4IM06
TÉLÉCOM PARIS

Post-hoc Explainability methods for Deep Learning

Project 18

Students:
Lorenza MARTINS
Luiz Fernando MOREIRA
Marina SANGINETO
Franz YURI

Professor:
Pietro GORI

Contents

1 Introduction

Nowadays deep convolutional neural networks are used in several computer vision tasks, but although they can achieve a high accuracy in most cases, it is still very hard to understand how and why they work so well. Without this understanding, designing a novel neural network archictecture remains in a big part trial-and-error.

In this work we present an implementation of two models [1, 2], both aimed at gaining deeper insight into how the network reaches its predictions by visualizing what happens inside under different scenarios.

In [1], a visualization technique is proposed based on a deconvolutional network (deconvnet) that projects activations from intermediate feature maps back into image space. With these visualizations, we can diagnose and improve neural network architecture and understand exactly which regions of an image the network focuses on when making its classification.

Moreover, in [2], a visualization technique is proposed based on gradient-weighted class activation mapping (Grad-CAM), which uses the gradients flowing into the final convolutional layer to generate a heatmap highlighting the regions most relevant to a given class. With these visualizations, we can verify that the network is attending to semantically meaningful parts of the image, detect when it relies on spurious or irrelevant cues, and gain deeper insight into its decision-making process.

2 Datasets

2.1 Exploratory Data Analysis

Before training the convolutional models and implementing the visualization techniques, we performed an exploratory analysis on the datasets to assess label coverage, class distribution, and input normalization requirements. We worked with the PathMNIST, BloodMNIST and DermaMNIST. This step helped validate preprocessing choices and guide the model setup.

First, we visualized raw input batches from the training loader to confirm the nature of the histology images and verify correct label mappings. Figure 1 shows one such batch and the corresponding labels were printed in the notebook for the PathMNIST.

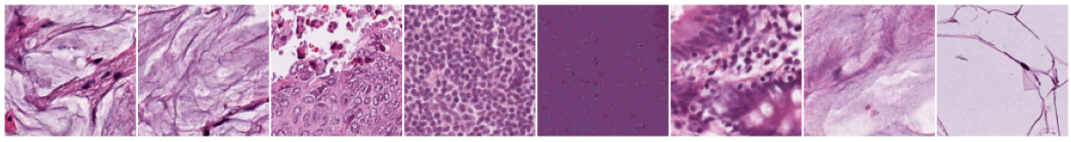


Figure 1: Sample batch from the training loader.

Next, we examined the class distribution. Understanding the class distribution is important because it affects the model's ability to learn minority classes and generalize to unseen data. To do so, we wrote two functions: `get_class_distribution` and `plot_class_distribution`.

The first function iterates over the dataloader, extracts the ground truth labels from each batch, handles cases where labels are one-hot encoded or multi-dimensional, and counts the occurrences of each class. This returns a dictionary mapping each class index

to its frequency in the dataset. The second one receives this dictionary and a mapping of class indices to names, sorts the class-frequency pairs, and generates the bar plot showing the number of samples per class. The function also adds the number of samples on top of each bar and formats the plot to make it easier to read. The resulting plot is shown in each dataset section. We could see that some were more imbalanced than other.

We also extracted one representative sample per class using a custom function named `visualize_one_sample_per_class`. This function iterates through the training dataloader and collects the first encountered image for each unique class label, storing them in a dictionary keyed by class index. After sorting the collected samples, it displays them in a 3×3 grid, with class names as titles. This visualization is useful for gaining an initial visual intuition about the dataset, ensuring that images are well distributed and distinct.

Below is a summary of the functions developed for this data analysis:

- `visualize_batch`: Displays `n` images from a batch, with raw labels.
- `get_class_distribution`: Computes frequency of each label in the dataset.
- `plot_class_distribution`: Plots a histogram from class counts.
- `compute_mean_std`: Calculates average pixel intensity and std deviation.
- `visualize_one_sample_per_class`: Shows one sample per class in a grid.

2.1.1 PathMNIST

The PathMNIST dataset consists of histopathology images of colon tissue. It contains 9 distinct classes representing different tissue types such as adipose, lymphocytes, mucus, and various epithelial structures including cancer-associated and cancerous epithelium. This dataset is often used for multi-class tissue classification tasks in computational pathology.

Figure 2 shows the class distribution, revealing a moderate imbalance across categories. For example, the "colorectal adenocarcinoma epithelium" class has over 12,000 samples, while "normal colon mucosa" and "mucus" are underrepresented with fewer than 8,000 examples each. We know that such imbalance may introduce bias during training and reduce generalization on rare classes, however, we were still able to achieve satisfactory results during training, as shown in Figure 8.

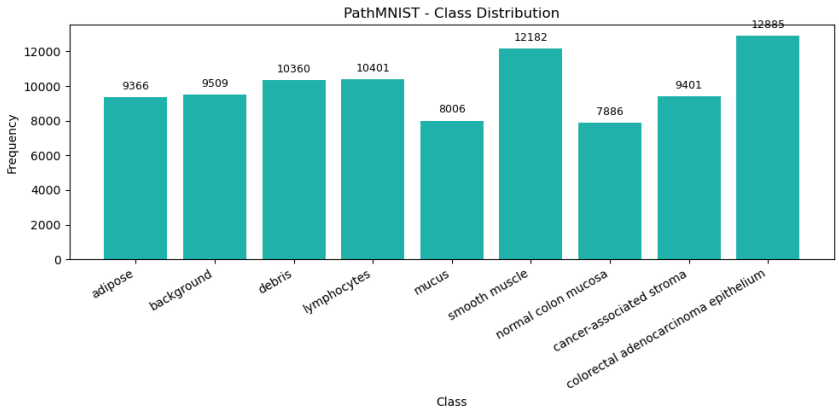


Figure 2: Class distribution in PathMNIST.

Figure 3 displays one representative sample per class. We can clearly distinguish textural and structural patterns among the classes. This helps verify that class labels are coherent with morphological features.

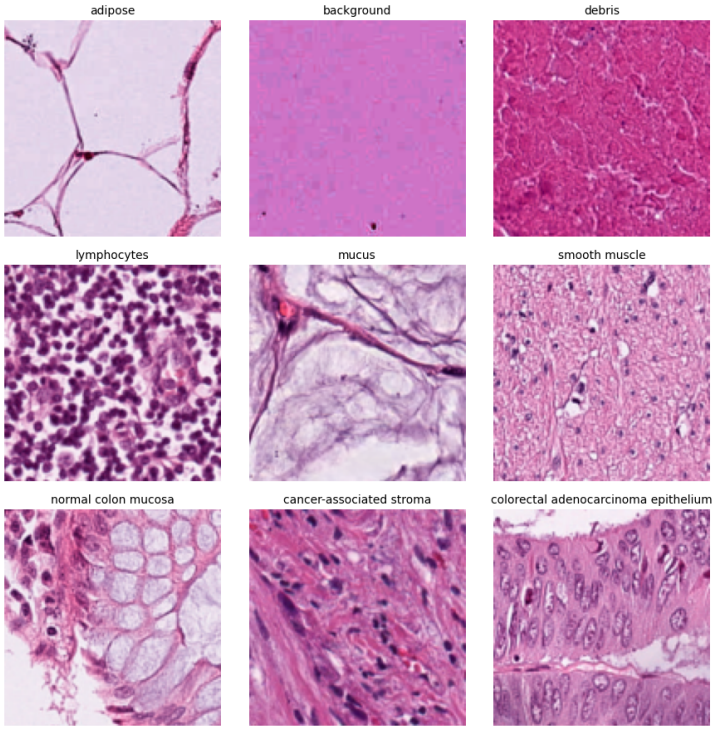


Figure 3: Grid with one image per class from PathMNIST.

2.1.2 BloodMNIST

BloodMNIST consists of microscopic blood cell images grouped into 8 categories such as neutrophils, eosinophils, lymphocytes, and platelets. The dataset is designed for tasks in hematology image classification and is relevant for identifying blood cell abnormalities.

Figure 4 shows a relatively well-balanced dataset, with most classes having between 800 and 2,300 samples. The most represented class is "neutrophil" while "lymphocyte" and "basophil" have the fewest samples. Although not perfectly balanced, this distribution allowed usto achieve satisfactory results during training, as shown in Figure 9.

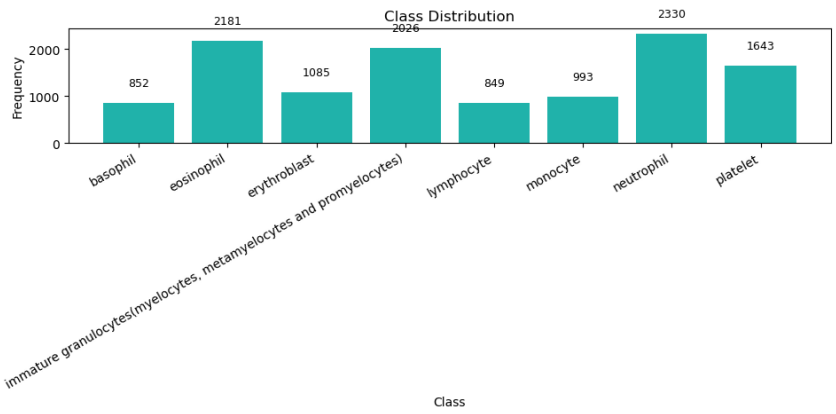


Figure 4: Class distribution in BloodMNIST.

As shown in Figure 5, the visual differences between cell types are clear, with notable variations in shape, nucleus segmentation, and texture. This confirms that a convolutional model should be able to learn meaningful features distinguishing these classes.

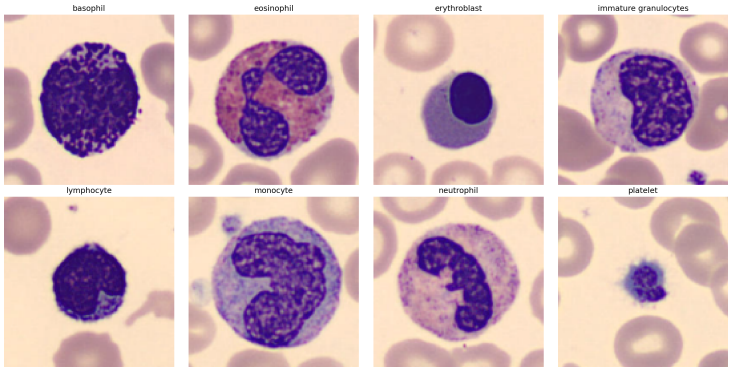


Figure 5: Grid with one image per class from BloodMNIST.

2.1.3 DermaMNIST

DermaMNIST contains over dermoscopic images of skin lesions labeled into 7 categories including melanoma and various types of carcinomas. The dataset serves as a benchmark for skin disease classification using medical imaging.

The class distribution in Figure 6 is highly imbalanced. The "melanocytic nevi" class dominates with 4,693 samples, while classes like "dermatofibroma" and "vascular lesions" have fewer than 100 examples. This heavy imbalance may bias the model toward the majority class. The significant class imbalance in DermaMNIST is further reflected in the considerably lower training and test accuracy, as shown in Figure 10.

Figure 6: Class distribution in DermaMNIST.

In Figure 7, we have the grid with one sample per class and we can observe a substantial visual variation across the classes. Some lesions appear highly textured, pigmented, or vascularized, providing strong visual cues. However, others show subtle differences, indicating the potential difficulty in distinguishing certain classes and the importance of model interpretability.

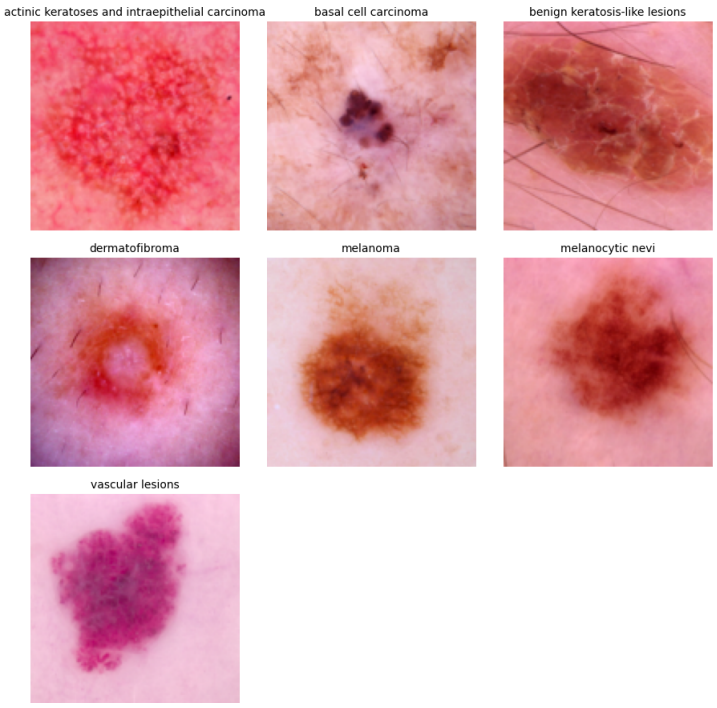


Figure 7: Grid with one image per class from DermaMNIST.

3 Training Model

We adapted the original AlexNet [3] to accept MedMNIST images [4] at 128×128 resolution.

Our adapted AlexNet begins by ingesting 128×128 RGB images and immediately reducing spatial resolution via a 5×5 convolution (stride 2, padding 2), yielding a $64 \times 64 \times 64$ feature map. A 3×3 max-pool (stride 2) then downsamples this to $64 \times 31 \times 31$.

The second block applies a 5×5 convolution (stride 1, padding 2) to preserve the 31×31 spatial size across 192 channels, followed by another 3×3 max-pool (stride 2) to produce a $192 \times 15 \times 15$ tensor.

Next, three successive convolutional layers (each with 3×3 kernels, stride 1, padding 1) expand and contract the channel depth—384 channels in layer 3, then 256 channels in layers 4 and 5—while holding the 15×15 spatial dimensions constant. A final 3×3 max-pool (stride 2) reduces this to $256 \times 7 \times 7$, after which an adaptive average-pool forces every feature map to a uniform 6×6 grid.

The resulting $256 \times 6 \times 6$ tensor is flattened and passed through two fully connected layers of size 4096 (each preceded by 50% dropout and a ReLU activation), before a final linear layer outputs class logits. This arrangement of strided convolutions, pooling, and adaptive averaging ensures a gradual reduction of spatial information while progressively abstracting features for classification.

Three subsets were considered—PathMNIST, BloodMNIST and DermaMNIST—and each was trained under the same overall scheme: Adam optimizer with learning rate 10^{-3}, weight decay 10^{-5}, and cross–entropy loss. Only batch size and number of epochs varied by dataset:

- **PathMNIST**: 23 epochs, batch size 512
- **BloodMNIST**: 65 epochs, batch size 2048

- **DermaMNIST**: 64 epochs, batch size 1024

Tables 1–3 report the training, validation and test accuracy and loss for each dataset.

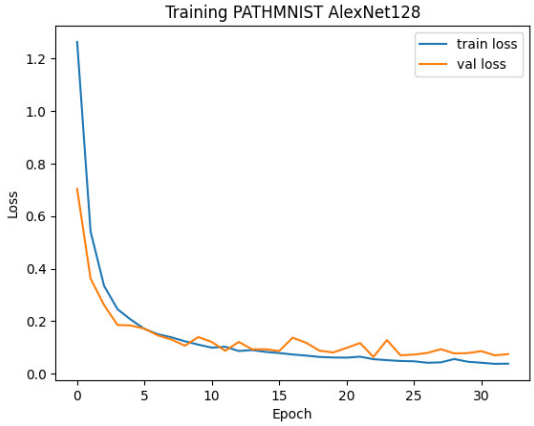


Table 1: AlexNet-128 on PathMNIST

Split	Accuracy (%)	Loss
Train	98.17	0.0549
Validation	98.06	0.0635
Test	91.20	0.3484

Figure 8: PathMNIST: training curve (left) and final accuracy & loss (right).

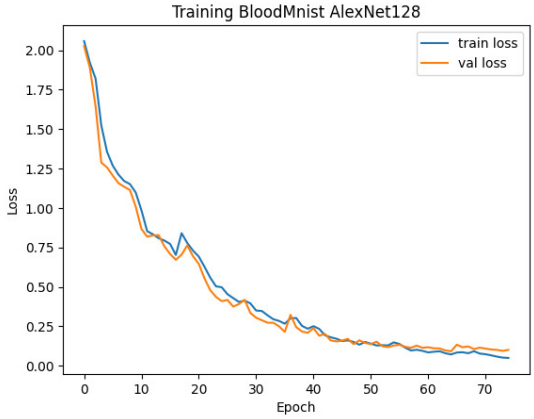


Table 2: AlexNet-128 on BloodMNIST

Split	Accuracy (%)	Loss
Train	97.55	0.0717
Validation	97.25	0.0915
Test	97.22	0.1308

Figure 9: BloodMNIST: training curve (left) and final accuracy & loss (right).

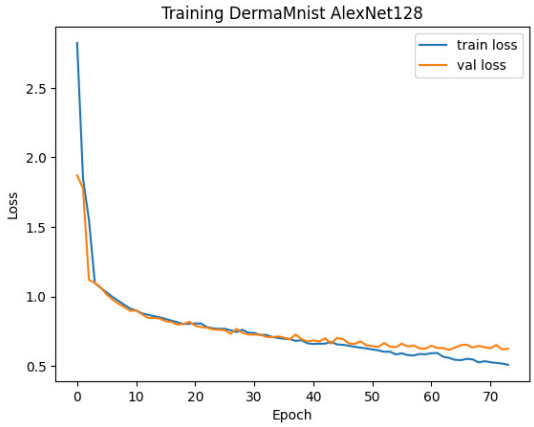


Table 3: AlexNet-128 on DermaMNIST

Split	Accuracy (%)	Loss
Train	78.85	0.5587
Validation	78.07	0.6164
Test	76.61	0.6296

Figure 10: DermaMNIST: training curve (left) and final accuracy & loss (right).

4 Visualizing and Understanding CNNs

Convolutional neural networks achieve remarkable accuracy across a wide range of vision tasks, yet their internal decision-making remains largely a black box. To better understand how these models arrive at their predictions, we apply two complementary visualization techniques. First, DeconvNet inverts intermediate feature-map activations back into pixel space, revealing the patterns and structures each filter has learned to detect. Second, the occlusion-sensitivity method systematically masks out regions of the input image to measure how each patch contributes to the network's final output. By combining these approaches, we gain both positive and negative evidence about the spatial cues and features that drive the network's behavior.

4.1 DeconvNet

In order to visualize the activations back into pixel space, we build a Deconvolutional Network (DeconvNet). This model simply performs the reverse operations needed to reconstruct the original image. The DeconvNet consists of five `ConvTranspose2d` layers, which undo the convolution steps, and three `MaxUnpool2d` layers, which invert the max-pooling operations. To correctly reverse each max-pooling, we must know exactly which indices were chosen during the forward pass of AlexNet, thus, after each layer our AlexNet model saves both the pooled feature map and its pooling indices in a dictionary called `acts`. Each `ConvTranspose2d` layer's weights are tied to its corresponding AlexNet convolution by copying the trained parameters directly into the deconv layers.

To display the patterns recovered by our DeconvNet, we have the following pipeline:

1. **Forward pass:** Run each test image through AlexNet, storing the pooled feature maps and pooling indices in the dictionary `acts`.

2. **Select image:** For each class, choose one representative test example and let its feature map at layer L be

$$F \;=\; \texttt{acts}[\texttt{f"feat\{L\}"}]\,[0] \quad \in\; R^{C\times H\times W}.$$

3. **Aggregate activations:** Zero out any negative responses so only excitatory filters remain:

$$F^{+} = \max(F, 0)\,.$$

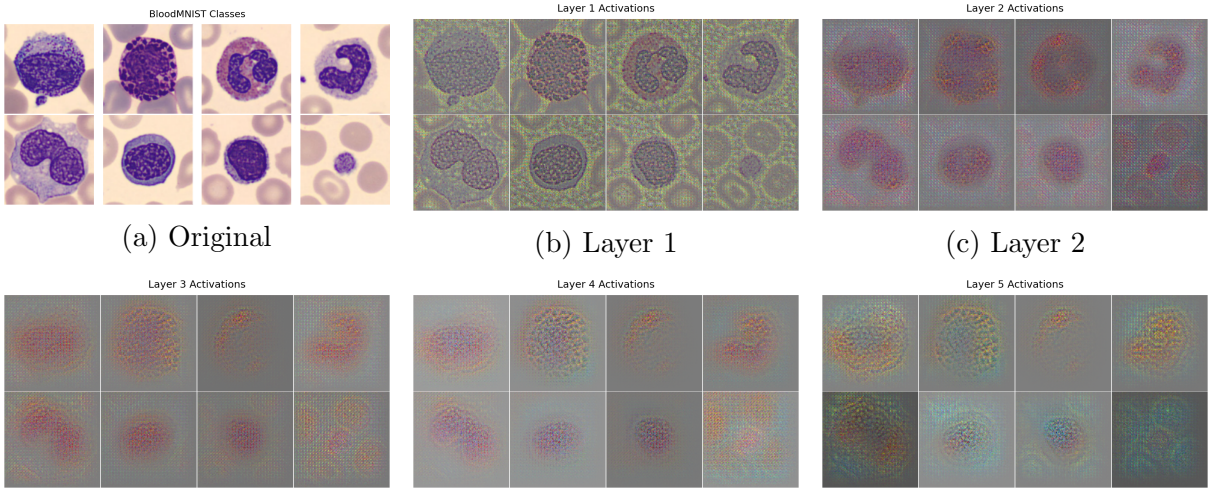
   This retains all channels at once instead of a single "one-hot" slice.

4. **Deconvolution:** Feed the full positive feature map F^{+} (plus the saved indices) into `DeconvNet`, which inverts each unpool–convolution stage back to a full-resolution RGB image.

5. **Restore pixel range:** Reverse the input normalization and linearly rescale the reconstructed pixels into the $[0, 1]$ interval for display.

6. **Visualization:** Arrange the original and reconstructed images side by side in a 2×4 (or 3×3) grid.

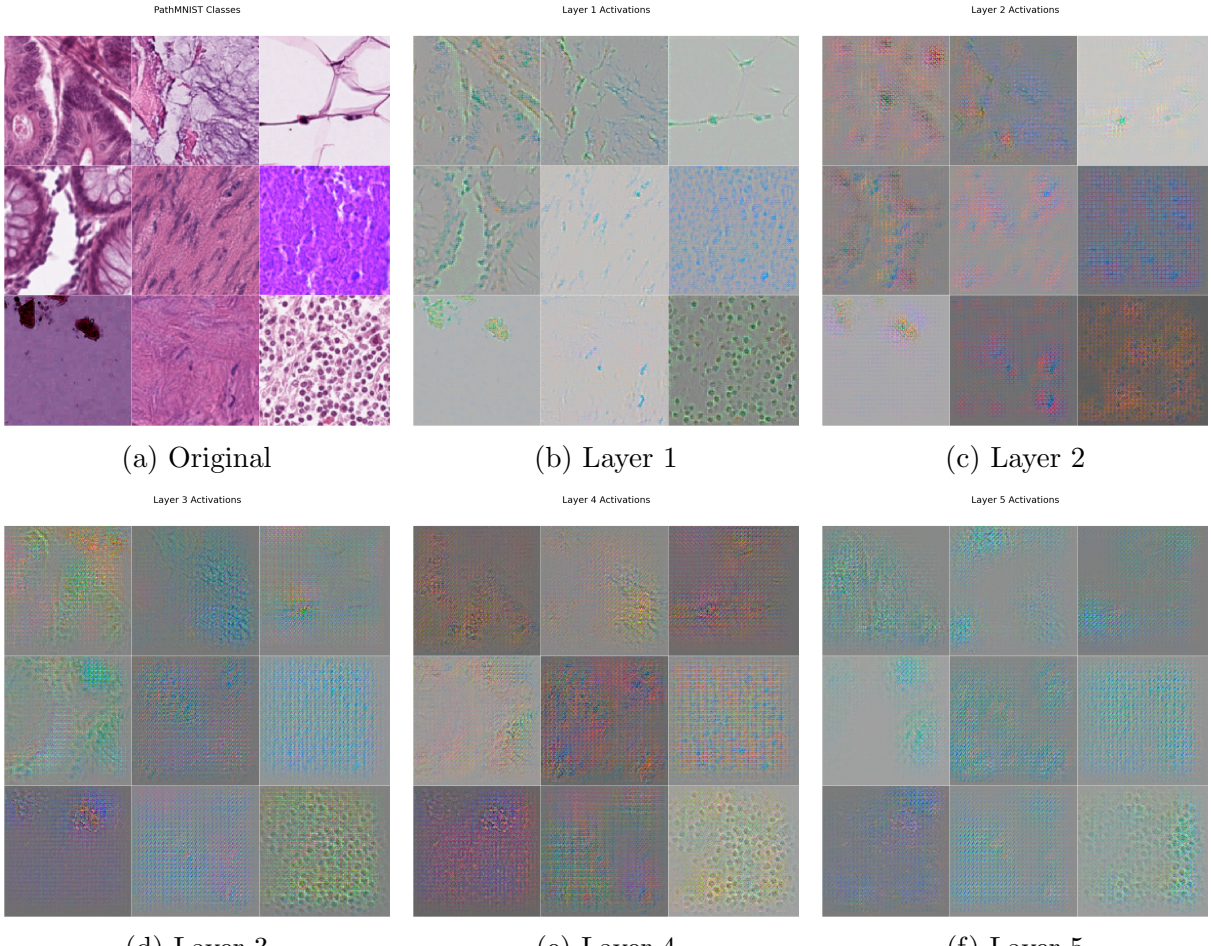
This process can be repeated for any convolutional layer, yielding an intuitive, pixel-level interpretation of internal feature detections.



(a) Original (b) Layer 1 (c) Layer 2

(d) Layer 3 (e) Layer 4 (f) Layer 5

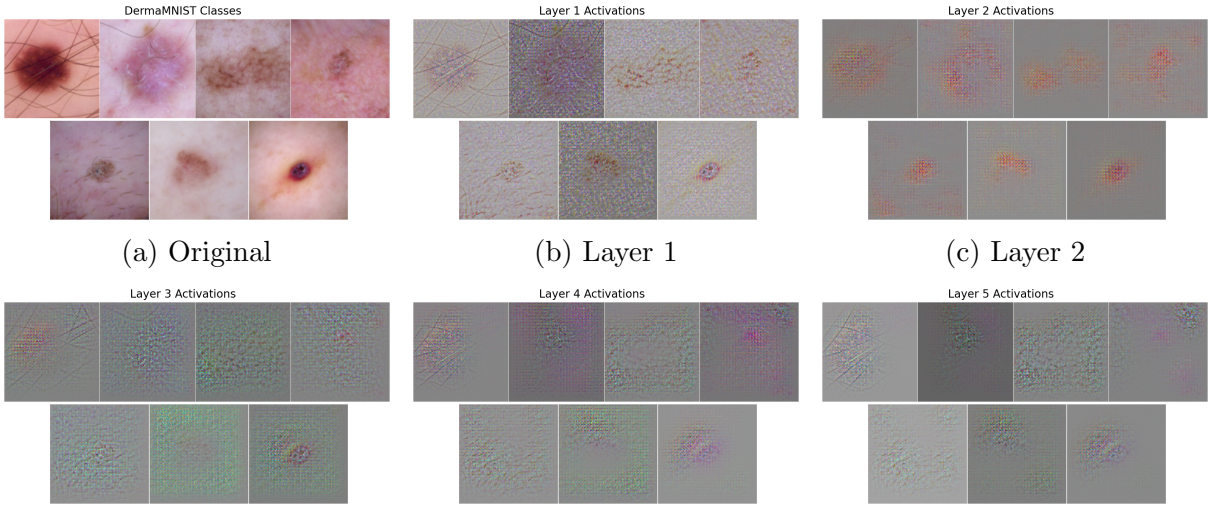
Figure 11: BloodMNIST classes input alongside DeconvNet reconstructions from layers 1–5.



(a) Original (b) Layer 1 (c) Layer 2

(d) Layer 3 (e) Layer 4 (f) Layer 5

Figure 12: PathMNIST classes input alongside DeconvNet reconstructions from layers 1–5.



(a) Original (b) Layer 1 (c) Layer 2

(d) Layer 3 (e) Layer 4 (f) Layer 5

Figure 13: DermaMNIST classes input alongside DeconvNet reconstructions from layers 1–5.

4.2 Occlusion Sensitivity Analysis

We implemented an occlusion-based sensitivity analysis to interpret the behavior of the model when classifying images. The steps involved systematically masking regions of the input image and observing how the model's predictions are affected. This process allowed the identification of image regions that are critical for the model's decision and better understand it's behavior.

The occlusion sensitivity approach was designed to evaluate the importance of different spatial regions in the input image with respect to the predicted class. By sliding a fixed-size occluding patch across the image and recording the model's output, it is possible to quantify the influence of each region on the prediction. Two main outputs are generated: a probability drop map, indicating the change in confidence for the true class when each region is occluded, and a prediction map, showing the new predicted class at each occlusion step.

4.2.1 Occlusion Implementation

`get_occlusion_result` : This function applies a gray patch to a specific region of the image and checks how the model reacts. Given the input coordinates and patch size, it replaces the selected area with a neutral gray value, runs the image through the model, and returns the true class probability, the predicted class, and the activation of the most responsive feature in layer 5.

`generate_occlusion_maps` : This function slides an occluding patch over the image using a defined stride. At each position, it calls 'get_occlusion_result' to record the activation, true class probability, and predicted class. These results are stored as three maps: activation, probability drop, and predicted class. The patch size controls how much of the image is hidden, while the stride sets the spatial resolution of the analysis.

`plot_occlusion_entry` : This function displays a comprehensive visualization of an occlusion implementation. It includes: (a) the occluded image with patch location, (b) the activation map showing how occlusion impacts the most active feature in layer 5, (c) a reconstruction of the most activated feature via deconvolution, (d) a heatmap of the true class probability when different regions are occluded, and (e) a map showing how the predicted class changes with occlusion. These subplots allow an intuitive and visual understanding of how different regions of the image influence the model's decision.

The full analysis pipeline was implemented in a set of notebook cells that begin by randomly selecting an image from the test set. The image and its true label are passed through the model to extract activations from layer 5, and the most strongly activated feature map is identified. A one-hot tensor is used to isolate and project this feature back to input space using the deconvolutional network. Then, the function 'generate_occlusion_maps' is used to compute the occlusion sensitivity maps, and one specific occlusion position (centered in the image) is selected to generate the visualization with 'plot_occlusion_entry'.

To assess how different occlusion configurations impact the model's decision-making process, we tested several scenarios across the three datasets using varying patch sizes and strides. Below, we show the occlusion results for different settings and datasets. A detailed interpretation of these results is provided in the Results section.

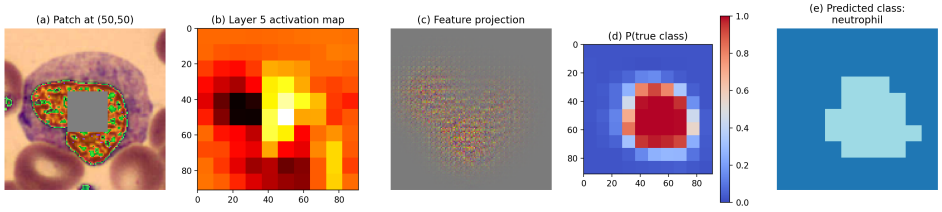


Figure 14: BloodMNIST occlusion, patch size 32×32, stride 10 and correct classification.

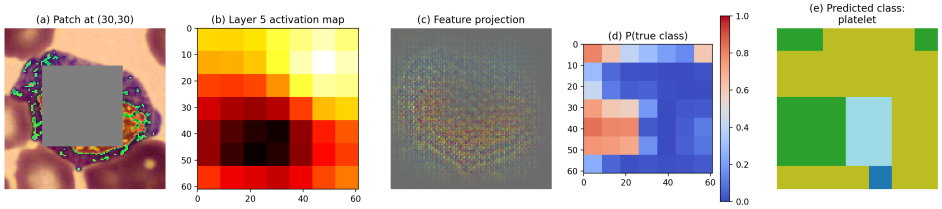


Figure 15: BloodMNIST occlusion, patch size 64×64, stride 10 and wrong classification.

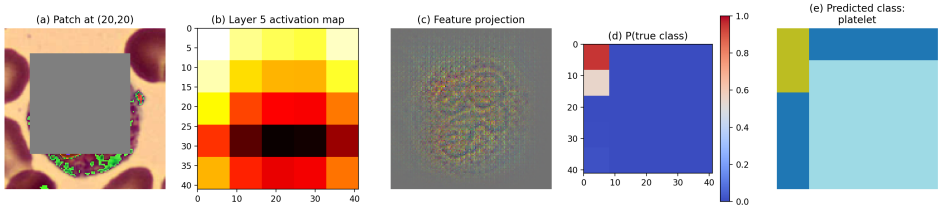


Figure 16: BloodMNIST occlusion, patch size 80×80, stride 10 and wrong classification.

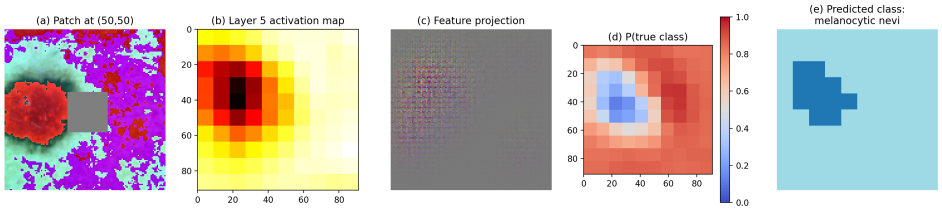


Figure 17: DermaMNIST occlusion, patch size 32×32, stride 10 and correct classification.

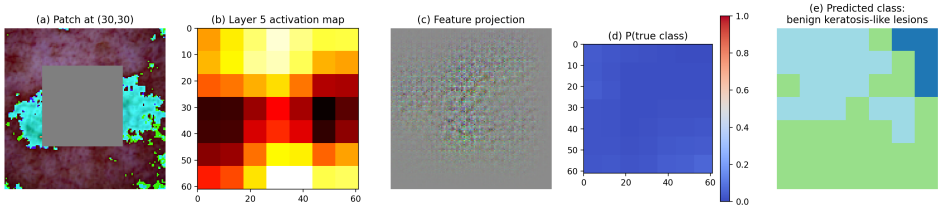


Figure 18: DermaMNIST occlusion, patch size 64×64, stride 10 and correct classification.

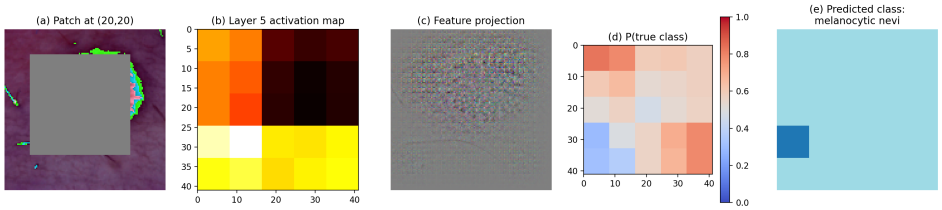


Figure 19: DermaMNIST occlusion, patch size 80×80, stride 10 and correct classification.

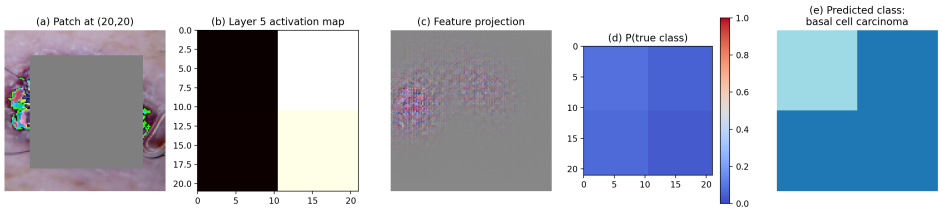


Figure 20: DermaMNIST occlusion, patch size 90×90, stride 20 and wrong classification.

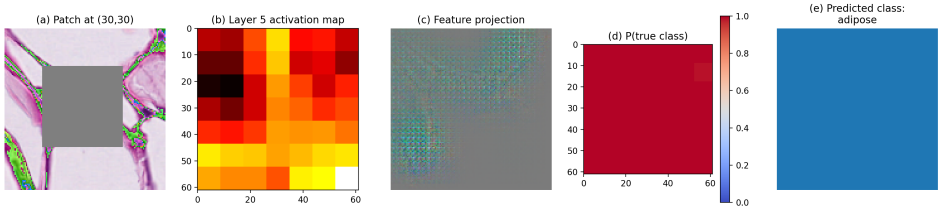


Figure 21: PathMNIST occlusion, patch size 64×64, stride 10 and correct classification.

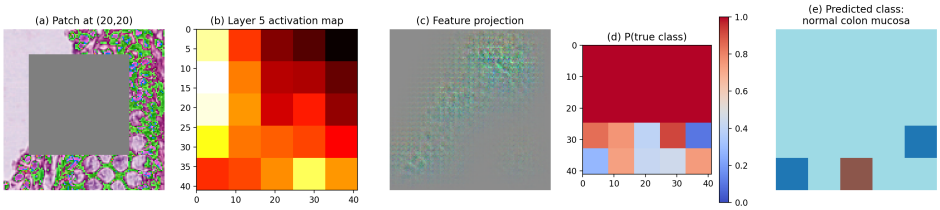


Figure 22: PathMNIST occlusion, patch size 80×80, stride 10 and correct classification.

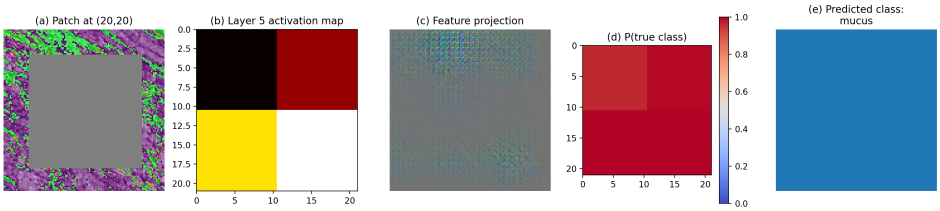


Figure 23: PathMNIST occlusion, patch size 90×90, stride 20 and wrong classification.

5 GradCAM

The second CNN visualization method explored by the group was through the Gradient-weighted Class Activation Mapping approach. This method's idea is to get, through the gradient of the final score y^c of a class c with respect to a set of feature maps A^k from a given layer k, an image that highlights the regions that have higher impact on the CNN's score for class c, i.e., a heat map.

5.1 Theoretical Basis

The central theme of the GradCAM method is that, given a trained CNN that takes an image X, of dimensions $n \times m$, and returns a score vector $y \in R^C$, it is possible to easily identify for a given layer the most important regions in an image for the final classification of the network. However, when we change the focus to identifying the regions that are the most relevant for a specific class c, the task becomes significantly harder, as all feature maps K in layer l are used for calculating the final scores.

The solution proposed by the GradCAM method is to measure the relevance of each feature map A_k for a given class y^c. This is done by calculating the partial derivative of y^c with respect to each feature map A_k in layer l:

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

This procedure is then followed by a ReLU-activated weighted summation of all the feature maps A_k:

$$A_f = \text{ReLU}\left(\sum_k \alpha_k^c A^k\right)$$

Finishing with a final feature map A_f.

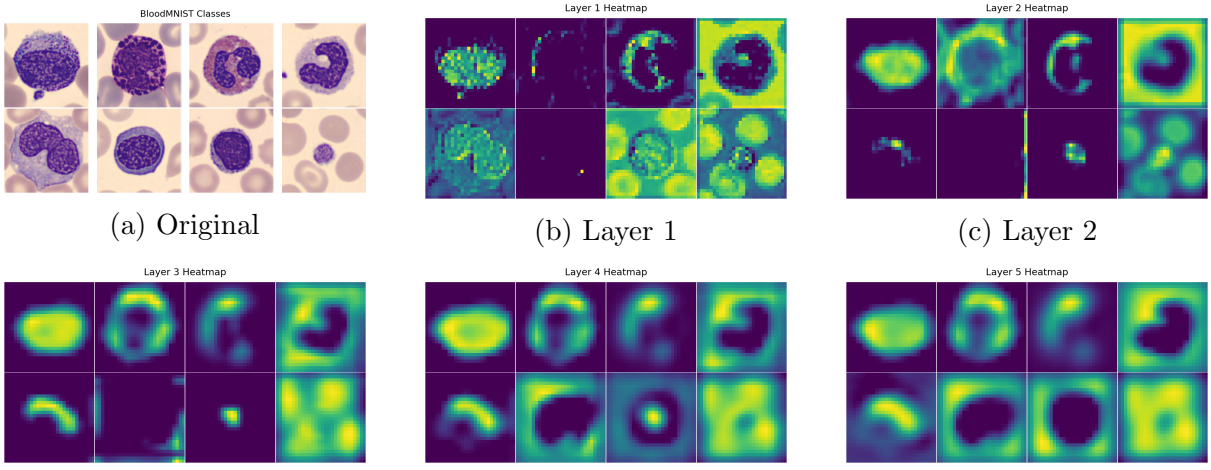
Finally, we can upsample A_f by a bilinear interpolation to arrive at an image of the desired dimensions $n \times m$.

5.2 Implementation

The implementation of the GradCAM method was done by creating a class gradcam, that received in its constructor the trained CNN. The central functionality of this class would then be achieved with the create_heatmap method, which would receive as inputs, besides the image, the indices of the layer and class we want to get the heatmap from. Auxiliary methods _upsample and _combine_feature_maps were also implemented to aid the development of the method create_heatmap. The first method upsamples an image using bilinear interpolation. The second takes as inputs a tensor storing the feature maps from the layer being analyzed and another one of the same dimensions storing the partial derivatives of the chosen class c with respect to these feature maps. It then takes the average intensity of pixels in each derivative map and does the weighted summation of the feature maps, storing the result in a final two-dimensional image.

5.2.1 create_heatmap

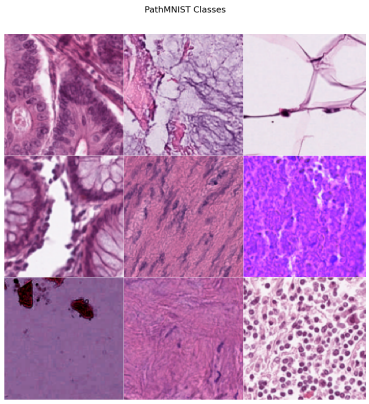
The create_heatmap method aims to get both the feature maps of the specified layer and the gradients linked to them. Then, it calls the _combine_feature_maps internal method and upsamples its output to the dimensions of the original image.



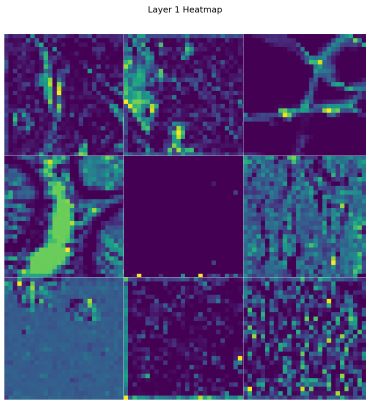
(a) Original (b) Layer 1 (c) Layer 2

(d) Layer 3 (e) Layer 4 (f) Layer 5

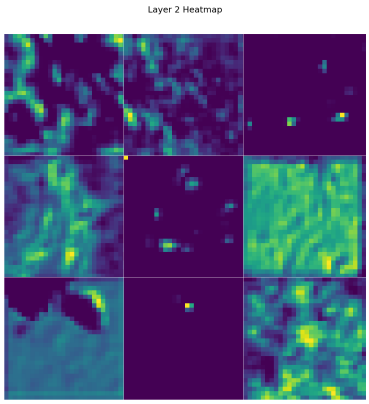
Figure 24: BloodMNIST classes input alongside GradCAM reconstructions from layers 1–5.



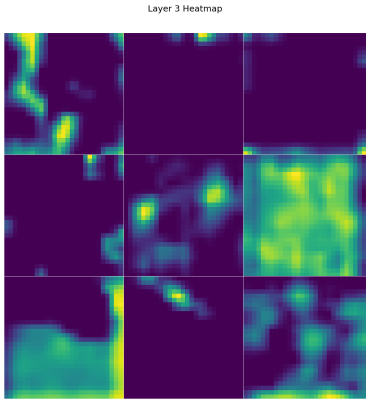
(a) Original



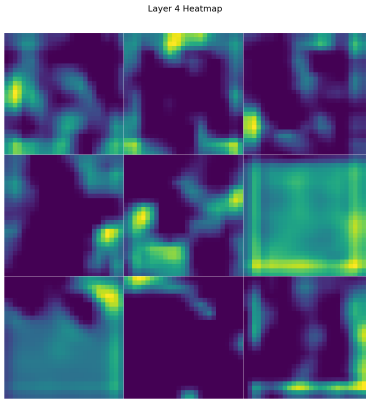
(b) Layer 1



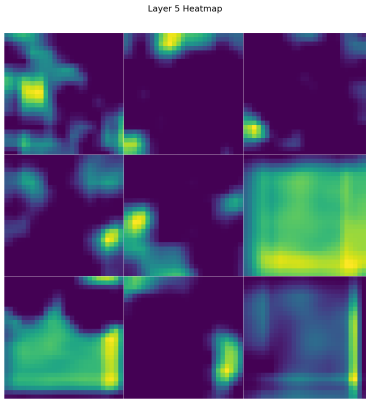
(c) Layer 2



(d) Layer 3



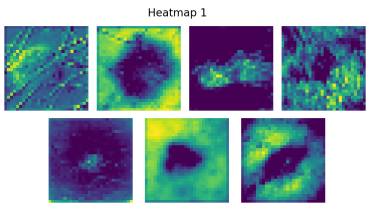
(e) Layer 4



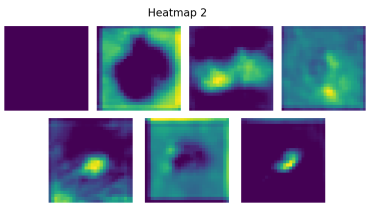
(f) Layer 5

Figure 25: PathMNIST classes input alongside GradCAM reconstructions from layers 1–5.

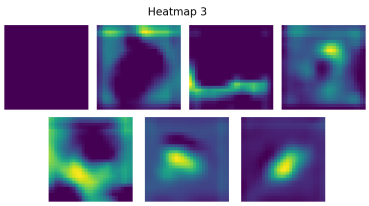
(a) Original



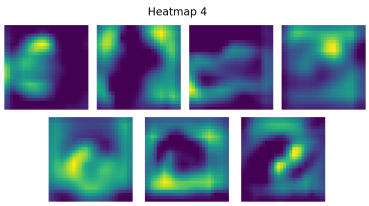
(b) Layer 1



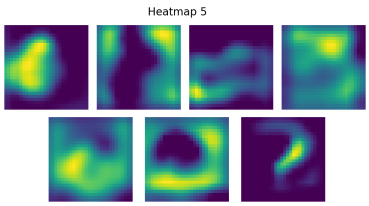
(c) Layer 2



(d) Layer 3



(e) Layer 4



(f) Layer 5

Figure 26: DermaMNIST classes input alongside GradCAM reconstructions from layers 1–5.

6 Results

By analysing the visualisations of neuron excitations obtained through the DeconvNet (as seen on 11, 12 and 13) we can notice that in the first layer, our CNN is focusing more in the general structure of the image, meaning that its getting edges and the shapes in general of the important structure being displayed at each image. In layer 2 it seems to be focusing on overall texture of each important part of the image. Layer 3 feels like it's a mixture of overall structure of each image, with the exception for the DermaMNIST, in which layer 3 seems to be more abstract than the other 2 datasets. This can be due to its inferior accuracy on the overall training. Layers 4 and 5 seems to be focusing on the specific class parts of each image, these are features that are define and distinguish each class.

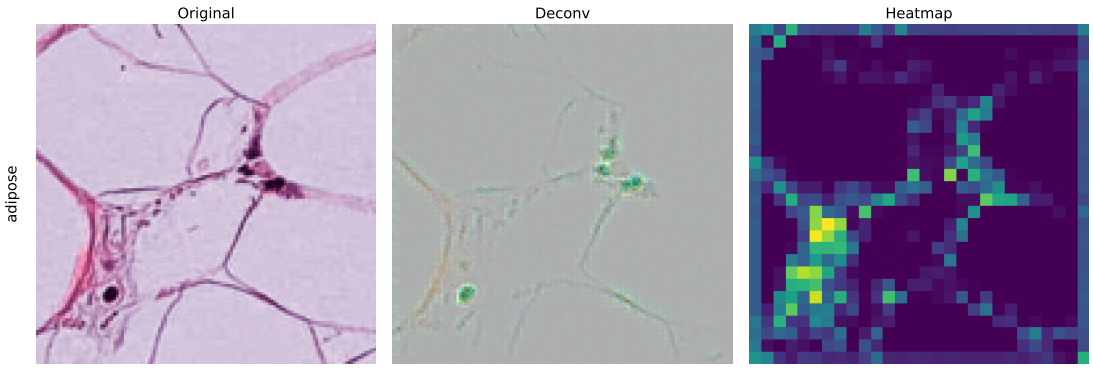
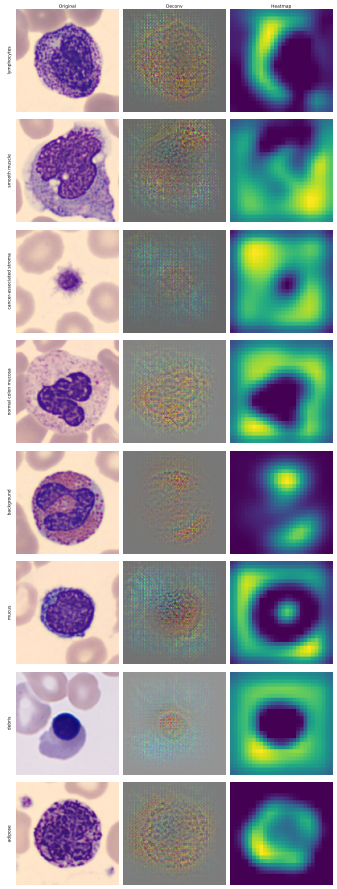
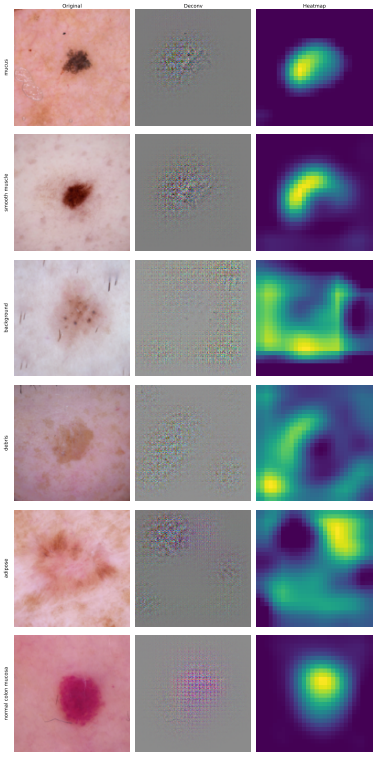


Figure 27: Layer 1 PathMNIST class adipose. As we can see on the initial layers the model focus on the general image contours

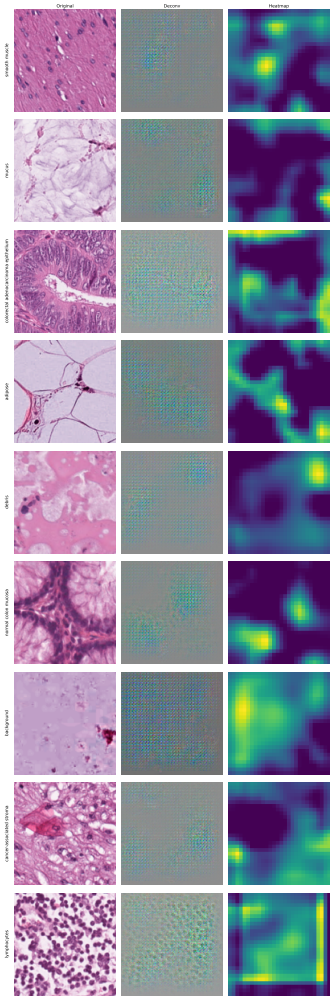
Each one of the 3 different explainable methods were applied to the three used datasets. Throughout the layers, the more external the layer, the more significant and interpretable the output visualization on each one of the used methods. As expected in the functioning of convolutional neural network they represent the input through a more complex non-linear that better aproaches the original one. Thus the last layer will foucus on our analysis.



(a) Blood



(b) Derma



(c) Path

Figure 28: Layer 5 comparision

By the GrandCAM method we can better analyse the highlighted of the main focus representation of the input data. Even so, some of the outputs obtained through this method have little information to be seen, as through the DeconvNet we can better see the details of neuron activation on each layer, as it is possible to see on figure 29. Some of the visualizations on GradCAM can even appear more empty as it lacks on details compared to the DeconvNet, as we can see on 30

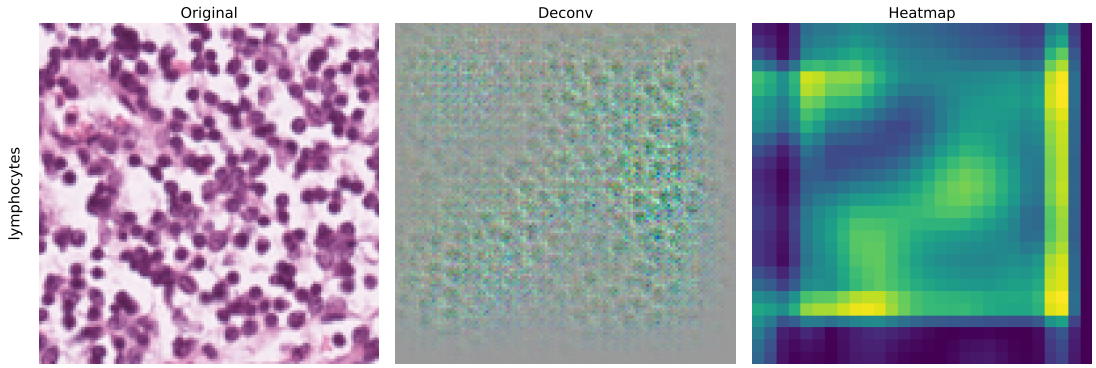


Figure 29: Layer 4 PathMNIST

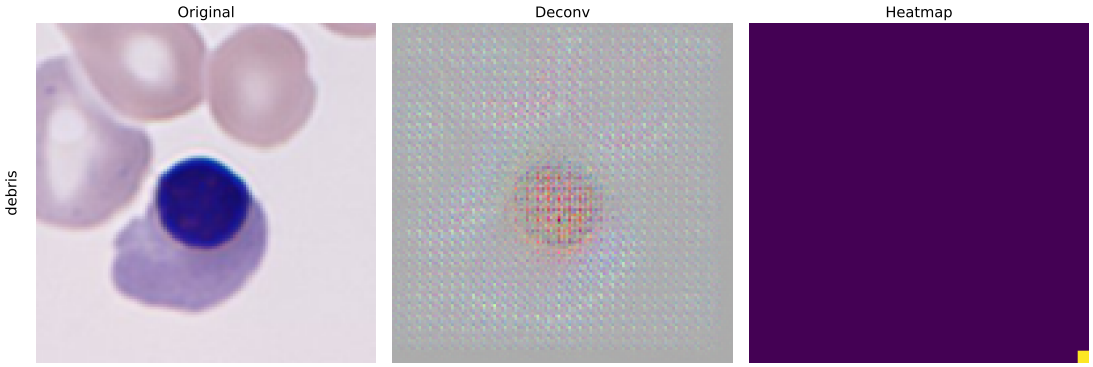


Figure 30: Void representation on GradCAM exemple

Regarding the occlusion results, we can better see its influence on the BloodMNIST dataset, as the main information is concentrated on the round blood cell on the image. When the part of the image that contains the blood cell is occluded, the model's accuracy highly decreases. While on the PathMNIST dataset as the main information covers the full image, when occluding any part of the image, the network still has sufficient information from the rest of the image in order to classify.

For the BloodMNIST dataset, we started with a small patch size of 32×32 with stride 10 (Figure 14) that led to a correct classification. However, with larger patches (64×64 and 80×80), as seen in Figures 15 and 16, occluding central regions significantly degraded model performance and resulted in misclassification. This suggests that the classifier relies heavily on localized features concentrated near the cell core.

In the DermaMNIST dataset, the model remained accurate with occlusions of 32×32, 64×64, and 80×80 (Figures 17 to 19), suggesting it relies on features spread across the lesion. However, with a larger 90×90 patch and stride 20 (Figure 20), the prediction failed, likely due to loss of contextual information. Also, it is worth saying that, probably due to the strong class imbalance in the dataset, nearly all test samples we had were from the majority class, *melanocytic nevi*.

For the PathMNIST dataset, the classifier remained robust to occlusions up to 80×80 (Figures 21 and 22). However, in Figure 23, the larger occlusion blocked key features, leading to misclassification. This reinforces the role of both patch size and stride: smaller strides provide finer detail, while larger patches reveal the model's sensitivity to specific regions

These examples show us how occlusion sensitivity maps can reveal the regions most influential to the prediction, and evaluate the robustness of learned features.

7 Conclusion

In conclusion, it was possible to obtain great results through each Explainable AI method. Each one of them generated different, still coherent, visualizations of the classification model latent space. With these visualizations it is possible to have a glance into the main features considered during classification.

The different aspects reveal DeconvNet's detailed output excitation of the layer's neurons; the occlusion model made it possible to find out the most relevant part of the input image for classification. Finally, the Grad-CAM model's visual representation of the latent space shows more focused feature highlights.

References

[1] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," 2013.

[2] R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, and D. Batra, "Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization," *CoRR*, vol. abs/1610.02391, 2016.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems* (F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.

[4] J. Yang, R. Shi, D. Wei, Z. Liu, L. Zhao, B. Ke, H. Pfister, and B. Ni, "Medmnist v2-a large-scale lightweight benchmark for 2d and 3d biomedical image classification," *Scientific Data*, vol. 10, no. 1, p. 41, 2023.