Handwritten digits recognition



**FOULON Alex**

**GAUTIER Erwan**

**May 22, 2024, ESIEE Paris**

# Introduction

The field of image classification has witnessed significant advancements over the past few decades, driven largely by the development of deep learning techniques. Among these techniques, both multilayer perceptrons (MLPs) and convolutional neural networks (CNNs) have become fundamental tools. This project aims to compare the performance of these two types of neural networks in the context of image classification, using the MNIST dataset as a benchmark.

The MNIST dataset, introduced by Yann LeCun and colleagues in the late 1990s, is a widely used dataset consisting of 60,000 training images and 10,000 testing images of handwritten digits, each sized 28x28 pixels. It has become a standard for evaluating the performance of image classification algorithms due to its simplicity and the variety of methods tested on it over the years.

In this study, we will implement and evaluate two models: a multilayer perceptron without convolutional layers and a convolutional neural network. By comparing these models, we aim to understand the strengths and weaknesses of each approach and provide insights into their applicability for different types of image classification tasks. This comparison will help in determining which architecture offers better performance in terms of accuracy and computational efficiency when applied to the MNIST dataset.

# First approach : The Multilayer Perceptron

## Description of the model

Our first version of the model is a multilayer perceptron (MLP). We built it using the Keras python library. Here is the part where the model is defined :

num\_classes = 10

input\_shape = (28, 28, 1)

model = keras.Sequential(

[

    keras.Input(shape=input\_shape),

    layers.Flatten(),

    layers.Dense(512, activation="relu"),

    layers.Dropout(0.5),

    layers.Dense(256, activation="relu"),

    layers.Dropout(0.5),

    layers.Dense(num\_classes, activation="softmax"),

])

It consists of 7 layers, each one having its own use :

1. **Input Layer** :This layer defines the shape of the input data, which in this case is an image of size 28x28 pixels with 1 color channel (grayscale).
2. **Flatten Layer** : This layer flattens the 2D image (28x28) into a 1D vector (784 elements). Dense layers expect 1D input. Flattening converts the 2D pixel grid into a format suitable for dense layers, enabling the network to process the image as a single vector.
3. **First Dense Layer** : This layer consists of 512 neurons, each fully connected to the input features. Dense layers are the core components of an MLP. The ReLU (Rectified Linear Unit) activation function introduces non-linearity, allowing the network to learn complex patterns. With 512 neurons, this layer has a sufficient capacity to capture and process detailed features from the input.
4. **First Dropout Layer** : This layer randomly sets 50% of the input units to 0 at each update during training time. Dropout is a regularization technique used to prevent overfitting. By randomly dropping units, the network becomes less sensitive to specific neuron weights, improving it’s generalization capabilities.
5. **Second Dense Layer** : This layer consists of 256 neurons, each fully connected to the previous layer's output. Like the first dense layer, this layer further processes and refines the features learned. With fewer neurons than the previous layer, it acts as a funnel, focusing on the most important features.
6. **Second Dropout Layer** : Similar to the first dropout layer, it sets 50% of the input units to 0 during training, further improving the generalization capabilities of the model.
7. **Output Dense Layer** : This layer consists of 10 neurons in this case (one for each digit), each one representing a class. The softmax activation function converts the output to a probability distribution over the classes, which is essential for classification tasks. The network outputs a probability for each digit, and the digit with the highest probability is chosen as the predicted class.

After the model is built, model.summary() gives us this output, which describes the architecture of the model :

┌─────────────────────────────────┬────────────────────────┬───────────────┐

│ Layer (type)                    │ Output Shape           │       Param # │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ flatten (Flatten)               │ (None, 784)            │             0 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ dense (Dense)                   │ (None, 512)            │       401,920 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ dropout (Dropout)               │ (None, 512)            │             0 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ dense\_1 (Dense)                 │ (None, 256)            │       131,328 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ dropout\_1 (Dropout)             │ (None, 256)            │             0 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ dense\_2 (Dense)                 │ (None, 10)             │         2,570 │

└─────────────────────────────────┴────────────────────────┴───────────────┘

 Total params: 535,818 (2.04 MB)

 Trainable params: 535,818 (2.04 MB)

 Non-trainable params: 0 (0.00 B)

We can see the size of each layer, along with the number of parameters that are gonna be fine-tuned during the training process.

## Results and performance

Let’s take a look at the confusion matrix of the predictions :

Une image contenant capture d’écran, texte, Rectangle, carré

Description générée automatiquement

The values along the diagonal represent the number of correct predictions for each class, and values off the diagonal represent misclassifications. We can observe that the majority of the predictions are correct as most values are along the diagonal. Misclassifications are minimal and scattered, indicating strong performance.

Now, we will analyse the evolution of the precision and the loss during training and validation :

Une image contenant texte, Tracé, diagramme, ligne

Description générée automatiquement

Training Accuracy is increasing steadily and reaches around 98%. Validation Accuracy starts high and remains slightly above training accuracy, indicating good generalization.

Training Loss decreases rapidly initially and then slowly continues to decrease, approaching zero. Validation Loss is decreasing as well, reaching very low values.

Here is the classification report table :

              precision    recall  f1-score   support

           0       0.99      0.99      0.99       980

           1       0.99      0.99      0.99      1135

           2       0.98      0.98      0.98      1032

           3       0.98      0.98      0.98      1010

           4       0.99      0.98      0.98       982

           5       0.98      0.98      0.98       892

           6       0.98      0.99      0.98       958

           7       0.99      0.98      0.98      1028

           8       0.97      0.98      0.98       974

           9       0.98      0.99      0.98      1009

* Precision is high across all classes (around 0.98-0.99).
* Recall measures the ability to capture all positive samples, and is high across all classes (around 0.98-0.99).
* F1-Score is the harmonic mean of precision and recall. Here, it indicates strong balance between precision and recall for all c lasses (around 0.98-0.99).

To summarize the results, the confusion matrix indicates high accuracy with minimal misclassifications. The graphs show effective learning and good generalization with minimal overfitting. The classification report table confirms strong performance across all metrics.

Overall, the model performs excellently on the MNIST dataset, achieving high accuracy and strong generalization capabilities.

# Multilayer Convolutional Neural Network

## Description of the model

Our second version of the model is a convolutional neural network (CNN). We built it using the Keras Python library. Here is the part where the model is defined:

num\_classes = 10

input\_shape = (28, 28, 1)

model = keras.Sequential(

[

    keras.Input(shape=input\_shape),

    layers.Conv2D(32, kernel\_size=(3, 3), activation="relu"),

    layers.MaxPooling2D(pool\_size=(2, 2)),

    layers.Conv2D(64, kernel\_size=(3, 3), activation="relu"),

    layers.MaxPooling2D(pool\_size=(2, 2)),

    layers.Flatten(),

    layers.Dropout(0.5),

    layers.Dense(num\_classes, activation="softmax"),

])

It consists of 8 layers, each one having its own use, with some similarities with the previous model :

* **Input Layer** : This layer defines the shape of the input data, which in this case is an image of size 28x28 pixels with 1 color channel (grayscale).
* **First Convolutional Layer** : This layer consists of 32 filters, each of size 3x3. The convolution operation scans the image with these filters to detect local patterns such as edges. The ReLU activation function introduces non-linearity, enabling the network to learn complex features.
* **First MaxPooling Layer** : This layer reduces the spatial dimensions of the feature maps produced by the previous convolutional layer. Using a pool size of 2x2, it downsamples the input, reducing the number of parameters and computational load, which helps in controlling overfitting.
* **Second Convolutional Layer** : This layer consists of 64 filters of size 3x3. It further processes the downsampled feature maps, allowing the network to learn more abstract and complex features.
* **Second MaxPooling Layer** : Similar to the first max-pooling layer, this layer reduces the spatial dimensions of the feature maps generated by the second convolutional layer. This additional downsampling helps in further reducing the model’s computational complexity and controlling overfitting.
* **Flatten Layer** : This layer flattens the 3D feature maps (output from the last max-pooling layer) into a 1D vector, enabling the network to process the entire image as a single vector.
* **Dropout Layer** : This layer randomly sets 50% of the input units to 0 at each update during training time, improving its generalization capabilities.
* **Output Dense Layer** : This layer consists of 10 neurons (one for each digit), each one representing a class. The softmax activation function converts the output to a probability distribution over the classes. The digit with the highest probability is chosen as the prediction.

model.summary() describes the architecture of the model :

┌─────────────────────────────────┬────────────────────────┬───────────────┐

│ Layer (type)                    │ Output Shape           │       Param # │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ conv2d (Conv2D)                 │ (None, 26, 26, 32)     │           320 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ max\_pooling2d (MaxPooling2D)    │ (None, 13, 13, 32)     │             0 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ conv2d\_1 (Conv2D)               │ (None, 11, 11, 64)     │        18,496 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ max\_pooling2d\_1 (MaxPooling2D)  │ (None, 5, 5, 64)       │             0 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ flatten (Flatten)               │ (None, 1600)           │             0 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ dropout (Dropout)               │ (None, 1600)           │             0 │

├─────────────────────────────────┼────────────────────────┼───────────────┤

│ dense (Dense)                   │ (None, 10)             │        16,010 │

└─────────────────────────────────┴────────────────────────┴───────────────┘

 Total params: 34,826 (136.04 KB)

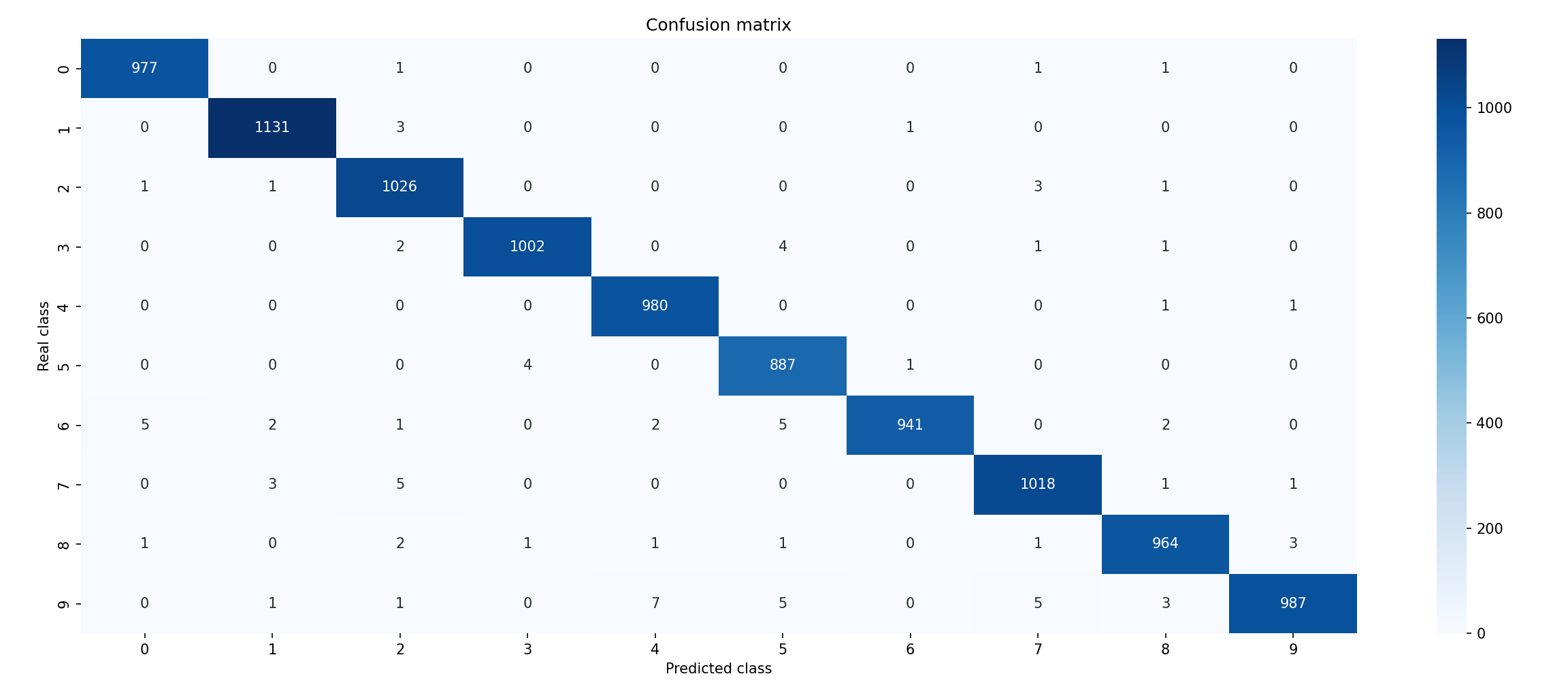
 Trainable params: 34,826 (136.04 KB)

 Non-trainable params: 0 (0.00 B)

We can see the size of each layer, along with the number of parameters that are gonna be fine-tuned during the training process. Note that there are a lot less trainable parameters in this model.

## Results and performance

Let’s take a look at the confusion matrix of the predictions :



As with the first model, we can observe that the majority of the predictions are correct as most values are along the diagonal. Misclassifications are minimal and scattered, indicating strong performance.

Now, we will analyse the evolution of the precision and the loss during training and validation :

Une image contenant diagramme, ligne, Tracé, capture d’écran

Description générée automatiquement

Training Accuracy is increasing steadily and reaches 99%. Validation Accuracy starts high and remains slightly above training accuracy, indicating good generalization.

Training Loss decreases rapidly initially and then slowly continues to decrease, approaching zero. Validation Loss is decreasing as well, reaching very low values.

Here is the classification report table :

Classification report:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00       980

           1       0.99      1.00      0.99      1135

           2       0.99      0.99      0.99      1032

           3       1.00      0.99      0.99      1010

           4       0.99      0.99      0.99       982

           5       0.99      0.99      0.99       892

           6       1.00      0.99      0.99       958

           7       0.99      0.99      0.99      1028

           8       0.99      0.99      0.99       974

           9       0.99      0.99      0.99      1009

* Precision is near perfect across all classes (0.99 - 1.00).
* Recall is near perfect across all classes (0.99 - 1.00).
* F1-Score indicates strong balance between precision and recall for all classes (0.99 - 1.00).

To summarize the results, the confusion matrix indicates high accuracy with minimal misclassifications. The graphs show effective learning and good generalization with minimal overfitting. The classification report table shows near perfect performance across all metrics.

Overall, the model performs excellently on the MNIST dataset, achieving high accuracy and strong generalization capabilities.

# Training and prediction time comparaison

In this section, we compare the training times of the two different neural network architectures. The comparison focuses on the relationship between accuracy and loss over time during the training process. Then, we will compare prediction time.

## Multilayer Perceptron :

Une image contenant texte, diagramme, Tracé, ligne

Description générée automatiquement

The training accuracy of the MLP rapidly increases within the first 5 seconds, reaching around 96%. The accuracy continues to improve gradually, stabilizing at approximately 98% after 20 seconds. Validation accuracy starts high and consistently remains slightly above the training accuracy.

The training loss decreases sharply within the first few seconds, from around 0.4 to below 0.1. The loss continues to decline slowly, approaching near-zero values by the end of the training period. Validation loss remains lower than training loss.

## Convolutional Neural Network :

Une image contenant ligne, diagramme, Tracé, texte

Description générée automatiquement

The CNN's training accuracy also shows a rapid increase, but it takes longer to stabilize compared to the MLP.It reaches around 97% within the first 20 seconds and gradually approaches 98% by 50 seconds. Validation accuracy starts high and remains slightly above training accuracy, similar to the MLP.

The training loss for the CNN drops quickly from 0.35 to around 0.1 within the first 15 seconds. The loss continues to decrease steadily, nearing zero by the 50-second mark. Validation loss stays consistently below the training loss as well.

## Comparative Insights

The MLP converges faster than the CNN, reaching its near-optimal accuracy and loss values in about 20 seconds, whereas the CNN takes around 50 seconds. This difference is expected because MLPs generally have simpler architectures compared to CNNs, which involve more complex computations due to convolutional and pooling operations.

Both models show good generalization, with validation accuracy and loss closely tracking their respective training metrics. The validation metrics for both models are slightly better than the training metrics, indicating effective regularization and no significant overfitting.

The CNN takes longer to train due to its increased computational complexity from convolutional layers, which process spatial hierarchies in the data. Despite the longer training time, CNNs are typically more effective for image data due to their ability to capture spatial features, as evidenced by the higher final accuracy compared to the MLP.

The MLP model trains faster, achieving high accuracy and low loss in a shorter amount of time. However, the CNN, while slower to train, ultimately provides better performance in terms of final accuracy.

# Conclusion

# References