Performing arithmetic using a neural network trained on images of digit permutation pairs

Marcus D. Bloice¹ · Peter M. Roth² · Andreas Holzinger³

Received: 10 December 2020 / Revised: 23 June 2021 / Accepted: 12 July 2021 /
Published online: 6 August 2021
© The Author(s) 2021

Abstract

In this paper, a neural network is trained to perform simple arithmetic using images of concatenated handwritten digit pairs. A convolutional neural network was trained with images consisting of two side-by-side handwritten digits, where the image's label is the summation of the two digits contained in the combined image. Crucially, the network was tested on permutation pairs that were not present during training in an effort to see if the network could learn the task of addition, as opposed to simply mapping images to labels. A dataset was generated for all possible permutation pairs of length 2 for the digits 0–9 using MNIST as a basis for the images, with one thousand samples generated for each permutation pair. For testing the network, samples generated from previously unseen permutation pairs were fed into the trained network, and its predictions measured. Results were encouraging, with the network achieving an accuracy of over 90% on some permutation train/test splits. This suggests that the network learned at first digit recognition, and subsequently the further task of addition based on the two recognised digits. As far as the authors are aware, no previous work has concentrated on learning a mathematical operation in this way. This paper is an attempt to demonstrate that a network can learn more than a direct mapping from image to label, but is learning to analyse two separate regions of an image and combining what was recognised to produce the final output label.

Keywords Neural networks · Cognition

✉ Marcus D. Bloice
marcus.bloice@medunigraz.at

Peter M. Roth
p.m.roth@ieee.org

Andreas Holzinger
andreas.holzinger@medunigraz.at

¹ Institute for Medical Informatics, Statistics, and Documentation, Medical University of Graz, 8036, Graz, Austria

² International AI Future Lab, Technical University of Munich (TUM), Ludwig-Bölkow-Campus, Willy-Messerschmitt-Straße 1, 85521, Taufkirchen, Germany

³ xAI Lab, Alberta Machine Intelligence Institute, Edmonton, T6G 2H1, Canada

1 Introduction

The aim of this study is to attempt to find experimental evidence that would suggest that a network can be trained to perform the task of addition, when supplied with image data containing two digits that should be summed. To ensure that the network has indeed learned this, and is not simply mapping images to labels, a constraint was applied whereby the network is tested with a held back test set of previously unseen permutation pairs. This forces the network to learn more than simply a mapping between individual images and labels as it is tested using digit combination pairs that it has not seen, meaning a direct mapping from an image or shape to a label would not function.

To test this hypothesis, a network was trained with data generated using 90 of the possible 100 combinations of the digits 0–9 up to length 2. Once trained, the network was tested by inputting images based on the remaining 10 permutation pairs, and was required to predict the summations for them. Some examples of the generated samples can be seen in Table 1. 1,000 samples were generated for each permutation pair. This was repeated 10 times for 10 different permutation pair train/test splits.

This was done by randomly selecting two MNIST digits for each permutation pair, and sampling as many times as required. In our initial experiments, 1000 images were created for each permutation pair, meaning 90,000 images for the training set and 10,000 images for the test set. The images for the training set were sampled from the MNIST training set, and the images for the test set were sampled from the MNIST test set. This ensured the two datasets were distinct in two ways: 1) the test set contained permutations not seen by the network during training, and 2) the test set contain MNIST images not seen by the network during training.

The task is challenging for several reasons:

- Any permutation pairs in the test set were not seen during training
- The individual digits are not labelled, and each input image (permutation pair) was labelled only by its summation
- The MNIST digits used to generate the permutation pairs were randomised and different for every image generated

Table 1 Example input images and their corresponding labels

Image	Interpretation	Label
$5 + 0$	5
$8 + 5$	13
$9 + 9$	18

Each image consists of two side-by-side MNIST digits merged into one single image where each image's label is the summation of the two digits. The label contains no information as to the individual digits contained within the image, nor is there any indication given to the network prior to training that each image consists of two digits or otherwise

- The MNIST digits used to generate the permutation pairs were separated into a train/test set prior to the generation of the data

All experimentation was performed using Keras, a Python deep learning library, with TensorFlow as its backend, running under Ubuntu Linux 14.04. Training was undertaken using an Nvidia Titan X GPU.

2 Background work

There has been previous work relating to this experiment. As opposed to most work, however, the goal of this study was not to recognise digits, extract their numerical values from the images, and then perform (after the network's character recognition procedure) some mathematical function on the numerical values. The task of this experiment was to learn if the network could learn the logical task of the mathematical operation itself using an end-to-end approach. For example, Hoshen and Peleg (2016) experimented with computer generated image data, however as output the network was trained to produce images containing the summations. Their work concentrated on the visual learning of arithmetic operations from images of numbers. In contrast, the work presented here outputs its predictions as a real number. Their approach used numbers of longer lengths and were therefore also able to generate many thousands of training samples, despite not using hand written digits. The input consisted of two images, each showing a 7-digit number and the output, also an image, displayed a number showing the result of an arithmetic operation (e.g., addition or subtraction) on the two input numbers. The concepts of a number, or of an operator, are not explicitly introduced. Their work, however, was more akin to the learning of a transformation function, rather than the task of learning a mathematical operation. Other operations, such as multiplication, were not learnable using this architecture. Some tasks were not learnable in an end-to-end manner, for example the addition of Roman numerals, but were learnable once broken into separate sub-tasks: first perceptual character recognition and then the cognitive arithmetic sub-task.

Similarly, a convolutional neural network was used by Liang et al. (2016) to recognise arithmetic operators, and to segment images into digits and operators before performing the calculations on the recognised digits. This again is different to the approach described here, as it is not an attempt to learn the operation itself, but to learn to recognise the operator symbols and equations (and perform the mathematics on the recognised symbols).

In Walach and Wolf (2016), the authors addressed the task of object counting in images where they applied a learning approach in which a density map was estimated directly from the input image. They employed convolutional neural networks with layered boosting and selective sampling. It would be possible to create an experiment based on their work, that would perform arithmetic by counting the values of domino tiles, for example.

Other work has indeed used MNIST as the basis for the training and testing data, such as in Trask et al. (2018), where the authors used so-called neural arithmetic logic units (NALUs). They showed that NALU-enhanced networks could perform arithmetic over sequences of images, track time, count objects in images, and translate numerical language in to scalars.

While much work has concentrated on learning to execute simple mathematical operations input in numerical form Zaremba and Sutskever (2014), Kaiser and Sutskever (2015), Kalchbrenner et al. (2015), Zaremba et al. (2016), and Freivalds and Liepins (2017), until now, as far as the authors are aware, no work has concentrated on learning the actual

mathematical operation itself through the analysis of images using standard convolutional networks. When considering work where data is input as images, previous work tends to concentrate on first recognising digits and operators within images, and then to perform the mathematical operations separately, after this extraction has been carried out. In the case of this work, an end-to-end algorithm has been developed that performs the digit recognition, representation learning of the values of the digits, and performs the arithmetic operation.

3 Experiment

A convolutional neural network was trained to perform arithmetic on images consisting of two side-by-side hand-written digits. Each image's corresponding label is the sum of the two digits, and the network was trained as a regression problem. For the digits 0–9, up to length 2, there are 100 possible permutation pairs. For each permutation pair, 1,000 unique images were generated using the MNIST hand written digit database (LeCun et al., 1998). MNIST was chosen due to its familiarity in machine learning, and because LeNet5 (used as the basis for the network trained in this work) clearly functions well on this dataset. Also, MNIST continues to be a focus of research and is used as a benchmarking dataset to this day (Delahunt & Kutz, 2019; Phong et al., 2018).

We focused on the task of addition, as opposed to other operations such as multiplication or subtraction for several reasons. First, the focus of this work was to learn if a network could learn a concept from visual data. To demonstrate this, focusing on one particular operation is sufficient, and we felt performing the experiment multiple times for other operations would detract from the main goal of the paper and complicate the reporting of our results. Second, operations such as multiplication and division have particular idiosyncrasies that would have complicated the experimental setup, such as how to deal with divide-by-zero in the case of the division operator, or having to deal with negative numbers and number order in subtraction (such as the pairs $(9, 3)=6$ and $(3, 9)=-6$ for example). We therefore wished to avoid such issues in order to keep the experiment as simple as possible without detracting from the main goal of the paper. However, we did experiment with other operations, and some preliminary results show promising results, which is described in more detail in Section 6.1 and shown in Table 8.

Training was performed on images generated from a random 90-long subset of the possible permutations, and testing was performed on images based on the remaining 10 permutations. A number of example input images and their labels are shown in Table 1 where it can be seen that a single input image consists of two MNIST digits side-by-side, and the image's label is the summation of the two digits. For each permutation, 1,000 combined images are generated resulting in 100,000 samples that are separated into a training and test set based on the permutation pairs.

The task of the experiment was to train a neural network with data generated from a subset of the possible 100 permutations, that when presented with a new samples, generated from the unseen permutation pairs, the network would be required to predict the correct summation. This was done in order to ascertain whether a network could learn a simple arithmetic operation such as addition, given only samples of images and their summations and no indication as to the value of each individual digit contained within the image, while only being trained on a subset of all possible permutation pairs and tested on the remaining pairs.

In summary, the experimental setting is as follows:

- By permutations, it is meant all possible combinations of the digits 0–9 of length 2. Formally, if the set of digits $D = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, all possible permutations is the Cartesian product of $D \times D$, which we define as P , so that $P = \{(0, 0), (0, 1), (0, 2), \dots, (9, 8), (9, 9)\}$.
- Of the 100 possible permutations pairs P , a random 90 are used as a basis to train the network and the remaining 10 pairs are used as a basis to test the network. These are the training permutations, P_t , and the test permutations, P_v . This permutation train/test split was repeated 10 times as a 10-fold cross validation.
- For each permutation, 1,000 samples are generated. So, for each of the permutations in P , 1,000 concatenated images are generated using random MNIST digits M (appropriate for that permutation).
- By appropriate this means that, for example, generating an image for the permutation pair (3, 1) a random MNIST digit labelled 3 is chosen and a random MNIST digit labelled 1 is chosen and these images are concatenated to create a single sample for this permutation pair. This means each sample is likely unique (likely, as each image is chosen at random with replacement, see Table 2). The generated images are contained in a matrix X , where X_t are the training samples and X_v are the test samples.
- For the generation of the training set images, X_t , only images from the MNIST training set, M_t , are used.
- For the test permutation images, X_v , only images from the MNIST test set, M_v , are used.
- The network is not given any label information regarding each individual digit within the concatenated images, only the summation is given as label data.
- The permutations pairs in the test set are not seen during training. This means the training set and test set are distinct in two ways: they contain different permutations pairs that do not overlap, and the individual MNIST images used to generate each permutation sample do not overlap between the training set and test set.

The decision to train the network as a regression problem is done for the following reasons. First, the number of output neurons would change depending on the train/test split. For example, the sum of (9, 9) is 18 and cannot be made by any other permutation, meaning a possible discrepancy between the number of possible output neurons of the training set and test set. Second, when training on permutations of longer length, an ever increasing number of output neurons would be required. Last, for measuring how well the network

Table 2 For each permutation pair, random MNIST digits are used for generating each sample

Sample 1	Sample 2	...	Sample 1000
...

For example, for the permutation pair (0, 2), each sample that is generated uses a random digit 0 combined with a random digit 2 obtained from MNIST

performs, classification poses problems which are mitigated by using regression and mean squared error loss.

The following sections describe the experiment itself, beginning with a description of the dataset and how it was created. Then, the neural network's architecture is described as well as the training strategy. Last, the results of the experiment are discussed, and the paper is concluded with a discussion.

4 Dataset

The dataset used for the creation of the concatenated image data was MNIST, a 70,000 strong collection of labelled hand written digits. As per the original dataset, 60,000 digits belong to the training set and 10,000 belong to the test set. Images in the MNIST dataset are 8-bit greyscale, 28×28 pixels in size. The generated images are therefore 28×56 pixels in size as they are the concatenation of two MNIST digits placed side-by-side and stored as a single image (see Table 3, for example). Each generated image's label is the summation of the two individual digits' labels (see Table 1 for several examples). For each permutation, one thousand samples are generated, and the MNIST digits are chosen at random in order to create distinct samples.

4.1 Train/Test split

As mentioned previously, for the digits 0–9, with a maximum of length of $l = 2$, there are n^l or $10^2 = 100$ possible permutation pairs. To generate the training and testing data, the permutations pairs are split into a permutation training set and a permutation test set at random, so that: $P = P_t \cup P_v$ and $P_t \cap P_v = \emptyset$. For training, 90% of the permutations were

Table 3 Example results for permutation samples from the test set passed through the trained network. As can be seen, all samples use distinct MNIST digits

Input image	Prediction	Actual
11.1652	11
10.3215	10
5.01775	5
8.99357	9
5.99666	6
8.6814	9

used to generate the training samples, X_t and the remaining 10% were used for generating the test set samples X_v , meaning $|P_t| = 90$ and $|P_v| = 10$ while $|X_t| = 90,000$ and $|X_v| = 10,000$ for any particular run. The experiment was performed using a 10-fold cross validation, based on the permutations pairs, and the loss was averaged across the 10 runs.

In terms of the generated samples, the generated training set images and generated test set images honoured the MNIST training set and test set split. This means that the generated training set samples are distinct from the generated test set samples both in terms of the permutation pairs and the images used to create each sample. It should be noted that the images were chosen from MNIST randomly *with replacement*, meaning images could appear twice in different permutation pairs within the training set or test set, but not between both. Therefore, the generated data set matrix X contains 100,000 samples, 10,000 for each permutation pair. This also means that X_t contains the corresponding samples for the permutations P_t , and X_v contains the samples for P_v . A label vector y contains the labels, which are the summations of the two digits in the sample. Similarly, $y = y_t \cup y_v$.

4.2 Network architecture and training strategy

The neural network used was a multi-layer convolutional neural network similar to the original LeNet5, which was first reported by LeCun et al. (1989). However, rather than treating the problem as a classification problem, the network is trained as a regression problem. The network was evaluated using mean squared error loss and optimised with ADADELTA (Zeiler, 2012). The network consisted of two convolutional layers, three dense layers, and approximately 5 million parameters. For exact details regarding the network type, please consult the paper's accompanying GitHub repository which contains the source code required to replicate the experiment. See Section 5 for the URL to the repository. All experimentation was performed with Keras using TensorFlow (Abadi et al., 2016) as its back-end, running under Ubuntu Linux 14.04.

A LeNet5-type network was chosen due to its association with MNIST, having been optimised and developed for this dataset, and has repeatedly been shown to work well at the general task of character recognition. As the inputs to this network are similar to the original MNIST images, having twice the width in pixels but having the same height in pixels, the only other modification that was made was with the output of the network. Instead of a 10 neuron, fully connected output layer with *Softmax*, the final layer was replaced with a single output neuron and trained as a regression problem, optimising mean squared error loss.

4.3 Data generation

The data generation procedure algorithm is shown in Algorithm 1. During data generation, the permutations, P , are iterated over and $m = 1000$ samples are generated for each permutation. A sample consists of two random MNIST images, corresponding to the labels in the current permutation, concatenated together as one image (this is represented by the function **ConcatenateImages**). As well as this, the label vector y is generated, which contains the sum of the two digit labels that make up each individual MNIST image used to create the sample. Note that in Algorithm 1 the symbol $\leftarrow$ represents append, as is the case for the matrix X and its corresponding label vector y .

The procedure shown in Algorithm 1 is repeated for the train set permutation pairs, P_t , and the test set permutation pairs, P_v . It is important to note that when generating the data for the training set permutation images, the MNIST training set is used, and conversely when generating the test set permutation images, the MNIST test set is used. This ensures

no overlap between the training set or test set in terms of the permutation pairs or the data used to generate the samples. Also important to note is that the label vector $\mathbf{y}$ contains the summation of the labels for each individual MNIST image contained in the generated sample.

Algorithm 1 Data Generation. The procedure by which the image data was generated is described below. Note that the symbol $\Leftarrow$ refers to append in this case.

Input: permutation pairs P , MNIST images M , labels y' , size $m \leftarrow 1000$.
 Initialise $X \leftarrow []$.
 Initialise $y \leftarrow []$.
for all (p_1, p_2) **in** P **do**
 for 1 **to** m **do**
 $r^1 \leftarrow$ random index from M with label p_1
 $r^2 \leftarrow$ random index from M with label p_2
 $X \Leftarrow \text{ConcatenateImages}(M_{r^1}, M_{r^2})$
 $y \Leftarrow y'_{r^1} + y'_{r^2}$
 end for
end for

5 Results

Averaged across the different training/test splits of a 10-fold cross validation of the permutation pairs, mean squared error was generally under 1.0 and averaged 0.85332, as shown in Table 7. Tables 3, 4, and 5 show a number of examples of a trained network's predictions on permutations from a test set. Table 3 shows a number of sample inputs from the test set and their predictions, as well as their true labels. It is interesting to note that the network learned to deal with permutations with images in reverse order, as is the case for (6, 4) and (4, 6) or (1, 3) and (3, 1) in Table 4. In some cases, three different permutation pairs exist in the test set which sum to the same number, and these were also predicted correctly, as seen in Table 5.

Although the network was trained as a regression problem, accuracy can also be measured by rounding the predicted real number output to the nearest integer and comparing it to the integer label. When rounding to the nearest digit, accuracy was as high as 92%, depending on the train/test split and averaged 70.9%. Accuracy increases, if the predicted value is used with a floor or ceiling function, and both values compared to the true value, achieving an accuracy of approximately 88% across a 10 fold cross validation. When allowing for an error of ± 1 , the accuracy, of course, increases further. Table 6 shows the accuracy of the trained network over a 10 fold cross validation. The accuracies are measured across all samples of all test set permutations—a total of 10,000 images. The accuracies are provided here merely as a guide, for regression problems it is of course more useful to observe the average loss of the predictions versus the labels. Errors presented here could be the result of misclassifications of the images themselves, although this is unlikely given that LeNet5 can classify MNIST with an accuracy of approximately 99%, albeit as a classification task and within a much different experimental setting, and having been trained as a regression problem rather than a classification problem. More likely is that some train/test splits would be more difficult to classify than others. For example, a test set with the combination of (9, 9), resulting in a label of 18, can only be achieved with this combination, while the label

Table 4 Example results of correct predictions for test set permutations that have the same label but consist of different pairs of digits such as (6, 6) and (4, 8) or (9, 2) and (3, 8)

Input image	Prediction	Actual
11.8673	12
11.8703	12
10.8862	11
10.8827	11
9.84883	10
9.9731	10
10.0746	10
3.7308	4
4.23593	4

Note also that the model was also able to deal with digits in swapped order, as is the case for (1, 3) and (3, 1). Table 5 shows a further example of this

of 10, for example, can be achieved through several combinations, such as (6,4), (5,5), and so on. Therefore, we find it likely that the mean squared error fluctuations are due to this. Also, even for folds with low accuracy there are always correct predictions for all permutation pairs, again a further reason why it is not entirely useful to report accuracies. The mean squared error loss is provided as a truer measure of the network’s performance, and

Table 5 Example of correct predictions for three permutations from the same test set that sum to the same value

Input image	Prediction	Actual
9.84883	10
9.9731	10
10.0746	10

Table 6 Accuracy of each run of a 10-fold cross validation

Fold	Rounding	Floor/ceiling	± 1
1	81.19%	85.51%	94.93%
2	43.00%	90.89%	96.23%
3	80.95%	86.95%	95.12%
4	55.08%	71.21%	86.56%
5	82.35%	93.56%	95.28%
6	92.23%	95.49%	96.76%
7	54.32%	86.94%	95.38%
8	74.86%	94.71%	96.67%
9	87.59%	94.33%	95.91%
10	57.48%	85.53%	96.23%
Avg.	70.91%	88.51%	94.90%

For each permutation there are always correct predictions, even for poorly performing folds, such as folds 2 and 4. By using floor and ceiling functions on the predicted values for each of the images, the accuracy increases significantly. Note that for the results here 2,000 samples per permutation were generated. The variance seen here could be attributed to several factors, including the possibility that network make erroneous predictions of digit classifications themselves, and that some digits are more difficult to classify than others and are more frequently confused, such as 1 and 7 or 0 and 8. Furthermore, and perhaps most likely, is that certain training and testing splits will be easier to classify than others. For example, a test set including the digits (9, 9) results in 18, which can only be achieved using this pair. However, the result of 10 can be achieved using (5, 5), (6, 4), (7, 3) and so on making this output more likely as more pairs sum to this value

in order to provide as accurate a loss as possible a 10-fold cross validation was performed. The results of a 10-fold cross validation of the permutation pairs can be seen in Table 7. The average mean squared error loss over the 10-fold cross validation was 0.853322.

The entire experimental setup can be reproduced, with source code available under <https://github.com/mbloice/MNIST-Calculator> as a self-contained Jupyter notebook with commentary and visualisations of the dataset and results.

Table 7 Results of each run of a 10-fold cross validation

	Fold	Test set MSE	Train set MSE
	1	1.1072	0.0632
	2	0.6936	0.0623
	3	0.7734	0.0661
	4	0.7845	0.0607
	5	0.9561	0.0694
	6	0.7732	0.0553
	7	1.2150	0.0803
	8	0.7278	0.0674
	9	0.9464	0.0602
The average mean squared error (MSE) across 10 runs was ≈ 0.85 on the test set	10	0.5556	0.0709
	Avg.	0.8533	0.0656

6 Future work

6.1 Further mathematical operations

During the course of this research, a number additional experiments were performed using other arithmetic operations such as multiplication and subtraction. The full results of these experiments are not yet ready for publication, however we can report briefly here on preliminary results. For example, the mathematical operation of multiplication proved more difficult to train, but preliminary results were encouraging (see Table 8 for some example predictions). The network was trained as a regression problem using an identically structured network as for the addition experiment. Difficulties in multiplication that do not occur in addition are for example multiplication by 0, where the presence of a 0 always results in 0 as an answer. Likewise, multiplication by 1 is challenging for similar reasons.

As can be seen in Table 8, preliminary work has shown that a network was trained that handles some of the difficulties inherent to multiplication, at least some of the time (e.g. 0×7 or 5×1). Further work will concentrate on refining these results until they are suitable for publication. We are confident this can be done, but will likely require a rethink of the experimental setup that worked for the addition task. For example, multiplication may require a redesign of the type of network used, it may perform better when framed as a classification rather than a regression problem, may require more generated data, or indeed the use of image augmentation (Bloice et al., 2019), which was not done in the case of addition, or the use of a larger data set, as described in Section 6.2.

6.2 Alternatives to MNIST

MNIST has been a benchmarking dataset for several decades now, and researchers are beginning to ask if alternatives to MNIST should perhaps be considered, particularly as

Table 8 Examples results of a network trained with multiplication as its target label as opposed to addition, including an example of multiplication by zero, resulting in the correction interpretation of the permutation $0 \times 7 = 0$

While results were not as good as with the addition task, this was preliminary work and several methods could be adopted to improve these results, such as the generation of a much larger data set

Input image	Interpretation	Prediction	Actual
4×5	20.3187	20
3×8	24.1791	24
0×7	0.152293	0
7×4	28.6621	28
2×2	4.1404	4
5×1	5.25269	5

the same, rather small, test set is used in most experiments resulting in a type of overfitting (Yadav & Bottou, 2019). While this does not affect the experiment reported here, as we are not excessively concerned with getting the absolute best feasible accuracy at classifying images due to averaging our summations across many samples, we believe an alternative dataset should be investigated in future work. For example, QMNIST (Yadav & Bottou, 2019) is a possible alternative to MNIST that could be swapped out with relative ease. QMNIST is an expanded version of MNIST, generated from the original NIST Special Database 19, with the focus on matching the pre-processing of MNIST as closely as possible.¹ QMNIST is included in PyTorch, and can be loaded using the datasets loader² and includes 60,000 test set samples rather than the standard 10,000 included in MNIST.

Another possible avenue could be a custom generated dataset, based on computer fonts with large amounts of data augmentation. Having tight control over the generation of the dataset used for our future experiments may be crucial in improving the overall accuracy of the models we train.

6.3 Classification vs. regression

The results presented here are when the network was trained as a regression problem (see Fig. 1), however, it would of course be possible to treat the problem as a classification problem. It is unclear whether regression or classification would result in better results, however, regression allowed us to round up and down the output of the network as seen in the results in Table 6. Also, when considering sequences of longer lengths (see Section 6.4) the number of output neurons explodes, and regression may be preferred in such cases.

6.4 Longer digit sequences

A further experiment that may warrant attention in future work is to perform addition with sequences of digits that are longer than 2. In this case, it is likely that regression will prove to be a more suitable way to frame the problem, as sequences of length $l = 3$ would require $10^3 = 1,000$ output neurons, and sequences of length $l = 4$ would require $10^4 = 10,000$ output neurons, and so on. This may affect results and it may become apparent that simply using regression is the preferred option, as was performed in the work presented in this paper.

6.5 Summarising future work

To summarise, future work will concentrate on a combination of enhancements to the experimental setting, including applying different mathematical operators, using a larger and more challenging dataset such as QMNIST, and using sequences of digits longer than length 2.

7 Interpretation

The question of whether or not a model's output, whether that of a linear model or deep learning model, can be trusted, is a long standing one (Lipton, 2018). In order to rule out,

¹<https://github.com/facebookresearch/qmnist>

²<https://pytorch.org/docs/stable/torchvision/datasets.html#qmnist>

Fig. 1 Mean squared error loss measured on a the test set and training set over 100 epochs, reported here for fold 10 of the 10-fold cross validation. Loss on the final epoch was 0.5556, and was also the lowest loss recorded during this training run

as confidently as this can be said, that the network has merely learned a mapping from image to label, the neural network was tested with permutations that were never encountered during training. However, to further understand what the network has learned a number visualisation techniques were also applied. For example, regions of the input image can be visualised that contribute most towards the activation of the output neuron using a saliency map. Figure 2 shows such a visualisation in the form of a guided back-propagation saliency map (Simonyan et al., 2013; Kotikalapudi, 2017; Adebayo et al., 2018).

Fig. 2 Guided back-propagation saliency map visualisation (Kotikalapudi, 2017) of a test set permutation fed through a trained model, in this case (5, 4). It can be seen that the network is focussing on two regions of the entire image. The saliency map suggests that the network has learned that the two regions to the left and right of each image influences the label of the image. Changing the left hand side region results in either a positively increased label or a negatively decreased label

Fig. 3 Saliency map of the negative gradients of the trained network, in this case (4, 8)

It can be seen from Fig. 2 that the network, which in this case has been fed an image from the test set of the permutation (5, 4), is focussing on two regions within the single, concatenated image. This suggests that the network has not learned a mapping from a single image to a label, but rather has learned to segment the image into two areas or sections and interpret them separately (and has learned some relationship between the two distinct areas).

Other visualisation techniques include visualisation of the areas of the image which contributed negatively to the output. Figure 3 shows this technique. It is also observable in this case that the network is focussing on two regions of the input image, and is not erroneously focussing on other, incorrect areas of the entire image.

The visualisations in Figs. 2 and 3 suggest that the network has not learned simply to map from an whole image, in its entirety, to a label. They suggest that the network has learned to segment the images into two regions, and to interpret each of these separately. The network has learned that some relationship, in this case addition, exists between these two regions, and altering the left hand side of the image either increases or decreases the image's label while altering the right hand side of the image also either increases or decreases the label's value. It has therefore determined that each side of the image contributes separately to the image's label. However, it has also recognised that different images belong to the same classes, as, for example, two quite differently written 5's do in fact represent the same integer value and have the same influence on the image's label.

It is worth reminding the reader that the model has done this without knowing what each digit represents in numerical form, and has had to learn this from scratch. Because this would require a quite large amount of data, the decision was made to use MNIST as the basis for the generation of the 100,000 samples used for the experiment. Because the network was tested on permutations it has never seen, suggests that it learned to interpret the digits within the image rather than a mapping from image to label. This argument can be further substantiated by examining the regions of interest that the network has learned to focus on during training.

8 Conclusion

In this work, we have presented a neural network that achieves good results at the task of addition when trained with images of side-by-side digits labelled with their summations,

and tested with digit combination pairs it has never seen. The network was able to predict the summation with an average mean squared error of 0.85 for permutation pairs it was not trained with. By testing the network on a distinct set of digit combinations that were unseen during training, it suggests the network learned the task of addition, rather than a mapping of individual images to labels.

A number of further experiments would be feasible using a similar experimental setup. Most obviously, the use of three digits per image could be performed using permutations up to length 3, or higher, about one of which we reported some preliminary results here. More generally, the applicability of this method to other datasets in other domains needs to be investigated more thoroughly, for example whether there is an analogous experiment which could be performed on a dataset that does not involve arithmetic but involves the combination and interpretation of unseen combinations of objects in order to make a classification, such as through the use of the Fashion-MNIST (Xiao et al., 2017) or ImageNet datasets. Furthermore, future work will also investigate other mathematical operators, such as multiplication and division, of which some preliminary results have already been reported here. Last, it may be worthwhile to investigate the impact of the digit image dataset itself, such as through the use of the expanded QMNIST dataset, or through a custom generated dataset based on computer fonts and expanded in size via augmentation.

Funding Open access funding provided by Medical University of Graz.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., & et al. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv:1603.04467.
- Adebayo, J., Gilmer, J., Muelly, M., Goodfellow, I., Hardt, M., & Kim, B. (2018). Sanity checks for saliency maps. In *Advances in neural information processing systems* (pp. 9505–9515).
- Bloice, M. D., Roth, P. M., & Holzinger, A. (2019). Biomedical image augmentation using Augmentor. *Bioinformatics*, 35(21), 4522–4524.
- Delahunt, C. B., & Kutz, J. N. (2019). Putting a bug in ML: The moth olfactory network learns to read MNIST. *Neural Networks*.
- Freivalds, K., & Liepins, R. (2017). Improving the neural gpu architecture for algorithm learning. arXiv:1702.08727.
- Hoshen, Y., & Peleg, S. (2016). Visual learning of arithmetic operations. In *AAAI Conference on artificial intelligence* (pp. 3733–3739).
- Kaiser, L., & Sutskever, I. (2015). Neural GPUs learn algorithms. arXiv:1511.08228.
- Kalchbrenner, N., Danihelka, I., & Graves, A. (2015). Grid long short-term memory. arXiv:1507.01526.
- Kotikalapudi, R. (2017). The keras-vis package. <https://github.com/raghakot/keras-vis>.
- LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278–2324.
- LeCun, Y., Jackel, L. D., Boser, B., Denker, J. S., Graf, H. P., Guyon, I., Henderson, D., Howard, R. E., & Hubbard, W. (1989). Handwritten digit recognition: applications of neural network chips and automatic learning. *IEEE Communications Magazine*, 27(11), 41–46.

- Liang, Z., Li, Q., & Liao, S. (2016). Character-level convolutional networks for arithmetic operator character recognition. In *International conference on educational innovation through technology* (pp. 208–212).
- Lipton, Z. C. (2018). The mythos of model interpretability. *Queue*, 16(3), 31–57.
- Phong, L. T., Aono, Y., Hayashi, T., Wang, L., & Moriai, S. (2018). Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, 13(5), 1333–1345.
- Simonyan, K., Vedaldi, A., & Zisserman, A. (2013). Deep inside convolutional networks: Visualising image classification models and saliency maps. arXiv:1312.6034.
- Trask, A., Hill, F., Reed, S. E., Rae, J., Dyer, C., & Blunsom, P. (2018). Neural arithmetic logic units. In *Advances in neural information processing systems* (pp. 8035–8044).
- Walach, E., & Wolf, L. (2016). Learning to count with CNN boosting. In *The european conference on computer vision* (pp. 660–676).
- Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. arXiv:1708.07747.
- Yadav, C., & Bottou, L. (2019). Cold case: the lost MNIST digits. In *Advances in neural information processing systems* (pp. 13443–13452).
- Zaremba, W., Mikolov, T., Joulin, A., & Fergus, R. (2016). Learning simple algorithms from examples. In *International conference on machine learning* (pp. 421–429).
- Zaremba, W., & Sutskever, I. (2014). Learning to execute. arXiv:1410.4615.
- Zeiler, M. D. (2012). ADADELTA: an adaptive learning rate method. arXiv:1212.5701.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.