Deep learning applications

Deep Learning Practical Work

Aymeric Delefosse & Charles Vin

2023 – 2024

Contents

Chapter 1

Transfer Learning

The exploration focuses on Transfer Learning, where we adapt a well-known deep learning model – VGG16 – for new applications. This process involves utilizing the VGG16 architecture, originally designed for extensive image recognition, to comprehend and perform image classification on the 15 Scene dataset. It highlights how transfer learning makes existing models adaptable to new tasks and showcases their flexibility in addressing diverse real-world image processing challenges.

1.1 VGG16 Architecture

Examining the depths of neural network structures reveals the strong capabilities of models such as VGG16. Initially created for extensive image recognition, VGG16's complex layers of convolution and pooling highlight the advancements in deep learning. In this section, we explore the architecture of VGG16, breaking down its layers and understanding how they work together to extract features and classify images. This investigation not only clarifies the model's design but also sets the foundation for its practical use in various image processing tasks.

1. ★ Knowing that the fully-connected layers account for the majority of the parameters in a model, give an estimate on the number of parameters of VGG16. There are three fully-connected layers at the end of the VGG16 architecture. Calculating their weights is relatively straightforward. We take into account the inclusion of biases.

- The first fully-connected layer receives an input of size 7 by 7 by 512 (the resulting output of the convolutional layers), which equals an input size of 25,088. Knowing that there are 4,096 neurons, this layer has a total of $(25,088+1)\times 4,096 = 102,764,544$ trainable weights.
- The second fully-connected layer receives the input from the previous layer, which is 4,096, and also consists of 4,096 neurons, resulting in $(4,096+1)\times 4,096 = 16,781,312$ trainable weights.
- Lastly, the third fully-connected layer, consisting of 1,000 neurons, has $(4,096+1)\times 1,000 = 4,097,000$ trainable weights.

Thus, the fully connected layers account for a total of 123,642,856 parameters. We can confidently state that they represent at least 85% of the model, implying there should be around **140 million parameters** to learn. If we consider a margin of 5%, there should be between 137,380,951 and 154,553,570 parameters.

We can readily confirm that the convolutional layers account for 14,714,688 parameters, meaning that there are actually 138,357,544 parameters in VGG16, meaning the fully-connected layers accounts to 89% of parameters.

2. ★ What is the output size of the last layer of VGG16? What does it correspond to? The output size of the last layer of VGG16 is 1000. It corresponds to the 1000 classes of the ImageNet dataset that the model has been trained on. Each element in this output vector represents the network's prediction scores for a specific class in the ImageNet dataset, and the class with the highest score is considered the predicted class for our given input image.

3. Bonus: Apply the network on several images of your choice and comment on the results. In Figure 1.1, we present the results of our network's application to six diverse images, encompassing different scenarios. These include two common images featuring a tuxedo cat and a Shih Tzu dog, along with two whimsical images and two images of Komondor dogs. The whimsical images consist of a close-up of a wombat and a photomontage featuring a dog. The Komondor images, in particular, provide an interesting challenge; one is clearly a dog, while the other's unique coat might humorously resemble a mop.

The network's predictions showcase its ability to recognize certain animal features, yet they also highlight its limitations in classification. It correctly identifies the wombat, cardigan dog, and Komondor without confusing the latter with a mop. However, it misclassifies the Shih Tzu as a Miniature Poodle, with a Lakeland Terrier as a close second, indicating confusion between breeds with similar appearances. The tuxedo cat is inaccurately labeled as an Egyptian Mau, emphasizing the network's difficulty in precise breed identification—a challenge partly attributed to the absence of highly specific categories within ImageNet's 1000 classes. Nevertheless, the tuxedo cat is still classified as a cat. These outcomes not only identify areas for improvement in the network's classification capabilities but also suggest the potential benefits of transfer learning. The identified errors offer valuable insights for further refining the model.

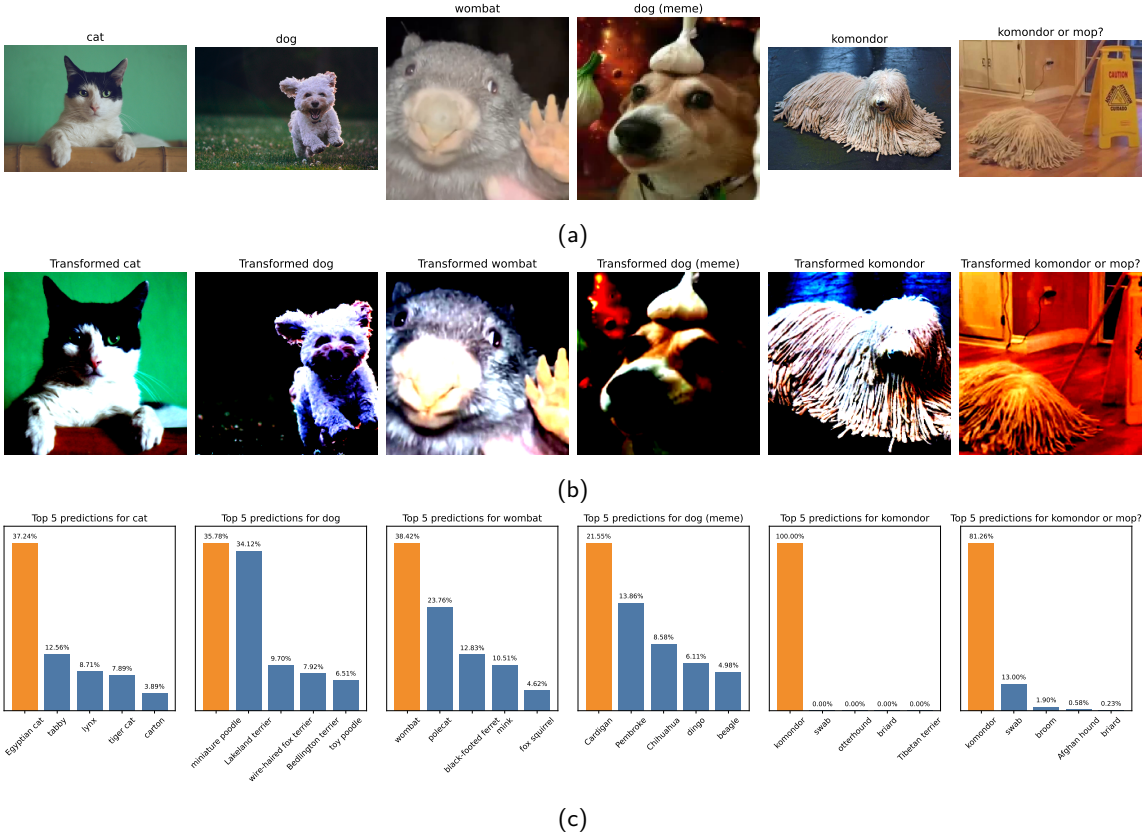


Figure 1.1: Performance evaluation of VGG16 pre-trained on ImageNet: (a) Original images, (b) Transformed images normalized and resized to 224×224, and (c) Top 5 predicted classes

4. Bonus: Visualize several activation maps obtained after the first convolutional layer. How can we interpret them? Let A_{ij} denote the activation map located at the ith row and jth column, where i and j are both in the range of $[1, 8]$.

In Figure 1.2, we showcase the 64 feature maps obtained after applying the first convolutional layer of VGG16 to the tuxedo cat image. Furthermore, we include activation maps for the five preceding images in Appendix Section 5.1.1.

Each activation map corresponds to a unique filter applied to the original image, capturing a diverse array of features. Some maps highlight edges or textures (e.g., A_{12}, A_{13}, A_{88}), while others respond to background elements (e.g., A_{26}, A_{28}), color contrasts (e.g., A_{41}, A_{62}), or specific shapes (e.g., A_{33}, A_{65}) within the image. It is essential to recognize that these initial layers are primarily designed to detect low-level features, which become progressively more abstract and complex in deeper layers.

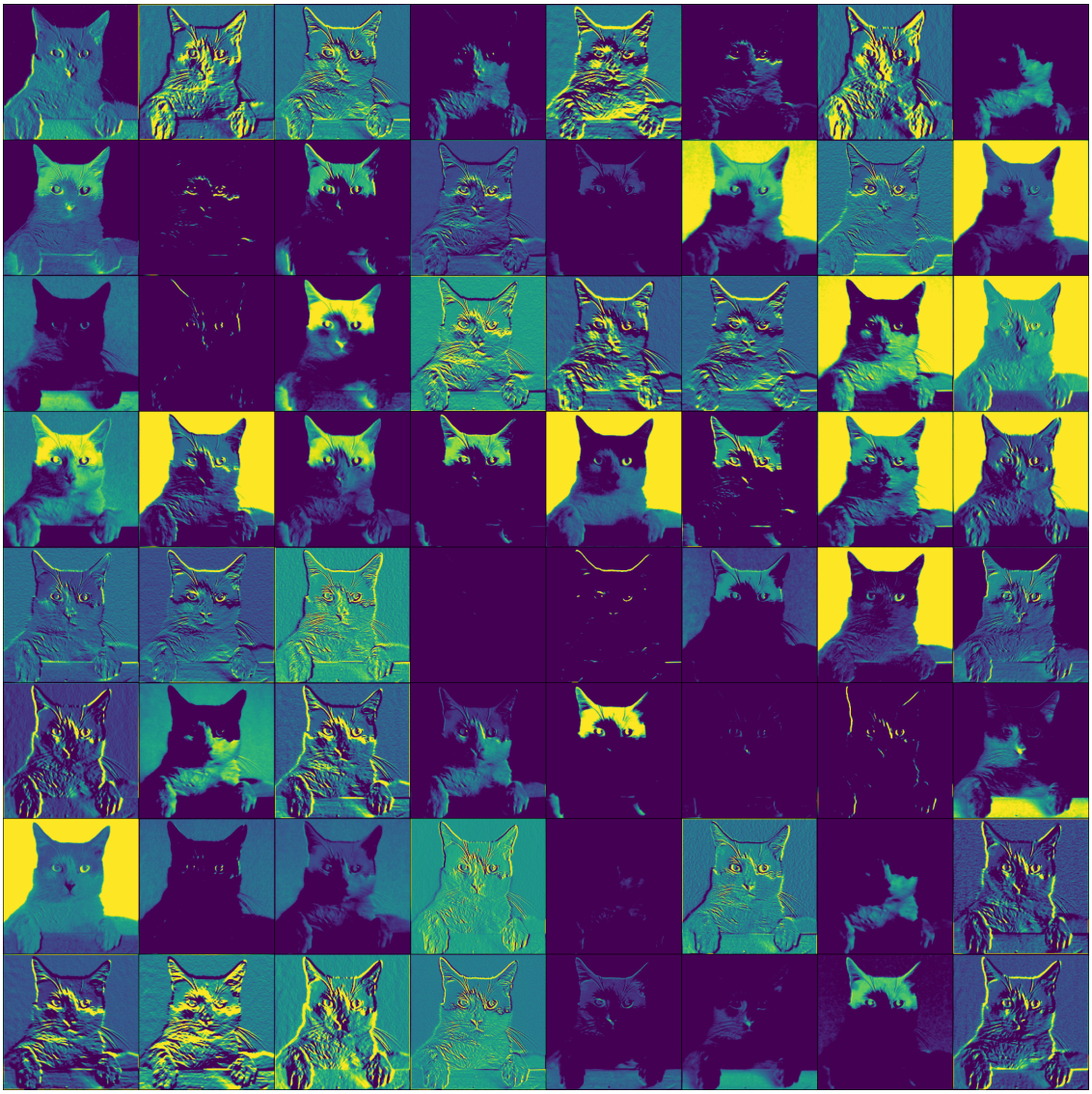


Figure 1.2: Activation maps obtained after the first convolutional layer of VGG16, on the tuxedo cat image

1.2 Transfer Learning with VGG16 on 15 Scene

The concept of transfer learning plays a significant role when it comes to applying pre-trained models to unfamiliar fields. In this section, our focus will be on how VGG16 can be cleverly repurposed for different yet related tasks. This approach highlights the adaptability of deep learning models and emphasizes the idea that existing knowledge encoded within a trained network can be successfully applied to new areas, a fundamental principle in modern machine learning research.

1.2.1 Approach

5. ★ Why not directly train VGG16 on 15 Scene? The 15 Scene dataset is quite small compared to the massive ImageNet dataset that VGG16 was originally trained on. VGG16 requires a lot of data to generalize well and to avoid overfitting. Moreover, training such a model from scratch is computationally expensive and time-consuming.

6. ★ How can pre-training on ImageNet help classification for 15 Scene? ImageNet is a vast and diverse dataset, containing millions of images distributed across numerous categories. A model pre-trained on this dataset acquires a broad spectrum of features, ranging from basic edge and texture detection to intricate patterns. These acquired features provide a robust initial foundation for extracting meaningful information from the 15 Scene images, even when the particular scenes or objects present in the 15 Scene dataset differ from those in ImageNet. This approach offers several advantages, especially considering the relatively small

size of our dataset. It helps address issues related to insufficient training data, such as overfitting. Additionally, it speeds up the training process because the model only needs fine-tuning of the previously learned features to adapt to the specific characteristics of the new dataset, eliminating the need to start the learning process from scratch.

7. What limits can you see with feature extraction? The effectiveness of transferred features depends on the similarity between the source task, for which the model was originally trained, and the target task. When the target task significantly differs from the source task, the extracted features may not be relevant or useful, and they may fail to capture the nuanced details required for achieving high accuracy. For instance, using a model trained on natural images for tasks like medical images or satellite imagery might not produce optimal results. To adapt such models to new domains, additional fine-tuning or even complete retraining with domain-specific data is often necessary, which can consume significant computational resources.

It's important to note that biases present in the pre-training dataset can influence the features extracted by the model. If the pre-training data is not representative or contains inherent biases, these biases can unintentionally affect the performance on the target task. Moreover, the utilization of models like VGG16 demands substantial computational resources, including both memory and processing power, which can present limitations, especially in resource-constrained environments.

1.2.2 Feature Extraction with VGG16

8. What is the impact of the layer at which the features are extracted? In CNNs, earlier layers typically capture fundamental features such as edges and textures, while deeper layers capture increasingly complex and high-level features that are more abstract and representative of the specific content within an image.

For some tasks, the simpler features extracted from the early layers may suffice, while for more intricate tasks (such as distinguishing between very similar categories), the deeper features can be more valuable. Early layers are generally more adaptable across various image types and tasks, whereas deeper layers tend to be more specialized and specific to the types of images and tasks the network was initially trained on.

9. The images from 15 Scene are black and white, but VGG16 requires RGB images. How can we get around this problem? Image from 15 scene are black and white so they only have one channel. VGG requires 3 channel RGB images. The easiest workaround is to replicate the single channel of the grayscale image across the three RGB channels. Another solution is to average the weights of the first convolutional layer (which is responsible for the RGB channels) so that it can directly accept grayscale images.

1.2.3 Training SVM classifiers

10. Rather than training an independent classifier, is it possible to just use the neural network? Explain. Absolutely, it is possible to use the neural network itself instead of training an independent classifier. In the case of models like VGG16, the final classification layer is essentially a neural network. If the classification task is similar to what VGG16 was originally trained for, we can fine-tune this neural network for our specific task. However, if the task is substantially different from the original one, it may not be ideal to retain the pre-trained classifier, as it might not perform well on new classes.

The decision between using the pre-trained neural network or training an independent classifier, such as an SVM, depends on the nature of the task and the available data. Using the pre-trained network can save time and computational resources, but fine-tuning or retraining may be necessary for optimal results on different tasks. Using a simpler classifier like an SVM can be a good compromise, leveraging the deep features extracted by the neural network while providing a more interpretable decision boundary.

1.2.4 Going further

11. For every improvement that you test, explain your reasoning and comment on the obtained results. TODO

In the study of transfer learning from VGG16 pretrained on ImageNet for Scene15 classification, the depth of the feature extraction layer emerged as a significant factor. Extracting features after the first fully connected layer (ReLU6) versus the second (ReLU7) offers the same dimensionality (4096 features), yet stopping after the first yields better results. Results are displayed on Table 1.1. This suggests that the initial fully connected layers may capture more generalizable features beneficial for the task at hand. Using all three fully connected layers (up to ReLU8) reduces feature dimensionality to 1000 and leads to poorer performance, likely due to over-specialization to ImageNet.

Layer Extracted	Feature Dimensionality	Accuracy (Non-Normalized)	Accuracy (Normalized)
ReLU6	4096	**0.906**	**0.912**
ReLU7	4096	0.896	0.896
ReLU8	1000	0.861	0.880

Table 1.1

Chapter 2

Visualizing Neural Networks

This exploration focuses on neural network visualization, a crucial aspect of understanding and analyzing how these models make decisions. It emphasizes the importance of interpretability and transparency in machine learning. Through the exploration of various visualization techniques, we can gain valuable insights into how these complex models interpret and process visual information. This endeavor helps bridge the divide between theoretical knowledge and real-world application, contributing to our deeper understanding of how neural networks function and their significance in modern AI solutions.

2.1 Saliency Map

This section focuses on identifying the most impactful pixels in an image for predicting its correct class. It employs a method proposed by Simonyan et al. (2014), which approximates the neural network around an image and visualizes the influence of each pixel.

1. ★ Show and interpret the obtained results. In the process of visualizing multiple saliency maps, most of them exhibited consistent patterns, while some displayed inconsistencies. Let's examine each case.

In Figure 2.1, we observe saliency maps that are generally consistent. When the model makes accurate predictions, we can observe high saliency values on the labeled object, i.e. it highlights only the important pixels. This makes it easy to recognize the images. This holds true for all the images in this figure, except for the fourth one where the model predicted "greenhouse" instead of "lakeside." In this case, the model did not focus on the lake boundaries, which led to the incorrect prediction. Instead, it primarily concentrated on the dark ground and the bright areas of the image, resulting in the "greenhouse" class prediction.

In Figure 2.2, we encounter saliency maps that appear inconsistent. Both the first and fourth images are examples where the model's prediction is correct, but the corresponding saliency maps lack informativeness and appear blurry. In contrast, for the second and last images, the model seems to be focusing on the right regions, but it still predicts the wrong class.

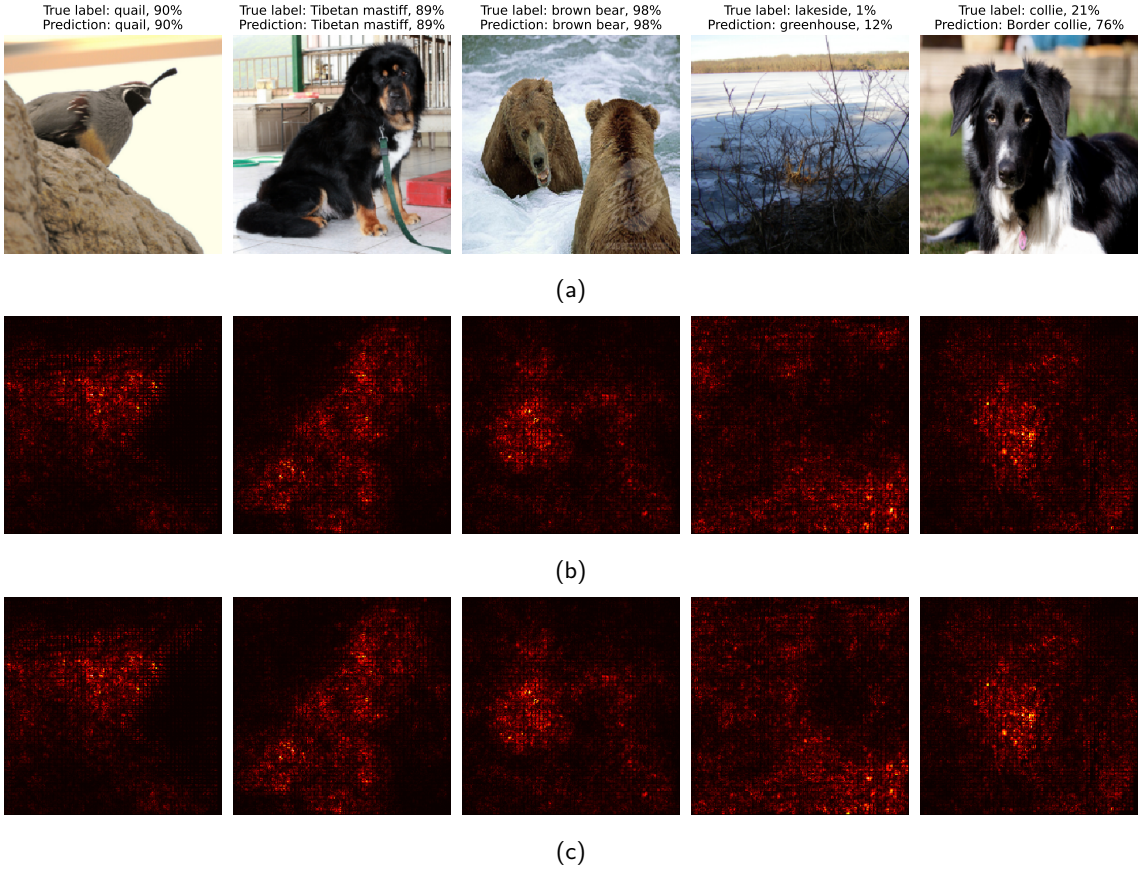


Figure 2.1: Illustration of (a) original images depicting the true label and predicted class by SqueezeNet, (b) saliency maps highlighting regions relevant to the true class, and (c) consistent saliency maps highlighting regions relevant to the predicted class.

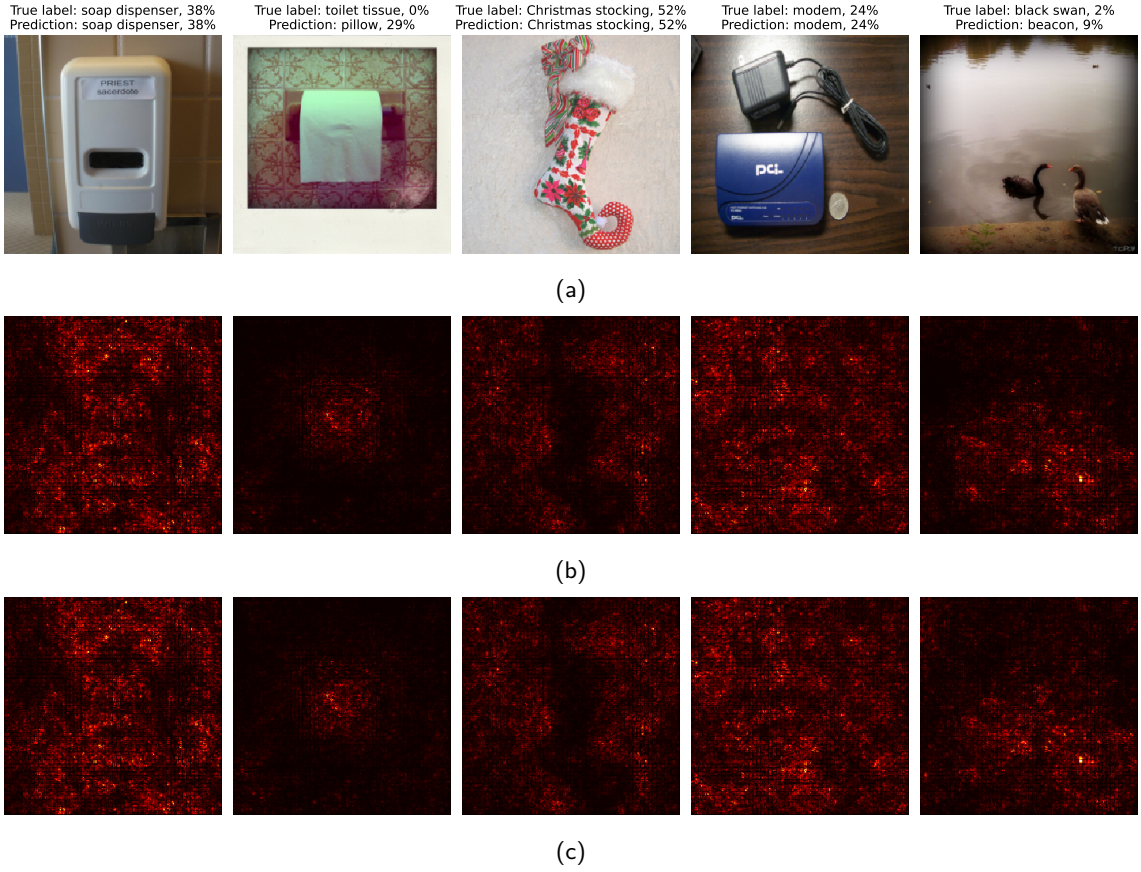


Figure 2.2: Illustration of (a) original images depicting the true label and predicted class by SqueezeNet, (b) saliency maps highlighting regions relevant to the true class, and (c) inconsistent saliency maps highlighting regions relevant to the predicted class.

2. Discuss the limits of this technique of visualization the impact of different pixels. As we discussed in the previous question, saliency maps can face challenges when dealing with images containing multiple objects or complex scenes. In such scenarios, these maps may become cluttered or unclear, making it challenging to extract the important pixels. They can also suffer from noise or, conversely, overemphasize certain features while downplaying others. This can result in a biased interpretation of the model's focus. Additionnaly, this technique (Vanilla Gradient) suffers from a saturation problem, as highlihted by Shrikumar et al. (2019). When ReLU is used, and when the activation goes below zero, it remains capped at zero and no longer change. This saturation issue may explain our previous observations regarding saliency maps. To address this problem, alternatives methods were subsequently introduced, such as SmoothGrad (Smilkov et al., 2017) and Grad-CAM (Selvaraju et al., 2019).

As with most explanation methods, interpreting saliency maps can be subjective, especially when they are ambiguous. Different individuals may draw different conclusions from the same saliency map, resulting in inconsistent interpretations of the model's behavior. This underscores the challenge of determining whether or not an explanation is correct. To gain a better understanding, we found it necessary to repeatedly analyze and visualize various saliency maps, particularly when they exhibited inconsistencies. It can be a complex task to keep in mind that each map represents the model's attention for a specific class, not the entire model. Examining saliency maps for other high-probability classes can provide additional context, contributing to a more complete understanding of the model's decision-making process.

It's important to note that saliency maps tend to highlight correlations rather than establishing causation. They reveal areas where the model directed its attention but do not necessarily imply that these areas directly influenced the model's decision.

3. Can this technique be used for a different purpose than interpreting the network? Saliency maps can be used to identify and localize important objects or regions in an image. This can be particularly useful in applications like automated image tagging or initial steps of object detection. This can also help to create more effective augmented images by applying transformations (like rotations, scaling, cropping) that preserve these key areas. This technique is primarily suited for image data. Its utility is limited in non-visual domains or for models that integrate multiple types of data (like text and images).

4. Bonus: Test with a diffrentent network, for example VGG16, and comment. We tried the same saliency maps using VGG16. Those are much better! VGG being better to classify images (no error this time in our examples) than SqueezeNet, this lead to less noise and better object frontier in saliency maps.

In our experimentation with VGG16, we observed notable improvements in the quality of the generated saliency maps compared to those produced with SqueezeNet. We think that VGG16's superior image classification capabilities contributed to reduced noise and more precise object boundaries in the saliency maps.

This outcome highlights the impact of the choice of neural network architecture on the effectiveness of saliency map generation. VGG16's strong performance in image classification results in more reliable and visually coherent saliency maps, making it a favorable choice for tasks that rely on accurate attention mapping within images.

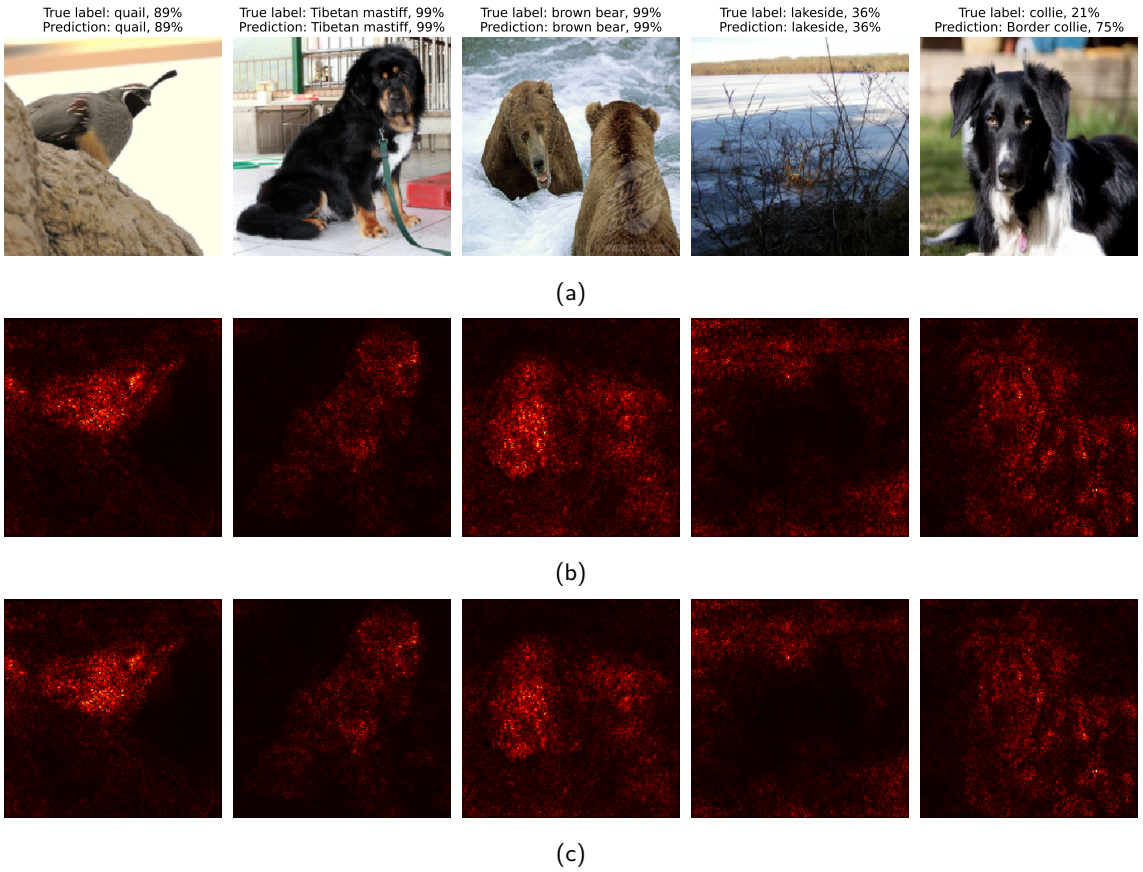


Figure 2.3: Illustration of (a) original images depicting the true label and predicted class by VGG16, (b) saliency maps highlighting regions relevant to the true class, and (c) consistent saliency maps highlighting regions relevant to the predicted class.

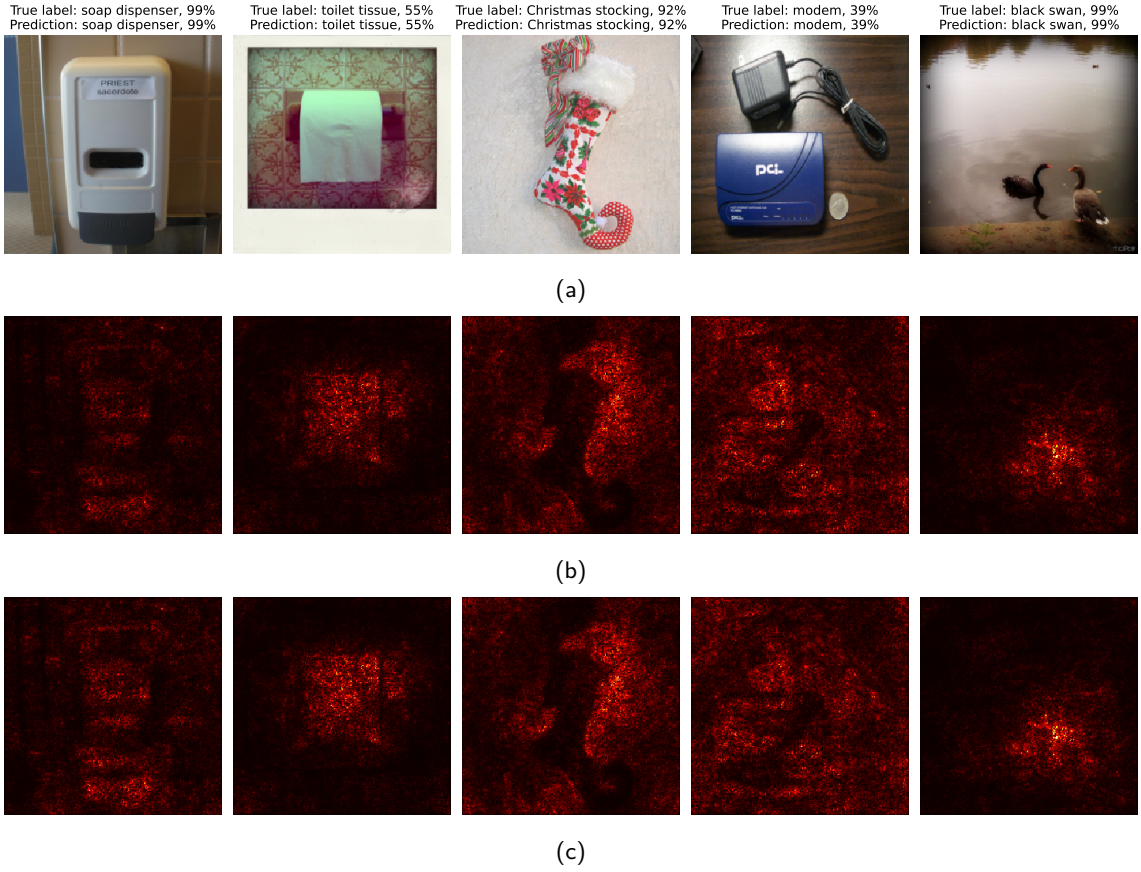


Figure 2.4: Illustration of (a) original images depicting the true label and predicted class by SqueezeNet, (b) saliency maps highlighting regions relevant to the true class, and (c) consistent saliency maps highlighting regions relevant to the predicted class.

2.2 Adversarial Example

Here, the goal is to study the vulnerability of CNNs to minor, imperceptible modifications in an image that lead to misclassification. This concept, introduced by Szegedy et al. (2014), reveals the limitations and unexpected behaviors of neural networks. Adversarial examples can be compared to counterfactual examples used in common artificial intelligence explainability methods, but their purpose is to deceive the model rather than interpret it.

5. ★ Show and interpret the obtained results. TODO

6. In practice, what consequences can this method have when using convolutional neural networks? For example, consider the ability to trick a model into misclassifying a banana as a toaster using a printable label, as demonstrated by Brown et al. (2018). While this may seem like a playful experiment, it raises significant concerns when applied to critical domains like autonomous vehicles or medical imaging.

Although our method requires access to model parameters, Athalye et al. (2018) has shown that it's possible to perform an adversarial attack without such access, known as a black-box attack. Consequently, an attacker could potentially deceive autonomous vehicles into making hazardous decisions, such as veering into another lane or perceiving non-existent information—incidents that have already occurred.

This method has the potential to be exploited by malicious individuals to intentionally mislead a model, particularly as machine learning models become increasingly integrated into systems. As a result, this topic becomes a serious concern in the realm of cybersecurity, leading to an arms race between attackers and defenders to safeguard against such vulnerabilities.

7. Bonus: Discuss the limits of this naive way to construct adversarial images. Can you propose some alternative or modified ways? (You can base these on recent research) One major limit of this naive way is that it requires many pixels to be changed. This led to the question: is it possible to deceive a neural network by modifying only one pixel? Su et al. (2019) demonstrated that it is actually possible.

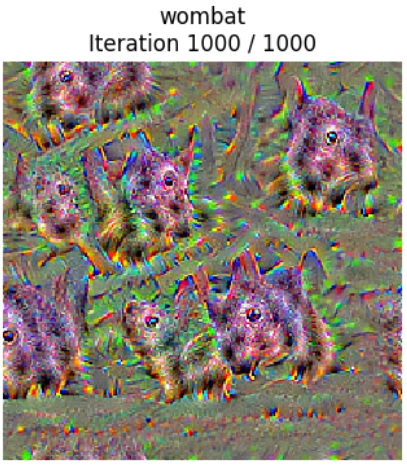
This method was inspired by biological evolution studies, and uses differential evolution to determine which to modify and how. It starts with a population of candidate solutions, each representing a potential pixel modification encoded by a five-element vector (coordinates and RGB values). The process then generates new generations of candidates (children) from the existing ones (parents) using a specific formula that involves mixing attributes from three random parent pixels. The process aims to find an adversarial example that causes the classifier to misidentify the image. It continues until such an example is found or until it reaches a user-specified iteration limit.

2.3 Class Visualization

This section aims to generate images that highlight the type of patterns detected by a network for a particular class, based on techniques developed by Simonyan et al. (2014) and Yosinski et al. (2015). This method helps in visualizing what features the network prioritizes for classification. To view the animated visuals in this section, you can use Adobe Acrobat Reader or access them from a separate folder provided with the materials.

8. ★ Show and interpret the obtained results. Class visualization is a technique aimed at understanding what features or patterns a CNN has learned to recognize for a specific class. It involves generating an input image that maximally activates a class score in the network. Here, we've done this by iteratively modifying an initial random image to increase the activation of the desired class through gradient ascent on the class score with respect to the input image.

As a first try of this method, we decided to visualize our prefered animal: a wombat. Figure 2.5 present a class visualization for a wombat after 1000 epochs and using the default regularization parameters in the given implementation (regularization factor of 10^{-3}, bluring every ten epoch, learning rate of 5).



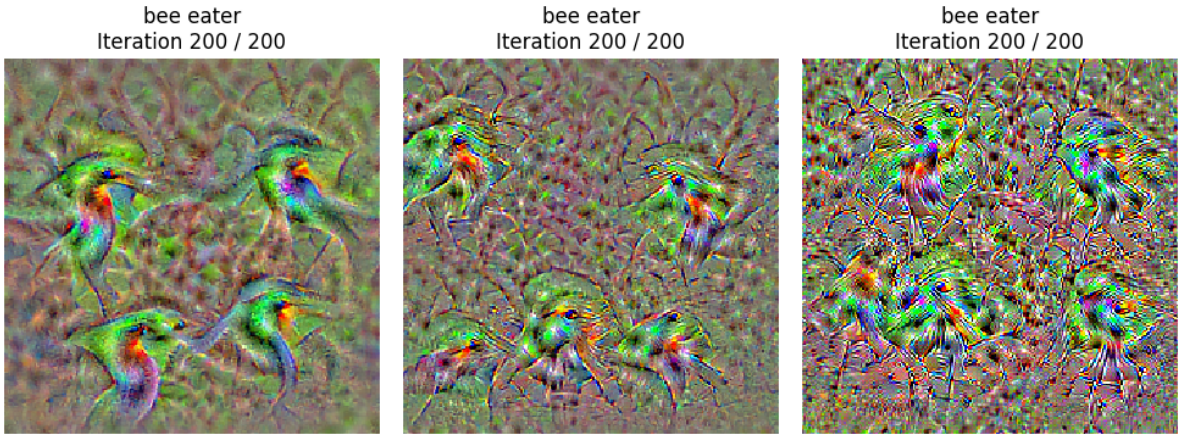
(a) Last iteration (b) Animation

Figure 2.5: Class visualization: started from random noise, maximising the score from the wombat class. Using default regularization parameters (regularization factor of 10^{-3}, bluring every ten epoch, learning rate of 5).

9. Try to vary the number of iterations and the learning rate as well as the regularization weight. While examining some of our animations, we observed the occurrence of blurring every 10 steps.

To improve class visualization, it appears that regularization plays a crucial role. In Figure 2.6, we experimented with disabling image blurring and weight regularization on SqueezeNet.

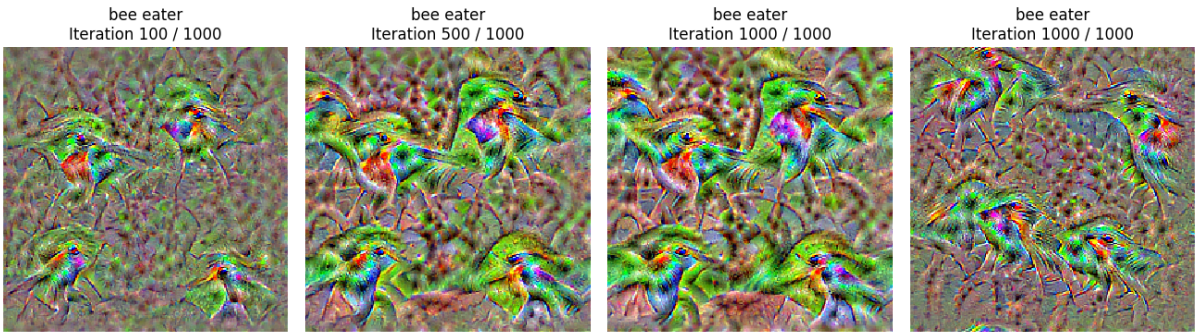
Non-regularized images appear to be overly saturated, making it challenging to discern the represented class clearly. Our suspicion is that without regularization, gradients become saturated in **every pixel** that could have contributed to a correct prediction of the "bee eater" class. By encouraging pixels that the model has already engaged in transformations and maintaining their direction, regularization facilitates the emergence of the true class image.



(a) (b) (c)

Figure 2.6: Comparaison of the bee eater class visualization using SqueezeNet (a) with strong regularizations, (b) with normal regularizations and (c) without regularizations.

About the number of epoch, our experiment show that we need to make a certain number of epoch to let the visualization converge, thus getting better images.

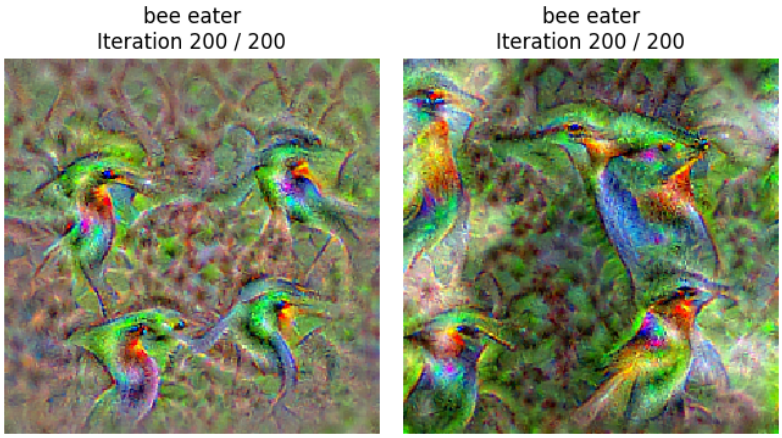


(a) (b) (c) (d)

Figure 2.7: Comparaison of the bee eater class visualization using SqueezeNet with strong regularizations (a) 200 iterations, (b) 500 iterations, (c) 1000 iterations and (d) at 1000 iterations but without having performed regularization.

About the learning rate, TODO

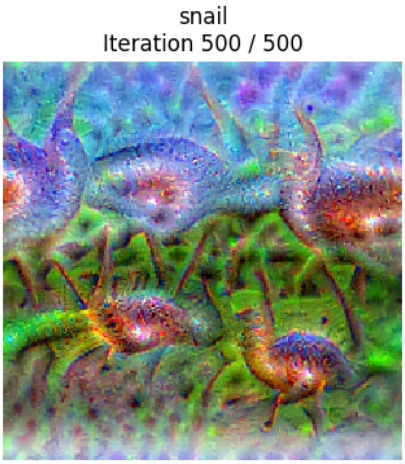
10. Try to use an image from ImageNet as the source image instead of a random image. You can use the real class as the target class. Comment on the interest of doing this. Initiating the process with an image from ImageNet is a valuable approach. It provides a meaningful starting point for gradient-based visualization, ensuring that the initial gradients are directed towards relevant features and patterns within the image. It prevents the initial gradients from scattering or diverging in various directions, potentially resulting in a more focused and interpretable visualization.



(a) (b) (c) Animation starting from same class.

Figure 2.8: Class visualization using SqueezeNet: (a) Last iteration starting from noise, (b) Last iteration starting from the same class, and (c) Animation starting from the same class to maximize the score for the bee eater class, all with strong regularization.

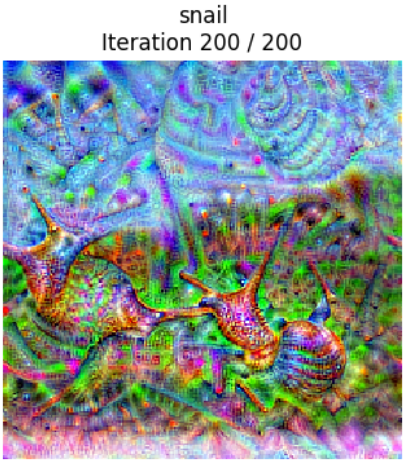
It pretty funny to create some objects from other objects. In Figure 2.9 we started from a image of hays to do a snail class visualization.



(a) Last iteration (b) Animation

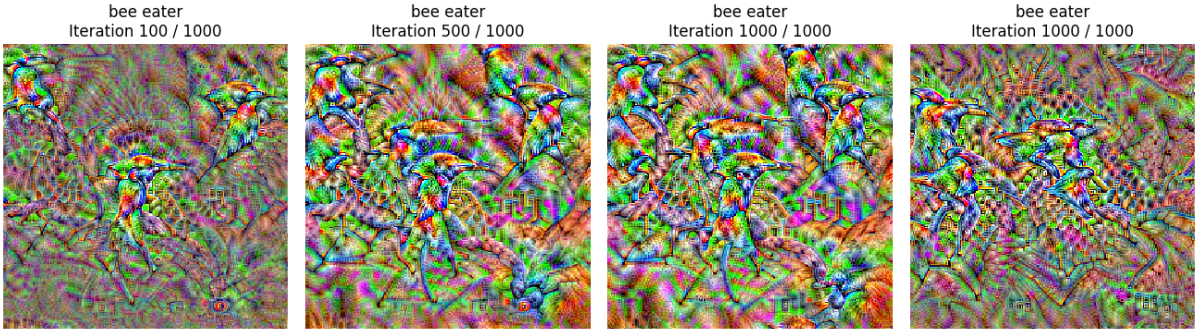
Figure 2.9: Class visualization using SqueezeNet with strong regularizations: started from a hay image to maximising the score for the snail class.

11. Bonus: Test with another network, VGG16, for example, and comment on the results. We conducted identical experiments using VGG16 this time. Visualizations are often superior due to VGG16's overall improved performance, as previously discussed in Section 2.1. Specifically, we found the hay to snail animation in Figure 2.10b to be more appealing. However, there are instances where regularization proves to be more challenging, as observed in Figure 2.11 or 2.12.



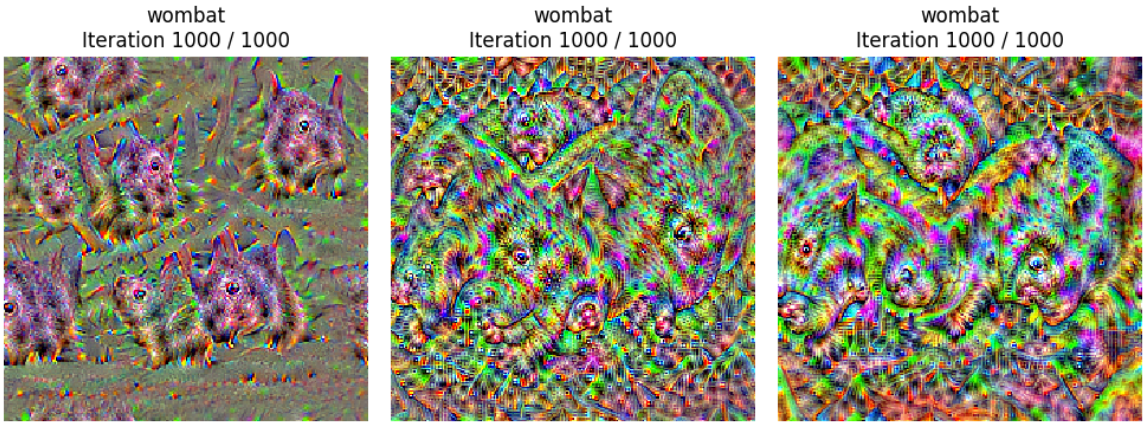
(a) Last iteration (b) Animation

Figure 2.10: Class visualization using VGG with strong regularizations: started from a hay image to maximising the score for the snail class.



(a) (b) (c) (d)

Figure 2.11: Comparaison of the bee eater class visualization using VGG with strong regularizations (a) 200 iterations, (b) 500 iterations, (c) 1000 iterations and (d) at 1000 iterations but without having performed regularization.



(a) (b) (c)

Figure 2.12: Class visualization of Wombat using VGG TODO

Chapter 3

Domain Adaptation

This exploration focuses on domain adaptation to tackle the task of applying models trained in one domain to a distinct yet related domain. This entails the comprehension and application of concepts like the DANN model and the Gradient Reversal Layer, which serve as tools to render a model agnostic to the domain. This practical exercise underscores the complexities of training a model on one dataset, such as labeled MNIST, and subsequently utilizing it effectively on a different dataset, such as unlabeled MNIST-M. This mirrors real-world situations where domain adaptation plays a crucial role, e.g. autonomous driving.

1. If you keep the network with the three parts (green, blue, pink) but didn't use the GRL, what would happen? Without the GRL, the domain classifier would become proficient at distinguishing between source and target domains, counteracting the aim of domain adaptation. This would lead to a more domain-specific model rather than a domain-generalized one.

2. Why does the performance on the source dataset may degrade a bit? The minor performance decrease on the source dataset result from the model adapting to features common to both domains, slightly reducing its specificity for the source domain.

3. Discuss the influence of the value of the negative number used to reverse the gradient in the GRL. The gradient reversal value balances learning domain-specific features and generalizing across domains. An optimal value is crucial for effective learning without compromising performance on either domain.

4. Another common method in domain adaptation is pseudo-labeling. Investigate what it is and describe it in your own words. Pseudo-labeling involves generating labels for the target domain using the model's predictions. These labels are then used for further training, helping the model adapt to the target domain by leveraging its existing knowledge and narrowing the domain gap.

Chapter 4

Generative Adversarial Networks

4.1 Generative Adversarial Networks

4.1.1 General principles

$$\min_G \max_D \mathbb{E}_{x^* \sim \text{Data}} [\log D(x^*)] + \mathbb{E}_{z \sim P(z)} [\log(1 - D(G(z)))] \tag{4.1}$$

$$\max_G \mathbb{E}_{z \sim P(z)} [\log D(G(z))] \tag{4.2}$$

$$\max_D \mathbb{E}_{x^* \sim \text{Data}} [\log D(x^*)] + \mathbb{E}_{z \sim P(z)} [\log(1 - D(G(z)))] \tag{4.3}$$

1. Interpret the equations 4.2 and 4.3. What would happen if we only used one of the two?

2. Ideally, what should the generator G transform the distribution $P(z)$ to?

3. Remark that the equation 4.2 is not directly derived from the equation 4.1. This is justified by the authors to obtain more stable training and avoid the saturation of gradients. What should the "true" equation be here?

4.1.2 Architecture of the networks

4. Comment on the training of of the GAN with the default settings (progress of the generations, the loss, stability, image diversity, etc.)

5. Comment on the diverse experiences that you have performed with the suggestions above. In particular, comment on the stability on training, the losses, the diversity of generated images, etc.

4.2 Conditional Generative Adversarial Networks

4.2.1 cDCGAN Architectures for MNIST

6. Comment on your experiences with the conditional DCGAN.

7. Could we remove the vector y from the input of the discriminator (so having $cD(x)$ instead of $cD(x, y)$)?

8. Was your training more or less successful than the unconditional case? Why?

9. Test the code at the end. Each column corresponds to a unique noise vector z. What could z be interpreted as here?

Chapter 5

Appendix

5.1 Transfer Learning

5.1.1 Activation maps

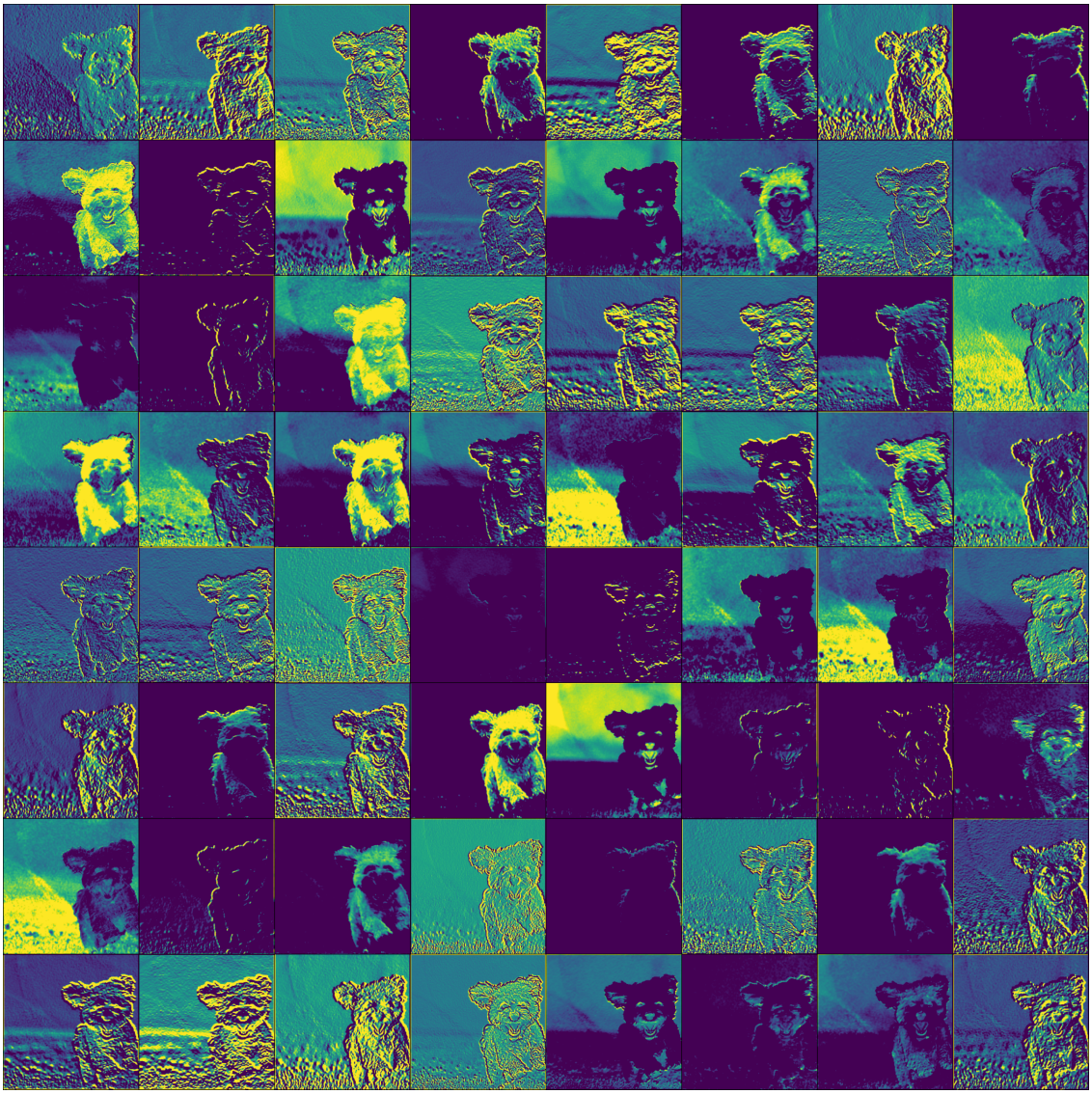


Figure 5.1: Activation maps obtanied after the first convolutional layer, on the Shih Tzu dog image

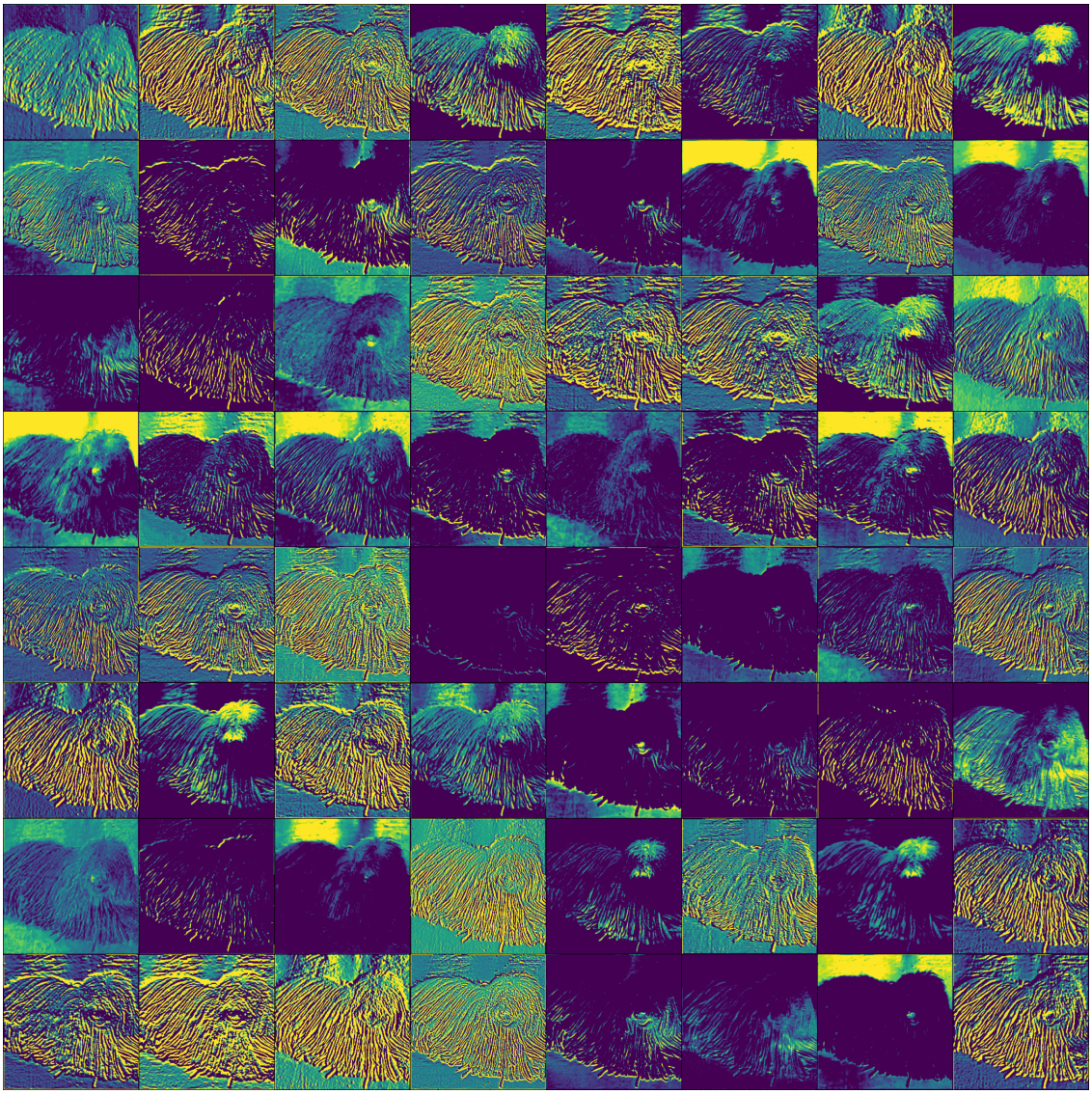


Figure 5.2: Activation maps obtanied after the first convolutional layer, on the Komondor dog image

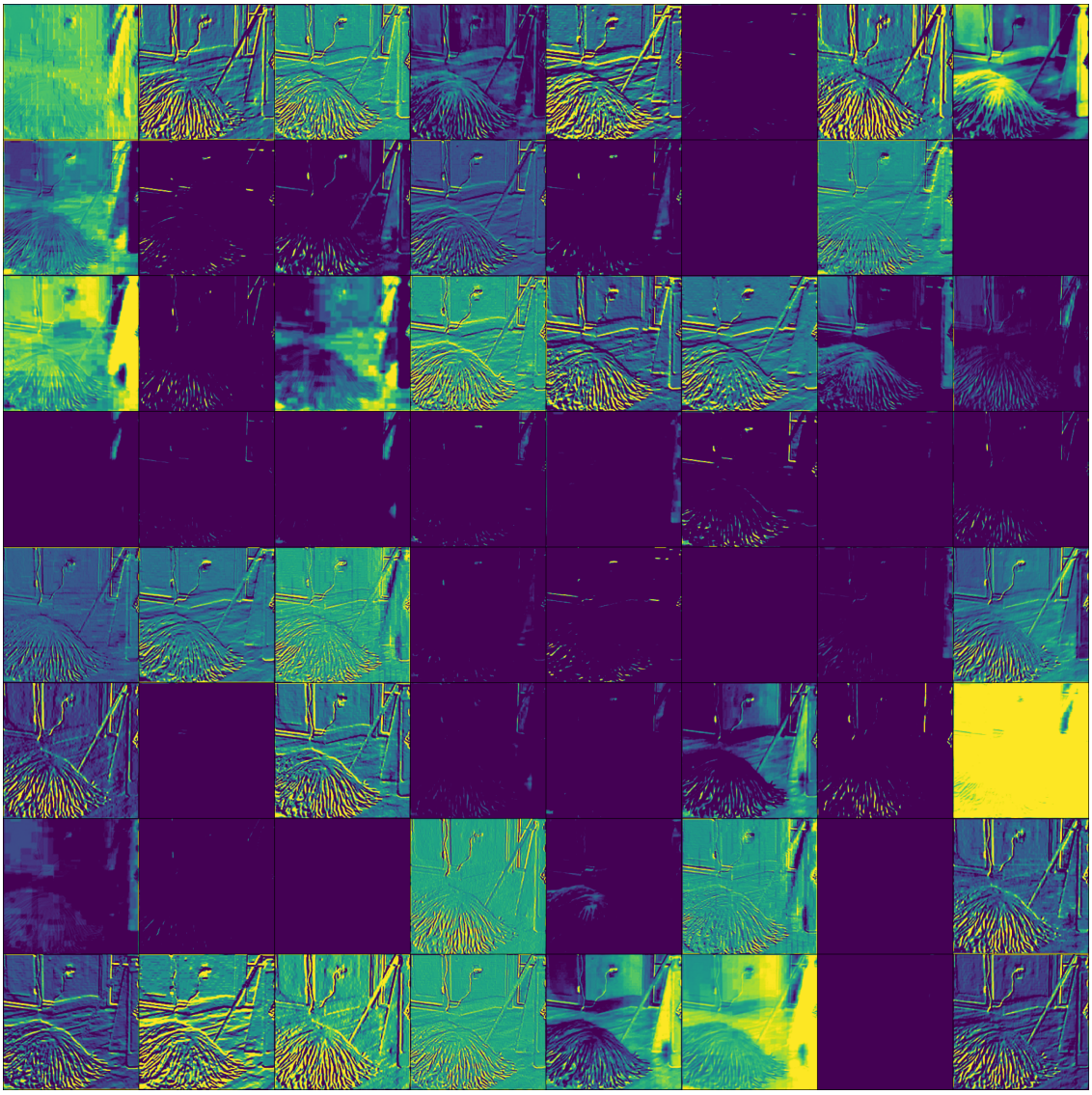


Figure 5.3: Activation maps obtanied after the first convolutional layer, on the "Komondor or mop?" image

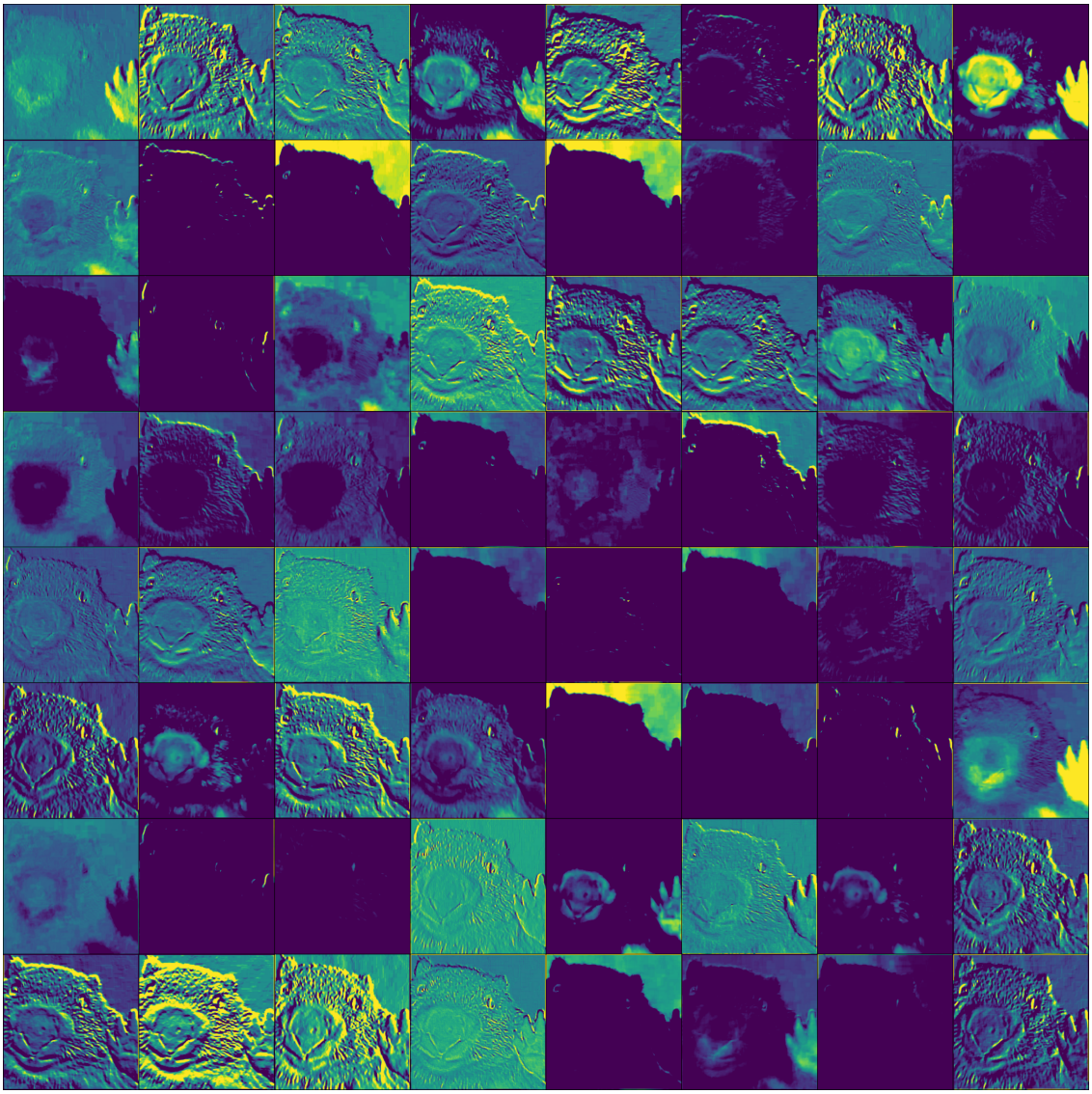


Figure 5.4: Activation maps obtanied after the first convolutional layer, on the wombat image

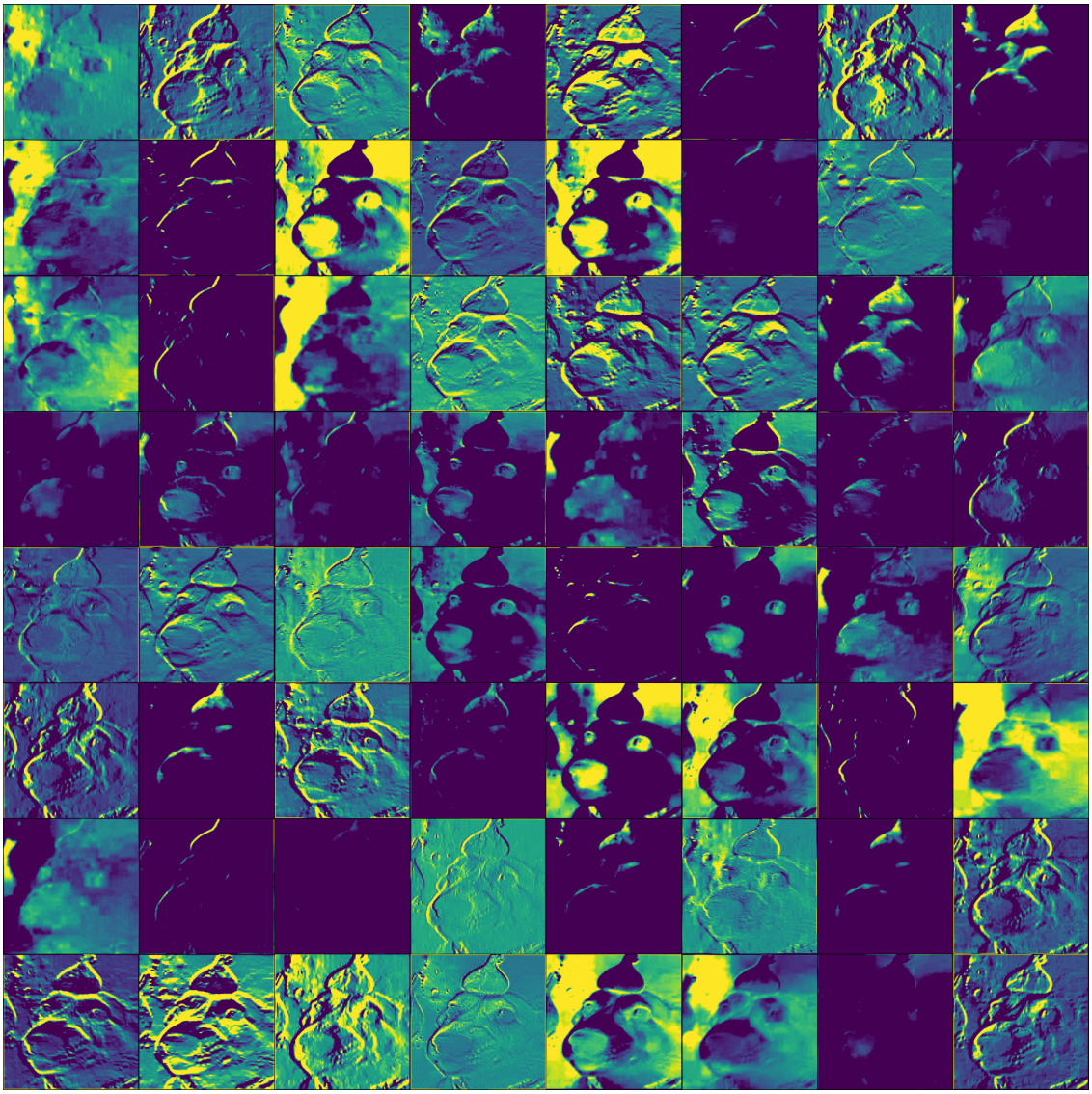


Figure 5.5: Activation maps obtanied after the first convolutional layer, on the dog (meme) image

Bibliography

Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples, 2018.

Tom B. Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. Adversarial patch, 2018.

Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, 128(2):336–359, October 2019. ISSN 1573-1405. doi: 10.1007/s11263-019-01228-7. URL http://dx.doi.org/10.1007/s11263-019-01228-7.

Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences, 2019.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps, 2014.

Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise, 2017.

Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841, October 2019. ISSN 1941-0026. doi: 10.1109/tevc.2019.2890858. URL http://dx.doi.org/10.1109/TEVC.2019.2890858.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, 2014.

Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization, 2015.