Applied Machine Learning Project

M.Sc. Data Science
University of Trento
A.Y. 2021/2022

3D Object Classification

Classification of Lego blocks employing PointNet

Emma Benedetti
223996
emma.benedetti@studenti.unitn.it

Lucia Hrovatin
223027
lucia.hrovatin@studenti.unitn.it

***Abstract*—PointNet is an efficient and effective Neural Network architecture suitable for many applications. This project exploits its robustness when facing 3D object classification tasks. The classified objects are Lego blocks extracted from RGB-D images and further transformed in Point Clouds, namely irregular geometric data structures based on a set of points.**

***Keywords*—PointNet, Point Cloud, RGB-D pictures, 3D object classification**

I. Introduction

The project explores the powerful features embedded in the PointNet classification network [1], a deep learning architecture that employs simple Point Cloud data structures [2], meaning sets of points in a 3D coordinate system, to perform 3D object classification tasks. Since the PointNet consumes exclusively point clouds, a two-step procedure is implemented, as shown in Fig.1. Firstly, the Legos' frames are extracted from RGB and monochromatic (i.e., *depth maps*) scenes through 2D bounding boxes. Secondly, each distinct 3D Lego block is rendered as Point Cloud and fed as input to the PointNet.

II. System Model

A. Collection and ingestion stage

As stated above, the PointNet inputs only Point Clouds. Therefore, the `lego_dataset.zip` and its `jpeg` files were rendered to meet these requirements. Three kinds of information were retained from the original dataset: the bounding boxes' coordinates, the RGB scenes, and the monochromatic depth maps. The coordinates have been constrained into the range $[0 - 1024]$ to avoid data anomalies, where the highest value corresponds to the number of pixels encoding the initial images. Overall, the cleansing procedure (performed by the function `extract_objects` in `data_ingestion.py`) removes around 33% of Lego blocks.
The remaining blocks are stored in the images-final.zip folder and needed further transformations. Thus, Point Clouds were created by joining RGB frames with depth maps and then stored as binary files. Whereas the file labels-final.pkl contains the label of each Point Cloud, images-final.pkl stores the Point Clouds rendered as arrays with dimensions nx3, where 3 refers to the *(x, y, z)* coordinates, and n is initially set to 1024. Therefore, as in the original implementation, 1024 points are sampled[1] from the original Point Cloud [1].
Further pre-processing steps were necessary due to outliers clustered in the lowest-right corner of all images and to the label's `string` format. The former issue was solved by exploiting the geometric median to identify outliers in 3d Euclidean Space [3], while the latter required the creation of a dictionary matching each label to an integer within the interval $[0 - 29]$. Lastly, the dataset was split into training and validation sets, including respectively 70% and 30% of the observations. The splitting procedure, randomly driven by the `train_test()` function (in `data_splitter.py`), guarantees reproducibility via the `seed`.

B. PointNet implementation

The deep-learning procedure takes after the **PointNet** architecture for classification tasks [1]. The architecture relies on three main stages: alignment networks applied on input points and, optionally, on features, Multi-Layer Perceptrons (*MLP*) with a different number of layers, and a max-pooling layer aggregating point features into a global feature. The input for each layer is re-centered and re-scaled using *Batch Normalization* followed by *ReLU* (Rectified Linear Unit) [4]. In the last MLP, a Dropout layer prevents overfitting during the training phase by randomly setting input units to 0 with a probability of 0.3.
A relevant feature of this model is its indifference to input permutations, that is achieved by the first alignment network, called `TNet3` (in `model.py`). Hence, the network takes as input the point cloud (with default size 1024x3) and returns a 3x3 transformation matrix that rotates or reflects the input. The second alignment network, `TNet64`, interests the points feature transformation and follows a similar procedure. Even though high dimensions greatly complicate the optimization process, the default model includes the transformation matrix in the feature space to boost the performance [1].

[1] If the Point Cloud's cardinality is less than n, it will be discarded.

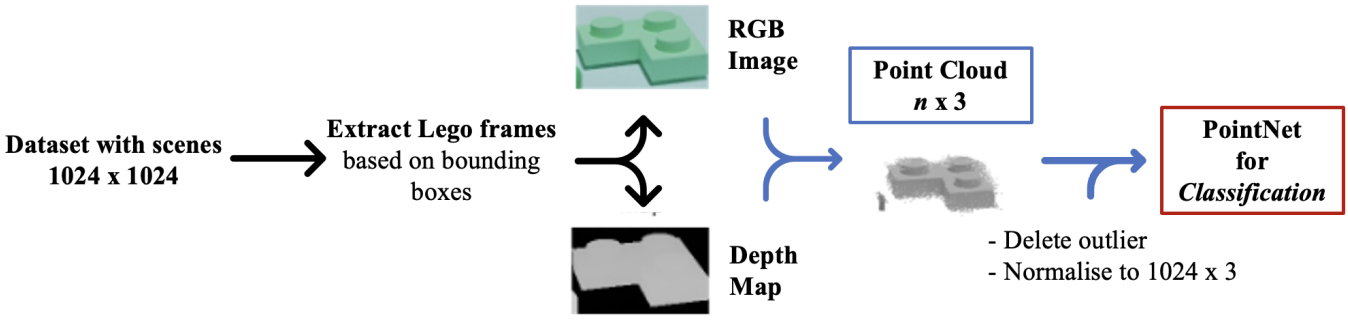

Fig. 1. **3D Object Classification Architecture**. The system transforms the initial scenes by splitting them into frames. Each Lego block is displayed in identical frames: an RGB picture and a depth map. A Point Cloud is generated and initialises the PointNet classification network. It takes n points as input, applies input (and feature) transformations, aggregates point features by max-pooling and outputs the classification scores for the k classes.

C. Evaluation stage

Excluding the Neural Network structure, there is no difference between image and point cloud classifications. Thus, similar measures can be used:

- **Batch Stochastic Gradient Descent** optimizer, in this case the *Adam* algorithm [5];
- **Cross-Entropy** as loss function;
- **Accuracy** as quality measure.

- The Adam algorithm

Neural Networks may be described as an optimization problem seeking the global optimum by employing a robust training trajectory and fast convergence. Among the several gradient descent algorithms, the Adam optimizer seems to better fit the projects' needs due to its robustness and performance [6], along with being consistent with the reference paper [1]. Specifically, it combines two other variants of the classical stochastic gradient descent procedure [7], namely the Adaptive Gradient Algorithm [8] and the Root Mean Square Propagation optimizer [6]. For a more detailed explanation, refer to the article by Kingma et al. [5].

The parameters are tuned according to the values reported in the reference paper [1], namely an initial learning rate of 0.001, and a batch size of 32. Moreover, the learning rate scheduler StepLR halved the learning rate every 20 epochs.

- Cross-Entropy

The choice of Cross-entropy (or log loss) as loss function is deeply rooted in the existing research and implementations [9]. In general terms, the loss function adjusts the model weights during the training phase, and its minimization reflects the model's goodness. Moreover, if the feature transformation (i.e., the `TNet64`) is activated, a regularization term is added to the Softmax training loss that should stabilize the optimization and boost the model's performance.

- Accuracy

The quality of the model is based on its accuracy, meaning the complementary of the misclassification error rate *ER* [10]. The ER corresponds to the sum of all wrongly assigned elements, y_j*, divided by the cardinality of the batch $|C_i|$, as expressed in equation (1).

$$ER_i = \sum_{j \in C_i} \frac{\mathrm{I}(y_i \neq y_j^*)}{|C_i|} \tag{1}$$

Within the architecture, the accuracy measure is employed twice. Firstly, it evaluates the goodness of the predictions by comparing them with the ground truth. Secondly, it selects the best train and test accuracy to avoid any risk of overfitting.

III. Results

The project's final result is an infrastructure that integrates a data cleansing procedure and a 3D image classification Deep Neural Network (Fig.1). Following the instructions contained in the `README.md` file on the GitHub repository, the deployed infrastructure can be trained and tested on the cleansed dataset images-final.zip, containing RGB and depth Lego frames, or on the given `pickle` files.

TABLE I
Results obtained after training the models for 100 epochs

Results in testing phase				
Name	**Learning rate**	**Accuracy**	**Loss**	**Runtime**
Vanilla PointNet	0.001	40.102	1.516	8h 07m
PointNet M1	0.01	22.842	2.165	23h 40m
PointNet M2	0.0001	23.579	2.221	20h 7m
PointNet standard	0.001	42.105	1.716	4h 55m
Weighted loss PointNet	0.001	7.779	2.029	4h 28m

Moreover, the model can be launched by command-line with default or tuned (hyper)parameters. The customisation of the following parameters is possible:

- Learning rate, whose default value is 0.001 and implies a scheduled decay;
- Number of epochs, set to 100 by default;
- Batch size for `train_loader` and `test_loader`. If not specified, it takes 32 as stated in the reference paper [1];

- a PointNet with both input alignment and feature alignment (set up by default), or a *Vanilla PointNet* with only the first `TNet`;
- the percentage that splits the dataset in train-test can be moved from $70/30$ to any other proportion;
- a subset of the dataset can be employed to speed up the training procedure. The model considers the whole dataset as default value;
- weighted loss function avoiding problems determined by an imbalanced dataset.

The different results[2] obtained when moving the parameters from the default values are reported in Table I. The best result is reached by the PointNet resembling the one in the reference paper, namely with both input and feature alignment and a learning rate of 10^{-3}. On the one hand, the absence of the feature alignment network impacts the performance of Vanilla PointNet but not in the expected extent. Thus, the performance gap displays a difference of only $2-3\%$ [1]. On the other hand, the learning rate variations returned opposite results. Any variation in learning rate, such as a lower one $lr = 10^{-4}$ or higher one (i.e. $lr = 10^{-2}$), affected the model's performance during training. In particular, the higher starting point had a stronger impact and almost halved the accuracy. However, from the similarity between the models' absolute test accuracy (see Tab. I), it can be assumed that the smaller learning rate $lr = 10^{-4}$ led to a local minimum but not to the global one.

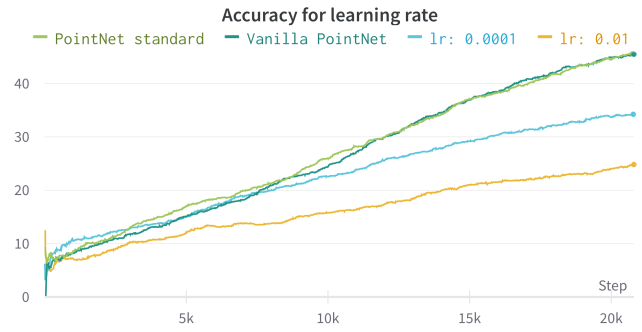

Fig. 2. Visualization of the training accuracy for four different models. On the x-axis the number of batches are reported, hence $20k = 200$ x $100(epochs)$. The most performing models in training are those having a learning rate of 10^{-3}, namely the Vanilla PointNet and the standard one.

A further customisation regards the number of points per Point Cloud. The random (re)sampling procedure set as input Point Clouds with three different cardinalities: 1024 (default cardinality), 512, and 2048 points. While with 512 points the test accuracy stops at around 16% (precisely 15.625), the model inputting 2048 points performs slightly worse, with a final accuracy equal to 13%. These results did not meet the expectations. Thus, an opposite behaviour compared to the reference paper arose. While the performance should grow as the number of points increases (and saturate with around $1K$ points) [1], it dropped, assessing the lowest value in the model with higher cardinality. Two possible reasons might explain this unexpected behaviour. Firstly, the training proceeds slower, and 100 epochs are insufficient. Secondly, having higher cardinality leads to a dropping out of all the Point Clouds whose number of points are less than the chosen size. Consequently, the number of batches per epoch decreases (e.g., 170 instead of 200 - standard PointNet with 1024 points- with the default buffer of 32) and the risk of an imbalanced dataset increases.

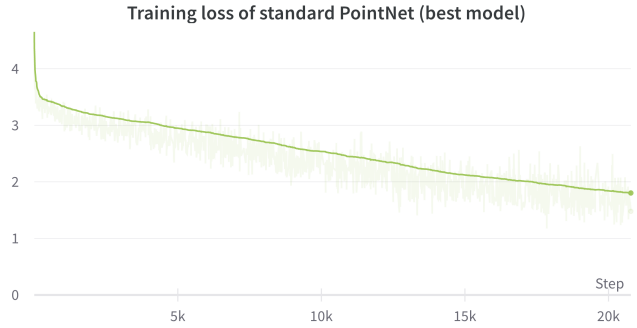

Fig. 3. A horizontal plateau characterizes a satisfactory training loss. In this model, the loss continuously decreases and does not reach the optimal condition (and convergence) due to the low number of epochs.

Overall, the best performance is unsatisfactory and should be boosted by implementing two adjustments. Firstly, the training loss continuously decreases (Fig. 3) and does not stabilize to a horizontal plateau [11], namely a convergence towards a minimum. Raising the number of epochs, for instance to 200, might stabilize the loss and lead to a better performance. Thus, the prolonged training implies an absolute accuracy of almost 63%, specifically 62.5% in test, $\overline{acc} = 43\%$ and loss of 1.72 in training[3].
Secondly, the results of Table I were obtained under the assumption of a *balanced* dataset. However, if the classes cardinality (regarding the whole dataset) are checked, an imbalance becomes evident (Table II): some classes are over-represented while others count few examples (e.g., class `9359`). To solve this issue, a weighted tensor (`weights`) was introduced into the loss function. It considers each class' cardinality and applies a weighted penalty. The weighted vector slows down the model's training and, after 100 epochs, displays a lower performance than the previous one, achieving an absolute accuracy of 7% in test, $\overline{acc} = 15.78\%$ and loss of 2.029 in training. The average accuracy and loss suggest insufficient training due to an early stopping.

IV. Conclusion

This project aimed at reproducing and applying the Deep Neural Network commonly referred to as PointNet [1]. Since

[2]The Loss refers to the final training loss.

[3]While implementing 200 epochs brought the best results, all but one models were trained on 100 epochs due to a limited computational power.

TABLE II
CLASSES CARDINALITY IN THE ORIGINAL DATASET

Class cardinality					
28	**2420**	**4772**	**3300**	**21**	**4019**
799	601	510	505	504	503
6474	**303**	**326a**	**6215**	**2456**	**30355**
496	489	484	484	473	461
3747	**3685**	**6156**	**65735**	**2291**	**712**
440	437	434	411	388	384
3030	**3433**	**2454**	**3011**	**30180**	**6213**
384	382	380	373	365	363
3027	**250**	**4854**	**971**	**10b**	**9359**
360	303	266	205	159	52

this network takes as input Point Clouds, a further effort was required to transform the original data. The deployed pipeline cleansed the data (RGB and monochromatic scenes) by extracting and normalizing Lego blocks and then rendered them as Point Clouds.

Even though the deployed architecture seems to work satisfactorily, some limitations and critical points could be found at different levels. Two main generalizations have been assumed at the pre-processing level: excluding all the Lego pieces whose dimensions were not correctly stored and overlooking the problem of overlapping bounding boxes. At the training level, a lack of proper machines (i.e., laptops equipped with a GPU) hindered the procedure that, in extreme cases, lasted almost two days and constrained the number of epochs to 100 (with the only exception stated above). Lastly, the imbalanced dataset slowed down the process and implied a verbose model. Overall, the architecture might be further enhanced by other combinations of parameters. Thus, any customisation affects the performance, and gives insights about the possibilities and limits of the generalization of PointNet to a new scenario [1].

REFERENCES

[1] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 652–660, 2017.

[2] C. K. Chua, C. H. Wong, and W. Y. Yeong, "Chapter four - software and data format," in *Standards, Quality Control, and Measurement Sciences in 3D Printing and Additive Manufacturing* (C. K. Chua, C. H. Wong, and W. Y. Yeong, eds.), pp. 75–94, Academic Press, 2017.

[3] K. R. Das and A. R. Imon, "Geometric median and its application in the identification of multiple outliers," *Journal of Applied Statistics*, vol. 41, no. 4, pp. 817–831, 2014.

[4] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, pp. 448–456, PMLR, 2015.

[5] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[6] E. M. Dogo, O. J. Afolabi, N. I. Nwulu, B. Twala, and C. O. Aigbavboa, "A comparative analysis of gradient descent-based optimization algorithms on convolutional neural networks," in *2018 International Conference on Computational Techniques, Electronics and Mechanical Systems (CTEMS)*, pp. 92–99, 2018.

[7] B. Léon, "Online algorithms and stochastic approximations," *Online Learning and Neural Networks; Cambridge University Press: Cambridge, UK*, 1998.

[8] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, p. 2121–2159, July 2011.

[9] K. Janocha and W. M. Czarnecki, "On loss functions for deep neural networks in classification," *arXiv preprint arXiv:1702.05659*, 2017.

[10] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*, vol. 112. Springer, 2013.

[11] L. N. Smith, "A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay," *arXiv preprint arXiv:1803.09820*, 2018.

V. CONTRIBUTORS

Project contributors		
Task/File	**Benedetti Emma**	**Hrovatin Lucia**
data_ingestion	15%	85%
data_splitter	50 %	50%
model	70 %	30%
solver	10%	90%
paper	50%	50%