DATS - 6303

DEEP LEARNING

Final Project - Individual Report

By

Manoj Padala

Explainability in Deep Learning - XAI Models

Overview of this project:

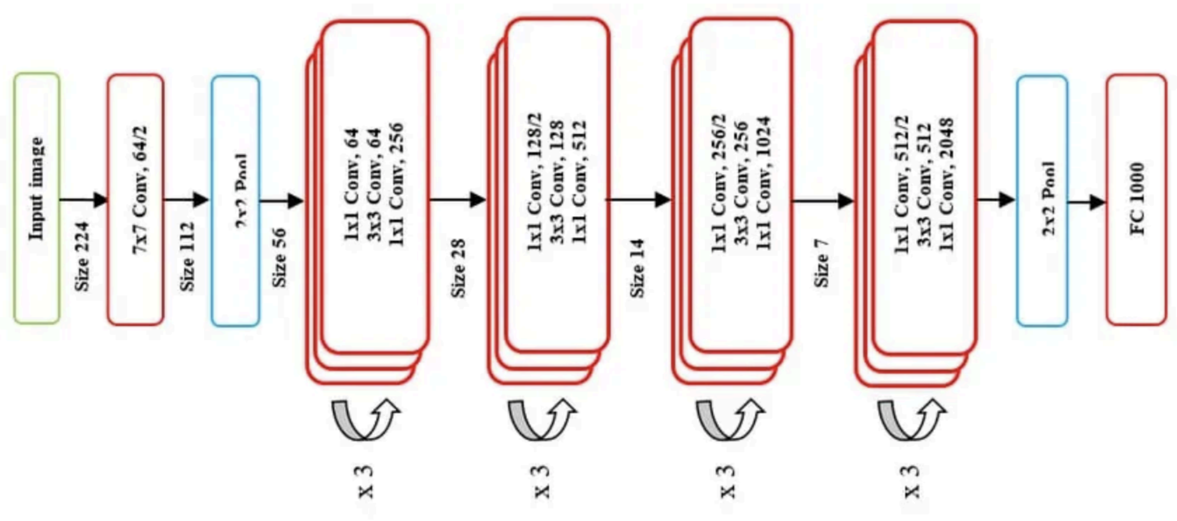
This project addresses these needs by developing and assessing state-of-the-art explainability techniques. Our objectives include investigating and implementing feature attribution methods, saliency maps, and model-agnostic approaches to demystify model decisions.
We aim to create user-friendly tools and visualizations that enhance the accessibility of model interpretations.
The methodology encompasses a thorough literature review, implementation of various explainability techniques like Integrated Gradients, SHAP, LIME, Grad-CAM, and Occlusion Sensetivity, 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.

Below are the topics which I worked on and helped my project mates while debugging their part of code:

1. GradCam.
2. Occlusion Sensitivity.

Initially, I have built a custom CNN model and tried to train images using Imagenet Data set.
But I didn't get results and it was taking a long time to train the data and instance was getting crashed.

Since the project aim to understand the explainabilty of AI, we dropped training our custom model and used Resnet50 as it was proven as a best pretrained model to classify images



Model Used: Resnet50:

Use of each layer:
Convolution layers:
Used to extract features from the input image by applying a series of convolutional filters to the image. Each convolutional filter is a small matrix of weights that is applied to a local region of the input image. The filter slides over the entire image, producing a feature map that highlights patterns and edges in the image. T**he weights of the filter are learned during the training process.**

Relu:
ReLU is a piecewise linear function, which makes it computationally efficient to compute and differentiate and effective way to introduce nonlinearity into the output of a neuron.

Global Average pooling:
This layer computes the average of each feature map, resulting in a feature vector with the same number of channels as the number of filters in the last convolutional layer. Reduces the spatial dimensions of the output tensor to a vector by applying global average pooling.

1st Stage:

First convolutional layer with 64 filters, kernel size (7, 7), and stride (2, 2).
Batch normalization and ReLU activation.
Max pooling with pool size (3, 3) and stride (2, 2).

2nd Stage:

9 convolutional layers, each with batch normalization and ReLU activation.
Identity blocks: Skip connections between layers.
First convolutional layer: 64 filters of size 1x1.
Second convolutional layer: 64 filters of size 3x3.
Third convolutional layer: 256 filters of size 1x1.
Skip connection from previous stage (input) to last layer.

3rd Stage:

12 convolutional layers, each with batch normalization and ReLU activation.
Projection block: If dimensions change, adjust using 1x1 convolutions.
First convolutional layer with 128 filters, 1x1 kernel, stride 2.
Followed by 3x3 conv with 128 filters, and 1x1 conv with 512 filters.

4th Stage:

18 convolutional layers, each with batch normalization and ReLU activation.
Projection block if dimensions change.
Convolutional layers to extract deep features.

5th Stage:

9 convolutional layers, each with batch normalization and ReLU activation.
Projection block if dimensions change.
First convolutional layer reduces size using 1x1 convolutions.
Final layers for deep feature extraction.

Global Average Pooling:

Reduces spatial dimensions to a vector.
Computes average of each feature map.
Output is a feature vector with same number of channels.

Resnet 18 works as similar to the above but resent50 is proven to provide more accuracy, So we chose resnet50 to understand the blackbox.

Here is how the model predicted that given image belongs to particular class
preds=model.predict(image)

The predict() method returns an array of predictions.

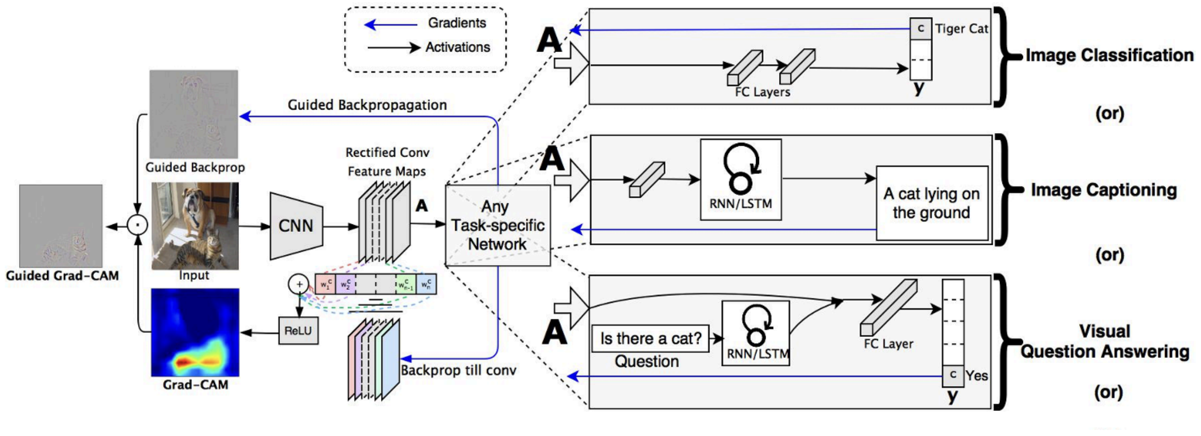
And we have used **np.argmax()** to find the index of the highest probability prediction in the **preds[0]** array. The index i corresponds to the class with the highest predicted probability and the model calculates the probability depends on the features of the image and also how the model was trained.

predicted_class = decode_predictions(preds, top=1)[0][0]

top=1 argument indicates that it only returns the top 1 prediction.

GradCAM:

Gradient-weighted Class Activation Mapping



Above architecture shows how the Gradcam works,

Grad-CAM leverages the activations of the last convolutional layer in the network. These activations capture high-level features learned by the network(Pretrained - Resnet50 in our case) during training, which are crucial for making predictions. Grad-CAM relies on the ability of a neural network to learn and extract features from input images.

Below are the steps on how the gradcam algorithm works,
Forward Pass:

Feed the input image through the CNN to obtain the feature maps of the last convolutional layer.
Let A^k be the activations of the last convolutional layer for the input image, where k is the index of the feature map.

Gradient Calculation:
It computes the gradients of the target class score Y_c with respect to the activations A^k of the last convolutional layer.

(This makes the main difference between CAM and GradCam)

Global Average Pooling (GAP):
Then it the computes the importance of each feature map by taking the global average of the gradients.

$$\alpha_k = \frac{1}{Z} \sum_i \sum_j \frac{\partial Y_c}{\partial A_{ij}^k}$$

Weighted Combination:

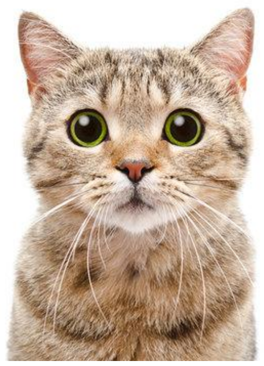
It weights the feature maps by their rank (according to weights)
And then it applies ReLU to the weighted combination and upsample it to match the dimensions of the input image.

This produces a weighted combination of the feature maps, emphasizing the regions that are most relevant for predicting the target class.

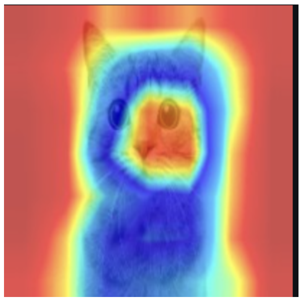
Visualization:
The heatmap is overlaid onto the input image to visualize the important regions. Typically, the heatmap is scaled and blended with the input image to produce a visually interpretable result.
The heatmap indicates which parts of the input image are most influential in the network's decision-making process for the target class.

Results:

Input Image:



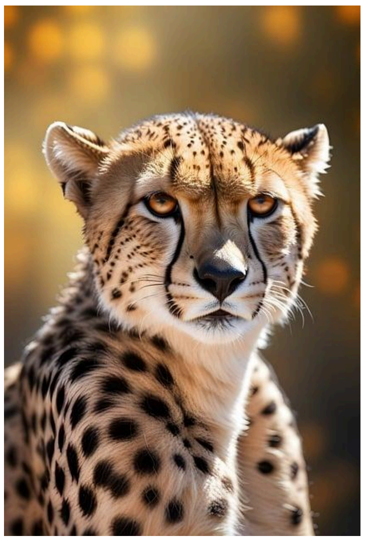
GradCAM HeatMap



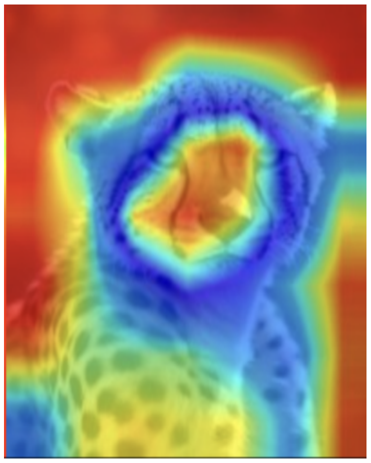
As we can identify from the above pictures that to identify for the model as Tabby (a breed in card), from the gradcam is from the features in Face,

Let’s give input image as cheetah and see the overlay map of Gradcam

Input Image:



GradCAM Heatmap:



In the above apart from identifying the features on the face of the cheetah, it also takes the input from the skin (Black marks).
By this GradCAM we can draw a consluion that our model has learned the importance black marks on the cheetah to identify it as a cheetah.

Occlusion Senstivity:

The use of Occlusion Sensitivty is to understand the importance of different parts of an image in the model's decision-making process.

Here is how Occlusion Sensitivity works,

The occlusion value is used to define the value that replaces occluded regions of the image, allowing the algorithm to assess the importance of different parts of the image for the model's predictions.

We will occlude different parts of the image by sliding the occlusion window across the image with a given stride. (This means that part of the image is effectively hidden or covered.)

The occluded image is then fed through the model, and the model's prediction is observed.

For each patch in the image, the model's prediction probability for the target class is calculated after occluding that patch. The probability represents the model's confidence in the presence of the target class when that patch is occluded. The higher the probability, the more important that patch is for the model's prediction of the target class.

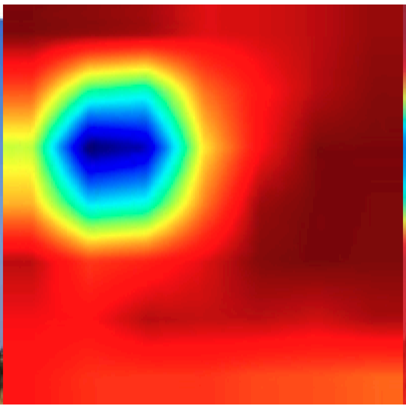
These probabilities are stored in a heatmap where each pixel's value represents the model's confidence when the corresponding patch is occluded. The heatmap is then normalized to values between 0 and 255 for visualization.

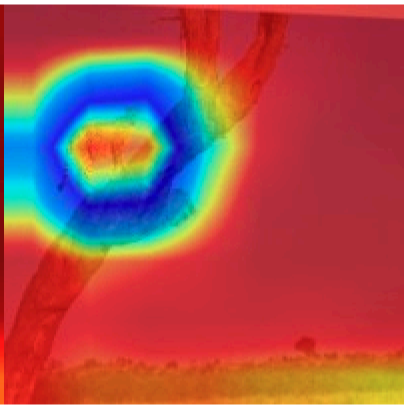
By comparing the model's predictions with and without occlusion, the algorithm can understand how important each part of the image is for the model's decision-making process.

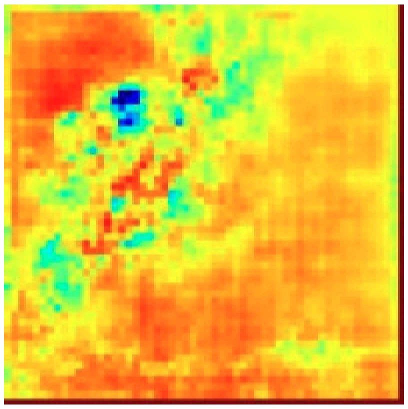
Predicted Class: cheetah

Probability: `0.880088`

Original Image


GradCAM Heatmap


GradCAM Overlayed Image


Occlusion Sensitivity Heatmap

Here is the Occlusion Sensitivty Heatmap for the cheetah in forest.

Summary and Conclusions:

I have developed a Streamlit application to deliver eXplainable Artificial Intelligence (XAI) models, specifically Grad-CAM and Occlusion Sensitivity. These models help users identify features crucial for our model to classify an object in an image as a particular class with associated probabilities.

This study can be further improved by constructing a custom Convolutional Neural Network (CNN) and training it on images of specific categories, such as cancer. By doing so, we can observe the explainability provided by the network. If the model's explanations are validated by medical professionals and proven effective, the model can be utilized for cancer detection.

Code:

I have used internet for the syntax but the code was built by me integrating with the stream lit application.

references:
https://www.youtube.com/watch?v=Y8mSngdQb9Q&ab_channel=SirajRaval
https://medium.com/@aimldl1984/explainable-ai-grad-cam-8acd04dd2dd3