Explainable AI - Project 1

Sebastiano D'Arconso (VR489066) - Tommaso Del Prete (VR488382)

February 5, 2024

Abstract

This project aims to implement a 2D Convolutional Neural Network (CNN) for image classification and then to use two eXplainable AI algorithms to extract the most important areas from the images. After that, a statistical analysis is performed on the final attributions. The eXplainable AI methods used in this project are the Integrated Gradients method (one implemented by us and one provided by the Captum library) and the LIME attribution method for post-hoc analysis. The dataset used is the Muffin vs Chihuahua dataset. The statistical analysis relies on the study of the histogram plots of each attribution and the correlations between each attribution.

1 Introduction

In this project we had to:

- Select a dataset for image classification
- Implement a Convolutional Neural Network
- Implement the Integrated Gradients algorithm
- Perform a post-hoc analysis with the Integrated Gradients method provided by the Captum library
- Perform post-hoc analysis using LIME (or SHAP) attribution method
- Compare the attributions extracted with the methods
- Comment the results

2 Dataset - Muffin vs Chihuahua

This dataset is composed of around 6000 images (5917) of chihuahuas and muffins. The images are scraped from google images and the count is around 2600 for muffins and 3400 for chihuahuas, one first basic preprocessing is performed in order to convert the non-RGB images in RGB and to assign the correct label to each photo (0 for muffins and 1 for chihuahuas). We then calculated the mean and standard deviation for the RGB values of each image in order to use those values later for the transformations.

2.1 Stratified K-Fold

To perform the k-fold subdivision we relied on the stratified k-fold function of sklearn, dividing the dataset in 5 different folds. This method Provides train/test indices to split data in train/test sets, the number of folds has been decided taking into account the resources available and also the fact that common values for "small" dataset are 5 or 10 folds. This cross-validation object is a variation of k-fold that returns stratified folds, the folds are made by preserving the percentage of samples for each class. We then trained the model on each fold and compared the results in order to keep the model with the best validation accuracy and to calculate the average performance of the model.

3 Model - CNN

The model we decided to implement is a 3-layers deep CNN, with batch normalization and max pooling in each layer except the last one (the classification layer). The model has been trained for 50 epochs on each of the 5 folds, for each fold we kept track of the losses and accuracy, saving them for each epoch and saving the model every time that it performs better in terms of validation accuracy, so at the and we had 5 models, one for each fold.

4 Post-hoc analysis - Integrated Gradients

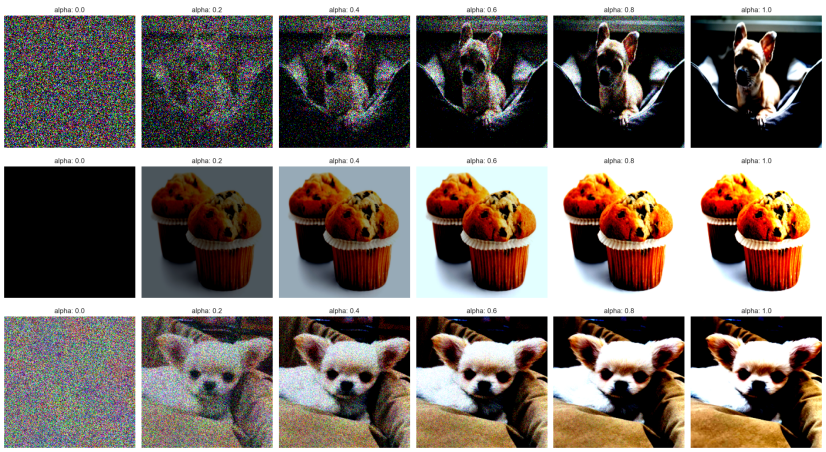
Integrated gradients represent the integral of the gradients with respect to inputs along the path from a given baseline to output. The integral can be approximated using a Riemann Sum. In the implementation we used the Trapezoidal rule.

$$IntegratedGrads_i^{approx}(x) ::= \overbrace{(x_i - x_i')}^{5.} \times \overbrace{\sum_{k=1}^{m}}^{4.} \frac{\overbrace{\partial\, F(x' + \overbrace{\frac{k}{m}}^{1.} \times (x - x'))}^{3.}}{\overbrace{\partial x_i}^{2.}} \times \overbrace{\frac{1}{m}}^{4.}$$

1. Generate alphas α
2. Generate interpolated images $= (x' + \frac{k}{m} \times (x - x'))$
3. Compute gradients between model F output predictions with respect to input features $= \frac{\partial F(\text{interpolated path inputs})}{\partial x_i}$
4. Integral approximation through averaging gradients $= \sum_{k=1}^{m}$ gradients $\times \frac{1}{m}$
5. Scale integrated gradients with respect to original image $= \sum_{k=1}^{m}$ gradients $\times \frac{1}{m}$. The reason this step is necessary is to make sure that the attribution values accumulated across multiple interpolated images are the same units and faithfully represent the pixel importance on the original image.

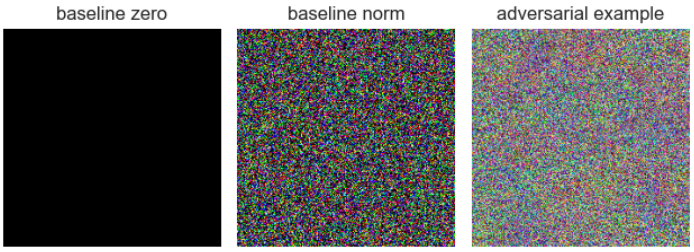
Each of the parts of the integrated gradients formula have been implemented separately and then merged together to perform the algorithm and can be seen in the **utils.py** file.

Example of interpolated images, first with normal baseline, then with zero baseline and the last with adversarial baseline (baseline with 50/50 output from the model):



4.1 Baseline choices

To be able to objectively determine which pixels are important to our predicted label, we'll use a baseline image as comparison. A good baseline is one that contains neutral or uninformative pixel feature information, and there are different baselines that can be used. In our experiments we tried with a black baseline, a noise baseline and a baseline chose using a method that helps to find a baseline that is completely uninformative. We called the last one "adversarial baseline" since is found using an adversarial method, in which we set a goal and then change the values of a random image for some fixed steos until the model scores 50/50 on that image. In the example seen in this report we started from a noise baseline, but with the code we can show how the results change when we use a random image taken from the dataset.

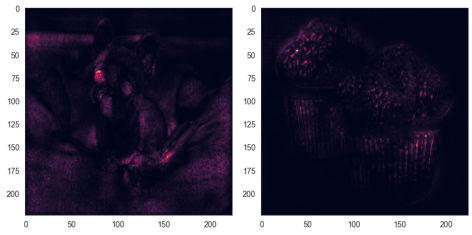


5 Post-hoc analysis - LIME

LIME stands for *local interpretable model-agnostic explanations*, it's a post hoc, perturbation-based explainability technique. To explain why the model makes a certain prediction for a given input instance, the LIME algorithm works by passing lots and lots of slightly perturbed examples of the original input to the model and then measures how the model predictions change with these small input changes. The perturbations occur at the feature level of the input instance, pixels and pixel regions are modified to create a new perturbed input. In this way, those pixels or pixel regions that most influence the model's prediction are highlighted as being more or less influential to the model's predicted output for the given input instance.

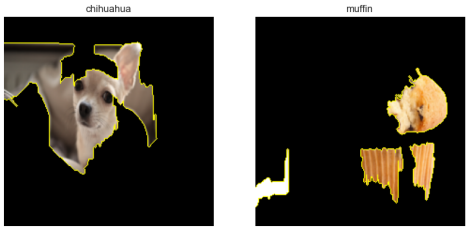
6 Attributions

In **gradient based methods** such as integrated gradients the goal is to estimate the attribution maps. An attribution map captures the importance of each input feature (in our case, pixels) for a specific output class.



(a) Chihuahua (b) Muffin

In **perturbation methods**, such as LIME, the method provides explanations for individual predictions by highlighting the important regions (superpixels: small, non-overlapping regions) in the input image that influenced the model's decision. The output of LIME for an image classification model like our case is typically a set of superpixels with assigned weights, indicating their influence on the model's prediction.

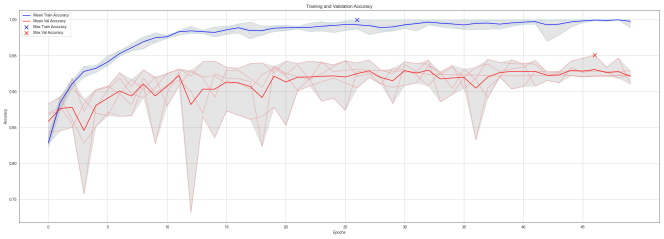


7 Results

After the training we performed k-fold cross validation to obtain the mean accuracy of the model as well as the F1 score across the k-folds.

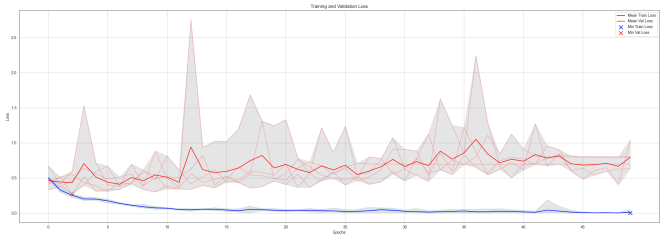
7.1 Accuracy

Max train	Mean train	Max val	Mean val
100%	97%	95%	91%



7.2 Loss

Min train	Mean train	Min val	Mean val
0.0	0.064	0.272	0.664



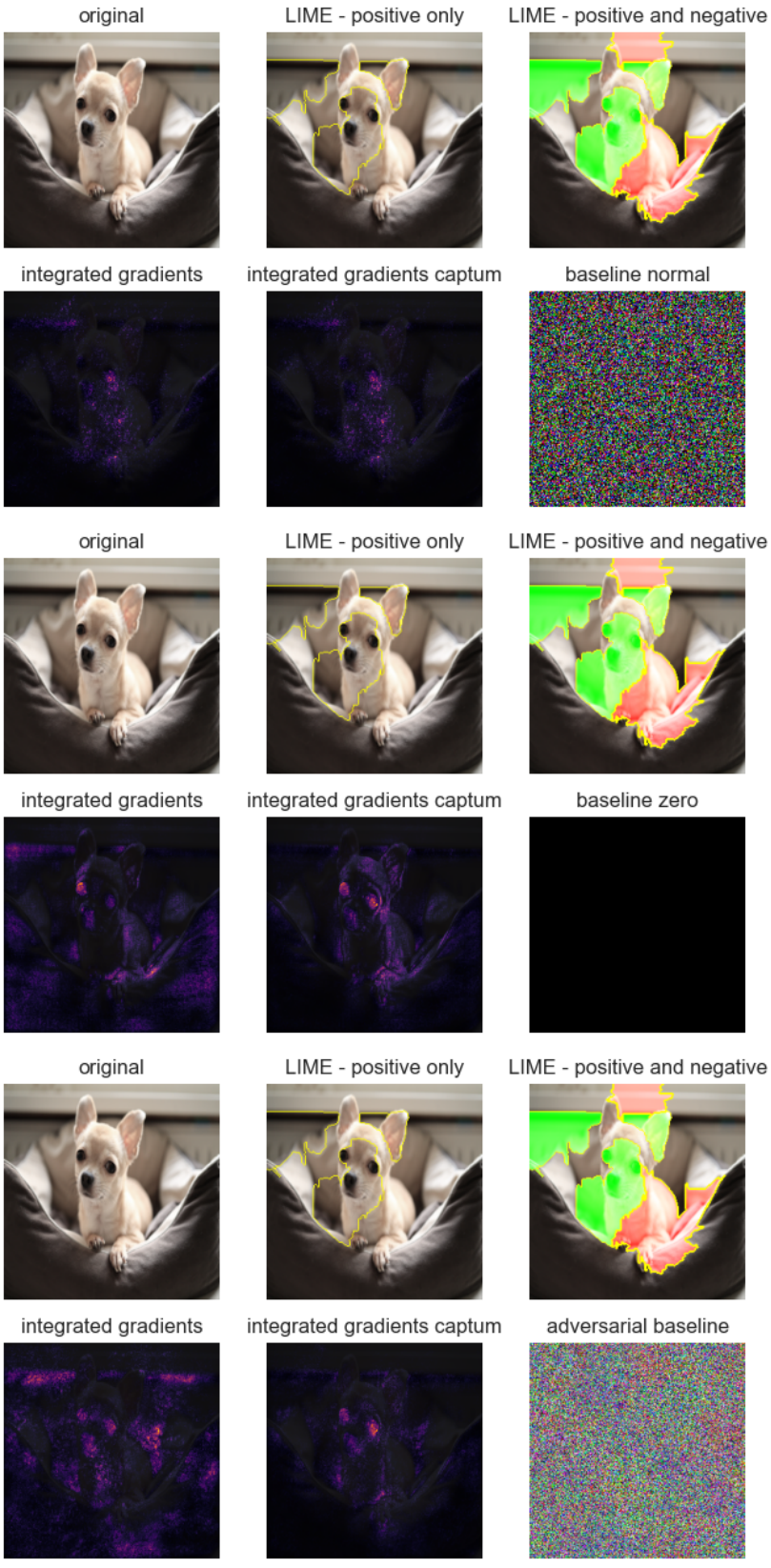
7.3 F1 score

For calculating the F1 score we used the f1 score function provided by sklearn.

Max/mean	0	1	2	3	4
Max	0.945	0.951	0.943	0.941	0.944
Mean	0.911	0.928	0.911	0.915	0.892

Total mean F1 score across the folds: 0.912.

7.4 Results on one sample image


original
LIME - positive only
LIME - positive and negative
integrated gradients
integrated gradients captum
baseline normal
original
LIME - positive only
LIME - positive and negative
integrated gradients
integrated gradients captum
baseline zero
original
LIME - positive only
LIME - positive and negative
integrated gradients
integrated gradients captum
adversarial baseline