Compact visualization of DNN classification performances for interpretation and improvement

Adrien Halnaut, Romain Giot, Romain Bourqui, David Auber

CHAPTER 3

Compact Visualization of DNN Classification Performances for Interpretation and Improvement

Chapter Subtitle

Adrien Halnaut,*, Romain Giot*, Romain Bourqui* and David Auber*
* Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, F-33400 Talence, France

Contents

Abstract

In the research field of automatic classification tasks, deep neural networks (or DNN) are frequently used for their efficiency and adaptive nature. State-of-the-art architectures and pre-trained networks exist and can be converted and fine tuned to handle other classification tasks. However, because of their training phase requiring many computations and adjustments on the many parameters of the models, they suffer from a black-box effect that

makes difficult to interpret their inner workings. As a result, the understanding of their successes and failures is as much as difficult. In this chapter, we make use of information visualization and present a method to help in the interpretability of these trained networks. By depicting both their architecture and way to process the different classes of a testing input data, it is possible to visually detect where the DNN starts to discriminate the classes at the layer level. Similarly, it is also possible to detect degradation in the classification process, mainly because of the usage of an over-sized network for a simplier classification task. We implemented and validated the method using several well-known datasets and networks. Results show a progressive classification process and is extended to an improvement on an over-sized network to solve a simpler task.

A. Introduction

Deep-learning [LBH15] based approaches are used in various contexts and dominate most historical methods, especially for classification problems. Even when training datasets are not large enough to train Deep Neural Networks (DNN), it is possible to use transfer learning with a pre-trained DNN by fine-tuning [GYA17] it, or by extracting features and feeding them to a conventional classifier [Tan13]. DNN can be represented by graph linking computational blocks between each other: each block processes the output of one or several previous one. Theses blocks are applying simple function defined during the model construction and its parameters are defined using weights. Such weights are data-dependent and computed during the training phase of the model.

The black-box feeling is one of the largest issues. Indeed, even if each block is individually well understood mathematically, its behavior depends mainly on the training data (*i.e.*, their impact on the learned weights). As a consequence, one can hardly know what processing these blocks are doing and why. However, it is well admitted that the model's first layers extract low-level features, while the latest ones extract high-level features specific to the application problem [EBCV09]. Two non-exclusive strategies can help to open this black box. (i) *Explainable deep-learning* where the architecture of the DNN emphasizes its explainability [ZNWZ18], even if it could negatively impact its performance, and (ii) *Interpretable deep-learning* where additional processes extract information by computing a more explainable model [ZYMW19], computing some saliency information [BML+16, AFMB+20] or using information visualization techniques [HKPC18]. Understanding, even partially, the classification process of a trained network helps in both understanding which data classes are easier to classify than other and where the network can be further improved for its task, in a different way than what layer pruning techniques do [HZS17].

This chapter is an extension of the paper [HGBA21], presenting a new method for

interpretable deep-learning based on information visualization techniques. The task to solve corresponds to the analysis of the data classification over layers. It aims at analyzing how all input samples of a dataset are globally treated by any part of the network. In opposite to most papers of the literature on attribution-based methods, the focus is not for a specific sample but for a full testing dataset. The method *allows focusing on* successive layers that better (or worst) discriminate the samples. It also displays the complete network and the data behavior for each of its computational blocks.

It originality relies on the fact it focuses on both all samples and full architecture and has the advantages of (i) using less screen space than existing methods despite the amount of information to display; (ii) fitting to any network that can be represented as a directed acyclic graph; (iii) using the same encoding for input data, inner blocks and final result.

The remaining of the paper is organized as follows. Section B presents related works in visualization for DNN, space filling curves. Section C describes the proposed method. Section D provides the details of the experimental protocol. Section E discusses the results and provide directions for future work. We finally draw conclusions in Section F.

B. Previous Works

Our proposed method aims at visually interpreting how DNN behave using a space-efficient method. For this reason, this section firstly presents previous works on deep neural networks visualization then focus on dense, pixel oriented visualization methods.

B.1. Visualization for the Interpretation of Deep Neural Networks

Some works focus on *single views* that can be reused in other works or embedded in more complex applications. GradCam [SCD+17] aims at generating a heatmap for a single input to highlight the spatial location that greatly supports the final decision. It is computed thanks to the gradients from the logit of a target class up to the latest convolutional layer. When the input feature is an image, the heatmap can be straightforwardly visualized and understood on it. Other methods rely on different concepts to achieve the same objective such as LRP [BML+16] on the concept of relevance, or other work [AFMB+20] that only uses information collected during the forward pass. Instead of focusing on a single input sample, it is also possible to focus on the complete dataset. Some use Sankey-diagram analogy [HGBA20] to highlight the processing flows. Others project activations obtained at a specific layer in a 2d space to verify how the network sees the data at this specific point [RFFT17]. Such an approach is also common in the literature using T-SNE projection [MH08]; however,

an important drawback remains: the representation is not space efficient and there is no guarantee that overlap does not occur. Our proposed method solves these two issues. Other works create *applications for educational purpose* to visually explain how some specific deep systems perform. For example, Tensorflow playground [SCS+17] focuses on simple DNN, CNN 101 [WTS+20] focuses on CNN, Ganlab [KTC+18a] focuses on Gan system and Adversarial Playground [NQ17] illustrates the concept of adversarial examples. Even if they are visually appealing, these systems can hardly be used for industrial scenarios.

In opposite, several complete tools treat *industrial problems*. Some of them are generalist enough to be used in almost any scenario, such as Activis [KAKC18] (that focuses on the visualization and comparison of activation of a single selected layer), while some others are restricted to some specific networks or evaluation scenarios. CNNVIS [LSL+17] is tailored for CNN and uses a visualization that relies on aggregation of layers (not all layers are depicted), filters (filters that behave similarly are grouped) and data (a subset of the samples are depicted). DQNVIZ [WGSY19] has been designed for Deep Q-Network explanation in the specific context of Atari play.

B.2. Hilbert Curve in Information Visualization

Dense pixel-oriented methods aim at improving both the data-ink ratio **??** and the visualization size by displaying a unit of information on a single pixel while avoiding unused pixels. Keim reviewed various pixel-oriented visualizations [Kei00] and asserts that space-filling curves, such as the Hilbert one [Hil35], are among the bests to project ordered elements in a screen space while preserving the distance of the one-dimensional ordering in the two-dimensional arrangement. Blanchard *et al.* [BHL05] have shown that to display images, reduced to a one pixel representation, on an Hilbert curve produces coherent and identifiable clusters. Auber *et al.* [ANM07] have also shown the interest of such visualization, when complemented by tailored interaction techniques, to explore datacubes of several dozen of millions of elements. Since these previous successes, we have selected the Hilbert curve to project our data in a square; a curve of order n contains 4^n elements [ANM07].

C. Proposed Method

Figure 3.1 describes the proposal with the "Nested blocks and guidelines model"[MSQM15] among various description levels: *domain* (who is concerned by which problem), *abstraction* (which data is used or generated to solve which task), *technique* (which methods are used) and *algorithms* (how these methods are implemented).

Additionally, Figure 3.2 lists the successive steps involved in the method. The requirements of the proposed method are: to be *space efficient* (R1) while displaying information from *all samples* (R2) in *all layers* (R3) of the network to solve the *task*

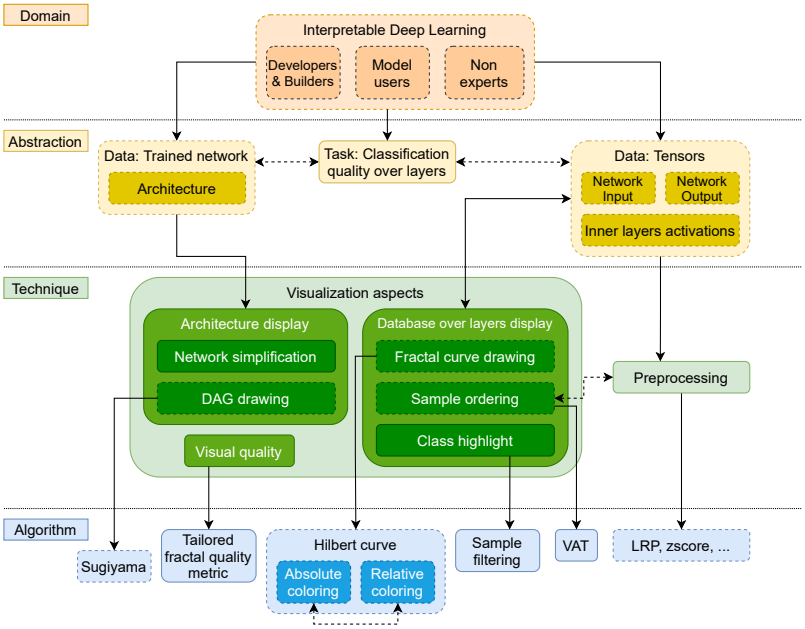


Figure 3.1: Nested blocks and guidelines [MSQM15] representation of the proposed method. Dotted italic blocks corresponds to existing ones; plain straight blocks are defined in the work.

"classification quality analysis over layers" (R4).

C.1. Domain Level

The proposed method fits the needs of *networks designers* and *trainers* that want to verify *how* the data is *grouped* by the various *layers* of their *classification network*. From the *analysis* of these groupings, they could infer *hypotheses* that aim at being verified with other techniques. Such hypotheses are related to input sample properties and network errors. *Non experts* would better understand how DNN work by looking at the representation of simple networks and datasets.

C.2. Abstraction Level

The proposed method considers an already trained DNN N with a compatible test dataset D_{test}. N is a network (*i.e.*, graph) of operations (*i.e.*, nodes) $N = (O, E)$. Its sources $s_\bullet \in O$ are the identity function on data input (*i.e.*, samples) and its sinks $t_\bullet \in O$ are its outputs (*i.e.*, classes probability). N has multiple sources for a multi-modal system, but always a single sink as we are restricted the use case of standard classification. The other nodes $o_\bullet \in O \setminus \{s_\bullet \cup t_\bullet\}$ correspond to any operations (*e.g.*,

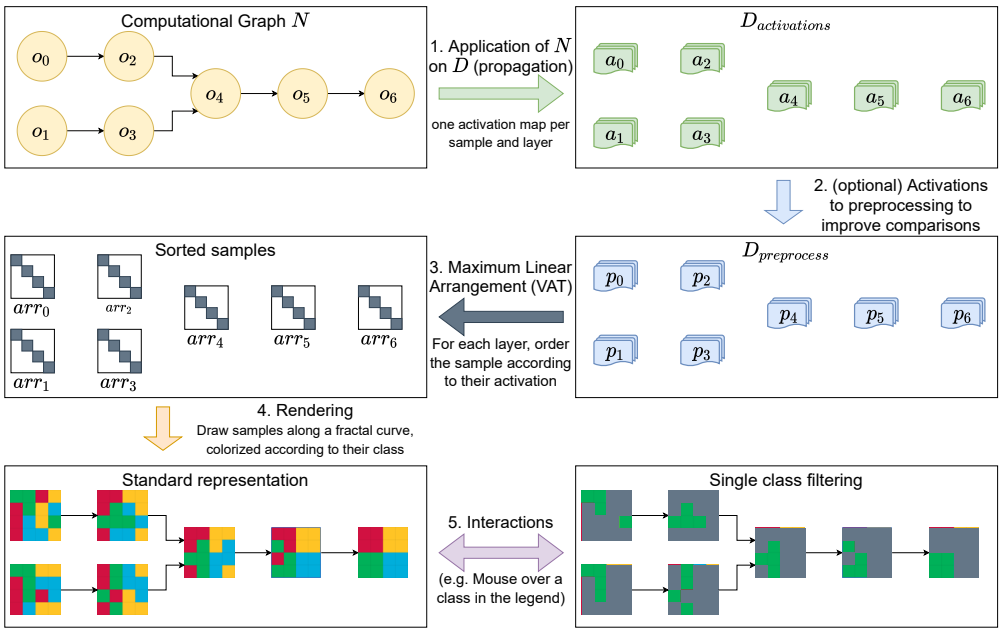


Figure 3.2: Summary of the proposed method. Dataset D is fed to the network N. All activations are collected, eventually preprocessed, and finally ordered at each layer. The ordered samples are drawn along a fractal curve at each operation of the network that is placed on the screen using a graph drawing method.

convolution, pooling, etc) that compose N; operations related to optimization (*e.g.*, dropout) are not included. The edges $E = O \times O$ model the flow of data over the operations of the network (*i.e.*, they link successive layers).

Each sample $d_i \in D_{test}$ is fed into the network and the output (*i.e.*, activations) of each operation o^j is stored in a_i^j; we assume operations are ordered depending on the execution flow. These activations consist of tensors whose order depends on the underlying operation and whose dimensions size depends on the input data of the network. Each operation o^j consumes at least one result $a_i^k | k < j$ computed by a previous operation except for the sources where $a_i^\bullet$ corresponds to the raw data (of the targeted modality in a multimodal scenario). Thus, a sample d_i is represented by a set of activations $A_i = \cup_j \{a_i^j\}$ and the complete dataset D_{test} is represented by an ensemble of sets of activations $D_{activations} = \cup_i \{A_i\}$.

The activations can be optionally preprocessed to fall within compatible domains as their domain is not controlled: $D_{preprocess} = \bigcup_i \bigcup_j \left\{Preprocessed\left(a_i^j\right)\right\}$. This preprocessing method is a parameter of the workflow.

C.3. Technique Level

As the method aims to display (a) the dataset and its *groundtruth* (R2), (b) the *architecture of the network* (R3) and (c) its impact on the *complete dataset* (R4), we propose an encoding relying on both $D_{preprocess}$ and N.

Groundtruth encoding. The groundtruth of the dataset is depicted with a legend where each class is represented by a colored rectangle (sample encoding) followed by a black text (classe name).

Network Encoding. It is straightforward to layout N operations with a graph-drawing algorithm tailored for Directed Acyclic Graphs (networks are always DAG). Such technique is common in the literature [KAKC18, WSW+17] and aims at computing the coordinates of each node (operation) in a plane while emphasizing the order of operations in the computing flow. Each node is depicted by a glyph that represents the whole dataset as viewed by the network at this specific operation. Thus, a specific encoding is used to map the activations $\bigcup_i \{Preprocessed(a_i^j)\}$ of each node o^j in the screen space.

Similarly to Ganlab [KTC+18b], a dotted line is drawn between nodes that represents consecutive operations; the flow of data is revealed by the dots moving in flow direction. Some networks can be very deep with successive layers that do not bring additional information because they consist of data reordering. We allow the user to request the visualization of a simplified network where the corresponding nodes are removed (thus, their successors are linked to their predecessors), as such information brings noise to the representation. No special encoding is used to represent this information shrinking.

Samples encoding. As mentioned, we have chosen a pixel-oriented technique that relies on fractal curves (R1). For a given node o^j, a maximal linear arrangement method is used to order the representation a_i^j of each sample d_i in such a way that samples are positioned closely in the ordering according to a *distance* function. We assume close samples in the output space of o^j corresponds to samples treated similarly by the network (*i.e.*, considered to be similar). Once the samples are ordered, they are projected into a discrete pixel grid using a fractal curve that respects proximity relations. This way, screen space usage can be maximized (1 pixel per sample) and we are assured that close samples are drawn closely on the screen (however close pixels on the screen are not necessarily close in data space). Two visual encodings can represent this curve. The first one, *absolute coloring*, explicitly draws samples of each class with the same color. The second one, *relative coloring*, uses a gray-scale to emphasize label difference between adjacent nodes and identify zones where different labels are present. It can be used *de facto* when the number of classes is too high to be efficiently discernable by a human using regular class colorization. When using the *absolute coloring* scheme, the user can choose to only visualize a specific class

to observe the spread of its samples over the layer. The name of the layer is written above its fractal representation, and a quality metric (presented later in this chapter) is written below it.

C.4. Algorithms Level

The model topology is drawn using the well-known Sugiyama [STT81] algorithm and each node is depicted with a specific fractal-based glyph that represents the ordered samples. The Euclidean distance is used to compare the activations generated for all the samples on the same operation. It reflects the dissimilarity between samples in the Euclidean space; we consider that each neuron activation has the same impact as others in the full network processing. These distances are then compiled into a $n \times n$ sized distance-matrix, n being the number of compared samples. In real use case, neurons have different impact on the final prediction than others. Some pre- or post-processing methods, such as the LRP [BML$^+$16] method as done in [HGBA20], can be applied to the activation maps in order to reflect that behavior. However, we decided not to apply those methods because of the unsure interpretation on model topologies using branches, such as our chimeric *DoubleLeNet5* (section **??**) or the widely used ResNet [HZRS16] which use residual connections. Using ordering methods [BBHR$^+$16], data can be ordered in a queue with similar elements placed next to each other using their dissimilarity matrix. By using the VAT algorithm [dSW18] on the dissimilarity matrices, we found a progressive definition of similarly processed samples, resulting in clusters (or "black squares" as defined in the original paper [dSW18]) reflecting the progressive recognition by the model over the layers we attempt to show. The order computed by this algorithm can then be applied on a 1d-space to display similar data indexes next to each other. Using a fractal curve, we transformed this 1d-space into a 2d-space which is more suitable for data visualization. The fractal curve chosen to map each sample into a pixel-grid is the Hilbert curve [Hil35] because of its ability to place points in a discrete space (this is not the case of Gosper curve [Gar76]) and the absence of "jumps" in the curve (this is not the case of the Z-order curve [Mor66]) which ensures that two consecutive samples are adjacent. The order in which each sample is positioned is following the same order computed by VAT on the previous step. When the number of test samples is lower than the number of pixels available in the curve, we skip half of the missing positions in the beginning of the curve (and thus half of the missing positions at the end of the curve); making a "hole" in the curve but keep the sample centered in the glyph.

In the absolute coloring, each pixel sample is being colored according to its ground-truth class, which is different for each class. In the relative coloring, the colors depend on the number of similar labels for the pixel of interest in its sample ordering. That gives three possible values (0 for an outlier with no neighbors of the same class, 1 for

Table 3.1: Number of layers, parameters and activations per sample for each network.

Network	Layers	Parameters	Memory Usage
LeNet5	10	1 182 006	447.16 ± 19.7 MB
DoubleLeNet5	18	1 646 370	54 570
VGG16	18	33 638 218	308 244

a previous or next label different, and 2 when the three successive samples are of the same class). The absolute colors come from a palette of diverging colors while the relative colors or black (0), gray (1) and white (2). Computing in the ordering space instead of the picture allows to no highlight the visual border inherent to the fractal curve. Placing the cursor on a class in the legend selects this specific class and draws only its samples with the appropriate absolute color.

The machine learning community provides various evaluation metrics (*e.g.*, accuracy or cross-entropy) to evaluate the quality of the network by comparing its output to a ground-truth. By definition, they cannot be applied at each layer, but we still need to provide hints to the user of their efficiency. We have defined a quality metric, based on the local homogeneity of the layer's visual representations, which counts the number of neighbors of a given pixel that are of the same color (*i.e.*, the number of samples that belong to the same class). We normalized it between 0 and 1 to ease its comparison (however, as the normalization does not consider the mandatory borders, 1 is an unreachable value). We assume that to quantify the quality of the visualization is strongly related to the ability of the layer to separate data.

D. Experimental Protocol

Several scenarios, that rely on a *test dataset* and a *trained network*, illustrate the efficiency of the proposed method. In this section, we present datasets and networks that we used in our evaluation.

Datasets. *MNIST* [LeC98] is a standard dataset used in handwritten recognition from 28×28 grayscale images. Even simple networks are able to perform almost perfectly on this 10-classes dataset. We use it to illustrate classification on easy data. *Fashion-MNIST* [XRV17] shares a similar distribution as *MNIST* but is composed of images of clothes instead of digits. Classification performance is usually lower than with *MNIST*. We use it to illustrate classification on averagely difficulty dataset, closer to actual classification problems.

Both datasets are composed of 60 000 samples to train the model and 10 000 samples to evaluate the model.

Networks. In our evaluation, we make use of three networks summarized in Table 3.1:

- *LeNet5* [LBBH98] is a simple and historical CNN that provides good accuracy results on *MNIST*. Its topology is simple enough to get a grasp on how data is being transformed across the model. It is also easy to train with its low parameter count, but that simplicity comes at the cost of lower accuracy results in more complex recognition tasks.
- *DoubleLeNet5* is a chimeric network we have created to illustrate the ability of the system to handle networks with several branches. It consists of two *LeNet5* minus the prediction layer that process in parallel the same input data, but one of the branches input has an image rotation step applied before being processed by the convolutional layers. The two branches are then concatenated before being fed to the prediction layer. The image rotation step is not represented in the DAGs has it has not be implemented in the same way as other layers during the evaluation. Performance wise, this model targets the same kinds of data as *LeNet5*, with a minor performance gain.
- *VGG16* [SZ14] is a deep CNN usually used on complex datasets composed of large color images, with a thousand recognizable classes, such as ImageNet [RDS+15]. Its robustness allows it to reach fairly good accuracy results on target tasks, but comes with a heavy computation cost and cannot be trained in a reasonable amount of time on standard computers. In this paper, the convolutional blocks of the *VGG16* model are already pre-trained with the ImageNet dataset, and are not re-trained when training the prediction layers.

D.1. Couples of Network and Dataset

We have selected meaningful combinations of network and dataset.

- *Easy scenario*: *LeNet5* uses *MNIST* which illustrates a well performing system.
- *Generalization scenario*: *LeNet5* uses *Fashion-MNIST* which illustrates a system with more classification errors.
- *Branch scenario*: *DoubleLeNet5* predicts *Fashion-MNIST* which illustrates a usage case with non-linear network architecture.
- *Simplification scenario*: *VGG16* is processing *MNIST*. It illustrates the use of a complex network to solve a simple task. By applying the visualization pipeline and observing resulting glyph, we propose a straightforward improvement for the model in both accuracy and complexity for the model.

D.2. Implementation and Execution Infrastructure

The TensorFlow framework [AAB+15] is solicited along with Keras. [C+15] to train the studied models with said datasets. Each layer output, processed as potentially very large high-dimensional data, are saved into machine cluster handling the dissimilarity matrix computation, which makes use of a large pool of memory (around 2 Terabytes

in our infrastructure). The resulting matrices are small enough (for our experiments) to fit and be processed on a recent laptop. The matrix manipulations of the original VAT algorithm [dSW18] are implemented using the ArrayFire library [YAM+15] for their efficient matrix computation abilities. This part of the process only produces the data for the visualization tool and thus can be seen as a backend infrastructure. Fractal images (*i.e.*, glyphs) are then generated by relying on the Rust*hilbert* library [Ski04]. The visual and interactive part corresponds to an `HTML` application written in `Typescript` relying on `D3.js` for the visualization, `D3-dag` for the Sugiyama implementation and `webpack` for the build system.

E. Results and discussion

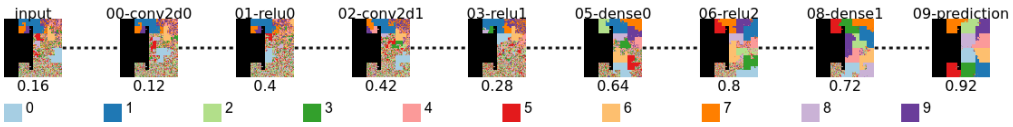
The complete results are accessible at the following address: `https://pivert.labri.fr/frac/index.html`. We strongly recommend to view the results online as the images here are severely undersized on the paper where 1 pixel represents several samples) Fig 3.3 depicts still resized representations of the proposed method for several scenarios, while their confusion matrices are presented in 3.4.

E.1. Results

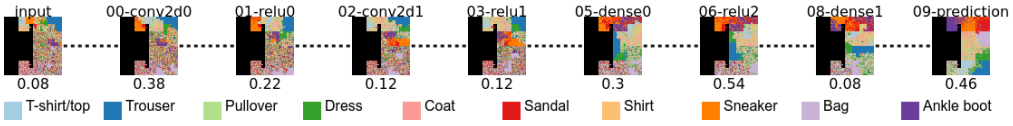
The accuracy for the system of the *easy scenario* is 98.96%. This is clearly reflected by the heterogeneous organization of the `09-prediction` glyph in the visualization. Looking at the successive operations that correspond to activation functions (`01-relu0`, `03-relu1`, `06-relu2`, `09-prediction`), we observe an improvement in the quality of the representation, and thus better discriminability ability over layers. Classes `1` and `0` seem to be discriminated early in the the network and can be considered as "easy" classes for the model. Several elements of the `6`, `7` and `8` classes are also recognized early. The first dense layer `05-dense0` brings a dramatic improvement of the discriminability in the classification.

The accuracy in the *generalization scenario* is 73.44%. Compared to the prediction layer's glyph from the *easy scenario*, this one is saltier. The previous layers overall are also less organized, which reflect the overall lower performances of the model on this dataset. The `T-shirt` and `Trouser` classes are differentiated early in the classification process while `Ankle boots` or `Sandal` are discriminated only at the end of the network.

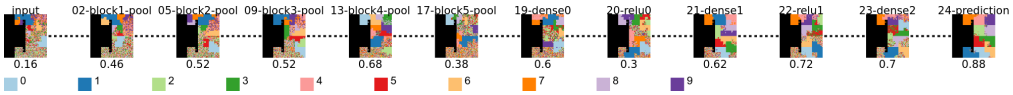
The *simplification scenario* has the accuracy of 97% (lower than *LeNet5* despite the more complex architecture). During the progressive classification, we can notice a succession of improvement and decrease in sample organization in the glyph. We assume this less efficient classification come from several reasons: they are pre-trained and not specialized for the task, and/or the model is too deep and layers are redundant. In the next subsection, we propose an improvement of the network architecture based



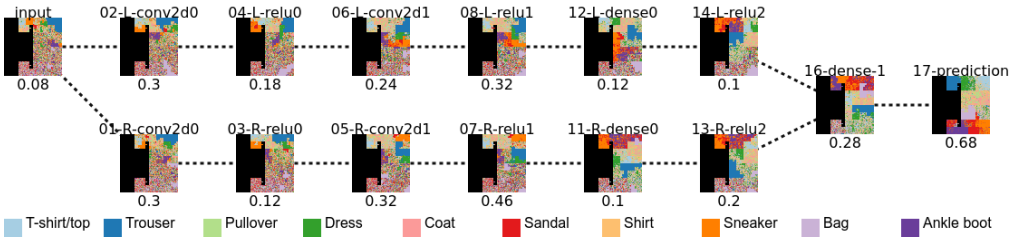
(a) Easy scenario: *LeNet5*, *MNIST* (accuracy=98.96%)



(b) Generalization scenario: *LeNet5* + *Fashion-MNIST* (accuracy=73.45%)



(c) Simplification scenario: *VGG16* + *MNIST* (accuracy=97%)



(d) Branch scenario: *DoubleLeNet5* + *Fashion-MNIST* (accuracy=73.75%)

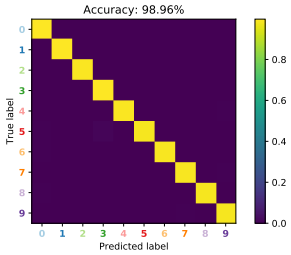
Figure 3.3: Illustration of results on some scenarios. The simplified version of the network is drawn. The confusion matrices are presented in 3.4 for comparison with a standard visual evaluation method. Larger images are available on the website `https://pivert.labri.fr/frac/`.

on this observation.

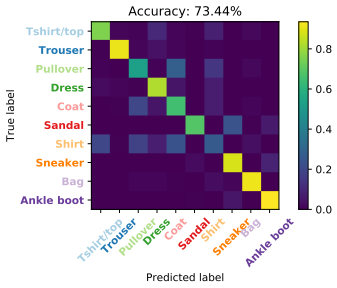
The *realistic scenario* illustrates the ability to draw networks with branches. The 73.75% accuracy of the model is very close to its linear counterpart in the *generalization scenario*. We observe the same tendencies in both branches as well as a similar sample organization than in the *generalization scenario*. They are also confident on the same classes.

E.2. Simplification Improvement

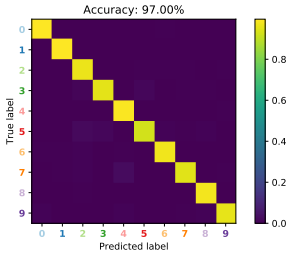
The glyph representing the fifth convolutional block of *VGG16* `17-block5-pool` shows a large degradation of the classification process, with fewer subsets of data being similarly processed than at the end of fourth convolutional block `13-block4-pool`.



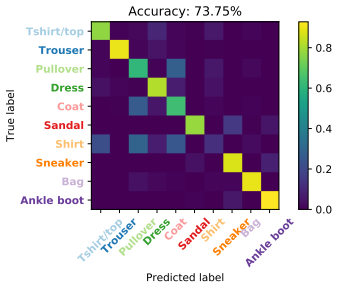
Easy scenario (*LeNet5 + MNIST*)



Generalization scenario (*LeNet5 + Fashion-MNIST*)



Simplification scenario (*VGG16 + MNIST*)



Branch scenario (*DoubleLeNet5 + MNIST*)

Figure 3.4: Confusion matrices of the systems presented in Figure 3.3.

We imply that data is being over-processed by convolution operations which degrades the model performances. We propose two ways to improve the model architecture based on this assumption :

- **VGG16-B4**: Remove the fifth convolutional block altogether and directly connect the fourth block to the top of the model. (*i.e.*, dense layers)
- **VGG16-B4+**: Remove the fifth convolutional block but keep the Pooling layer before connecting the model's top. This ensure that the number of weight parameters for each neuron of the next layer is not increased after the modification.

In both cases, the model's top is being re-trained from scratch (with only results from ImageNet processing for the convolutional blocks) to ensure that any noticeable improvement is due to the architecture modification instead of possible transfer learning from previous iteration.

The results of *VGG16*'s simplification is shown in Table 3.2. Removing the fifth convolutional block of VGG16 improved the classification performances of the model while reducing its complexity. Which improvement is better for the use case is up to the user's decision. However, even without knowing the parameters of each VGG16's layers nor the nature of the processed data, the user can notice the classification degra-

Architecture	Accuracy	Parameters	Memory Usage
VGG16	97.00%	33 638 218	447.16 ± 19.7 MB
VGG16-B4	98.65%	32 850 250	325.96 ± 3.95 MB
VGG16-B4+	98.35%	26 558 794	347.83 ± 7.9 MB

Table 3.2: Performances of different *VGG16* architectures following modification based on observations of progressive classification. *VGG16-B4* architecture has 1.67% more accuracy than the original model while having 2.34% less parameters and having 27.1% smaller footprint in RAM. *VGG16-B4+* has 1.37% more accuracy than the original but with 21% less parameters. However, its RAM footprint is only 22.2% smaller than the original model.

dation behavior of the network, and engage in network simplification manually.

E.3. Discussion

Compared to widely used t-SNE projection [RFFT17], the method presented in this chapter has a fairly more efficient use of the screen space. Furthermore, all of the layer activations can be displayed on the same screen space without overploting data. Focus on pixel-scale usage is emphasized, but not all points of the curves are used; black pixels correspond to unused pixels because test datasets are smaller than what is technically possible with such display size. Indeed, in the case of Hilbert's curve, only datasets of size 4^n can entirely fit into the curve. In our experiment, $4^7 - 10000$ pixels are lost, which is roughly 39% of the picture for each layer. It is thus possible to evaluate larger datasets without using more space on the screen, resulting in better data-ink usage. Another observation is the effectiveness of samples projection over the fractal curve to depict the classification performance over layers. Usually, representation of sample ordering is getting better over layers which means the network is progressively better at separating classes of samples. Dataset and subsets of dataset classification difficulty are also represented: *Fashion-MNIST*, which is a problem more difficult than *MNIST*, is thus less well organized.

By construction, the very first node corresponds to the projection of the raw dataset; the noisier it is, the more complex it is to distinguish its samples without extracting additional features. The representation clearly depicts this point and its quality metric is worst for *Fashion-MNIST* than *MNIST*. The very last node corresponds to the projection of the softmax values; the noisier it is, the worst the network's accuracy is. The final representation is complementary of a confusion matrix (see Figure 3.4) as it provides more information about classification efficiency of the models.

A labeled dataset is currently needed to color the pixels. It limits the use of the method to a test dataset and not a real world unlabeld dataset. However, it is still

(a) Good system, absolute coloring (b) Good system, relative coloring (c) Bad system, absolute coloring (d) Bad system, relative coloring

Figure 3.5: Comparison of the absolute and relative color schemes. No data is depicted in black for absolute and light blue for relative color schemes.

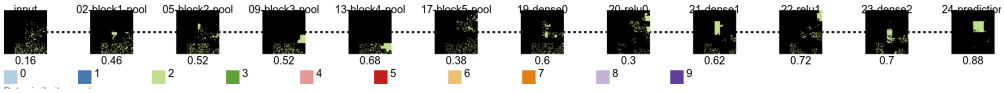


Figure 3.6: Analysis of the spread of samples of class 2 over layers. Such representation could indicated an oversizing of the network by looking at the separation effect around layer `17-block5-pool`.

possible to use the predicted labels instead of the groundtruth ones to obtain a view of how the network interprets the data.

Fig 3.5 compares the absolute and relative color schemes for one operation able to differentiate the samples and another one yet not able to differentiate them. Thanks to the color, absolute colorization allows to clearly see which classes is subject to more noise than the others, while the relative colorization rule allows to better see the relative quantity of errors, and is also able by construction to handle many classes.

Focusing on a specific class helps to track the evolution of the sample processing for that chosen class over layers. Figure 3.6 illustrates a the oversizing of the *VGG16* network on *MNIST* by observing that samples of selected class tend to be processed similarly at the `09-block3-pool` layer, whereas it is not the case anymore around layer `17-block5-pool`.

E.4. Future Work

The method is resource consuming, mainly due to the need of storing dissimilarity matrices in memory, which is of size N^2, with N being the size of the tested dataset. As a future work, it would be interesting to study whether one could use smaller part, or estimation, of the dissimilarities to approach a similar visualization. Additionally, the ordering of the samples highly depend on the Euclidean distance that is known to not be efficient in high dimensional spaces; other metrics need to be compared. The approach is satisfactory using interaction, but is not yet self-sufficient. Indeed, it provides a good overview of how the classification is handled but lacks of interactions to track the progression of a single sample or group of samples (in opposite to our

previous work that specifically focus on this point [HGBA20]) in the network. Such investigations have to be held; for example, some sort of consistency in the sample position between two successive glyph would help in tracking elements. Furthermore, being able to focus a single sample instead of the whole class would help in determining the cause of miss-classification by the network (*e.g.* the model made confusion between a 6 and a 8 at the first convolutional layer, which led in miss-classification for the rest of the processing).

The Hilbert curve is very efficient to place the samples in its reserved space. However, there is a high probability that the number of elements in the dataset to visualize is lower than what is possible with the curve. It would be interesting to implement additional interactions that use this additional space; or use grid-based projection methods instead of fractal ones. To subsample or sample with replacement the dataset with a number of samples equals to the curve length, and that follows data distribution, could also be interesting. The standard Sugiyama algorithm does not consider the screen space size; a modified method should be used in order to project the graph on the screen in a way that does not necessitate to horizontally scroll the screen to see it [LLS+18].

F. Conclusion

Deep learning classifiers are progressively replacing handcrafted and understood standard classifiers for various fields. This significant gain in performance and accuracy is counterbalanced by a steep difficulty in understanding how and why they perform so well. Information visualization is one solution to this fill this lack of interpretability. We have presented a pipeline consuming a trained network and a dataset which produces an interactive representation depicting both the network's architecture and the behaviors of each layer when they process the test samples. Such system allows to visually analyze the classification quality over layers of a dataset and could be used to visually detect patterns in the data. This analysis would lead to a hypothesis about the performance of the network. However, such hypothesis would need then to be verified by other means.

This approach has been validated on various scenarios and shows its interest and limits that could be overcome in the future. Extensions with various specific interaction methods to also focus on individual data, efficient data subsampling and dense pixel-based glyph construction with better screen-space usage and/or less restrictions would imrpove the method for more complex and precise network analysis.

ACKNOWLEDGMENTS

his work has been carried out with financial support from the French State. Experiments presented in this paper were carried out using the Labo's in the Sky with Data(LSD), the LaBRI data platform partially funded by Region Nouvelle Aquitaine.

References

AAB+15. Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, and *et al.* TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

AFMB+20. Kazi Ahmed Asif Fuad, Romain Martin, Pierre-Etienne andGiot, Romain Bourqui, Jenny Benois-Pineau, and Akka Zemmari. Features understanding in 3d cnns for actions recognition in video. In *The tenth International Conference on Image Processing Theory, Tools and Applications (IPTA 2020)*, page 6, 2020.

ANM07. David Auber, Noël Novelli, and Guy Melançon. Visually mining the datacube using a pixel-oriented technique. In *2007 11th International Conference Information Visualization (IV'07)*, pages 3–10. IEEE, 2007.

BBHR+16. Michael Behrisch, Benjamin Bach, Nathalie Henry Riche, Tobias Schreck, and Jean-Daniel Fekete. Matrix reordering methods for table and network visualization. *Computer Graphics Forum*, 35(3):693–716, 2016.

BHL05. Frédéric Blanchard, Michel Herbin, and Laurent Lucas. A new pixel-oriented visualization technique through color image. *Information Visualization*, 4(4):257–265, 2005.

BML+16. Alexander Binder, Grégoire Montavon, Sebastian Lapuschkin, Klaus-Robert Müller, and Wojciech Samek. Layer-wise relevance propagation for neural networks with local renormalization layers. volume 9887 of *Lecture Notes in Computer Science*, pages 63–71. Springer Berlin / Heidelberg, 2016.

C+15. François Chollet et al. Keras, 2015.

dSW18. L. E. B. d. Silva and D. C. Wunsch. A study on exploiting vat to mitigate ordering effects in fuzzy art. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2018.

EBCV09. Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. Technical report, University of Montreal, 2009.

Gar76. Martin Gardner. Mathematical games–in which "monster" curves force redefinition of the word "curve". *Scientific American*, 235(6):124–133, 1976.

GYA17. Mostafa Mehdipour Ghazi, Berrin Yanikoglu, and Erchan Aptoula. Plant identification using deep neural networks via optimization of transfer learning parameters. *Neurocomputing*, 235:228–235, 2017.

HGBA20. Adrien Halnaut, Romain Giot, Romain Bourqui, and David Auber. Deep dive into deep neural networks with flows. In *15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, pages 231–239, 2020.

HGBA21. Adrien Halnaut, Romain Giot, Romain Bourqui, and David Auber. Samples classification analysis across dnn layers with fractal curves. In *ICPR 2020's Workshop Explainable Deep Learning for AI*, 2021.

Hil35. David Hilbert. Über die stetige abbildung einer linie auf ein flächenstück. In *Dritter Band: Analysis· Grundlagen der Mathematik· Physik Verschiedenes*, pages 1–2. Springer, 1935.

HKPC18. Fred Hohman, Minsuk Kahng, Robert Pienta, and Duen Horng Chau. Visual analytics in deep learning: An interrogative survey for the next frontiers. *IEEE transactions on visualization and computer graphics*, 25(8):2674–2693, 2018.

HZRS16. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

HZS17. Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1389–1397, 2017.

KAKC18. M. Kahng, P. Y. Andrews, A. Kalro, and D. H. Chau. Activis: Visual exploration of industry-scale deep neural network models. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):88–97, 2018.

Kei00. Daniel A Keim. Designing pixel-oriented visualization techniques: Theory and applications. *IEEE Transactions on visualization and computer graphics*, 6(1):59–78, 2000.

KTC+18a. Minsuk Kahng, Nikhil Thorat, Duen Horng Polo Chau, Fernanda B Viégas, and Martin Wattenberg. Gan lab: Understanding complex deep generative models using interactive visual experimentation. *IEEE transactions on visualization and computer graphics*, 25(1):1–11, 2018.

KTC+18b. Minsuk Kahng, Nikhil Thorat, Duen Horng Polo Chau, Fernanda B Viégas, and Martin Wattenberg. Gan lab: Understanding complex deep generative models using interactive visual experimentation. *IEEE transactions on visualization and computer graphics*, 25(1):1–11, 2018.

LBBH98. Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

LBH15. Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

LeC98. Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

LLS+18. Mengchen Liu, Shixia Liu, Hang Su, Kelei Cao, and Jun Zhu. Analyzing the noise robustness of deep neural networks. In *2018 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pages 60–71. IEEE, 2018.

LSL+17. Mengchen Liu, Jiaxin Shi, Zhen Li, Chongxuan Li, Jun Zhu, and Shixia Liu. Towards better analysis of deep convolutional neural networks. *IEEE transactions on visualization and computer graphics*, 23(1):91–100, 2017.

MH08. Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9:2579–2605, 2008.

Mor66. Guy M Morton. A computer oriented geodetic data base and a new technique in file sequencing. Technical report, International Business Machines Company New York, 1966.

MSQM15. Miriah Meyer, Michael Sedlmair, P Samuel Quinan, and Tamara Munzner. The nested blocks and guidelines model. *Information Visualization*, 14(3):234–249, 2015.

NQ17. Andrew P Norton and Yanjun Qi. Adversarial-playground: A visualization suite showing how adversarial examples fool deep learning. In *Visualization for Cyber Security (VizSec), 2017 IEEE Symposium on*, pages 1–4. IEEE, 2017.

RDS+15. Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

RFFT17. Paulo E Rauber, Samuel G Fadel, Alexandre X Falcao, and Alexandru C Telea. Visualizing the hidden activity of artificial neural networks. *IEEE transactions on visualization and computer graphics*, 23(1):101–110, 2017.

SCD+17. R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *2017 IEEE International Conference on Computer Vision (ICCV)*, page 618–626, 2017.

SCS+17. Daniel Smilkov, Shan Carter, D Sculley, Fernanda B Viégas, and Martin Wattenberg. Direct-manipulation visualization of deep networks. *arXiv preprint arXiv:1708.03788*, 2017.

Ski04. John Skilling. Programming the hilbert curve. In *AIP Conference Proceedings*, volume 707, pages 381–387. American Institute of Physics, 2004.

STT81. Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11:109–125, 1981.

SZ14. Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *preprint arXiv:1409.1556*, 2014.

Tan13. Yichuan Tang. Deep learning using linear support vector machines. In *In ICML*, 2013.

WGSY19. Junpeng Wang, Liang Gou, Han-Wei Shen, and Hao Yang. Dqnviz: A visual analytics approach to understand deep q-networks. *IEEE transactions on visualization and computer graphics*, 25(1):288–298, 2019.

WSW+17. Kanit Wongsuphasawat, Daniel Smilkov, James Wexler, Jimbo Wilson, Dandelion Mane, Doug Fritz, Dilip Krishnan, Fernanda B Viégas, and Martin Wattenberg. Visualizing dataflow graphs of deep learning models in tensorflow. *IEEE transactions on visualization and computer graphics*, 24(1):1–12, 2017.

WTS+20. Zijie J Wang, Robert Turko, Omar Shaikh, Haekyu Park, Nilaksh Das, Fred Hohman, Minsuk Kahng, and Duen Horng Chau. Cnn 101: Interactive visual learning for convolutional neural networks. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–7, 2020.

XRV17. Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

YAM+15. Pavan Yalamanchili, Umar Arshad, Zakiuddin Mohammed, Pradeep Garigipati, Peter Entschev, Brian Kloppenborg, James Malcolm, and John Melonakos. ArrayFire - A high performance software library for parallel computing with an easy-to-use API, 2015.

ZNWZ18. Quanshi Zhang, Ying Nian Wu, and Song-Chun Zhu. Interpretable convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8827–8836, 2018.

ZYMW19. Quanshi Zhang, Yu Yang, Haotian Ma, and Ying Nian Wu. Interpreting cnns via decision trees. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6261–6270, 2019.