50.039 Theory and Practice of Deep Learning

W12-S1 Interpretability in Deep Learning

Matthieu De Mari, Berrak Sisman

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

About this week (Week 12, an informative lecture about interpretability and AI ethics)

1. What is **interpretability** and why is it necessary?
2. What are typical examples of **models mistakes** and **biases**?
3. What are the **two great families of algorithms for interpretability**?
4. What is the **t-SNE** algorithm and how can it help to interpret a model?
5. What is the **activation maximization algorithm**?
6. What are **occlusion-based approaches**?
7. What are **gradient-based approaches for interpretability** and their limits?
8. What is the **LIME** algorithm?
9. What is the **guided backpropagation algorithm**?
10. What is the **LRP algorithm**?
11. (What are **open questions** in research in interpretability?)

What is interpretability? (or eq. Explainability)

Definition (Interpretability in Neural Networks):

Interpretability is the degree to which a human can understand the cause of a decision, i.e. the degree to which a human can consistently predict the model's result.

The higher the interpretability of a machine learning model, the easier it is for someone to comprehend why certain decisions or predictions have been made.

Typically, a model is more interpretable than another model if its decisions are easier for a human to comprehend than decisions from the other model.

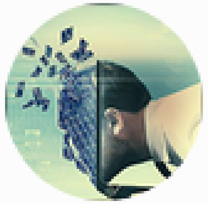
Why do we need interpretability?

There are many reasons why we need interpretability.

- (**Reason #0:** Because humans are easily scared by things they do not understand and do not like things that are not easily explainable.)
- **Reason #1:** Improving Neural Networks decisions and training methods.
- **Reason #2:** Confirming what a Neural Network has learnt and what it seems to implement to reach a decision.
- **Reason #3:** Identify the reasons for mistakes and cognitive biases in the decisions of a Neural Network in an attempt to fix them.

Reason #0: Because humans do not like things that are not easily explainable.

Should We Be Afraid of AI?



Ron Schmelzer Contributor
COGNITIVE WORLD Contributor Group ⓘ
AI

Philosophers, computer scientists, and even Elon Musk are concerned artificial intelligence could one day destroy the world. Some scholars argue it's is the most pressing existential risk humanity might ever face, while others mostly dismiss the hypothesized danger as unfounded doom-mongering.

https://theconversation.com/people-dont-trust-ai-heres-how-we-can-change-that-87129

https://www.vice.com/en/article/7x48kg/the-divide-between-people-who-hate-and-love-artificial-intelligence-is-not-real

Reason #0: Because humans do not like things that are not easily explainable.

https://theconver
trust-ai-heres-how-we-can-change-that-87129

en/article/7x48kg/the-
-who-hate-and-love-
artificial-intelligence-is-not-real

Reason #1

Reason #1: Improving Neural Networks decisions and training methods.

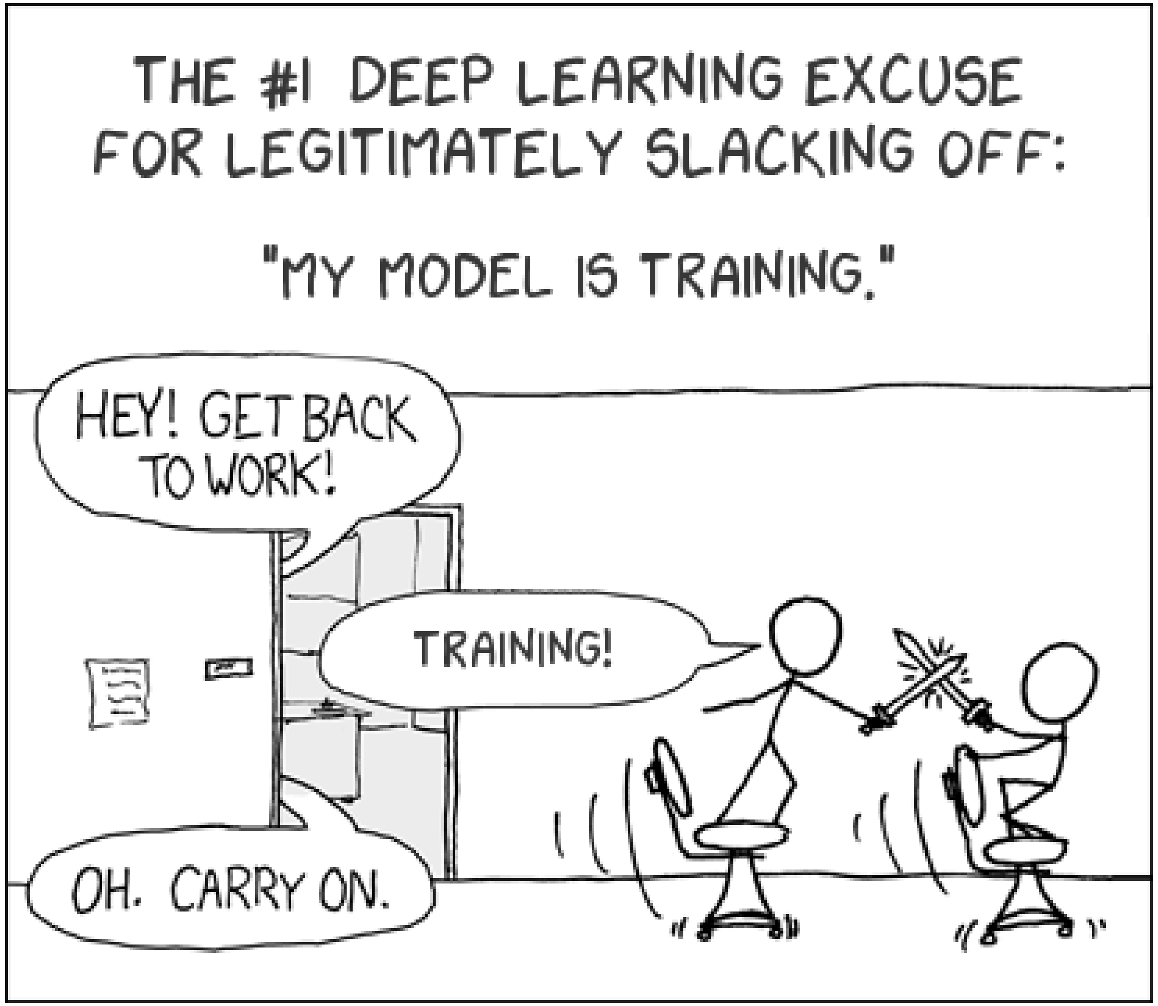
- Neural Networks are black-boxes, and a very empirical science.
- Experience shows that these algorithms have been able to (partially) answer some problems we had no clue how to address.
- But still very obscure…

https://www.explainxkcd.com/wiki/index.php/1838:_Machine_Learning

Reason #1

Reason #1: Improving Neural Networks decisions and training methods.

- Understanding how NNs tend to learn could lead to more efficient training procedures.
- (Right now we kind of pray that an empirically validated model architecture and some Gradient Descent will make it work?)

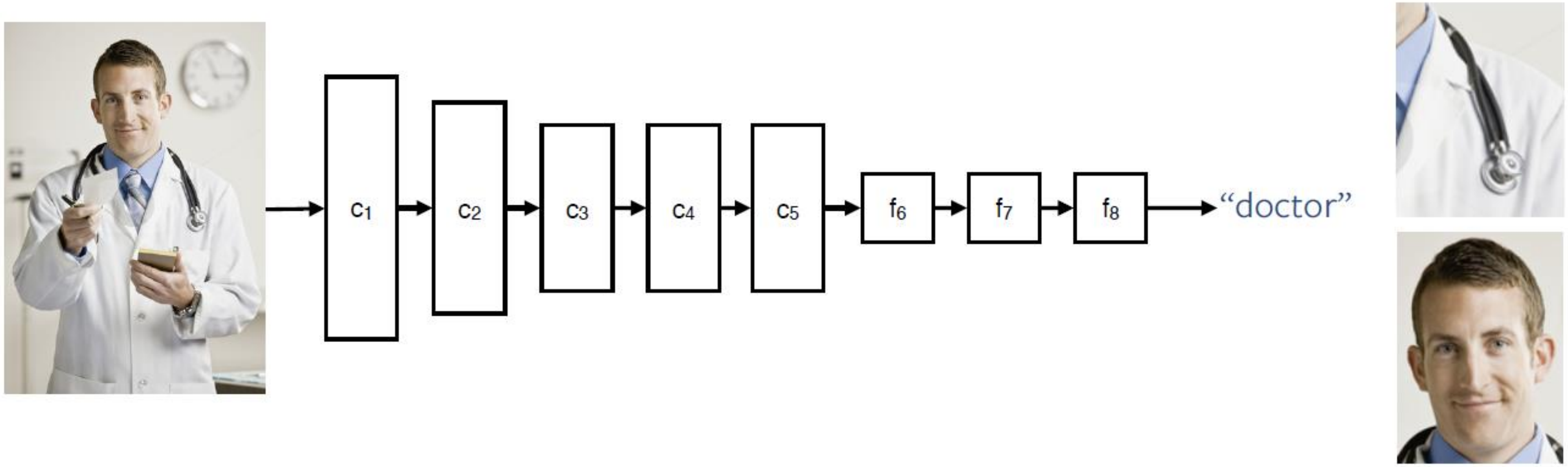


https://www.explainxkcd.com/wiki/index.php/303:_Compiling

Reason #2

Reason #2: Confirming what a Neural Network has learnt and what it seems to implement to reach a decision.

- What features is the neural network learning to reach a decision?
- What similarities is the neural network looking for in the dataset?
- What are the neurons doing anyway?



Reason #3

Reason #3: Identify the reasons for mistakes and cognitive biases in the decisions of a Neural Network in an attempt to fix them.

Reason #3

Reason #3: Identify the reasons for mistakes and cognitive biases in the decisions of a Neural Network in an attempt to fix them.

Definition (cognitive bias):

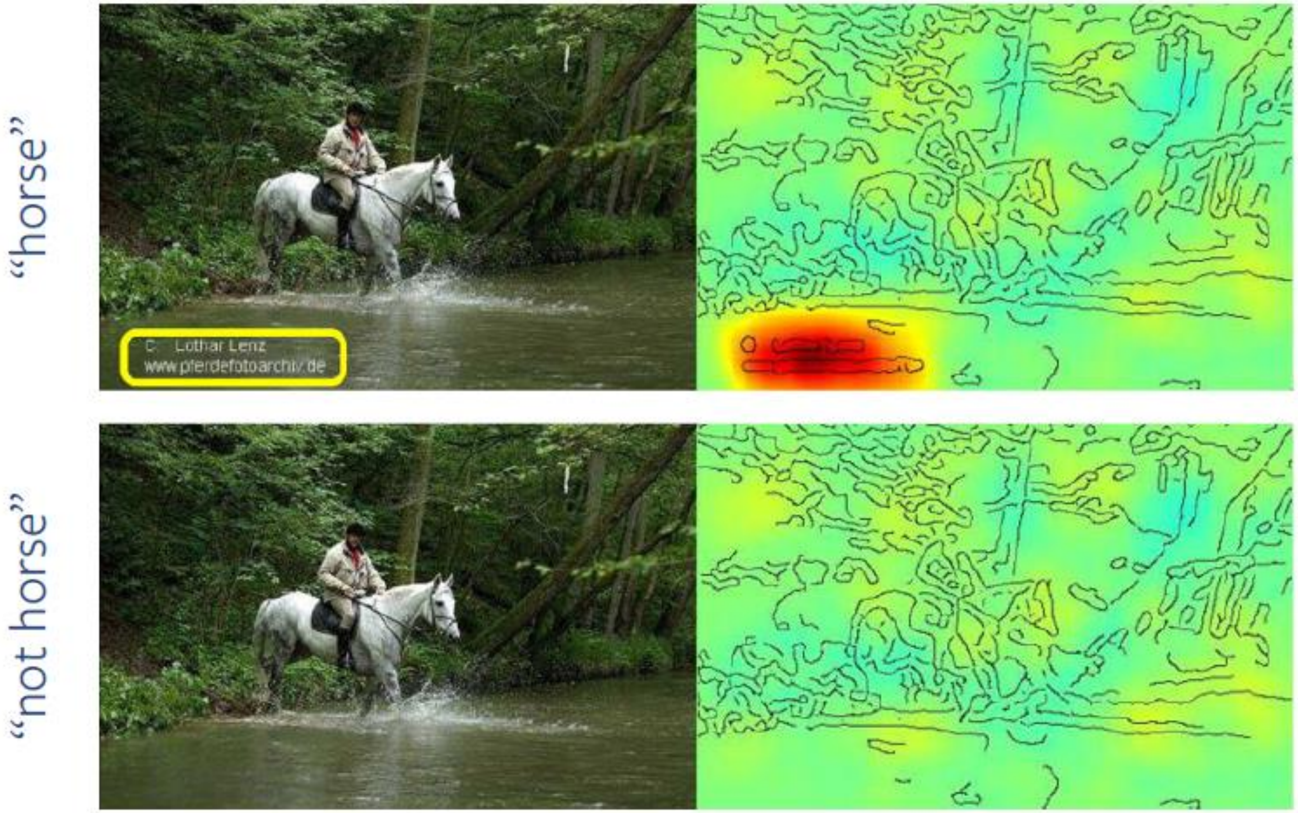
Similar to cognitive biases in psychology.

In Neural Networks, a **cognitive bias** refers to a feature, which has been used by the NN to reach a certain decision, but should not have been used as it seems illogical to humans.

Reason #3

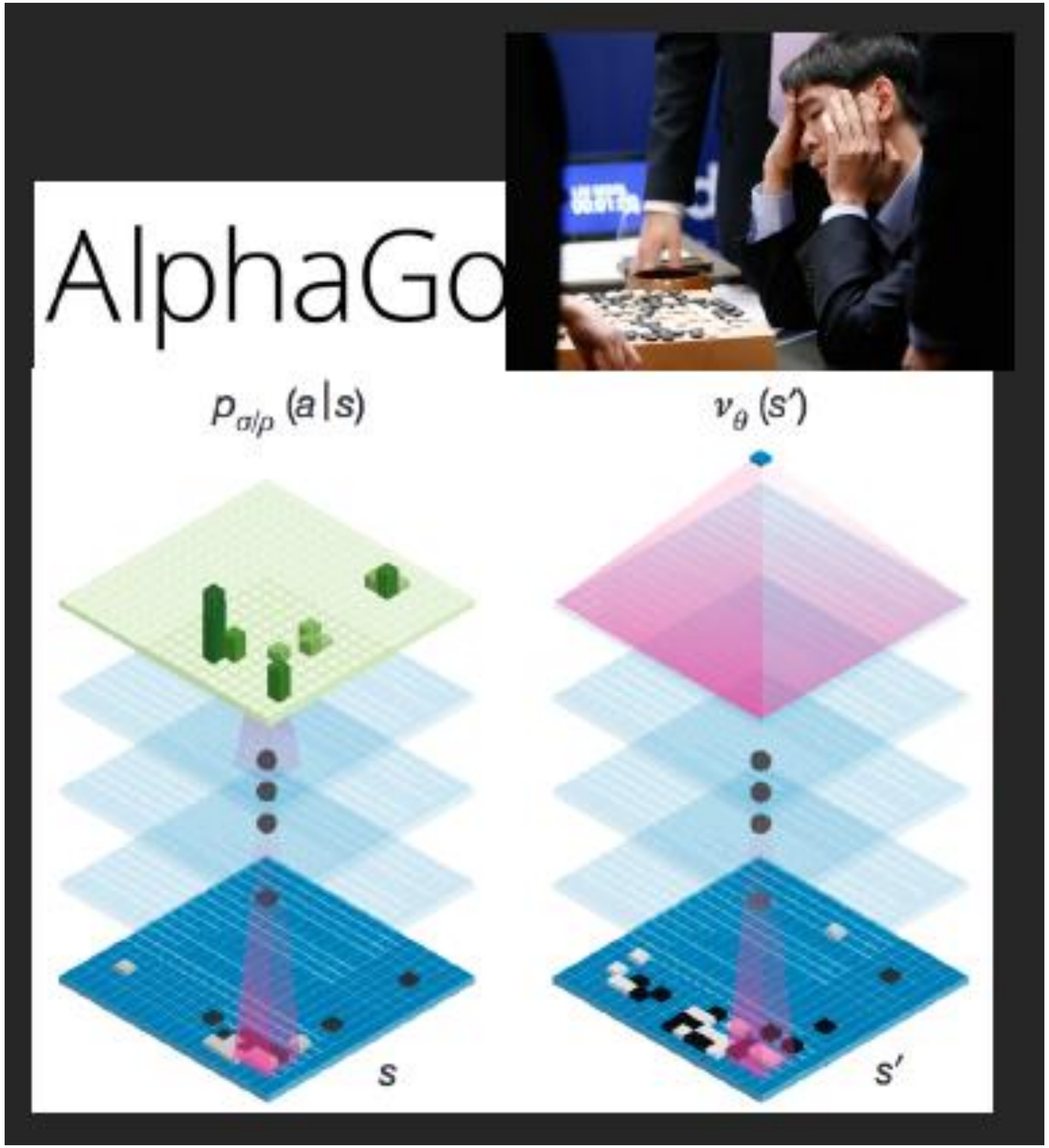
Reason #3: Identify the reasons for mistakes and cognitive biases in the decisions of a Neural Network in an attempt to fix them.

- The Neural network on the right draws information from the photo legend and recognizes that this photographer likes to take pictures of horses…
- That is a cognitive bias!



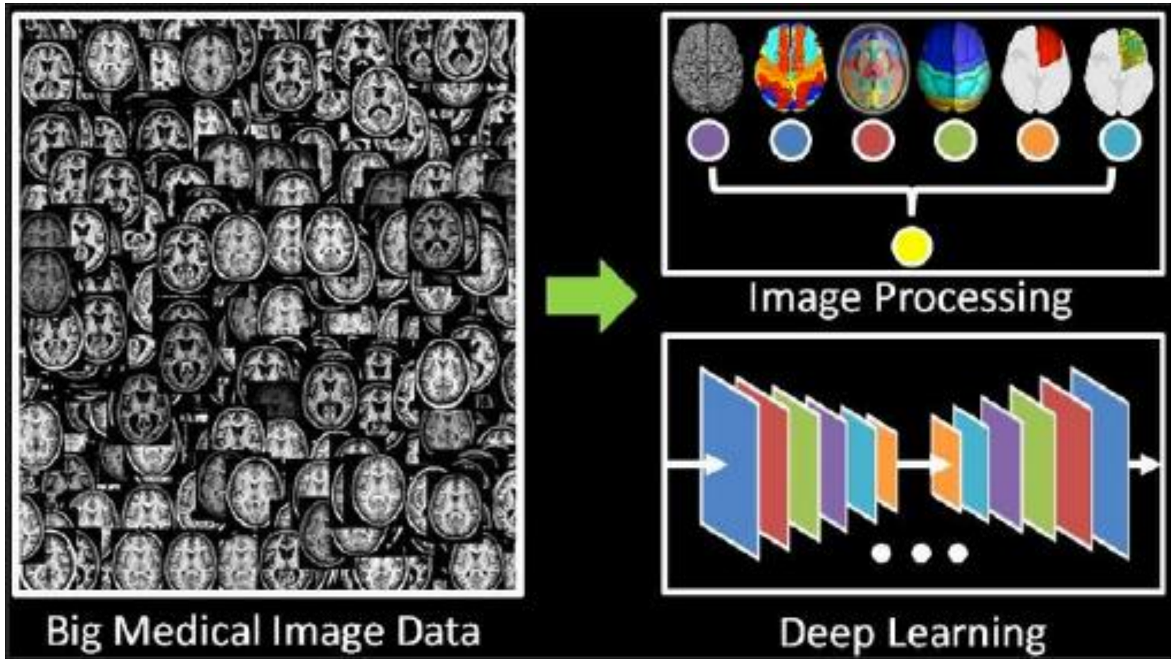
Reason #3

- **Causality:** Check that only causal relationships are picked up. No cognitive biases.



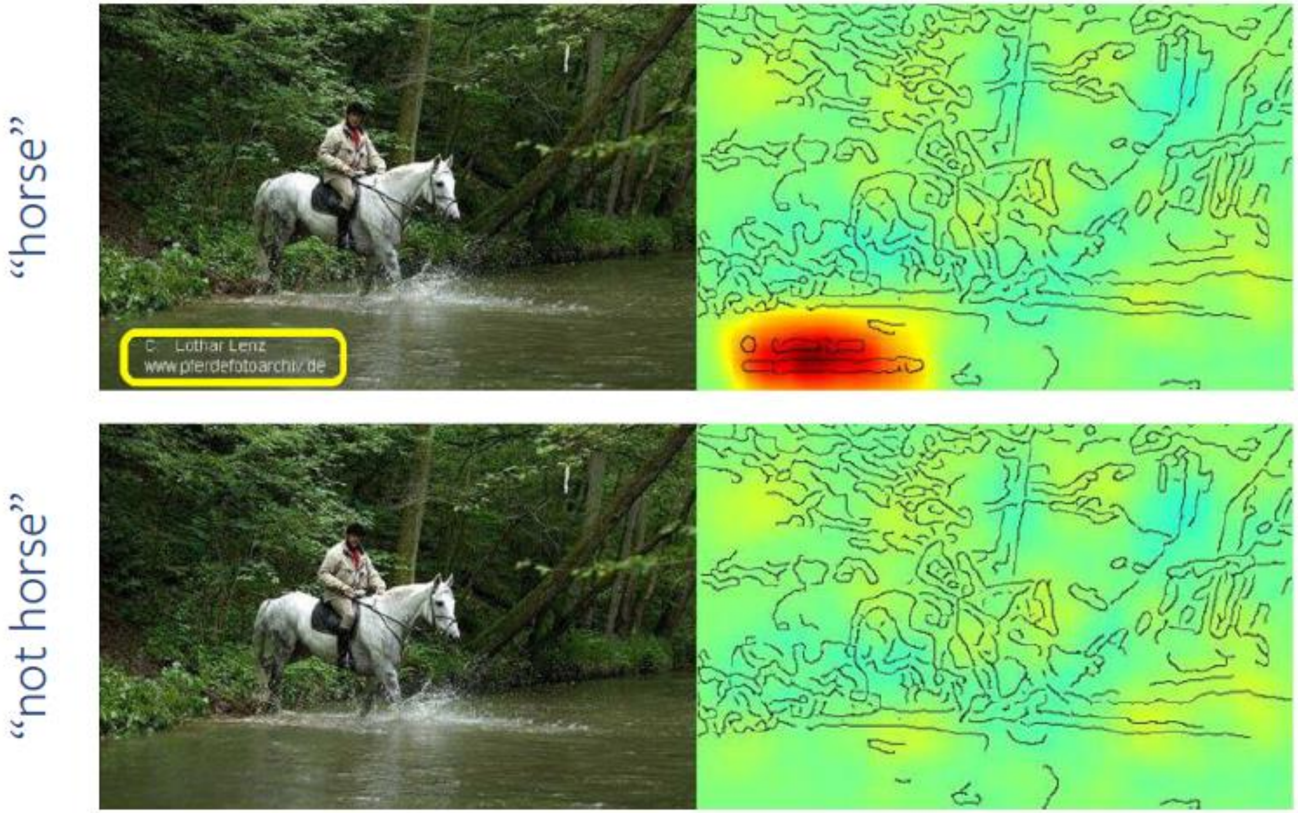
Reason #3

- **Causality:** Check that only causal relationships are picked up. No cognitive biases.



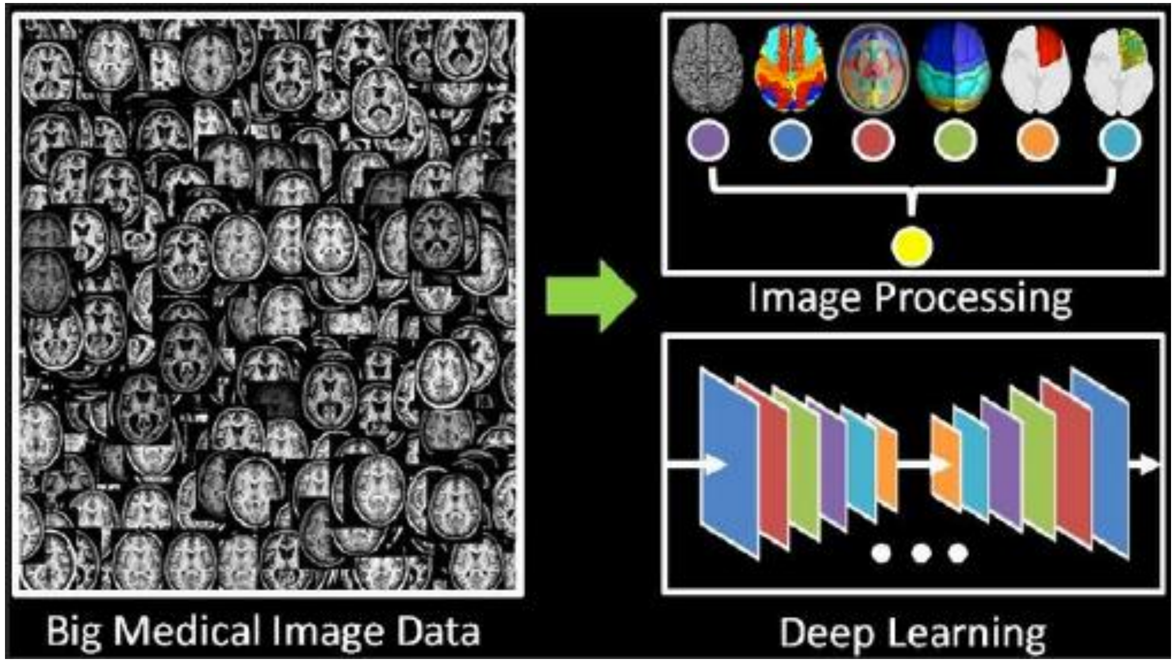
Reason #3

- **Causality:** Check that only causal relationships are picked up. No cognitive biases.
- **Reliability or Robustness:** Ensuring that small changes in the input do not lead to large changes in the prediction.



Reason #3

- **Causality:** Check that only causal relationships are picked up. No cognitive biases.
- **Reliability or Robustness:** Ensuring that small changes in the input do not lead to large changes in the prediction.



Reason #3

- **Causality:** Check that only causal relationships are picked up. No cognitive biases.
- **Reliability or Robustness:** Ensuring that small changes in the input do not lead to large changes in the prediction.
- **Privacy:** Ensuring that sensitive information in the data is protected.

Reason #3

- **Fairness:** Ensuring that predictions are unbiased and do not implicitly or explicitly discriminate against underrepresented groups.
- An interpretable model can tell you why it has decided that a certain person should not get a loan, and it becomes easier for a human to judge whether the decision is based on a learned demographic (e.g. racial) bias.

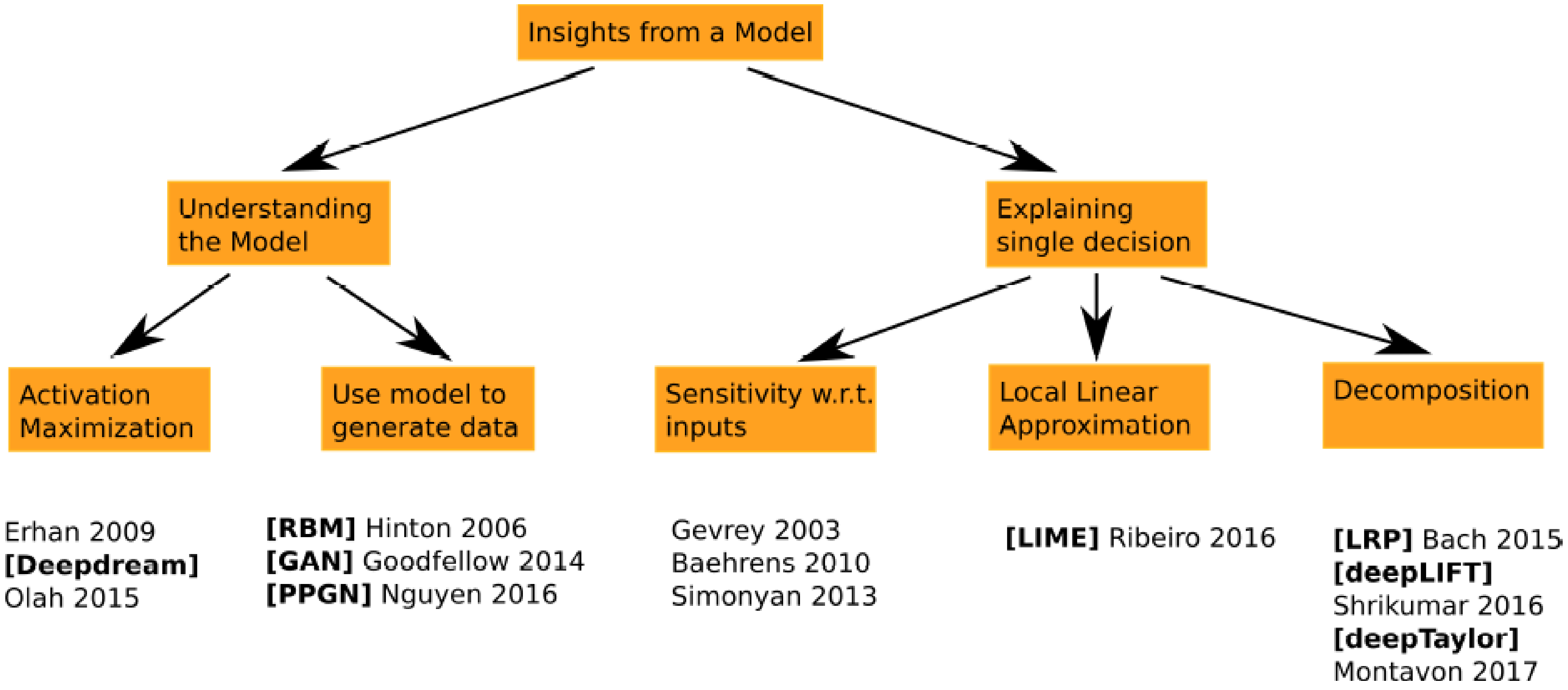
NEWS · 24 OCTOBER 2019 · UPDATE 26 OCTOBER 2019

Millions of black people affected by racial bias in health-care algorithms

Study reveals rampant racism in decision-making software used by US hospitals — and highlights ways to correct it.

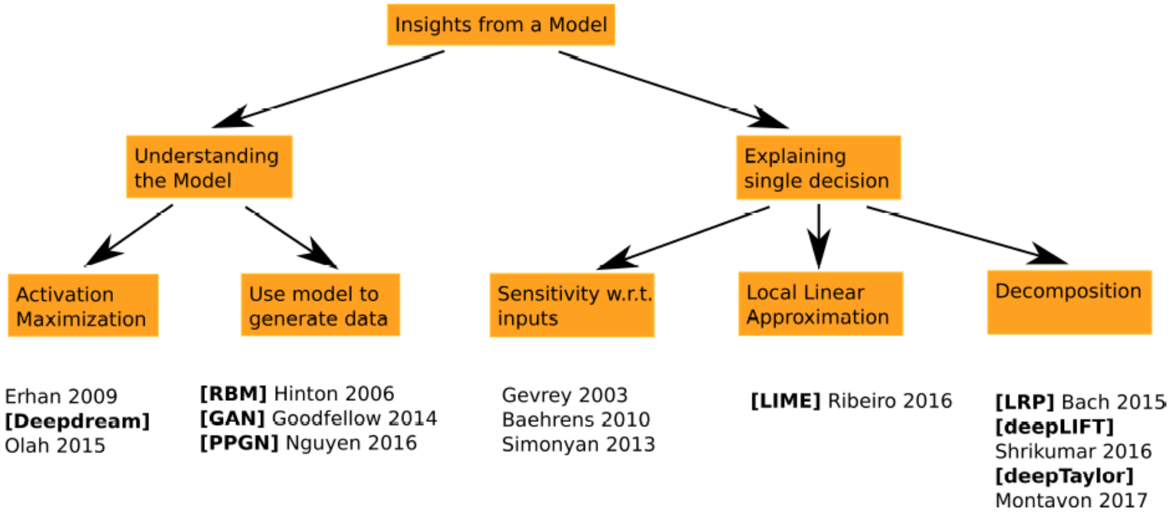
https://www.nature.com/articles/d41586-019-03228-6

The two families of interpretability methods



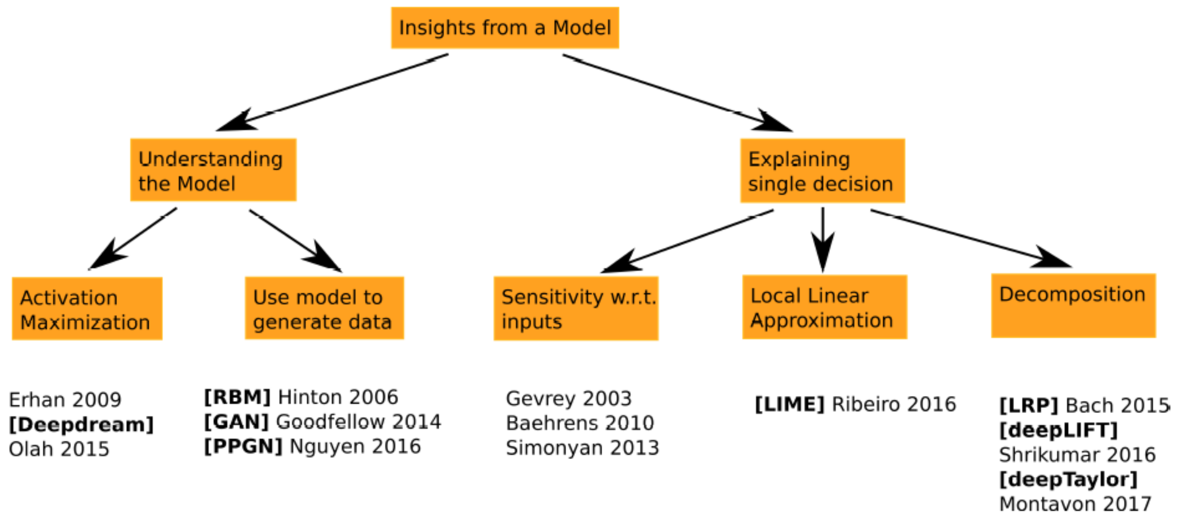
The two families of interpretability methods

- **Explaining a single decision:** this first class of methods attempts to understand how the data is interpreted by the model.
- Which part of the image is used for the decision?
- Which image would maximize this particular class?
- Which part of the image would have to be modified to affect the decision?



The two families of interpretability methods

- **Understanding the model:** this second class of methods focuses on understanding the meaning of trained parameters in the models.
- What is this neuron computing?
- What feature is this convolution filter checking?
- Etc.



Methods to be discussed in this class

- t-SNE
- Dataset evaluation
- Gradient maps and gradient-input maps
- LIME
- Occlusion-based methods
- Activation maximization on samples
- Activation maximization on layers
- Guided back-propagation
- LRP

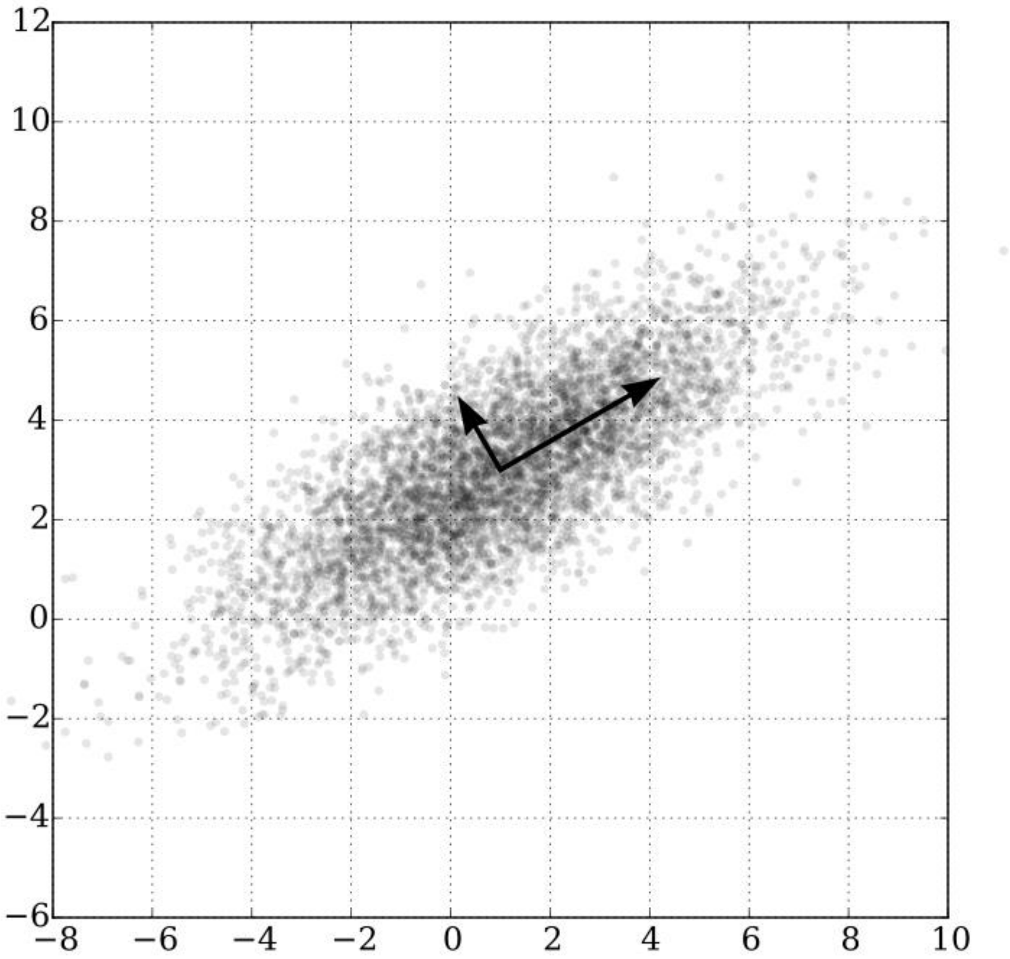
- **Note:** we will not have time to explore the implementation of said methods, and that is alright!
- Mostly interested in the intuition and general discussion of this topic anyway!

Reminder: Principal Component Analysis (PCA)

In machine learning, **PCA** is a **linear dimension reduction technique** that seeks to maximize variance and preserves large pairwise distances.

The principal components of a collection of samples of dimension p, is a sequence of direction vectors.

Each i-th vector is the direction of a line that best fits the data, while being orthogonal to the previous $(i-1)$-th vectors.



t-SNE, a definition

Definition (t-SNE):

The **t-Distributed Stochastic Neighbor Embedding (t-SNE)** is an unsupervised, non-linear technique primarily used for data exploration and visualizing high-dimensional data.

In simpler terms, t-SNE gives you a feel or intuition of how the data is arranged in a high-dimensional space.

The t-SNE differs from PCA by preserving only the small pairwise distances or local similarities, whereas PCA is concerned with preserving large pairwise distances to maximize variance.

t-SNE implementation

The t-SNE algorithm calculates a **similarity measure between pairs** of instances in the high dimensional space and in the low dimensional space.

It then tries to optimize these two similarity measures using a cost function.

Objective: given N datapoints, create N 2-D embeddings according to their similarities.

t-SNE implementation

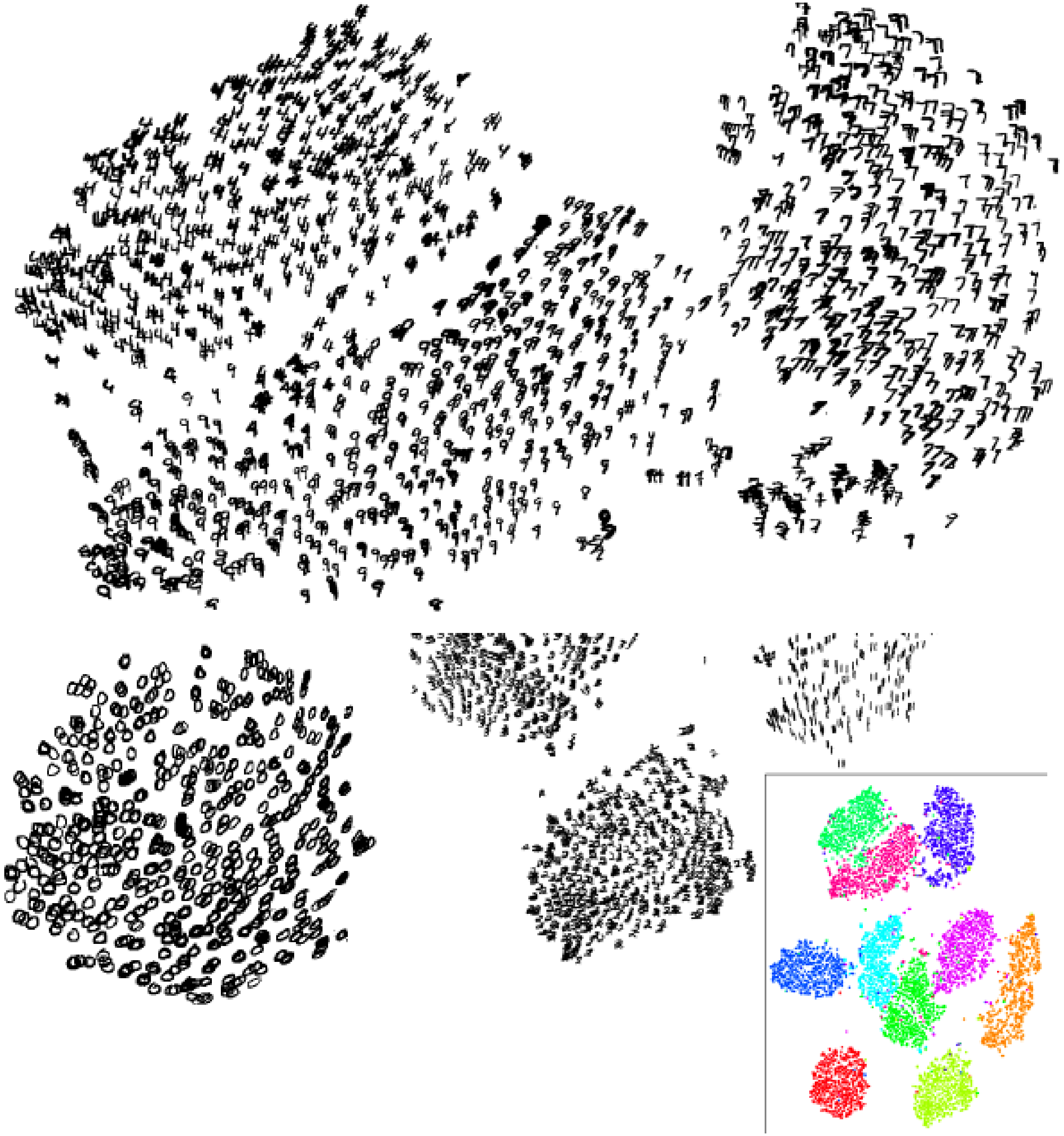
The t-SNE algorithm calculates a **similarity measure between pairs** of instances in the high dimensional space and in the low dimensional space.

It then tries to optimize these two similarity measures using a cost function.

Objective: given N datapoints, create N 2-D embeddings according to their similarities.

The t-SNE method can be broken down in three steps.

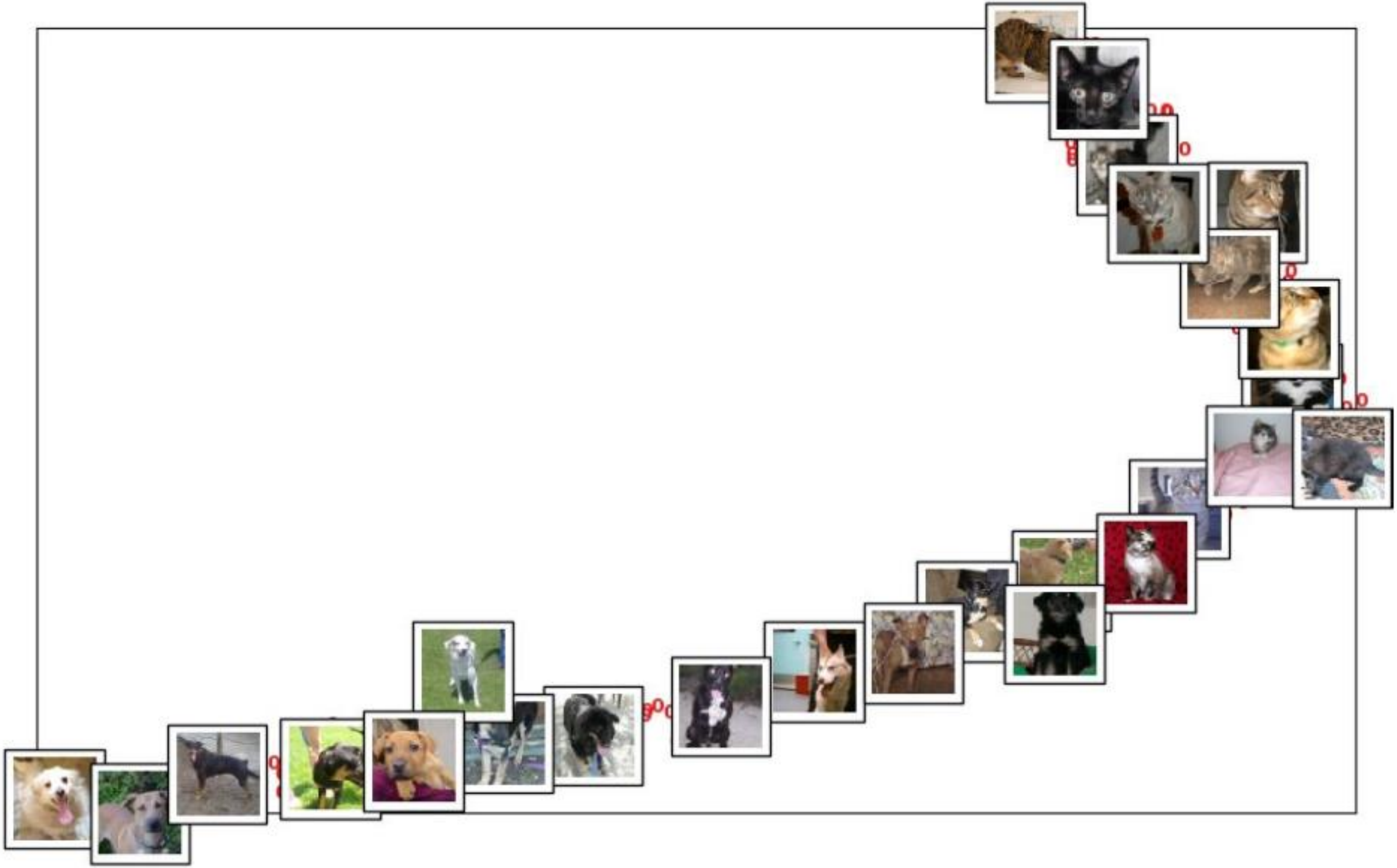
- **Step 1:** Gaussian distribution between samples in high-dimension space.
- **Step 2:** Cauchy distribution between representations in low-dimension space.
- **Step 3:** Minimize the KL divergence between both distributions.



Dos and Don’ts for t-SNE

Do

- Use it on your dataset and trained model outputs!
- Use t-SNE to get some qualitative hypotheses on what your features capture.
- Use t-SNE to get insights as to whether or not the data is separable in the first place (if not, very little hope to train a classifier!)



Dos and Don’ts for t-SNE

Do

- Use it on your dataset and trained model outputs!
- Use t-SNE to get some qualitative hypotheses on what your features capture.
- Use t-SNE to get insights as to whether or not the data is separable in the first place (if not, very little hope to train a classifier!)

Don’t

- Present a "proof by t-SNE": your map is not the data!
- Attempt to interpret distances between points in the t-SNE visualization.

Dos a

Do

- Use it c E": your trained
- Use t-S ances qualitat SNE your fe
- Use t-S whethe separal not, ve classifie

Dos and Don’ts for t-SNE

Do

- Use it on your dataset and trained model outputs!
- Use t-SNE to get some qualitative hypotheses on what your features capture.
- Use t-SNE to get insights as to whether or not the data is separable in the first place (if not, very little hope to train a classifier!)

Don’t

- Present a "proof by t-SNE": your map is not the data!
- Attempt to interpret distances between points in the t-SNE visualization.
- Forget that low-dimensional metric spaces cannot capture non-metric similarities.

Do

Do

- Us ': your
  tra
- Us nces
  qu NE
  you
- Us nal
  wh ture
  sep
  no
  cla

Photo reproduced with permission by Stefano Costatini (CC 2.0).

Photo reproduced with permission by Elliott Brown (CC 2.0).

To conclude about t-SNE

- t-SNE is a valuable tool in generating hypotheses and understanding.
- However, it does not produce conclusive evidence.

- This is automatically done with libraries such as sklearn.
- For instance, see https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html)

Analyzing features via dataset evaluation

- Consider the images in your dataset and a trained model.
- For a given class c, find the top-K images (and maybe the bottom-K images) in your dataset with maximal scores for this class probability.
- Can also be done with certain neurons/layers instead of the outputs to identify the images that seem to activate a given neuron/layer the most.

Restricted

Analyzing features via dataset evaluation

- Consider the images in your dataset and a trained model.
- For a given class c, find the top-K images (and maybe the bottom-K images) in your dataset with maximal scores for this class probability.
- Can also be done with certain neurons/layers instead of the outputs to identify the images that seem to activate a given neuron/layer the most.

- **Question:** if the top-K images which maximally activate the outputs of my CNN layer #10 in my ResNet are dog images, does it mean that this layer is detecting dogs?

Analyzing features via dataset evaluation

- **Question:** if the top-K images which maximally activate the outputs of my CNN layer #10 in my ResNet are dog images, does it mean that this layer is detecting dogs?

Analyzing features via dataset evaluation

- **Question:** if the top-K images which maximally activate the outputs of my CNN layer #10 in my ResNet are dog images, does it mean that this layer is detecting dogs?

- This is simple to implement, but does not show much.
- **This does not show what features your model has learned.**
- **It simply shows what your model seems to be focusing on in your given dataset.**
- Need something more sophisticated.

Activation maximization on output classes

- Consider a trained model.
- **Question:** for a given class c, e.g. dog, what would be the most “doggish” image one could give to our model?

- This is what **activation maximization** does.
- **Find the input x, which maximizes a certain output of interest (either a final probability class or the output of a layer).**

Activation maximization on output classes

- Find the input x^*, which maximizes a certain output of interest $f_c(x)$, e.g. loss for class c

$$x^* = \arg\max_x \left(f_c(x)\right)$$

- Start from image x_0 (either noise or maximal candidate in dataset).

Activation maximization on output classes

- Find the input x^*, which maximizes a certain output of interest $f_c(x)$, e.g. loss for class c

$$x^* = \arg\max_x \left(f_c(x)\right)$$

- Start from image x_0 (either noise or maximal candidate in dataset).

- Produce new candidate, like in attacks, except this time we attempt to produce the best sample for a given class c.

$$x_{n+1} = x_n - \nabla f_c(x_n)$$

- Iterate for given number of iterations or convergence.

Activation maximization on output classes

Example: what would be the most flower-ish image for a given network?

Activation maximization on layers output

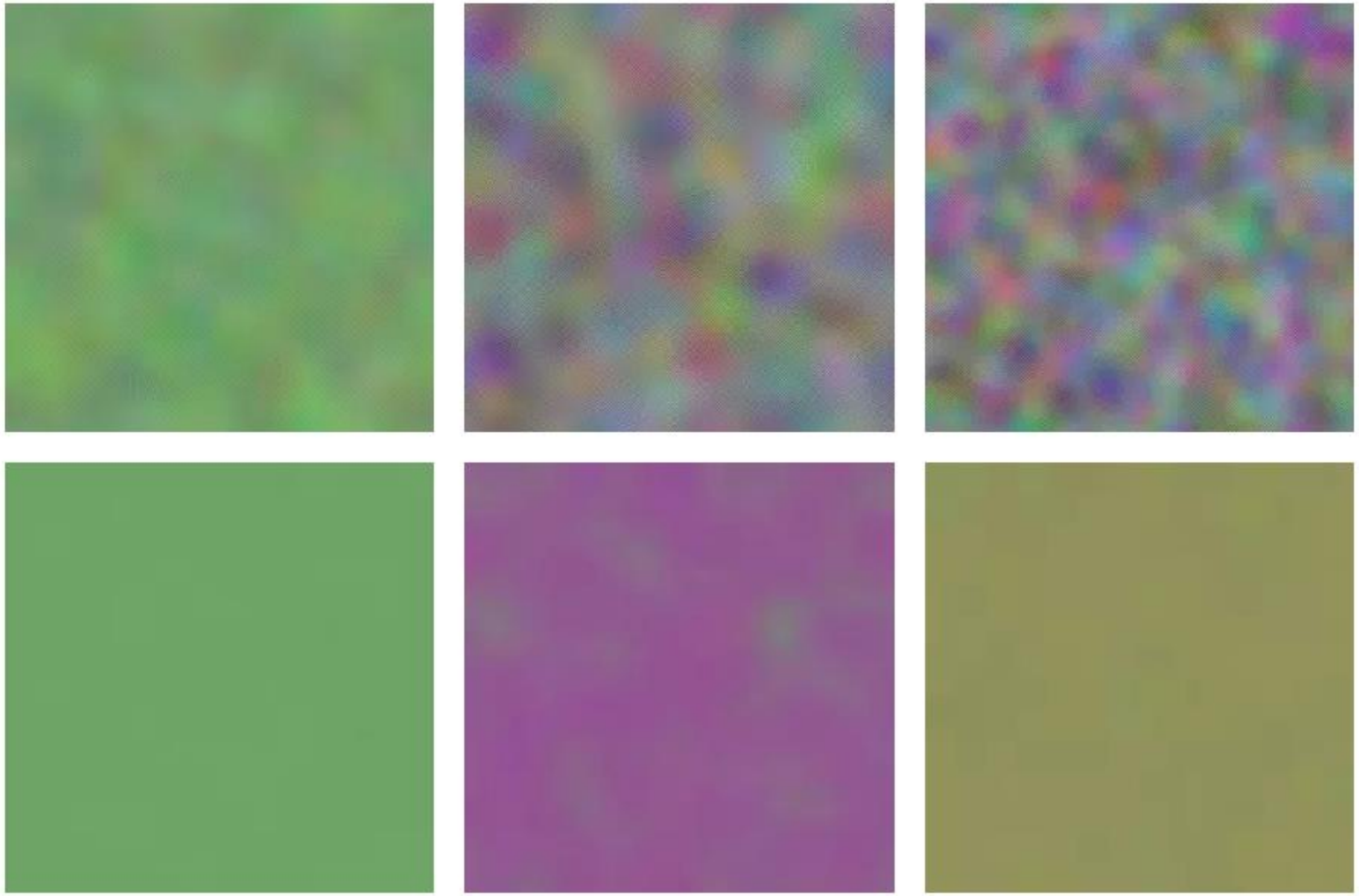
Also works with CNN layer outputs!

Example: Is layer 4b:409 in ResNet detecting dog features in images?

Activation maximization on layers output

Block1Conv1

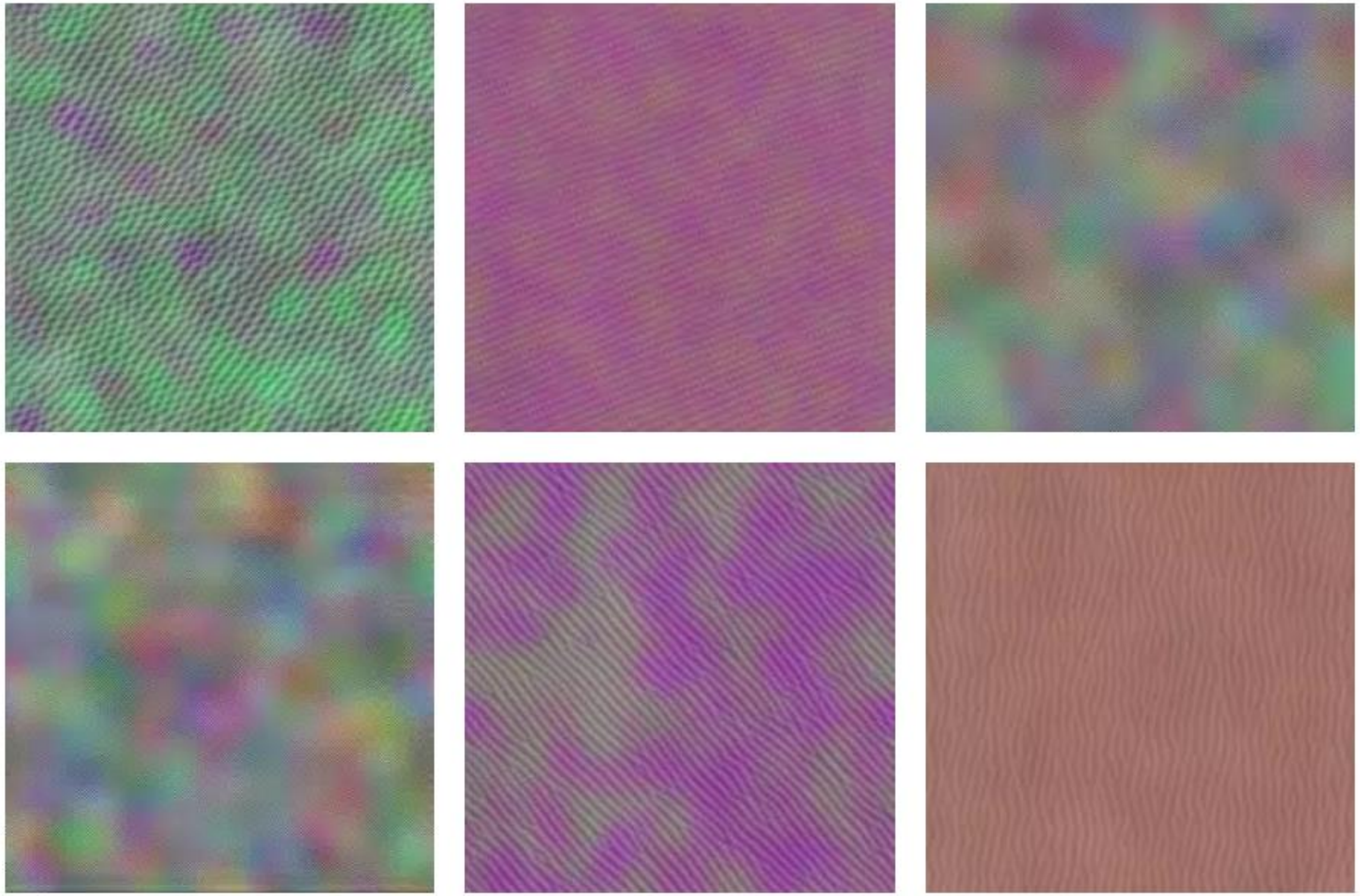
For the first Conv layer in the first Conv block, the results are not very detailed. However, filters clearly distinguish from each other, as can be seen from the results:



Activation maximization on layers output

Block2Conv1

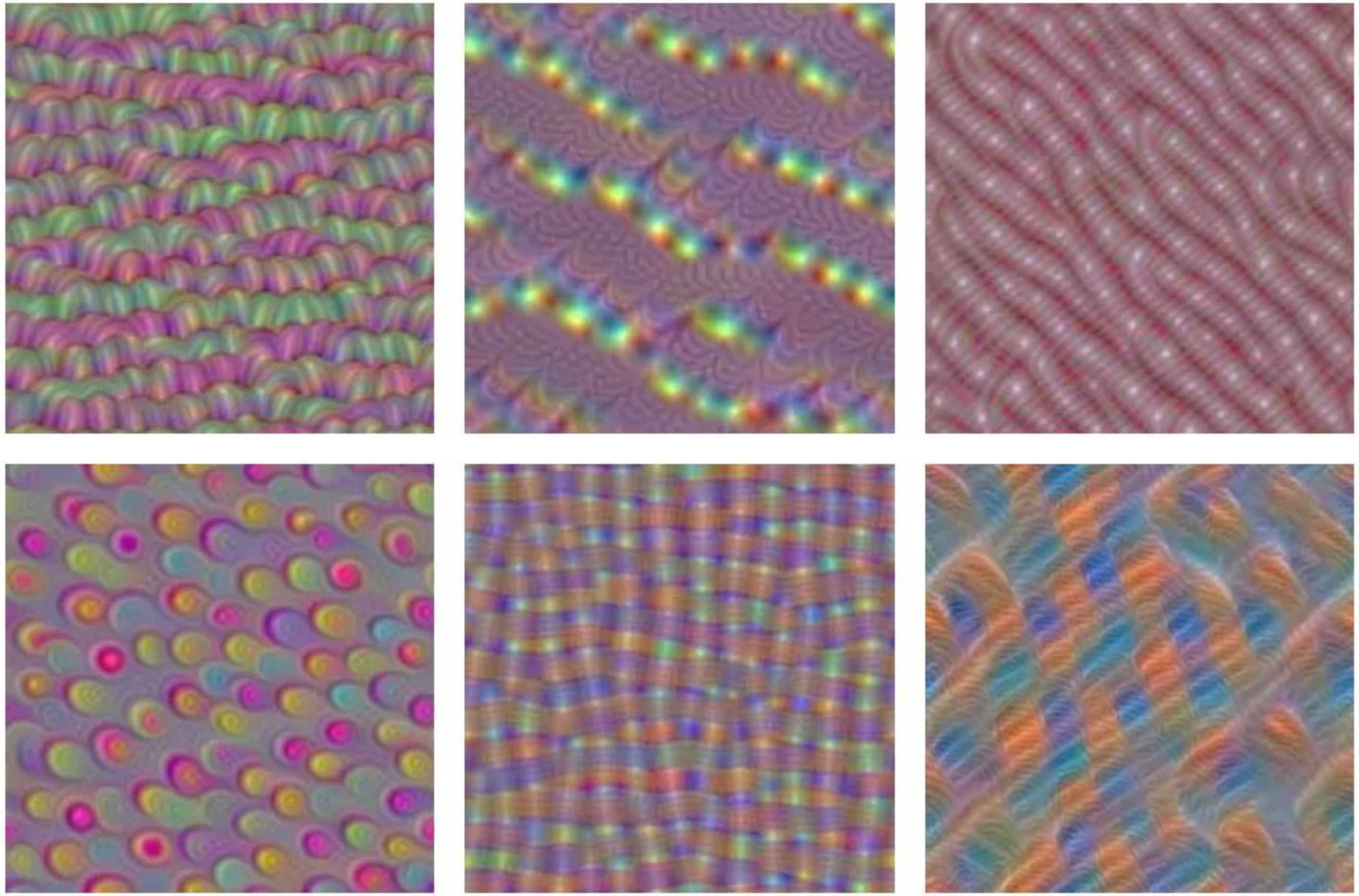
In the second block, a little bit more detail becomes visible. Certain stretched patterns seem to be learnt by the filters.



Activation maximization on layers output

Block4Conv1

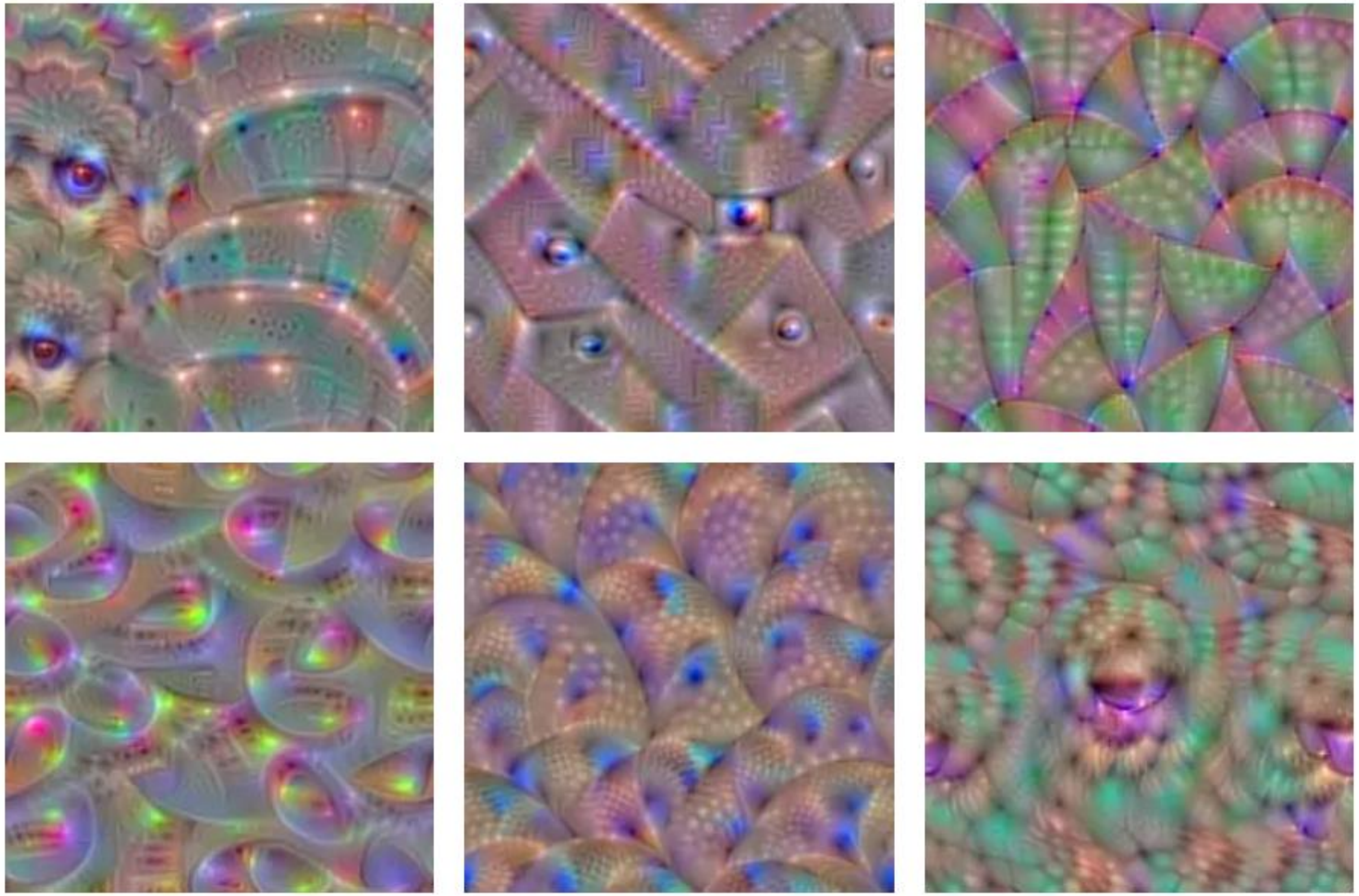
Details become visible in the fourth convolutional block. It's still difficult to identify real objects in these visualizations, though.



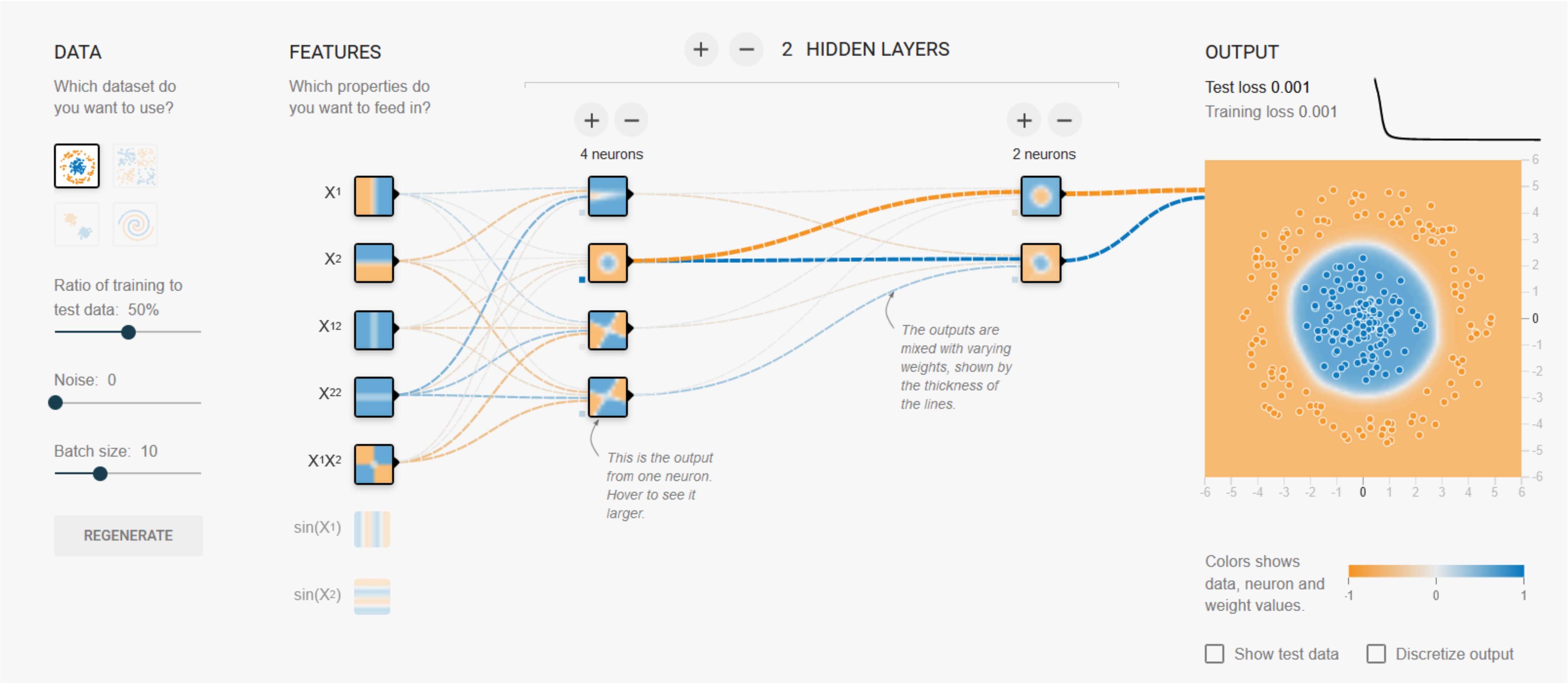
Activation maximization on layers output

Block5Conv2

This latter becomes possible in the visualizations generated from the fifth block. We see eyes and other shapes, which clearly resemble the objects that this model was trained to identify.



Activation maximization on layers output



Deep Learning Tensorflow Playground: https://playground.tensorflow.org

Activation maximization on output classes

- Consider a trained model.
- **Question:** for a given class c, e.g. dog, what would be the most "doggish" image one could give to our model?

- This is what **activation maximization** does.
- Find the input x, which maximizes a certain output of interest (either a final probability class or the output of a layer).

- **Still, a very subjective approach.**

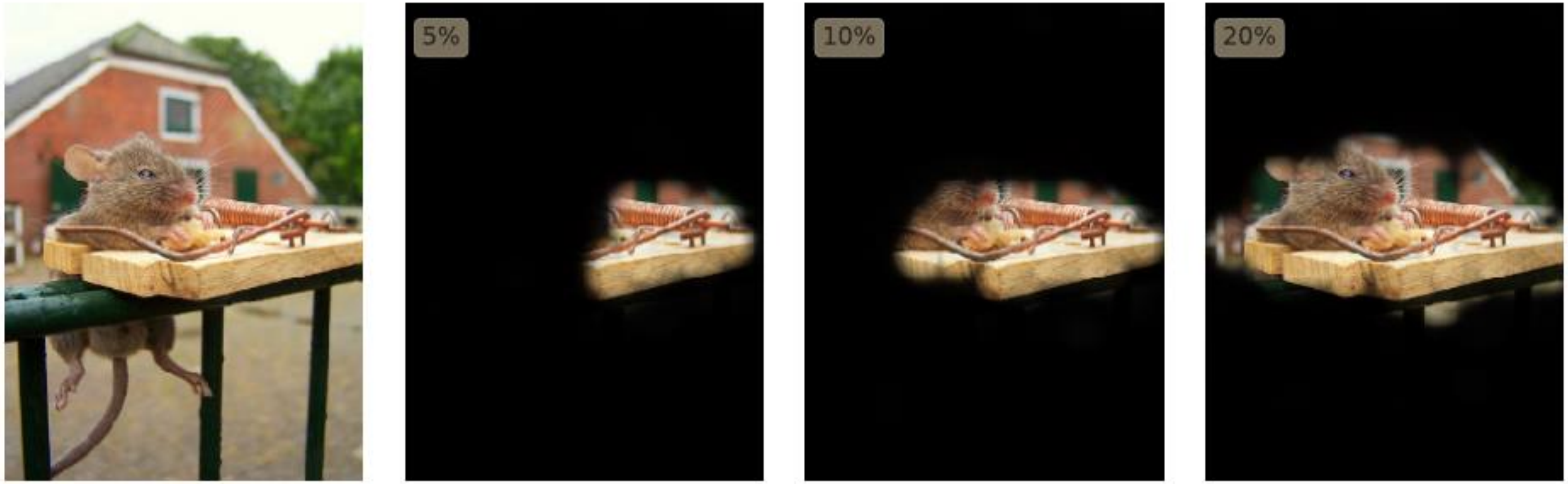
Occlusion-based methods

- Consider an image x, with D-dimensions $(x_1, \dots, x_d, \dots, x_D)$, and a given decision metric, for instance $f_c(x)$, the probability of x being classified as it ground truth class c.

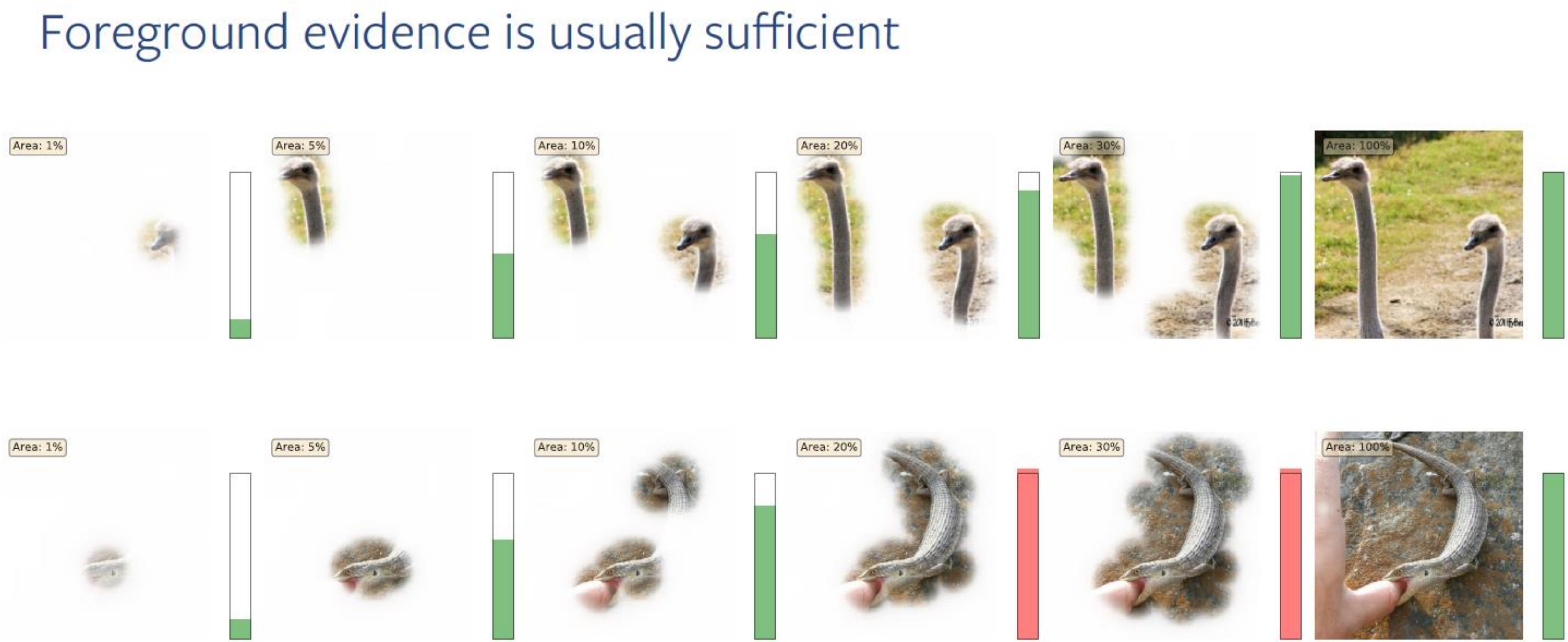
- **Following the dropout idea:** If I was to wipe out parts of the image and remove some dimensions of the input, which ones would most likely affect the decision?
- **This would indicate that the pixels x_d are important or not in classifying x as c!**

Occlusion-based methods

- Typically, one can wipe out certain pixels of x and replace them with black values.
- Then, we could attempt to find the pixels (with a given percentage p), to maximize the output probability $f_c(x)$.

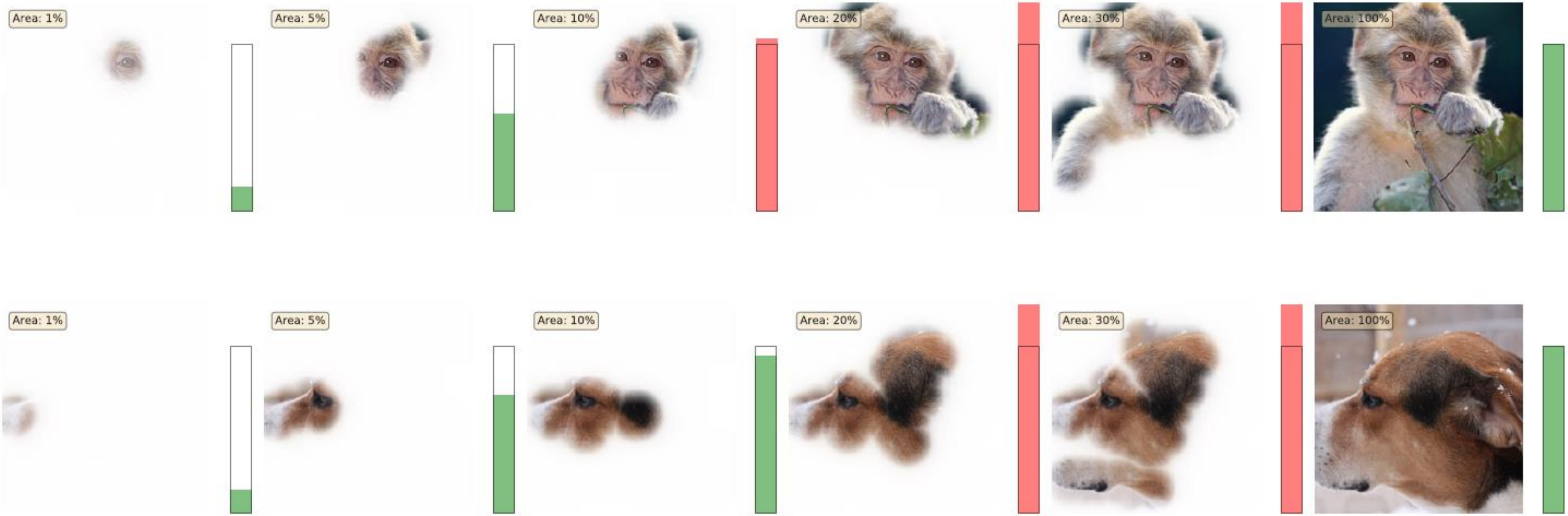


Occlusion-based methods



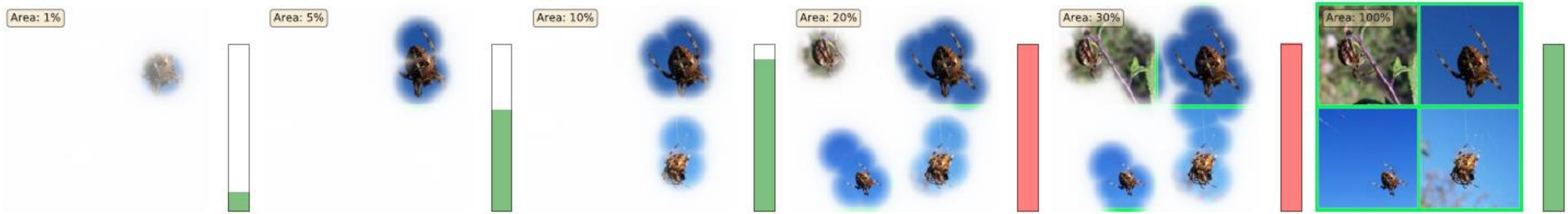
Occlusion-based methods

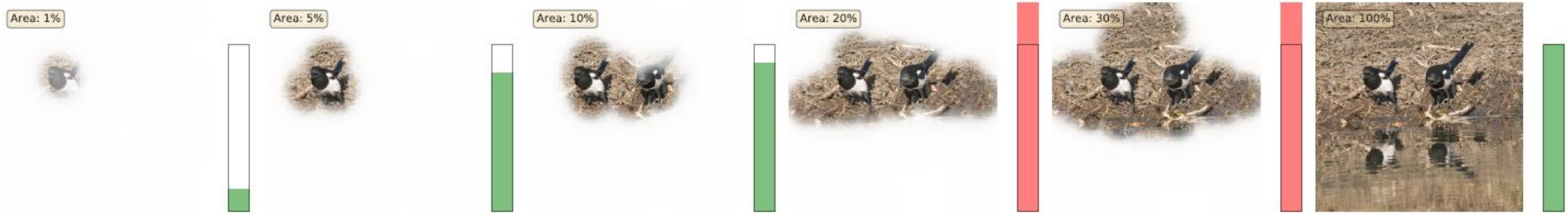
Large objects are recognized by their details



Occlusion-based methods

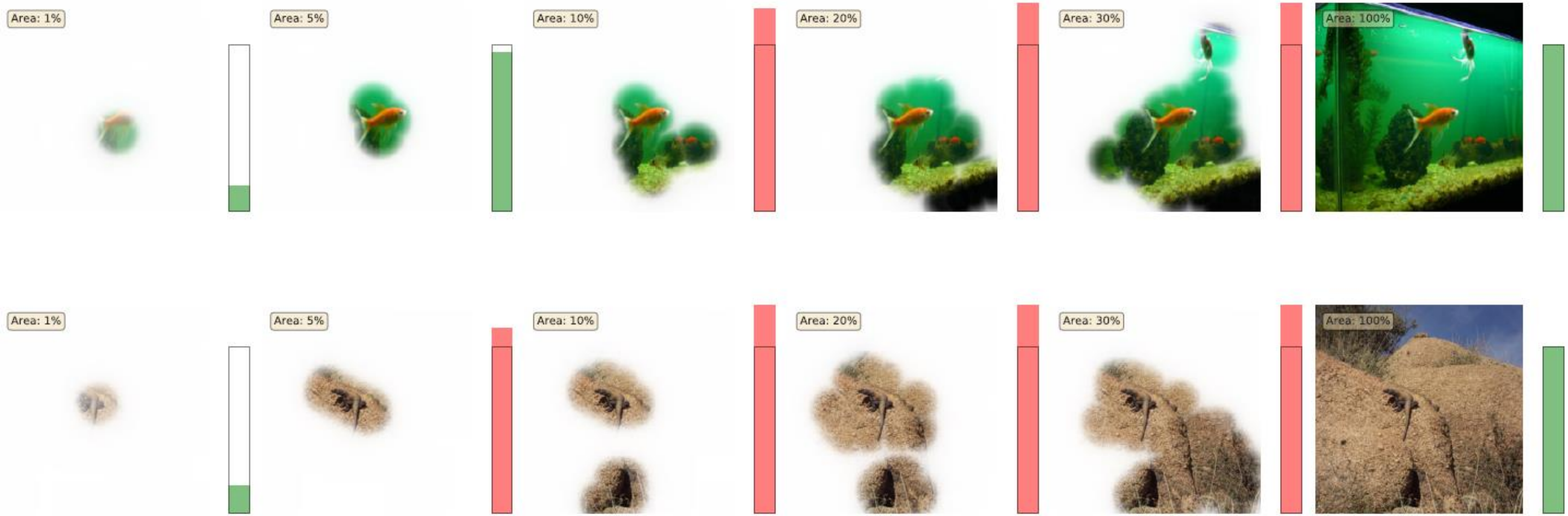
Multiple objects contribute cumulatively





Occlusion-based methods

Suppressing the background may overdrive the network



Occlusion-based methods

Original

"cat" probability
1.00

Redact-out

"cat" probability
0.5
(ineffective)

Blur-out

"cat" probability
0.01
(more meaningful)

Occlusion-based methods

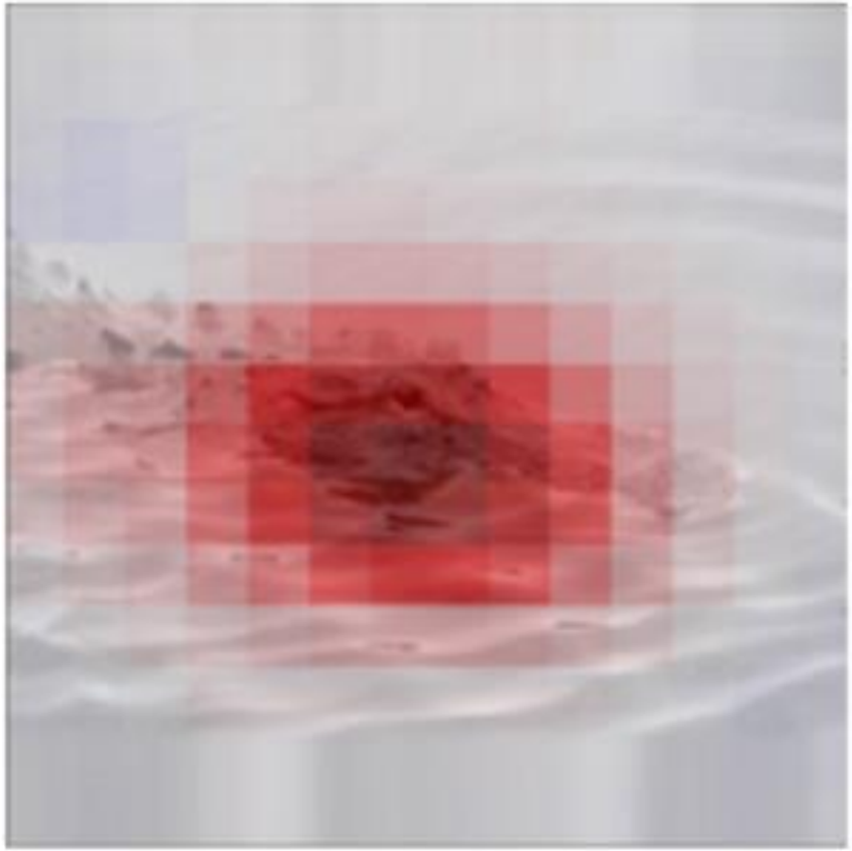
- One could also wonder what could be more meaningful perturbation to the image to make it adversarial in the most efficient manner.
- For instance, blurring seems to give more narrowed results.
- **A blurring attack (instead of randomly noising as in W8) could prove to be devastating to image recognition models?**

Gradient sensitivity

- Consider a trained model f and a dataset.
- **What we would like:** for a given input x, with D-dimensions $(x_1, \dots, x_d, \dots, x_D)$, define some importance measure $r_d(x)$ quantifying how element x_d contributes to the decision returned by a given model f.

- This would typically tell us which pixels of an image were important in deciding for the class $f(x)$!
- Which pixels in the image are making the image dog-ish?

Gradient sensitivity



Gradient sensitivity

- A first simple model would be the gradient sensitivity, defined as

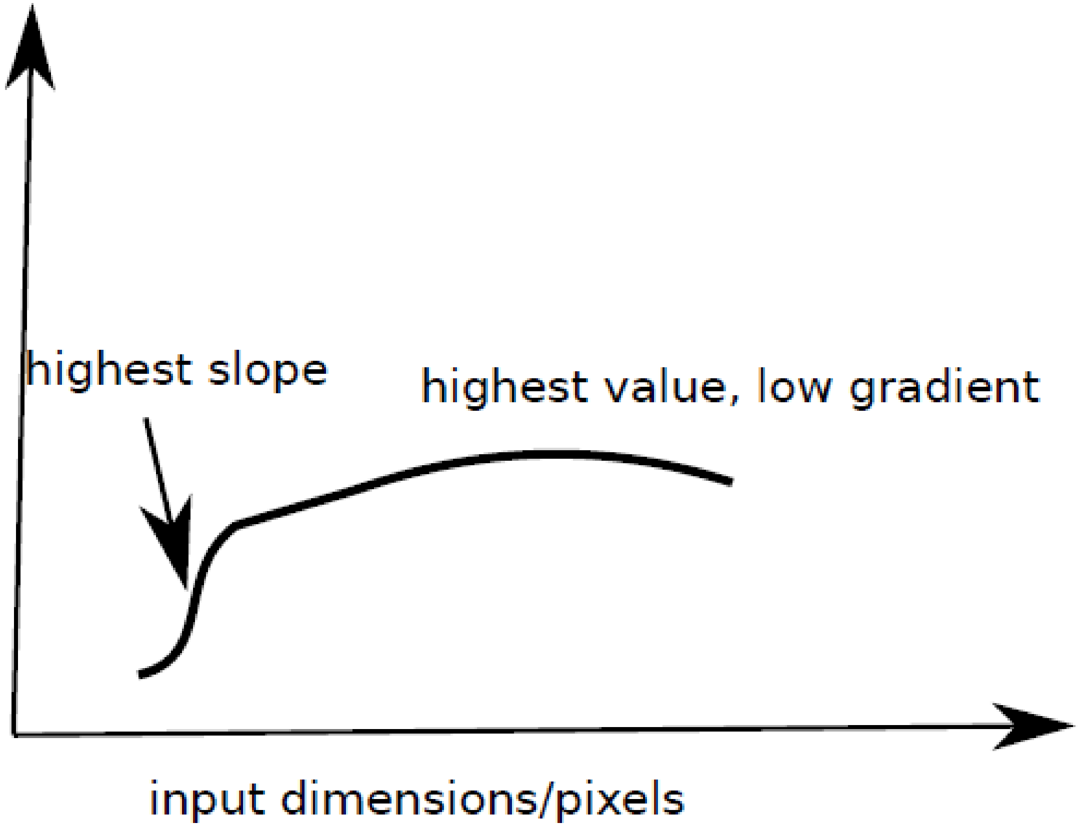
$$r_d(x) = \left(\frac{\partial f}{\partial x_d}(x)\right)^2$$

Gradient sensitivity

- A first simple model would be the gradient sensitivity, defined as

$$r_d(x) = \left(\frac{\partial f}{\partial x_d}(x)\right)^2$$

- Easily implemented, but only gives an information about the pixels, which are most sensitive to change the prediction.



Gradient sensitivity

- A first simple model would be the gradient sensitivity, defined as

$$r_d(x) = \left(\frac{\partial f}{\partial x_d}(x)\right)^2$$

- Easily implemented, but only gives an information about the pixels, which are most sensitive to change the prediction.

- The gradient does not explain **which pixels are most contributing to the prediction of a dog**.
- The gradient explains **which pixels are most sensitive to change the prediction of a dog**.

Most sensitive to change

$\neq$

Most contributing

Gradient sensitivity

- A first simple model would be the gradient sensitivity, defined as

$$r_d(x) = \left(\frac{\partial f}{\partial x_d}(x)\right)^2$$

- Easily implemented, but only gives an information about the pixels, which are most sensitive to change the prediction.

- In the case of a fully connected, with no activation functions, i.e.

$$f(x) = w.x$$

- With $x = (x_1, \dots, x_D)$ and $w = (w_1, \dots, w_D)$.
- We then have

$$r_d(x) = w_d^2$$

Gradient sensitivity

- A first simple model would be the gradient sensitivity, defined as

$$r_d(x) = \left(\frac{\partial f}{\partial x_d}(x)\right)^2$$

- Easily implemented, but only gives an information about the pixels, which are most sensitive to change the prediction.

- The gradient does not explain **which pixels are most contributing to the prediction of a dog**.
- The gradient explains **which pixels are most sensitive to change the prediction of a dog**.

Most sensitive to change

$\neq$

Most contributing

Gradient x input sensitivity

- A first simple model would be the gradient x input sensitivity, defined as

$$r_d(x) = \left(\frac{\partial f}{\partial x_d}(x)\right)^2 x_d$$

- Easily implemented.

Gradient x input sensitivity

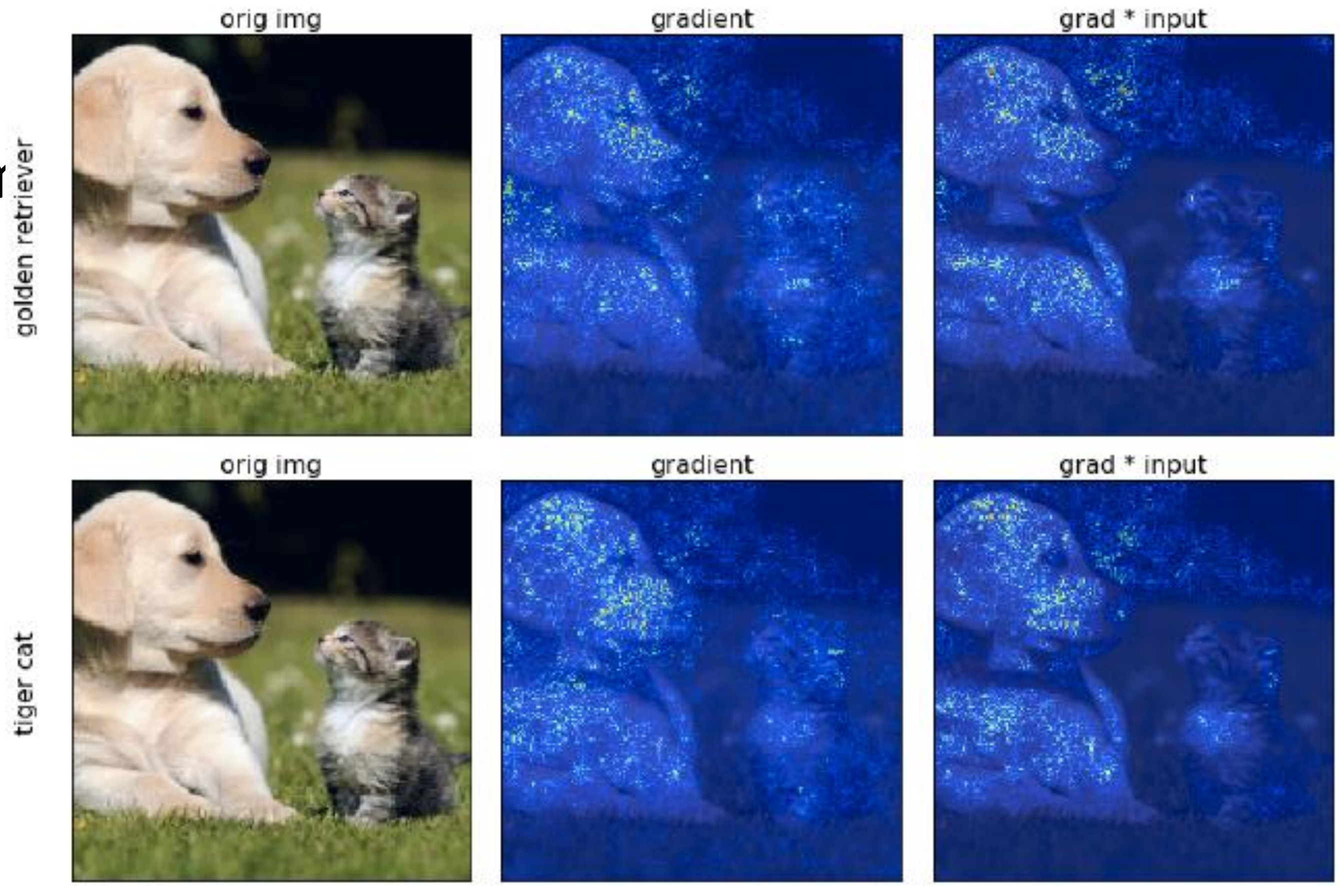
- A first simple model would be the gradient x input sensitivity, defined as

$$r_d(x) = \left(\frac{\partial f}{\partial x_d}(x)\right)^2 x_d$$

- Easily implemented.
- Fixes (partially) the previously mentioned problem.
- Works well for shallow nets and sigmoid networks.

Gradients vs Gradient x Input Sensitivity

- Overall, Gradient x Input seems to perform slightly better at identifying subpixels which seem to matter but…
- **Still very noisy**
- **Strongly affected by activation functions (ReLU?!)**
- **And only showing sensitivity, not the most contributing pixels!**
- **Need to come up with more advanced measures!**



The LIME algorithm

- We have seen in the gradients sensitivity that decisions made by a linear model is easily explainable

$$f(x) = \sum_d w_d x_d$$

- Simply given by weights w_d!

The LIME algorithm

- We have seen in the gradients sensitivity that decisions made by a linear model is easily explainable

$$f(x) = \sum_d w_d x_d$$

- Simply given by weights w_d!

- If it has a bias, there is an unexplained component thought, the bias b, which cannot be naturally assigned into contributions of single dimensions d

$$f(x) = \sum_d w_d x_d + \mathrm{b}$$

The LIME algorithm

- **Core idea:** given a test sample x learn a locally linear approximation to f around x.

$$f(x) \approx A(x) = \sum_d w'_d x_d$$

How to compute a locally linear approximation of f?

The LIME algorithm

- **Core idea:** given a test sample x learn a locally linear approximation to f around x.

$$f(x) \approx A(x) = \sum_d w'_d x_d$$

How to compute a locally linear approximation of f?

- **Step 1:** Generate samples around a given input x, denote them $(z_i)_{i \in [1,N]}$, as in attacks.
- Get their scores $f(z_i)$ as well.
- The samples can be noised versions of the original input x, with maximal noise amplitude ϵ.
- Or something even fancier (occlusion, blurring, etc.).

The LIME algorithm

- **Core idea:** given a test sample x learn a locally linear approximation to f around x.

$$f(x) \approx A(x) = \sum_d w'_d x_d$$

How to compute a locally linear approximation of f?

- **Step 2:** Find the combination of weights w'_d by training a K-Lasso on the pairs $\left(z_i, f(z_i)\right)_{i \in [1,N]}$.

$$w' = \arg\min_w [(f(z_i) - wz_i)^2 + \lambda|w|]$$

- L1 norm for regularization will encourage weights to be zero.
- Select the K-highest weights.

The LIME algorithm

- **Core idea:** given a test sample x learn a locally linear approximation to f around x.

$$f(x) \approx A(x) = \sum_d w'_d x_d$$

How to compute a locally linear approximation of f?

- **Step 3:** After the regression, use those weight only to obtain the locally linear approximation A.

$$A(x) = \sum_d w'_d x_d$$

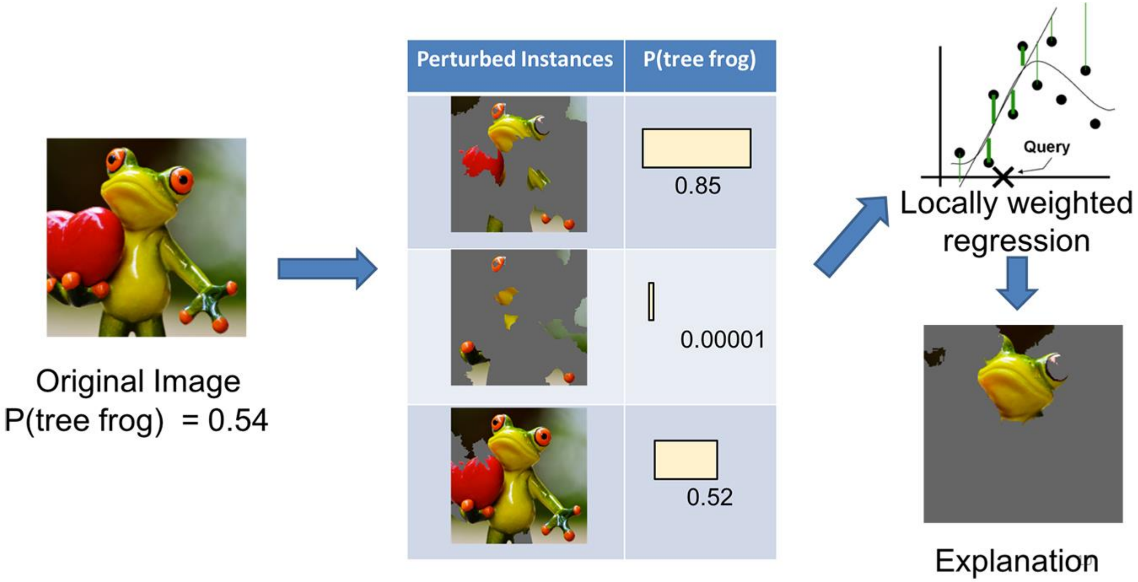
- Try it for different values of the noise amplitude ϵ, number of weights K, and regularization parameter λ.

The LIME algorithm

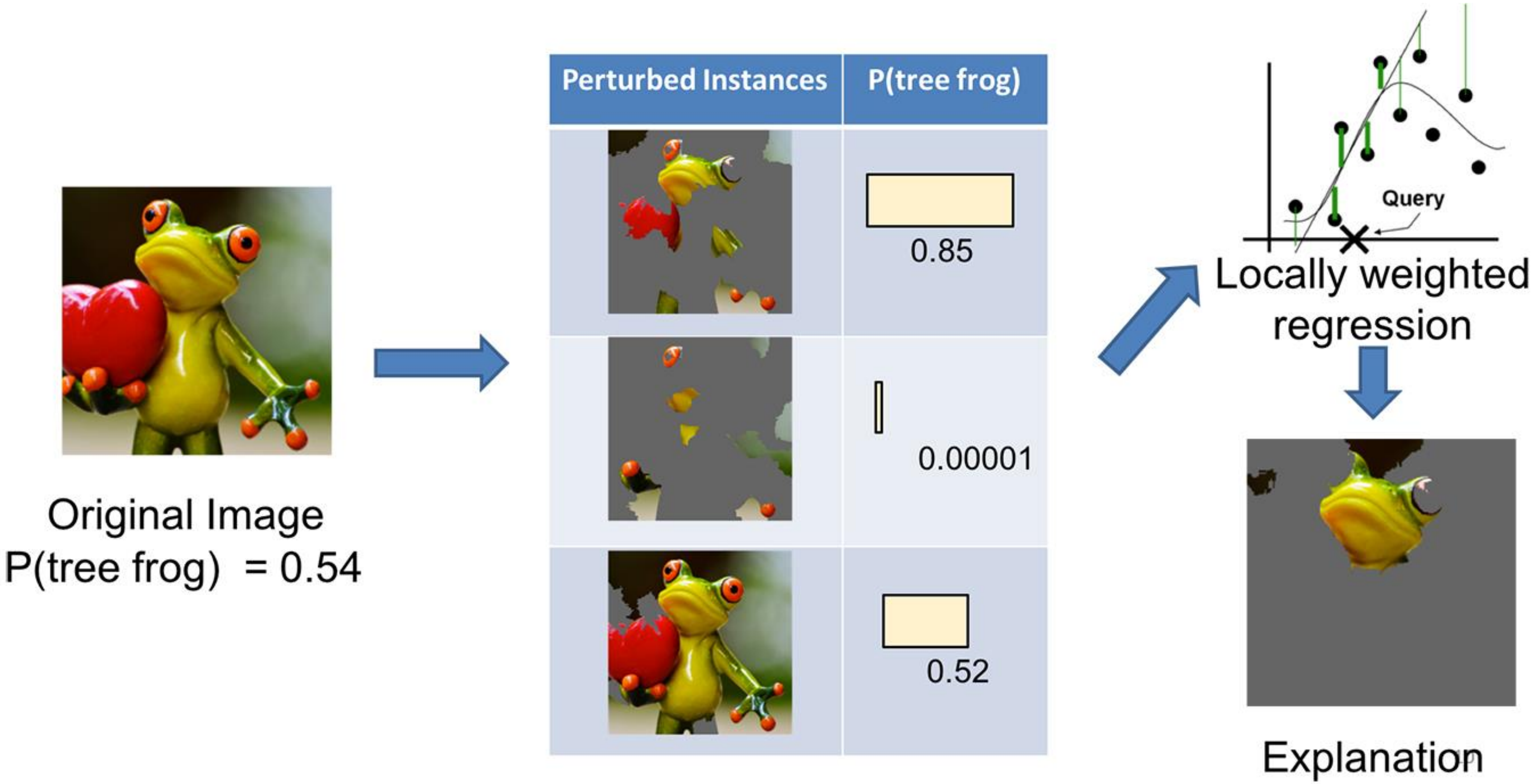
- **Step 4: The pixel importance score $r_d(x)$ is then simply defined as**

$$r_d(x) = w'_d x_d$$

- The K value indicates the K most important pixels or dimensions.
- and $r_d(x) = w'_d x_d$ gives a score. (not the most useful though)



The LIME algorithm



The LIME algorithm

- The **Local Interpretable Model-Agnostic Explanations (LIME)** algorithm is a **model-agnostic** method.
- It provides an explanation for any type of model.
- The quality of the explanation unfortunately suffers for the generality of this method.

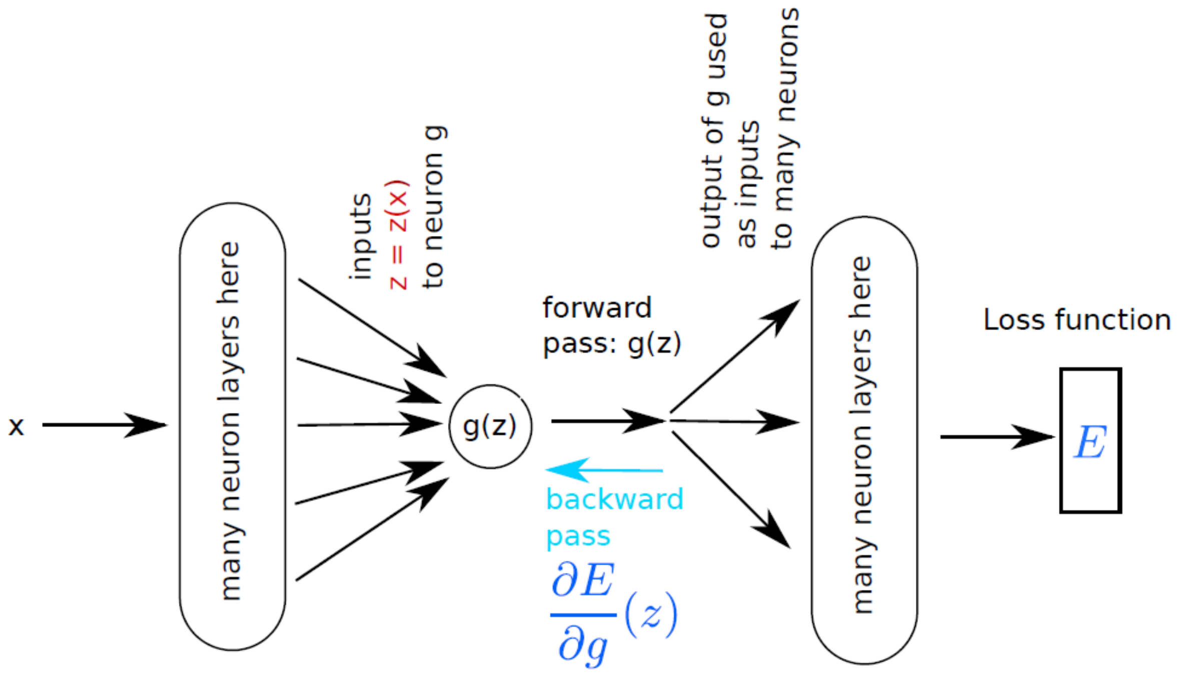
A quick note for good practices

- The radius of generated samples ϵ has a strong effect.
- A large sampling radius allows to learn correlations between neighboring data points more than just the gradient.
- For a small sampling radius LIME converges to the gradient.

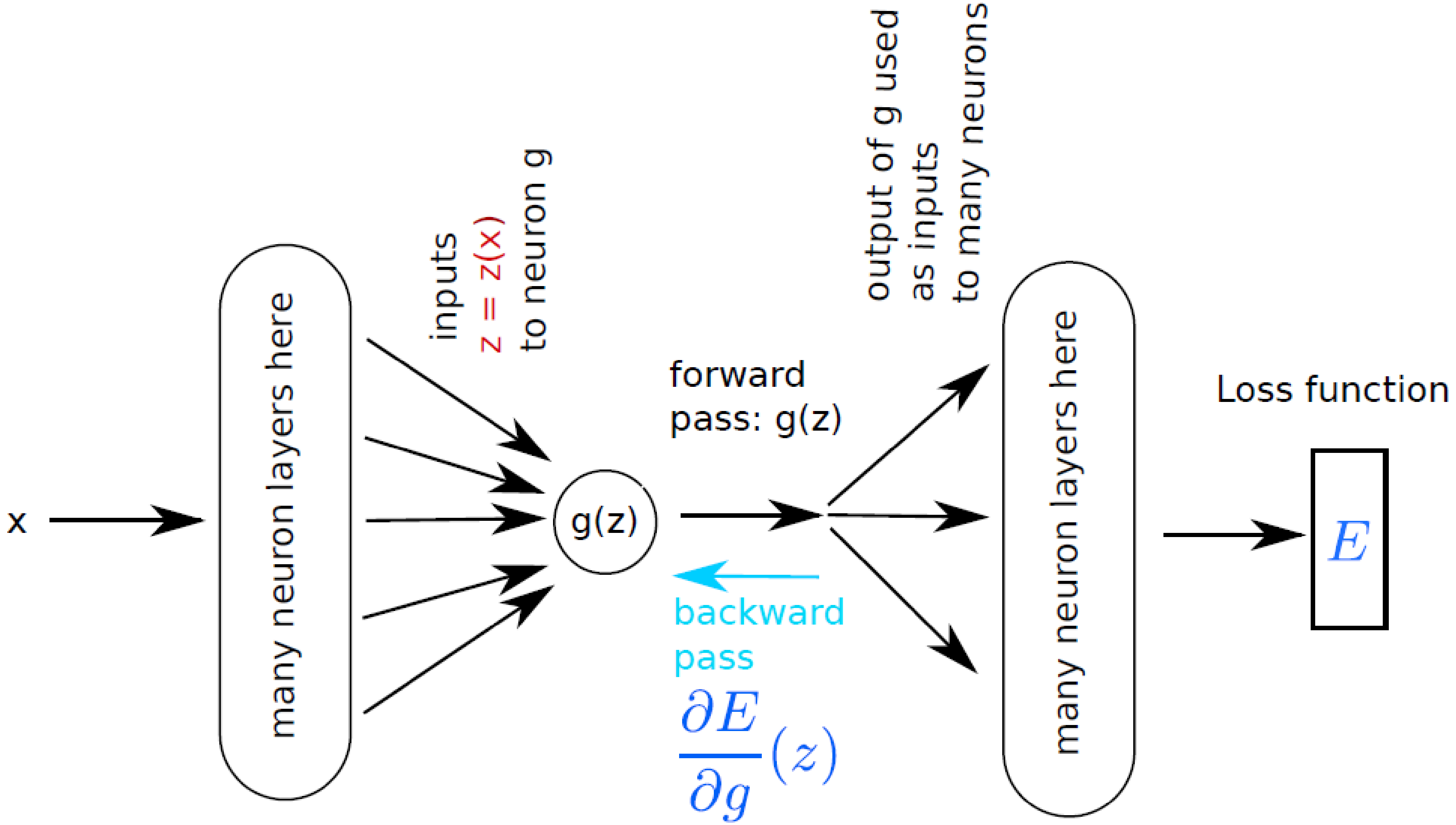
Guided backpropagation

Code idea: do a backpropagation for all fully connected layers $g(z)$, but zero out gradients if

- The activation of the neuron is negative, i.e. $g(z) \; < \; 0$
- Or if the gradient arriving at this neuron . $\frac{\partial E}{\partial g}(z)$ is negative, i.e. $\frac{\partial E}{\partial g}(z) < 0$.



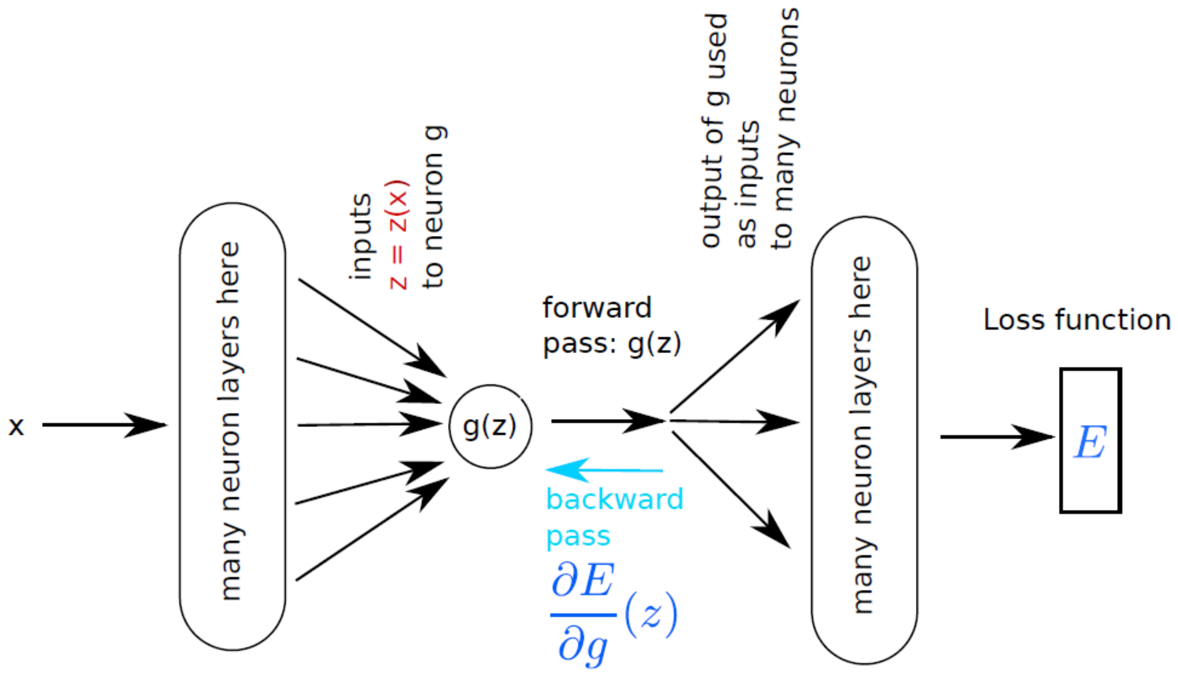
Guided backpropagation



Guided backpropagation

Code idea: do a backpropagation for all layers $g(z)$, but zero out gradients if

- The activation of the neuron is negative, i.e. $g(z)\ <\ 0$
- Or if the gradient arriving at this neuron . $\frac{\partial E}{\partial g}(z)$ is negative, i.e. $\frac{\partial E}{\partial g}(z) < 0.$



Guided backpropagation

Code idea: do a backpropagation for all layers $g(z)$, but zero out gradients if

- The activation of the neuron is negative, i.e. $g(z) \ < \ 0$
- Or if the gradient arriving at this neuron . $\frac{\partial E}{\partial g}(z)$ is negative, i.e. $\frac{\partial E}{\partial g}(z) < 0.$

Reason:

- If the activation of the neuron is negative $g(z) \ < \ 0$, then $g(z)$ is a suppressing neuron. Ignore gradients from suppressing neurons, pass through only gradient signal from firing neurons.
- Ignore gradients which decrease the function value, look only at gradients which increase the prediction
- It is a **heuristic** to look only at activating signals and gradients.

Guided backpropagation

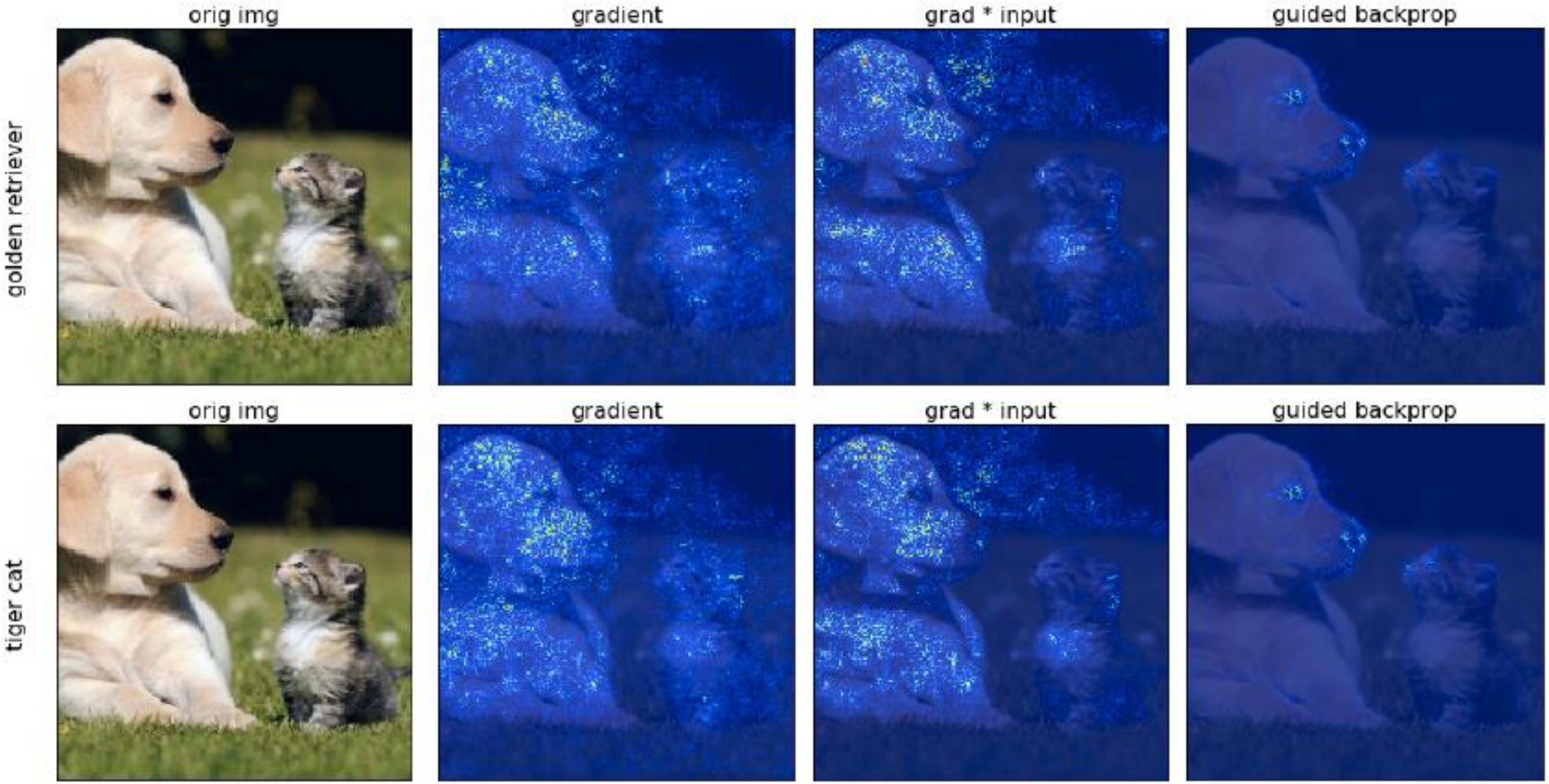
Code idea: do a backpropagation for all layers $g(z)$, but zero out gradients if

- The activation of the neuron is negative, i.e. $g(z) \ < \ 0$
- Or if the gradient arriving at this neuron . $\frac{\partial E}{\partial g}(z)$ is negative, i.e. $\frac{\partial E}{\partial g}(z) < 0$.

Takeaways:

- **It is a heuristic, no mathematical or theoretical underpinning.**
- **<u>We are not really sure what it does.</u>**
- Looking only at one sign of signals and gradients often gives cleaner heatmaps as a result!

Guided backpropagation



The promised land, a.k.a. decomposition?

- **Most approaches are trying to avoid, as much as possible to open the black-boxes that Neural Networks are.**
- **Most approaches rely on linear approximations, final layers activations, gradients information, or simply toy with the model.**
- **But none really try to explain it properly.**

The promised land, a.k.a. decomposition?

- **Most approaches are trying to avoid, as much as possible to open the black-boxes that Neural Networks are.**
- **Most approaches rely on linear approximations, final layers activations, gradients information, or simply toy with the model.**
- **But none really try to explain it properly.**

- The most promising direction (IMO) for interpretability is methodology called Layer-wise Relevance Propagation (LRP).
- It is a general approach to explain predictions of AI, which relies on the mathematical **Deep Taylor Decomposition** of the neural network.

The LRP method (out-of-class)

- **Core idea:** Layer-wise Relevance Propagation (LRP) attempts to calculate the relevance score $r_d(x)$ as a decomposition of a prediction, with some additional constraints.

$$f(x) = \sum_d r_d(x)$$

The LRP method (out-of-class)

- Given a neuron

$$y = g\left(\sum_d w_d x_d\right)$$

LRP distributes a quantity, relevance, top-down = from a neuron output y, onto the inputs x_d of that neuron.

- It then repeats the operation on each neuron of each layer.
- LRP is somewhat related to the Taylor decomposition of the propagation rule for every single neuron.
- LRP results in different rules which can be applied to different types of neural network layers.

The LRP method (out-of-class)

- For instance, in the case of a FC layer, use the ϵ-rule to propagate the relevance coefficients R.

$$R_j = \sum_k \frac{a_j \rho(w_{jk})}{\epsilon + \sum_{0,j} a_j \rho(w_{jk})} R_k$$

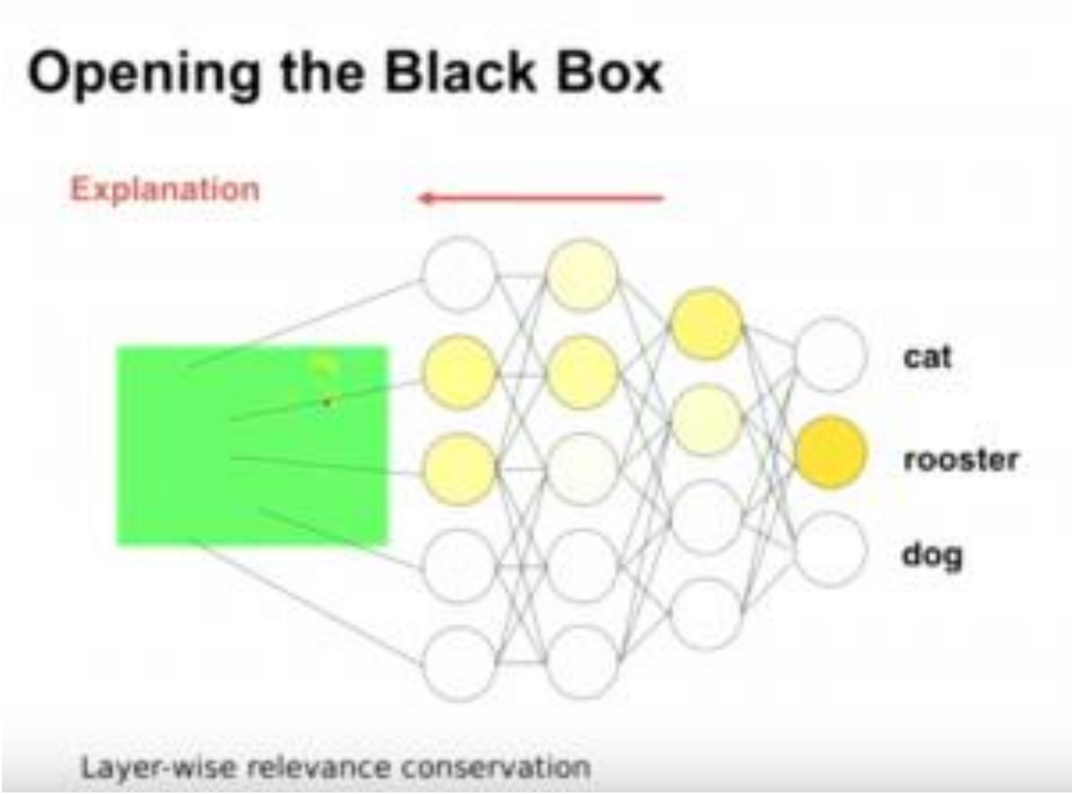
- For Convolution layers, use the β-rule.

$$R_i = \sum_j \frac{a_i w_{ij} - l_i w_{ij}^{+} - h_i w_{ij}^{-}}{\sum_i a_i w_{ij} - l_i w_{ij}^{+} - h_i w_{ij}^{-}} R_j$$

- Can even get the best of both worlds by combining both the ϵ-rule and the β-rule to get the γ-rule.

The LRP method (out-of-class)

- **Mathematical meaning behind these formulas? Unclear, but definitely empirical approximations, heuristics.**
- **Important message:** People are trying some formulas to backpropagate the relevance score from the final layers to the first one/inputs with the highest fidelity as possible.

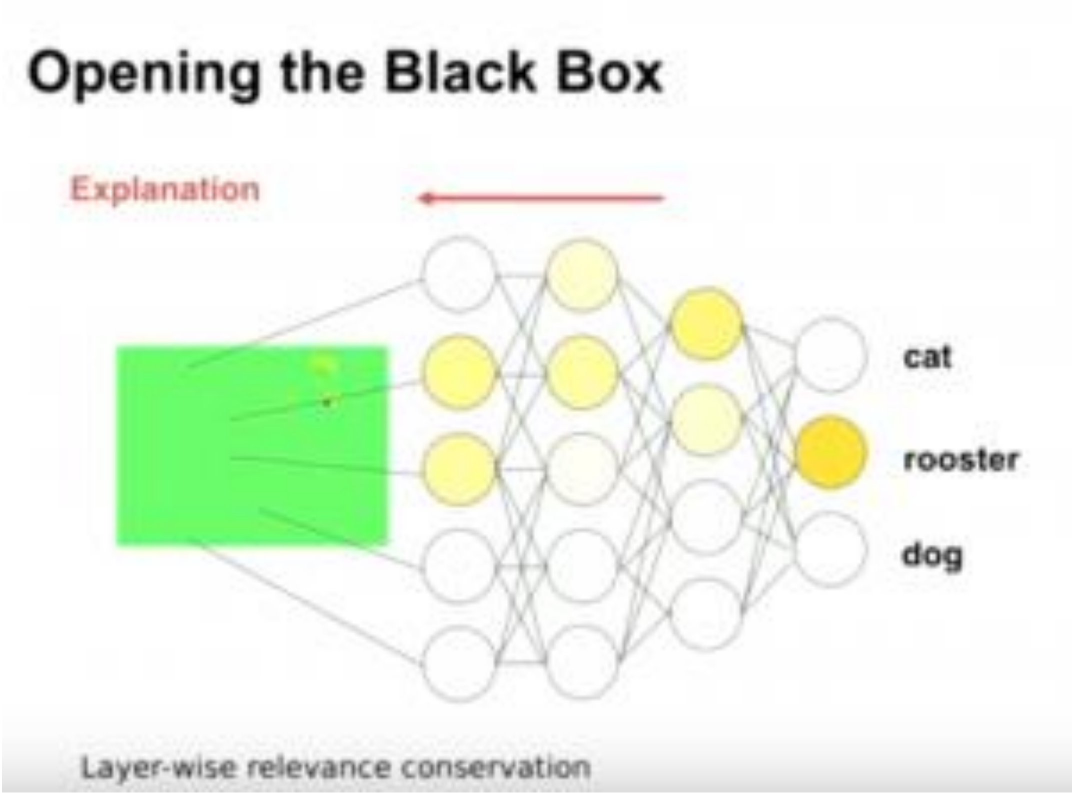


A good PyTorch tutorial for LRP. https://git.tu-berlin.de/gmontavon/lrp-tutorial

The LRP method (out-of-class)

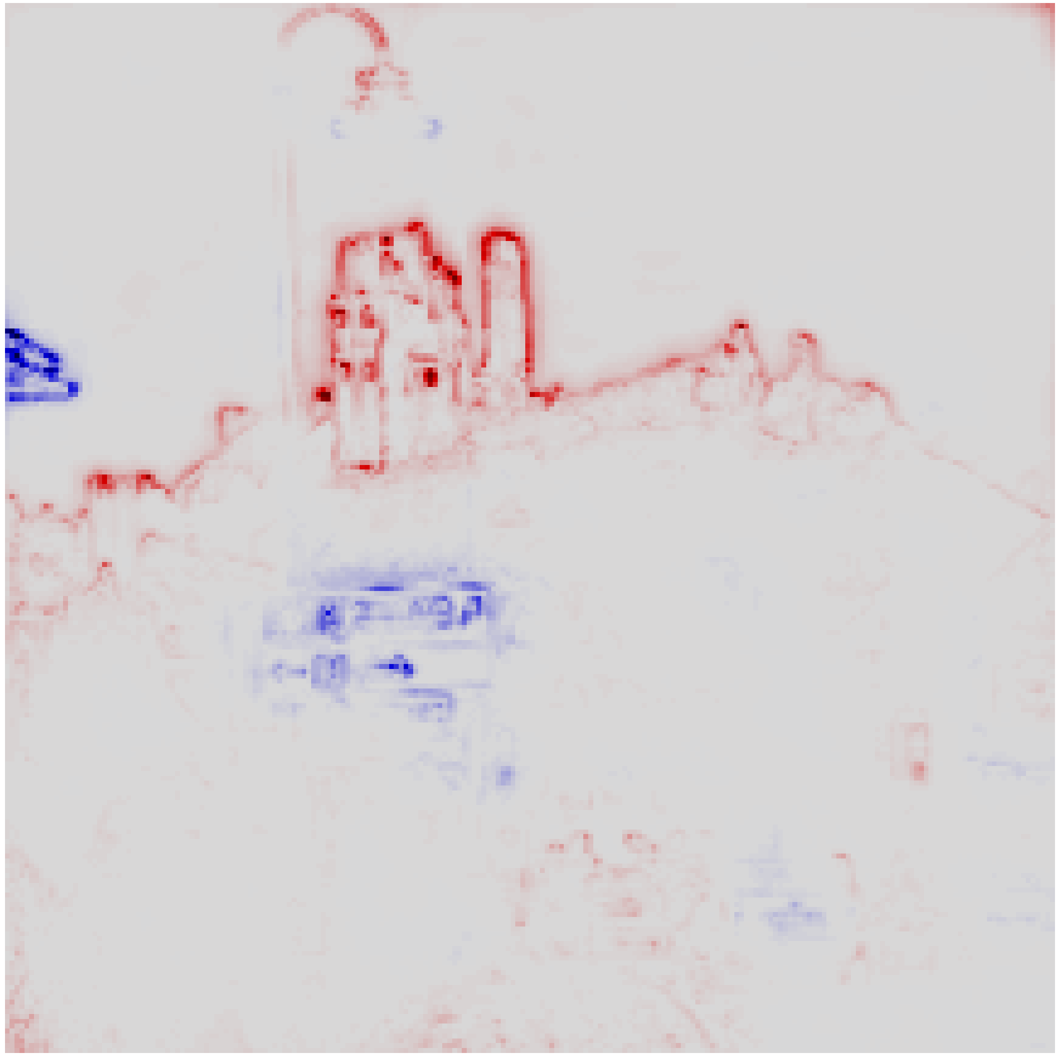
Important message #2: Formulas do not really have a mathematical foundation, but we understand that, intuitively, it should rely on both

- the gradients (account for sensitivity of inputs to decision),
- and the weights (account for contribution of inputs to decision).



A good PyTorch tutorial for LRP.
https://git.tu-berlin.de/gmontavon/lrp-tutorial

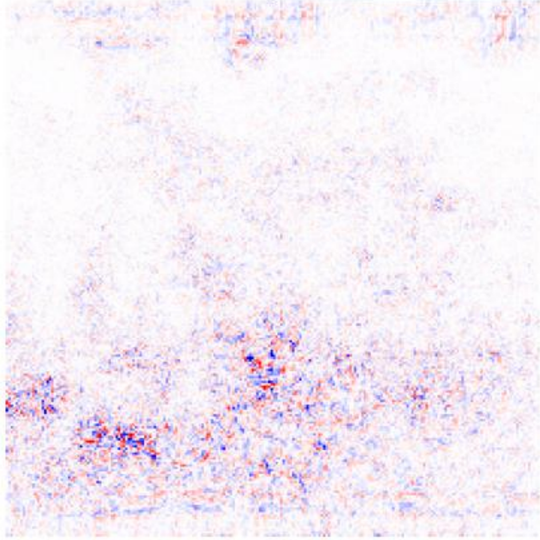
The LRP method (out-of-class)

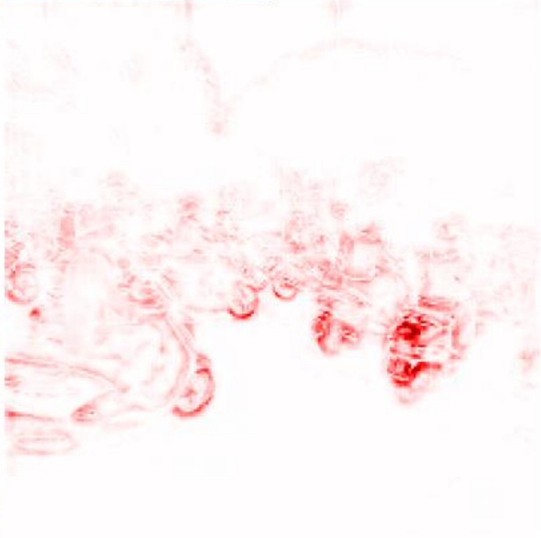


Restricted

The LRP method (out-of-class)

Gradients





LRP

Conclusion

In this lecture

- Concept of interpretability
- Why is interpretability needed
- t-SNE method
- Analyzing features with activation maximization
- Occlusion methods
- Gradient and Gradient x Input sensitivity methods
- LIME
- Guided Backpropagation
- LRP

Learn more about these topics

Out of class, for those of you who are curious

- About t-SNE
  http://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf
  And https://lvdmaaten.github.io/publications/papers/JMLR_2008.pdf
- About gradient sensitivity
  https://papers.nips.cc/paper/5422-on-the-number-of-linear-regions-of-deep-neural-networks.pdf
  And http://proceedings.mlr.press/v70/balduzzi17b/balduzzi17b.pdf.
- Playing with occlusion maps
  https://github.com/akshaychawla/Occlusion-experiments-for-image-segmentation/blob/master/Occlusion%20experiments%20for%20segmentation.ipynb

Learn more about these topics

Out of class, for those of you who are curious

- [LIME] Ribeiro et al., "Why Should I Trust You?: Explaining the Predictions of Any Classifier", 2016. https://arxiv.org/abs/1602.04938
- [GuidedBP] Springenberg et al., "Striving for Simplicity: The All Convolutional Net", 2014. https://arxiv.org/abs/1412.6806
- [CAM] Oquab et al., "Is Object Localization for Free? - Weakly-Supervised Learning With Convolutional Neural Networks", 2016. http://openaccess.thecvf.com/content_cvpr_2015/papers/Oquab_Is_Object_Localization_2015_CVPR_paper.pdf

Learn more about these topics

Out of class, for those of you who are curious

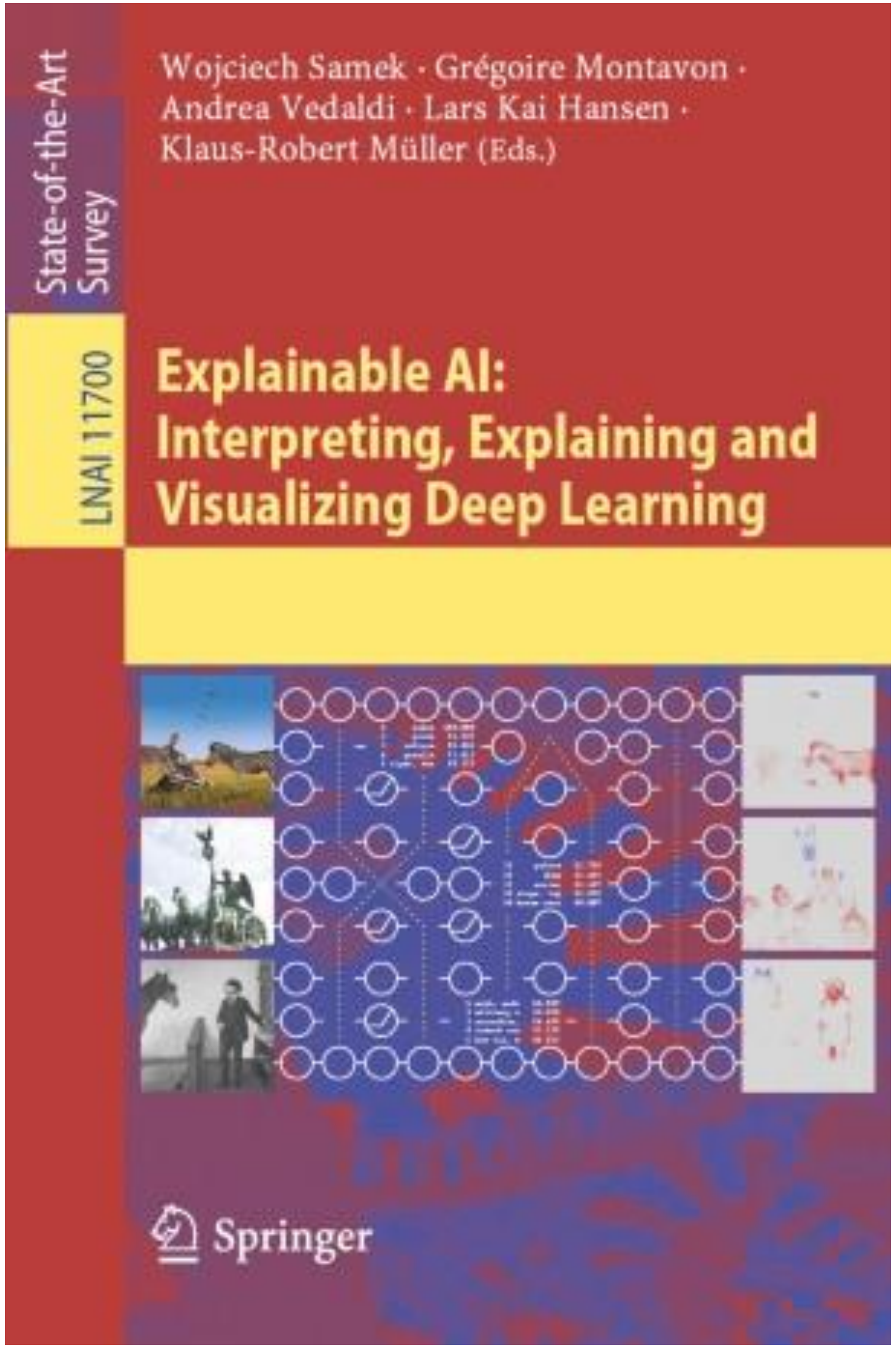
- [Grad-CAM] Selvaraju et al., “Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization”, 2017. https://arxiv.org/abs/1610.02391
- [Grad-CAM++] Chattopadhyay et al., “Grad-CAM++: Improved Visual Explanations for Deep Convolutional Networks”, 2018. https://arxiv.org/abs/1710.11063
- [LRP] Bach et al. “On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation”, 2015. https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0130140 https://git.tu-berlin.de/gmontavon/lrp-tutorial

Learn more about these topics

The Bible for interpretability? (as of Early 2021)

Montavon et al., "Layer-Wise Relevance Propagation: An Overview", 2019.

https://link.springer.com/chapter/10.1007/978-3-030-28954-6_10

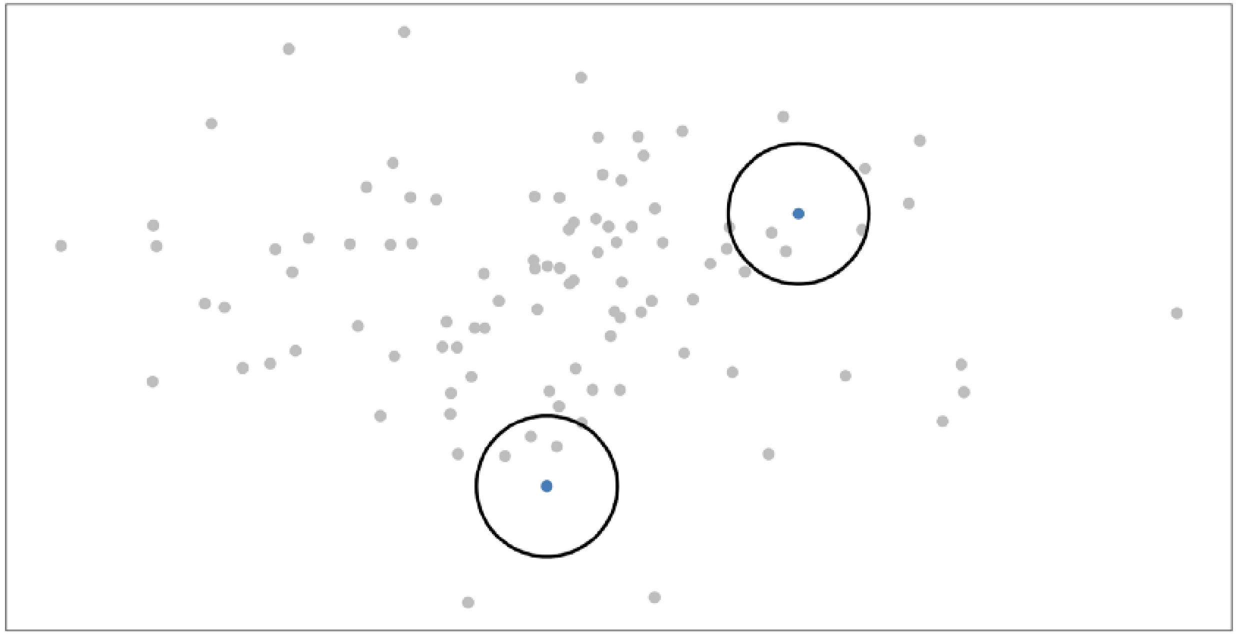


t-SNE implementation

Step 1: Gaussian distribution between samples in high-dimension space.

- Compute the **probability** that i would vote for j as being his neighbor based on a gaussian model which is centered on x_i.

$$p_{j|i} = \frac{\exp\left(\frac{-\|x_i - x_j\|^2}{2\sigma^2}\right)}{\sum_{k\neq \mathrm{i}} \exp\left(\frac{-\|x_i - x_k\|^2}{2\sigma^2}\right)}$$



t-SNE implementation

Step 1: Gaussian distribution between samples in high-dimension space.

- Compute the **probability** that i would vote for j as being his neighbor based on a gaussian model which is centered on x_i.

$$p_{j|i} = \frac{\exp\left(\frac{-\left\|x_i - x_j\right\|^2}{2\sigma^2}\right)}{\sum_{k\neq \mathrm{i}} \exp\left(\frac{-\|x_i - x_k\|^2}{2\sigma^2}\right)}$$

- The σ parameter is used to define **perplexity**. It allows to define how many neighbors should be taken considered for x_i. Normal range for perplexity is between 5 and 50.
- Symmetrize by defining

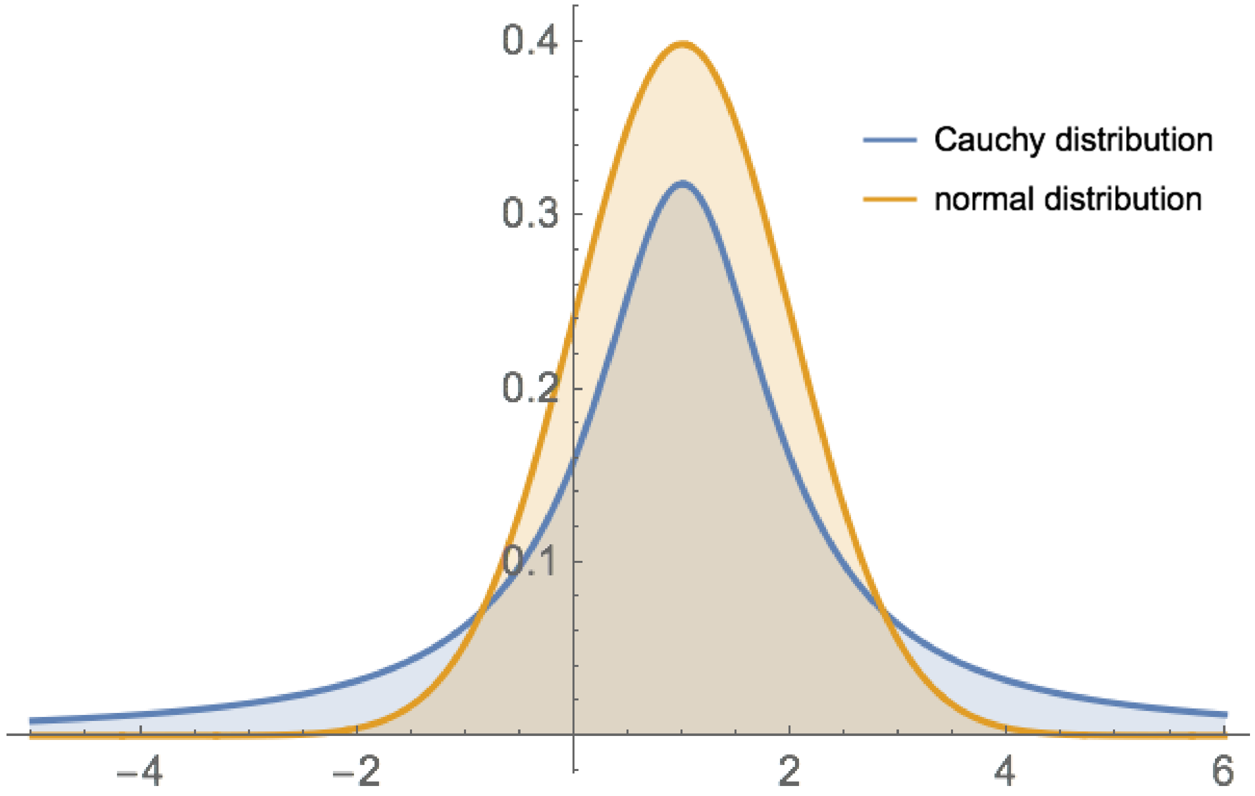
$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}$$

$$and\ p_{ii} = 0$$

t-SNE implementation

Step 2: Cauchy distribution between representations in low-dimension space.

- Learn a similar but **heavy-tailed** distribution model of y_i voting for neighbor y_j as heavy-tailed **Cauchy** distribution.



t-SNE implementation

Definition (Cauchy distribution):

The **Cauchy distribution** is often used in statistics as the canonical example of a "pathological" distribution, because both its expected value and its variance are undefined.

While similar in shape to the Normal distribution, it is also **heavy-tailed**.

Its PDF is given by

$$f(x, x_0, \gamma) = \frac{1}{\pi\gamma\left(1 + \left(\frac{x - x_0}{\gamma}\right)^2\right)}$$

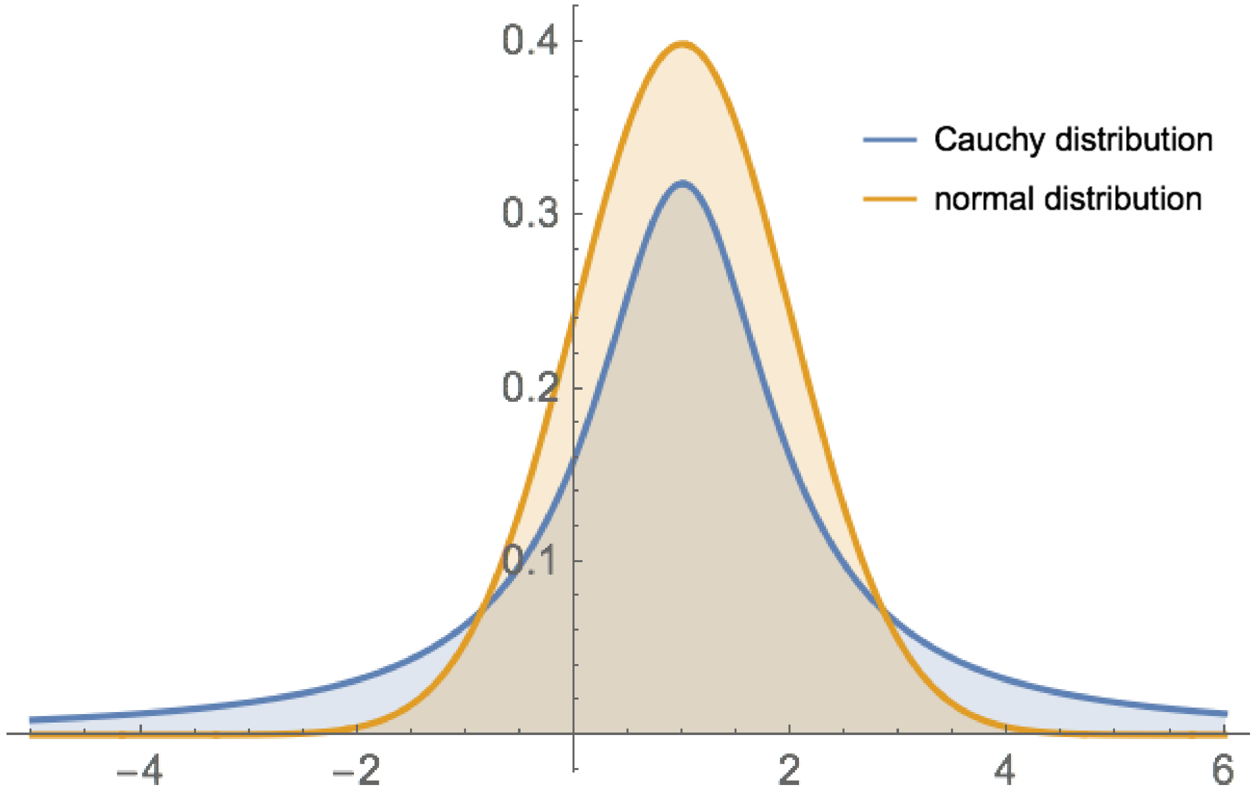
$$f(x, x_0, \gamma) = \frac{1}{\pi\gamma}\left(1 + \left(\frac{x - x_0}{\gamma}\right)^2\right)^{-1}$$

t-SNE implementation

Definition (heavy-tailed distribution):

In probability theory, **heavy-tailed** distributions are probability distributions whose tails are not exponentially bounded.

They have heavier tails than the normal/exponential distribution, and therefore give higher probabilities to elements far from the center/mean.



t-SNE implementation

Step 2: Cauchy distribution between representations in low-dimension space.

- Learn a similar but heavy tailed distribution model of y_i voting for neighbor y_j as **heavy-tailed Cauchy** distribution.

$$q_{ij} = \frac{\left(1 - \|y_i - y_j\|^2\right)^{-1}}{\sum_{\substack{k,l \\ k \neq l}} (1 - \|y_k - y_l\|^2)^{-1}}$$

t-SNE implementation

Step 2: Cauchy distribution between representations in low-dimension space.

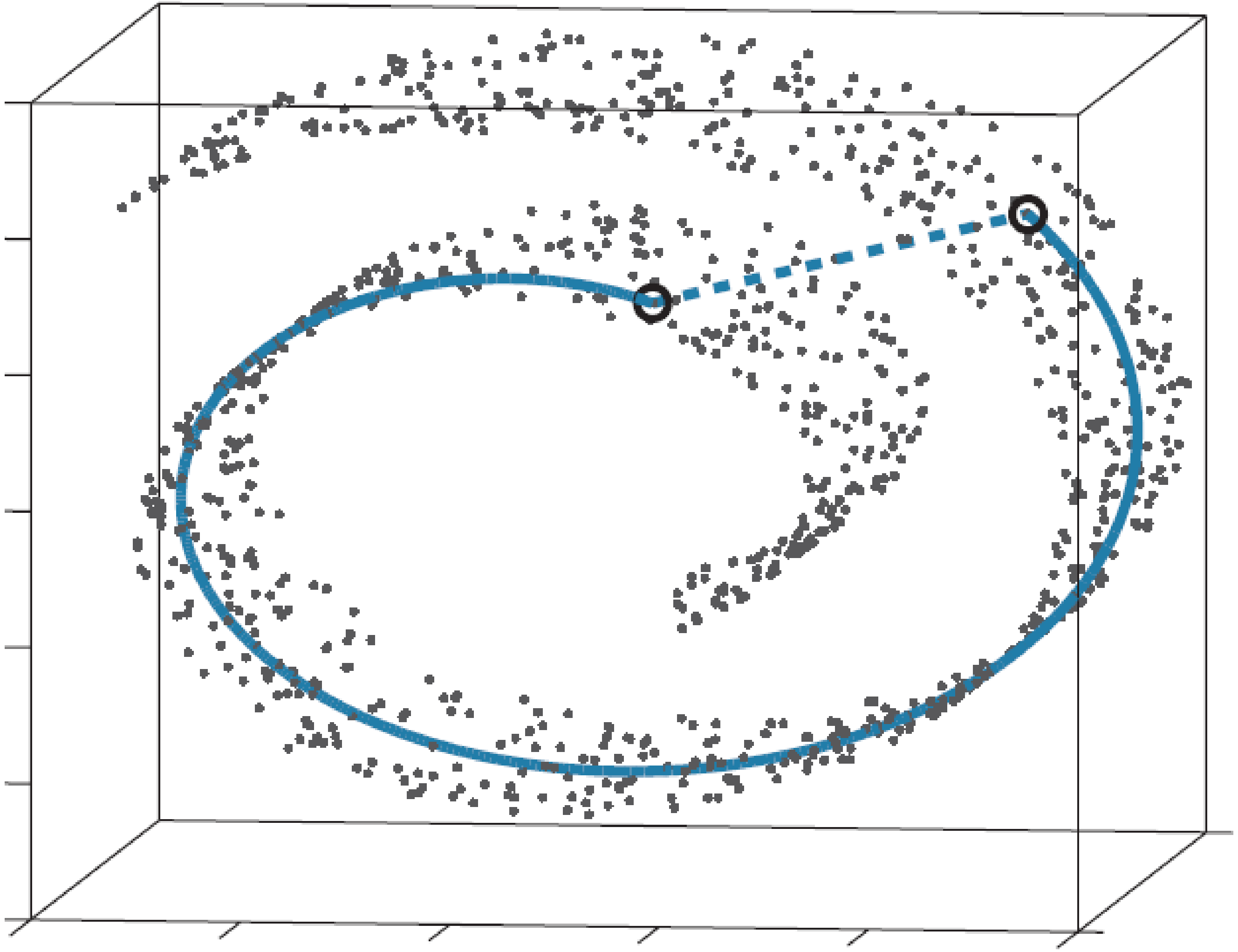
- Learn a similar but heavy tailed distribution model of y_i voting for neighbor y_j as **heavy-tailed Cauchy** distribution.

$$q_{ij} = \frac{\left(1 - \|y_i - y_j\|^2\right)^{-1}}{\sum_{\substack{k,l \\ k \neq l}} (1 - \|y_k - y_l\|^2)^{-1}}$$

Why for the y_i use a heavy-tailed probability distribution instead of a Normal one?

- This allows a moderate distance in the high-dimensional space to be faithfully modeled by a much larger distance in the map.
- As a result, it eliminates the unwanted attractive forces between map points that represent moderately dissimilar datapoints."

t-SNE implementation



t-SNE implementation

Step 3: Minimize the KL divergence between both distributions.

- In order to optimize for y_i, minimize the KL divergence.

$$q_{ij}, i, j = \underset{q_{ij}, i, j}{\arg\min} [KL(p||q)]$$

With KL as seen in lecture W11S1.

t-SNE implementation

Step 3: Minimize the KL divergence between both distributions.

- In order to optimize for y_i, minimize the KL divergence.

$$q_{ij}, i, j = \arg\min_{q_{ij}, i, j} [KL(p||q)]$$

With KL as seen in lecture W11S1.

- We then minimize for y_i by computing the gradient of $KL(p||q)$ with respect to y_i.

- **Note:** Here q depends on y_i.

- This is automatically done with libraries such as sklearn. (For instance, see https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html)

Restricted

Gradient x input sensitivity

- A first simple model would be the gradient x input sensitivity, defined as

$$r_d(x) = \left(\frac{\partial f}{\partial x_d}(x)\right)^2 x_d$$

- Easily implemented.

- **Motivation as heuristic:** close to Taylor distribution as explanation for a point x_0 close to the point x to be explained, if x_0 is chosen orthogonal to the gradient ($\nabla f(x). x_0 = 0$).
- This is an explanation relative to a point x_0 such that $f(x) \approx f(x_0)$, which has for classification the same certainty and is close to the point of interest x.

A quick word on convolutions

- So far, all the algorithms (except for LIME) suggest to play with fully connected layers only, as they are easily interpretable. But not what we use for image processing…

- ConvNets have become very efficient lately by outperforming the human eye in visual tasks of different levels of complexity…
- **But are we able to explain why these networks work so well**?

- A few attempts in literature exist: CAMs, Grad-CAMs and its variations, LRP, etc.

The idea of CAMs

- Class Activation Map (CAM) is a technique that allows highlighting discriminative regions used by a CNN to identify a class in the image.
- To be able to build a CAM, CNNs must have a global average pooling layer after the final convolutional layer and then a linear (dense) classification layer.
- This means that this method cannot, unfortunately, be directly applied to all CNN architectures.
- When used on a non-compatible CNN architecture, the trick to make it work is to plug an existing pre-trained network into a global average pooling layer.

The idea of CAMs

- CAM is still powerful and easy to put in place.
- Let’s now see how it’s generated for a simple 2D image.

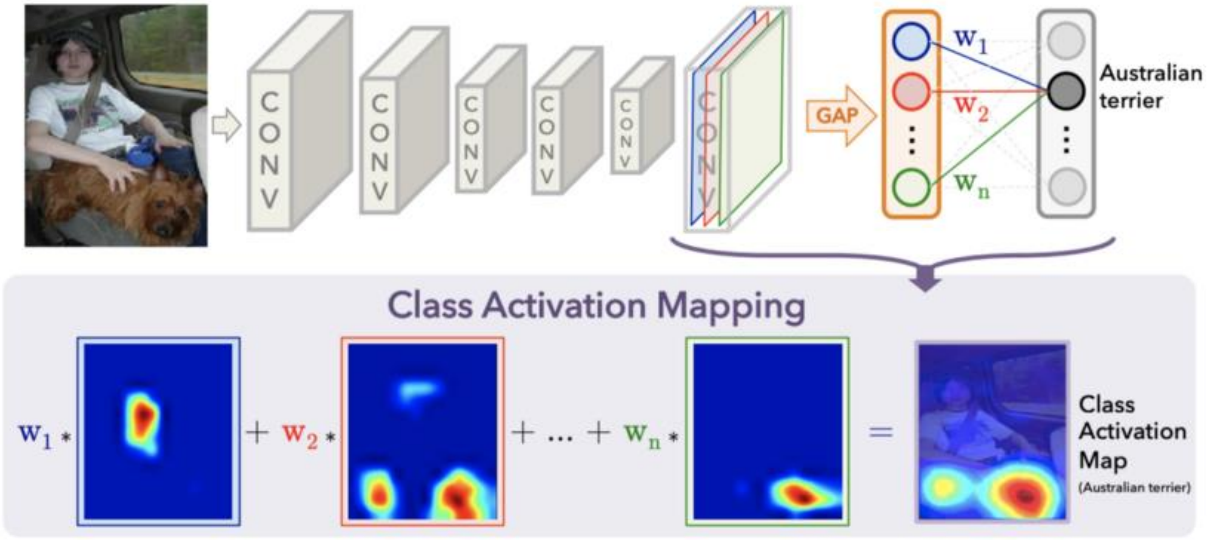
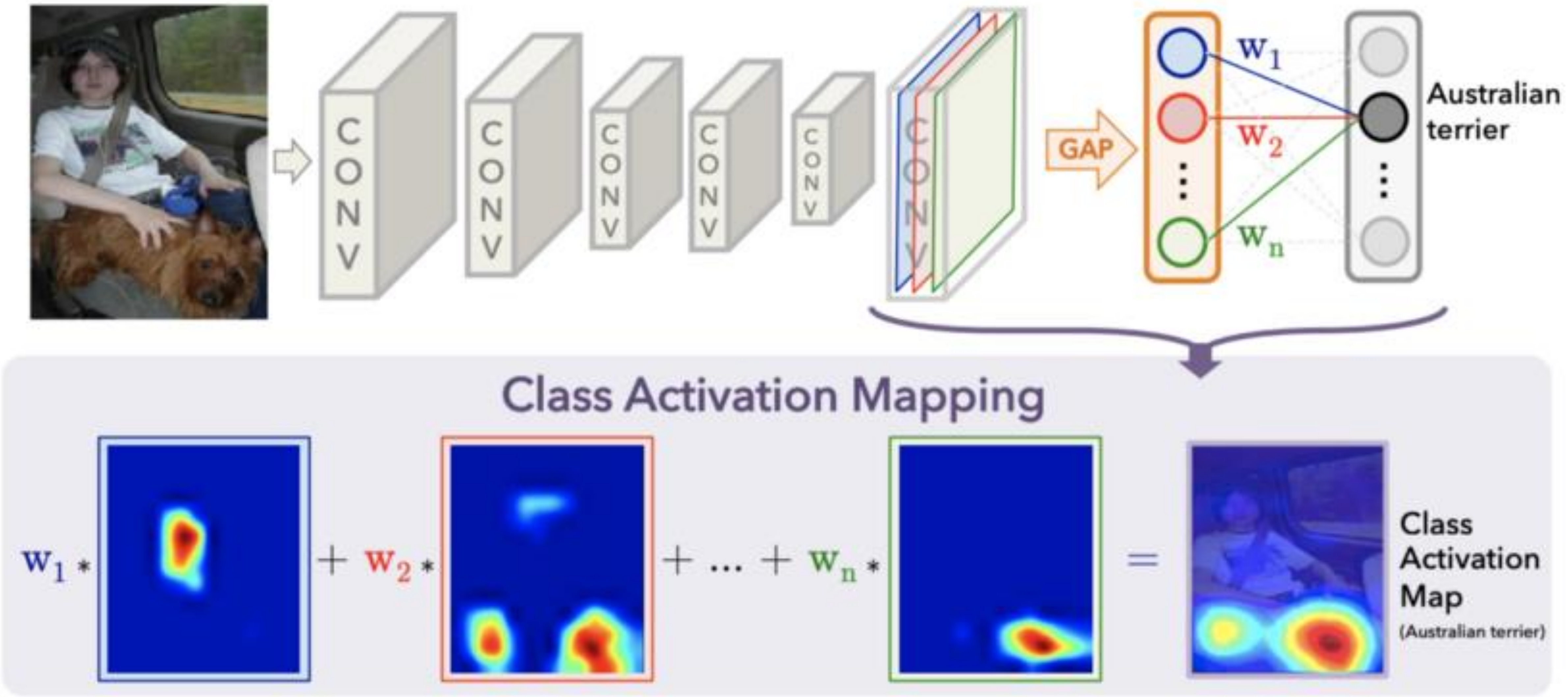


Figure 2. Class Activation Mapping: the predicted class score is mapped back to the previous convolutional layer to generate the class activation maps (CAMs). The CAM highlights the class-specific discriminative regions.

The idea of CAMs



The idea of CAMs

- When fed with a 2D color image, a CNN constrained with a global average pooling layer, outputs after the last convolutional layer a set of filters (blue, red, green) that get reduced to a vector after global average pooling.
- This vector is then connected to a final classification layer to output the predicted class.

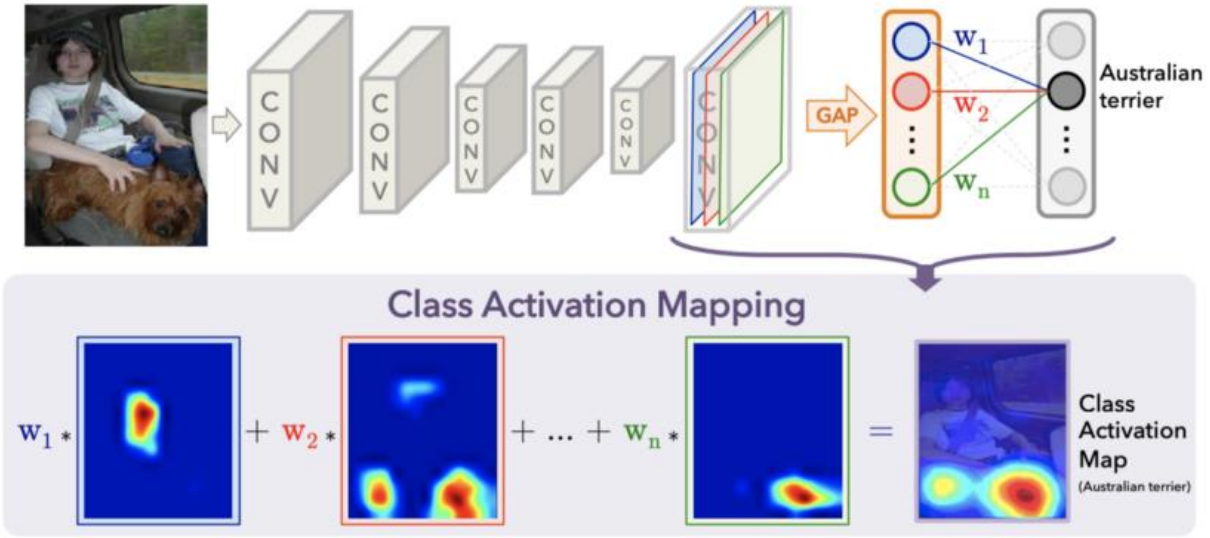


Figure 2. Class Activation Mapping: the predicted class score is mapped back to the previous convolutional layer to generate the class activation maps (CAMs). The CAM highlights the class-specific discriminative regions.

The idea of CAMs

- Each filter contains a 2D low-level spatial information about the image that got distilled after layers of successive convolutions.
- Each weight $(w_1, w_2, \dots, w_n)$ represents the partial importance of each reduced filter in computing the output class.
- Filters are adjusted throughout training by the back-propagation algorithm, but <u>preserve the spatial configuration of the image!</u>

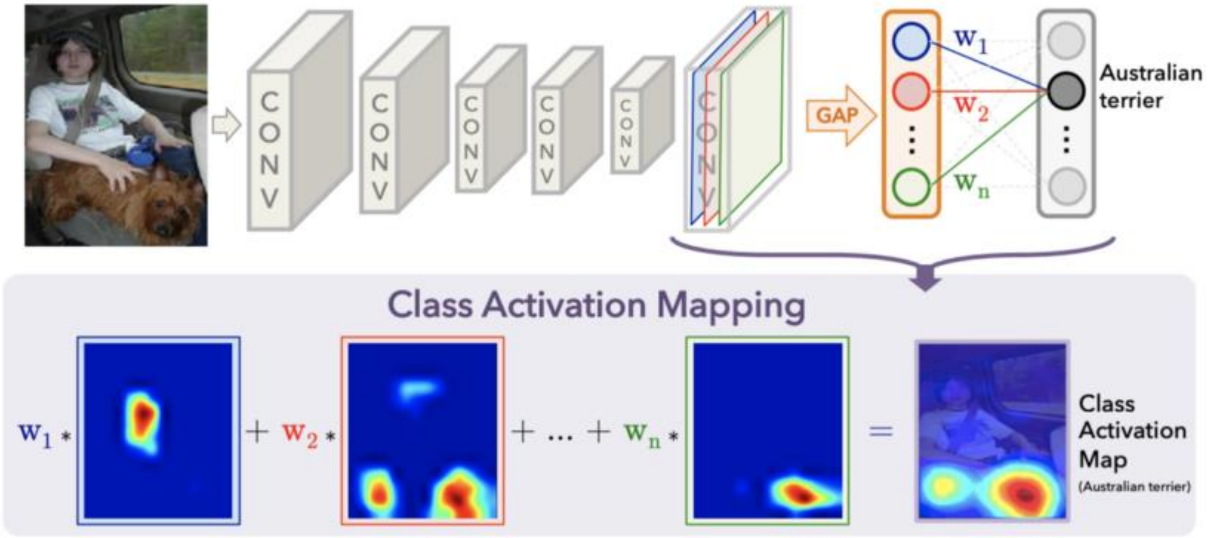


Figure 2. Class Activation Mapping: the predicted class score is mapped back to the previous convolutional layer to generate the class activation maps (CAMs). The CAM highlights the class-specific discriminative regions.

The idea of CAMs

- **A Class Activation Map is, therefore, a sum of a set of spatial 2D activations (i.e. filters) weighted with respect to their relative importance in determining the output class.**
- When superposed to the initial image, CAM is represented as a heatmap in which highly discriminative regions are painted in **red**.

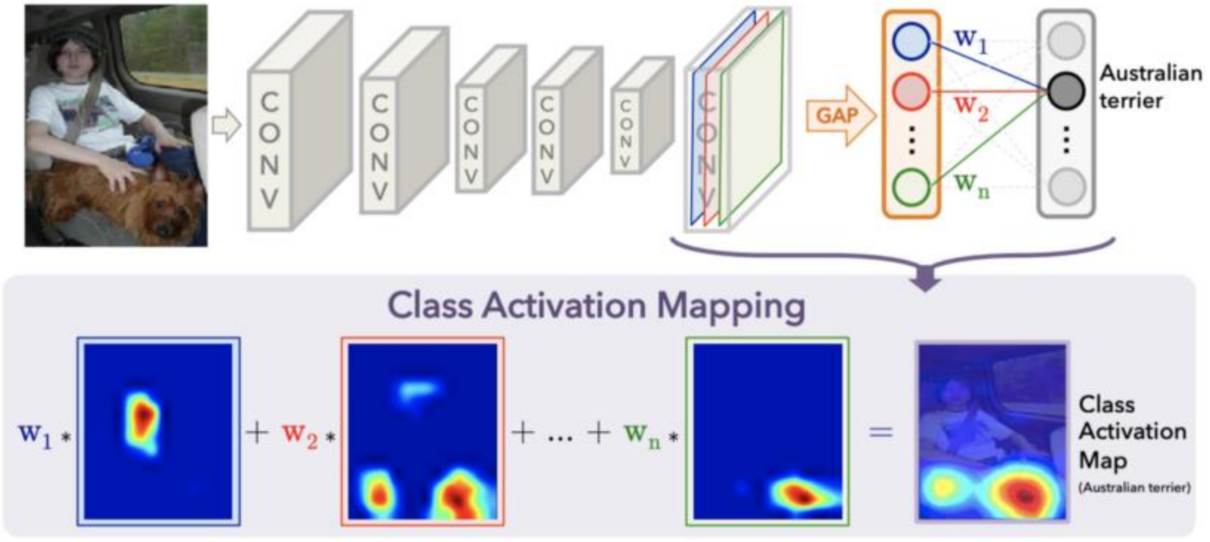


Figure 2. Class Activation Mapping: the predicted class score is mapped back to the previous convolutional layer to generate the class activation maps (CAMs). The CAM highlights the class-specific discriminative regions.

The idea of CAMs

- **Important Note:** When a class activation map is generated, it obviously has not the dimension of the initial image!
- It has the dimension of the outputs of the last Conv2d layer, which is small!
- To be able to use it and superpose it to the input, an upsampling operation (nearest neighbour? 2D-interpolation?) has to be done to resize it.

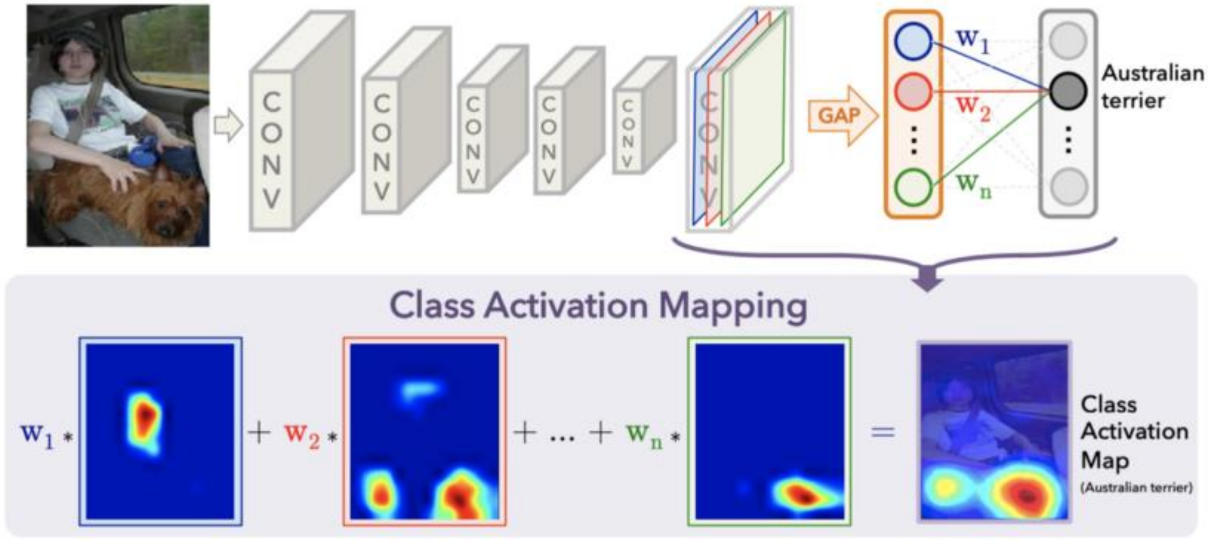
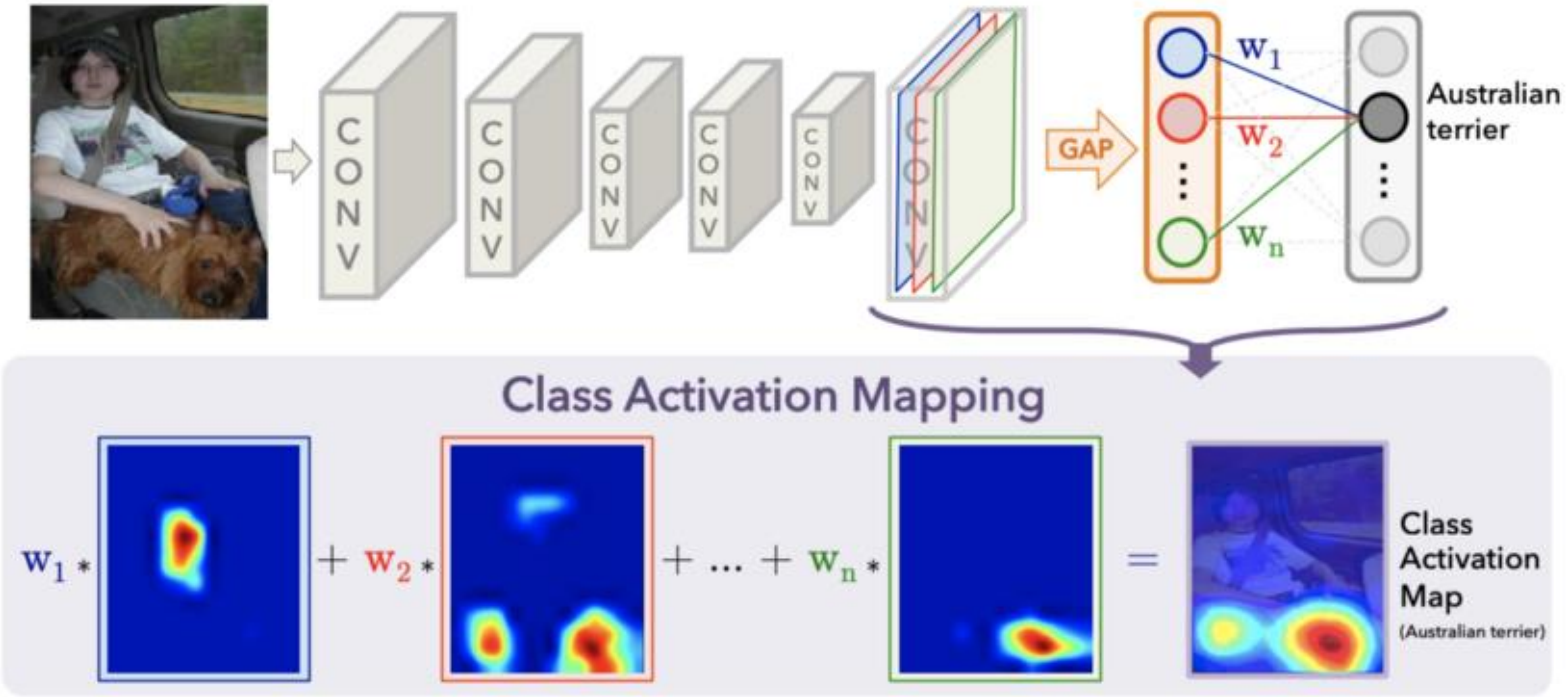


Figure 2. Class Activation Mapping: the predicted class score is mapped back to the previous convolutional layer to generate the class activation maps (CAMs). The CAM highlights the class-specific discriminative regions.

The idea of CAMs



A quick word on advanced CAMs

- Several variations of CAM exist and are, these days, considered the most interesting direction for explaining ConvNets.
- **Grad-CAM:** was built to address some of the inherent issues of CAM, so that it does not need any retraining or architectural modification (pooling needed in CAM!). Grad-CAM backpropagates to calculate and uses a fancier heuristic formula to calculate the pixel relevance score $r_d(x)$ based on weights.

$$w_k^c = \frac{1}{Z}\sum_i \sum_j \frac{\partial Y^c}{\partial A_{ij}^k}$$

A quick word on advanced CAMs

- Several variations of CAM exist and are, these days, considered the most interesting direction for explaining ConvNets.
- **Guided Grad-CAM:** Combining the ideas of Guided backpropagation (mentioned earlier) and Grad-CAM together. Best of both worlds type of approach.

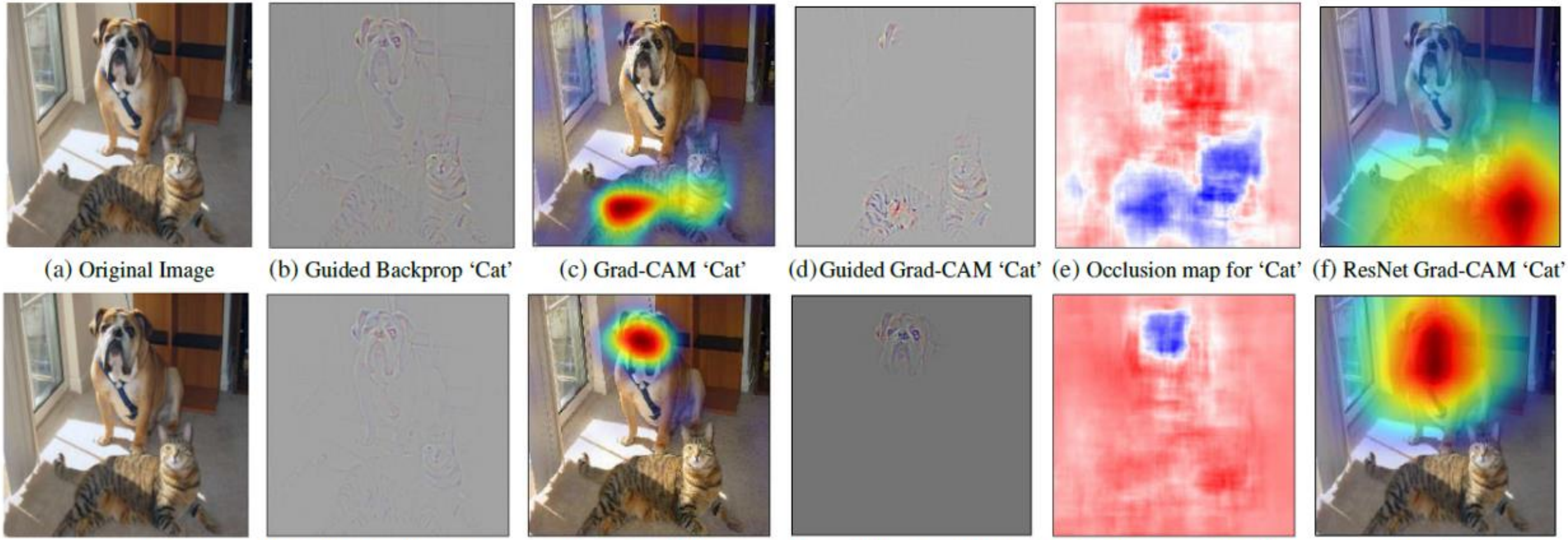
A quick word on advanced CAMs

- Several variations of CAM exist and are, these days, considered the most interesting direction for explaining ConvNets.
- **Grad-CAM++:** Built on Grad-CAM, provides better visual explanations of CNN model predictions, in terms of better object localization as well as explaining occurrences of multiple object instances in a single image. A weighted combination of the positive partial derivatives of the last convolutional layer feature maps, with respect to a specific class score as weights, is used to generate a visual explanation for the corresponding class label.

$$w_k^c = \sum_i \sum_j \left[\frac{\frac{\partial^2 Y^c}{(\partial A_{ij}^k)^2}}{2\frac{\partial^2 Y^c}{(\partial A_{ij}^k)^2} + \sum_a \sum_b A_{ab}^k \{ \frac{\partial^3 Y^c}{(\partial A_{ij}^k)^3} \}} \right] . relu\left(\frac{\partial Y^c}{\partial A_{ij}^k}\right)$$

$$\alpha_{ij}^{kc} = \frac{\frac{\partial^2 Y^c}{(\partial A_{ij}^k)^2}}{2\frac{\partial^2 Y^c}{(\partial A_{ij}^k)^2} + \sum_a \sum_b A_{ab}^k \{ \frac{\partial^3 Y^c}{(\partial A_{ij}^k)^3} \}}$$

A quick word on advanced CAMs



(a) Original Image (b) Guided Backprop 'Cat' (c) Grad-CAM 'Cat' (d) Guided Grad-CAM 'Cat' (e) Occlusion map for 'Cat' (f) ResNet Grad-CAM 'Cat'

(g) Original Image (h) Guided Backprop 'Dog' (i) Grad-CAM 'Dog' (j) Guided Grad-CAM 'Dog' (k) Occlusion map for 'Dog' (l) ResNet Grad-CAM 'Dog'

A quick word on advanced CAMs (case of image captioning?)