SOFTCOMPUTING

FIRST SUBJECT REPORT

Translation system based on a Multilayer Perceptron: pictures of characters and digits into a Braille alphabet

Pablo Fernández Ivars - 287764

Group 2

INDEX

DATA RECOLLECTION AND PREPARATION 2

MULTILAYER PERCEPTRON ARCHITECTURE 2

TRAINING AND TESTING 3

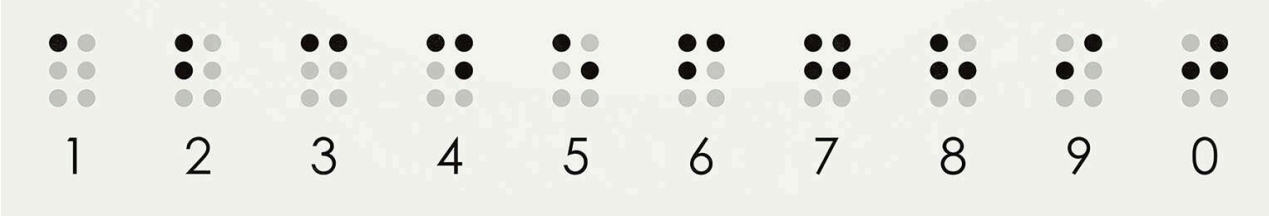
BRAILLE TRANSLATOR 4

MODEL DEPLOYMENT AND RESULTS 4

CONCLUSION 6

INTRODUCTION

Braille is a tactile writing system used by visually impaired people. It encodes characters into patterns of raised dots arranged in a 2x3 cell. In the case of numbers, there exists the following standardized mapping between digits (0-9) into specific Braille patterns:



With this project we aim to develop a translation system that accurately converts images of handwritten or printed digits into their corresponding Braille representation, implementing image preprocessing techniques and a Multilayer Perceptron neural network (MLP).

DATA RECOLLECTION AND PREPARATION

For our dataset we used the chars74k dataset from the Kaggle website, which contains English character images. With the `def process_image(route_image),` each image undergoes the same preprocessing:

- **Resizing**: Adjusts the input image to 28 x 28 pixels
- **Grayscale Conversion**: Reduces the dimensionality of the data, lowering computational requirements for processing the data
- **Binarization**: Simplifies the data by converting it to black-and-white pixels using a certain threshold
- **Normalization**: Scales pixel values to the [0, 1] to improve the dataset's convergence

MULTILAYER PERCEPTRON ARCHITECTURE

The Multilayer Perceptron is a type of feedforward neural network consisting of multiple layers of interconnected networks. For our task of digit recognition, the proposed architecture of the network is the following:

Input layer	Hidden layers	Output layer
Each 28 x 28 image is converted to a 1 x 784 vector for processing	The first dense layer is composed by 128 neurons with ReLU activation, which introduces non-linearity to the model and enables the network to learn complex patterns	Composed of 10 neurons (one for each digit), with softmax activation so the output are probabilities

TRAINING AND TESTING

First, we check if the model already exists. If it exists, it loads it. If it does not exist, it creates a new Sequential model based on the architecture we previously defined.

```
model = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
```

In order to train the model, we proceed to the splitting of the data into a training set (80% of the data) and a test set(20%). To continue with the training, we reshape the data to 28 x 28 x 1 for input to the model, and we compile the model:

- `optimizer='adam'`: Works well for problems like image classification
- `loss='sparse_categorical_crossentropy'` : computes the difference between the true labels and the predicted probabilities, which is ideal for multi-class classification where the true labels are integers
- `metrics=['accuracy']`

The training is then executed for 10 epochs, meaning that the dataset is passed through the model 10 times, and dividing the dataset into 32 batches. As the model processes more batches and epochs, it learns better weights, improving accuracy. Finally, we save our trained model. By the end of the training, the model should have learned to classify digits with high accuracy, ready to be used for digit recognition and Braille translation.

BRAILLE TRANSLATION

Before translating to Braille, an image of a digit is preprocessed as it is explained in the 'Data Recollection and Preparation' part. Then, the resulting `pixels` array is reshaped to include the batch dimension, ensuring its compatibility with the model's input structure. Once it is fully preprocessed, we use our trained model to classify the digit. The model outputs a probability distribution over the 10 classes of the given image. For example, the output might look like this:

```
[0.01, 0.03, 0.02, 0.05, 0.87, 0.01, 0.01, 0.0, 0.0, 0.0]
```

In this case, the model is 87% confident that the digit is 4. Finally, we use the *argmax* function so that it selects the index with higher probability. In the case of the previous example, the `predicted_number` is 4.

Once the digit is predicted, it is mapped to its corresponding Braille representation using the following braille dictionary.

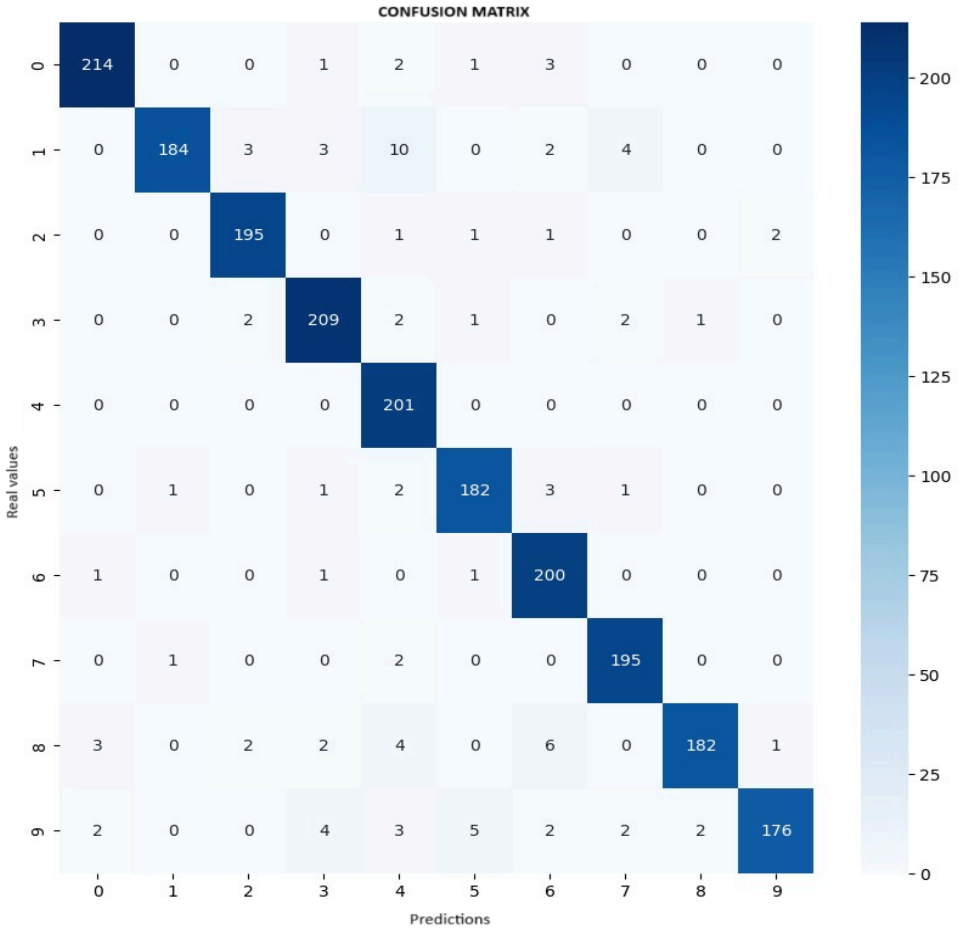
```
braille_dict = {
    0: "⠚",  # Braille for 0
    1: "⠁",  # Braille for 1
    2: "⠃",  # Braille for 2
    3: "⠉",  # Braille for 3
    4: "⠙",  # Braille for 4
    5: "⠑",  # Braille for 5
    6: "⠋",  # Braille for 6
    7: "⠛",  # Braille for 7
    8: "⠓",  # Braille for 8
    9: "⠊",  # Braille for 9
```

The `predicted_number` Braille is used as a key to retrieve its corresponding Braille character from the dictionary. For instance, following the previous case, the result would be ⠙

MODEL DEPLOYMENT AND RESULTS

To ensure an enhanced performance of the model, we used many different images of digits, including distorted and cropped versions of these:

To obtain a detailed view of the model's performance, we created the following confusion matrix, which compares the true labels `y_test` with the predicted labels `y_pred_classes`.



As we can see from the table offer really good results:

- **Accuracy = 95.19%**

Metrics per class:

Class	Precision	Sensitivity	F1-Score
0	97.27%	95.96%	96.61%
1	100%	89.32%	94.36%
2	97.01%	97.50%	97.26%

3	93.72%	96.31%	95.00%
4	88.94%	100%	94.15%
5	92.86%	95.29%	94.06%
6	94.79%	98.52%	96.62%
7	91.98%	97.99%	94.89%
8	98.91%	91.00%	94.79%
9	98.32%	89.80%	93.87%

The high accuracy indicates that the neural network learned robust patterns and features from the training data, enabling it to generalize well to unseen samples-

According to the class-specific, we can see that it was able to achieve **perfect sensitivity in class 4**, which means that the model identified all instances of 4 correctly and all precision values range between **88.94% (class 4)** and **100% (class 1)**, demonstrating that the model rarely misclassified samples as an incorrect digit. F1-score ranges from **93.87% (class 9)** to **97.26% (class 2),** which shows a balanced performance.

Moreover, the high values along the diagonal indicate that the model classifies most samples correctly and that misclassifications, when present, tend to occur between similar digits such as **1** and **7**, **5** and **6.**

CONCLUSION

Our proposition demonstrated reliable recognition of all digits, with only minor inconsistencies in specific classes. The lightweight MLP architecture delivered these results in an efficient way, making it suitable for deployment in resource-constrained environments. In addition, this design allows for easy expansion to include alphabets or symbols in future versions.