciting. Such algorithms are already being used every day, but now as we are studying predictive modelling, it might be the time for us to explore further and to understand how those models really work. We have decided to make our own version of the Digit Recogniser and therefore try to implement a machine that could ‘see’ human writing. To achieve this task, we will use the well-known MNIST database used in a Kaggle competition. There have been several scientific papers on attempts to achieve the lowest error rate, so we have selected different approaches to implement algorithms into R and Python. Our objective won’t be to try to improve the error rate but to learn and compare algorithms. With the package Keras, an implementation of a recurrent neural network will be coded into Python, also a convolutional neural network, a HRNN and a k-nearest neighbour will be coded into R. By using complex method as well as simplest approach, we could compare time processing and accuracy to be able to list advantages and limits.

1. **Introduction:**

Be able to read digit numbers write by human’s hands can be very useful for company as “La Poste” which is the French post office. Order letters by the postal code destination is a long process for human, but if a machine can do this action, the distribution time process will be improved. To succeed in that task, a machine need to learn how to recognize digit by pictures. Our objective is to create a strong model that allow the machine to learn from pictures.

This project is for us a great opportunity to improve our machine learning skills by learning neural networks. We knew when we have decided to make this project that simplest models will not be the best choice for this problem. In the predictive modelling unit, we learnt how to implement algorithms in R such as linear regression, logistic regression, k-nearest neighbour and others simpler models, but we just had an overview on deep learning which is often use for image recognition. So, we will need to read scientist articles, books and Mooc to be able to understand and then implement such complex algorithms as the convolutional neural network or the recurrent neural network. We will also use others supports to help us achieving our task as articles from Kaggle web site and online tutorials. In fact, MNIST database is overuse since it competition.

Frist, we read the French book “*Comprendre le deep learning: une introduction aux réseaux de neurones »*, which is an introduction of the deep learning concept with examples in JavaScript. Then to go further we read “*Deep Learning by* [*Ian Goodfellow*](https://www.google.com/search?client=firefox-b-ab&q=Ian+Goodfellow&stick=H4sIAAAAAAAAAOPgE-LVT9c3NEwuNjCzNEiqUoJyC8zTs03TTbVkspOt9JPy87P1y4syS0pS8-LL84uyrRJLSzLyiwAB-2HaPgAAAA&sa=X&ved=2ahUKEwiy_cuRxKHfAhVDx4UKHeRxAVQQmxMoATAXegQIBxAH)*,* [*Yoshua Bengio*](https://www.google.com/search?client=firefox-b-ab&q=Yoshua+Bengio&stick=H4sIAAAAAAAAAOPgE-LVT9c3NEwuNjCzNEiqUuLSz9U3SDZOq6qo1JLJTrbST8rPz9YvL8osKUnNiy_PL8q2SiwtycgvAgD7oL_wOwAAAA&sa=X&ved=2ahUKEwiy_cuRxKHfAhVDx4UKHeRxAVQQmxMoAjAXegQIBxAI)*,* [*Aaron Courville*](https://www.google.com/search?client=firefox-b-ab&q=Aaron+Courville&stick=H4sIAAAAAAAAAOPgE-LVT9c3NEwuNjCzNEiqUoJyTSuSKsvTTbVkspOt9JPy87P1y4syS0pS8-LL84uyrRJLSzLyiwCoBSjvPgAAAA&sa=X&ved=2ahUKEwiy_cuRxKHfAhVDx4UKHeRxAVQQmxMoAzAXegQIBxAJ)*”*. This bookhelped us to cover mathematical and conceptual background for deep learning technique. In the meantime, we were trying to implement shared algorithms from the web to cross mathematical concepts and algorithms knowledge.

After all those research, we had decided our approach for this problem. We wanted to use a model learnt from the predictive modelling unit and compare it with more complex methods both using R and Python environments. But this MNIST dataset has not the same size as all example made on the unit at ESIGELEC. It is a real-world problem with a huge amount of data. Computer power needed will probably be very important to train our models. We thought that it might be interesting to use a Big Data software as Hadoop for distributing our calculation, but, as our own laptop get multiple core and enough ram, we forgot this idea to stay focus on models and how accurate they are.

1. **Data description:**

The famous MNIST database (Mixed National Institute of Standards and Technology) is a large dataset of handwritten digits. It contains grey scale images into two data files, a training and a test file. Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. The pixel-value is an integer between 0 and 255. Also, each image represents only one digit between 0 and 9.

In the training file, each column is a pixel except for the first one. Called the ‘label’, this column is the value drawn by the user.

The test file is almost identical to the training one, the only difference is that ‘label’ column doesn’t exist.

The MNIST database contains 42,000 training images and 28,000 testing images.

|  |  |
| --- | --- |
| Figure 1: Overview of the data | Figure 2 : MNIST labels |
| Figure 3: R code for Figure 3 | Figure 4: sample visualisation |

1. **Methods:**

We use four different methods to test the digit recognition dataset and get the strong model usable for the post office problem.

* 1. **K-nearest neighbour KNN**
  + *CORElearn* package in R

We wanted to try a simple model such as the K-nearest neighbour method to be able to compare it with complex model. KNN assumes that the data points are in a metric space. Since the points are in feature space, they have a notion of distance and we can calculate vectors. The number K decides how many data point influence the classification. To find the appropriate K we must test multiple values of it by using the cross-validation technique.

* 1. **Convolutional neural networks**
  + *Keras* package in R
  + *Keras* package in Python

In a convolutional neural network, a convolutional layer can have multiple different convolution kernels (also known as filters), and each convolution kernel slides over the input image and processes only a small number of images at a time. Such a convolutional layer at the input can extract the most basic features in the image.

Each neuron is only connected to a local area of the upper layer. The spatial size of the connection is called the receptive field of the neuron. Also, the current layer uses the same weight and bias for each channel’s neurons in the depth direction, which called weight sharing.

Local connections and weight sharing reduce the number of parameters, greatly reducing training complexity and reducing overfitting. At the same time, weight sharing also gives the convolution network tolerance to translation. So, it seems that CNN is a great choice for image identification.

A CNN require that we make some data preparations, like normalisation and reshaping. Then we can create the model. We use a 6-layer model. The first four layers are the convolutional (Conv2D) layer. It is like a set of learnable filters (32 filters for the two firsts conv2D layers and 64 filters for the last two ones). Each filter transforms a part of the image (defined by the kernel size) using the kernel filter. The kernel filter matrix is applied on the whole image. We can see filters as a transformation of the image.

Another type of important layer is the pooling (MaxPool2D) layer. This layer simply acts as a down sampling filter. It looks at the two neighbouring pixels and picks the maximal value. These are used to reduce computational cost, and to some extent also reduce overfitting.

We must choose the pooling size (i.e. the area size pooled each time). Higher the pooling dimension is, more important the down sampling is. Combining convolutional and pooling layers allow learning more global features of the image. Other layers are used, like dropout and flatten to optimizer converge and make it faster and closer to the true value.

In a convolutional neural network, a convolutional layer can have multiple different convolution kernels also known as filters. Each convolution kernel slides over the input image and then processes only a small number of images. Convolutional layer at the input can extract the most basic features in the image.

Each neuron is connected to a local area of the upper layer. The spatial size of the connection is called the receptive field of the neuron. Also, the current layer uses the same weight and bias for each channel’s neurons in the depth direction, which is called weight sharing.

Local connections and sharing weights reduce the number of parameters, and greatly reducing the training complexity and the overfitting. At the same time, sharing weights also gives the convolution network tolerance to translation. So, CNN is a great choice for image identification.

* 1. **Hierarchical recurrent neural network**
  + *Keras* package in R

HRNN learn across multiple levels of temporal hierarchy over a complex sequence, where long short-term memory (LSTM) units are used.

In a traditional recurrent neural network, during the gradient back-propagation phase, the gradient signal can end up being multiplied a large number of times by the weight matrix associated with the connections between the neurons of the recurrent hidden layer. If the weights in this matrix are small (vanish), it can lead the gradient signal to be so small that learning either becomes very slow or stops working altogether. Conversely, if the weights in this matrix are large, it can lead to a situation where the gradient signal is so large that it can cause learning to diverge.

Therefore, LSTM is introduced into HRNN. A LSTM unit is composed of a memory cell, an input gate, an output gate and a forget gate.

An LSTM cell takes an input and stores it for some period of time. This is equivalent to applying the identity function to the input. Because the derivative of the identity function is constant, when an LSTM network is trained with backpropagation through time, the gradient does not vanish. So, we can find out that the input gate controls the extent to which a new value flows into the cell, the forget gate controls the extent to which a value remains in the cell and the output gate controls the extent to which the value in the cell is used to compute the output activation of the LSTM unit.

In our model, firstly like CNN model we shape the input layer as 28\*28 and **t**ransform RGB values into [0,1] range, then we encode a row of pixels using TimeDistributed Wrapper which will apply a layer to every temporal slice of an input. After that we encodes columns of encoded rows using LSTM layer. Before output we apply a dense layer using softmax as the activation function.

After the implementation of those tree models, we are can compare the result and choose which is the best for our problem.

1. **Results:**
   1. **K-nearest neighbour (KNN)**

The simple model, k-nearest neighbour need a huge training time process, the amount of data and the cross validation make it unusable in our laptop. So, we only train a part of the data. The result wasn’t so bad, by using the first one thousand rows, the model reaches a 0.697 accuracy. And when we added rows, for example the first five thousand, the accuracy reaches 0.8284. So, we can easily think that for all rows, the result will be very good. But there is a cost, the computation needed.

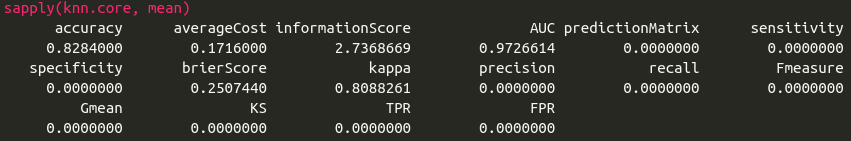
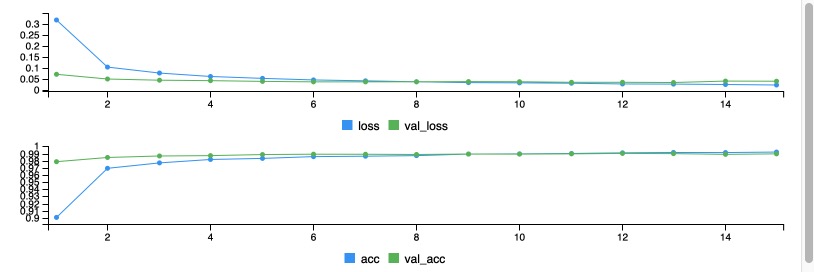


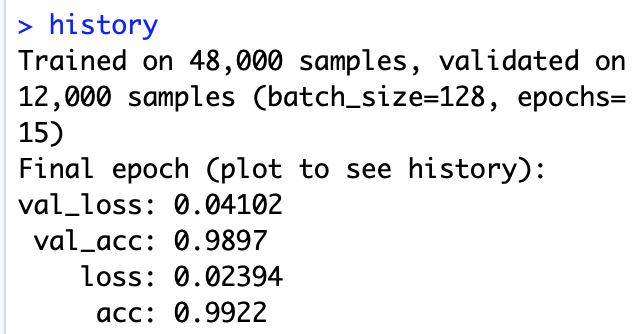
Figure 5: KNN result for 5000 rows

* 1. **Convolutional neural network (CNN)**

The CNN is good for image recognition and its result is impressive.

In R the training process uses 2184s for 15 epochs so 145.6s/epoch. Finally, we got an accuracy of 0.9913.

*Figure 6: CNN loss and val\_loss graph*



*Figure 7: CNN R result*

In Python, the result is similar. However, in this case we don’t separate data into serval epochs and choose adam as the optimizer. The accuracy falls to 0.9357 and the training time of one epoch is 196s.

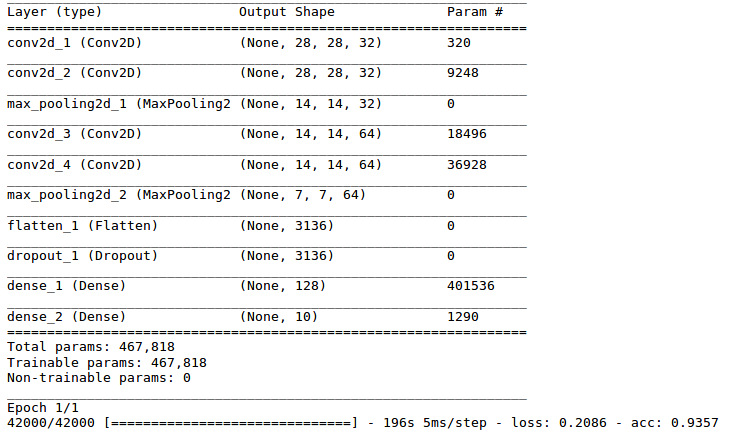


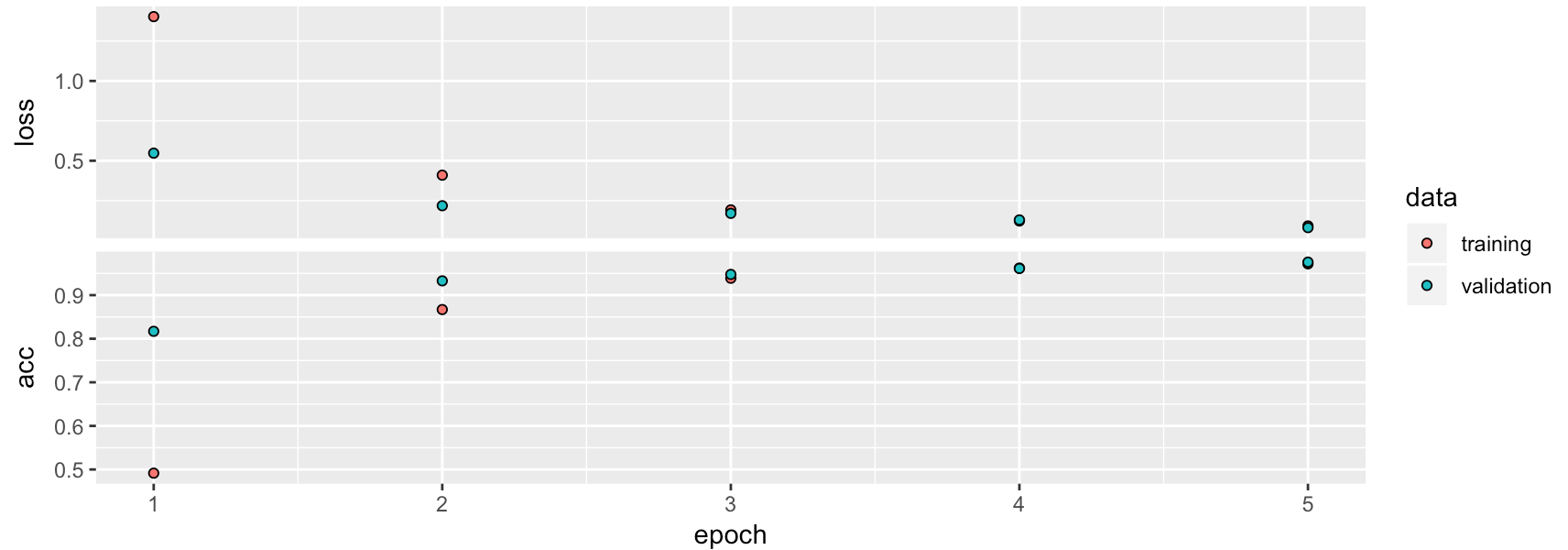
Figure 8: RNN Python result

But CNN still has some inconveniences. It is not easy to understand, we can't observe the revolution in the hidden layer and it also needs a large calculate power.

* 1. **Hierarchical recurrent neural network (HRNN)**

In HRNN we took 5 epochs which costed totally 3863s. The final accuracy is 0.9718, very high but still lower than the CNN model.

The HRNN should have a better performance in continue handwritten problem.



*Figure 9: CNN loss and val\_loss graph*

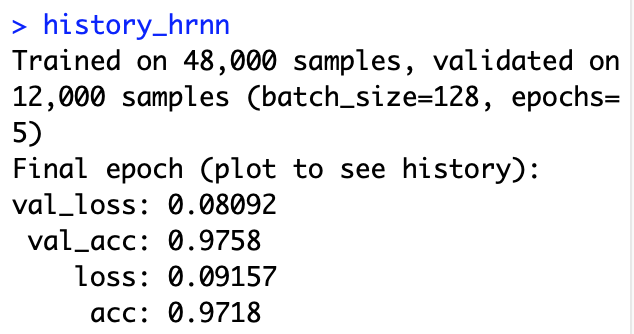


Figure 10: HRNN accuracy

* 1. **Comparison of different models:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **KNN** | **CNN in R** | **CNN in Python** | **HRNN** |
| **Trained sample** | 5000 | 48000 | 42000 | 48000 |
| **Training time per epoch** | / | 145.6s | 196s | 772.6s |
| **Optimizer** | / | Adadelta | Adam | Adadelta |
| **Accuracy** | 0.8284 | 0.9922 | 0.9357s | 0.9718 |

1. **Conclusion:**

This project was for us an opportunity to learn neural networks methods to create a model which can be use in a real-world problem, such as the postal office distribution. By using four methods, we have an overview of the pros and cons of each of them. Some methods need a huge amount of calculation during the training to get a good accuracy. The MNIST dataset is huge but it exists bigger database, even a new version of the MNIST dataset called the EMNIST, which contains 240,000 training images, and 40,000 testing images of handwritten digits and characters. In this situation, our laptops could have real difficulties. We also learn that model can be difficult to understand and so, to explain.

1. **References:**

* Emiru Tsunoo, Peter Bell, Steve Renals, **“Hierarchical Recurrent Neural Network for Story Segmentation”**, in *INTERSPEECH 2017*, August 20–24, 2017
* Jean-Claude Heudin, **“Comprendre le deep learning: une introduction aux réseaux de neurones”,** 2016
* Ian Goodfellow, Yoshua Bengio, Aaron Courville, **“Deep Learning”**, 2015
* Gareth James, Daniela Witten Trevor Hastie, Robert Tibshirani, **“An Introduction to Statistical Learning – with Applications in R”**, 2013
* Santiago Fernandez, Alex Graves and Jurgen Schmidhuber, **“Sequence Labelling in Structured Domains with Hierarchical Recurrent Neural Networks”**, in *proc. 20th int. Joint conf. On artificial intelligence*, 2007
* Brandon Rohrer, “**How do Convolutional Neural Networks work?**”, *[ http://brohrer.github.io/how\_convolutional\_neural\_networks\_work.html?fbclid=IwAR1KbKb-FN0lqxr4XVN\_mt0Ca7MIR\_ZcyENhQDd1qNZ4DHQ6uhbttMmaoRQ]*
* Sambit Mahapatra, “**A simple 2D CNN for MNIST digit recognition**”, in *Towards Data Science*, 2018, *[https://towardsdatascience.com/a-simple-2d-cnn-for-mnist-digit-recognition-a998dbc1e79a?fbclid=IwAR1bTUzXHK9gFLnjShf2t93yeI2A70Hiwk7HUbqsBux1VbkDJVsA4v5AaFY]*
* “**LSTM Networks for Sentiment Analysis**”, *[http://deeplearning.net/tutorial/lstm.html]*
* **“Keras”,** *[https://keras.rstudio.com/index.html]*
* JJ Allaire**, “Keras for R”,** 5 September 2017, *[https://blog.rstudio.com/2017/09/05/keras-for-r/]*