

GPEX, A Framework For Interpreting Artificial Neural Networks

Amir Akbarnejad, Gilbert Bigras, and Nilanjan Ray, *IEEE Member.*

Abstract—Machine learning researchers have long noted a trade-off between interpretability and prediction performance. On the one hand, traditional models are often interpretable to humans but they cannot achieve high prediction performances. At the opposite end of the spectrum, deep models can achieve state-of-the-art performances in many tasks. However, deep models' predictions are known to be uninterpretable to humans. In this paper we present a framework that shortens the gap between the two aforementioned groups of methods. Given an artificial neural network (ANN), our method finds a Gaussian process (GP) whose predictions almost match those of the ANN. As GPs are highly interpretable, we use the trained GP to explain the ANN's decisions. We use our method to explain ANNs' decisions on many datasets. The explanations provide intriguing insights about the ANNs' decisions. With the best of our knowledge, our inference formulation for GPs is the first one in which an ANN and a similarly behaving Gaussian process naturally appear. Furthermore, we examine some of the known theoretical conditions under which an ANN is interpretable by GPs. Some of those theoretical conditions are too restrictive for modern architectures. However, we hypothesize that only a subset of those theoretical conditions are sufficient. Finally, we implement our framework as a publicly available tool called GPEX. Given any pytorch feed-forward module, GPEX allows users to interpret any ANN subcomponent of the module effortlessly and without having to be involved in the inference algorithm. GPEX is publicly available online: www.github.com/Nilanjan-Ray/gpex

Index Terms—Machine Learning, Interpretability, Explainability, Artificial Neural Networks, Gaussian Processes.

1 INTRODUCTION

Artificial neural networks (ANNs) are widely adopted in machine learning. Their wide adoption is due to several reasons: theoretically proven expressive power [1] [2], the availability of tools and methods for training [3] [4] [5] [6] [7], and achieving state-of-the-art performance in several tasks [8] [9] [10]. Despite these benefits, ANNs are known to be black-box to humans, meaning that their inner mechanism for making predictions is not necessarily interpretable/explainable. ANN's black-box property impedes its deployment in safety-critical applications like medical imaging or autonomous driving. Moreover, the black-box property makes ANNs hard-to-troubleshoot for machine learning researchers. Therefore, interpreting/explaining ANNs has received a lot of interest recently [11] [12] [13] [14] [15] [16].

Explanation methods like LIME [13] and SHAP [14] consider an explainer model. Although the explainer model is encouraged to be faithful to the original model, actually it is way simpler than the model itself. Moreover, the explainer model is faithful only locally around an instance. Because of this "local assumptions", the explanations from LIME [13] and SHAP [14] might be unreliable and can be easily manipulated by an adversary model [17] [18].

Gradient-based explanation methods have been successful in providing explanations for ANNs. The simplest gradient-based method computes the gradient of the output

activation with respect to input features. This gradient is computed via the normal backpropagation procedure. More sophisticated gradient-based methods like DeepLIFT [19] modify the Jacobian matrices when doing backpropagation. Gradient-based methods are easily applicable to ANNs as they use the normal backpropagation procedure. However, similar to LIME [13] and SHAP [14] they implicitly presume a locally linear model. Among gradient-based methods, with the best of our knowledge only Integrated Gradients [20] has a weak sense of ANN's global behaviour over the feature space.

In this paper we are interested in explainer models which are globally faithful to the ANN. We opt the explainer model to be a Gaussian process (GP) [21]. Among Gaussian processes' elegant properties, here we are interested in two: 1. Gaussian processes are highly interpretable as they make predictions based on kernel similarity between a test instance and training instances. 2. Researchers have long known that large classes of ANNs are equivalent to GPs. More precisely, with some conditions on an ANN, there is a GP whose mean is globally faithful to the ANN [22].

To explain an ANN's decisions, we firstly find a Gaussian process that makes almost the same predictions as the ANN. Afterwards, given a test instance we find, e.g., 10 training instances which are closest to the test instance. This closeness is measured in terms of the GP's kernel-similarity function. For instance in Fig.1, the first column illustrates two test instances. In each row of Fig.1, 10 training instances which are the closest to the test instance (in terms of the GP's similarity function) are illustrated. Fig.1 demonstrates that the ANN has classified the first test image (i.e. the test image from a horse's head in column 1) as a horse, because it is similar to some training images from some horses'

A.Akbarnejad and N.Ray are with the Department of Computing Science, University of Alberta, Edmonton, AB, e-mail: (see <http://webdocs.cs.ualberta.ca/~nray1/contact.html>).

G.Bigras is with the Department of Laboratory Medicine & Pathology, University of Alberta, Edmonton, AB.

Manuscript received April TODO, 2021; revised August TODO, 2021.



Fig. 1: The first column depicts two testing instances. Columns 2-11 depict the 10 closest training instances to the test instance shown in column 1.

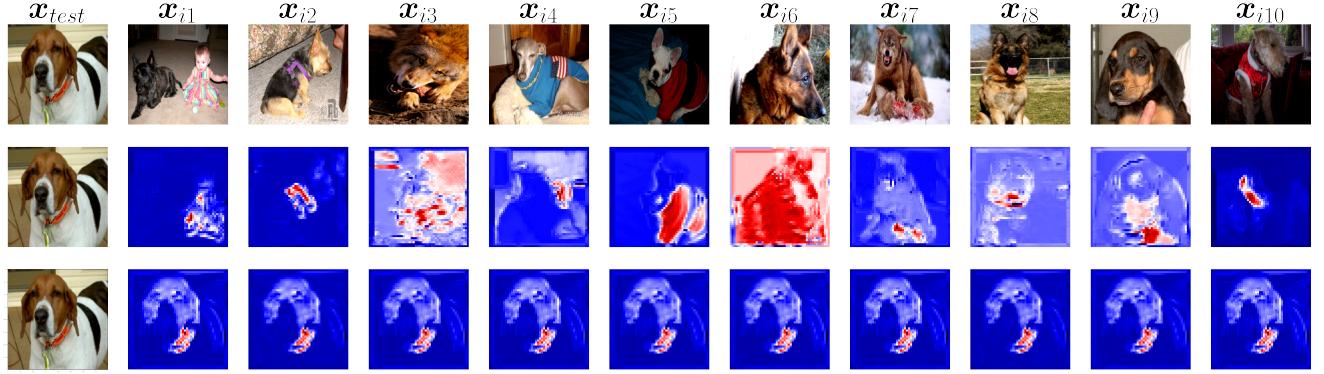


Fig. 2: For each pair of instances, we can measure to what degree each feature (in this case each pixel) contributes to the similarity between the two instances. The second row highlights the most contributing pixels of x_{ij} -s, while the third row highlights the most contributing pixels of x_{test} .

heads. However, the second row of Fig.1 demonstrates that the second test image (i.e. the image taken from faraway) is classified as horse because it is similar to some training images from horses which are also taken from faraway. Fig.1 shows that the ANN has had a correct reason to label the two test images as such.

Besides finding training instances similar to the testing set, we also provide explanations as to why those two instances are considered similar. Row 1 of Fig.2 illustrates a test instance as well as the 10 closest training instances. The heatmaps at the second (resp. third) row highlight the pixels from x_{ij} -s (resp. x_{test}) that contribute the most to the similarity between the testing instance and each training instance. Fig.2 demonstrates that the ANN has classified the test image as a dog by making use of clues like the dog collar, the baby next to the dog, human finger, etc. In this case our proposed GPEX shows that the ANN's decisions are not reliable, and the ANN has to be improved by, e.g., increasing the size of training set.

The contributions of this paper are as follows:

- We derive an evidence lower-bound (ELBO) in which a GP and an ANN are encouraged to behave similarly. Although approximate inference for GPs is well explored in literature, with the best of our knowledge we propose the first ELBO formulation in which an ANN and a similarly behaving GP naturally appear.
- In literature there are existing frameworks for applying GPs to image processing tasks. However, those frameworks either are not applicable to large datasets or are unable to use GPU acceleration. Our framework scales to datasets containing hundreds of

thousands of instances. Moreover, it makes use of GPU acceleration which is critical in deep learning. By doing so, we can train GPs whose outputs largely agree with the corresponding ANNs.

- Theoretical results on ANN-GP analogy impose some restrictions on ANNs under which the ANN will be equivalent to a GP. Some of these conditions are too restrictive for recently used deep architectures. In this paper we empirically show that an ANN is required to fulfill only a subset of those theoretical conditions.
- Having solved practical and computational issues, we propose a python library called GPEX (Gaussian Processes for EXplaining ANNs) that enables effortless application of GPs. GPEX can be used by machine learning researchers to interpret/troubleshoot their artificial neural networks. Moreover, GPEX can be used by researchers working on the theoretical side of ANN-GP analogy to empirically test their hypotheses.

2 RELATED WORK

2.1 Methods for Explaining Machine Learning Models

Given a test instance like x_{test} , LIME [13] interprets the ANN's decision $g(x_{test})$ by assuming that $g(\cdot)$ is locally linear around x_{test} and takes the local linear approximation as the explanation. Although the local model provides intriguing insights, the linear explainer might be too sensitive to small perturbations on x_{test} , because the ANN's decision boundary might be highly non-linear [17]. For those

methods the explanation may not reflect the ANN's internal mechanism as the linear explainer is way simpler than the ANN. Let x_{test} be an image in the test set containing M superpixels. The SHAP [14] framework explains the ANN's decision by computing to what degree any subset of superpixels contribute to the ANN's decision. To avoid considering all 2^M subsets, SHAP [14] assigns M values $[\phi_1, \dots, \phi_M]$ to superpixels. The ϕ_j -s are called Shapley values [14] and for any subset of superpixels like i_1, \dots, i_s the value $(\phi_{i1} + \dots + \phi_{is})$ is a good measure for the contribution of the superpixels i_1, \dots, i_s on the ANN's decision. Although the Shapley values are provably the optimal values for cooperative game theory [14], the machine learning setting is slightly different. For example, when some superpixels are excluded from an image, it is not clear what value(s) should fill-in the excluded pixels. More importantly, as SHAP [14] considers a local explainer model based on perturbed versions of an instance, its explanations are unreliable. For instance, a model (potentially an adversary model) may behave differently on the dataset instances and the perturbed ones [18].

The simple gradient method computes the gradient of output activation with respect to input pixels. In a different viewpoint, the importance of the last layer's neurons on the ANN's output is easily understood as the output is the weighted sum of the neurons in the last layer. Starting from the last layer, the simple gradient method relates the importance of the ℓ -th layer neurons to the importance of the $(\ell - 1)$ -th layer neurons until it reaches the input features. More sophisticated gradient-based methods like DeepLIFT [19] address the practical limitations of the simple-gradient method, and are shown to perform better. Gradient-based explanation methods use a backpropagation-like procedure, and therefore, they are easily applicable to ANNs. With the best of our knowledge the shortcomings of gradient-based methods is not empirically studied in literature. Nonetheless, an oft-said limitation is that a group of input pixels may have a negligible immediate effect (i.e. gradient) on output activations, but removing/adding those pixels simultaneously may have a large effect on output activations.

Influence functions has been used to explain machine learning models [15]. This method computes the effect of each training instance on the parameters of the trained model. It is prohibitively slow to discard each training instance and observe the effect of the instance on the model. Therefore, influence functions [15] efficiently computes the gradient of model parameters with respect to an instance's weight in the training loss. Influence functions [15] is similar to our approach in that it spots the most influential training instances. One issue with influence functions [15] is that the computed influence number deviates from the actual change in parameters when the model is retrained without the instance. Another closely-related explanation method is representer point selection [12]. Having mild conditions on an ANN, this method decomposes the decision $g(x_{test})$ to a weighted sum of similarities between x_{test} and training instances. One distinction between our kernel space and that of representer point selection [12] is that our kernel depends on all parameters of the ANN, whereas the kernel derived in representer point selection [12] depends on all parameters except the weights of the last layer.

2.2 Gaussian Processes

Gaussian process (GP) is a non-parametric model with elegant properties: it is interpretable, capable of modeling uncertainty, and it rarely overfits to training data. Training a GP is challenging specially because its kernel function interconnects all training instances. This interconnection makes the stochastic training (i.e. training using mini-batches) impossible because it causes the i.i.d assumption to be violated. There exist recent works for stochastic training in a correlated setting [23]. However, the common practice is to consider a set of instances called *inducing points* which parameterize the GP. Inducing points can be, e.g., a random subset of training instances. Incorporating the inducing points unties the interconnection of the whole training set and facilitates stochastic training. Like many other methods in literature [24] [25], we train a GP using inducing points.

Researchers have long known the close connection between Gaussian processes and artificial neural networks. The first theoretical connection was that under some conditions, a random single-layer neural network converges to the mean of a Gaussian process [26]. This connection was recently proven for ANNs with many layers [27]. Although the first discovered connections were only for ANNs with random parameters, more recent results hold even for ANNs trained with gradient descent [28]. Our proposed method is inspired by these theoretical results. However, we empirically show that only a subset of the theoretical conditions on ANNs are sufficient.

Several attempts have been made to adopt GPs for deep learning and image processing. For example, SV-DKL [24] derives a lower-bound for training a GP with a deep kernel. Although SV-DKL [24] is scalable, unfortunately it cannot make use of GPU acceleration. A more recent framework called GPYtorch [29] provides GPU acceleration. However, its computational complexity is quadratic in number of training instances which makes it prohibitively slow for large datasets. Neural tangents [22] is a python library based on GP-ANN analogy. It requires all layers (including the intermediate layers) to be infinitely wide. Afterwards, it computes the kernel of Gaussian processes layer by layer. Requiring all layers to be infinitely wide is too restrictive specially for recent deep models for image processing. In our framework we empirically show that it is sufficient to make only the last layer wide. We hypothesize that the batch-normalization layers [4] impose the finite-variance condition on neurons in intermediate layers, and therefore, the CLT-like (the central limit theorem like) theorem applies to the last wide layer. Moreover, neural tangents [22] presumes the model is trained on a dataset of fixed size. This assumption is often violated for image datasets because data augmentation is often applied during training. Unlike neural tangents [22], in our framework the dataset can be infinite and/or augmented.

3 PROPOSED METHOD

3.1 Notation

In this article the function $g(\cdot)$ always denotes an ANN. The kernel of a Gaussian process is denoted by the double-input function $\mathcal{K}(\cdot, \cdot)$. We assume the kernel similarity between

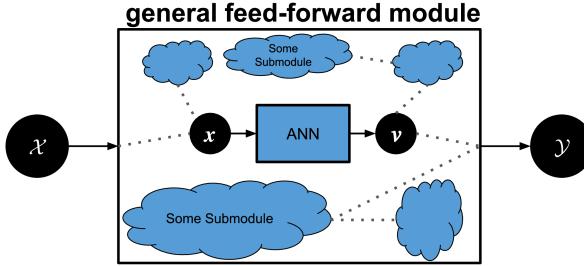


Fig. 3: Our framework takes in a general feed-forward pipeline with arbitrary submodules. The only requirement is that the pipeline must have at least one ANN submodule to be explained by Gaussian processes.

two instances x_i and x_j is equal to $f(x_i)^T f(x_j)$, where $f(\cdot)$ maps the feature-space to the kernel space. In this paper \mathbf{u} (resp. v) denotes a vector in the kernel-space (resp. the posterior mean) of a GP. In some sense \mathbf{u} and v denote the input and the output of a GP, respectively. We have that

$$\mathcal{K}(x_i, x_j) = f(x_i)^T f(x_j) = \mathbf{u}_i^T \mathbf{u}_j.$$

The number of GPs is equal to the number of the outputs from the ANN. In other words, we consider one GP per scalar output from the ANN. We use index ℓ to specify the ℓ -th GP as follows: $\mathcal{K}_\ell(x_i, x_j) = f_\ell(x_i)^T f_\ell(x_j) = \mathbf{u}_i^{(\ell)T} \mathbf{u}_j^{(\ell)}$. We parameterize the ℓ -th GP by a set of M inducing points $\{\tilde{\mathbf{u}}_m^{(\ell)}, \tilde{v}_m^{(\ell)}\}_{m=1}^M$. The tilde in $(\tilde{\mathbf{u}}_m^{(\ell)}, \tilde{v}_m^{(\ell)})$ indicates that $\tilde{\mathbf{u}}$ is one of the M inducing points in the kernel space. However, \mathbf{u} (without tilde) can be an arbitrary point in the continuous kernel space.

3.2 The Proposed Framework

To make our framework as general as possible, we consider a general feed-forward pipeline that contains an ANN as a submodule. In Fig.3 the bigger square illustrates the general module. The input-output of the general pipeline are denoted in Fig.3 by \mathcal{X} and \mathcal{Y} . The general pipeline has at least one ANN submodule to be explained by GPEX. Fig.3 illustrates this ANN by the small blue rectangle within the general pipeline. The input-output of the ANN are denoted in Fig.3 by x and y . Note that \mathcal{X} and \mathcal{Y} can be anything, including without any limitation, a set of vectors, labels, and meta-information. However, input-output of the ANN (i.e. x and y) are required to be in tensor format. The exact requirements are provided in the online documentation for GPEX. Moreover, the general module can have other arbitrary submodules, which are depicted by the blue clouds. The relations between the submodules, as illustrated by the dotted-lines in Fig. 3, can also be quite general. Our probabilistic formulation only needs access to the conditional distributions $p(\mathbf{x}|\mathcal{X})$ and $p(\mathcal{Y}|\mathbf{x}, \mathcal{X})$. Similarly, the proposed GPEX is completely agnostic about the general pipeline and it only requires the ANN's input-output to be in the tensor format. Given a PyTorch module, the proposed GPEX tool automatically grabs the distributions $p(\mathbf{x}|\mathcal{X})$ and $p(\mathcal{Y}|\mathbf{x}, \mathcal{X})$ from the main module it is given.

The inducing points $\{\tilde{\mathbf{u}}_m^{(\ell)}, \tilde{v}_m^{(\ell)}\}_{m=1}^M$ parameterize the ℓ -th GP. A feature point like x is first mapped to the kernel-space

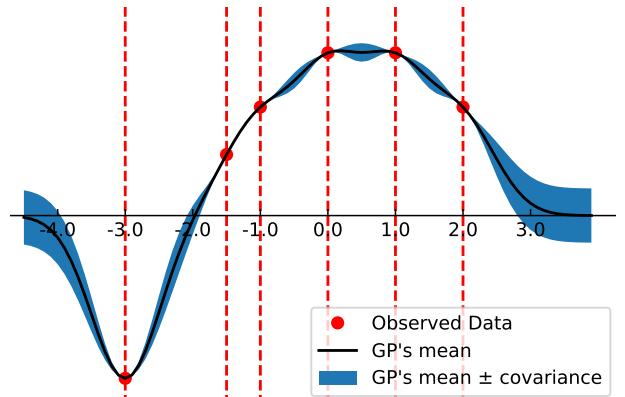


Fig. 4: Typical behaviour of Guassian process posterior given a set of observed values.

as $\mathbf{u}^{(\ell)} = f_\ell(x)$. Afterwards, the GP's output on x depends on the kernel similarities between $\mathbf{u}^{(\ell)}$ and the inducing points $\{\tilde{\mathbf{u}}_m^{(\ell)}\}_{m=1}^M$. More precisely, the posterior of the ℓ -th GP on x is a random variable $v^{(\ell)}$ whose distribution is as follows:

$$\begin{aligned} p(v^{(\ell)} | \mathbf{u}^{(\ell)}, \tilde{\mathbf{u}}_{1:M}^{(\ell)}, \tilde{v}_{1:M}^{(\ell)}) &= \\ \mathcal{N}\left(v^{(\ell)} ; \mu_v(\mathbf{u}^{(\ell)}, \tilde{\mathbf{u}}_{1:M}^{(\ell)}, \tilde{v}_{1:M}^{(\ell)}), cov_v(\mathbf{u}^{(\ell)}, \tilde{\mathbf{u}}_{1:M}^{(\ell)}, \tilde{v}_{1:M}^{(\ell)})\right), \end{aligned} \quad (1)$$

where $\mu_v(\cdot, \cdot, \cdot)$ and $cov_v(\cdot, \cdot, \cdot)$ are the mean and covariance of a GP's posterior computed as:

$$\begin{aligned} \mu_v(\mathbf{u}^{(\ell)}, \tilde{\mathbf{u}}_{1:M}^{(\ell)}, \tilde{v}_{1:M}^{(\ell)}) &= \\ \mathcal{K}(\mathbf{u}^{(\ell)}, \tilde{\mathbf{u}}_{1:M}^{(\ell)}) [\mathcal{K}(\tilde{\mathbf{u}}_{1:M}^{(\ell)}, \tilde{\mathbf{u}}_{1:M}^{(\ell)}) + \sigma_{gp}^2 \mathbf{I}_{M \times M}]^{-1} \tilde{v}_{1:M}^{(\ell)} \end{aligned} \quad (2)$$

and

$$\begin{aligned} cov_v(\mathbf{u}^{(\ell)}, \tilde{\mathbf{u}}_{1:M}^{(\ell)}, \tilde{v}_{1:M}^{(\ell)}) &= \mathcal{K}(\mathbf{u}^{(\ell)}, \mathbf{u}^{(\ell)}) - \\ \mathcal{K}(\mathbf{u}^{(\ell)}, \tilde{\mathbf{u}}_{1:M}^{(\ell)}) [\mathcal{K}(\tilde{\mathbf{u}}_{1:M}^{(\ell)}, \tilde{\mathbf{u}}_{1:M}^{(\ell)}) + \sigma_{gp}^2 \mathbf{I}_{M \times M}]^{-1} \mathcal{K}(\tilde{\mathbf{u}}_{1:M}^{(\ell)}, \mathbf{u}^{(\ell)}). \end{aligned} \quad (3)$$

As the variables $\{v_m^{(\ell)}\}_{m=1}^M$ and v are latent or hidden, we train the model parameters by optimizing a variational lower-bound. We consider the following variational distributions:

$$\begin{aligned} q_1(v^{(\ell)} | x) &= \mathcal{N}(v^{(\ell)} ; g_\ell(x), \sigma_g^2), \\ q_2(\tilde{v}_m^{(\ell)}) &= \mathcal{N}(\tilde{v}_m^{(\ell)} ; \varphi_m^{(\ell)}, \sigma_\varphi^2). \end{aligned} \quad (4)$$

In Eq.4, the function $g_\ell(\cdot)$ is the ℓ -th output from the ANN. Note that as the set of hidden variables $\{v_m^{(\ell)}\}_{m=1}^M$ is finite, we have parameterized their variational distribution by a finite set of numbers $\{\varphi_m^{(\ell)}\}_{m=1}^M$. However, as the variables x can vary arbitrarily in the feature space, the variable $\mathbf{u}^{(\ell)}$ varies arbitrarily in the kernel space. Therefore, the set of values $v^{(\ell)}$ may be infinite. Accordingly, the variational distribution for $v^{(\ell)}$ is conditioned on x and is parameterized by the ANN $g(\cdot)$.

3.3 The Derived Evidence Lower-Bound (ELBO)

The proposed evidence lower-bound (ELBO) is the main objective function for training both the Gaussian process and the ANN. Due to space limitations, the derivation of

the lower-bound is moved to Sec.S1 of the supplementary material. In this section we only introduce the derived ELBO and discuss how it relates the GP, the ANN and the training cost of the main module in an intuitive way. The ELBO terms containing the GP parameters (i.e. the parameters of the kernel function $f(\cdot)$) is denoted by \mathcal{L}_{gp} . According to Eq.S9 of the supplementary material \mathcal{L}_{gp} is as follows:

$$\begin{aligned} \mathcal{L}_{gp} = & -\frac{1}{2} \mathbb{E}_{\sim q} \left[\sum_{\ell=1}^L \frac{(\mu_v(\mathbf{u}^{(\ell)}, \tilde{\mathbf{u}}_{1:M}^{(\ell)}, \tilde{v}_{1:M}^{(\ell)}) - g_\ell(\mathbf{x}))^2 + \sigma_g^2}{\text{cov}_v(\mathbf{u}^{(\ell)}, \tilde{\mathbf{u}}_{1:M}^{(\ell)}, \tilde{v}_{1:M}^{(\ell)})} \right] \\ & -\frac{1}{2} \mathbb{E}_{\sim q} \left[\sum_{\ell=1}^L \log \left(\frac{\text{cov}_v(\mathbf{u}^{(\ell)}, \tilde{\mathbf{u}}_{1:M}^{(\ell)}, \tilde{v}_{1:M}^{(\ell)})}{\sigma_g^{2L}} \right) \right] \\ & + (\text{some constants}), \end{aligned} \quad (5)$$

where $q(\cdot)$ is the variational distribution that factorizes to the $q_1(\cdot)$ and $q_2(\cdot)$ distributions defined in Eq.4. In the first term of Eq.5, the nominator encourages the GP and the ANN to have the same output. More precisely, for a feature point \mathbf{x} we can compute the corresponding point in the kernel space as $\mathbf{u}^{(\ell)} = f_\ell(\mathbf{x})$ and then compute the GP's mean based on kernel similarities between \mathbf{u} and the inducing points to get the GP's mean μ_v . In Eq.5 the GP's mean μ_v is encouraged to match the ANN's output $g_\ell(\mathbf{x})$. In Eq.5, because of the denominator of the first term, the ANN-GP similarity is not encouraged uniformly over the feature-space. Wherever the GP's uncertainty is low, the term $\text{cov}_v(\mathbf{u}^{(\ell)}, \tilde{\mathbf{u}}_{1:M}^{(\ell)}, \tilde{v}_{1:M}^{(\ell)})$ in the denominator becomes small. Therefore, the GP's mean is highly encouraged to match the ANN's output. On the other hand, in regions where the GP's uncertainty is high, the GP-ANN analogy is less encouraged. This formulation is quite intuitive according to the behaviour of Gaussian processes. Fig.4 illustrates the posterior of a GP with radial-basis kernel for a given set of observations. In regions like $[3, \infty)$ and $(-\infty, -4]$ there are no nearby observed data. Therefore, in these regions the GP is highly uncertain and the blue uncertainty margin is thick in such regions. Intuitively, our derived ELBO in Eq.5 encourages the GP-ANN analogy only when GP's uncertainty is low and excludes regions similar to $[3, \infty)$ and $(-\infty, -4]$ in Fig.4. Note that this formulation makes no difference for the ANN as ANNs are known to be global approximators. However, this formulation makes a difference when training the GP, because the GP is not required to match the ANN in regions where there are no similar training instances. The ELBO terms containing the ANN parameters is denoted by \mathcal{L}_{ann} . According to Sec.S1.2 of the supplementary material, \mathcal{L}_{ann} is as follows:

$$\begin{aligned} \mathcal{L}_{ann} = & -\frac{1}{2} \mathbb{E}_{\sim q} \left[\sum_{\ell=1}^L \frac{(\mu_v(\mathbf{u}^{(\ell)}, \tilde{\mathbf{u}}_{1:M}^{(\ell)}, \tilde{v}_{1:M}^{(\ell)}) - g_\ell(\mathbf{x}))^2}{\text{cov}_v(\mathbf{u}^{(\ell)}, \tilde{\mathbf{u}}_{1:M}^{(\ell)}, \tilde{v}_{1:M}^{(\ell)})} \right] \\ & + \mathbb{E}_{\sim q} [\log p(\mathcal{Y}|\mathbf{y}, \mathcal{X})]. \end{aligned} \quad (6)$$

In the above objective the first term encourages the ANN to have the same output as the GP. Similar to the objective of Eq.5, the denominator of the first term gives more weight to ANN-GP analogy when GP's uncertainty is low. In the right-hand-side of Eq.6, the second term is the likelihood of the pipeline's output(s), i.e. \mathcal{Y} in Fig.3. This term can be,

e.g., the cross-entropy loss when \mathcal{Y} contains class scores in a classification problem, or the mean-squared error when \mathcal{Y} is the predicted value for a regression problem.

4 ALGORITHM

During training, to compute GP's posterior we firstly need to have the M inducing points $\{(\tilde{\mathbf{u}}_m^{(\ell)}, \tilde{v}_m^{(\ell)})\}_{m=1}^M$. It is computationally prohibitive to repeatedly update $\{\tilde{\mathbf{u}}_m^{(\ell)}\}_{m=1}^M$ by mapping all M images to the kernel space as $\tilde{\mathbf{u}}_m^{(\ell)} = f_\ell(\tilde{\mathbf{x}}_m)$. On the other hand, as the kernel-space mappings $\{f_\ell(\cdot)\}_{\ell=1}^L$ keep changing during training, we need to somehow track how the inducing points $\{\tilde{\mathbf{u}}_m^{(\ell)}\}_{m=1}^M$ change during training. To this end, we consider a matrix whose m -th row contains the value of $f_\ell(\tilde{\mathbf{x}}_m)$ at some point during training, where $\tilde{\mathbf{x}}_m$ is the m -th inducing point or image. During training, we keep updating the rows of this matrix by feeding mini-batches of images to $f_\ell(\cdot)$. Note that we have as many GPs as the number of ANN's output heads. Therefore, for each GP we consider a separate matrix containing the representations of the inducing images in the ℓ -th kernel space. In Algs.1, 2, 3, and 4 the variable \mathbf{U} is a list containing all of the the aforementioned matrices. To explain a given ANN, we let the ANN to be fixed and we only train the GPs' parameters. This procedure is explained in Alg.4. In each iteration, the kernel-mappings are updated according to the objective function of Eq.5 (line 3 of Alg.4). Afterwards, to make the matrices in \mathbf{U} track the changes in $[f_1(\cdot), \dots, f_L(\cdot)]$, we map an inducing image (or a mini-batch of inducing images) to the kernel spaces, and we update the corresponding matrices and rows in \mathbf{U} according to the newly obtained kernel-space representations. Updating \mathbf{U} is done in line 5 of Alg.4. The method in Alg.1 computes the GPs' posterior means and covariances at any image like \mathbf{x} , given the observed inducing points as specified by \mathbf{U} and \mathbf{V} . Note that this method returns two outputs, because a GP's posterior at \mathbf{x} is a normal distribution described by its mean and variance. In Alg.1 lines 8 and 9 correspond to the equations of GP posterior (i.e. Eqs. 2 and 3). The method in Alg.1 is used both during training and testing. During training, this method is called whenever ANN's output and GP's posterior are encouraged to be close. During training, according to line 6 of Alg.1 only the matrix row(s) corresponding to the fed inducing image(s) are the result of mapping the inducing image(s) via the kernel-mapping, and all other rows are kept fixed. Line 6 of Alg.1 allows for computing the gradient of loss with respect to kernel-mappings $[f_1(\cdot), \dots, f_L(\cdot)]$. During testing we call Alg.1 to get the GP's posterior at a test instance like \mathbf{x}_{test} . Alg.3 initializes the GP parameters \mathbf{U} and \mathbf{V} . For the ℓ -th GP, the vector $\mathbf{V}[\ell]$ is initialized to the ℓ -th output head of the ANN at all inducing images. In Alg.3, the vector $\mathbf{V}[\ell]$ is initialized in line 2. Moreover, for the ℓ -th GP the matrix $\mathbf{U}[\ell]$ is initialized by mapping all inducing images to the ℓ -th kernel-space via the mapping $f_\ell(\cdot)$. In Alg.3 the matrix $\mathbf{U}[\ell]$ is initialized in line 5. The method in Alg.3 is called only once before training the GP. For instance, when explaining an ANN in Alg.4, the initialisation is done once at the beginning of the procedure.

Algorithm 1 Method Forward_GP

Input: Images \mathbf{x} and $\tilde{\mathbf{x}}$, list of matrices \mathbf{U} , list of vectors \mathbf{V} .
Output: List of GP posterior means μ , and covariances cov .

Initialisation : $\mu = list(L)$, $cov = list(L)$.

- 1: **for** $\ell = 1$ to L **do**
- 2: $u = f_\ell(\mathbf{x})$ //map \mathbf{x} to the kernel space of the ℓ -th GP.
- 3: $\mathbf{U}_\ell \leftarrow \mathbf{U}[L]$ //get the inducing points of the ℓ -th GP.
- 4: $\mathbf{V}_\ell \leftarrow \mathbf{V}[L]$ //observed values at the inducing points.
- 5: **if** training **then**
- 6: $\mathbf{U}_\ell[\tilde{\mathbf{x}}.\text{index}] \leftarrow f_\ell(\tilde{\mathbf{x}})$ //to pass gradient w.r.t. $f_\ell(\cdot)$
- 7: **end if**
- 8: $\mu[\ell] \leftarrow \mathbf{u}^T \mathbf{U}_\ell^T (\mathbf{U}_\ell \mathbf{U}_\ell^T + \sigma_{gp}^2 \mathbf{I})^{-1} \mathbf{V}_\ell$.
- 9: $cov[\ell] \leftarrow \mathbf{u}^T \mathbf{u} - \mathbf{u}^T \mathbf{U}_\ell^T (\mathbf{U}_\ell \mathbf{U}_\ell^T + \sigma_{gp}^2 \mathbf{I})^{-1} \mathbf{U}_\ell \mathbf{U}_\ell^T$.
- 10: **end for**
- 11: **return** μ and cov

Algorithm 2 Method Optim_KernMappings

Input: Images \mathbf{x} and $\tilde{\mathbf{x}}$, list of matrices \mathbf{U} , list of vectors \mathbf{V} .
Output: Kernel-space mappings $[f_1(\cdot), \dots, f_L(\cdot)]$.

Initialisation : $loss \leftarrow 0$.

- 1: $\mu, cov \leftarrow \text{forward_GP}(\mathbf{x}, \tilde{\mathbf{x}}, \mathbf{U}, \mathbf{V})$ //feed \mathbf{x} to GPs.
- 2: $\mu_{ann} \leftarrow g(\mathbf{x})$ //feed \mathbf{x} to ANN.
- 3: **for** $\ell = 1$ to L **do**
- 4: $loss \leftarrow loss + \frac{(\mu[\ell] - \mu_{ann}[\ell])^2}{cov[\ell]} + \log(cov[\ell])$. //Eq. 5.
- 5: **end for**
- 6: $\delta \leftarrow \frac{\partial loss}{\partial \text{params}([f_1(\cdot), \dots, f_L(\cdot)])}$ //the gradient of loss.
- 7: $\text{params}([f_1, \dots, f_L]) \leftarrow \text{params}([f_1, \dots, f_L]) - lr \times \delta$ //update the parameters.
- 8: $lr \leftarrow$ updated learning rate
- 9: **return** $[f_1(\cdot), \dots, f_L(\cdot)]$

4.1 Efficiently Computing Gaussian Process Posterior

One difficulty in training GPs is the matrix inversion of Eqs.2 and 3, as done in lines 8 and 9 of Alg.1. To address this issue, we adopted computational techniques recently used for fast spectral clustering [30]. These computational techniques allow us to efficiently compute the GP-posterior for hundreds of thousands of inducing points in each training iteration. Let \mathbf{A} be an arbitrary $M \times D$ matrix where $M \gg D$. Moreover, let \mathbf{b} be a M -dimensional vector and let σ be a scalar. The computational techniques [30] allow us to efficiently compute: $(\mathbf{A}\mathbf{A}^T + \sigma^2 \mathbf{I}_{M \times M})^{-1} \mathbf{b}$. The idea is that $\mathbf{A}\mathbf{A}^T$ and therefore its inverse are of rank D . Therefore, one can do the computations efficiently in the space of D eigenvectors that correspond to non-zero eigenvalues. Further details are provided in supplementary material in Sec.S2. The pseudo-code is provided in Alg.5.

4.2 Computing Feature Contributions to the Similarity

Besides finding similar training instances to the test instance, given two instances \mathbf{x}_1 and \mathbf{x}_2 we can measure to what degree each feature or pixel contributes to the similarity $\mathcal{K}(\mathbf{x}_1, \mathbf{x}_2)$, as introduced in rows 2 and 3 of Fig.2. For images, an idea similar to class activation maps (CAM) [31] is applicable. To this end, the kernel mappings

Algorithm 3 Method Init_GPparams

Input: Dataset of inducing images $[\tilde{\mathbf{x}}_1, \dots, \tilde{\mathbf{x}}_M]$.
Output: List of matrices \mathbf{U} , list of vectors \mathbf{V} .

Initialisation : $\mathbf{U} = list(L)$, $\mathbf{V} = list(L)$.

- 1: **for** $\ell = 1$ to L **do**
- 2: $\mathbf{V}[\ell] \leftarrow [g(\tilde{\mathbf{x}}_1)[\ell], \dots, g(\tilde{\mathbf{x}}_M)[\ell]]$.
- 3: **end for**
- 4: **for** $\ell = 1$ to L **do**
- 5: $\mathbf{U}[\ell] \leftarrow [f_\ell(\tilde{\mathbf{x}}_1), \dots, f_\ell(\tilde{\mathbf{x}}_M)]$.
- 6: **end for**
- 7: **return** \mathbf{U} and \mathbf{V}

Algorithm 4 Method Explain ANN

Input: Training dataset ds_train , and the inducing dataset $ds_inducing$.

Output: Kernel-space mappings $[f_1(\cdot), \dots, f_L(\cdot)]$, and the other GP parameters \mathbf{U} and \mathbf{V} .

Initialisation : $\mathbf{U}, \mathbf{V} \leftarrow \text{Init_GPparams}(ds_inducing)$.

- 1: **for** $iter = 1$ to max_iter **do**
- 2: $\mathbf{x} \leftarrow \text{randselect}(ds_train)$.
- 3: $\tilde{\mathbf{x}} \leftarrow \text{randselect}(ds_inducing)$
- 4: $[f_1(\cdot), \dots, f_L(\cdot)] \leftarrow \text{Optim_KernMapings}(\mathbf{x}, \tilde{\mathbf{x}}, \mathbf{U}, \mathbf{V})$.
- 5: $\tilde{\mathbf{x}} \leftarrow \text{randselect}(ds_inducing)$.
- 6: **for** $\ell = 1$ to L **do**
- 7: //update kernel-space representations.
- 8: $\mathbf{U}[\ell][\tilde{\mathbf{x}}.\text{index}] \leftarrow f_\ell(\tilde{\mathbf{x}})$
- 9: **end for**
- 10: **end for**
- 11: **return** $[f_1(\cdot), \dots, f_L(\cdot)], \mathbf{U}, \mathbf{V}$

$[f_1(\cdot), \dots, f_L(\cdot)]$ should be convolutional neural networks that produce volumetric maps followed by spatial average pooling. However, the kernel-mappings that we used have a slightly different architecture and the original CAM [31] formulation should be adjusted. In Sec.S3 of supplementary material the details are elaborated upon.

5 EXPERIMENTS

We conducted several experiments on four publicly available datasets: MNIST [32], Cifar10 [33], Kather [34], and DogsWolves [35]. MNIST [32] and Cifar10 [33] are famous datasets for digit classification and object classification, respectively. Kather dataset [34] contains 100,000 microscopic images from hematoxylin & eosin (H&E) stained samples from human colon tissue. Finally, DogsWolves dataset [35] contains 1000 images from dogs and 1000 images from wolves, and the task is to classify each image as either dog or wolf. For MNIST [32] and Cifar10 [33] we used the standard split to training and test sets provided by the datasets. For Kather [34] and DogsWolves [35] we randomly selected 70% and 80% of instances as our training set. Training GPEX involves several details that we have not discussed yet in this article. Therefore, we firstly discuss the experimental results in Secs. 5.1, 5.2, and 5.3. Afterwards, in supplementary material in Sec.S4 we elaborate upon the training details and the parameter settings in each of our experiments. Note that we trained the ANNs as usual rather than using Eq.6,

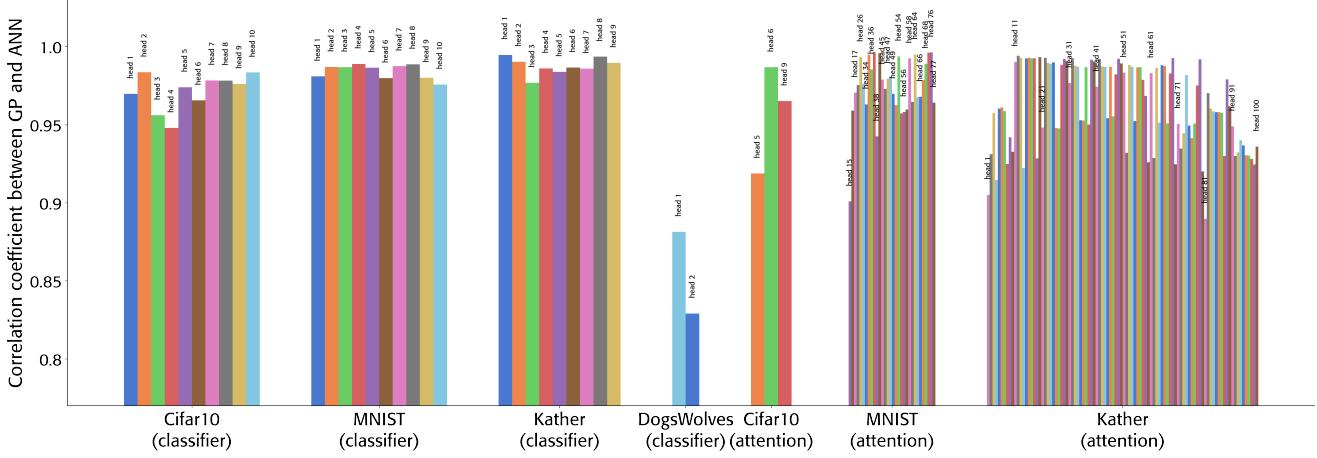


Fig. 5: Faithfulness of GPs to ANNs measured by the Pearson correlation coefficient.

Algorithm 5 Method Efficiently_Compute_AATinvb

Input: Matrix \mathbf{A} of size $M \times D$, vector \mathbf{b} of size $M \times 1$, and positive scalar σ .
Output: The vector $\text{output} = (\mathbf{A}\mathbf{A}^T + \sigma^2\mathbf{I})^{-1}\mathbf{b}$.

- 1: $\tilde{\mathbf{E}}, \tilde{\boldsymbol{\lambda}} \leftarrow \text{eigendecom}(\mathbf{A}^T\mathbf{A} + \sigma^2\mathbf{I})$.
- 2: $[\tilde{\mathbf{e}}_1, \dots, \tilde{\mathbf{e}}_D] \leftarrow \tilde{\mathbf{E}}$
- 3: $[\tilde{\lambda}_1, \dots, \tilde{\lambda}_D] \leftarrow \tilde{\boldsymbol{\lambda}}$
- 4: $[\mathbf{e}_1, \dots, \mathbf{e}_D] \leftarrow [\mathbf{A}\tilde{\mathbf{e}}_1, \dots, \mathbf{A}\tilde{\mathbf{e}}_D]$
- 5: $[\lambda_1, \dots, \lambda_D] \leftarrow [\tilde{\lambda}_1, \dots, \tilde{\lambda}_D]$
- 6: $\mathbf{E} \leftarrow [\mathbf{e}_1, \dots, \mathbf{e}_D]$
- 7: $\mathbf{\Lambda} \leftarrow \text{diagonal}(\frac{1}{\lambda_1 + \sigma^2}, \dots, \frac{1}{\lambda_D + \sigma^2})$
- 8: $\text{output} \leftarrow \mathbf{E}\mathbf{\Lambda}\mathbf{E}^T\mathbf{b} + \frac{1}{\sigma^2}(\mathbf{b} - \mathbf{E}\mathbf{E}^T\mathbf{b})$ //according to //Eq.S21 in supplementary material
- 9: **return** output

because our proposed GPEX should be applicable to ANNs which are trained as usual.

5.1 Measuring Faithfulness of GPs to ANNs

We trained a separate convolutional neural network (CNN) on each dataset to perform the classification task. For MNIST [32], Cifar10 [33], and Kather [34] we used a ResNet-18 [36] backbone followed by some fully connected layers. DogsWolves [35] is a relatively small dataset, and very deep architectures like ResNet-18 [36] overfit to training set. Therefore, we used a convolutional backbone which is suggested in the dataset website [35]. For all datasets, we set the width (i.e. the number of neurons) of the second last fully-connected layer to 1024. Because according to theoretical results on GP-ANN analogy, the second last layer of ANN is required to be wide. We used an implementation of ResNet [36] which is publicly available online [37]. We trained the pipelines for 20, 200, 20, and 20 epochs on MNIST [32], Cifar10 [33], Kather [34], and DogsWolves [35], respectively. For Cifar10 [33], we used the exact optimizer suggested by [37]. For other datasets we used an Adam [38] optimizer with a learning-rate of 0.0001. The test accuracies of the models are equal to 99.56%, 95.43%, 96.80%, and 80.50% on MNIST [32], Cifar10 [33], Kather [34], and DogsWolves [35], respectively.

We explained each classifier CNN using our proposed GPEX framework (i.e. Alg.4). As discussed in Sec.4, given an ANN we have as many kernel-spaces (and as many GPs) as the number of ANN's output heads. The exact parameter settings and practical considerations for training the GPs is elaborated upon in Sec.S4 of the supplementary material. To measure the faithfulness of GPs to ANNs, we compute the Pearson correlation coefficient for each ANN head and the mean of the corresponding GP posterior. The results are provided in Fig.5. In Fig.5, the first four groups of bars (i.e. the groups labeled as Cifar10 (classifier), MNIST (classifier), Kather (classifier), and DogsWolves (classifier)) correspond to applying the proposed GPEX on the four classifier CNNs trained on the four datasets. Note that within each group of bars, for each ANN head and the corresponding GP we have included a separate bar whose height is equal to the correlation coefficient between the ANN head and the corresponding GP. According to Fig.5, our trained GPs almost perfectly match the corresponding ANNs. Only for DogsWolves [35], as illustrated by the 4-th bar group in Fig.5, the correlation coefficients are lower compared to other datasets. We hypothesize that this is because the DogsWolves dataset [35] has very few images. GP posterior mean can be changed only by moving the inducing points in the kernel-space. Therefore, when very few inducing points are available GP posterior mean is less flexible. This is consistent with our parameter analysis of Fig.10, and explains the lower correlation coefficients for the DogsWolves dataset [35] in Fig.5. In supplementary material, the scatter plots in Figs.S2, S4, and S6 illustrate the faithfulness of GPs to ANNs. In Figs.S2, S3, and S4 of the supplementary material each plot corresponds to a specific head of an ANN.

In Fig.3 we discussed that our proposed GPEX is not only able to explain a classifier ANN, but it can explain any ANN which is a subcomponent of any feed-forward pipeline. To evaluate this ability, we trained three classifiers with an attention mechanism [39]. Each classifier has two ResNet-18 [36] backbones: one extracts a volumetric map containing deep features, and the other produces a spatial attention mask. The attention mask is multiplied the extracted deep features to produce a masked volumetric map.

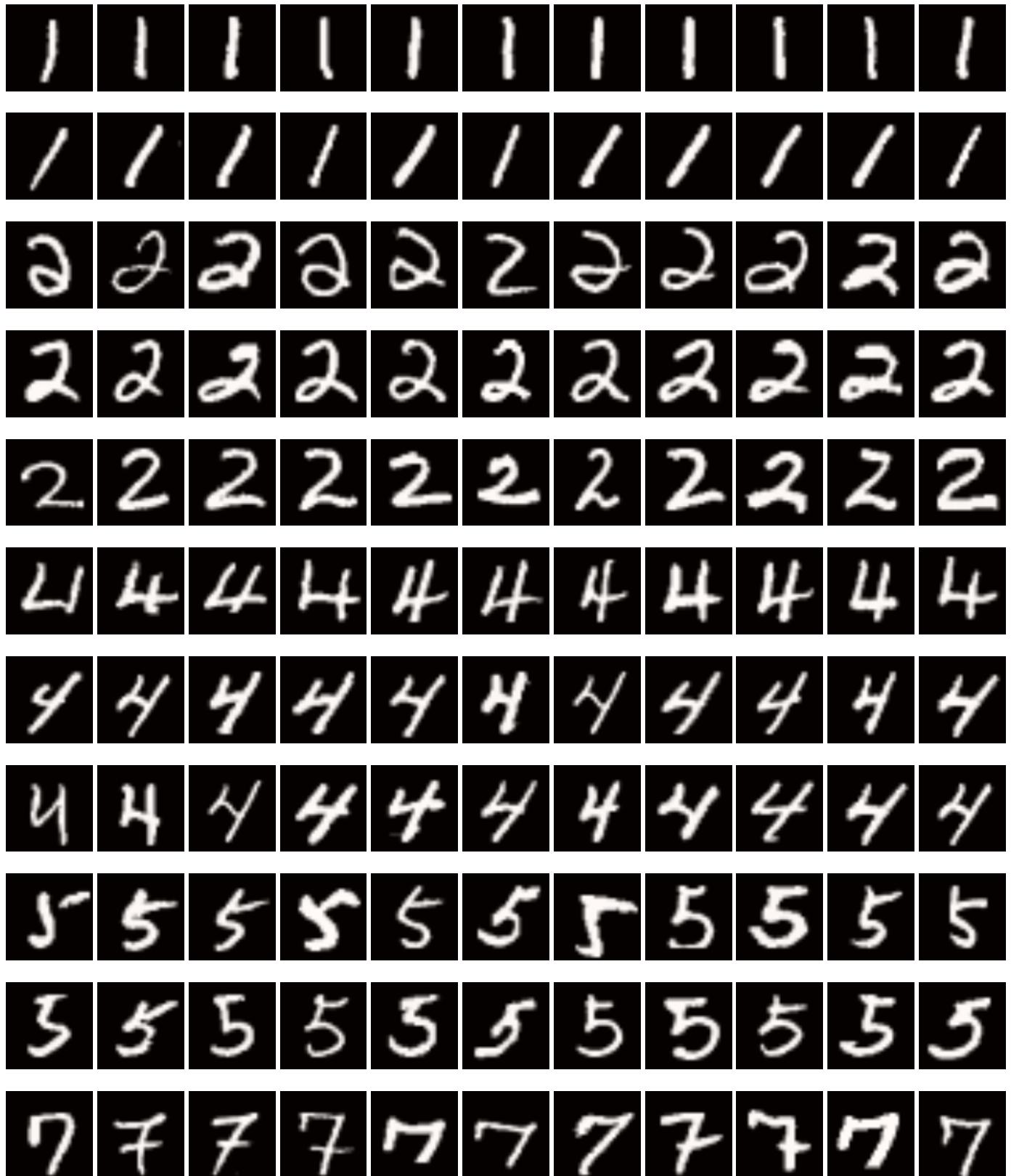


Fig. 6: Sample explanations for MNIST (set 1).



Fig. 7: Sample explanations for MNIST (set 2).

Afterwards, this masked volumetric map is fed to spatial pooling and linear layers to produce class activations. For each attention backbone, we set the width of the second last layer to 1024. We add a sigmoid activation function at the end of each attention backbone, so as to make the values of the attention masks between 0 and 1. To see whether our proposed GPEX can find GPs which are faithful to the attention backbones, we applied Alg.4 to each classifier, but this time the ANN to be explained (i.e. the box called "ANN" in Fig.3) is set to be the attention submodule. Note that each attention backbone produces a spatial attention mask of size h by w . We think of each attention backbone as an ANN which has $h \times w$ output heads. We trained three classifier pipelines with attention mechanism on Cifar10 [33], MNIST [32], and Kather [34]. We used the same training procedure that we used for the four classifier CNNs in previous part. In Fig.5, 5-th, 6-th, and 7-th bar groups show the correlation coefficients between the GPs found by our proposed method and the attention backbones. Note that we didn't include all attention heads, because some pixels in attention masks are always off. For instance, for Cifar10 [33] each attention mask is 3 by 3. But, as illustrated by Fig.S5 in supplementary material, some output heads like head 1, head 2, and head 3 change around -2. Note that the sigmoid activation function is small around -2. Therefore, according to the scatters, those attention pixels do not turn on for any instance. Therefore, in Fig.5 we have excluded the attention heads which are always off. According to Fig.5,

our proposed GPEX is able find GPs which are faithful to attention subcomponents of the classifier pipelines.

For the experiments of Fig.5, the scatter plots are provided in supplementary material in Figs. S2, S3, S4, S5, S6, and S7. Moreover, the accuracies of GPs and ANNs are provided in Tab.S1 of the supplementary material. According to Figs.S59, S60, S61, and S62 the disagreement between GPs prediction and ANN prediction mostly happens when either some output activations are very close to one another or all activations are close to zero. This is consistent with the scatters of Figs.S2, S4, and S6 in which the scatters are slightly dispersed for intermediate values.

5.2 Explaining ANNs' Decisions

In Sec.5.1 we trained four CNN classifiers on Cifar10 [33], MNIST [32], Kather [34], and DogsWolves [35], respectively. Afterwards, we applied our proposed method (i.e. Alg.4) to each CNN classifier. In this section, we are going to explain the decisions made by the CNN classifiers via the GPs and the kernel-spaces that our proposed GPEX has found. We explain the decision made for a test instance like x_{test} as follows. We consider the GP and the kernel-space that correspond to the ANN's head with maximum value (i.e. the ANN's head that relates to the predicted label). Consequently, among the instances in the inducing dataset, we find the 10 closest instances to x_{test} like $\{x_{i1}, x_{i2}, \dots, x_{i10}\}$. Intuitively the ANN has labeled x_{test} in that way because it has found x_{test} to be similar to $\{x_{i1}, x_{i2}, \dots, x_{i10}\}$. Besides



Fig. 8: Sample explanations for Cifar10 (set 1).

finding the nearest neighbours, we provide explanation as to why x_{test} and an instance like $x_{ij}, 1 \leq j \leq 10$ are considered similar by the model. The procedure is explained in Sec.4.2.

For MNIST digit classification, some test instances and nearest neighbours in training set are shown in Figs.6 and 7. In these figures each row corresponds to a test instance. The first column depicts the test instance itself and columns 2 to 11 depict the 10 nearest neighbours. According to rows 1 and 2 of Fig.6, the classifier has labeled the two images as digit 1 because it has found 1 digits with similar inclinations in the training set. We see the model has also taken the inclination into account for the test instances of rows 7 and 8 of Fig.6 and rows 4, 5, and 6 of Fig.7. In Fig.6, according to rows 3, 4, and 5 the test instances are classified as digit 2 because 2 digits with similar styles are found in the training set. We see the model has also taken the style into account for the test instances of rows 6, 7, 8, 9, 10, 11 of Fig.6 and rows 1, 2, 3, 4, 5, and 6 of Fig.7. For instance, the test instance in row 6 of Fig.6 is a 4 digit with a short tail and the two nearest neighbours are alike. Or for the test instances in rows 2, 3, and 4 of Fig.7 the test instances have incomplete circles in the same way as their nearest neighbours. For MNIST [32], more explanations are provided in the supplementary material in Figs.S8, S9, S10, and S11. Fig.2 illustrates a sample explanation for similarities. For instance row 1 of Fig.2 illustrates a test instances as well as the 10 nearest neighbours. The second row of Fig.2 highlights to what degree each region of each nearest neighbour contributes to its similarity to the test instance. The third row of Fig.2 illustrates to what degree each region of the test instance contributes to its similarity to each of the nearest neighbours. In the supplementary material, we have included many explanations similar to Fig.2. According to rows 1, 2, and 3 or Fig.S17 in the supplementary material, the cross pattern of the 8 digits have had a significant contribution to their similarities. For MNIST [32], several explanations are included in the supplementary material in Figs.S12, S13, S14, S15, S16, S17, S18, and S19.

Fig.8 illustrates some sample explanations for Cifar10 [33]. Like before, each row corresponds to a test instance, the first column depicts the test instance itself and columns 2 to 11 depict the 10 nearest neighbours. In Fig.8, the test instances of rows 1, 2, 3, 4, and 5 are captured from horses' heads from closeby, and the nearest neighbours are alike. However, in rows 6, 7, 8, 9, 10, and 11 of Fig.8 the test images are taken from faraway and the found similar training images are also taken from faraway. Intuitively, as the classifier is not aware of 3D geometry, it finds training images which are captured from the same distance. We constantly observe this pattern in more explanations in the supplementary material: row 6, 7, 8, 9, 10, and 11 in Fig.S39, all rows of Fig.S40, rows 1, 2, 6, 7, 8, 9, 10 and 11 of Fig.S41, rows 1, 7, 8, 9, 10, and 11 of Fig.S42, rows 1, 2, 3, 4, 5, 6, 7, and 8 of Fig.S43, rows 8, 9, 10, and 11 of Fig.S44, all rows of Fig.S45 and rows 1-10 of Fig.S46. Moreover, animal faces tend to be recognized by similar faces. We see this pattern in rows 2, 3, 4, 5 and 6 of Fig.S40, rows 6, 7, 8, and 9 of Fig.S41, rows 7 and 8 of Fig.S43, rows 8, 9, 10, and 11 of Fig.S44 and rows 1, 2, 10, and 11 of Fig.S45. To classify airplanes, the model have taken into account the

inclination. For instance, in Fig.S36, the model has taken into account whether the airplane is taking off (rows 1, 8, 9, 10, and 11 of Fig.S36), flying straight (rows 2 and 4 of Fig.S36) or is inclined downwards (rows 3, 5, 6 and 7 of Fig.S36). Furthermore, the bat-like airplanes are recognized by the model because similar planes are found in the training set, as we see in rows 1, 2, 3, 4, 5, 6 and 7 of Fig.S37. Cessnas are often classified by finding cessnas in the training set, as we see in rows 8, 9 and 10 of Fig.S37 and row 1 of Fig.S38. As the classifier has no knowledge about 3D geometry, it tends to find training instances which are captured from the same angle as the test instance, as we see in rows 6, 7, 8, 9, 10 and 11 of Fig.39, rows 7, 8, 9, 10 and 11 of Fig.S42, rows 9, 10 and 11 of Fig.S43, rows 1, 2, 3, 4, 5, 6 and 7 of Fig.S44, row 11 of Fig.S46, all rows of Fig.S47, and rows 1, 2, 3, 4, 5, 6, 7, and 8 of Fig.S48. In rows 3, 4, and 5 of Fig.S41 it seems the model takes into account the ostrich-like shape of the animal. In rows 2, 3, 4, and 5 of Fig.S42 the horns seem to have an effect. In rows 6, 7, 8, and 9 of Fig.S45, we see the model have made use of the riders to classify the test instances as horse. According to rows 1, 2, 3, 4, 5, 6, 7, 8, 9, and 10 of Fig.S46, the model distinguishes between medium sized ships and huge cargo ships. To classify firefighter trucks, model tends to find similar firefighter trucks in the training set, as we see in rows 10 and 11 of Fig.S47, and rows 1, 2, 3, and 4 of Fig.S48. For some testing instances, the model finds training instances which are almost identical to the test instance, as we see in rows 2 and 5 of Fig.S40, row 7 of Fig.S42, row 8 of Fig.S43, and row 8 of Fig.S48. In rows 2, 4, 5, 6, 7, 8, 9, 10, and 11 of Fig.S38 it seems the classifier has taken into account the blue background. We used the proposed GPEX to explain as to why some testing instances get missclassified. Rows 9, 10, and 11 of Fig.S48 and all rows of Fig.S49 illustrate some instances which are misclassified. For instance in row 10 of Fig.S48 the test image shows an airplane, but the model has classified it as a cat, because it is similar to the cat faces shown in columns 2 to 11 (can you find the cat face in the airplane image?). In row 11 of Fig.S48, the car is classified as truck partially because it very similar to the truck at column 2. In row 2 of Fig.S49, the deer is classified as horse partially because it is very similar to the training image shown in column 2. In row 3 of Fig.S49, we hypothesize the dog is classified as cat because the model has taken into account the cyan and red colors in the background. In this case, adding dog images with cyan and red background may make the model classify this test instance correctly. In rows 5 and 6 of Fig.S49, the model correctly understands the test images are similar to some faces from other animals, but it fails to find similar frog faces in the training set. In this case, adding more images from frog faces may solve this issue. You can find other interesting points in Figs.S36-S49 of the supplementary material.

For the DogsWolves dataset [35], the explanations are provided in Figs.S29-S35 of supplementary material. According to row 3 of Fig.S29, the red ball in the test instance has the most contribution to the similarity. According to row 2 of Fig.S29, patterns like human hand in column 4 or woody or pink background in columns 8, 10, and 11 are highlighted in nearest neighbours. Our explanations consistently show that the model detects dogs by any pattern that rarely appear in a wolf image. For instance in rows

4-6 of Fig.S29, the brick wall in the test instance, humans in columns 3, 9, and 11, and dog collars or costumes in columns 4, 5, 6, and 10 are used by the model. According to rows 9, 12, and 15 of Fig.S29, the flowers, the red ball in the dogs mouth, and children are used by the model, respectively. According to rows 3, 6, 9, 12, and 15 of Fig.S30, the red rope, the dog's color, red patterns, brown background and brown background are used by the model respectively. According to rows 3, 6, 9, 12, and 15 of Fig.S31, brown background, human, brown background, the red wallet, and the pink ball are used by the model, respectively. According to rows 3, 6, 9, 12, and 15 of Fig.S32, human, pink pillow, brown color, orange background, and red blood are used by the model, respectively. Note that in Fig.S32 the last two instances (rows 10-15) are misclassified. In Fig.S33 all test instances get misclassified. According to rows 3, 6, 9, 12, and 15 of Fig.S33, colorful background, the red object attached to the wolf, background, white background, and dark-green background are used by the model, respectively. Figs.S34 and S35 illustrate more explanations. For instance, according to row 6 of Fig.S34 and row 12 of Fig.S35, the test instances are misclassified due to their dark background. Moreover, according to rows 3, 6, and 15 of Fig.S35, the test instances are misclassified due to their background. All in all, our explanations reveal that for DogsWolves dataset [35] the model makes use of potentially incorrect clues to label instances. This is not surprising because the dataset has only 2000 images.

For Kather dataset [34], some explanations are shown in supplementary material in Figs.S20, S21, S22, S23, and S24. Like before, each row corresponds to a test instance, the first column depicts the test instance itself and columns 2 to 11 depict the 10 nearest neighbours. In row 1 of Fig.S22, the test image is classified as fat tissue. According to rows 1, 2, and 3 of Fig.S22, the similarity is due to the wire mesh formed by cellular membranes described by our expert pathologist. Row 13 of Fig.S22 shows cancer-associated stroma which is classified correctly. All 10 nearest neighbours are also cancer-associated stroma. Distinguishing between cancer-associated stroma and normal smooth muscle is a challenging task even for expert pathologists, and they often look similar. According to rows 13, 14, and 15 of Fig.S22 in the supplementary material, the model sometimes cares about both the stroma and nuclei. In row 7 of Fig.S22, the test image is correctly classified as lymphocytes. For a pathologist they represent scattered well defined round structures. According to rows 7, 8, and 9 of Fig.S22, the model considers all regions which matches the way pathologists recognize lymphocytes. In rows 1, 2 and 3 of Fig.S23 and rows 1, 2, and 3 of Fig.S24, for the two test instances the model takes into account nuclei which is not the same way that a pathologists would classify the images. We hypothesize that for the model it is easier to extract features from nuclei than to consider the context information. Because even small changes in nuclei is easily measurable by the model while it is not easily noticeable by human eyes. The test image in row 7 of Fig.S24 gets missclassified. According to rows 7, 8, and 9 of Fig.S24 the artificial white holes are considered as glandular lumens by the model and that explains why the test instance gets misclassified. The test image in row 10 of Fig.S24 gets missclassified. According to rows 10, 11,

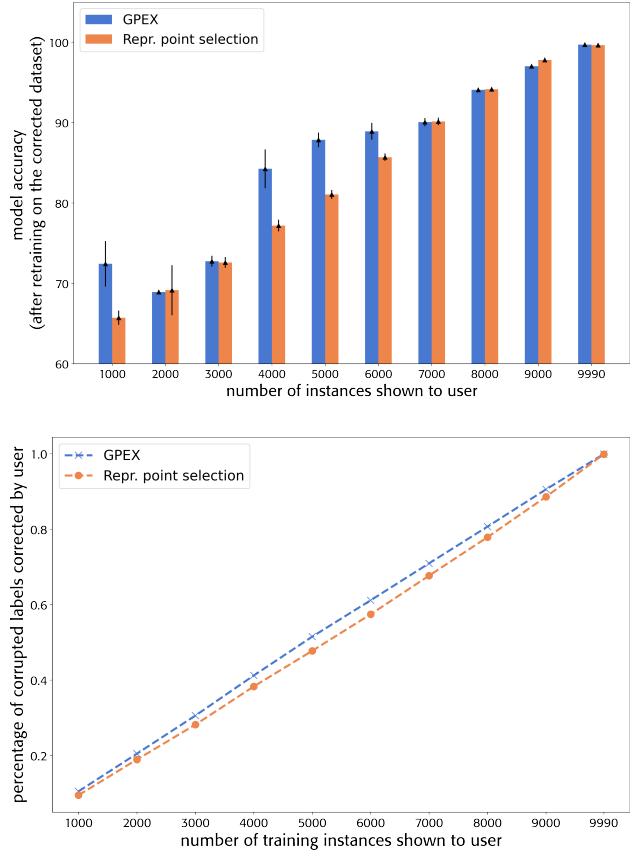


Fig. 9: Comparing the proposed GPEX to representer point selection [12] in dataset debugging task.

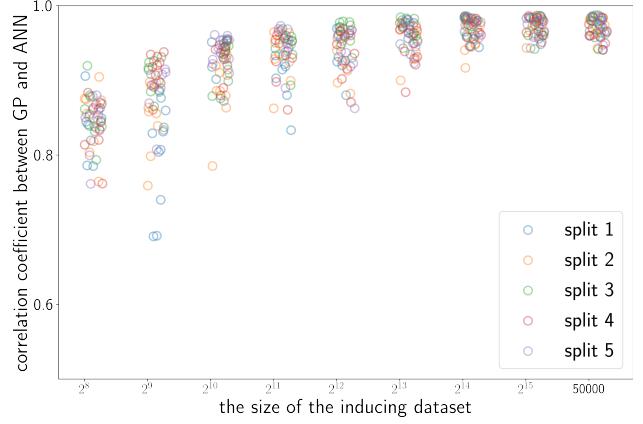


Fig. 10: Analyzing the effect of the size of inducing dataset.

and 12 of Fig.S24, the test image is smooth muscle. But it contains artifactual white spaces (retractions) which make the model think the test image is similar to debris images that contain artifactual white spaces. For Kather dataset [34], more sample explanations are provide in the supplementary materials in Figs.S20-S24.

5.3 Evaluating GPEX in Dataset Debugging Task

For Cifar10 [33] we only selected images which are labeled as either automobile or horse. To corrupt the labels, we randomly selected 45% of training instances and changed their labels. Afterwards, we trained a classifier CNN with

ResNet18 [36] backbone for this binary classification task. We used the training procedure that we explained in Sec.5.1. Because 45% of labels in the dataset are corrupted, the model accuracy understandably dropped to 64.8%. In dataset debugging task, training instances are shown to a user in some order. After seeing an instance, the user checks the label of the instance and corrects it if needed. One can use explanation methods to bring the corrupted labels to the user's attention more quickly, because going through the training instances one by one is tedious for the user. Given an explanation method, we repeatedly select a test instance which is misclassified by the model. Afterwards, we show to the user the closest training instance (of course among the training instances which are not yet shown to the user). We repeat this process for test instances in turn until all training instances are shown to the user. Note that we show the nearest neighbour of a misclassified test instance to the user, because intuitively the nearest neighbour may have had a corrupted label and has caused the model to misclassify the test instance. We compared our proposed GPEX to representer point selection [12] in dataset debugging task. The result is shown in Fig.9. According to the upper plot in Fig.9, when correcting the dataset by GPEX, the model accuracy becomes close to 90% after showing about 4000 instances to user. But when using representer point selection [12], this happens when the user has seen about 7000 training instances. As some labels in the dataset are corrupted, model training becomes unstable. Therefore, in the upper plot of Fig.9 we repeat the training 5 times and we report the standard errors by the lines in top of the bars. According to the lower plot of Fig.9, after showing a fixed number of training instances to the user, when using the proposed GPEX more corrupted labels are shown to the user. Indeed, GPEX brings the corrupted labels to the user's attention quicker than representer point selection [12] does. Besides the quantitative analysis of Fig.9, we quantitatively compare GPEX explanations to those of representer point selection [12]. The results are provided in the supplementary material in Figs.S50-S57. In each triple, the first row shows the test instance and the 10 nearest neighbours found by our proposed GPEX. The second row shows the 10 nearest neighbours selected by representer point selection [12]. The third row shows the 10 nearest neighbours according to the kernel-space of representer point selection [12]. Representer point selection [12] assigns an importance weight to each training instance. Therefore, some training instances tend to appear as nearest neighbours regardless of what the testing instance is. We see this behaviour in rows 2, 5, 8, 11, and 14 of Figs.S50-S57. However, for our proposed GPEX the nearest neighbours can freely change for different test instances. We see this behaviour in rows 1, 4, 7, 10, and 13 of Figs.S50-S57. If we ignore the importance weights in representer point selection [12], the aforementioned issue in that method happens less frequently, as we see in rows 3, 6, 9, 12, and 15 of Figs.S50-S57. However, the issues is that without the importance weights, the explainer model in representer point selection [12] will not be faithful to the ANN itself.

6 PARAMETER ANALYSIS

To analyze the effect of the number of inducing points (i.e. the variable M in Sec.4) we applied the proposed GPEX to the classifier CNN that we trained in Sec.5.1 on Cifar10 dataset [33]. This time, instead of considering all training instances as the inducing dataset, we randomly selected some training instances. In Fig.10, the horizontal axis shows the size of the inducing dataset. For each size, we repeated the experiment 5 times (i.e. split 1-5 in Fig.10). According to Fig.10, to obtain GPs which are faithful to ANNs one needs to have a lot of inducing points. This highlights the importance of the scalability technique that we used in Sec.4.1. Another intriguing point in Fig.10 is that if we are to select a few training images as inducing points, the correlation coefficients highly depend on which instances are selected. More precisely, Fig.10 suggests that one may be able to reach high correlation coefficients by selecting few inducing points from the training set in a subtle way. We analyzed two other important factors: the width of the second last layer and the number of epochs for which the ANN has been trained. We trained ANNs with different number of neurons in the second last layer and we analyzed the ANN at different checkpoints of training (10, 50, 100, 150, and 200 epochs). The result is shown in the supplementary material in Fig.S58. According to Fig.S58, increasing the width of the second last layer increases correlation coefficients. However, as illustrated by Fig.S58, the proposed GPEX can achieve almost perfect match even when the second last layer of the ANN is not wide. Moreover, according to Fig.S58, our proposed GPEX can reach high correlation coefficients even when the ANN's parameters are not a local minimum of the classification loss. This empirical results show that most theoretical results like requiring all layers of the ANN to be wide [27], or requiring the ANN to be optimized on a loss [22] may not be necessary.

7 CONCLUSION

In this paper, we presented a framework for explaining ANNs by Gaussian processes. The obtained GPs are faithful to ANNs, and therefore the explanations are highly reliable and provide intriguing insights about the decision-making mechanism of ANNs. Our framework called GPEX is publicly available as a tool, which enables the effortless adoption of our framework. Besides explaining ANNs, our framework can obtain more insights about GP-ANN analogy (like we did in parameter analysis section), and to discover new theoretical findings based on empirical results. The proposed GPEX can open the ANN black-box, which might provide significant improvements in theoretical and empirical aspects of ANNs.

ACKNOWLEDGMENTS

The authors would like to thank Compute Canada for providing computational resources. Moreover, we thank Namitha Guruprasad for helping in experiments.

REFERENCES

- [1] A. Pinkus, "Approximation theory of the mlp model in neural networks," *Acta Numerica*, vol. 8, p. 143–195, 1999.
- [2] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang, "The expressive power of neural networks: A view from the width," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017.
- [3] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," *CoRR*, vol. abs/1206.5533, 2012.
- [4] Sergey Ioffe et al, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 448–456.
- [5] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.
- [6] Martin Abadi et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org.
- [7] Adam Paszke et al., "Automatic differentiation in pytorch," 2017.
- [8] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7132–7141.
- [9] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Cham: Springer International Publishing, 2015, pp. 234–241.
- [10] F. Milletari, N. Navab, and S.-A. Ahmadi, "V-net: Fully convolutional neural networks for volumetric medical image segmentation," in *2016 Fourth International Conference on 3D Vision (3DV)*, pp. 565–571, 2016.
- [11] E. Tjoa and C. Guan, "A survey on explainable artificial intelligence (xai): Toward medical xai," *IEEE transactions on neural networks and learning systems*, vol. PP, October 2020.
- [12] Chih-Kuan Yeh et al., "Representer point selection for explaining deep neural networks," in *Advances in Neural Information Processing Systems*, vol. 31. Curran Associates, Inc., 2018.
- [13] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should I trust you?": Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, CA, USA, August 13–17, 2016, 2016, pp. 1135–1144.
- [14] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 2017, pp. 4765–4774.
- [15] P. W. Koh and P. Liang, "Understanding black-box predictions via influence functions," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 70. PMLR, 06–11 Aug 2017, pp. 1885–1894.
- [16] Marco Ancona et al., "Towards better understanding of gradient-based attribution methods for deep neural networks," in *International Conference on Learning Representations*, 2018.
- [17] A. Ghorbani, A. Abid, and J. Zou, "Interpretation of neural networks is fragile," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, pp. 3681–3688, Jul. 2019.
- [18] D. Slack, S. Hilgard, E. Jia, S. Singh, and H. Lakkaraju, "Fooling lime and shap: Adversarial attacks on post hoc explanation methods," in *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, ser. AIES '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 180–186.
- [19] Avanti Shrikumar et al., "Learning important features through propagating activation differences," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 70. PMLR, 06–11 Aug 2017, pp. 3145–3153.
- [20] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 70. PMLR, 06–11 Aug 2017, pp. 3319–3328.
- [21] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.
- [22] R. Novak, L. Xiao, J. Hron, J. Lee, A. Alemi, J. Sohl-dickstein, and S. Schoenholz, "Neural tangents: Fast and easy infinite neural networks in python," 2020.
- [23] H. Chen, L. Zheng, R. al Kontar, and G. Raskutti, "Stochastic gradient descent in correlated settings: A study on gaussian processes," *Advances in neural information processing systems*.
- [24] A. G. Wilson, Z. Hu, R. R. Salakhutdinov, and E. P. Xing, "Stochastic variational deep kernel learning," in *Advances in Neural Information Processing Systems*, vol. 29. Curran Associates, Inc., 2016.
- [25] A. H. Akbarnejad and M. S. Baghshah, "An efficient semi-supervised multi-label classifier capable of handling missing labels," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 2, pp. 229–242, 2019.
- [26] R. Neal, *Bayesian Learning for Neural Networks*, ser. Lecture Notes in Statistics. Springer New York, 2012.
- [27] A. G. de G. Matthews, J. Hron, M. Rowland, R. E. Turner, and Z. Ghahramani, "Gaussian process behaviour in wide deep neural networks," in *International Conference on Learning Representations*, 2018.
- [28] Jaehoon Lee et al., "Wide neural networks of any depth evolve as linear models under gradient descent," in *Advances in Neural Information Processing Systems*, vol. 32. Curran Associates, Inc., 2019.
- [29] J. Gardner, G. Pleiss, D. Bindel, K. Weinberger, and A. Wilson, "Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration," *Advances in Neural Information Processing Systems*, vol. 2018–December, pp. 7576–7586, 2018.
- [30] L. He, N. Ray, Y. Guan, and H. Zhang, "Fast large-scale spectral clustering via explicit feature mapping," *IEEE Transactions on Cybernetics*, vol. 49, no. 3, pp. 1058–1071, 2019.
- [31] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.
- [32] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.
- [33] A. Krizhevsky, "Learning multiple layers of features from tiny images," pp. 32–33, 2009. [Online]. Available: <https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf>
- [34] J. N. Kather, C.-A. Weis, F. Bianconi, S. M. Melchers, L. R. Schad, T. Gaiser, A. Marx, and F. G. Zöllner, "Multi-class texture analysis in colorectal cancer histology," *Scientific Reports*, vol. 6, 2016.
- [35] H. Vutukuri, "Dogs vs Wolves Classification of Dogs and Wolves," <https://www.kaggle.com/harishvutukuri/dogs-vs-wolves>, 2019, [Online; accessed 19-Dec-2021].
- [36] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.
- [37] "An implementation for ResNet in pytorch," <https://github.com/kuangliu/pytorch-cifar>, [Online; accessed 19-Dec-2021].
- [38] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.
- [39] Meng-Hao Guo et al., "Attention mechanisms in computer vision: A survey," *CoRR*, vol. abs/2111.07624, 2021.

Amir Akbarnejad received his B.Sc. and M.Sc. degrees from Sharif University of Technology, Tehran, Iran. He is a PhD student at University of Alberta.

Nilanjan Ray received Bachelor of Mech. Eng. (Jadavpur Univ., India, 1995), M.Tech. in Comp. Sc. (Indian Statistical Institute, India, 1997), and Ph.D. in Electrical Eng. (Univ. of Virginia, USA, 2003). He is a Professor at Computing Science, University of Alberta, Canada. His research interest includes computer vision and medical image analysis. He published more than 100 peer reviewed papers in these areas. Nilanjan served as an associate editor at IEEE Transactions on Image Processing (2013-2017) and IET Image Processing (2016-2021). He co-chaired AI-GI-CRV conference in 2017.

Gilbert Bigras Gilbert Bigras received a M.D. degree (Université de Montréal, 1985), a specialist certificate in human pathology (F.R.C.P. Pathology, Université de Montréal 1993) and a Ph.D. in Biomedical Engineering (Université Joseph Fourier, Grenoble, France 1997). He is associate professor at the Faculty of Medicine, University of Alberta, Canada. His research interest includes image analysis applied to Breast Biomarkers. He published more than 50 peer reviewed papers in this area. He is member of the Canadian Association Pathologists National Standards Committee for High Complexity Testing. He is a practicing pathologist at the Cross Cancer Institute (Edmonton, Alberta) and the medical lead for immunohistochemistry and Breast Biomarkers in North Alberta.

Supplementary Material for GPEX, A Framework For Interpreting Artificial Neural Networks

Amir Akbarnejad, Gilbert Bigras, and Nilanjan Ray, *IEEE Member.*

arXiv:2112.09820v1 [cs.LG] 18 Dec 2021

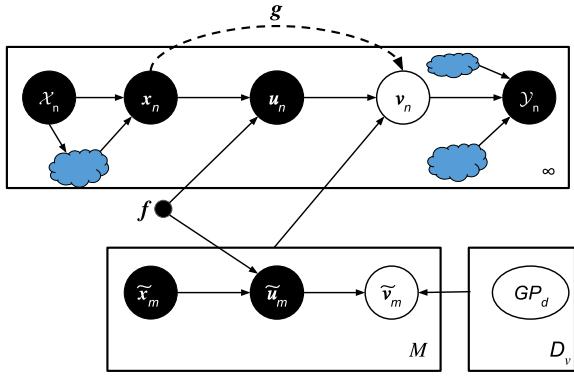


Fig. S1: The proposed framework as a probabilistic graphical model.

S1 DERIVING THE VARIATIONAL LOWER-BOUND

In this section we derive the variational lower-bound introduced in Sec.3.3 of the main article. We firstly introduce Lemmas 1 and 2 as they appear in our derivations.

Lemma 1. The KL-divergence between two normal distributions $\mathcal{N}_1(\cdot; \mu_1, \Sigma_1)$ and $\mathcal{N}_2(\cdot; \mu_2, \Sigma_2)$ can be computed as follows:

$$\begin{aligned} KL(\mathcal{N}_1 \parallel \mathcal{N}_2) &= \frac{1}{2} \left(\log \left(\frac{|\Sigma_2|}{|\Sigma_1|} \right) - D + \text{trace}\{\Sigma_2^{-1} \Sigma_1\} \right. \\ &\quad \left. + (\mu_2 - \mu_1)^T \Sigma_2^{-1} (\mu_2 - \mu_1) \right). \blacksquare \end{aligned} \quad (S1)$$

Lemma 2. Let p_1 and p_2 be two normal distributions:

$$\begin{aligned} p_1(x) &= \mathcal{N}(x; \mu_1, \sigma_1^2), \\ p_2(x) &= \mathcal{N}(x; \mu_2, \sigma_2^2). \end{aligned}$$

We have that

$$\begin{aligned} \mathbb{E}_{x \sim p_2} [\log p_1(x; \mu_1, \sigma_1^2)] &= \\ &- \frac{(\mu_1 - \mu_2)^2 + \sigma_2^2}{2\sigma_1^2} - \frac{1}{2} \log(\sigma_1^2) - \frac{1}{2} \log(2\pi). \blacksquare \end{aligned} \quad (S2)$$

Fig.S1 illustrates the framework as a probabilistic graphical model. A general feed-forward pipeline takes in a set of

input(s) \mathcal{X} and produces a set of output(s) \mathcal{Y} . The general pipeline is required to have at least one ANN as a submodule. The ANN submodule is required to take in only one input x and to produce only one output v , where x and v are tensors of arbitrary sizes. As illustrated in Fig.S1, the ANN's input x can depend arbitrarily on some other intermediate variables in the pipeline. This relation is modeled by the conditional distribution $p(x_n | \text{Parent}(x_n))$ where $\text{Parent}(x_n)$ is the set of all variables which are connected to x_n . Similarly, as illustrated in Fig.S1 the pipeline's output \mathcal{Y} can arbitrarily depend on some intermediate variables in the pipeline. This relation is modeled by the conditional distribution $p(y_n | \text{Parent}(y_n))$. In Fig.S1 the lower boxes are the inducing points and other variables that determine the GPs' posterior. More precisely, in Fig.S1 $\{\tilde{x}_m\}_{m=1}^M$ are some inducing points (e.g. some training images). Vectors in the kernel space are denoted by \tilde{u} and u . Moreover, the observed values are denoted by v and \tilde{v} . Informally, u and v denote the input/output of the GPs. When referring to one of the M inducing points a "tilde" is used (as (\tilde{u}, \tilde{v})), however (u, v) corresponds to a point that can be anywhere in the kernel-space.

The inducing instances $\{\tilde{x}_m\}_{m=1}^M$ are mapped to the kernel-spaces by the kernel mappings $\{f_1(\cdot), \dots, f_L(\cdot)\}$. In Fig.S1 the variables $\{\tilde{u}_m\}_{m=1}^M$ are the kernel-space representations of the inducing points $\{\tilde{x}_m\}_{m=1}^M$. Moreover, $\{\tilde{v}_m\}_{m=1}^M$ are the GP's output values at the inducing points. Given an instance x_n , it is firstly fed to the kernel mappings $\{f_1(\cdot), \dots, f_L(\cdot)\}$ and the kernel-space representations u_n are obtained. Afterwards, the GPs' outputs on u_n depend on u_n as well as all other inducing points because the inducing points actually determine the GPs' posterior on all kernel-space points including u_n . Therefore, in Fig.S1 the variable v_n is not only connected to u_n but it is also connected to the box at the bottom (i.e. all inducing points and other variables associated with them).

As usual, the variational lower-bound is equal to

$$\mathcal{L} = \mathbb{E}_{\sim q} [\log p(\text{all variables})] - \mathbb{E}_{\sim q} [\log q(\text{hidden variables})]. \quad (S3)$$

The likelihood of all variables in Eq.S3 factorizes as the product of conditional distributions of each variable given

its parents. Therefore

$$p(\text{all variables}) = \prod_{\text{variable } t} p(t|\text{Parent}(t)). \quad (\text{S4})$$

In Eq.S4 only some conditional distributions appear in our derivations which are discussed at the following.

- The variable \mathbf{x}_n : the ANN's input \mathbf{x}_n can depend arbitrarily on some other intermediate variables in the pipeline. In our derivations we leave this conditional distribution as $p(\mathbf{x}_n|\text{Parent}(\mathbf{x}_n))$.
- The variable \mathbf{u}_n : Given a training instance \mathbf{x}_n , the kernel-space representations \mathbf{u}_n are deterministically obtained by feeding the instance to the kernel-mappings $[f_1(\cdot), \dots, f_L(\cdot)]$.
- The variable \mathbf{v}_n : The ANN's output is required to depend only on the input, so

$$p(\mathbf{v}_n|\text{Parent}(\mathbf{v}_n)) = p(\mathbf{v}_n|\mathbf{u}_n, \mathbf{x}_n, \{\tilde{\mathbf{x}}_m, \tilde{\mathbf{u}}_m, \tilde{\mathbf{v}}_m\}_{m=1}^M). \quad (\text{S5})$$

The above distribution is actually the GPs' posterior at \mathbf{u}_n (i.e. the normal distribution of Eq.1 of the main article).

- The variable $\tilde{\mathbf{x}}_m$: the inducing point $\tilde{\mathbf{x}}_m$ can depend arbitrarily on some other intermediate variables in the pipeline. In our derivations we leave this conditional distribution as $p(\tilde{\mathbf{x}}_m|\text{Parent}(\tilde{\mathbf{x}}_m))$.
- The variable $\tilde{\mathbf{u}}_m$: Given an inducing point $\tilde{\mathbf{x}}_m$, the kernel-space representations $\tilde{\mathbf{u}}_m$ are deterministically obtained by feeding the inducing point $\tilde{\mathbf{x}}_m$ to the kernel-mappings $[f_1(\cdot), \dots, f_L(\cdot)]$.
- The variables $\tilde{\mathbf{v}}_m$: Given the kernel-space representations $\{\tilde{\mathbf{u}}_m^{(\ell)}\}_{m=1}^M$, the variables $\{\tilde{\mathbf{v}}_1^{(\ell)}, \dots, \tilde{\mathbf{v}}_M^{(\ell)}\}$ follow a M -dimensional Gaussian distribution with zero mean and a covariance matrix determined by the GP prior covariance among the variables $\{\tilde{\mathbf{u}}_m^{(\ell)}\}_{m=1}^M$.
- The variable \mathcal{Y}_n : the pipeline's output \mathcal{Y} can arbitrarily depend on some intermediate variables in the pipeline. In our derivations we leave this conditional distribution as $p(\mathcal{Y}_n|\text{Parent}(\mathcal{Y}_n))$.

According to Eq.S4, the likelihood of all variables factorizes as

$$\begin{aligned} p(\text{all variables}) &= \prod_{\text{variable } t} p(t|\text{Parent}(t)) \\ &= (\prod_n p(\mathbf{x}_n|\text{Parent}(\mathbf{x}_n))) \times (\prod_n p(\mathbf{u}_n|\mathbf{x}_n)) \times \\ &\quad (\prod_n \prod_\ell p(v_n^{(\ell)}|\mathbf{u}_n, \mathbf{x}_n, \{\tilde{\mathbf{x}}_m, \tilde{\mathbf{u}}_m, \tilde{\mathbf{v}}_m\}_{m=1}^M)) \times \\ &\quad (\prod_m p(\tilde{\mathbf{x}}_m|\text{Parent}(\tilde{\mathbf{x}}_m)) \times (\prod_m p(\tilde{\mathbf{u}}_m|\tilde{\mathbf{x}}_m)) \times \\ &\quad (\prod_\ell p(\tilde{\mathbf{v}}_{1:M}^{(\ell)}|\mathbf{0}, \mathcal{K}_{\text{prior}}(\tilde{\mathbf{u}}_{1:M}^{(\ell)}, \tilde{\mathbf{u}}_{1:M}^{(\ell)}))) \times \\ &\quad (\prod_n p(\mathcal{Y}_n|\text{Parent}(\mathcal{Y}_n)) \times (\prod_{\text{other vars } t} p(t|\text{Parent}(t))). \end{aligned} \quad (\text{S6})$$

Now we derive the lower-bound \mathcal{L} with respect to each parameter separately.

S1.1 Deriving the Lower-bound With Respect to the Kernel-mappings

In the right-hand-side of Eq.S6 only the following terms are dependant on the kernel-mappings $[f_1(\cdot), \dots, f_L(\cdot)]$:

$$\begin{aligned} & [\prod_m p(\tilde{\mathbf{u}}_m|\tilde{\mathbf{x}}_m) \times \prod_\ell p(\tilde{\mathbf{v}}_m^{(\ell)}|\mathbf{0}, \mathcal{K}_{\text{prior}}(\tilde{\mathbf{u}}_{1:M}^{(\ell)}, \tilde{\mathbf{u}}_{1:M}^{(\ell)}))] \times \\ & [\prod_n p(\mathbf{u}_n|\mathbf{x}_n) \times \prod_\ell p(v_n^{(\ell)}|\mathbf{u}_n, \mathbf{x}_n, \{\tilde{\mathbf{x}}_m, \tilde{\mathbf{u}}_m, \tilde{\mathbf{v}}_m\}_{m=1}^M)]. \end{aligned} \quad (\text{S7})$$

Please note that in the above equation the terms $p(\tilde{\mathbf{u}}_m|\tilde{\mathbf{x}}_m)$ and $p(\mathbf{u}_n|\mathbf{x}_n)$ are equal to 1 because $\tilde{\mathbf{u}}_m$ and \mathbf{u}_n are deterministically obtained from $\tilde{\mathbf{x}}_m$ and \mathbf{x}_n . Therefore, in Eq.S3 the terms containing the kernel mappings $[f_1(\cdot), \dots, f_L(\cdot)]$ are as follows:

$$\begin{aligned} \mathcal{L}_f &= \mathbb{E}_{\sim q} [\sum_\ell \log p(v^{(\ell)}|\mathbf{u}, \mathbf{x}, \{\tilde{\mathbf{x}}_m, \tilde{\mathbf{u}}_m, \tilde{\mathbf{v}}_m\}_{m=1}^M)] + \\ &\quad \sum_\ell \mathbb{E}_{\sim q} [\log p(\tilde{\mathbf{v}}_{1:M}^{(\ell)}|\mathbf{0}, \mathcal{K}_{\text{prior}}(\tilde{\mathbf{u}}_{1:M}^{(\ell)}, \tilde{\mathbf{u}}_{1:M}^{(\ell)}))] - \\ &\quad \sum_\ell \mathbb{E}_{\sim q} [\log q_2(\tilde{\mathbf{v}}_{1:M}^{(\ell)})] \\ &= \mathbb{E}_{\sim q} [\sum_\ell \log p(v^{(\ell)}|\mathbf{u}, \mathbf{x}, \{\tilde{\mathbf{x}}_m, \tilde{\mathbf{u}}_m, \tilde{\mathbf{v}}_m\}_{m=1}^M)] - \\ &\quad \sum_{\ell=1} \mathbb{E}_{\sim q} [KL(q_2(\tilde{\mathbf{v}}_{1:M}^{(\ell)}) \parallel p(\tilde{\mathbf{v}}_{1:M}^{(\ell)}|\mathbf{0}, \mathcal{K}_{\text{prior}}(\tilde{\mathbf{u}}_{1:M}^{(\ell)}, \tilde{\mathbf{u}}_{1:M}^{(\ell)})))]. \end{aligned} \quad (\text{S8})$$

We simplify the two terms on the right-hand-side of Eq.S8. The first term is the expected log-likelihood of a Gaussian distribution (i.e. the conditional log-likelihood of $\tilde{\mathbf{v}}^\ell$ as in Eq.1 of the main article). Also the variational distribution $q(\cdot)$ is Gaussian. Therefore, we can use Lemma.2 to simplify the first term:

$$\begin{aligned} \mathbb{E}_{\sim q} [\sum_{\ell=1}^L \log p(v^{(\ell)}|\mathbf{u}, \mathbf{x}, \{\tilde{\mathbf{x}}_m, \tilde{\mathbf{u}}_m, \tilde{\mathbf{v}}_m\}_{m=1}^M)] &= \\ \sum_{\ell=1}^L \mathbb{E}_{\sim q} [\log p(v^{(\ell)}|\mathbf{u}, \mathbf{x}, \{\tilde{\mathbf{x}}_m, \tilde{\mathbf{u}}_m, \tilde{\mathbf{v}}_m\}_{m=1}^M)] &= \\ \sum_{\ell=1}^L \left[-\frac{(\mu_v(\mathbf{u}^{(\ell)}, \tilde{\mathbf{u}}_{1:M}^{(\ell)}, \tilde{\mathbf{v}}_{1:M}^{(\ell)}) - g_\ell(\mathbf{x}))^2 + \sigma_g^2}{\text{cov}_v(\mathbf{u}^{(\ell)}, \tilde{\mathbf{u}}_{1:M}^{(\ell)}, \tilde{\mathbf{v}}_{1:M}^{(\ell)})} \right. & \quad (\text{S9}) \\ \left. - \frac{1}{2} \log (\text{cov}_v(\mathbf{u}^{(\ell)}, \tilde{\mathbf{u}}_{1:M}^{(\ell)}, \tilde{\mathbf{v}}_{1:M}^{(\ell)})) \right. \\ \left. - \frac{1}{2} \log(2\pi) \right]. \end{aligned}$$

Please note that the two terms of Eq.S9 are the two terms which were presented and discussed in Eq.5 of the main article.

Now we simplify the KL-term on the right-hand-side of

Eq.S8. According to Lemma.1 we have that

$$\begin{aligned} KL\left(q_2(\tilde{\mathbf{v}}_{1:M}^{(\ell)}) \parallel p(\tilde{\mathbf{v}}_{1:M}^{(\ell)} | \mathbf{0}, \mathcal{K}_{prior}(\tilde{\mathbf{u}}_{1:M}^{(\ell)}, \tilde{\mathbf{u}}_{1:M}^{(\ell)}))\right) = \\ + 0.5 \left(\log \left(\frac{\sigma_{gp}^2}{\sigma_\varphi^2} \right) \right) \\ - 0.5M \\ + \frac{\sigma_\varphi^2}{\sigma_{gp}^2} \\ + \frac{\varphi_{1:M}^{(\ell)T} \varphi_{1:M}^{(\ell)}}{\sigma_{gp}^2}, \end{aligned} \quad (S10)$$

where φ are the variational parameters of $q_2(\cdot)$ as in Eq.4 of the main article. Therefore, the KL-term of Eq.S8 is a constant with respect to the kernel mappings $[f_1(\cdot), \dots, f_L(\cdot)]$ and can be discarded. All in all, the lower-bound for optimizing the kernel-mappings is equal to the right-hand-side of Eq.S9 which was introduced and discussed in Sec.3.3. of the main article.

S1.2 Deriving the Lower-bound With Respect to the ANN Parameters

According to Eq.4 of the main article, in our formulation the ANN's parameters appear as some variational parameters. Therefore, the likelihood of all variables (Eq.S6) does not generally depend on the ANN's parameters. But according to the general ELBO formulation in Eq.S3 the ELBO \mathcal{L} depends on ANN's parameters, because when computing the expectation the variables are drawn from the variational distribution $q(\cdot)$. We estimated the ELBO of Eq.S3 by the average over few samples. More precisely, given a training instance \mathbf{x} , we firstly computed the kernel-space representations as:

$$\mathbf{u}^{(\ell)} = f_\ell(\mathbf{x}), \quad 1 \leq \ell \leq L. \quad (S11)$$

Afterwards, we used the reparametrization trick for Eq.1 of the main article to draw a sample for $\mathbf{v}^{(\ell)}$ as follows:

$$\begin{aligned} z_{q2}^{(\ell)} &\sim \mathcal{N}(0, 1), \\ \mathbf{v}^{(\ell)} &\sim \mu_v(\mathbf{u}^{(\ell)}, \tilde{\mathbf{u}}_{1:M}^{(\ell)}, \tilde{\mathbf{v}}_{1:M}^{(\ell)}) + z_{q2}^{(\ell)} cov_v(\mathbf{u}^{(\ell)}, \tilde{\mathbf{u}}_{1:M}^{(\ell)}, \tilde{\mathbf{v}}_{1:M}^{(\ell)}), \end{aligned} \quad (S12)$$

where $\mu_v(\cdot, \cdot, \cdot)$ and $cov_v(\cdot, \cdot, \cdot)$ are defined in Eqs.2 and 3 of the main article. Moreover, we continue the forward pass of the original pipeline to get a sample \mathcal{Y} . Having drawn \mathbf{x} , \mathbf{u} , \mathbf{v} , and \mathcal{Y} from the variational distribution, we estimate the ELBO of Eq.S3 by these samples.

$\mathcal{L} =$

$$\begin{aligned} \mathbb{E}_{\sim q} [\log p(\text{all variables})] - \mathbb{E}_{\sim q} [\log q(\text{hidden variables})] \\ \approx \log p(\text{all variables}) \Big|_{\mathbf{x}, \mathbf{u}, \mathbf{v}, \mathcal{Y}} - \sum_m^M \sum_\ell^L \mathbb{E}_{\sim q_2} [\log q_2(\tilde{\mathbf{v}}_m^{(\ell)})] \end{aligned} \quad (S13)$$

In the above equation, the second term on the right-hand-side is the entropy of a normal distribution and it only depends on the variance of the q_2 distribution. As we let

the variance of q_2 be fixed (σ_q^2 in Eq.4 of the main article), the second term is a constant. Therefore,

$$\mathcal{L} \approx \log p(\text{all variables}) \Big|_{\mathbf{x}, \mathbf{u}, \mathbf{v}, \mathcal{Y}}. \quad (S14)$$

Among the likelihood term on the right-hand-side of Eq.S6 the conditional distribution of all variables before \mathbf{u}_n (e.g. \mathbf{x}_n and \mathcal{X}_n) are independent of the ANN's parameters (i.e. the parameters of the function $g(\cdot)$). On the other hand, for all variables that appear after \mathbf{u}_n , the conditional distribution depends on the ANN's parameters. Indeed, according to Eq.S14

$$\begin{aligned} \mathcal{L}_{ann} \approx \left[\sum_{\ell=1}^L \log p(v^{(\ell)} | \mathbf{u}, \mathbf{x}, \{\tilde{\mathbf{x}}_m, \tilde{\mathbf{u}}_m, \tilde{\mathbf{v}}_m\}_{m=1}^M) \right] \Big|_{\mathbf{x}, \mathbf{v}} + \\ \log p(\mathcal{Y} | Parent(\mathcal{Y})) \Big|_{\mathbf{x}, \mathbf{v}, \mathcal{Y}} + \\ \left(\sum_{\text{other vars after } \mathbf{u}_n} \log p(t | Parent(t)) \right) \Big|_{\mathbf{x}, \mathbf{v}, \mathcal{Y}}. \end{aligned} \quad (S15)$$

In the above equation, the first term on the right-hand-side is the log-likelihood of the normal distribution of Eq.1:

$$\begin{aligned} \log p(v^{(\ell)} | \mathbf{u}, \mathbf{x}, \{\tilde{\mathbf{x}}_m, \tilde{\mathbf{u}}_m, \tilde{\mathbf{v}}_m\}_{m=1}^M) = \\ - \frac{1}{2} \left[\sum_{\ell=1}^L \frac{(\mu_v(\mathbf{u}^{(\ell)}, \tilde{\mathbf{u}}_{1:M}^{(\ell)}, \tilde{\mathbf{v}}_{1:M}^{(\ell)}) - g_\ell(\mathbf{x}))^2}{cov_v(\mathbf{u}^{(\ell)}, \tilde{\mathbf{u}}_{1:M}^{(\ell)}, \tilde{\mathbf{v}}_{1:M}^{(\ell)})} \right] \\ + (\text{some terms independent from } g(\cdot)). \end{aligned} \quad (S16)$$

In Eq.S15 the term $p(\mathcal{Y} | Parent(\mathcal{Y}))$ is the likelihood of the output(s) of the whole pipeline as illustrated by Fig.1 of the main article, given the ANN's output and all other intermediate variables on which the final output \mathcal{Y} depends. This likelihood turns out to be equivalent to commonly-used losses like the cross-entropy loss or the mean-squared loss. Here we elaborate upon how this happens. Let the task be a classification, and let $\hat{\mathcal{Y}} \in \mathbb{R}^L$ be the pipeline's output. The final model prediction \mathcal{Y} is done as follows:

$$\mathcal{Y} \sim Categorical(\hat{\mathcal{Y}}_1, \dots, \hat{\mathcal{Y}}_K) \quad (S17)$$

Therefore we have that

$$p(\mathcal{Y} | Parent(\mathcal{Y})) = (\hat{\mathcal{Y}}_1)^{I[\mathcal{Y}==1]} \times \dots \times (\hat{\mathcal{Y}}_K)^{I[\mathcal{Y}==K]}, \quad (S18)$$

where $I[\cdot]$ is the indicator function. So, we have that

$$\begin{aligned} \log p(\mathcal{Y} | Parent(\mathcal{Y})) = \\ I[\mathcal{Y} == 1] \log(\hat{\mathcal{Y}}_1) + \dots + I[\mathcal{Y} == K] \log(\hat{\mathcal{Y}}_K). \end{aligned} \quad (S19)$$

Therefore, when the pipeline is for classification, $\log p(\mathcal{Y} | v, \text{etc.})$ will be equal to the cross-entropy loss. This conclusion was introduced and discussed in Eq.6 of the main article. We can draw similar conclusions when the pipeline is for other tasks like regression, or even a combination of tasks.

In the general pipeline of Fig.S1, if all stages after \mathbf{v} are deterministic (of course except the final stage which is probabilistic like Eq.S17), the third term on the right-hand-side of Eq.S15 becomes 1. Therefore, the right-hand-side of Eq.S15 is equal to Eq.6 of the main article. As we discussed in Sec.3.3 of the main article, \mathcal{L}_{ann} has two terms: the first term encourages the GP-ANN analogy and the second term seeks to lower the task-loss.

S1.3 Deriving the Lower-bound With Respect to $q_2(\cdot)$ Parameters

In Eq.4 of the main article we considered the variational parameters $\{\varphi_m^{(\ell)}\}_{m=1}^M$ for the hidden variables $\{\tilde{v}_m^{(\ell)}\}_{m=1}^M$. The ELBO of Eq.S3 can be optimized with respect to $\{\varphi_m^{(\ell)}\}_{m=1}^M$ as well. But we noticed that optimizing $\{\varphi_m^{(\ell)}\}_{m=1}^M$ is computationally unstable. Therefore, we set $\{\varphi_m^{(\ell)}\}_{m=1}^M$ according to the following rule:

$$\varphi_m^{(\ell)} = g_\ell(\tilde{x}_m), \quad 1 \leq m \leq M, \quad 1 \leq \ell \leq L. \quad (\text{S20})$$

We set $\{\varphi_m^{(\ell)}\}_{m=1}^M$ as above because $\tilde{v}_m^{(\ell)}$ is simply the ℓ -th GP posterior mean at the inducing point \tilde{x}_m . To make the GP's posterior mean equal to the ANN's output, $\tilde{v}_\ell^{(m)}$ should be equal to the ANN's (i.e. $g(\cdot)$'s) output at the m -inducing point.

S2 EFFICIENTLY COMPUTING GAUSSIAN PROCESS POSTERIOR

Let \mathbf{A} be an arbitrary $M \times D$ matrix where $M \gg D$. Moreover, let \mathbf{b} be a M -dimensional vector and let σ be a real number. The computational techniques [30] allow us to efficiently compute:

$$(\mathbf{A}\mathbf{A}^T + \sigma^2 \mathbf{I}_{M \times M})^{-1} \mathbf{b}.$$

The idea is that $\mathbf{A}\mathbf{A}^T$ and therefore its inverse are of rank D . Therefore, $(\mathbf{A}\mathbf{A}^T)^{-1}$ has D non-zero eigenvalues like $\{\lambda_1, \dots, \lambda_D\}$ and the rest of its eigenvalues are zero. Let the corresponding eigenvectors be $\{e_1, \dots, e_D\}$. To compute $(\mathbf{A}\mathbf{A}^T)^{-1} \mathbf{b}$ we can simply project \mathbf{b} to the D -dimensional space of the eigenvectors. By doing so, we avoid the $\mathcal{O}(M^3)$ computational complexity. Let $\{\lambda_1, \dots, \lambda_D\}$ be the non-zero eigenvalues of $\mathbf{A}\mathbf{A}^T$ and let $\{e_1, \dots, e_D\}$ be the corresponding eigenvectors. From linear algebra, it follows that for $\mathbf{A}\mathbf{A}^T + \sigma^2 \mathbf{I}_{M \times M}$ the eigenvalues and the eigenvectors are $\{\lambda_1 + \sigma^2, \dots, \lambda_D + \sigma^2, \sigma^2, \dots, \sigma^2\}$ and $\{e_1, \dots, e_D\}$, respectively. Please note that $M - D$ eigenvectors are added all of which are equal to σ^2 . Similarly, from linear algebra it follows that for the inverse of $\mathbf{A}\mathbf{A}^T + \sigma^2 \mathbf{I}_{M \times M}$ the eigenvalues and eigenvectors are $\{\frac{1}{\lambda_1 + \sigma^2}, \dots, \frac{1}{\lambda_D + \sigma^2}, \frac{1}{\sigma^2}, \dots, \frac{1}{\sigma^2}\}$ and $\{e_1, \dots, e_D, e_{D+1}, \dots, e_M\}$ respectively. Please note that although there are M eigenvectors, only the first D eigenvectors appear in our computations. More precisely, let $\mathbf{E} \in \mathbb{R}^{M \times D}$ be a matrix whose columns are $\{e_1, \dots, e_D\}$. Let $\mathbf{\Lambda}$ be a diagonal matrix whose diagonal is formed by $\{\frac{1}{\lambda_1 + \sigma^2}, \dots, \frac{1}{\lambda_D + \sigma^2}\}$. In the space of the D eigenvectors the linear transformation on any vector like \mathbf{b} is equal to $\mathbf{E}\mathbf{\Lambda}\mathbf{E}^T \mathbf{b}$, meaning that multiplication by \mathbf{E}^T transforms \mathbf{b} to the space of the D eigenvectors, multiplication by $\mathbf{\Lambda}$ performs the transformation in that space, and multiplication by \mathbf{E} transforms the result back to the original space. The $(M - D)$ eigenvalues that correspond to the rest of the eigenvectors are all the same and are equal to $\frac{1}{\sigma^2}$. Therefore, there is no need to project \mathbf{b} to the space of the $(M - D)$ eigenvectors because the linear transformation in that space is simply a scaling by $\frac{1}{\sigma^2}$. All in all, we have that

$$(\mathbf{A}\mathbf{A}^T + \sigma^2 \mathbf{I}_{M \times M})^{-1} \mathbf{b} = \mathbf{E}\mathbf{\Lambda}\mathbf{E}^T \mathbf{b} + \frac{1}{\sigma^2} (\mathbf{b} - \mathbf{E}\mathbf{E}^T \mathbf{b}). \quad (\text{S21})$$

Complexity of computing the right-hand-side of Eq.S21 is way lower than the $\mathcal{O}(M^3)$ requirement of the standard matrix inversion. We borrowed more computational ideas from the work on fast spectral clustering [30]. To compute the first D eigenvalues and eigenvectors of $\mathbf{A}\mathbf{A}^T$, we worked with the D -by- D matrix $\mathbf{A}^T \mathbf{A}$ rather than the M -by- M matrix $\mathbf{A}\mathbf{A}^T$ (recall that $D \ll M$), because given the eigenvalues and eigenvectors of $\mathbf{A}^T \mathbf{A}$, those of $\mathbf{A}\mathbf{A}^T$ are easily computable [30]. The procedure is explained in Alg.5. In Alg.5, lines 1-3 compute the eigenvalues/vectors of the matrix $\mathbf{A}^T \mathbf{A}$. Afterwards, lines 4 and 5 compute the first D eigenvalues/vectors of $\mathbf{A}\mathbf{A}^T$ using those of $\mathbf{A}^T \mathbf{A}$. Finally, line 8 computes $(\mathbf{A}\mathbf{A}^T + \sigma^2 \mathbf{I})^{-1} \mathbf{b}$ according to the right-hand-side of Eq.S21. To make the computations faster, we made use of the following equation $\mathbf{A}\mathbf{A}^T = \sum_m \mathbf{A}[m, :] \mathbf{A}[m, :]^T$, where $\mathbf{A}[m, :]$ is the m -th row of the matrix \mathbf{A} . Thanks to this equation, we compute $\mathbf{A}\mathbf{A}^T$ only once at the beginning of the training. Afterwards, as each mini-batch alters only some rows of \mathbf{A} , we update the previously computed $\mathbf{A}\mathbf{A}^T$ by considering only the effect of the modified rows.

S3 COMPUTING PIXEL CONTRIBUTIONS TO THE SIMILARITY

Let the kernel mapping $f(\cdot)$ be a convolutional neural network that produces a volumetric map of size $C \times H \times W$ followed by a spatial average pooling that produces the C -dimensional vector in the kernel-space. In this case, $\mathcal{K}(\mathbf{x}_1, \mathbf{x}_2)$ is as follows:

$$\begin{aligned} \mathcal{K}(\mathbf{x}_1, \mathbf{x}_2) &= f(\mathbf{x}_1)^T f(\mathbf{x}_2) \\ &= \left(\sum_{i=1}^H \sum_{j=1}^W \mathbf{z}_{ij}^{(1)} \right)^T \left(\sum_{k=1}^H \sum_{\ell=1}^W \mathbf{z}_{k\ell}^{(2)} \right) \\ &= \sum_{i=1}^H \sum_{j=1}^W \sum_{k=1}^H \sum_{\ell=1}^W (\mathbf{z}_{ij}^{(1)T} \mathbf{z}_{k\ell}^{(2)}), \end{aligned} \quad (\text{S22})$$

where $\mathbf{z}^{(1)}$ and $\mathbf{z}^{(2)}$ are the volumetric maps of size $C \times H \times W$ and the indices (i, j) and (k, ℓ) index the spatial locations over the volumetric maps. The last term in Eq.S22 shows that the total similarity $\mathcal{K}(\mathbf{x}_1, \mathbf{x}_2)$ is the sum of the contributions from each pair of positions (i, j) on \mathbf{x}_1 and (k, ℓ) on \mathbf{x}_2 . To compute the contribution of a specific location like (i, j) on \mathbf{x}_1 , we sum up the contributions of (i, j) on \mathbf{x}_1 and all possible locations $\{(k, \ell)\}_{k=1}^H \}_{\ell=1}^W$ on \mathbf{x}_2 .

The kernel-mappings that we used have a slightly different architecture than a volumetric map followed by spatial average pooling. Our kernel mappings produce a volumetric map of size $C \times H \times W$ followed by a spatial average pooling that produces a C -dimensional vector. Afterwards, the resulting vector is divided by its ℓ_2 -norm to produce a vector of norm 1. Consequently, this vector of norm 1 is fed to a leaky ReLU layer that produces the final kernel-space representation $f(\mathbf{x})$. For this architecture the pixel contributions can be computed according to an equation similar to Eq.S22 as follows. Our kernel mappings produce the volumetric map \mathbf{z} of size $C \times H \times W$ followed by a spatial average pooling that produces the C -dimensional vector \mathbf{a} :

$$\mathbf{a} = \sum_{i=1}^H \sum_{j=1}^W \mathbf{z}_{ij}. \quad (\text{S23})$$

Afterwards, the resulting vector is divided by its ℓ_2 -norm to produce the vector \mathbf{b} of norm 1:

$$\mathbf{b} = \left[\frac{a_1}{\|\mathbf{a}\|_2}, \dots, \frac{a_C}{\|\mathbf{a}\|_2} \right]. \quad (\text{S24})$$

Consequently, this vector of norm 1 is fed to a leaky ReLU layer that produces the final kernel-space representation $f(\mathbf{x})$:

$$f(\mathbf{x}) = \text{leakyReLU}(\mathbf{b}). \quad (\text{S25})$$

We begin with simplifying Eq.S25. The leaky ReLU activation function multiplies the input by a constant and this constant depends on the sign of the input. Therefore, applying the leaky ReLU activation is equivalent to multiplication by a diagonal matrix Λ . Therefore,

$$f(\mathbf{x}) = \Lambda \mathbf{b}. \quad (\text{S26})$$

Let \mathbf{x}_1 and \mathbf{x}_2 be two images, and $\mathbf{z}^{(1)}$ and $\mathbf{z}^{(2)}$ be the corresponding volumetric maps. We have that

$$\begin{aligned} \mathbf{a}^{(1)} &= \sum_{i=1}^H \sum_{j=1}^W \mathbf{z}_{ij}^{(1)}, \\ \mathbf{a}^{(2)} &= \sum_{k=1}^H \sum_{\ell=1}^W \mathbf{z}_{k\ell}^{(2)}. \end{aligned} \quad (\text{S27})$$

And

$$\begin{aligned} \mathbf{b}^{(1)} &= \left[\frac{a_1^{(1)}}{\|\mathbf{a}^{(1)}\|_2}, \dots, \frac{a_C^{(1)}}{\|\mathbf{a}^{(1)}\|_2} \right], \\ \mathbf{b}^{(2)} &= \left[\frac{a_1^{(2)}}{\|\mathbf{a}^{(2)}\|_2}, \dots, \frac{a_C^{(2)}}{\|\mathbf{a}^{(2)}\|_2} \right]. \end{aligned} \quad (\text{S28})$$

And

$$\begin{aligned} f(\mathbf{x}^{(1)}) &= \Lambda^{(1)} \mathbf{b}^{(1)}, \\ f(\mathbf{x}^{(2)}) &= \Lambda^{(2)} \mathbf{b}^{(2)}. \end{aligned} \quad (\text{S29})$$

Now we simplify the similarity $\mathcal{K}(\mathbf{x}_1, \mathbf{x}_2)$:

$$\begin{aligned} \mathcal{K}(\mathbf{x}_1, \mathbf{x}_2) &= (\Lambda^{(1)} \mathbf{b}^{(1)})^T (\Lambda^{(2)} \mathbf{b}^{(2)}) \\ &= (\Lambda^{(1)^T} \Lambda^{(2)}) (\mathbf{b}^{(1)^T} \mathbf{b}^{(2)}) \\ &= \frac{(\Lambda^{(1)^T} \Lambda^{(2)})}{\|\mathbf{a}^{(1)}\|_2 \|\mathbf{a}^{(2)}\|_2} \left(\sum_{i=1}^H \sum_{j=1}^W \mathbf{z}_{ij}^{(1)} \right)^T \left(\sum_{k=1}^H \sum_{\ell=1}^W \mathbf{z}_{k\ell}^{(2)} \right) \\ &= \frac{(\Lambda^{(1)^T} \Lambda^{(2)})}{\|\mathbf{a}^{(1)}\|_2 \|\mathbf{a}^{(2)}\|_2} \sum_{i=1}^H \sum_{j=1}^W \sum_{k=1}^H \sum_{\ell=1}^W (\mathbf{z}_{ij}^{(1)^T} \mathbf{z}_{k\ell}^{(2)}). \end{aligned} \quad (\text{S30})$$

Indeed, as the used architecture for kernel-mappings is slightly different than producing a volumetric map followed by spatial average pooling, instead of Eq.S22, we used Eq.S30 that we derived above.

S4 PRACTICAL DETAILS AND PARAMETER SETTINGS

In this section we discuss some practical details which we have not yet discussed in this paper. Moreover, we provide the exact parameter settings that we used throughout our experiments. As explained in Sec.4, there are L kernel mappings that we denoted by $[f_1(\cdot), f_2(\cdot), \dots, f_L(\cdot)]$. One

can implement this kernel mappings by, e.g., considering L independent CNNs. However, doing so dramatically increases the computation cost. Therefore, we modeled the L mappings by a common ResNet-50 [38] backbone. After the common backbone, we placed L branches. Each branch has two convolutional layers followed by global spatial average pooling that produce a vector. Each branch ends with an L2 normalizer layer (that sets the L2-norm of the vector to 1) followed by a leaky-ReLU layer. During our experiments, we noticed that the L2-normalization layer and the final leaky-ReLU layer are essential. Without the L2 normalization layer, the vectors in the kernel-space can have arbitrarily-small or arbitrarily-big elements, and this makes the training unstable. We included the last leaky-ReLU layer, because according to GP posterior mean formula, vectors in the kernel-space go through a linear transformation. Therefore, without the last leaky-ReLU layer, the pipeline would have two consecutive linear layers. Throughout our experiments, we set the output of each branch (i.e. vectors in the kernel-space of each GP) to be 20-dimensional.

As illustrated by Fig.10, we need to make the inducing dataset as large as possible. Therefore, throughout our experiments we selected the whole training dataset as the inducing dataset. Unlike training instances, we didn't apply data-augmentation on inducing instances. By doing so, the training dataset and the inducing dataset will have very similar instances. This causes a difficulty that we are going to discuss in this part. The kernel-mappings $[f_1(\cdot), \dots, f_L(\cdot)]$ are trained according to Alg.4. After selecting an instance like \mathbf{x} from the training dataset, \mathbf{x} is actually the augmented version of an inducing instance like $\tilde{\mathbf{x}}_m$. Indeed, we have that $\mathbf{x} = \text{DataAug}(\tilde{\mathbf{x}}_m)$. Because \mathbf{x} and $\tilde{\mathbf{x}}_m$ are very similar, they will be very close to one another regardless of what parameters $[f_1(\cdot), \dots, f_L(\cdot)]$ have. Therefore, regardless of the kernel-mappings, the GP-mean will match the ANN value at \mathbf{x} , and there will be no training signal for the kernel-mappings $[f_1(\cdot), \dots, f_L(\cdot)]$. Please note that in this case the GPs match the ANNs only on training instances and not testing instances. To avoid this issue, we optimized the GP-ANN analogy (i.e. the objective in Eq.5 of the main article) on instances like $\lambda \mathbf{x}_i + (1 - \lambda) \mathbf{x}_j$, where \mathbf{x}_i and \mathbf{x}_j are two instances randomly selected from the training set and λ is a scalar uniformly selected from $[-1, 2]$.

When applying our proposed GPEX we used Adam optimizer [40]. Although the AMSGrad version [42] of this optimizer is often recommended, for our proposed GPEX we noticed the Adam optimizer [40] without AMSGrad works the best. For explaining classifier ANNs, we used a learning-rate of 0.0001 while for explaining the attention submodules we used a learning rate of 0.00001.

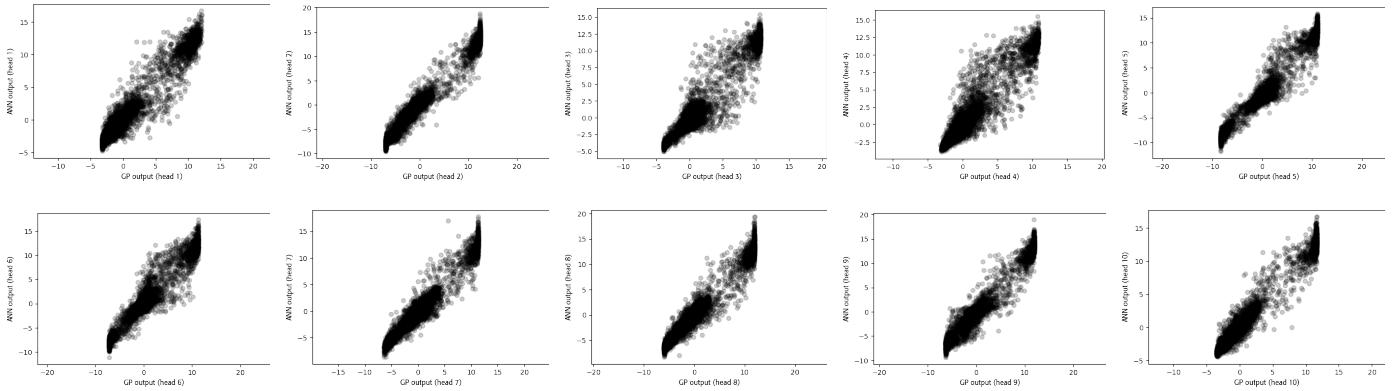


Fig. S2: Scatters for Cifar10 (classifier).

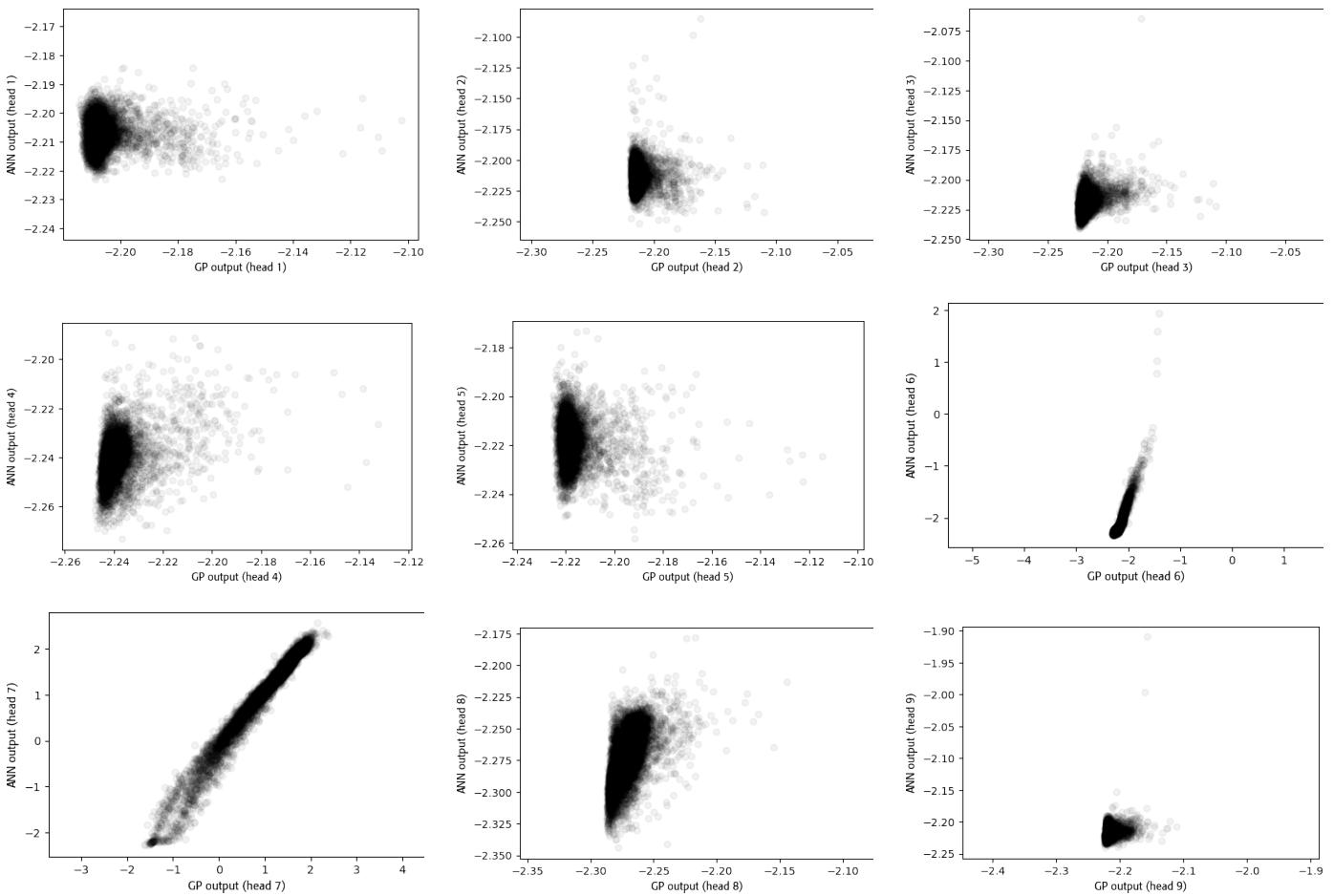


Fig. S3: Scatters for Cifar10 (attention).

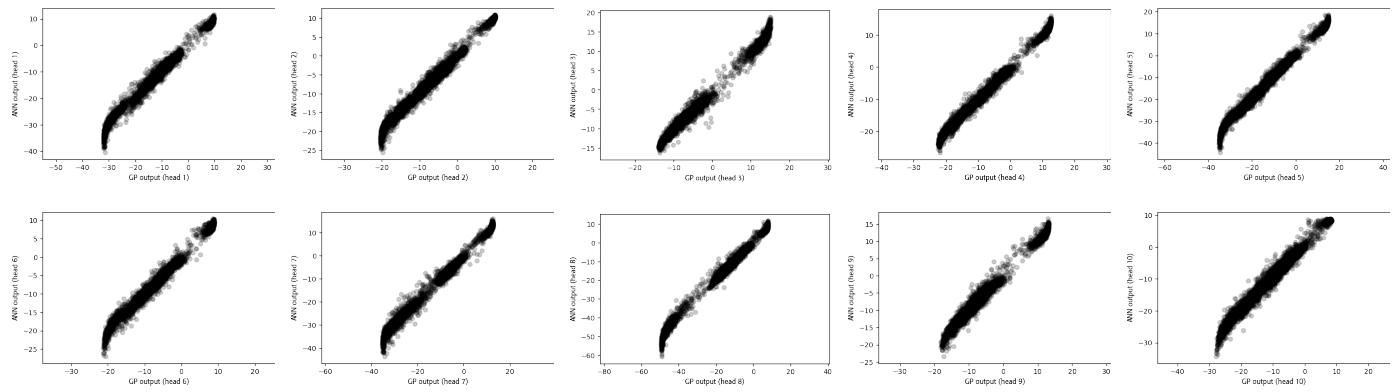


Fig. S4: Scatters for MNIST (classifier).

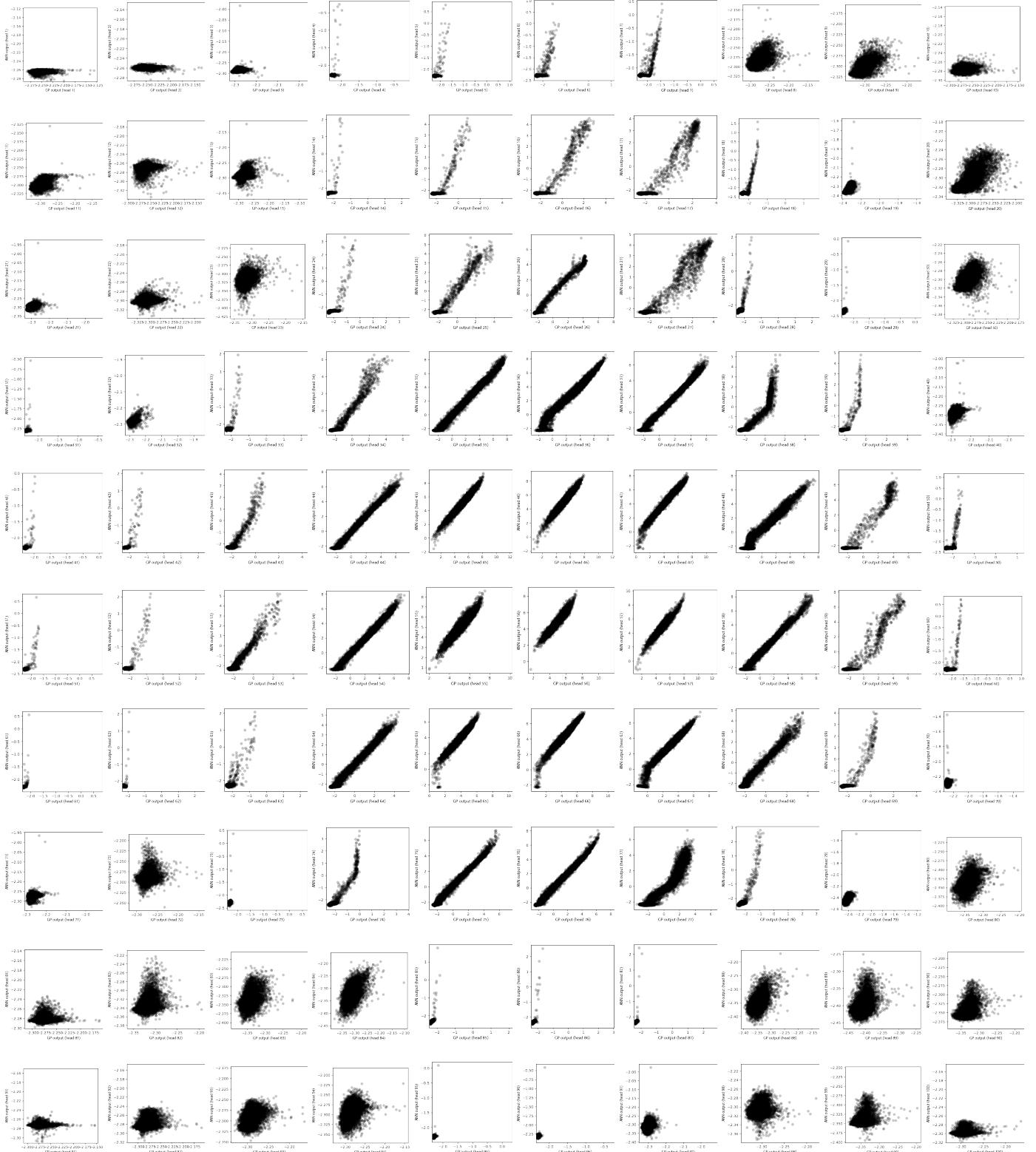


Fig. S5: Scatters for MNIST (attention).

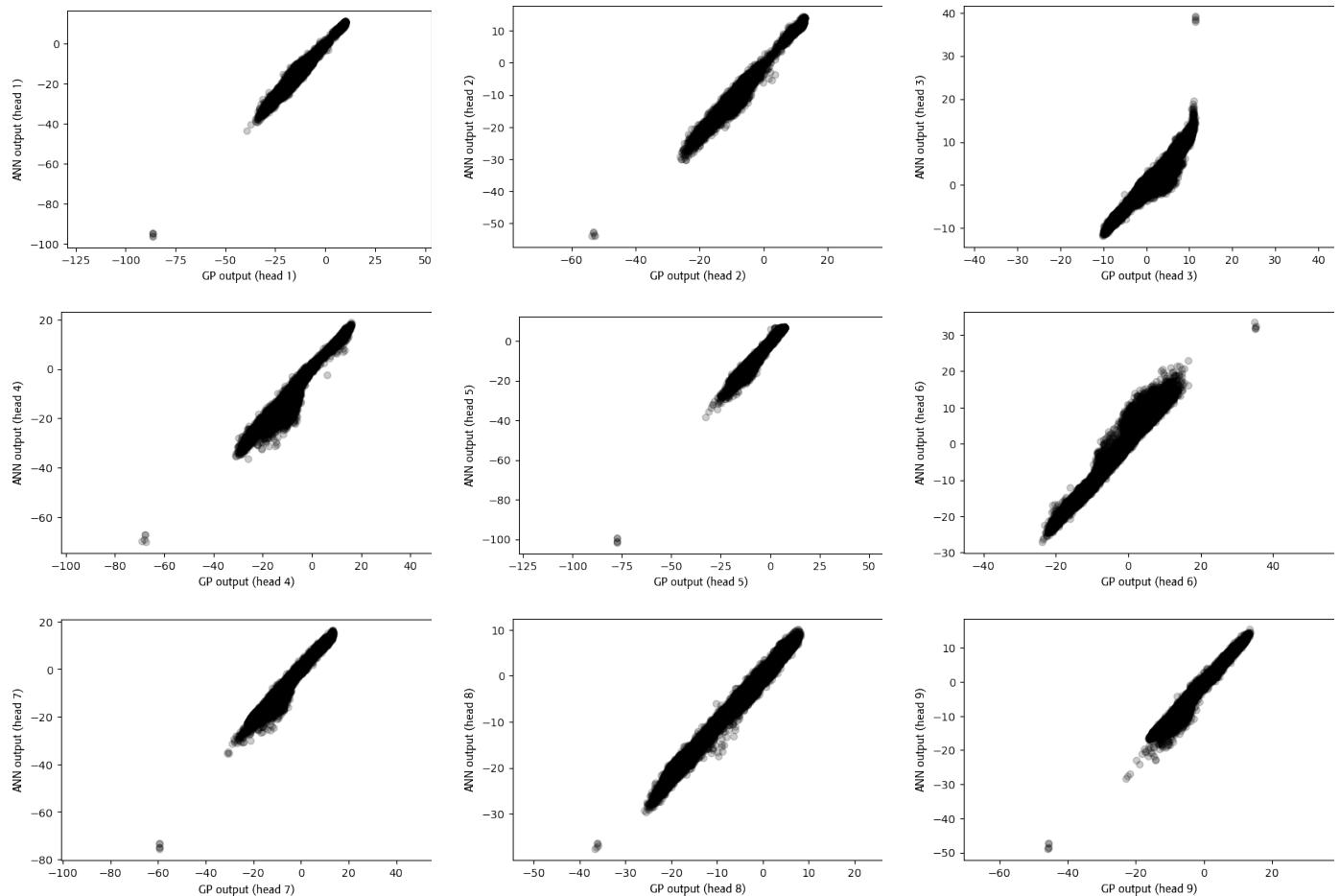


Fig. S6: Scatters for Kather dataset (classifier).

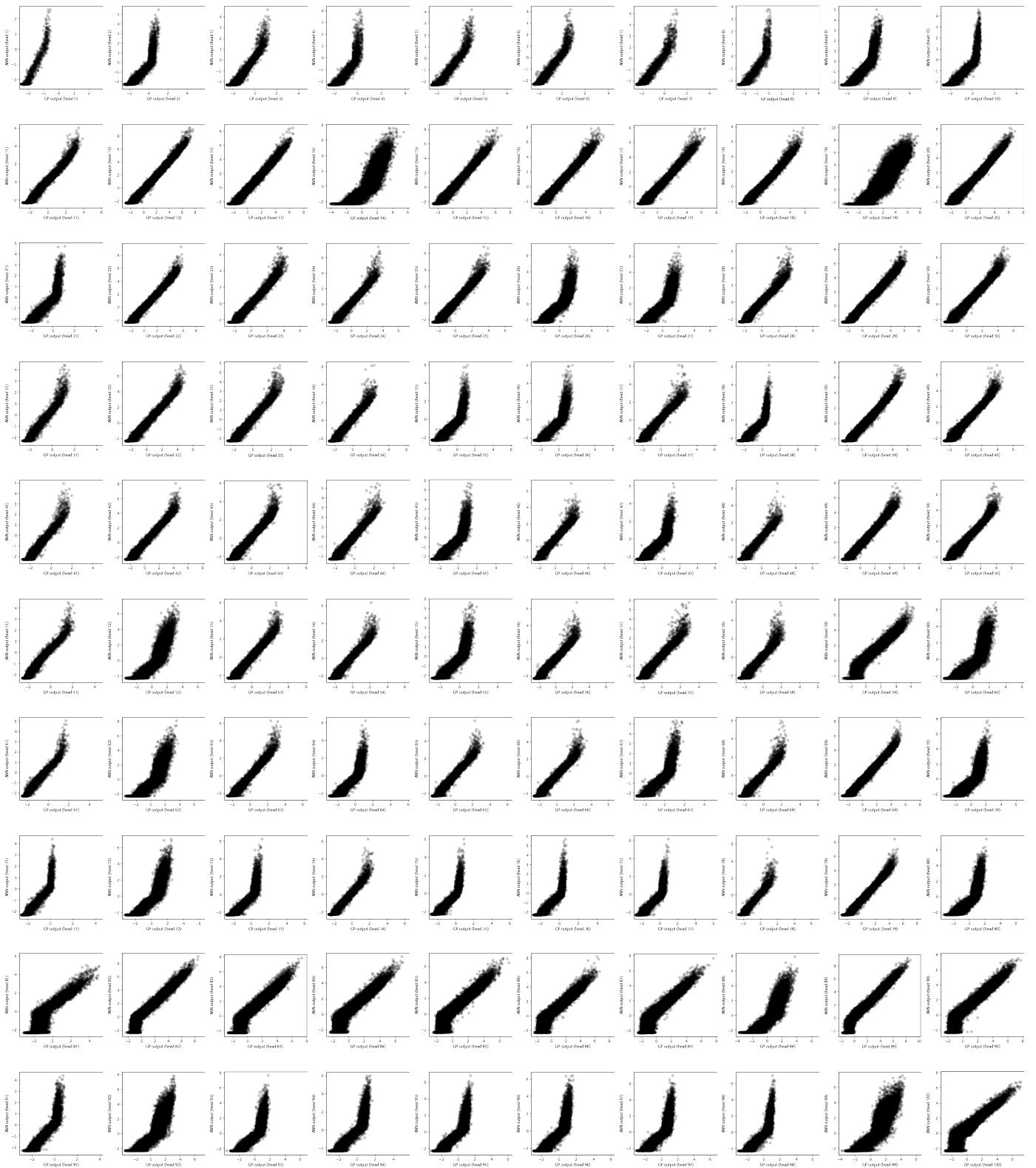


Fig. S7: Scatters for Kather dataset (attention).

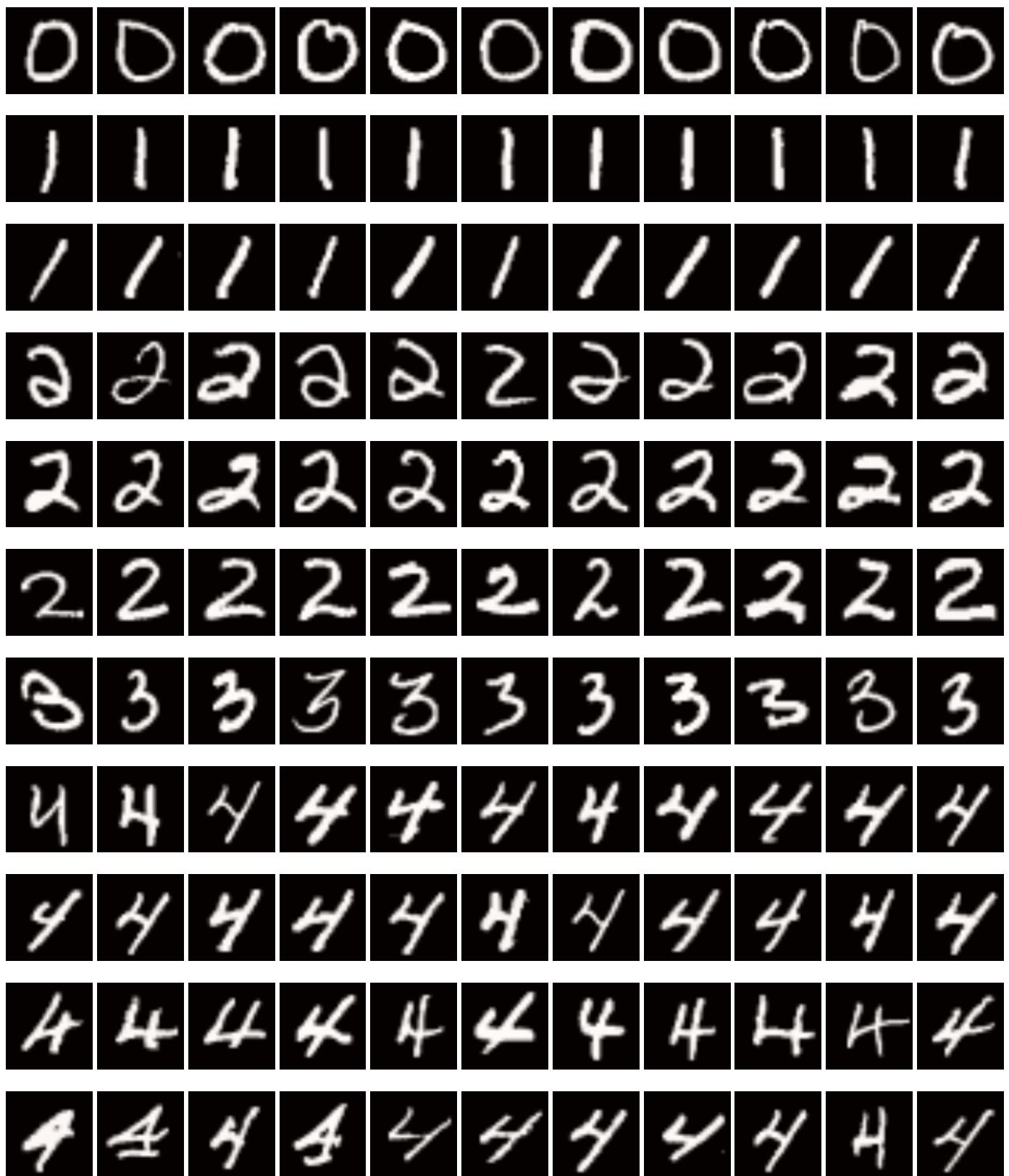


Fig. S8: Explanations for MNIST (set 1).

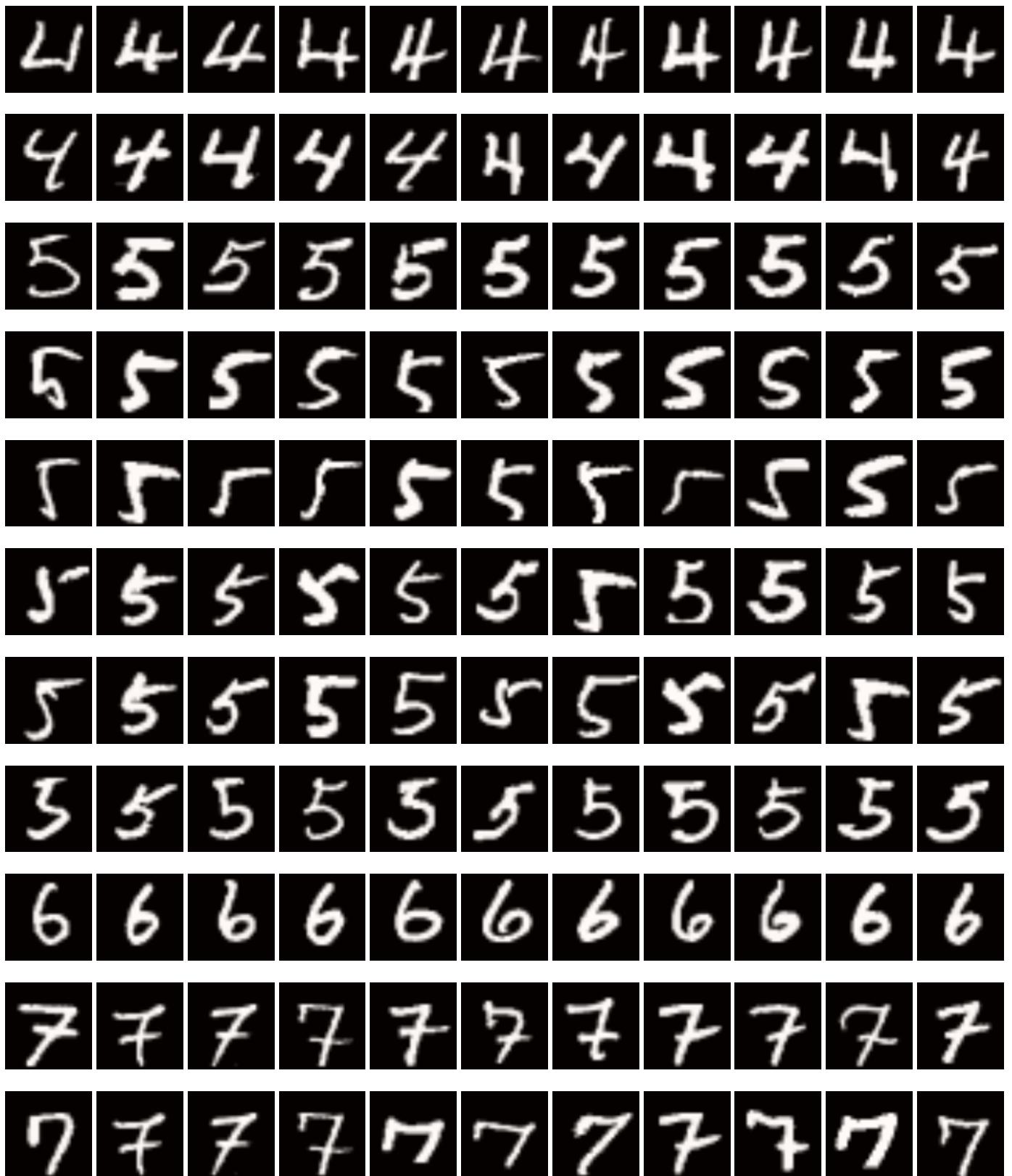


Fig. S9: Explanations for MNIST (set 2).



Fig. S10: Explanations for MNIST (set 3).



Fig. S11: Explanations for MNIST (set 4).

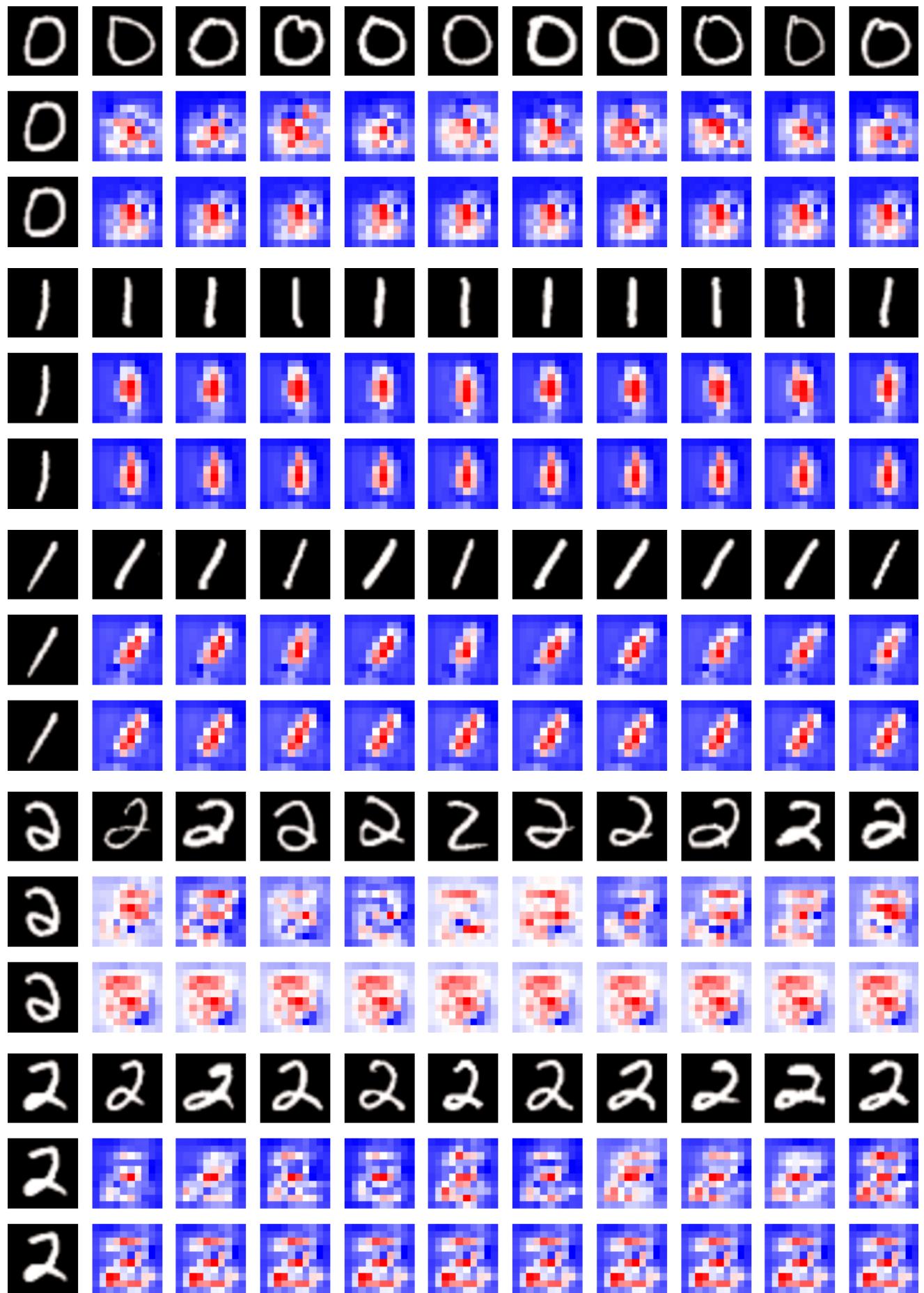


Fig. S12: Explanations for MNIST (set 5).

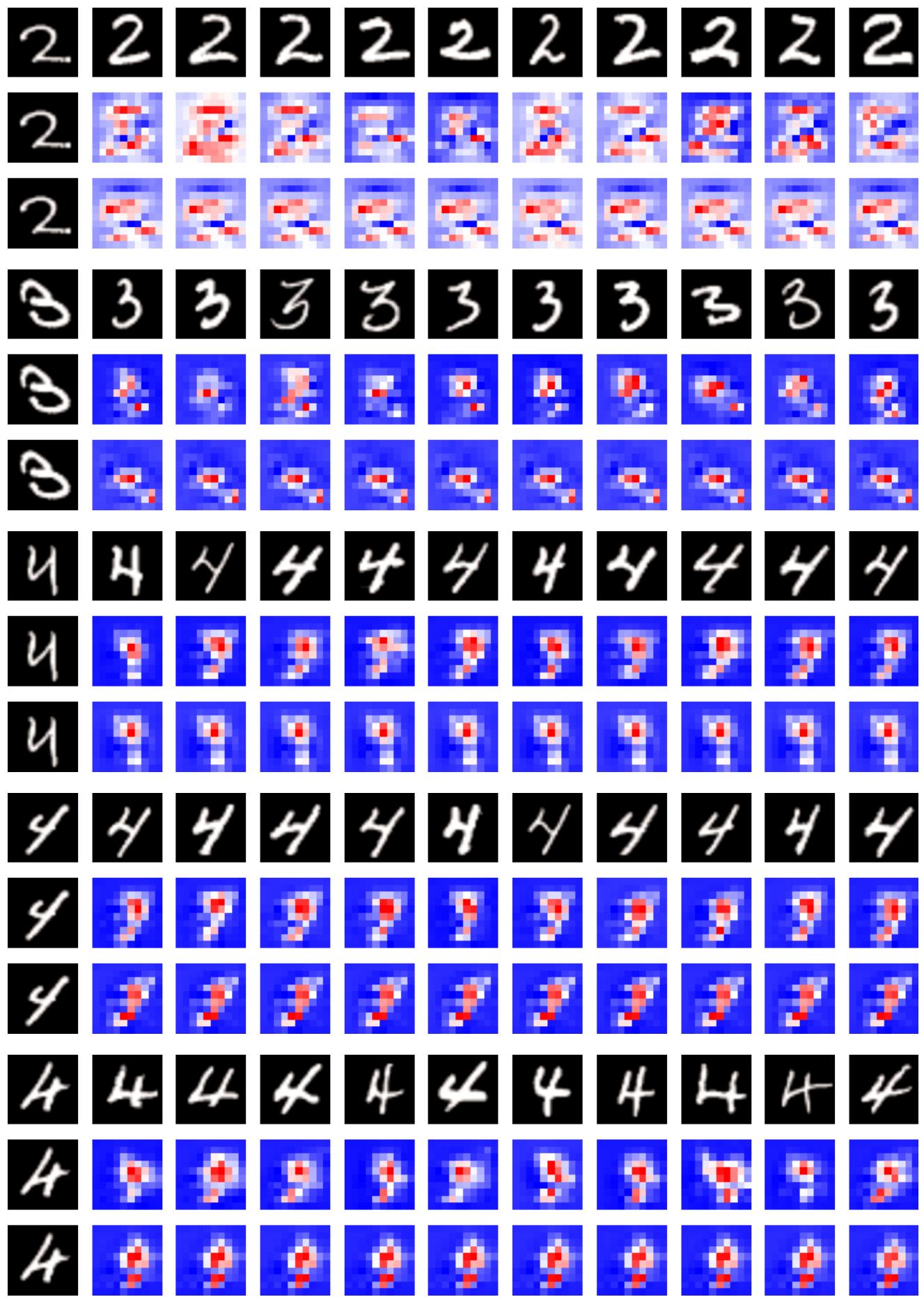


Fig. S13: Explanations for MNIST (set 6).

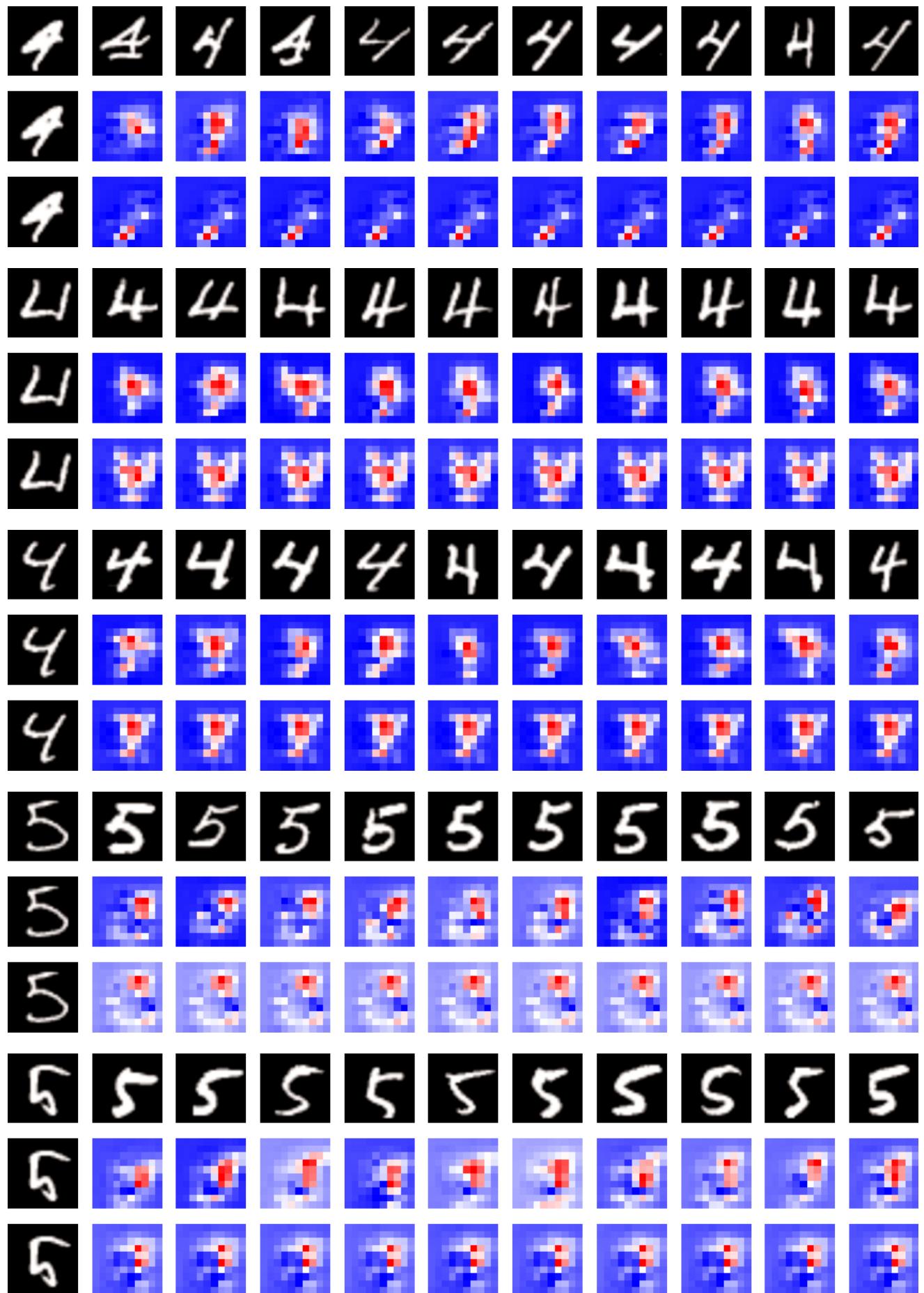


Fig. S14: Explanations for MNIST (set 7).

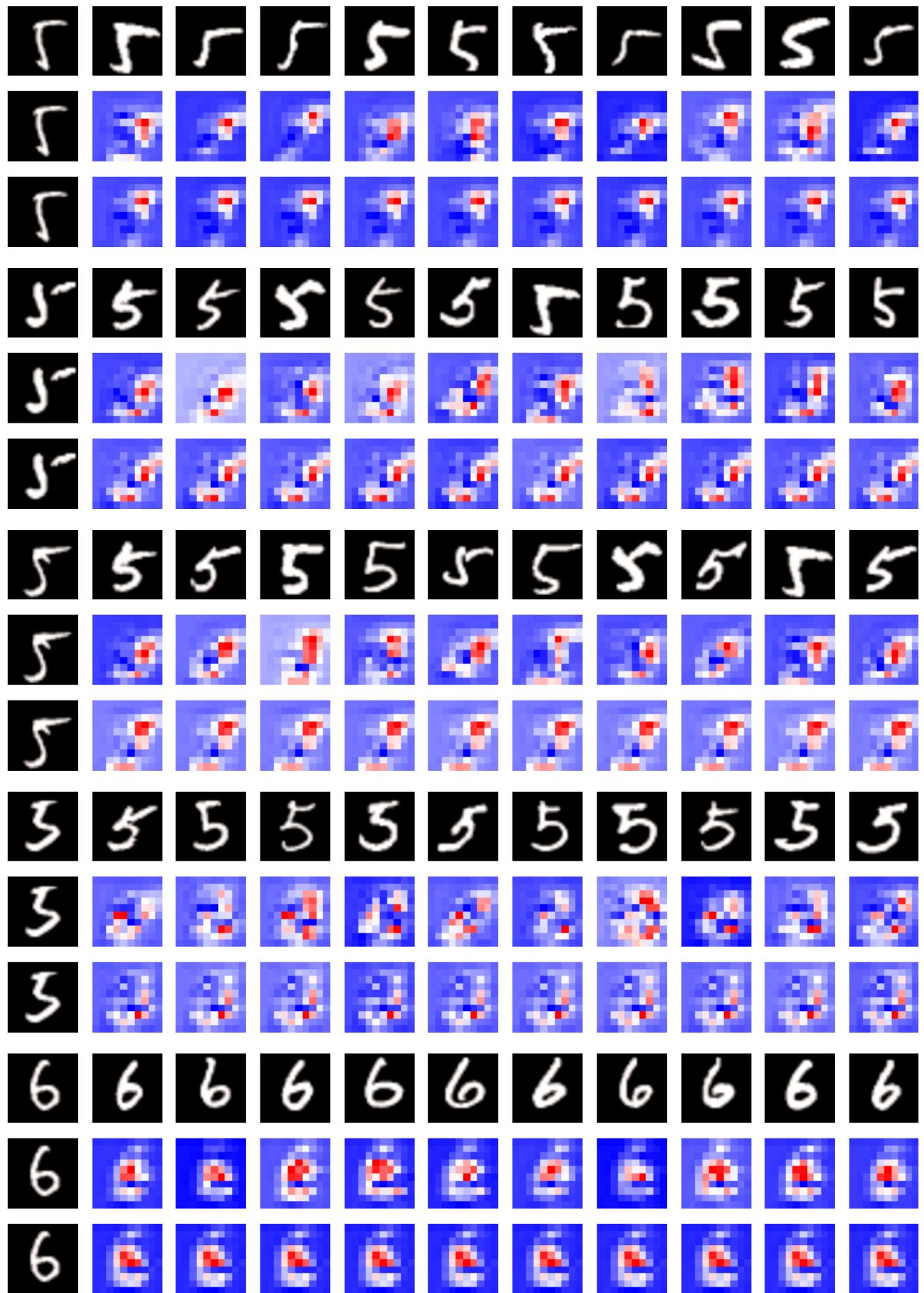


Fig. S15: Explanations for MNIST (set 8).

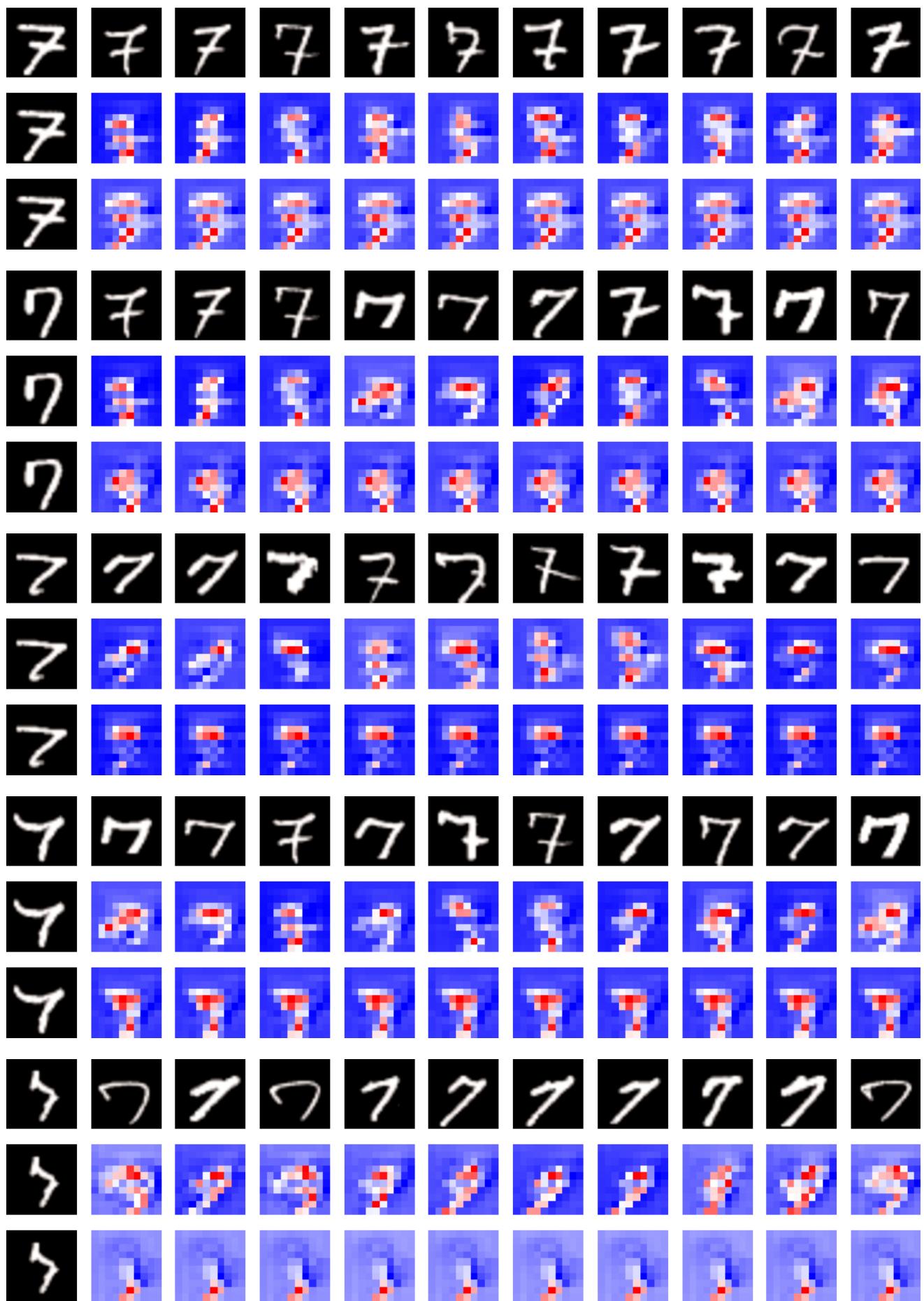


Fig. S16: Explanations for MNIST (set 9).

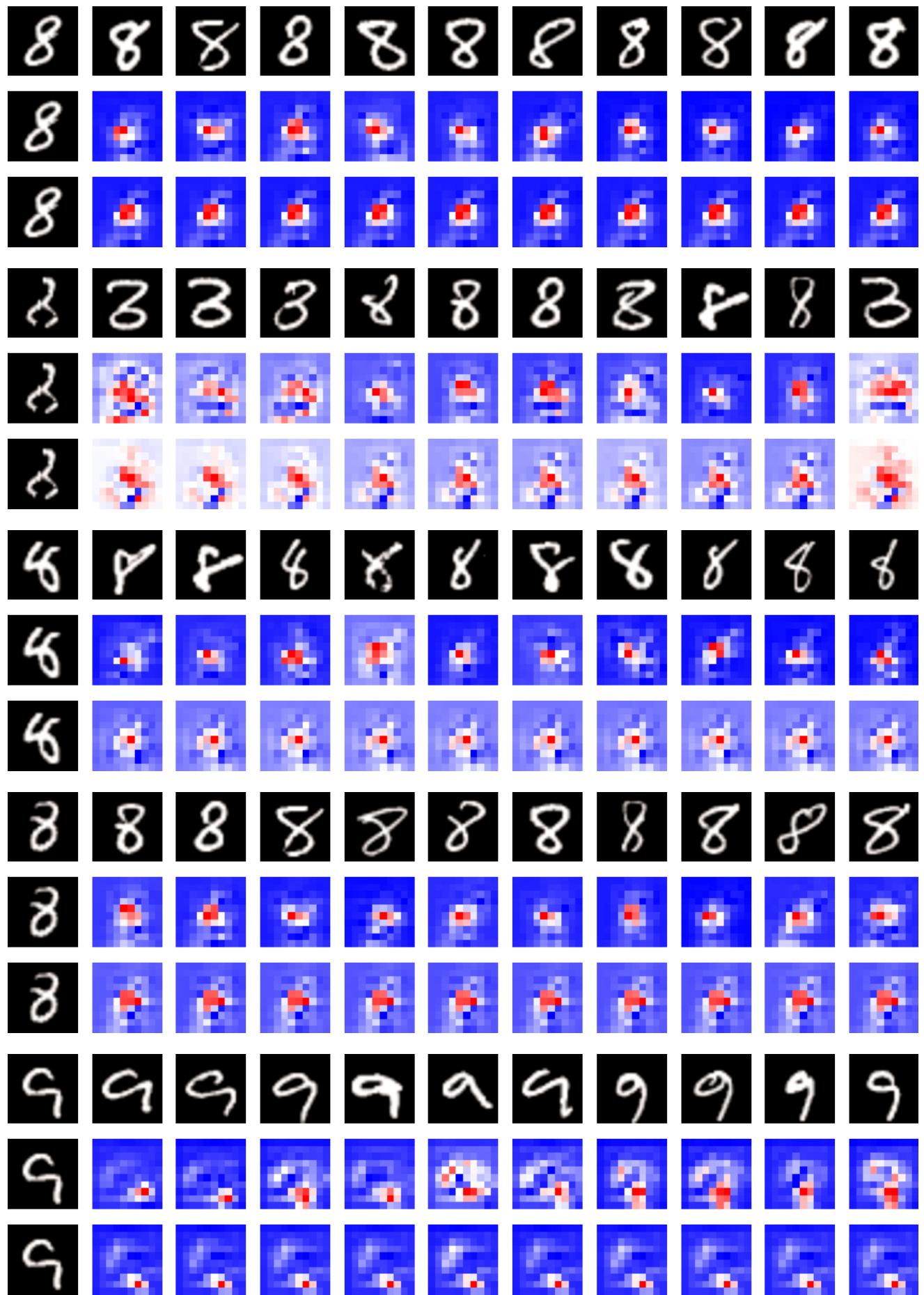


Fig. S17: Explanations for MNIST (set 10).

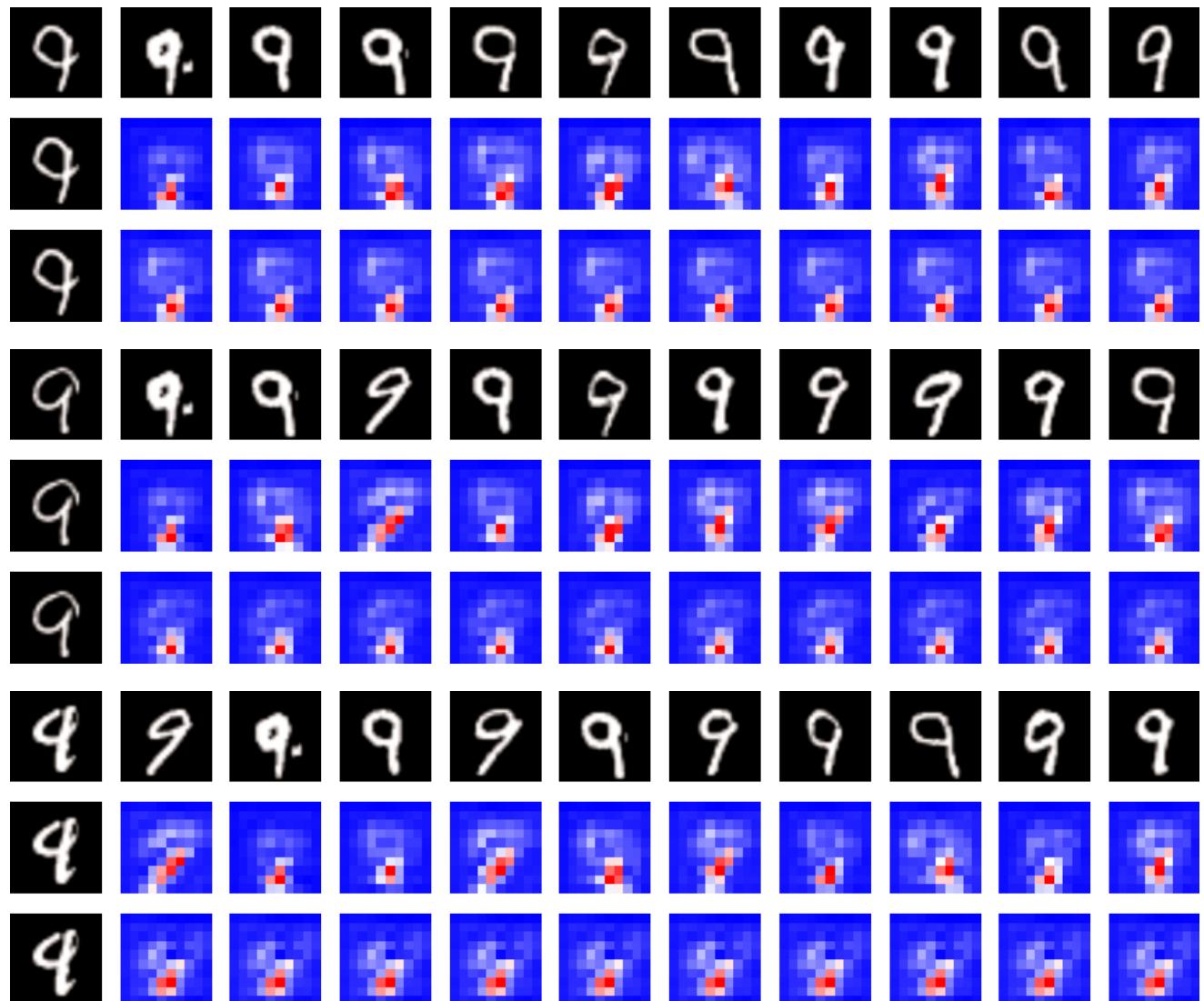


Fig. S18: Explanations for MNIST (set 11).

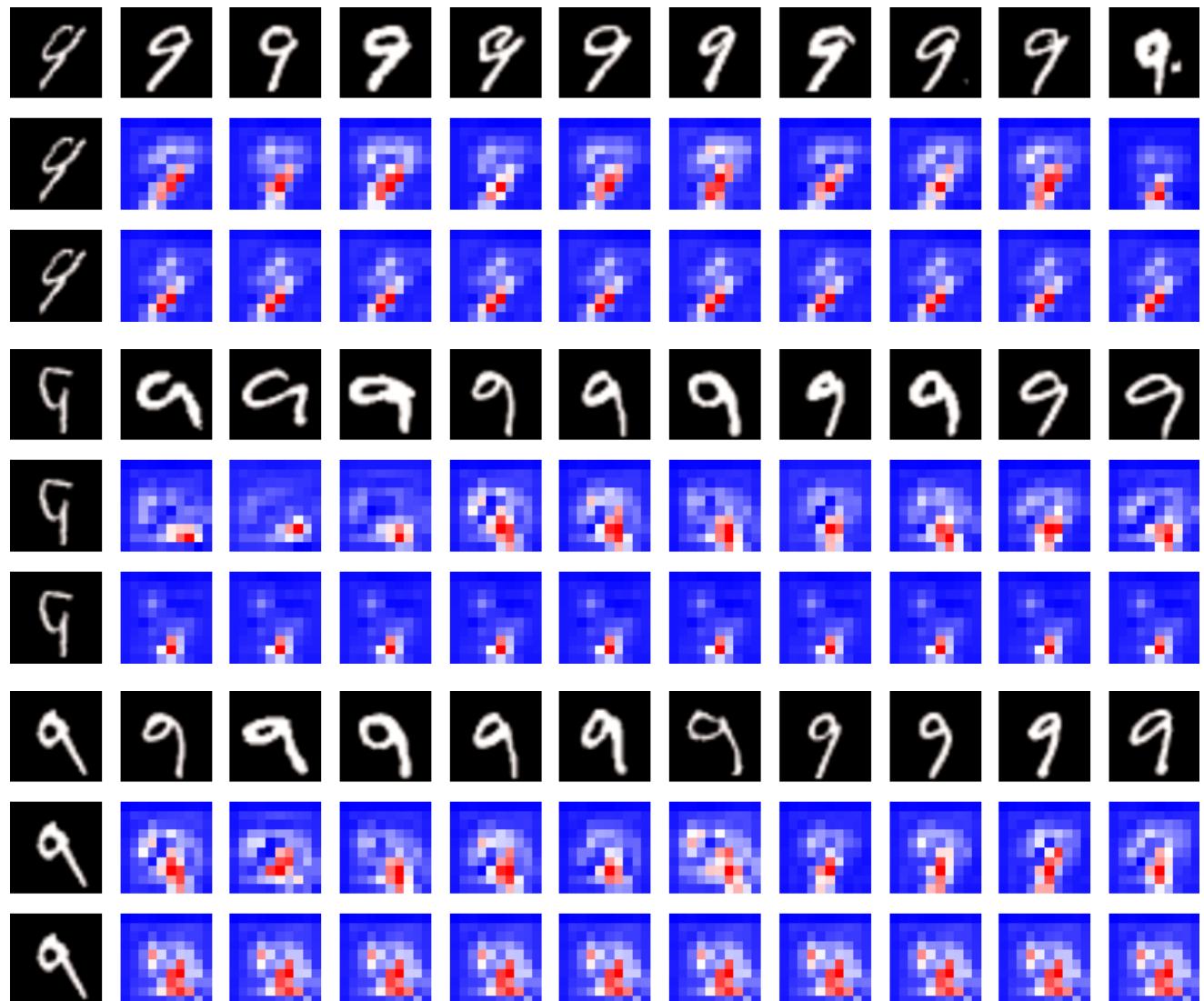


Fig. S19: Explanations for MNIST (set 12).

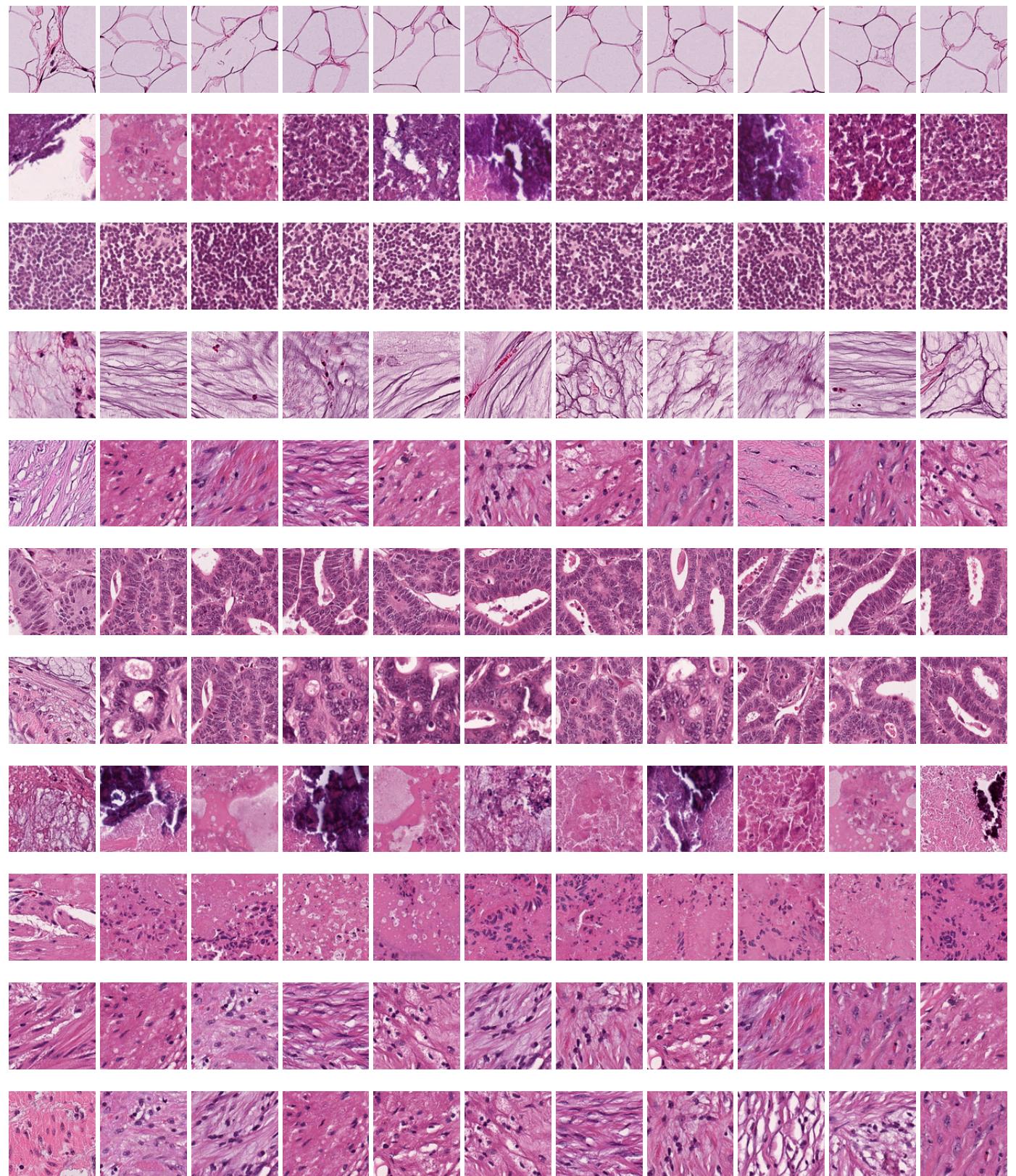


Fig. S20: Explanations for Kather dataset (set 1).

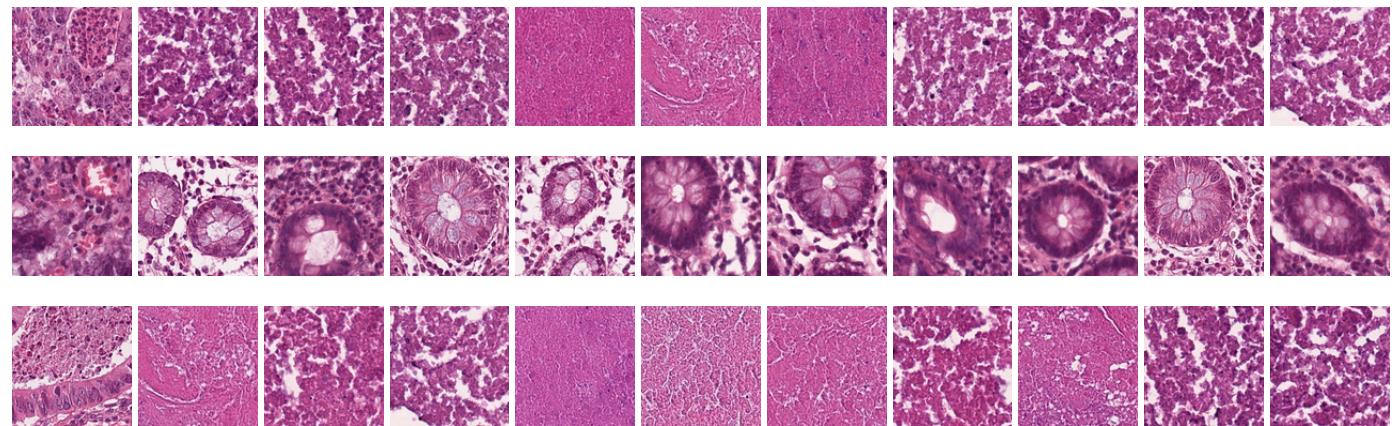


Fig. S21: Explanations for Kather dataset (set 2).

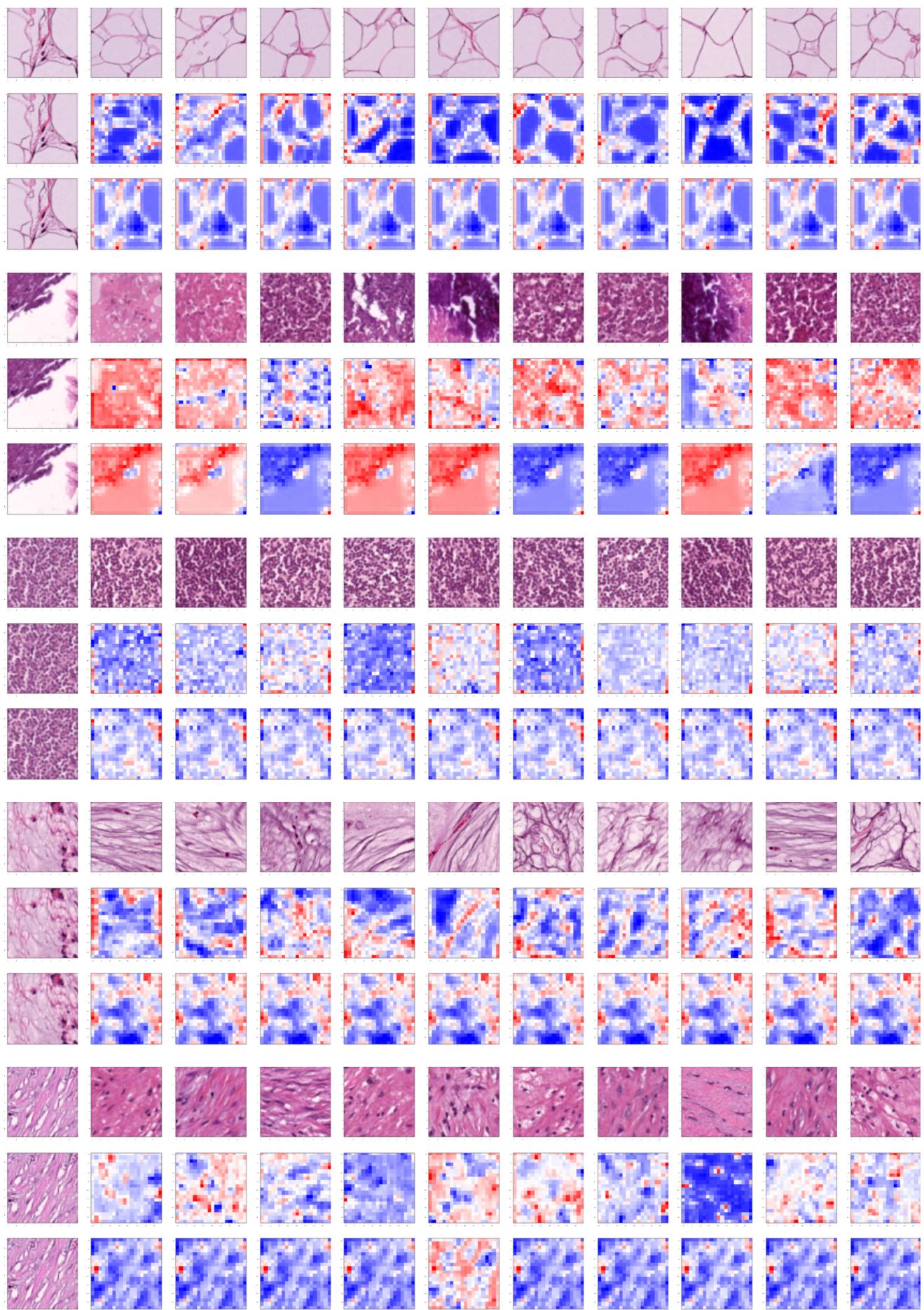


Fig. S22: Explanations for Kather dataset (set 3).

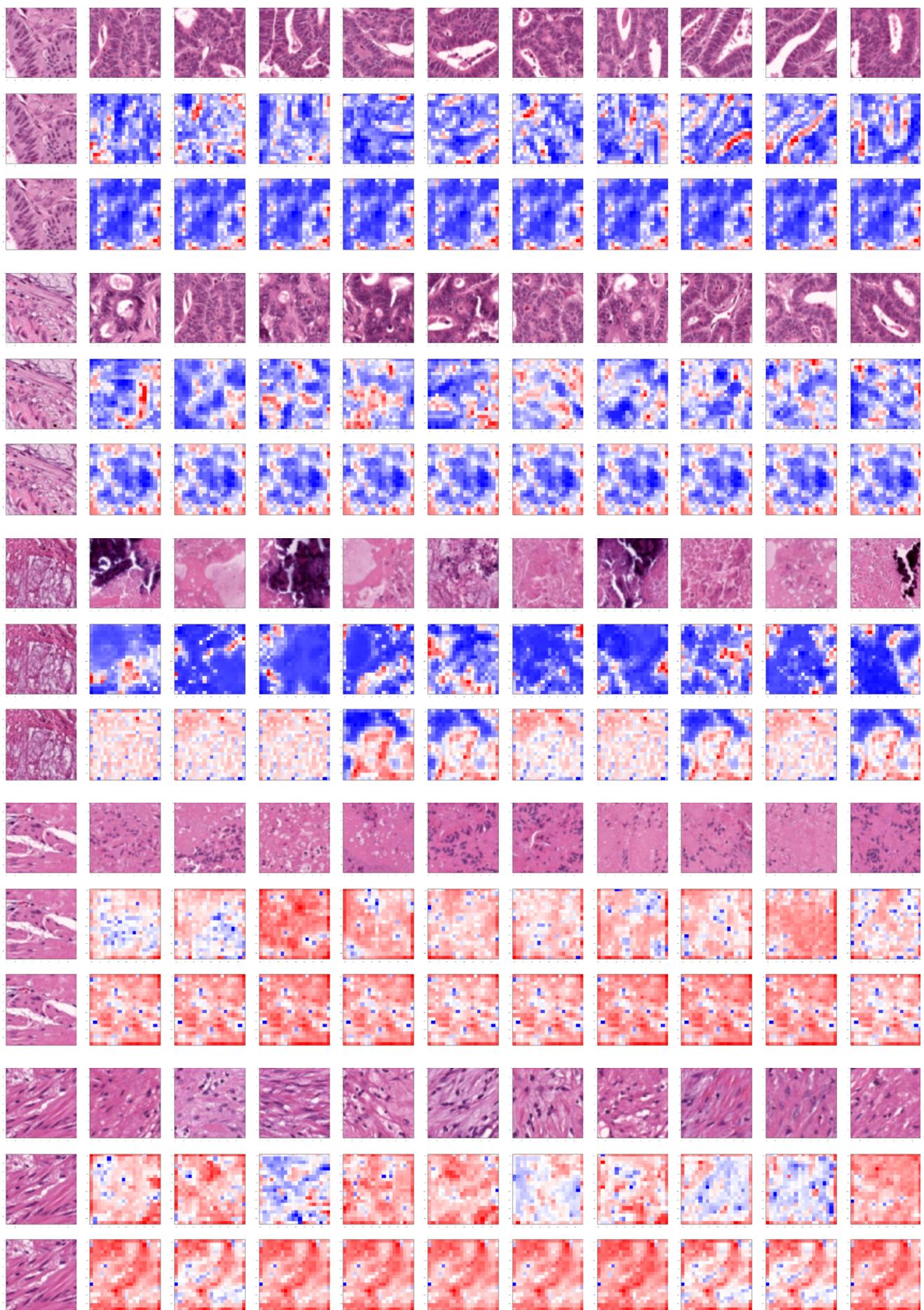


Fig. S23: Explanations for Kather dataset (set 4).

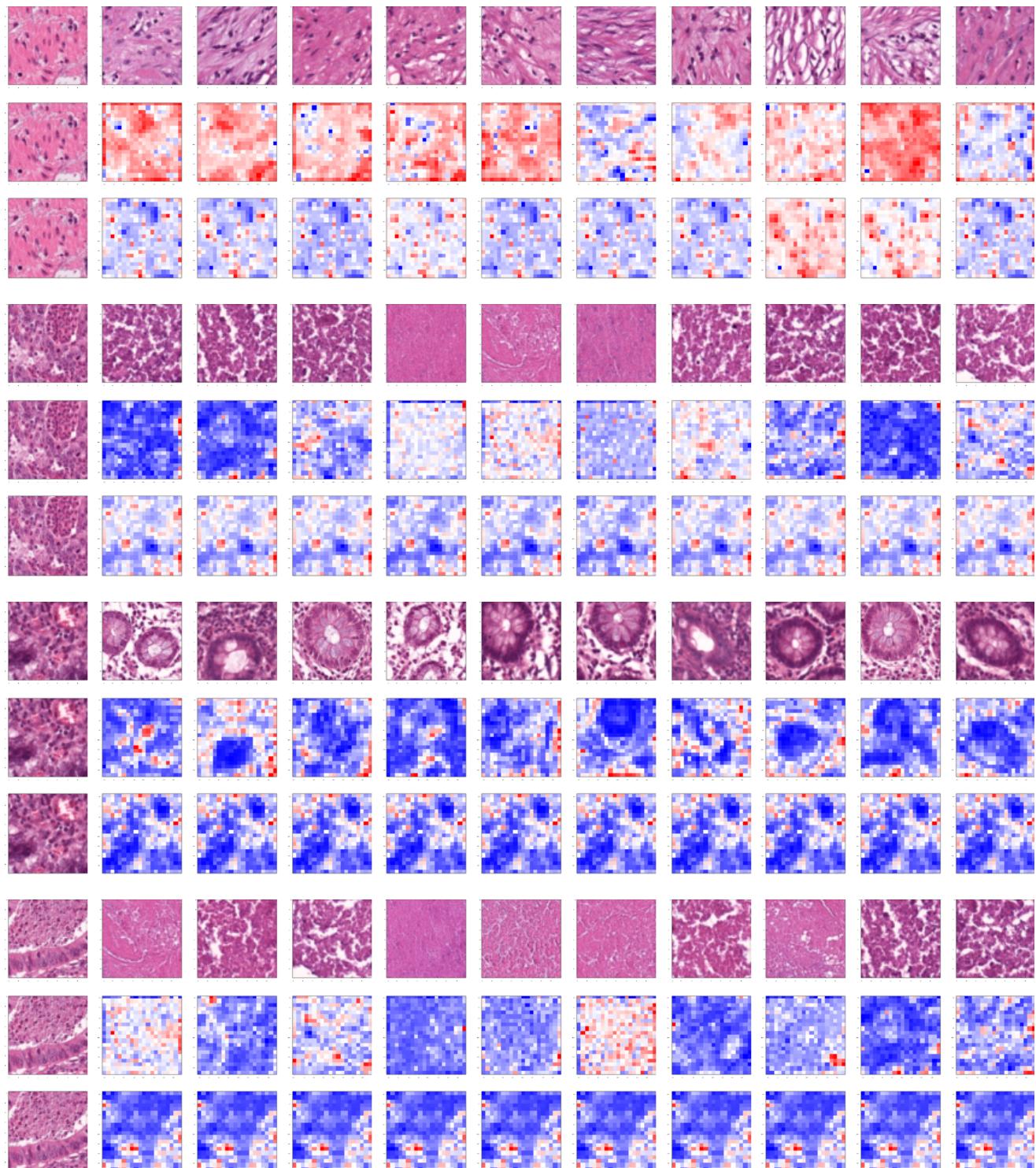


Fig. S24: Explanations for Kather dataset (set 5).

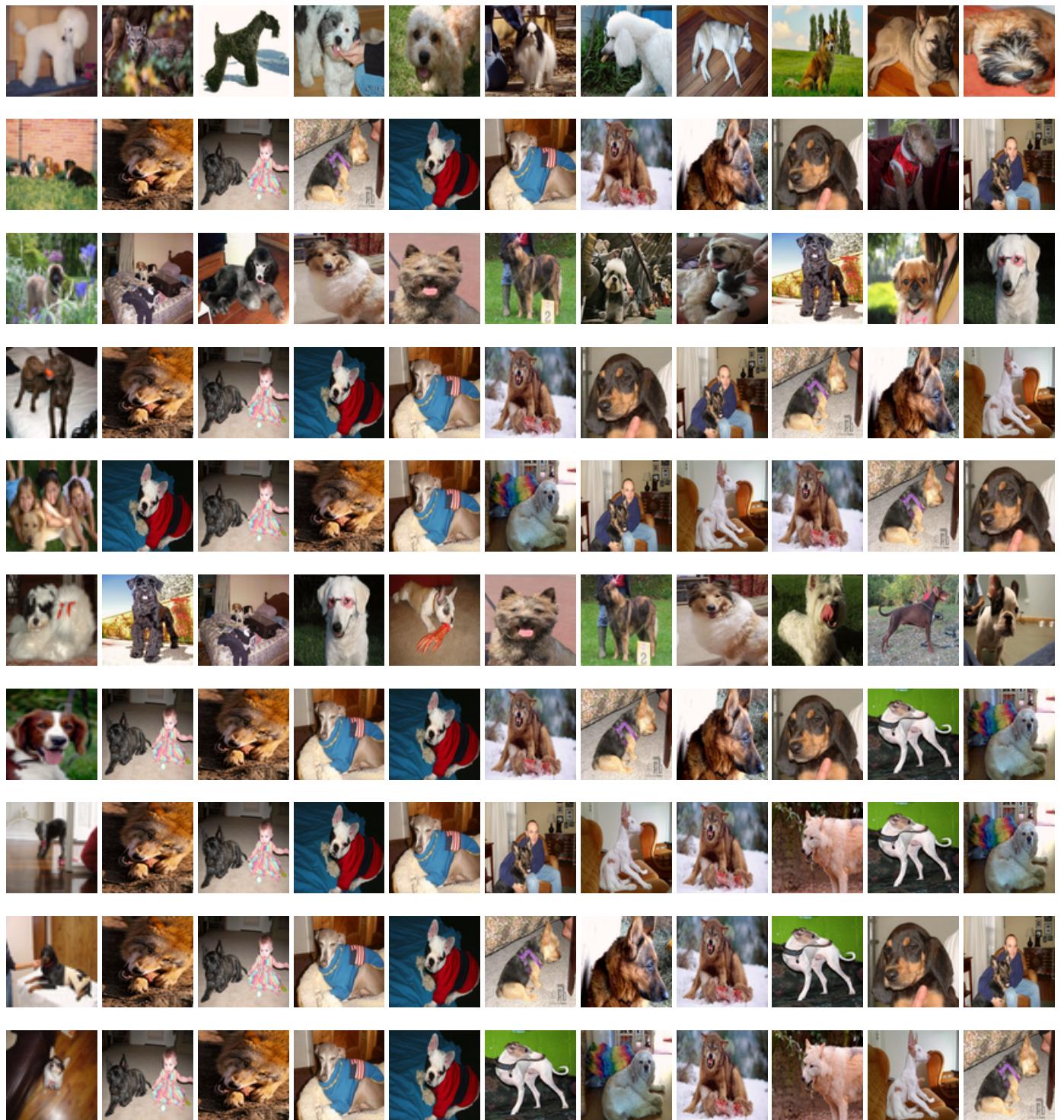


Fig. S25: Explanations for DogsWolves (set 1).

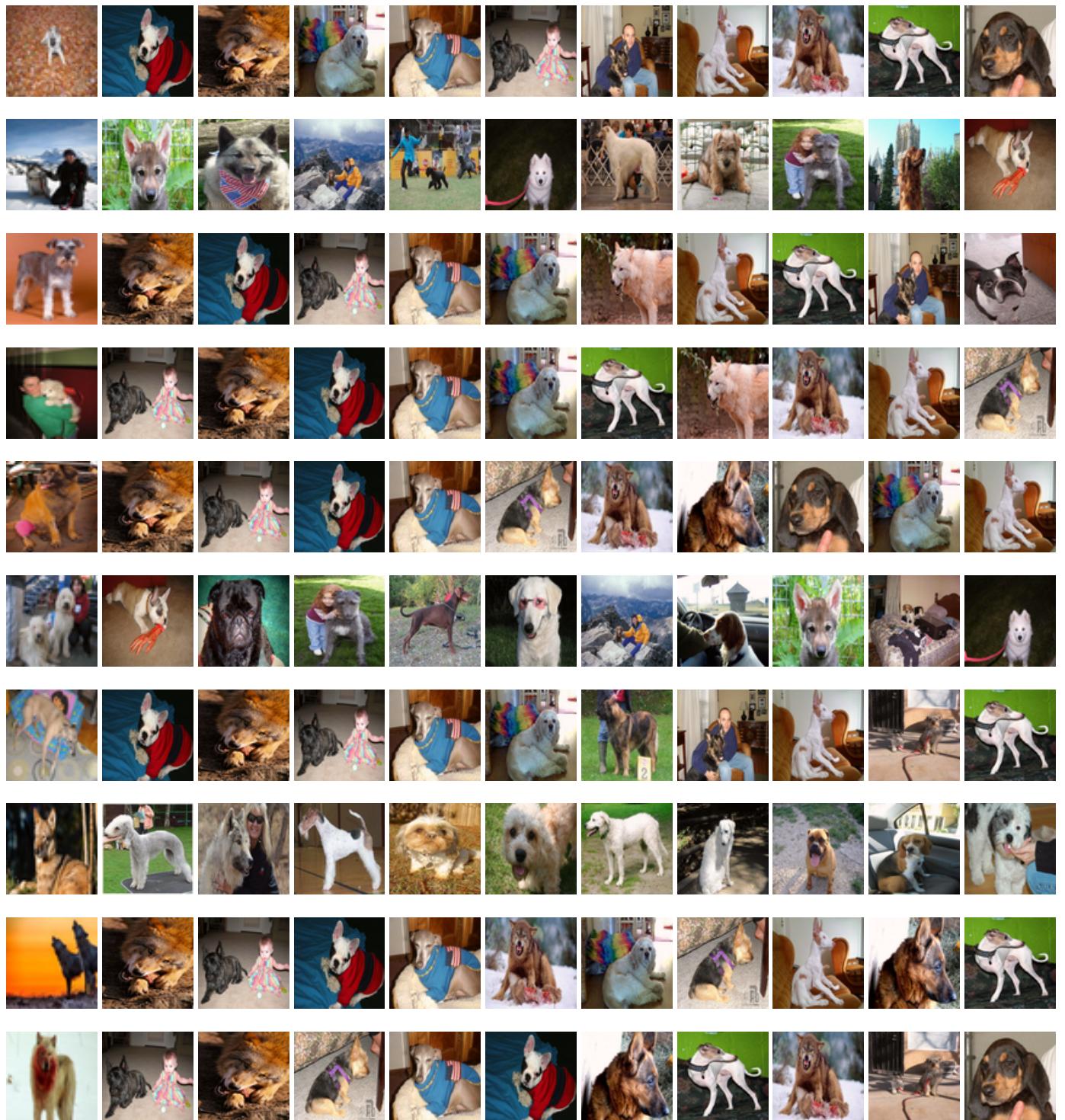


Fig. S26: Explanations for DogsWolves (set 2).

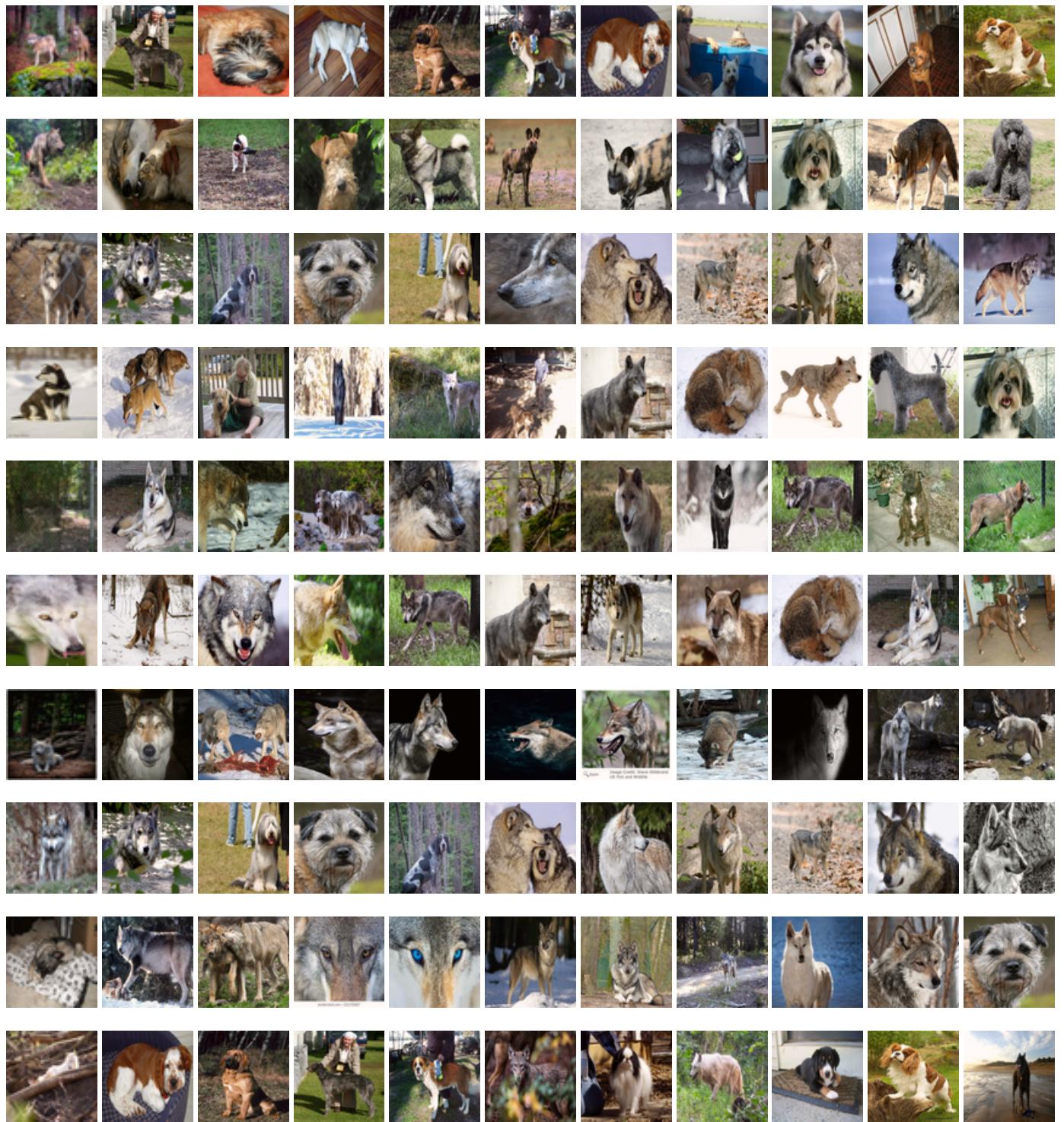


Fig. S27: Explanations for DogsWolves (set 3).

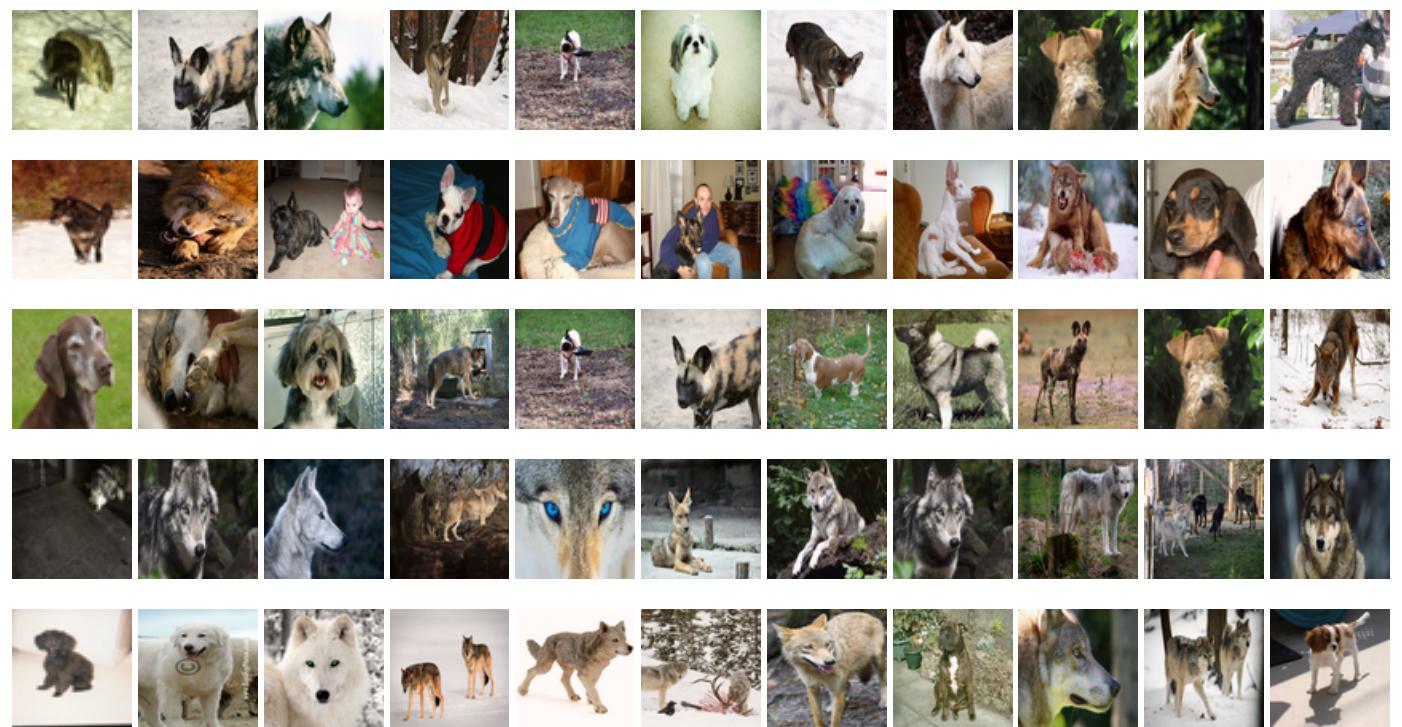


Fig. S28: Explanations for DogsWolves (set 4).



Fig. S29: Explanations for DogsWolves (set 5).

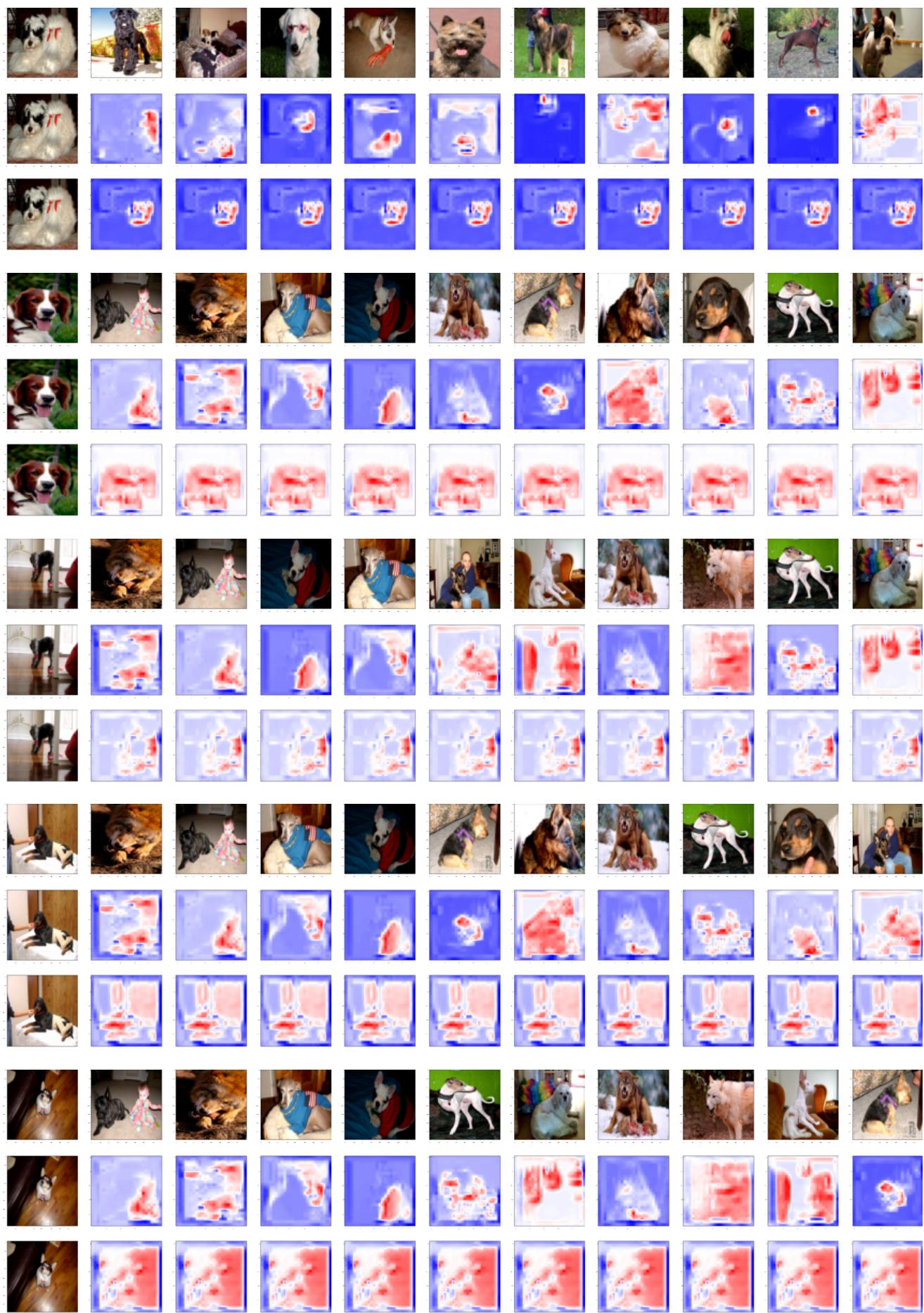


Fig. S30: Explanations for DogsWolves (set 6).



Fig. S31: Explanations for DogsWolves (set 7).

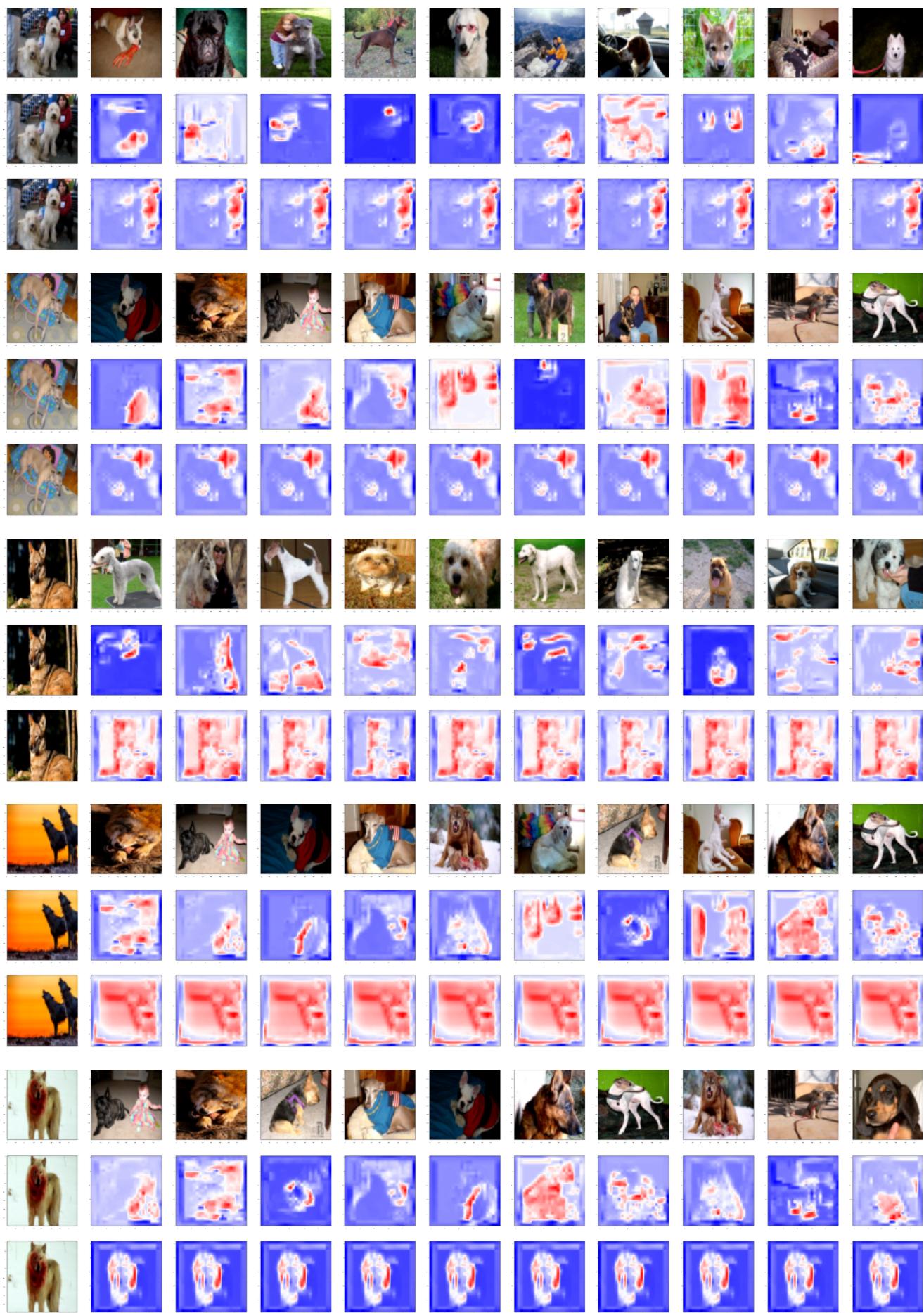


Fig. S32: Explanations for DogsWolves (set 8).



Fig. S33: Explanations for DogsWolves (set 9).



Fig. S34: Explanations for DogsWolves (set 10).



Fig. S35: Explanations for DogsWolves (set 11).



Fig. S36: Explanations for Cifar10 (set 1).



Fig. S37: Explanations for Cifar10 (set 2).

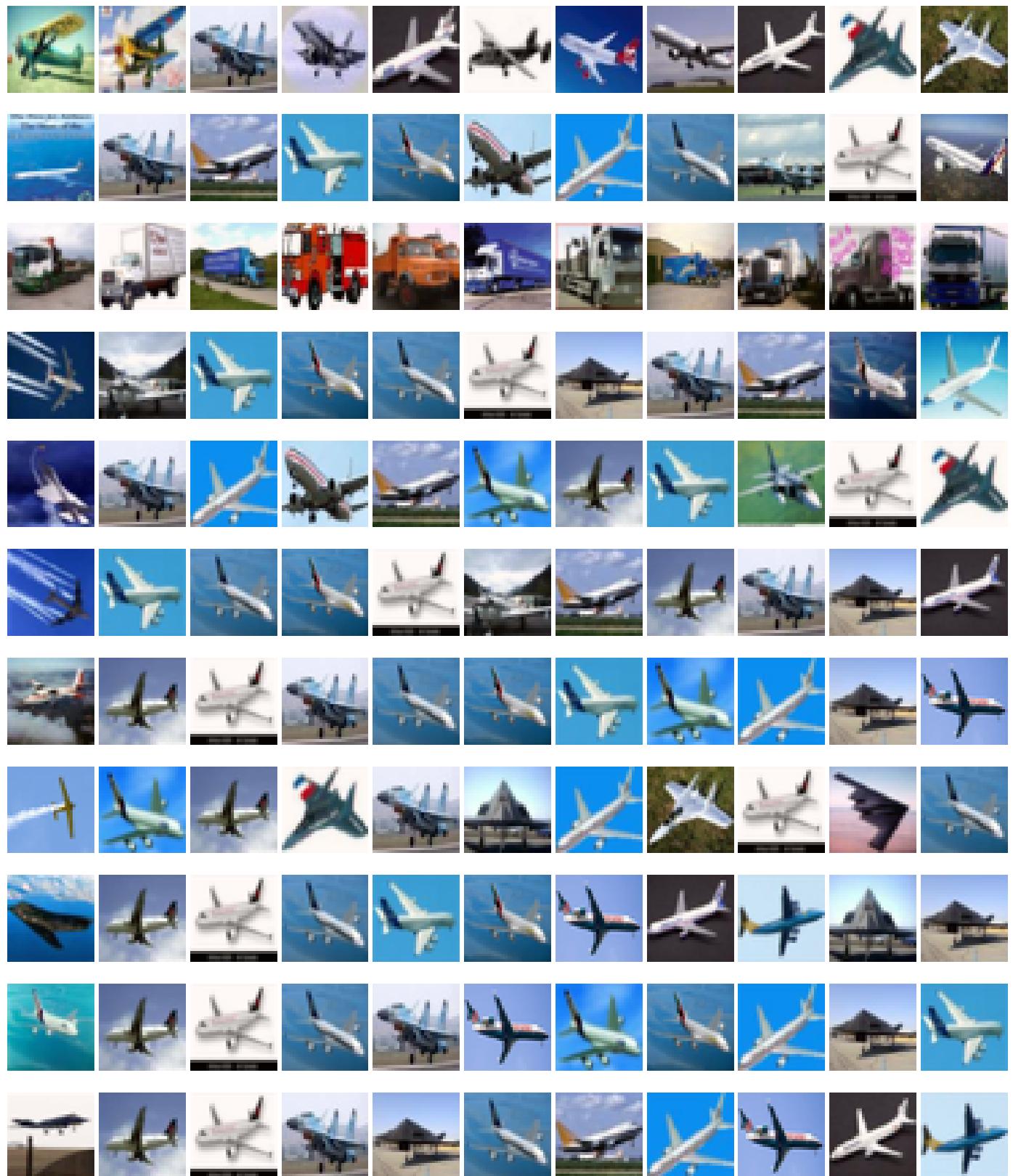


Fig. S38: Explanations for Cifar10 (set 3).

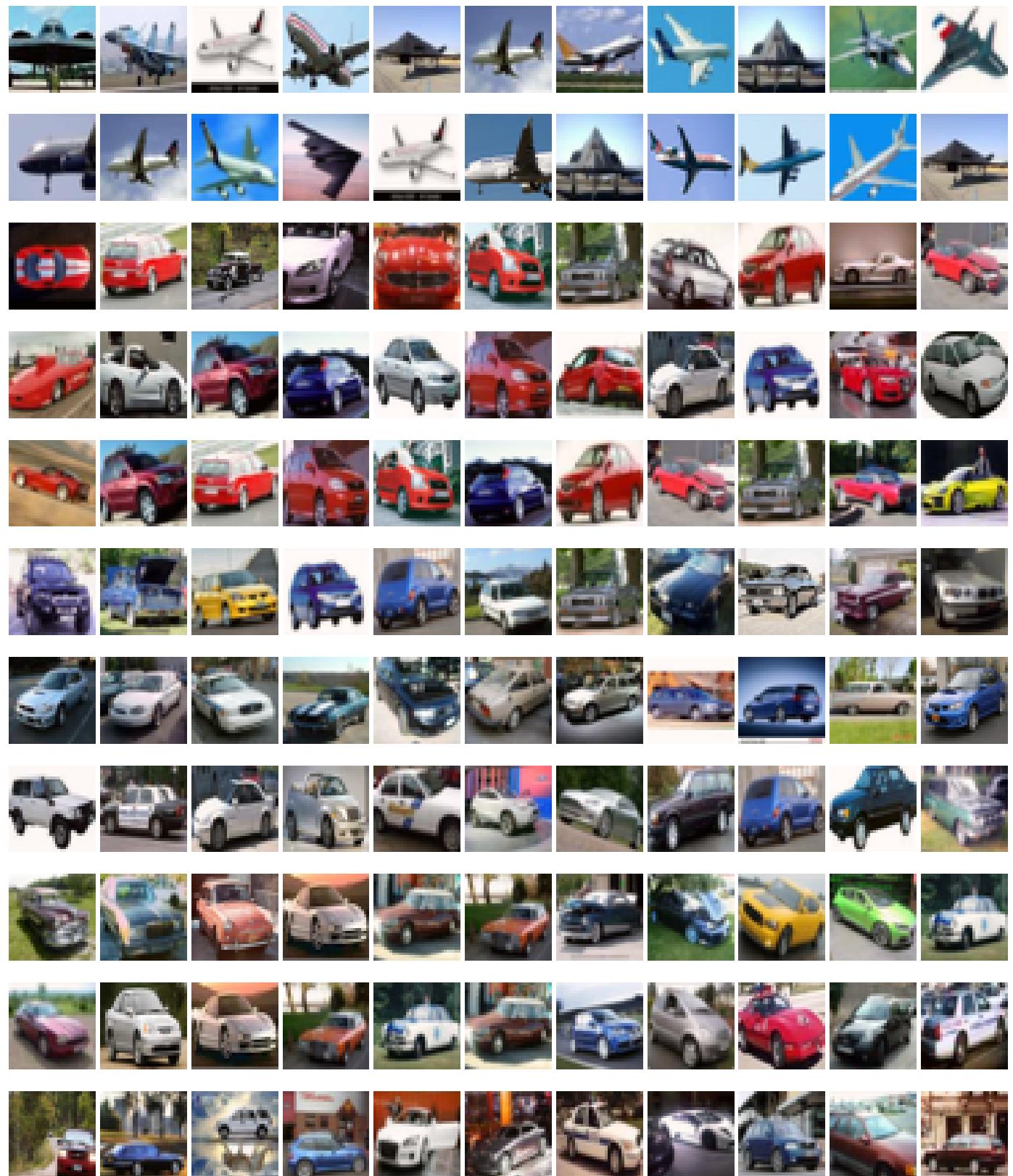


Fig. S39: Explanations for Cifar10 (set 4).



Fig. S40: Explanations for Cifar10 (set 5).



Fig. S41: Explanations for Cifar10 (set 6).

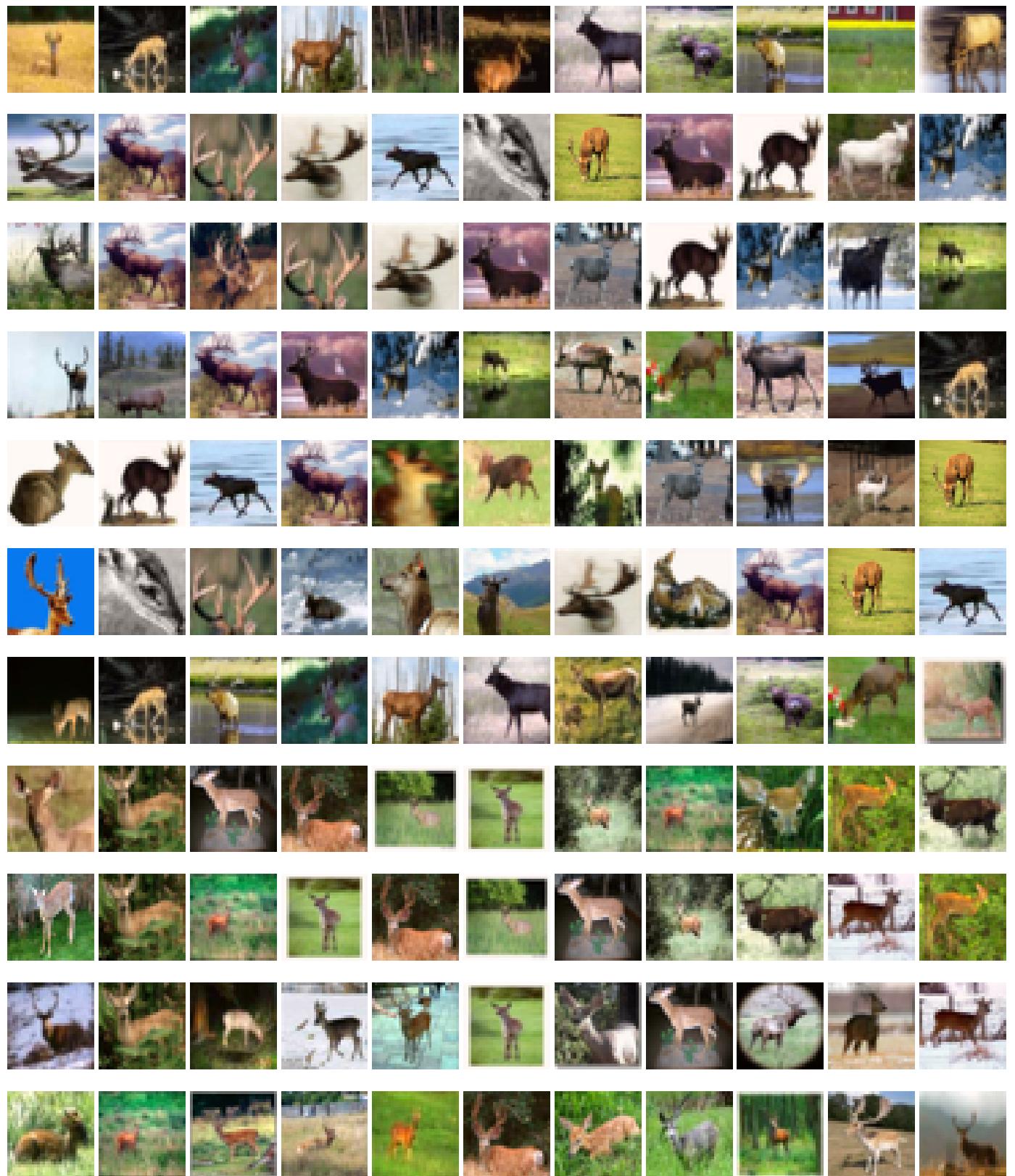


Fig. S42: Explanations for Cifar10 (set 7).

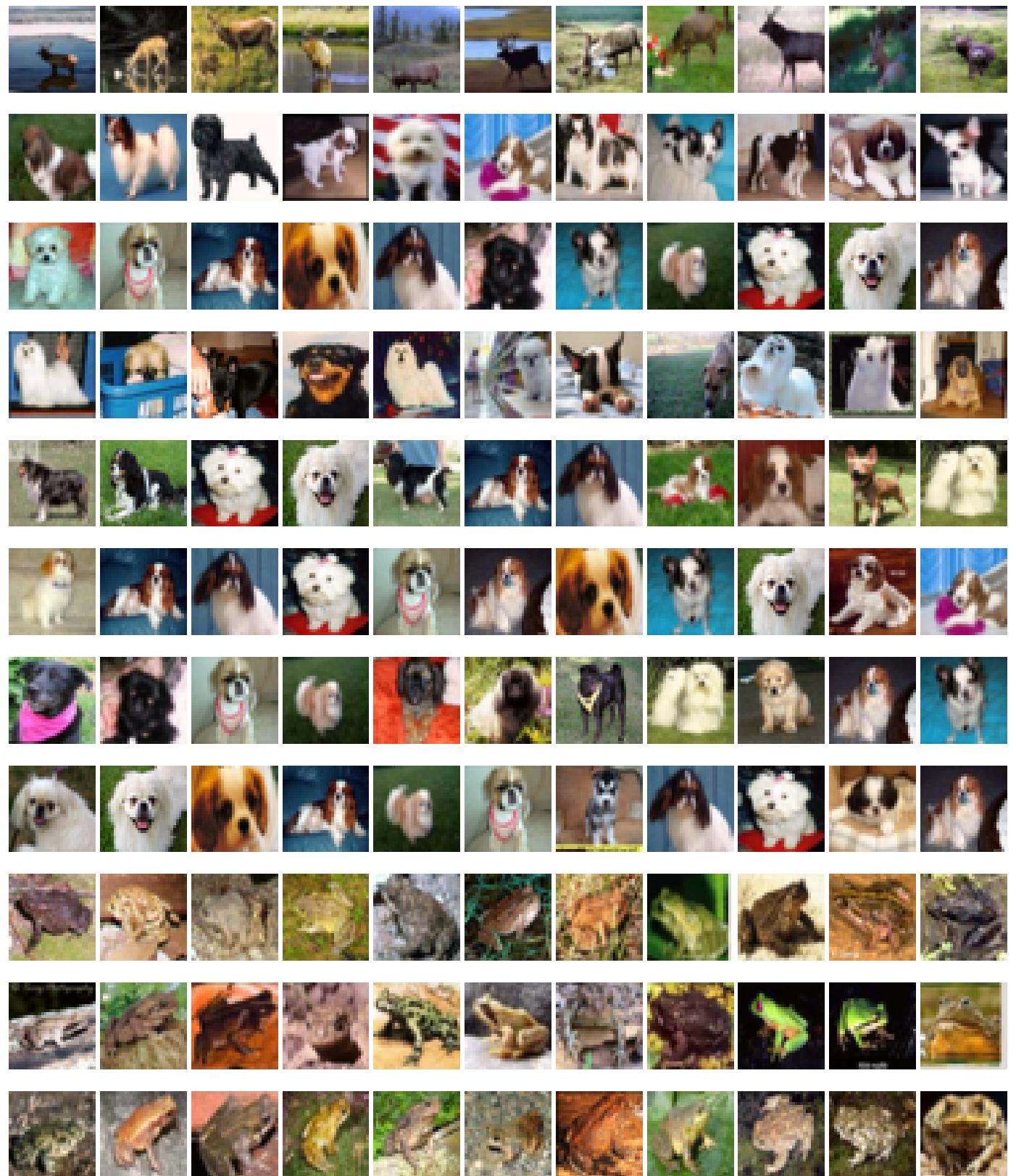


Fig. S43: Explanations for Cifar10 (set 8).



Fig. S44: Explanations for Cifar10 (set 9).

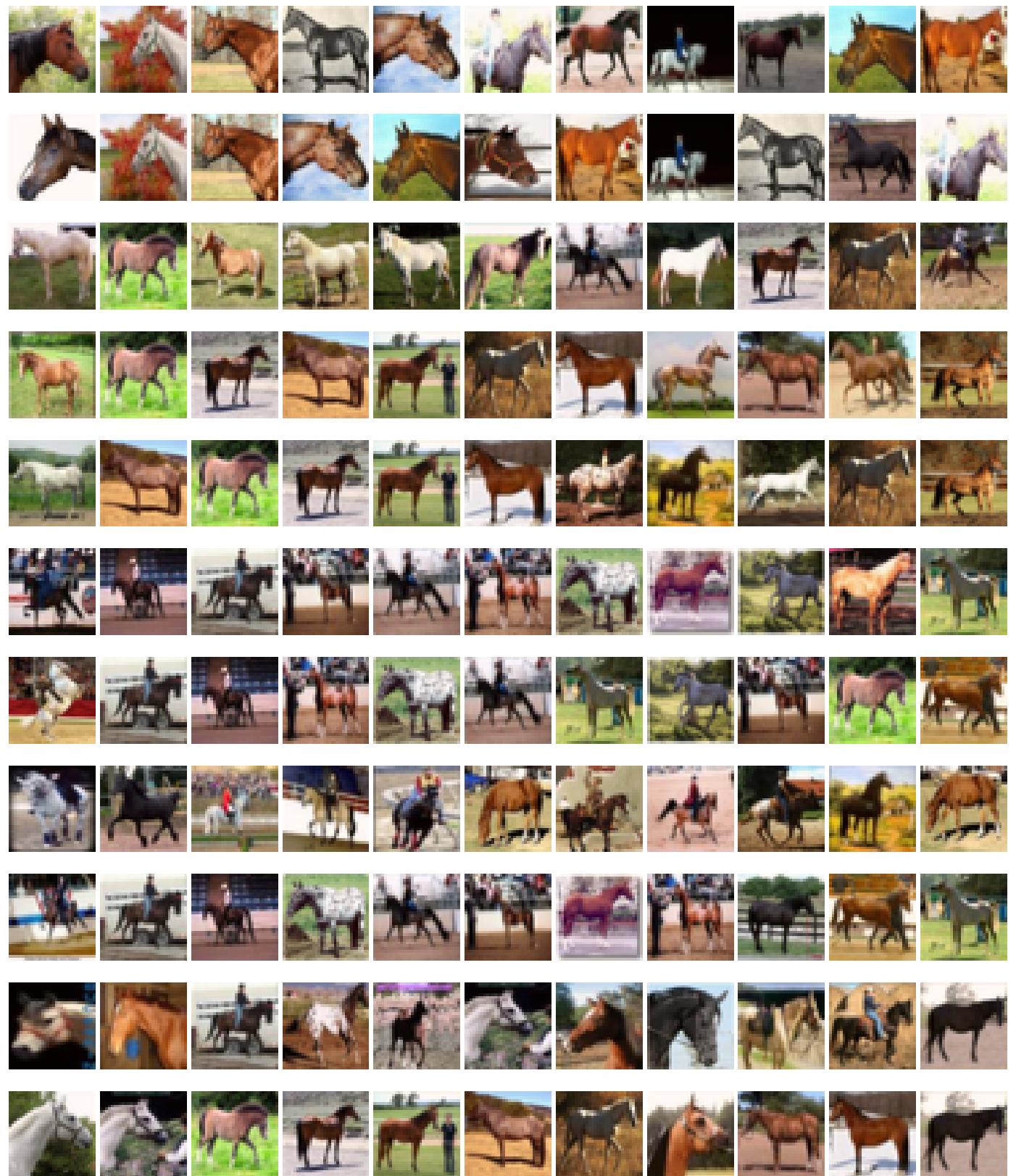


Fig. S45: Explanations for Cifar10 (set 10).

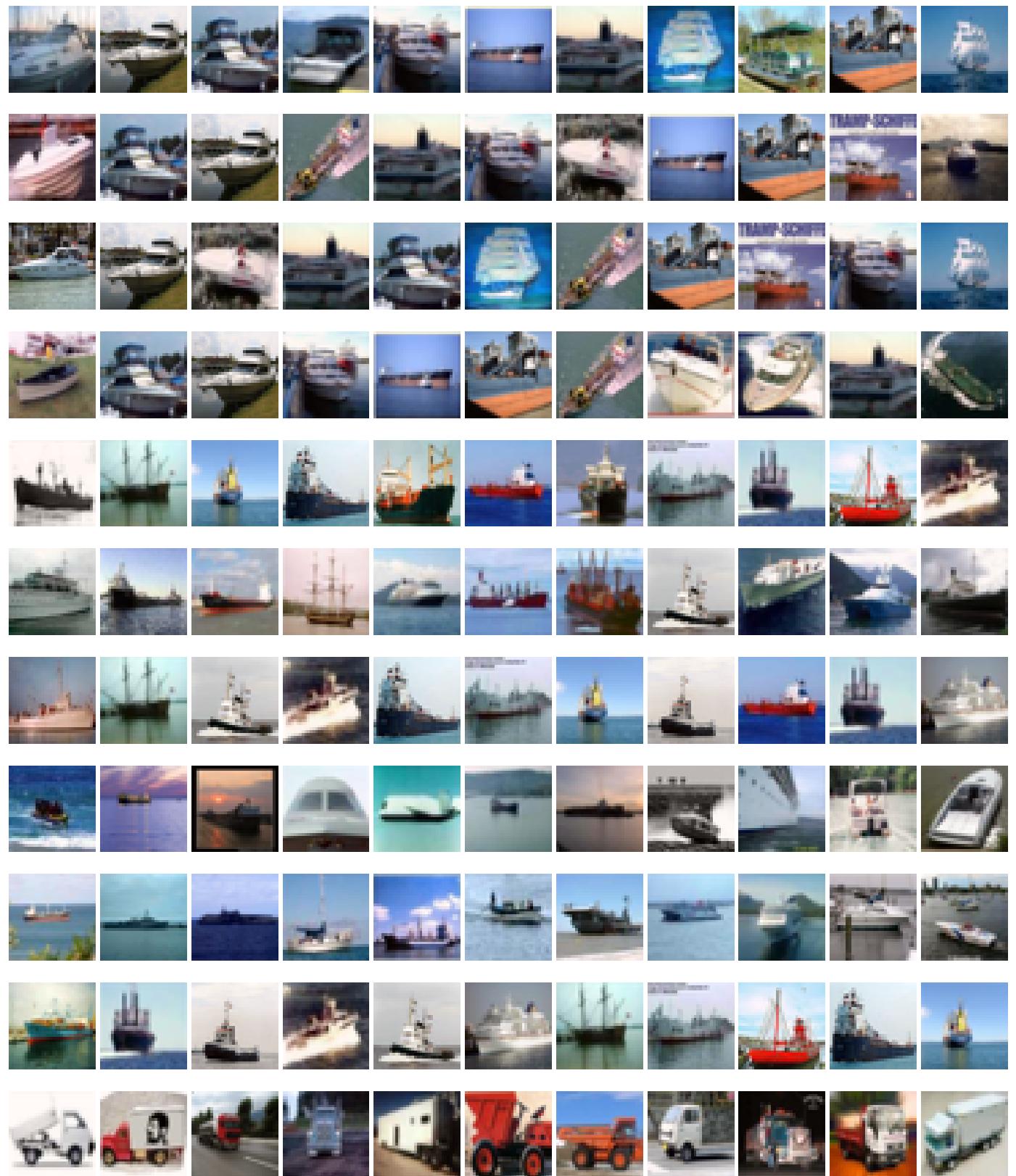


Fig. S46: Explanations for Cifar10 (set 11).

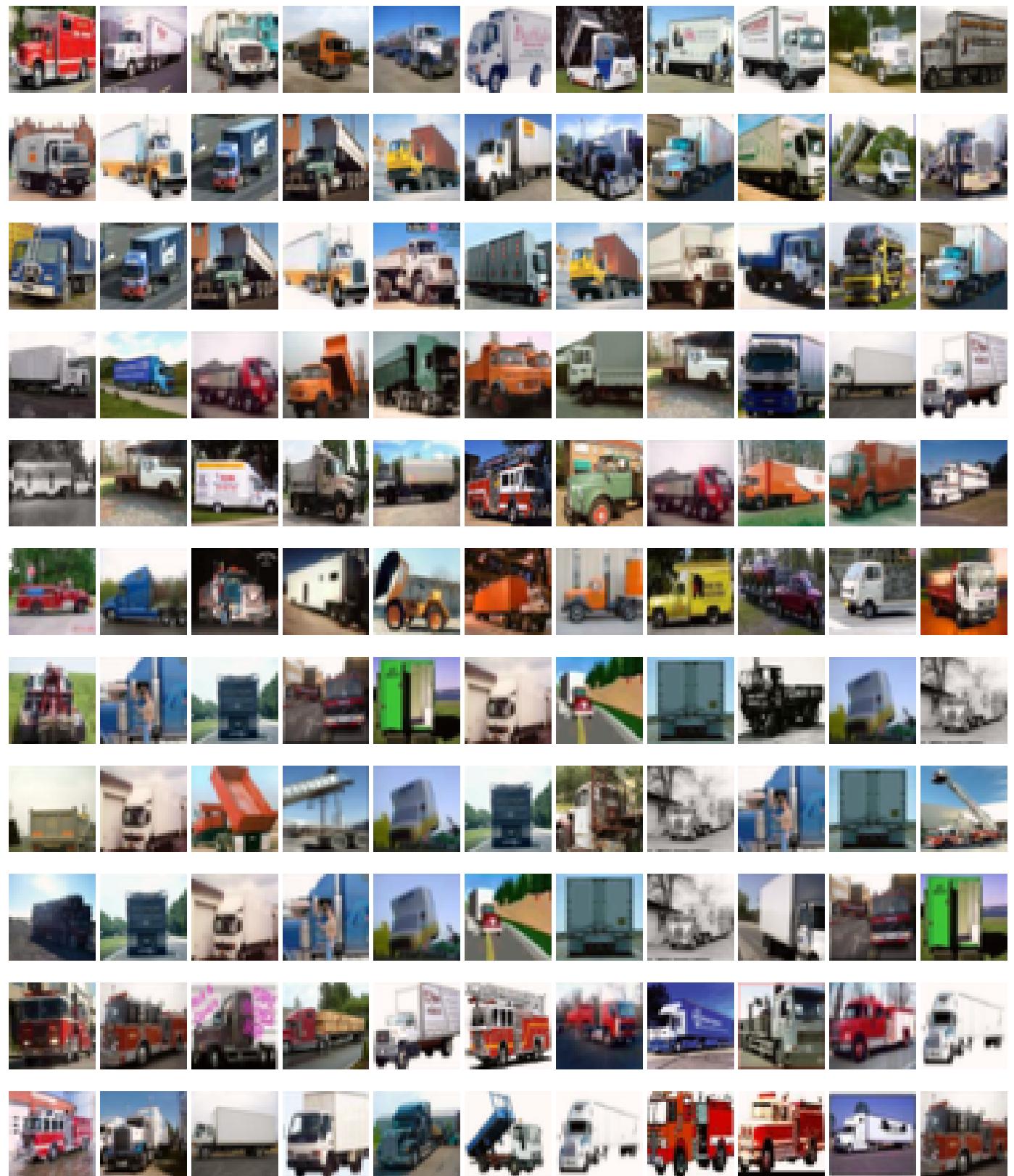


Fig. S47: Explanations for Cifar10 (set 12).

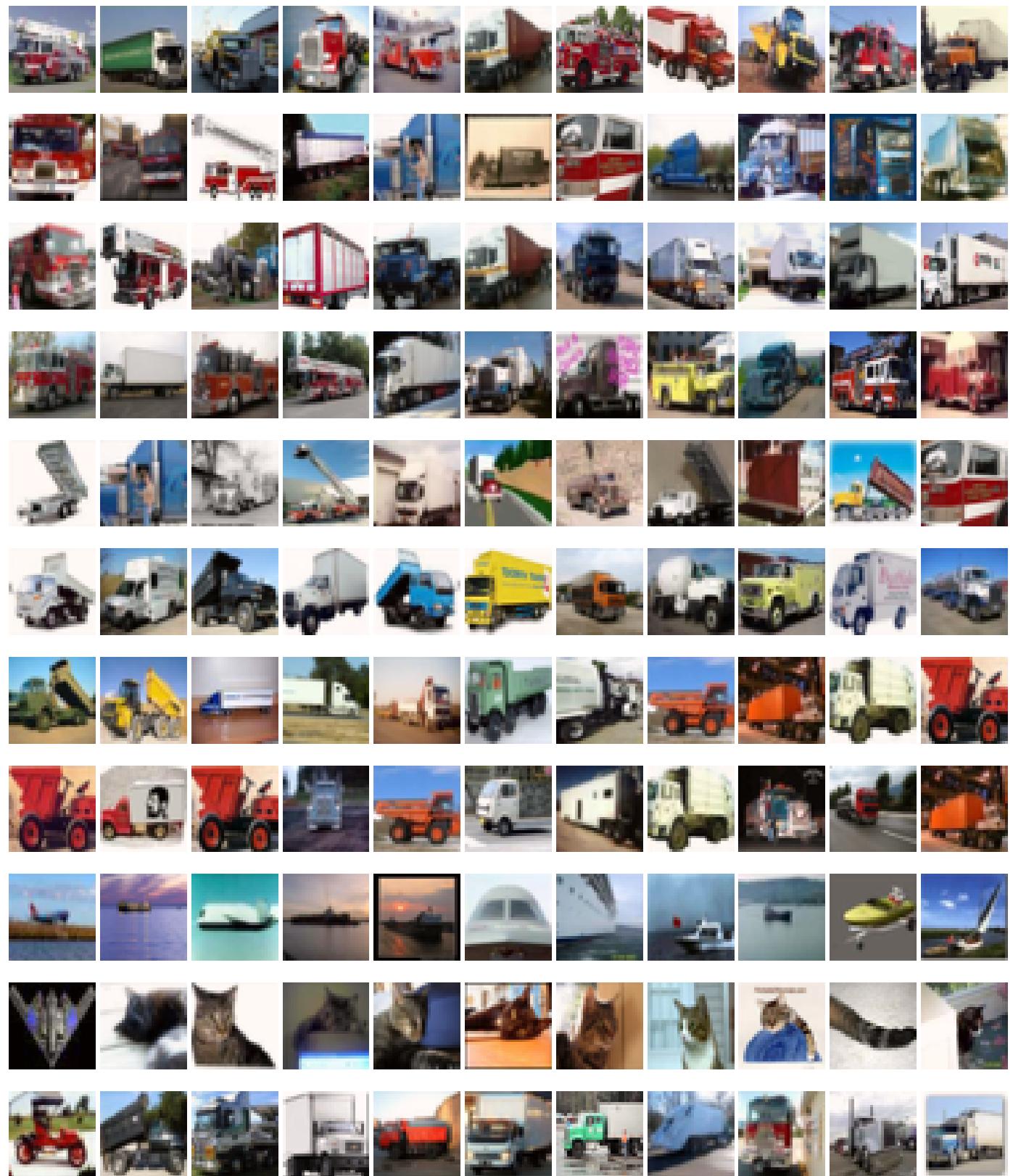


Fig. S48: Explanations for Cifar10 (set 13).

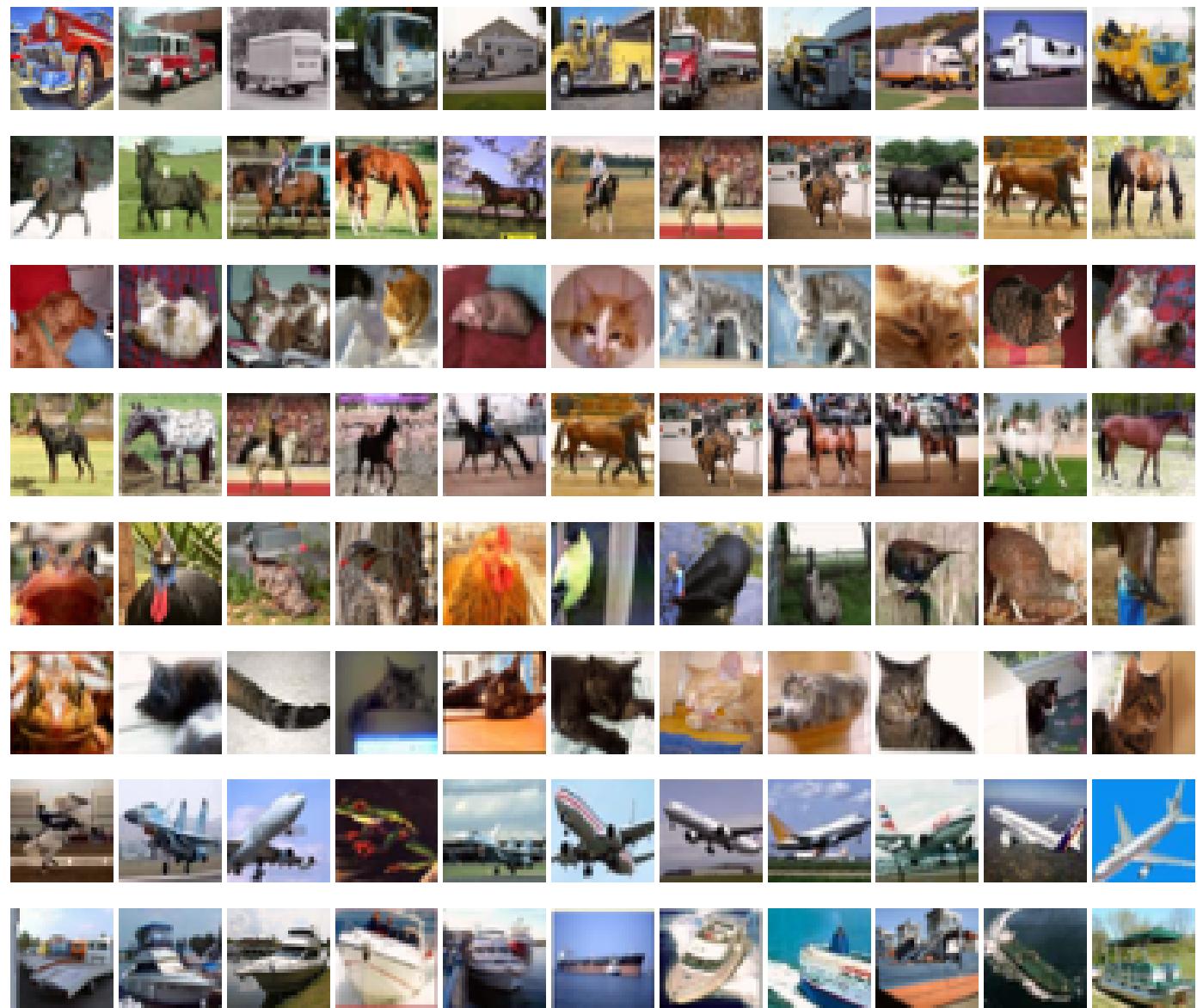


Fig. S49: Explanations for Cifar10 (set 14).

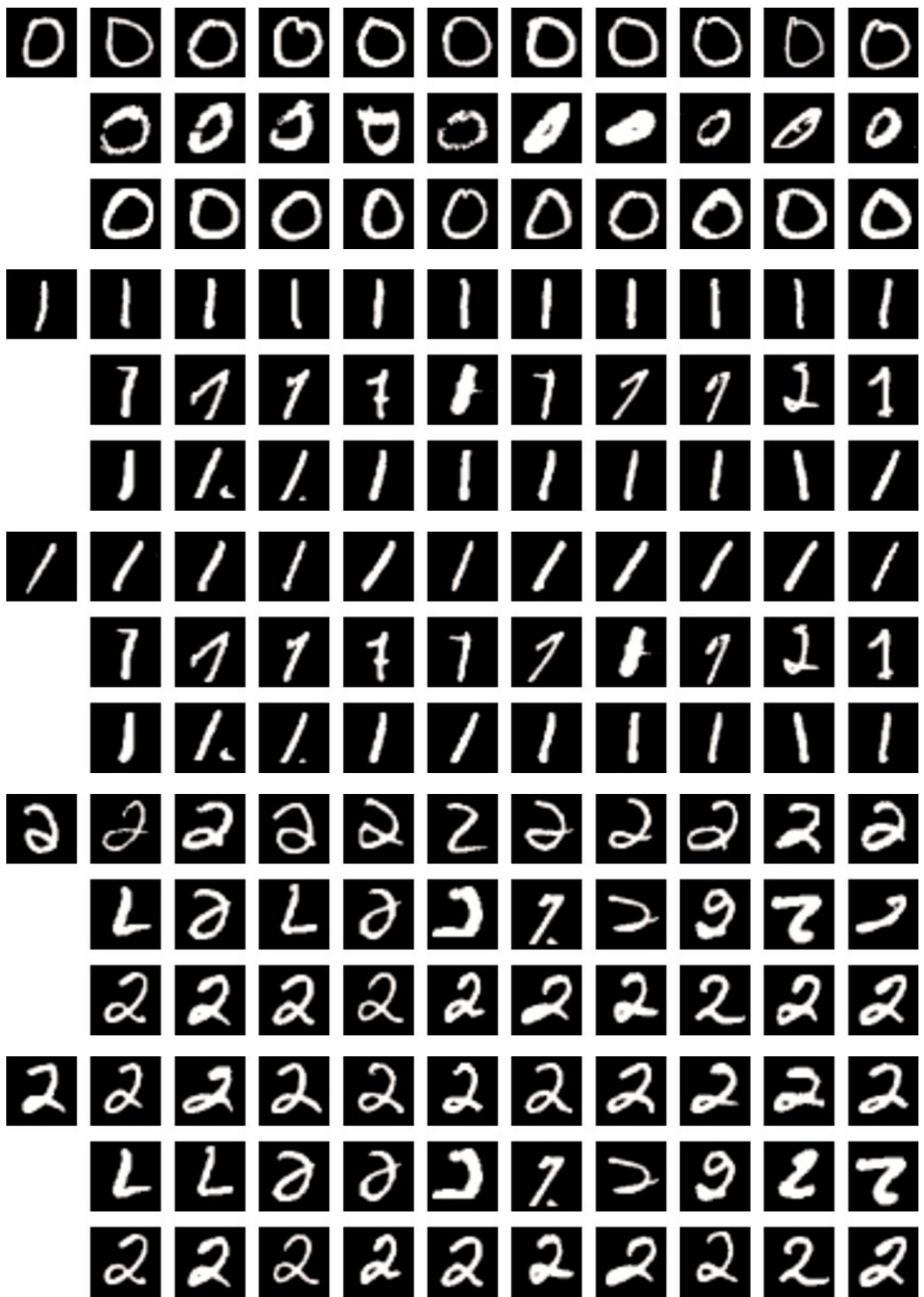


Fig. S50: Comparing the proposed GPEX with representer point selection [12] (set 1).

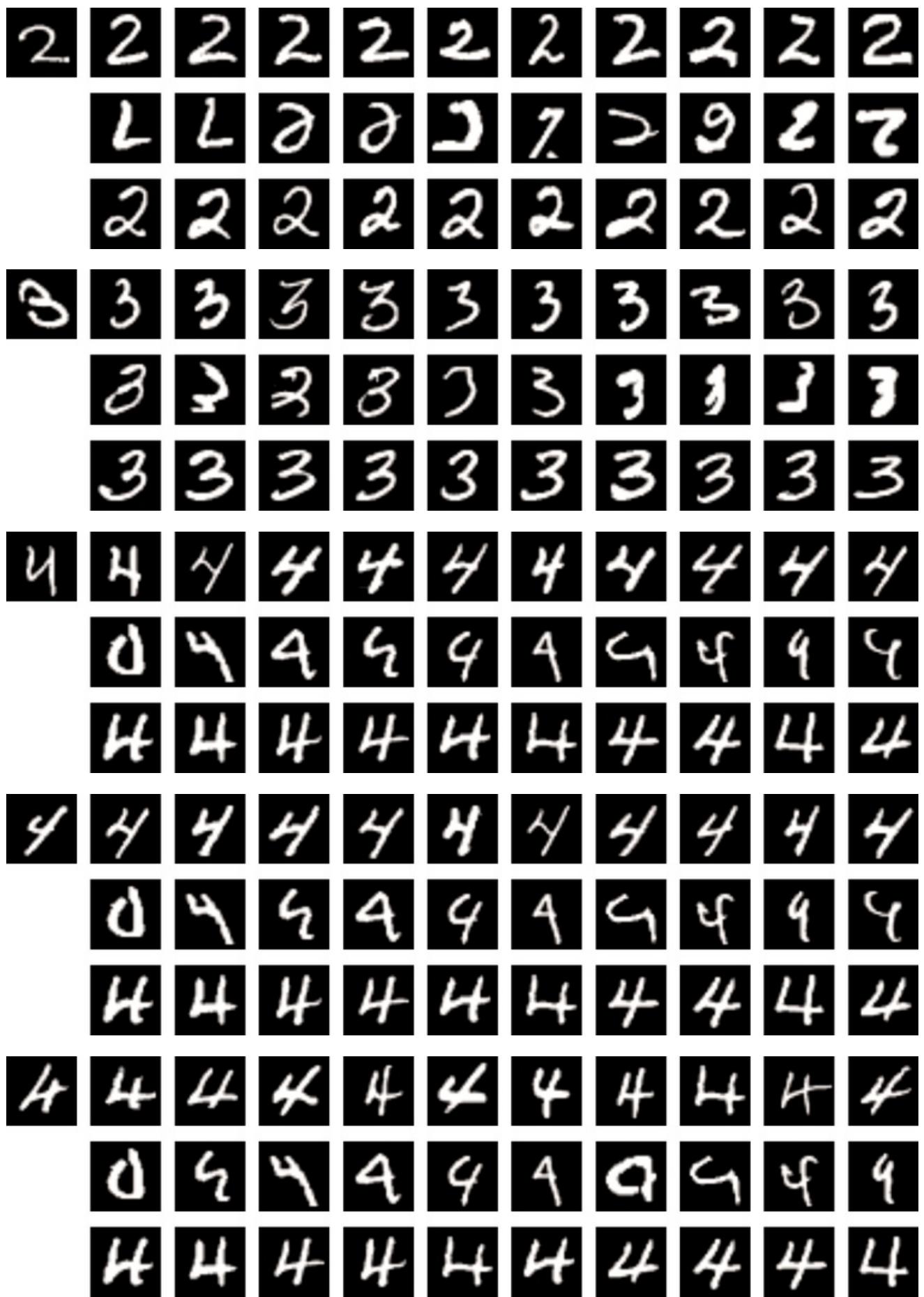


Fig. S51: Comparing the proposed GPEX with representer point selection [12] (set 2).

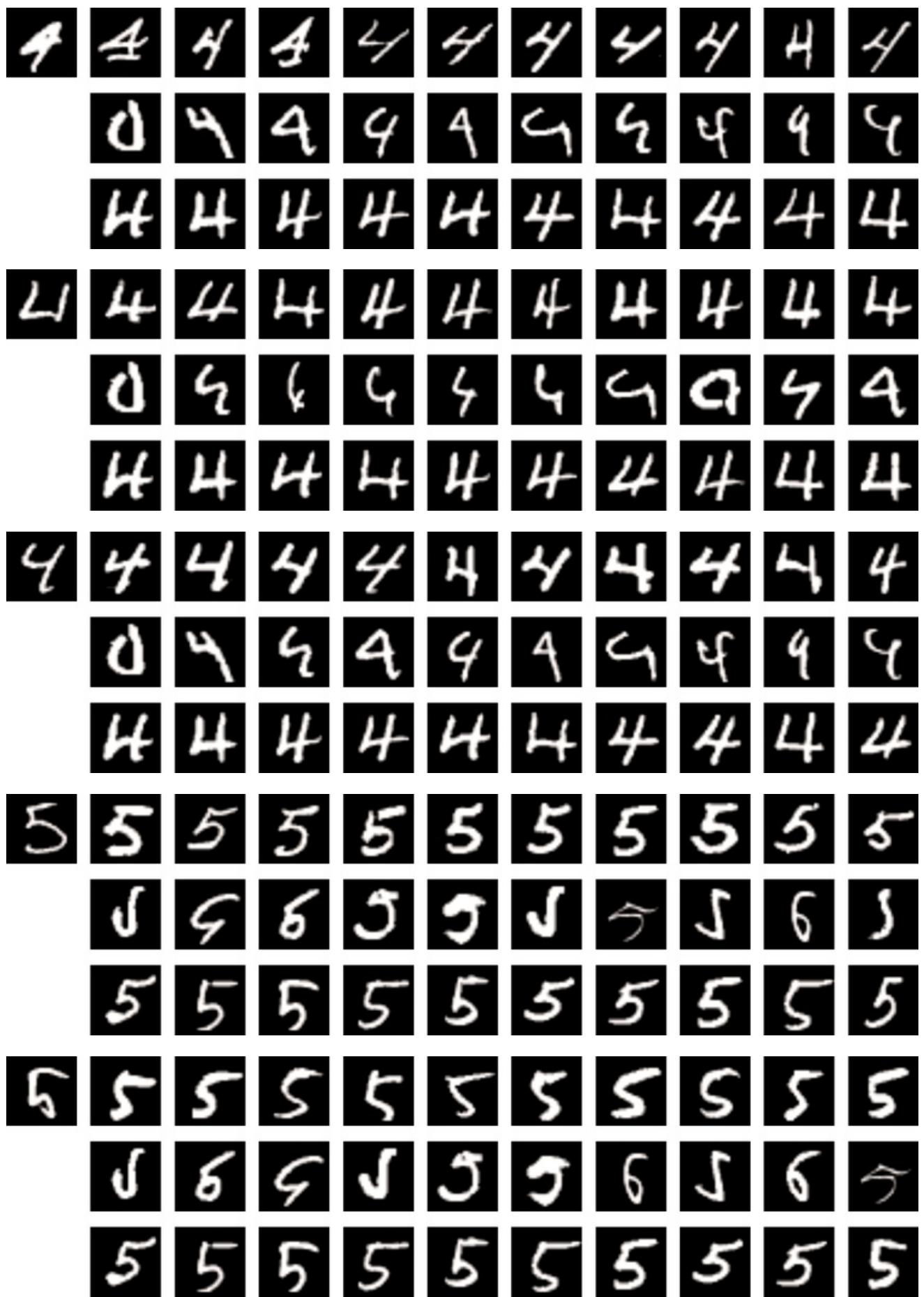


Fig. S52: Comparing the proposed GPEX with representer point selection [12] (set 3).

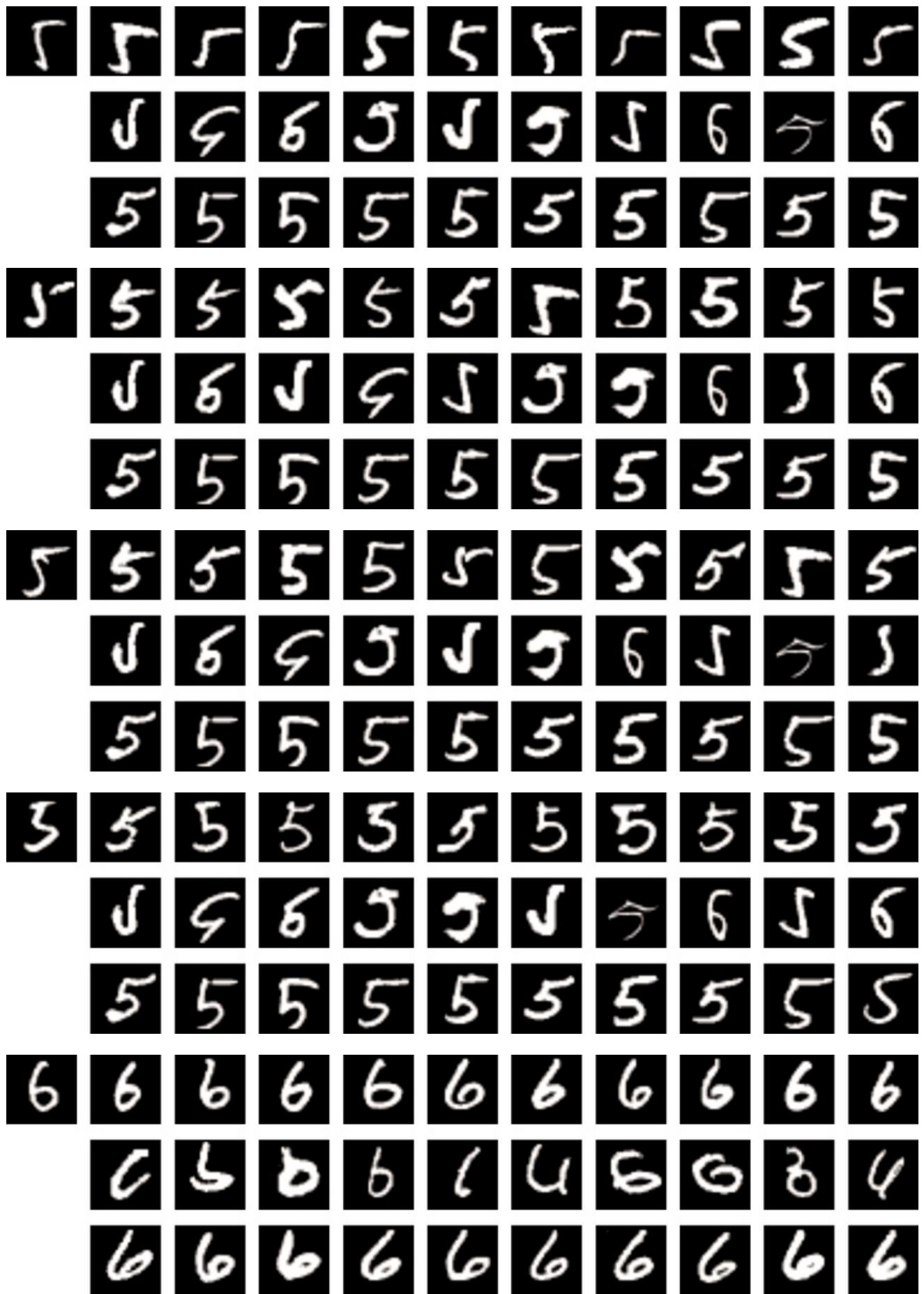


Fig. S53: Comparing the proposed GPEX with representer point selection [12] (set 4).

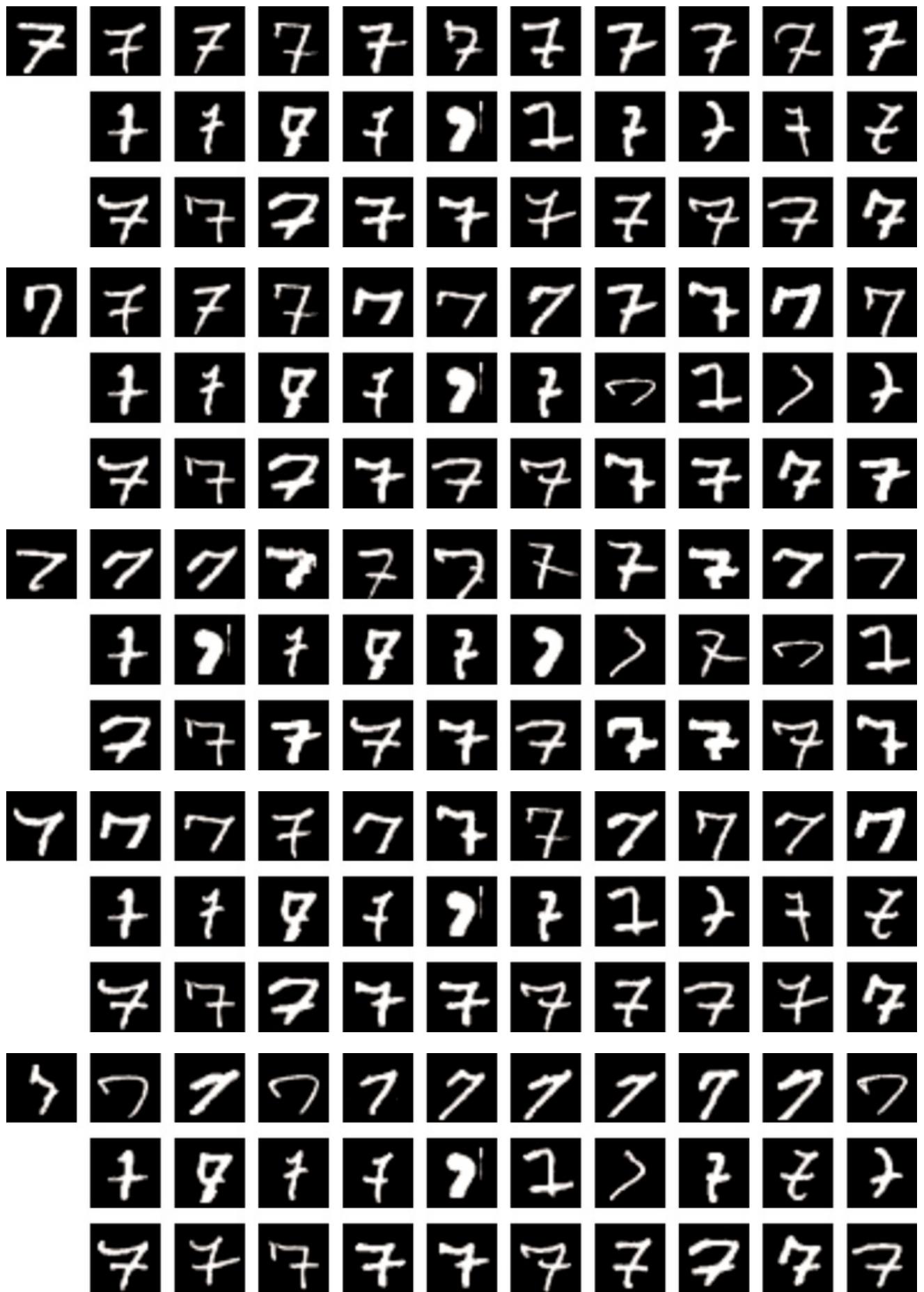


Fig. S54: Comparing the proposed GPEX with representer point selection [12] (set 5).

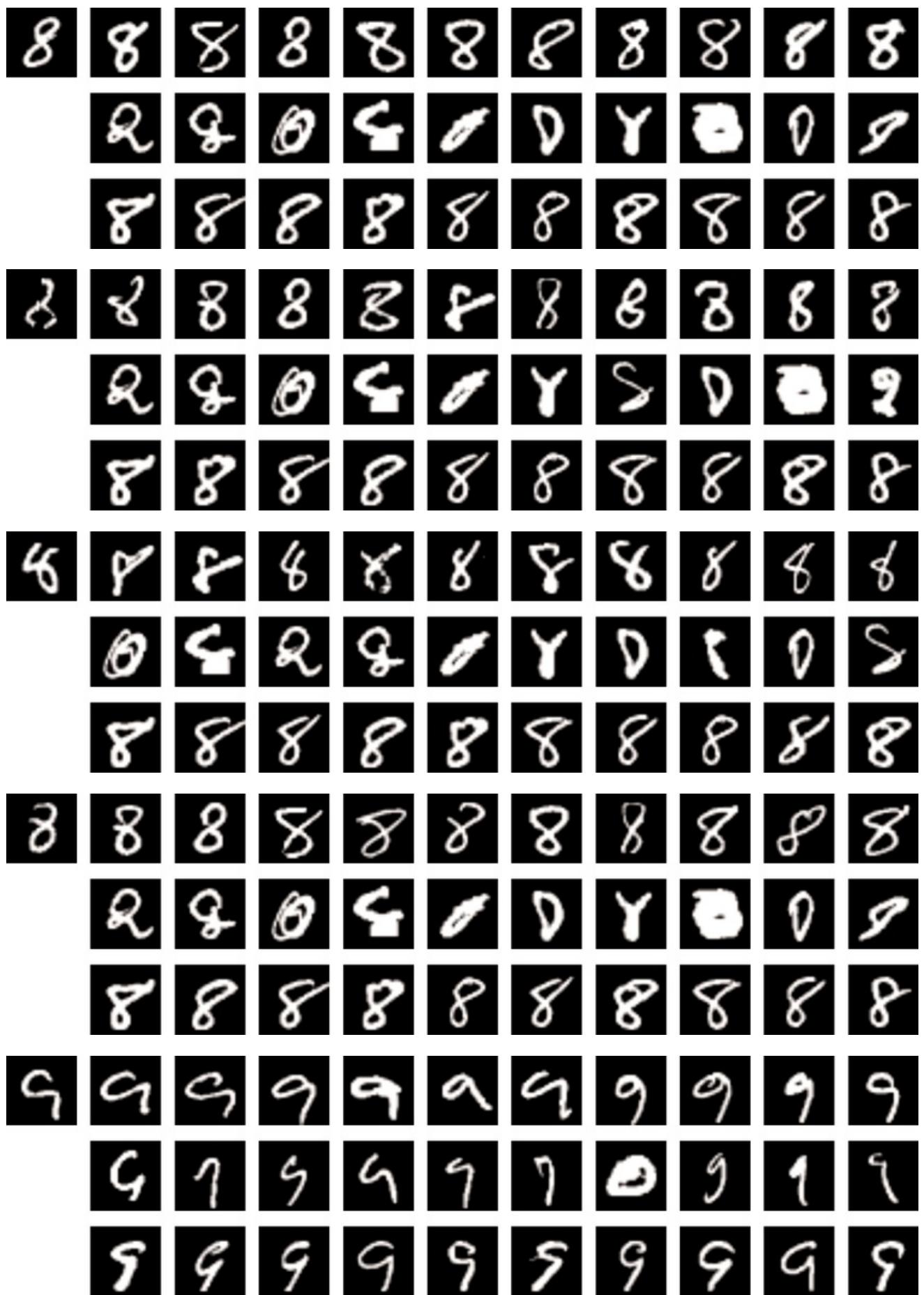


Fig. S55: Comparing the proposed GPEX with representer point selection [12] (set 6).

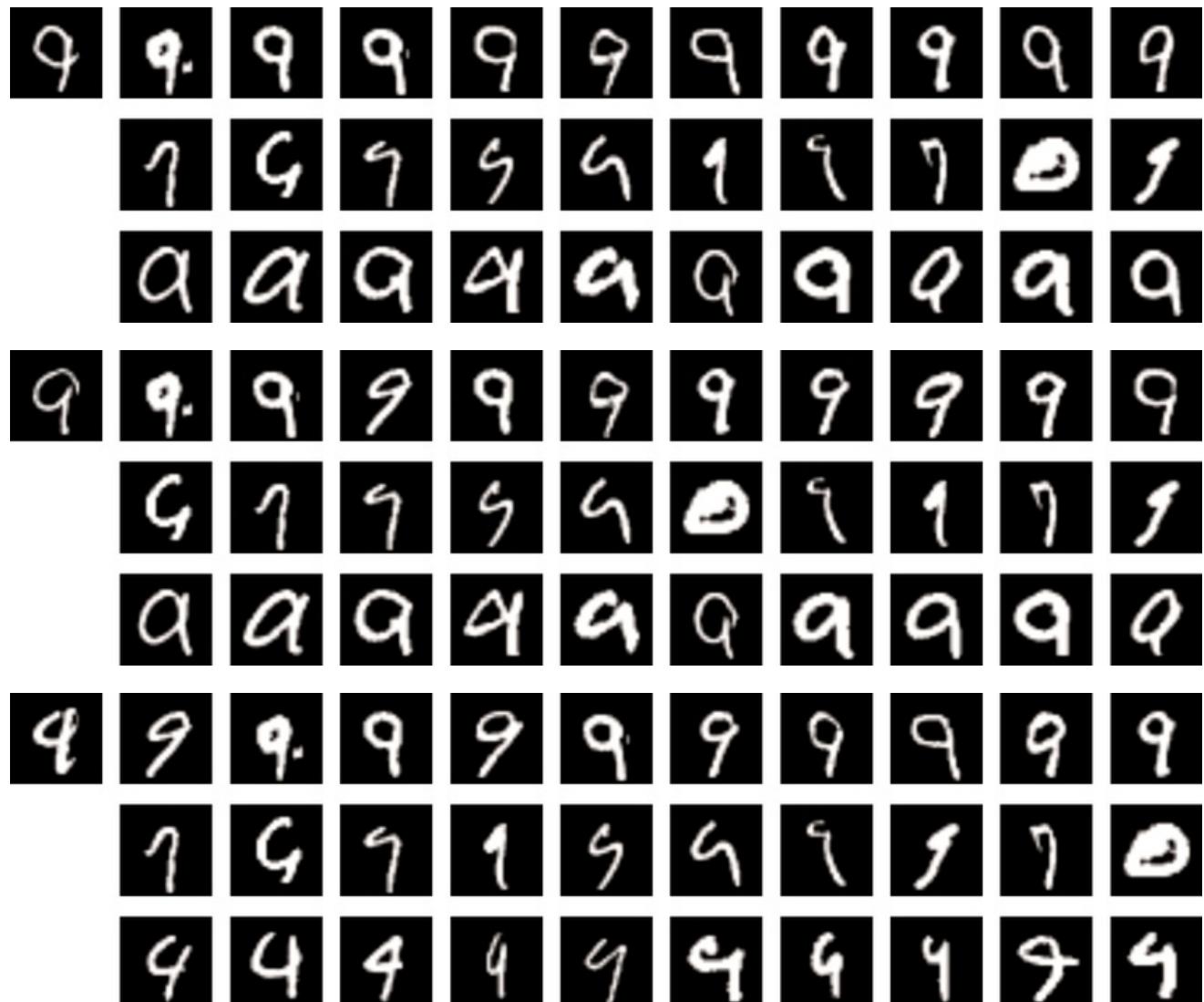


Fig. S56: Comparing the proposed GPEX with representer point selection [12] (set 7).

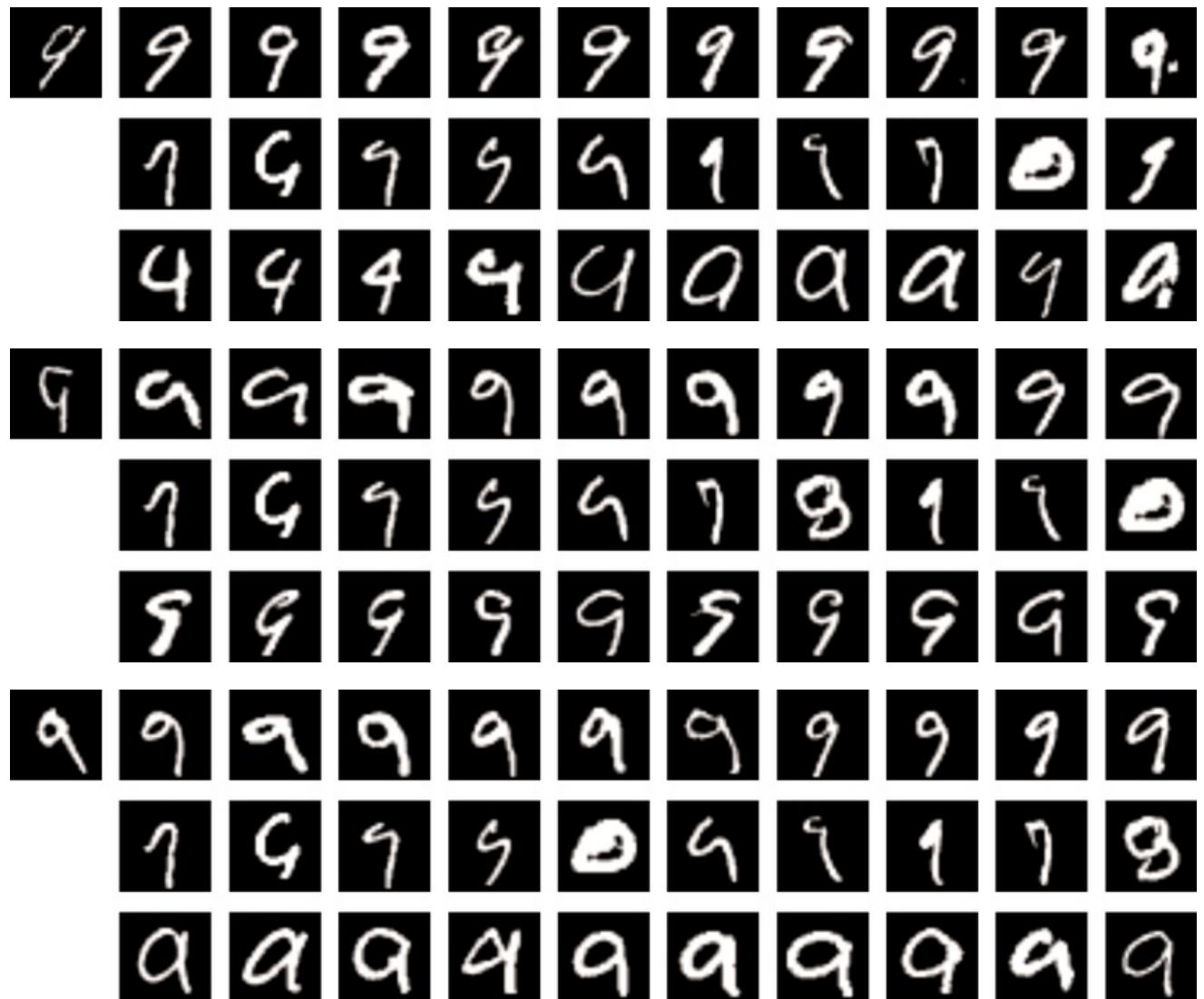


Fig. S57: Comparing the proposed GPEX with representer point selection [12] (set 8).

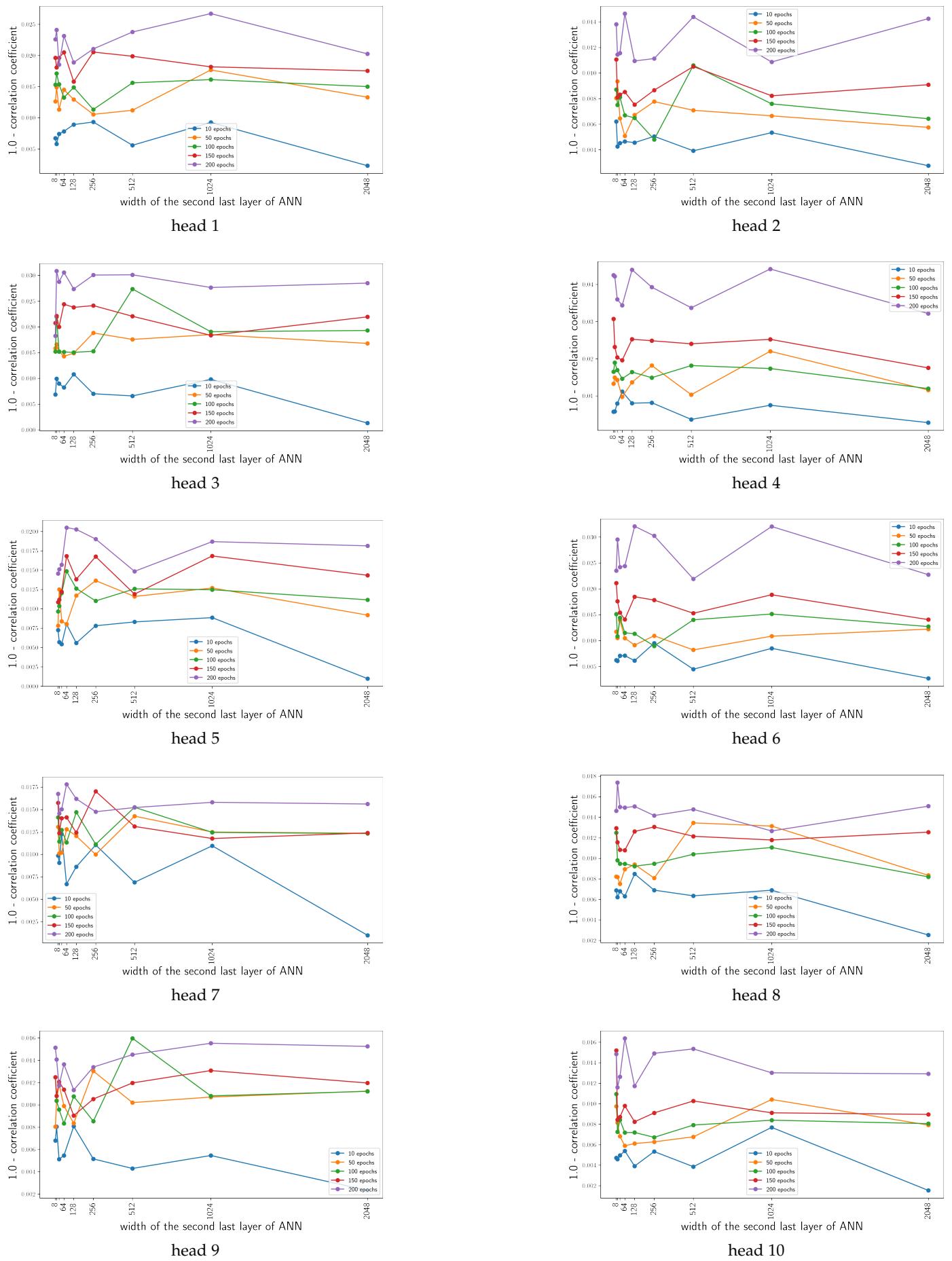


Fig. S58: Parameter analysis of Sec.6 of the main article.

	Cifar10 [33]	MNIST [32]	Kather [34]	DogsWolves [35]
ANN accuracy	95.43	99.56	96.80	80.50
GPs accuracy	92.26	99.41	93.60	78.75

TABLE S1: Accuracies of ANN classifiers versus the accuracies of the explainer GPs on four datasets.

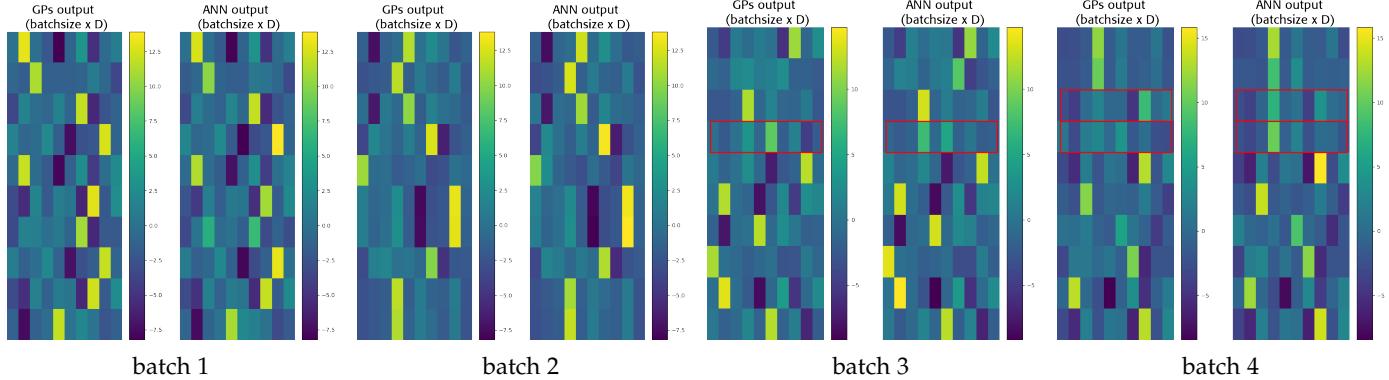


Fig. S59: Comparing GP and ANN outputs for four batches of Cifar10 dataset [33]. The red rectangles highlight the instances for which the predictions of GP and ANN (i.e. the class with maximum score) are different.

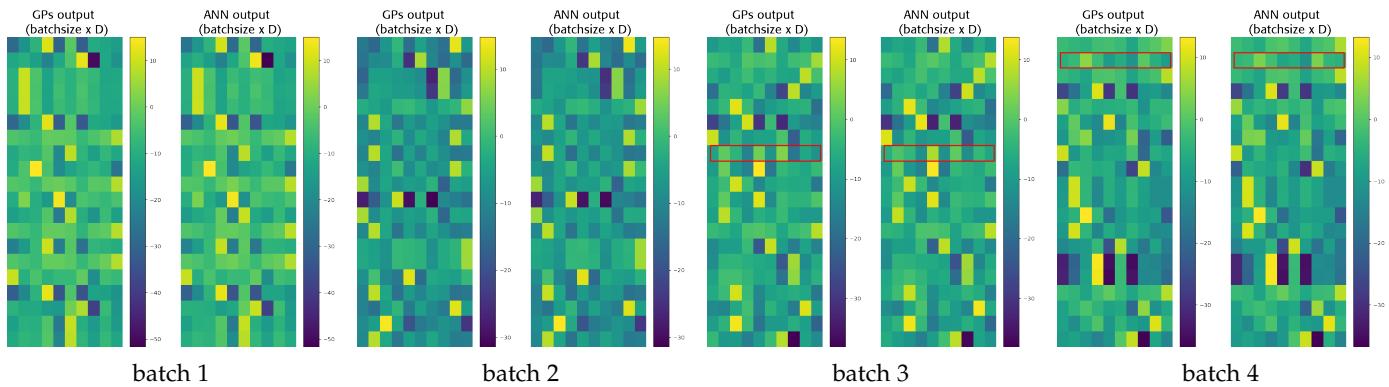


Fig. S60: Comparing GP and ANN outputs for four batches of MNIST dataset [32]. The red rectangles highlight the instances for which the predictions of GP and ANN (i.e. the class with maximum score) are different.

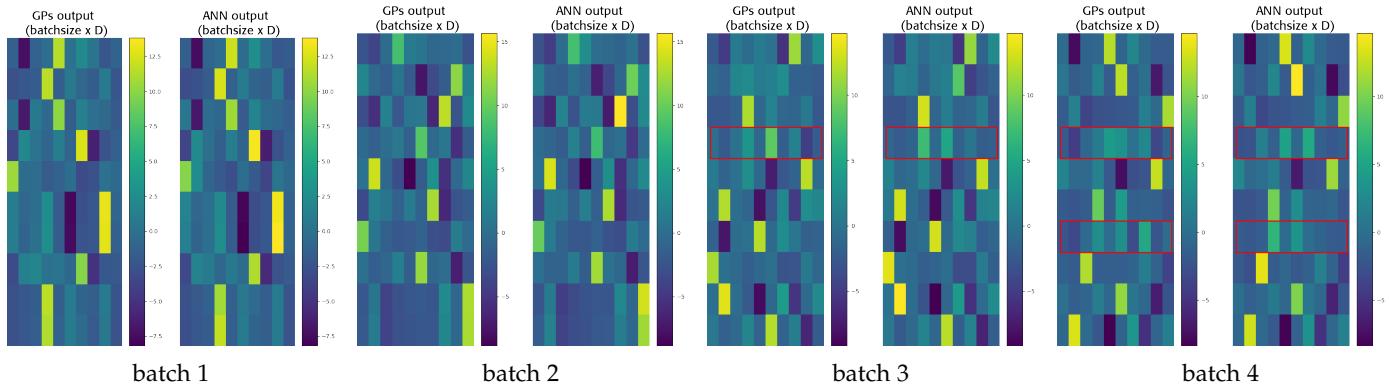


Fig. S61: Comparing GP and ANN outputs for four batches of Kather dataset [34]. The red rectangles highlight the instances for which the predictions of GP and ANN (i.e. the class with maximum score) are different.

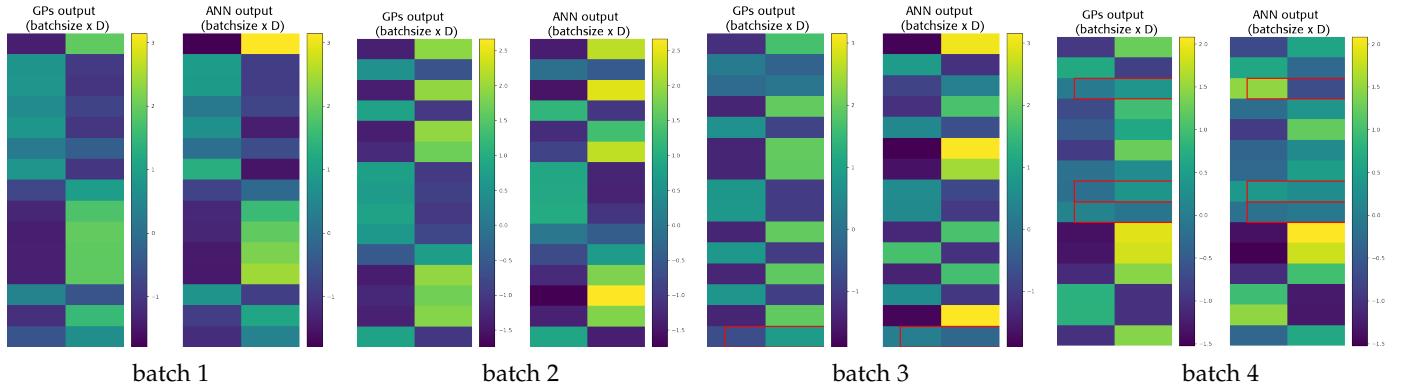


Fig. S62: Comparing GP and ANN outputs for four batches of DogsWolves dataset [35]. The red rectangles highlight the instances for which the predictions of GP and ANN (i.e. the class with maximum score) are different.