image & video understanding

Visual Calculator

Project Report

IVU Practicals (710.241)

2022/2023

David Mihola
david.mihola@student.tugraz.at

Robert Stojanovic
robert.stojanovic@student.tugraz.at

January 17, 2023

Graz

1 Application description

The core idea of our application is to take a photograph of an equation handwritten on a paper and provide the result. We limit the symbols in equations to digits (`0` to `9`) and basic mathematical operators, i.e. addition, subtraction, multiplication and division represented by `+`, `-`, `*` and `/` respectively.

We accomplish the laid out task using an ensemble of two approaches described in the following sections.

1.1 MSER detection and single digit recognition

First approach of the ensemble is using an algorithm for detecting individual symbols and then classifying them one-by-one. The detection of the symbols is done with the MSER algorithm described by Matas et al. [2]. This algorithm is used to find continuous handwritten regions in an image and return their positions. Since simple mathematical equations are rarely written with a single stroke we can use this algorithm to detect the regions and positions of the individual symbols. This works well, when the equations are written with a dark pen on a white paper, but once shadows appear or graph paper is used as the writing surface, the results tend to become less reliable. To circumvent this issue, we first pre-process the image by computing the adaptive threshold with a Gaussian kernel to better highlight individual symbols and then apply the MSER algorithm on the result.

Once we have found the relevant regions containing symbols, we crop those from the original image and apply Otsu's thresholding algorithm [3] to get a binary result. Lastly, we resize and pad the result to be of uniform size and proceed to classify it. The classification is done using a machine learning model, which takes a single symbol as an input and outputs the probabilities of it being a specific digit or an operator. The model is explained in further detail in section 1.1.2.

1.1.1 Data pre-processing and data sets

The recognition happens on a digit by digit basis so we need a data set containing all of the chosen symbols. The Dataset: Handwritten Digits and Operators[1] was a good starting point. It is a relatively large data set and the symbols are encoded as 28 by 28 pixel binary images, meaning the pixels of the digits are either white or black to better capture the stroke of a symbol. One downside of this data set is that the digits are all written by a single person, so the handwriting is likely biased and might not match the handwriting of other people. We attempted to solve this by augmenting the data set with our own handwritten digits and operators. To simplify the creation and labeling process we used a python interface originally based on Thevenot's [7] implementation.

1.1.2 Recognition model and training

The digit recognition model is a relatively small convolutional neural network made up of five layers. The input layer is a convolution layer with 16 filters and a 3 by 3 kernel using the `ReLU` activation function that takes a single 28x28 binary image as an input. This is followed by a pooling layer with the size (2x2). Then the output is flattened and connected with a dense layer with an output of 128 units activated by the `ReLU` activation function, which feeds into the output layer with a `Softmax` activation function for all 14 symbols. We trained the network using our combined data set using the Adam optimizer with 0.01 learning rate. The stopping criterion was early stopping, which maximized the validation accuracy with a patience of 3

[1] https://www.kaggle.com/datasets/michelheusser/handwritten-digits-and-operators

epochs. The final accuracy on the test set was 0.981. The process of training can be seen in figure 1 with the resulting confusion matrix being shown in figure 2.

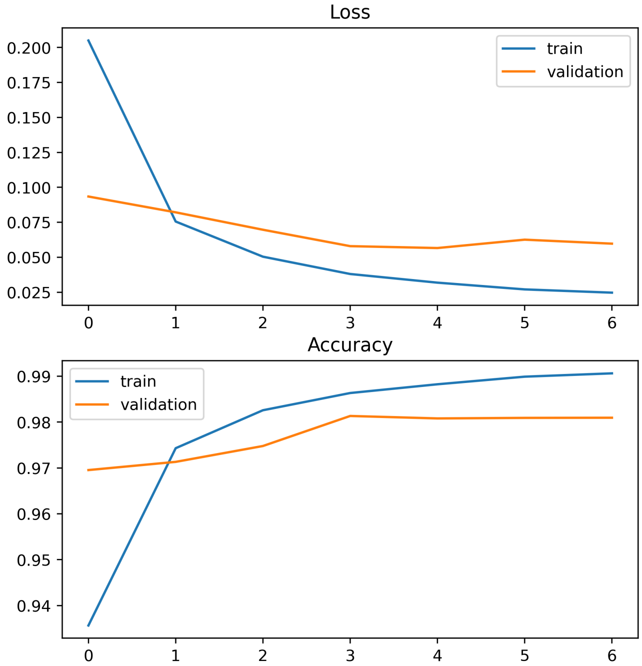
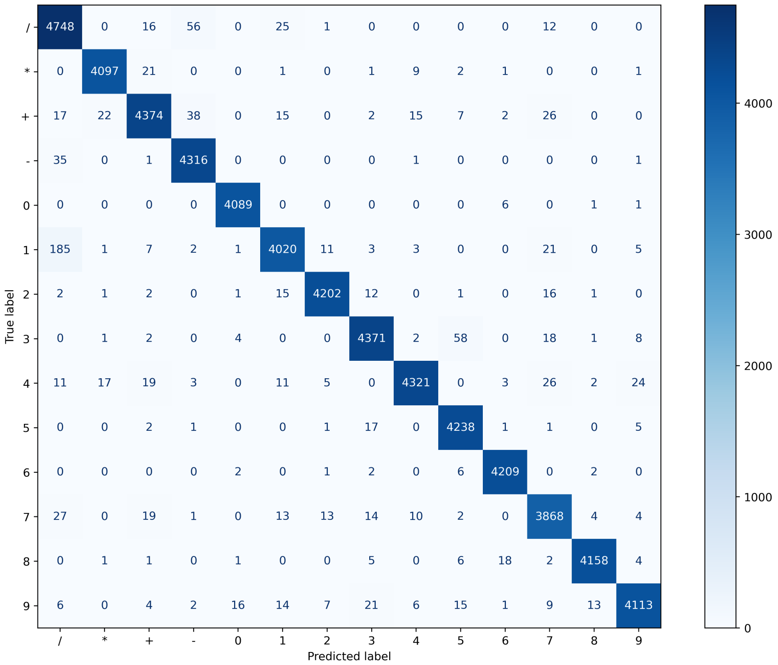


Figure 1: Development of loss and accuracy during training



	/	*	+	-	0	1	2	3	4	5	6	7	8	9
/	4748	0	16	56	0	25	1	0	0	0	0	12	0	0
*	0	4097	21	0	0	1	0	1	9	2	1	0	0	1
+	17	22	4374	38	0	15	0	2	15	7	2	26	0	0
-	35	0	1	4316	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	4089	0	0	0	0	0	6	0	1	1
1	185	1	7	2	1	4020	11	3	3	0	0	21	0	5
2	2	1	2	0	1	15	4202	12	0	1	0	16	1	0
3	0	1	2	0	4	0	0	4371	2	58	0	18	1	8
4	11	17	19	3	0	11	5	0	4321	0	3	26	2	24
5	0	0	2	1	0	0	1	17	0	4238	1	1	0	5
6	0	0	0	0	2	0	1	2	0	6	4209	0	2	0
7	27	0	19	1	0	13	13	14	10	2	0	3868	4	4
8	0	1	1	0	1	0	0	5	0	6	18	2	4158	4
9	6	0	4	2	16	14	7	21	6	15	1	9	13	4113

Figure 2: Test data set confusion matrix

1.1.3 Evaluation

The detector and classifier was tested on a few static images ranging from digital black on white to handwritten on graph paper. We found, that on digitally written equations the results tended to be very solid with only occasional misclassifications, while handwritten equations had more mixed results. The solution still provides good results for images without shadows, while equations, which are shadowed, fared much worse. Some exemplary results of such equations taken at varying conditions are shown in figures 3, 4 and 5.

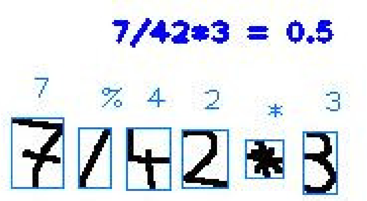


Figure 3: Digital equation on pure white background

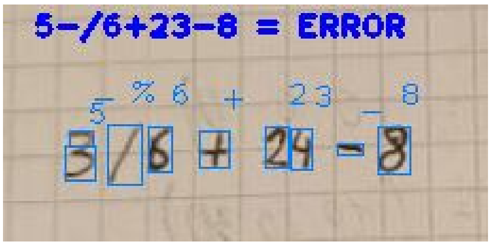


Figure 4: Equation on graph paper

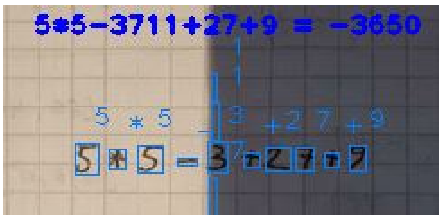


Figure 5: Equation on graph paper with shadows

1.2 Fully machine learning based approach

The second approach used in the ensemble is based only on a machine learning model. The model takes an image of a whole equation as an input and outputs a set of predictions, which are processed into a string representation of the equation. The following sections describe, how we design, trained and evaluated the model using publicly available data sets of handwritten symbols.

1.2.1 Data pre-processing and data sets

We used 3 publicly available data sets of hand written digits, operators and other symbols. The data sets are:

- Handwritten math symbols dataset[2],
- Dataset: Handwritten Digits and Operators[3],
- Handwritting characters database[4].

Initially, we pre-processed the downloaded data sets and created our own data set by:

1. selecting only the relevant symbols for our task, i.e digits, `+`, `-`, `*` and `/`,
2. resizing the images to 28x28 pixels if not already in this size,
3. thresholding the images and converting them to black background with the symbol being white,
4. merging all of the images randomly to a single file and creating file with corresponding labels,
5. cleaning the images by removing outliers (mislabeled samples) with the use of a simple underfitted CNN,
6. cropping the images, so that the symbol spans the whole width of the image (the 28x28 size is no longer kept),
7. separating the symbols to train, validation and test files per class with labels being already part of these files,
8. generating syntactically correct equations with 3 to 10 symbols by randomly sampling (from respective files for each of the train, validation and test data sets) and concatenating them,
9. randomly shifting the concatenated image across the width of the final image with width of 288 pixels and height of 38 pixels,
10. labeling the final image.

Each image has 18 labels, which makes it 16 pixels per label for the width dimension. The label is given based on the center of a symbol appearing in a given part of the image. We made sure to add padding when two symbols appear in the same part of the image.

Later on, after evaluating the first working network on actual images of handwritten equations, we added augmentations into the pipeline, as the performance on these dropped substantially in comparison to the test data set performance, see section 1.2.3. We augmented the data by:

- resizing the symbols after stage 6 of the pipeline,
- applying dilatation after stage 10 of the pipeline.

[2] https://www.kaggle.com/datasets/xainano/handwrittenmathsymbols

[3] https://www.kaggle.com/datasets/michelheusser/handwritten-digits-and-operators

[4] https://github.com/sueiras/handwritting_characters_database

When resizing we always kept the original symbol as well as a larger and a smaller version of it for digits and two smaller versions for operators, as we noticed that it is common to write operators smaller than digits. The dilatation was added because the thickness of symbols after pre-processing of the actual handwritten equations was mostly larger than the thickness in the training samples.

1.2.2 Model architecture and training details

Our initial idea was to use multi object detection and classification by exploring and modifying the YOLO networks as described in [4], [5] and [6] to our needs. This primarily meant changing the input shape from a square aspect ration to aspect ratio with dominant width and simplifying the network to be able to train it on our machines. At the same time, we did not need to detect exact positions of each digit and character in an image, as knowing their relative positions is sufficient for our task. When trying experiments with networks inspired by the YOLO networks, we were constantly running into two issues. First, we could not make the network simple enough to train it sufficiently on our machines while keeping the YOLO loss function. Second, we struggled with defining our simpler loss function, which would enable training on much simplified models. One such a failed architecture is defined in the YOLO_inspired_CNN.py[5] script. The network, we think, is quite effective at recognizing single characters, but it is not capable of assigning them the correct position.

Nevertheless, by doing these experiments, we learned, what works and what does not, and designed our own network architecture. We completely removed the detection part of the problem and focused only on classification. We made our architecture such that it classifies parts of the images. Its output is therefore a set of predictions, from which an equation can be constructed, as the relative position of the predictions is given by the predicted part of the image. The architecture consist of two parts.

The first part operates on the whole image and has the following layers:

- convolutional with 32 kernels of size 3x3 and 0x1 padding,
- max pooling with 2x2 kernel size,
- convolutional with 64 kernels of size 3x2 and 0x1 padding,
- max pooling with 2x2 kernel size,
- convolutional with 128 kernels of size 5x1 and 0x0 padding.

Keeping the kernel widths of the second and third convolutional layers 2 and 1 respectively ensures, that the per width receptive field of the first part of the network is relatively small, i.e. every of the output values contains information from an 8 pixels wide part of the original image (apart from the edge values, which also capture the added padding).

The second part of the network is applied separately on a divided output of the first part into 16 slices of size 128x4x4 (CxHxW) given the input image size described in 1.2.1. On slice captures information from a 32 pixels wide part of the original image (again that differs for edge slices due to padding). This part of the network has the following layers:

- convolutional with 256 kernels of size 3x3 and 0x0 padding,
- dropout with rate of 0.5,
- fully connected with 128 output units,
- dropout with rate of 0.5,

[5] https://github.com/xmihol00/visual_calc/blob/main/networks/YOLO_inspired_CNN.py

- fully connected with 15 output units.

Each of the output units corresponds to a probability of a class appearing on that part of the image. Apart from classes for digits and operators, there is one additional empty class (no digit or operator appears on that part of the image).

For both parts of the network, we used the `LeakyReLU` function with negative slope of 0.1 as the activation function for all hidden layers and the `Softmax` function as the output layer activation function. Additionally, we batch normalized the output of each hidden layer before applying the activation function.

Although this architecture performs much better than any other of the previously tried architectures, we decided to further improve it by introducing recurrence. We feed the predictions for the N-1th part of the image (zeros in case of the first part) to the input for the Nth part of the image by concatenating it with the flattened output of the last convolutional layer. The architectures are available in the custom_CNN.py[6] and custom_recursive_CNN.py[7] scripts respectively.

We trained these networks using the `Adam` optimizer with 0.01 initial learning rate. The learning rate was set to decay after every 5 epochs by a factor of 0.25. We used early stopping to maximize the validation accuracy with a patience for 3 epochs as the stopping criterion.

1.2.3 Evaluation and results on generated test set

We evaluate the performance of the described models by converting the ground truth labels and the predicted labels to strings and by comparing them using the Levenshtein distance, see [1] for a good explanation. In short, when the Levenshtein distance is 0, the two input strings match, when the distance is larger than 0, the string differs and at least distance deletions, additions or replacements of characters or combination of those must be done in order to make the strings match. The results are shown in figure 6.

1.2.4 Inference and results on actual handwritten equations

We pre-process the given RGB input image by converting it to gray scale and further thresholding it to black and white duo-tone. We locate interesting areas of the image by summing up pixel values. Then we resize these areas to have heights of 38, 36, 34 and 32. Afterwards we place each of these resized areas shifted by 1 pixel each time in the final 288x38 image. This gives us 64 images in total, which we predict. The the most occurring syntactically valid prediction is regarded as the final prediction. The results are shown in figure 6.

1.3 Evaluation of the combined model

We combined the two recognition methods in to an ensemble to get more robust solution than either of the individual methods would provide by gathering both of their predictions and selecting the most frequent one. Specifically, we retrieved the 4 most likely predictions of the MSER variant by building the equations from the digits combinations with the highest probabilities and gave them decreasing weights, so that the best prediction had a weight of 4, while the lowest had a weight of 1. We then combine these along with the 64 predictions of the multi-classifier and select most occurring syntactically valid prediction as the final one. The final result of this combined classification is shown in figure 7.

[6] `https://github.com/xmihol00/visual_calc/blob/main/networks/custom_CNN.py`

[7] `https://github.com/xmihol00/visual_calc/blob/main/networks/custom_recursive_CNN.py`

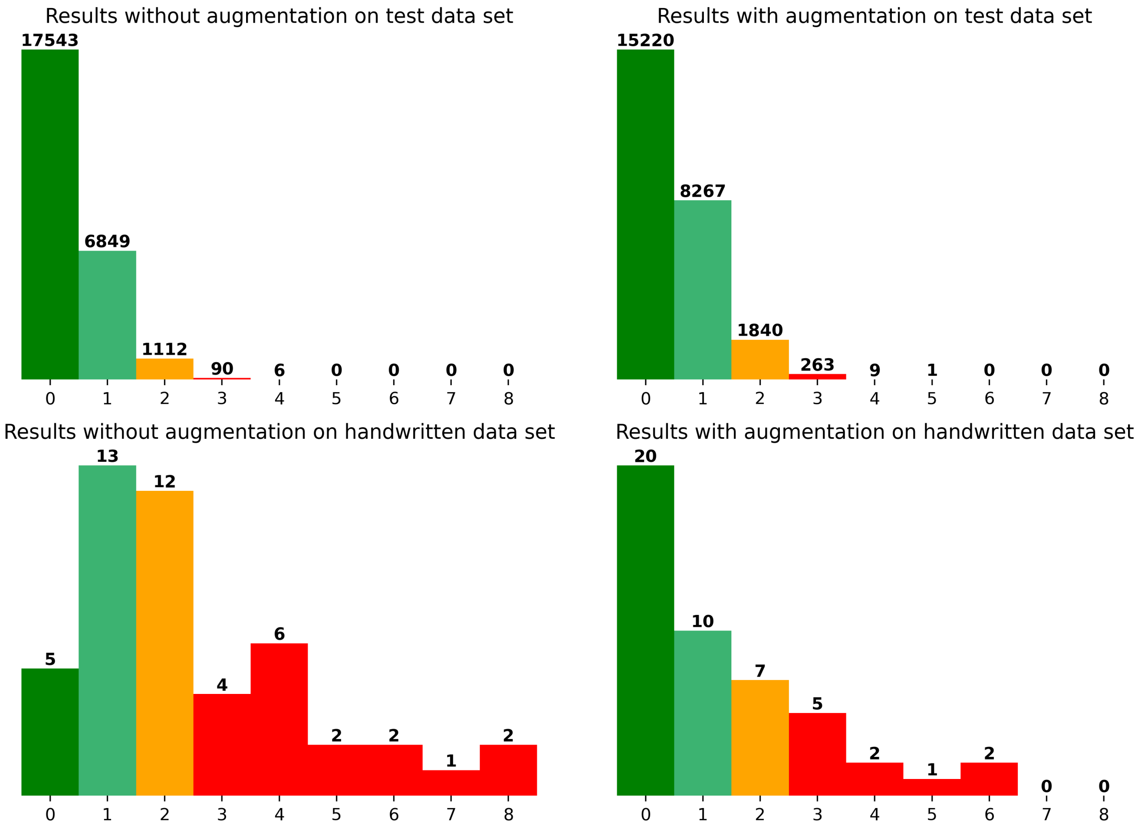


Figure 6: Levenshtein distance between ground truth labels and predictions of the final model architecture trained on not augmented and augmented train data sets

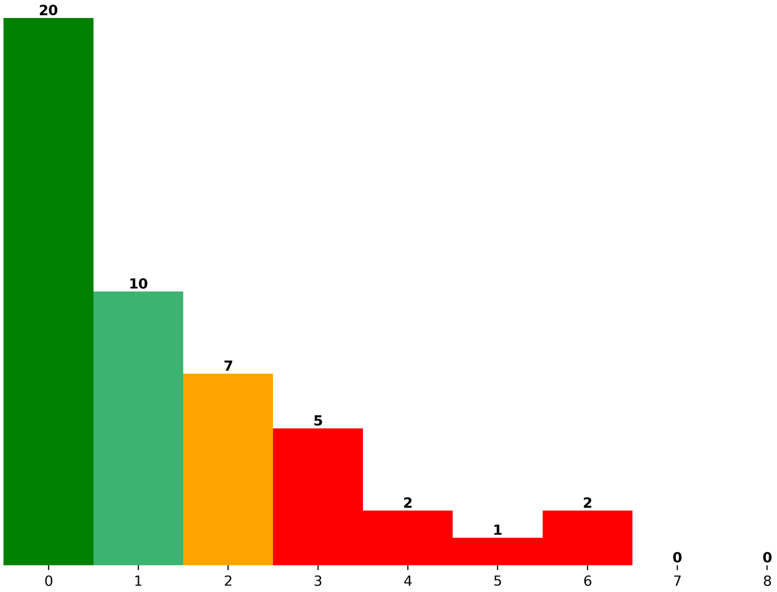


Figure 7: Levenshtein distance between ground truth labels and predictions of the final ensemble

2 Updated Project Plan

The plan listed below is an update version of the proposed project plan. Changes are highlighted in red and details of the individual tasks can be found in the proposal document.

- **Research the common approaches to multi-object classification.**
  Date: 20. 10. 2022
  Responsibility: Robert, David, Raul
- **Clean and pre-process existing data sets, randomly generate equations.**
  Date: ~~31. 10. 2022~~ 10. 11. 2022
  Responsibility: David
- **Build simplified models for the promising approaches with generated equations.**
  Date: 6. 11. 2022
  Responsibility: Robert implements a single digit classifier, David implements a small multi digit detector.
- **Implement a script that pre-processes handwritten equations by us.**
  Date: st15. 11. 2022 27. 12. 2022
  Responsibility: ~~Raul~~, David
- **Implement a script for annotating handwritten digits and operators.**
  Date: 30. 11. 2022
  Responsibility: Robert
- **Improve generated data set.**
  Date: ~~10. 12. 2022~~ 23. 12. 2022
  Responsibility: Robert, David, ~~Raul~~
- **Implement an algorithm to reconstruct and solve the labeled equations.**
  Date: ~~20. 12. 2022~~ 28. 12. 2022
  Responsibility: ~~Raul~~ Robert, David
- **Finish implementation of the final models.**
  Date: 31. 12. 2022
  Responsibility: David, Robert
- **Fine-tune and evaluate the chosen model using our own data.**
  Date: 5. 1. 2023
  Responsibility: David, Robert
- **Combine all the parts into a single application.**
  Date: 10. 1. 2023
  Responsibility: ~~Raul~~ Robert, David
- **Finish writing the report.**
  Date: ~~15. 1. 2023~~ 17. 1. 2023
  Responsibility: ~~Raul~~ David, Robert

References

[1] Haldar, R. and Mukhopadhyay, D. Levenshtein Distance Technique in Dictionary Lookup Methods: An Improved Approach. *CoRR*. 2011, abs/1101.1232. Available at: http://arxiv.org/abs/1101.1232.

[2] Matas, J., Chum, O., Urban, M. and Pajdla, T. Robust wide-baseline stereo from maximally stable extremal regions. *Image and Vision Computing*. 2004, vol. 22, no. 10, p. 761–767. DOI: https://doi.org/10.1016/j.imavis.2004.02.006. ISSN 0262-8856. British Machine Vision Computing 2002. Available at: https://www.sciencedirect.com/science/article/pii/S0262885604000435.

[3] Otsu, N. A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man, and Cybernetics*. 1979, vol. 9, no. 1, p. 62–66. DOI: 10.1109/TSMC.1979.4310076.

[4] Redmon, J., Divvala, S., Girshick, R. and Farhadi, A. *You Only Look Once: Unified, Real-Time Object Detection*. arXiv, 2015. DOI: 10.48550/ARXIV.1506.02640. Available at: https://arxiv.org/abs/1506.02640.

[5] Redmon, J. and Farhadi, A. YOLO9000: Better, Faster, Stronger. *CoRR*. 2016, abs/1612.08242. Available at: http://arxiv.org/abs/1612.08242.

[6] Redmon, J. and Farhadi, A. YOLOv3: An Incremental Improvement. *CoRR*. 2018, abs/1804.02767. Available at: http://arxiv.org/abs/1804.02767.

[7] Thevenot, A. *Python Interface to Create Handwritten dataset* [online]. 2020. Available at: https://www.github.com/AxelThevenot/Python-Interface-to-Create-Handwritten-dataset.