Batch size importance on the performance of Regression CNNs

LAMON Julien, KATO Yuko, TURAN Taylan, VIERING Tom, WANG Ziqi, LOOG Marco, TAX David

Delft University of Technology

J.A.D.Lamon@student.tudelft.nl,{Y.Kato, O.T.Turan, T.J.Viering, Z.Wang-8, M.Loog, D.M.J.Tax}@tudelft.nl

Abstract

TODO

***Keywords*—** Deep Learning, Convolutional Neural Network, Regression, Sensitivity Analysis, Batch Size

1 Introduction

A Convolutional Neural Network (CNN) is a "deep learning neural network designed for processing structured arrays of data such as images"[1]. The breakthrough of CNNs [2] was mostly seen in the field of image recognition. Its benefit and innovation come from the fact that it can efficiently detect features without any human supervision. Whilst CNNs have first been discussed in the 1980s [1], the arrival of autonomous cars and much more makes it the preeminence in its favourable field. However, while this method is very helpful, it is yet treated as a black box: studying its structure will not give any judgment on the structure being approximated. Although some scientists such as Zeiler and Fergus [3] came up with solutions to visualize the work of a CNN, various sections of it are still not fully understood. Furthermore, image recognition is a subject that can be treated in two different ways: classification and regression tasks. Recent interest has been growing in the classification alternative, such as Ye Zhang and Byron Wallace [4] where they evaluate the sensitivity of a CNN to its "input vector representations; filter region size(s); the number of feature maps; the activation functions; the pooling strategy; and regularization terms". However, until now, there have been only a few amounts of studies on the sensitivity of CNNs in a regression task [5].

As demonstrated by Satya Mallick and Sunita Nayak [6], the number of parameters a CNN contains is vast: as an example, the first successful CNN architecture that proved its usefulness - AlexNet [7] - contains over 60 million parameters in total. In consequence, a significant amount of research is being produced around the optimization of hyperparameters [8]. However, before diving into the optimization, there is a need to first improve our know-how of the network's sensitivity to specific hyperparameters. The aim of this work is therefore to get a more profound understanding of how sensitive CNNs are to the batch size, as well as why they are sensitive to them. More precisely, the analysis will be made over CNNs training on a regression task, namely: the mean, standard deviation and median. All experiments will be conducted on a baseline model, with the assumption that all hyperparameters - except for the batch size - do not have any impact on the model's performance.

The paper is structured as follows. To first have a broader understanding of the problem, section 2 will be charged of that. That is, it will give a bigger definition of a convolutional neural network, as well as sensitivity analysis. Then, in section 4, a detailed description of the methods will be given. This encircles the dataset used with the preprocessing made on it, with a high end description of the baseline model chosen, followed by the training and performance evaluation. In section 5, the experiments that were done as well as their results will be described. The responsible research paragraph will be given in section 6. A discussion on our results compared to the results of related works will be undergo in section 7. Finally, section 8 will conclude the paper and also give use some ideas of future work.

2 Problem Description

This section will develop to a greater extend the problem we are trying to answer, namely why is a regression CNN sensitive (or not) to different batch sizes. To fully understand the design and context there is first a need to have a more in-depth explanation of a Convolutional Neural Network. Then, the section will end with a detailed description on what is a sensitivity analysis.

2.1 Convolutional Neural Network

A Convolutional Neural Network (CNN) is a class of Deep Learning Methods. While an Artificial Neural Network (ANN) would need to flatten the data – being an image in our research context – and feed it in a network of the same size as the vector, a CNN is able to learn, and extract patterns present in an image by running a filter on an image. This gives the advantage of it being able to autonomously detect features without the need of human supervision. Moreover, CNNs are composed of multiple blocks, namely: convolution layers, pooling and fully connected layers. More generally, the model can be divided into two distinctive parts: feature extraction and the mapping of these extracted features to a final output, being in our case the mean, standard deviation, or median. We will now go through each of the distinctive parts.

Feature Extraction

This is the major part of the model. It is made out of a sequence of one or more convolutional and pooling layers.

The role of a convolutional layer is to extract features of a picture by the use of a kernel. That is, this kernel – also known as a filter – is being dragged on the picture, left to right, top to bottom, and a matrix multiplication between the kernel and each portion of the scanned image is made. The output matrix is then summed to obtain the output value at the position of the filter. The result of all output given by the kernel is called a feature map (as seen in figure 1). That feature map is then passed through an activation function, such as the ReLu, tanh, etc.

Pooling layers are used to reduce the special size given by the convolutional layer. This way, the number of learnable parameters

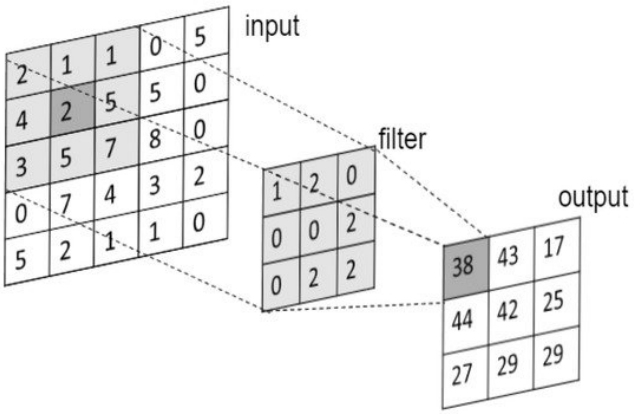

Figure 1: Outline of the Convolutional Layer [9]

is reduced. A vast amount of different pooling layers exists, but the common pooling layers are the average and max pooling layers. The latter is used in our research project due to its noise suppression ability. The max pooling function, as the convolutional layer, will "drag" a window (i.e. a filter/kernel) through the feature map returned by the ReLu function, and return the biggest digit in that window, as shown in figure 2.

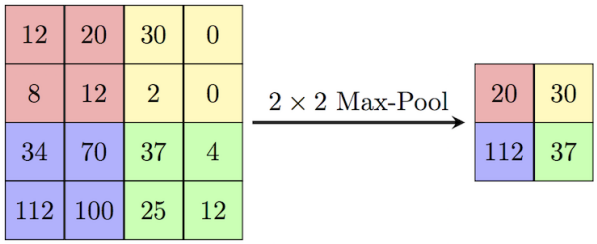

Figure 2: Pictoral representation of a Max Pooling layer

add source ?

Mapping features to an output

At the end of the aforementioned step, the output result of the last convolution or pooling layer is flattened to be fed to the Fully Connected Layer(s). The fully connected (FC) layer is a layer where ingoing/outgoing edges are equal to the number of neurons on the previous/next layer. The last layer then maps this flattened vector to final outputs, which can be the probabilities of the input image belonging to a specific label, or its computed mean.

2.2 Sensitivity Analysis

3 Related Work

4 Methods

Following a more profound description of the research problem, there is now a need to get a more detailed description of the method. This way, any skilled reader will be able to reproduce this setup and obtain the same results. The section will first start with a description of the dataset used during the research, as well as the pre-processing conducted before feeding it to the network. Then, as a second subsection, a more detailed description of the model will be given, with explanations on why such a custom model was created. In section 4.4, details on the training will be given. That is, details on parameters that will be kept constant for the whole research process. And finally, to end the section, an explanation of how the network is evaluated will be drawn.

4.1 Experimental Environment

In order to be able to fully replicate all experiments, there is a need to detail the experimental environment. Then, it should be noted that all experiments have been conducted on Windows 10 installed on a machine with an Intel Core i5-10400F 2.90GHz and 16GB RAM. In order to significantly improve the speed of training, a single RTX2060 GPU is used in our experiments, with the use of CUDA.

Considering the implementation of the CNN, the whole network is created using the python package named PyTorch. PyTorch is one of the strongest Deep Learning libraries, with Tensorflow on its side. With its support for C, C++ and Tensor computing, it permits to have a much faster model while still being easy to use. All experiments are publicly available in a GitHub repository [source].

4.2 Dataset

All experiments will be operated on the MNIST dataset [10], being a collection of images representing handwritten digits. This specific dataset contains 60,000 training images, with an additional 10,000 images for the test set. All images have already been preprocessed by these researchers [10], resulting in size-normalized and centred 28x28 images.

Since the project is focussing on the computation of regression tasks, labels given with the MNIST dataset are of no help to us. Therefore, a need to preprocess the dataset by ourselves was needed. Therefore, to approximate the mean, median, or standard deviation, these values were computed for all images.

4.3 Baseline Model

A baseline model was implemented following a tutorial called "MNIST Handwritten Digit Recognition in PyTorch"[11]. However, since this tutorial is concentrated on classification tasks, the code was modified to meet our needs. Furthermore, it was found during our research that dropouts are deactivated while the model is evaluating. The model was then changed to remove the dropout.

The model is constructed out of two sequential layers and then a set of fully connected layers (FC), as seen in figure 3. The two sequential layers are similar; that is, they both have one 2D Convolutional Layer with a kernel size of 5x5, following by a 2D max-pooling layer with a 2x2 sized kernel, and then a ReLu activation function. The only difference between the two sequential layers are the input and output channels: the first layer has one input and 10 output channels while the second layer has 10 input and 20 output channels. The first input channel is of size one because the MNIST dataset is grayscaled. Finally, a set of fully-connected layers is created. This set is comprised of two FC: the first contains 320 nodes and is connected to the second layer, which has 50 nodes and finished with only one, being the mean or std or, lastly, the median. It should be noted that right after the two sequential layers, the output is first flattened in order to be fed to the set of fully connected layer.

4.4 Training

With the help of PyTorch, discussed in the earlier "Experimental Environment" section, most of the hyperparameters are kept as default. In other words, the number of hyperparameters being set by ourselves is kept to a minimum to concentrate the most on the research question, and not the fine tune of the model. As an example, the 2D Convolutional Layer class provided by PyTorch [12] requires the following parameters:

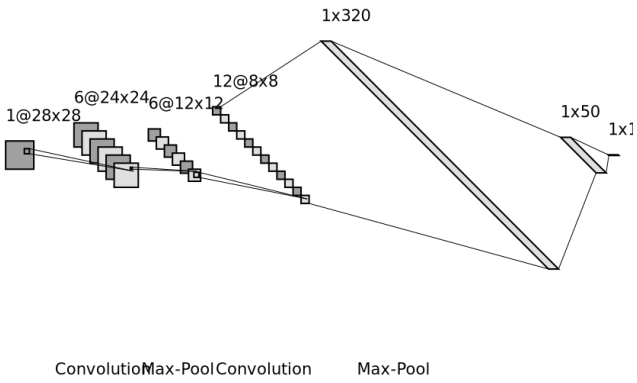

Figure 3: Outline of CNN model

torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')

However, we are only modifying the first three parameters, the rest is kept as default, as seen in the class above. This also applies to the pooling layers, fully connected layers, ReLu activation function, and the optimizer used.

Concerning the optimizer, the choice was being considered between the Stochastic Gradient Descent (SGD) and the Adaptive Moment Estimation (Adam)

add a definition of SGD and Adam ?

. To make up our mind on which optimizer to choose, we considered the results of each of them while computing the mean of images in the MNIST dataset, as seen in fig

ADD FIGURE

. It can therefore be seen that the Adam optimizer is achieving a lower Mean Squared Error loss, compared to the SGD. We, therefore, chose to continue with the Adam optimizer.

It should be noted that no validation set has been used in the training of the model. The reason is that the use of a validation set is to evaluate different models on it to choose the best performing one. However, due to our research question a baseline model is needed to have its performance differing only due to the parameters manually changed. Therefore, since the model is decided beforehand, a validation set is not needed.

4.5 Performance Evaluation

To test the model, we had to chose between various regression loss function. After some research the choice was reduced to only two functions: the Mean Squared Error (MSE) and Mean Absolute Error (MAE) losses. The MSE is the commonly used regression function, being the sum of squared distances between the targeted and predicted values: $\frac{1}{n}\sum_{t=1}^{n} e_t^2$. On the other side, the MAE is the sum of absolute differences between the targeted and predicted values: $\frac{1}{n}\sum_{t=1}^{n} |e_t|$.

The choice to continue with the MSE was made due to the results found in figure

ADD FIGURE

. This figure shows that the MAE is not well performing compared to the MSE, with more spikes, etc.

5 Results

TODO

6 Responsible Research

TODO

7 Discussion

TODO

8 Conclusions and Future Work

TODO

References

[1] Thomas Wood. Convolutional neural network, May 2019.

[2] Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[3] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *Computer Vision – ECCV 2014 Lecture Notes in Computer Science*, page 818–833, 2014.

[4] Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification, 2016.

[5] Stephane Lathuiliere, Pablo Mesejo, Xavier Alameda-Pineda, and Radu Horaud. A comprehensive analysis of deep regression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(9):2065–2081, 2020.

[6] Satya Mallick and Sunita Nayak. Number of parameters and tensor sizes in a convolutional neural network (cnn): Learn opencv, Apr 2021.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[8] Erik Bochinski, Tobias Senst, and Thomas Sikora. Hyperparameter optimization for convolutional neural network committees based on evolutionary algorithms. 09 2017.

[9] Saifullahi Aminu Bello, Shangshu Yu, Cheng Wang, Jibril Muhmmad Adam, and Jonathan Li. Review: Deep learning on 3d point clouds. *Remote Sensing*, 12(11), 2020.

[10] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[11] Gregor Koehler. Mnist handwritten digit recognition in pytorch, Feb 2020.

[12] Conv2.