EXPLAINABLE AI - Project 2
Follow-up
VISUAL INTELLIGENCE - Project 1

Author:
Serena De Antoni, Gaetano Alberto Caporusso, Enrico Zorzi

Abstract

Implementation of a 2D Convolutional Neural Network (CNN) and a ScatNet for image classification, comparison of the parameters for understanding which is the best for the task under analysis, and display the filters extracted from the CNN and the ScatNet. Reduction of the number of images until ScatNet performs better than CNN. Finally, application of XAI algorithms (IG, LIME and SHAP) on the two networks implemented and a statistical analysis on the final attributions.

1 PIPELINE

- Select a dataset of our choice.
- Implement the Convolutional Neural Network.
- Perform k-fold cross validation to assess the model's generalization ability and compute the mean accuracy and the mean F1 score across the k folds.
- Implement the ScatNet using the same classifier used for the CNN.
- Perform k-fold cross validation also for the ScatNet like the point 3 for the CNN, and extract the mean accuracy and F1 score across the k folds.
- Extract the filters from both models.
- Reduce the number of images to see when ScatNet works better than CNN.
- Implement the Integrated Gradients XAI algorithm from scratch and perform a post-hoc analysis on three different test images.
- Using the Captum library, perform a post-hoc analysis using the Integrated Gradients provided by the library.
- Perform a post-hoc XAI analysis using LIME and SHAP attribution methods.
- Compare the attributions extracted from your XAI model (IG) and the models defined by the library choice (IG, LIME and SHAP), performing a statistical analysis to assess the differences/similarities between the two methods and also a statistical analysis regarding the attributions extracted.
- Comment the results you obtained.

2 DATASET

We select a dataset that contains 4920 RGB labelled images of people with glasses (2769) or without glasses (2151) with shape 1024x1024. The dataset is divided into 3936 train images and 984 test images. We resize all the images to 128x128 and apply data augmentation (RandomRotation, RandomHorizontalFlip, RandomVerticalFlip, ColorJitter) on the train dataset to increase the image's diversity.

Figure 1: Sampled images from dataset after the data augmentation

3 CNN IMPLEMENTATION

3.1 CNN model

We implement a CNN with 3 convolutional layers and 3 fully connected layers. Each convolutional layer is followed by a max pooling operation, and Rectified Linear Unit (ReLU) activation function is applied after each layer.

3.2 5-Fold Cross Validation

K-fold cross-validation is a technique used in machine learning to assess the model's generalization ability. It helps to ensure that the model's performance estimated is less dependent on the particular way the data is split into training and testing sets. The basic idea is to split the dataset into K subsets (or "folds") of approximately equal size. The model is then trained K times, each time using K-1 of the folds as training data and the remaining fold as validation data. This process is repeated K times, with each fold used exactly once as the validation data. The value of K can vary depending on the size of the dataset and computational resources available: common choices for K include 5-fold, 10-fold, and leave-one-out (where K equals the number of samples). We decided to use 5 folds for our model, each one containing about 787 images. After training and validating the model 5 times, we compute accuracy, f1, loss for each fold and, in the end, the mean accuracy, the mean f1 and the standard deviation of the accuracy across all the folds.

3.3 CNN's Train and Validation

We trained our net for 50 epochs across the 5 folds. The validation step and the metrics (train and validation accuracy, loss and f1) are computed after each epoch. After each fold the model is saved. At the end of the training step we save the best model and all the train metrics in a dataframe. We also calculate the mean and the standard deviation of the accuracy across all the folds.

Fold	F1 (Val)	F1 (Train)	Accuracy (Val)	Accuracy (Train)	Loss (Val)
fold 1	0.92625	0.932328	0.909188	0.915203	0.387399
fold 2	0.938097	0.930022	0.921449	0.912906	0.392215
fold 3	0.921491	0.92541	0.901728	0.907666	0.404772
fold 4	0.910947	0.921576	0.892681	0.903138	0.411598
fold 5	0.934723	0.937001	0.918551	0.924725	0.37091

Table 1: CNN mean metrics (F1 score, accuracy and Loss) for each fold in train and test

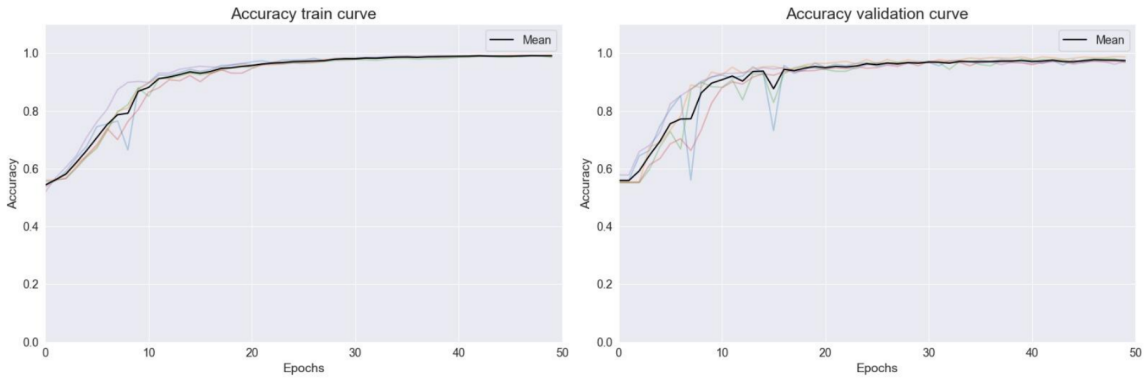


Figure 2: CNN accuracy curve. In the images the coloured lines represent the accuracy for each fold, the black one represents the total average accuracy across all folds

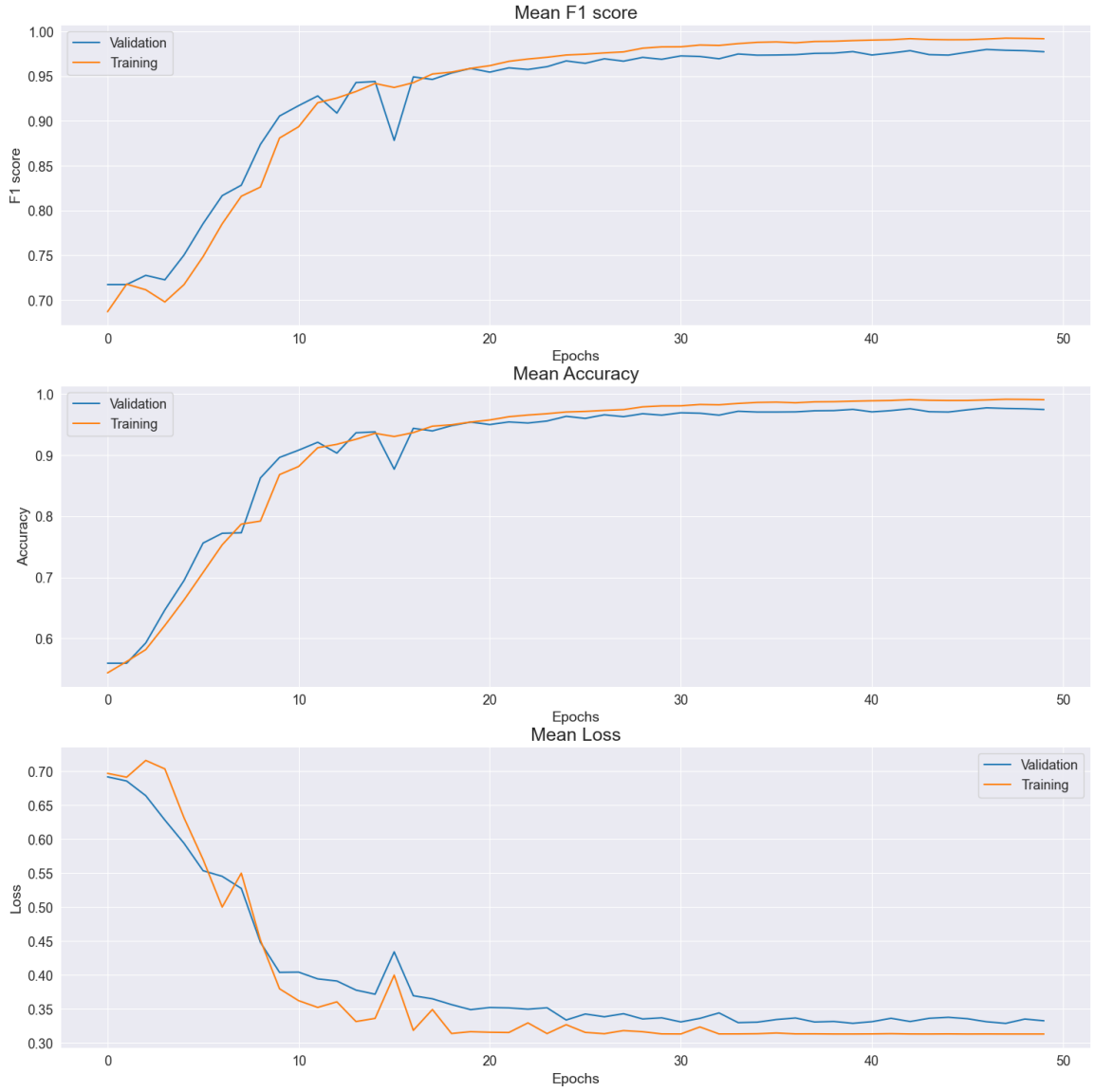


Figure 3: CNN comparison between mean train and mean validation metrics

4 SCATNET IMPLEMENTATION

4.1 ScatNet model

We implement a Scatnet using the same classifier as the CNN: 3 fully connected layers in which ReLU activation functions are applied after each layer. The main difference between the CNN and the Scatnet implementation is the presence of the Scattering as a feature extractor. Scattering in image processing refers to a technique used for extracting meaningful features from images. It's a method that aims to create a more robust representation of images by capturing both low and high-frequency information. Scattering works:

- First with a wavelet transform of the input image, convolving the image with a wavelet (passa-banda) filter at multiple (L) scales and orientations.
- After is calculated the modulus of the resulting coefficient for discarding their phase information. This step helps in making the representation invariant to phase changes.
- Then the modulus coefficients are spatially pooled, and that involves dividing the images into smaller spatial regions and aggregating the modulus coefficients within each region. Pooling helps in reducing the dimensionality of the representation and making it more robust to small spatial translation or deformations in the input image. The above steps are repeated J times for each scale and in each iteration the input to the wavelet transform is the output of the previous iteration

$$U[\lambda]x = |x \star \psi_\lambda| \qquad p = (\lambda_1, \lambda_2, ..., \lambda_n)$$

$$U[p]x = U[\lambda_m]...U[\lambda_2]U[\lambda_1]x = ||x \star \psi_{\lambda_1}| \quad \star \psi_{\lambda_2}| \quad ... \star \psi_{\lambda_m}| \tag{1}$$

$$U[0]x = x$$

$$S[p]x(u) = U[p]x \star \phi_{2J}(u) = \int U[p]x(v)\phi_{2J}(u-v)dv = ||x \star \psi_{\lambda_1}| \quad \star \psi_{\lambda_2}| \quad ... \star \psi_{\lambda_m}| \quad \star \phi_{2J}(u) \tag{2}$$

$$S[0]x = x \star \phi_{2J}(u)$$

4.2 CNN vs ScatNet

The main difference between CNN and Scatnet for the feature extraction is that in the Scatnet the filters are known and are usually wavelets. Therefore the filters don't have to be calculated in the network training phase and this saves us a lot of computing power.

4.3 Scatnet's Train and Validation

Also in this case we perform 5-fold cross validation and we trained our net for 100 epochs across the folds. All the steps and the extracted parameters are the same as in the CNN.

Fold	F1 (Val)	F1 (Train)	Accuracy (Val)	Accuracy (Train)	Loss (Val)
fold 1	0.92625	0.932328	0.909188	0.915203	0.387399
fold 2	0.938097	0.930022	0.921449	0.912906	0.392215
fold 3	0.921491	0.92541	0.901728	0.907666	0.404772
fold 4	0.910947	0.921576	0.892681	0.903138	0.411598
fold 5	0.934723	0.937001	0.918551	0.924725	0.37091

Table 2: Scatnet mean metrics (F1 score, accuracy and Loss) for each fold in train e test

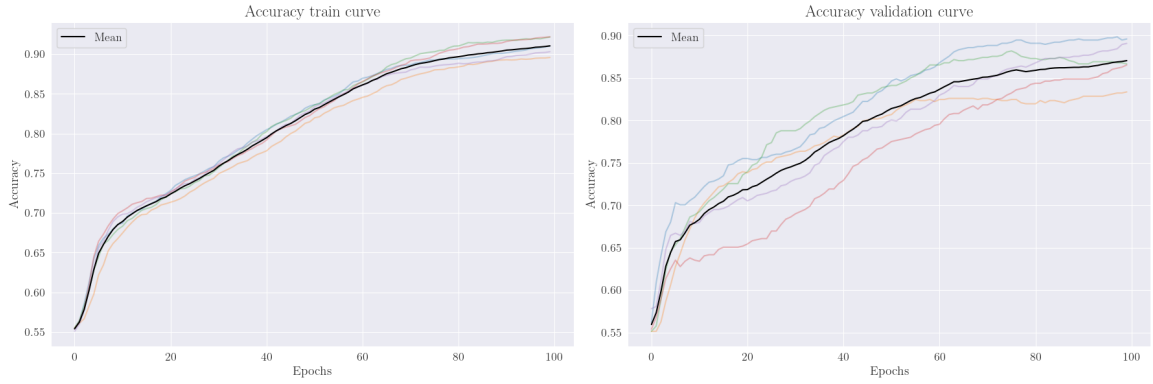


Figure 4: Scatnet accuracy curve. The coloured lines represent the accuracy for each fold, the black one represents the total average accuracy across all folds.

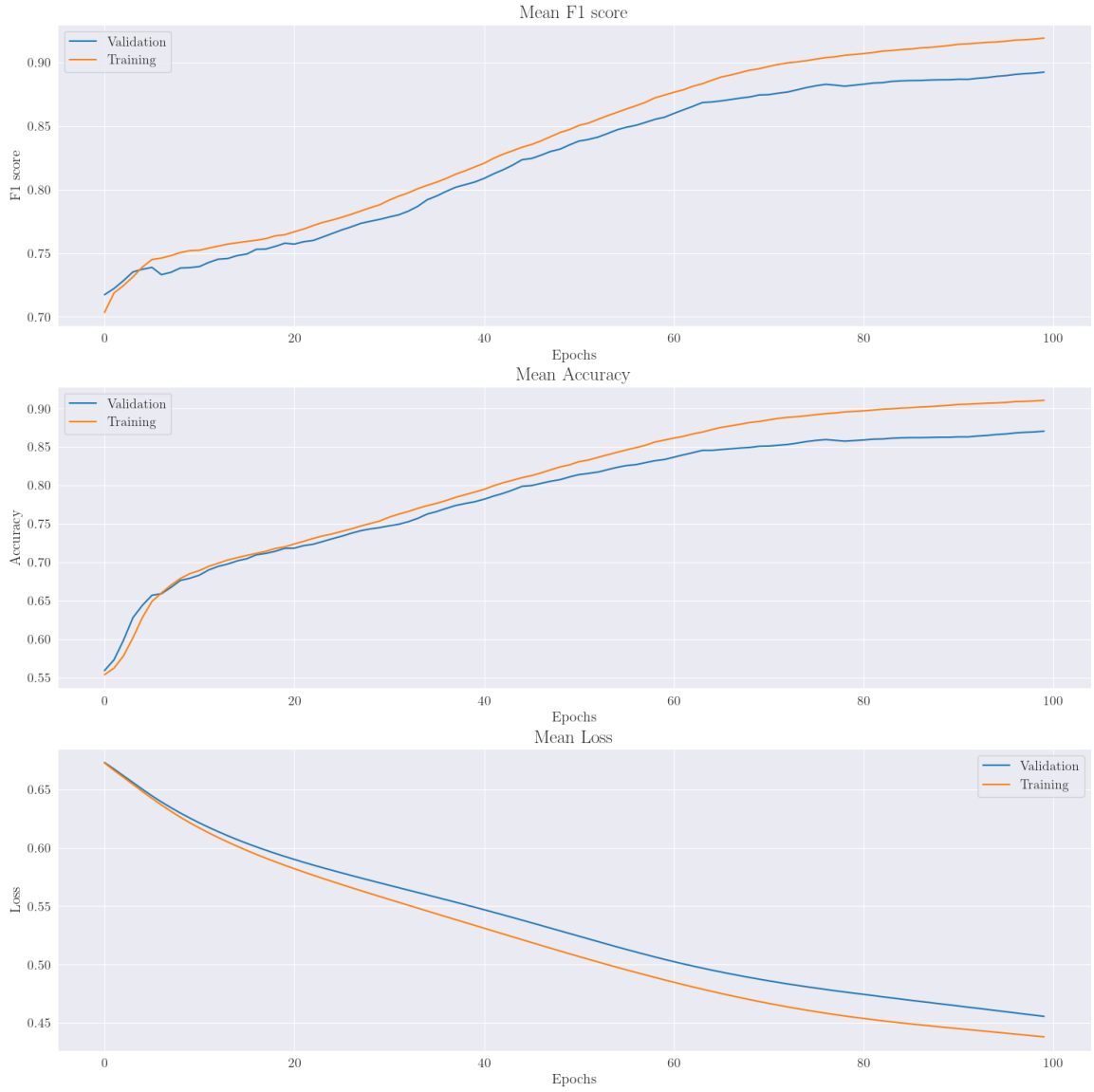


Figure 5: Comparison between mean train and validation metrics

5 RESULT

5.1 CNN's test

We load the best CNN model that has a 98.73% of accuracy and make inference on it with the test set. We computed f1 score, accuracy and the confusion matrix and the returned values are:

$$F1score = 0.99 \qquad Accuracy = 0.99$$

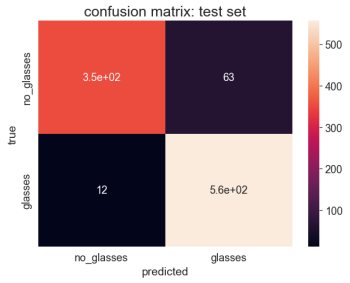


Figure 6: CNN confusion matrix of test

5.2 ScatNet's test

We load the best Scatnet model that has a 89.59% of accuracy and make inference on it with the test set. We computed f1 score, accuracy and the confusion matrix and the returned values are:

$$F1score = 0.94 \qquad Accuracy = 0.92$$

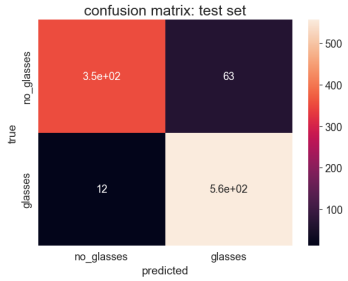
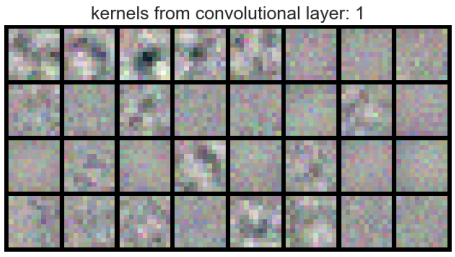


Figure 7: Scatnet confusion matrix of test

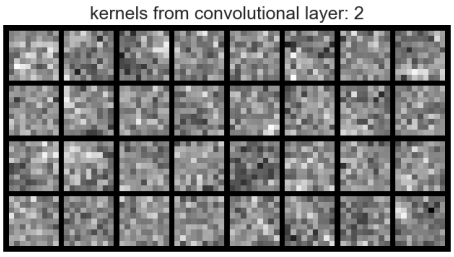
6 FILTER

6.1 CNN's filters

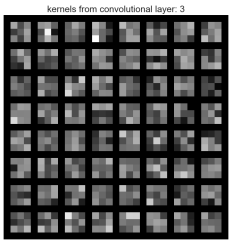
In the next images we have the kernels of the convolutional layers. The kernels of the first one usually have an edge detector behaviour, in our case this doesn't happen and since the model has still a good accuracy we might deduct that it learns more deep and complex features



(a) First layer



(b) Second layer



(c) Third layer

Figure 8: CNN filter extracted

Figure 9: Two CNN feature maps: one of a person without glasses and another of a person with glasses.

6.2 ScatNet's filters

As we already said, in this case the filters are known and not learned during the training phase. We can visualize them using the Fourier transform:

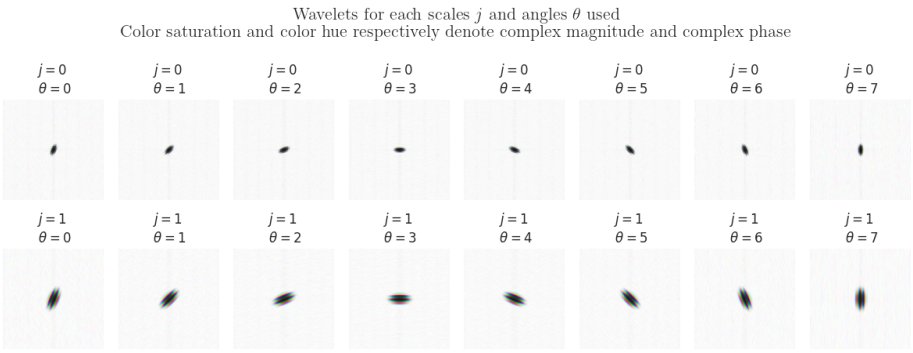


Figure 10: Scatnet filter

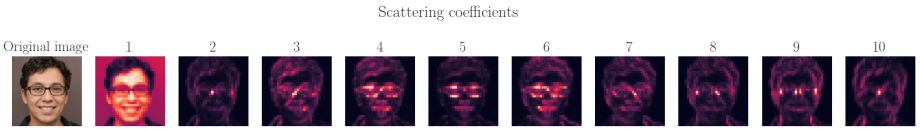


Figure 11: Scattering output

7 WHEN SCATNET WORKS BETTER THAN CNN

We take the initial dataset and we select only 46 images of people with glasses and 32 images of people with no glasses. We found that CNN's best accuracy on fold on train is 66.7% and on test is 59.3%. On the other hand, Scatnet's best accuracy on fold on train is 73.33% and on test is 62.0%.

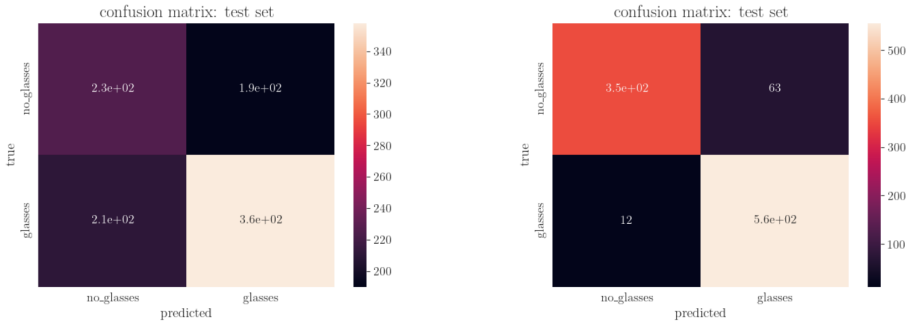


Figure 12: CNN (sx) and Scatnet (dx) confusion matrix of tests with less images

8 NTEGRATED GRADIENTS

8.1 Integrated Gradients (IG)

Integrated Gradients (IG) is a method employed in eXplainable AI to explain the influence of input features on the predictions of machine learning models. It addresses the challenge of model interpretability by attributing a prediction to its relevant features. The technique calculates the importance of each feature by integrating the gradients of the model's prediction function with respect to the input features along a straight path from a baseline input to the actual input. The integral of integrated gradients can be efficiently approximated via a summation. Formally , for a specific feature:

$$IntegratedGrads_i^{approx}(x) ::= (x_i - x_i') \sum_{k=1}^{m} \frac{\partial F(x' + \frac{k}{m} \cdot (x - x'))}{\partial x_i} \cdot \frac{1}{m} \tag{3}$$

where:

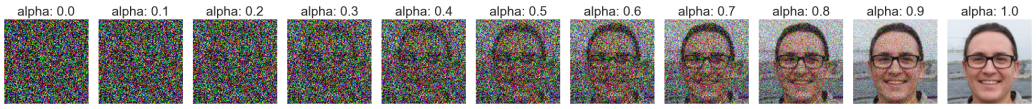
- x_i' = Baseline value of feature i
- x_i = Actual value of feature i in input x
- F = Model's prediction function

- m = Number of steps in the Riemman approximation of the integral
- $\sum_{k=1}^{m}$ = Summation computes the average of the gradients with respect to feature i along this path

8.2 Baseline

For our tests we implement a zero (black) baseline and a random normalized baseline. In the integrated gradients we use both the baseline but for Lime and Shap we use only the black one.

(a) Baseline Zero



(b) aseline Rand

8.3 Integrated gradients by scratch

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

1. Generate alphas α
2. Generate a linear interpolation between the baseline and the original image:

$$IntegratedGrads_i^{approx}(x) ::= (x_i - x_i') \times \sum_{k=1}^{m} \frac{\partial F(\overbrace{x' + \frac{k}{m} \cdot (x - x')}^{\text{interplate m images at k intervals}})}{\partial x_i} \times \frac{1}{m} \sum$$

3. Compute gradients between model output predictions with respect to input features to measure the relationship between changes to a feature and changes in the model's predictions:

$$IntegratedGrads_i^{approx}(x) ::= (x_i - x_i') \times \sum_{k=1}^{m} \frac{\overbrace{\partial F(\text{interpolated images})}^{\text{Compute Gradients}}}{\partial x_i} \times \frac{1}{m} \sum$$

4. Integral approximation through averaging gradients

$$IntegratedGrads_i^{approx}(x) = (x_i - x_i') \times \sum_{k=1}^{m} \textbf{gradients(interpolated images)} \times \frac{1}{m} \sum$$

5. Scale integrated gradients with respect to original image

$$IntegratedGrads_i^{approx}(x) ::= (x_i - x_i') \times \text{integrated gradients}$$

The reason this step is necessary is to make sure that the attribution values accumulated across multiple interpolated images are in the same units and faithfully represent the pixel importances on the original image.

Figure 13: Example of the results of integrated gradients by scratch, in the first case with the zero baseline and in the second case with the random normalized baseline

8.4 Integrated gradients by Captum

Captum provides a generic implementation of integrated gradients that can be used with any PyTorch model.

Figure 14: Example of the results of Captum's integrated gradients, in the first case with the zero baseline and in the second case with the random normalized baseline

9 LIME

LIME (Local Interpretable Model-Agnostic Explanations) [4] is a technique designed to provide local interpretability for machine learning models. In image classification tasks, deep learning models often act as "black-boxes" making it challenging to understand why they classify images the way they do. LIME offers a solution to this problem by providing local and interpretable explanations for individual image predictions. Formally when applied to images, the steps involved in LIME are as follows:

- **Select an Image:** Choose a specific image for which you want to understand the model's prediction.
- **Generate Perturbed Images:** Create slightly modified versions of the original image by applying small changes, such as adding noise or masking parts of the image.
- **Obtain Predictions:** Use the machine learning model to predict the class labels for the perturbed images. This step generates a dataset with predictions for the modified images.
- **Train an Interpretable Model:** Train a simple and interpretable model, such as a linear classifier or decision tree, using the perturbed images and their corresponding predictions from the machine learning model.
- **Interpretation:** Analyze the coefficients, decision boundaries, or feature importance of the interpretable model to understand which parts of the image influenced the model's prediction the most. This helps provide insights into why the original model made the prediction it did for the selected image.

Figure 15: CNN (sx) and Scatnet (dx) Results of Lime in three different images

10 SHAP

SHAP (SHapley Additive exPlanations) [5] is a powerful technique used for explaining the predictions of machine learning models. It offers a principled approach to understanding the importance of different features. When applied to images, SHAP attributes the importance of each pixel to the model's prediction. The main steps are:

- **Feature Attribution:** Pixels are treated as features, and SHAP calculates the contribution of each pixel to the model's prediction.
- **SHAP Values:** These values quantify the impact of each pixel on the model's output, providing local interpretability.

- **Visualization:** SHAP values can be visualized, helping to identify influential regions in the image through techniques like summary plots or heatmaps.

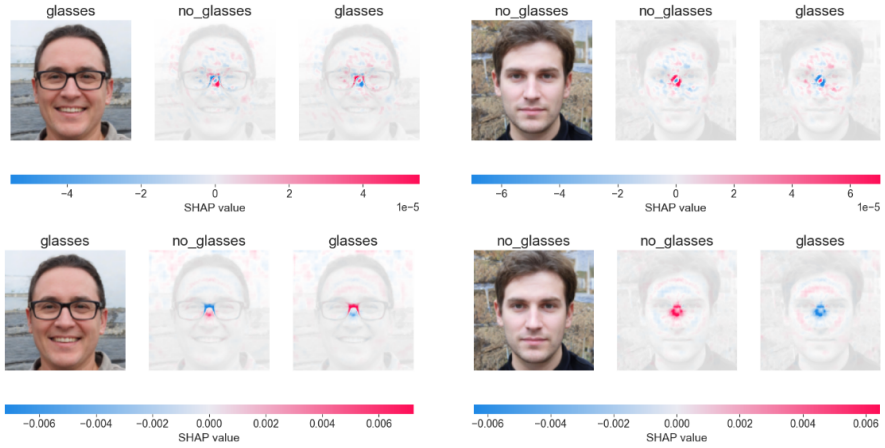


Figure 16: CNN (first line) and Scatnet (second line) examples of SHAP on two different images, in the first case with glasses and in the second one without glasses

11 COMPARISON AMONG XAI METHODS

11.1 Test Images

For all our comparisons and tests we used these 3 images :

(a) Glasses **(b)** No Glasses **(c)** Sun Glasses

Figure 17: tests original images

11.2 SHAP vs IG

Visual comparison of SHAP and Integrated Gradients methods.

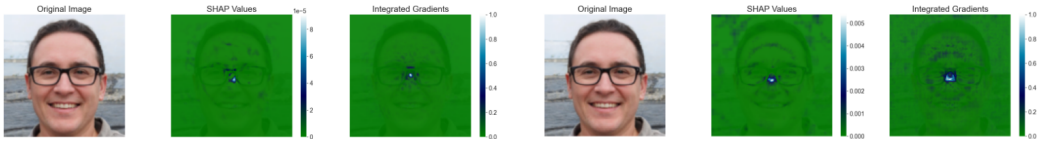


Figure 18: CNN (sx) and Scatnet (dx) comparison between SHAP and integrated gradients methods

11.3 Attributions Histograms

For visualization purposes we have normalized IG, IG captum and SHAP values. We considered values above 0.2 for our histograms. We can notice that IG and IG captum have similar distributions of attributions and we can find some similarities also with SHAP.

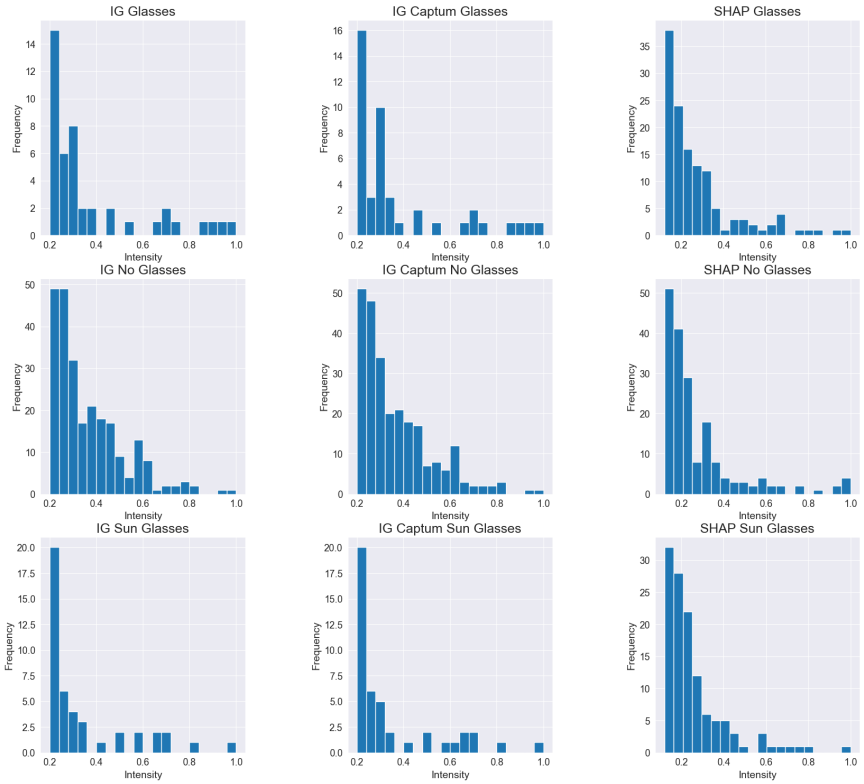


Figure 19: CNN Histograms

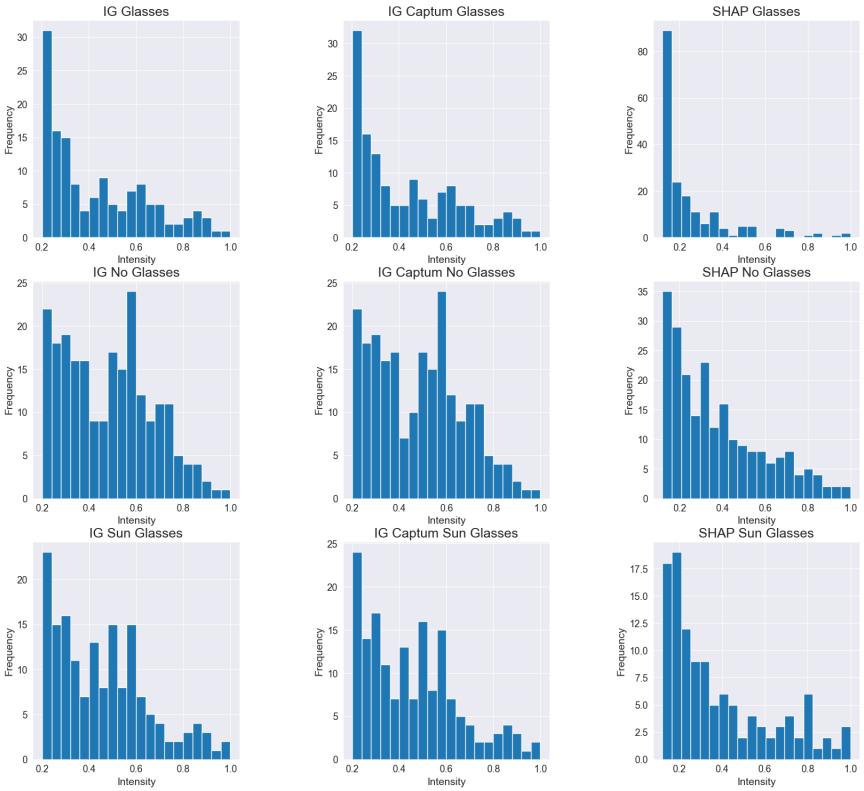
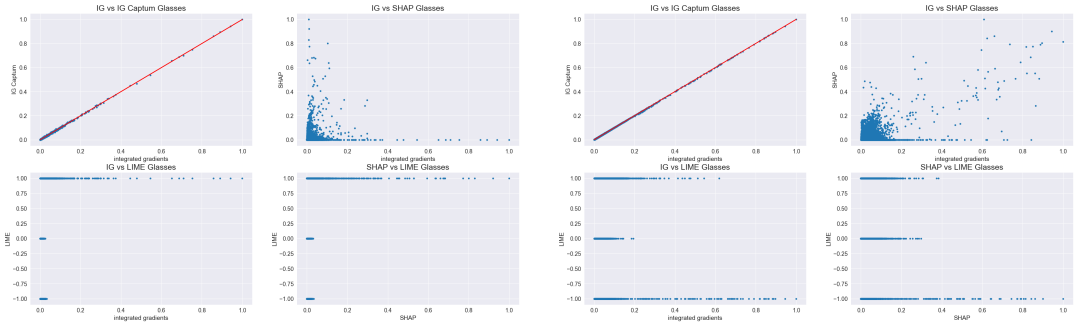


Figure 20: Scatnet Histograms

11.4 Correlation between attributions

In figs 22,23,24 can see a linear correlation between IG and IG Captum in both the models. But on the other hand we can see that there is a correlation between IG and SHAP only with the Scatnet model. Regarding SHAP, we can observe that in the CNN, the majority of points concentrate around 1, while in the SCATNET, we have more -1. This could be attributed to the fact that LIME focuses on the overall image, whereas the other two methods assign values only to a small portion of the image, in our case, the central area of the face.
In tables 3,4,5 we calculate the distances between the baricenters (mean of the distributions) and the distances between the spreads (standard deviations). The most significant results are those concerning the comparison between the two methods of Integrated Gradients (IG) and between IG and SHAP, as we are able to compare values that have the same distribution range. As expected, we found a very low distance in both models, given the linear relationship between the IG methods. However, we found a higher value, which we still consider significant, between IG and SHAP. The considerations made so far are also reflected in the correlation coefficient values obtained (Table 6).



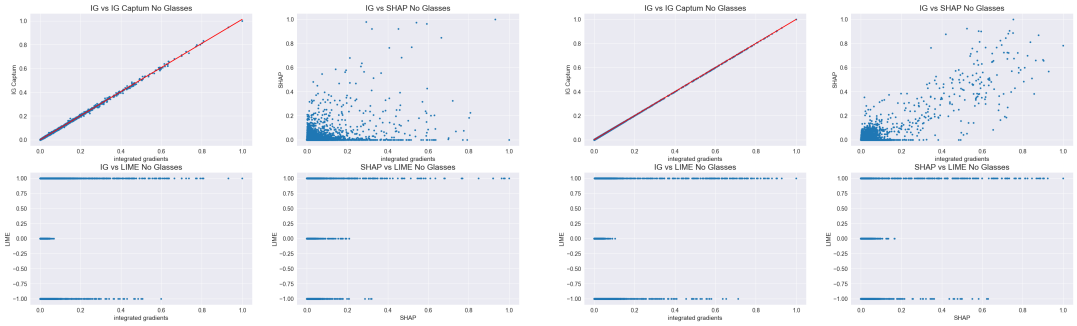
(a) CNN (Glasses) **(b)** ScatNet(Glasses)

Figure 21: The correlation between CNN and ScatNet attribution methods (IG, IG captum, SHAP, and Lime)

Comparison	Distance between baricenters	Distance between the spreads
ig to ig_captum	0.000486544	2.88071e-05
ig to SHAP	0.0645191	0.0263511
ig to LIME	0.239745	0.66364
SHAP to LIME	0.304264	0.689992

Comparison	Distance between baricenters	Distance between the spreads
ig to ig_captum	0.000800579	9.31656e-06
ig to SHAP	0.181166	0.0486867
ig to LIME	0.179407	0.857501
SHAP to LIME	0.00175928	0.906188

Figure 22: CNN (sx) and Scatnet(dx) table distances of the image with the person with glasses



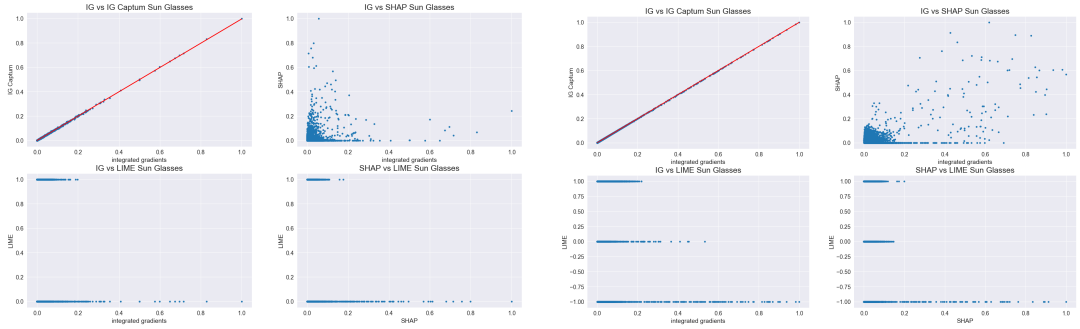
(a) CNN(No Glasses) **(b)** ScatNet(No Glasses)

Figure 23: The correlation between CNN and ScatNet attribution methods (IG, IG captum, SHAP, and Lime)

Comparison	Distance between baricenters	Distance between the spreads
ig to ig_captum	0.000811202	0.000746074
ig to SHAP	0.0892134	0.0526515
ig to LIME	0.336064	0.729236
SHAP to LIME	0.24685	0.781888

Comparison	Distance between baricenters	Distance between the spreads
ig to ig_captum	0.000120524	8.60591e-06
ig to SHAP	0.0958175	0.0617619
ig to LIME	0.87992	0.631787
SHAP to LIME	0.784103	0.693549

Figure 24: CNN (sx) and Scatnet(dx) table distances of the image with the person without glasses



(a) CNN(Sun Glasses) **(b)** ScatNet(Sun Glasses)

Figure 25: The correlation between CNN and ScatNet attribution methods (IG, IG captum, SHAP, and Lime)

Comparison	Distance between baricenters	Distance between the spreads
ig to ig_captum	0.000834473	0.000746074
ig to SHAP	0.090034	0.0526515
ig to LIME	0.144581	0.729236
SHAP to LIME	0.234615	0.781888

Comparison	Distance between baricenters	Distance between the spreads
ig to ig_captum	0.00173692	8.60591e-06
ig to SHAP	0.11413	0.0617619
ig to LIME	0.316708	0.631787
SHAP to LIME	0.202578	0.693549

Figure 26: CNN (sx) and Scatnet(dx) table distances of the image with the person with glasses

Comparison	IG vs IG Captum	IG vs SHAP	IG vs LIME	SHAP vs LIME
Glasses	0.999302	0.198223	0.109916	0.0577796
No Glasses	0.999554	0.49414	0.201026	0.116117
Sun Glasses	0.999613	0.452962	0.109372	0.00364763

Comparison	IG vs IG Captum	IG vs SHAP	IG vs LIME	SHAP vs LIME
Glasses	0.999961	0.574102	-0.0499731	-0.113782
No Glasses	0.999999	0.831061	0.134356	0.164829
Sun Glasses	0.999995	0.646029	-0.107265	-0.0988894

Figure 27: CNN (sx) and Scatnet(dx) correlation coefficients

References

[1] Glasses or No Glasses Dataset

[2] Integrated Gradients from scratch - Tensorflow

[3] LIME: Local Interpretable Model-agnostic Explanations

[4] Integrated Gradients - Captum Documentation

[5] A Unified Approach to Interpreting Model Predictions, Scott M. Lundberg, Su-In Lee