REAL TIME HANDWRITTEN DIGIT RECOGNITION BASED ON CONVOLUTIONAL NEURAL NETWORKS (CNN)

F.Lousada Fernández, M. Martínez Montaña

Abstract— **The objective of this project is to recognize handwritten digits based on real time image acquisition from a camera. Currently, several algorithms are suitable for this classification task, such as Bag of Words approach, Decision Trees or Artificial Neural Networks (ANN). However, the most powerful image recognition tools are based on Convolutional Neural Network (CNN), which is going to be the approach chosen for this project. For training and testing MNIST dataset was used. The final net achieved a 98.54% train accuracy and 98.40% test accuracy. For real time segmentation, blob analysis was carried out, and for digit preprocessing, resize and Gauss filter were used to adequate each detected digit to CNN input. The final model was tested in a 1280x720 Webcam with sample time of 1 sec.**

Index Terms— digit recognition, Convoulutional Neural Nework, deep learning, MNIST dataset.

I. INTRODUCTION

HANDWRITING recognition has been one recurrent problem of image classification. There is an increasing demand on solving this task in some different fields, such as mobile applications for character detection, camera-based vigilance with car plates recognition, and more generally for handrwritten-to-digital document transcription.

Among all techniques Computer Vision has for classification, Convolutional Neural Networks (CNN) are the most powerful tool. Some researches have achieved much higher accuracy with CNN algorithms than with any other previous state-of-art solution. There are some developed nets with high number of layers that classify thousands of objects with error rates lower than 1%, such as LeNet, AlexNet or GoogleNet[1]. But the problem tried to be solve here appears to be simpler, since the algorithm has to deal only with 10 different classes, one per digit.

The idea of this project is to develop a model able to receive frame-by-frame video information, and recognize all the handwritten digits on it, locating its position at first and then recognizing them at the end. In order to accomplish the best possible result, this project has been divided in several tasks, as shown below [2].

1. **Training the net.** CNN are supposed to be trained with the largest amount of data available to get the best results (the larger the training set, the better the algorithm will learn the features). The algorithm is trained with the popular MNIST dataset [3], which collects 70 000 handwritten digits of 28-by-28 grayscale pixels. From those images, 60 000 are used for training the net and 10 000 for testing. As will be seen, this task depends also on some parameters that need to be set beforehand, so several iterations were tried to improve the accuracy.

2. **Digit segmentation** The final model should be able to detect multiple digits per image. This segmentation will be implemented with Matlab blob analysis tool. The net is going to be tested with all blobs detected, assuming most of all will be written numbers.

3. **Simple digit pre-processing**. In order to make all detected digits suitable for CNN input some processing is needed. Each detected blob will be resized, binarized and filtered for this task.

4. **Testing the net**. Final task will be pass each digit through the net in order to get its prediction. Final image will be plotted only with those predictions that exceed a given probability threshold, to filter those junk detections that are not digits. To get fancier results, bounding box and labels will be also added to final image.

The application runs in a looped Matlab script, repeating tasks 2 to 4, so that they could be implemented per frame in a real time video.

II. CNN APPROACH

Convolutional Neural Networks (CNN) are based on the same principles as regular Artificial Neural Networks (ANN): both use similar architecture (each layer output is computed by weighing each input variable and sum all with a bias term), and the same rules to update parameters epoch by epoch (back propagation gradient descent). In fact, final layers of CNN are usually regular ANN layers.

However, CNNs are intended to deal with images as inputs, so that the main problem to solve is the high dimensionality. By definition, the most important assumption made here is that image points situated in similar regions contain closer information than with more distant ones. That is the reason why convolution is used to capture most important features from training images, instead of weighting every single pixel with different values [1][2][4].

A. CNN architecture

CNN architecture is based on superposition of different types of layers, so each one process as input the previous layer's output. Among all of them, the layers used in this project are the following [4]:

1. ***Convolutional Layer (CL).*** They compute the core task of the algorithm. Each one is defined by the size of kernels (or filters) used, with three dimensions – height, width and depth–, the number of filters used and two more values: stride and padding. The first represent how the kernel is slipped along the image (if stride equals to 2, the filter is convolved jumping 2 by 2 pixels, if equals to 1, is convolved on every pixel). The second one allows adding rows and column of zeros when convolving the filter at image boundaries. CLs receive an image as input, and convolves it with each filter, resulting a size-reduced feature map. The number of feature maps returned is the same of filters has the layer, and the output size depends on Equation 1. By tuning those values, the desired output size and number can be achieved.

$$size(out) = \frac{size(in) - size(kernel) + 2 \cdot padding}{stride} + 1 \qquad (1)$$

2. ***ReLu Layer (RELU).*** It applies element wise the function $f(x) = \max\{0, x\}$, so that every negative value is set at zero. It leaves the size of the input unchanged, but without negative values.

3. ***MaxPooling Layer (MAXP)***. By using the same idea of Convolutional layers, it performs a down sampling by sliding a rectangular window over the input image, and taking the maximum value. The output size follows the same Equation 1, but with zero-padding.

4. ***Fully Connected Layer (FC).*** Each one composed by a given number of neurons, it weights every input pixel value, computes a sum with a bias and applies an activation function to the result. The function used in this project is *softmax* function**.** The output size will be directly the number of neurons used, and the output scores of the last fully connected layer represent the probability of belonging to a certain class.

B. CNN training procedure

When trained, the CNN algorithm initializes the train parameters (CLs filter values and FC weights and biases) usually to random values. To update those parameters, gradient descent back-propagation algorithm is used with mean square error (MSE) used as cost function [1].

When a training image batch is passed through the net, the final scores are computed for each sample, MSE is computed with respect to its desired label scores. At the end, kernel values, weights and biases are updated such that they minimize the cost function. The training continues until desired error rate (misclassified percentage) is achieved.

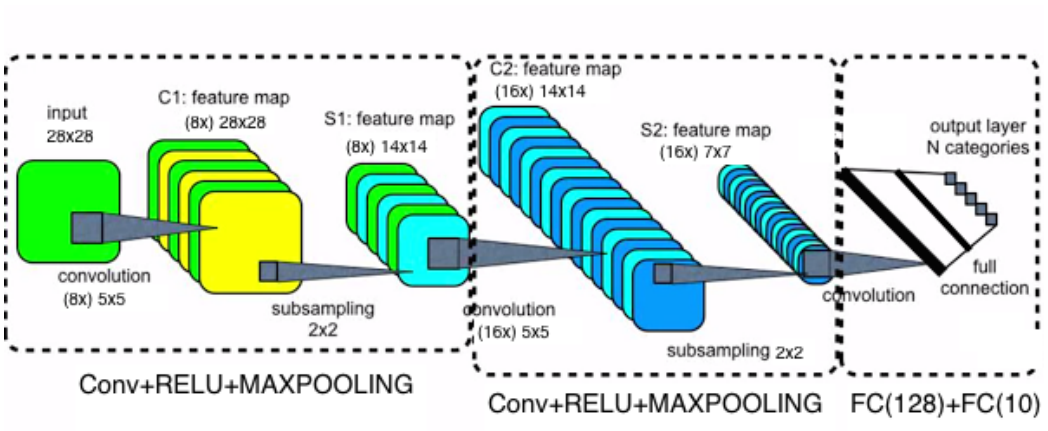

Figure 1. CNN architecture used.

III. Technical Implementation

A. Training CNN on MNIST dataset [2][5]

The problem of handwriting-digit classification is relatively less complex than others, so the architecture used has a less volume of layers than for some other tasks. The architecture used is shown at Figure 1. Basically, two CL-RELU-MAXP blocks are concatenated, so at the end two FC separated by a RELU compute the scores. First CL is composed by eight 5-by-5 kernels, and the second one by sixteen 5-by-5 kernels. Both padding and stride are set to 2 and 1 respectively. All MAXP layers use a 2-by-2 window with stride 2. The first FC is defined by 128 neurons, and the last one needs to be composed by 10, so the output is a vector of 10 class probabilities.

The net is trained with the MNIST train set, composed by 60000 digit samples. Maximum number of epochs is set to 5, the learning rate is initialized to 5e-4, and the batch size is set to 128 samples. Figure 2 shows the error rate evolution along every iteration.

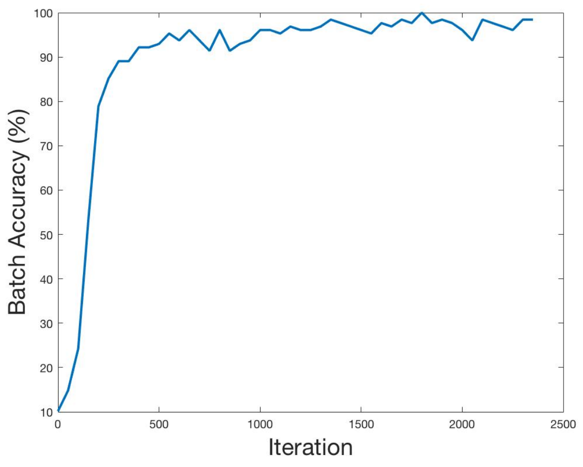

Figure 2. Batch accuracy evolution along training iterations.

Once the CNN is trained, it is tested again on the whole train set (instead batch-by-batch) and also on the test set. The accuracies achieved are 98.54% at the first case and 98.40% on the test set.

B. Digit segmentation

In order to allocate possible digits, blob analysis is carried out. The minimum blob area is initialized to 100 pixels. Frames are initially adaptive binarized with threshold 0.7. The result returns the location and size of each blob by bounding box assignation.

In order get similar sizes as training data, the bounding box margins are a bit expanded with additional vertical and horizontal pixels, with this value set at 20 pixels. The maximum number of blobs detected is let as free parameter, with values between 30 and 60 used for testing.

C. Digit pre-processing

The purpose of preprocessing is to test the CNN with data in most similar conditions as the data it was trained with. Each detected region by blob analysis is processed inside each bounding box limits, and the result extracted as individual images to be tested in the CNN.

At first, each image is resized to 28x28 grayscale. Then pixel values are transformed to *uint8* and convolved with a 5x5 Gaussian filter with sigma 0.9. Finally, the image is normalized with values between 0 and 255. Some snapshots of both digit segmentation and preprocessing are shown at Figure 3.

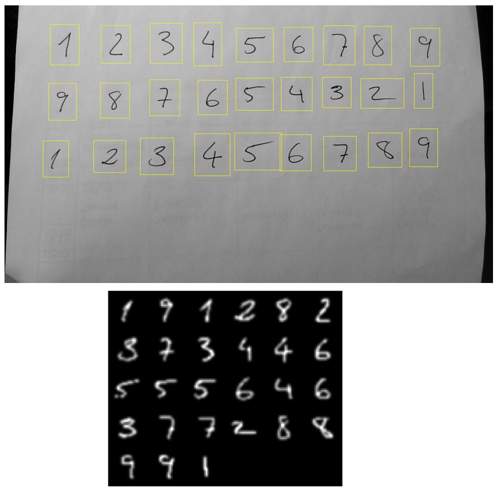

Figure 3. Example snapshot of detected blobs (top), digits extracted after preprocessing (bottom).

D. Testing CNN

The last step is to test the CNN with the extracted digits. The net is looped along every sample, so a probability vector is assigned to each one. In order to remove those "bad predictions" –those blobs that do not correspond to actual digits, and also those samples which the algorithm is not sure enough about–, a threshold parameter is let as free. If this threshold is set to 0.90 for instance, the script will only show those classifications with at least 90 % chance of belonging to its actual class. The final step is to show the resulting image, with each bounding box labeled as the predicted class. This result is shown at Figure 4.

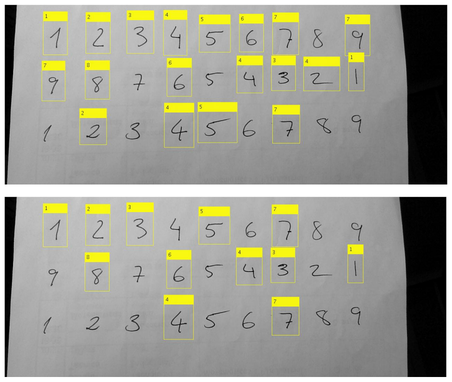

Figure 4. Result image with reliability threshold 0.6 (top) and 0.9 (bottom).

IV. Conclusions

This project has shown the power of CNN algorithm based on digit recognition. Although the model is not so complex it achieves an accuracy of 98.40% on MNIST test set.

However, the real performance is noticeable when the net has to process every video frame, dealing with variations of luminance, rotation and other aspects of camera acquisition. But despite all of this, the model works reasonably well as shown at Figure 4.

Several changes could be added to improve results. First of all, blob analysis only looks for blobs and they may be numbers or not, so a pre-number detection could be implemented (such as corner detector) in order not to test the CNN with fake images. Also more convolutional layers could be added to improve the accuracy.

References

[1] M. Nielsen, "*Neural Networks and Deep Learning*", 2017. Available: http://neuralnetworksanddeeplearning.com/

[2] X. Yang, J. Pu, "*Multi Digit Recognition using Convolutional Neural Networks on Mobile*". Stanford University, Stanford CA 9435

[3] Y. LeCun, C. Cortes, C. J. C. Burges, "*The MNIST database of handwriting digits*". Available: http://yann.lecun.com/exdb/mnist/

[4] F. Li, J. Johnson, S. Yeung, "*Convolutional Neural Networks for visual recognition*", Course CS231, Spring 2017, Stanford University, Stanford CA 9435 Available: http://cs231n.github.io/convolutional-networks/#overview

[5] J. Brownlee, "*Handwritting Digit Recognition using Convolutional Neural Networks in Python with Keras*", 2016. Available: http://machinelearningmastery.com/handwritten-digit-recognition-using-convolutional-neural-networks-python-keras/