Using instance hardness measures in curriculum learning

Gustavo H. Nunes[1], **Gustavo O. Martins** [2], **Carlos H. Q. Forster**[2], **Ana C. Lorena**[2]

[1]Universidade Federal de São Paulo (UNIFESP)
Av. Cesare Monsueto Giulio Lattes, 1201 - São José dos Campos, SP, Brazil

[2]Instituto Tecnológico de Aeronáutica (ITA)
Praça Marechal Eduardo Gomes, 50 - São José dos Campos, SP, Brazil

{forster,aclorena}@ita.br, gustavo.nunes@unifesp.br, oliveiramartinsgustavo@gmail.com

Abstract. *Curriculum learning consists of training strategies for machine learning techniques in which the easiest observations are presented first, progressing into more difficult cases as training proceeds. For assembling the curriculum, it is necessary to order the observations a dataset has according to their difficulty. This work investigates how instance hardness measures, which can be used to assess the difficulty level of each observation in a dataset from different perspectives, can be used to assemble a curriculum. Experiments with four CIFAR-100 sub-problems have demonstrated the feasibility of using the instance hardness measures, the main advantage is on convergence speed and some datasets accuracy gains can also be verified.*

1. Introduction

In human learning, individuals usually learn subjects in an order of increasing difficulty, so that the learner may first grasp easier concepts before trying to assimilate harder topics. In contrast, most Machine Learning (ML) methods present the training data to the learning system without much regard to their difficulty level. Curriculum learning (CL) is a learning paradigm which attempts to approximate the ML process to that of human learning [Bengio et al. 2009], by presenting the training data to the learning algorithm in a meaningful order, beginning with easy examples but increasing their difficulty as training proceeds. This methodology has been found to improve learning for some domains, by either increasing generalization performance and/or decreasing convergence time [Weinshall et al. 2018].

Seeking to characterize the difficulty of the data points in a classification problem, there is some recent literature which extracts simple measures from the training datasets in order to quantify their hardness level, that is, how hard each observation is to classify [Arruda et al. 2020, Smith et al. 2014]. Instance hardness is defined in [Smith et al. 2014] as the likelihood an instance has to be misclassified by a diverse set of classification models. These authors also propose to quantify different aspects which may expound the hardness level of an instance, in a set of what they call "hardness measures" (HM).

The topics of CL and HM can be considered somewhat related. While the former needs to know which examples are easy and hard in order to rank them properly during the learning process, the latter is able to assess the difficulty levels of the individual observations from a dataset. Therefore, one is expected to profit from their combination. This is the hypothesis investigated in this work: "*Can instance hardness measures be used to properly rank the instances in curriculum learning?*". The work [Smith and Martinez 2016] has investigated how instance hardness as measured by a posterior class probability of multiple ML models could be used in CL. More recently, [Hacohen and Weinshall 2019] have employed a Support Vector Machine (SVM) classifier trained over features extracted from pre-trained deep learning network models, in a transfer learning strategy. Here we use a similar approach, but employing simpler measures able to describe why a given instance is hard to classify. Experimentally, we have observed that the average hardness level of a dataset is in general a proxy to whether CL can be beneficial and gains are verified mainly for datasets with higher average hardness levels.

This paper is organized as follows: Section 2 presents background on CL and HM. Section 3 presents the methodology adopted in the work. Section 4 presents and discusses experimental results achieved for different subsets of the CIFAR-100 dataset. Section 5 concludes this paper and discusses future work opportunities.

2. Background

This section presents background on CL and HM. In the following definitions, let T denote a training dataset composed of N pairs $(\mathbf{x}_i, y_i)$, where $\mathbf{x}_i$ is a particular observation described by d input features and y_i is its label.

2.1. Curriculum Learning

Curriculum learning (CL) consists of guiding the training of Machine Learning (ML) models such that the observations of a dataset are presented in a particular meaningful order [Bengio et al. 2009, Weinshall et al. 2018]. Usually the observations are ranked according to their difficulty level and ordered from easier to harder, so that the system first learns the simpler aspects of the task from the easier examples before learning the more complex ones from the more difficult cases [Bengio et al. 2009]. One of the centerpieces in developing a curriculum is finding a good ranking (or scoring) system, which depends on the problem at hand [Silva and Costa 2018].

Weinshall et al. [Weinshall et al. 2018] actually understand that defining a CL strategy in practice consists of solving two major problems: scoring the data according to their difficulty; and defining a scheduling procedure based on the resulting ranking. Formalizing this, Hacohen and Weinshall [Hacohen and Weinshall 2019] define *scoring* and *pacing* functions, which together define a curriculum.

A scoring function is one function which determines the difficulty level of the individual observations in the training dataset, enabling ranking and sorting the data. Formally, it is any function $\mathrm{f} : T \to \mathbb{R}$ which allows us to say that an observation $(\mathbf{x}_i, y_i)$ is more difficult than another $(\mathbf{x}_j, y_j)$ if $\mathrm{f}(\mathbf{x}_i, y_i) > \mathrm{f}(\mathbf{x}_j, y_j)$ [Hacohen and Weinshall 2019]. A pacing function is one function which determines the pace in which the data is presented to the learning algorithm. It is a function which determines the subsets $T_1^{'}, \ldots, T_M^{'} \subseteq T$ of size $|T_i^{'}| = \mathrm{g}(i)$ from where M training mini-batches $\{\mathbb{B}_i\}_{i=1}^{M}$ are sampled. A particular subset $T_i^{'}$ contains the $\mathrm{g}(i)$ first examples from the training dataset, when sorted in ascending order by the value of the scoring function. In this way, the first subsets contain the easiest examples but harder cases are progressively included as training proceeds.

Algorithm 1 presents a pseudocode of CL for producing the training mini-batches [Hacohen and Weinshall 2019]. Given a dataset T, a scoring function for ordering the data and a pacing function for sampling them, the algorithm outputs a sequence of mini-batches that are recommended for training the learning algorithm. Line 1 first sorts the data items in ascending order of difficulty. The for loop in line 3 assembles the mini-batches sampling without replacement from the ordered set T, following the pace of increasing difficulty given by the pacing function.

Algorithm 1: CL pseudo-code [Hacohen and Weinshall 2019]

```
   Input: scoring function f, pacing function $g_\vartheta$, dataset $T$, desired number
          of batches $M$.
   Output: sequence of mini-batches $[\mathbb{B}'_1, ..., \mathbb{B}'_M]$.
1  sort $T$ according to f, in ascending order
2  $result \leftarrow []$
3  forall $i$=1, ..., $M$ do
       $size \leftarrow g(i)$
       $T'_i \leftarrow T[1, ..., size]$
       uniformly sample $\mathbb{B}'_i$ from $T'_i$
       append $\mathbb{B}'_i$ to $result$
4  return result
```

The pacing function employed in this work is the exponential pacing function. It has a fixed step length, but increases the batch size at each step. Let *step_length* be the number of iterations in each step, *inc* be an exponential factor used to increase the amount of data used for sampling the mini-batches in each step and *starting_percent* be the fraction of the data used in the initial step, the fixed exponential pacing function can be defined by Equation 1 [Hacohen and Weinshall 2019], where N is the number of instances in the dataset:

$$\mathrm{g}(i) = \min(starting\_percent \cdot inc^{\lfloor \frac{i}{step\_length} \rfloor}, 1) \cdot N. \tag{1}$$

2.2. Instance Hardness Measures

Smith et al. [Smith et al. 2014] introduced the concept of *instance hardness* as the likelihood each observation of a dataset has to be misclassified, despite the learning algorithm employed. Given a training dataset T, let h be a hypothesis mapping the input feature vectors to their corresponding labels, induced from T. The probability of h correctly classifying a particular observation $\mathbf{x}_i$ is $p(y_i|\mathbf{x}_i, \mathrm{h})$. Smith et al. [Smith et al. 2014] define the hardness level of an instance $\mathbf{x}_i$ as:

$$\mathrm{IH}_A(\mathbf{x}_i, y_i) = 1 - \frac{1}{|A|}\sum_{j=1}^{|A|} p(y_i|\mathbf{x}_i, \mathrm{h}_j). \tag{2}$$

Where A is a set of representative learning algorithms. Therefore, the hardness level of an instance depends not only on its own features but also on the remaining training data and the performance of a set of representative algorithms producing classification hypothesis h_j from T. Discussions on how the set of representative algorithms is chosen are beyond the scope of this work. Our interest is on the subsequent analysis of reasons why an instance is hard to classify.

Whilst instance hardness as defined in Equation 2 measures the probability that an instance will be misclassified, giving insight on the complexity and hardness levels of the observations, there are various underlying aspects that influence the difficulty of an observation. Seeking to explore *why* an instance is misclassified, Smith et al. [Smith et al. 2014] define a set of hardness measures (HM), each one measuring an aspect that may influence in an observation being misclassified and, in consequence, in its hardness level. Next we present the set of hardness measures from the literature [Smith et al. 2014] employed in this work.

- **k-Disagreeing Neighbors** (kDN): estimates the local overlap of an instance in the input space by regarding its nearest neighbors. The kDN of an instance is the percentage of its k nearest neighbors that do not share its label. Letting $k\mathrm{NN}(\mathbf{x}_i)$ be the set of the k nearest neighbors of the instance $\mathbf{x}_i$, kDN can be formally defined as:

$$k\mathrm{DN}(\mathbf{x}_i) = \frac{\sharp\{\mathbf{x}_j : \mathbf{x}_j \in k\mathrm{NN}(\mathbf{x}_i) \wedge y_j \neq y_i\}}{k}. \tag{3}$$

  Higher values of kDN imply the instance is surrounded by observations from different classes, resulting in a strong class overlap and a more complex classification.
- **Disjunct Class Percentage** (DCP): this measure first builds a decision tree using T and considers the percentage of instances in the disjunct of $\mathbf{x}_i$ which share the same label as $\mathbf{x}_i$. The disjunct of an observation corresponds to the leaf node where it is classified by the decision tree.

$$DCP(\mathbf{x}_i) = 1 - \frac{\sharp\{\mathbf{x}_j | \mathbf{x}_j \in Disjunct(\mathbf{x}_i) \wedge y_j = y_i\}}{\sharp\{\mathbf{x}_j | \mathbf{x}_j \in Disjunct(\mathbf{x}_i)\}} \tag{4}$$

  where Disjunct($\mathbf{x}_i$) represents the set of instances contained in the disjunct where $\mathbf{x}_i$ is placed. Here we adopt the complement of the original DCP measure such that easier instances will register lower DCP values, having a larger percentage of examples sharing the same label as them in their disjunct.
- **Class Likelihood Difference** (CLD): takes the difference between the likelihood $\mathbf{x}_i$ has to y_i and the maximum likelihood it has to any other class:

$$CLD(\mathbf{x}_i) = \frac{1 - (P(\mathbf{x}_i|y_i)P(y_i) - \max_{y_j \neq y_i}[P(\mathbf{x}_i|y_j)P(y_j)])}{2} \tag{5}$$

  The difference in the class likelihood is larger for easier instances, because the confidence they belong to their classes is larger than that of any other class. Here we take a normalized version of the measure, so that easier instances show a lower CLD value and it is bounded in the $[0, 1]$ interval.

Other HMs are defined in the work of Smith et al. [Smith et al. 2014] and, more recently, in [Arruda et al. 2020]. We have chosen here measures with highlighted performance in describing the hardness level of the observations of a dataset, as experimentally evaluated in [Arruda et al. 2020].

3. Methodology

The usage of HM in CL is simple using the methodology proposed in [Hacohen and Weinshall 2019]. We embed HM as the scoring function f in Algorithm 1, allowing the use of those metrics with minimal modifications to the CL framework, which we describe next. All HMs as described in Section 2 are already standardized so that higher values are output for harder instances. The PyHard package was used in the computation of the HM[1]. In our experimental evaluation, we use as base an experimental framework delineated in the recent work of [Hacohen and Weinshall 2019][2], which uses a convolutional network (CNN) of moderate size for classifying a group of images from the CIFAR-100 dataset. The CIFAR-100 dataset contains 60,000 32x32x3 colored images divided into 100 classes [Krizhevsky et al. 2009]. The classes are also grouped into super-classes, each one comprising five similar classes. Each super-class contains 3,000 images, further divided into 2,500 training images and 500 test images. The datasets used in this work are: (i) "people", (ii) "small mammals", (iii) "trees" and (iv) "vehicles_2", which show different difficulty levels, as discussed in Section 4.

The CNN used in the experiments has eight convolutional layers with 32, 32, 64, 64, 128, 128, 256 and 256 filters, respectively. The first six layers have filters of size 3 x 3, while the last two layers have filters of size 2 x 2. Additionally, in every second layer there is a 2 x 2 max-pooling layer and a 0.25 dropout layer. After the convolutional layers, the units are flattened and there is a fully-connected layer with 512 units followed by a 0.5 dropout layer. The output layer is a fully-connected layer with a number of output units equal to the number of classes, followed by a softmax layer. Finally, the network is trained using a SGD optimizer, with cross-entropy loss and batch size of 100.

Hacohen and Weinshall [Hacohen and Weinshall 2019] used a *transfer scoring function* in their experiments. It takes the Inception network [Szegedy et al. 2016], pre-trained on the ImageNet dataset [Deng et al. 2009], runs it through each training set observation and uses the activation levels of the penultimate layer of the network as a feature vector. This yields a new dataset consisting of the extracted feature vectors of the images (with 2048 features) and their original labels. Finally, a SVM classifier is trained over this new dataset and this classifier's classification probability scores are used as the scoring function. Given these scores, the network is trained using a fixed exponential pacing function. We also use the features from the Inception network to build the datasets fed into the HM, but without the need of making use of the SVM classifier.

Furthermore, results for four distinct curricula are presented for each dataset and hardness measure, namely:

[1]https://pypi.org/project/pyhard/0.2/
[2]Publicly available at www.github.com/GuyHacohen/curriculum_learning

- *Curriculum*: uses the sequence of mini-batches $[\mathbb{B}_i]_{i=1}^M$ resulting from the CL algorithm with the scoring function, i.e. uses training data in an order of ascending difficulty;
- *Anti-curriculum*: uses an anti-curriculum scoring function, meaning it uses the inverse training order demanded by the curriculum, training on data in an descending order of difficulty;
- *Random*: uses a randomized training order, independent of the scores. This is the standard strategy to train neural network models;
- *SVM* [Cortes and Vapnik 1995]: uses the SVM scoring function of [Hacohen and Weinshall 2019].

The complete pipeline for the use of HM is illustrated in Figure 1. First the set of images, originally unstructured, is structured into an feature-value format by the extraction of a fixed set of features from the images using a pre-trained Inception neural network model. Next, the HMs take the structured data as input to estimate the hardness level of the instances. Allied to a pacing function, the CL strategy is employed to generate a given sequence of batches to train the CNN models.

Since the main interest of the study lies in the scoring functions, the pacing function used is kept the same as that employed by [Hacohen and Weinshall 2019], namely the *fixed exponential pacing* function, with parameter values $step\_length = 100$, $inc = 1.9$ and $starting\_percent = 0.04$.

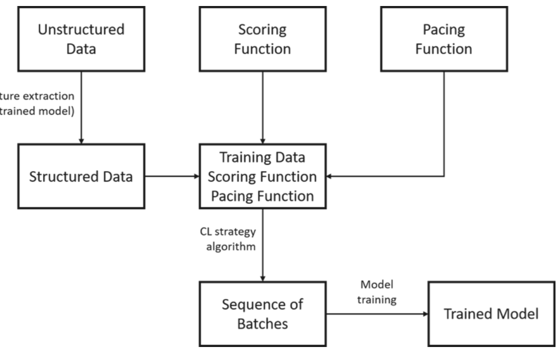


Figure 1. Flowchart of steps in the framework implemented, based on Algorithm 1.

For the evaluation of the experimental results, both the test accuracy and standard deviation are observed for multiple runs of each scoring function, as done in the reference work [Hacohen and Weinshall 2019]. Each evaluated curriculum was executed 30 times. Since the main effects we want to observe from the use of the curriculum are the improvement in generalization ability and in convergence speed, as in [Hacohen and Weinshall 2019] these metrics were analysed from two perspectives: how they behave across the epochs and which are their final values, concerning a mean and standard deviation from the various runs.

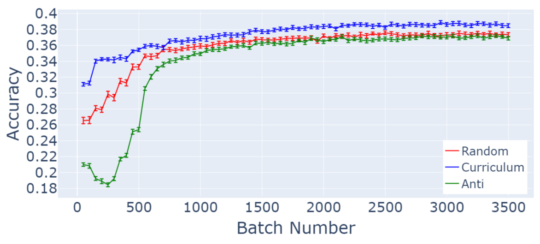
4. Results

We present next the experimental results of this work. The codes employed for generating the results can be consulted at `https://github.com/ghnunes/bracisCurriculum`.

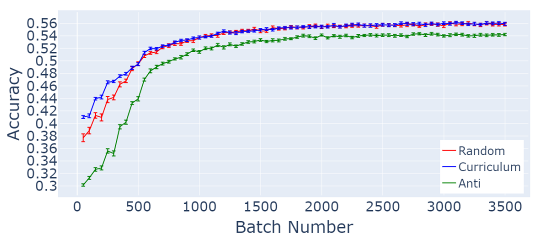
4.1. Hardness measures results

We first analyze the results obtained when each of the HMs experimented are employed to assemble the curriculum. The average and standard deviation accuracies of the CNN models are monitored for increasing numbers of batches. The results are contrasted to those of an anti-curriculum using the same measures and of a standard random sampling of the images for training, disregarding their hardness levels.

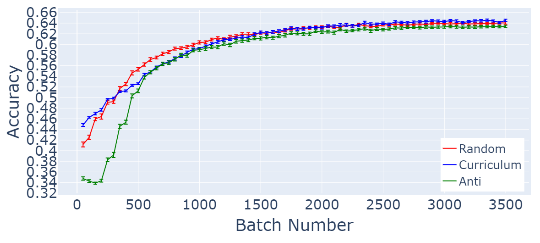
The results for the k-DN HM, using $k = 5$ as in [Smith et al. 2014], are plotted in Figure 2. The curriculum strategy is presented in blue, the anti-curriculum is colored in green and the random strategy is shown in red.



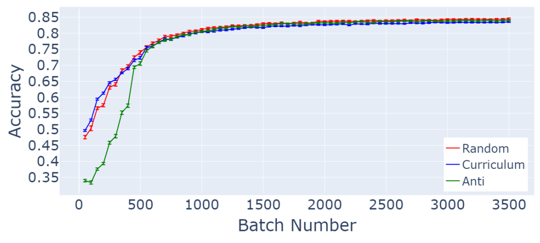
(a) Dataset "people"



(b) Dataset "small mammals"



(c) Dataset "trees"

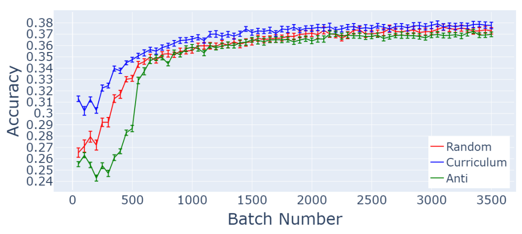


(d) Dataset "vehicles_2"

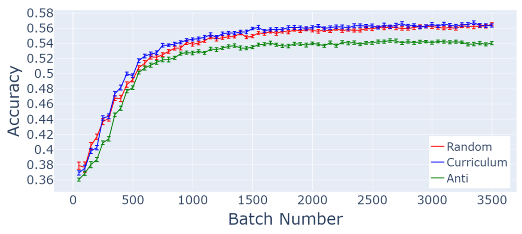
Figure 2. Average accuracy results of the CNN models using the k-DN measure as scoring function, for increasing batches.

The plots from Figure 2 show that the curriculum in general allows to obtain higher accuracy rates earlier than the other strategies, speeding up the network convergence. In contrast, the anti-curriculum impairs the accuracy performance in earlier stages of the CNN training, when the batches present predominantly hard instances. For some datasets, such as "people" and "trees", there is even a decrease of performance in early training epochs. As training progresses, the accuracy raises for all strategies. In fact, as more batches are generated, they tend to show an average hardness level and include the same instances, justifying the raise in the accuracy values even for the anti-curriculum strategy. Eventually the accuracy of the anti-curriculum may catch up the accuracies of the other strategies, as in the dataset "vehicles_2". But in other datasets such as "people" and "small mammals", the accuracy rates of the anti-curriculum remain inferior to those of the other strategies, specially when compared to the curriculum. The inferior performance of the anti-curriculum compared to the curriculum counterpart, in speed of convergence and/or generalization performance, evidences the k-DN measure is effective in ordering the training data according to their hardness levels. The random scoring had in general intermediate performance. It is worth noting that in the dataset "trees", the random strategy was quite effective for lower batch numbers. Whilst the anti-curriculum was the worst strategy in all cases, specially for early training batches.

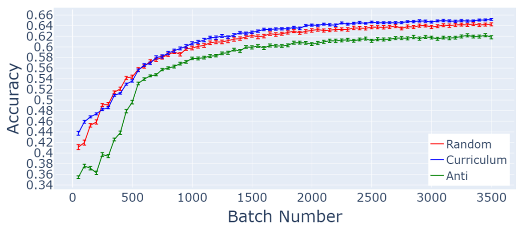
Similar tendencies are verified when DCP is used as a scoring function (Figure 3), although the accuracies in earlier training epochs seem more unstable and present more variation than those observed in the case of kDN. The DCP curriculum (in blue) in general improves the convergence speed more than the random (in red) and anti-curriculum (in green) strategies, as evidenced by the blue line being above the red and green lines in the early stages of the training (and eventually through all the training process, as in the "people" dataset). The curriculum has also shown superior test accuracy performance than the anti-curriculum in most of the cases. Therefore, DCP was also effective in ordering the training instances according to their hardness levels. In dataset "trees", the difference of the curriculum to the random strategy for early training batches was more subtle than that observed when kDN was used as a scoring function. And in the dataset "vehicles_2" all training strategies tend to present similar test accuracies as training progresses.



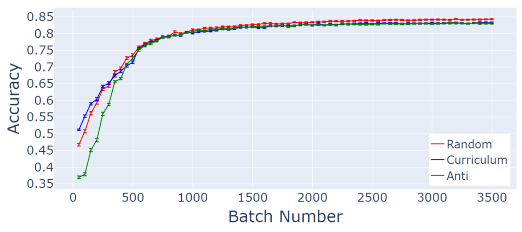
(a) Dataset "people"



(b) Dataset "small mammals"



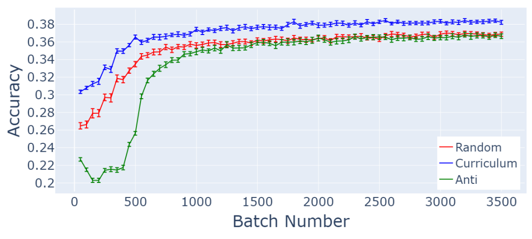
(c) Dataset "trees"



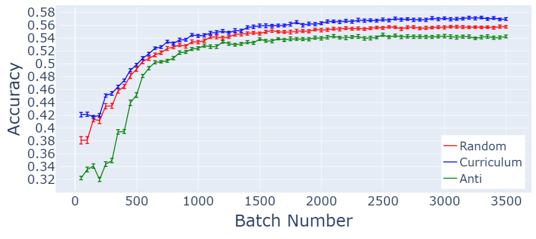
(d) Dataset "vehicles_2"

Figure 3. Average accuracy results of the CNN models using the DCP measure as scoring function, for increasing numbers of batches.

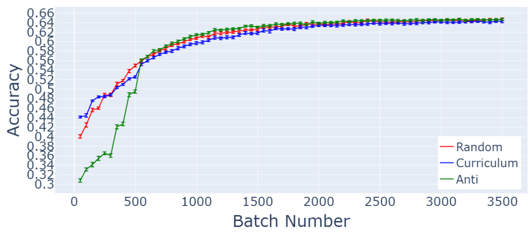
For CLD we observe again (Figure 4) that the curriculum in general improves the convergence speed of the CNN models and in some cases also improves the generalization ability as measured by the test set accuracy (as observed for datasets "small mammals" and "people"). As kDN and DCP, CLD was in general effective for ordering the training instances according to their hardness levels. But a counterintuitive result is observed for the dataset "trees" and after 500 batches the anti-curriculum achieves higher accuracy rates than all other strategies. Overall, the "trees" dataset has shown more distinct results compared to the other three datasets used. In dataset "vehicles_2", again all strategies behave similarly as more batches are presented.



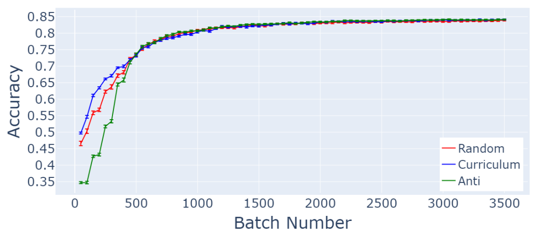
(a) Dataset “people”



(b) Dataset “small mammals”



(c) Dataset “trees”



(d) Dataset “vehicles_2”

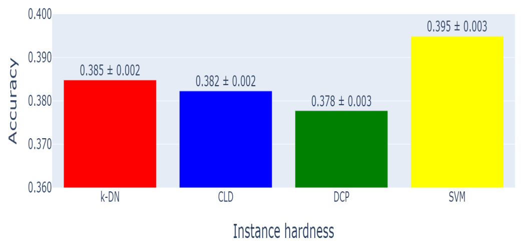
Figure 4. Average accuracy results of the CNN models using the CLD measure as scoring function, for increasing numbers of batches.

4.2. Discussions

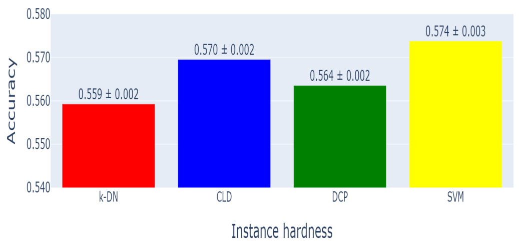
The results from the previous section prove that different HMs are suitable for CL, although the effectiveness may vary per dataset. Overall, using a CL strategy might be of particular interest if one wants to train the CNN models for a lower number of epochs due to computational resource constraints, as the curriculum in general gets better accuracy results earlier than the other strategies.

Figure 5 presents the final accuracy values achieved per dataset and curriculum ordering strategy, that is, the test accuracy after 3500 batches of data are presented. In these plots the final accuracy values achieved when the SVM curriculum strategy from [Hacohen and Weinshall 2019] is used are also presented. In general, the SVM curriculum achieved the best accuracy results. Only in the dataset “trees” the results of the SVM curriculum were very similar to those of the HM scoring functions. While taking the probabilities of classification output by a robust SVM model seem to provide better orderings, the computational cost involved is far superior to that of computing the HMs. An SVM can require up to cubic operations in the number of observations N [Bottou and Lin 2007]. kDN and CLD have a linear asymptotic cost and DCP has an asymptotic cost of $O(NlogN)$, for $N > d$. Therefore, they are much simpler to compute compared to the SVM-based scoring function. As the results of the HMs vary for different datasets, one strategy worth future studies is to aggregate the ranking provided by multiple measures and test other instance hardness. In this way one may take into account different characteristics which can influence the hardness level of an observation.

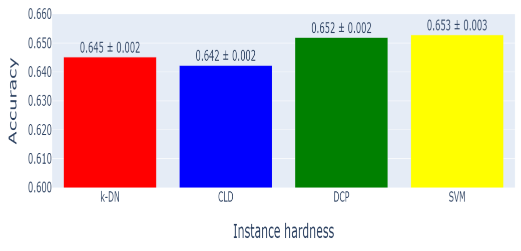
Taking a closer look at the datasets themselves, Table 1 presents the average (and standard deviation in parenthesis) of the different HMs per dataset. That is, we compute the HMs for each individual observation of a dataset and take the average of such values as an aggregate measure of difficulty, aka complexity level



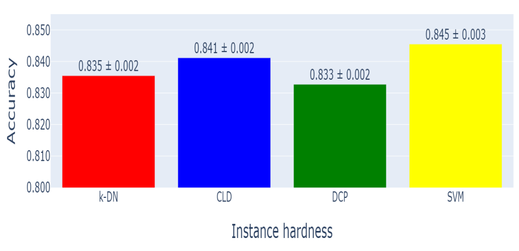
(a) Dataset “people”



(b) Dataset “small mammals”



(c) Dataset “trees”



(d) Dataset “vehicles_2”

Figure 5. Comparison of the accuracy obtained with the baseline.

of the dataset [Lorena et al. 2019]. The highest values per HM are highlighted in boldface and the lowest values are highlighted in italics. What stands from this table is that the “people” dataset was in average the hardest, whilst the “vehicles_2” dataset is the easiest. Standard deviation values are usually quite high, specially for the CLD measure. This indicates a large variation of values for all datasets and HMs. The dataset “trees” figures as the second hardest dataset, followed by the “small mammals” dataset.

Table 1. Difficulty of datasets based on average instance hardness measures (standard deviations in parenthesis)

	Dataset			
HM	**people**	**mammals**	**trees**	**vehicles**
kDN	**0.608 (0.292)**	0.363 (0.325)	0.526 (0.317)	*0.148 (0.254)*
DCP	**0.676 (0.207)**	0.501 (0.305)	0.611 (0.261)	*0.271 (0.318)*
CLD	**0.428 (0.489)**	0.274 (0.443)	0.417 (0.491)	*0.098 (0.297)*
SVM	**0.604 (0.010)**	0.233 (0.228)	0.237 (0.020)	*0.052 (0.040)*

CL was particularly effective for the datasets “people” and “small mammals” regarding the final generalization performance. In the datasets “vehicles_2” and “trees”, CL was not so advantageous. For “vehicles_2”, the easiest of the problems, the results are in accordance to what is expected in CL theory [Hacohen and Weinshall 2019, Weinshall and Amir 2020, Weinshall et al. 2018]. In fact, when a topic is already easy, CL may not be necessary at all. In that case, a conventional random sampling will already approximate the order of the curriculum. But harder problems tend to be easier to grasp by employing a CL strategy. The "trees" dataset breaks the pattern, since it is the second hardest from our pool and yet CL did not present outstanding results for it. There may be other sources of difficulty for this particular dataset which are not being captured by the scoring functions employed in this paper.

5. Conclusion

This work has explored the synergy of the Curriculum Learning and Instance Hardness measures topics, as by definition the former require the instances of a dataset to be ordered according to their difficulty level, which can be measured by the latter. We have employed the hardness measures as scoring functions in a general CL algorithm. Knowing that different hardness measures quantify distinct aspects that contribute to an observation being misclassified, we have also compared the results of three different instance hardness measures extracting distinct characteristics from the data.

Experiments with four CIFAR-100 sub-problems have demonstrated the feasibility of using the instance hardness measures to assemble a curriculum for training CNN models. The main advantage is on convergence speed, which is usually faster when a CL strategy is employed. For some datasets accuracy gains can also be verified. Interestingly, these gains are more evident for some harder datasets compared to easier problems, as measured by the average hardness level of the instances they contain. This makes evaluating the hardness of a dataset a possible initial proxy to decide whether employing a curriculum learning strategy for a new problem would be advisable.

Even though this work serves mainly as a proof of concept that curriculum learning and instance hardness measures may be applied together, a more thorough experimentation is still necessary in order to derive more definitive conclusions. More datasets should be used and other ML/deep learning techniques can also be tested. The usage of a pre-trained Inception network for extracting features from the images can also bias the results and other pre-trained models can be used instead. There are also a plethora of other hardness measures based on other aspects of the data, whose performances can differ from those presented here. Therefore, the test of other instance hardness measures in curriculum learning poses as another possible future work. A combination of the rankings provided by multiple hardness measures is also worth investigating, as each one of them provides an alternative perspective on the difficulty level of the instances.

In addition, this work has only focused on the effect that the scoring function may have on curriculum learning. However, the pacing function is also an important component which was not explored. Therefore, experimenting on the use of other pacing functions is also a worth future work.

Acknowledgements

To the Brazilian research agency CNPq, to Guy Hacohen who made the implementation behind his paper publicly available in GitHub, which facilitated the replication of the curriculum learning algorithm, and to Pedro Paiva, developer of the PyHard package, which implements the instance hardness measures used in this work and more. Finally, thanks to Intel for providing the DevCloud environment where the executions were made.

References

Arruda, J. L., Prudêncio, R. B., and Lorena, A. C. (2020). Measuring instance hardness using data complexity measures. In *BRACIS 2020*, pages 483–497. Springer.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th ICML*, pages 41–48.

Bottou, L. and Lin, C.-J. (2007). Support vector machine solvers. *Large scale kernel machines*, 3(1):301–320.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE CVPR*, pages 248–255.

Hacohen, G. and Weinshall, D. (2019). On the power of curriculum learning in training deep networks. In *ICML*, pages 2535–2544. PMLR.

Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images. *Citeseer*.

Lorena, A. C., Garcia, L. P., Lehmann, J., Souto, M. C., and Ho, T. K. (2019). How complex is your classification problem? a survey on measuring classification complexity. *ACM Computing Surveys (CSUR)*, 52(5):1–34.

Silva, F. L. D. and Costa, A. H. R. (2018). Object-oriented curriculum generation for reinforcement learning. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1026–1034.

Smith, M. R. and Martinez, T. (2016). A comparative evaluation of curriculum learning with filtering and boosting in supervised classification problems. *Computational Intelligence*, 32(2):167–195.

Smith, M. R., Martinez, T., and Giraud-Carrier, C. (2014). An instance level analysis of data complexity. *Machine learning*, 95(2):225–256.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *IEEE CVPR*, pages 2818–2826.

Weinshall, D. and Amir, D. (2020). Theory of curriculum learning, with convex loss functions. *Journal of Machine Learning Research*, 21(222):1–19.

Weinshall, D., Cohen, G., and Amir, D. (2018). Curriculum learning by transfer learning: Theory and experiments with deep networks. In *ICML*, pages 5238–5246. PMLR.