RDFIA - M2 IMA

Practical work report 2-b
Visualizing Neural Networks

Done by :

Carlos GRUSS & Oussama RCHAKI

25/09/24

SORBONNE UNIVERSITÉ

Contents

Objective

This practical work aims to explore and review several commonly used methods for understanding how Convolutional Neural Networks (CNNs) process input images in classification tasks. Formally, a CNN can be described as a function f that maps an input image $\mathbf{x}$ to a score vector $\tilde{\mathbf{y}}$, where the vector's dimension corresponds to the number of possible classes:

$$f : \mathbb{R}^{h \times w \times c} \to \mathbb{R}^K, \quad \mathbf{x} \mapsto \tilde{\mathbf{y}}$$

The focus of this work is on three distinct methods that share a common principle: the analysis of the gradient of the output score with respect to the input image:

$$\frac{\partial \tilde{y}_i}{\partial \mathbf{x}}$$

Rather than training a new network, we will utilize the pre-trained SqueezeNet model from Iandola et al. (2016). SqueezeNet, trained on the ImageNet dataset, is a compact and efficient architecture that achieves competitive performance while maintaining a lightweight design.

1 – Saliency Map

Saliency maps were first proposed by Simonyan et al. (2014) as a method to visualize the contribution of each pixel in the input image to the classification decision for a specific class. The approach involves approximating the neural network around a given input image belonging to class i by a linear function:

$$\tilde{y}_i = f_i(\mathbf{x}) \approx \mathbf{w}_i^\top \mathbf{x} + b_i, \quad \text{where} \quad \mathbf{w}_i = \frac{\partial \tilde{y}_i}{\partial \mathbf{x}}.$$

To generate the saliency map, the gradient $\mathbf{w}_i$ is first computed. The authors then propose taking the absolute value of this gradient and, for each pixel, selecting the maximum value across the 3 RGB channels. This process results in a 2D matrix, which serves as the saliency map, highlighting the regions of the image most influential to the classification decision.

Question 1

The results are presented in Figure 1. Saliency maps highlight the pixels in the input image that have the greatest influence on the logit corresponding to the true label of the image. For instance, in the case of an image depicting haystacks in a field, the saliency map distinctly emphasizes the regions corresponding to the haystacks when predicting the class "hay."

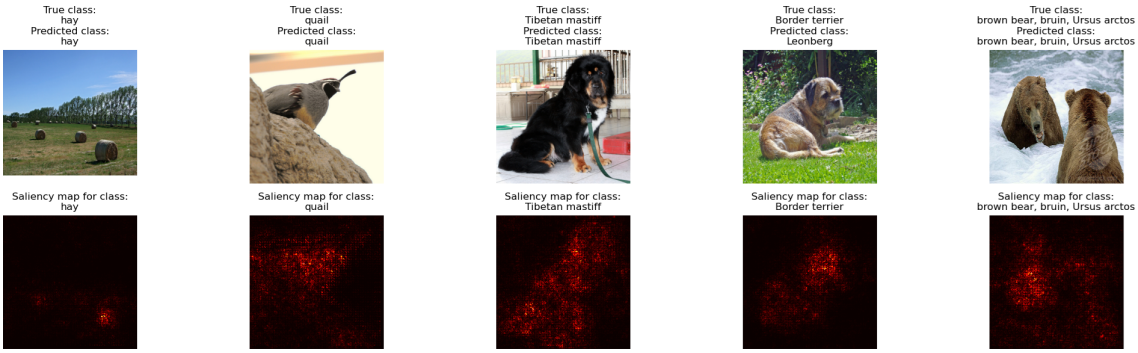

Figure 1 : Top row: Input images. Bottom row: Saliency maps for the true class.

To further investigate, we computed the saliency map with respect to a different entry in the logits vector, corresponding to an incorrect class. This approach allows us to interpret what the network focuses on when assessing whether the image belongs to another (incorrect) class. The network appears to focus on approximately the same regions of interest, indicating that these regions are broadly informative across multiple class predictions (Figure 2).

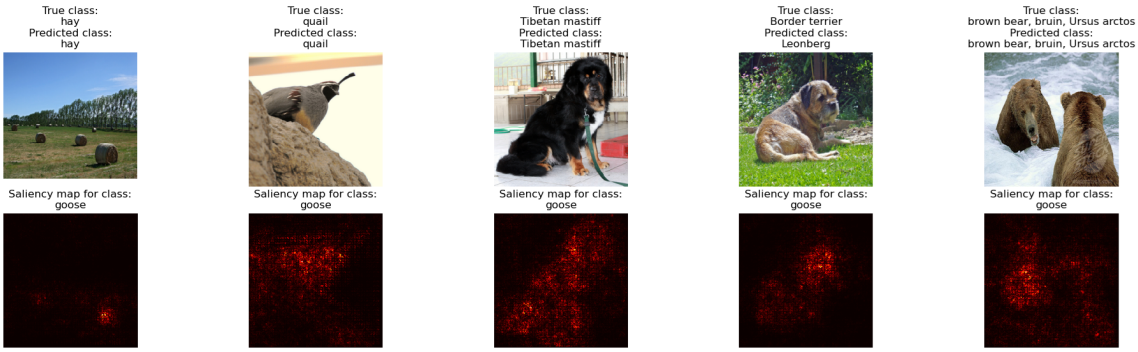

FIGURE 2 : Top row: Input images. Bottom row: Saliency maps for a wrong class.

QUESTION 2

As shown in some examples in Figure 3, saliency maps can be less informative in certain cases where nearly every pixel in the image is highlighted. This issue often arises in images with high complexity or intricate structure, such as scenes where the impact of individual pixels is less significant than the overall context they collectively represent.

Moreover, saliency maps only highlight the pixels relevant to a specific class. While the relevant pixels may overlap across the majority of classes, it could be insightful to visualize the saliency maps for other highly probable classes. This would provide a more comprehensive understanding of how the model makes its predictions and differentiates between classes.

The limitations of this basic method are also evident in examples where the model makes incorrect predictions (see Figure 4). For instance, in the lakeside image, the model incorrectly classifies the input as belonging to the greenhouse class. However, the saliency map fails to provide meaningful insights into the rationale behind this misclassification, highlighting its limitations in interpreting erroneous predictions.

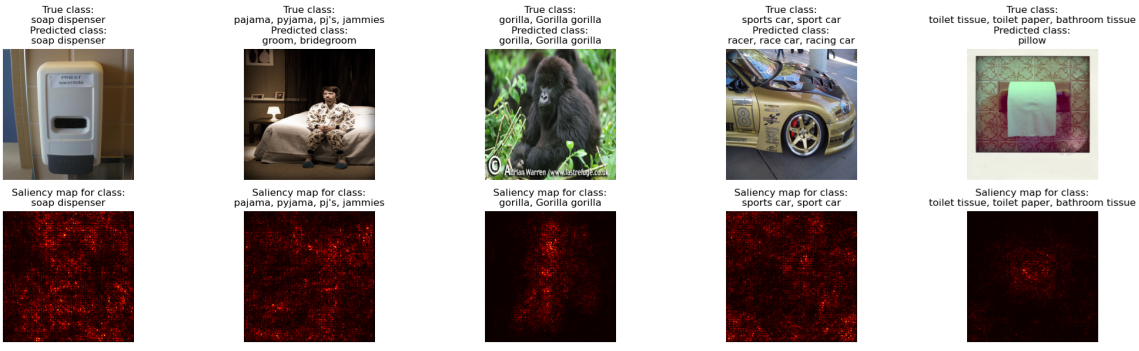

FIGURE 3 : On some examples, saliency maps highlight almost the whole image.

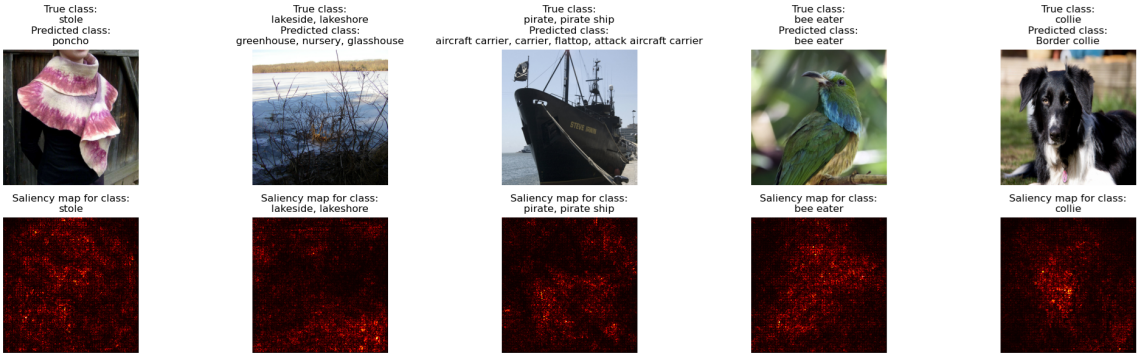

FIGURE 4 : On examples on which the image makes errors (second column).

QUESTION 3

This technique can be applied to other tasks such as pixel-wise image segmentation or object localization. By highlighting the regions of the input image most relevant to a specific class, saliency maps can help identify the spatial extent of objects within an image. This can provide valuable insights for tasks where precise localization or segmentation of objects is required, even without explicitly training the model for these purposes.

QUESTION 4

The same experiments were repeated using the VGG16 model pre-trained on ImageNet. The results were similar to those obtained previously, but the new saliency maps exhibit improved spatial resolution (Figure 5). This enhancement makes the saliency maps easier to interpret and provides greater clarity in understanding the regions of interest.

Additionally, the VGG16 model correctly classifies some examples that were previously misclassified, such as the lakeside image.

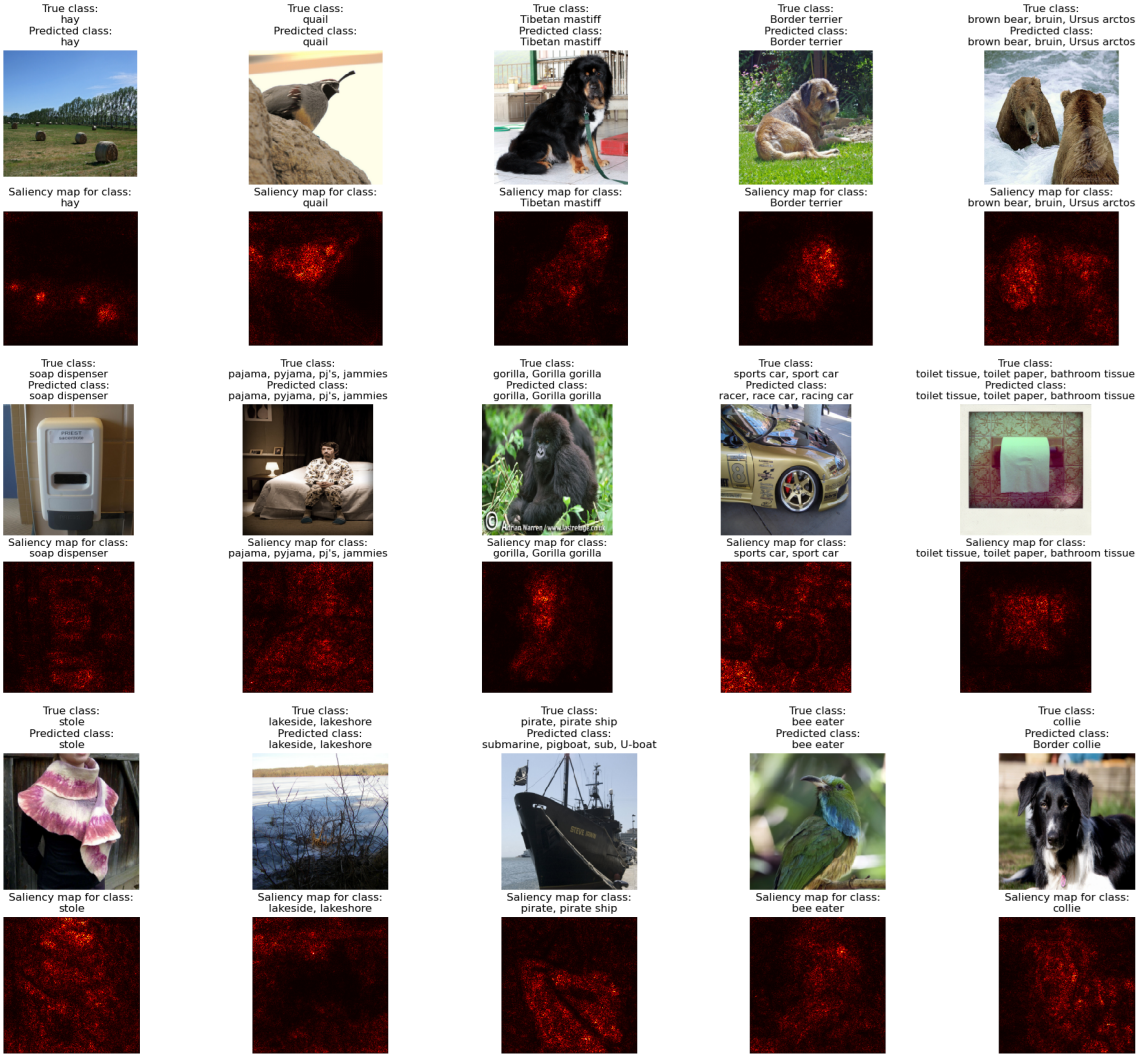

FIGURE 5 : Saliency maps using VGG16.

2 – Adversarial Examples

Adversarial examples, also called fooling samples, represent a significant aspect of neural network analysis. This practice, developed over the past few years, involves taking an image correctly classified by a pre-trained network and modifying it minimally to cause misclassification.

To create an adversarial example, we consider an image $\mathbf{x}$ correctly classified in class i. The goal is to modify $\mathbf{x}$ so it is classified into class j while keeping the network unchanged. This is achieved by computing the gradient:

$$g = \frac{\partial \tilde{y}_j}{\partial x}$$

where $\tilde{y}_j$ is the score for class j. The image is updated iteratively using the following rule:

$$x \leftarrow x + \eta \frac{g}{\|g\|_2}$$

where η is the learning rate. This process is repeated until the network's prediction for $\mathbf{x}$ becomes class j.

Question 5

The analysis of adversarial examples reveals that the modified image is visually almost identical to the original image from the perspective of the human eye. As shown in Figure 6, the difference between the original and fooled images is imperceptible. However, the difference map clearly highlights that the modifications primarily affect the **key features** or **distinctive characteristics** of the true class in the original image. These targeted changes are sufficient to deceive the model into misclassifying the image, illustrating the vulnerability of neural networks to adversarial attacks.

This raises a significant concern: Adversarial attacks demonstrate that neural networks can be manipulated by changes so subtle that they are practically invisible to humans. Such small perturbations, often within the range of numerical rounding errors, can drastically alter a network's predictions. For example, the research by Su, Vargas, and Sakurai (2019) in their article titled *"One Pixel Attack for Fooling Deep Neural Networks"* illustrates that altering a **single pixel** in an image can lead to misclassification with high confidence. The study employs a black-box optimization technique called *differential evolution*, which crafts adversarial examples without requiring access to the model's internal gradients. This demonstrates the fragility of deep learning models to even the smallest perturbations.

The implications of such vulnerabilities are particularly alarming in high-stakes applications where the model's predictions must be accurate and reliable. Consider, for instance, **medical imaging systems**: An adversarial perturbation could cause a model to misdiagnose a critical condition, potentially endangering a patient's life. Similarly, in **autonomous vehicles**, even a minor perturbation in sensor data could result in catastrophic errors in object detection or path planning.

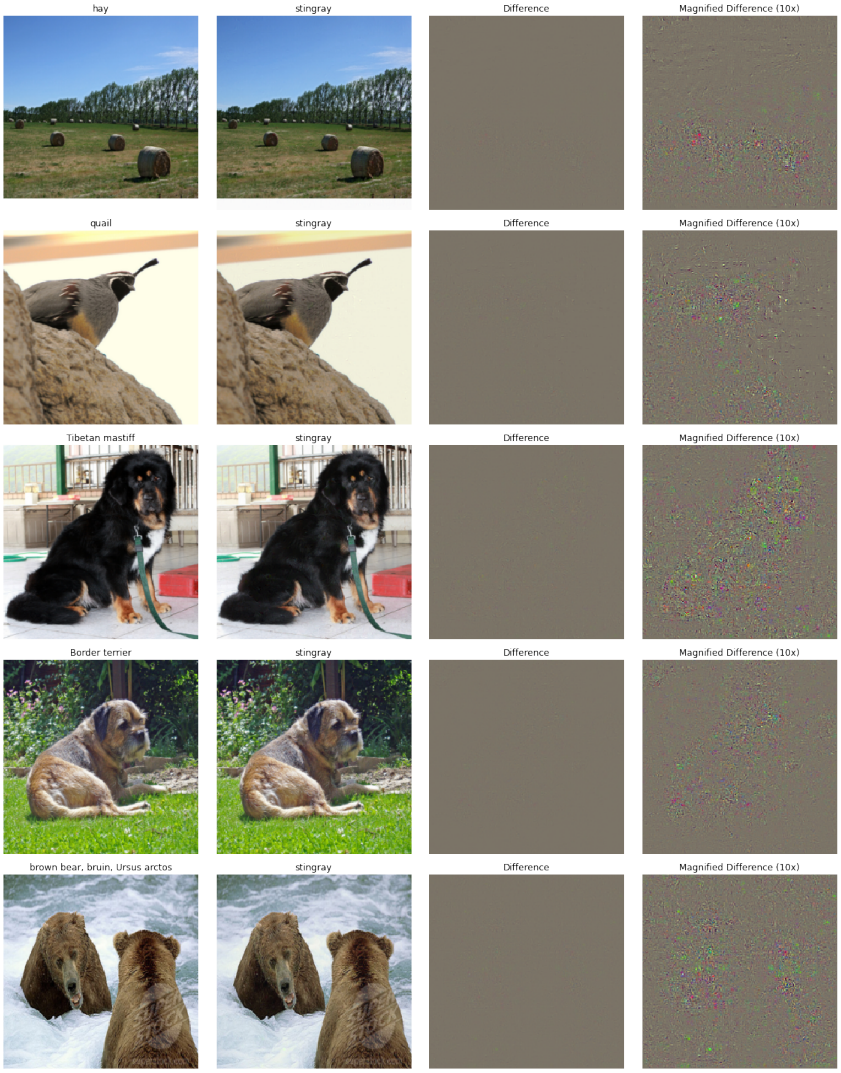

FIGURE 6 : Adversarial Examples.

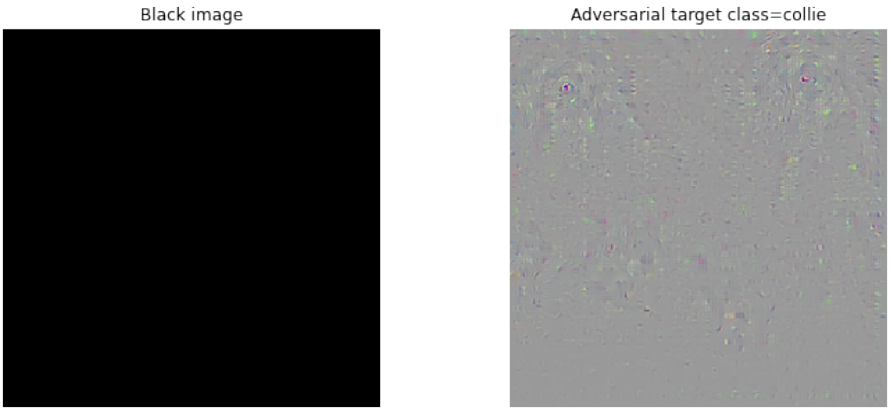

FIGURE 7 : Adversial example from a black image.

Figure 7 shows that even starting from a black image containing no information, we can modify the image in such a way that the model predicts the desired class for the modified image.

Question 6

The method of generating adversarial examples has several significant consequences for convolutional neural networks (CNNs):

- **Vulnerability to Adversarial Attacks:** CNNs, despite their high performance in image classification tasks, are highly sensitive to small, targeted perturbations in input data. This demonstrates a fundamental lack of robustness in their predictions. The existence of adversarial examples undermines trust in CNN-based systems, necessitating the development of defenses and robust training methods to mitigate such risks. Adversarial attacks pose serious risks in applications where CNNs are deployed for critical tasks, such as medical diagnostics, autonomous driving, or financial decision-making. Even minor perturbations can lead to catastrophic outcomes, as the predictions become unreliable and untrustworthy.
- **Reliance on Learned Features:** The success of adversarial attacks highlights that CNNs rely heavily on specific learned local features of the data, which can be easily manipulated. This suggests that CNNs may fail to generalize to the core semantics of the input.

Question 7

The naive approach to constructing adversarial images described in the problem has several limitations:

- **Computational Efficiency:** Iteratively applying gradient updates can be computationally expensive, particularly for large images or complex networks. This makes the method impractical for real-time scenarios.
- **Global Perturbation:** The method modifies all pixels in the image, even those that may not contribute significantly to the classification. This is inefficient and can lead to unnecessary changes.
- **Detectability:** The generated adversarial examples may still be detectable by adversarial defense mechanisms, particularly if the perturbations follow a recognizable pattern.
- **Transferability:** The generated adversarial examples are often specific to the architecture and weights of the targeted model. They may not transfer effectively to other models, limiting their practical utility.

Some proposed alternatives in more recent research are:

- **One-Pixel Attacks:** As demonstrated by Su et al. (2019), adversarial examples can be created by modifying only a single pixel in the image. This approach is computationally efficient and harder to detect.
- **Fast Gradient Sign Method (FGSM):** Proposed by Goodfellow et al. (2014), FGSM uses a single step in the gradient direction to generate perturbations, making it computationally faster and more practical.
- **Adversarial Training:** Incorporating adversarial examples into the training process helps improve the robustness of CNNs by exposing them to potential attacks during training.
- **Black-Box Attacks:** Using optimization-based methods like differential evolution or evolutionary algorithms to generate adversarial examples without requiring access to the model's gradients, improving transferability and applicability to unknown models.

In conclusion, while the naive approach provides a foundation for generating adversarial examples, more advanced techniques offer improved efficiency, stealth, and generalizability. These methods continue to evolve as researchers strive to address the vulnerabilities of neural networks.

3 – Class Visualization

Question 8

Some examples of the results produced by the implemented algorithm are shown in Figure 8. Although the spatial resolution is relatively low and the outputs are quite noisy, certain structures can be discerned that provide an abstract representation of the intended class. These visualizations offer insights into the features that the model considers characteristic of a specific class, helping us better understand its internal abstractions.

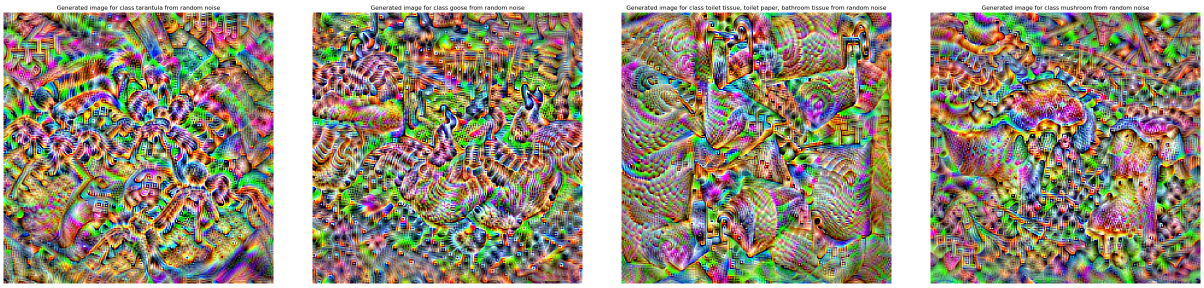

Figure 8 : Generated examples for different classes starting from random noise.

These images can also be generated by initializing the algorithm with an image that is not random noise, as demonstrated in Figure 9.

Figure 9 : Adversial example from a black image.

Question 9

The effect of the number of iterations is quite evident: the more iterations the algorithm is allowed to perform, the more intricate the patterns ingrained into the image. However, the gradient ascent eventually appears to converge to a local maximum, after which no further significant changes occur.

The effects of varying the regularization weight are more interesting and are illustrated in Figure 10. As the regularization weight decreases, fewer recognizable high-level features of the target class emerge. Our intuition suggests that reducing the regularization penalty causes the algorithm to focus more heavily on maximizing the objective loss through changes in low-level, high-frequency features. These features, such as textures, are often difficult for humans to interpret or distinguish from noise.

In contrast, the effects of varying the learning rate are less pronounced. It can be observed, however, that certain structures appear more prominently as the learning rate increases (e.g., the lines in the background of the images in Figure 11).

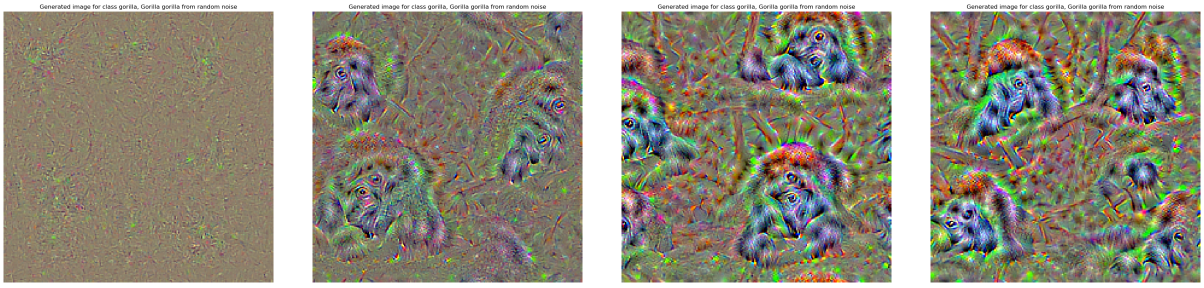

FIGURE 10 : Results obtained for gorilla class after 200 iterations with a fixed learning rate of 5 and varying regularization weight. From left to right: 1×10^{-1}, 1×10^{-2}, 1×10^{-3}, 1×10^{-4}

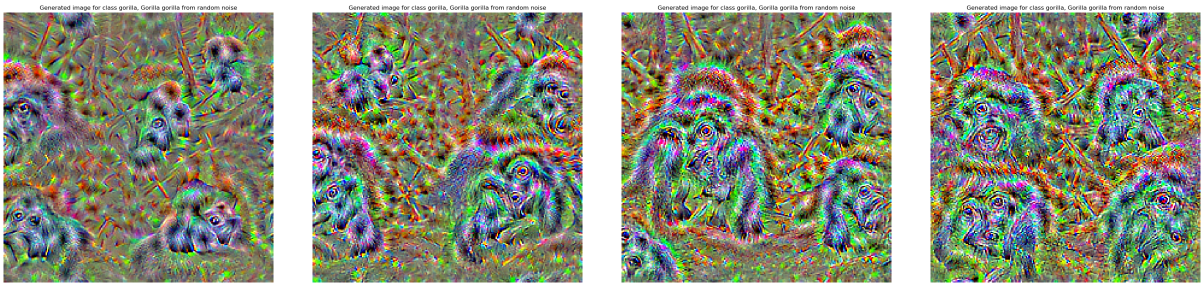

FIGURE 11 : Results obtained for gorilla class after 200 iterations with a fixed regularization weight of 1×10^{-3} and varying learning rate. From left to right: 5, 10, 15, 20

QUESTION 10

In Figure 12, we observe the results of visualizing a class by initializing the algorithm with an image belonging to the same class. This approach can be advantageous, as the initialization is likely closer to the final optimum in the image space. Consequently, the algorithm may focus on enhancing and better expressing the features already present, leading to more discernible outcomes.

Interestingly, despite initializing with an image of the target class, the final result differs significantly from the input image. This highlights how neural networks "perceive" images in an abstract, feature-based manner, providing insights into both their internal representations and the inherent limitations of statistical learning models.

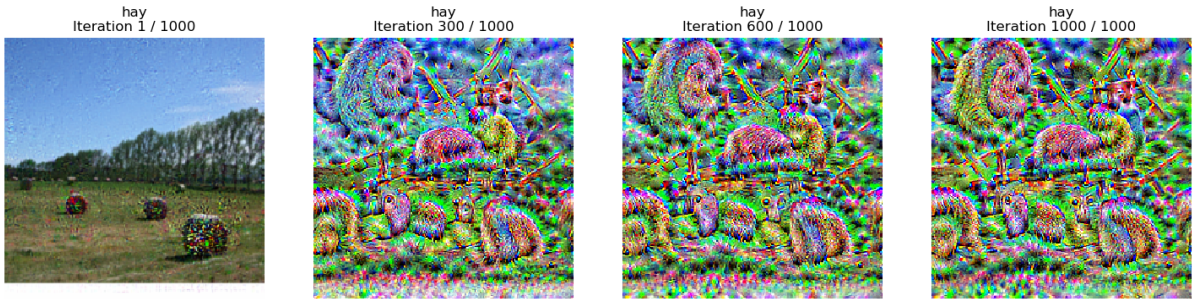

FIGURE 12 : Visualization of class hay starting from an image of the same class. Learning rate set to 15 and regularization weight 1×10^{-4}.

QUESTION 11

Finally, we experimented with using the VGG16 model pre-trained on ImageNet instead of the original network. The results are shown in Figure 13.

Notably, these experiments were significantly slower due to the increased complexity of the VGG16

architecture. This higher complexity is also reflected in the class visualizations, where the learned features appear more intricate and expressive compared to those from the simpler network.

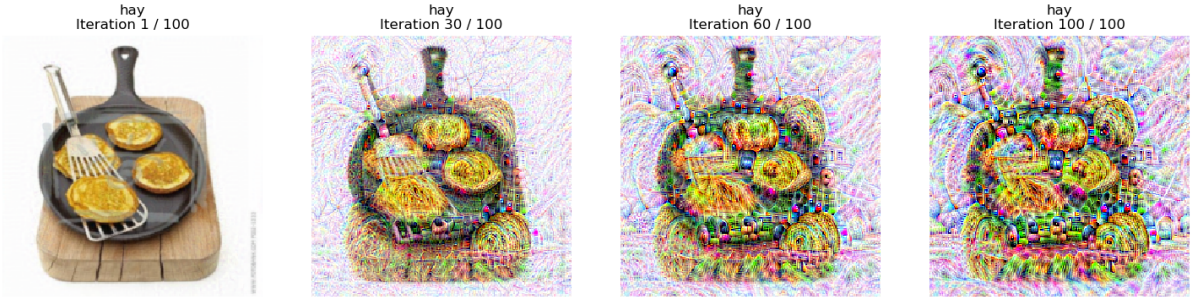

FIGURE 13 : Visualization of class hay starting from an image of pancakes, using VGG16. Learning rate is 1×10^{-4} and regularization weight is 5.