Investigate the influence of batch size and learning rate on a simple deep learning model

Aurélien Chassagne under supervision of Olivier Debeir* & Adrien Foucart*

Abstract—This paper focuses on the influence of learning rate and batch size on the learning process on a basic deep learning architecture for computer vision. A dataset is randomly created for a trivial object detection problem. A small sequential architecture model consisting of convolution and max pooling layers is trained using the Intersection over Union metric. The model is trained on different combinations of batch size and learning rate to evaluate their influence on the accuracy curve and training time. The results show a certain correlation between batch size and learning rate, as well as the importance to use an architecture adapted to the problem. In short, it is possible to say that the batch size mainly influences training time and seems to give an upper bound for accuracy, the learning rate mainly influences accuracy. In addition, the influence of the learning rate is proportional to the batch size. Therefore, it is better to first choose a batch size and then find a learning rate with a good trade-off between cost and accuracy.

Index Terms—Deep Learning, Batch Size, Learning Rate, Computer Vision, Object Detection, Box Regression

1 Introduction

Deep learning is a growing field finding its place in many applications over a wide variety of fields. Therefore it becomes more and more important to take an interest and understand the ins and outs of this technology, or at least to become familiar with the concepts and notions revolving around this field.

To take a first steps in this broad field, it is necessary to practice and experiment in order to become familiar with the concepts and develop a good Data Scientist instinct. To do this, nothing better than tackling a simple problem to study the influence of various parameters.

2 Problem

Simplicity is better than complexity to analyze a problem. However, to be slightly more original than in the classical literature, it will not be focus on a binary classification problem but on a box regression problem.

2.1 Dataset

Data for box regression problem can quickly become more complicated to obtain because of the need for coordinates of object within the image. Therefore, the dataset and its labels will be generated instead of relying on an existing open source dataset for box regression. Morevoer, this approach allows to create a really simple and didactic example.

To continue in this philosophy, the dataset is simply a binary image with a black background on which a white box is randomly added, as shown in Fig. 1. The coordinates of the upper left and lower right corner are saved as labels. As good practice, the labels are normalized (values between 0 and 1) to improve the speed and accuracy of the model as well as to allow better generalization on images of different sizes.

- *M. Olivier Debeir professor and researcher at the Laboratory of Image Synthesis and Analysis (LISA) of Université Libre de Bruxelles (ULB).*
- *M. Adrien Foucart teaching assistant and PhD student at LISA in ULB.*

Since there is no point in having high resolution images for such a simple problem. The image size is only 128x128 sparing a lot of time, space and energy.

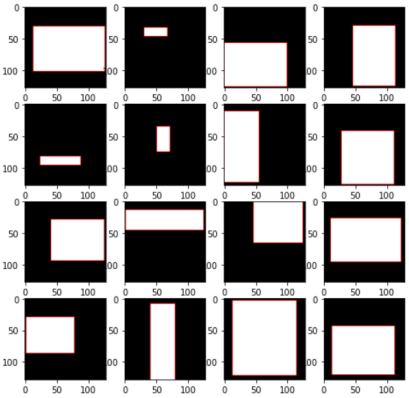


Fig. 1: Preview of the generated dataset

About the number and distribution of data, it remains standard. These are 10.000 images with a split of 80, 10 and 10 percent, respectively for the training, validation and testing set.

2.2 Model

Since the problem is defined and dataset is known. The next step is the design of the neural network model.

2.2.1 Architecture

A good practice for the design of neural networks is to start with a simple model and to complexify it later if necessary.

Since data are particularly simple, it is obvious that a small model is more than enough in this example. Therefore this section will not be specifically developed in this article.

The architecture is a simple model using relu activation function. The architecture is a succession of max pooling and convolution. Then the 2D-images are flatten into an 1-D vector to go through some dense layer with a final output of four element to map with the coordinates of the labeled box.

Layer (type)	Output Shape	Param #
max_pooling2d (MaxPooling2D)	(None, 64, 64, 1)	0
conv2d (Conv2D)	(None, 64, 64, 4)	40
max_pooling2d_1 (MaxPooling2	(None, 32, 32, 4)	0
conv2d_1 (Conv2D)	(None, 32, 32, 4)	148
max_pooling2d_2 (MaxPooling2	(None, 16, 16, 4)	0
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 16)	16400
dense_1 (Dense)	(None, 4)	68

Total params: 16,656
Trainable params: 16,656
Non-trainable params: 0

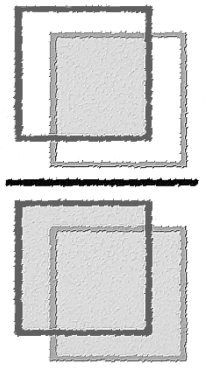
Fig. 2: Architecture of the simple neural network

Max pooling uses a 2x2 kernel to reduce image size by a factor of 4. Convolution uses a 3x3 kernel with 4 output channels to extract features from the image. Since the images only behave as simple shapes, it is neither necessary to use too many layers, nor too large layers (i.e. 4 output channels are enough for these convolutions).

2.2.2 Metrics

Unlike architecture, metrics are more interesting to develop in this example. Indeed, the basic metrics pre-defined on Keras are not particularly suited to object detection problems.

Therefore a custom metric, such as intersection over union (IoU), is needed. IoU is the fraction of intersection between area cover by the labels and the prediction over their union area. The main idea is to have the best superposition between the predictions and labels in term of position an also size.



The loss of the model will simply be "1-IoU" since this metrics is between 0 and 1, and the larger it is, the better it is.

3 Results

Now that the generic model is built. The interesting part can begin, the training of the model by varying some hyper-parameters. In this experiment, the parameters of interest are batch size and learning rate.

To automate the process, the model will be trained inside a loop to be able to iterate on a pre-defined list of batch size and learning rate. The generated data is saved in files for later analysis.

Each model (ie each combination of learning rate and batch size) is evaluated 10 times to obtain reliable data.

3.1 Accuracy

A good way to evaluate a deep neural network is the accuracy of the predictions.

3.1.1 Methodology

To avoid biasing the final model, it is forbidden to evaluate the precision on the test set. In our particular instance, as the data is randomly generated on the spot, it would be possible to create a second test set to assess the accuracy of the intermediate models. However, this is usually not a good option in real world problems due to lack of data. This approach will therefore not be used in order to mimic real cases.

Another good approach for estimate the accuracy is to rely on the validation curves. However, this approach assumes that the data are correctly distributed, that the validation set is sufficiently general and different from the training set to asses that the model generalizes well enough and does not fall into overfitting[1]. Since data are randomly generated, this assumption can be considered to be satisfied.

Plots of validation loss curves have then been generated to analyze the training. Some of theses plots are shown on the figure 3.[2]

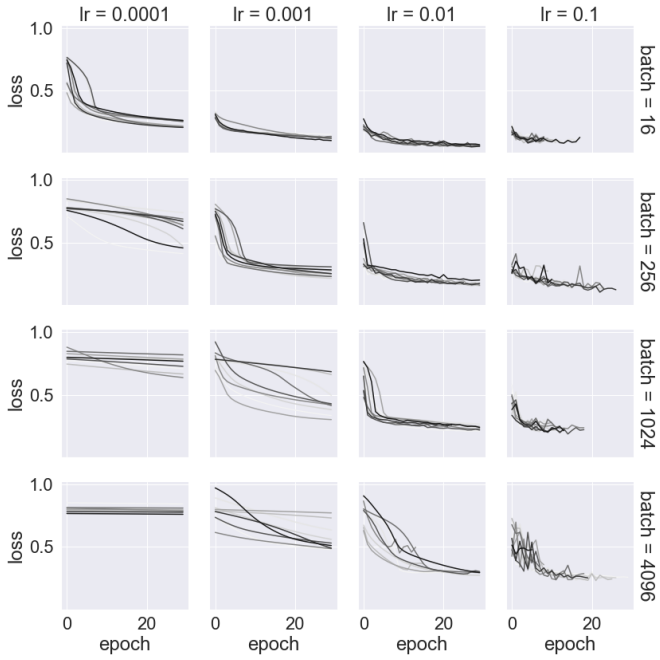


Fig. 3: Influence of learning rate & batch size on validation losses over 10 iterations

As you may see on some plots of the figure 3 some curves stop earlier. This behavior is completely intended since an early stopping condition has been implemented. This means that the AI stops its training prematurely if it notices no more improving in validation loss.

In addition, the two first epochs are not represented in the graphs to improve the readability.

1. Tables in appendix confirm the generalization power of the model.
2. For interested readers, these graphs, with some additional ones, are included in appendix with an enlarged size.

3.1.2 Analysis

The analysis is developed on 3 axes of influence; the learning rate, the batch size and both at the same time.

The influence of learning rate is highlighted by analyzing the plots line by line (ie. fixed batch size).

The graphs show a faster decrease in validation loss by increasing the learning rate. This means the model requires fewer epochs to reach its full potential, saving on training time.

However, too high learning rate, as in the last column (ie. learning rate = 0.1), causes oscillations on the learning curve for this model. This phenomenon is quite easy to understand. Remember that the process in the background is a gradient descent and that the learning rate is the step with which it follows this derivative. If the learning rate is too high, it will exceed the minimum, then go back during the next iteration and so on causing this oscillation.

The influence of the batch size is highlighted by analyzing the plots column by column (ie. fixed learning rate).

The interesting detail is the final loss. Indeed, the final loss for a fixed number of epoch, is smaller when the size of the batch is reduced. This phenomenon is a bit more complicated to interpret. One hypothesis could be that a small batch size implies more weight updating. Indeed, the weights are updated after each batch. A small batch therefore implies using more batches per epochs and thus more updates. Therefore, the model need less epochs for smaller batch size.

These graphs clearly show a relation between learning rate and batch size. It is mainly visible on the main diagonal. By increasing both the batch size and the learning rate, it is possible to keep very similar learning curves.

In other words, when the model gives good predictions and the batch size is increased or decreased, the learning rate must respectively be increased or decreased and vice versa.

Another interesting relation between learning rate and batch size is the sensitivity of the batch size to learning rate. This observation is more visible by taking a look at the first and third rows (ie. batch of respectively 16 and 1024 images) of the fig 3. Indeed, the graph corresponding to a batch size of 16 and a learning rate of 0.0001 (ie. plot (1,1)) and the graph corresponding to a batch size of 1024 and a learning rate of 0.01 (ie. plot (3,3)) have similar curves. To start observing oscillations in these learning curves, it is necessary to increase the learning rate by 2 to 3 orders of magnitude for a batch size of 16, while it only takes 1 order of magnitude for a batch size of 1024. This means that a small batch size is less sensitive to the learning rate. In other words, a variation of learning rate is more significant for bigger batch size.

3.2 Learning Time

A second important criterion for evaluating a model is its learning time. This criterion is often neglected over accuracy of the predictions.

However, deep learning training often takes considerable time, and therefore significant energy consumption. In a climate of conscience where ecology takes an increasingly important place, adopting an eco-responsible approach becomes more and more important.

Therefore, it is important to find a good balance between cost and precision. Is it really interesting to double the learning time to gain a small percentage of precision ? These kinds of questions should be seen more often in a field like this, focused on precision competition.

3.2.1 Methodology

To evaluate the training time of the different models, a stopwatch has been set up to measure and save this value. As for the accuracy measurement, the timing is computed for each combination of learning rate and batch size over 10 iterations (i.e. 10 training sessions of 30 epochs). Those data were then gathered on the graph of Fig 4.

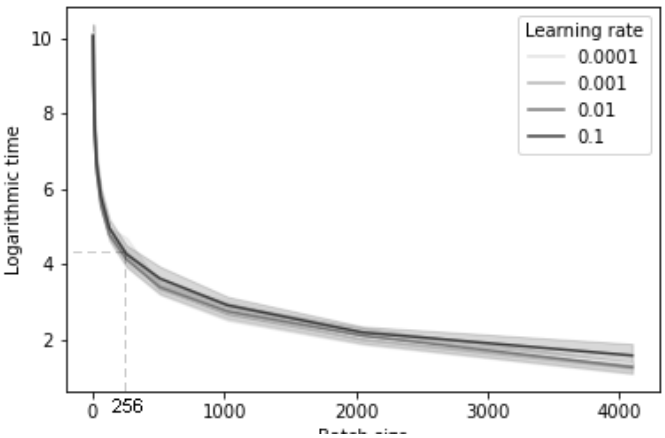


Fig. 4: Evolution of training time according to batch size on fixed number of epochs

3.2.2 Analysis

The first thing that comes out from the graph 4 is the training time following a curve of the form $\frac{1}{x}$ depending on the batch size.

Although the values of time and batch size are not very important on their own, it is important to realize that time follows a logarithmic scale. This means that decreasing the batch size, exponentially increases the training time.

A second characteristic is that the training time is independent of the learning rate. Indeed, the different learning rate curves overlap quite well, meaning that the learning rate has no real impact on learning time with a fixed number of epochs.

3.3 Interpretation

The batch size gives a compromise between training time and and accuracy. Too large a batch size will decrease the accuracy of the model on a fixed number of epochs. Conversely, a batch size that is too small will drastically increase the learning time of an epoch.

The learning rate gives a trade-off between learning speed and stability. Indeed, a high learning rate should allow the model to obtain good accuracy more quickly. However, a too high learning rate creates an oscillation phenomenon. This means that the gradient descent uses a too large displacement step, thus missing the optimum each time and creating back and forth movements around the optimum point. At the model level, this is reflected in by a learning curve improving very quickly at the beginning and then oscillates, degrading the accuracy. Conversely, a too low learning rate will barely modify the weights of the neural network at each update, giving an almost flat learning curve. The model thus takes a considerable time to get an acceptable accuracy. However, the training is stable with continuously improving accuracy (but very slowly), preventing the model from spinning around an optimal and getting quite random accuracy.

4 Discussion

As seen in the previous section, the choice of the learning rate and the batch size is not insignificant. Indeed, a good combination of learning rate and batch size is important to obtain a good compromise between the cost (ie. training time) and the accuracy of the model.

From what has been highlighted during the experiment, it is possible to establish a strategy to select the learning rate and batch size. Since there seems to be a relationship between batch size and learning rate, it is possible to fix one and determine the best other one.

What should be fixed first ? It is logically preferable to set the batch size which is the most impacting and restrictive. Indeed, the batch size is limited by the size of the data and the limits of the GPU. In addition, it plays an essential role in the learning duration. Therefore it is more interesting to fix the batch size to then determine a good learning rate.

First, the batch size must allow to achieve sufficient precision (small batch size) in a reasonable time (big batch size). A good trade-off would be one of the batch sizes, just after the exponential slope of fig 4, i.e. a batch size around 256. However, as our model is quite light and quick to train. It is not absurd to reduce the batch size to 16 which seems to gain a non-negligible precision, justifying the increase in training time.

Second, the choice of learning rate must be taken with more attention. A way to determine a good learning rate is to choose one at random and train the model on a few epochs. Then observe the learning curves and increase or decrease the learning rate by an order of magnitude depending on the observed results. Repeat several times, possibly refining the step, until you find a satisfying one. For example, thanks to the data previously collected, the figure 3 makes it possible to define a learning rate between 0.01 and 0.001 as being a good value for a batch size of 16.

A trick to avoid wasting too much time on fine-tuning the learning rate and get good first-time results is a decreasing learning rate. This functionality can be implemented thanks to the callback function *ReduceLROnPlateau* of Keras. This function reduce the learning rate if there does not seem to be any improvement. This trick allows to set a learning rate that is a little too high at the beginning to learn quickly, then if an oscillation phenomenon occurs, then the learning rate is reduced.

5 Towards Improvement

After all these observations, it is possible to train a model with a good parametrization. Then take a look at its performance and predictions to detect certain characteristics in order to identify new areas for improvement to investigate.

5.1 Best Parametrization

As discussed in the section 4, a good combination of parameters would be a batch size of 256 with a learning rate of 0.01.

However, since our model is trivial and quick to train, it is not very expensive to reduce the batch size a little to obtain better results. In addition, to slightly improve performance, a decreasing learning rate will be applied. Therefore, the model is trained over 30 epochs with a batch size of 16 and a decreasing learning rate of 0.1, divided by 2 every time the validation does not improve on 2 epochs.

5.1.1 Predictions Accuracy

This time, since it is the final evaluation, the model can be challenged on the testing set.

The results give a loss of 0.038 which is excellent for an *Intersection over Union* metric (ie. overlapping measure). However, this result is only a number that may sometimes hide a certain *"field reality"*. To go further in improving the model, it is relevant to analyze a little the predictions obtained. A sample of predictions is shown in the figure 5.

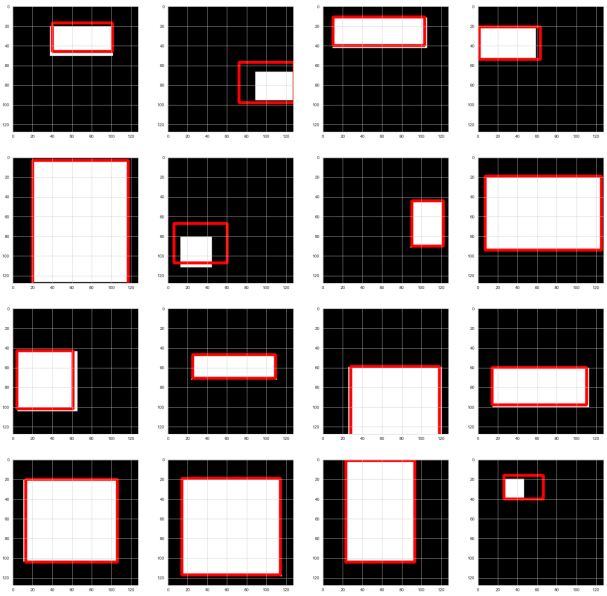


Fig. 5: Predictions of the model trained on 30 epochs with a batch size of 16 and learning rate from 0.1 to 0.0016

Visualizing the predictions highlights an important feature of the model, its inaccuracy for small objects.

5.1.2 Model Limits

To ensure the inaccuracy of small objects, it is not conceivable to manually analyze all the predictions. Therefore, a histogram of errors as a function of the smallest dimension of the object (fig. 6) could be a relevant visualization. For the sake of readability of the results, the errors are quantified by levels[3].

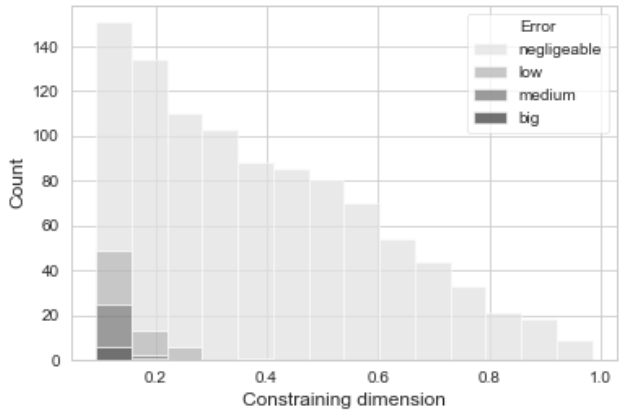


Fig. 6: Frequency and magnitude of the prediction error as a function of the constraining dimension (i.e. smallest) of the object to detect.

This result is clearly in line with the hypothesis put forward. However, it ignores one of the dimensions of the object. To be certain, a scatter plot of the error as a function of the height and width of the object (fig. 7) is more appropriate.

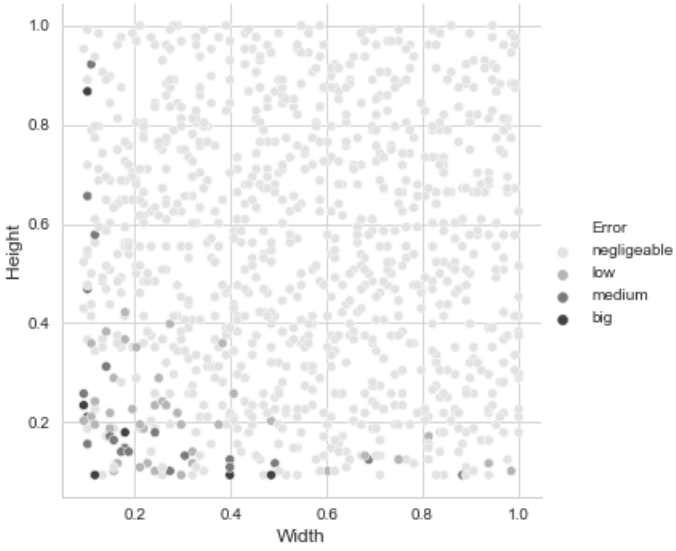


Fig. 7: Magnitude of the error on the predictions depending on the height and width of the object to detect

These results clearly demonstrate an anti-correlation between the prediction error and the size of the object, the smaller the object, the greater the error probability. It is enough for the object to have only one too small dimension to greatly degrade the precision.

3. Error levels are defined as:
negligeable $\leq 0.1 <$ low $\leq 0.2 <$ medium $\leq 0.3 <$ big

The most logical cause that comes in mind is the max pooling. Since the max pooling is mapping an image into a smaller one, it prevents to perfectly reverse the transformation. Consequently, this generates a slight inaccuracy on the position of pixel. By the nature of the metric used for box regression (IoU), a small inaccuracy has much more impact on a small object than a large one.

However, it is not possible to get rid of max pooling layers in computer vision. Otherwise the dense layer at the end of the structure would be far too huge, with a colossal computation time. The architecture used for the model shows its limits and does not seem to be suitable for a box regression problem.

5.2 Better Architectures

Since the architecture is not particularly suitable for a box regression problem. It is interesting to look into the literature to discover new architectures and designs for deep learning. However, it is a bit outside of the scope of this paper.

5.2.1 Region of Interest

A commonly used architecture for detection problems in computer vision is the Region of Interest (RoI). The key idea is to separate the image into lots of regions of interest (possible overllaping) and to train the AI as a classification problem on these RoIs. [2]

One of the advantages, compared to the approach used in this article, is that the RoI can detect several objects in an image.

6 Conclusion

To conclude, there is no magic solution to parameterize a model. However, having a good feeling, a good methodology and good practices to follow, as the ones describes below, help greatly.

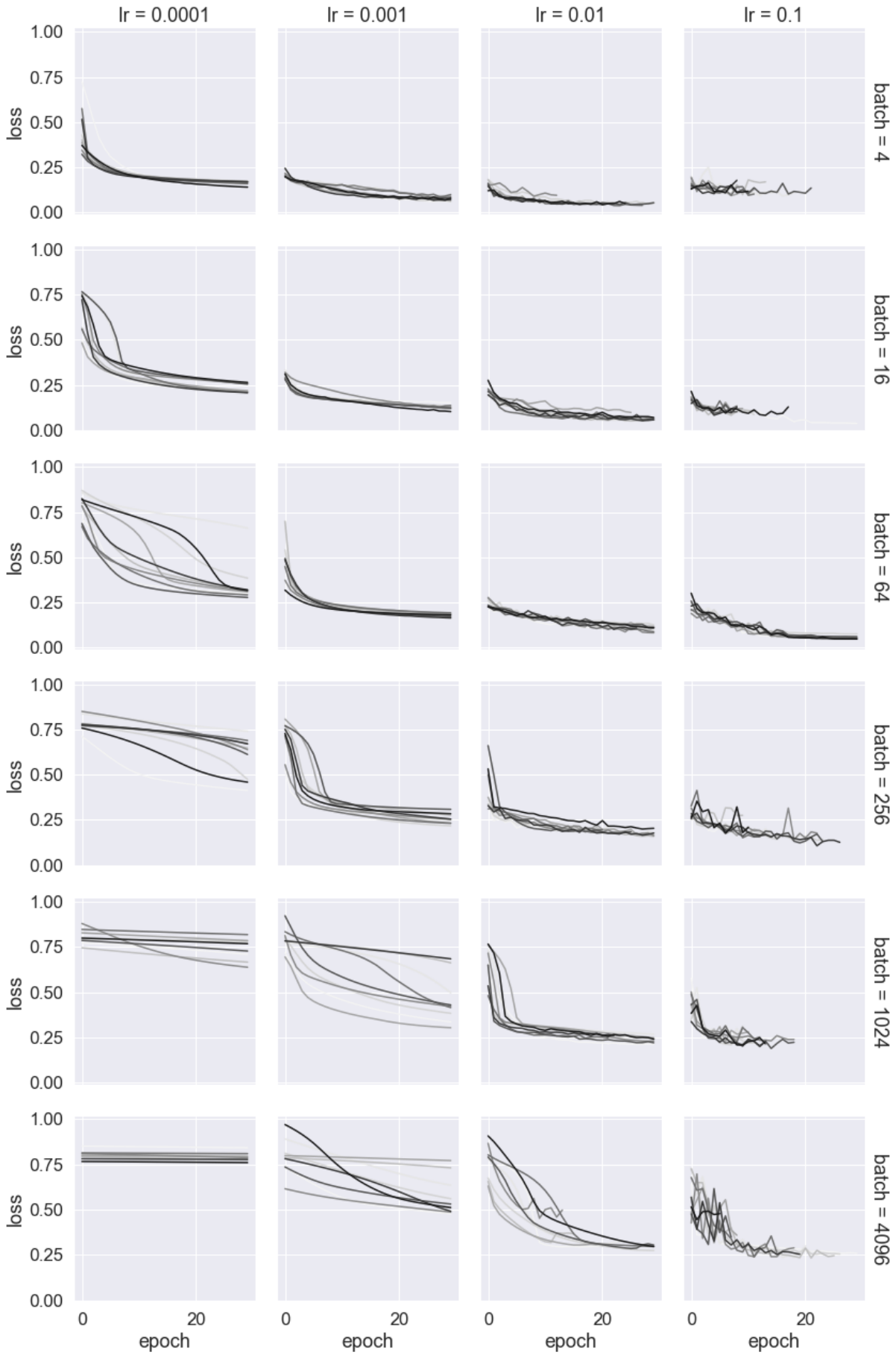
First the selection of batch size which is more constraining, and then find the best learning rate. In addition, the impact of learning rate depend of the batch size. A large batch size requires a more precise learning rate. For rapid prototyping, a decreasing learning rate is particularly useful.

After training phase, an analysis of the results allows to deduct some potentially interesting parameters to modify. For example, an oscillating loss indicates that the learning rate should be reduced. On the contrary, if learning is slow, an increase in learning rate is beneficial.

Accuracy can be improved by increasing the learning rate or decreasing the batch size. However, a batch size reduction increases significantly the learning time. Therefore, it is only a question of finding a good trade-off between accuracy and cost.

At the beginning, the most important parameter is the whole architecture. Indeed, the first careful choice concerns the architecture, its limits and its adequacy with the problem. If at the end, the accuracy is still not as good as expected after multiples parametrization combinations then a new architecture should perhaps be considered.

Appendix A
Influence of learning rate and batch size on validation losses over 10 iterations



APPENDIX B
FINAL LOSSES & OVERFITTING

Training Curves | Validation Curves | Similitude

batch	lr	train loss
4	0.0001	0.166290
	0.001	0.076296
	0.01	0.048530
16	0.0001	0.233798
	0.001	0.125542
	0.01	0.060065
64	0.1	0.036349
	0.0001	0.354742
	0.001	0.181704
	0.01	0.106987
256	0.1	0.054588
	0.0001	0.608280
	0.001	0.258224
	0.01	0.174654
1024	0.1	0.743774
	0.0001	0.463807
	0.001	0.239661
4096	0.01	0.794044
	0.0001	0.583059
	0.001	0.293586
	0.01	0.258153

batch	lr	valid loss
4	0.0001	0.165973
	0.001	0.079723
	0.01	0.054255
16	0.0001	0.231780
	0.001	0.126675
	0.01	0.061508
64	0.1	0.037583
	0.0001	0.348920
	0.001	0.180731
	0.01	0.105992
256	0.1	0.054651
	0.0001	0.601600
	0.001	0.255394
	0.01	0.173817
1024	0.1	0.741449
	0.0001	0.458969
	0.001	0.236623
4096	0.01	0.793787
	0.0001	0.577436
	0.001	0.292560
	0.01	0.258545

batch	lr	similitude
4	0.0001	100.190878
	0.001	95.700459
	0.01	89.447357
16	0.0001	100.870977
	0.001	99.105699
	0.01	97.653473
64	0.1	96.715316
	0.0001	101.668507
	0.001	100.538254
	0.01	100.938109
256	0.1	99.883565
	0.0001	101.110271
	0.001	101.108387
	0.01	100.481289
1024	0.1	100.313635
	0.0001	101.054074
	0.001	101.284211
4096	0.01	100.032380
	0.0001	100.973717
	0.001	100.350532
	0.01	99.848392

Fig. 8: Comparison of the average value of the loss curves at the 30^{th} from previous figure

ACKNOWLEDGMENTS

I would like to thank Sanjeev TRIPATHI for his GitHub repositories [3] [4] and Dr. Sreenivas BHATTIPROLU for his GitHub repository [1] and youtube channel about small deep learning topics, for inspiring my research axis.

Finally, I would also like to thank Olivier DEBEIR, my computer vision professor at ULB, and his teaching assistant Adrien FOUCART for their time invested in the coordination and discussions on the project.

REFERENCES

[1] Dr. Sreenivas BHATTIPROLU, *Python for microscopists*, April 2022, on *GitHub* url: https://github.com/bnsreenu/python_for_microscopists

[2] Tomasz GREL, *Region of interest pooling explained*, February 2017, on *DeepSense.IA* url: https://deepsense.ai/region-of-interest-pooling-explained/

[3] Sanjeev TRIPATHI, *Deep Bounding Box Regression using Keras*, March 2017, on *GitHub* url: https://github.com/sanjeev309/deep_bbox_regression_keras/

[4] Sanjeev TRIPATHI, *Synthetic Bounding-Box Regression Database Tool*, October 2019, on *GitHub* url: https://github.com/sanjeev309/synthetic_bbox_regression_db_tool

[5] Adrian ROSEBROCK, *Object detection: Bounding box regression with Keras, TensorFlow, and Deep Learning*, October 2020, on *PyImageSearch* url: https://pyimagesearch.com/2020/10/05/object-detection-bounding-box-regression-with-keras-tensorflow-and-deep-learning/

[6] Adrian ROSEBROCK, *R-CNN object detection with Keras, TensorFlow, and Deep Learning*, July 2020, on *PyImageSearch* url: https://pyimagesearch.com/2020/07/13/r-cnn-object-detection-with-keras-tensorflow-and-deep-learning/