J Vis (2022) 25:593–612
https://doi.org/10.1007/s12650-021-00809-4

REGULAR PAPER

Martha Dais Ferreira · Gabriel D. Cantareira · Rodrigo F. de Mello · Fernando V. Paulovich

Neural network training fingerprint: visual analytics of the training process in classification neural networks

Received: 20 July 2021 / Revised: 1 October 2021 / Accepted: 5 October 2021 / Published online: 2 November 2021

Abstract The striking results of deep neural networks (DNN) have motivated its wide acceptance to tackle large datasets and complex tasks such as natural language processing, facial recognition, and artificial image generation. However, DNN parameters are often empirically selected on a trial-and-error approach without detailed information on convergence behavior. While some visualization techniques have been proposed to aid the comprehension of general-purpose neural networks, only a few explore the training process, lacking the ability to adequately display how abstract representations are formed and represent the influence of training parameters during this process. This paper describes *neural network training fingerprint (NNTF)*, a visual analytics approach to investigate the training process of any neural network performing classification. NNTF allows understanding how classification decisions change along the training process, displaying information about convergence, oscillations, and training rates. We show its usefulness through case studies and demonstrate how it can support the analysis of training parameters.

Keywords Neural network visualization · Neural network training · Deep learning · Visual analytics · Visualization

1 Introduction

Deep neural networks (DNNs) are currently among the state-of-the-art for analyzing large-scale and complex datasets due to their ability to abstract high-level patterns and model data beyond most heuristics (Goodfellow et al. 2016; LeCun et al. 2015). They are widely used for natural language processing, face and speech recognition, and artificial data generation. One common application for DNNs is data classification consisting of inferring a model $f : \mathcal{X} \rightarrow \mathcal{Y}$ to correctly label unknown data based on a set of known labeled examples $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$, in which x_i is a set of features in X, and y_i is its corresponding label in Y.

M. D. Ferreira (✉) · F. V. Paulovich
Faculty of Computer Science, Dalhousie University, Halifax, NS, Canada
E-mail: daismf@dal.ca

F. V. Paulovich
E-mail: paulovich@dal.ca

G. D. Cantareira
Department of Informatics, King's College London, London, UK
E-mail: gabriel.dias_cantareira@kcl.ac.uk

R. F. de Mello
University of São Paulo, São Carlos, Brazil
E-mail: mellorf@gmail.com

Although it is well established in the literature that DNNs yield outstanding results in classification for different domains, even with fixed architectures, training parameters are usually empirically selected based on time-consuming trial-and-error strategies (Goodfellow et al. 2016; Krizhevsky 2010; Krizhevsky et al. 2012; LeCun et al. 2015; Srivastava et al. 2014).

These parameters strongly influence the training process in terms of convergence speed and overfitting, such as the learning rate and the momentum of error derivatives (Goodfellow et al. 2016; LeCun et al. 2015), and they weigh heavily in how training advances through the optimization space, often impacting the difference between good results and failures.

Novel DNN techniques make use of even more parameters. Regularization methods (Srivastava et al. 2014; LeCun et al. 2015), for instance, are designed to avoid overfitting by controlling how many iterations or data instances steer the model in specific directions. The weight decay regularization term (Goodfellow et al. 2016; Krogh and Hertz 1992; LeCun et al. 2015) is an example of a method to improve generalization while training DNNs. Still, it adds costs to the parametrization process since more elements need to be set and tested by users every time a new network is trained.

Many visual approaches were proposed to aid the comprehension of general-purpose neural networks focusing on the abstract representations generated by a model and the behavior of the filters (Mahendran and Vedaldi 2016; Zeiler and Fergus 2014). Only, a few attempts to represent the training process itself, and those dedicated to it, focus on single regularization terms or convergence values (Cantareira et al. 2020; Rauber et al. 2017; Srivastava et al. 2014; Yosinski et al. 2015), lacking the ability to display how abstract representations are formed and to express the influence of training parameters during this process.

Aiming at filling this gap, we propose *neural network training fingerprint (NNTF)*, a visual analytics approach to improve analysis and understanding of the training process of neural networks with a particular interest in deep neural networks (DNNs). Although NNTF can be applied to any classification model that uses an iterative training process, we focus on DNNs due to their popularity, shedding some light on the complex convergence process and decision boundary formation. Our approach is presented as a complement to usual quality metrics, such as accuracy or confusion matrices, providing information about the convergence behavior not conveyed by such metrics to aid in understanding training parameters and observing their effects in the convergence process.

The analysis supported by NNTF provides information about the relationship between consecutive training iterations, allowing the study of the learning progress, existing patterns, loops, robustness, and potential pitfalls. NNTF makes it possible to understand the influence of training parameters, convergence, and even how network complexity and training parameters impact the learning process by combining visualization techniques with methods from dynamic systems. Our approach is independent of the model architecture, as it only relies on network outputs and training iterations.

In summary, the main contributions of this paper are as follows:

- A new visual analytics framework to support the interpretation of DNNs learning processes, being non-intrusive and simple to use. It can be employed with well-established network implementations, such as the TensorFlow (Abadi et al. 2016), without requiring changes in code;
- The use of *recurrence quantification analysis (RQA)*, a widely known dynamical system tool, to support the analysis of the training process of DNNs.

The remainder of this paper is organized as follows. In Sect. 2, we discuss related work and present techniques with similar goals to our approach. In Sect. 3, the proposed visualization method is described, showing how to analyze the network training behavior. In Sect. 4, case studies are discussed, and in Sect. 5 the limitations of our approach are outlined. Conclusions are drawn in Sect. 6.

2 Related work

Over the past few years, several visual analytics tools and techniques have been devised to support the understanding of *artificial neural networks (ANNs)* (Hohman et al. 2018; Liu et al. 2017) primarily focusing on the analysis of filter activations and features. For filter analysis, Samek et al. (2017) proposed a quantitative evaluation of representing activations throughout heatmaps. They compare three heatmap algorithms, concluding that the layer-wise relevance propagation (LRP) (Bach et al. 2015) is the best choice to reveal filter information of Deep Neural Networks (DNN). Shang et al. (2016) also proposed a visual representation to compare filters resulting from different activation functions.

Although useful, filter visualizations cannot be used to show what is modeled by an artificial neural network. Aiming to overcome such limitations, some visualization techniques give support to understand the produced features. A well-known example is the Deconvnet (Zeiler and Fergus 2014), which assists the analysis of Convolutional Neural Networks (CNN) architectures, helping to understand the reasons behind the attained results and to improve their performances. Deconvnet reconstructs input data (images) at each CNN layer to show the features extracted by filters, supporting the detection of incidental problems based on user inspections. (Simonyan et al. 2014) developed two visual representations to support image segmentation based on Deconvnet, allowing feature inspection after the execution of the back-propagation algorithm and the summarization of the features produced for each dataset class. Another feature-based visualization tool is introduced in Zintgraf et al. (2017), which helps to identify the impact of filter sizes on classification tasks and how the decision process is conducted.

Supporting both filter and feature analysis, Erhan et al. (2009) proposed a strategy to identify the features detected by filters after their activation functions, allowing visual inspection of the impact of model initialization and if features are humanly understandable. Similarly, Mahendran and Vedaldi (2016) proposed a method to inspect images and check whether they are understandable using three different visualizations: one for reconstructing image representations, one for presenting filter activations, and another to improve the stimulus of the feature extractor. Babiker and Goebel (2017) also proposed a visualization tool to understand features by supporting the identification of unnecessary features filtered along with layers. Kahng et al. (2018) introduced a method to explore features produced by CNNs through projecting the activation distances using the t-Distributed Stochastic Neighbor Embedding (t-SNE) Van Der Maaten (2014). Finally, Gu et al. (2020) proposed a visualization tool to interpret the relevance of the internal features in the classification results, supporting diagnosis tasks conducted by experts. Although relevant, those visualization techniques present information about the features and filters of a network. They cannot give insights into the influence of the training parameters and the overall learning process.

There are a few visualization techniques designed to support the training process analysis. Yosinski et al. (2015) proposed a framework to support the study of DNN architectures, i.e., the number of layers and filters per layer and the weight decay regularization term (Krogh and Hertz 1992; LeCun et al. 2015). Two visual metaphors are proposed, providing features and filter activation visualizations. Both allow the analysis of the training process states to support the interpretation of the influences resultant from the training parameters. However, they do not offer any insight regarding the convergence of the learning generalization. Our approach allows the study of network convergence and generalization easily and intuitively while also aids the analysis of individual training parameters.

Liu et al. (2017) proposed DGMTracker, a visualization method for exploring relationships between neuron output, loss function, and data flow for generative networks. While informative, DGMTracker can be costly, as it saves a snapshot for the entire model for every observed iteration. Liu et al. (2018) is another approach focused on the evolution of neuron outputs, allowing the observation of weights for different neurons over training epochs, different data instances, and the correlation between the two. Other models dedicated to showing specific network structures over time exist, such as Wang et al. (2018), a visualization that can display the evolution of movement-reward patterns in Q-networks during reinforcement training.

To avoid overfitting while combining different architectures, Srivastava et al. (2014) introduced the Dropout regularization method. They used visual representations to evaluate the dropout impact on the filters and CNN features after the training process. Such visualizations allow inspecting the generalization process and the study of the dataset size. However, the supported analysis is simplistic, missing information about essential training parameters, such as learning rate and momentum. In our approach, we also allow a refined analysis of all critical training parameters with the visual interpretation of the learning process.

Rauber et al. (2017) also presented a visualization approach for analyzing the learning evolution and training convergence by showing the relation among features extracted by DNNs and processing units. The method consists of projecting the filters activation and layers before and after the training process using t-SNE and multidimensional scaling (MDS) (Borg and Groenen 2003) techniques. Despite allowing the comparison between different networks at a certain level, the proposed strategy does not offer enough information to thoroughly analyze how parameters affect the training stage or how and when convergence is achieved.

DeepEyes, developed by Pezzotti et al. (2018) provides an overview of DNNs, being capable of identifying when a network architecture requires more or fewer filters and layers. DeepEyes provides the identification of filter problems and unnecessary layers with a progressive visualization. It employs scatterplots and heatmaps to show filter activations and analyze the feature space. On the other hand, Cantareira

et al. (2020) introduced a novel approach to investigate neural networks structure through vector fields, representing the general flow of information. Such vector fields are obtained from the trajectories produced by projections of multiple stages of the model.

Finally, there are a few approaches dedicated to visualizing the properties of training data. The OoDAnalyzer, by Chen et al. (2020), is dedicated to investigating the properties of data outliers and how they differ from other training samples. Gschwandtner and Erhart (2018) proposed a model to explore potential quality problems for time-series data. Ma et al. (2019) presented an approach to visualize adversarial vulnerabilities in training data. Hohman et al. (2020) presented a visualization approach to analyze evolving training data sets. These approaches aim to provide a better understanding of the relationship between training data and model behavior.

All of the discussed techniques for training visualization rely on displaying elements from the internal structure of neural networks. In general, they are essentially directed at analyzing network architecture, giving little support to guide users to evaluate the impact of the training parameters on the convergence process of an ANN model, which is directly associated with the network generalization. To the best of our knowledge, our approach is the first that combines dynamic system measures with visualization techniques to guide the parametrization process of ANNs, with the potential to reduce their overall complexity, while keeping as model-agnostic as possible, using only model outputs.

3 Neural network training fingerprint

The *artificial neural network (ANN)* structure was inspired by the mammals' cortex, composed of simple and complex visual cells. Simple cells are responsible for extracting basic features, while the complex ones combine local features producing representations (Scherer et al. 2010). This structure hierarchically manipulates the data through layers (complex cells), each having a set of processing units (simple cells) to extract local features.

In ANN classification tasks, each processing unit divides the data space using a linear function (a.k.a. hyperplane), which is placed to obtain the best separation between different class labels. The connections among processing units are responsible for combining the half-spaces built up by those linear functions to produce nonlinear separability of data spaces in an attempt to produce good classification results (Goodfellow et al. 2016; LeCun et al. 2015). The learning process requires labeled examples to measure error and

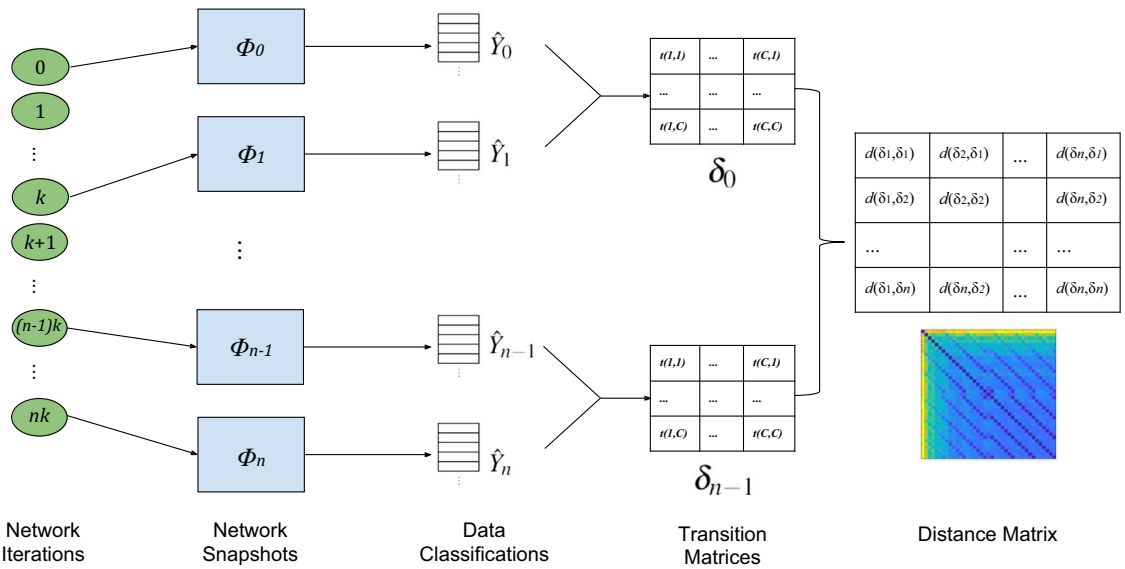


Fig. 1 Overview of our approach to generating the neural network training fingerprints. Every k iterations, the network output is saved in the form of snapshots Φ_i, resulting in p classifications $\hat{Y}_i$. Then, for every pair of classifications $(\hat{Y}_i, \hat{Y}_{i+1})$, a transition matrix δ_i is obtained, in which an entry δ_{mn} is the number of elements classified as m in $\hat{Y}_i$ and n in $\hat{Y}_{i+1}$. Those matrices represent the network learning progress along with the k iterations. Finally, we compute the $(p-1) \times (p-1)$ distance fingerprint matrix D, which contains the dissimilarity levels between all matrices δ_i. This process is non-intrusive, given it uses only the network outputs

perform training by adapting the weights associated with those processing units. Such adaptation follows the gradient descent of some loss function measured regarding the expected and the obtained network outputs (Goodfellow et al. 2016; LeCun et al. 2015).

In this paper, our goal is to help machine learning experts to understand the network training process and the impact of different parameterizations on it. The proposed approach assists in the visualization of convergence details, something that accuracy or loss cannot provide since even for stable accuracy, instances may still be transitioning between classes, indicating that there are still uncertain on these instances' classification despite the model convergence. While it is possible to use information from every training iteration in this process, we use model snapshots taken every k iterations that represent 1 epoch. Parameter k determines the visualization resolution, and although our choice for it is empiric, k may also be set to match certain relevant aspects of the training data. If k corresponds to less than one epoch, a more detailed view is presented, showing the impact that the batch caused to the model. On the other hand, if k is larger than one epoch, an overview of the model convergence is provided, showing its robustness and stability. Our approach only requires the labels (classes) produced by the network along learning iterations to assess how classifications evolve, so it is a non-intrusive process that can be used coupled to any existing available code without requiring drastic changes. However, it is worth mentioning that our approach cannot provide information when there are no transitions between classes, i.e., instances do not move from one class to another along iterations.

Our approach is summarized in Fig. 1. Considering $\hat{Y}_i$ to be the network output labels for training set in snapshot Φ_i, at the training iteration $i \cdot k$, a transition matrix δ_i is computed between $\hat{Y}_i$ and $\hat{Y}_{i+1}$. This transition matrix is represented as a matrix $C \times C$, in which C is the total number of labels. Each entry δ_{nm} summarizes the number of instances classified as n at snapshot Φ_i and classified as m at snapshot Φ_{i+1}. In a converging scenario, the δ_is should approach more and more to a diagonal matrix as iterations are executed. The transition matrix provides valuable information on how training evolves and can be seen as a feature vector containing values for every possible class transition in the system. However, such a matrix cannot provide an overview of the entire convergence process. Based on that, we construct a distance matrix D to capture similarity patterns among transition matrices, allowing the analysis of cyclic or repetitive patterns and abrupt changes in classification. D contains the pairwise distance between all transition matrices δ_i ordered from left-to-right, top-to-bottom, from the first to the last transition matrices produced as the network training evolves. In our model, each entry $d_{ij} \in D$ is calculated using a function $d_{ij} = d(\delta_i, \delta_j)$ that seeks to reflect the flow between labels in absolute terms over time, i.e., the actual number of instance transitions that are different between matrices.

As a requirement, d_{ij} should be independent of the dataset size, that is, the total number of possible transitions. Additionally, we want to capture the dissimilarity between transition matrices with small and large inter-class transitions. As the amount of inter-class transitions is expected to reduce over time as the network converges, each transition's relative importance must increase. With this in mind, we developed a simple function based on the Manhattan (city block) distance to ignore the diagonal elements of δ_i that is normalized by the total number of inter-class transitions.

$$d_{ij} = \frac{\sum_{m \neq n} |\delta_{i_{mn}} - \delta_{j_{mn}}|}{\sum_{m \neq n} (\delta_{i_{mn}} + \delta_{j_{mn}})}, \tag{1}$$

where $\delta_{i_{mn}}$ and $\delta_{j_{mn}}$ are the elements in row m, column n of $\delta_{i_{mn}}$ and $\delta_{j_{mn}}$ and $1 \leq m, n \leq C$. If both δ_i and δ_j are diagonal matrices, we assume $d_{ij} = 0$.

Other options of distance formulations were tested, such as the Earth Mover's Distance. However, we opted for simplicity so that the interpretation of the conveyed information is as straightforward as possible. Another possibility is the Hamming distance between raw classification data from the network at iterations $i \cdot k$. Still, there would be no normalization factor, and the relative aspect of the distance matrix would be lost. Also, other normalizations can be used instead of the one employed in Eq. (1), adjusting the proposed requirement according to different analytical perspectives. For instance, the distance could be divided by the number of samples to give insights into the training process considering data sets sizes.

Once a distance matrix D is produced, it can be analyzed to discover features regarding the training process behavior, as discussed in the next section. Another possibility is the comparison between two distance matrices to understand the differences in their convergence progressions. While many conclusions can be drawn by observing the matrices side by side, we also use a differentiation matrix to highlight differences that may not be as easy to perceive. For two distances matrices D^A and D^B, the differentiation matrix is given by:

$$\hat{D}_{AB} = D^A - D^B \tag{2}$$

Although a simple idea, such a matrix allows various analytical scenarios, such as comparing different sets of training parameters or analyzing differences in convergence for training and test sets. It is worth mentioning that the differentiation matrix does not represent a distance metric, as $\hat{D}_{AB} = -\hat{D}_{BA}$, but it is not required to be.

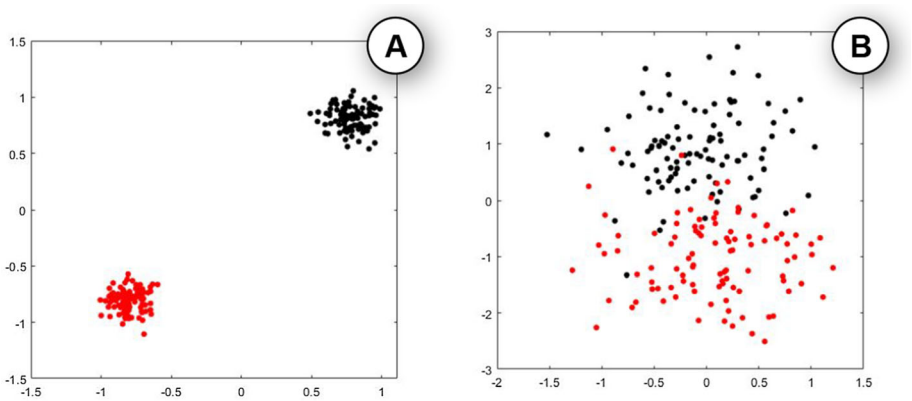
3.1 Fingerprint visualization

To illustrate how the visualization of these matrices as heatmaps can reflect relevant information about ANN training, we introduce some examples generated with synthetic datasets. Here we consider a single-layer network with one processing unit, which linearly separates the space using one hyperplane. The produced datasets are composed of 200 elements using two attributes (points in 2D) also divided into two classes. We considered two scenarios, A and B, in which the first is linearly separable, the second presents class overlapping. Figure 2 presents these two synthetic datasets.

The network is expected to converge in scenario A since classes are linearly separable. Figure 3 shows the visualization obtained for the first 200 iterations, with $k = 10$. Figure 3a consists of the distance matrix D, representing the distances between class transitions, which show a burst of high dissimilarity before reaching a zone that is completely similar to itself but dissimilar to everything else. It is safe to assume that this constant zone is where the network has stabilized, and no more transitions are happening. Figure 3b consists of the transition matrices δ_i, whose transformation into diagonal matrices confirms the network has indeed stabilized—the only transitions observed are from/to every class to itself.

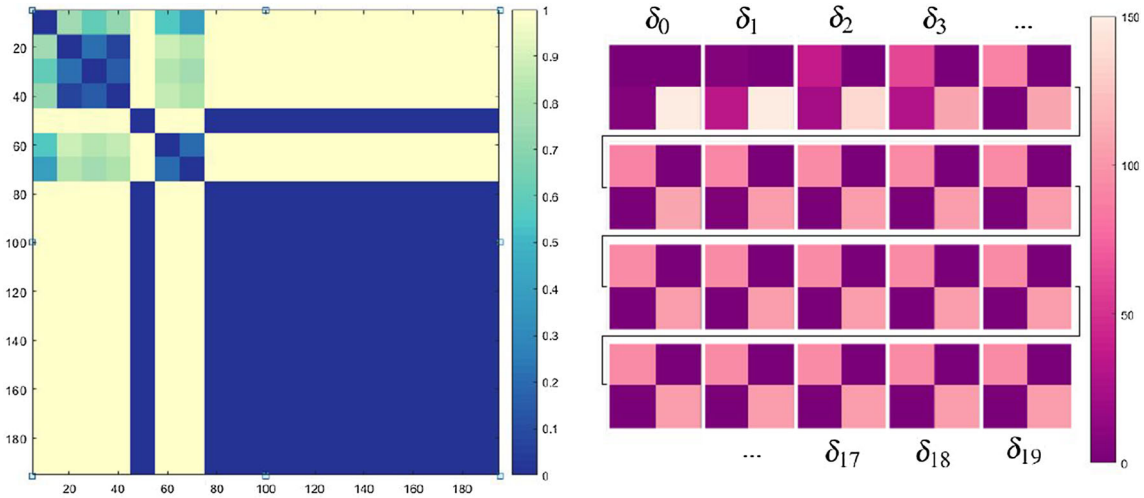
Figure 4 illustrates the visualization obtained for dataset B, which contains class overlapping. The visualization corresponds to 6000 training iterations with $k = 50$. In this example, the hyperplane dividing the space cannot completely separate the classes, resulting in oscillations. Such oscillations result in diagonal patterns in the dissimilarity matrix (Fig. 4a), indicating similarities among current and past network states, which may also be seen in the transition matrices themselves (Fig. 4b). In summary, similarities among states reveal that the optimization follows a repeating pattern, possibly indicating that processing units are not complex enough to separate the data space. Another factor that may result in repeating patterns is an excessively large learning rate, making it difficult to find the minimum of some error function while computing the gradient descent method.

In general, these heatmap visual representations present the classification behavior along with iterations, which relies on network weight adaptation. This behavior may reveal some insights into the decision process and network complexity. The dissimilarity matrix presents an overview of the learning process, indicating whether an ANN model requires changes in its architecture or training parameters.



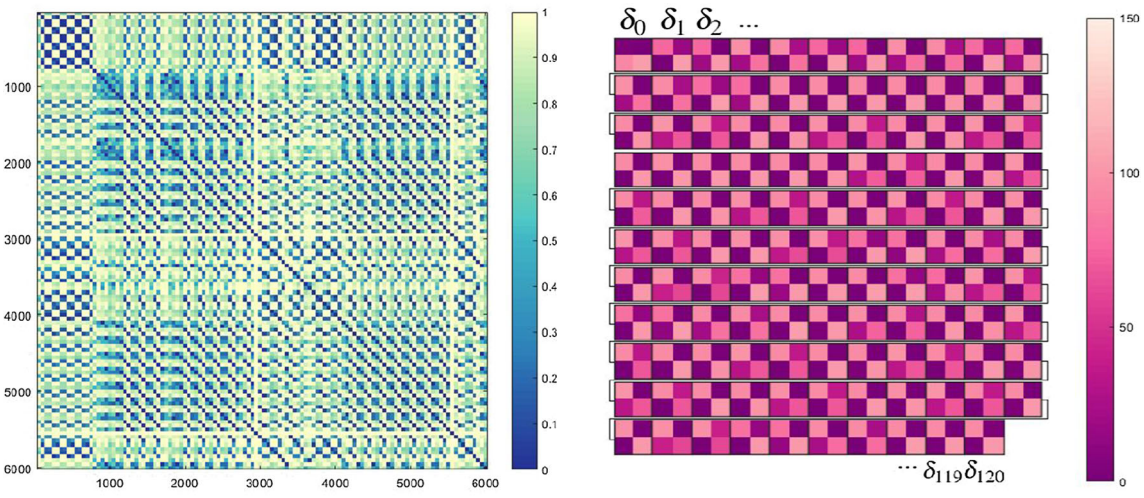
(a) Linearly separable data. **(b)** Class overlapping data.

Fig. 2 Two dimensional synthetic datasets used to exemplify our approach. Different scenarios to show the resulting visual representations for a simple linearly separable case and another much more complex



(a) Distance matrix visualization. **(b)** Transitions matrices visualization.

Fig. 3 Network visualization for scenario *A*. The network quickly converges to an optimal result, correctly classifying all training instances. In the dissimilarity matrix, it is possible to see that after a burst of transition dissimilarities, around the 80th iteration, no more transitions happen (**a**). The transition matrices confirm that indeed no more inter-class transitions are detected (**b**)



(a) Distance matrix visualization. **(b)** Transitions matrices visualization.

Fig. 4 Network visualization for scenario *B*. According to the distance matrix, the network convergence is stuck into a loop, in which the dark diagonal lines highlight that transition sequences repeat themselves (**a**). This is expected, since the data is not linearly separable. Repeating patterns are noticed in the transitions matrices as well (**b**)

3.2 Recurrence quantification analysis

Beyond visual inspections, distance and differentiation matrices can be assessed quantitatively to give insights into the training process. Here, we use *recurrence quantification analysis (RQA)* to aid in this process. In the context of dynamical systems, RQA provides measures to evaluate time-series patterns taking as input *recurrence plots (RP)* and *cross-recurrence plots (CRPs)* (Webber and Zbilut 1994). RQA supports analyzing recurrent patterns present in a given time series.

Here, we introduce the CRP as a generalization of RPs, which extends this ability of analyzing recurrent patterns while comparing two time series. Consider two time series $\mathbf{s} = s_1, s_2, \ldots, s_n$ and $\mathbf{t} = t_1, t_2, \ldots, t_m$ reconstructed in phase spaces using Takens' embedding theorem (Takens 1981). The CRP computes similarities among all phase space states obtained from $\mathbf{s}$ and $\mathbf{t}$ and produces an $n \times m$-binary matrix, in which

$B_{i,j}$ corresponds to the comparison of state i from $\mathbf{s}$ with j from $\mathbf{t}$. In summary, when $B_{i,j} = 1$, states i and j are similar enough in terms of an open-ball defined in the phase space. When the CRP is applied to compare a time series to itself, we have the RP in which the principal diagonal of B is filled out with 1s.

By considering the distance matrix D as the RP of a time series representing the transitions in an ANN, the RQA can be used to perform quantitative analysis to assist its interpretation. However, the distance matrix D needs to be binarized to represent the similarities and dissimilarities. To perform this binarization, ranges of values $[b_i, b_j]$ are transformed into 1, while the remaining values are set as 0. In our case, those ranges are used to study the network convergence regarding the probability of classification transitions. This probability tends to one when the classification result for a given example continually changes along with training iterations, approaching zero when training stabilizes.

Firstly, in this process, the RQA identifies all nonzero diagonals present in the binarized version of D. Next, the entropy *ent* is computed over the identified diagonals on matrix B (binarized D), as defined in Eq. (3), in which p is the frequency distribution of a diagonal at length v, *lmin* is the minimum value of length for a diagonal, and L is the total number of diagonals (Webber and Zbilut 1994).

$$ent = -\sum_{v=lmin}^{L} (p(v)log(p(v))) \tag{3}$$

Another measure that can be computed from the diagonals is the laminarity *lam*. Laminarity is defined in Eq. (4), where p is the frequency distribution of a diagonal length v, with *lmin* being the minimum value of length obtained and L the total number of diagonals. Laminarity (Webber and Zbilut 1994) counts the number of vertical lines present in the binary matrix according to the provided band. In our case, laminarity can be used to analyze training convergence since each line represents a significant, somewhat unique transition matrix.

$$lam = \frac{\sum_{v=lmin}^{L} vp(v)}{\sum_{v=1}^{L} vp(v)} \tag{4}$$

In our model, we obtain *ent* and *lam* values for different value ranges of the matrices summarizing classification transitions. We then save pairs (b_{ent}, ent) and (b_{lam}, lam), being the lowest ranges considered more valuable since they are related to steady classification models, i.e., when classification transitions stabilize.

Considering all this information, images producing high laminarity at lower ranges offer the best scenario since they indicate that the classification transitions stabilized while at smaller learning steps rather than jumping to some overfitting solution. A lower entropy at a lower range is also associated with the best possible scenario once it indicates the network found a learning setting with a small uncertainty with a slow enough convergence, meaning it finds a good solution.

4 Results

In this section, we present different usage scenarios for two different popular network architectures, showing how to use the *neural network training fingerprint (NNTF)*[1] to understand the training process and estimate parameters.

4.1 Usage scenario with Lenet architecture

In the first scenario, we investigate training parameters and the overall convergence of Lenet-5 (LeCun et al. 1999), a well-known CNN architecture. Lenet-5 is composed of seven layers, leading to a relatively fast training process that allows us to swap and assess parameters quickly.

We use the Cifar-10 (Krizhevsky 2010) dataset to proceed with experiments, which is widely known in the field of object recognition. Moreover, the Cifar-10 dataset is a complex enough dataset to explore the features provided by our approach. Results with the Cifar-10 dataset are typically reported using pre-defined 50, 000-RGB images for training and 10, 000 for testing (Krizhevsky 2010; Srivastava et al. 2014). We follow the same protocol.

[1] Source code available at https://github.com/marthadais/NeuralNetworkTrainingFingerprint.

Given the employed CNN, it is important to introduce some of its parameters to make our analysis clearer to the reader. The CNN update rule is defined in Eq. (5), which is used to adapt filter weights to minimize classification error. Term w_i corresponds to the filter weight at iteration i, w_{i+1} is the weight but at the next iteration, $\mathcal{L}(w)$ is the loss function, λ represents the weight decay, η is the learning rate, γ represents the momentum, $\nabla\mathcal{L}(w)$ is associated with the gradient, and v_i is the weight adaptation step.

$$\begin{aligned} v_{i+1} &= \gamma v_i - \eta\left[\nabla\mathcal{L}(w) + \frac{1}{2}\lambda\|w\|\right] \\ w_{i+1} &= w_i + v_{i+1} \end{aligned} \tag{5}$$

In this context, the learning rate is responsible for the gradient step size, the momentum is used to maintain a percentage of the past gradient information, and the weight decay is a regularization term to make the average of weight values be around zero, helping to bring weights back to the quasi-convex region of the error function. We attempt to assess the impact of those parameters, setting them to $\eta = 0.01$, $\gamma = 0.9$ and $\lambda = 0.005$ as found in the literature (Krizhevsky et al. 2012; LeCun et al. 2015). We collect results for Lenet-5 considering 50 epochs with a step of 1, resulting in a distance matrix with 50×50 transition states. In terms of iterations, we set 128 as batch size, producing approximately 19, 532 iterations and leading to a $k \approx 391$ iterations that represent 1 epoch.

4.1.1 Assessing the weight decay parameter

The first parameter we analyze is weight decay. Experiments were performed with a fixed learning rate of $\eta = 0.01$ and a fixed momentum set to $\gamma = 0.9$. The weight decay was set using one of the values in $\{0, 0.001, 0.005, 0.01, 0.05\}$, allowing enough changes to be pointed out by our approach.

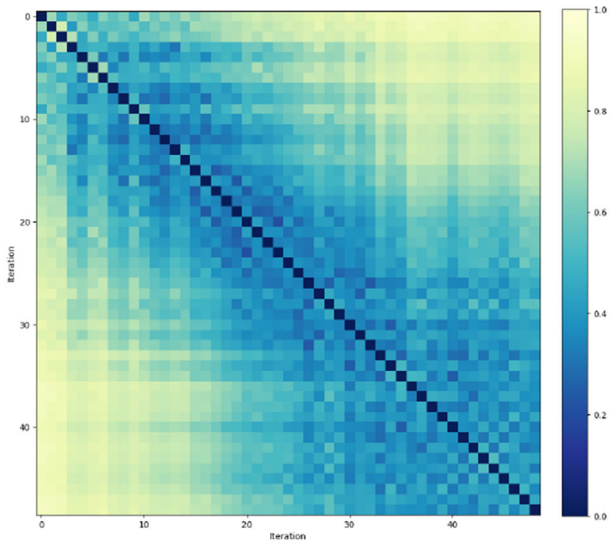
Our first step is to observe the training process setting the weight decay to zero for the Cifar-10 dataset. Figure 5 shows the attained results. In Fig. 5a, the distance matrix D is presented, along with a heatmap of the difference between classification results at each iteration. In Fig. 5b, the transition and accuracy plots to support insights drawn from analyzing the matrix are displayed. Since the weight decay is responsible for slowing down the rate at which the training process converges to avoid overfitting, this example stabilizes relatively quickly, attaining an accuracy value approximately to one (virtually 100%) for the training set itself and the transition count dropping in later iterations.

However, a small number of transitions between classes are still happening, even when the loss function is closed to zero, indicating uncertainty in the model. This is an example of how NNFT can provide more details on the training process, besides the accuracy and loss function, allowing a better analysis of the model optimization. The distance matrix illustrates this process by showing some similarities and repetitions at the beginning, leading to more yellow zones that are entirely dissimilar to each other toward the end. From a pure optimization point of view, highly dissimilar areas are good since they mean the objective function quickly navigates through an area that has never been before and likely reaches a minimum.

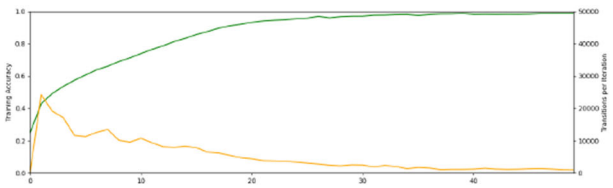
However, this does not necessarily mean that the accuracy results will be as positive for the test set since it may indicate overfitting. Running this network through the test set yields an accuracy value of 0.5672. Figure 6 presents different distance matrices varying λ values. Although composed of highly dissimilar segments, the matrix for the large λ value (0.05) does not show the yellow zones, meaning a sequence that is dissimilar in its components but repeats itself over and over. The addition of cyclic elements to the process gives the network more time to fine-tune its filters, possibly resulting in better generalization.

At first glance, the differences between the distance matrices are not expressive. However, we can subtract one matrix from another to better understand what has changed as the weight decay increase, as presented in Fig. 7. It is worth mentioning that the larger parameter value shows more transitions in the training process than the lower parameter value since more reddish colors are observed close to the main diagonal. The first noticeable thing is that when the weight decay increases too much, the number of transitions seems to stabilize, indicating a slower convergence. This can be further confirmed by counting transitions. Transitions tend to happen in bulk with lower weight parameters values at the beginning of the training process (represented in the top-left of the figure).

RQA results for the matrices generated in this experiment are summarized in Table 1. According to the interpretation discussed in Sect. 3, we can analyze these values in the following manner: the distance matrix generated from the weight decay $\lambda = 0$ has the lowest entropy value, meaning that convergence is attained in an orderly manner. However, laminarity along low bands is also small, meaning that if convergence is attained, there is no evidence of it being achieved through small changes. The matrix generated from



(a) Distance matrix visualization.



(b) Number of transitions (orange) versus classification accuracy (green).

Fig. 5 Visual analysis of Lenet-5 training process using the CIFAR-10 dataset. In **a**, the distance matrix shows that highly dissimilar transitions happen toward the end of the training process, culminating in a completely similar spot. In **b**, the total number of transitions as iterations go by (in orange) and the accuracy of network classifications considering the training set (in green) are shown. While the later iteration transitions are highly dissimilar, the actual amount of transitions is considerably low, meaning that the network is stabilizing. High dissimilarity areas like the ones shown in this image indicate the objective function is navigating through a new region in the solution space

$\lambda = 0.001$ has a high laminarity score. However, it also provides a high entropy. Such high entropy means that the convergence process contains too much noise, indicating a poor parametrization choice for training. The matrix obtained from weight decay value $\lambda = 0.01$ seems to generate good results on both fronts, suggesting that it is a better choice if the goal is to obtain a well-balanced training process. Notably, $\lambda = 0.005$ offers a good laminarity result without sacrificing much in entropy and band location, matching the parameter commonly employed in the literature.

4.1.2 Assessing learning rate parameter

The learning rate was also analyzed considering a fixed momentum set to $\gamma = 0.9$ and weight decay set to $\lambda = 0.005$. Four CNNs were trained considering the following learning rates $\{0.0005, 0.001, 0.005, 0.01\}$. Figure 8 presents the obtained results. The network was not capable of converging for $\eta = 0.01$ and $\eta = 0.005$. The transition counts for such matrices confirm this claim, showing a repeating pattern and small accuracy values even for the training set. While we conclude that we have a bad parameter choice by simply analyzing accuracy plots, network training information points out a locking state, such as a loop, which is relevant to studying and exploring the learning rate mechanism itself. Since those steps are too large for this scenario, the weights heavily influence the gradient adaptation steps and, therefore, get off the minimum region.

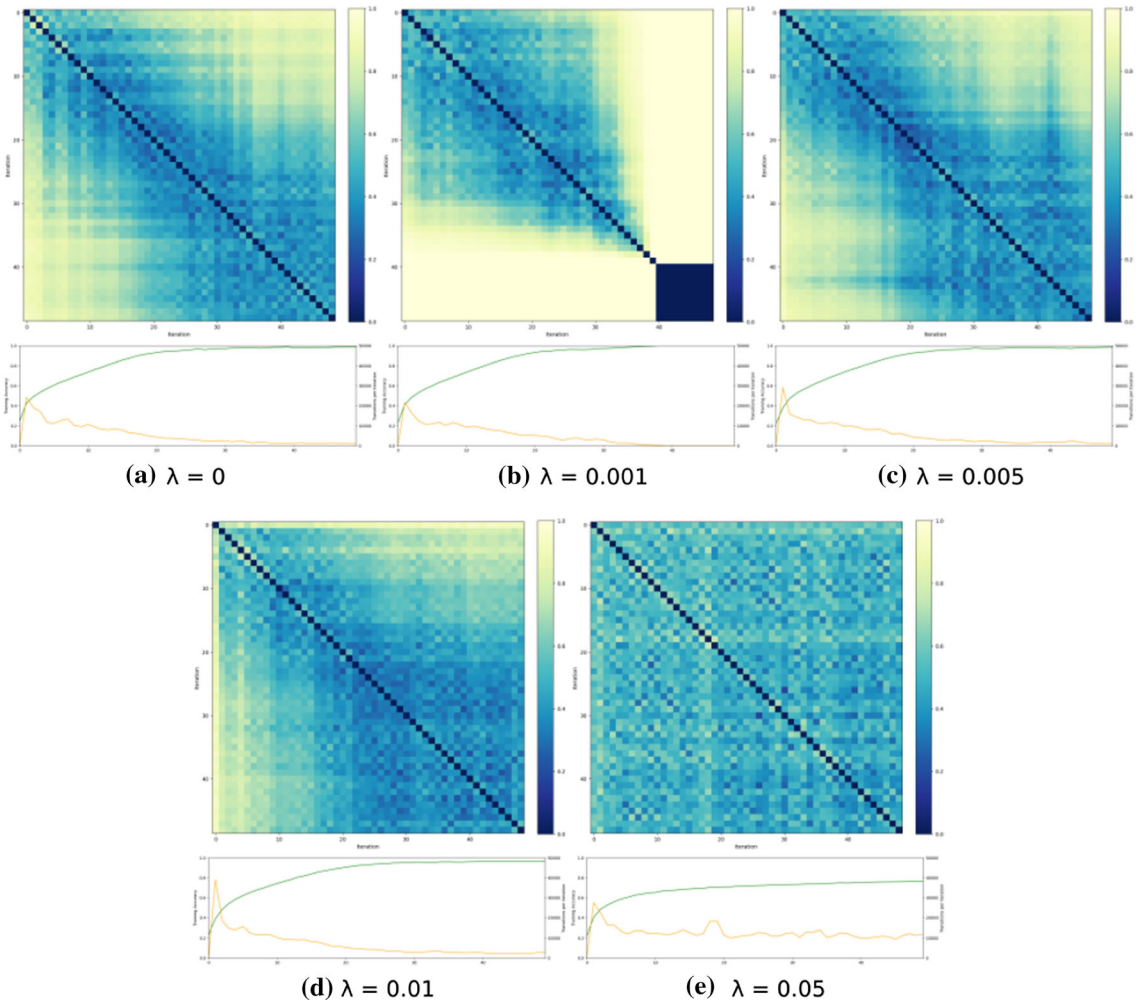


Fig. 6 Distance matrices D for different weight decay regularization values. As weight decay increases, the highly dissimilar areas give way to cyclic patterns that slow down the training process to avoid overfitting. With large weight decay values, only the cyclic pattern remains, hindering convergence

RQA results for the matrices obtained while analyzing learning rates are shown in Table 2. While the slow process shown by the learning rate $\eta = 0.005$ could seem like a promising approach for training at first—assuming the running time is not an issue—the RQA results show that the optimization is not good enough (high entropy) and that low band laminarity is not as high as expected. This looks to be related to bumps present in the error function so that the optimization gets stuck at a local minimum at certain points. Learning rates closer to $\eta = 0.001$ seem to be a better choice, which is also what we found in the literature.

4.1.3 Assessing momentum parameter

The momentum was the third parameter analyzed. Learning rate and weight decay were set as $\eta = 0.01$ and $\lambda = 0.005$, respectively. The momentum values analyzed were $\{0.5, 0.7, 0.9, 0.95\}$, leading to 4 CNN executions. In this case, a large momentum usually brings more past gradient information to smooth the error function and avoid local minima. However, this also leads to slower convergence.

The results provided by the visualization in Fig. 9 confirm that lower momentum values generate images that exhibit a more texture-like appearance, shaped like grids. Dark diagonals are also more pronounced, indicating that classification goes through more repeating patterns as the network converges. Lower momentum values lead to less predictable matrix behaviors, with sharp and bright lines that appear less periodically.

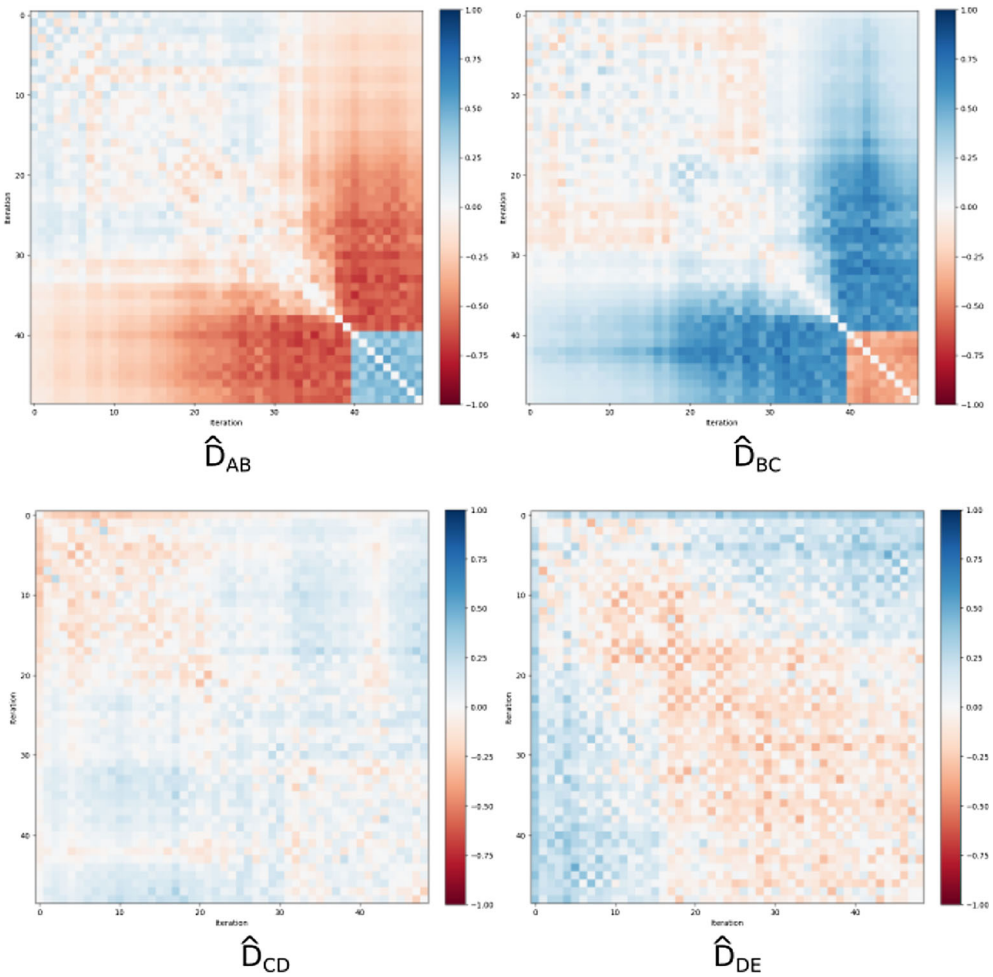


Fig. 7 Differentiation matrices $\hat{D}_{AB}$, $\hat{D}_{BC}$, $\hat{D}_{CD}$ and $\hat{D}_{DE}$ between the distance matrices in Fig. 6

Table 1 RQA results for Weight Decay parameter experiments

wd	b_{lam}	*lam*	b_{ent}	*ent*	*Acc*
0.05	0.30–0.35	0.0761	0.00–0.05	0.0794	0.6733
0.01	0.25–0.30	0.2453	0.00–0.05	0.0794	0.6344
0.005	**0.25–0.30**	**0.3125**	**0.00–0.05**	**0.0794**	**0.6098**
0.001	0.00–0.05	0.6694	0.00–0.05	0.52094	0.5882
0	0.25–0.30	0.1429	0.00–0.05	0.0794	0.5672

From left to right, the columns describe the weight decay value used to generate the distance matrix analyzed, the band in which the laminarity value b_{lam} was obtained, the *lam* value itself, the band in which entropy b_{ent} was obtained, the entropy value *ent*, and the accuracy in the test set *Acc*
They represent the most adequated values according to the analysis

One interesting element of this view is that the horizontal and vertical lines are clearly visible in these examples, indicating high transition values, showing that the network abruptly changes the classification. Those aspects are more easily perceived on the matrices corresponding to small momentum values, resulting in more aggressive optimization steps.

RQA results for the momentum parameter are shown on Table 3. Parameter $\gamma = 0.9$ seems to be the best choice for a balanced approach since it offers a reasonable combination of laminarity and entropy values. It is worth mentioning that this value is the same usually employed in the literature.

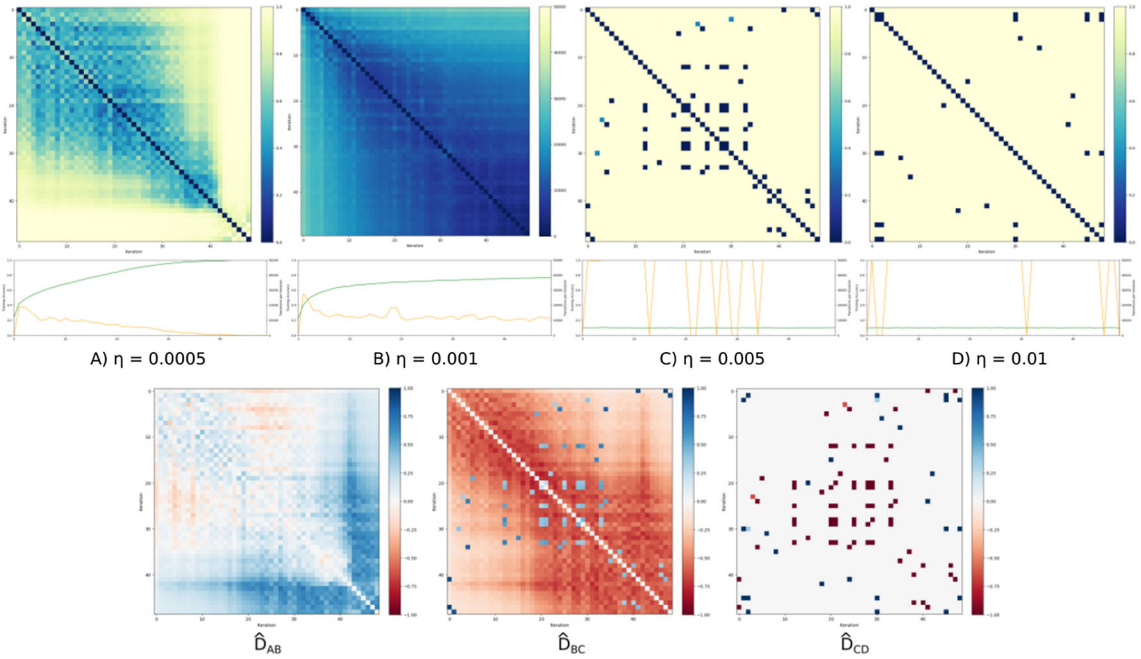


Fig. 8 Distance matrices D_A, D_B, and D_C for different learning rate parameters in the first row and differentiation matrices $\hat{D}_{AB}$ and $\hat{D}_{BC}$ between them in the second row. The effect of reducing learning rates can easily be seen as the concentration of brighter areas shifts from the first rows/columns of the matrix in $\eta = 0.01$ to covering most of its length in $\eta = 0.001$. The differentiation matrices show exactly this trend between matrices, as well as the absence of certain sudden steps as can be seen by the blue lines in $\hat{D}_{AB}$

Table 2 RQA results in the learning rate parameter

lr	b_{lam}	*lam*	b_{ent}	*ent*	*Acc*
0.01	0.00–0.05	0.1266	0.00–0.05	0.0794	0.1000
0.005	0.00–0.05	0.2353	0.00–0.05	0.2955	0.1000
0.001	**0.20–0.25**	**0.3125**	**0.00–0.05**	**0.0794**	**0.6098**
0.0005	0.3–0.35	0.1500	0.00–0.05	0.0794	0.6299

From left to right, the columns describe the learning rates used to generate the distance matrices, the band in which the laminarity value b_{lam} was obtained, the *lam* value itself, the band in which entropy b_{ent} was obtained, the entropy value *ent*, and the accuracy in the test set *Acc*
They represent the most adequated values according to the analysis

4.2 Usage scenario with VGG16 architecture

In this scenario, we evaluate training parameters for the VGG16 (Simonyan and Zisserman 2014), a well-known CNN architecture, using NNTF. VGG16 is composed of 16 layers and presents positive results for large datasets such as Imagenet (Krizhevsky et al. 2012). We once again used the Cifar-10 data set and collected results for every epoch in 50 total epochs for this experiment.

4.2.1 Assessing the weight decay parameter

As performed in the LeNet architecture evaluation, the first parameter analyzed for VGG16 is weight decay. Similarly, we use a fixed learning rate of $\eta = 0.001$ and a fixed momentum of $\gamma = 0.9$. Five CNNs were trained, each one using one of the values in $\{0.05, 0.01, 0.005, 0.001, 0\}$ as weight decay. RQA results for the matrices generated in this experiment are shown in Table 4. In this scenario, the most suitable values for laminarity and entropy were generated using the weight decay $\lambda = 0$, which is not similar to the one obtained for LeNet architecture and the typical value used in literature. However, the standard value $\lambda = 0.005$ also presented a good balance between laminarity and entropy.

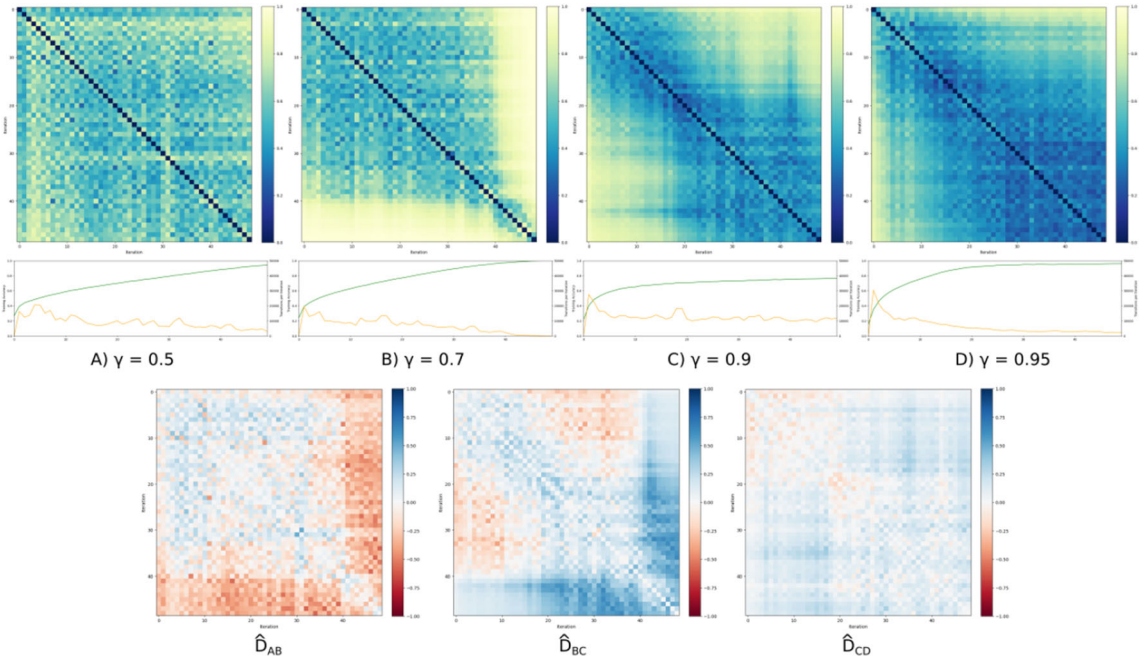


Fig. 9 Distance matrices D for different momentum values on the top and a differentiation matrix $\hat{D}$ between lower and greater values parameters the bottom. Lower momentum values result in almost texture-like images, as there are many bright areas but at the same time many diagonals. In later areas, the diagonals and grid-like bright squares do not seem as pronounced, suggesting less periodic convergence

Table 3 RQA results for the momentum parameter

mnt	b_{lam}	*lam*	b_{ent}	*ent*	*Acc*
0.5	0.40–0.45	0.2013	0.00–0.05	0.0794	0.6114
0.7	0.35–0.40	0.0625	0.00–0.05	0.0794	0.6403
0.9	**0.25–0.30**	**0.3125**	**0.00–0.05**	**0.0794**	**0.6098**
0.95	0.20–0.25	0.2353	0.00–0.05	0.0794	0.6167

From left to right, the columns describe the value of γ used to generate the distance matrix, the band in which the laminarity value b_{lam} was obtained, the *lam* value itself, the band in which entropy b_{ent} was obtained, the entropy value *ent*, and the accuracy in the test set *Acc*
They represent the most adequated values according to the analysis

Table 4 RQA results for weight decay parameter considered the VGG16 architecture

wd	b_{lam}	*lam*	b_{ent}	*ent*	*Acc*
0.1	0.00–0.05	0.5200	0.00–0.05	0.3898	0.1000
0.05	0.00–0.05	0.4647	0.00–0.05	0.4540	0.1000
0.01	0.25–0.30	0.0833	0.00–0.05	0.0794	0.7274
0.005	0.30–0.35	0.1111	0.00–0.05	0.0794	0.7438
0.001	0.00–0.05	0.3623	0.00–0.05	0.3772	0.7883
0	**0.30–0.35**	**0.2500**	**0.00–0.05**	**0.0794**	**0.7803**

From left to right, the columns contain, the employed weight decay value used to generate the distance matrices, the band in which the laminarity value b_{lam} was obtained, the *lam* value itself, the band in which entropy b_{ent} was obtained, the entropy value *ent*, and the accuracy in the test set *Acc*
They represent the most adequated values according to the analysis

Figure 10 shows fingerprints for the conducted training, while Fig. 11 presents the differences between these fingerprints. As the weight decay parameter increases, concentrated areas disappear, and the heatmaps become more homogeneous but noisier as well. Lines indicating rifts in the similarity between epochs also become absent, indicating that larger weight decay values inhibit leaps in optimization. In the last image, the

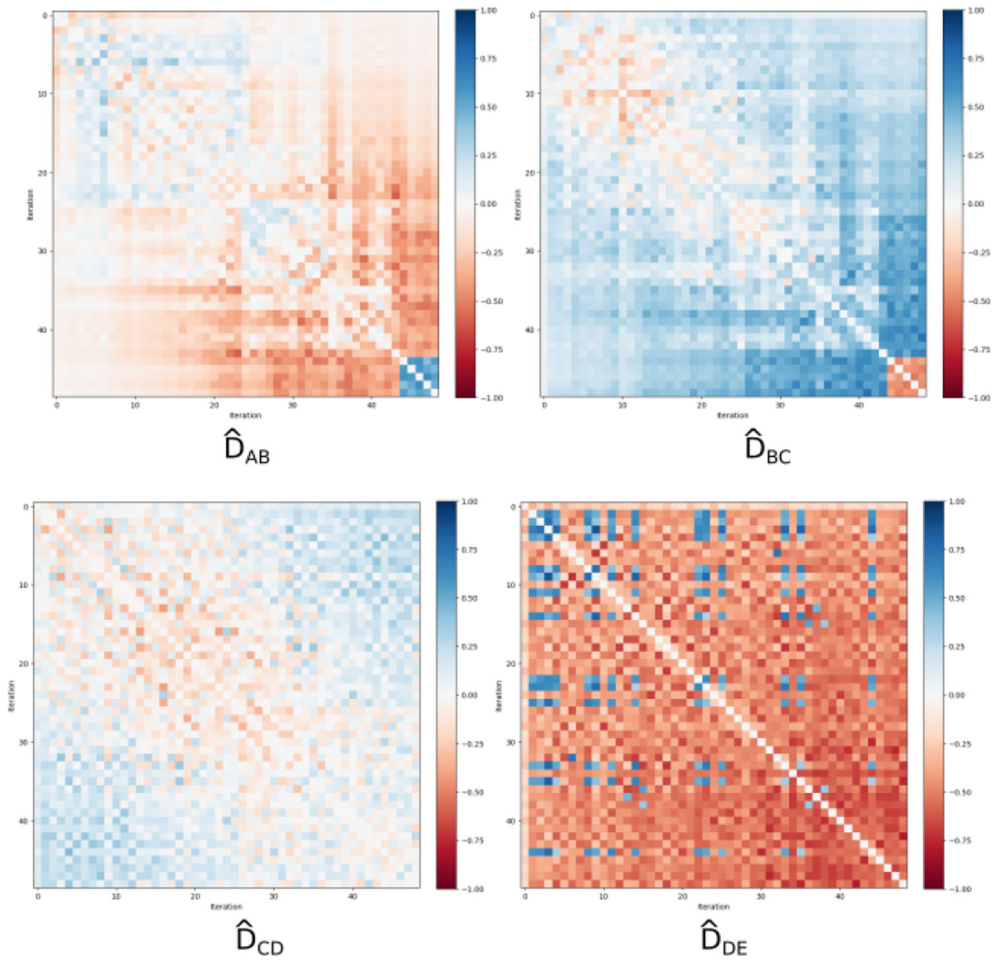


Fig. 10 Fingerprints for VGG16 training with varying weight decay parameters λ. As λ increases, concentrated areas disappear and the heatmaps gain noise. Lines indicating rifts in similarity between epochs also become less frequent, indicating that larger weight decay values inhibit leaps in optimization. In the last image, the value $\lambda = 0.05$ inhibits convergence, getting the model stuck in a state where any attempt of optimization would cause it to shift classification for all data instances

value $\lambda = 0.05$ inhibits convergence, getting the model stuck in a state where any optimization attempt would cause it to shift classification for all data instances.

4.2.2 Assessing learning rate parameter

The learning rate was analyzed considering a fixed momentum set to $\gamma = 0.9$ and weight decay set to $\lambda = 0.005$. Four CNNs were trained considering the following learning rates $\{0.0005, 0.001, 0.005, 0.01\}$. RQA results for the matrices generated in this experiment are shown in Table 5. In this scenario, the most suitable values for laminarity and entropy were generated using the learning rate $\eta = 0.001$, which is similar to the one obtained for the LeNet architecture. As VGG16 is a more complex network, we believe that the produced error function is slightly wide and smooth, leading to $\eta = 0.005$, a suitable value for the learning rate.

Figure 12 shows the different training behaviors for each η value. Despite producing similar results, it is possible to notice that $\eta = 0.005$ produces an image with several horizontal and vertical yellow lines when compared to $\eta = 0.001$, indicating a less steady convergence that is not visible by just comparing accuracy curves. At $\eta = 0.01$, this increase becomes excessive, as certain iterations generate huge shifts in the classification that the model has difficulties recovering from.

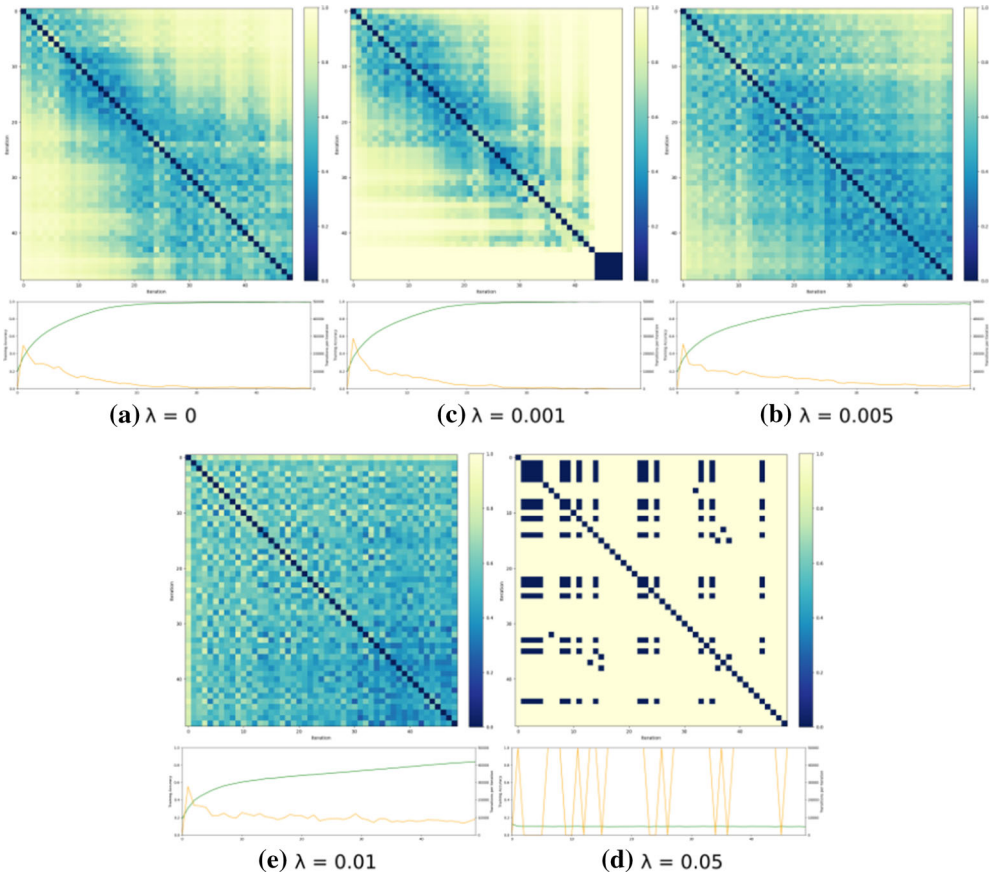


Fig. 11 Differentiation matrices $\hat{D}_{AB}$, $\hat{D}_{BC}$, $\hat{D}_{CD}$ and $\hat{D}_{DE}$ between the distance matrices in Fig. 6

Table 5 RQA results varying the learning rate parameter considering VGG16

lr	b_{lam}	*lam*	b_{ent}	*ent*	*Acc*
0.01	0.30–0.40	0.2778	0.00–0.05	0.0794	0.3367
0.005	0.25–0.30	0.1477	0.00–0.05	0.0794	0.8004
0.001	**0.25–0.30**	**0.2000**	**0.00–0.05**	**0.0794**	**0.7438**
0.0005	0.35–0.40	0.1190	0.00–0.05	0.0794	0.7153

From left to right, the columns describe the learning rates used to generate the distance matrices, the band in which the laminarity value b_{lam} was obtained, the *lam* value itself, the band in which entropy b_{ent} was obtained, the entropy value *ent*, and the accuracy in the test set *Acc*
They represent the most adequated values according to the analysis

4.2.3 Assessing momentum parameter

Momentum was also analyzed for the VGG16, fixing the learning rate and weight decay to $\eta = 0.01$ and $\lambda = 0.005$, respectively. The momentum values analyzed were $\{0.5, 0.7, 0.9, 0.95\}$, resulting in 4 CNN executions. RQA results for the matrices generated in this experiment are shown in Table 6. In this scenario, the most suitable values for laminarity and entropy were generated using the momentum $\gamma = 0.9$, similar to the one obtained for LeNet architecture and found in the literature.

Figure 13 shows training fingerprints for different γ values. Lower values generate more vibrant colored heatmaps (highly different transitions), indicating that transitions at every iteration follow very particular patterns that do not repeat often. As γ increases, the central diagonal of the heatmaps starts to become darker since the iterations immediately close to one another move in a more similar direction in the solution space and generate more similar transitions.

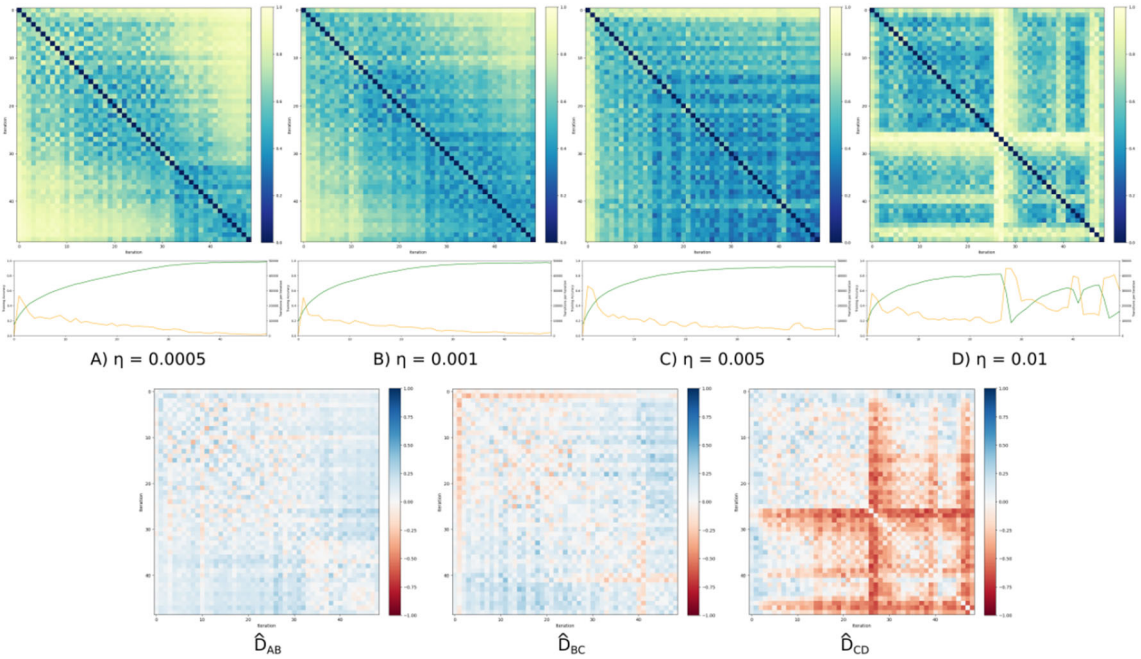


Fig. 12 Fingerprints for different learning rates η for VGG16 training with CIFAR10 dataset. Despite increasing step size, $\eta = 0.005$ shows a smoother heatmap, indicating steady convergence. The image on the right, depicting $\eta = 0.01$, shows that excessively large steps may result in erratic movement through solution space

Table 6 RQA results for the momentum parameter considering VGG16

mnt	b_{lam}	*lam*	b_{ent}	*ent*	*Acc*
0.5	0.35–0.40	0.0714	0.00–0.05	0.0794	0.6896
0.7	0.45–0.50	0.2093	0.00–0.05	0.0794	0.7103
0.9	**0.25–0.30**	**0.2000**	**0.00–0.05**	**0.0794**	**0.7438**
0.95	0.20–0.25	0.0833	0.00–0.05	0.0794	0.7706

From left to right, the columns describe the value of γ used to generate the distance matrices, the band in which the laminarity value b_{lam} was obtained, the *lam* value itself, the band in which entropy b_{ent} was obtained, the entropy value *ent*, and the accuracy in the test set *Acc*
They represent the most adequated values according to the analysis

In summary, while the proposed visual approach can support the optimization analysis, showing the impact of the learning parameters in the convergence, the RQA measures assist in understanding model uncertainty (entropy) and how stable and robust the convergence is (laminarity). Therefore, their combination shows to be a powerful tool for understanding the convergence behavior, showing abrupt changes in the classification, repetitive patterns, uncertainty, and robustness.

5 Discussion and limitations

The *neural network training fingerprint (NNTF)* offers information that can support the configuration of training parameters and the analysis of network and data complexity. It allows assessing the impact of parameter changes and how they relate to network convergence, which is not supported by the usual loss/accuracy values. NNTF describes transitions in the classification of instances during training, which shows how the decision changes over time as the model is adapted using training data. It is worth mentioning that the decision boundaries changes as the model converges toward the loss function derivative. Thus, the transitions will stop occurring if the model achieves a minimal region in the loss function, indicating an overfitting or a local minimum that does not ensure the learning.

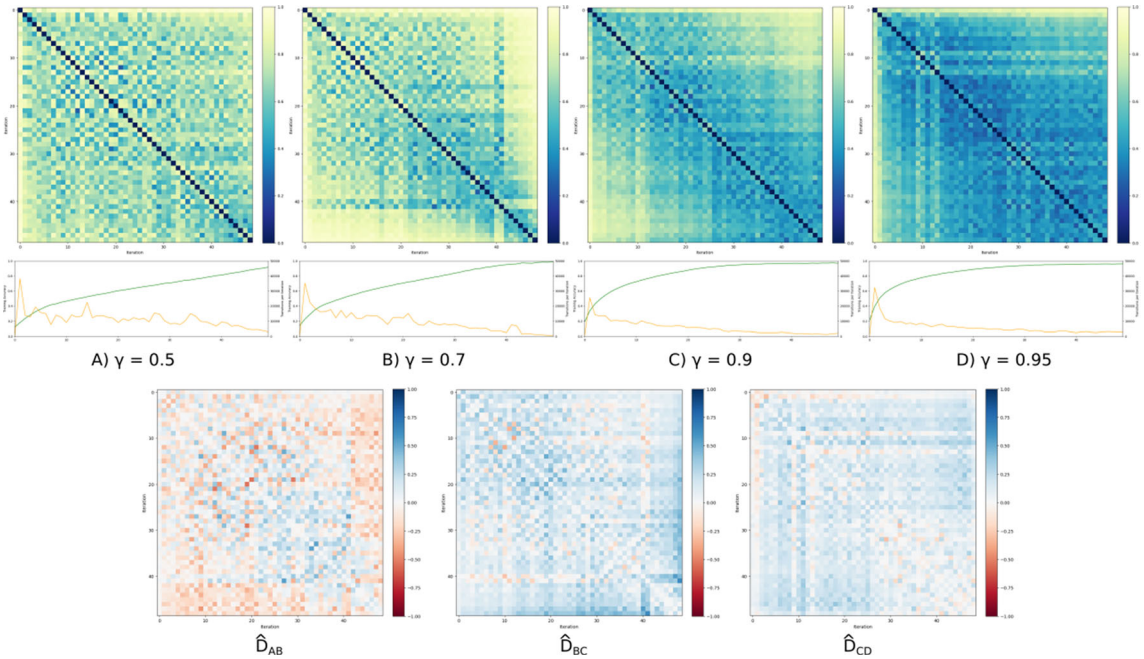


Fig. 13 Training fingerprints for different γ values in VGG16 using Cifar-10 data. Images on the left side are more vibrant (highly different transitions), indicating that transitions at every iteration follow very particular patterns that do not repeat often (smaller steps in different directions in solution space). As γ increases, the central diagonal starts to become darker, as solution space movement becomes smoother and iterations immediately close to one another generate more similar transitions

Moreover, the proposed approach is a non-intrusive visualization framework to support the interpretation of artificial neural networks. As it only relies on model outputs after training iterations, it can be easily extended to any classifier based on an iterative learning process. The visualization can indicate when the optimization is undergoing oscillation caused by bad parametrization or when the model is not complex enough to deal with the input dataset. Information about the convergence is also provided, indicating when it is too fast, too slow, or not happening at all. Loops and repeating patterns are visible, meaning that the network is oscillating toward the gradient.

Considering the RQA measure, it is possible to conduct a quantitative evaluation of the produced visualizations, leading to a better analysis of the optimization process. RQA is a well-known metric employed on dynamical systems that measure the similarity of RPs and CRPs. The proposed visualization is similar to the matrices produced by RP and CRP methods, making it possible to apply the same measures. The results obtained with RQA show that our approach can support the understanding of ANNs training and assist in the analysis of the parameters, in which the entropy represents the uncertainty of the training process and the laminarity of how stable and robust is the convergence.

In summary, NNTF can be applied to any classifier to assist in evaluating the optimization process, indicating cyclic and repetitive patterns, abrupt changes in classification, uncertainty, and robustness. NNTF supports understanding the training parameters' impact, helping users get insights on selecting adequate values. However, despite the promising results, our technique presents two main limitations: (i) the visual tool cannot provide information when there are no transitions, and (ii) the proposed solution to compare iterations with no transitions can be visually confusing. In these cases, other tools are preferred.

6 Conclusions

A visual approach, called *neural network training fingerprint (NNTF)*, was proposed to better understand the training process of artificial neural networks used to classify complex and large datasets. NNTF does not require information about the architecture or the model's internal features, being characterized as a non-intrusive method. The only information needed to perform the network analysis is the classification outputs

for each instance during the training, which allows the observation of many supervised models and their learning process.

We presented and discussed the application of NNTF for neural network classification. As future work, we plan to use the repeating sequence patterns identified during the training process to investigate their impact (if there is any) in the context of DNN optimization, including an analysis on validation and test sets. Also, we plan to adapt the NNTF to understand the training process of non-classification-focused tasks, such as regression or natural language processing, which allows the analysis for different datasets domains, such as signals and text. The challenge is to derive a measure to replace the class changing to represent the network transitions. Another aspect to check in the future is the impact of uncertainty in true labels and multi-label classification tasks on the training process of DNNs. Finally, we plan to investigate the use of RQA in the analysis of other visual representations to quantify changes, for instance, in time-varying dimensionality reduction visualizations (Alencar et al. 2012; de Araújo Tiburtino Neves et al. 2021).

Acknowledgements We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M, et al. (2016) Tensorflow: a system for large-scale machine learning. In: OSDI, vol 16, pp 265–283

Alencar AB, Börner K, Paulovich FV, de Oliveira MCF (2012) Time-aware visualization of document collections. In: Proceedings of the 27th annual ACM symposium on applied computing, SAC '12. Association for Computing Machinery, New York, pp 997–1004. https://doi.org/10.1145/2245276.2245469

Babiker HKB, Goebel R (2017) An introduction to deep visual explanation. ArXiv preprint arXiv:1711.09482

Bach S, Binder A, Montavon G, Klauschen F, Müller KR, Samek W (2015) On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. PLoS One 10(7):e0130140

Borg I, Groenen P (2003) Modern multidimensional scaling: theory and applications. J Educ Meas 40(3):277–280

Cantareira GD, Etemad E, Paulovich FV (2020) Exploring neural network hidden layer activity using vector fields. Information 11(9):426

Chen C, Yuan J, Lu Y, Liu Y, Su H, Yuan S, Liu S (2020) Oodanalyzer: interactive analysis of out-of-distribution samples. IEEE Trans Vis Comput Graph 27(7):3335–3349

de Araújo Tiburtino Neves TT, Martins RM, Coimbra DB, Kucher K, Kerren A, Paulovich FV (2021) Fast and reliable incremental dimensionality reduction for streaming data. Comput Graph. https://doi.org/10.1016/j.cag.2021.08.009

Erhan D, Bengio Y, Courville A, Vincent P (2009) Visualizing higher-layer features of a deep network. Univ Montr 1341(3):1

Goodfellow I, Bengio Y, Courville A, Bengio Y (2016) Deep learning. MIT Press, Cambridge

Gschwandtner T, Erhart O (2018) Know your enemy: identifying quality problems of time series data. In: 2018 IEEE Pacific visualization symposium (PacificVis). IEEE, pp 205–214

Gu D, Li Y, Jiang F, Wen Z, Liu S, Shi W, Lu G, Zhou C (2020) Vinet: a visually interpretable image diagnosis network. IEEE Trans Multimedia 22(7):1720–1729

Hohman F, Kahng M, Pienta R, Chau DH (2018) Visual analytics in deep learning: an interrogative survey for the next frontiers. IEEE Trans Vis Comput Graph 25(8):2674–2693

Hohman F, Wongsuphasawat K, Kery MB, Patel K (2020) Understanding and visualizing data iteration in machine learning. In: Proceedings of the 2020 CHI conference on human factors in computing systems, pp 1–13

Kahng M, Andrews PY, Kalro A, Chau DHP (2018) Activis: visual exploration of industry-scale deep neural network models. IEEE Trans Vis Comput Graph 24(1):88–97

Krizhevsky A (2010) Convolutional deep belief networks on cifar-10. Unpublished manuscript 40(7):1–9

Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems, pp 1097–1105

Krogh A, Hertz JA (1992) A simple weight decay can improve generalization. In: Advances in neural information processing systems, pp 950–957

LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436

LeCun Y, Haffner P, Bottou L, Bengio Y (1999) Object recognition with gradient-based learning. In: Shape, contour and grouping in computer vision. Springer, Berlin, Heidelberg, pp 319–345. https://doi.org/10.1007/3-540-46805-6_19

Liu D, Cui W, Jin K, Guo Y, Qu H (2018) Deeptracker: visualizing the training process of convolutional neural networks. ACM Trans Intell Syst Technol (TIST) 10(1):1–25

Liu M, Shi J, Cao K, Zhu J, Liu S (2017) Analyzing the training processes of deep generative models. IEEE Trans Vis Comput Graph 24(1):77–87

Liu S, Wang X, Liu M, Zhu J (2017) Towards better analysis of machine learning models: a visual analytics perspective. Vis Inf 1(1):48–56

Ma Y, Xie T, Li J, Maciejewski R (2019) Explaining vulnerabilities to adversarial machine learning through visual analytics. IEEE Trans Vis Comput Graph 26(1):1075–1085

Mahendran A, Vedaldi A (2016) Visualizing deep convolutional neural networks using natural pre-images. Int J Comput Vis 120(3):233–255

Pezzotti N, Höllt T, Van Gemert J, Lelieveldt BP, Eisemann E, Vilanova A (2018) Deepeyes: progressive visual analytics for designing deep neural networks. IEEE Trans Vis Comput Graph 24(1):98–108

Rauber PE, Fadel SG, Falcao AX, Telea AC (2017) Visualizing the hidden activity of artificial neural networks. IEEE Trans Vis Comput Graph 23(1):101–110
Samek W, Binder A, Montavon G, Lapuschkin S, Müller KR (2017) Evaluating the visualization of what a deep neural network has learned. IEEE Trans Neural Netw Learn Syst 28(11):2660–2673
Scherer D, Müller A, Behnke S (2010) Evaluation of pooling operations in convolutional architectures for object recognition. In: International conference on artificial neural networks, pp 92–101. Springer
Shang W, Sohn K, Almeida D, Lee H (2016) Understanding and improving convolutional neural networks via concatenated rectified linear units. In: International conference on machine learning, pp 2217–2225
Simonyan K, Vedaldi A, Zisserman A (2014) Deep inside convolutional networks: visualising image classification models and saliency maps. In: 2nd international conference on learning representations (ICLR), workshop track proceedings
Simonyan K, Zisserman A (2014) Very deep convolutional networks for large-scale image recognition. ArXiv preprint arXiv:1409.1556
Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 15(1):1929–1958
Takens F et al (1981) Detecting strange attractors in turbulence. Lect Notes Math 898(1):366–381
Van Der Maaten L (2014) Accelerating t-SNE using tree-based algorithms. J Mach Learn Res 15(1):3221–3245
Wang J, Gou L, Shen HW, Yang H (2018) Dqnviz: a visual analytics approach to understand deep q-networks. IEEE Trans Vis Comput Graph 25(1):288–298
Webber CL Jr, Zbilut JP (1994) Dynamical assessment of physiological systems and states using recurrence plot strategies. J Appl Physiol 76(2):965–973
Yosinski J, Clune J, Nguyen A, Fuchs T, Lipson H (2015) Understanding neural networks through deep visualization. arXiv preprint arXiv:1506.06579
Zeiler MD, Fergus R (2014) Visualizing and understanding convolutional networks. In: European conference on computer vision, pp 818–833. Springer
Zintgraf LM, Cohen TS, Adel T, Welling M (2017) Visualizing deep neural network decisions: prediction difference analysis. ArXiv preprint arXiv:1702.04595