Interpretability Tools as Feedback Loops

BoilerMake X

J. Setpal

January 21, 2023

Outline

1. Setting the Stage
2. Baselining Interpretability
3. Leveraging Interpretability

Outline

1. Setting the Stage
2. Baselining Interpretability
3. Leveraging Interpretability

DagsHub

Here's a Scenario

Consider the following:

a. We want to build a classifier (classifiers are cool).

Here's a Scenario

Consider the following:

a. We want to build a classifier (classifiers are cool).
b. This classifier differentiates between an Orca and a Leopard.

Here's a Scenario

Consider the following:

a. We want to build a classifier (classifiers are cool).

b. This classifier differentiates between an Orca and a Leopard.

c. We use the Caltech-256 dataset to obtain images of both:

Here's a Scenario

Consider the following:

a. We want to build a classifier (classifiers are cool).

b. This classifier differentiates between an Orca and a Leopard.

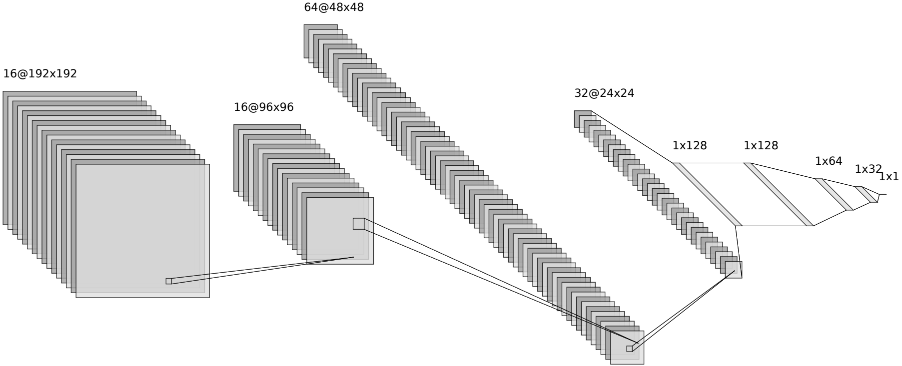
c. We use the Caltech-256 dataset to obtain images of both:

d. There are 188 leopard images and 89 orca images.

DagsHub

More Scenario Stuff

Here's our model architecture:



Last Bit of Scenario, I Promise

We use:

a. Optimizer: `SGD`
   - Learning Rate: 10^{-2}
   - Epsilon: 10^{-8}

b. Loss: `BinaryCrossEntropy`

c. Epochs: 5

DagsHub

Last Bit of Scenario, I Promise

We use:

a. Optimizer: `SGD`
   - Learning Rate: 10^{-2}
   - Epsilon: 10^{-8}

b. Loss: `BinaryCrossEntropy`

c. Epochs: 5

During training, our training and validation accuracies are $\approx$ `1.000` 🎉

DagsHub

Last Bit of Scenario, I Promise

We use:

a. Optimizer: `SGD`
   - Learning Rate: 10^{-2}
   - Epsilon: 10^{-8}

b. Loss: `BinaryCrossEntropy`

c. Epochs: 5

During training, our training and validation accuracies are $\approx$ 1.000 🎉
However, we achieve a **test accuracy** of only 0.5938. This *sucks*.

Last Bit of Scenario, I Promise

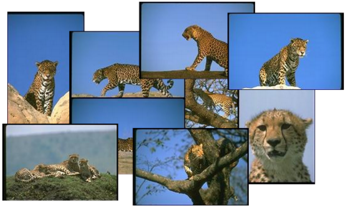
We use:

a. Optimizer: `SGD`
   - Learning Rate: 10^{-2}
   - Epsilon: 10^{-8}

b. Loss: `BinaryCrossEntropy`

c. Epochs: 5

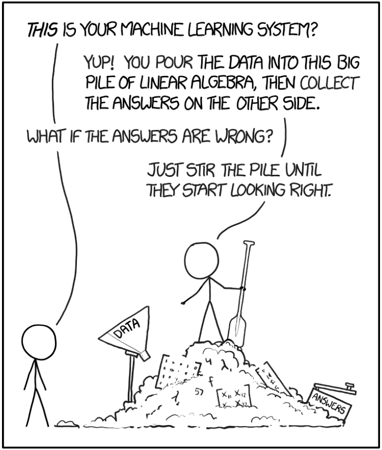
During training, our training and validation accuracies are $\approx$ `1.000` 🎉
However, we achieve a **test accuracy** of only `0.5938`. This *sucks*.
Here are some misclassified samples:



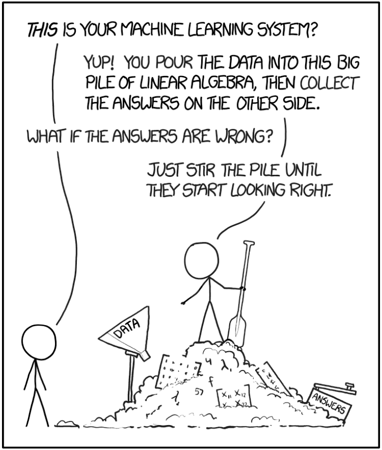
Outline

1. Setting the Stage
2. Baselining Interpretability
3. Leveraging Interpretability

What even *is* Interpretability?



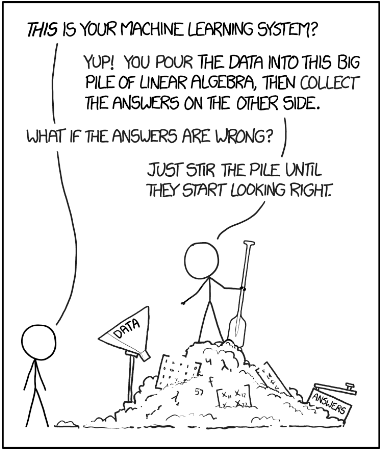
DagsHub



Interpretability within Machine Learning is the **degree** to which we can understand the **cause** of a decision, and use it to consistently <u>predict the model's prediction</u>.

DagsHub

What even *is* Interpretability?

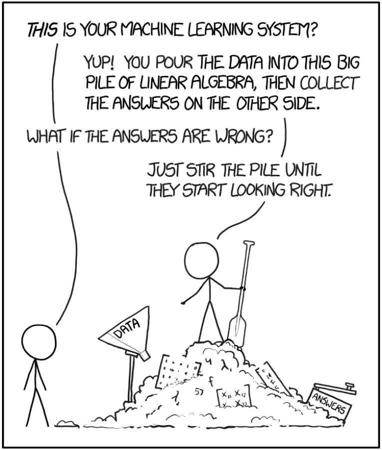


Interpretability within Machine Learning is the **degree** to which we can understand the **cause** of a decision, and use it to consistently predict the model's prediction.

This is easy for shallow learning.

DagsHub

What even *is* Interpretability?



Interpretability within Machine Learning is the **degree** to which we can understand the **cause** of a decision, and use it to consistently <u>predict the model's prediction</u>.

This is easy for shallow learning. For deep learning however, it is a **lot harder**.

A Cautionary Tale

Let's explore: `https://interaktiv.br.de/ki-bewerbung/en/`

Start-up attempting to make the application process 'faster, but also more objective and fair'.

A Cautionary Tale

Let's explore: `https://interaktiv.br.de/ki-bewerbung/en/`

Start-up attempting to make the application process 'faster, but also more objective and fair'.

They were not successful.

Class Activation Mappings

For deep learning, interpretability techniques today involve a fairly straightforward formula:

Class Activation Mappings

For deep learning, interpretability techniques today involve a fairly straightforward formula:

- Split hidden layers.
- Expose weights.
- *Observe!*

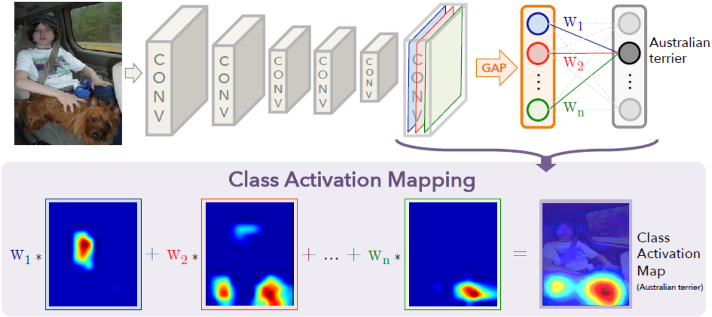
DagsHub

Class Activation Mappings

For deep learning, interpretability techniques today involve a fairly straightforward formula:

- Split hidden layers.
- Expose weights.
- *Observe!*

We'll focus today's discussion on **Class Activation Mappings (CAMs)**:



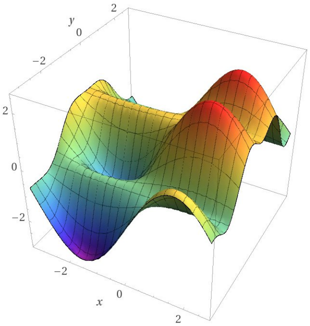
DagsHub

Building Feedback Loops

Finding optimal model weights is an **NP-hard** problem.

Building Feedback Loops

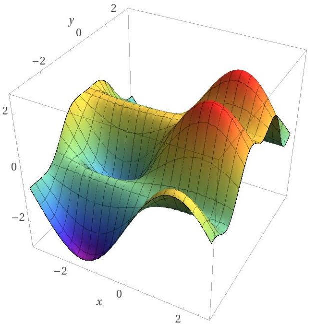
Finding optimal model weights is an **NP-hard** problem.



Model Search Space

Building Feedback Loops

Finding optimal model weights is an **NP-hard** problem.

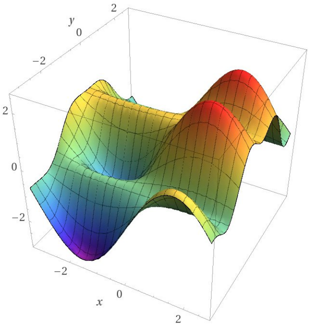


Model Search Space

We can't speed this up. However, we do have information about our training set that we can use to **motivate training behaviour**.

Building Feedback Loops

Finding optimal model weights is an **NP-hard** problem.



Model Search Space

We can't speed this up. However, we do have information about our training set that we can use to **motivate training behaviour**.

So, the idea here is simple: use shared knowledge (+ common sense) to modify how we train our models.

Updating the Search Space

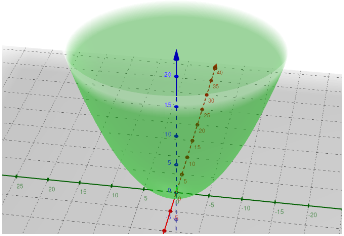
We approach the challenge in the *opposite* direction.

Updating the Search Space

We approach the challenge in the *opposite* direction. Instead of optimizing how we solve the search space, let's **update the search space** itself!

Updating the Search Space

We approach the challenge in the *opposite* direction. Instead of optimizing how we solve the search space, let's **update the search space** itself!

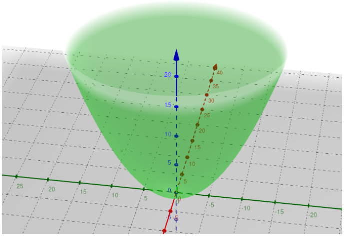


Updated Search Space!

By updating our loss function to eliminate *pseudo-correctness*, we can:

Updating the Search Space

We approach the challenge in the *opposite* direction. Instead of optimizing how we solve the search space, let's **update the search space** itself!



Updated Search Space!

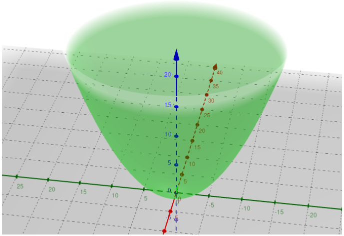
By updating our loss function to eliminate *pseudo-correctness*, we can:

- Make the optimal weights incredibly easy for our optimzer to find.

DagsHub

Updating the Search Space

We approach the challenge in the *opposite* direction. Instead of optimizing how we solve the search space, let's **update the search space** itself!



Updated Search Space!

By updating our loss function to eliminate *pseudo-correctness*, we can:

- Make the optimal weights incredibly easy for our optimzer to find.
- Allow our generalized model to extrapolate on implicit information.

Outline

Getting Back to the Challenge

There are some obvious causes for why it performs poorly:

a. There are too few, unbalanced training samples.

DagsHub

There are some obvious causes for why it performs poorly:

a. There are too few, unbalanced training samples.
   Solution: Data Augmentation / Covariate Shift

DagsHub

Getting Back to the Challenge

There are some obvious causes for why it performs poorly:

a. There are too few, unbalanced training samples.
   Solution: Data Augmentation / Covariate Shift
b. **The images have a sharp color dominance.**

DagsHub

Getting Back to the Challenge

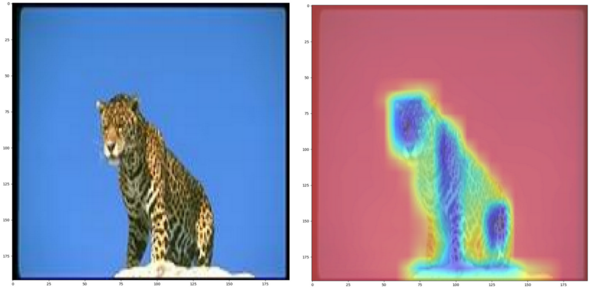
There are some obvious causes for why it performs poorly:

a. There are too few, unbalanced training samples.
   Solution: Data Augmentation / Covariate Shift

b. **The images have a sharp color dominance.**

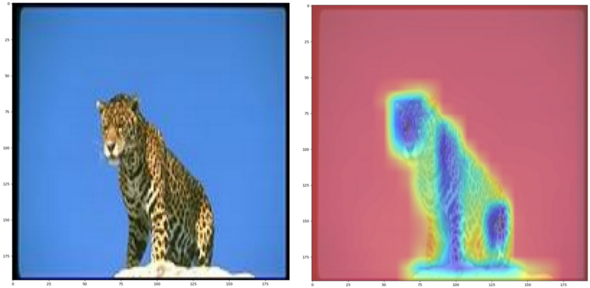
Diagnosing the Model

When we obtain a Class Activation Map of a sample image, we observe:



Diagnosing the Model

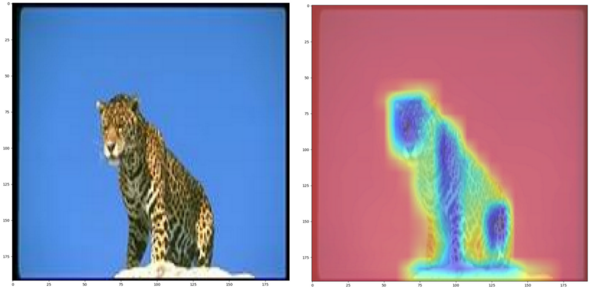
When we obtain a Class Activation Map of a sample image, we observe:



It **does not** use the leopard to base it's prediction! This is prevalent across the dataset.

Diagnosing the Model

When we obtain a Class Activation Map of a sample image, we observe:

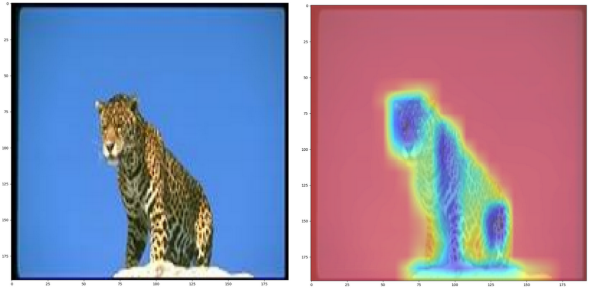


It **does not** use the leopard to base it's prediction! This is prevalent across the dataset.

Observation: The targets in our entire training dataset are centered.

Diagnosing the Model

When we obtain a Class Activation Map of a sample image, we observe:



It **does not** use the leopard to base it's prediction! This is prevalent across the dataset.

Observation: The targets in our entire training dataset are centered.

Q: Can we exploit this?

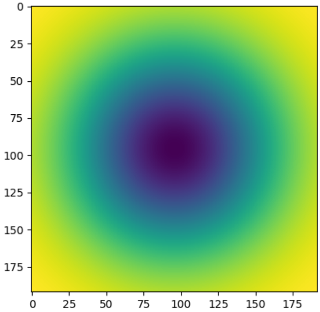
Exploiting a Centred Dataset

Idea: Let's penalize our batch whenever the CAM is off-center.

Exploiting a Centred Dataset

Idea: Let's penalize our batch whenever the CAM is off-center.

We can achieve this by inverting a 2D Gaussian Kernel.

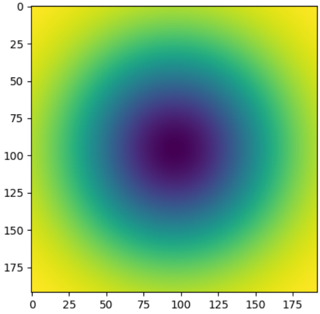


Inverted Gaussian Kernel

Exploiting a Centred Dataset

Idea: Let's penalize our batch whenever the CAM is off-center.

We can achieve this by inverting a 2D Gaussian Kernel.



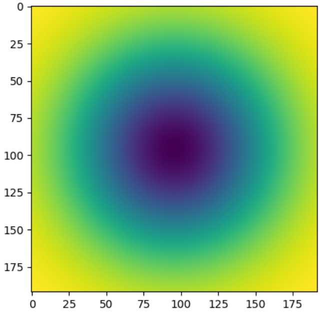
Inverted Gaussian Kernel

Weights scale sharply as the operation approaches the image boundary.

Exploiting a Centred Dataset

Idea: Let's penalize our batch whenever the CAM is off-center.

We can achieve this by inverting a 2D Gaussian Kernel.



Inverted Gaussian Kernel

Weights scale sharply as the operation approaches the image boundary.

It has to not be perfect, since we don't intend to overfit our model to this setup. This kernel filter is a **guideline**.

DagsHub

Introducing `CAMLoss`!

Here's how it all comes together:

a. In addition to the prediction, we output the class activation map.

Introducing `CAMLoss`!

Here's how it all comes together:

a. In addition to the prediction, we output the class activation map.

b. We apply our kernel filter on every batch.

Introducing `CAMLoss`!

Here's how it all comes together:

a. In addition to the prediction, we output the class activation map.

b. We apply our kernel filter on every batch.

c. We return the mean of the returned features. Weights $\propto \frac{1}{\text{Fit Quality}}$

DagsHub

Introducing `CAMLoss`!

Here's how it all comes together:

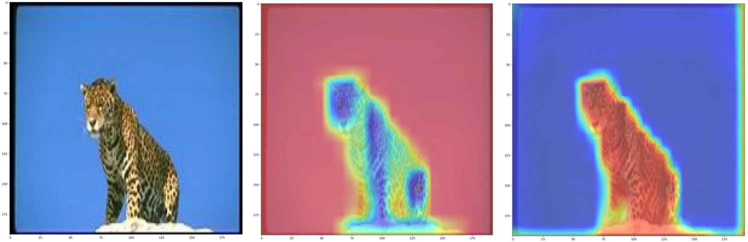
a. In addition to the prediction, we output the class activation map.
b. We apply our kernel filter on every batch.
c. We return the mean of the returned features. Weights $\propto \frac{1}{\text{Fit Quality}}$
d. This is our additional self-supervised loss function!

Introducing CAMLoss!

Here's how it all comes together:

a. In addition to the prediction, we output the class activation map.
b. We apply our kernel filter on every batch.
c. We return the mean of the returned features. Weights $\propto \frac{1}{\text{Fit Quality}}$
d. This is our additional self-supervised loss function!

Obtaining the Class Activation Map of the updated model, we observe:

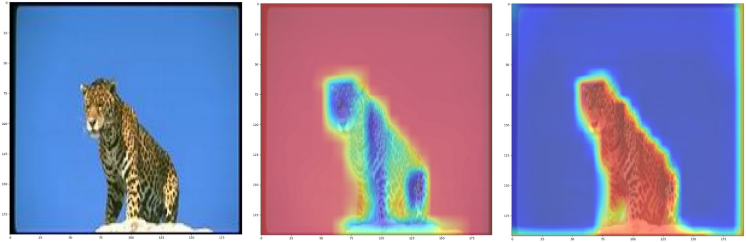


Introducing CAMLoss!

Here's how it all comes together:

a. In addition to the prediction, we output the class activation map.
b. We apply our kernel filter on every batch.
c. We return the mean of the returned features. Weights $\propto \frac{1}{\text{Fit Quality}}$
d. This is our additional self-supervised loss function!

Obtaining the Class Activation Map of the updated model, we observe:



Great Success!

Implementation Details

If you can view this screen, I am making a mistake.

Thank you!

Have an awesome rest of your day! Any questions for me?

Code, Experiments, Data, Slides:

https://dagshub.com/jinensetpal/lint.git