DEBUGGING AND UNDERSTANDING DEEP LEARNING MODELS

Narine Kokhlikyan

Research Scientist

@NARINEK

AGENDA

01
INTRO TO MODEL INTERPRETABILITY AND EXPLAINABILITY

02
EXPLAINING BBX VS BUILDING INHERENTLY INTERPRETABLE MODELS

03
GRADIENT AND PERTURBATION-BASED ATTRIBUTION ALGORITHMS

04
CASE STUDY: SEMANTIC SEGMENTATION

05
CONCEPT-BASED MODEL INTERPRETABILITY

06
MODEL COMPARISON AND CORRELATION ANALYSIS

07
RECAP & THE FUTURE

INTERPRETABILITY VS EXPLAINABILITY

THE LINE BETWEEN THESE TWO CONCEPTS IS BLURRY AND OFTEN ILL-DEFINED

EXPLAINABILITY

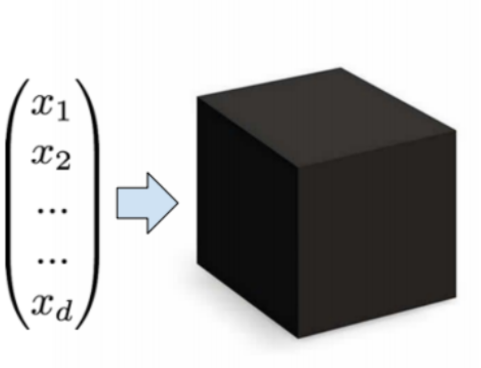
“SEEKS ANSWERS TO ‘WHY QUESTION’ ABOUT THE DECISIONS AND BEHAVIOR OF OUR MODEL*”

INTERPRETABILITY

“DESCRIBES AI MODEL INTERNALS AND THEIR PREDICTIONS IN HUMAN UNDERSTANDABLE TERMS*”

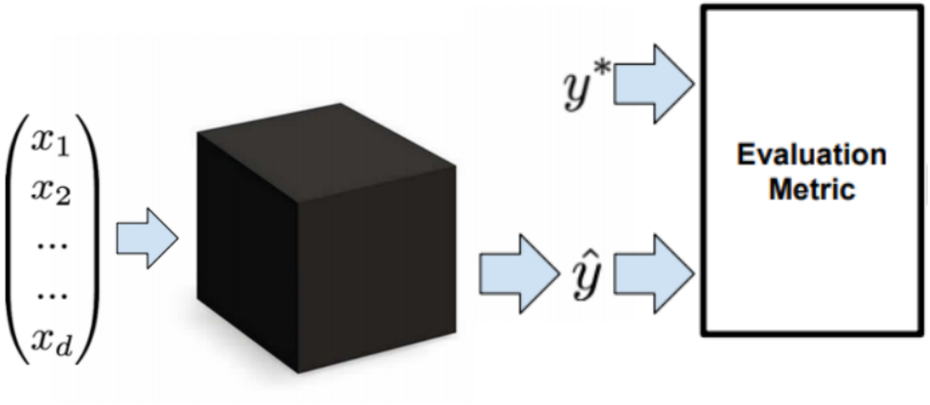
* LH Gilpin, et. al., Explaining explanations: An overview of interpretability of machine learning in IEEE 5th International Conference on data science and advanced analytics (DSAA), 2018

BLACK BOX MODELS



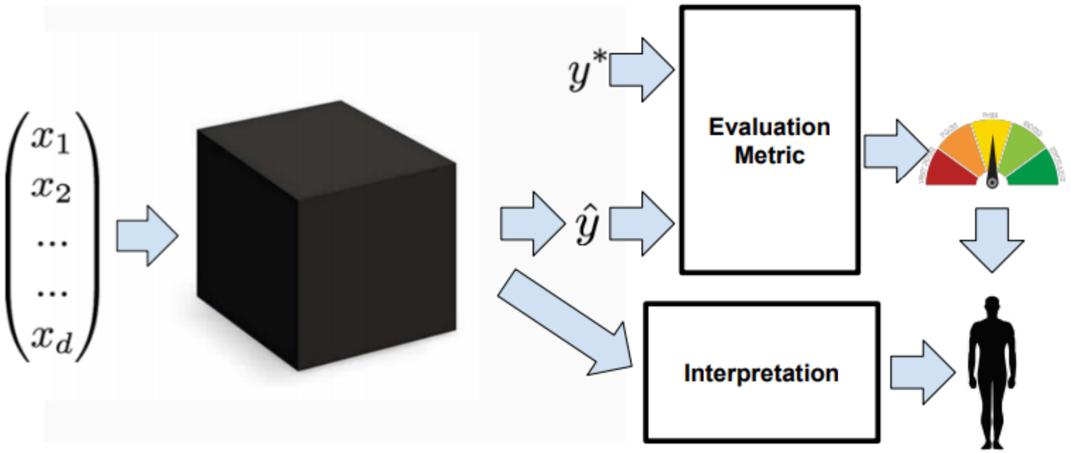
The Mythos of Model Interpretability, Zachary C. Lipton, 2017

BLACK BOX MODELS AND EVALUATION METRICS



The Mythos of Model Interpretability, Zachary C. Lipton, 2017

INTERPRETING BLACK BOX MODELS



The Mythos of Model Interpretability, Zachary C. Lipton, 2017

DESIDERATA OF INTERPRETABILITY*

- Trust
- Causality
- Transferability
- Informativeness
- Fair and ethical decision making

* The Mythos of Model Interpretability, Zachary C. Lipton, 2017

TRUST

- Is this the confidence about model performance ?
- Given an input what prediction can we expect from our model
- Model's performance can be different in deployment environment
    - We can't expect that our models always account for all kinds of biases in the data

CAUSALITY

- Inferring causal relationships from observational data
    - e.g. smoking and cancer, thalidomide use and birth defects

TRANSFERABILITY

- Generalizability and transferring knowledge to different domains
  - Ability to adopt to new environments
- ML models are susceptible to adversarial attacks and can be easily fooled

INFORMATIVENESS

- ML models convey information about the prediction only through output value
  - This is not very informative; there should be more descriptive ways of doing it
  - e.g. **Input**: `Where I can buy best chocolate ? ` **Answer**: 'Chocolatier Desiree'

FAIR AND ETHICAL DECISION MAKING

- How can we make sure that our model made decision fairly ?
  - e.g. predictions related to recidivism
- Regulations for algorithmic decisions
  - Contesting the propositions and modifying the decisions

EXPLAINING / INTERPRETING BLACK BOX

VS

BUILDING INHERENTLY INTERPRETABLE MODELS

EXPLAINING BLACK BOX MODELS

- Aka Post-Hoc Interpretability
- Infer behavior of a pre-trained model
    - based on perturbed inputs
    - gradient back-propagation
    - visualization
- No sacrifice of predictive performance

BUILDING INHERENTLY INTERPRETABLE MODELS

- By looking at the output of the model we can tell how the model came to a decision
  - e.g. decision trees, simple linear models
- More challenging for Deep Neural Networks (DNNs)
  - Model's decision making is attributed to a number of prototype samples based on similarity (`This Looks Like That`, Chen C, et. al., 2019)
- Might require performance sacrifice

EXPLAINING / INTERPRETING BLACK BOX DEEP LEARNING MODELS

GRADIENT AND PERTURBATION - BASED ATTRIBUTION METHODS

GRADIENT-BASED ATTRIBUTION ALGORITHMS

- Describes the infinitesimal change in inputs that changes the output
- Requires model's forward and backward passes

PERTURBATION-BASED ATTRIBUTION ALGORITHMS

- Observe changes of model output when the inputs are perturbed
  - e. g. Feature Ablation, Permutation, Shapley Values

GRADIENT - BASED ATTRIBUTION METHODS

SALIENCY

- Infinitesimal change in inputs that changes the output
- $\Phi_c(x) = \dfrac{\partial F_c(x)}{\partial x}^{*}$, where $x \in \mathbb{R}^N$ is the input

  $F : \mathbb{R}^N \to \mathbb{R}_c$ is the NN function

  c is the number of classes

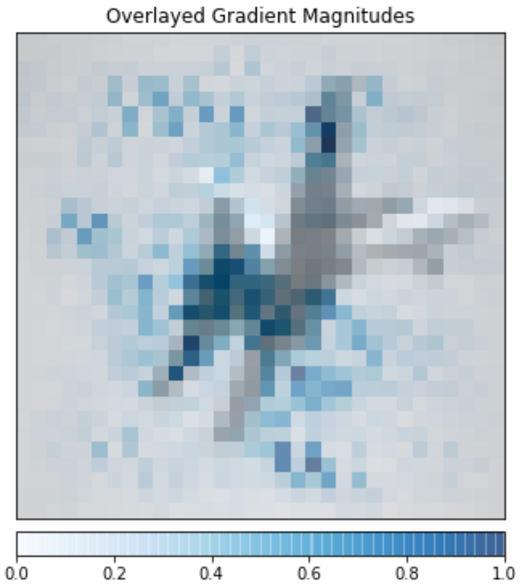
* Deep Inside Convolutional Networks, Simonyan, 2014

SALIENCY

- Infinitesimal change in inputs that changes the output
- $\Phi_c(x) = \dfrac{\partial F_c(x)}{\partial x}^{*}$, where $x \in \mathbb{R}^N$ is the input

  $F : \mathbb{R}^N \rightarrow \mathbb{R}_c$ is the NN function

  c is the number of classes



Saliency Maps trained on CIFAR10 dataset

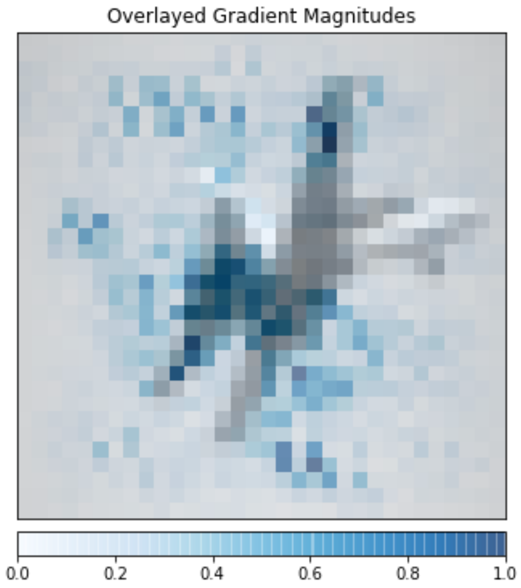
* Deep Inside Convolutional Networks, Simonyan, 2014

SALIENCY

- Infinitesimal change in inputs that changes the output
- $\Phi_c(x) = \dfrac{\partial F_c(x)}{\partial x}^{*}$, where $x \in \mathbb{R}^N$ is the input

  $F : \mathbb{R}^N \rightarrow \mathbb{R}_c$ is the NN function

  c is the number of classes



Saliency Maps trained on CIFAR10 dataset

Limitations

- Gradients are noisy and sensitive to functions input

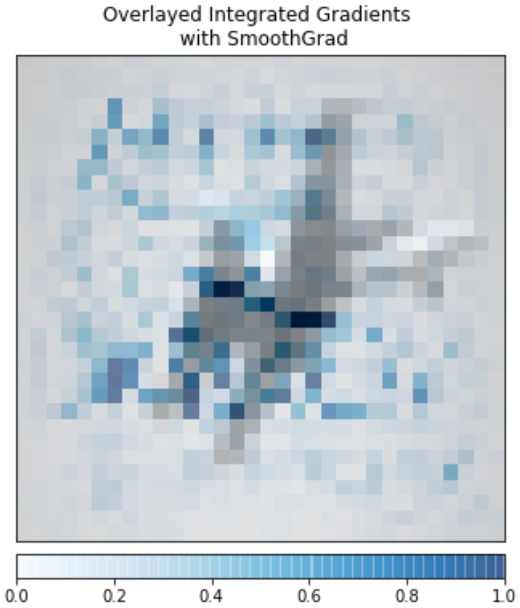
[*] Deep Inside Convolutional Networks, Simonyan, 2014

SMOOTHGRAD

- Adds noise to remove noise, SmoothGrad, Smilkov 2017
  - Samples n samples in the neighborhood of input x and averages the explanations across all those samples

$$\hat{\Phi}_c(x) = \frac{1}{n}\sum_{0}^{n} \Phi_c(x + \mathcal{N}(0,\sigma)),\ \sigma \text{ is std}$$

- Helps to stabilize explanations



Saliency Maps with SmoothGrad trained on CIFAR10 dataset

SMOOTHGRAD

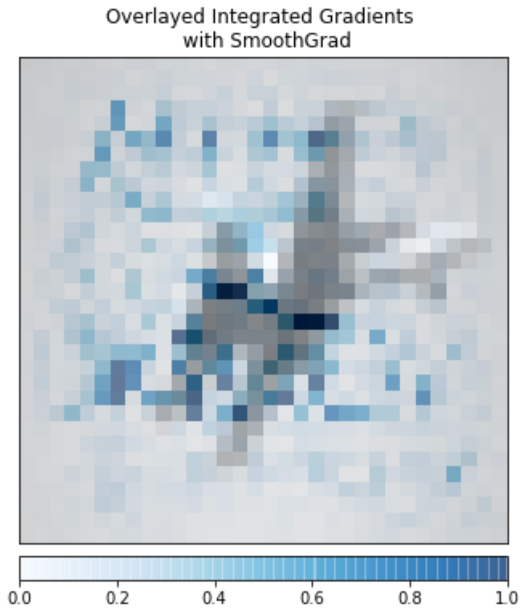
- Adds noise to remove noise, SmoothGrad, Smilkov 2017
  - Samples n samples in the neighborhood of input x and averages the explanations across all those samples

$$\hat{\Phi}_c(x) = \frac{1}{n}\sum_{0}^{n} \Phi_c(x + \mathcal{N}(0,\sigma)),\ \sigma \text{ is std}$$

- Helps to stabilize explanations

Limitations

- Finding optimal n can be challenging
- Computationally expensive depending on how large n is



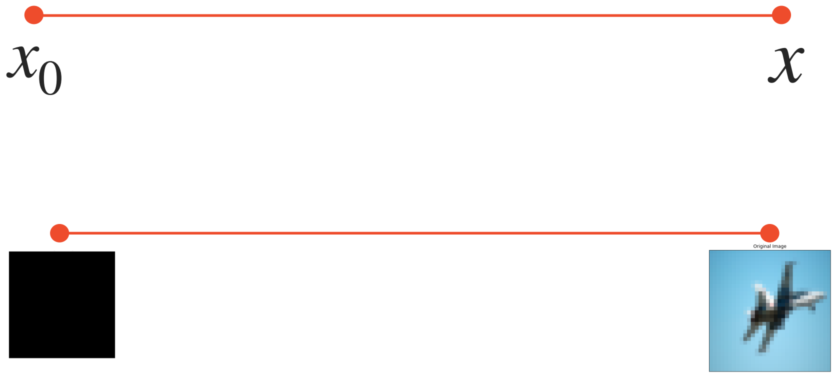
Saliency Maps with SmoothGrad trained on CIFAR10 dataset

INTEGRATED GRADIENTS

- Integrates the gradients along the path from baseline $x_0 \in \mathbb{R}^N$ to inputs $x \in \mathbb{R}^N$ (Axiomatic Attribution for Deep Networks, Sundararajan, et. al., 2017)

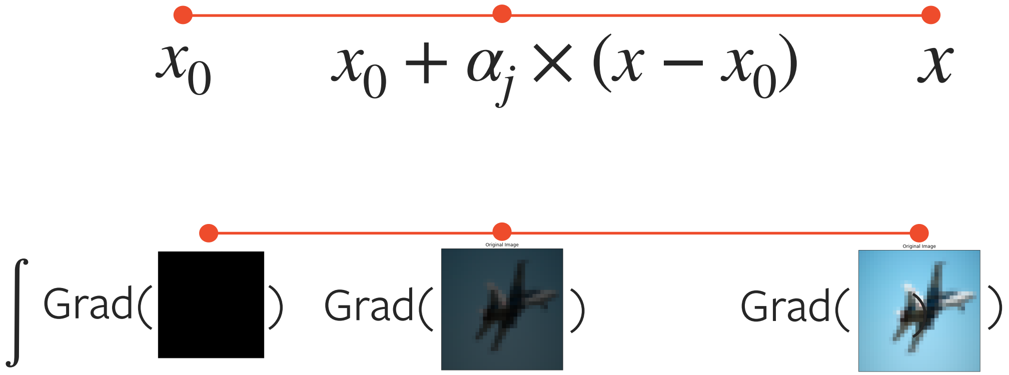
INTEGRATED GRADIENTS

- Integrates the gradients along the path from baseline $x_0 \in \mathbb{R}^N$ to inputs $x \in \mathbb{R}^N$ (Axiomatic Attribution for Deep Networks, Sundararajan, et. al., 2017)



INTEGRATED GRADIENTS

- Integrates the gradients along the path from baseline $x_0 \in \mathbb{R}^N$ to inputs $x \in \mathbb{R}^N$ (Axiomatic Attribution for Deep Networks, Sundararajan, et. al., 2017)



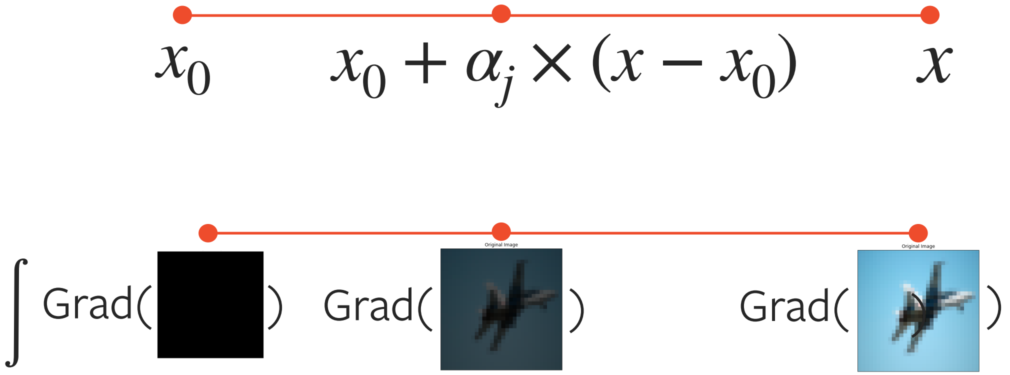
x_0 $\quad x_0 + \alpha_j \times (x - x_0) \quad$ x

$\int$ Grad( ) Grad( ) Grad( )

INTEGRATED GRADIENTS

- Integrates the gradients along the path from baseline $x_0 \in \mathbb{R}^N$ to inputs $x \in \mathbb{R}^N$ (Axiomatic Attribution for Deep Networks, Sundararajan, et. al., 2017)
- $\Phi(x^i) = (x^i - x_0^i) \cdot \int_0^1 \frac{\partial F(x_0^i + \alpha \cdot (x^i - x_0^i))d\alpha}{dx^i},$

  where $\alpha \in [0,1]$ is a scaling factor

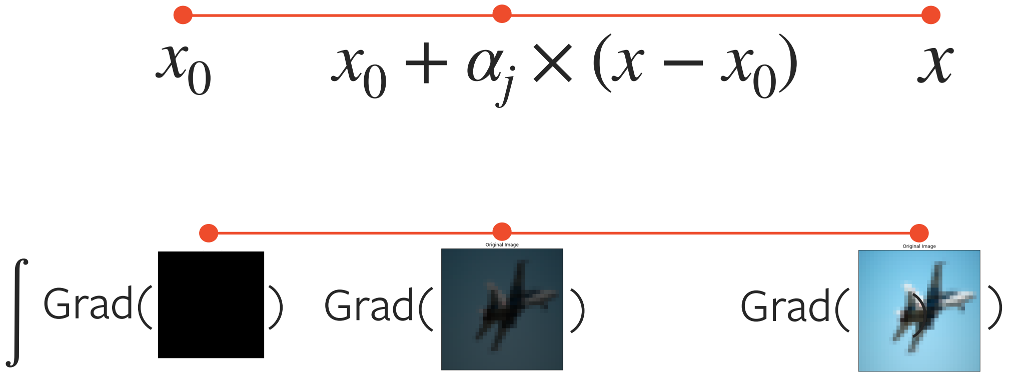


x_0 $\qquad x_0 + \alpha_j \times (x - x_0) \qquad x$

$\int$ Grad( ) Grad( ) Grad( )

INTEGRATED GRADIENTS

- Integrates the gradients along the path from baseline $x_0 \in \mathbb{R}^N$ to inputs $x \in \mathbb{R}^N$ (Axiomatic Attribution for Deep Networks, Sundararajan, et. al., 2017)
- $\Phi(x^i) = (x^i - x_0^i) \cdot \int_0^1 \frac{\partial F(x_0^i + \alpha \cdot (x^i - x_0^i))d\alpha}{dx^i}$, where $\alpha \in [0,1]$ is a scaling factor



x_0 $x_0 + \alpha_j \times (x - x_0)$ x

$\int$ Grad( ) Grad( ) Grad( )

Limitations

- Finding good baselines can be challenging
- Feature correlations and interactions aren't taken into account

AXIOMS OF INTEGRATED GRADIENTS

- Completeness
  - Sum of the attributions is equal to the Neural Network (NN) function's differences at its input and baseline

$$\sum_{i=1}^{n} \Phi(x^i) = F(x) - F(x')$$

AXIOMS OF INTEGRATED GRADIENTS

- Completeness
  - Sum of the attributions is equal to the Neural Network (NN) function's differences at its input and baseline

$$\sum_{i=1}^{n} \Phi(x^i) = F(x) - F(x')$$

- Sensitivity
  - Sensitive to differing features and output predictions

AXIOMS OF INTEGRATED GRADIENTS

- Completeness
  - Sum of the attributions is equal to the Neural Network (NN) function's differences at its input and baseline

$$\sum_{i=1}^{n} \Phi(x^i) = F(x) - F(x')$$

- Sensitivity
  - Sensitive to differing features and output predictions
- Implementation Invariance
  - Attributions for two functionally equivalent NNs are identical

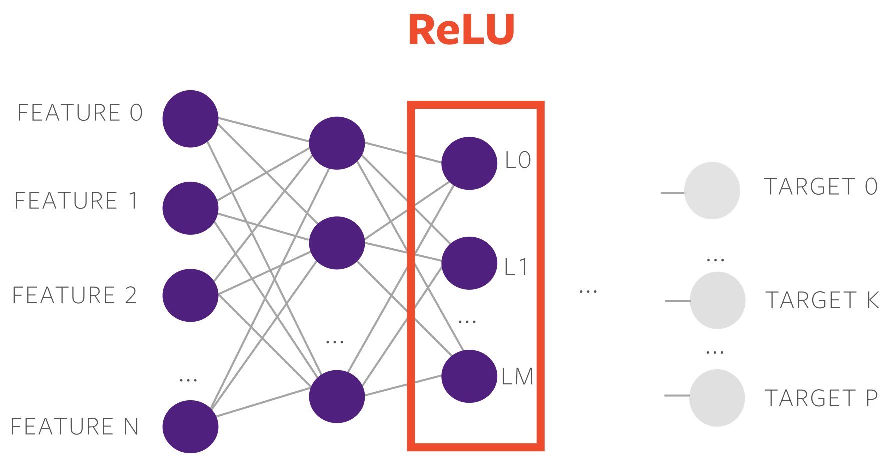
GRADIENTS - BASED METHODS WITH MODIFIED BACKPROPAGATION

- Backpropagates custom relevance score instead of gradients
- The algorithms in this category include
  - DeepLIFT, LRP, GuidedBackprop, GradCAM, GuidedGradCam, Deconvolution

GRADIENTS - BASED METHODS WITH MODIFIED BACKPROPAGATION

- Backpropagates custom relevance score instead of gradients
- The algorithms in this category include
  - DeepLIFT, LRP, GuidedBackprop, GradCAM, GuidedGradCam, **Deconvolution**

Deconvolution

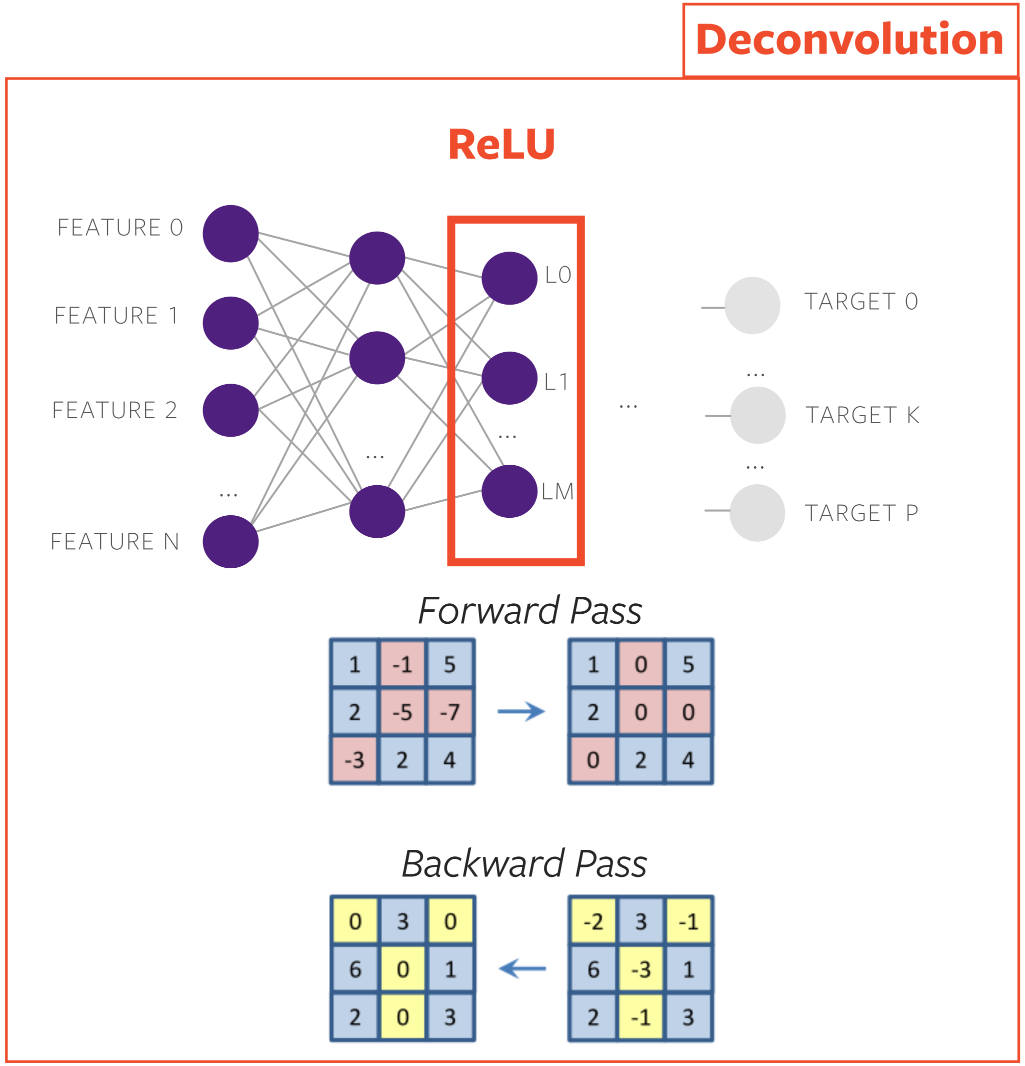


Forward Pass

$$\begin{bmatrix} 1 & -1 & 5 \\ 2 & -5 & -7 \\ -3 & 2 & 4 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 5 \\ 2 & 0 & 0 \\ 0 & 2 & 4 \end{bmatrix}$$

GRADIENTS - BASED METHODS WITH MODIFIED BACKPROPAGATION

- Backpropagates custom relevance score instead of gradients
- The algorithms in this category include
  - DeepLIFT, LRP, GuidedBackprop, GradCAM, GuidedGradCam, Deconvolution



Forward Pass

1	-1	5
2	-5	-7
-3	2	4

1	0	5
2	0	0
0	2	4

Backward Pass

0	3	0
6	0	1
2	0	3

-2	3	-1
6	-3	1
2	-1	3

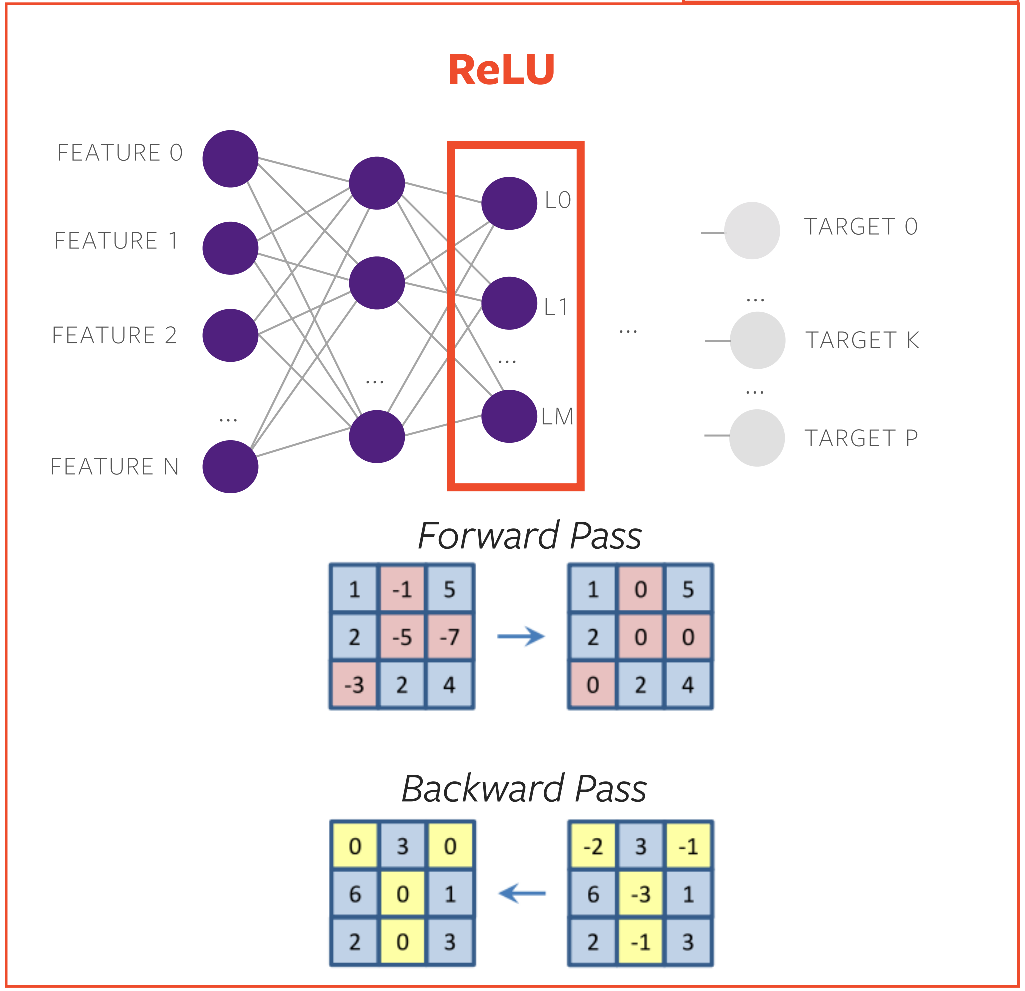
GRADIENTS - BASED METHODS WITH MODIFIED BACKPROPAGATION

- Backpropagates custom relevance score instead of gradients
- The algorithms in this category include
  - DeepLIFT, LRP, GuidedBackprop, GradCAM, GuidedGradCam, Deconvolution

Limitations

- Most of these methods except DeepLIFT are insensitive to parameter randomization (When Explanations Lie, Sixt 2020)

Deconvolution



Forward Pass

1	-1	5
2	-5	-7
-3	2	4

→

1	0	5
2	0	0
0	2	4

Backward Pass

0	3	0
6	0	1
2	0	3

←

-2	3	-1
6	-3	1
2	-1	3

PERTURBATION - BASED ATTRIBUTION METHODS

FEATURE ABLATION

- Measures the importance of feature(s) based on the magnitude changes in prediction scores or measures of prediction goodness when feature(s) are ablated

FEATURE ABLATION

- Measures the importance of feature(s) based on the magnitude changes in prediction scores or measures of prediction goodness when feature(s) are ablated

input $x \in \mathbb{R}^N$

0.29	0.51	0.25	0.68
0.07	0.86	0.13	0.10
0.18	0.72	0.31	0.68
0.07	0.31	0.31	0.40

mask $m \in \mathbb{Z}^{0+}$

0	1	0	2
0	1	0	2
0	1	0	2
0	1	0	2

baseline $b \in \mathbb{R}^N$

0.0	0.0	0.0	0.0	Default
1.0	-1.0	1.0	0.0	Custom

FEATURE ABLATION

- Measures the importance of feature(s) based on the magnitude changes in prediction scores or measures of prediction goodness when feature(s) are ablated
- $\Phi(F, x, m, b) = F(x) - F(x, m, b)$, where $m \in \mathbb{Z}^{0+}$ is the ablation mask and $b \in \mathbb{R}^N$ ablation values

FEATURE ABLATION

- Measures the importance of feature(s) based on the magnitude changes in prediction scores or measures of prediction goodness when feature(s) are ablated
- $\Phi(F, x, m, b) = F(x) - F(x, m, b)$, where $m \in \mathbb{Z}^{0+}$ is the ablation mask and $b \in \mathbb{R}^{N}$ ablation values
- We can also combine it with a loss function or any evaluation metric

  $\Phi(F, x, m, b, l, t) = l(F, x, t) - l(F, x, m, b, t)$, where $l : \mathbb{R}^{N} \rightarrow \mathbb{Z}^{M}$ is the loss and $t \in \mathbb{Z}^{M}$ the labels

FEATURE ABLATION

- Measures the importance of feature(s) based on the magnitude changes in prediction scores or measures of prediction goodness when feature(s) are ablated
- $\Phi(F, x, m, b) = F(x) - F(x, m, b)$, where $m \in \mathbb{Z}^{0+}$ is the ablation mask and $b \in \mathbb{R}^N$ ablation values
- We can also combine it with a loss function or any evaluation metric $\Phi(F, x, m, b, l, t) = l(F, x, t) - l(F, x, m, b, t)$, where $l : \mathbb{R}^N \rightarrow \mathbb{Z}^M$ is the loss and $t \in \mathbb{Z}^M$ the labels
- Occlusion (Visualizing and Understanding Convolutional Networks, Zeiler, et. al., 2013)
  - Ablates rectangular patches of inputs and computes the differences of output function with and without ablation

FEATURE ABLATION

- Measures the importance of feature(s) based on the magnitude changes in prediction scores or measures of prediction goodness when feature(s) are ablated
- $\Phi(F, x, m, b) = F(x) - F(x, m, b)$, where $m \in \mathbb{Z}^{0+}$ is the ablation mask and $b \in \mathbb{R}^N$ ablation values
- We can also combine it with a loss function or any evaluation metric

  $\Phi(F, x, m, b, l, t) = l(F, x, t) - l(F, x, m, b, t)$, where $l : \mathbb{R}^N \rightarrow \mathbb{Z}^M$ is the loss and $t \in \mathbb{Z}^M$ the labels

Limitations

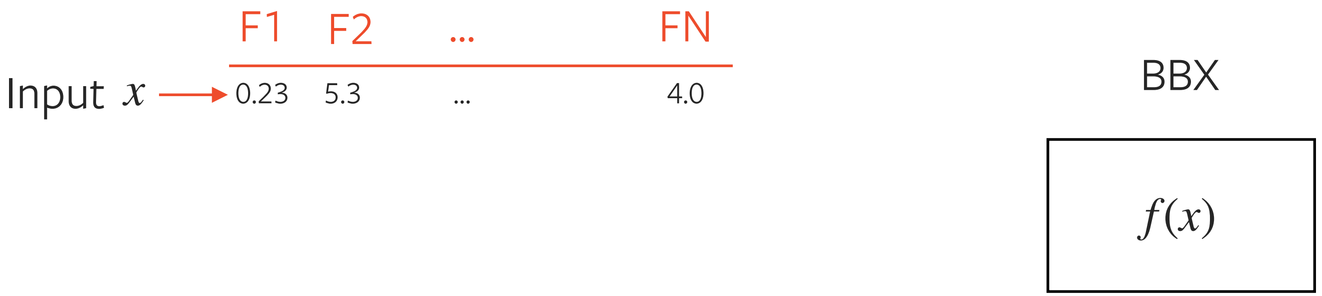
- Identifying which features mask together and what ablation values use
- Inputs with ablated features might be out of test/train/valid data distributions

LIME

- Approximates the predictions of black-box model with an interpretable surrogate model such as linear model or a decision tree [Why should I trust you? Explaining the predictions of any classifier, Ribeiro, 2016]

LIME

- Approximates the predictions of black-box model with an interpretable surrogate model such as linear model or a decision tree [Why should I trust you? Explaining the predictions of any classifier, Ribeiro, 2016]



Input x →

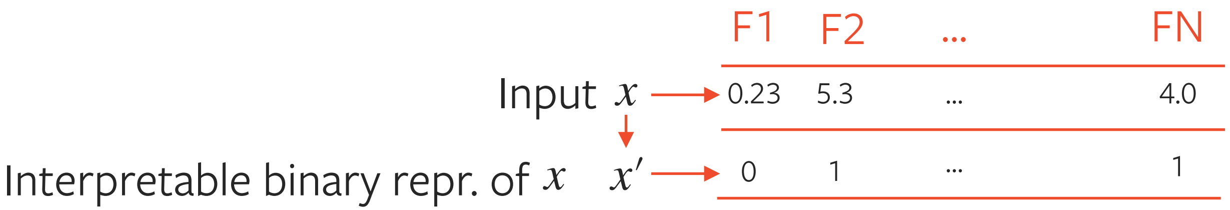
F1	F2	...	FN
0.23	5.3	...	4.0

BBX

$f(x)$

LIME

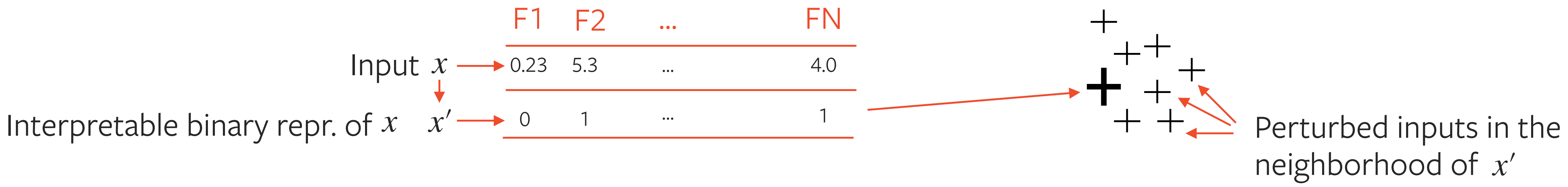
- Approximates the predictions of black-box model with an interpretable surrogate model such as linear model or a decision tree [Why should I trust you? Explaining the predictions of any classifier, Ribeiro, 2016]



	F1	F2	...	FN
Input x	0.23	5.3	...	4.0
Interpretable binary repr. of x x'	0	1	...	1

LIME

- Approximates the predictions of black-box model with an interpretable surrogate model such as linear model or a decision tree [Why should I trust you? Explaining the predictions of any classifier, Ribeiro, 2016]

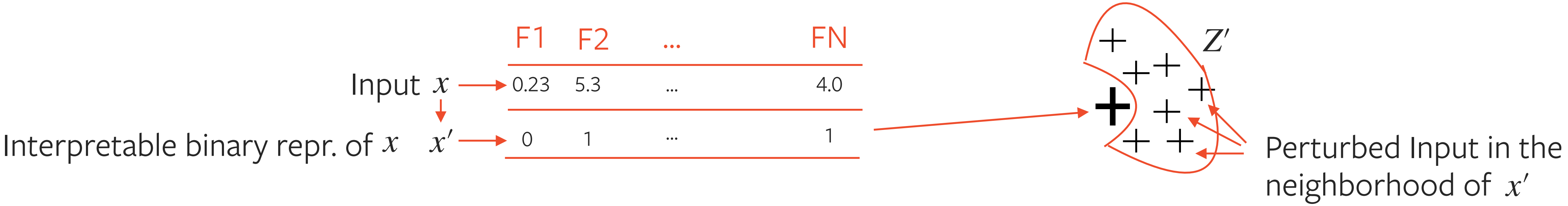


	F1	F2	...	FN
Input x	0.23	5.3	...	4.0
Interpretable binary repr. of x x'	0	1	...	1

Perturbed inputs in the neighborhood of x'

LIME

- Approximates the predictions of black-box model with an interpretable surrogate model such as linear model or a decision tree [Why should I trust you? Explaining the predictions of any classifier, Ribeiro, 2016]

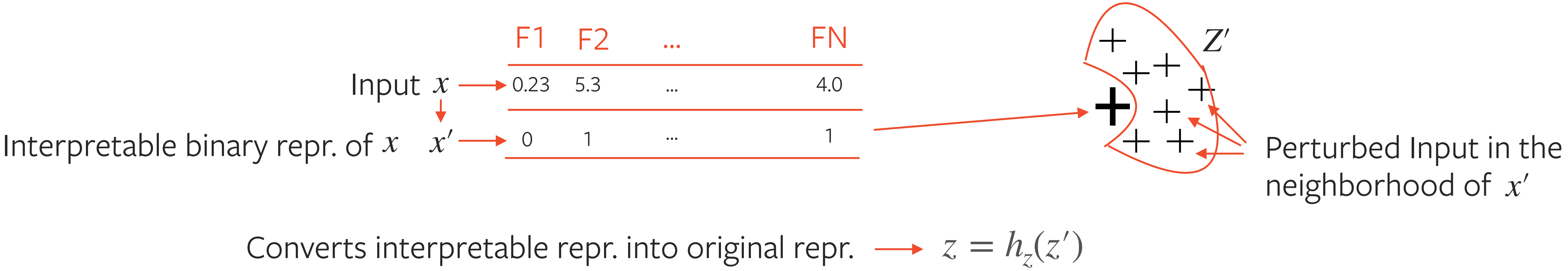


	F1	F2	...	FN
Input x	0.23	5.3	...	4.0
Interpretable binary repr. of x — x'	0	1	...	1

Perturbed Input in the neighborhood of x'

LIME

- Approximates the predictions of black-box model with an interpretable surrogate model such as linear model or a decision tree [Why should I trust you? Explaining the predictions of any classifier, Ribeiro, 2016]



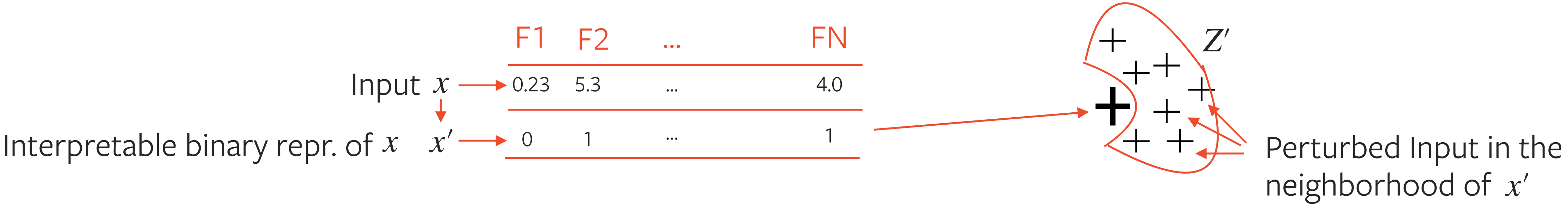
	F1	F2	...	FN
Input x	0.23	5.3	...	4.0
Interpretable binary repr. of x x'	0	1	...	1

Perturbed Input in the neighborhood of x'

Converts interpretable repr. into original repr. → $z = h_z(z')$

LIME

- Approximates the predictions of black-box model with an interpretable surrogate model such as linear model or a decision tree [Why should I trust you? Explaining the predictions of any classifier, Ribeiro, 2016]



	F1	F2	...	FN
Input x	0.23	5.3	...	4.0
Interpretable binary repr. of x x'	0	1	...	1

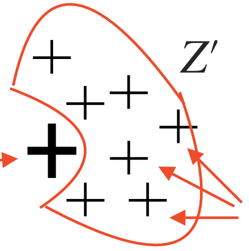
Perturbed Input in the neighborhood of x'

Converts interpretable repr. into original repr. → $z = h_z(z')$

Compute similarity between x and $z \in Z$ → $\pi_x(z)$

LIME

	F1	F2	...	FN
Input x	0.23	5.3	...	4.0
Interpretable binary repr. of x x'	0	1	...	1



Perturbed Input in the neighborhood of x'

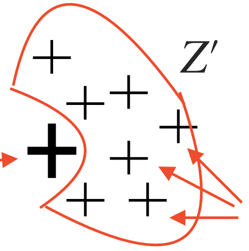
Converts interpretable repr. into original repr. → $z = h_z(z')$

Compute similarity between x and $z \in Z$ → $\pi_x(z)$

Interpretable model → $g(z')$

LIME

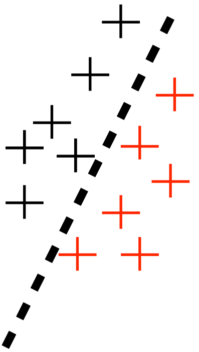
	F1	F2	...	FN
Input x	0.23	5.3	...	4.0
Interpretable binary repr. of x x'	0	1	...	1



Perturbed Input in the neighborhood of x'

Converts interpretable repr. into original repr. → $z = h_z(z')$

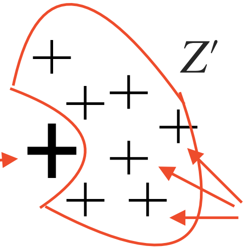
Compute similarity between x and $z \in Z$ → $\pi_x(z)$



Train interpretable model $g(z')$

LIME

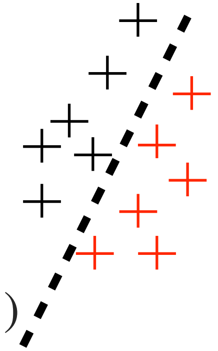
	F1	F2	...	FN
Input x	0.23	5.3	...	4.0
Interpretable binary repr. of x x'	0	1	...	1



Perturbed Input in the neighborhood of x'

Converts interpretable repr. into original repr. → $z = h_z(z')$

Compute similarity between x and $z \in Z$ → $\pi_x(z)$



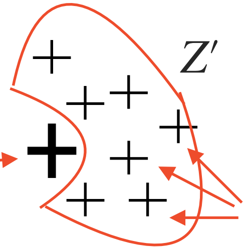
Train interpretable model $g(z')$

BBX

$\boxed{f(z)}$

LIME

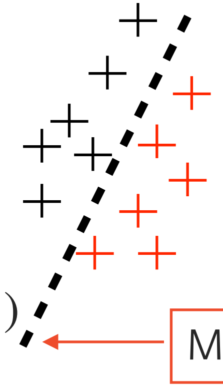
	F1	F2	...	FN
Input x →	0.23	5.3	...	4.0
Interpretable binary repr. of x x' →	0	1	...	1



Perturbed Input in the neighborhood of x'

Converts interpretable repr. into original repr. → $z = h_z(z')$

Compute similarity between x and $z \in Z$ → $\pi_x(z)$



Train interpretable model $g(z')$

BBX

$f(z)$

Minimize the weighted distance

$$L(f, g, \pi) = \sum_{z \in Z, z' \in Z'} \pi_x(z)(f(z) - g(z'))^2$$

LIME

Minimizing $L(f, x, g)$ helps us to estimate w_i^g which serve as importance scores for interpretable features

$$L(f, g, \pi) = \sum_{z \in Z, z' \in Z'} \pi_x(z)(f(z) - g(z'))^2$$

		F1	F2	...	FN
Input	$x \rightarrow$	0.23	5.3	...	4.0
Interpretable binary repr. of x	$x' \rightarrow$	0	1	...	1
Importance scores	$\phi \rightarrow$	w_1^g	w_2^g	...	w_N^g

LIME

Limitations

- Depends on choice sampling technique, sample size and similarity function
- Depends on the accuracy and complexity of the interpretable model

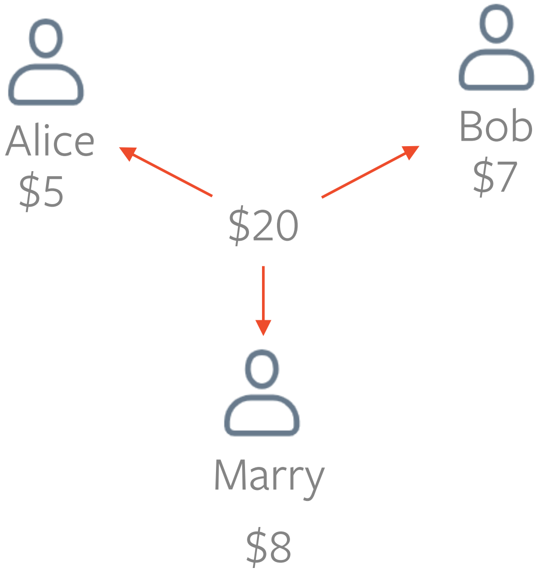
SHAPLEY VALUES

- Measures expected marginal contributions of each feature for given sample and prediction - a technique adopted from cooperative game theory*

* A value for n-person games. Contributions to the Theory of Games 2.28 (1953): 307-317, L.S. Shapley, 1952

SHAPLEY VALUES

- Measures expected marginal contributions of each feature for given sample and prediction - a technique adopted from cooperative game theory*



* A value for n-person games. Contributions to the Theory of Games 2.28 (1953): 307-317, L.S. Shapley, 1952

SHAPLEY VALUES

Age	#Rooms	Location	**Price**
40 years	3 rooms	SF Downtown	**$500.000**

F(40 years, 3 rooms, SF Downtown) = $500.000

SHAPLEY VALUES

Age	#Rooms	Location	**Price**
40 years	3 rooms	SF Downtown	**$500.000**

F(40 years, 3 rooms, SF Downtown) = $500.000

What are the contributions of each feature for this prediction ?

SHAPLEY VALUES

0 1 2 ← Feature IDs

Age	#Rooms	Location	**Price**
40 years	3 rooms	SF Downtown	**$500.000**

F(40 years, 3 rooms, SF Downtown) = $500.000

What are the contributions of each feature for this prediction ?

SHAPLEY VALUES

0 1 2 ← Feature IDs

Age	#Rooms	Location	**Price**
40 years	3 rooms	SF Downtown	**$500.000**

F(40 years, 3 rooms, SF Downtown) = $500.000

What are the contributions of each feature for this prediction ?

Feature Permutations

1	0	1	2
2	1	0	2
3	2	0	1
4	0	2	1
5	2	0	0
6	0	2	0

SHAPLEY VALUES

0	1	2	← Feature IDs
Age	#Rooms	Location	**Price**
40 years	3 rooms	SF Downtown	**$500.000**

F(40 years, 3 rooms, SF Downtown) = $500.000

What are the contributions of each feature for this prediction ?

Feature Permutations

1	0	1	2
2	1	0	2
3	2	0	1
4	0	2	1
5	2	0	0
6	0	2	0

← Adding features in this order

SHAPLEY VALUES

Feature IDs

0	1	2	
Age	#Rooms	Location	**Price**
40 years	3 rooms	SF Downtown	**$500.000**

Feature Permutations

Adding features in this order

1	0	1	2
2	1	0	2
3	2	0	1
4	0	2	1
5	2	0	0
6	0	2	0

F(40 years, 3 rooms, SF Downtown) = $500.000

What are the contributions of each feature for this prediction ?

out0_1 = F(Empty, Empty, Empty)

SHAPLEY VALUES

0	1	2	← Feature IDs
Age	#Rooms	Location	**Price**
40 years	3 rooms	SF Downtown	**$500.000**

F(40 years, 3 rooms, SF Downtown) = $500.000

What are the contributions of each feature for this prediction ?

out0_1 = F(Empty, Empty, Empty)

out1_1 = F(40 years, Empty, Empty)

Feature Permutations

1	0	1	2
2	1	0	2
3	2	0	1
4	0	2	1
5	2	0	0
6	0	2	0

← Adding features in this order

SHAPLEY VALUES

0	1	2	
Age	#Rooms	Location	**Price**
40 years	3 rooms	SF Downtown	**$500.000**

Feature IDs

F(40 years, 3 rooms, SF Downtown) = $500.000

What are the contributions of each feature for this prediction ?

out0_1 = F(Empty, Empty, Empty)

out1_1 = F(40 years, Empty, Empty)

out2_1 = F(40 years, 3 rooms, Empty)

Feature Permutations

1	0	1	2
2	1	0	2
3	2	0	1
4	0	2	1
5	2	0	0
6	0	2	0

Adding features in this order

SHAPLEY VALUES

0	1	2	← Feature IDs
Age	#Rooms	Location	**Price**
40 years	3 rooms	SF Downtown	**$500.000**

Feature Permutations

1	0	1	2
2	1	0	2
3	2	0	1
4	0	2	1
5	2	0	0
6	0	2	0

← Adding features in this order

F(40 years, 3 rooms, SF Downtown) = $500.000

What are the contributions of each feature for this prediction ?

out0_1 = F(Empty, Empty, Empty)

out1_1 = F(40 years, Empty, Empty)

out2_1 = F(40 years, 3 rooms, Empty)

out3_1 = F(40 years, 3 rooms, SF Downtown)

SHAPLEY VALUES

0	1	2	← Feature IDs
Age	#Rooms	Location	**Price**
40 years	3 rooms	SF Downtown	**$500.000**

F(40 years, 3 rooms, SF Downtown) = $500.000

What are the contributions of each feature for this prediction ?

out0_1 = F(Empty, Empty, Empty)

out1_1 = F(40 years, Empty, Empty)

out2_1 = F(40 years, 3 rooms, Empty)

out3_1 = F(40 years, 3 rooms, SF Downtown)

Feature Permutations

1	0	1	2
2	1	0	2
3	2	0	1
4	0	2	1
5	2	0	0
6	0	2	0

← Adding features in this order

Feature Contributions for the 1st permutation

Age	#Rooms	Location
out1_1 - out0_1	out2_1 - out1_1	out3_1 - out2_1

SHAPLEY VALUES

0 1 2 ← Feature IDs

Age	#Rooms	Location	**Price**
40 years	3 rooms	SF Downtown	**$500.000**

F(40 years, 3 rooms, SF Downtown) = $500.000

What are the contributions of each feature for this prediction ?

Feature Permutations

1	0	1	2
2	1	0	2
3	2	0	1
4	0	2	1
5	2	0	0
6	0	2	0

← Adding features in this order

SHAPLEY VALUES

0	1	2	Feature IDs
Age	#Rooms	Location	**Price**
40 years	3 rooms	SF Downtown	**$500.000**

F(40 years, 3 rooms, SF Downtown) = $500.000

What are the contributions of each feature for this prediction ?

out0_2 = F(Empty, Empty, Empty)

Feature Permutations

1	0	1	2
2	1	0	2
3	2	0	1
4	0	2	1
5	2	0	0
6	0	2	0

Adding features in this order

SHAPLEY VALUES

0	1	2		
Age	#Rooms	Location	**Price**	
40 years	3 rooms	SF Downtown	**$500.000**	

← Feature IDs

F(40 years, 3 rooms, SF Downtown) = $500.000

What are the contributions of each feature for this prediction ?

out0_2 = F(Empty, Empty, Empty)

out1_2 = F(Empty, 3 rooms, Empty)

Feature Permutations

1	0	1	2
2	1	0	2
3	2	0	1
4	0	2	1
5	2	0	0
6	0	2	0

← Adding features in this order

SHAPLEY VALUES

0	1	2	
Age	#Rooms	Location	**Price**
40 years	3 rooms	SF Downtown	**$500.000**

← Feature IDs

F(40 years, 3 rooms, SF Downtown) = $500.000

What are the contributions of each feature for this prediction ?

out0_2 = F(Empty, Empty, Empty)

out1_2 = F(Empty, 3 rooms, Empty)

out2_2 = F(40 years, 3 rooms, Empty)

Feature Permutations

1	0	1	2
2	1	0	2
3	2	0	1
4	0	2	1
5	2	0	0
6	0	2	0

← Adding features in this order

SHAPLEY VALUES

0	1	2	
Age	#Rooms	Location	**Price**
40 years	3 rooms	SF Downtown	**$500.000**

← Feature IDs

F(40 years, 3 rooms, SF Downtown) = $500.000

What are the contributions of each feature for this prediction ?

out0_2 = F(Empty, Empty, Empty)

out1_2 = F(Empty, 3 rooms, Empty)

out2_2 = F(40 years, 3 rooms, Empty)

out3_2 = F(40 years, 3 rooms, SF Downtown)

Feature Permutations

1	0	1	2
2	1	0	2
3	2	0	1
4	0	2	1
5	2	0	0
6	0	2	0

← Adding features in this order

...

SHAPLEY VALUES

0	1	2	← Feature IDs
Age	#Rooms	Location	**Price**
40 years	3 rooms	SF Downtown	**$500.000**

F(40 years, 3 rooms, SF Downtown) = $500.000

What are the contributions of each feature for this prediction ?

Feature Permutations

1	0	1	2
2	1	0	2
3	2	0	1
4	0	2	1
5	2	0	0
6	0	2	0

← Adding features in this order

...

out0_2 = F(Empty, Empty, Empty)

out1_2 = F(Empty, 3 rooms, Empty)

out2_2 = F(40 years, 3 rooms, Empty)

out3_2 = F(40 years, 3 rooms, SF Downtown)

Feature Contributions for the 2nd permutation

Age	#Rooms	Location
out2_2 - out1_2	out1_2 - out0_2	out3_2 - out2_2

SHAPLEY VALUES

0 1 2 ← Feature IDs

Age	#Rooms	Location	**Price**
40 years	3 rooms	SF Downtown	**$500.000**

F(40 years, 3 rooms, SF Downtown) = $500.000

What are the contributions of each feature for this prediction ?

Age	#Rooms	Location
out1_1 - out0_1	out2_1 - out1_1	out3_1 - out2_1
out2_2 - out1_2	out1_2 - out0_2	out3_2 - out2_2

...

Feature Permutations

1	0	1	2
2	1	0	2
3	2	0	1
4	0	2	1
5	2	0	0
6	0	2	0

↓ ...

SHAPLEY VALUES

0	1	2	← Feature IDs
Age	#Rooms	Location	**Price**
40 years	3 rooms	SF Downtown	**$500.000**

F(40 years, 3 rooms, SF Downtown) = $500.000

What are the contributions of each feature for this prediction ?

Age	#Rooms	Location
out1_1 - out0_1	out2_1 - out1_1	out3_1 - out2_1
out2_2 - out1_2	out1_2 - out0_2	out3_2 - out2_2
	...	

Sum and normalize by 6
the number of permutations

Feature Permutations

1	0	1	2
2	1	0	2
3	2	0	1
4	0	2	1
5	2	0	0
6	0	2	0

↓ ...

SHAPLEY VALUES

- Formally Shapley values is defined as

$\Phi(x_i) = \Sigma_{S \subseteq N \setminus i} \frac{S!(N-S-1)!}{N!}[F_{S \cup \{i\}}(x_{S \cup \{i\}}) - F_S(x_S)]$, where $N \in \mathbb{Z}^M$ is the total number of features and $S \in N$ is a subset of features

SHAPLEY VALUES

- Formally Shapley values is defined as

$\Phi(x_i) = \Sigma_{S \subseteq N \backslash i} \frac{S!(N-S-1)!}{N!}[F_{S \cup \{i\}}(x_{S \cup \{i\}}) - F_S(x_S)]$, where $N \in \mathbb{Z}^M$ is the total number of features and $S \in N$ is a subset of features

Limitations

- Evaluates exponential number of feature perturbations
  - To mitigate this issue we approximate Shapley values using only a limited number of feature perturbations (Shapely Value Sampling, Polynomial calculation of the Shapley value based on sampling, Castro, et. at., 2009)

AXIOMS OF SHAPLEY VALUES

- **Symmetry**
  - For any function F and all $S_{\setminus i,j}$, if features i and j are interchangeable then

    $F(S \cup \{i\}) = F(S \cup \{j\})$

AXIOMS OF SHAPLEY VALUES

- **Symmetry**
  - For any function F and all $S_{\backslash i,j}$, if features i and j are interchangeable then

    $F(S \cup \{i\}) = F(S \cup \{j\})$
- **Dummy**
  - For any function F and all $S_{\backslash i}$ if i is a dummy feature, $F(S \cup \{i\}) = F(S)$

AXIOMS OF SHAPLEY VALUES

- **Symmetry**
  - For any function F and all $S_{\backslash i,j}$, if features i and j are interchangeable then $F(S \cup \{i\}) = F(S \cup \{j\})$
- **Dummy**
  - For any function F and all $S_{\backslash i}$ if i is a dummy feature, $F(S \cup \{i\}) = F(S)$
- **Additivity**
  - The attribution of the linear combination of two functions F_1 and F_2 is equal to the linear combination of attributions for each of two functions

SHAPLEY ADDITTIVE EXPLANATIONS (SHAP)

- Approximates Shapley Values by computing the conditional expectation of the contributions [Lundberg et. al. 2017]

SHAPLEY ADDITTIVE EXPLANATIONS (SHAP)

- Approximates Shapely Values by computing the conditional expectation of the contributions [Lundberg et. al. 2017]
- $x' \in \{0,1\}^M$ is a interpretable representation of input x, M is the number of interpretable features

SHAPLEY ADDITTIVE EXPLANATIONS (SHAP)

- Approximates Shapely Values by computing the conditional expectation of the contributions [Lundberg et. al. 2017]
- $x' \in \{0,1\}^M$ is a interpretable representation of input x, M is the number of interpretable features
- $g(x')$ a interpretable explanation model

SHAPLEY ADDITTIVE EXPLANATIONS (SHAP)

- Approximates Shapely Values by computing the conditional expectation of the contributions [Lundberg et. al. 2017]
- $x' \in \{0,1\}^M$ is a interpretable representation of input x, M is the number of interpretable features
- $g(x')$ a interpretable explanation model
- $x = h_x(x')$ - transforming interpretable input into original version

SHAPLEY ADDITTIVE EXPLANATIONS (SHAP)

- Approximates Shapely Values by computing the conditional expectation of the contributions [Lundberg et. al. 2017]
- $x' \in \{0,1\}^M$ is a interpretable representation of input x, M is the number of interpretable features
- $g(x')$ a interpretable explanation model
- $x = h_x(x')$ - transforming interpretable input into original version
- $f(x) = g(x') = \phi_0 + \sum_{i=1}^{M} \phi_i x'_i$ - additivity property of explanations

SHAPLEY ADDITTIVE EXPLANATIONS (SHAP)

- Let's assume that
  - S is the set of non-zero indices in z' and z_S has missing values for features that are not in S
  - $\bar{S}$ is a set of features that are not in S and $z_{\bar{S}}$ has missing values that are not in $\bar{S}$

SHAPLEY ADDITTIVE EXPLANATIONS (SHAP)

- Let's assume that
  - S is the set of non-zero indices in z' and z_S has missing values for features that are not in S
  - $\bar{S}$ is a set of features that are not in S and $z_{\bar{S}}$ has missing values that are not in $\bar{S}$
- Approximating with conditional expectation
  - $f_x(z) = f(h_x(z')) = E[f(z) \mid z_s]$ SHAP explanation model interpretable for input representation

    $\approx f([z_s, E[z_{\bar{s}}]])$ Assume interpretable model linearity

SHAPLEY ADDITTIVE EXPLANATIONS (SHAP)

- Approximating similar to Lime
  - $$L(f, g, \pi) = \sum_{z\in Z, z'\in Z'} \boxed{\pi_x{}'(z')}(f(z) - g(z'))^2$$

    Similarity metric

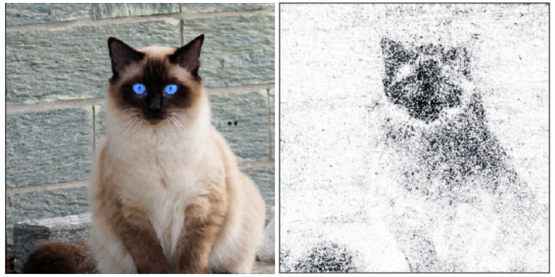
$$\pi_{x'}(z') = \frac{(M-1)}{(M\ choose\ |z'|)|z'|(M-|z'|)}$$

PRACTICAL APPLICATIONS OF ATTRIBUTION ALGORITHMS

Captum

A MODEL INTERPRETABILITY LIBRARY FOR PYTORCH

MULTIMODAL



What color are the cats eyes?

Predicted

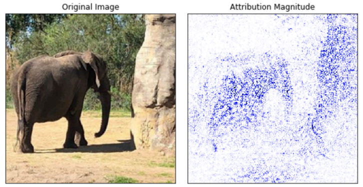
Blue (0.517)

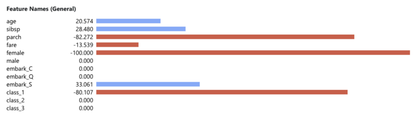
EXTENSIBLE

```
class MyAttribution(Attribution):

   def attribute(self, input, ...):
       attributions = self._compute_attrs(input, ... )
       # <Add any logic necessary for attribution>
        return attributions
```

EASY TO USE

visualize_image_attr(attr_algo.attribute(input), ...)





this movie is awful , just awful . someone bought it for me as a christmas present because they knew i liked a good horror flick . i do n't think they understood the " good " part . all i can say is next year this person is getting slipper socks from me . avoid this movie-- it makes you bitter . peace.<br /><br / >

https://arxiv.org/abs/2009.07896

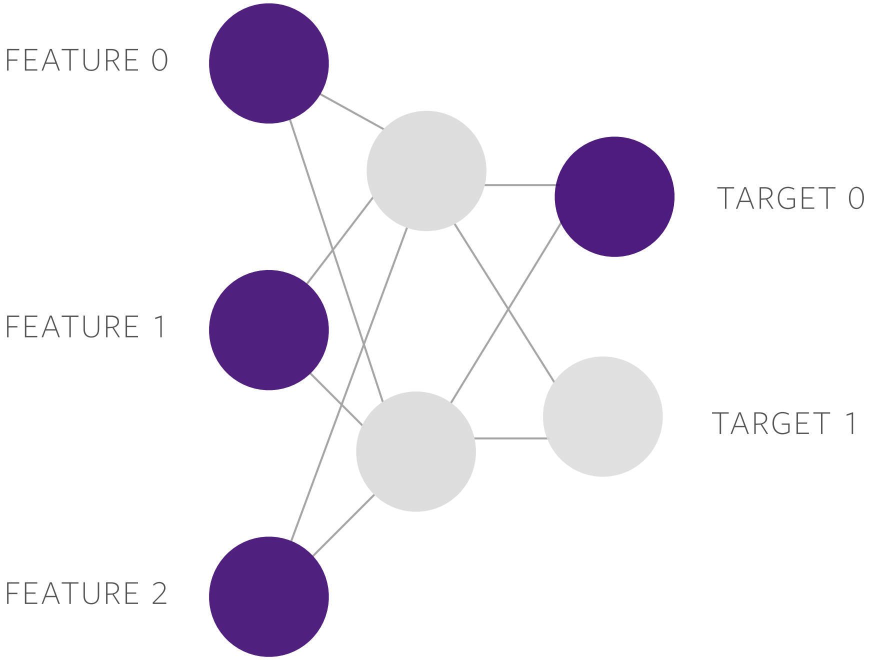
WHAT DOES THE CAPTUM LIBRARY OFFER ?

A number of gradient and perturbation-based attribution algorithms to interpret:

WHAT DOES THE CAPTUM LIBRARY OFFER ?

A number of gradient and perturbation-based attribution algorithms to interpret:

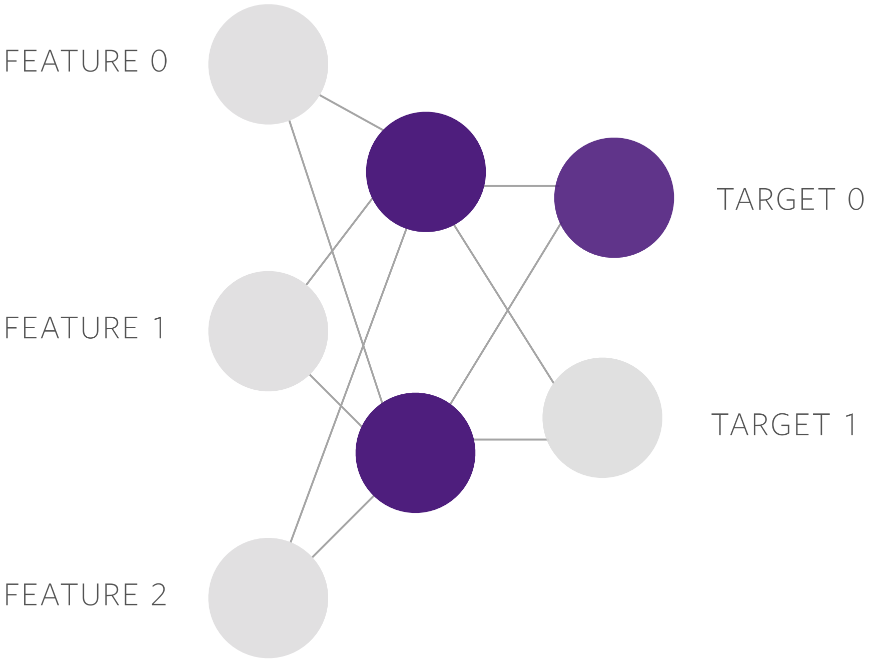
- **Primary Attribution -> Output predictions with respect to inputs**
- Layer Attribution -> Output predictions with respect to all neurons in the layers
- Neuron Attribution -> Neurons with respect to inputs



WHAT DOES THE CAPTUM LIBRARY OFFER ?

A number of gradient and perturbation-based attribution algorithms to interpret:

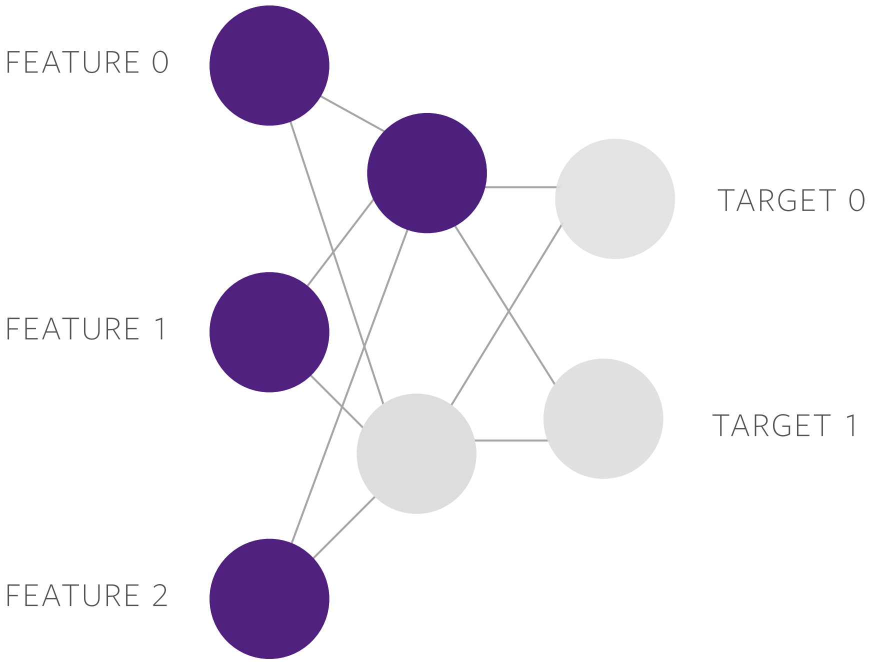
- Primary Attribution -> Output predictions with respect to inputs
- **Layer Attribution -> Output predictions with respect to all neurons in the layers**
- Neuron Attribution -> Neurons with respect to inputs



WHAT DOES THE CAPTUM LIBRARY OFFER ?

A number of gradient and perturbation-based attribution algorithms to interpret:

- Primary Attribution -> Output predictions with respect to inputs
- Layer Attribution -> Output predictions with respect to all neurons in the layers
- **Neuron Attribution -> Neurons with respect to inputs**

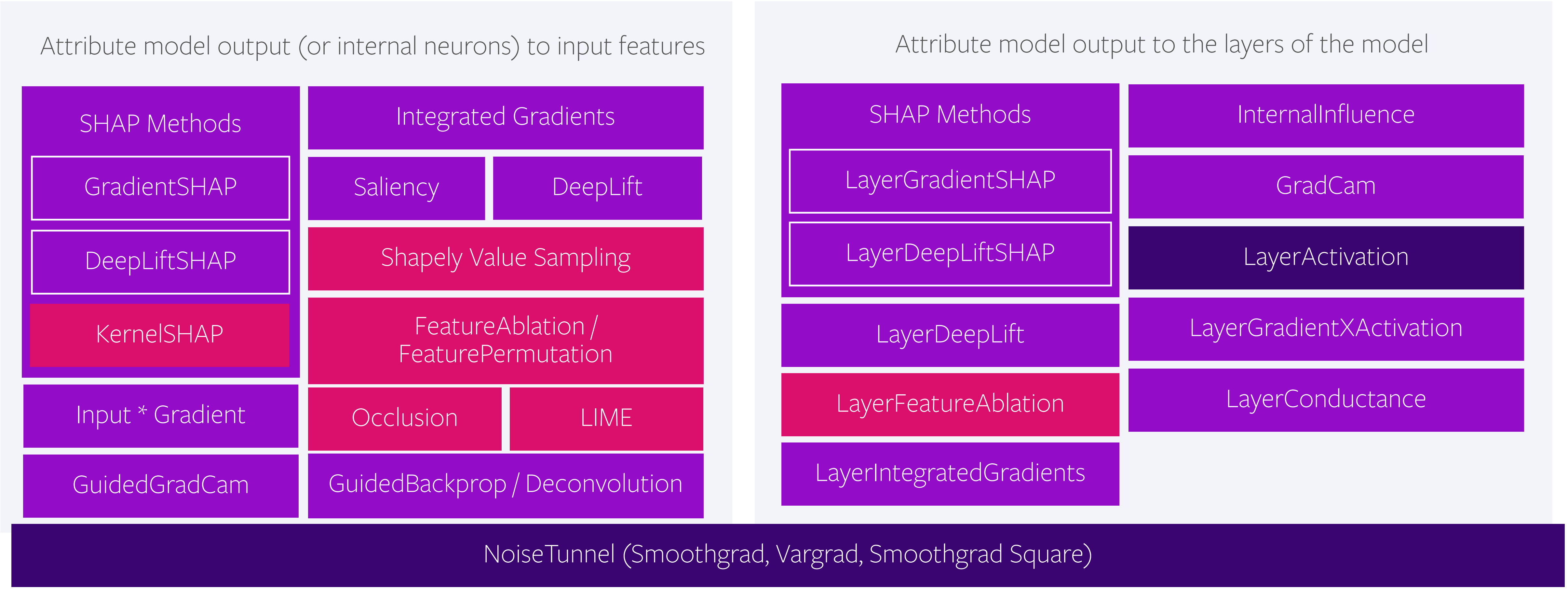


ATTRIBUTION ALGORITHMS

🟪 Gradient

🟥 Perturbation

⬛ Other



Attribute model output (or internal neurons) to input features

SHAP Methods

- GradientSHAP
- DeepLiftSHAP
- KernelSHAP

Integrated Gradients

Saliency

DeepLift

Shapely Value Sampling

FeatureAblation / FeaturePermutation

Input * Gradient

Occlusion

LIME

GuidedGradCam

GuidedBackprop / Deconvolution

Attribute model output to the layers of the model

SHAP Methods

- LayerGradientSHAP
- LayerDeepLiftSHAP

InternalInfluence

GradCam

LayerActivation

LayerDeepLift

LayerGradientXActivation

LayerFeatureAblation

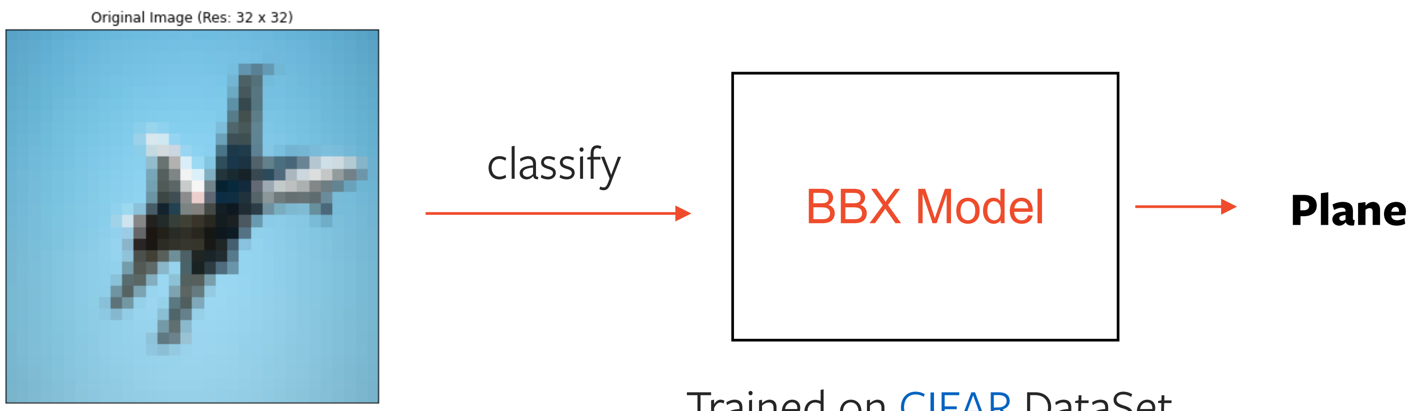
LayerConductance

LayerIntegratedGradients

NoiseTunnel (Smoothgrad, Vargrad, Smoothgrad Square)

PRACTICAL APPLICATIONS WITH CAPTUM

INTERPRETING THE PREDICTIONS OF AN IMAGE CLASSFICATION MODEL



Original Image (Res: 32 x 32)

classify

BBX Model

Plane

Trained on CIFAR DataSet

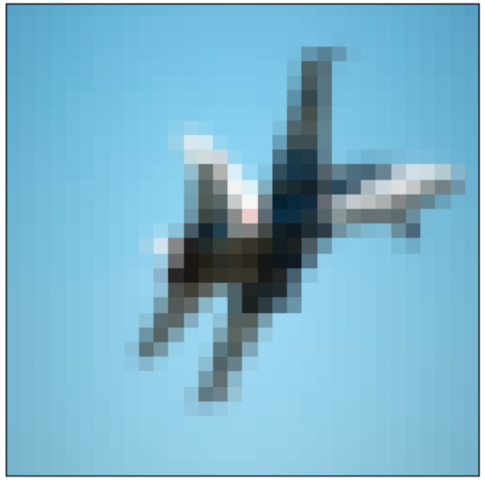
SALIENCY

Original Image (Res: 32 x 32)

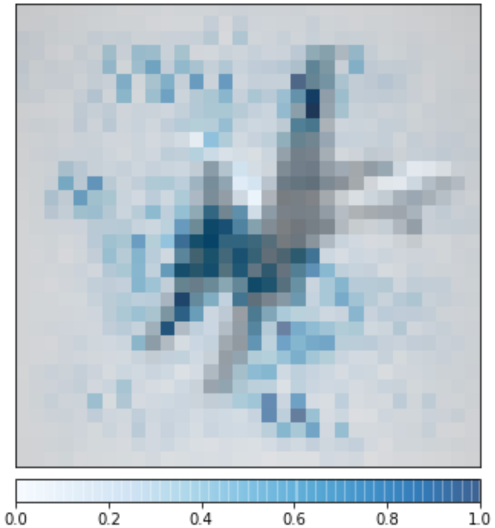
```
from captum.attr import Saliency
from captum.attr import visualization as viz

# Creating and instance of Saliency algorithm
attr_algo = Saliency(cifar_net)

# Computing the attributions for plane w.r.t. inputs
attrs = attr_algo.attribute(image,
                            target = plane_label_ind)


# Visualizing attributions
viz.visualize_image_attr(attr,
                         original_image,
                         method="blended_heat_map",
                         sign="absolute_value",
                         show_colorbar=True,
                         title="Overlayed Gradient
                                Magnitudes")
```

SALIENCY

Original Image (Res: 32 x 32)



Overlayed Gradient Magnitudes



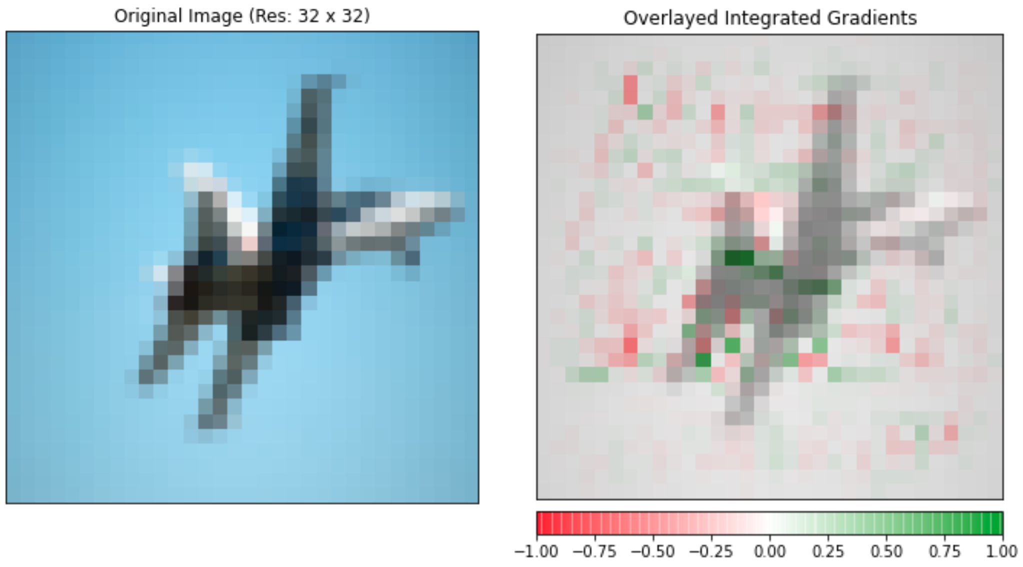
```
from captum.attr import Saliency
from captum.attr import visualization as viz

# Creating and instance of Saliency algorithm
attr_algo = Saliency(cifar_net)

# Computing the attributions for plane w.r.t. inputs
attrs = attr_algo.attribute(image,
                            target = plane_label_ind)

# Visualizing attributions
viz.visualize_image_attr(attr,
                         original_image,
                         method="blended_heat_map",
                         sign="absolute_value",
                         show_colorbar=True,
                         title="Overlayed Gradient
                                Magnitudes")
```

INTEGRATED GRADIENTS

Original Image (Res: 32 x 32)

```
from captum.attr import IntegratedGradients
from captum.attr import visualization as viz

# Creating and instance of Integrated Gradients algorithm
attr_algo = IntegratedGradients(cifar_net)

# Computing the attributions for plane w.r.t. inputs
attrs = attr_algo.attribute(image,
                            target = plane_label_ind)

# Visualizing attributions
viz.visualize_image_attr(attrs,
                         original_image,
                         method="blended_heat_map",
                         sign="all",
                         show_colorbar=True,
                         title="Overlayed Integrated
                                Gradients Magnitudes")
```

INTEGRATED GRADIENTS


Original Image (Res: 32 x 32)

Overlayed Integrated Gradients

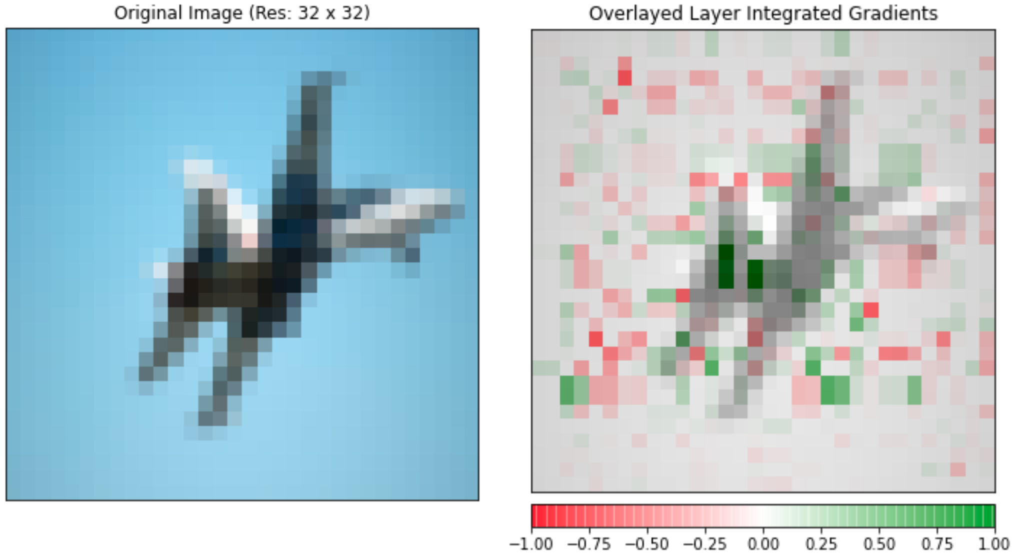
```
from captum.attr import IntegratedGradients
from captum.attr import visualization as viz

# Creating and instance of Integrated Gradients algorithm
attr_algo = IntegratedGradients(cifar_net)

# Computing the attributions for plane w.r.t. inputs
attrs = attr_algo.attribute(image,
                            target = plane_label_ind)

# Visualizing attributions
viz.visualize_image_attr(attrs,
                         original_image,
                         method="blended_heat_map",
                         sign="all",
                         show_colorbar=True,
                         title="Overlayed Integrated
                                Gradients Magnitudes")
```

LAYER INTEGRATED GRADIENTS

Original Image (Res: 32 x 32)

```
from captum.attr import LayerIntegratedGradients
from captum.attr import visualization as viz

# Creating and instance of Layer Integrated Gradients
# algorithm
attr_algo = LayerIntegratedGradients(cifar_net,
                                     cifar_net.conv1)

# Computing the attributions for plane w.r.t. conv1 layer
# output
attrs = attr_algo.attribute(image,
                            target = plane_label_ind)

# interpolate layer output in order to match input size
attrs = LayerAttribution.interpolate(attrs, (32,32))

# Visualizing attributions
viz.visualize_image_attr(attrs,
                         original_image,
                         method="blended_heat_map",
                         sign="all",
                         show_colorbar=True,
                         title="Overlayed Integrated
                                Gradients Magnitudes")
```

LAYER INTEGRATED GRADIENTS



Original Image (Res: 32 x 32)

Overlayed Layer Integrated Gradients

```
from captum.attr import LayerIntegratedGradients
from captum.attr import visualization as viz

# Creating and instance of Layer Integrated Gradients
# algorithm
attr_algo = LayerIntegratedGradients(cifar_net,
                                     cifar_net.conv1)

# Computing the attributions for plane w.r.t. conv1 layer
# output
attrs = attr_algo.attribute(image,
                            target = plane_label_ind)

# interpolate layer output in order to match input size
attrs = LayerAttribution.interpolate(attrs, (32,32))

# Visualizing attributions
viz.visualize_image_attr(attrs,
                         original_image,
                         method="blended_heat_map",
                         sign="all",
                         show_colorbar=True,
                         title="Overlayed Integrated
                                Gradients Magnitudes")
```

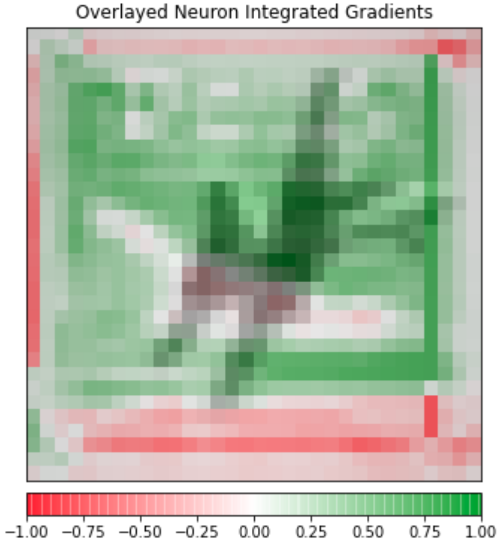
NEURON INTEGRATED GRADIENTS

Original Image (Res: 32 x 32)

```
from captum.attr import NeuronIntegratedGradients
from captum.attr import visualization as viz

# Creating and instance of Neuron Integrated Gradients
# algorithm
attr_algo = NeuronIntegratedGradients(cifar_net,
                                      cifar_net.conv1))

# Computing the attributions for the sum of all neurons
# in the output of conv1 layer w.r.t. inputs
attrs = attr_algo.attribute(image,
        neuron_selector = lambda x: x.sum(axis=(0,1,2))

# Visualizing attributions
viz.visualize_image_attr(attrs,
                         original_image,
                         method="blended_heat_map",
                         sign="all",
                         show_colorbar=True,
                         title="Overlayed Integrated
                                Gradients Magnitudes")
```

NEURON INTEGRATED GRADIENTS

Original Image (Res: 32 x 32)


Overlayed Neuron Integrated Gradients

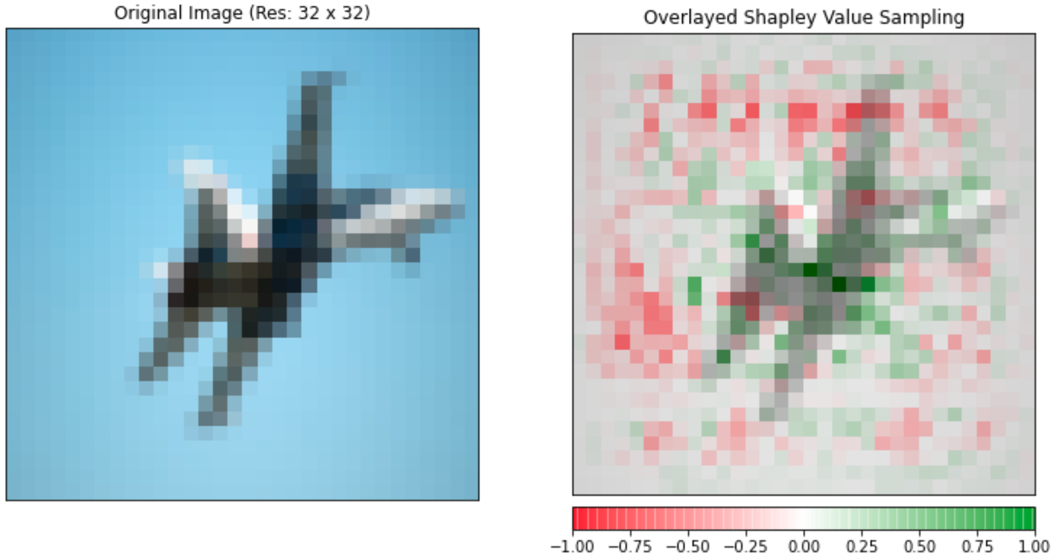
```
from captum.attr import NeuronIntegratedGradients
from captum.attr import visualization as viz

# Creating and instance of Neuron Integrated Gradients
# algorithm
attr_algo = NeuronIntegratedGradients(cifar_net,
                                      cifar_net.conv1))

# Computing the attributions for the sum of all neurons
# in the output of conv1 layer w.r.t. inputs
attrs = attr_algo.attribute(image,
        neuron_selector = lambda x: x.sum(axis=(0,1,2))

# Visualizing attributions
viz.visualize_image_attr(attrs,
                         original_image,
                         method="blended_heat_map",
                         sign="all",
                         show_colorbar=True,
                         title="Overlayed Integrated
                                Gradients Magnitudes")
```

SHAPLEY VALUE SAMPLING

Original Image (Res: 32 x 32)

```
from captum.attr import ShapleyValueSampling
from captum.attr import visualization as viz


# Creating and instance of Shapley Value Sampling algorithm
attr_algo = ShapleyValueSampling(cifar_net)


# Computing the attributions for plane w.r.t. inputs
attrs = attr_algo.attribute(image,
                            target = plane_label_ind)

# Visualizing attributions
viz.visualize_image_attr(attrs,
                         original_image,
                         method="blended_heat_map",
                         sign="all",
                         show_colorbar=True,
                         title="Overlayed Shapley
                                Value Sampling")
```

SHAPLEY VALUE SAMPLING


Original Image (Res: 32 x 32)

Overlayed Shapley Value Sampling

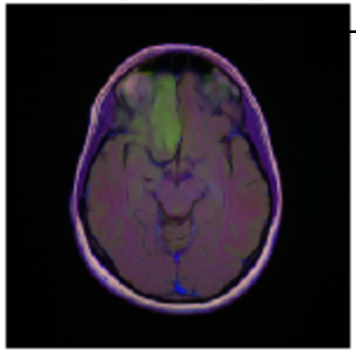
```
from captum.attr import ShapleyValueSampling
from captum.attr import visualization as viz


# Creating and instance of Shapley Value Sampling algorithm
attr_algo = ShapleyValueSampling(cifar_net)


# Computing the attributions for plane w.r.t. inputs
attrs = attr_algo.attribute(image,
                            target = plane_label_ind)


# Visualizing attributions
viz.visualize_image_attr(attrs,
                         original_image,
                         method="blended_heat_map",
                         sign="all",
                         show_colorbar=True,
                         title="Overlayed Shapley
                                Value Sampling")
```

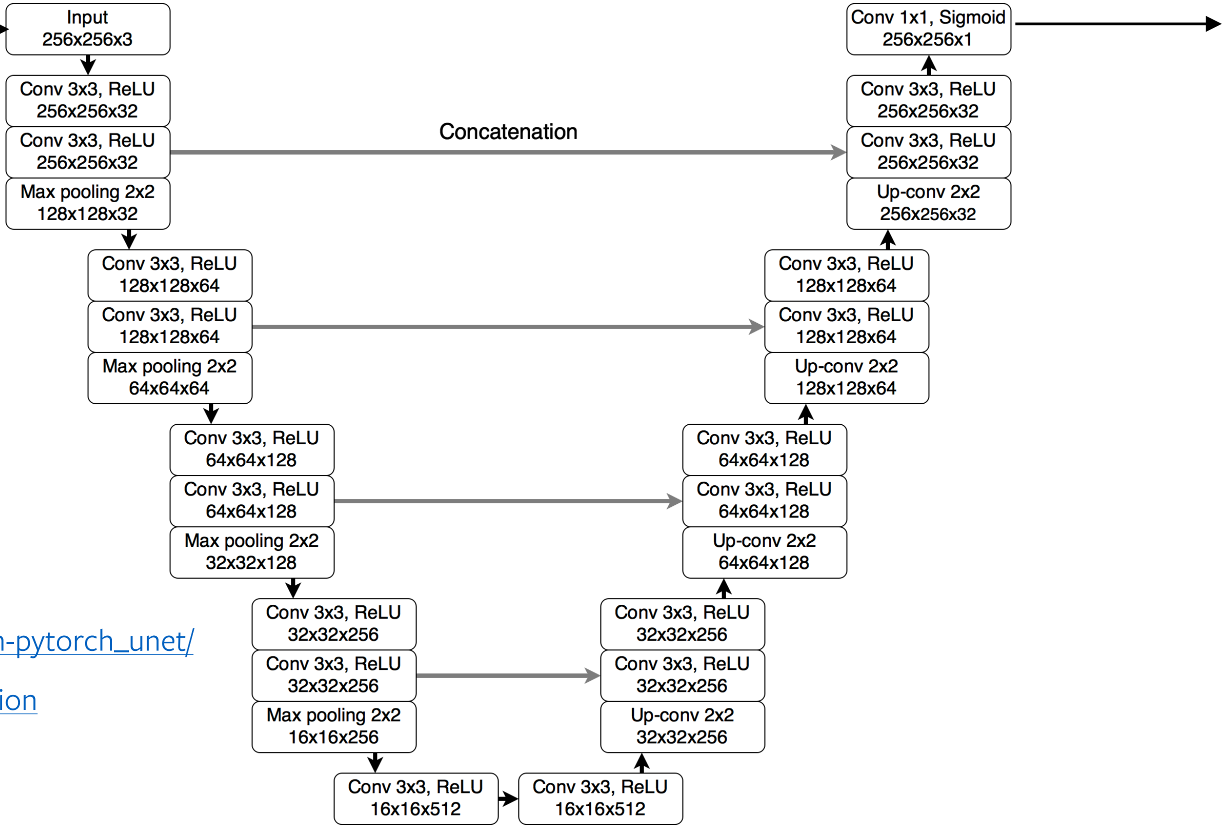
```
attributions = Attribution(forward_func, ...).attribute(input, ...)
```

USING CAPTUM FOR BRAIN MRI SEGMENTATION

U-NET MODEL FOR BRAIN MRI ABNORMALITY SEGMENTATION

Original Image





Input 256x256x3

Conv 3x3, ReLU 256x256x32

Conv 3x3, ReLU 256x256x32

Max pooling 2x2 128x128x32

Conv 3x3, ReLU 128x128x64

Conv 3x3, ReLU 128x128x64

Max pooling 2x2 64x64x64

Conv 3x3, ReLU 64x64x128

Conv 3x3, ReLU 64x64x128

Max pooling 2x2 32x32x128

Conv 3x3, ReLU 32x32x256

Conv 3x3, ReLU 32x32x256

Max pooling 2x2 16x16x256

Conv 3x3, ReLU 16x16x512

Conv 3x3, ReLU 16x16x512

Up-conv 2x2 32x32x256

Conv 3x3, ReLU 32x32x256

Conv 3x3, ReLU 32x32x256

Up-conv 2x2 64x64x128

Conv 3x3, ReLU 64x64x128

Conv 3x3, ReLU 64x64x128

Up-conv 2x2 128x128x64

Conv 3x3, ReLU 128x128x64

Conv 3x3, ReLU 128x128x64

Up-conv 2x2 256x256x32

Conv 3x3, ReLU 256x256x32

Conv 3x3, ReLU 256x256x32

Conv 1x1, Sigmoid 256x256x1

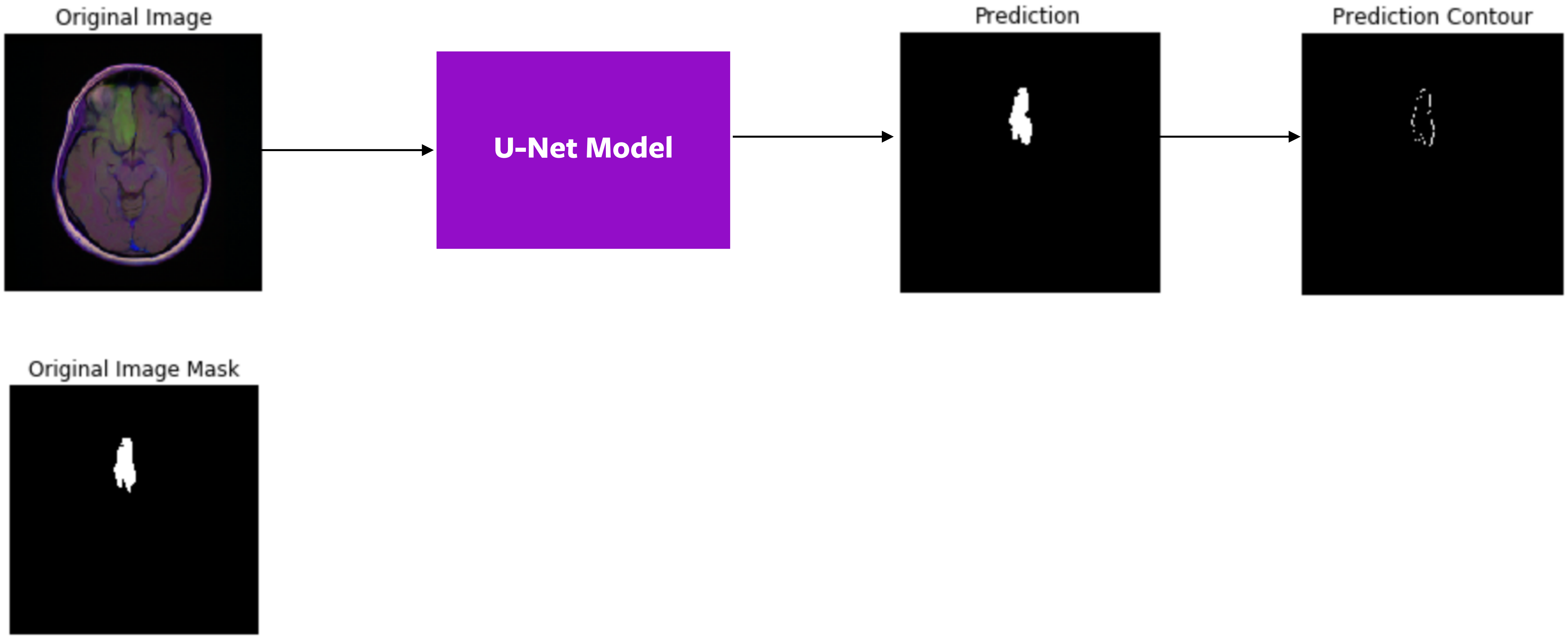
Concatenation

Prediction

Model Link: https://pytorch.org/hub/mateuszbuda_brain-segmentation-pytorch_unet/

Dataset Link: https://www.kaggle.com/mateuszbuda/lgg-mri-segmentation

U-NET MODEL FOR BRAIN MRI ABNORMALITY SEGMENTATION



Ground Truth

INTEGRATED GRADIENTS FOR MRI ABNORMALITY SEGMENTATION

- Loading U-Net model trained on Brain MRI images

```python
import torch


# load u-net model from torch.hub
unet_model = torch.hub.load('mateuszbuda' \
        '/brain-segmentation-pytorch',
        'unet',
         in_channels=3, out_channels=1,
         init_features=32, pretrained=True)
```

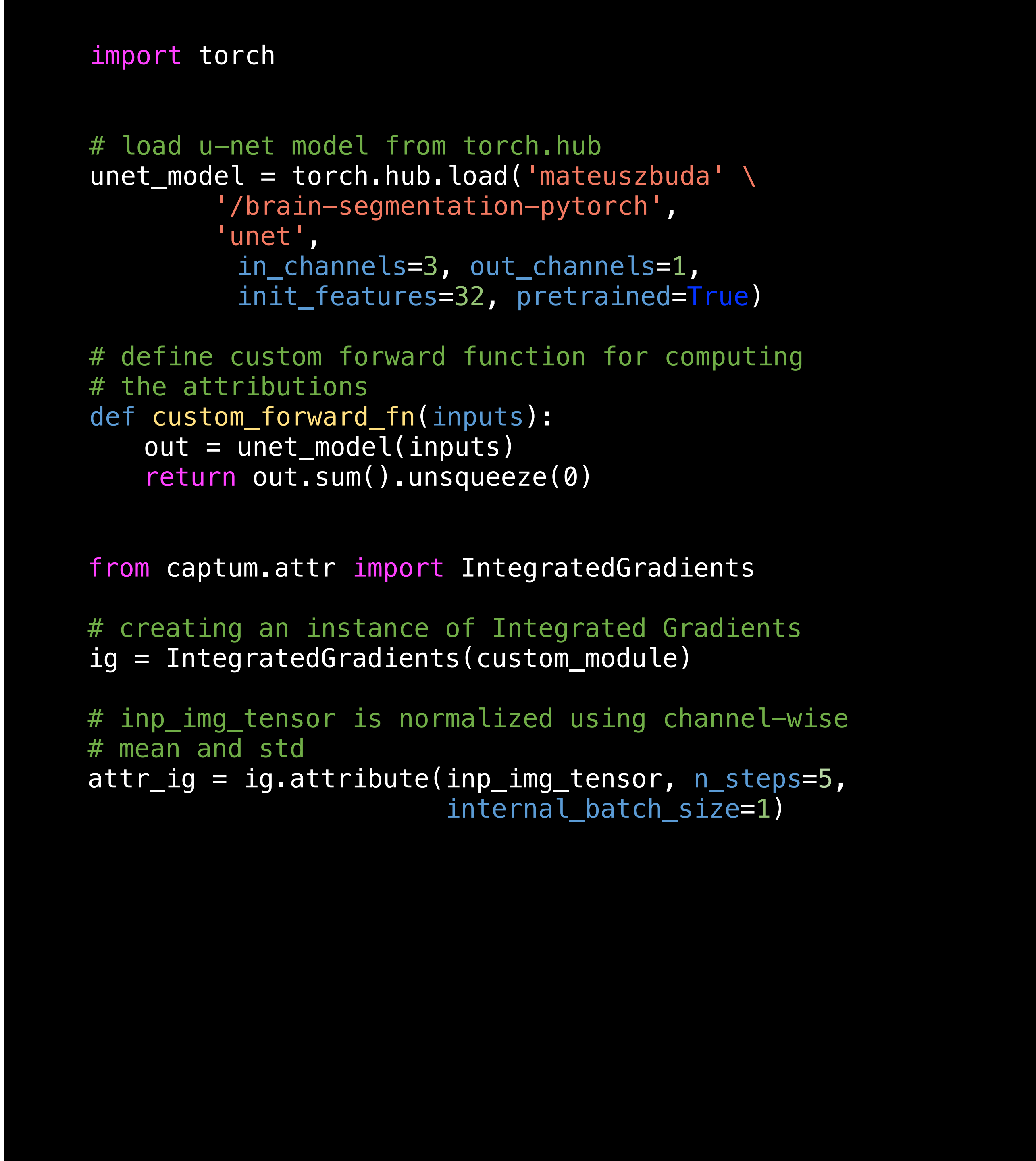
INTEGRATED GRADIENTS FOR MRI ABNORMALITY SEGMENTATION

- Loading U-Net model trained on Brain MRI images
- Summarizing the output of the prediction using custom forward function

```
import torch


# load u-net model from torch.hub
unet_model = torch.hub.load('mateuszbuda' \
        '/brain-segmentation-pytorch',
        'unet',
         in_channels=3, out_channels=1,
         init_features=32, pretrained=True)

# define custom forward function for computing
# the attributions
def custom_forward_fn(inputs):
    out = unet_model(inputs)
    return out.sum().unsqueeze(0)
```

INTEGRATED GRADIENTS FOR MRI ABNORMALITY SEGMENTATION

- Loading U-Net model trained on Brain MRI images
- Summarizing the output of the prediction using custom forward function
- Compute Integrated Gradients for the prediction segment



```python
import torch


# load u-net model from torch.hub
unet_model = torch.hub.load('mateuszbuda' \
        '/brain-segmentation-pytorch',
        'unet',
         in_channels=3, out_channels=1,
         init_features=32, pretrained=True)

# define custom forward function for computing
# the attributions
def custom_forward_fn(inputs):
    out = unet_model(inputs)
    return out.sum().unsqueeze(0)


from captum.attr import IntegratedGradients

# creating an instance of Integrated Gradients
ig = IntegratedGradients(custom_module)

# inp_img_tensor is normalized using channel-wise
# mean and std
attr_ig = ig.attribute(inp_img_tensor, n_steps=5,
                       internal_batch_size=1)
```

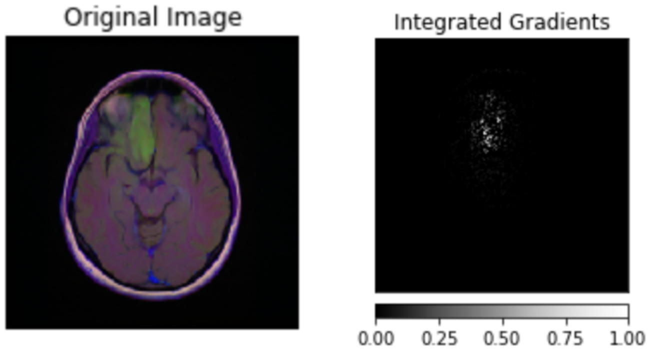
INTEGRATED GRADIENTS FOR MRI ABNORMALITY SEGMENTATION

- Loading U-Net model trained on Brain MRI images
- Summarizing the output of the prediction using custom forward function
- Compute Integrated Gradients for the prediction segment
- Visualize Attributions

```
import torch


# load u-net model from torch.hub
unet_model = torch.hub.load('mateuszbuda' \
        '/brain-segmentation-pytorch',
        'unet',
         in_channels=3, out_channels=1,
         init_features=32, pretrained=True)

# define custom forward function for computing
# the attributions
def custom_forward_fn(inputs):
    out = unet_model(inputs)
    return out.sum().unsqueeze(0)


from captum.attr import IntegratedGradients

# creating an instance of Integrated Gradients
ig = IntegratedGradients(custom_module)

# inp_img_tensor is normalized using channel-wise
# mean and std
attr_ig = ig.attribute(inp_img_tensor, n_steps=5,
                       internal_batch_size=1)


from captum.attr import visualization as viz

# visualize attributions
viz.visualize_image_attr_multiple(attr_ig,
                                  orig_img_test,
                                  methods=["heat_map"],
                                  signs=["positive"],
                                  ... )
```

INTEGRATED GRADIENTS FOR MRI ABNORMALITY SEGMENTATION

- Loading U-Net model trained on Brain MRI images
- Summarizing the output of the prediction using custom forward function
- Compute Integrated Gradients for the prediction segment



Original Image

Integrated Gradients

```python
import torch


# load u-net model from torch.hub
unet_model = torch.hub.load('mateuszbuda' \
        '/brain-segmentation-pytorch',
        'unet',
         in_channels=3, out_channels=1,
         init_features=32, pretrained=True)

# define custom forward function for computing
# the attributions
def custom_forward_fn(inputs):
    out = unet_model(inputs)
    return out.sum().unsqueeze(0)


from captum.attr import IntegratedGradients

# creating an instance of Integrated Gradients
ig = IntegratedGradients(custom_module)

# inp_img_tensor is normalized using channel-wise
# mean and std
attr_ig = ig.attribute(inp_img_tensor, n_steps=5,
                       internal_batch_size=1)


from captum.attr import visualization as viz

# visualize attributions
viz.visualize_image_attr_multiple(attr_ig,
                                  orig_img_test,
                                  methods=["heat_map"],
                                  signs=["positive"],
                                  ... )
```

GUIDED BACK PROP FOR MRI ABNORMALITY SEGMENTATION

- Loading U-Net model trained on Brain MRI images
- Summarizing the output of the prediction using custom forward function
- Compute Guided Back Prop for the prediction segment

```python
import torch


# load u-net model from torch.hub
unet_model = torch.hub.load('mateuszbuda' \
        '/brain-segmentation-pytorch',
        'unet',
         in_channels=3, out_channels=1,
         init_features=32, pretrained=True)

# define custom forward function for computing
# the attributions
def custom_forward_fn(inputs):
    out = unet_model(inputs)
    return out.sum().unsqueeze(0)


from captum.attr import GuidedBackprop

# creating an instance of Guided Back Prop
gbp = GuidedBackprop(custom_module)

# inp_img_tensor is normalized using channel-wise
# mean and std
attr_gbp = gbp.attribute(inp_img_tensor)
```

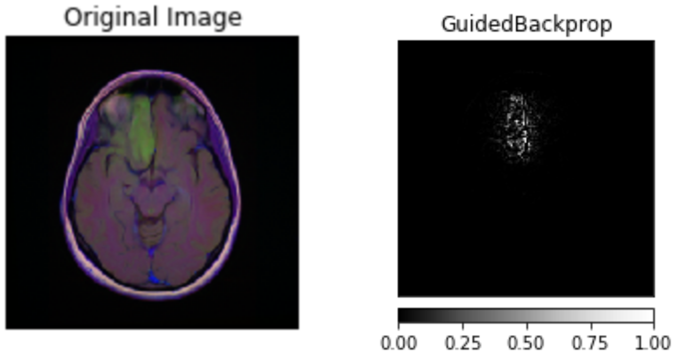
GUIDED BACK PROP FOR MRI ABNORMALITY SEGMENTATION

- Loading U-Net model trained on Brain MRI images
- Summarizing the output of the prediction using custom forward function
- Compute Guided Back Prop for the prediction segment
- Visualize Attributions

```python
import torch


# load u-net model from torch.hub
unet_model = torch.hub.load('mateuszbuda' \
        '/brain-segmentation-pytorch',
        'unet',
         in_channels=3, out_channels=1,
         init_features=32, pretrained=True)

# define custom forward function for computing
# the attributions
def custom_forward_fn(inputs):
    out = unet_model(inputs)
    return out.sum().unsqueeze(0)


from captum.attr import GuidedBackprop

# creating an instance of Guided Back Prop
gbp = GuidedBackprop(custom_module)

# inp_img_tensor is normalized using channel-wise
# mean and std
attr_gbp = gbp.attribute(inp_img_tensor)


from captum.attr import visualization as viz

# visualize attributions
viz.visualize_image_attr_multiple(attr_gbp,
                                  orig_img_test,
                                  methods=["heat_map"],
                                  signs=["positive"],
                                  ... )
```

GUIDED BACK PROP FOR MRI ABNORMALITY SEGMENTATION

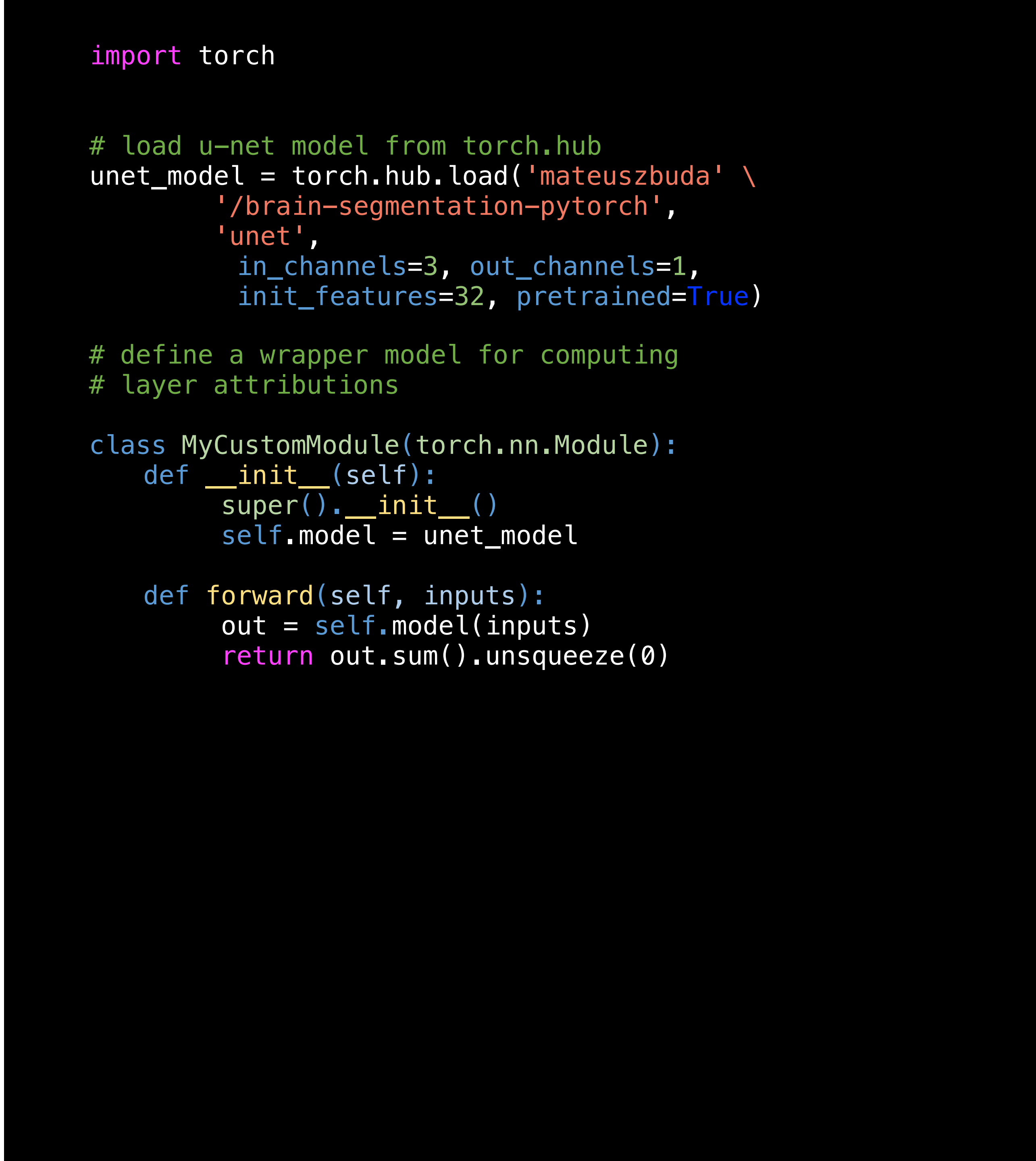
- Loading U-Net model trained on Brain MRI images
- Summarizing the output of the prediction using custom forward function
- Compute Guided Back Prop for the prediction segment



```
import torch


# load u-net model from torch.hub
unet_model = torch.hub.load('mateuszbuda' \
        '/brain-segmentation-pytorch',
        'unet',
         in_channels=3, out_channels=1,
         init_features=32, pretrained=True)

# define custom forward function for computing
# the attributions
def custom_forward_fn(inputs):
    out = unet_model(inputs)
    return out.sum().unsqueeze(0)


from captum.attr import GuidedBackprop

# creating an instance of Guided Back Prop
gbp = GuidedBackprop(custom_module)

# inp_img_tensor is normalized using channel-wise
# mean and std
attr_gbp = gbp.attribute(inp_img_tensor)


from captum.attr import visualization as viz

# visualize attributions
viz.visualize_image_attr_multiple(attr_gbp,
                                  orig_img_test,
                                  methods=["heat_map"],
                                  signs=["positive"],
                                  ... )
```

LAYER GRADCAM FOR MRI ABNORMALITY SEGMENTATION

- Loading U-Net model trained on Brain MRI images
- Wrapping unet model with a wrapper model and returning the sum of output predictions



```
import torch


# load u-net model from torch.hub
unet_model = torch.hub.load('mateuszbuda' \
        '/brain-segmentation-pytorch',
        'unet',
         in_channels=3, out_channels=1,
         init_features=32, pretrained=True)

# define a wrapper model for computing
# layer attributions

class MyCustomModule(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.model = unet_model

    def forward(self, inputs):
        out = self.model(inputs)
        return out.sum().unsqueeze(0)
```

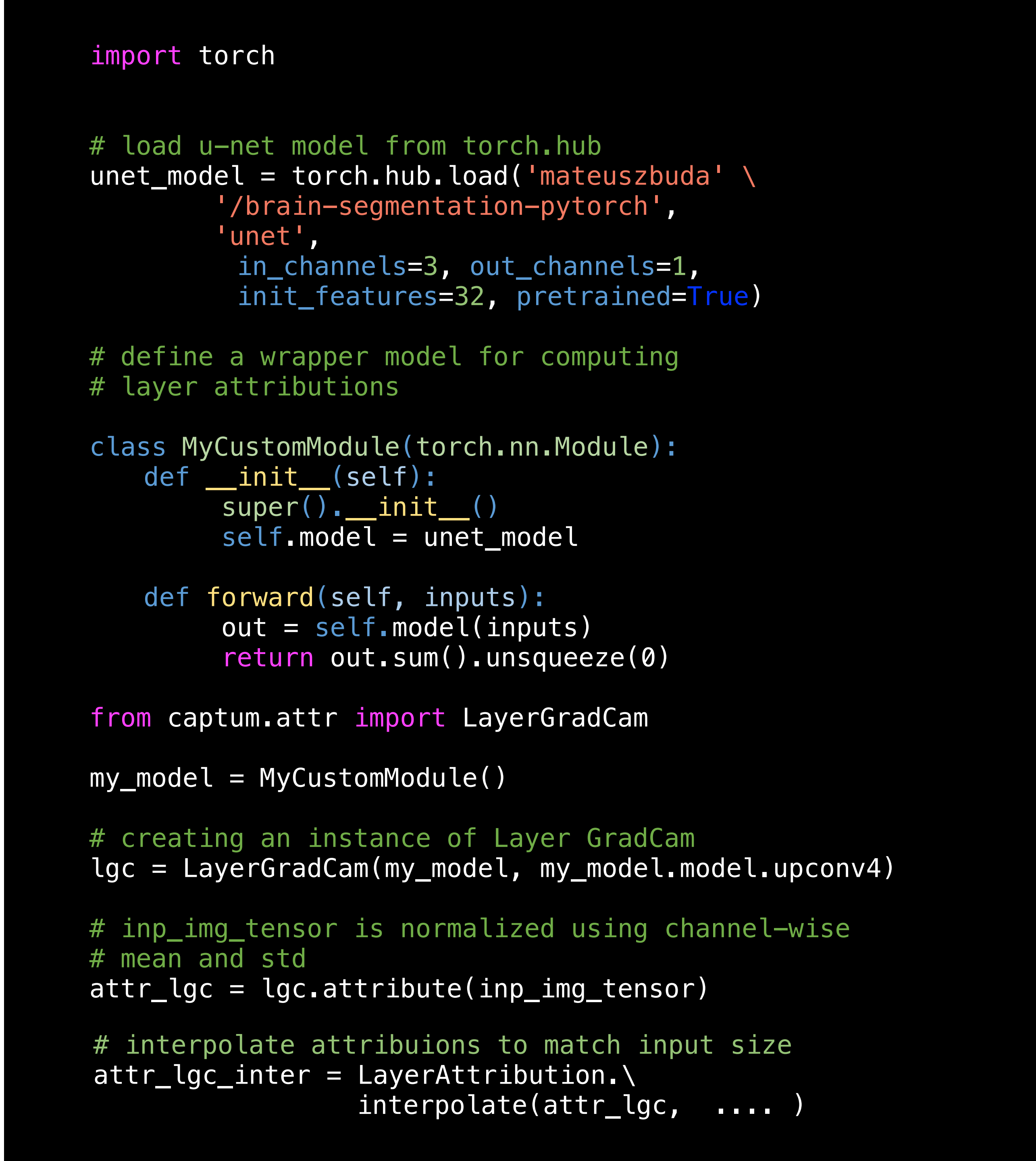
LAYER GRADCAM FOR MRI ABNORMALITY SEGMENTATION

- Loading U-Net model trained on Brain MRI images
- Wrapping unet model with a wrapper model and returning the sum of output predictions
- Compute Layer Grad Cam for a specific layer **`upconv4’**

```
import torch


# load u-net model from torch.hub
unet_model = torch.hub.load('mateuszbuda' \
        '/brain-segmentation-pytorch',
        'unet',
         in_channels=3, out_channels=1,
         init_features=32, pretrained=True)

# define a wrapper model for computing
# layer attributions

class MyCustomModule(torch.nn.Module):
   def __init__(self):
        super().__init__()
        self.model = unet_model

   def forward(self, inputs):
        out = self.model(inputs)
        return out.sum().unsqueeze(0)

from captum.attr import LayerGradCam

my_model = MyCustomModule()

# creating an instance of Layer GradCam
lgc = LayerGradCam(my_model, my_model.model.upconv4)

# inp_img_tensor is normalized using channel-wise
# mean and std
attr_lgc = lgc.attribute(inp_img_tensor)
```

LAYER GRADCAM FOR MRI ABNORMALITY SEGMENTATION

- Loading U-Net model trained on Brain MRI images
- Summarizing the output of the prediction using custom forward function
- Compute Layer Grad Cam for a specific layer **`upconv4’**
- Interpolate attributions to the input size



```
import torch

# load u-net model from torch.hub
unet_model = torch.hub.load('mateuszbuda' \
        '/brain-segmentation-pytorch',
        'unet',
         in_channels=3, out_channels=1,
         init_features=32, pretrained=True)

# define a wrapper model for computing
# layer attributions

class MyCustomModule(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.model = unet_model

    def forward(self, inputs):
        out = self.model(inputs)
        return out.sum().unsqueeze(0)

from captum.attr import LayerGradCam

my_model = MyCustomModule()

# creating an instance of Layer GradCam
lgc = LayerGradCam(my_model, my_model.model.upconv4)

# inp_img_tensor is normalized using channel-wise
# mean and std
attr_lgc = lgc.attribute(inp_img_tensor)

# interpolate attribuions to match input size
attr_lgc_inter = LayerAttribution.\
                 interpolate(attr_lgc,  .... )
```

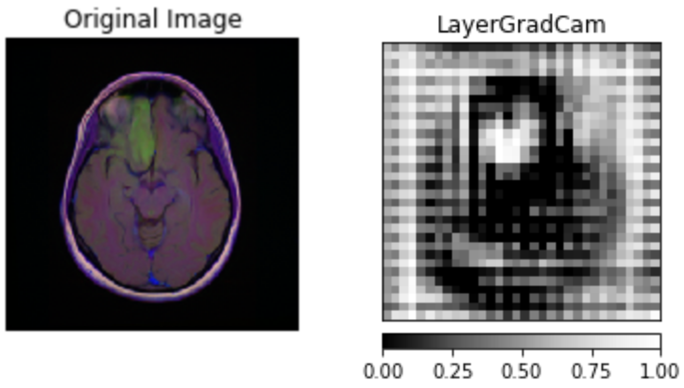
LAYER GRADCAM FOR MRI ABNORMALITY SEGMENTATION

- …
- Visualizing attribution scores for layer **`upconv4’**

```
from captum.attr import visualization as viz

# visualize attributions
viz.visualize_image_attr_multiple(attr_lgc_inter,
                                  orig_img_test,
                                  methods=["heat_map"],
                                  signs=["positive"],
                                  ... )
```

LAYER GRADCAM FOR MRI ABNORMALITY SEGMENTATION

- …
- Visualizing attribution scores for layer **`upconv4’**



```
from captum.attr import visualization as viz

# visualize attributions
viz.visualize_image_attr_multiple(attr_lgc_inter,
                                  orig_img_test,
                                  methods=["heat_map"],
                                  signs=["positive"],
                                  ... )
```

EVALUATION OF MODEL INTERPRETABILITY

EVALUATION OF MODEL INTERPRETABILITY

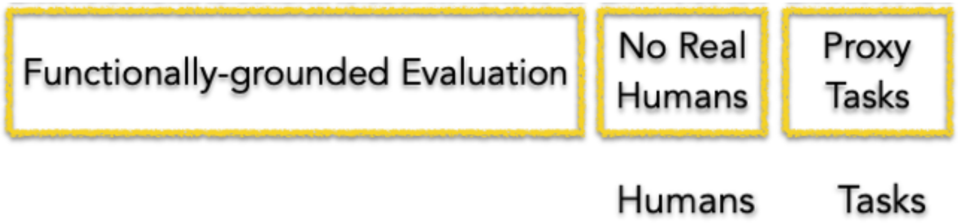
- No clear guidance on how to quantify the quality of interpretations / explanations
- Quantitative metrics are often domain specific
- Visual evaluation can be misleading or seen as a confirmation bias

A TAXONOMY OF INTERPRETABILITY EVALUATION

- Three levels of interpretability (Towards A Rigorous Science of Interpretable Machine Learning, Doshi-Velez and Kim, 2017)

A TAXONOMY OF INTERPRETABILITY EVALUATION

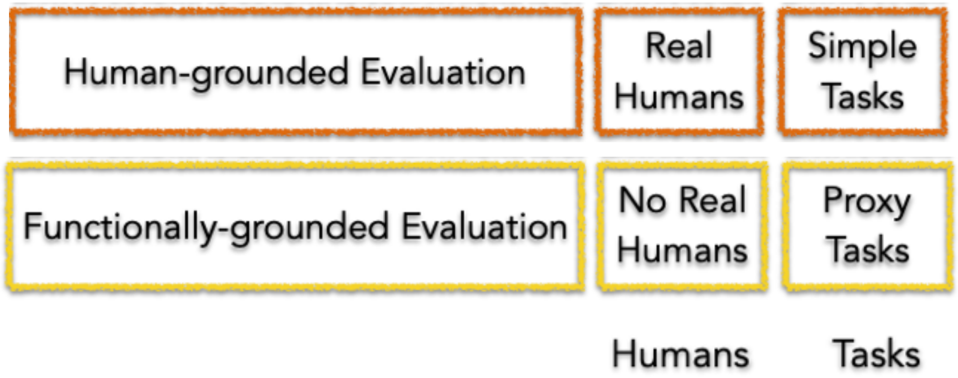
- Three levels of interpretability (Towards A Rigorous Science of Interpretable Machine Learning, Doshi-Velez and Kim, 2017)



- No humans required, quality assessment is performed by proxy tasks
- Examples of proxies are the depth of the tree, prediction performance improvement

A TAXONOMY OF INTERPRETABILITY EVALUATION

- Three levels of interpretability (Towards A Rigorous Science of Interpretable Machine Learning, Doshi-Velez and Kim, 2017)



Human-grounded Evaluation

Real Humans

Simple Tasks

Functionally-grounded Evaluation

No Real Humans

Proxy Tasks

Humans

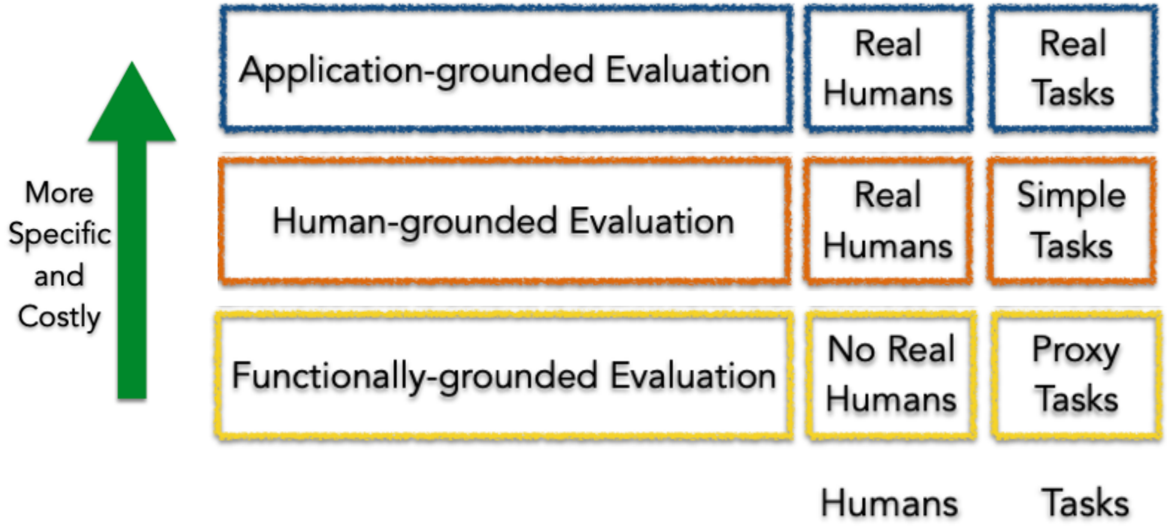
Tasks

- More strict evaluation than functional grounded one
- Human evaluation is required but not by domain experts

- No humans required, quality assessment is performed by proxy tasks
- Examples of proxies are the depth of the tree, prediction performance improvement

A TAXONOMY OF INTERPRETABILITY EVALUATION

- Three levels of interpretability (Towards A Rigorous Science of Interpretable Machine Learning, Doshi-Velez and Kim, 2017)



Application-grounded Evaluation	Real Humans	Real Tasks
Human-grounded Evaluation	Real Humans	Simple Tasks
Functionally-grounded Evaluation	No Real Humans	Proxy Tasks
	Humans	Tasks

- Validate explanations in products by end users
- Requires domain experts, good definition of how to evaluate the quality in an unbiased manner

- More strict evaluation than functional grounded one
- Human evaluation is required but not by domain experts

- No humans required, quality assessment is performed by proxy tasks
- Examples of proxies are the depth of the tree, prediction performance improvement

EVALUATION METRICS > SENSITIVITY - MAX

- Measures the sensitivity of explanations to subtle input perturbations using Monte-Carlo sampling-based approximation (On the (In)fidelity and Sensitivity of Explanations, Yeh, et. al., 2019)
- Given input $x \in \mathbb{R}^N$, perturbed input $y \in \mathbb{R}^N$, perturbation radius $r \in \mathbb{R}$, a NN function $F : \mathbb{R}^N \to \mathbb{R}$ and an explanation function $\Phi : F \times \mathbb{R}^N \to \mathbb{R}^N$

$$SENS_{MAX}(\Phi, F, x, r) = \underset{||y-x|| \leq r}{max} \frac{||\Phi(F, y) - \Phi(F, x)||}{||\Phi(F, x)||}$$

EVALUATION METRICS > INFIDELITY

- Measures mean-squared error between dot product of input perturbation and explanation and differences between the predictor function at its input and perturbed input (On the (In)fidelity and Sensitivity of Explanations, Yeh, et. al., 2019)
- Completeness property is a special case of infidelity metric

Given input $x \in \mathbb{R}^N$, a NN function $F : \mathbb{R}^N \rightarrow \mathbb{R}$, a meaningful perturbation $I \in \mathbb{R}^N$ with a probability measure μ_I

$$INFD_{\mu_I}(\Phi, F, x) = \mathbb{E}_{I \sim \mu_I}[(I^T \Phi(F, x) - (F(x) - F(x - I)))^2]$$

CONCEPT-BASED MODEL INTERPRETABILITY

CONCEPT-BASED MODEL INTERPRETABILITY

- Explaining model predictions on the basis of pre-defined concepts

Stripes

Random

CONCEPT-BASED MODEL INTERPRETABILITY

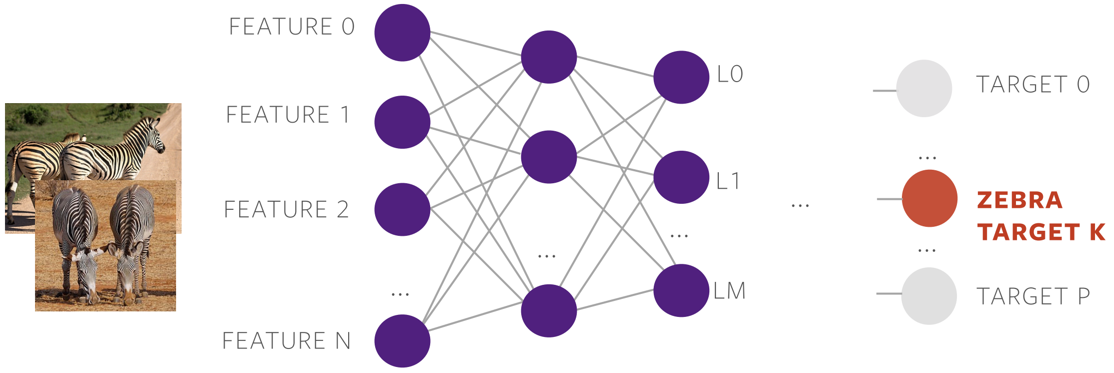
- Explaining model predictions on the basis of pre-defined concepts

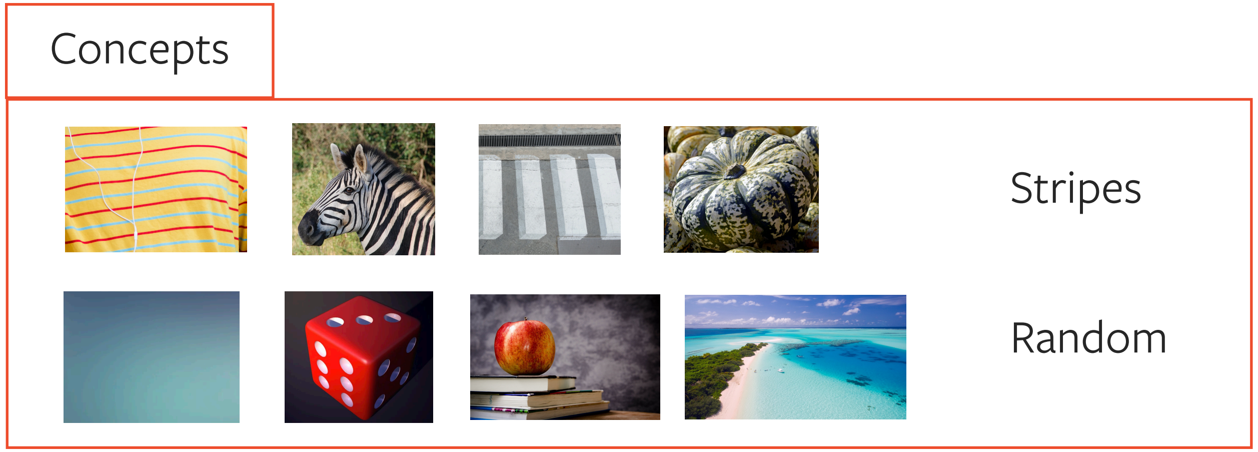
Stripes

Random

- Measures prediction sensitivity to high-level concepts

CONCEPT-BASED MODEL INTERPRETABILITY





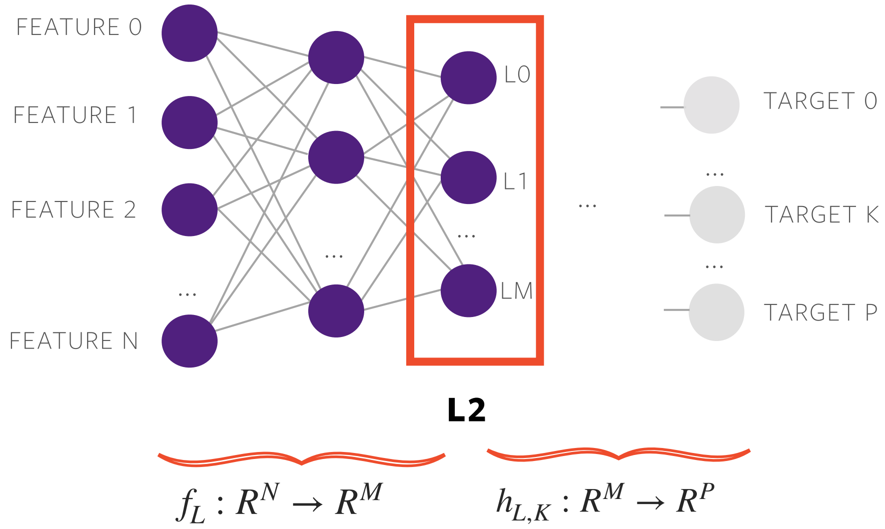
TESTING WITH CONCEPT ACTIVATION VECTORS (TCAV)

- Two Step Procedure (Kim, et.al., 2018)
  1) Concept Activation Vector (CAV) Generation
  2) Directional Sensitivity Computations

TESTING WITH CONCEPT ACTIVATION VECTORS (TCAV)

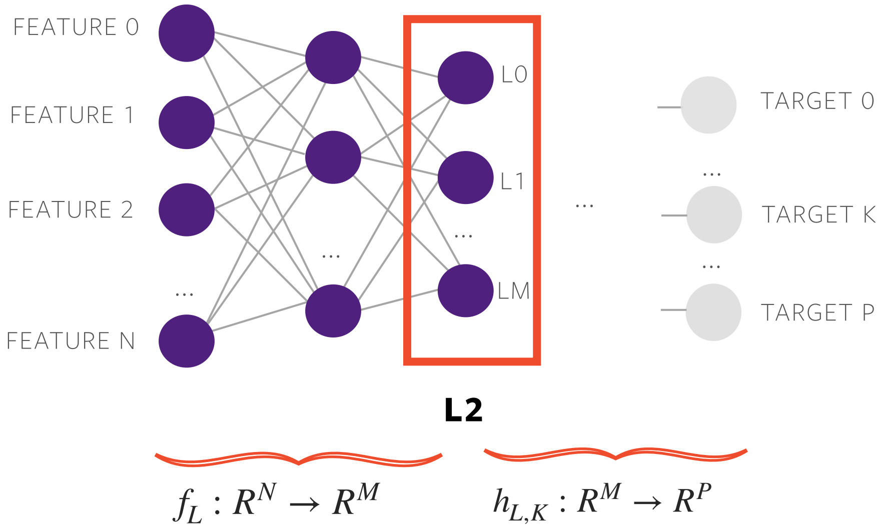
- Two Step Procedure (Kim, et.al., 2018)

  1) Concept Activation Vector (CAV) Generation

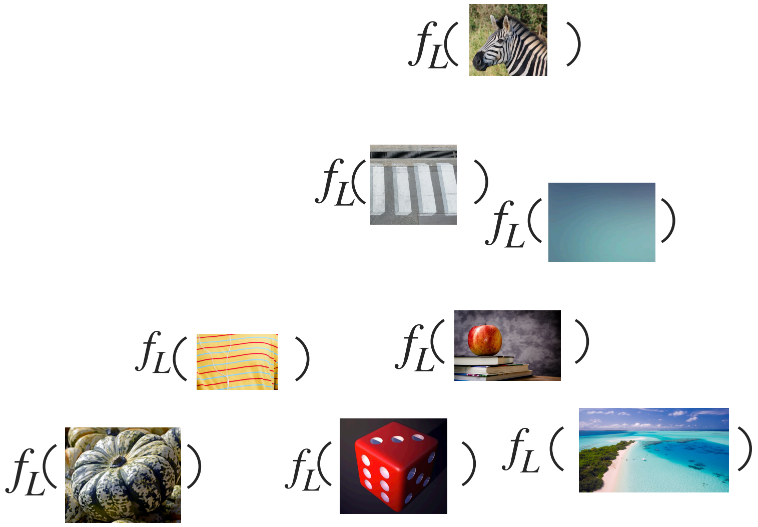


TESTING WITH CONCEPT ACTIVATION VECTORS (TCAV)

- Two Step Procedure (Kim, et.al., 2018)
  1) Concept Activation Vector (CAV) Generation

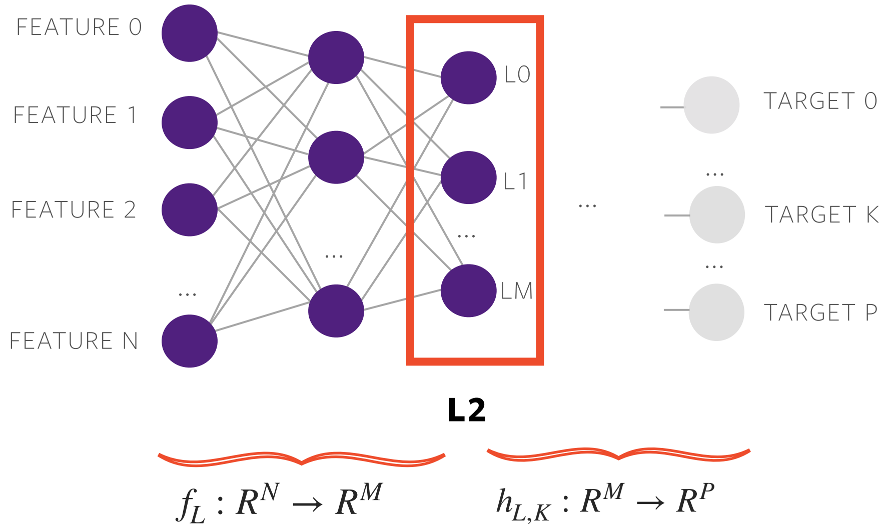


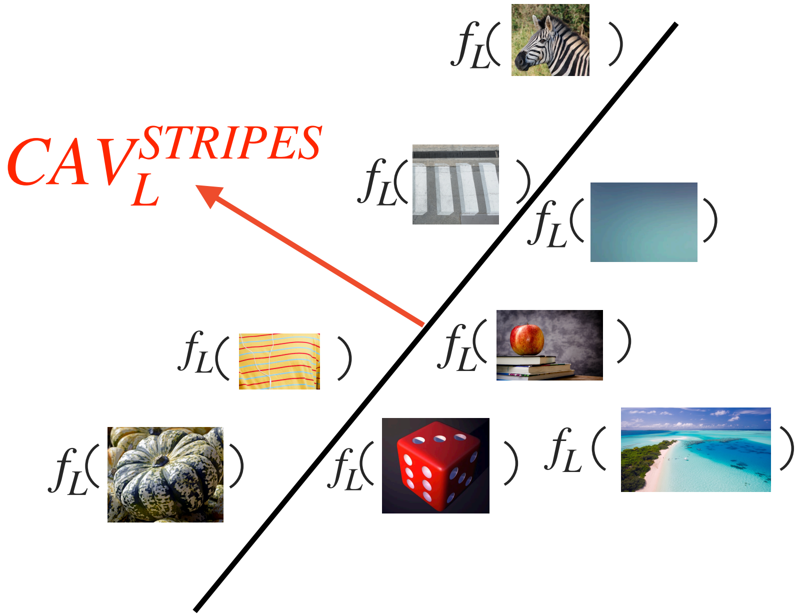
$f_L : R^N \rightarrow R^M$ $h_{L,K} : R^M \rightarrow R^P$



TESTING WITH CONCEPT ACTIVATION VECTORS (TCAV)

- Two Step Procedure (Kim, et.al., 2018)
  1) Concept Activation Vector (CAV) Generation

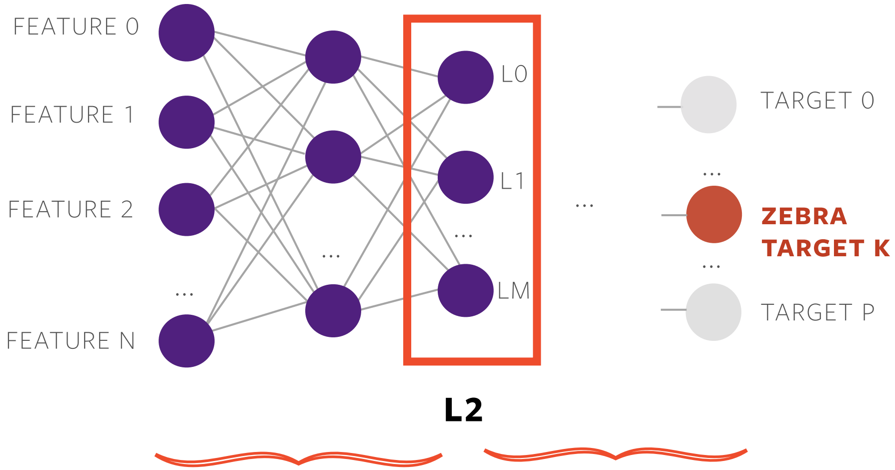




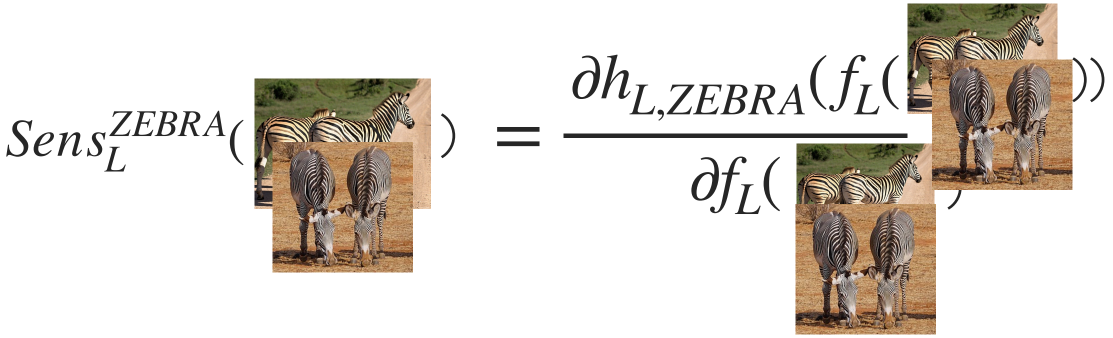
CAV is the vector orthogonal to the hyperplane of concept linear classifier

TESTING WITH CONCEPT ACTIVATION VECTORS (TCAV)

- Two Step Procedure (Kim, et.al., 2018)
  1) Concept Activation Vector (CAV) Generation
  2) **Directional Sensitivity Computations**



$f_L : R^N \rightarrow R^M$ $h_{L,K} : R^M \rightarrow R^P$



$$Sens_L^{ZEBRA}(\ ) = \frac{\partial h_{L,ZEBRA}(f_L(\ ))}{\partial f_L(\ )}$$

TESTING WITH CONCEPT ACTIVATION VECTORS (TCAV)

- Two Step Procedure (Kim, et.al., 2018)
  1) Concept Activation Vector (CAV) Generation
  2) Directional Sensitivity Computations

$CAV\_SENS^{STRIPES}_{L,ZEBRA}($ $) = CAV^{STRIPES}_{L} \, . \, Sens^{ZEBRA}_{L}($ $)$

TESTING WITH CONCEPT ACTIVATION VECTORS (TCAV)

- Two Step Procedure (Kim, et.al., 2018)
  1) Concept Activation Vector (CAV) Generation
  2) Directional Sensitivity Computations

$CAV\_SENS^{STRIPES}_{L,ZEBRA}($ $) = CAV^{STRIPES}_{L} \, . \, Sens^{ZEBRA}_{L}($ $)$

$TCAV^{STRIPES}_{L,ZEBRA}($ $) = \dfrac{\left| CAV\_SENS^{STRIPES}_{L,ZEBRA}(\right.}{}$ $\left.) > 0 \right|$ / $|$ $|$

TESTING WITH CONCEPT ACTIVATION VECTORS (TCAV)

- Two Step Procedure (Kim, et.al., 2018)
  1) Concept Activation Vector (CAV) Generation
  2) Directional Sensitivity Computations

In a general case

$$TCAV_{L,CLASS}^{CONCEPT}(inputs_{CLASS}) = \frac{|CAV\_SENS_{L,CLASS}^{CONCEPT}(inputs_{CLASS}) > 0|}{|inputs_{CLASS}|}$$

TCAV > STATISTICAL SIGNIFICANCE TESTING

- TCAV can potentially learn meaningless CAVs for a meaningful concept
- A CAV generated for a random concept can potentially be meaningful

TCAV > STATISTICAL SIGNIFICANCE TESTING

- TCAV can potentially learn meaningless CAVs for a meaningful concept
- A CAV generated for a random concept can potentially be meaningful
- Steps we can take to mitigate those issues
    - Two sided statistical significance tests
        - Against large number of random concepts
        - A meaningful concept will stand out with high TCAV score among most random concepts

TCAV > LIMITATIONS

- Concepts has to be pre-defined in advance
  - Time consuming process
- Learning meaningless CAVs
  - Statistical significance testing for multiple random concepts
    - Computationally time and memory intensive

CONCEPT-BASED MODEL INTERPRETABILITY > MORE TECHNIQUES

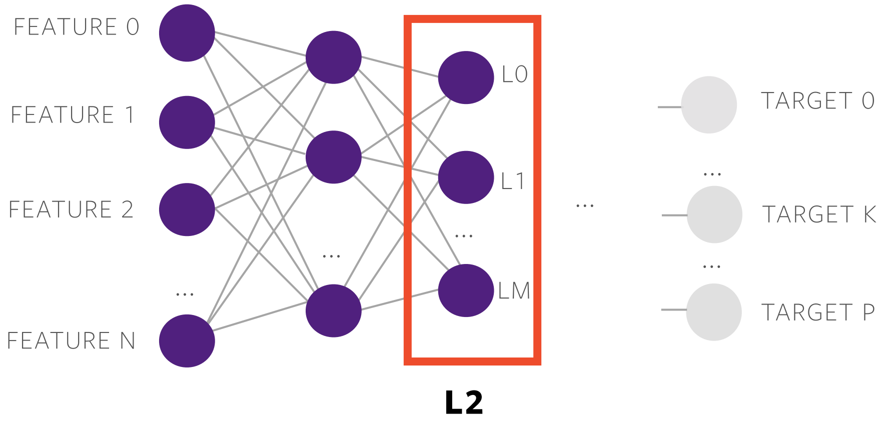
- Automatic Concept Extraction (ACE) for images (Towards Automatic Concept-based Explanations, Ghorbani et. al., 2019)
- Identifying a sufficient set of concepts that describe our prediction, ConceptSHAP (On Completeness-aware Concept-Based Explanations in Deep Neural Networks, Yeh, et. al., 2020)

MODEL COMPARISION

PRINCIPLE COMPONENT ANALYSIS (PCA)

- Visualizing high dimensional embedding spaces
- Principal Component Analysis (PCA) (LIII. On lines and planes of closest fit to systems of points in space, Pearson F.R.S, 1901)

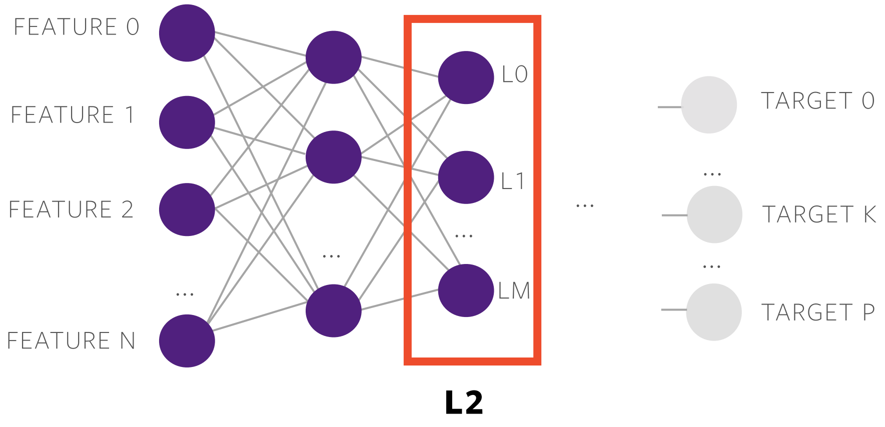
$f_L : R^N \rightarrow R^M$

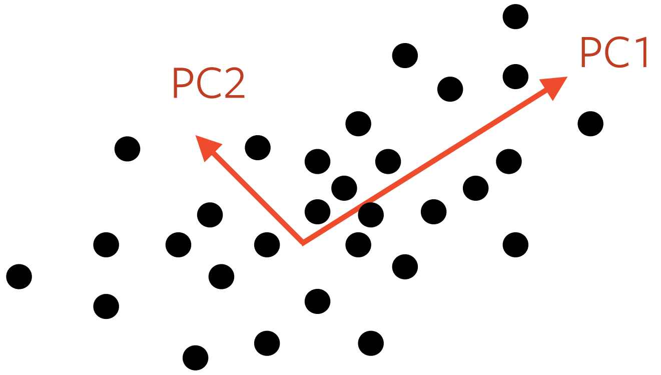


PRINCIPLE COMPONENT ANALYSIS (PCA)

- Visualizing high dimensional embedding spaces
- Principal Component Analysis (PCA) (LIII. On lines and planes of closest fit to systems of points in space, Pearson F.R.S, 1901)
- Projects high dimensional layer embedding vectors to a lower dimensional space that captures maximum variance in the data

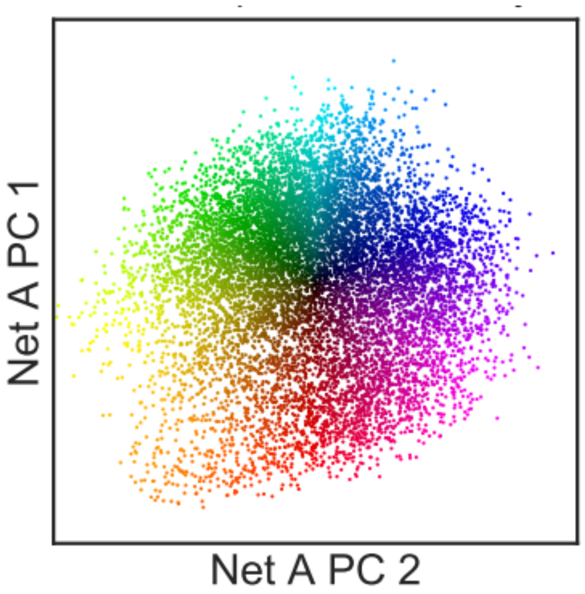
$$f_L : R^N \to R^M$$





PRINCIPLE COMPONENT ANALYSIS (PCA)

- Two Principle components for CIFAR-10 test dataset
- Colored dots are examples from test dataset

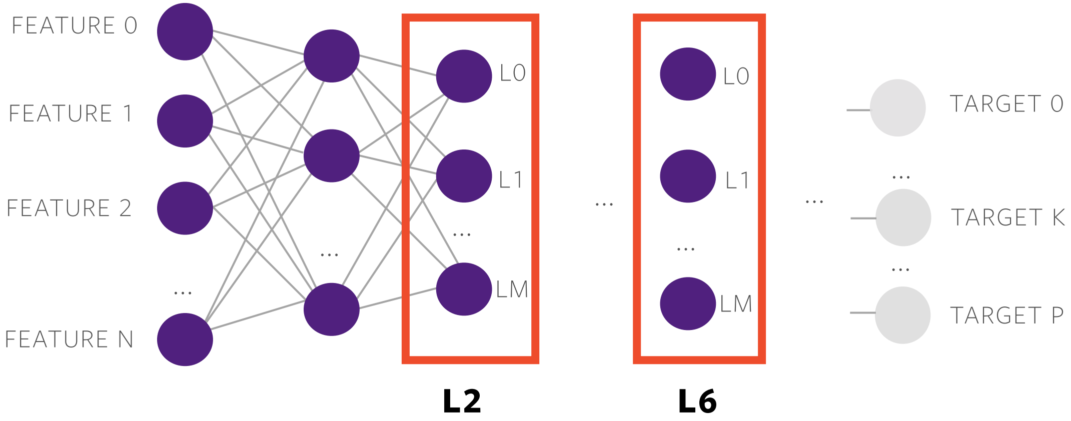


Source: Similarity of Neural Network Representations Revisited, Kornblith, et. al., 2019

CORRELATION ANALYSIS

- Comparing embedding representations for a pair of layers in a model or across multiple models

$f_L : R^N \rightarrow R^M$ $f_L : R^N \rightarrow R^M$

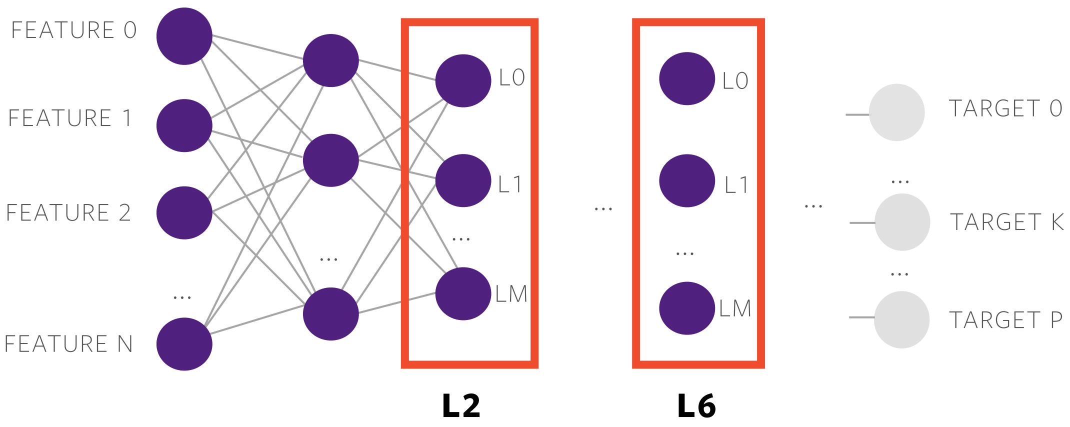


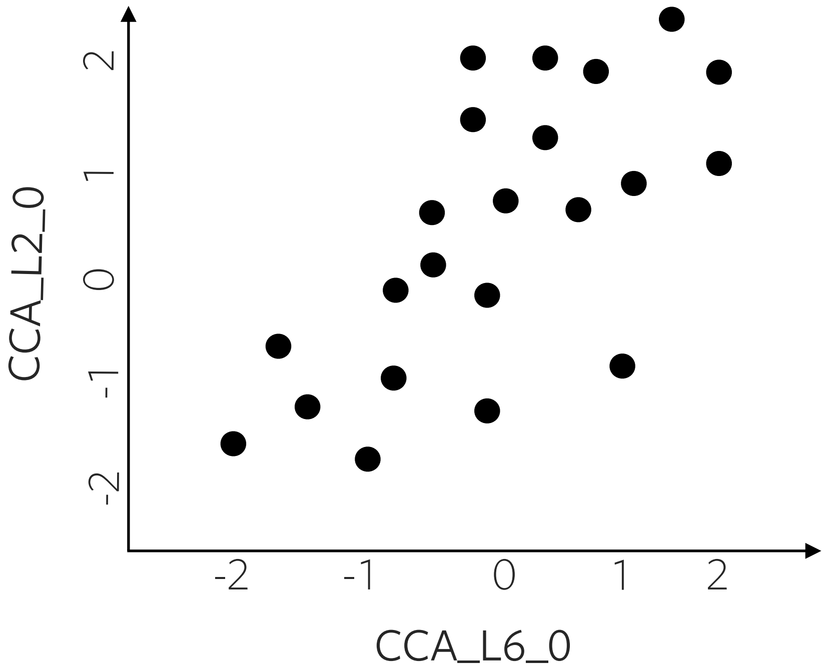
CANONICAL CORRELATION ANALYSIS (CCA)

- Comparing embedding representations for a pair of layers in a model or across multiple models
- **Canonical Correlation Analysis (CCA)** ([Relations between two sets of variates, Hotelling, 1936](#))

  Explains correlation between embedding representations of a pair of layers

$f_L : R^N \to R^M \quad f_L : R^N \to R^M$

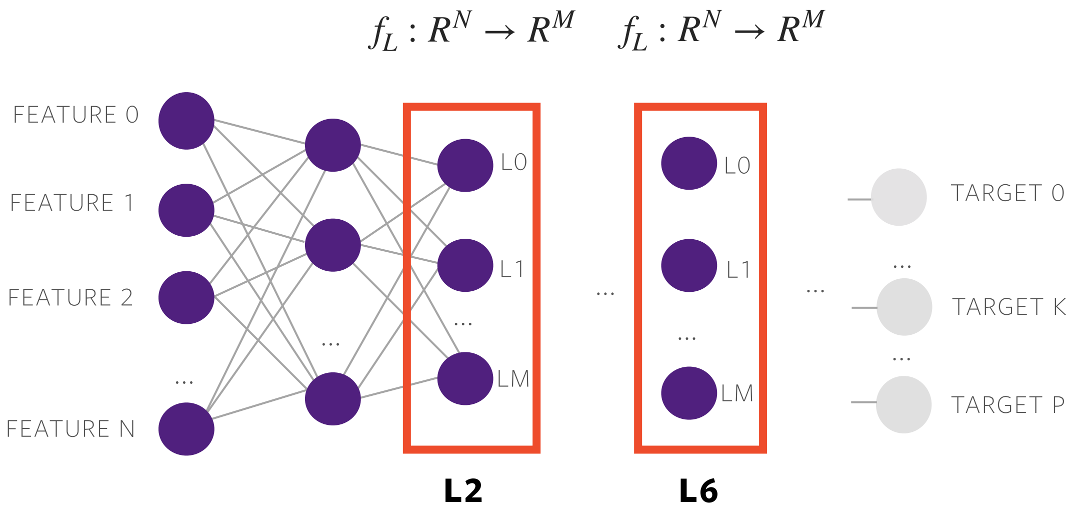




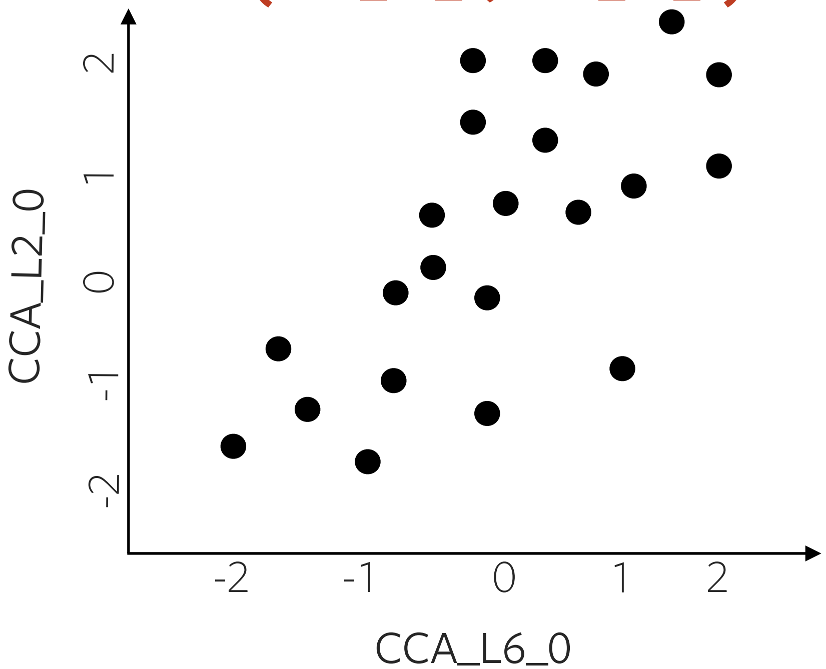
CANONICAL CORRELATION ANALYSIS (CCA)

- Comparing embedding representations for a pair of layers in a model or across multiple models
- **Canonical Correlation Analysis (CCA)** (Relations between two sets of variates, Hotelling, 1936)

  Explains correlation between embedding representations of a pair of layers



Corr(CCA_L2_0, CCA_L6_0) = 0.7



SINGULAR VECTOR CANONICAL CORRELATION ANALYSIS (SVCCA)

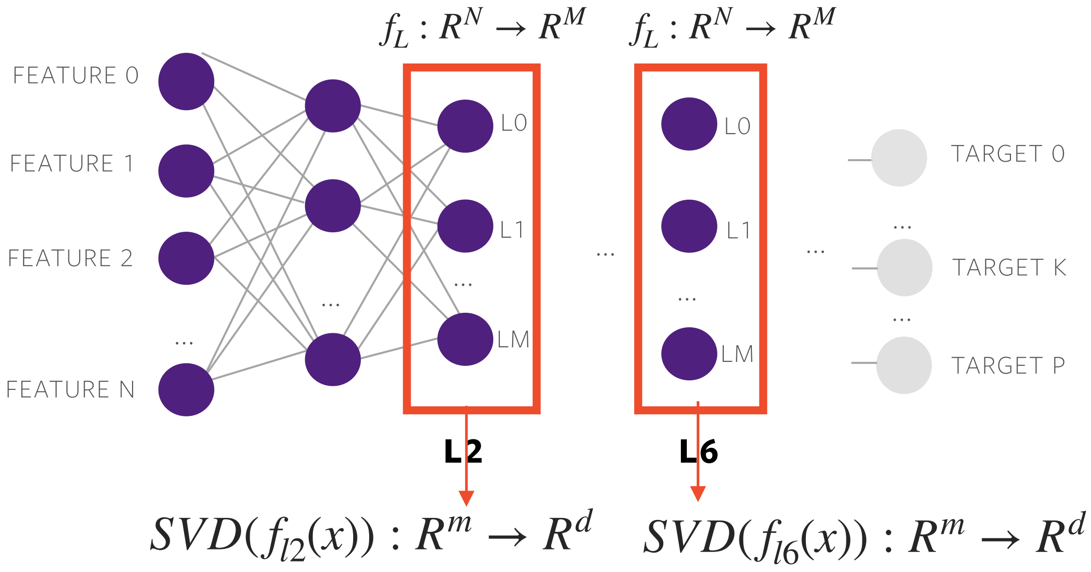
- **Singular Value Canonical Correlation Analysis (SVCCA)** (SVCCA, Raghu, et. al., 2017)

  Applies CCA on the top directions of both layers that capture the most variance

SINGULAR VECTOR CANONICAL CORRELATION ANALYSIS (SVCCA)

- **Singular Value Canonical Correlation Analysis (SVCCA)** ([SVCCA, Raghu, et. al., 2017)](SVCCA, Raghu, et. al., 2017)

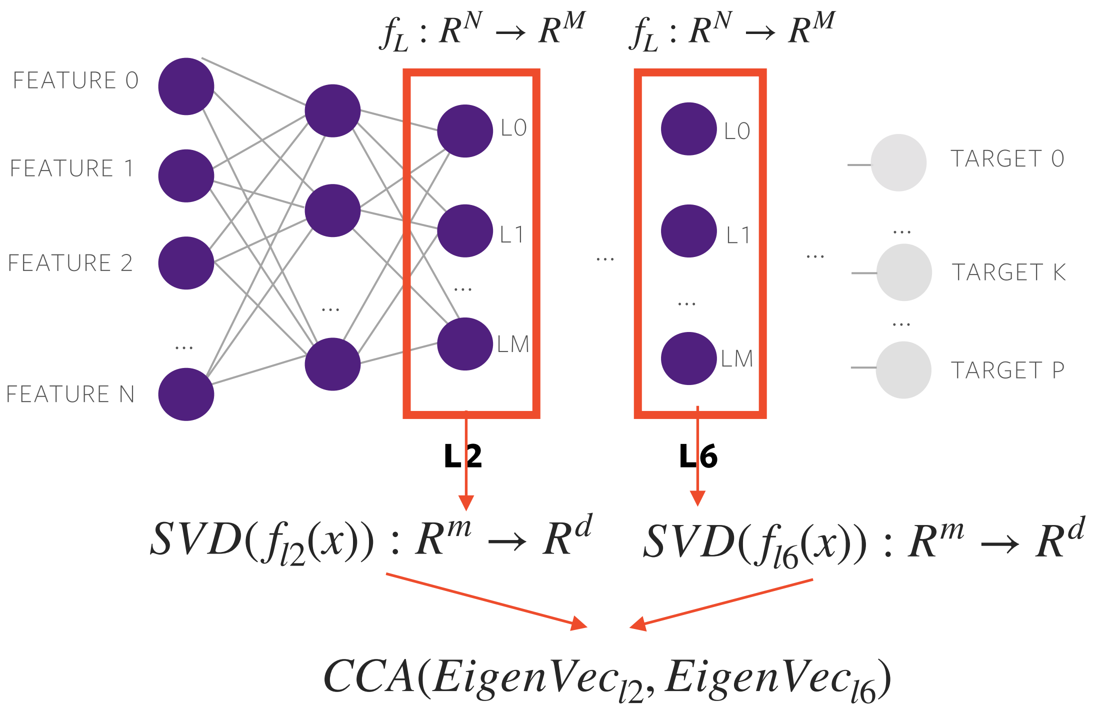
  Applies CCA on the top directions of both layers that capture the most variance


$f_L : R^N \rightarrow R^M$
$f_L : R^N \rightarrow R^M$
FEATURE 0
FEATURE 1
FEATURE 2
...
FEATURE N
...
L0
L1
...
LM
...
L0
L1
...
LM
...
TARGET 0
...
TARGET K
...
TARGET P
L2
L6
$SVD(f_{l2}(x)) : R^m \rightarrow R^d$
$SVD(f_{l6}(x)) : R^m \rightarrow R^d$

SINGULAR VECTOR CANONICAL CORRELATION ANALYSIS (SVCCA)

- **Singular Value Canonical Correlation Analysis (SVCCA)** (SVCCA, Raghu, et. al., 2017)

  Applies CCA on the top directions of both layers that capture the most variance



$f_L : R^N \to R^M$ $f_L : R^N \to R^M$

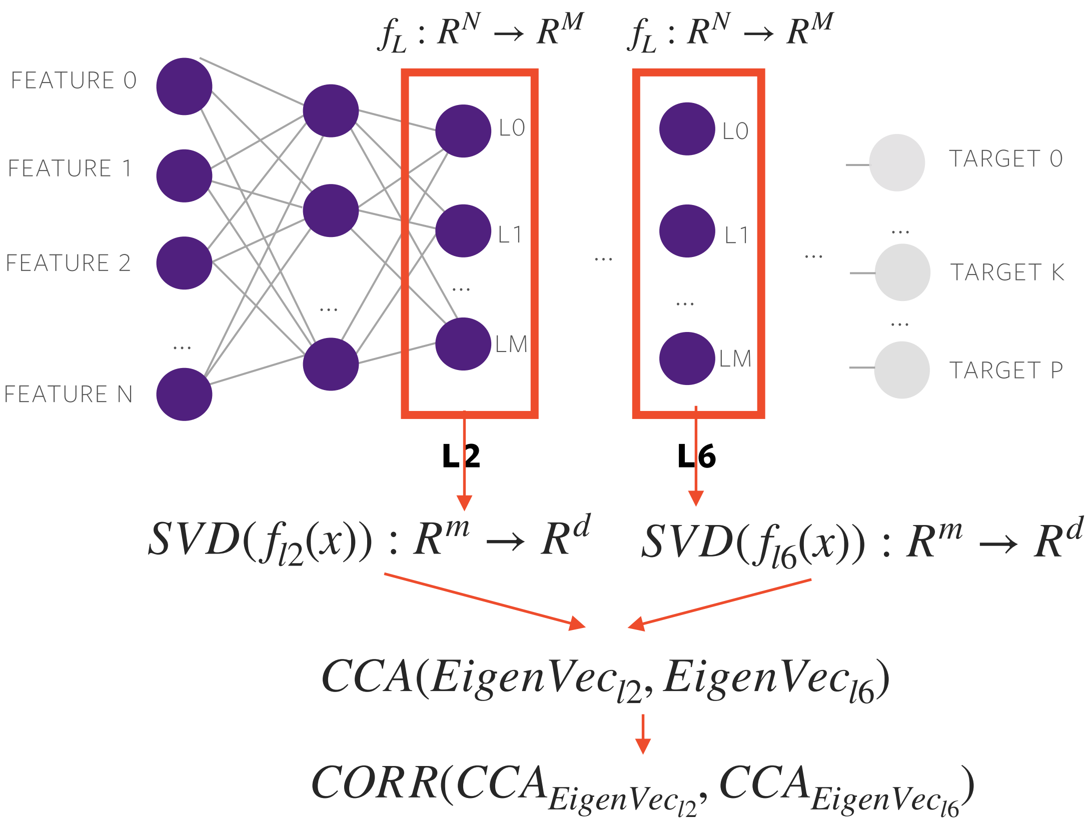
$SVD(f_{l2}(x)) : R^m \to R^d$ $SVD(f_{l6}(x)) : R^m \to R^d$

$CCA(EigenVec_{l2}, EigenVec_{l6})$

SINGULAR VECTOR CANONICAL CORRELATION ANALYSIS (SVCCA)

- **Singular Value Canonical Correlation Analysis (SVCCA)** (SVCCA, Raghu, et. al., 2017)

  Applies CCA on the top directions of both layers that capture the most variance



$f_L : R^N \rightarrow R^M$ $\quad f_L : R^N \rightarrow R^M$

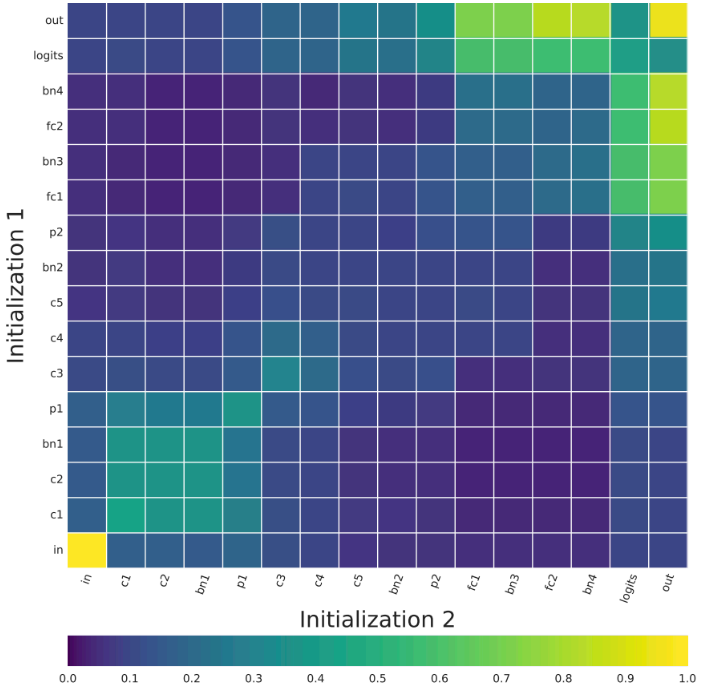
$SVD(f_{l2}(x)) : R^m \rightarrow R^d$ $\quad SVD(f_{l6}(x)) : R^m \rightarrow R^d$

$$CCA(EigenVec_{l2}, EigenVec_{l6})$$

$$CORR(CCA_{EigenVec_{l2}}, CCA_{EigenVec_{l6}})$$

SINGULAR VECTOR CANONICAL CORRELATION ANALYSIS (SVCCA)

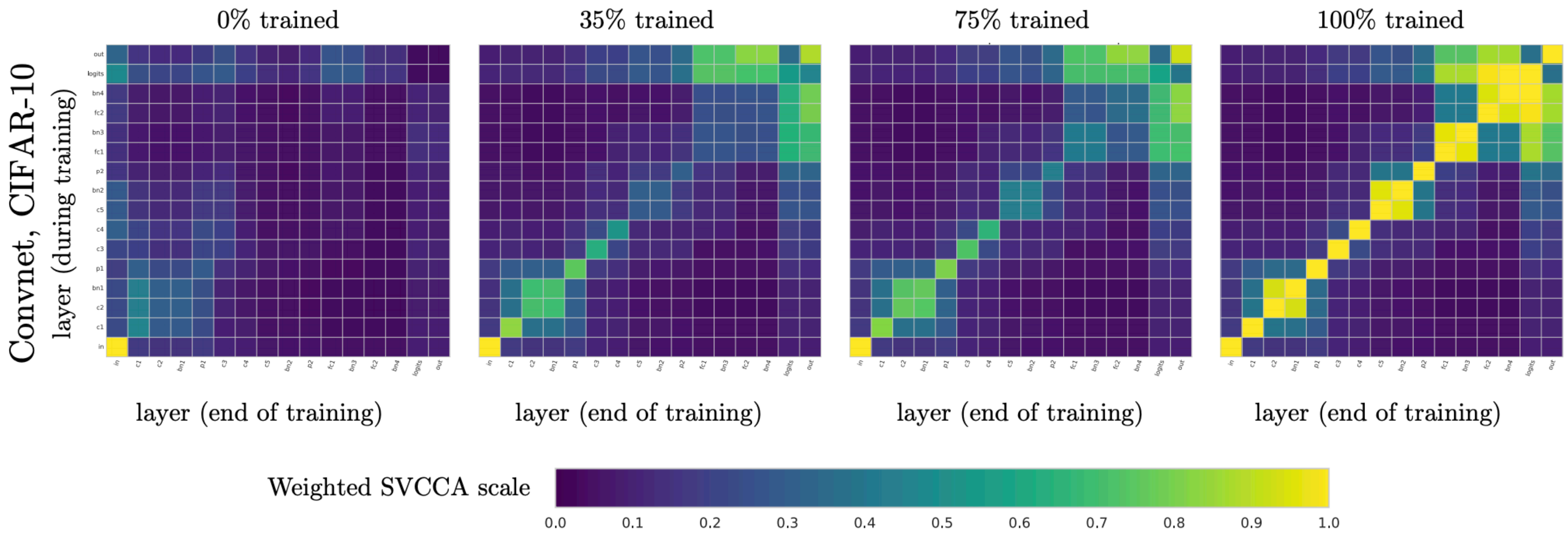
- Comparing the layers of two pre-trained CIFAR-10 models that used different model initialization



Source: [SVCCA, Raghu, et. al., 2017]

SINGULAR VECTOR CANONICAL CORRELATION ANALYSIS (SVCCA)

- Layer similarities between trained and during different stages of training for CIFAR-10 model



Source: SVCCA, Raghu, et. al., 2017

PROJECTED WEIGHT CANONICAL CORRELATION ANALYSIS (PWCCA)

- An improvement of SVCCA that its better at distinguishing noise from important signal
- **Projected Weight Canonical Correlation Analysis (PWCCA)** (Insights on representational similarity in neural networks with canonical correlation, Marcos, et. al., 2018)

  Weights CCA vectors based on how much the original input vector accounts for the CCA canonical covariates.

CENTERED KERNEL ALIGNMENT (CKA)

- Compares embedding representations of layer pairs in a model or across multiple models
- **Centered Kernel Alignment (CKA)** (Similarity of Neural Network Representations Revisited, Kornblith, et. al., 2019)

  A generalization of dot product similarity metric by Kernel Hilbert Spaces

CENTERED KERNEL ALIGNMENT (CKA)

- Compares embedding representations of layer pairs in a model or across multiple models
- **Centered Kernel Alignment (CKA)** (Similarity of Neural Network Representations Revisited, Kornblith, et. al., 2019)

  A generalization of dot product similarity metric by Kernel Hilbert Spaces

  Given $X = F_{L2}(input), Y = F_{L6}(input)$

$$\langle vec(XX^T), vec(YY^T)\rangle = tr(XX^T YY^T) = ||Y^T X||_F^2 \quad (1)$$

CENTERED KERNEL ALIGNMENT (CKA)

- Given $X = F_{L2}(input), Y = F_{L6}(input)$

$$\langle vec(XX^T), vec(YY^T) \rangle = tr(XX^T YY^T) = ||Y^T X||_F^2 \qquad (1)$$

can be generalized with Hilbert-Schmidt Independence Criterion for centered X and Y

Linear or RBF kernels: $K_{ij} = k(x_i, x_j), L_{ij} = l(y_i, y_j)$

$HSIC(K, L) = \frac{1}{(n-1)^2} tr(KHLH)$, H is centering matrix

CENTERED KERNEL ALIGNMENT (CKA)

- Given $X = F_{L2}(input), Y = F_{L6}(input)$

$$\langle vec(XX^T), vec(YY^T)\rangle = tr(XX^TYY^T) = ||Y^TX||_F^2$$

can be generalized with Hilbert-Schmidt Independence Criterion for centered X and Y

Linear or RBF kernels: $K_{ij} = k(x_i, x_j), L_{ij} = l(y_i, y_j)$

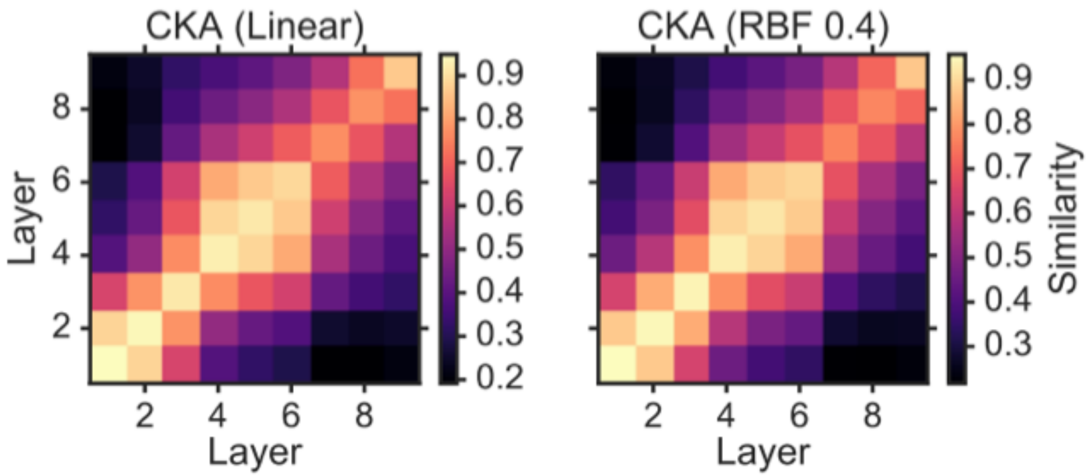
$HSIC(K, L) = \frac{1}{(n-1)^2} tr(KHLH)$, H is centering matrix

In order to be invariant to isotopic scaling

$$CKA(K, L) = \frac{HSIC(K, L)}{\sqrt{HSIC(H, H)HSIC(L, L)}}$$

CENTERED KERNEL ALIGNMENT (CKA)

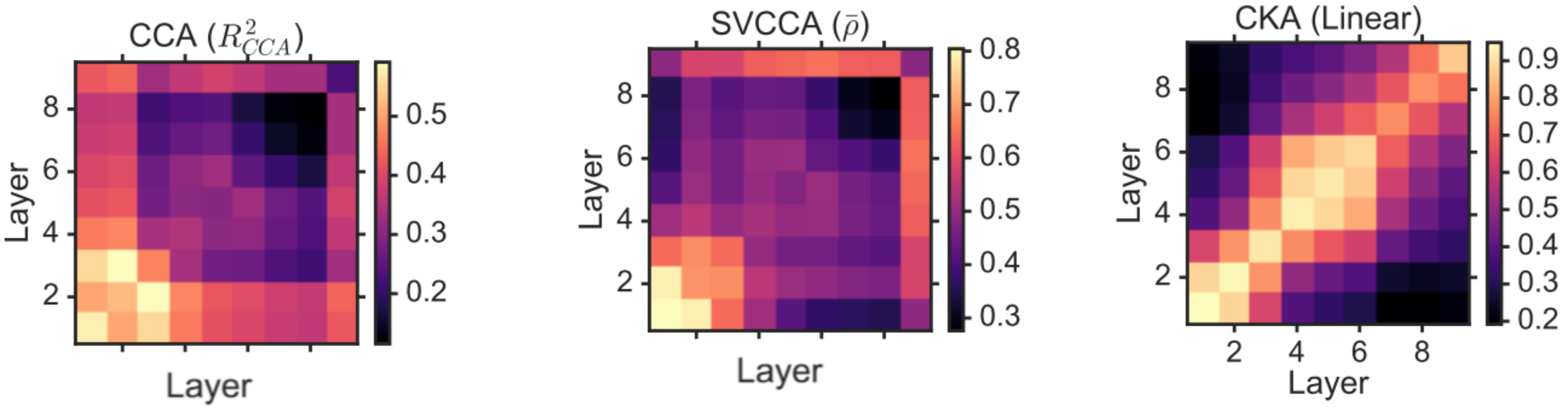
- CKA using linear and RBF Kernels for CIFAR-10 model trained with two different parameter initializations



Source: Similarity of Neural Network Representations Revisited, Kornblith, et. al., 2019

CENTERED KERNEL ALIGNMENT (CKA)

- CCA vs SVCCA vs CKA for CIFAR-10 model trained with two different parameter initializations



Source: Similarity of Neural Network Representations Revisited, Kornblith, et. al., 2019

SUMMARY

- Five desiderata of model interpretability research
- Black Box vs inherently interpretable models
- Explaining with gradient and perturbation-based attribution algorithms
- Attribution algorithms for image classification and segmentation
- Evaluating the quality of model explanations
- Concept-based model interpretability
- Model comparison and correlation analysis

OTHER DIRECTIONS OF MODEL INTERPRETABILITY RESEARCH

- Optimization-based visualizations

[Identifying concepts learned by a neuron or groups of neurons]

- Adversarial robustness and model interpretability
  - More robust models and more interpretable gradients