Computing Fast and Accurate Maps for Explaining Classification Models

Yu Wang[a,*], Cristian Grosu[a], Alexandru Telea[a]

[a] *Department of Information and Computing Sciences, Utrecht University, Utrecht, 3584 CS, The Netherlands*

Abstract

Image representations of the behavior of trained machine learning classification models can help machine learning engineers examine various aspects of a model such as how it partitions its data space into decision zones separated by decision boundaries; how training samples support the decision in various parts of the data space; and how close training data is to decision boundaries. Yet, for an image of $n \times n$ pixels, all current methods that create such images have a computational complexity of $O(n^2)$ which precludes their use in interactive visual analytics scenarios. We present a set of techniques for the fast computation of such image-based classifier representations. Compared to earlier work in this area, we accelerate both so-called decision maps, that compute categorical labels, and classifier maps, that compute real-valued quantities, in $O((\log n)^2)$ time. Practically, our method has a speed-up of about one order of magnitude and yields results very similar to the ground-truth maps; has no free parameters; is model agnostic; and is simple to implement. We demonstrate our method on several combinations of maps, datasets, and classification models.

Keywords: Decision Maps, Inverse Projection, Fast Computation, Explainable AI, Visual Analytics

1. Introduction

Machine learning research proposes increasingly many, and more complex, models for classification and regression. As such, there is a growing need for techniques and tools that help both researchers and practitioners to understand how these models work. In particular, visual analytics techniques approach this task by depicting various aspects of such models [1, 2, 3]. When combined with interaction, such techniques allow users to effectively explore the behavior of trained models and further answer questions concerning their generalizability, robustness, trust, and ways of improving their training to gain accuracy [4, 2].

Decision maps are a simple but effective instrument in the above class [5]. Such methods map a part of the high-dimensional data space on which a trained classification model operates to a 2D image. The hue, saturation, and brightness values of image pixels encode inferred label and model confidence of the trained model at the respective data locations. Such images show the model's so-called *decision zones*, *i.e.* areas where the model infers the same label; and *decision boundaries*, *i.e.*, locations in the data space where the model changes decision. The same image-based idea can be used to create so-called *classifier maps* which encode more advanced aspects of the model, beyond inferred classes and confidences. Such aspects include the model's sensitivity to small changes in its inputs [6]; the distance to training data or decision boundaries; or the model's sensitivity to mislabeled samples [7].

Several techniques for computing decision maps have been proposed [8, 9, 10, 7]. However, computing a decision map image, even at quite small resolutions of hundreds of pixels squared, takes tens of seconds up to tens of minutes, depending on the decision map technique [11]. This precludes using such decision maps in scenarios where users aim to *interactively and iteratively* improve a classification model by *e.g.* changing its hyperparameters or performing data pseudo-labeling in active learning settings [12, 9, 13].

To alleviate this, we recently proposed FastDBM, a set of techniques that speeds up the computation of decision maps [14]. This method can speed up any decision map that encodes classifier label and confidence without inner knowledge of how the model operates. Also, the method is simple to implement, has no hidden parameters, and creates images practically identical to the ground-truth, slow to compute, ones. However, FastDBM cannot be applied to accelerate the computation of classifier maps that depict properties beyond inferred class value and classification confidence. For instance, FastDBM cannot be used to accelerate gradient maps [6] or differential decision maps [7].

In this work, we show that FastDBM can be easily extended to compute any real-value map created via inverse projection that depends only on sample positions, such as the gradient maps and differential decision maps mentioned above, referred next as classifier maps. This requires only a simple modification of the original FastDBM method. We show that our modification still keeps the attractive speed-up and low error rates proposed by the original FastDBM technique. Besides this key extension, we also explore additional combinations of techniques and quality metrics to gauge the added value of our proposal.

We next summarize our contributions:

- We present FastDBM, an acceleration technique for computing decision maps for trained classification models, which enables such maps to be used in interactive settings.

*Corresponding author

Email addresses: y.wang6@uu.nl (Yu Wang), c.grosu@students.uu.nl (Cristian Grosu), a.c.telea@uu.nl (Alexandru Telea)

- We extend FastDBM to accelerate the creation of so-called classifier maps that depict any (smooth) real-valued property of the studied classification model, such as gradient maps and distance to decision boundary maps.

- We present a detailed evaluation of FastDBM on different classifiers, datasets, decision map and classifier map methods, and quality metrics. Our evaluation confirms the high accuracy and speed of FastDBM in all tested cases.

The structure of this paper is as follows. Section 2 introduces related work on decision maps, classifier maps, and techniques used for computing these. Section 3 presents the core of our FastDBM technique which is used to compute decision maps. Section 4 evaluates the three acceleration heuristics we proposed for FastDBM and outlines the winning heuristic: binary split. Section 5 presents additional evaluations focusing on the binary split heuristic. Section 6 presents our extension of FastDBM to handle real-valued classifier maps and shows examples of accelerating three such map types. Section 7 discusses the features of our method. Finally, Sec. 8 concludes the paper.

2. Related work

Let $D = \{\mathbf{x}_i\} \subset \mathbb{R}^n$ be a high-dimensional dataset. A classification model $f : \mathbb{R}^n \to C$, trained and/or tested on D, maps samples from the data space to a categorical (label) domain C. Let $c : \mathbb{R}^n \to [0, 1]$ denote the confidence of this classification. A *decision map* is a two-dimensional image I that aims to capture f's behavior by extrapolating it from D. Two elements are key to the construction and use of I, as follows:

Direct projection: Let $P : D \to \mathbb{R}^2$ be a so-called dimensionality reduction, or projection, operation, such as t-SNE [15], UMAP [16], PCA [17], or any of the many other such techniques [18, 19]. Let $P(D) = \{P(\mathbf{x}) | \mathbf{x} \in D\}$ be the mapping of D to a 2D scatterplot computed by P. $P(D)$ only depicts the behavior of f over the *discrete set* of samples D. One can next color points $P(\mathbf{x}) \in P(D)$ by the value of the inferred class $f(\mathbf{x})$ and study the resulting colored scatterplot to get an idea of how f acts on groups of similar or different samples. Yet, one has no idea what f does *between* samples, in the gaps between scatterplot points. This is especially important when one is concerned with f's behavior close to its so-called decision boundaries, *i.e.*, places where f changes value. Such boundaries most likely will pass between points in $P(D)$, so are not shown by the scatterplot.

Decision maps aim to solve precisely this – namely, present users with a *dense* image that shows f's behavior at every pixel [8]. To construct such maps, we must extrapolate information from $P(D)$ over the entire image $I \subset \mathbb{R}^2$.

Inverse projection: Inverse projections provide precisely what is needed for the above-mentioned extrapolation. These are functions $P^{-1} : \mathbb{R}^2 \to \mathbb{R}^n$ that inversely map, or backproject, any pixel $\mathbf{p} \in I$ to a data space location $P^{-1}(\mathbf{p})$. Inverse projections allow one to explore the gap areas between the points of a projection scatterplot $P(D)$ – either interactively or simply by having such information displayed there – for many applications such as data augmentation [20, 21], morphing and data imputation [22, 6], and, closer to our context, analyzing trained ML classification models by decision maps [23, 10, 9], as discussed next

Map creation – overview: Using P^{-1}, one can now depict, at every image pixel $\mathbf{p}$, any property of interest that is measured in the data space $\mathbb{R}^n$. A first example hereof are decision maps

$$F(\mathbf{p}) = f(P^{-1}(\mathbf{p})) \tag{1}$$

which color each $\mathbf{p}$ by the class label inferred by f at that back-projected location. Additionally, the confidence c of the model can be evaluated at $\mathbf{p}$ and encoded in *e.g.* saturation or brightness [8, 9, 10, 7]. Figure 2a shows a decision map that encodes the model's inference for the well-known MNIST dataset [24].

Classifier maps extend this idea by allowing one to substitute f in Eqn. 1 by any real-valued function of interest defined on the data space. For instance, *gradient maps* [6, 7] compute the (approximate) norm of the gradient of P^{-1} at $\mathbf{p} = (x, y)$ as

$$G(\mathbf{p}) = \sqrt{\left(\frac{\partial P^{-1}}{\partial x}(\mathbf{p})\right)^2 + \left(\frac{\partial P^{-1}}{\partial y}(\mathbf{p})\right)^2}. \tag{2}$$

Visualizing G over I shows areas where P^{-1} has high gradients, *i.e.*, where extrapolating the model f away from samples in D can be risky due to the so-called compression of the data space to the 2D space created by the projection P [25, 18].

Another classifier map visualizes, for each pixel $\mathbf{p}$, the distance to the closest decision boundary

$$d_B(\mathbf{p}) = \min_{\Delta\mathbf{x}\in\mathbb{R}^n} \{||\Delta\mathbf{x}||_2 \mid f(P^{-1}(\mathbf{p}) + \Delta\mathbf{x}) \neq f(P^{-1}(\mathbf{p}))\}, \tag{3}$$

which allows one to find different areas in the data space where the trained model may be brittle [8, 7]. Computing d_B is however expensive as it requires bisection-like search for the closest decision boundary [8] or running adversarial example generation [26]. Our acceleration proposal is thus highly relevant here.

A final classifier map example is the distance to the closest training sample

$$d_D(\mathbf{p}) = \min_{\mathbf{x}\in D} ||P^{-1}(\mathbf{p}) - \mathbf{x}|| \tag{4}$$

which helps finding areas where f extrapolates far from its training data D, *i.e.*, where the model's behavior can be less reliable, despite high confidence values [7]. Summarizing, as opposed to decision maps which depict the model $f : \mathbb{R}^n \to C$, classifier maps depict any real-valued function $g : \mathbb{R}^n \to \mathbb{R}$ that helps understanding f's behavior directly or indirectly.

Decision and classifier maps are useful tools for explainable AI. At a basic level, decision maps help users understand how a model works by showing where the model is confident and where it is uncertain [23]. This insight supports active learning where the user annotates training-set samples located in low-confidence decision map areas or close to decision boundaries [21, 27]. In this scenario, quickly recomputing the decision map after user annotation is crucial to support the visual analytics 'human in the loop' process. Additionally, they can also be used to evaluate a classifier's brittleness against backdoor and data poisoning attacks [9, 7].

Map creation – technical choices: The choices for P and P^{-1} strongly affect the resulting decision maps. Unlike direct projections, only a few inverse projection methods P^{-1} exist. An early such method, iLAMP [22], builds local affine mappings that revert the LAMP [28] direct projection. To address iLAMP's lack of continuity and global mapping, a later study proposed a Radial Basis Function (RBF) based inverse projection method [29]. NNInv massively accelerated computing inverse projections by deep learning the 2D to $\mathbb{R}^n$ mapping [30]. Self-Supervised Neural Projection (SSNP) deep learns P and P^{-1} jointly [31] using an autoencoder approach [32]. SSNP inherits the speed of NNInv but produces smoother decision maps [10, 11].

To create decision maps, Rodrigues et al.[8] used t-SNE and LAMP [28] for P and iLAMP [22] for P^{-1}, respectively. Further on, Rodrigues et al. [23] presented DBM, an approach in which one can freely choose P and P^{-1} to create decision maps. They evaluated 28 methods for P and two methods for P^{-1} and found out that t-SNE and UMAP are optimal choices for P; and NNInv is the optimal choice for P^{-1}. We next use in our work DBM to denote decision maps that use NNInv as P^{-1}. DBM was next refined by Self-Supervised Decision Maps to produce higher-quality decision maps (SDBM, [10]). Separately, DeepView [9] proposed discriminative dimensionality reduction to construct decision maps using UMAP for P^{-1}. Recently, Wang et al. [11] presented a detailed evaluation of decision map techniques from the perspective of quality and computational scalability. They found that t-SNE and UMAP (for P) and NNInv (for P^{-1}) yield very good results, surpassed only by DeepView. DeepView is, however, *orders* of magnitude slower than other methods, so our proposed acceleration cannot bring it to work at near-interactive rates. Summarizing the above, we next consider the following decision map techniques as targets to accelerate: autoencoders (for both P and P^{-1}); SSNP (for both P and P^{-1}); and DBM (with PCA, UMAP, and t-SNE for P and NNInv for P^{-1}).

Scalability: All current decision map techniques are slow – on a typical commodity PC, computing a decision map for resolutions of 250^2 pixels takes about 10 seconds for all tested methods except DeepView; for DeepView, this takes several hours, depending on the classifier used [11]. As we shall see in Sec. 4, costs increase quadratically with the decision map resolution – and higher resolutions are needed to create maps in which users see the exact shape of decision boundaries; variations due to compression in gradient maps [25, 18]; and distances to decision boundary or to training samples [7]. Current decision map methods are thus not suitable for visual analytics scenarios that require fast recomputation of decision and/or classifier maps upon re-training of the studied model.

3. FastDBM computation

For an image I of $n \times n$ pixels, the complexity of current decision map methods is $O(n^2 K)$, where K is the cost of a single $f(P^{-1}(\cdot))$ operation. Decreasing K is hard if we allow any generic inverse projections P^{-1} and classifier model f. Hence, to improve speed, we next aim to reduce the n^2 term.

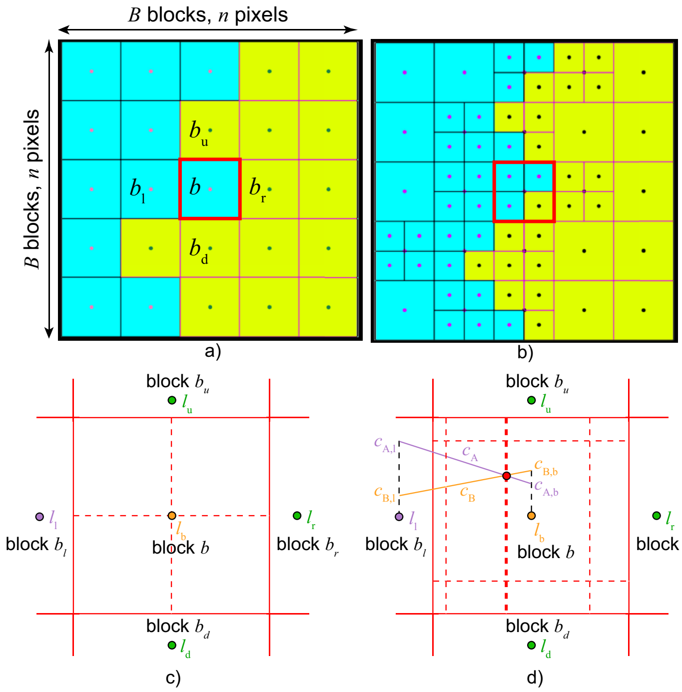

Figure 1: Illustration for binary and confidence-based splitting heuristics. Given the block-set in (a), binary split creates the refined block-set in (b). For the block in (c), binary split would create four equal-sized blocks along the thin dashed lines. In contrast, confidence-based splitting (d) examines the confidence values and splits the block in up to 9 smaller blocks along the thin dashed lines.

A classification model f, in general, must fit its decision boundaries so that they (a) surround same-class training points, but (b) the boundaries are sufficiently *smooth* to allow for generalization without overfitting. Given (b), f, and thus a decision map that aims to accurately capture f, has in general relatively *few* compact decision zones (not necessarily one zone per class). We use this property to devise our acceleration as follows.

3.1. Binary split

We start by dividing the image I into B^2 blocks – each such block is a square of $\frac{n}{B} \times \frac{n}{B}$ pixels from I. For each block b, we evaluate the label $l_b = f(P^{-1}(\mathbf{p}))$ at its central pixel $\mathbf{p}$. Figure 1a shows this for a binary classifier (cyan and yellow are the two classes). Let l_u, l_d, l_l, l_r be the labels computed similarly for the up, down, left, and right neighbor blocks of b. Let N be the number of neighbors with labels different from l_b. If $N = 0$, then b is surrounded by same-label blocks, so, if we assume that a decision zone in the decision map is locally *thicker* than $\frac{n}{B}$ pixels, no decision boundary crosses it. Hence, we can assign l_b to all pixels in b. If $N > 0$, we split b into four equal smaller blocks. Figure 1b shows the results of this splitting. We repeat the process, in a quadtree-like fashion, until we arrive at pixel-sized blocks or blocks do not need splitting anymore. During this, we note that (1) splitting larger blocks first helps to ensure a uniform refinement all over the image; and (2) splitting blocks having several neighbors with different labels is better than splitting blocks having a single such neighbor since the former cover more decision boundary fragments. We model this by keeping blocks to split in a priority queue sorted decreasingly on $d \cdot d \cdot \frac{N}{C}$ where d is the size of a block, N is its number of different-label neighbors, and C is its neighbor count (4 for blocks inside the decision map, 3 for blocks on the map boundary, and 2 for blocks on the map corners).

As Sec. 2 outlines, a decision map also often shows the *confidence* of the visualized model f at each map pixel. Per block, however, we have a single data sample $P^{-1}(\mathbf{p})$, computed at the block's center pixel $\mathbf{p}$. This is fine for class labels since these are *constant* over decision zones, thus also per block as per our splitting heuristic. In contrast, confidence varies *continuously* within a decision zone, hence can also vary within a block. We avoid computing additional confidence values apart from $c(\mathbf{p})$ by interpolating these values, computed at the blocks' centers $\mathbf{p}$, using nearest-neighbor, bilinear, and bicubic schemes.

3.2. Confidence split

The binary split is a simple bisection procedure to find the places in $\mathbb{R}^2$ where decision boundaries are, up to the pixel precision of I. We can potentially use the confidence values $c(\mathbf{p})$ to refine this process as follows. Take the block b shown in Fig. 1c. Binary split would divide b along the dashed lines in the image. Consider the confidence values c_A and c_B for the inferred classes purple, respectively orange, sampled at the centers of cells b_l and b, denoted next as $c_{A,l}$, $c_{B,l}$, $c_{A,b}$ and $c_{B,b}$ respectively (Fig. 1d). We next linearly interpolate these values to find the point where $c_A = c_B$ (red point, Fig. 1d). This is likely a good point to split cell b (along the thick dashed line, Fig. 1d) since, left to this point, class A has a higher confidence than class B (so the decision zone there should tell A) and, right to this point, class B has a higher confidence than class A (so the decision zone there should tell B). Note how this confidence maximization is precisely similar to how classification models internally decide on the class to output, albeit using more complex interpolation schemes than our linear one. We proceed in the same way for all class values with respect to all four boundaries of cell b. This yields a possible set of 2, 3, 4, 6, or 9 cells that split b as opposed to the fixed 4 cells done by binary split (see thin dashed red lines in Fig. 1d). Confidence is next interpolated as for the binary split method.

3.3. Confidence sampling

Our final acceleration heuristic uses the underlying idea that, if we can capture the confidence c at a *coarse* sampling resolution, then we can find decision zones (and boundaries) at pixel resolution by maximizing c over all inferred labels. This will reduce the costs implied by the block-splitting process. For this, given our initial B^2 blocks, we compute confidences $c(\mathbf{p})$ at block centers $\mathbf{p}$, for *all* inferred $|C|$ classes, and next interpolate these over I using nearest-neighbor, bilinear, or bicubic techniques – as described above, but now only over the initial blocks, which we do not further split. Next, for each pixel $\mathbf{p} \in I$, we compute which class yields the highest interpolated confidence and assign that class to $\mathbf{p}$.

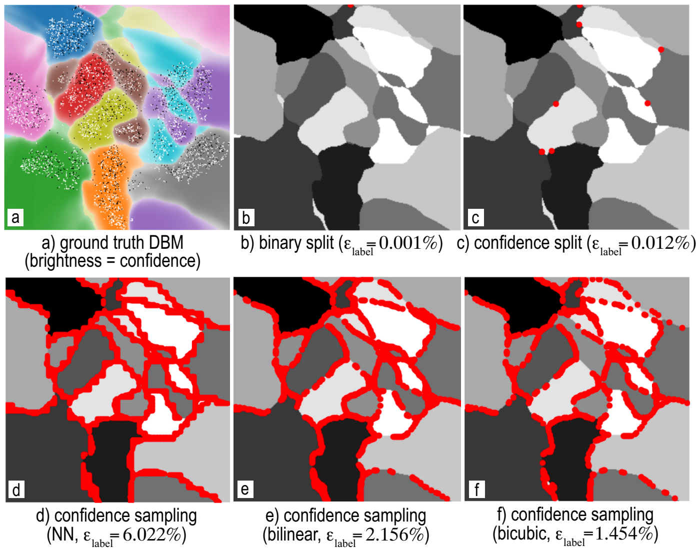

Figure 2: a) Ground-truth DBM with labels and confidence encoded into colors, respectively saturation, MNIST dataset. b-f) Class assignment errors for FastDBM method variants.

4. Evaluation of acceleration heuristics

4.1. Comparison of acceleration heuristics

We now compare our three acceleration heuristics (binary split, confidence split, confidence sampling) against each other and with the ground truth. For this, we use two metrics:

Label errors: We ideally want to get the same labels for a FastDBM image I_{fast} and the ground-truth decision map image I. We evaluate this by the error

$$\epsilon_{label} = \frac{100}{n^2} \sum_{1 \leq x \leq n, 1 \leq y \leq n} \delta(I(x,y), I_{fast}(x,y)), \tag{5}$$

where $\delta(a,b)$ is 0 if $a = b$ and 1 otherwise. That is, ϵ_{label} measures the percent of the $n \times n$ FastDBM map image which is different from the ground truth.

Confidence errors: Our interpolated confidence c_{fast} should be as close as possible to the ground-truth one c. We evaluate this by the normalized MSE error

$$\epsilon_{conf} = \frac{\sum_{1 \leq x \leq n, 1 \leq y \leq n} (c(x,y) - c_{fast}(x,y))^2}{\sum_{1 \leq x \leq n, 1 \leq y \leq n} c(x,y)^2}. \tag{6}$$

Figure 2 shows our results for the MNIST dataset [24], classified with a simple deep learning network f (flatten layer, dense 10-unit layer and softmax activation, 20 training epochs, 3.5K training samples, 1.5K test samples). We use DBM (P set to t-SNE, P^{-1} set to NNInv) to create the decision maps; map image size $n = 256$ pixels, $B = 8$ blocks. Image (a) shows the ground-truth DBM with labels and confidence color- respectively saturation-coded. Images (b-f) show the results of our binary split, confidence split, and confidence sampling heuristics, the latter using nearest neighbors, bilinear, and bicubic interpolation. Red points show pixels where ground-truth labels differ from our results. Our heuristics yield practically the same DBMs, with only a few different pixels. The binary split method is best – only 8 pixels of the 256^2 are different; the confidence sampling method is the worst; for the latter, errors appear *strictly* on the decision boundaries. This is likely since confidence varies slowly inside decision zones but rapidly close to boundaries (see Fig. 2a), so our interpolation has difficulties in the latter areas.

Figure 3 shows the errors ϵ_{label} and ϵ_{conf} and computing time for the above experiment for different image resolutions n (100 to 2000 pixels squared). For confidence sampling, we only use bicubic interpolation as this yields lower errors than nearest neighbor and bilinear (see Fig. 2). Error-wise, the binary split and confidence split methods are very similar and consistently lower than confidence sampling since the latter method uses a single *fixed* block resolution which, if too low, is unable to capture complex signal variations over the map image. Also, the binary split and confidence split errors are virtually constant with n, while confidence sampling errors show a slight increase with n. Speed-wise, the binary and confidence-sampling methods show near-linear behavior in n (with a very small slope) as opposed to the quadratic behavior of ground-truth DBM, with the confidence-split method in between the two. The binary and confidence-sampling methods are over one order of magnitude faster than ground-truth DBMs. The confidence split method's relative low speed can be explained by the fact that it can create up to 9 cells when splitting a single block as opposed to exactly four for the binary split (see Fig. 1 and related text). Note also that our maximal resolution $n = 2000$ exceeds *by far* all reported DBM results in the literature. From the above, we conclude that the binary split method is the clear winner when considering computational speed and accuracy factors. As such, we focus only on this method in our further evaluations.

4.2. Parameter setting for binary split heuristic

Binary split has one parameter – the initial block count B – so how to set its value? A high B will limit errors due to dense sampling of the image, but will be slow, since $f(P^{-1})$ must be evaluated on many blocks. A low B will be fast, but as Sec. 3 notes, decision map details under $\frac{n}{B}$ may be lost. To find a good initial value for B, we measure both speed and label errors ϵ_{label} for various B settings ranging from 8 to 96. To generalize our findings, we test several combinations of P and P^{-1} to compute our ground-truth decision maps, specifically autoencoders (AE, used for both P and P^{-1}); SSNP (used for both P and P^{-1}); and DBM (PCA, UMAP, and t-SNE used for P, NNInv used for P^{-1}). Figure 4 shows the speed and label errors as function of B for our maximally considered resolution $n = 2000$. We see that, label-error-wise, all B values above roughly 32 yield (very) low errors. Speed-wise, B values in the interval 32-64 offer best results, which confirms our earlier observations that too low or too high B will be slow. Also, we see that the overall speed trend as function of B does not strongly depend on the choice of (P, P^{-1}), up to a constant bias factor. Hence, we conclude that a block size $B = 32$ is a good preset for FastDBM.

4.3. Implementation details

Our FastDBM method is implemented in Python and runs fully on the CPU. The full source code, including datasets and experiments presented here, is publicly available [33].

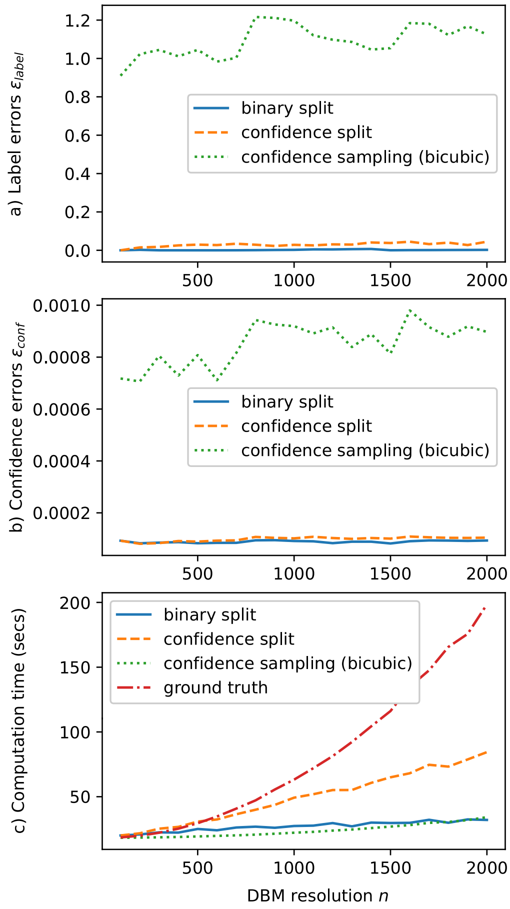

Figure 3: Label errors ϵ_{label} (a), confidence errors ϵ_{conf} (b), and computation time (c) for our three acceleration heuristics, MNIST dataset.

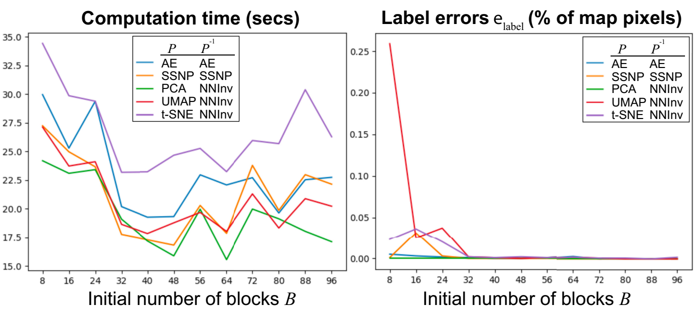

Figure 4: Speed (left) and label errors (right) of binary split method as function of initial block count B for decision maps constructed by various (P, P^{-1}) methods.

5. In-depth evaluation of binary split acceleration

We found that binary split works the best among the three proposed heuristics (Sec. 4.1). We now further evaluate the binary split heuristic using more classifiers, an additional quality metric, and using decision maps constructed with all inverse projection techniques that we are aware of.

5.1. Using additional classifiers

We evaluate the binary split method with additional combinations of datasets and classifiers used to compute the ground-truth DBM (P set to t-SNE or UMAP, P^{-1} set to NNInv). The datasets include FashionMNIST [34], HAR [35], and Iris [36]. Classifiers included logistic regression (LR), support vector machines (SVM), k-nearest neighbors (kNN), decision trees (DT), random forests (RF), and the neural network (NN) we used earlier for MNIST. It is important to note that the accuracy of the trained models is of no concern in this experiment. If FastDBM approximates well the ground-truth DBM, FastDBM can be next used next to assess how well (or poorly) the models behave.

Ground-truth DBMs were created by the DBM (t-SNE, NNInv) and DBM (UMAP, NNInv) combinations at resolution $n = 400$ pixels squared. Figure 5 shows the ground-truth DBMs; those created by our binary split method; 2D projections of training samples in green and the label difference encoded by red dots as in Fig. 2, for the MNIST, FashionMNIST, and HAR datasets. We see that our method yields visually almost identical label results as the ground-truth – there are only few red points in the 'difference' images. This occurs consistently for quite different DBMs, *e.g.*, the smooth decision-zone DBMs created for LR, NN, SVM, and KNN, but also the far noisier DBM created for DT, and the overall low-confidence DBM created for RF. Additional results for all other tested combinations, present in the supplementary material, confirm this observation.

5.2. Consistency evaluation

To further confirm the visual similarity between the ground-truth decision maps and the FastDBM versions shown in Fig. 5, we next compare the *map consistency* metric computed for both cases. In detail, map consistency

$$Cons_p = \frac{\left| \{\mathbf{p} \in I \mid f(P^{-1}(P(P^{-1}(\mathbf{p})))) = f(\mathbf{p})\} \right|}{|I|} \tag{7}$$

adapts the data consistency metric [9], earlier used to measure how well an inverse projection P^{-1} reverts the effects of a direct projection P, to points outside a given dataset D for which we have ground-truth for P^{-1}. In other words, $Cons_p$ computes the fraction of 'consistent' pixels in a decision map image, *i.e.*, pixels whose corresponding data points (obtained by P^{-1}) have the same class label after a round-trip of projection and inverse projection [11]. If our acceleration technique works well, then the consistency $Cons_p^{fast}$ of the images it produces should be very close to the consistency $Cons_p$ of the ground-truth decision map images. Table 1 shows, for several datasets and classifiers, the values of $Cons_p^{fast}$ computed by binary split for DBM (UMAP+NNInv) and SDBM compared to the ground-truth $Cons_p$. We see that, although both $Cons_p^{fast}$ and $Cons_p$ are less than the ideal value 1 (which would imply that P^{-1} is an exact inverse of P), their values are very close to each other. That is, the quality of the images produced by FastDBM is very close to the ground-truth images.

5.3. Accelerating additional direct and inverse projection techniques for creating decision maps

So far, we computed our (accelerated) decision maps using NNInv and SSNP for the inverse projection P^{-1} since, as mentioned in Sec. 2, earlier work showed that NNInv and SSNP are fast and accurate for this task. Yet, other inverse projection techniques do exist, most notably iLAMP [22] and the inverse projection using radial basis functions (RBF) [29]. Earlier work has shown that both these techniques are slower than NNInv [30]. However, it is interesting to see how these techniques fare given our acceleration. Separately, iLAMP and RBF have a quite different behavior from the already-tested NNInv and SSNP. Hence, if our acceleration technique can create *accurate* approximations of decision maps using these inverse projections, this increases the claims of generality of our proposal.

Figure 6 shows, for the MNIST dataset, the decision maps computed by four ground-truth technique pairs (t-SNE and iLAMP, UMAP and iLAMP, t-SNE and RBF, and UMAP and RBF) and their counterparts produced by our binary split acceleration. The ground-truth maps are noisier than those we computed so far using NNInv and SSNP for P^{-1}, in line with earlier findings [30, 8]. Our binary split method captures these ground truth images quite well – the label difference images show only a few pixels where our results differ from the ground truth, much like in Fig. 5. Our method speeds up the computation of most maps – see timing figures in the lower-left corners of the images. Speed up overall ranges from 140% to 450% except for the t-SNE and iLAMP combination which is only 15% faster. This is due to the high irregularity of the decision boundaries in this case, which generates a very large number of cell splits – see the corresponding 'binary split process' images in Fig. 6.

Concluding, we claim that our binary split heuristic creates accurate decision maps for all existing inverse projections we are aware of; and, for most cases except very noisy decision maps (which are likely not useful in practice), it also accelerates the map computation by several factors.

6. Accelerating the computation of continuous maps

So far, our binary split method only works for maps with *label* values like classification functions $f : \mathbb{R}^n \rightarrow C$. However, several maps used in classifier visualization have continuous values, *i.e.*, are of the form $f : \mathbb{R}^n \rightarrow \mathbb{R}$. Examples are the gradient maps G (Eqn. 2), distance-to-closest-training sample d_D (Eqn. 4), and distance-to-decision boundary d_B (Eqn. 3). In general, one cannot reduce such continuous maps to the computation and comparison of purely categorical (label) values. Yet, we would like to accelerate their computation.

To do this, we generalize the binary split idea by replacing the label comparison (see Sec. 3) with a threshold comparison. For a dataset D, we compute this threshold globally as

$$T = \alpha \cdot \tau \cdot \left(\max_{\mathbf{p} \in \mathcal{B}} f(P^{-1}(\mathbf{p})) - \min_{\mathbf{p} \in \mathcal{B}} f(P^{-1}(\mathbf{p})) \right). \tag{8}$$

Simply put, T is a fraction of the range of the function f over the map. $\mathcal{B}$ denotes the set of center pixels of the initial B^2 blocks.

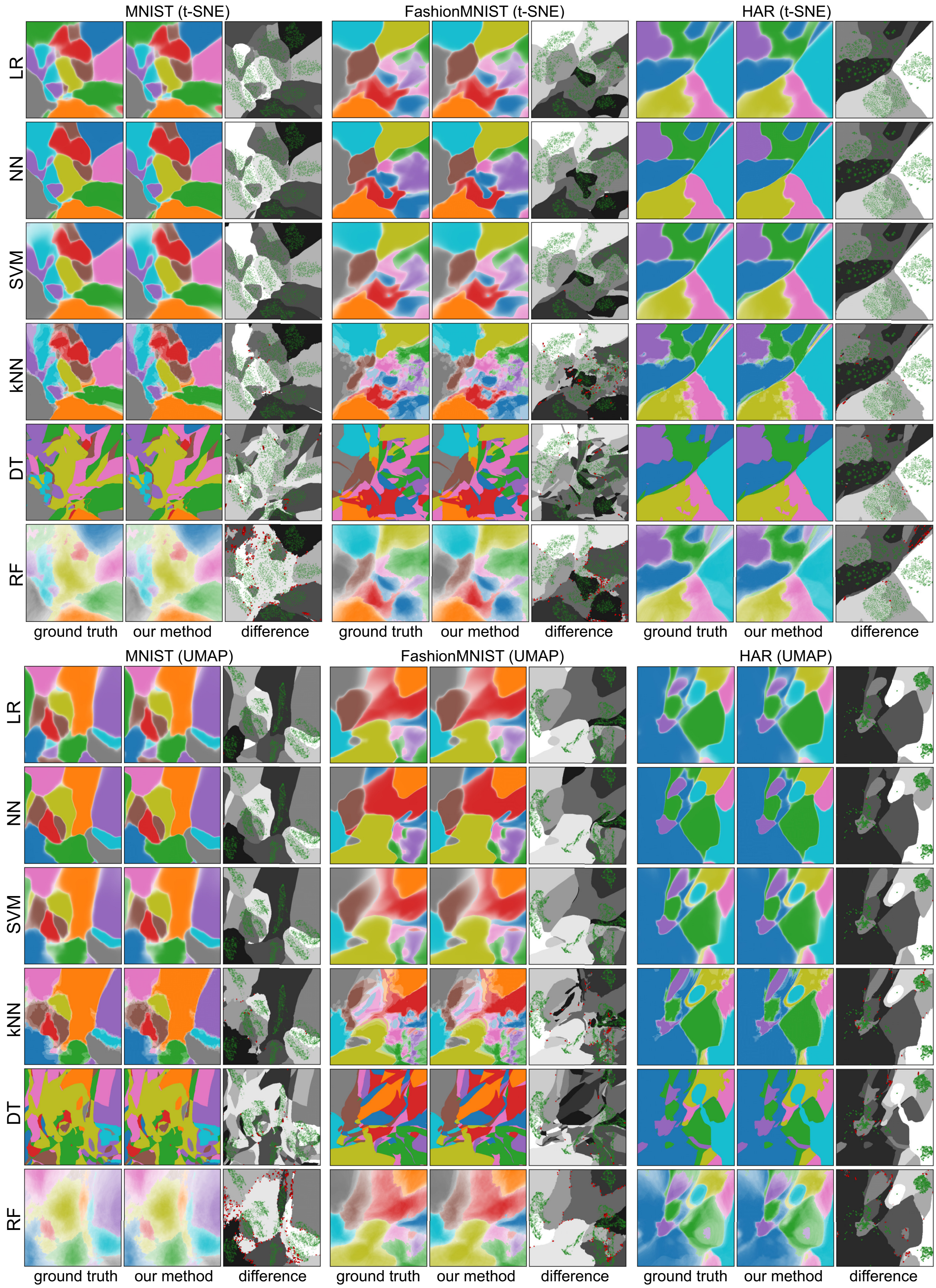

Figure 5: Comparison between ground-truth DBM and our binary split method for three datasets, six classifiers, t-SNE and UMAP projections.

Table 1: $Cons_p$ of FastDBM *vs* two ground truth methods for three datasets and six classifiers. $\Delta\ Cons_p = Cons_p^{fast} - Cons_p$. See Sec. 4.

(a) DBM (UMAP+NNInv)

Classifier	Metric	FashionMNIST	HAR	MNIST
DT	$Cons_p$	0.4033	0.3704	0.4718
	$Cons_p^{fast}$	0.4041	0.3659	0.4651
	$\Delta\ Cons_p$	0.0009	-0.0045	-0.0067
KNN	$Cons_p$	0.2152	0.0816	0.1414
	$Cons_p^{fast}$	0.2159	0.0735	0.1364
	$\Delta\ Cons_p$	0.0007	-0.0081	-0.0049
LR	$Cons_p$	0.2787	0.1776	0.2759
	$Cons_p^{fast}$	0.2792	0.1745	0.2727
	$\Delta\ Cons_p$	0.0005	-0.0031	-0.0032
NN	$Cons_p$	0.2959	0.1740	0.2810
	$Cons_p^{fast}$	0.2969	0.1712	0.2785
	$\Delta\ Cons_p$	0.0010	-0.0028	-0.0025
RF	$Cons_p$	0.2480	0.2477	0.3009
	$Cons_p^{fast}$	0.2464	0.2400	0.2955
	$\Delta\ Cons_p$	-0.0016	-0.0077	-0.0053
SVM	$Cons_p$	0.2153	0.1591	0.2470
	$Cons_p^{fast}$	0.2149	0.1551	0.2458
	$\Delta\ Cons_p$	-0.0004	-0.0039	-0.0013

(b) SDBM

Classifier	Metric	FashionMNIST	HAR	MNIST
DT	$Cons_p$	0.2685	0.1515	0.3594
	$Cons_p^{fast}$	0.2678	0.1510	0.3616
	$\Delta\ Cons_p$	-0.0007	-0.0005	0.0022
KNN	$Cons_p$	0.1145	0.0778	0.0950
	$Cons_p^{fast}$	0.1149	0.0767	0.0956
	$\Delta\ Cons_p$	0.0004	-0.0011	0.0007
LR	$Cons_p$	0.0589	0.0432	0.0909
	$Cons_p^{fast}$	0.0591	0.0431	0.0910
	$\Delta\ Cons_p$	0.0003	-0.0001	0.0002
NN	$Cons_p$	0.0725	0.0320	0.0872
	$Cons_p^{fast}$	0.0726	0.0321	0.0881
	$\Delta\ Cons_p$	0.0001	0.0001	0.0009
RF	$Cons_p$	0.1455	0.0655	0.2174
	$Cons_p^{fast}$	0.1456	0.0658	0.2178
	$\Delta\ Cons_p$	0.0001	0.0002	0.0004
SVM	$Cons_p$	0.0597	0.0364	0.0893
	$Cons_p^{fast}$	0.0603	0.0364	0.0898
	$\Delta\ Cons_p$	0.0006	-0.0000	0.0006

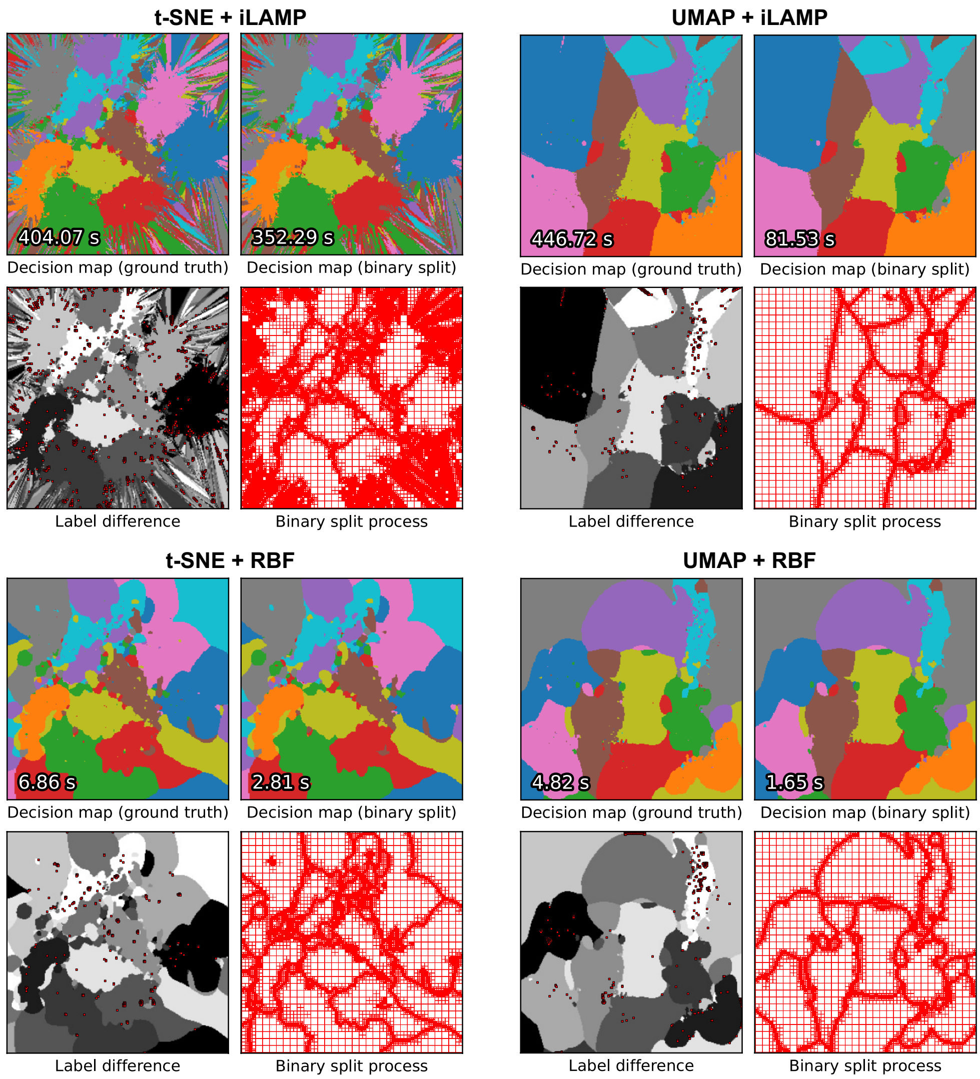

Figure 6: Maps computed using the iLAMP and RBF inverse projections in combination with the t-SNE and UMAP direct projections for the MNIST dataset. Resolution: 256^2. See Sec. 5.3.

$\tau = e^{-\frac{B \cdot d}{n}}$ is a decreasing function of the block size d, *i.e.*, smaller blocks will use a higher threshold. The intuition behind this is that smaller blocks already capture f at a higher resolution so we make them harder to further split to reduce over-refinement. Conversely, if f exhibits even a small variation over large blocks, this is a reason to split these to capture further details. The last parameter α is a scaling factor. When the difference between the maximum and minimum values of the four neighbors of a block including the block itself exceeds T, we split the block.

Distance-to-boundary maps which compute d_B (Eqn. 3) are a first example of such continuous maps. Figure 7 shows the results of accelerating the computation of d_B at resolution 512^2 pixels for the three datasets as in Fig. 5, and for all four inverse projection techniques we are aware of (NNInv, SSNP, RBF, and iLAMP) with UMAP as the direct projection. For all inverse projection techniques, we show the ground-truth d_B map, the map computed by our binary split acceleration, and the blocks created by the binary split process. Ground-truth maps are visually almost identical from those computed by our binary split heuristic. Speed-wise, our binary split heuristic is up to roughly ten times faster than computing the ground truth, see the figures in the bottom-left corners in the respective images in Fig. 7. This speed-up is in line with what we visually see as amounts of cells being split in the rightmost columns in Fig. 7 – the largest cells in those columns indicate the original block sizes, that is, using $B = 32$ initial cells for the acceleration heuristic, as explained earlier in Sec. 4.

Gradient maps G (Eqn. 2) are a second example of continuous maps we can accelerate. Figure 8 shows gradient maps computed by ground truth and our generalized binary split acceleration for the same dataset-projection-inverse projection combinations as shown in Fig. 7. Our accelerated maps are very similar to the ground truth, while the computation time is up to 10 times lower – see timing figures in the lower-left corners of the images.

Figure 9 shows a final example of continuous maps, namely distance-to-closest sample maps d_D (Eqn. 4). As for d_B and G, our acceleration yields practically the same images with high speed-ups *vs* ground truth. Given these results and the fact that our binary split works entirely agnostically on the nature of the function f, we claim that similar results can be obtained for *any* function $f : \mathbb{R}^n \to \mathbb{R}$ that produces a real value from an inversely-projected 2D pixel. The only implicit assumption our acceleration method makes for f is that it should be locally smooth, *i.e.*, not have unbounded variations on a small spatial extent, so that we can use the threshold computed by Eqn. 8 to locate map areas needing subdivision.

To find a suitable choice for α, we executed a grid search over the range $[0, 0.6]$ by evaluating both computation time and MSE error of our resulting map *vs* the ground-truth maps G, d_D, and d_B for the MNIST dataset. The MSE error is computed analogously to ϵ_{conf} (Eqn. 6). Figure 10 shows the search results. For larger α values, we get higher errors since the split threshold T is larger, so fewer block refinements (splits) occur; for smaller α values, the error decreases but the computational time increases, since there are more splits. We found that $\alpha \in [0.1, 0.2]$ is a good choice balancing between speed and accuracy. Specifically, we set $\alpha = 0.125$ for G, $\alpha = 0.1$ for d_D, and $\alpha = 0.15$ for d_B consistently in all our following experiments.

Figure 11 shows the computation times and normalized MSE errors of our generalized binary split method for different image sizes and for all the three maps d_D, G, and d_B. The results are quite similar with the binary split used for label-based maps (Fig. 3c): Our method is roughly linear in the map resolution (as compared to quadratic in resolution for the brute-force ground truth computation), while errors decrease inversely quadratically with resolution. All in all, the above results show us that the generalized binary split is a computationally effective and accurate way to accelerate the construction of continuous maps.

7. Discussion

We next discuss several aspects of our method.

Genericity: Our acceleration method based on the binary split can accommodate the construction of classifier maps for *any* function $f : \mathbb{R}^n \to \mathbb{R}$. This covers, but is not restricted to, the actual classification label F, gradient maps G, distance-to-boundary maps d_B, and distance-to-closest-training sample maps d_D. We can accelerate the computation of all such functions, while preserving their accuracy, in a black-box manner, *i.e.*, without knowing anything additional about what the respective functions capture or how they are computed. The only constraint we have is that such functions are smooth.

Performance: Our experiments showed that our binary split is roughly 5 times faster than the brute-force computation of the decision maps. This factor varies mainly as a function of the *smoothness* of the inverse projection method P^{-1} used: NNInv, SSNP, and RBF are relatively smooth mappings so fewer block splits are needed to capture their variation, which yields higher speed-ups. iLAMP is far less smooth so it requires more block splits, thereby reducing our speed-up to roughly 1 to 2 times. For the inverse projection methods NNInv and SSNP, which were earlier found to be the most reliable for computing decision maps, FastDBM is linear in the map resolution as compared to quadratic time for the brute-force computation.

A related point involves using GPU for further acceleration. Take a decision map algorithm which uses some projection P and inverse projection P^{-1}. Here, both, one of, or none of P and P^{-1} can use the GPU, depending on how these methods were desired by their creators. For example, considering P^{-1}, NNInv [30], a deep-learning method, uses the GPU; while iLAMP [22] does not. Our method accelerates the decision map construction *independently* on how P and P^{-1} work internally – in a nutshell, we reduce the number of times one needs to evaluate P^{-1} over a given map. Our acceleration does not currently use the GPU but works 'atop' a set of algorithms which themselves are CPU or GPU based. This leaves an interesting open opportunity of further accelerating our algorithm using the GPU – again, independently on whether P or P^{-1} are CPU or GPU based.

Quality: All our experiments showed that we can obtain the maps virtually identical visually to the ground-truth ones no matter which type of function we visualize. Moreover, the quality, measured in terms of normalized MSE *vs* ground truth, only increases with the image resolution. This means that our

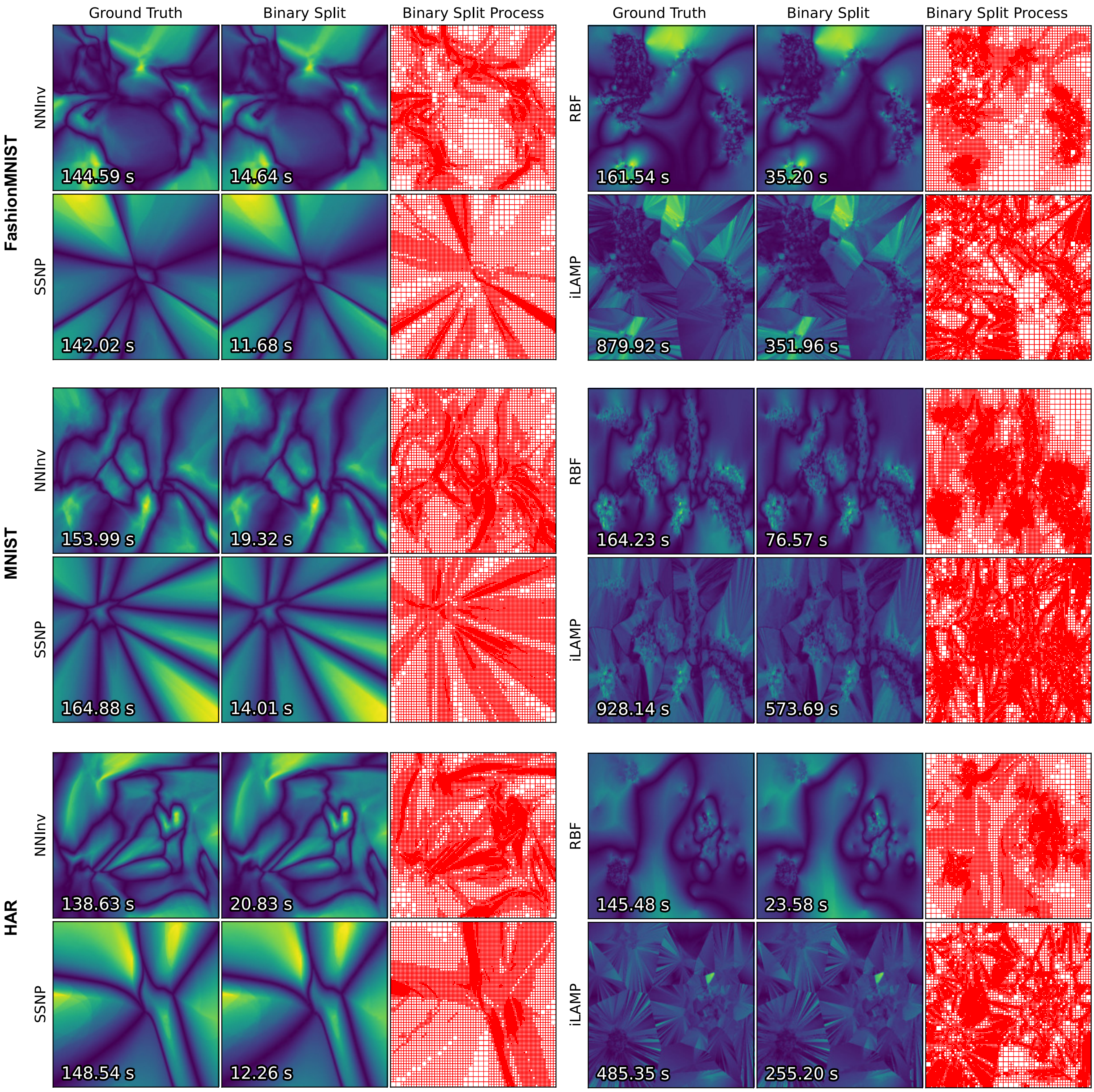

Figure 7: Distance-to-decision-boundary maps d_B for the generalized binary split method, three datasets (resolution: 512^2). See Sec. 6.

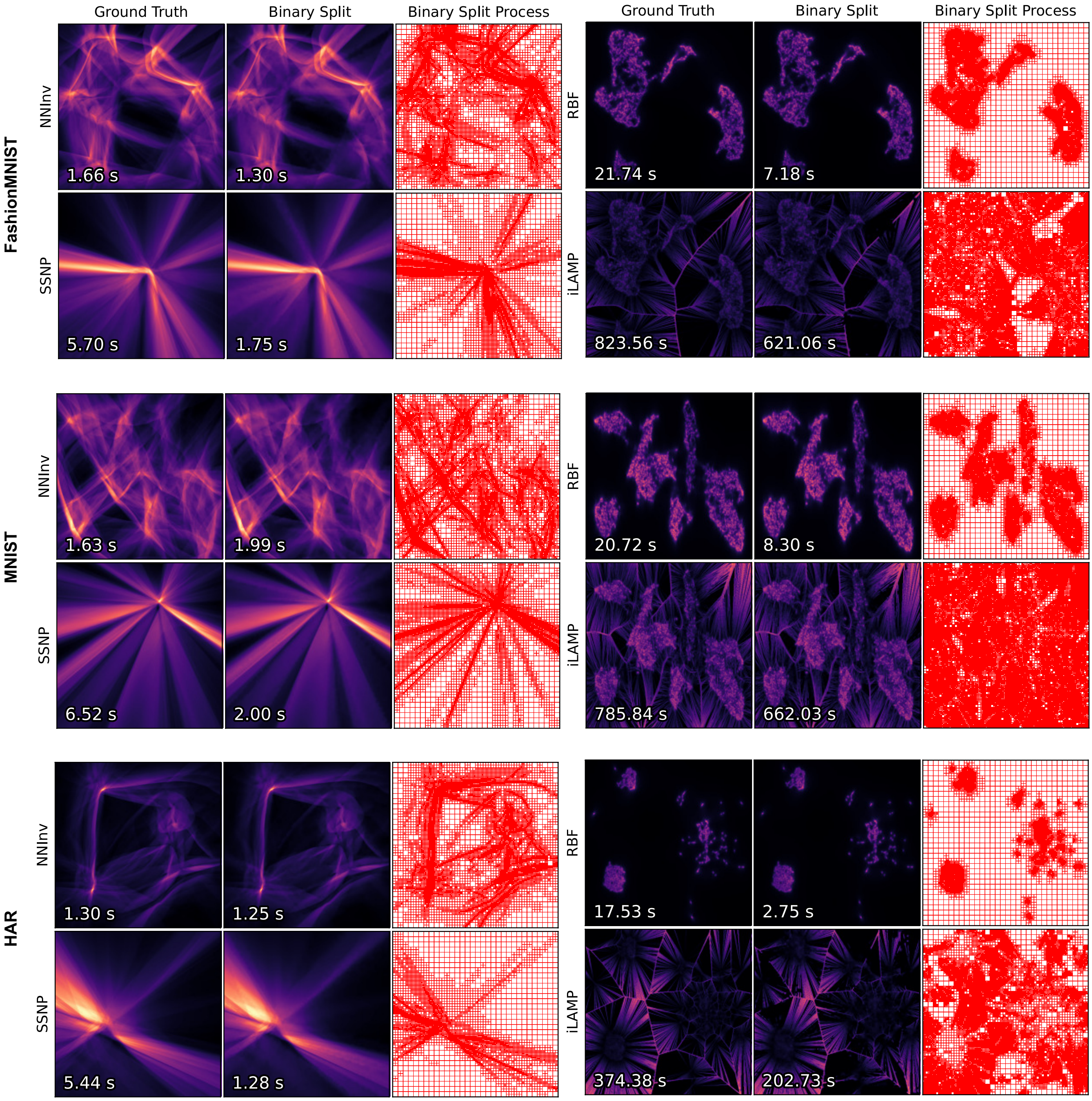

Figure 8: Gradient maps G for the generalized binary split, three datasets (resolution: 512^2). See Sec. 6.

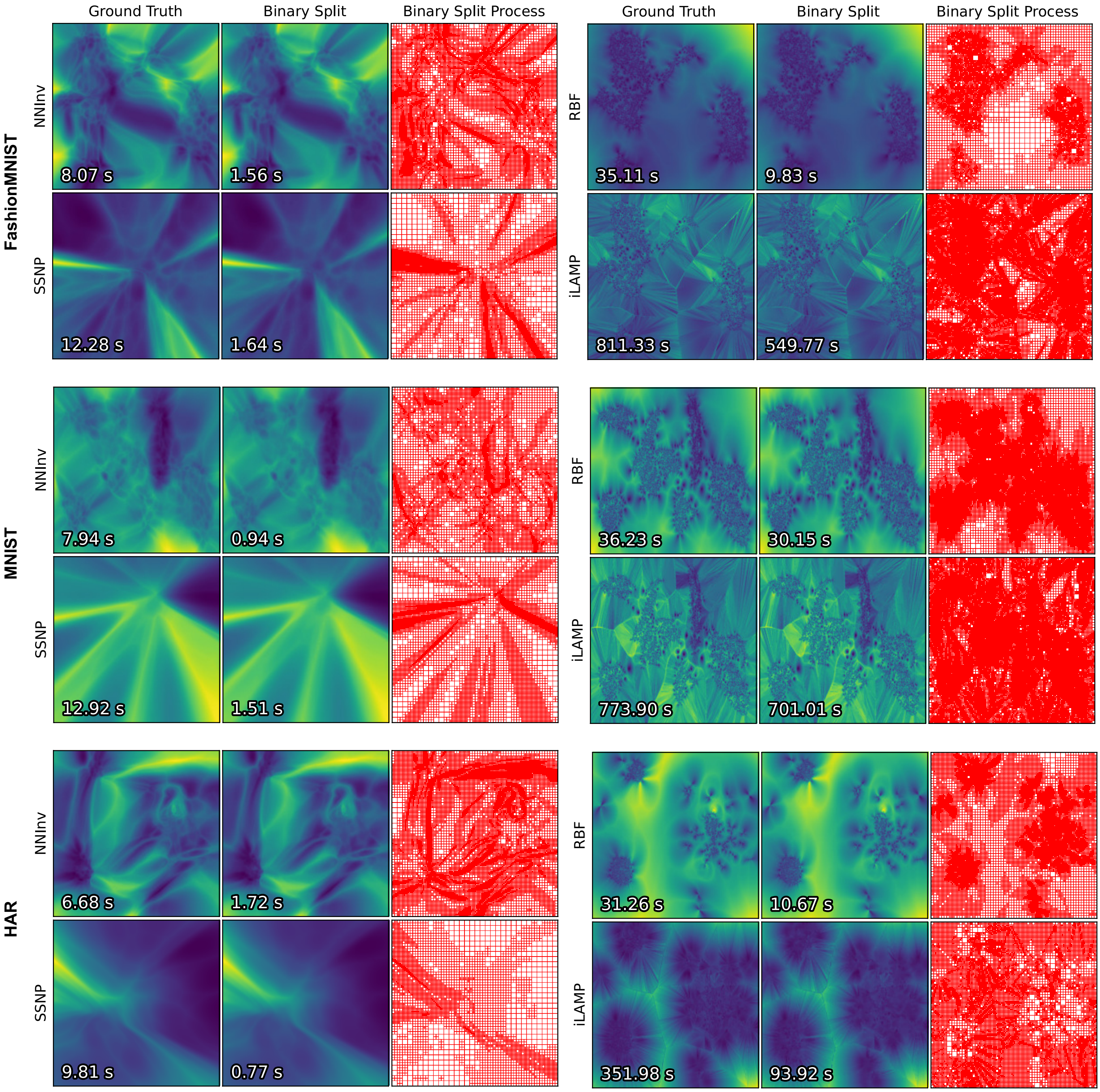

Figure 9: Distance to nearest sample maps d_D for the generalized binary split method, three datasets (resolution: 512^2). See Sec. 6.

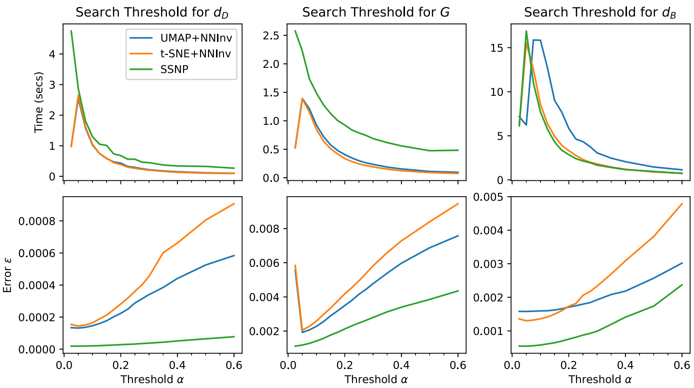

Figure 10: Generalized binary split method: Search for the best threshold α for the MNIST dataset. Columns show different classifier maps (d_D, G, d_B). Top row shows computation time. Bottom row shows computation error.

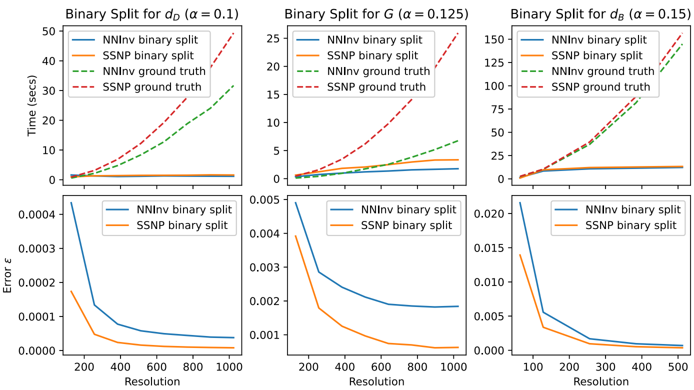

Figure 11: Performance of the generalized binary split method with varying grid resolutions for NNInv and SSNP on the MNIST dataset. Columns show different classifier maps (d_D, G, d_B) with optimized thresholds α. Top row shows computation time for binary split and ground truth methods. Bottom row shows the error as the image resolution increases.

accelerated maps can be safely substituted for the brute-force-computed ones for all practical reasons.

Ease of use: Our acceleration method is essentially parameter-free – the only two parameters B (initial block count, see Sec. 3.1) and α (controlling the split threshold for continuous mappings, see Eqn. 8) have well-tested presets which are independent on the classification model, choice of direct and inverse projections P and P^{-1}, and type of map being computed.

Limitations: The key assumption behind our acceleration is that the *combination* of function f we aim to visualize with the inverse projection function P^{-1} is smooth and has bounded variation over $\mathbb{R}^2$. While this is true of all f and P^{-1} we know of, and is also in line with the well-known smoothness assumption underlying most machine learning methods for f, that cases could exist where smoothness would not hold. In such cases, it is possible that our acceleration does not yield worthwhile speed-ups and/or the accelerated maps have visible errors as compared to the ground truth ones. Separately, we believe that the confidence split method (Sec. 3.2) has not yet reached its true potential. Better interpolation schemes than our current linear one should be able to decrease the number of generated cells and thereby achieve higher performance at the same quality level as compared to the so far currently best-ranked binary split. Another open challenge lies in scaling decision map visualizations to a large number of classes (*e.g.*, dozens or more). In that case, encoding class values in categorical colors will not work well. This limitation is broadly shared by many visualizations that use categorical color maps to encode class values. Potential solutions can group class values hierarchically to reduce the needed color count and offer detail-on-demand interactively. Note that this scalability problem only affects decision maps (which depict class value) and not the classifier maps (which depict real-valued quantities).

8. Conclusion

We have presented FastDBM, a technique for accelerating the computation of maps that describe the working of general-purpose classification models. Our technique is agnostic of the exact type of maps being computed as shown by its application to create maps of classification label, classification confidence, distance-to-classification-boundary, distance-to-closest-training sample, and gradient maps. Compared to earlier work [14], we show that our technique can be applied also to real-valued maps; and also show high speed-ups and accuracy for more combinations of direct and inverse projection methods used to compute the maps. Practically, we show that our method can compute classifier maps that are visually almost identical to ground-truth ones with a speed-up of one order of magnitude on average. This allows the further deployment of such visualizations in interactive visual analytics workflows for classifier engineering. Our method depends on just two free parameters for which we provide good preset values. Our method can accelerate any current classifier map computation technique, and can be applied to any trained classifier model, as it only requires access to the inverse projection function this technique uses, respectively to the black-box execution of the trained model.

Future work aims to explore our acceleration technique to compute additional classifier maps. Also, we consider speeding up our method by more advanced sampling and interpolation schemes, GPU execution of our block splitting scheme, and evaluating it on novel direct and inverse projection methods which arrive in the infovis arena. In parallel, measuring the added value of computing near-real-time classifier maps for classifier engineering, *e.g.*, in the context of visual active learning, is a key goal we aim at.

References

[1] A. Chatzimparmpas, R. M. Martins, I. Jusufi, A. Kerren, A survey of surveys on the use of visualization for interpreting machine learning models, Information Visualization 19 (3) (2020) 207–233.

[2] J. Yuan, C. Chen, W. Yang, M. Liu, J. Xia, S. Liu, A survey of visual analytics techniques for machine learning, Comp. Visual Media 7 (1) (2021) 3–36.

[3] D. Sacha, M. Kraus, D. A. Keim, M. Chen, VIS4ML: An Ontology for Visual Analytics Assisted Machine Learning, IEEE TVCG 25 (1) (2019) 385–395.

[4] A. Chatzimparmpas, K. Kucher, A. Kerren, Visualization for trust in machine learning revisited: The state of the field in 2023, IEEE Computer Graphics and Applications 44 (3) (2024) 99–113.

[5] A. Telea, A. Machado, Y. Wang, Seeing is Learning in High Dimensions: The Synergy Between Dimensionality Reduction and Machine Learning, SN Computer Science 5 (3) (2024) 279.

[6] M. Espadoto, G. Appleby, A. Suh, D. Cashman, M. Li, C. E. Scheidegger, E. W. Anderson, R. Chang, A. C. Telea, UnProjection: Leveraging Inverse-Projections for Visual Analytics of High-Dimensional Data, IEEE TVCG 29 (2) (2021) 1559–1572.

[7] A. Machado, M. Behrisch, A. Telea, Exploring classifiers with differentiable decision boundary maps, Computer Graphics Forum 43 (3) (2024) e15109.

[8] F. C. M. Rodrigues, R. Hirata, A. C. Telea, Image-based visualization of classifier decision boundaries, in: Proc. SIBGRAPI, IEEE, 2018, pp. 353–360.

[9] A. Schulz, F. Hinder, B. Hammer, DeepView: Visualizing Classification Boundaries of Deep Neural Networks as Scatter Plots Using Discriminative Dimensionality Reduction, in: Proc. IJCAI, 2020, pp. 2305–2311.

[10] A. A. A. M. Oliveira, M. Espadoto, R. Hirata Jr, A. C. Telea, SDBM: Supervised Decision Boundary Maps for Machine Learning Classifiers, in: Proc. IVAPP, 2022, pp. 77–87.

[11] Y. Wang, A. Machado, A. Telea, Quantitative and Qualitative Comparison of Decision Map Techniques for Explaining Classification Models, Algorithms 16 (9) (2023) 438.

[12] B. C. Benato, A. C. Telea, A. X. Falcao, Semi-Supervised Learning with Interactive Label Propagation Guided by Feature Space Projections, in: Proc. SIBGRAPI, IEEE, 2018, pp. 392–399.

[13] B. C. Benato, J. F. Gomes, A. C. Telea, A. X. Falcão, Semi-Automatic Data Annotation guided by Feature Space Projection, Pattern Recognition 109 (2021) 107612.

[14] C. Grosu, Y. Wang, A. Telea, Computing fast and accurate decision boundary maps, in: Proc. EuroVA, The Eurographics Association, 2024.

[15] L. van der Maaten, G. Hinton, Visualizing data using t-SNE, J. Mach. Learn. Res. 9 (11) (2008) 2579–2605.

[16] L. McInnes, J. Healy, J. Melville, UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction, arXiv:1802.03426 [cs, stat] (2018).
URL `http://arxiv.org/abs/1802.03426`

[17] I. Jolliffe, Principal component analysis, Springer, 2002.

[18] L. Nonato, M. Aupetit, Multidimensional projection for visual analytics: Linking techniques with distortions, tasks, and layout enrichment, IEEE TVCG 25 (2018) 2650–2673.

[19] M. Espadoto, R. Martins, A. Kerren, N. Hirata, A. Telea, Toward a quantitative survey of dimension reduction techniques, IEEE TVCG 27 (3) (2019) 2153–2173.

[20] F. C. M. Rodrigues, Visual Analytics for Machine Learning, PhD Thesis, University of Groningen (2020).

[21] B. C. Benato, C. Grosu, A. X. Falcão, A. C. Telea, Human-in-the-loop: Using classifier decision boundary maps to improve pseudo labels, Computers & Graphics 124 (2024) 104062.

[22] E. P. dos Santos Amorim, E. V. Brazil, J. Daniels, P. Joia, L. G. Nonato, M. C. Sousa, iLAMP: Exploring high-dimensional spacing through backward multidimensional projection, in: Proc. IEEE VAST, 2012, pp. 53–62.

[23] F. C. M. Rodrigues, M. Espadoto, R. Hirata, A. C. Telea, Constructing and Visualizing High-Quality Classifier Decision Boundary Maps, Information 10 (9) (2019) 280.

[24] Y. LeCun, C. Cortes, C. Burges, MNIST handwritten digit database, http://yann.lecun.com/exdb/mnist (2010).
URL `http://yann.lecun.com/exdb/mnist`

[25] M. Aupetit, Visualizing distortions and recovering topology in continuous projection techniques, Neurocomputing 10 (7–9) (2007) 1304–1330.

[26] S.-M. Moosavi-Dezfooli, A. Fawzi, P. Frossard, DeepFool: a simple and accurate method to fool deep neural networks, arXiv:1511.04599 [cs] (2016).
URL `http://arxiv.org/abs/1511.04599`

[27] R. Monarch, Human-in-the-Loop Machine Learning: Active Learning and Annotation for Human-Centered AI, Simon and Schuster, 2021.

[28] P. Joia, D. Coimbra, J. A. Cuminato, F. V. Paulovich, L. G. Nonato, Local affine multidimensional projection, IEEE TVCG 17 (12) (2011) 2563–2571.

[29] E. Amorim, E. Vital Brazil, J. Mena-Chalco, L. Velho, L. G. Nonato, F. Samavati, M. Costa Sousa, Facing the high-dimensions: Inverse projection with radial basis functions, Computers & Graphics 48 (2015) 35–47.

[30] M. Espadoto, F. C. M. Rodrigues, N. S. T. Hirata, R. Hirata Jr, Deep Learning Inverse Multidimensional Projections, in: Proc. EuroVA, The Eurographics Association, 2019.

[31] M. Espadoto, N. Hirata, A. Telea, Self-supervised Dimensionality Reduction with Neural Networks and Pseudo-labeling, in: Proc. IVAPP, SciTePress, 2021, pp. 27–37.

[32] G. E. Hinton, R. R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science 313 (5786) (2006) 504–507.

[33] Y. Wang, C. Grosu, A. Telea, Generalized FastDBM implementation source code (2025).
URL `https://github.com/yuwang-vis/generalized_fastDBM`

[34] H. Xiao, K. Rasul, R. Vollgraf, Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms, arXiv 1708.07747 [cs.LG] (2017).

[35] D. Anguita, A. Ghio, L. Oneto, X. Parra, J. L. Reyes-Ortiz, Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine, in: Proc. Intl. Workshop on ambient assisted living, Springer, 2012, pp. 216–223.

[36] R. A. Fisher, Iris Plants Database, UCI Machine Learning Repository (1988).