

Explainability for Deep Learning Models

DATS 6303 Deep Learning Project Individual Report

Sailesh Baabu Suresh Babu

Abstract

Deep learning models are integral to critical applications across healthcare, finance, and autonomous systems, necessitating high levels of trust, transparency, and interpretability. This project addresses these needs by developing and assessing state-of-the-art explainability techniques. Our objectives include investigating and implementing feature attribution methods, saliency maps, and model-agnostic approaches to demystify model decisions. We aim to create user-friendly tools and visualizations that enhance the accessibility of model interpretations. The methodology encompasses a thorough literature review, and implementation of various explainability techniques like Integrated Gradients, SHAP, LIME, Grad-CAM, and SmoothGrad, followed by the development of an interactive platform compatible with major deep learning frameworks. We will evaluate these methods across various architectures and domains to determine their effectiveness in improving model transparency and user trust. The expected outcomes include a suite of validated explainability techniques and a comprehensive tool for interpreting deep learning predictions, ultimately contributing to informed decision-making and model refinement in practice. This project proposes a robust framework for enhancing the explainability of deep learning models, promoting broader adoption and trust in automated systems.

Outline of project

In this project, I have developed XAI across various machine-learning models for image classification. The final outcome of this is I have produced a complete no-code product that allows users to assess the robustness of image classification system using XAI tools.

What is my contribution?

- Delved deeply into the LIME XAI utility using Python. I developed a comprehensive no-code product that is compatible with both TensorFlow and PyTorch frameworks.
- This product allows users to upload any trained model, whether custom or pre-trained, along with a specific observation, to explore the inner workings of the model.

Detailed description of the contribution:

1. Lime
2. Shap

1. Lime:

LIME (**L**ocal **I**nterpretable **M**odel-agnostic **E**xplanations) is an XAI technique that explains the predictions of any classifier in an interpretable and faithful manner, by approximating it locally with an interpretable model. LIME works by perturbing the input data and observing the corresponding changes in the output, allowing users to understand how individual features affect predictions.

- Local refers to explanations that are specifically tailored to and valid for individual predictions made by a model.
- Interpretable refers to the ability of a model or an explanation to be easily understood by humans.
- Model agnostic means model independent. This can be used with any type of model (Pretrained, Custom, or Pretrained + Custom).
- I have implemented only pre-trained models with Pytorch and Tensorflow frameworks.
- Explanation - How individual input features contribute to a specific prediction.

How Lime Works:

Here I have used an example to explain it using a cat

Step 1: Select the Instance for Explanation

Suppose we have an image that our model has classified as a cat. We want to understand why the model made this decision.

Step 2: Perturb the Data

To explore how the model behaves near our specific instance (the original cat image), we create new images by making slight modifications. This could involve altering the brightness, adding noise, or slightly shifting parts of the image. Each of these new images is a perturbed version of the original.

Step 3: Predict Outcomes

We then input these perturbed images into the original model to see what it predicts for each. Maybe some altered images are still classified as a cat, while others are not.

Step 4: Weight the New Samples

Each perturbed image is given a weight based on its similarity to the original image. Images that are more similar to the original (less perturbed) have higher weights because the explanation focuses on the local environment around the original image.

Step 5: Fit an Interpretable Model

Using the perturbed images and their predictions, we fit a simple, interpretable model, such as a linear model. This model uses the perturbed data (features from images and their weights) to approximate how the original complex model behaves around the original instance. The goal is to predict the outcome (cat or not) based on these simplified inputs.

Step 6: Extract Explanation

From the simple model, we can now extract an explanation. For instance, the model might indicate that the presence of pointy ears and whiskers heavily influences the prediction of a "cat." If the model assigns high positive coefficients to these features, it suggests they are crucial for the model's decision to classify the image as a cat.

2. SHAP :

SHAP (**SH**apley **A**dditive **eX**planations) is an XAI technique derived from cooperative game theory, used to explain the output of any machine learning model. SHAP values provide a measure of the importance of each feature in a prediction, indicating how each contributes to pushing the model output from the baseline prediction to the current prediction. This method ensures fairness by distributing the "payout" (the prediction) fairly among the features based on their contribution.

- For SHAP implementation I have used Pytorch and Tensorflow as the framework.
- SHAP provides explanations for individual predictions, allowing users to understand how the model behaves at a specific instance.
- Unlike LIME, SHAP consider every pixel as a feature and gives the shapely value.
- SHAP is compatible with any model, offering flexibility to work with different machine learning frameworks and types of models, whether they are custom-built, pre-trained, or a combination of both.

- SHAP values explain the influence of each feature on the specific prediction by showing the contribution of each feature to the prediction.

How SHAP Works:

Using a similar example with an image classified as a cat:

Step 1: Select the Instance for Explanation

We start with the same image of a cat that our model has classified as containing a cat. We aim to uncover the reasoning behind this classification.

Step 2: Compute SHAP Values

SHAP values are computed for each feature of the input image. This involves using permutations of the input features to evaluate the change in the prediction output when a feature is present versus when it is absent.

Step 3: Aggregate Contributions

For image data, this might involve considering how specific regions of the image (like pixels or segments) contribute to the model's prediction. This aggregation can be visualized as a heatmap where areas that significantly influence the prediction are highlighted.

Step 4: Visualize the Explanation

The SHAP values can be visualized to show the impact of each region of the image. For example, regions with high SHAP values that positively affect the prediction (like the presence of whiskers or ears) will be highlighted, indicating their importance in the classification decision.

Step 5: Interpret the Results

The visualization helps us understand which features (or image regions) are most influential. If SHAP values show high positive values for areas capturing the cat's ears and whiskers, it suggests these features are key determinants in recognizing the image as a cat.

Experimental Setup:

For this project, I have chosen Streamlit as the Python framework for developing web applications to build interactive dashboards for the LIME and SHAP demonstrations. Streamlit is a popular Python library that enables developers to quickly create and share beautiful, interactive web applications for data science and machine learning projects, using only Python scripts without the need for complex web development skills.

Outcome:

LIME Dashboard:

AI Explainability Dashboard

LIME for Image Classification Model

Select the Deep Learning framework:

☐ PyTorch

☒ TensorFlow

Pretrained

☒ Pre-trained

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

```
model = tf.keras.applications.MobileNetV2(weights='imagenet')
```

```
model = tf.keras.applications.MobileNetV2(weights='imagenet')
```

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

Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG

Browse files

photo-1560275619-4662e36fa65c.jpg 166.4KB

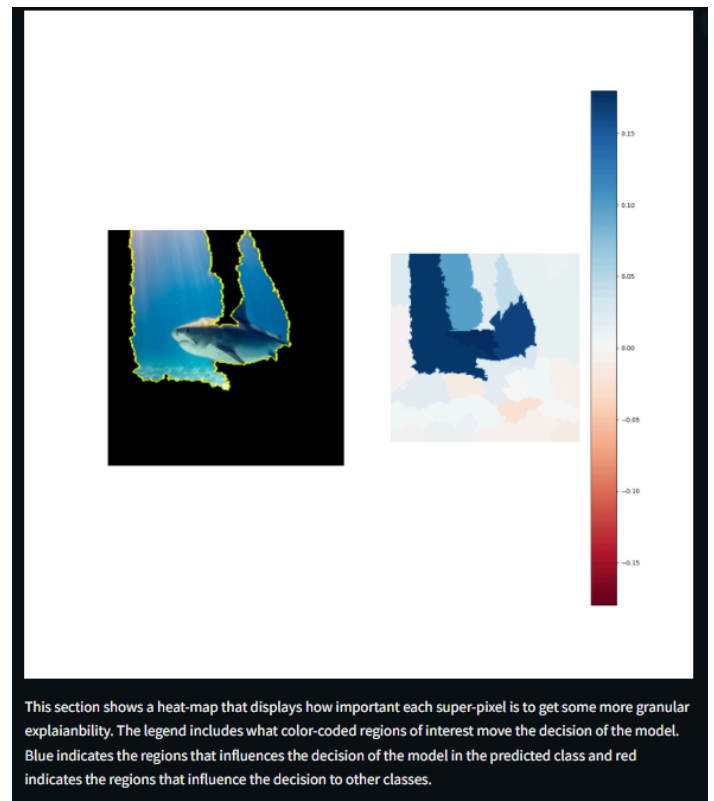
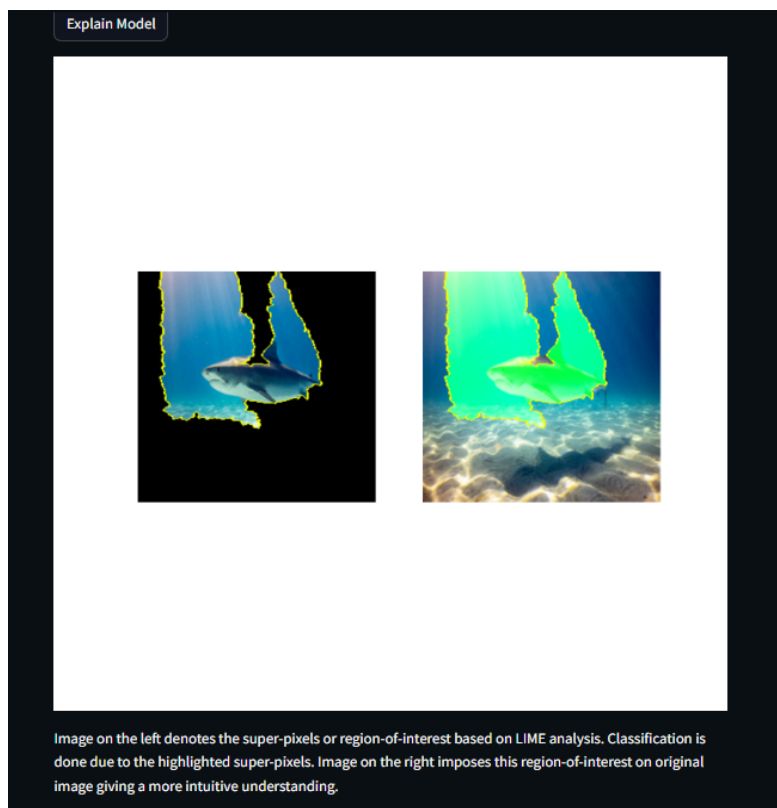


Uploaded Image

Your Predicted Output from the model is as follows:

0	1	2	3	4	5	6	7	8	9	10	11	12
0.0002	0.0002	0.0029	0.8631	0.0201	0.0021	0.0046	0	0.0001	0.0001	0.0001	0.0001	0.0001

I have chosen Tensorflow as the framework and mobilenetv2 as the pre-trained model. Kept the image size at 224, and added a few normalization to the image. Then I used a shark as the image.

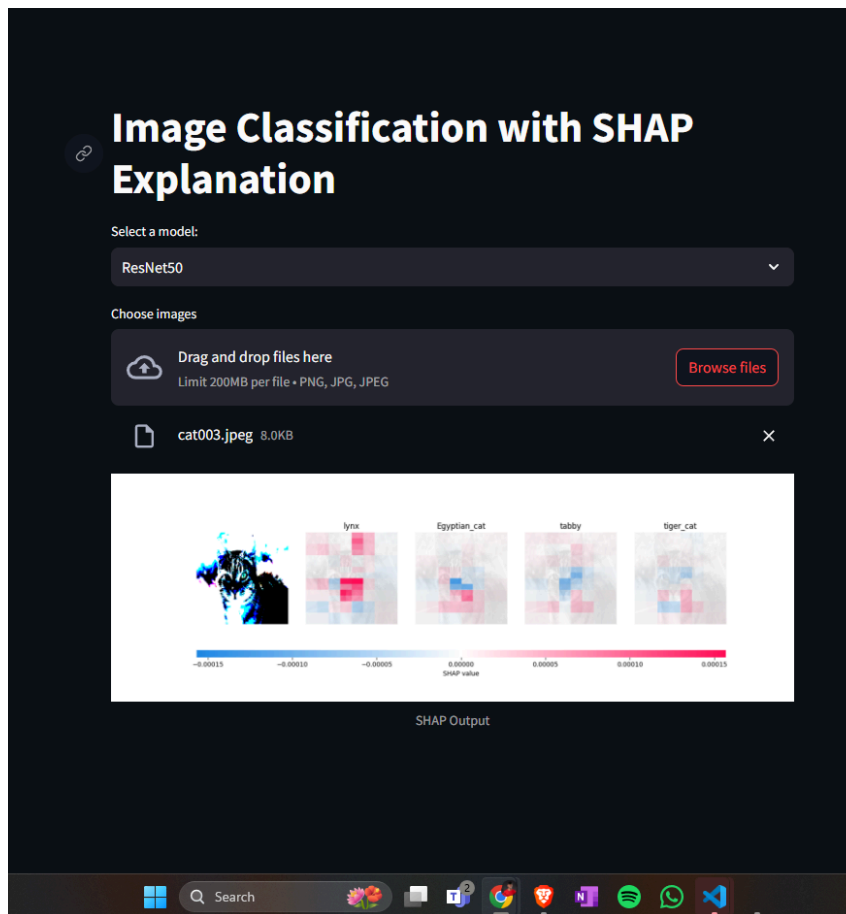


The image on the left has formed the super-pixels or regions of interest based on the analysis.

The image on the right imposes this region of interest on the original image which is giving us more understanding.

The final output can be seen on the right side image with a bar value varying from red to blue.

SHAP Dashboard:



For the SHAP image classification, I have used 2 pre-trained models ResNet50 and VGG16.

I have chosen the ResNet50 as the pre-trained model and cat as my dataset image. The output, it's segmented into pixels, here all the pixels will be considered for the overall prediction of the model.

The below bar values represent the **Shapely values**, originates from cooperative game theory and is used to fairly distribute the total payoff to players depending on their contribution to the overall game. In the context of machine learning, Shapley values are used to determine the contribution of each feature in a model's prediction.



Conclusion:

I have acquired a deep understanding of XAI tools, particularly LIME and SHAP, which is essential for any successful data scientist to grasp. These XAI's will be used to understand the black box by everyone. I also want to thank my teammates who chose to introduce me to this new topic **Explainability AI**.

Future Work: I will try to implement these XAI's into the products.

References:

▶ Explainable AI explained! | #3 LIME - Youtube Explanation

▶ SHAP values for beginners | What they mean and their applications - Lime and Shap

["Why Should I Trust You?": Explaining the Predictions of Any Classifier](#) - Paper (Why should I trust You?)

[XAI - LIME: Binary Dog and Cat Image Classifier Model](#) - GitHub Repo

[the SHAP documentation](#) - SHAP

Code Contribution:

Lines of code I modified in SHAP_XAI - 80%

Lines of code I modified in LIME_XAI - 65%