Feature Visualization of Robust Neural Networks (RNNs)

Team 5: Ewan Golfier, Ulysse Widmer, Mariia Eremina

Spring 2023

Abstract

This project aims to employ feature visualization techniques to analyze sensitive neurons, channels, or layers in both robust and non-robust neural networks. Specifically, we will focus on the ResNet34 model and compare its non-robust variant obtained from the PyTorch library with the robust variant obtained from Robust.Art. To identify the sensitive neurons, we will generate adversarial images using the validation dataset of ImageNet100 to deceive the models. Subsequently, we will compare and contrast the feature visualizations of the sensitive neurons across the different models.

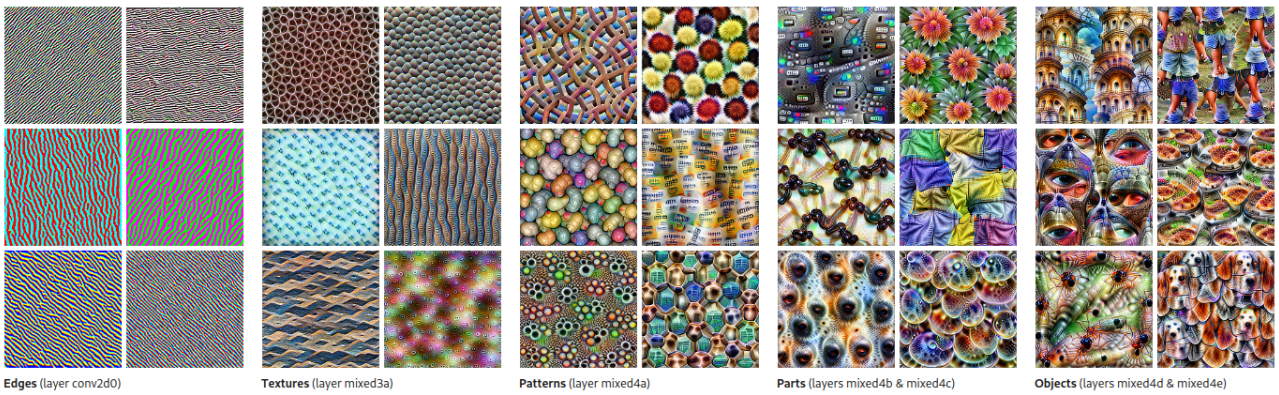

Figure 1: Feature visualizations examples from `https://distill.pub/2017/feature-visualization/`

1 Introduction

1.1 Feature visualization

Feature visualization is an important technique used in the study of neural networks, aiming to generate visual representations that highlight the most activating aspects of a targeted component within the network. The process of feature visualization generally involves the following steps::

1. Selection of the Targeted Component: A specific component of the network, such as a neuron, channel, or layer, is chosen as the target for activation.

2. Image Optimization: Starting from an image composed of random noise, an optimization process is applied to the image in order to maximize the activation of the targeted component.

Unlike the conventional process of optimizing model parameters during training, feature visualization reverses the optimization process to focus on optimizing the input itself without modifying the weights or other network parameters.

For detail information and additional visual examples, the following resource can be explored `https://distill.pub/2017/feature-visualization/`

To facilitate feature visualization in this project, we will utilize the Lucent library library, which provides useful tools and functionalities for this purpose.

1.2 Robust Neural Networks

The neural networks we are interested in for this project are Convolutional Neural Networks (CNNs) used for image classification. For any CNN, we can presumably find adversarial input: images designed to get misclassified by the model, but are easy to classify for a human. Robust networks (RCNN) are defined by their ability to not be as easily fooled by adversarial images.

There are many ways to create robust newtorks, some are very transformative and it can therefore become hard to compare a robust and a non-robust network. In this project, we will compare two versions of Resnet34 for which the network weights differ. To achieve a robust network by only changing the weights, most implementations augment the training set with adversarial examples.

This project involves generating adversarial examples using the Fast Gradient Sign Method (FGSM) as a white-box attack. We will create these examples based on internal information

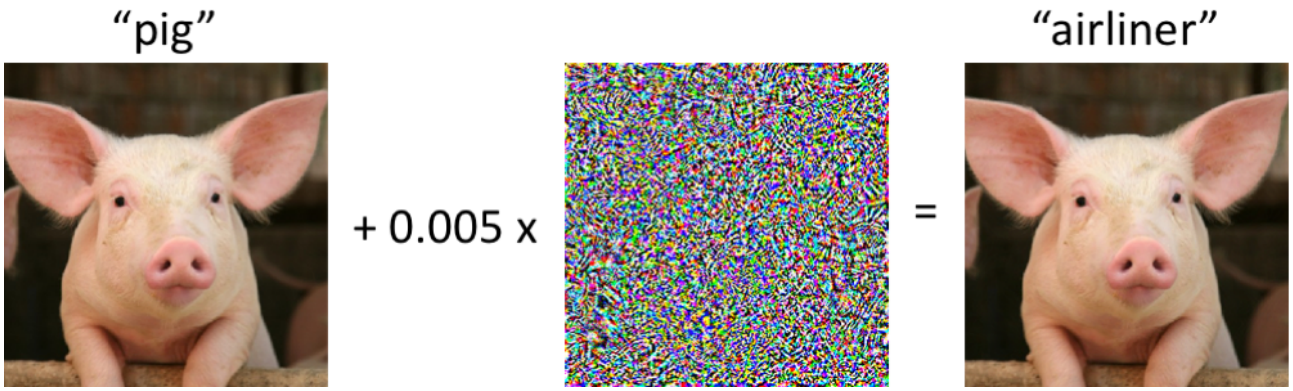

Figure 2: Example of an adversarial image fooling a network by adding a small amount of Gaussian noise to original image

about the model and evaluate both networks using these adversarial inputs. This comparison will allow us to assess the relative robustness of each network.

1.3 Resnet34

Resnet34 is a CNN for image classification, trained on the ImageNet1K dataset (1.2 million images, 1'000 classes), with the architecture described in 3.

In this project, we will be using ImageNet100's validation dataset. ImageNet100 is a subset of ImageNet1K with only 100 labels and its validation dataset reduces the number of images to 5000 (50 per label).

The non-robust model we use is the default ResNet34 implementation that comes with the PyTorch library. It's robust counterpart comes from Robust.Art.

You can find more about ResNet34 in the following paper: `https://arxiv.org/pdf/1512.03385.pdf`

1.4 Feature sensitivity

Neuron sensitivity is defined in this paper (by Zhang, Liu et al.) using the following formula :

$$\sigma(F_l^m, \bar{\mathbf{D}}) = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{\dim(F_l^m(x_i))} ||F_l^m(x_i) - F_l^m(x_i')||_1 \tag{1}$$

Where $\bar{\mathbf{D}} = \{(x_i, x_i')\}, x_i \in \mathbf{D}, x_i' \in \mathbf{D}'$. $\mathbf{D}$ is the original dataset and $\mathbf{D}'$ is the corresponding adversarial dataset. $F_l^m(x_i)$ and $F_l^m(x_i')$ respectively represent the output for the original image and its adversarial counterpart at the m'th neuron in layer l of the network.

Sensitive features can be explained as the neurons, channels or layers that are most sensitive to adversarial inputs and therefore responsible for their missclassification by the network. In the paper, as well as our project, these were chosen to be the top-k neurons with highest value of sensitivity.

2 Adversarial attack

2.1 FGSM attack

To create our adversarial examples, we used a Fast Gradient Sign Method (FGSM) attack. This method is very efficient as it is computed in a single step. It works by adding a pixel-

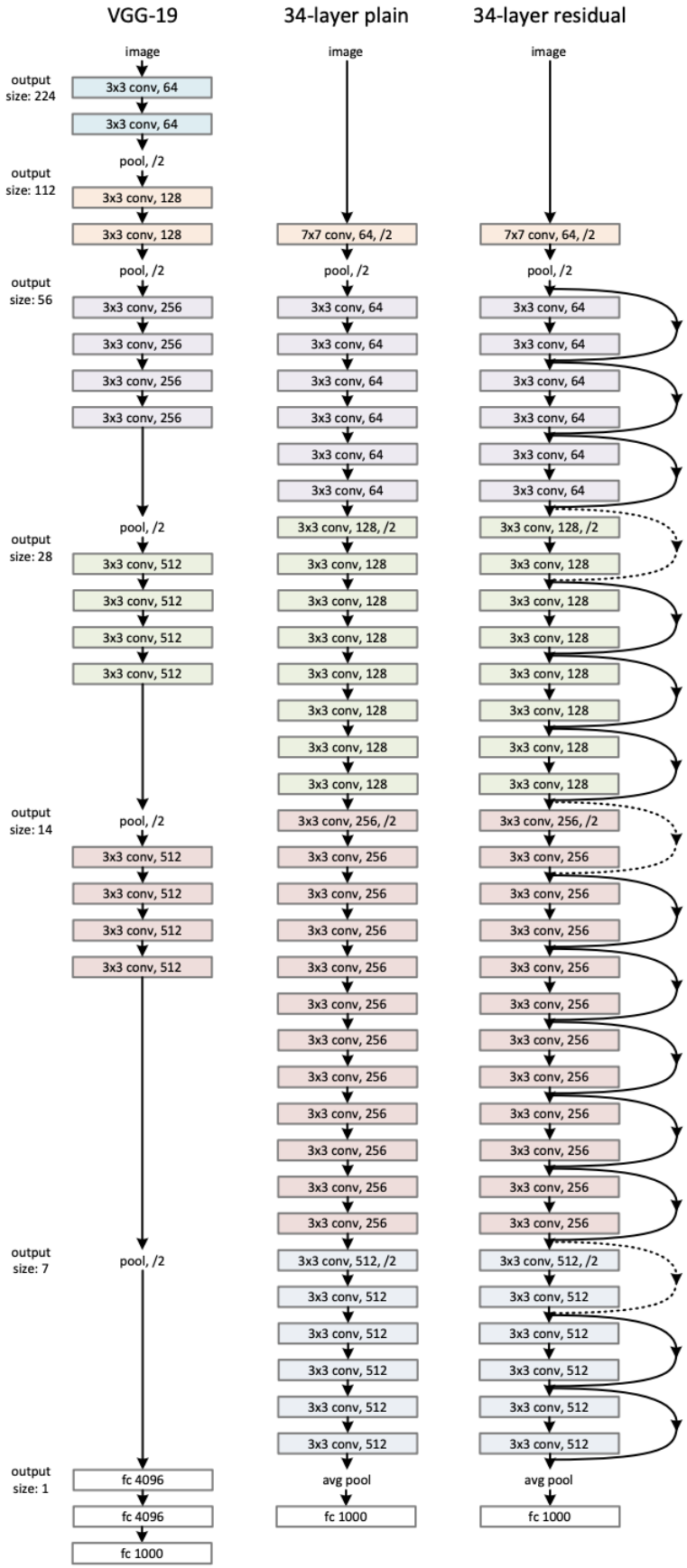

Figure 3: ResNet34 network architecture

wide perturbation in the direction of the gradient of the loss function for an input image. The formula is defined as follows :

$$x_{adv} = x + \epsilon \operatorname{sign}(\nabla_x J(x, y_{true})) \tag{2}$$

Where x is the input image, x_{adv} is the perturbed adversarial example, J is the loss function and y_{true} is the true label of the image x. ϵ is a parameter controlling the amount of perturbation in the adversarial image.

In our study, we utilize a range of epsilon values, specifically [0.0, 0.05, 0.1, 0.15, 0.20, 0.25, 0.3], to create our own adversarial dataset. These epsilon values determine the magnitude of the perturbation added to the original images. When epsilon is set to 0, it corresponds to the original image without any added noise or perturbation. By varying the epsilon values, we can generate a dataset with different levels of adversarial perturbations, enabling us to analyze the impact of varying degrees of perturbation on the performance of the models.

2.2 Robustness comparison

We chose to compare the robustness of both networks by feeding our adversarial examples to the network and plotting their accuracies as a function of ϵ.

To do this, we first precomputed and saved the signs of the sign of the gradients for all 5000 of our images in both networks, as they take a long time to compute and can be reused for all the ϵ values.

However, as we are using the validation set of ImageNet100, we only have 100 possible labels in our dataset, whereas our network output has 1'000 labels. Furthermore, these labels are not in the same format, as our dataset uses a number code to refer to the label, whereas the model outputs a number between 0 and 999. To alleviate this problem, we made a translation table, which was guaranteed to work as ImageNet100 is a subset of ImageNet1K. This lets us know y_{true} for the model while still using our dataset's number codes. Therefore we are able to compute the loss function to get our gradient.

Once we had the gradient signs, we just had to apply the FGSM attack to all our images for all epsilon. We then plotted both model accuracies on these datasets to see what model was more resistant to the perturbations.

2.3 Results

As we can see in 4, the robust model performs better than its counterpart as ϵ becomes bigger, although the non-robust model performs slightly better when little perturbation is present.

As expected, we can therefore conclude that the robust model is more robust to an FGSM attack than the non-robust one. We can also see that our attack worked, as both models decrease in accuracy as ϵ grows.

In **??** and **??**, we have an example of our generated adversarial inputs, as well as their missclassification by the network as ϵ grows. Notice how for for $\epsilon = 0.01$, the robust network correctly classifies the flamingo, whereas the non-robust one does not.

3 Feature visualization

3.1 Sensitive neurons

For the neurons sensitivity computation, we had to save a dataset of perturbed images. We settled on $\epsilon = 0.1$ and $\epsilon = 0.01$ as that is where there is a significant enough difference in accuracy, but the robust model still isn't performing too bad.

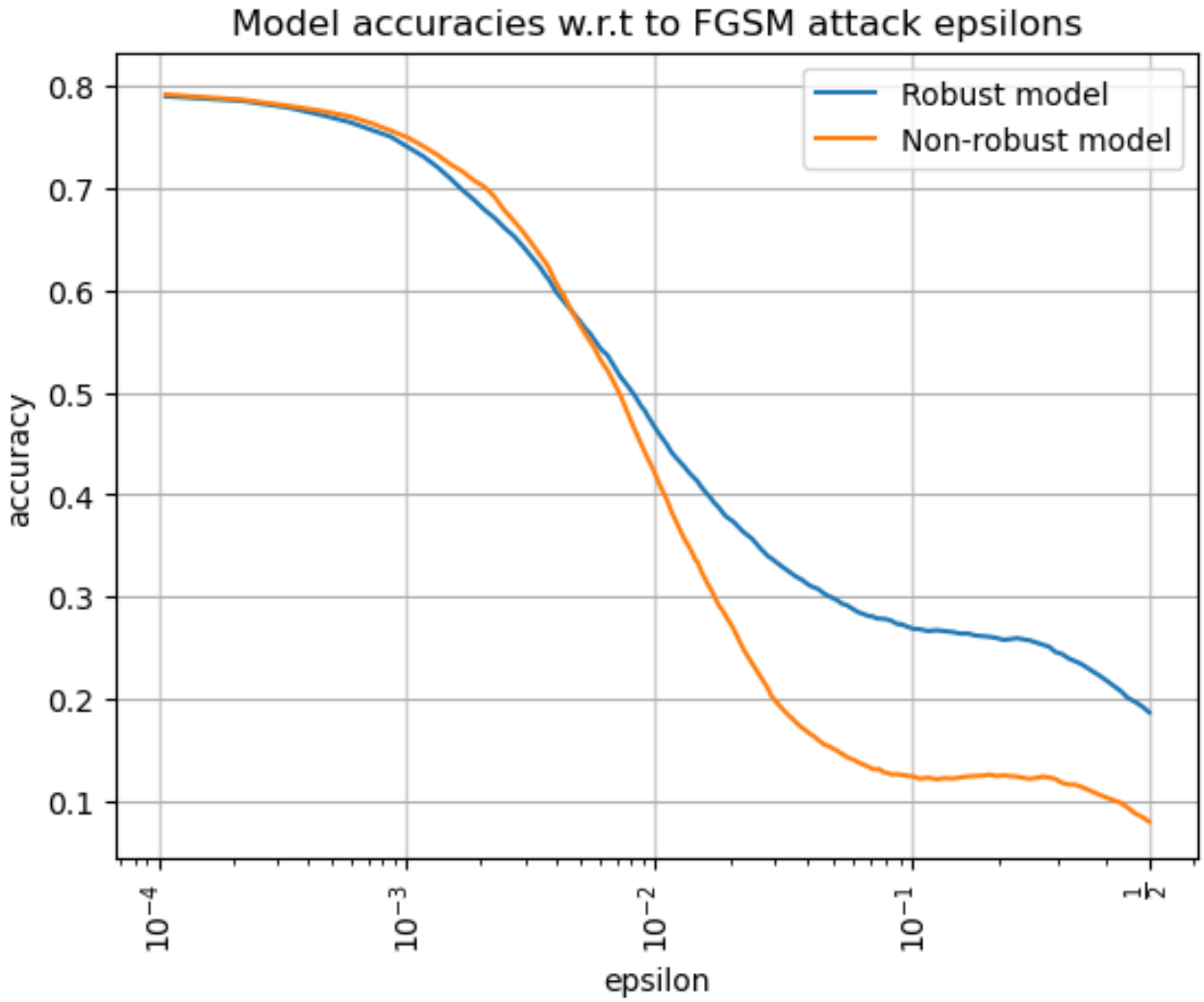

Figure 4: Robust vs non-robust model accuracies w.r.t. ϵ

Figure 5: Adversarial example for the robust network for $\epsilon = 0$ (original), 0.01 and 0.35

Figure 6: Adversarial example for the non-robust network for $\epsilon = 0$ (original), 0.01 and 0.35

To compute neuron sensitivities, we used the formula described in section 1.4. As we want to find the sensitivities for all the neurons in the network, we use PyTorch's feature extraction tool : `create_feature_extractor`. This lets us get the output for every feature in the network for a given input as a Python dictionnary of tensors. Following the formula, we sum the absolute value of the difference in network outputs, then divide by the number of images. We also divide by the dimension layer-by-layer, which is just its number of elements in the feature's layer.

Once we have the sensitivities for both the robust and non-robust model, we get their top-k sensitivities, as well as their location in the network. Those will be the features we will choose to visualize.

3.2 Sensitive feature visualization

To visualize the sensitive features, we use the Lucent library, which lets us visualize particular neurons in a network by gradually optimizing a random noise input image to maximize that neuron's activation.

In practice, we feed images from a single class to the networks and find the most sensitive neurons associated with that subset. We then use feature visualization to be able to try and interpret what those particular channels are the most sensitive.

Upon reflection, we discovered that it would have been more beneficial to first identify the sensitive neurons in the network using the entire dataset. Subsequently, we could have found interpretations for those neurons by analyzing the images that activate them the most. However, in our current approach, where we feed input to the model first and then identify the top k sensitive neurons, we observed that the sensitivity of neurons varies depending on the type of picture we feed into the model whats proof the definition of neurons.

3.3 Results

We find that, for both networks, the most sensitive neurons are located in the last layers of the network. For the non-robust model they are mostly concentrated in the fully connected layer, and for the robust model, they are generally split between the avg pool and fully connected layers.

As expected, what neurons are most sensitive depends heavily on the input. This means that for most flamingo images, the sensitive neurons will be the same, but they will be different than the sensitivities found using cat images.

We can see that by changing the ϵ value, most sensitive neurons stay the same, especially the most sensitive ones. However, this is not always the case.

Some of the sensitive neurons are present in both networks, which seems weird at first, because there shouldn't be a reason for the same neurons in a layer to be activated by the same image features. One possible explanation is that because the sensitive neurons tend to be at the end of the network, the neurons that are activating correspond to the actual image labels.

We find that the interpretability of the feature visualizations is the biggest difficulty of this project, as there exists no concreate evaluation metric. Some of the sensitive channels sometimes seem to evocate the input images, however most of them are impossible for humans to confidently interpret.

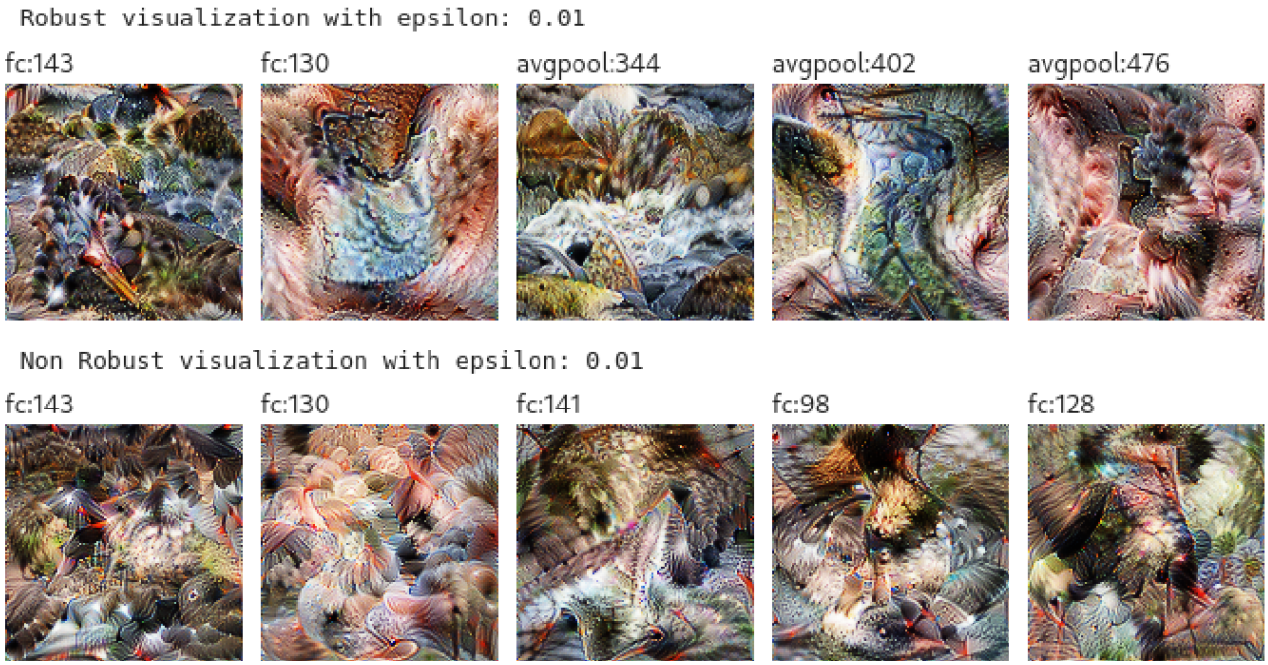

Figure 7: Sensitive neurons for the networks using flamingo images as input, $\epsilon = 0.01$

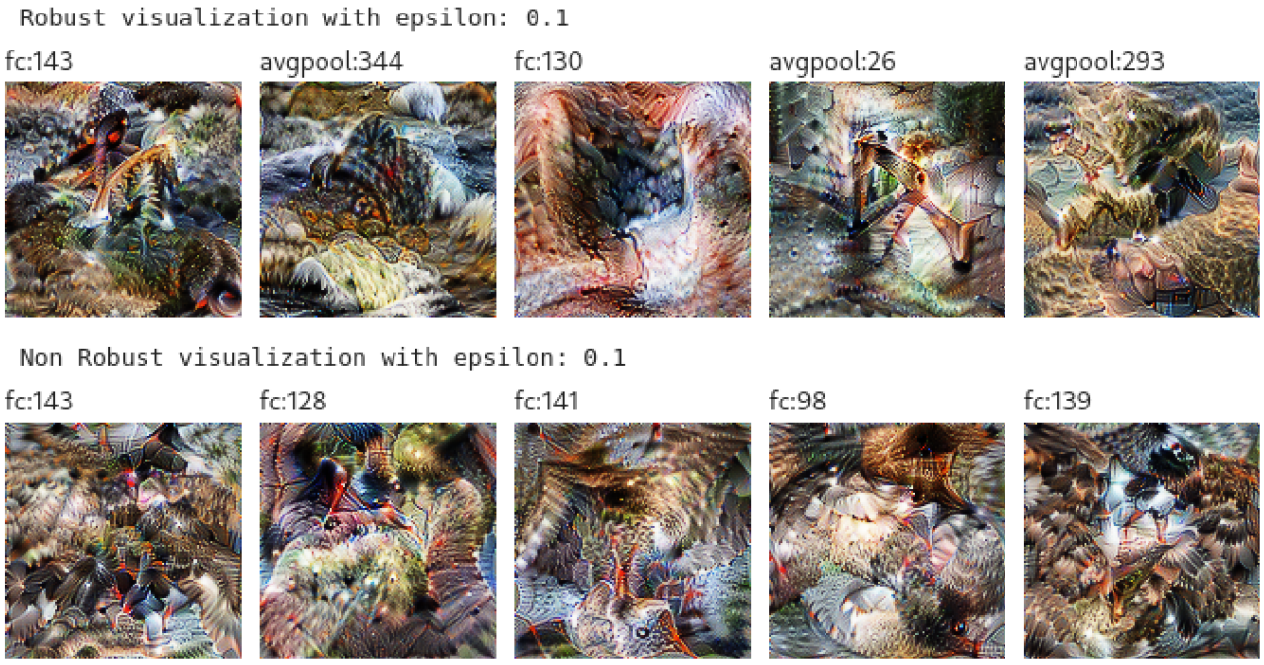

Figure 8: Sensitive neurons for the networks using flamingo images as input, $\epsilon = 0.1$

4 Conclusion

4.1 Outcomes

We managed to find, visualize and compare the sensitive features in two ResNet34 networks. In doing so, we managed to fool the models using and untargeted adversarial attack and proved that the robust network was indeed more robust to an FGSM attack.

We find our technique of finding sensitive neurons associated to image labels to be difficult to interpret. This may also be due to the relatively small amount of data availablel per class (50 images).

4.2 Future work

One limitation of our project is the interpretability of the results. Indeed, while our method shows us what neurons are most responsible for the missclassifications, the feature visualizations and the neuron sensitivities do not tell us a lot on how the networks actually missclassify adversarial examples.

Another limitation would be the size of our dataset. To be more confident in what neurons are the most sensitive, one should augment the dataset. Adding more data and diversity would help avoiding bias in what neurons get activated the most. One solution would be to first find the sensitive neurons using the whole dataset, and then finding the images that activate those neurons the most. This has the added benefit of letting the network tell us how it interprets the images instead of us having to guess.

As further improvements, we propose trying our method on different CNNs, and trying different adversarial attack methods. One could also find other ways to visualize the features and find a better way to compare the models.