Explainability AI (XAI) – Understanding the black-box of complex machine learning models.

A DATS 6303 Deep Learning Project report by Team – 2

Aditya Kumar, Medhasweta Sen & Tyler Wallet

Abstract

As we solve more complex problems, the complexity of our models also increases. This means that most machine learning models are doing something under the hood that is so complex that we don't have a clue anymore why they behave the way they do. In other words, we don't know the thought process of our model in predicting something. Understanding the behavior of our machine learning model is becoming very important. Judging the model's performance based on its accuracy or other metrics alone is not sufficient anymore as your model can trick you.

This is where Explainable AI or XAI comes to the rescue. XAI aims to make the model's behavior understandable. The explanations produced by this concept should help you to understand why the model behaves the way it does. If the model isn't behaving as expected, there's a good chance you did something wrong in the data preparation phase, or choice of the modeling technique.

Outline of Project Components

In this work, we have developed several XAI demonstrations across various kinds of machine learning models for Image Classification. The purpose of these demonstrations is for the user to understand these concepts in detail before utilizing them. Ultimately, we have produced a complete no-code product offering to evaluate the robustness of Image Classification solutions leveraging XAI utilities.

A comprehensive list of all work done is –

- Inspection of MLP and LSTM model architectures on demo datasets for image classifications through base PyTorch.
- XAI for object Detection with no code hyperparameter tuning
- Leveraging existing Python XAI utilities to provide an in-depth explainer on their internal workings along with a demo for each, to educate users on the right utility for their application. The utilized utilities are as follows:
  - ➢ GradCAM
  - ➢ Occlusion Sensitivity
  - ➢ Rise

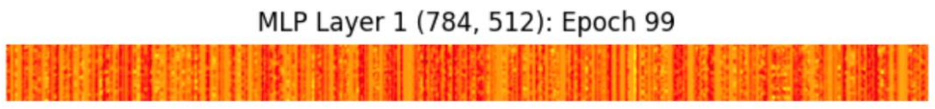
  - Integrated Gradient
  - Lime
- Lastly, developed an entire no-code product offering that will work with TensorFlow and PyTorch frameworks. Users can upload any trained model, Custom or Pre-Trained, along with an observation of their choice to analyze the model's inner working.

Description of XAI Concepts

A Simple Approach:

I. MLP:

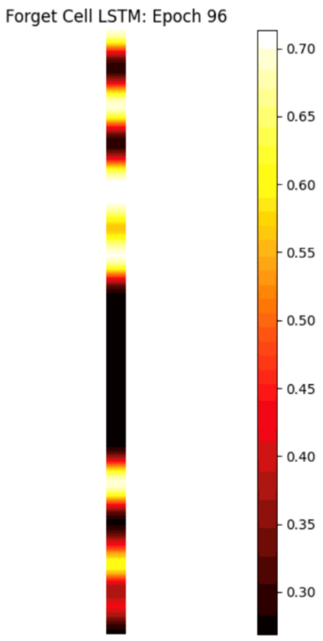
Multilayered perceptron's (MLPs), like most statistical models, are susceptible to overfitting and underfitting. To demonstrate the reliability of a model at the inference stage, we created an animated heatmap of the class activations to visualize whether these are overfitting or underfitting. The purpose of this visualization is for the user to note whether the intermediary ReLU activations are deactivating or activating hidden layer neurons. One sign of MLPs overfitting is their class activation maps converging to zero. To demonstrate the previously mentioned we purposely overfitted a MNIST Classification model resulting in the following illustration:



Here we can observe the straight opaque lines as zero vectors, which indicate a clear case of overfitting. Thus, in this manner, students can check and refer to whether their model is overfitting by looking at this animation.

II. LSTM

Long-Short Term Memory (LSTMs) networks possess a unique module inside its unrolled computational graph called the 'forget gate'. To properly explain how the LSTM model is behaving it is pertinent to look at the behavior of the forget gate. The forget gate in formal terms is a sigmoid activated vector that projects to the previous 'memory cells' of the recurrent network. The role behind this projection is to tell the memory cell what to forget and what to remember (0 weights representing forget; 1 weight representing remember) Our purpose then was to create an animation to visualize which part of the vector is remembering and which part is forgetting:



Here we can see that the bright spots of the forget vector are remembering, as opposed to the opaque spots which are forgetting. In this manner we can clearly explain what is happening inside a LSTM network.

XAI for Object Detection:

Object detection in computer vision is a demanding task, and achieving precise results involves locating objects in images and accurately classifying them. Class Activation Maps (CAM) are a popular tool for highlighting important image regions for classification. However, when applied to object detection, they face unique challenges due to the need to work with complex, multi-object scenes.

Gradient-Free CAM Methods for Object Detection:

In object detection scenarios, the usual CAM methods that rely on gradients may not be suitable because they can't easily differentiate between objects or efficiently handle complex object detection pipelines. Instead, we turn to gradient-free CAM methods, two of which stand out:

- **EigenCAM**: EigenCAM is a lightning-fast method for identifying crucial image regions, making it ideal for scenarios where speed is essential, such as object detection. However, it may not excel at distinguishing between different object categories within an image, which is acceptable in many object detection applications.

- **AblationCAM**: AblationCAM, on the other hand, is a more sophisticated gradient-free CAM method. It's particularly valuable when the goal is to achieve fine-grained class discrimination. This is crucial in object detection when we want to differentiate between various object categories present in the same image. AblationCAM achieves this by selectively "ablating" or deactivating specific regions in the image and observing how these changes affect the model's output. It assigns higher importance to areas that significantly influence the model's ability to classify objects.

The Role of the FPN Backbone:

In the context of object detection models like Faster R-CNN, the Feature Pyramid Network (FPN) backbone plays a vital role. It generates essential activations that are used to locate and classify objects within an image. When applying CAM methods, we focus on this FPN backbone to compute meaningful activations.

Custom Reshape Transform:

Working with tensors of varying spatial sizes generated by the FPN backbone can be a technical challenge in object detection. To address this, a custom "reshape" transform called **fasterrcnn_reshape_transform** is introduced. This transform aggregates differently sized tensors into a common shape, ensuring effective utilization of the activations.

Custom Target Function for CAM:

For CAM methods to be effective in object detection, it's crucial to define a custom "target" function. This function determines what aspect of the bounding boxes to maximize when generating the CAM. In object detection, this could involve maximizing bounding box scores, matching object categories, or considering properties like width or height. The introduces a target function called **FasterRCNNBoxScoreTarget**, which calculates a score for each bounding box based on factors like Intersection over Union (IoU) and category matching.

Adjusting the ratio_channels_to_ablate Parameter:

An intriguing feature discussed is the ability to modify the number of layers to ablate in AblationCAM. This flexibility is governed by the **ratio_channels_to_ablate** parameter, which allows for the adjustment of the proportion of channels (activations) to ablate. Fine-tuning this parameter provides a balance between computation speed and the level of detail captured in the CAM. For example, setting it to 0.1 means ablating 10% of the channels, potentially speeding up computation while still capturing essential information. Conversely, setting it to 0.01 reduces ablation to just 1% of the channels, resulting in a more detailed but computationally intensive CAM.

In summary, this exploration demonstrates the adaptability of gradient-free CAM methods, such as EigenCAM and AblationCAM, to address the specific challenges and goals of object detection projects. These methods provide valuable insights into detected objects within images while considering computational efficiency, making them powerful tools in the field of computer vision.

Existing Solutions:

I. GradCAM

GradCAM is a technique for producing visual explanations that can be used on Convolutional Neural Network (CNN) which uses both gradients and the feature maps of the last convolutional layer (Xplique) As quoted in the research paper *Visual Explanations from Deep Networks via Gradient-based Localization*, GradCAM uses the gradients of any target concept (say logits for "dog" or even a caption), flowing into the final convolutional layer to produce a coarse localization map highlighting the important regions in the image for predicting the concept (arXiv) Further, we can derive the weights of each feature map as:

$$w_k = \frac{1}{Z} \sum_i \sum_j \frac{\partial f(x)}{\partial A^k_{i,j}}$$

Then, to create a visualization we can aggregate the feature maps:

$$\phi = \max(0, \sum_k w_k A^k)$$

To output heatmap with bicubic interpolation.

II. Occlusion Sensitivity

Occlusion Sensitivity is a method that sweeps a patch that occludes pixels over the images and uses the variations of the model prediction to deduce critical areas (Xplique) To this extent, the user passes the desired patch size and patch stride to produce special class scores to each occluded input as:

$$\phi_i = S_c(x) - S_c(x_{[x_i = \bar{x}]})$$

Where Sc is the unnormalized class score (net input of last layer prior to SoftMax activation function) and phi is the corresponding occluded feature map.

III. Rise

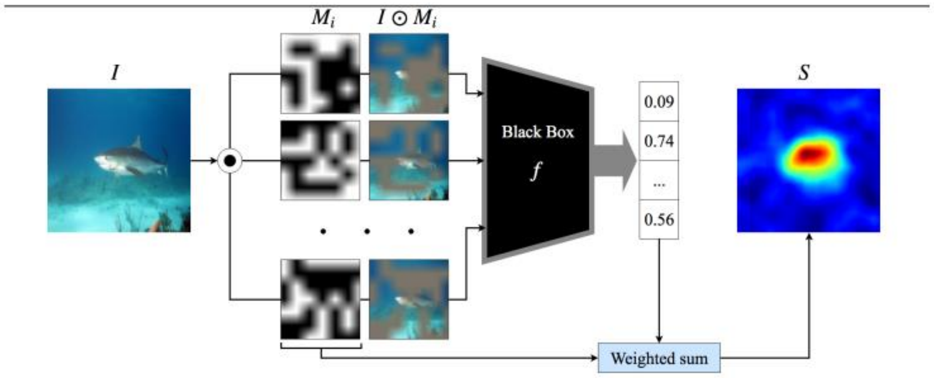


Figure 3: Overview of RISE: Input image I is element-wise multiplied with random masks M_i and the masked images are fed to the base model. The saliency map is a linear combination of the masks where the weights come from the score of the target class corresponding to the respective masked inputs.

Rise is an attribution method designed primarily for computer vision tasks, offering insights into why a machine learning model makes specific decisions when processing images. It has been seamlessly integrated into the Xplique framework, which provides tools for explaining and interpreting machine learning models.

At its core, Rise falls under the umbrella of attribution methods, which aim to shed light on the critical factors or variables that influence a model's decision-making process. In the context of computer vision, this typically involves identifying the pixels within an input image that have the most significant impact on the model's output or prediction.

Rise, however, stands out as a perturbation-based attribution method, which places it in one of two major categories of attribution methods. Unlike back-propagation methods, perturbation-based methods focus on perturbing or modifying the input data and observing how these perturbations affect the model's output. The underlying assumption is that perturbing important input elements will result in more substantial changes in the model's output, thus highlighting their significance.

Rise employs a unique approach to attribution, distinct from gradient-based methods or other perturbation-based techniques. Here's a closer look at how the Rise method operates:

1. **Binary Masks**: Rise generates binary masks that cover various portions of an input image. These binary masks act as a way to either hide or reveal specific pixels within the image.
2. **Scoring Pixel Influence**: For each pixel in the image, Rise calculates a score. This score is determined by observing how the model's output changes when that particular pixel is either masked (hidden) or unmasked (revealed). Essentially, Rise assesses the influence of each pixel on the model's decision.

3. **Averaging Scores**: To provide a comprehensive explanation, Rise takes the mean of the scores obtained for each pixel. This aggregation of scores allows Rise to identify which pixels, on average, have the most significant impact on the model's decision.

Visualising Rise Masks

To help our understand the concept of Rise masks better, consider the analogy of a grid. Rise generates these random masks by following a grid pattern, where each grid cell can be set to either 0 or 1, indicating whether the corresponding pixel is hidden or revealed. This grid-based approach allows Rise to explore various combinations of pixel masks to understand their collective impact.

Hyperparameters in Rise

To effectively use the Rise method, we need to be aware of and tune several hyperparameters. These hyperparameters significantly affect the quality and efficiency of our attribution results. Some crucial hyperparameters in Rise include:

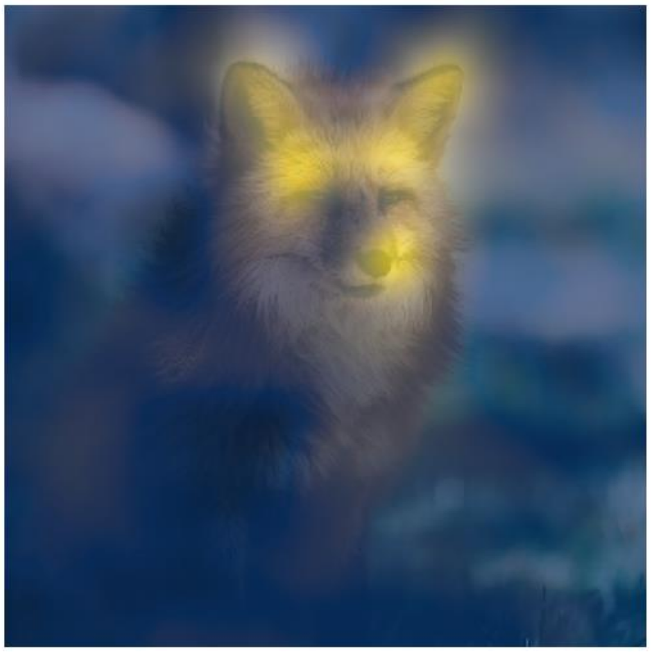
4. **batch_size**: Determines the number of perturbed samples processed simultaneously. It should align with our hardware capabilities and image size.
5. **nb_samples**: Specifies the number of masks generated for Monte Carlo sampling. Finding the right balance is essential, as too few masks result in noisy explanations, while too many increase computation time.
6. **grid_size**: Sets the size of the grid used to generate masks. Smaller values create smoother explanations, while larger values increase precision.
7. **preservation_probability**: Controls the percentage of non-masked pixels in each mask. It affects the quality of the explanation but not computation time.

Tuning Rise Hyperparameters

To achieve the best results with Rise, we'll need to fine-tune these hyperparameters based on our specific use case. Here are some tips for tuning them effectively:

- Match the batch size with the one used during our model's training to ensure efficient memory usage.
- Experiment with the number of samples (nb_samples) to strike a balance between explanation precision and computation time. Larger input sizes typically require more samples.
- Adapt the grid size (grid_size) to our image dimensions and the size of elements of interest. Start with a default value of 7 and adjust from there.

- Modify the preservation probability (preservation_probability) based on our goals. Increase it to highlight more elements simultaneously or decrease it for a more focused explanation



nb_samples=20000, grid size=13, preservation probability=0.5

IV. Integrated Gradient:

Integrated Gradients is a powerful technique used in the realm of machine learning to shed light on the decision-making processes of complex models, especially in tasks related to Computer Vision. In simpler terms, it helps us understand why a machine learning model makes a particular prediction by identifying the critical factors or features that influence that decision.

To grasp the essence of Integrated Gradients, let's start with the broader context of attribution methods. Attribution methods, in the world of machine learning, are like detectives trying to figure out which clues led to a specific outcome. In the case of Computer Vision, their mission is to highlight the pixels in an image that have the most significant impact on the model's output.

Integrated Gradients belongs to a category of attribution methods known as back-propagation methods. These methods, alongside perturbation-based techniques, are pivotal in understanding model decisions. What sets back-propagation methods apart is their reliance on the model's internal weights. Essentially, they backtrack from the model's final prediction to the input data, following the path of significance.

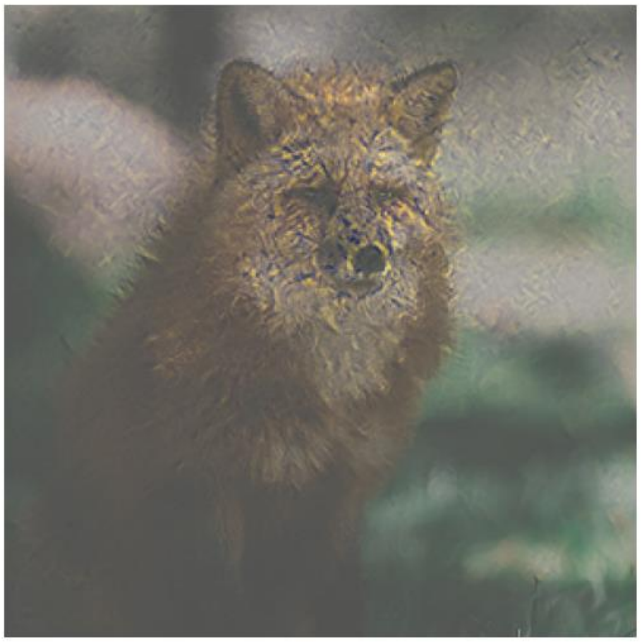
Now, let's dive into the heart of Integrated Gradients. Imagine we have an image, and we want to understand why our model classified it the way it did. Instead of just looking at the image's gradient, Integrated Gradients takes a more sophisticated approach. It calculates the gradients not just at a single point but along a path from a starting point (called the baseline) to the image we're interested in.

Here's a breakdown of the key elements:

1. **Baseline State**: The journey begins with a baseline state, often representing the complete absence of features. Think of it as the "zero" state where nothing relevant is happening.
2. **The Path**: Integrated Gradients computes gradients at various points along a straight-line path from this baseline state to the input image. These gradients tell us how sensitive the model's prediction is to changes along this path.
3. **Averaging**: Unlike basic gradient calculations, Integrated Gradients doesn't stop at just one point. It averages the gradient values across the entire path. This helps provide a more comprehensive view of feature importance.

Now, why is Integrated Gradients so valuable? It helps us understand not just the "what" but the "why" behind a model's decisions. It uncovers which parts of an image or data contributed most to a particular prediction. This insight can be used for debugging

models, improving their transparency, and even identifying biases or unexpected behaviour.
In practice, when using Integrated Gradients, we'll need to consider a few parameters, such as the output layer, batch size, steps, and baseline value. These parameters help fine-tune the method to suit our specific use case and computational resources.
In summary, Integrated Gradients is a method that brings transparency and interpretability to machine learning models, especially in tasks like image recognition. By tracing the path of significance from a baseline state to the input data, it helps us understand why models make certain decisions, making it an invaluable tool in the world of explainable AI.



output layer=-1, batch size=16, steps=50, baseline value=0

V. Lime

The acronym LIME stands for Local Interpretable Model-agnostic Explanations. LIME supports explanations for tabular models, text classifiers, and image classifiers. However, for the purpose of this work, the focus is specifically on Image Classifiers.
LIME can provide model agnostic local explanations for solving both regression and classification problems and it can be applied with both structured datasets and even with unstructured datasets like text and images. Lime is a Python Library based on a paper titled “Why Should I Trust You?”, which is referenced towards the end.

The abbreviation of LIME itself gives an intuition about the core idea behind it. LIME is:

- Model agnostic, which means that LIME is model-independent. In other words, LIME is able to explain any black-box classifier.
- Interpretable, which means that LIME provides a solution to understand why your model behaves the way it does, by displaying higher level features of an image such as the super-pixels.
- Local, which means that LIME tries to find the explanation of black-box models by approximating the local linear behavior of the model. Intuitively, an explanation is a local linear approximation of the model's behavior. While the model may be very complex globally, it is easier to approximate it around the vicinity of a particular instance.

How LIME Works:
Internally, LIME tries to interpret a black box model by conducting these four steps:

1. Input data permutation
The first step that LIME does is to create several artificial data points that are close with the instance (image) in the feature space, that is used to explain the model.
LIME will generate several samples that are similar to the input image by turning on and off some of the super-pixels of the image.

2. Predict the class of each artificial data point
Next, LIME will predict the class of each of the artificial data points that has been generated using the trained model. If your input data is an image, then the prediction of each perturbed image will be generated at this stage.

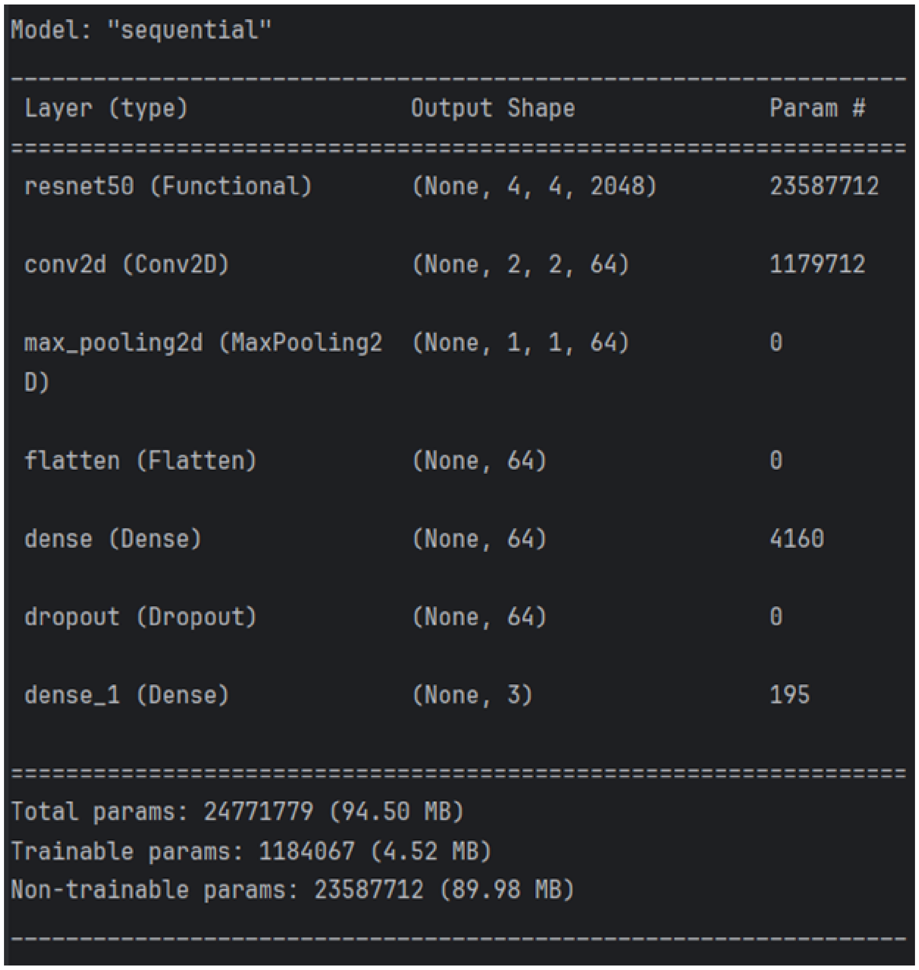
3. Calculate the weight of each artificial data point
The third step is to calculate the weight of each artificial data to measure its importance. To do this, first the cosine distance metric is usually applied to calculate how far the distance of each artificial data point with respect to the original input data. Next, the distance will be mapped into a value between zero to one with a kernel function. The closer the distance, the closer the mapped value to one, and hence, the bigger the weight. The bigger the weight, the bigger the importance of a certain artificial data point.
If the input data is an image, then the cosine distance between each perturbed image and the original image will be computed. The more similarity between a perturbed image to the original image, the bigger its weight and importance.

4. Fit a linear classifier to explain the most important features
The last step is fitting a linear regression model using the weighted artificial data points. After this step, the fitted coefficient of each feature, just like the usual linear regression analysis, is retrieved. Now upon sorting the coefficients, the features that have larger coefficients are the ones that play a big role in determining the prediction of the black-box machine learning model.

Methodology
To analyze the LIME framework, the No-Code Product Offering is utilized as that is integrated with LIME. For the purpose of this analysis, a Kaggle dataset of images containing Dogs, Cats, Pandas is utilized for an image classification task.

1. A custom model architecture is built using ResNet18 as the base followed by Convolutional Layers for image classification. TensorFlow is used as the framework of choice. This model is used to evaluate the working of the no-code

product offering for model Explainability. The results of this are shown in the next section. Model architecture is shown below -



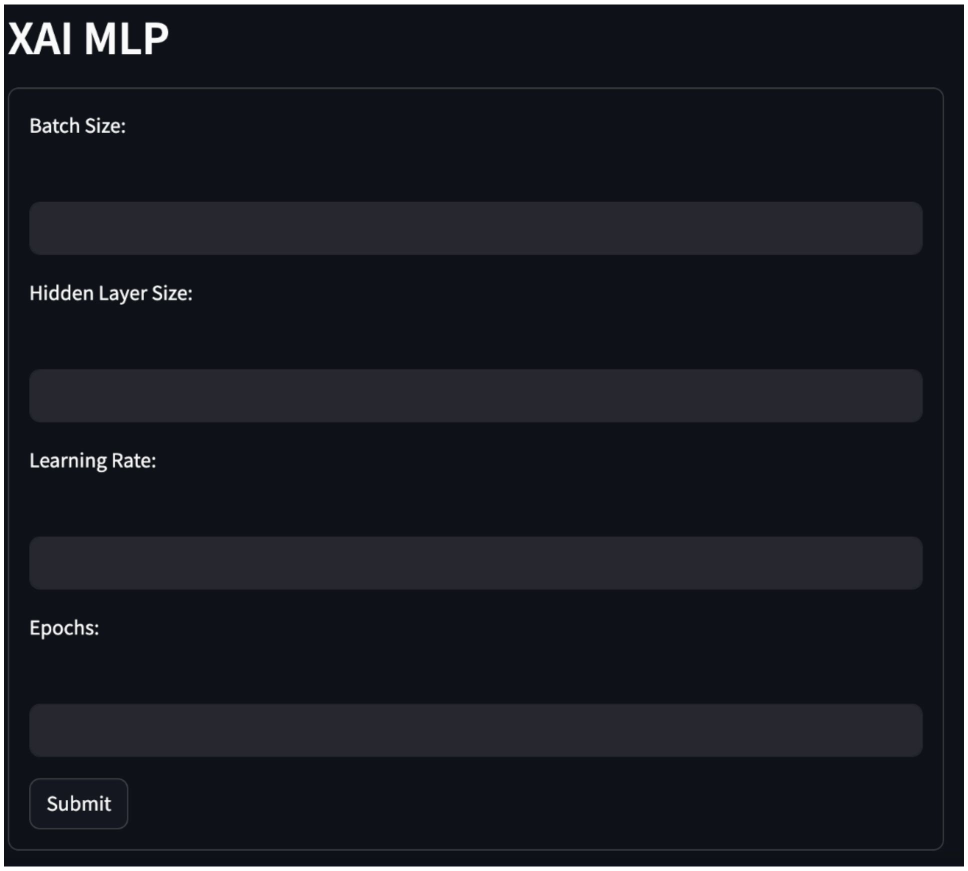
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 4, 4, 2048)        23587712

 conv2d (Conv2D)             (None, 2, 2, 64)          1179712

 max_pooling2d (MaxPooling2  (None, 1, 1, 64)          0
 D)

 flatten (Flatten)           (None, 64)                0

 dense (Dense)               (None, 64)                4160

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 3)                 195

=================================================================
Total params: 24771779 (94.50 MB)
Trainable params: 1184067 (4.52 MB)
Non-trainable params: 23587712 (89.98 MB)
_________________________________________________________________
```

2. A pre-trained model, InceptionNet_V3, from the PyTorch library is utilized as the subsequent model to evaluate and explain its working using the no-code model Explainability offering.

Experimental Setup

Streamlit has been selected as the Web Application Python framework to create an interactive dashboard for both the XAI Demos, as well as the no-code Model Explainability Offering. Streamlit is a free and open-source framework to rapidly build and share machine learning and data science web apps. It is a Python-based library specifically designed for machine learning engineers.

Results



XAI MLP

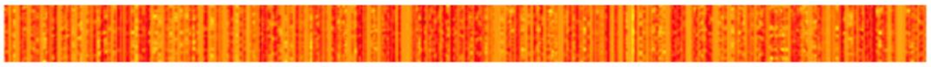
Batch Size:

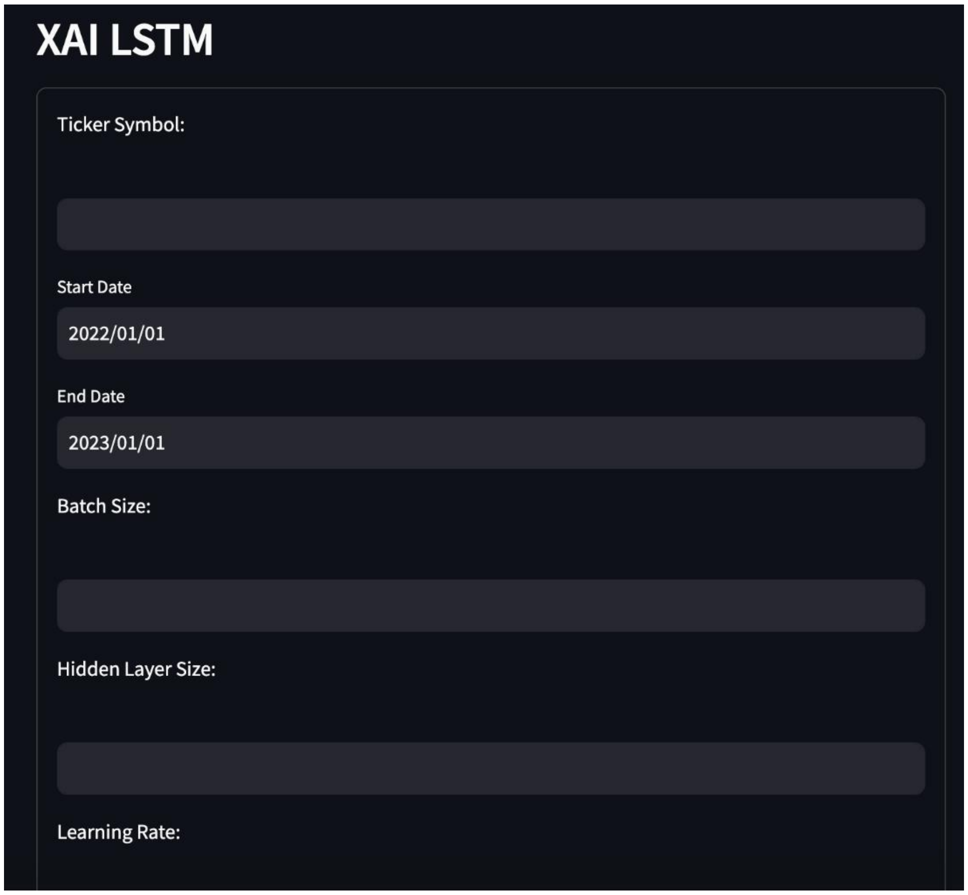
Hidden Layer Size:

Learning Rate:

Epochs:

MLP Layer 1 (784, 512): Epoch 99




XAI LSTM

Ticker Symbol:

Start Date

2022/01/01

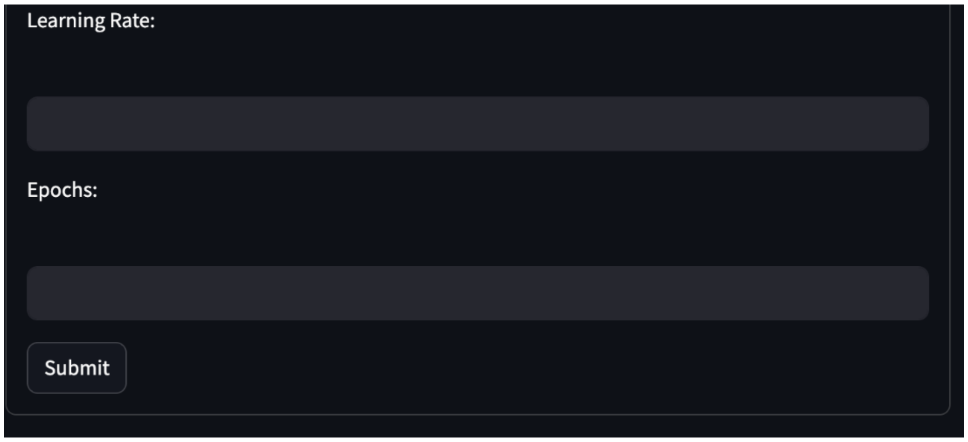
End Date

2023/01/01

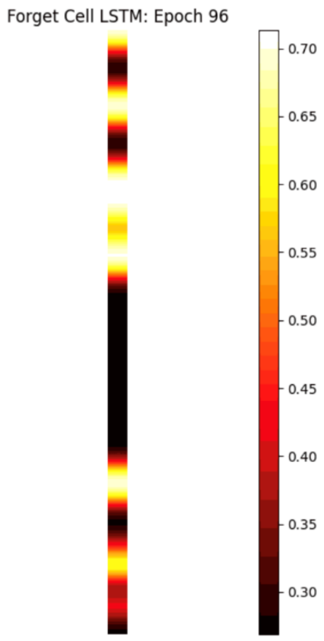
Batch Size:

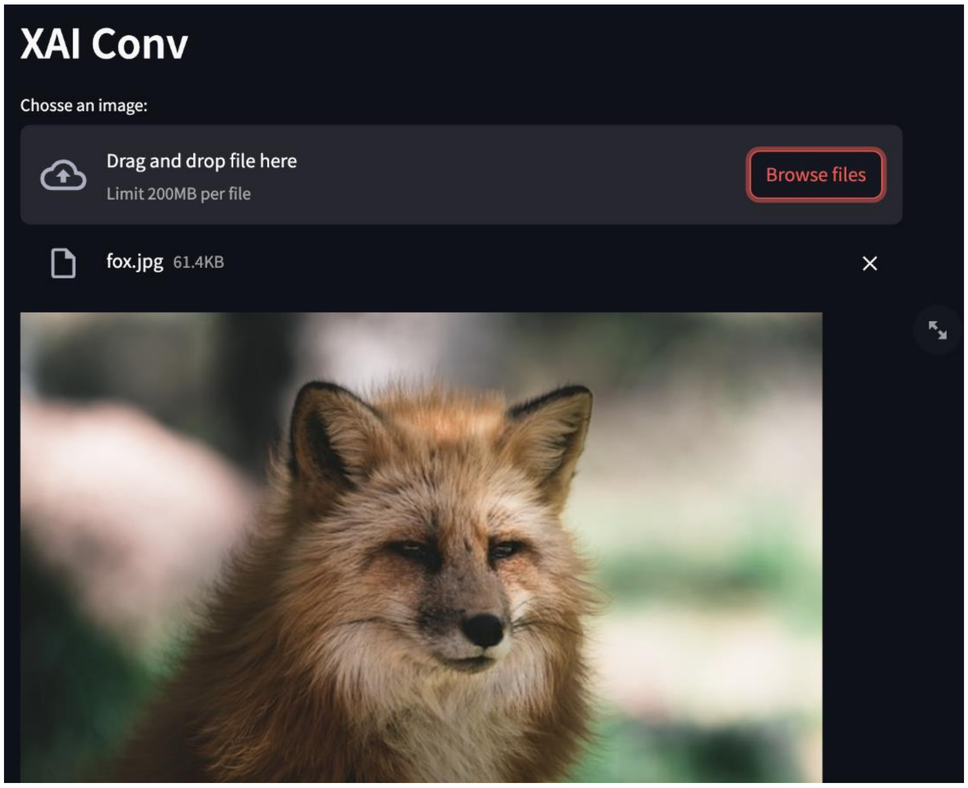
Hidden Layer Size:

Learning Rate:


Learning Rate:

Epochs:

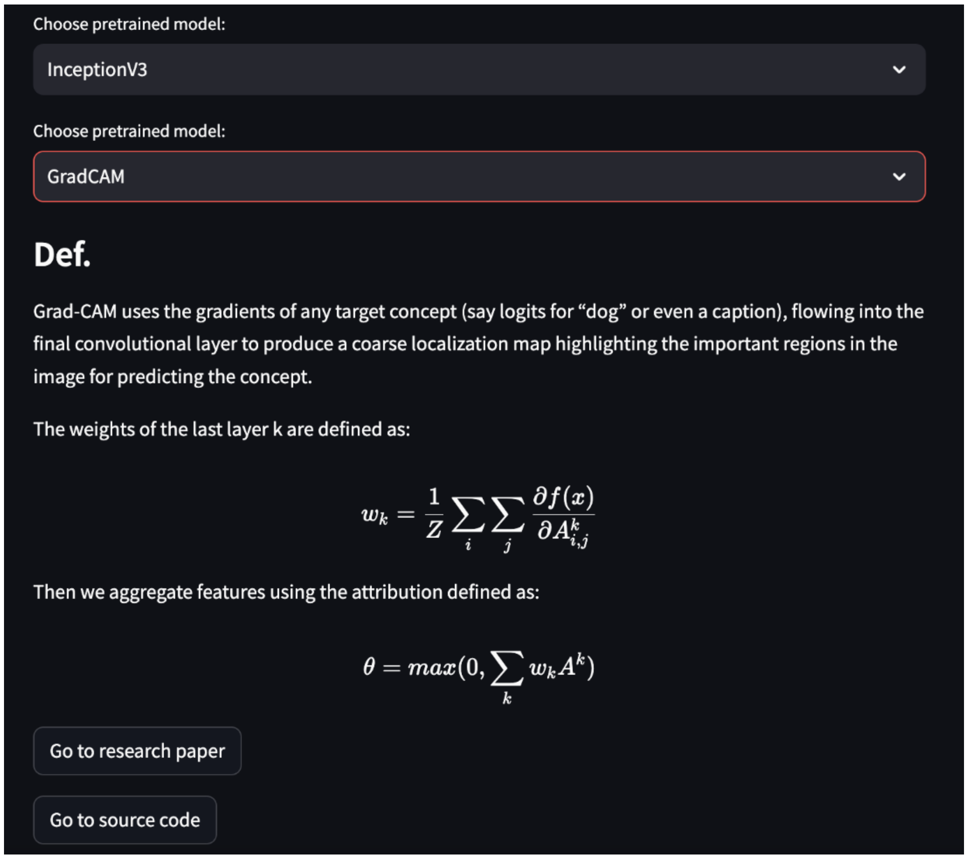



XAI Conv

Chosse an image:

Drag and drop file here
Limit 200MB per file

fox.jpg 61.4KB


Choose pretrained model:

InceptionV3

Choose pretrained model:

GradCAM

Def.

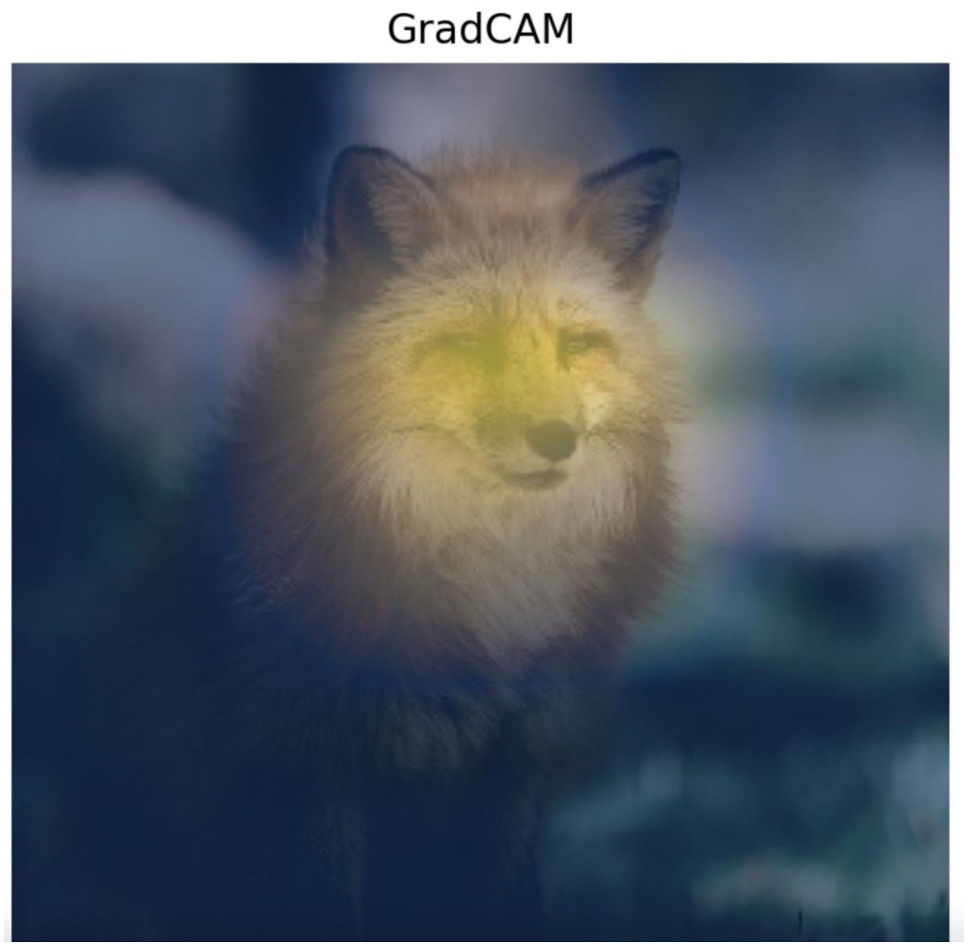
Grad-CAM uses the gradients of any target concept (say logits for “dog” or even a caption), flowing into the final convolutional layer to produce a coarse localization map highlighting the important regions in the image for predicting the concept.

The weights of the last layer k are defined as:

$$w_k = \frac{1}{Z} \sum_i \sum_j \frac{\partial f(x)}{\partial A_{i,j}^k}$$

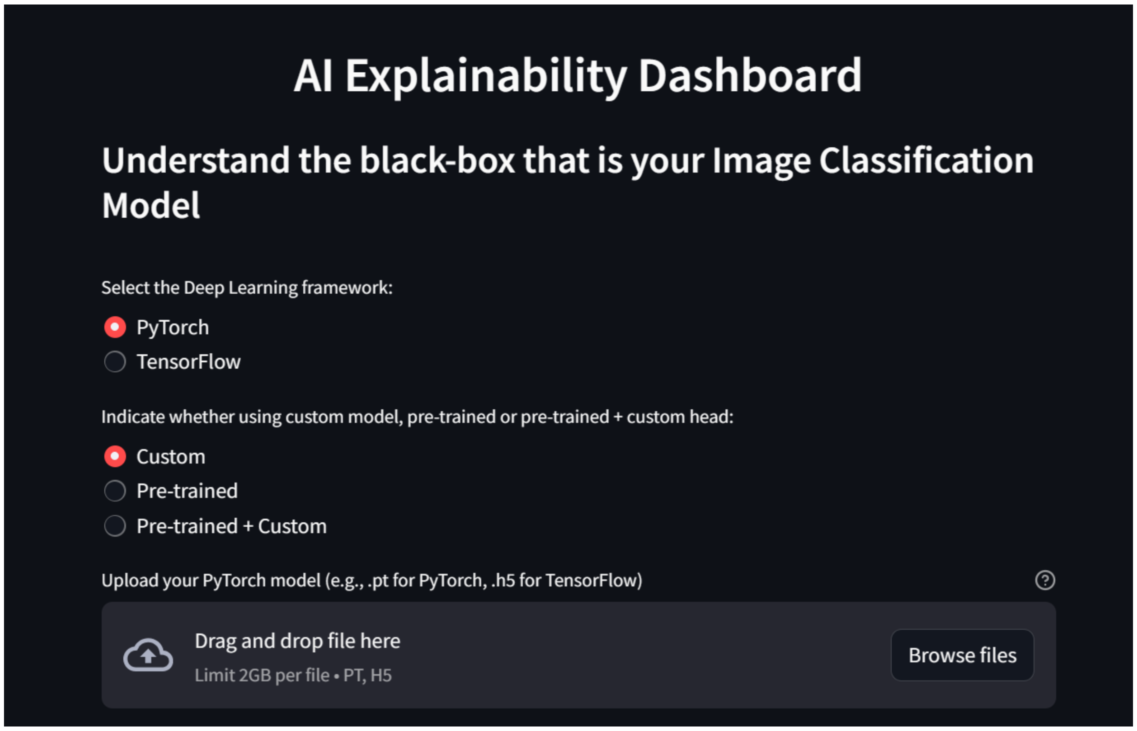
Then we aggregate features using the attribution defined as:

$$\theta = max(0, \sum_k w_k A^k)$$



XAI Product Offering:

The page for the XAI Product Offering looks like this –



AI Explainability Dashboard

Understand the black-box that is your Image Classification Model

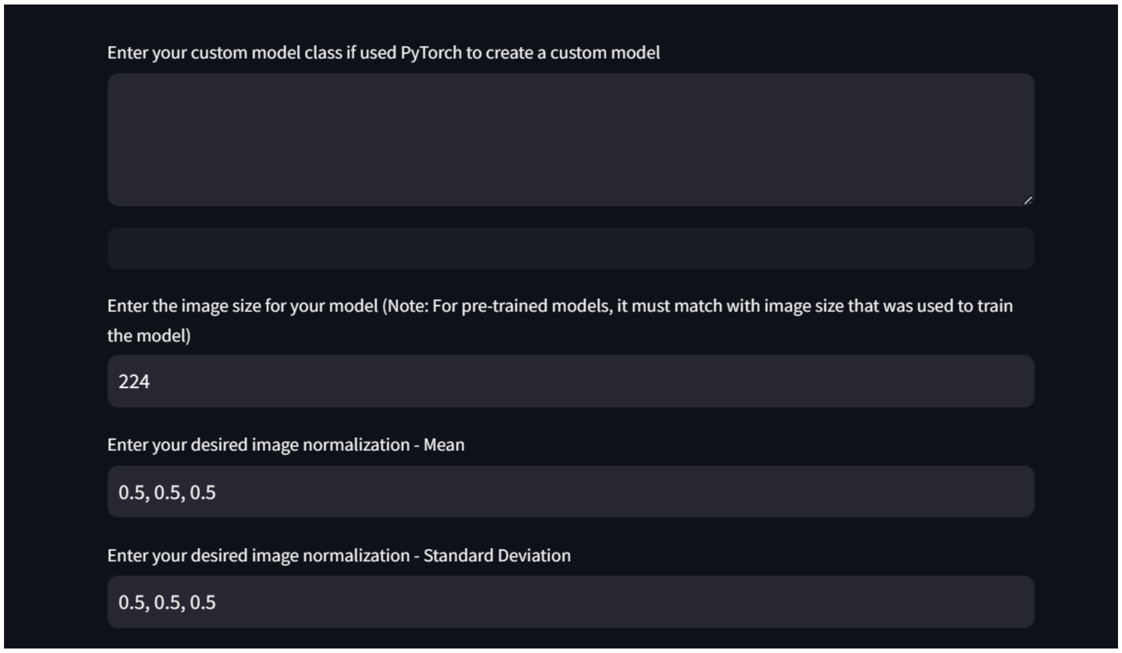
Select the Deep Learning framework:

[✓ PyTorch]
TensorFlow

Indicate whether using custom model, pre-trained or pre-trained + custom head:

[✓ Custom]
Pre-trained
Pre-trained + Custom

Upload your PyTorch model (e.g., .pt for PyTorch, .h5 for TensorFlow)



Enter your custom model class if used PyTorch to create a custom model

Enter the image size for your model (Note: For pre-trained models, it must match with image size that was used to train the model)

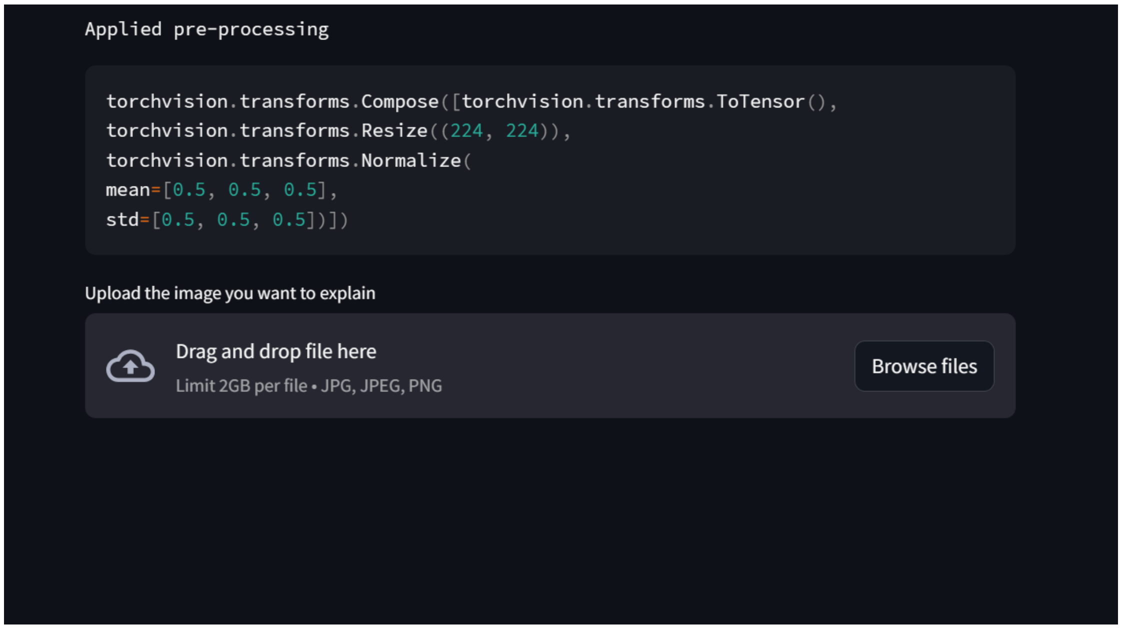
224

Enter your desired image normalization - Mean

0.5, 0.5, 0.5

Enter your desired image normalization - Standard Deviation

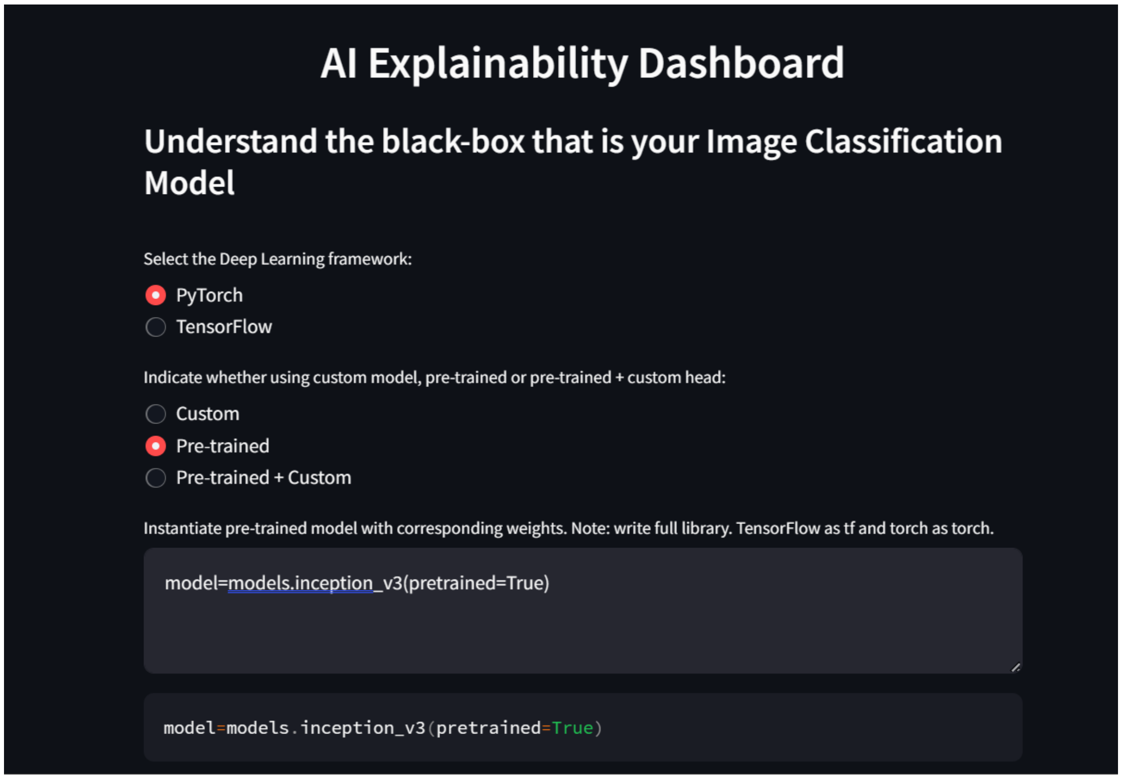
0.5, 0.5, 0.5


Applied pre-processing

```
torchvision.transforms.Compose([torchvision.transforms.ToTensor(),
torchvision.transforms.Resize((224, 224)),
torchvision.transforms.Normalize(
mean=[0.5, 0.5, 0.5],
std=[0.5, 0.5, 0.5])])
```

Upload the image you want to explain

Lime Demonstration: Utilized the problem and methodology defined in the LIME section above.


AI Explainability Dashboard

Understand the black-box that is your Image Classification Model

Select the Deep Learning framework:

- [✓ PyTorch]
- TensorFlow

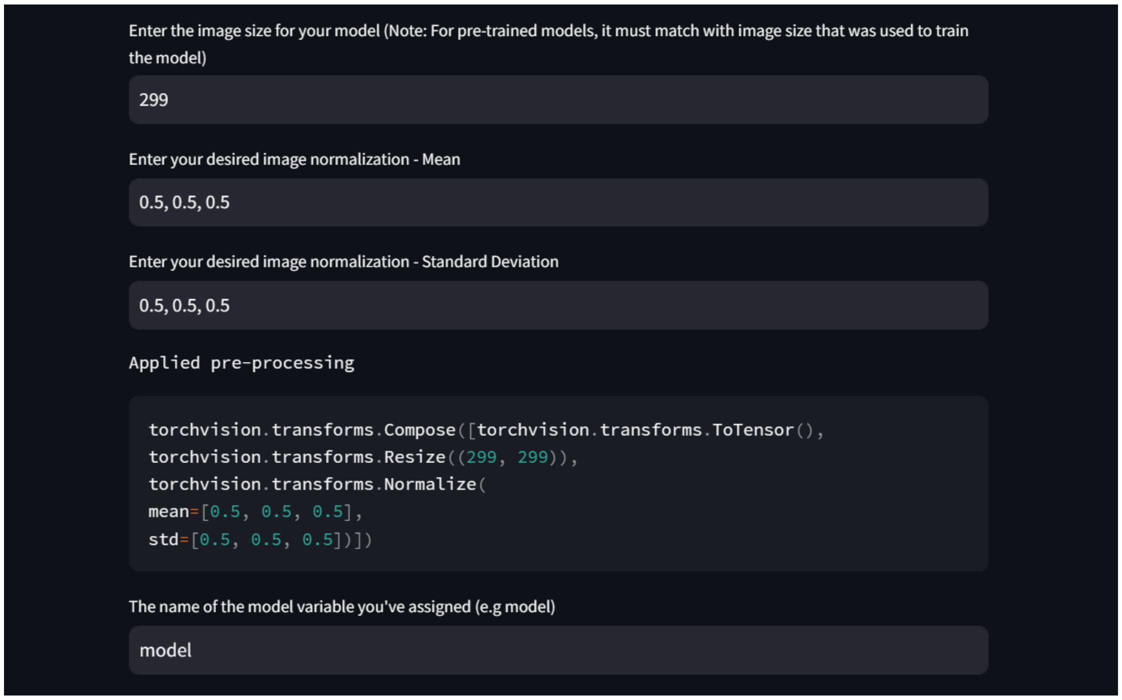
Indicate whether using custom model, pre-trained or pre-trained + custom head:

- Custom
- [✓ Pre-trained]
- Pre-trained + Custom

Instantiate pre-trained model with corresponding weights. Note: write full library. TensorFlow as tf and torch as torch.

model=models.inception_v3(pretrained=True)

```
model=models.inception_v3(pretrained=True)
```


Enter the image size for your model (Note: For pre-trained models, it must match with image size that was used to train the model)

299

Enter your desired image normalization - Mean

0.5, 0.5, 0.5

Enter your desired image normalization - Standard Deviation

0.5, 0.5, 0.5

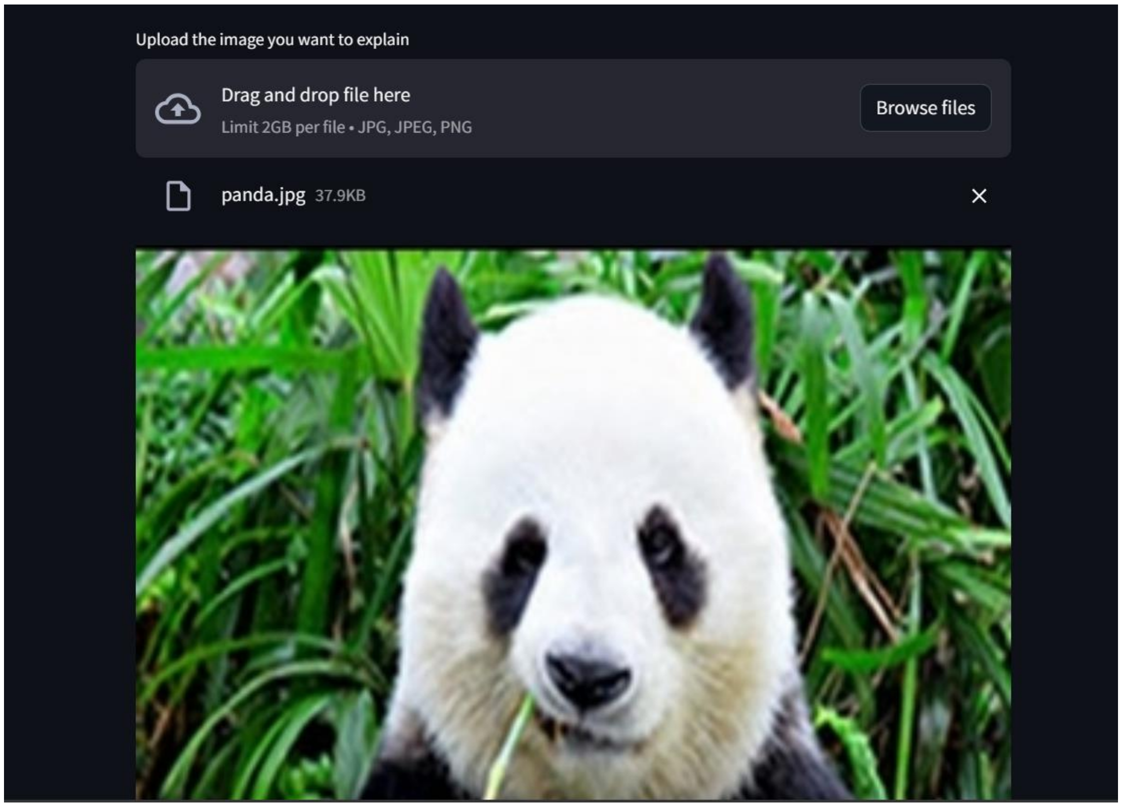
Applied pre-processing

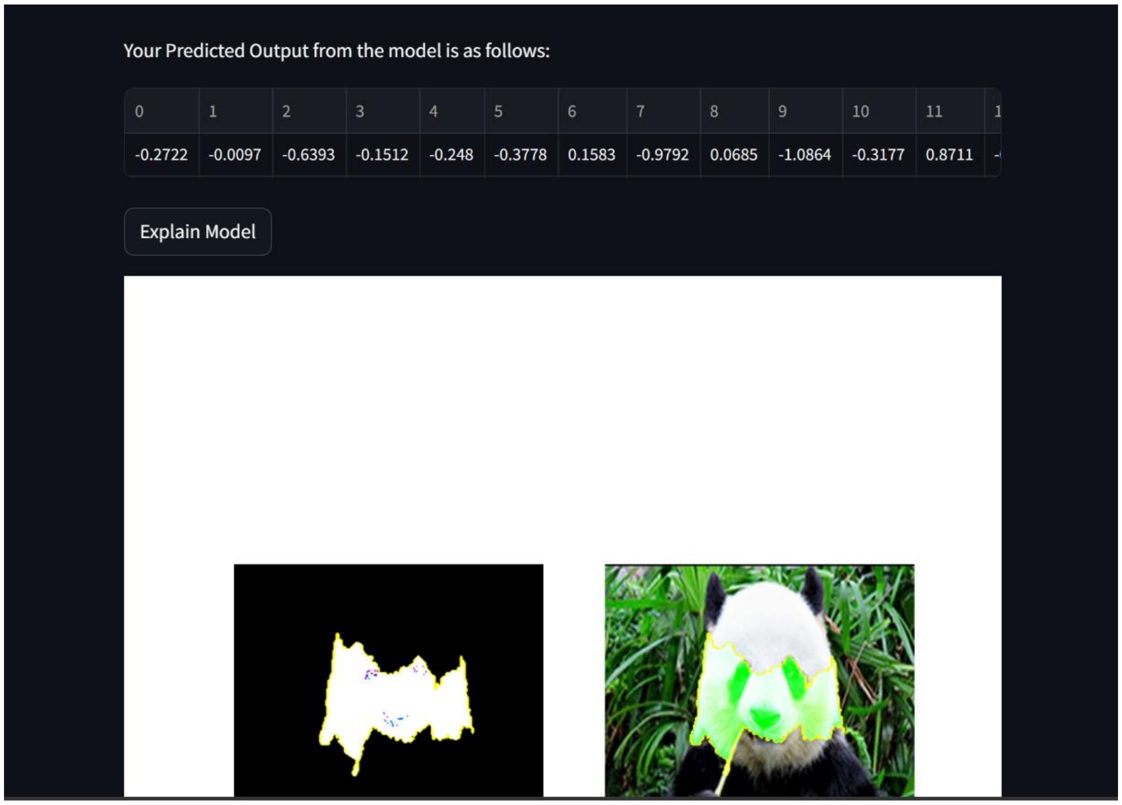
```
torchvision.transforms.Compose([torchvision.transforms.ToTensor(),
torchvision.transforms.Resize((299, 299)),
torchvision.transforms.Normalize(
mean=[0.5, 0.5, 0.5],
std=[0.5, 0.5, 0.5])])
```

The name of the model variable you've assigned (e.g model)

model

Pre-processing parameters need to match the pre-processing done while training the model.


Upload the image you want to explain



Your Predicted Output from the model is as follows:

0	1	2	3	4	5	6	7	8	9	10	11	1
-0.2722	-0.0097	-0.6393	-0.1512	-0.248	-0.3778	0.1583	-0.9792	0.0685	-1.0864	-0.3177	0.8711	-

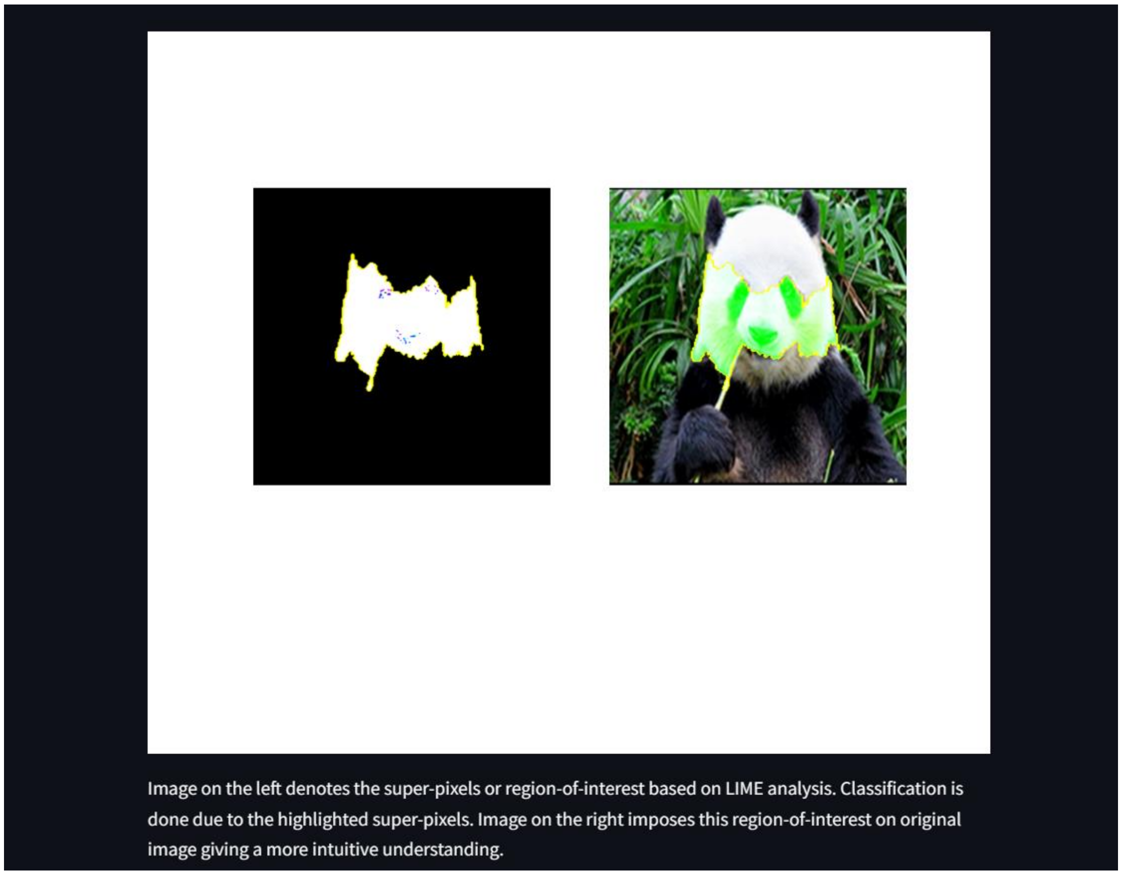
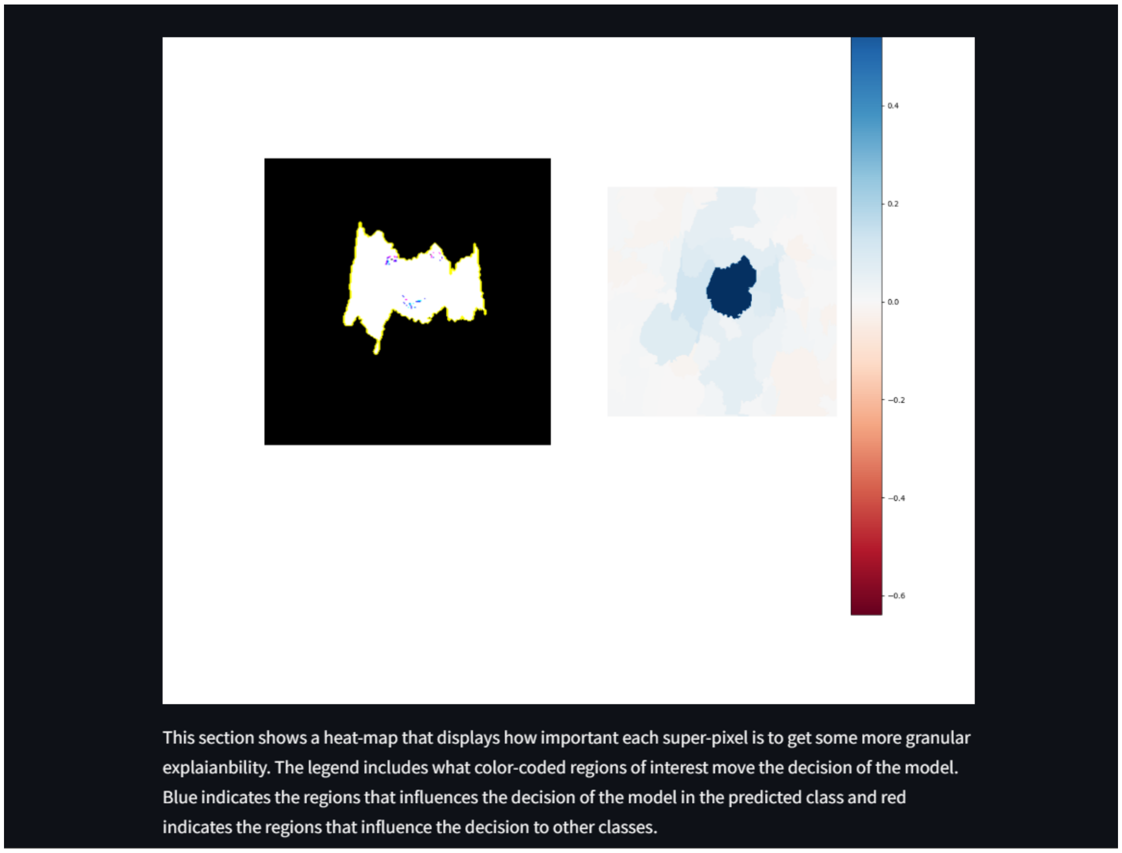


Image on the left denotes the super-pixels or region-of-interest based on LIME analysis. Classification is done due to the highlighted super-pixels. Image on the right imposes this region-of-interest on original image giving a more intuitive understanding.



This section shows a heat-map that displays how important each super-pixel is to get some more granular explaianbility. The legend includes what color-coded regions of interest move the decision of the model. Blue indicates the regions that influences the decision of the model in the predicted class and red indicates the regions that influence the decision to other classes.

The XAI analysis is done when prompted by clicking on the button. A detailed LIME analysis is presented at the end, and appropriate explanations for the plot are provided to help the user understand these results. This analysis helps the user to make the call as to whether their black box model has learnt the right features to be able to make the decision or not. If it hasn't, the modeling approach needs to change.

Conclusion

Future work – To integrate all XAI Python capabilities that are part of the demos into the XAI product offering.

References

- https://towardsdatascience.com/interpreting-image-classification-model-with-lime-1e7064a2f2e5

- https://towardsdatascience.com/how-to-explain-image-classifiers-using-lime-e364097335b4
- https://towardsdatascience.com/interpreting-image-classification-model-with-lime-1e7064a2f2e5
- https://github.com/PacktPublishing/Applied-Machine-Learning-Explainability-Techniques/blob/main/Chapter05/LIME_with_image_data.ipynb
- https://github.com/marcotcr/lime
- https://towardsdatascience.com/lime-how-to-interpret-machine-learning-models-with-python-94b0e7e4432e
- https://towardsdatascience.com/lime-vs-shap-which-is-better-for-explaining-machine-learning-models-d68d8290bb16
- https://www.kaggle.com/code/bygbrains/dog-cat-pandas-image-classifier
- https://deel-ai.github.io/xplique/latest/api/attributions/methods/grad_cam/
- https://arxiv.org/pdf/1610.02391.pdf
- https://deel-ai.github.io/xplique/latest/api/attributions/methods/occlusion/