# Clustering for Data-driven Unraveling Artificial Neural Networks

Felipe Costa Farias[1,2][0000−0001−7411−5562], Teresa Bernarda Ludermir[1][0000−0002−8980−6742], and Carmelo J. A. Bastos-Filho[3][0000−0002−0924−5341]

[1] Universidade Federal de Pernambuco, Centro de Informática, Recife PE, Brasil
`tbl@cin.ufpe.br`
[2] Instituto Federal de Educação, Ciência e Tecnologia de Pernambuco, Paulista PE, Brasil
`felipefariax@gmail.com`
http://www.springer.com/gp/computer-science/lncs
[3] Universidade de Pernambuco, Recife PE, Brasil
`carmelofilho@ieee.org`

**Abstract.** This work presents an investigation on how to define Neural Networks (NN) architectures adopting a data-driven approach using clustering to create sub-labels to facilitate the learning process and to discover the number of neurons needed to compose the layers. We also increase the depth of the model aiming to represent the samples better, the more in-depth it flows into the model. We hypothesize that the clustering process identifies sub-regions in the feature space in which the samples belonging to the same cluster have strong similarities. We used seven benchmark datasets to validate our hypothesis using 10-fold cross validation 3 times. The proposed model increased the performance, while never decreased it, with statistical significance considering the p-value $< 0.05$ in comparison with a Multi-Layer Perceptron with a single hidden layer with approximately the same number of parameters of the architectures found by our approach.

**Keywords:** Neural networks · Data-driven architecture · Sub-labels · Clustering · Representation learning.

## 1 Introduction

Computational Intelligence studies adaptive mechanisms to facilitate intelligent behavior in complex and dynamic environments. It is often applied in situations where heuristics are insufficient to solve a problem associated with uncertainty or stochastic behavior. Machine Learning is a branch of Computational Intelligence that allows computers to learn by experience (data) without being explicitly programmed [3]. The research's attention has grown due to the amount of available data and the proliferation of technologies, such as Internet of Things and Big Data [6]. Consequently, we have experienced the proposal of sophisticated models to learn from datasets with a large number of examples.

There are several types of Machine Learning algorithms. In Supervised Machine Learning, the algorithms during the training phase try to process inputs $X$ and create an association with known outputs $Y$. One of the most used families of algorithms to perform this task is Artificial Neural Networks (ANN) [5]. Several works investigate the automatic construction of ANN architectures. This problem is known as Neural Architecture Search (NAS). The NAS approaches aim to decrease human intervention during the modeling process of an ANN. It offers mechanisms to propose ANN architecture automatically. However, most of the strategies presented in the literature does not consider the characteristics of the data involved in the problem to build the ANN architecture.

Among the strategies for solving the NAS problem, we observed the application of global optimizers such as Genetic Algorithms (GA) [11]. We highlight three works that use Genetic Algorithms and Reinforcement Learning to perform this task. The NeuroEvolution of Augmenting Topologies (NEAT) [14, 15] uses GAs to find the structure and weights of ANNs, encoding these attributes as part of the individual deployed in the evolutionary process. The strategy of Reinforcement Learning was used in [16]. Despite the good results, they have used 800 NVidia K-40 GPUs for 28 days, needing 22,400 GPU hours of processing time. The same strategy was proposed in [17]. However, they used the concept of transferability to allow learning from a simple dataset and applying it to a more complex dataset. Again, the processing time is a big challenge. They used 500 NVidia P100 GPUs for four days, totaling 2,000 GPU hours. In order to explore the architecture space based on the current network and reusing its weights, a Reinforcement Learning meta-controller was used to grow the network depth/layer width in [1]. They used 5 GPUs during 2 days, training 450 networks. These methods can achieve reasonable results, but they are often time-consuming.

In a different branch, we have the work [9] that focuses on searching for the architecture composition progressively, starting from the simplest candidates to the more complex ones. This proposal is based on *Blocks* constructing *Cells* that will compose entire *Networks*. A Binary Particle Swarm Optimization (BPSO) algorithm was used in [10] to define the architecture of an ANN that has no regular layers. These proposal are interesting but does not explicitly take into account the distribution information regarding the dataset. It is important to observe this information since the model will use samples of the dataset to train and the neurons should behave plausibly with respect to its inputs. For example, the neurons should neither explode or vanish its activations for all the dataset but have specific activation patterns for each label.

Even though the NAS has good results, these methods try several architectures to find which one is the best through a search algorithm, often using several GPUs for several hours. Differently from NAS approaches, in this work we investigate how to design a data-driven ANN Architecture using clustering to discover the number of neurons of a given layer and iteratively increase the depth of this ANN without the need to initializing and training many different architectures. We only start with the inputs and apply our strategy to create the subsequent hidden and output layers. The layer that will be created is based on the current data representation that this layer will have as inputs. We use the dataset distribution in order to start with weights that activates more to specific labels and less to all the other labels. We have focused on tabular datasets since several real-world applications share this data representation. To investigate this problem, We have used benchmark datasets to evaluate our hypothesis. Our goals are to infer (i) the width (number of neurons) through clustering of the feature space to find sub-labels that share strong similarities, creating specific neurons that activate to specific sub-labels; and (ii) the depth (number of layers) of an ANN regarding the specificity of each dataset. The main idea concerns on each layer representing the samples in a more straightforward manner to the next ones. We aim to disentangle the representations into a space that is easier for the classifier on the last layers of the model to recognize the patterns. We name this process as Clustering for Data-driven Unraveling (DDU) Artificial Neural Networks.

The remainder of this work was organized as follows. We present the background information in Section 2. We describe the proposed methodology to data-driven evolve a ANN architecture in Section 3. We show the experimental arrangement and the results in Section 4. Finally, we present the discussion, conclusions, and future works in Section 5.

## 2 Our proposal

In this section, we present the proposal's details, comprising the creation of sub-labels and the definition of the layers.

Since some samples of a specific label can lay near the same region while other samples of the same label may be in other regions, it may harm the learning process. We have used clustering techniques to find the essential regions of interest. The clustering process aims to create sub-labels based on the proposal presented in [4]. The primary goal is to enhance the learning process. We use the clusters to map each one of the sub-labels to a single neuron in the forward layer and compose the entire layer. We add layers composed of neurons from clustering in an iterative manner. Fig. 1 presents an example of a possible data-driven defined architecture of a ANN through the processing of our proposal.

The *width* of each layer is defined by the number of neurons that compose a specific layer. We find the number of layers by applying the clustering process. On the other hand, the *depth* of the DDU ANN is the number of layers. The number of layers is also determined automatically in the appending new layers process. The appending process is applied iteratively until we reach a stop criterion. Fig. 2 shows the steps that compose our proposal.

Given the inputs $x_i \in X$ and the outputs $y_i \in Y$, we use GMMs (Gaussian Mixture Models) clustering algorithm to create sub-labels as shown in [4] using the $X$ as inputs. This process create the clusters regarding each label separately and apply the prediction of each created cluster to all the data in order to calculate the scores discussed in the next paragraph. We generate GMMs with

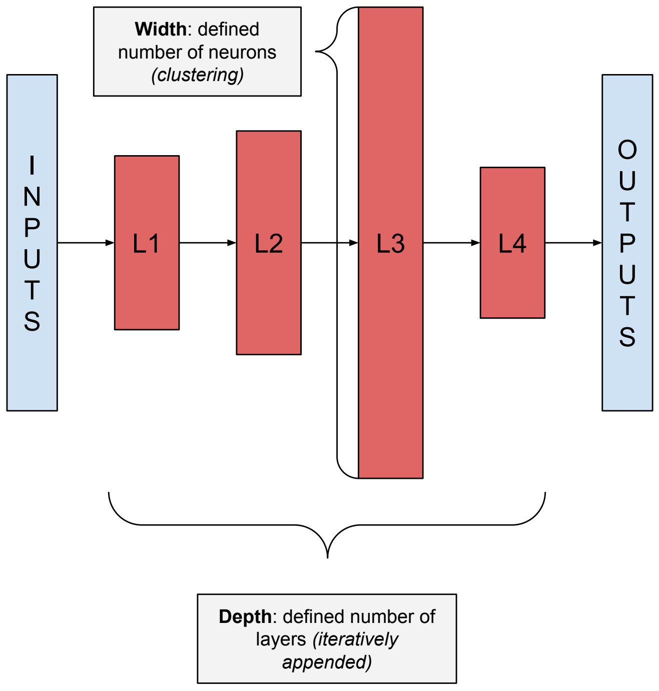
**Width**: defined number of neurons *(clustering)*
INPUTS
L1
L2
L3
L4
OUTPUTS
**Depth**: defined number of layers *(iteratively appended)*

**Fig. 1.** Example of architecture found by the Data-driven Unraveling technique. The inputs are the features of the samples, while the outputs are its labels.

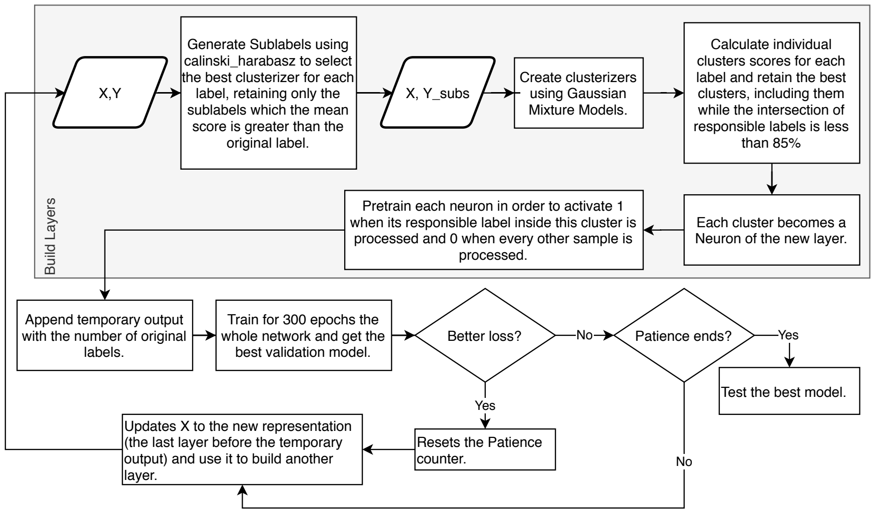
Build Layers
X,Y
Generate Sublabels using calinski_harabasz to select the best clusterizer for each label, retaining only the sublabels which the mean score is greater than the original label.
X, Y_subs
Create clusterizers using Gaussian Mixture Models.
Calculate individual clusters scores for each label and retain the best clusters, including them while the intersection of responsible labels is less than 85%
Each cluster becomes a Neuron of the new layer.
Pretrain each neuron in order to activate 1 when its responsible label inside this cluster is processed and 0 when every other sample is processed.
Append temporary output with the number of original labels.
Train for 300 epochs the whole network and get the best validation model.
Better loss?
No
Patience ends?
Yes
Test the best model.
Yes
Resets the Patience counter.
Updates X to the new representation (the last layer before the temporary output) and use it to build another layer.
No

**Fig. 2.** Diagram of the proposed methodology.

the number of clusters in $[2, 3, 4, 5]$ repeating the process for 2 times, totaling 8 GMMs generation. We decided to deploy the GMM technique since it presented the highest Calinski-Harabasz score [2] when compared to Silhouette Score for related. We also expect that the samples of the same label could converge to the same spatial region as the samples are processed by layers and we believe that it could be approximated by Gaussian distributions due to the Central Limit Theorem.

We defined purity and belongingness metrics, combining them into a single score assigned to each cluster. Since each cluster is assigned to a specific label or sub-label (responsible label), the score is related to that specific responsible label inside the cluster found. Purity is defined as the percentage of a specific label regarding all the elements inside this cluster. Belongingness is the percentage of a specific label within the cluster regarding all samples of this label. For each responsible label, we assign the cluster with maximum score. The rationale relies on the idea that the maximum score value returns all the samples of a single label (belongingness=100%) in a cohesive region without other labels inside (purity=100%). The score is minimum when it has a low number of an assigned label inside this cluster, regarding all the feature space (low belongingness), and the cluster has a significant number of samples not belonging to the assigned responsible label (low purity). As we need to define a single indicator in our methodology to assess the clustering process, we decided to combine these two metrics as follows:

$$score = kappa * purity + (1 - kappa) * belongingness \tag{1}$$

*kappa* is automatically calculated using the following strategy to give the same importance to the two metrics:

$$kappa = k_p \tag{2}$$

$$k_b = (1 - k_p) \tag{3}$$

$$k_p * \overline{p} = k_b * \overline{b} \tag{4}$$

where $k_p$ is the *kappa* regarding the purity and $k_b$ is the *kappa* regarding the belongingness, $\overline{b}$ is the average belongingness and $\overline{p}$ is the average purity for each label.

It is easy to observe that Eq. (4) can be rewritten as

$$kappa = 1 - \frac{1}{(\frac{\overline{b}}{\overline{p}} + 1)} \tag{5}$$

After defining the sub-labels, we evaluate if the sublabeled scenario is better than the original one. For each sub-label, we evaluate if the average score regarding the clusters with this responsible sub-label has increased compared to the averaged score of the clusters responsible for its original label that derived the sub-label. In this process, we may have chosen some, all, or none of the created sub-labels.

If we have chosen some sub-labels, we adjust the $Y$ to use the proposed sub-labels, treating them as different labels because it has shown different features to be considered for the same class. By doing this, we believe we can create more concise groups of samples, facilitating the classifier's learning process.

After the generation of possible sub-labels, we apply the GMMs clustering algorithm with the number of clusters in $[L, L+1, L+2, ...L+7]$ for 2 times, totaling 16 GMMs generation. We analyze each one of the clusters independently of the GMM generation process. We calculate their scores and assign their responsible labels. Then, we sort them, for each label, from the highest to lowest score. After this, for each label (or sub-label), we select the clusters until the intersection of the elements of the responsible label inside the specific cluster - we save the indexes of the responsible labels handled by each cluster - is less than 85% of the union of the responsible labels already retained by the clusters already chosen for this specific label (or sub-label).

Each one of the retained clusters is mapped into a neuron and pre-trained for 5 epochs with a higher learning rate of 0.01 trying to activate 1 for the responsible label elements inside this cluster and 0 otherwise. Our goal is to make this neuron trigger specifically for activations of this label (or sub-label) when the model processes a sample near that region of clustering. So we applied a traditional SGD training with this modified outputs. This process finishes with a layer created based on the clusters and the weights pre-trained to activate in the responsible labels. We append this layer with a SELU [8] function activation since it appears to self-normalize the network, into the model, and append a temporary output layer with the number of labels of the original problem together with a LogSoftmax activation.

We train the model for 300 epochs and select the model with the lowest validation loss. If the global loss in this validation set decreases, we reset the patience parameter and do the same process to add another layer using the activations of the last layer (before the temporary output) as inputs to the new layer. If the loss increases, we increment the patience counter and stops the depth growing when the patience counter reaches 3.

We present the pseudo-code describing the algorithm to perform the DDU method in Algorithm 1.

**Algorithm 1** Algorithm for DDU

```
Require: X, Y
 1: C = []
 2: patience = 0
 3: while patience < 3 do
 4:    current_y = Y
 5:    Y_subs = generate_sublabels(X)
 6:    if better_performance(Y_subs) then
 7:       current_y = Y_subs
 8:    end if
 9:    clusters = create_GMMs(X)
10:    calculate_scores(clusters, Y_subs)
11:    sort(clusters) // from highest to lowest score
12:    for label in labels: do
13:       for cluster in clusters[label]: do
14:          if responsible_label_intersection < 85% then
15:             C.append(cluster)
16:          end if
17:       end for
18:    end for
19:    Pretrain neurons regarding C
20:    Append temporary output
21:    Train entire model for 300 epochs
22:    loss = model(validation_split)
23:    if loss decreased then
24:       best_model = model
25:       patience = 0
26:    else
27:       patience++
28:    end if
29:    X = model.represent(X)
30: end while
31: return best_model.
```

## 3 Experiments and Results

We select seven benchmark datasets to validate our proposal. We present the datasets in Table 1.

**Table 1.** Datasets used as benchmarks

| **Dataset** | **# Instances** | **# Features** | **# Labels** |
|---|---|---|---|
| Glass | 214 | 9 | 6 |
| Ionosphere | 315 | 34 | 2 |
| Iris | 150 | 4 | 3 |
| Pima | 768 | 8 | 2 |
| Satimage | 6430 | 36 | 6 |
| Tic-tac-toe | 958 | 9 | 2 |
| Vehicle | 846 | 18 | 4 |

We have used a Stratified 10-Fold and repeated it three times, resulting in 30 trials. In the process of the Stratified 10-fold, we divided the entire dataset into ten mutually exclusive splits. We maintained the proportion of labels in each subset. The first run uses the first split for test, whereas the other nine are used for Train/Validation. In the second run, the split number 2 is used for test, whereas the other nine are used for train/Validation. We repeat the process until the last run uses the split number 10 for test and the first nine splits are used for train/Validation. We also have saved the indexes that were presented in each fold to be used during the comparison with a different model: Multilayer Perceptrons (MLPs) with only one hidden layer with the number of weights approximately the same of the architecture generated by our proposal, regarding the samples presented in that arrangement.

We have used the PyTorch [12] and Scikit-Learn [13] libraries to implement our proposal. We have chosen the ADAM optimizer [7] with a learning rate of 0.001 for the training of the entire model with the Negative Log-likelihood loss and 0.1 for the pre-training of each neuron given their related clusters with the Mean Squared Error loss.

Table 2 compares the results of the proposed technique (DDU) and MLPs with only one hidden layer with the number of weights - and not the number of neurons - approximately equal to each one of the architectures created by the DDU. We have used this strategy to compare the neuron arrangement's importance in different layers, with approximately the same number of parameters. For each dataset, the average accuracy of 30 runs for each set (train, validation, and test) is given, and its standard deviation appears between parenthesis. We also have applied the Wilcoxon statistical test between the two techniques comparing the same dataset, highlighting the text where it presented a $p - value < 0.05$.

We split the data into Train, Validation and Test sets, and we expect these three splits to share the same sample distribution. The Validation Set is often

**Table 2.** Results in Train, Validation and Test sets regarding DDU and MLP models for each dataset

| **Model** | **Dataset** | **Train** | **Validation** | **Test** |
|---|---|---|---|---|
| DDU | pima | 77.99 (0.01) | + **78.48 (0.02)** | 76.74 (0.03) |
| MLP | pima | 78.07 (0.01) | 76.71 (0.03) | 76.39 (0.02) |
| DDU | vehicle | + **84.12 (0.02)** | + **80.98 (0.03)** | + **80.90 (0.04)** |
| MLP | vehicle | 82.81 (0.01) | 79.34 (0.02) | 79.40 (0.03) |
| DDU | glass | 66.88 (0.05) | + **65.01 (0.05)** | 62.16 (0.10) |
| MLP | glass | 66.09 (0.03) | 60.83 (0.06) | 62.33 (0.09) |
| DDU | tic-tac-toe | 82.33 (0.06) | 77.89 (0.06) | 67.14 (0.11) |
| MLP | tic-tac-toe | 82.46 (0.04) | 77.98 (0.05) | 70.79 (0.10) |
| DDU | iris | 96.04 (0.02) | 97.16 (0.03) | 95.33 (0.06) |
| MLP | iris | + **97.36 (0.01)** | 95.98 (0.03) | 95.56 (0.06) |
| DDU | satimage | + **91.16 (0.01)** | + **89.52 (0.01)** | + **88.86 (0.02)** |
| MLP | satimage | 89.58 (0.01) | 88.17 (0.01) | 88.06 (0.01) |
| DDU | ionosphere | 93.50 (0.02) | + **89.37 (0.03)** | 84.82 (0.08) |
| MLP | ionosphere | 93.70 (0.01) | 88.06 (0.03) | 85.96 (0.08) |

used as a proxy to decide when to stop the learning process of a model because we assume that it would share the distribution/behavior of the unseen test data. Our technique presented better accuracies in the Validation Set, which was not necessarily reflected in the Test Set. It maybe has occurred due to our process of K-Fold splitting. As we have used the Stratified K-Fold, we expected that the train, validation and test splits should have approximately the same feature distribution. Still, it probably has not happened since it guarantees the same percentage of labels in each split, but each split may still not be statistically equivalent in the feature space distribution. Also, the split was based on the original labels. That probably could generate completely different sub-labels during the process once the feature space distribution information was not used at the splitting process, as it is based solely on the output labels. Considering that in each trial, the same indexes were presented to both techniques, the DDU appears to have a better knowledge extraction since the validation accuracy improved and is statistically significant, and the training accuracy not necessarily accompanied. Maybe a better way to split the data should be a stratification based in regions that each sample occupies in the feature space. In this way, inside the same region/cluster, we should stratify the samples into the K splits.

As one can see in Table 3, the DDU generated some architectures, including ones with more than one hidden layer, that show better accuracies in different splits. It is worth remembering that the number of neurons of the MLPs was calculated to match the number of parameters (weights) found by DDU in each run. As it shows, meaning that not only the learning capacity (stored in weights) matters but also the arrangement of neurons in different layers influences the results.

The architectures with the minimum, median, and maximum number of neurons/layers generated for each dataset by the DDU are presented in Table 3. The column **Architecture** shows the complete NN architecture starting from the input layer to the output layer. The column **Weights** presents the number of weights existent in that specific architecture. The columns **Train**, **Val.** and **Test** show the accuracy for each split.

**Table 3.** Minimum, Median and Maximum number of weights in found Architectures

| **Dataset** | **Architecture** | **Weights** | **Train** | **Val.** | **Test** |
|---|---|---|---|---|---|
| glass | [9 9 6] | 216 | 64,58 | 60,42 | 50,00 |
| glass | [9 12 12 6] | 405 | 67,36 | 65,31 | **71,43** |
| glass | [9 11 14 16 13 13 10 6] | 1125 | **72,92** | **73,47** | **71,43** |
| ionosphere | [34 2 3 2] | 1236 | 92,83 | 88,61 | 80,00 |
| ionosphere | [34 5 4 2] | 1354 | 92,41 | 88,61 | **94,29** |
| ionosphere | [34 5 8 6 2 6 3 5 2] | 1481 | **96,20** | **89,87** | 82,86 |
| iris | [4 3 3] | 37 | **97,03** | 97,06 | 93,33 |
| iris | [4 4 5 3] | 67 | 96,04 | 97,06 | **100,00** |
| iris | [4 9 6 7 6 5 3] | 235 | **97,03** | **100,00** | **100,00** |
| pima | [8 2 2] | 84 | **77,22** | 78,61 | 76,62 |
| pima | [8 5 3 4 3 5 2] | 168 | **77,22** | 76,88 | **79,22** |
| pima | [8 6 7 5 5 2 7 5 3 4 2] | 308 | 76,88 | **79,77** | 77,63 |
| satimage | [36 10 11 6] | 1832 | 89,91 | 88,80 | 88,02 |
| satimage | [36 13 12 15 6] | 2190 | 90,90 | 89,15 | 88,18 |
| satimage | [36 11 12 15 13 14 18 13 13 12 13 12 13 12 11 6] | 4014 | **93,32** | **90,81** | **90,67** |
| tic-tac-toe | [9 5 8 2] | 182 | 73,99 | 75,00 | 71,88 |
| tic-tac-toe | [9 7 6 6 5 7 3 2] | 314 | 87,62 | 82,41 | **72,92** |
| tic-tac-toe | [9 8 8 3 3 5 5 5 5 3 4 4 4 3 4 2] | 431 | **88,08** | **86,57** | 47,92 |
| vehicle | [18 12 4] | 588 | 79,82 | 78,53 | 78,82 |
| vehicle | [18 13 12 4] | 762 | 82,84 | **81,15** | 86,90 |
| vehicle | [18 12 15 9 8 4] | 959 | **83,19** | **81,15** | **89,29** |

We highlight that the generated architectures do not necessarily create shapes that always increase or decrease the number of hidden neurons. It might increase, decrease, and increase again if the representation became locally worse at a given depth. The layers near the output tend to have a low number of hidden neurons, probably indicating that less resource is needed to treat the problem at that depth. It happens since representations of the samples were facilitated by the possible disentanglement of the feature space done by all the previous layers.

Regarding the Validation split, the accuracy increased as the architecture used more weights and/or layers in most cases. It occurs since the samples are processed through the layers, thus probably creating better representations which the final output layer can classify with fewer efforts. On the other hand, the layers near the input tend to have a larger number of neurons due to the high entanglement of the features at the early stages.

## 4 Conclusions

In this paper, we presented a preliminary study on how to define a Data-Driven Neural Network Architecture without creating several Networks. To assess this, we have used GMMs clustering to define each layer's width and iteratively appending layers, increasing depth, aiming to find better and possibly disentangled representations, easing the model learning process.

As presented in Table 2, the performance has increased statistically (Wilcoxon significance test with $alpha = 0.05$) on two occasions in the Test split while never decreased it. It also corroborates with our initial hypothesis that clustering the feature space may reveal the number of necessary neurons representing the samples in a more organized way. The quantity of neurons seems correlated with the high non-linearity entanglement at the first layers and with the possible low entanglement at the last layers of the model. If we consider the Validation split, that is considered as a proxy to the accuracy in Test split, our approach was better in 5 cases out of 7.

Our proposal has shown some exciting results, and our hypothesis could be evaluated in these initial experiments. For future works, we intend to assess the technique in more benchmark datasets and adapt the method to work with datasets usually used in Deep Learning tasks such as MNIST and CIFAR10. We also want to perform more evaluations using the data-driven found architectures to evaluate if a pruning process would perform successfully or if it would fail, leading us to believe that the DDU is capable of creating effective architectures with a small number of useless neurons/weights. Another valid investigation is to split the data stratified by region and by labels inside each region. The evaluation of other Clustering Techniques and strategies to retain the cluster may impact the performances. Also, more tests on hyperparameters values need to be done to have more robust conclusions about its impact on the learning process. We also plan to compare the technique with other approaches, such as the NEAT.

## References

1. Cai, H., Chen, T., Zhang, W., Yu, Y., Wang, J.: Efficient architecture search by network transformation. In: Thirty-Second AAAI conference on artificial intelligence (2018)
2. Caliński, T., Harabasz, J.: A dendrite method for cluster analysis. Communications in Statistics-theory and Methods **3**(1), 1–27 (1974)
3. Eberhart, R.C., Shi, Y.: Computational Intelligence: Concepts to Implementations. Elsevier Science (2011)
4. Farias, F.C., Bernarda Ludermir, T., Bastos-Filho, C.J.A., Rosendo da Silva Oliveira, F.: Analyzing the impact of data representations in classification problems using clustering. In: 2019 International Joint Conference on Neural Networks (IJCNN). pp. 1–6 (July 2019)

5. Goodfellow, I., Bengio, Y., Courville, A., Bengio, Y.: Deep learning, vol. 1. MIT press Cambridge (2016)
6. Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of things (iot): A vision, architectural elements, and future directions. Future generation computer systems **29**(7), 1645–1660 (2013)
7. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
8. Klambauer, G., Unterthiner, T., Mayr, A., Hochreiter, S.: Self-normalizing neural networks. In: Advances in neural information processing systems. pp. 971–980 (2017)
9. Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.J., Fei-Fei, L., Yuille, A., Huang, J., Murphy, K.: Progressive neural architecture search. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 19–34 (2018)
10. Liu, J., Gong, M., He, H.: Nucleus neural network: A data-driven self-organized architecture. arXiv preprint arXiv:1904.04036 (2019)
11. Mitchell, M.: An introduction to genetic algorithms. MIT press (1998)
12. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Wallach, H., Larochelle, H., Beygelzimer, A., dAlché Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32, pp. 8024–8035. Curran Associates, Inc. (2019)
13. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
14. Stanley, K.O., Clune, J., Lehman, J., Miikkulainen, R.: Designing neural networks through neuroevolution. Nature Machine Intelligence **1**(1), 24–35 (2019)
15. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary computation **10**(2), 99–127 (2002)
16. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578 (2016)
17. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 8697–8710 (2018)