Supplementary Materials for Interpretations Steered Network Pruning via Amortized Inferred Saliency Maps

Alireza Ganjdanesh*, Shangqian Gao*, and Heng Huang

Department of Electrical and Computer Engineering, University of Pittsburgh, Pittsburgh, PA 15261, USA
{alireza.ganjdanesh, shg84, heng.huang}@pitt.edu
(* indicates equal contribution)

1 REAL-X Formulation Development for Interpretation of Classifiers

This section provides details about the connections of REAL-X formulation presented in [6] for dimensionality reduction of samples and its version for interpreting a classifier. At first, we show the REAL-X formulation for dimensionality reduction for completeness and then derive its formulation for the interpretation of classifiers. Please refer to Section 3.2 of the main text for a description of the notations.

1.1 REAL-X for Dimensionality Reduction

Amortized Explanation Models (AEMs) [6, 1, 16, 2] aim to learn to predict a 'sample-specific' mask for each sample such that preserved features contain all information related to an outcome. An outcome in REAL-X [6] is the target (label) of the sample, *i.e.*,

$$\mathcal{P}(\mathbf{y}|\mathbf{x} = x) = \mathcal{P}(\mathbf{y}|\mathbf{x} = m(x)) \tag{1}$$

Here $\mathcal{P}$ is the joint population distribution on inputs and targets that is unknown in practice.

We emphasize that the goal is not to explain a classifier's predictions in this formulation. Instead, it is dimensionality reduction by only keeping input features (pixels in images) that preserve the information related to labels of samples.

REAL-X trains a selector model $f_\beta(\cdot)$ implemented with a Deep Neural Network (DNN) function parameterized by β that predicts a distribution over possible explanatory masks for a given sample, and $f_\beta(\cdot) \in \mathbb{R}^D$. The distribution is factorized as a product of marginal Bernoulli distributions over mask's pixels, *i.e.*,

$$q_{sel}(m|x;\beta) = \prod_{i=1}^{D} q_i(m_i|x;\beta) \tag{2}$$
$$q_i(m_i|x;\beta) \sim Bernoulli((f_\beta(x))_i)$$

During training, the selector model is encouraged to predicts masks that follow Eq. (1). To do so, a predictor model is used that estimates population conditional distribution of targets given masked inputs $\mathcal{P}(\mathbf{y}|\mathbf{x} = m(x))$ and trained by the following objective:

$$\max_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{P}} \mathbb{E}_{m'\sim\mathcal{B}(0.5)}[\log(q_{pred}(\mathbf{y} = y|\mathbf{x} = m'(x);\theta))] \tag{3}$$

This is equivalent to

$$\min_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{P}} \mathbb{E}_{m'\sim\mathcal{B}(0.5)}[KL(\mathcal{P}(\mathbf{y}|\mathbf{x} = x)\|q_{pred}(\mathbf{y}|\mathbf{x} = m'(x);\theta))] \tag{4}$$

where we represent the conditional population distribution $\mathcal{P}(\mathbf{y}|\mathbf{x} = x)$ with one-hot vectors. Finally, given a pretrained predictor model, REAL-X trains a selector model to maximize:

$$\max_{\beta} \mathbb{E}_{(x,y)\sim\mathcal{P}} \mathbb{E}_{m'\sim\ q_{sel}(m|x;\beta)}[\log(q_{pred}(\mathbf{y} = y|\mathbf{x} = m'(x))) - ||m'||_0] \tag{5}$$

Similar to Eqs. (3, 4), the log-likelihood term in Eq. (5) can be replaced with

$$KL(\mathcal{P}(\mathbf{y}|\mathbf{x} = x)||q_{pred}(\mathbf{y}|\mathbf{x} = m'(x))) \tag{6}$$

1.2 REAL-X for Interpretation of Classifiers

Now, we develop the formulation of REAL-X for interpreting a classifier's predictions. The new goal is to learn to find a mask for each sample such that it preserves information related to the classifier's predictions in the remaining input features, *i.e.*,

$$\mathcal{Q}_{class}(\mathbf{y}|\mathbf{x} = x^{(i)}) = \mathcal{Q}_{class}(\mathbf{y}|\mathbf{x} = m(x^{(i)})) \tag{7}$$

Therefore, similar to Eq. (4), the objective for training a predictor model that estimates the conditional distribution of the classifier given masked inputs will be:

$$\min_{\theta} {}_{x\sim\mathcal{P}(\mathbf{x})m'\sim\mathcal{B}(0.5)} L_\theta(x, m'(x)) \tag{8}$$

$$L_\theta(x, m'(x)) = KL(\mathcal{Q}_{class}(\mathbf{y}|\mathbf{x} = x), q_{pred}(\mathbf{y}|\mathbf{x} = m'(x);\theta)) \tag{9}$$

where we have replaced the population conditional distribution $\mathcal{P}(\mathbf{y}|\mathbf{x} = x)$ in Eq. (4) with the one for the classifier $\mathcal{Q}_{class}(\mathbf{y}|\mathbf{x} = x)$, which is usually implemented as a softmax distribution.

Lastly, we can train a selector model guided by a pretrained predictor to obey Eq. (7) by minimizing:

$$\min_{\beta} \mathbb{E}_{x\sim\mathcal{P}(\mathbf{x})}\mathbb{E}_{m'\sim\ q_{sel}(m|x;\beta)}[L(x, m'(x)) + \lambda||m'||_0] \tag{10}$$

Again, this is equivalent to Eq. (5) by replacing the population conditional distribution $\mathcal{P}(\mathbf{y}|\mathbf{x} = x)$ in Eq. (6) with the one for the classifier, *i.e.*, $\mathcal{Q}_{class}(\mathbf{y}|\mathbf{x} = x)$. We use Eqs. (8, 10) to train REAL-X to explain decisions of a ResNet-56 architecture on CIFAR-10 in section 3.3 of the paper.

2 Implementation Details of Our AEM

As a recall, the formulation for training our AEM model is the selector minimizing:

$$\min_{\beta} L_{sel}(\beta) = \mathbb{E}_{x\sim\mathcal{P}(\mathbf{x})}\mathbb{E}_{m'\sim q_{sel}(m|x;\beta,\mathbf{v})}[L(x, m'(x)) + \lambda_1\mathcal{R}(m') + \lambda_2\mathcal{S}(m')] \tag{11}$$

$$\mathcal{R}(m') = ||m'||_0$$

$$\mathcal{S}(m') = \sum_{i=1}^{M}\sum_{j=1}^{N}[(m'_{i,j} - m'_{i+1,j})^2 + (m'_{i,j} - m'_{i,j+1})^2]$$

such that we factorize $q_{sel}(m|x;\beta,\mathbf{v})$ as a product of marginal Bernoulli distributions over the pixels. The parameters of Bernoulli distributions have RBF form over pixel locations, *i.e.*, we calculate the parameter for a pixel at location (z, t) as:

$$f_{BP}(z, t; c_z, c_t, \sigma) = \exp\left(\frac{-1}{2\sigma^2}[(z - c_z)^2 + (t - c_t)^2]\right) \tag{12}$$

In this section, we provide more practical details about the implementation of our AEM, namely the U-Net architecture's layers and training procedure of the selector model for ImageNet [13] and CIFAR-10 [8].

2.1 U-Net Architecture

We use a U-Net [12] architecture with a feature filter module proposed in [2] to implement the selector module of our AEM model. It provides the flexibility to use the feature extractor backbone of the pretrained classifier (*e.g.*, scales 1-5 of ResNet [4] before its global average pooling layer) as the encoder of U-Net and only train the decoder part for computational efficiency. The feature filter is an embedding layer (denoted by C) learned along with the decoder that performs the initial localization of the target class by attenuating spatial locations that do not contain the target [2]. It applies such operation on the output of the encoder before inputting it to the decoder. Formally, the output filtered feature Z at spatial location (i, j) given input features X and target class embedding C_y is calculated as:

$$Z_{i,j} = X_{i,j}\sigma(X_{i,j}^T C_y) \tag{13}$$

3 Experimental Setup

We use CIFAR-10 [8] and ImageNet [3] to validate the effectiveness of our proposed model. For all experiments, we train the original classifier from scratch (CIFAR-10) or adopt PyTorch [11] pretrained models (ImageNet). To train the AEM model, we follow two main steps: 1) We train the predictor with Eq. 6 in the paper using the same architecture as the classifier starting from scratch (CIFAR-10) or PyTorch pretrained checkpoint (ImageNet). 2) We use the convolutional feature extraction backbone of the classifier as the encoder of the selector's U-Net architecture and keep it frozen during training the decoder for computational efficiency. (Fig. 2 in the paper) The decoder is trained with objective 9 in the paper and steered by the trained predictor from the previous step.

CIFAR-10: For CIFAR-10 experiments, we evaluate our model on ResNet-56 [4] and MobileNetV2 [14]. We train the predictor model for 300 epochs for both models with a mini-batch size of 128 using ADAM optimizer [7] with a learning rate of 0.0001, exponential decay rates $(\beta_1, \beta_2) = (0.9, 0.999)$, and weight decay of 0.0001. We train the selector model for 10 epochs with mini-batch size 16 and the same optimization configuration. We found that the parameter setting $\lambda_1 = 0.2$ and $\lambda_2 = 0.001$ perform well for both models. We randomly partition the official training set with a 0.9/0.1 ratio to form our training/validation sets and use the official test set as our test partition. During pruning, we select 5% of the official training set as the subset for pruning. We choose $\gamma_1 = 0.5$ and $\gamma_2 = 2.0$ for pruning. We optimize Eq. 11 in the paper for pruning for 200 epochs by using the subset with ADAM optimizer. After pruning, we finetune the model for 200 epochs with SGD optimizer of momentum 0.9, weight decay 0.0001, and start learning rate 0.1. The mini-batch size is 128 for both pruning and finetuning. As mentioned in section 3.5 and Fig. 2 of the paper, we use the convolutional feature extraction backbone of the

classifier as the encoder of the selector's U-Net architecture and keep it frozen during training the decoder for computational efficiency. ResNet-56 [4] and MobileNetV2 [14] architectures generate three scales of representations before their average pooling layer considering (number of channels, spatial dimensions) from a raw input with $3 \times 32 \times 32$ spatial dimensions. ResNet-56 scales' representations have $\{(16, 32 \times 32), (32, 16 \times 16), (64, 8 \times 8)\}$ dimensions, and these values for MobileNetV2 are $\{(32, 32 \times 32), (96, 16 \times 16), (1280, 8 \times 8)\}$. Therefore, their feature filter embedding layers have 10×64 and 10×1280 dimensions respectively (CIFAR-10 has 10 classes). We use two upsampling blocks for CIFAR-10 experiments. These blocks are characterized by 3 parameters: number of their input features' channels, number of pass-through features' channels (input features from the U-Net's encoder), and number of their output channels [2]. These values are $\{(64, 32, 32), (32, 16, 16)\}$ and $\{(1280, 96, 96), (96, 32, 32)\}$ for upsampling layers of ResNet-56 and MobileNetV2 respectively. An upsampling layer, at first, upsamples a low-resolution feature map by a factor of 2 using $2D$-Convolution and Pixel Shuffle blocks [15]. Then, it concatenates upsampled features with pass-through features and applies three Bottleneck blocks [4] on them. Please refer to our code implementations for more details.

Finally, we use a $2D$-Convolution layer followed by three non-linearities that map the last upsampling layer's output to 3 values corresponding to predicted (c_z, c_t, σ) parameters for the output RBF kernel. We set the kernel size of the convolution filter to the upsampling layer's output spatial dimension (32). In addition, we set the number of output channels to 3. We use two different non-linear activation functions, namely one for calculating c_z, c_t and the other for σ that we elaborate on them in Section 3.

ImageNet: Most of the details are similar to the CIFAR-10 experiments described above. We use ResNet-34, 50, 101 [4] and MobileNetV2 [14] to assess our model's performance on ImageNet [13]. Training on the full dataset is computationally intensive. We randomly select 0.1/0.02 of the official training set as our training/validation partitions and use the official validation set as our test set for our AEM model's training and evaluation. We train the predictor for 100 epochs with batch size 128 and the selector for 5 epochs with batch size 64. The optimization parameters are the same as CIFAR-10 experiments. The configuration $\lambda_1 = 1.0$ and $\lambda_2 = 0.0001$ showed convincing performance for all architectures. In ImageNet, we use the previously mentioned subset for pruning. During pruning, we optimize Eq. 11 in the paper for 100 epochs with the ADAM optimizer. γ_1 and γ_2 are the same as the CIFAR-10 setting. After pruning, we finetune all architectures for 100 epochs by using SGD with a momentum of 0.9 with a start learning rate of 0.1. For MobileNet-V2, we use a start learning rate of 0.05 and a cosine annealing learning rate scheduler, as mentioned in the original paper [14]. We set the weight decay to 0.0001 for ResNet models and 0.00004 for MobileNetV2. We implement our method using PyTorch [11]. Given an input image with 3 channels and 224×224 spatial dimensions, ResNet-34, 50, and 101 calculate 5 scales with (number of channels, spatial dimensions) as follows:

- $\{(64,\ 56\times56),\ (64,\ 56\times56),\ (128,\ 28\times28),\ (256,\ 14\times14),\ (512,\ 7\times7)\}$ for ResNet-34,
- $\{(64,\ 56\times56),\ (256,\ 56\times56),\ (512,\ 28\times28),\ (1024,\ 14\times14),\ (2048,\ 7\times7)\}$ for ResNet-50 and ResNet 101,
- $\{(32,\ 112\times112),\ (24,\ 56\times56),\ (32,\ 28\times28),\ (96,\ 14\times14),\ (1280,\ 7\times7)\}$ for MobileNetV2.

Thus, the embedding layer of their feature filter layer is 1000×512 for ResNet-34, 1000×2048 for ResNet-50 as well as ResNet-101, and 1000×1280 for MobileNetV2.

We use 3 upsampling blocks for these architectures. The values of (# input features' channels, # pass-through features' channels, # output channels) for upsampling layers are as follows:

- $\{(512,\ 256,\ 256),\ (256,\ 128,\ 128),\ (128,\ 64,\ 64)\}$ for ResNet-34.
- $\{(2048, 1024, 1024), (1024, 512, 512), (512, 256, 256)\}$ for ResNet-50 and ResNet-101.
- $\{(1280,\ 96,\ 96),\ (96,\ 32,\ 32),\ (32,\ 24,\ 24)\}$ for MobileNetV2.

Finally, we use a $2D$-Convolution with kernel size 56 and 3 output channels to map the last upsampling layer's outputs to 3 numbers for (c_z, c_t, σ).

Proposed Output Nonlinearities for the Selector Model Center's Coordinates Nonlinearity: As mentioned in Section 3.5 in the paper, if we consider a two-axis coordinate system for an image's spatial dimensions and the system's origin being on the center of it, the values c_z, c_t can take any real values theoretically. However, we know that the salient part of the image is within its spatial dimensions, *e.g.*, $c_z, c_t \in [-16, 16]$ for CIFAR-10 images with 32×32 size. Therefore, we use `Tanh` non-linearity to ensure that the output values are in the range of image dimensions. In addition, to prevent the vanishing gradients phenomenon in `Tanh`, we set its 'active' range being roughly equal to spatial dimensions of an input image, *i.e.*, we calculate c_z, c_t as:

- $c_z = 14 * \texttt{Tanh}(u_z/14) + 16$
- $c_t = 14 * \texttt{Tanh}(u_t/14) + 16$

We show this functional form in Fig. 1. As can be seen, when its input u lies in the interval $[-14, 14]$, the output lies in $[2, 30]$, which corresponds to the 28×28 frame into a 32×32 image. Moreover, when $u \in [-14, 14]$, the `Tanh` function is in its active form, which prevents the known vanishing gradient challenge with `Tanh`.

For ImageNet experiments, we use the following form to ensure that the center of predicted RBF is in the central 220×220 frame of a 224×224 image, and we show it in Fig. 3.

- $c_z = 108 * \texttt{Tanh}(u_z/108) + 112$
- $c_t = 108 * \texttt{Tanh}(u_t/108) + 112$

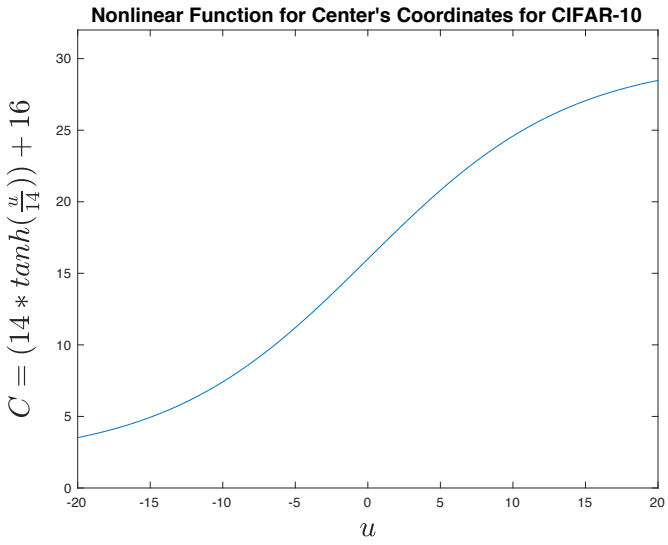

Fig. 1: Our proposed nonlinear function to calculate the center's coordinates of a predicted RBF Kernel of the selector for the CIFAR-10 dataset.

Expansion Parameter's (σ) Nonlinearity: the parameter σ in Eq. (12) determines the degree of RBF expansion on an input image's surface and should be a positive real number. Therefore, we use 'Softplus' non-linearity to calculate it, which is a smooth approximation of `ReLU` [10] and is always positive. It is calculated with the formula $SoftPlus(u) = \log(1 + \exp(u))$ and is shown in the Fig. 2. We use this function to calculate the expansion parameter for both sets of experiments on CIFAR-10 and ImageNet.

Selector Network's Training Initialization: We use the standard network initialization implemented in PyTorch [11] to train a decoder of a selector's U-Net. However, this initialization makes the output values of the output $2D$-convolution layer close to zero at the beginning of training. As a result, the output σ will be close to zero, and it may cause instability in the starting iterations of training. To prevent such instability, we empirically found that setting the $2D$-convolution layer's bias weight corresponding to σ to be about a third of the input image's spatial dimension can make the model's convergence faster and training more stable. Thus, we initialize the bias weight to be 10 for CIFAR-10 experiments and 80 for ImageNet ones.

3.1 Gumbel-Sigmoid Reparameterization Strategy for Training Selector Model

We use Eq. (11) as the objective to train our selector model. Thus, we need to minimize the expectation on masks that the selector model parameterizes their

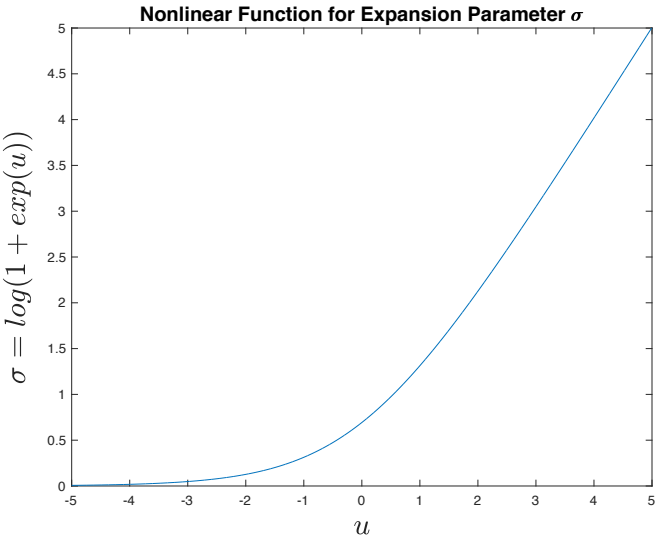

Fig. 2: Our proposed nonlinear function to calculate the expansion parameter of a predicted RBF Kernel of the selector for the CIFAR-10 dataset.

distribution. Empirically, we use Monte-Carlo sampling and sample one mask for each input image x. However, sampling is not a differentiable operation. Hence, it is impossible to train the selector's parameters by optimizing them using back-propagation schemes when we directly sample from the predicted distribution. A workaround to this problem is to replace non-differentiable sampling from a categorical distribution with a differentiable sampling from the Gumbel-Sigmoid distribution [5, 9]. In summary, the binary mask can be generated by using the following function with the Gumbel-Sigmoid trick:

$$m_{i,j} = \frac{1}{1 + \exp(-\frac{\log(f_{BP}(i,j;c_z,c_t,\sigma)) + g_j}{\tau})} \tag{14}$$

such that g_i values are sampled from the Gumbel distribution, and τ is called the 'temperature' parameter that determines 'sharpness' of the sample. Low τ values result in samples close to Bernoulli distribution (binary), but higher τ values make the output distribution more similar to uniform. We set $\tau = 1$ for training our selector model for all experiments. We then can insert Eq. (14) into Eq. (11) to optimize the selector model.

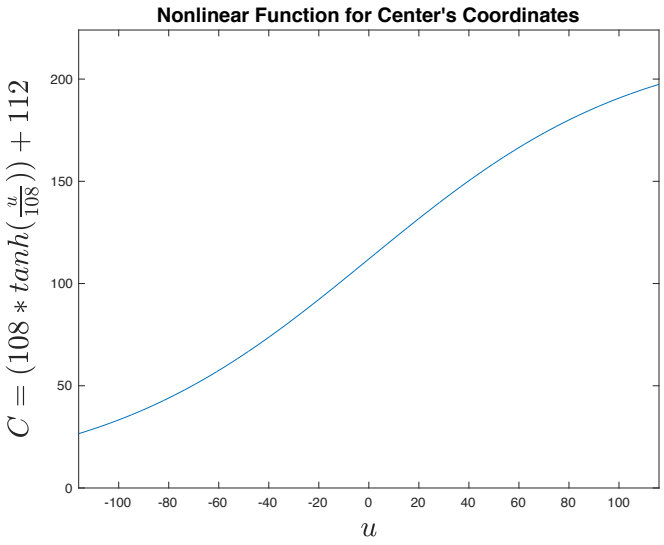

Fig. 3: Our proposed nonlinear function to calculate the center's coordinates of a predicted RBF Kernel of the selector for the ImageNet dataset.

3.2 More Details about Pruning

Similarly, we also use Gumbel-Sigmoid trick to characterize each channel:

$$v_l = \frac{1}{1 + \exp(-\frac{\theta_{g_l} + g_j + b}{\tau})} \tag{15}$$

where θ_{g_l} is the parameters for l-th layer, and $\mathbf{v} = [v_1, \cdots, v_L]$. We also insert a constant b for starting pruning from the whole model. In experiments, we set $b = 3$ and $\tau = 0.4$ for pruning channels. To achieve pruning, we multiply v_l to its corresponding feature map $\mathcal{F}_l$ after activation functions:

$$\hat{\mathcal{F}}_l = v_l \odot \mathcal{F}_l \tag{16}$$

where $\odot$ is element-wise product, and v_l is first expanded to the same size of $\mathcal{F}_l$.

In practice, we let $\mathcal{R}_{res}(x, y) = \log(\max(x, y)/y)$, which effectively push $\mathcal{R}_{res}$ to 0. Other regression loss functions like MSE and MAE can not achieve similar functions, and they often fail when applied to compact models like MobileNet-V2 [14].

3.3 More Samples of our AEM's Predictions

We show visual examples of our proposed AEM's predictions on ImageNet and CIFAR-10 in the following pages. In each row from left to right, we show an

input image, the predicted distribution of explanatory masks over the input, the predicted distribution shown over the input image, a mask sampled from the predicted distribution, and the input masked by the sampled mask. ImageNet images have higher resolution than CIFAR-10 ones. Thus, their sampled masks look more coherent than CIFAR-10 images. However, we can see that our selector model almost always puts the mode of its RBF kernel on the salient part of the input.

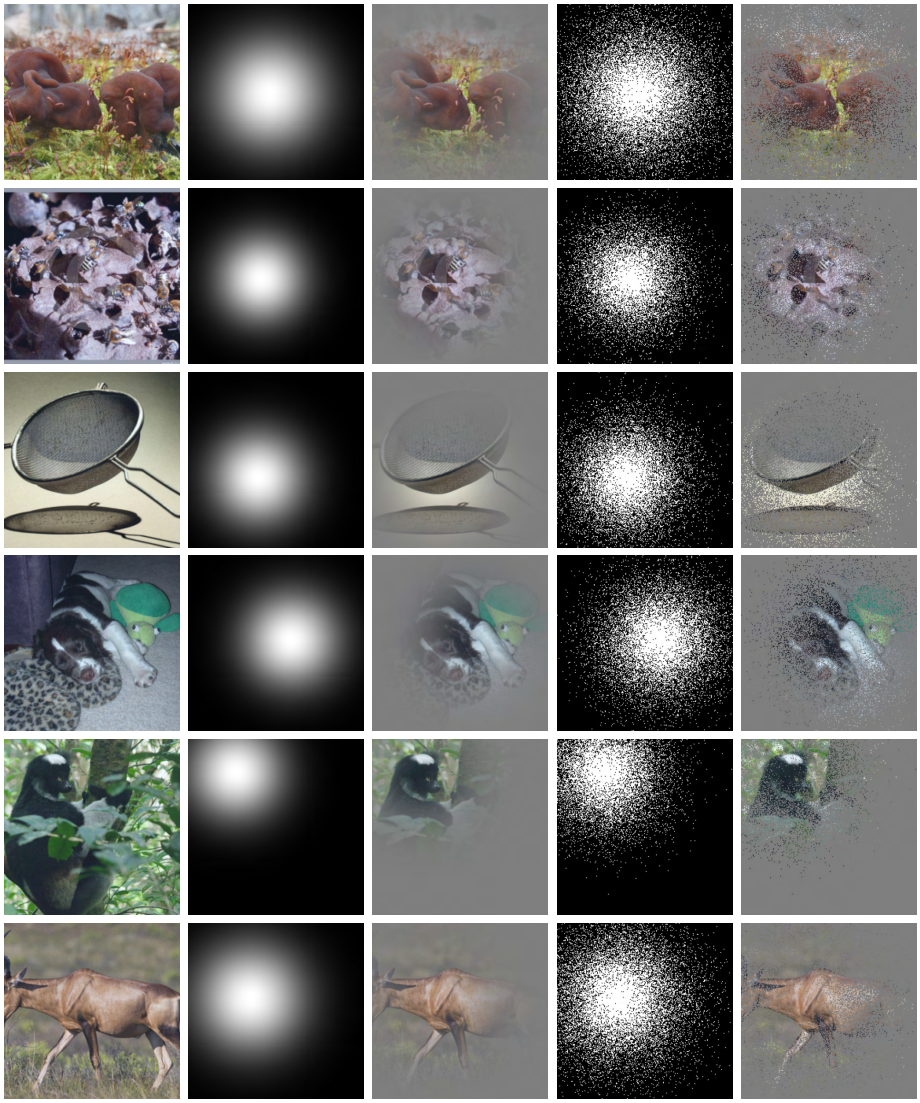

Fig. 4: **ImageNet Examples. Columns from left to right:** input image, distribution over explanatory masks predicted by selector, predicted distribution shown over input, a sampled mask from the predicted distribution, and input image masked by the sampled mask. **Class of input images from top to bottom:** 'Gyromitra', 'Honeycomb', 'Strainer', 'English springer', 'Indri brevicaudatus', 'Hartebeest'.

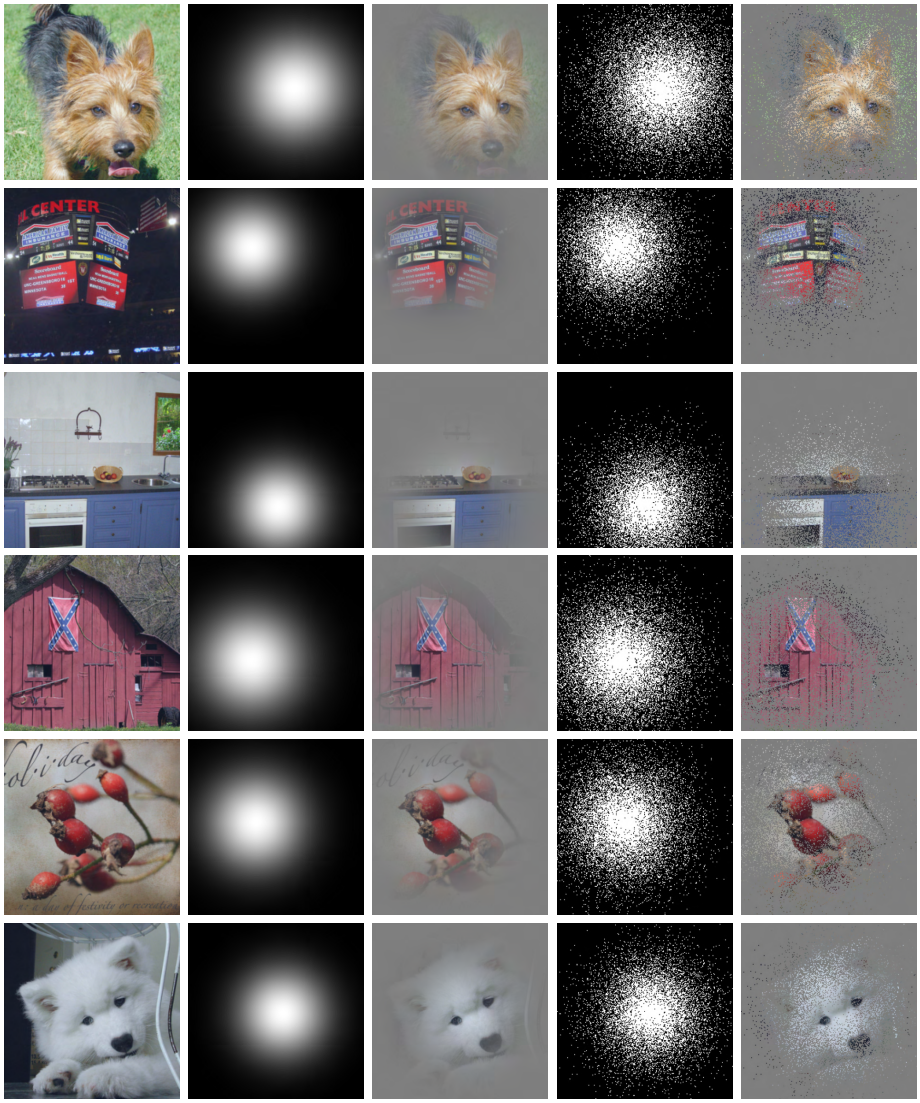

Fig. 5: **ImageNet Examples. Columns from left to right:** input image, distribution over explanatory masks predicted by selector, predicted distribution shown over input, a sampled mask from the predicted distribution, and input image masked by the sampled mask. **Class of input images from top to bottom:** 'Australian terrier', 'Scoreboard', 'Microwave oven', 'Barn', 'Rosehip', 'Samoyed'.

Fig. 6: **ImageNet Examples. Columns from left to right:** input image, distribution over explanatory masks predicted by selector, predicted distribution shown over input, a sampled mask from the predicted distribution, and input image masked by the sampled mask. **Class of input images from top to bottom:** 'Miniskirt', 'Soccer ball', 'Jeep', 'Albatross', 'Tench', 'China cabinet'.

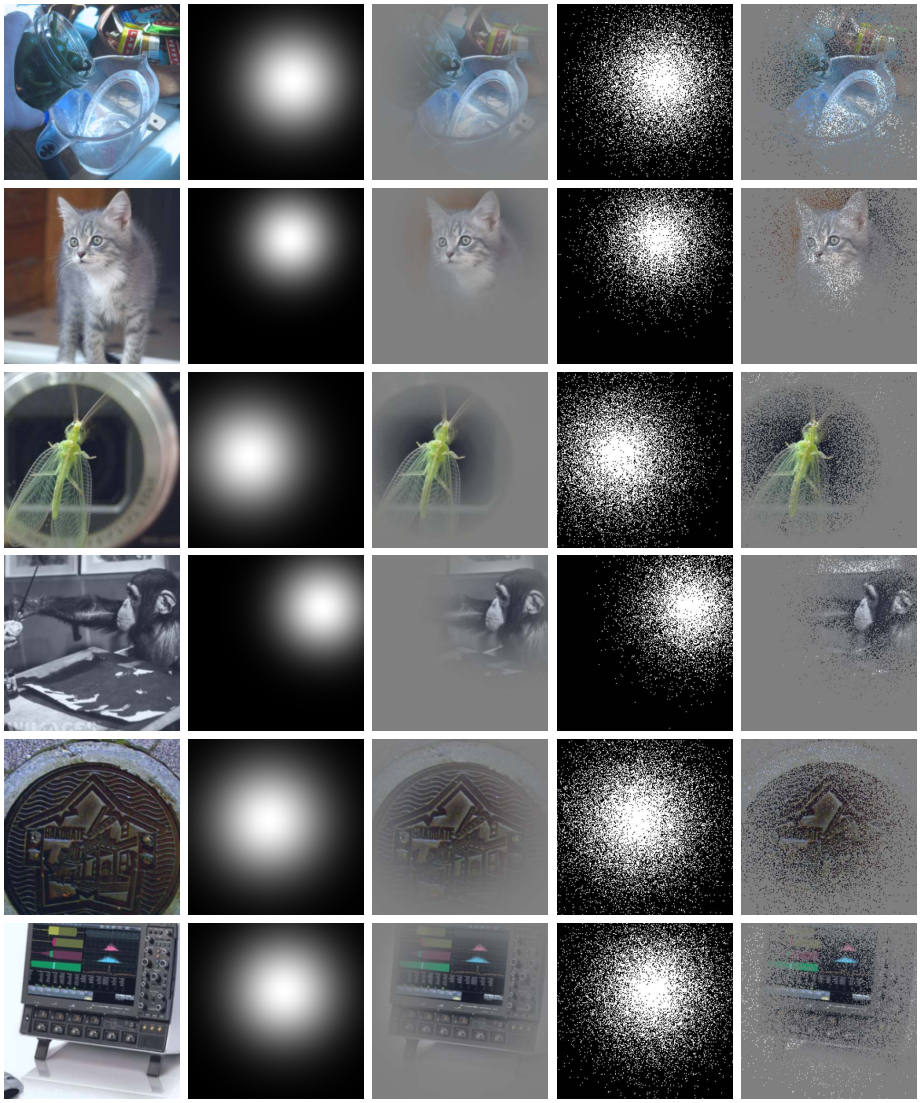

Fig. 7: **ImageNet Examples. Columns from left to right:** input image, distribution over explanatory masks predicted by selector, predicted distribution shown over input, a sampled mask from the predicted distribution, and input image masked by the sampled mask. **Class of input images from top to bottom:** 'Kimono', 'Whippet', 'Poncho', 'Drilling Platform', 'Steel Drum', 'Black Grouse'.

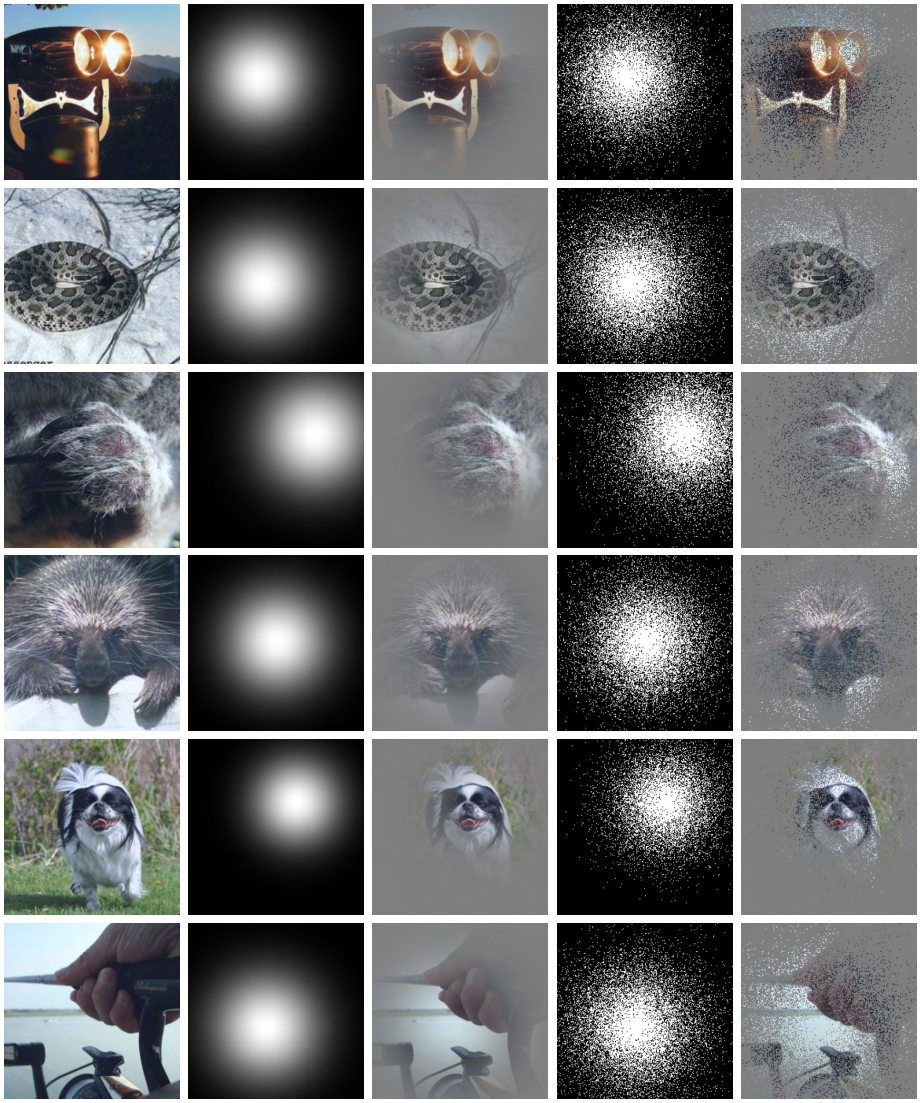

Fig. 8: **ImageNet Examples. Columns from left to right:** input image, distribution over explanatory masks predicted by selector, predicted distribution shown over input, a sampled mask from the predicted distribution, and input image masked by the sampled mask. **Class of input images from top to bottom:** 'Binoculars', 'Horned viper', 'Native bear', 'Hedgehog', 'Japanese spaniel', 'Reel'.

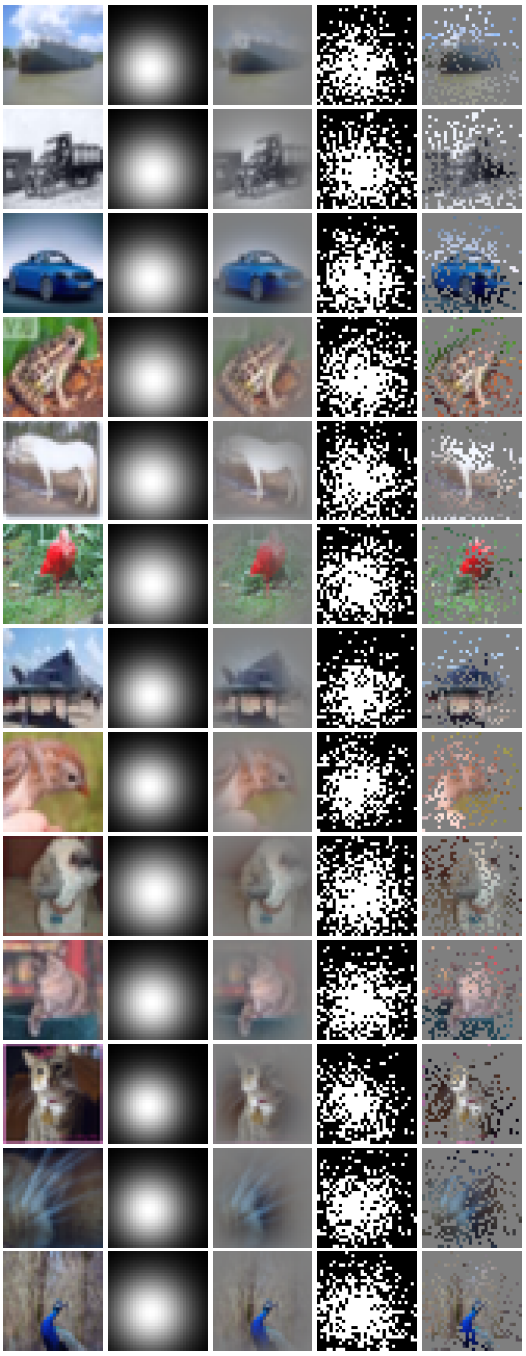

Fig. 9: **CIFAR-10 Examples. Class of input images from top to bottom:** 'Ship', 'Truck', 'Automobile', 'Frog', 'Horse', 'Bird', 'Airplane', 'Bird', 'Dog' 'Cat', 'Cat', 'Cat', 'Bird'.

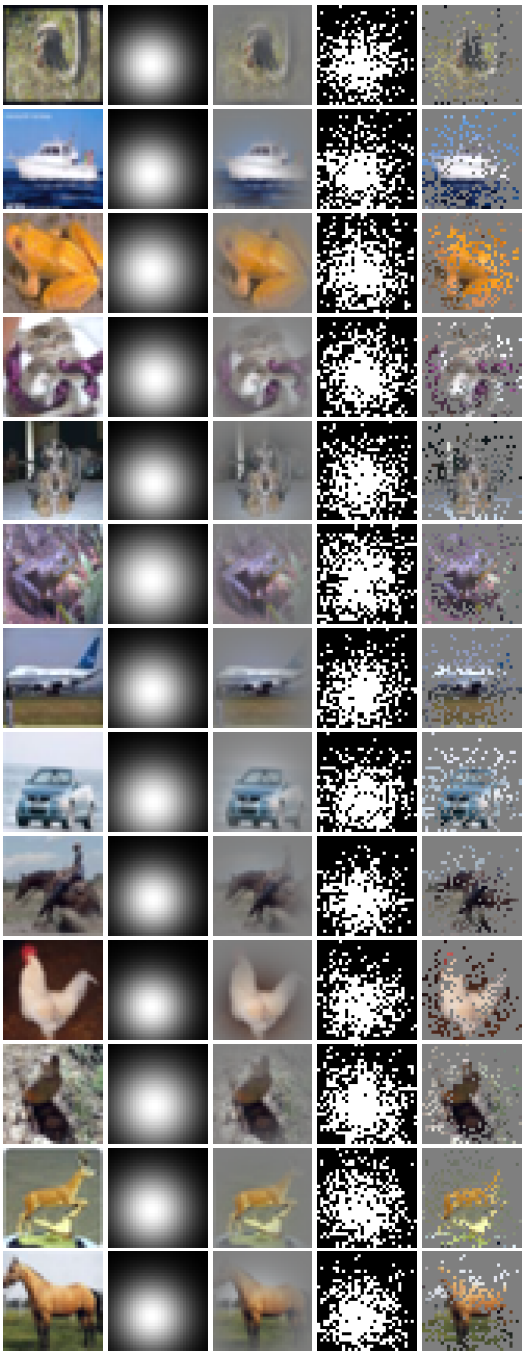

Fig. 10: **CIFAR-10 Examples. Class of input images from top to bottom:** 'Bird', 'Ship', 'Frog', 'Cat', 'Dog', 'Frog', 'Airplane', 'Automobile', 'Horse' 'Bird', 'Bird', 'Deer', 'Horse'.

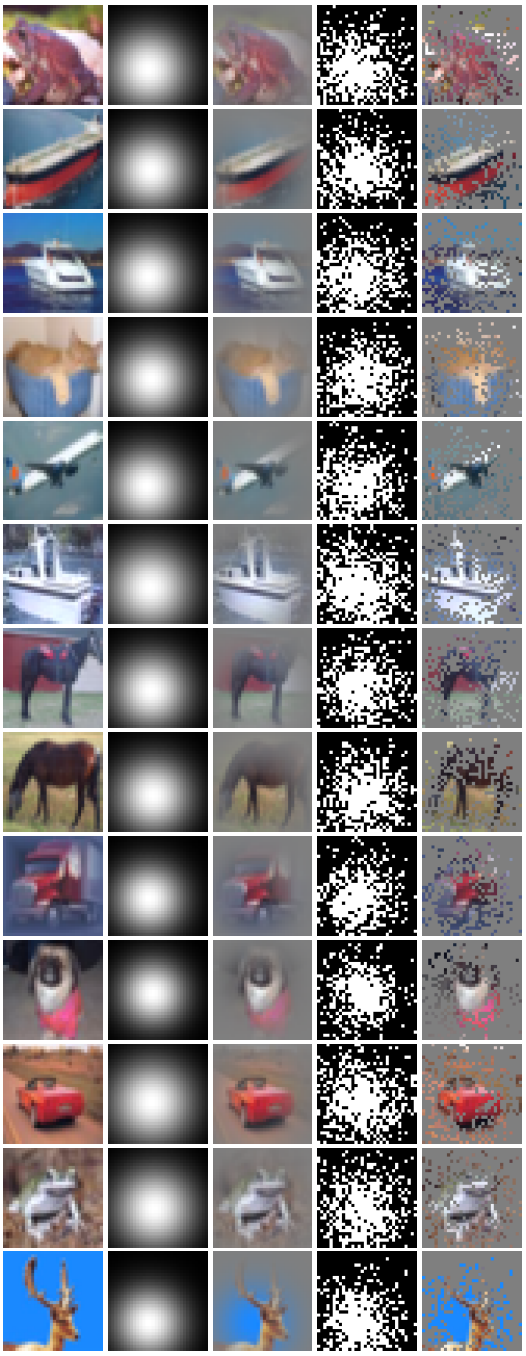

Fig. 11: **CIFAR-10 Examples. Class of input images from top to bottom:** 'Frog', 'Ship', 'Ship', 'Cat', 'Airplane', 'Ship', 'Horse', 'Horse', 'Truck', 'Dog', 'Automobile', 'Frog', 'Deer'.

Fig. 12: **CIFAR-10 Examples. Class of input images from top to bottom:** 'Dog', 'Airplane', 'Horse', 'Automobile', 'Horse', 'Ship', 'Ship', 'Automobile', 'Cat', 'Airplane', 'Ship', 'Airplane', 'Dog'.

References

1. Chen, J., Song, L., Wainwright, M., Jordan, M.: Learning to explain: An information-theoretic perspective on model interpretation. In: International Conference on Machine Learning. pp. 883–892. PMLR (2018)
2. Dabkowski, P., Gal, Y.: Real time image saliency for black box classifiers. In: Guyon, I., von Luxburg, U., Bengio, S., Wallach, H.M., Fergus, R., Vishwanathan, S.V.N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA. pp. 6967–6976 (2017)
3. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition. pp. 248–255. Ieee (2009)
4. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
5. Jang, E., Gu, S., Poole, B.: Categorical reparameterization with gumbel-softmax. In: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. OpenReview.net (2017), https://openreview.net/forum?id=rkE3y85ee
6. Jethani, N., Sudarshan, M., Aphinyanaphongs, Y., Ranganath, R.: Have we learned to explain?: How interpretability methods can learn to encode predictions in their interpretations. In: International Conference on Artificial Intelligence and Statistics. pp. 1459–1467. PMLR (2021)
7. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015), http://arxiv.org/abs/1412.6980
8. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009)
9. Maddison, C.J., Mnih, A., Teh, Y.W.: The concrete distribution: A continuous relaxation of discrete random variables. In: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. OpenReview.net (2017), https://openreview.net/forum?id=S1jE5L5gl
10. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: Fürnkranz, J., Joachims, T. (eds.) Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel. pp. 807–814. Omnipress (2010), https://icml.cc/Conferences/2010/papers/432.pdf
11. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems **32**, 8026–8037 (2019)
12. Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical image computing and computer-assisted intervention. pp. 234–241. Springer (2015)
13. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M.S., Berg, A.C., Fei-Fei, L.: Imagenet large scale visual recognition challenge. Int. J. Comput. Vis. **115**(3), 211–252 (2015). https://doi.org/10.1007/s11263-015-0816-y, https://doi.org/10.1007/s11263-015-0816-y

14. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4510–4520 (2018)
15. Shi, W., Caballero, J., Huszár, F., Totz, J., Aitken, A.P., Bishop, R., Rueckert, D., Wang, Z.: Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1874–1883 (2016)
16. Yoon, J., Jordon, J., van der Schaar, M.: Invase: Instance-wise variable selection using neural networks. In: International Conference on Learning Representations (2018)