Is the batch size affecting the performance of Regression CNNs ?

LAMON Julien[1] , **KATO Yuko**[1] , **TURAN Taylan**[1] , **VIERING Tom**[1] , **WANG Ziqi**[1] , **LOOG Marco**[1] , **TAX David**[1]

[1]Delft University of Technology

Abstract

With an expectation of 8.3 trillion photos stored in 2021 [1], convolutional neural networks (CNN) are beginning to be preeminent in the field of image recognition. However, with this deep neural network (DNN) still being seen as a black box, it is hard to fully employ its capabilities. A need to tune hyperparameters is required to have a robust CNN that can more accurately do its task. In this study, the batch size, being one of the most important hyperparameters, is our main concern. The batch size is the number of samples that will be propagated through the network before updating the weights. Moreover, we show how the batch affects the performance of Regression CNNs to the following regression tasks: the mean, median, standard deviation (std) and variance of the pixel intensities of a grey-scale MNIST[2] input image. This will be analyzed by how well regression CNNs converge, given different batch sizes and a fixed learning rate. Additionally, we will also be comparing the final mean squared error given by all different batch sizes. At the end of the research, our findings concluded that a higher batch size leads to a higher Mean Squared Error (MSE) and a slower convergence. Additionally, the best performance obtained was for batch sizes of size 8 to 32, with slight differences between the four different regressions tasks.

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"[3]. The breakthrough of CNNs [4] was mostly seen in the field of image recognition. Its benefit and innovation come from the fact that it can efficiently learn features that are not constructed by human experts. Whilst CNNs have first been discussed in the 1980s [3], 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 task being approximated. Although some scientists such as Zeiler and Fergus [5] 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 [6] 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 [7].

With, for example, the AlexNet [8] containing over 60 million parameters in total, a significant amount of research is being produced around the optimization of hyperparameters [9]. However, before diving into the optimization, there is a need to first improve our knowledge of the network's sensitivity to specific hyperparameters. The aim of this work is therefore to get a better understanding of how sensitive CNNs are to the batch size. Sensitivity, as described by Maosen Cao, Nizar F. Alkayem, Lixia Pan and Drahomir Novak [10], is how a slight variation in the input parameters shows a significant response to the results of the model. More precisely, the analysis will be made over CNNs training on a regression task, namely: the mean, median, standard deviation (std) and variance of the pixel intensities of a grey-scale MNIST[2] input image. All experiments will be conducted on a baseline model with fixed hyperparameters, except for the batch size and the learning rate. The learning rate is being fine-tuned for every regression task, on a batch size of 32 samples.

2 Problem Description

This section develops to a greater extent the problem we are trying to answer, namely, if the batch size affects the performance of regression CNNs. 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 ends with a description of the main hyperparameters related to the training of the baseline model.

2.1 Convolutional Neural Network

A Convolutional Neural Network (CNN) is a class of Deep Learning Methods. While a Multilayer Perceptron (MLP) 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 can learn, and extract patterns present in an image by running a filter on an image. This gives the advantage of it being able to detect features having pixel dependencies. Moreover, CNNs are composed of multiple blocks, namely: convolution layers, pooling and fully connected layers. Their respective role can be found below:

- **Convolution Layer**: extracts features of a picture by applying filters, also known as a kernel.

- **Pooling Layer**: used to reduce the spatial size given by the convolutional layer. With a vast amount of different pooling layers, it should be noted that we use the max-pooling function in our research.
- **Fully Connected Layer (FC)**: the flattened results are fed to the FC and then mapped to final outputs, which in our situation is the computed mean, median, standard deviation, or variance.

A more profound explanation of CNNs can be found in the paper of Yamashita, Nishio, Do and Togashi [11].

2.2 Hyperparameters

To correctly train the model, many hyperparameters need to be adjusted. A hyperparameter is a value set before the learning process that is internal to the model. A correct adjustment of hyperparameters enables the model to reach its best performance, and reduce the time to convergence. The main hyperparameters related to the training are the following: the learning rate, momentum, number of epochs, and batch size.

The learning rate is a hyperparameter that defines the rate at which a CNN updates its weights during training. It is a crucial parameter to tune since a too small learning rate leads to a longer training phase and it might get stuck in a local minima, while a larger learning rate might lead to an unstable training process. An explanation of how this hyperparameter is tuned can be found in the Results section.

The number of epochs is defined as the number of times the whole train set will be fed to the network. This hyperparameter is important since a lack of epochs will result in an underfitting problem, while on the other hand, too many epochs will lead to overfitting. The number of epochs for our research and every experiment is set to 10, as seen in the Results section.

The batch size, one of the main and most important parameter that need to be tuned, is the central problem in the research. The batch size is the number of samples that will be propagated through the network before updating the weights. As said by Ibrahem Kandel [12], "setting this hyperparameter too high can make the network take too long to achieve convergence [...]; however, if it is too low, it will make the network bounce back and forth without achieving acceptable performance". While this comment gives us already some clarity on the sensitivity of the model to the batch size, it is, unfortunately, being stated on a classification task. In this research paper, we will then confront this statement on different regression tasks, as seen in section 4.2. In addition to the faster convergence of the model it should be noted that, for some datasets, we are unable to fit the entire model at once in the memory. This is then another advantage given by the batch size.

3 Related Work

With the AlexNet architecture containing over 60 million parameters in total [13], a vast amount of parameters needs to be set up and adjusted. With the batch size being one of the most dominant ones [14], it is already known that a smaller batch size reduces the time of convergence with the disadvantage of getting stuck in a local minima and that a larger batch size might make the model take too long to converge. Yet, only a few research papers are being produced on the topic. Below is a research paper that investigates the effect of batch size on the performance of CNNs for image classification, therefore being a close subject to our research question.

In 2020, Ibrahem Kandel and Mauro Castelli published a paper named "The effect of batch size on the generalizability of the convolutional neural networks on the histopathology dataset". In this paper, the authors investigated the accuracy of other researchers' claims.

Firstly, the authors state the paper of Radiuk [15], which goal is to "find an impact of training set batch size on the performance". The use of the MNIST and CIFAR-10 datasets are both used for consistent results, as well as different CNN architectures. The experiments were all produced on powers of 2 varying from 2^4 to 2^{10}, as well as the following batch sizes: [50, 100, 150, 200, 250]. With the SGD optimizer, and a learning rate of 0.001 and 0.0001 for the MNIST and CIFAR-10 dataset, respectively, the results concluded that "the greater the parameter value, the higher the image recognition accuracy". More specifically, it is claimed that the optimal batch size is of 200 samples or greater.

Then, the authors consider the results found in the work of Masters and Luschi [16]. Their research is concentrated on comparing the test performance of different batch sizes through various experiments. Specifically, their experiments were all performed on three different datasets, namely the CIFAR-10, CIFAR-100 and ImageNet dataset. With the use of different CNN architectures (AlexNet, ResNet and ResNet-50), learning rates in the range $[2^{-12}; 2^0]$, the SGD optimizer, and the same number of epochs for all experiments, it was concluded that the models were all performing better when operating on batches between 2 and 32 samples. Moreover, the authors claim that their results "contrast with recent work advocating the use of mini-batch sizes in the thousands". It was also stated that increasing the batch size leads to a lower "range of learning rates that provide stable convergence and acceptable test performance".

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 section will therefore get more in depth with what was the experimental environment of the research, followed by a description of the datasets used. Then, a more detailed description of the model is given, with explanations on why such a custom model was created. In section 4.4, details on parameters that are kept constant for the whole research process is given. And finally, to end the section, an explanation of how the network is evaluated is explained.

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 [17]. All experiments are publicly available in the following GitHub repository: github.com/Jlamon/sensitivity_batch_size_regression_cnn.

4.2 Dataset

All experiments will be operated on the MNIST dataset [2], 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 [2], 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. To approximate the mean, median, standard deviation or variance,

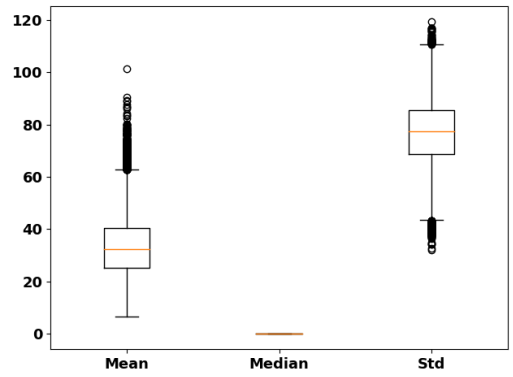


Figure 1: Distribution of computing the mean, median and standard deviation of the image pixel intensities of the train MNIST dataset.

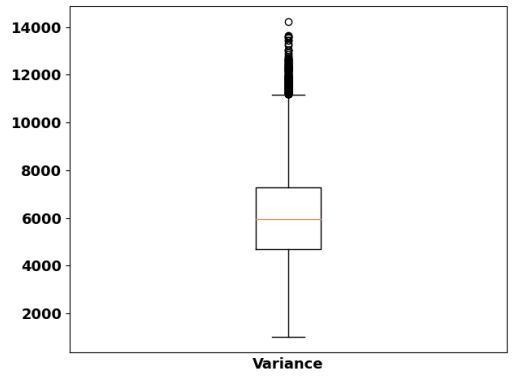


Figure 2: Distribution of computing the variance of the image pixel intensities of the train MNIST dataset.

these values were computed for all images. To facilitate the understanding of what regression task is being approximated, a various number of abbreviations will be used, as listed below:

- **MeanMNIST**: this abbreviation is used when the model is approximating the mean of the pixel intensities of a grey-scaled MNIST input image. The formula is as follows: $\frac{\text{Sum of all pixel values}}{\text{Number of pixel values}}$
- **MedianMNIST**: this abbreviation is used when the model is approximating the median of the pixel intensities of a grey-scaled MNIST input image. The median is the middle value of the set of ordered pixel values. In other words, it is the $\frac{n+1}{2}^{th}$ value.
- **StdMNIST**: this abbreviation is used when the model is approximating the standard deviation of the pixel intensities of a grey-scaled MNIST input image. The formula is as follows: $\sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2}$. Where x_i is the i^{th} pixel value in the image, $\overline{x}$ is the mean value of the pixel intensities of the image, and N is the number of pixel values.
- **VarMNIST**: this abbreviation is used when the model is approximating the variance of the pixel intensities of a grey-scaled MNIST input image. The formula is as follows: $\frac{\sum_{i=1}^{N}(x_i - \overline{x})^2}{N}$.

An observation should be made on the various distribution of these preprocessed datasets. It can be seen in figure 1 that the distribution of the MeanMNIST and StdMNIST dataset are quite resemblant. However, for the MedianMNIST dataset, we can see that the distribution is almost null, therefore showing us that the experiments conducted on this dataset might not give us helpful results for our research question due to its inaccuracy. Then, as seen in figure 2 the distribution of the VarMNIST dataset needs to be set apart from the others due to its large difference in the MSE loss.

4.3 Baseline Model

A baseline model was implemented following a tutorial called "MNIST Handwritten Digit Recognition in PyTorch"[18]. A modification in the code was needed to fit our needs. The adjustments made concerned the removal of the dropout layer, but most importantly the need to adapt the model to regression tasks.

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 gray scaled. 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.

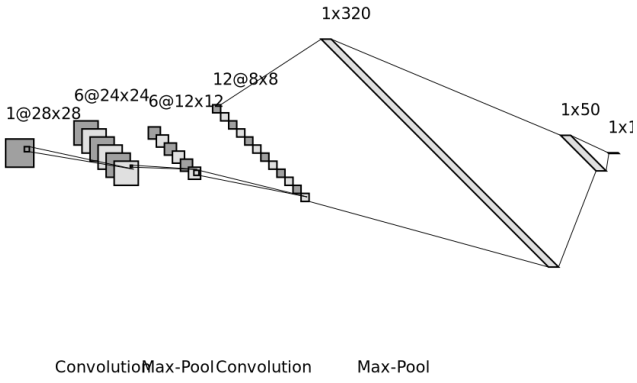


Figure 3: Outline of CNN model

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 is kept to a minimum to concentrate the most on the research question, and not the fine-tuning of the model. As an example, the 2D Convolutional Layer class provided by PyTorch [19] requires the following parameters:

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

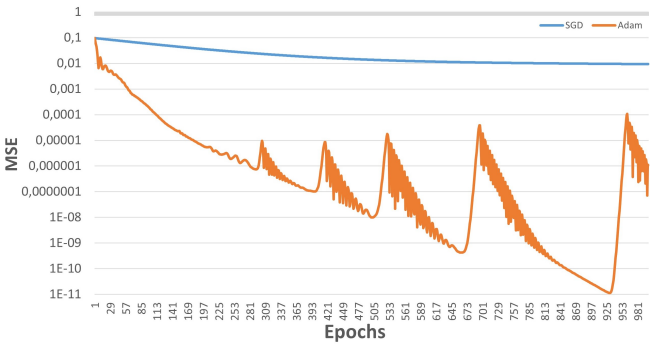


Figure 4: The testing accuracy of two different optimizers (SGD and Adam) over a fixed number of epochs [1]

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). In figure 4 we see that the Adam optimizer is achieving a lower MSE than the SGD. The Adam optimizer is then chosen for the rest of the experiments.

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

A regression loss function should be picked to evaluate if the predicted labels correspond to the targeted label. With a vast choice of regression loss functions such as the Mean Squared Error (MSE), Mean Absolute Error (MAE) or Smooth Mean Absolute Error, it is chosen to continue our experiments with the most commonly used regression loss function, namely the MSE.

5 Results

In the interest of answering the research question, we executed several experiments. It is in this section that we present the multiple evaluations performed to determine how the batch sizes affect the performance of Regression CNNs. All results given in this section are created using the methods described in section 4.

For each regression task (mean, median, std and variance), the learning rate of the baseline model is tuned considering a batch size of 32. The learning rates first tested are on the log-scale curve:

$$[0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1]$$

Then, considering the best result found on the log-scale, values around it are tested as well. As an example, if the model trains better with an initial learning rate of 0.01, these following values are then tested:

$$[0.0025, 0.005, 0.0075, 0.01, 0.015, 0.0175, 0.02]$$

Graphs displaying the results of these different learning rates can be found in Appendix A.

As in the paper of Pavlo M. Radiuk [15], the different batch sizes that will be tested are all on the power of 2. However, to have a broader view of the sensitivity, the number of batch sizes being tested is augmented, resulting in the following array:

$$[2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048]$$

For each one of them, the model is trained from scratch and the accuracy is tested at the end of each epoch, as well as before any prior training. It should be noted that we mostly added smaller batch sizes, namely the values 2, 4 and 8 since it was noted in the early experiments that the higher the batch size, the higher the MSE loss. We, therefore, wanted to try and observe an increase in the MSE loss for smaller batch sizes.

In the end, a total amount of four experiments will be performed, with each of them being conducted on the four regression tasks. For each experiment, a different set of graphs is given to compare the results. For each experiment and batch size, the baseline model is trained for 10 epochs. A higher number of epochs would result in the model overfitting.

5.1 Experiment 1: How well do they converge ?

In this experiment, graphs are created to see if there are any poor fluctuations in the test loss. This experiment is conducted because the sensitivity of a model to different batch sizes can be noticed by how the model converges. Then, if there are any significant differences between the batch sizes, some hypothesis can be deducted.

Figure 5 visualizes the trend of different batch size on the four constructed datasets, namely the MeanMNIST, MedianMNIST, StdMNIST and VarMNIST datasets. Therefore, each graph in figure 5 represents the results of the baseline model approximating the mean, median, standard deviation or variance of the pixel intensities of its input images. It should be noted that the curves - representing the MSE at the end of each epoch - are computed on the test set. Additionally, the baseline model applies the best performing learning rate found for each regression dataset, with a batch size of 32, as seen in appendix A.

For our experiments, it can be observed that there are two different trends in the performance of the four datasets. It can be seen that the convergence in the mean and variance is converging normally, with still the highest batch sizes (such as the batch size with 1024 samples) having an inferior performance to the small batch sizes, such as a batch size of 64 samples.

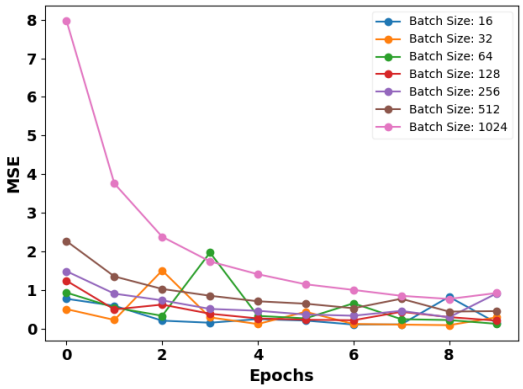
Considering the second trend, being observed on the StdMNIST and MedianMNIST datasets, we can see that the curves are not as smooth as for the MeanMNIST and VarMNIST dataset. Again, just as for the first trend, the larger the batch size value, the higher the MSE result. It is apparent that for both figures 5(b) and 5(c), the noisiest batch size value is 512 samples, followed by 1024 samples.

The main difference between the two trends is therefore not in the results but its global convergence. To these specific experiments, we can state that the StdMNIST and MedianMNIST are being complex to compute for the baseline model. Conclusions can be drawn from the distribution of the synthetic datasets, as seen in figures 1 and 2. But this will all be discussed in the Discussion section.

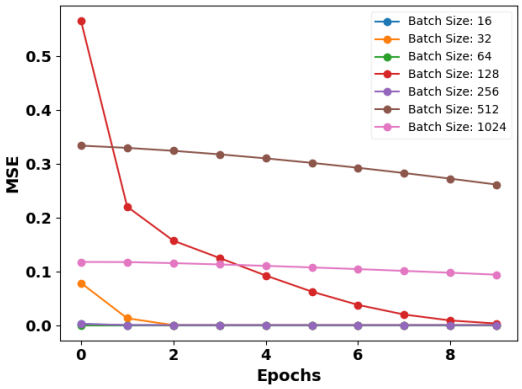
5.2 Experiment 2: The time it takes to converge

The second experiment that is performed concerns the effect of the batch size on the time the model takes to train. As seen in figure 6, the graph only represents results computed on the MeanMNIST dataset. That is because the convergence of the time is similar for all regression tasks, therefore making it counterproductive to display the results of all task. Then, as seen in figure 6, the bar charts visible in red represents the results of the MSE computed at the very end of the training phase. On the other hand, the bar charts visible in blue represents the convergence of the time the model takes to train, in seconds.

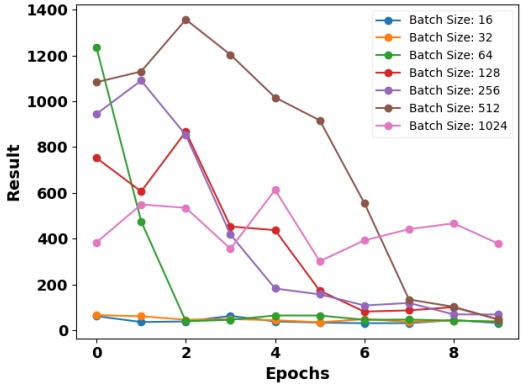
[1] Credits to Remco den Heijer for the Graph Making



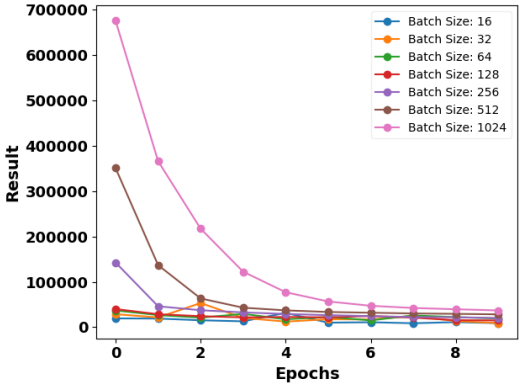
(a) The convergences of different batch sizes on the MeanMNIST dataset. The best performing batch size is of 64 samples



(b) The convergences of different batch sizes on the MedianMNIST dataset. The best performing batch size is of 32 samples.



(c) The convergences of different batch sizes on the StdMNIST dataset. The best performing batch size is of 16 samples



(d) The convergences of different batch sizes on the VarMNIST dataset. The best performing batch size is of 256 samples

Figure 5: The convergence of testing different batch sizes on the four regression tasks. Figure 5(a) represents the results of testing one the MeanMNIST dataset, following with figure 5(b) on the top right corner, testing on the MedianMNIST dataset. On the bottom-left corner, figure 5(c) is picturing the results on the StdMNIST dataset. Finally, figure 5(d) displays the results on the VarMNIST dataset.

What can be observed from the figure is that the time greatly converges as the batch size is growing. The first batch size – a batch size of 2 samples – takes almost 600 seconds, being 10 minutes while the largest batch size – a batch size of 2048 samples – takes less than 100 seconds, being a minute. This, therefore, shows a difference by a factor of 10, which should not be negligible. However, as we can see, as the batch size is larger, the MSE is growing up as well, therefore stipulating that the model might need more time to converge.

5.3 Experiment 3: Sensitivity of CNN to different batch sizes

The third experiment, being the most important one, represents the last MSE loss registered for each different batch size. Then, as seen in figure 7, it can be observed that a slight increase in the error can be seen for smaller batch sizes. A significant increase in the error rate for larger batch sizes can be noticed. It should be noted that the computation of the variance is separated from the computation of the mean, median and standard deviation due to the notable difference in the MSE. While the maximum value in figure 7(a) is 4, the maximum value in figure 7(b) is over 80.000. This then means that compiling the two graphs would result in an inability to compare the results.

As a first look, it can be seen that, globally, the MSE loss is decreasing while going from a batch size of 2 to a batch size of 4, before it gradually increases, except for the median. Again, the median might behave this way due to its distribution, as seen in figures 1 and 2. The computation on the StdMNIST and MeanMNIST do also continue to decrease while going from a batch size of 4 to 16, but the decrease is only by a difference of 0.1 in the MSE loss.

5.4 Experiment 4: Error difference in 10 different runs

Sensitivity can also be measured by how different results are on multiple runs. That is, the bar charts seen in figure 8 are representing the mean results given by the last epoch of the model, while the error bars indicate the standard deviation. Moreover, the baseline model is being trained ten times from scratch, and only the MSE of the last layer is kept. Then, the mean and standard deviation of each batch

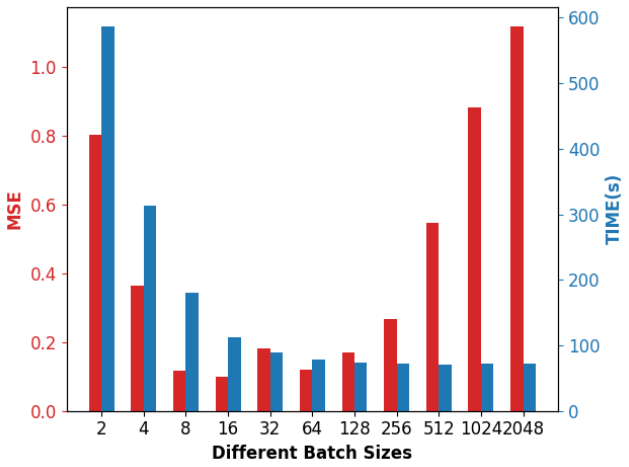


Figure 6: Results from the computation on the MeanMNIST dataset. The MSE is represented in red while the time it takes to train the model is represented in blue. The time is represented in seconds.

size is computed and represented on the graph. It should be noted that for each run, a new random seed is being created. The digit used to set the random seed is computed as follows: current batch size + current run index.

From the graphs, it can be seen that there are similarities to the graphs in figure 6, such as the fact that the larger the batch size value, the higher the MSE loss. But, also, we can observe some clear differences, such as in figure 8(a) with a standard deviation around 0.5 while the maximum mean value is about 0.9. This clearly shows some sensitivity to this specific batch size.

Again, as for the other figures (figures 7(a) and 5(b)), the computation on the MedianMNIST dataset shows some severely small results, therefore making us question if figure 8(b) is accurate. It can be observed that the results on the small batch sizes (batch sizes from 16 to 64) gives results of 0.0 due to a too-small number that a float datatype is unable to handle. We also indicate a relative difference in the following batch sizes, with the standard deviation being mostly higher than the mean, therefore suggesting an inability to interpret the results.

As for the two other graphs (figure 8(c) and 8(d)), we can see less noise in the results than in figure 8(a) and 8(b). We can however witness a high standard deviation for the batch size of value 64 and 128 in figure 8(c). It can also be indicated that with the high MSE loss of the VarMNIST in figure 8(d) the standard deviation is much more important than for the other figures, such as a standard deviation of approximately 10.000 for a batch size of 32 samples.

6 Discussion

This research paper answers the question of whether CNNs are affected by the batch size hyperparameter or not. Various experiments have been conducted to answer this question, as seen in the Results section. These results, all being conducted on the very same baseline model but having its learning rate and batch size hyperparameters changed, indicate that smaller batch sizes do give better results than the larger ones. Except for the time to convergence, where time is steadily decreasing before stagnating after a batch size of 64 as seen in figure 6, it can be seen that batch sizes in the interval [2; 64] give considerably better results before starting to increase. Moreover, in the interval stated before, it must be observed that the batch sizes 2, 4 and 8 are generally giving the least favourable results in the 6 batch sizes studied in the interval. However, the change is significantly smaller than the differences in the MSE loss found between batch sizes [2; 64] and [128; 2048]. As an example, for the std in figure 7(a), the worst MSE in the first interval is only around 0.7, while the worst result in the second interval is almost 4. The difference is then by a factor of around 0.57. This difference is even bigger for the variance (as seen in figure 7(b)), where the factor is 4, considering that the maximum error loss in the first interval is 20.000 and the second interval is over 90.000.

Concerning the noise found in the standard deviation in figure 8, it can be seen that the distribution is quite different for all regression tasks. A hypothesis as to why such noise appears can be explained by their distribution. As it can be seen, when the distribution is dense, as for the MedianMNIST in figure 1, the noise is much higher than for the VarMNIST dataset, which, a contrary, has a much more sparse distribution, as seen in figure 2, where the variance of the pixel intensities of the input image is between around 1000 and 14000.

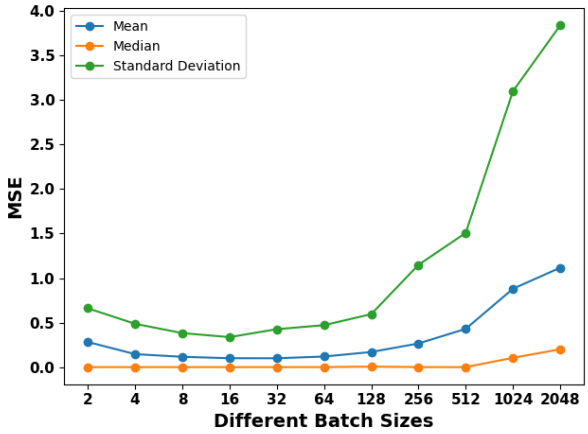
With previous research papers discussed in the related work section focusing on classification tasks, it is then natural to try and find correlations between CNNs performing classification tasks, and regression tasks. Then, it can be seen that our results do not agree with the conclusions declared by Radiuk [15], stating that the optimal batch sizes are of 200 samples or greater. It should however be noted that this statement is accompanied by the fact that the author does also increase the learning rate, and that it requires more computational power. This is therefore an issue that should be resolved in future research, being the fact that the learning rate used by each regression task is fine-tuned only on a batch size of 32, being suggested by Bengio [20] as a good default value. With a fine-tuned learning rate and more epochs, the MSE loss might be lower than the results found with a fixed learning rate, as done in our research.

Then, concerning the paper of Masters and Luschi [16], it can be seen that our research results relate more to this paper. However, our results yet do not agree completely with each other. As the authors are stating, the optimal results are obtained when training the model with a batch size of 32 samples or lower, with the batch sizes of 2 and 4 being usually performing better than the rest. This is therefore not applying to us; as seen in figure 7(a) and 7(b), our optimal batch sizes are between 8 and 32, with this interval differing fairly between the four different regression tasks.

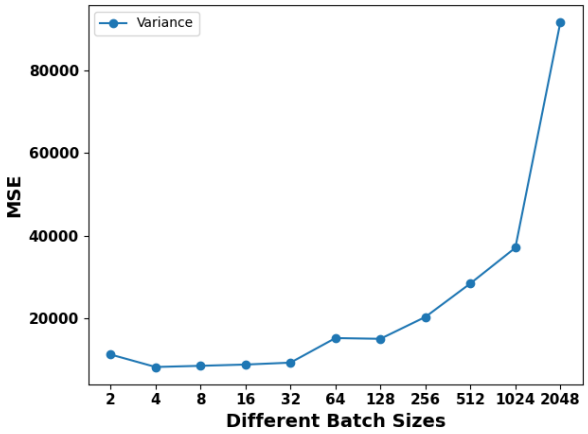
Concerning the limitations, the generalizability of the results is limited by the fact that only the MNIST dataset is used for our experiments. The main issue with this dataset is the fact that all digits are greyscaled, therefore greatly limiting the differences in the pixel intensities of the input image. While a pixel value in a grayscaled image is only represented by a single number, serving as the brightness of the pixel, a coloured image, on the other hand, needs to be represented by a vector of three number, if the RGB format is used. This would then increase the complexity of the task given to the baseline model, which might have some effect on the performance, as well as its rapidity to convergence. A further improvement for the research would then be to test our model on coloured images from the CIFAR-10 dataset, for example.

Another improvement would be to test our model on a synthetic dataset created entirely for this research. That specific dataset would not be representing any object or digit, but would only be containing randomly chosen pixel intensities. This would then show the limitations in the model approximating any of the regression tasks used in this research.

Nonetheless, the results found in this research paper are yet valid for answering our research question, being rather or not the batch size is affecting the performance of Regression CNNs.



(a) The results of the baseline model approximating the MeanMNIST, MedianMNIST and StdMNIST datasets, with different batch size on a base 2 logarithmic scale



(b) The results of the baseline model approximating the VarMNIST dataset, with different batch sizes on a base 2 logarithmic scale.

Figure 7: The results of testing different batch sizes on the baseline model. Figure 7(a) is representing the results of the baseline model computing the MeanMNIST, MedianMNIST and StdMNIST datasets. The results are being computed with a learning rate of `0.0015`, `0.1` and `0.001`, respectively. The results computed on the VarMNIST dataset are displayed in figure 7(b) due to the considerable gaps in the results.

7 Responsible Research

In addition to the time taken to represent and discuss the results found during the research, an appropriate amount of time was taken to represent the truth to its fullest, and to cite the work and code established by others. That is, all information is given to prove that results and representations can be reproducible on any other machine if the conditions are met.

Concerning the code, it was already noted that the code was taken from a blog, namely "MNIST Handwritten Digit Recognition in PyTorch" [18]. It was then changed, with the help of Julian Biesheuvel, Remco den Heijer and Ratish Thakoersingh, the teammates with whom I indirectly worked during the whole research project. As the baseline was created, each teammate forked from it, and work on his specific research question. It is then that the code, followed with its experiments, plots and saved states of the model, can be publicly found in a GitHub repository [source], all sorted by different weeks. Additionally, a profound explanation of experiments, with the hyperparameters used and why can be found in the previous sections 4.

Concerning the main python package, PyTorch, with its strong community behind it, makes the source code easy to learn and understand. This advantage, therefore, enables anybody to reproduce the results if they have a powerful enough machine.

With all the conditions met, this is research can be fully reproduced with similar results. A minor difference in the results can be found if another GPU is used, with mostly a change in the time taken to train the model. Also, due to rounding errors, minor differences can be found in the MSE/MAE results, but yet having a global similarity to the results shown in this research paper. Finally, some results might also change due to the random seeds.

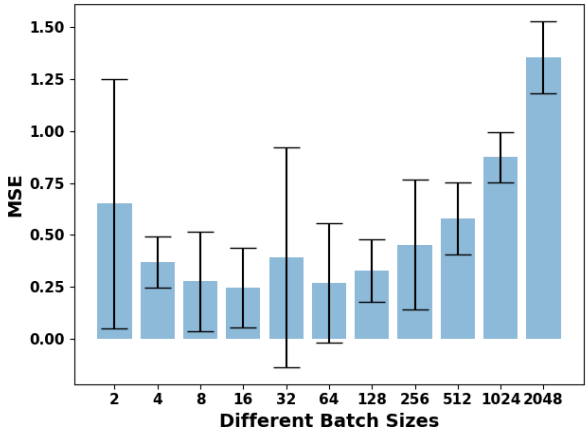
Furthermore, the experiment does not involve human subjects, thus unethical human research cannot occur.

8 Conclusion

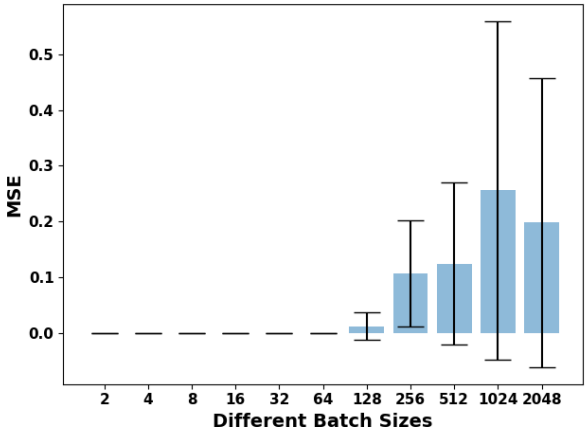
This research paper presented a study on whether or not the batch size affects the performance of Regression CNNs. With our experiments, we then reviewed the correlations found with the related work, being performed on classification tasks.

The current results demonstrated that the batch size has a significant effect on the performance of CNNs. It was observed that the optimal batch sizes were generally found in the interval of [8; 32], with this interval differing fairly between the four different regression tasks. On the other hand, it should be noted that the MSE is higher for batch sizes of 2 and 4 samples before decreasing, therefore contradicting the findings of Masters and Luschi [16], stating that the optimal batch sizes are of size 32 or lower, and more often as small as 2 or 4.

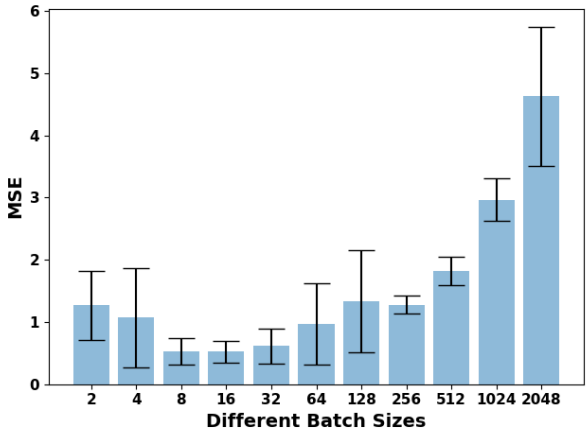
While our results highlight the fact that larger batch sizes do not perform well, due to the significant difference in the MSE loss, it should yet be taken as a grain of salt. It should therefore be noted that the learning rate, being with the batch size one of the main hyperparameters, was fine-tuned on a batch size of 32, instead of fine-tuning it for all batch sizes. This might then result in a learning rate not being set right for higher batch sizes.



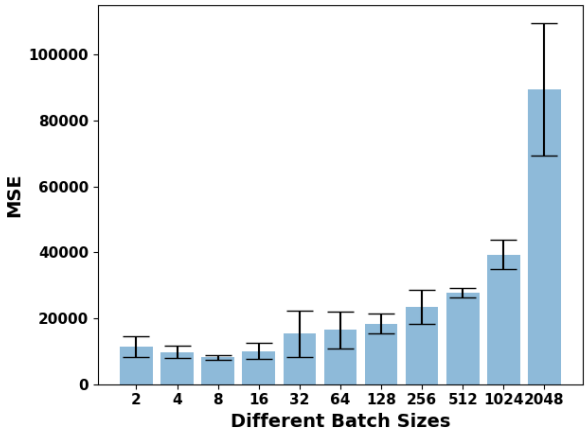
(a) Results of testing different batch sizes - on the MeanMNIST dataset - on a base 2 logarithmic scale.



(b) Results of testing different batch sizes - on the MedianMNIST dataset - on a base 2 logarithmic scale.



(c) Results of testing different batch sizes - on the StdMNIST dataset - on a base 2 logarithmic scale.



(d) Results of testing different batch sizes - on the VarMNIST dataset - on a base 2 logarithmic scale.

Figure 8: The testing accuracy of different batch sizes tested 10 times with different random seeds. The bar charts represent the mean results, with the error bar displaying the standard deviation. Figure 8(a) represents the results of the baseline model computing the MeanMNIST dataset, following with figure 8(b) on the top right corner, testing on the MedianMNIST dataset. On the bottom-left corner, figure 8(c) is picturing the results on the StdMNIST dataset. Finally, figure 8(d) displays the results on the VarMNIST dataset.

References

[1] David Carrington. How many photos will be taken in 2021?, Mar 2021.

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

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

[4] Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdelrahman 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.

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

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

[7] 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.

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

[9] Erik Bochinski, Tobias Senst, and Thomas Sikora. Hyperparameter optimization for convolutional neural network committees based on evolutionary algorithms. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 3924–3928, 09 2017.

[10] Maosen Cao, Nizar F. Alkayem, Lixia Pan, and Drahomír Novak. Advanced methods in neural networks-based sensitivity analysis with their applications in civil engineering. *Artificial Neural Networks - Models and Applications*, 2016.

[11] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, 9:611–629, 8 2018.

[12] Ibrahem Kandel and Mauro Castelli. The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset. *ICT Express*, 6:312–315, 12 2020.

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

[14] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *PMLR*, pages 448–456, 6 2015.

[15] Pavlo M. Radiuk. Impact of training set batch size on the performance of convolutional neural networks for diverse datasets. *Information Technology and Management Science*, 20, 1 2018.

[16] Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks, 2018.

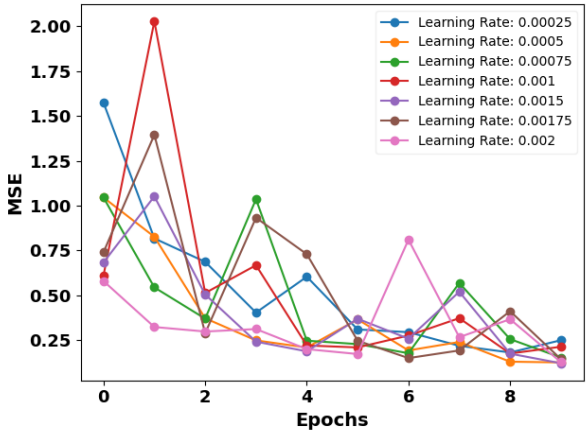
[17] Stephen Welch. Pytorch vs. tensorflow: What you need to know, May 2020.

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

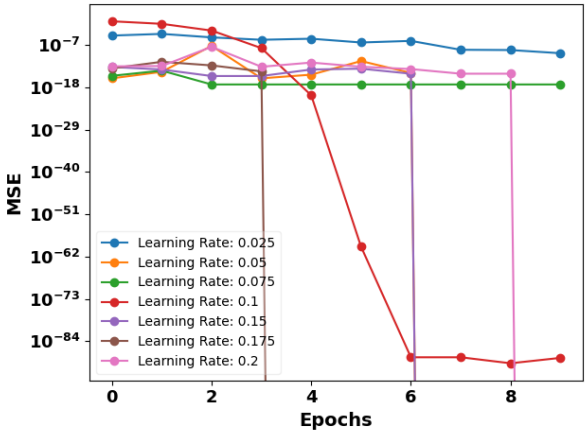
[19] Conv2. *Conv2d - PyTorch 1.8.1 documentation*.

[20] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7700 LECTURE NO:437–478, 2012.

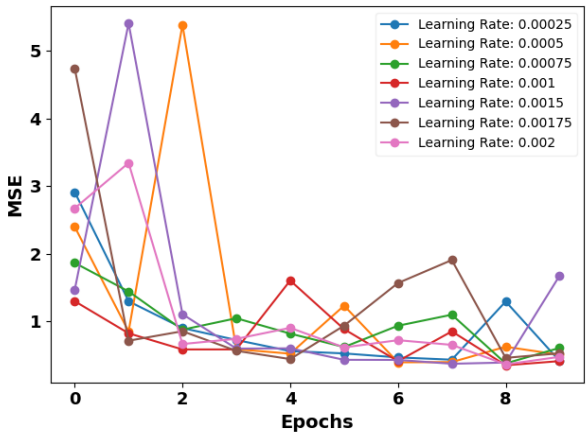
A Learning rate results



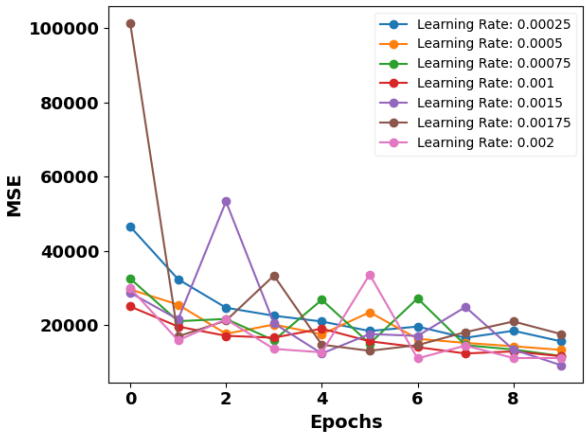
(a) The results of testing the learning rates around the best performing learning on the log-curve (`0.001`) on the MeanMNIST dataset. The best performing learning rate is found to be `0.0015`



(b) The results of testing the learning rates around the best performing learning on the log-curve (`0.1`) on the MedianMNIST dataset. **This graph is log-scaled** due to the sparse differences between all learning rates. The best performing learning rate is found to be `0.1`. The learning rates `0.175`, `0.15` and `0.2` is not chosen due to the rounding error leading to a MSE of `0.0`



(c) The results of testing the learning rates around the best performing learning on the log-curve (`0.001`) on the StdMNIST dataset. The best performing learning rate is found to be `0.001`



(d) The results of testing the learning rates around the best performing learning on the log-curve (`0.001`) on the VarMNIST dataset. The best performing learning rate is found to be `0.0015`

Figure 9: The results of testing the learning rates on the four regression tasks. Figure 9(a) represents the results of testing on the MeanMNIST dataset, following with figure 9(b) on the top right corner, testing on the MedianMNIST dataset. On the bottom-left corner, figure 9(c) is picturing the results on the StdMNIST dataset. Finally, figure 9(d) displays the results on the VarMNIST dataset.