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.

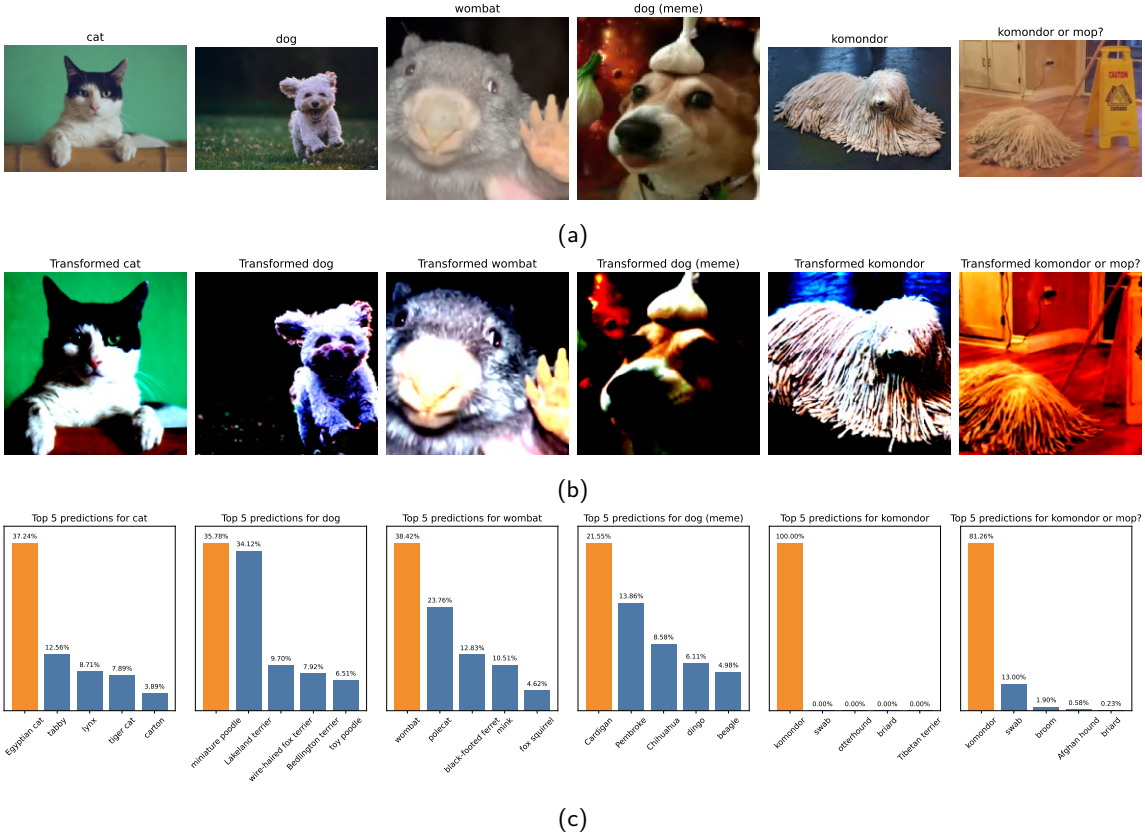


Figure 1.1: Performance evaluation of VGG16 pre-trained on ImageNet: (a) Original images, (b) Transformed images normalized and resized to 224×224, and (c) Top 5 predicted classes.

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 important to note that these initial layers are primarily designed to detect low-level features, which become progressively more abstract and complex in deeper layers. Additionally, these maps consistently serve specific purposes regardless of the input image, indicating a dedicated function for each activation map. Thus, our interpretation stays true for the images in Appendix Section 5.1.1.

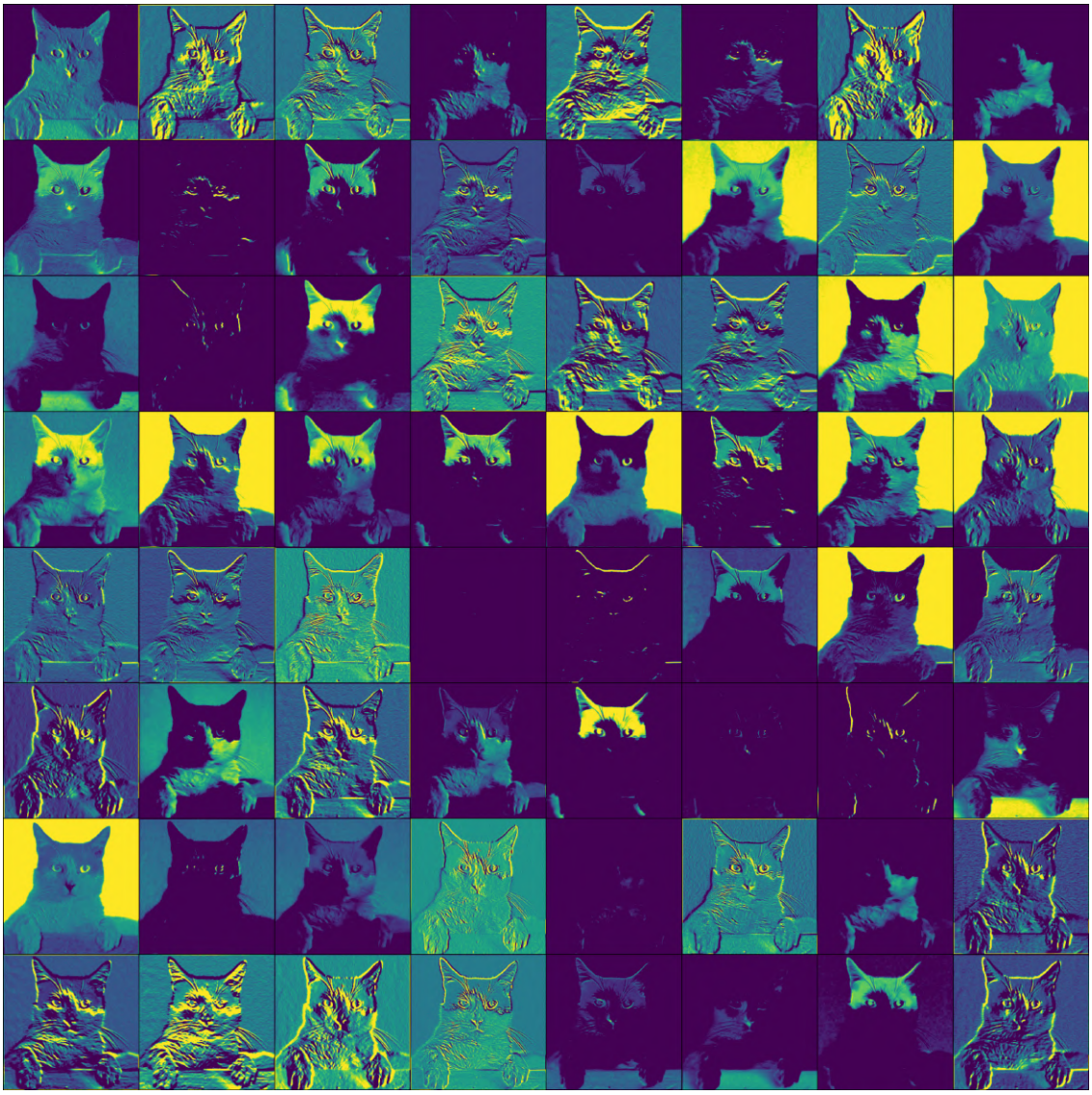


Figure 1.2: Activation maps obtained after the first convolutional layer of VGG16, on the tuxedo cat image.

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? The distinction in feature extraction primarily depends on the classifier or fully connected layers that we choose to retain at the end of the network. This process of feature extraction allows us to preserve and harness the features captured by the CNN layers. Consequently, modifying or replacing these layers for a specific task leads to a shift in the nature of the extracted features. The impact of the extraction layer is closely related to the number of parameters retained and the level of abstraction and complexity (or dimensionality) within the feature space, which is customized by the architecture of the classifier. For example, in cases where multiple connected layers are present, the first connected layer typically contains the largest number of parameters. In our case with VGG16, the first layer alone accounts for more than 83% of the parameters in the final classifier. As a concluding note, 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. For example, if we aim to leverage VGG16's classifier for a task like Scene15, we would need to adapt the neural network by adding a final layer that produces an output of 15, but further training and fine-tuning will be necessary. Alternatively, we could opt for an independent neural network tailored for this specific task.

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 will 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. 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 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: Performance comparison based on the depth of feature extraction in VGG16, indicating accuracy for features extracted at ReLU6, ReLU7, and ReLU8 using an SVM with $C = 1$, and whether or not the data were normalized.

Additionally, the choice of feature extraction layer, whether it be the first fully-connected layer, second, or third, did not have a discernible impact on the time required for feature extraction or the time taken to train a SVM. We conducted a grid search over 5-fold cross-validation to fine-tune the SVM parameters and present the results in Table 1.2. The variation in the optimal C values across different layers (ReLU6, ReLU7, ReLU8) sheds light on key aspects of feature complexity and generalizability. Higher C values suggest the need for stronger regularization, possibly due to the features being more intricate and specialized towards the ImageNet dataset, hence less adaptable to the Scene15 dataset.

Layer Extracted	Best C value	Cross-validation score	Accuracy
ReLU6	10	**0.920**	**0.910**
ReLU7	100	0.905	0.899
ReLU8	5	0.894	0.888

Table 1.2: Tuning results for SVM using features extracted from different layers of VGG16, highlighting the best C value, cross-validation score, and accuracy, using normalized data.

We also explored the effectiveness of employing a custom Multi-Layer Perceptron (MLP) as an alternative to using a SVM for classification. The architecture of our MLP is outlined in Table 1.3. The motivation behind this approach was to examine whether a neural network, capable of end-to-end learning, could outperform the conventional method of SVM classification when working with features extracted from VGG16. This experiment was carried out in two variations: one where the MLP served as a standalone classifier for the extracted features, and another where the MLP was integrated into a modified VGG16, functioning as the final layers, with the flexibility of either freezing or fine-tuning the earlier layers.

The experiment yielded two key outcomes: the standalone MLP achieved an accuracy of 0.8958, while the MLP integrated into VGG16 achieved an accuracy of 0.8938, both after 20 epochs. These results, though slightly lower than those obtained with an SVM, suggest that the more complex end-to-end learning approach of a neural network does not necessarily lead to superior performance compared to using a simpler model like SVM on pre-extracted features. This observation reinforces the notion that in certain transfer learning scenarios, particularly when the extracted features are already robust, simpler models can be equally, if not more, effective than their more complex counterparts, while being much more computationally efficient.

Layer Type	Output Size	Parameters
Linear	1024	$4096 \times 1024 + 1024 = 4{,}195{,}328$
ReLU	-	-
Dropout	-	rate $= 0.2$
Linear	512	$1024 \times 512 + 512 = 524{,}800$
ReLU	-	-
Dropout	-	rate $= 0.2$
Linear	15	$512 \times 15 + 15 = 7695$
Total parameters		4,727,823

Table 1.3: Architecture of the Multi-Layer Perceptron (MLP) used in the VGG16 modification.

Lastly, we explored alternative architectures for transfer learning, extending our considerations beyond VGG16 to leverage the advancements made in pre-trained networks since 2014. Our focus shifted to attention-based models, particularly Vision Transformers, renowned for their superior performance on ImageNet. In our experiments, we chose the `vit_b_16` model, celebrated for its efficient parameter utilization (with 37.43% fewer parameters compared to VGG16) and improved accuracy (a 9.48% boost in top-1 accuracy compared to VGG16). In our initial analysis, we evaluated the performance of this model using the first six images and observed a noticeable increase in the model's assertiveness in its predictions. For instance, the model exhibited significantly higher confidence in correctly identifying a wombat as a wombat, with a confidence level of 97.48% for the fine-tuned weights compared to 38.42% in VGG16. Results are available in Appendix Section 5.1.2, presented in Figure 5.6 and Figure 5.7.

Feature extraction occurred after the encoder, specifically at the linear layer of the classifier. Notably, we evaluated weights refined via end-to-end fine-tuning (IMAGENET1K_SWAG_E2E_V1) against those trained from scratch (IMAGENET1K_V1). Our findings are presented in Table 1.4 and Table 1.5. It's worth noting that tuning the SVM with Vision Transformer features led to superior results when compared to the tuned SVM trained using VGG16 features, resulting in a significant 2.1% increase in accuracy. As expected, the utilization of IMAGENET1K_SWAG_E2E_V1 weights naturally provided the best performance, owing to their superior performance on ImageNet-1K. Additionally, it's important to mention that the feature extraction process for Vision Transformers did indeed take a bit longer. Specifically, Vision Transformers processed at a rate ranging from 6 to 12 batches per second, while VGG ranged from 15 to 20 batches per second.

Weights	**Accuracy (Non-Normalized)**	**Accuracy (Normalized)**
IMAGENET1K_V1	0.911	0.918
IMAGENET1K_SWAG_E2E_V1	**0.917**	**0.925**

Table 1.4: Performance comparison based on the weights used for Vision Transformer, using an SVM with $C = 1$, and whether or not the data were normalized.

Weights	**Best C value**	**Cross-validation score**	**Accuracy (Normalized)**
IMAGENET1K_V1	1	0.921	0.918
IMAGENET1K_SWAG_E2E_V1	5	**0.944**	**0.931**

Table 1.5: Tuning results for SVM using different sets of weights for Vision Transformer, highlighting the best C value, cross-validation score, and accuracy, using normalized data.

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. This practical was our favourite yet.

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. It appears that the third image of a Christmas sock may be inverted. We also observe a similar effect when using VGG16, as we will explore later in our experiments.

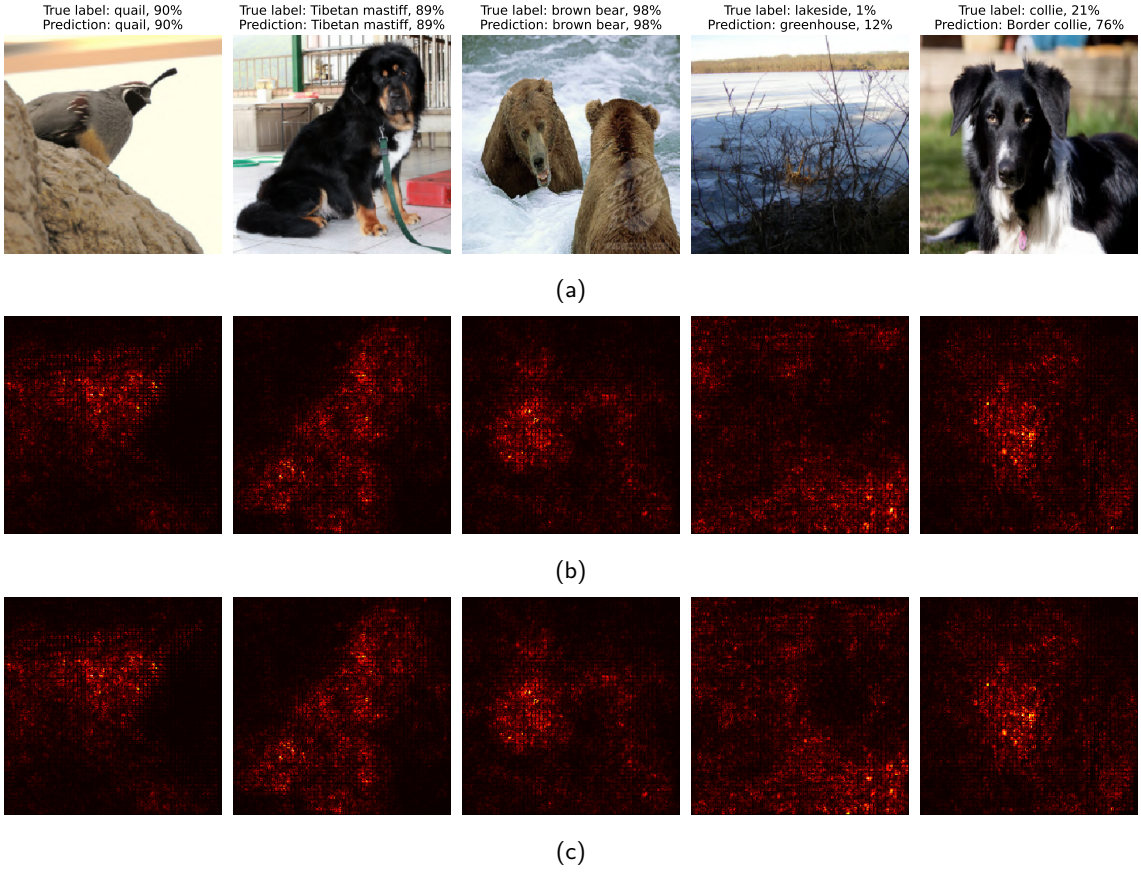


Figure 2.1: Illustration of (a) original images depicting the true label and predicted class by SqueezeNet, (b) saliency maps highlighting regions relevant to the true class, and (c) consistent saliency maps highlighting regions relevant to the predicted class.

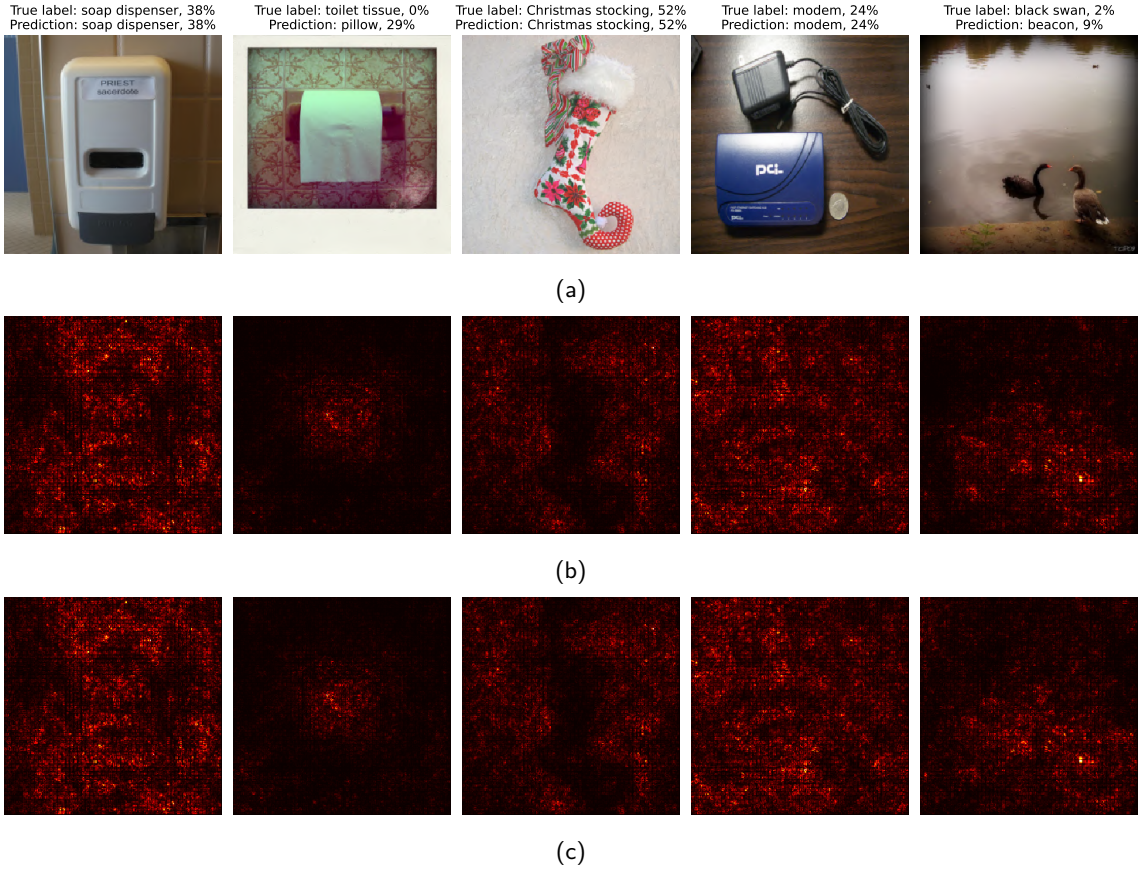


Figure 2.2: Illustration of (a) original images depicting the true label and predicted class by SqueezeNet, (b) saliency maps highlighting regions relevant to the true class, and (c) inconsistent saliency maps highlighting regions relevant to the predicted class.

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. They can find applications in image segmentation, for instance 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 generated saliency maps using VGG16 on the same examples. Notably, we observed that there were no misclassifications due to VGG16's superior image classification capabilities, as seen in examples like Lakeside and Black Swan. Consequently, the saliency maps generated by VGG16 exhibited significant improvements. They displayed reduced noise in examples like Quail and featured more precise object boundaries, as seen with the Tibetan Mastiff. Saliency maps for objects like the soap dispenser, toilet tissue, and model also showed improvements. However, the Christmas sock still exhibited an unusual inversion in the saliency map.

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.

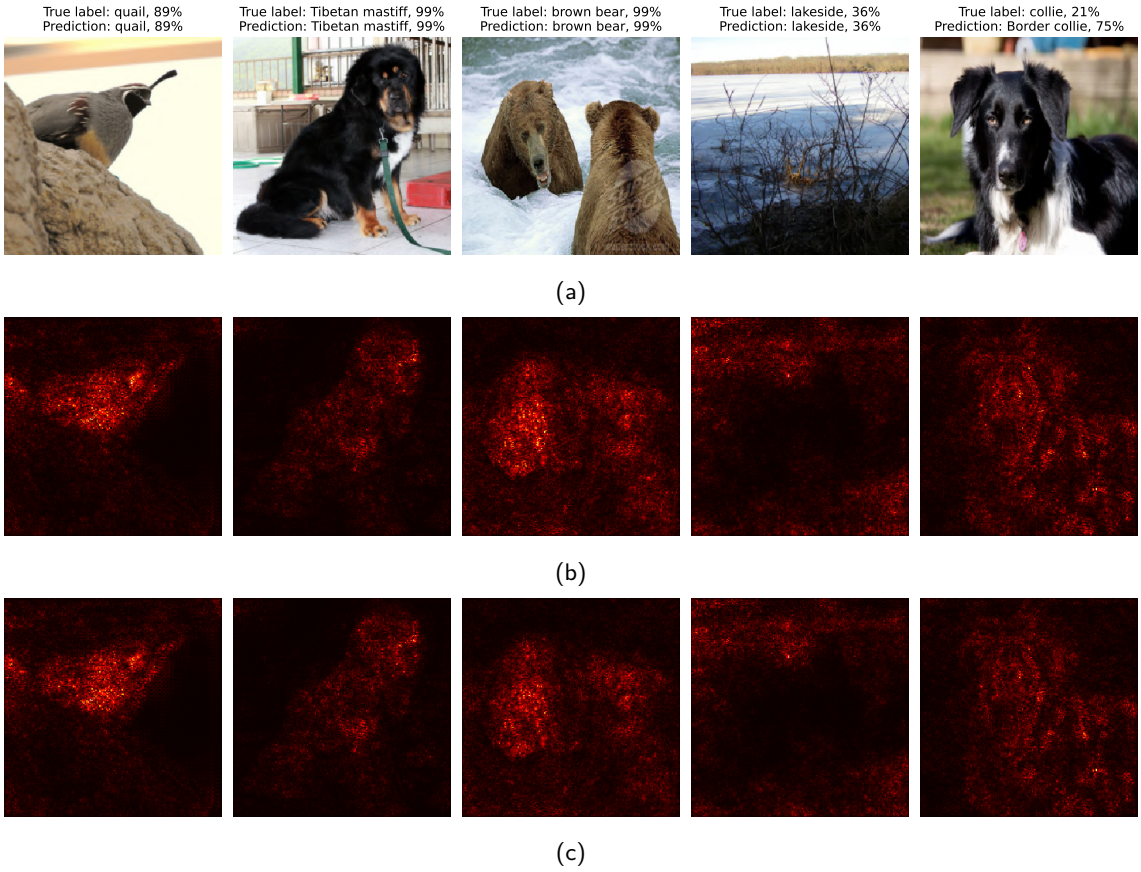


Figure 2.3: Illustration of (a) original images depicting the true label and predicted class by VGG16, (b) saliency maps highlighting regions relevant to the true class, and (c) consistent saliency maps highlighting regions relevant to the predicted class.

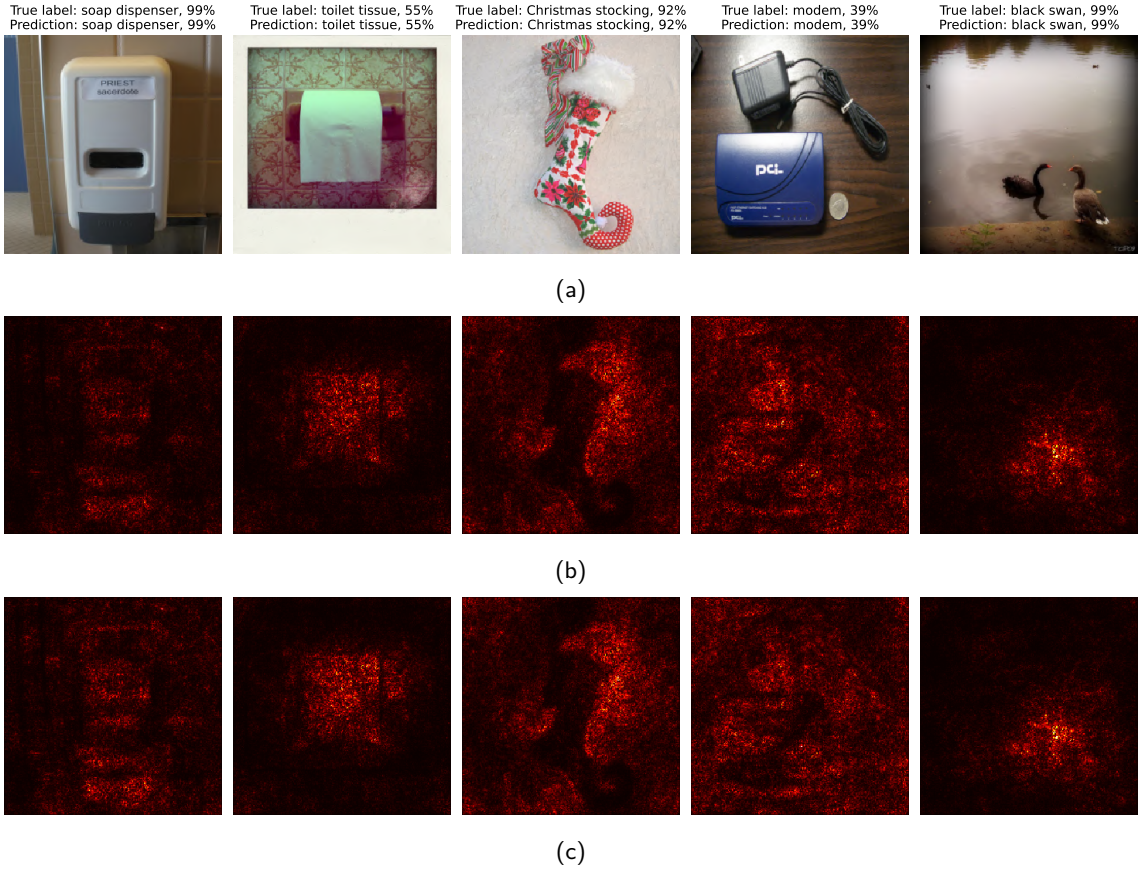


Figure 2.4: Illustration of (a) original images depicting the true label and predicted class by VGG16, (b) saliency maps highlighting regions relevant to the true class, and (c) consistent saliency maps highlighting regions relevant to the predicted class.

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. In the part of the pratical, we created adversarial examples. We apply imperceptible, slight and carefully calculated modifications to the input image such that it is misclassified by a neural network. This is done by applying gradient backpropagation to find out how to tweak the pixel values of the original image so as to change the network's prediction. The adjustments are made according to the gradient of the loss with respect to the input image, and the process iteratively continues until the image is misclassified.

Our results are presented in Figure 2.5. The right column present original image, before any intervention, the left column the modified misclassified image, and in the middle column, the magnified difference (10x) between them. Despite the visual similarity of the two images on either side, the neural network classifies them as two different breeds due to the small, calculated changes that showed in the center image. Without magnification, these changes are imperceptible to our human eyes, underscoring the minimal nature of the alterations and yet how significantly they can impact a neural network's classification.

Furthermore, it's worth noting that in many cases, adversarial examples tend to focus on the main object that determines the image's classification, as evident in Figures 2.5a, 2.5c, and 2.5e. This aligns with the concept of saliency maps. However, it's important to acknowledge that this isn't always the case. In some instances, successful adversarial examples may require alterations to the background to effectively deceive the model, as demonstrated in Figures 2.5b, 2.5d, and 2.5f. Notably, to deceive the pirate ship as a wombat, we only needed to focus on the ship itself. But to trick the pirate ship into classifying as a fireboat, it was necessary to fool both the boat and the background. This illustrates the nuanced nature of crafting adversarial examples and the need to consider various factors to achieve misclassification.

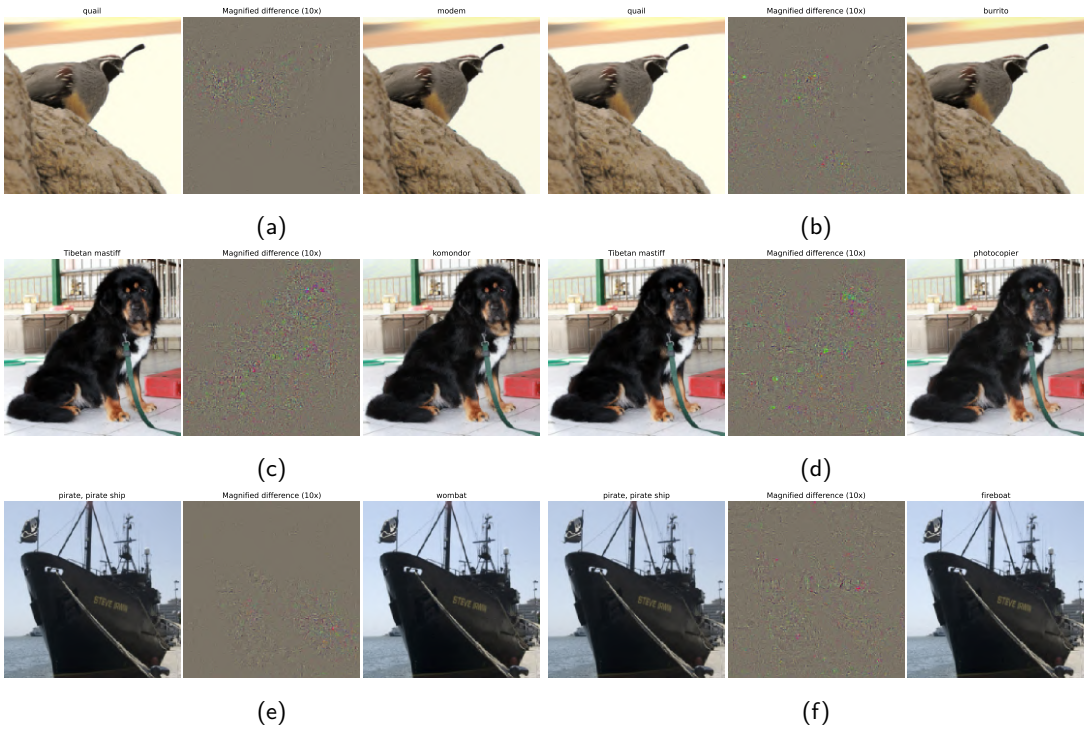


Figure 2.5: Adversarial examples. The left column displays the original image, the middle column shows the magnified difference added to the original image to deceive the model, and the right column presents the image that successfully fools the model.

6. In practice, what consequences can this method have when using convolutional neural networks? Adversarial techniques have practical applications beyond academic interest. 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.

Moreover, these techniques can be employed to enhance privacy and counteract mass surveillance. As an example, Cap_able's Manifesto Collection is a clothing line that uses patterns designed to shield facial recognition technologies. These garments are not recognized by models such as Yolo, instead causing misidentifications as various animals or inanimate objects. This line of clothing opens up a discussion on privacy and the protection of individuals from the misuse of biometric recognition cameras, offering a way to defend against unlawful intrusion into personal data and safeguard fundamental rights in public spaces.

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

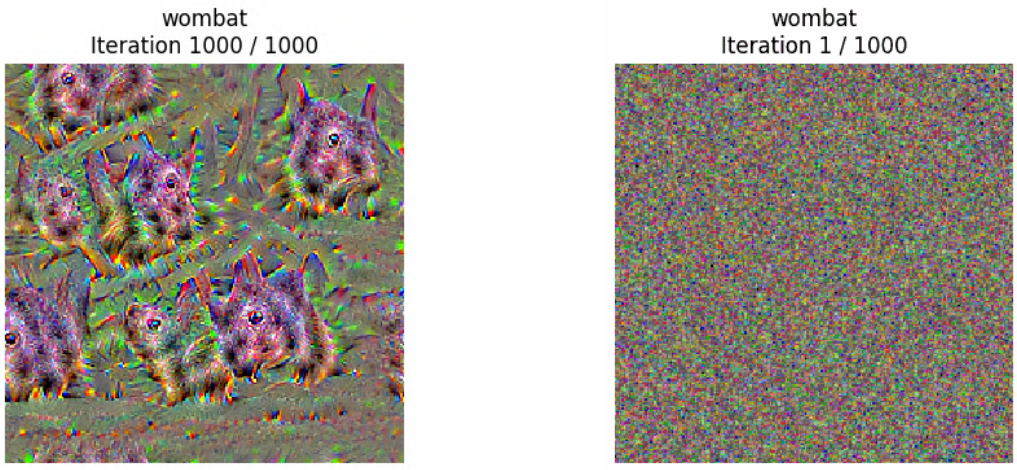
This section aims to generate images that highlight the type of patterns detected by a network for a particular class, based on techniques developed by Simonyan et al. (2014) and Yosinski et al. (2015). This method helps in visualizing what features the network prioritizes for classification. To view 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 2.6 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). While examining some of our animations, we observed the occurrence of blurring every 10 steps.

Interpretation: The reason the figure appears as it does, featuring multiple, somewhat distorted wombats and a somewhat psychedelic color scheme, is because the features that the network has learned to identify wombats may not precisely align with human perception.

The distortions and repetitions are artifacts of the process. The network is not trained to maintain the global coherence of the wombat's shape but rather to accentuate the specific features—such as fur texture, eye or nose shape—that it has associated with wombats. These features are then amplified to the extent that they activate the 'wombat' classification neurons as strongly as possible.



(a) Last iteration (b) Animation

Figure 2.6: 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. To improve class visualization, it appears that regularization and blurring plays a crucial role. In the following figure, we experimented with modifiying the learning rate, the image blurring frequency and the weight regularization on SqueezeNet.

As a baseline for comparison with upcoming experiments, Figure 2.7 represents the class visualization of the "bee eater" class using the default parameters from the practical implementation, which include a learning rate of 5, a regularization of $1e^{-3}$, and applying blurring every ten epochs.

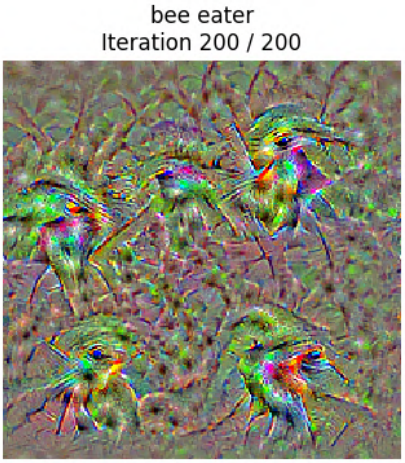
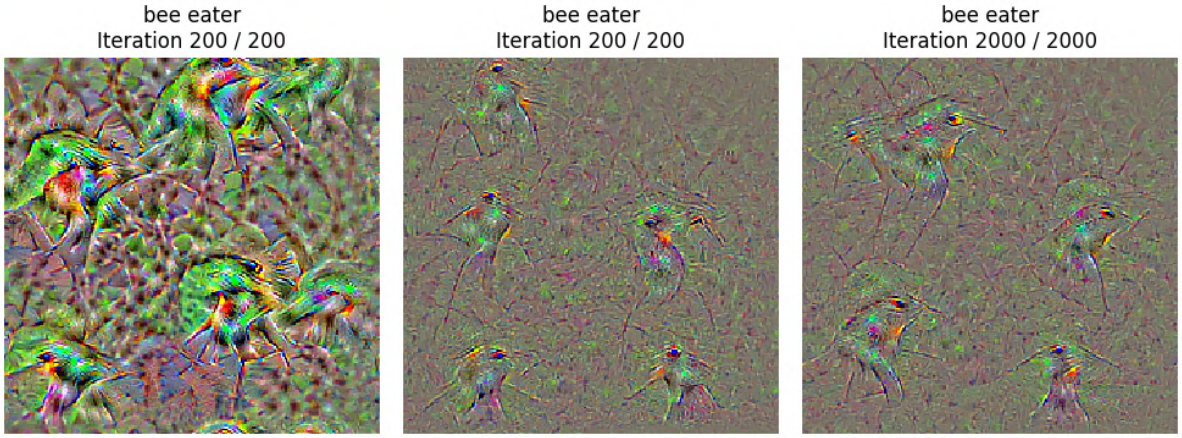


Figure 2.7: Baseline class visualization for the "bee eater" class, provided here for comparison, was generated using a learning rate of 5, a regularization strength of $1e^{-3}$, and blurring applied every ten epochs.

In our first experiment, we explore the impact of the l2-regularization parameter. As a reminder, the default regularization factor is set to $1e^{-3}$. Figure 2.8 presents the results obtained using a $1e^{-5}$ regularization factor for 200 epochs and $1e^{-2}$ regularization factors for 200 and 2000 epochs.

Non-regularized images appear to be overly saturated and noisy, even away from bird figures, making it challenging to discern the represented class clearly. Our suspicion is that without regularization, the image become saturated in **every pixel** that could have contributed to a correct prediction of the "bee eater" class to maximise the class score.

Conversely, when over-regularization is applied, the resulting images appear to have "lighter" class visualizations, even after 2000 epochs. However, the noise in these images remains low and consistent, and the boundaries of the birds are clear and distinguishable. On the downside, the birds themselves appear to be less colorful in these images. By constraining pixel values, we encourage the model to make judicious decisions about which pixels to modify, creating better frontiers.



(a) (b) (c)

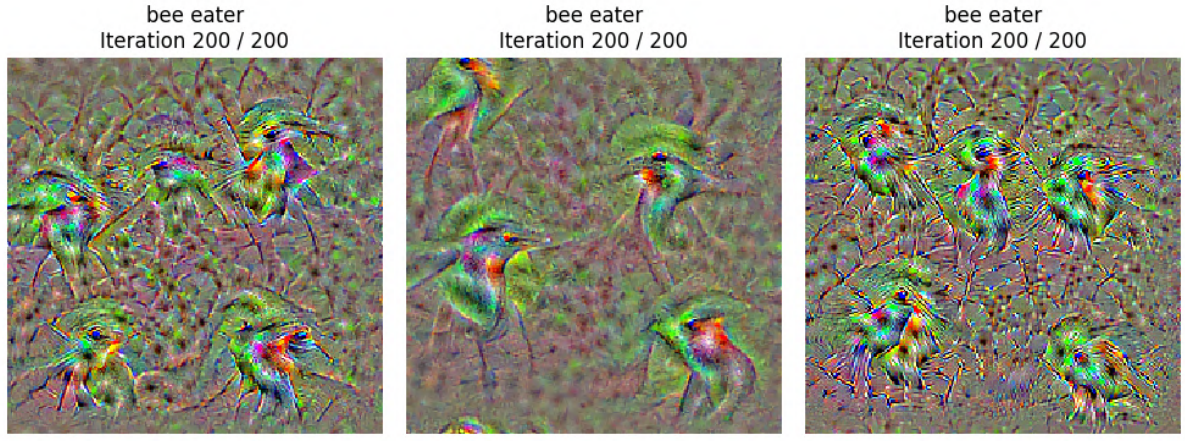
Figure 2.8: Comparaison of the bee eater class visualization using SqueezeNet (a) using low regularization ($1e^{-5}$), (b) using high regularization ($1e^{-2}$). (c) using high regularization ($1e^{-2}$) on 2000 epochs

In our second experiment, we focus on modifying the blurring frequency. In the baseline (Figure 2.7), blurring occurs every ten steps. In Figure 2.9, we explore two variations: blurring every two steps and no blurring at all. The first image represents the baseline (blur every ten epochs), provided here for quick comparison.

The second image, where blurring happens every two steps, showcases birds that are considerably more detailed and clear in our assessment. In contrast to the last image, there are no peculiar patterns extending around the birds.

Blurring appears to prevent the extension of repeated patterns around the visualization. By encouraging pixels that the model has already engaged in transformations and maintaining their direction, blurring facilitates

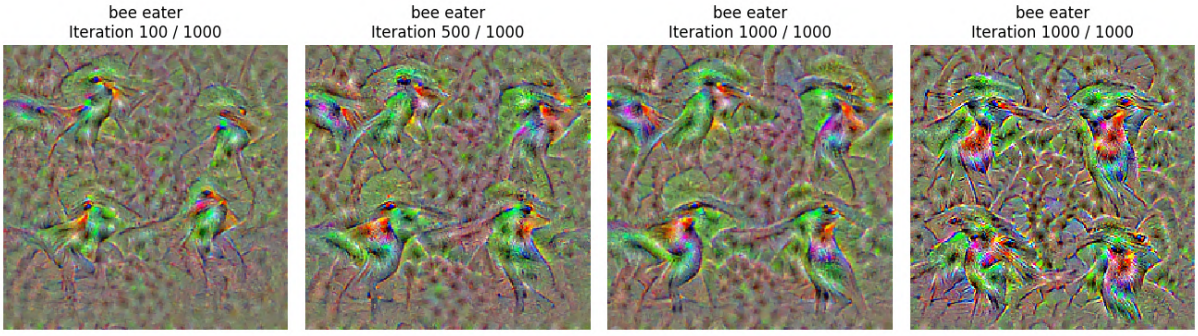
the emergence of a more genuine and detailed class image.



(a) (b) (c)

Figure 2.9: Comparison of the bee eater class visualization using SqueezeNet: (a) Blurring every ten steps (baseline, provided here for easier comparison). (b) Blurring every two steps. (c) Without blurring.

Regarding the number of epochs, our experiments indicate that a certain number of epochs are needed for the visualization to converge and produce better images. However, it's important to note that as the process continues, the images can become saturated or reach a point where further training may not significantly improve the results. This suggests that there's a trade-off between the number of epochs for convergence and the risk of overfitting or saturation in the generated images.



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

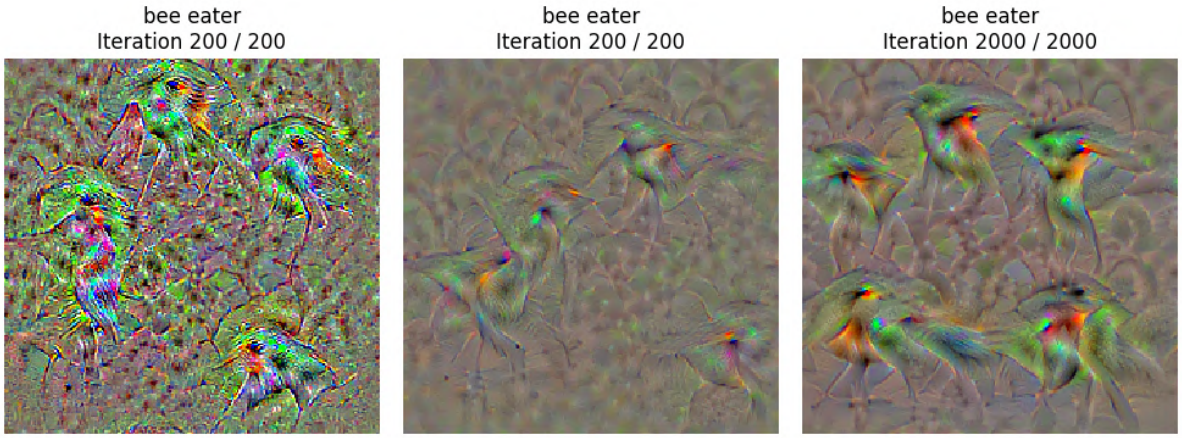
Figure 2.10: Comparaison of the bee eater class visualization using SqueezeNet with strong regularizations (a) 200 iterations, (b) 500 iterations, (c) 1000 iterations and (d) 1000 iterations without applying regularization.

About the learning rate, we know from previous courses that it permit a better exploration of loss landscape but modifiy the convergence speed.

We conducted experiments with two variations of the learning rate: 1. Increasing it by a factor of two (from 5 in the baseline to 10). 2. Reducing it by a factor of ten (from 5 in the baseline to 0.5). It's worth noting that reducing the learning rate makes the convergence significantly slower. To compensate for this, we also tried the reduced learning rate setting with 2000 iterations. Results are presented in Figure 2.11.

Using a high learning rate results in images that appear rushed, with a significant amount of noise, and the birds are nearly indistinguishable. On the other hand, with a low learning rate, especially at 2000 steps, the birds seem to merge with the background noise, and patterns appear larger and lighter.

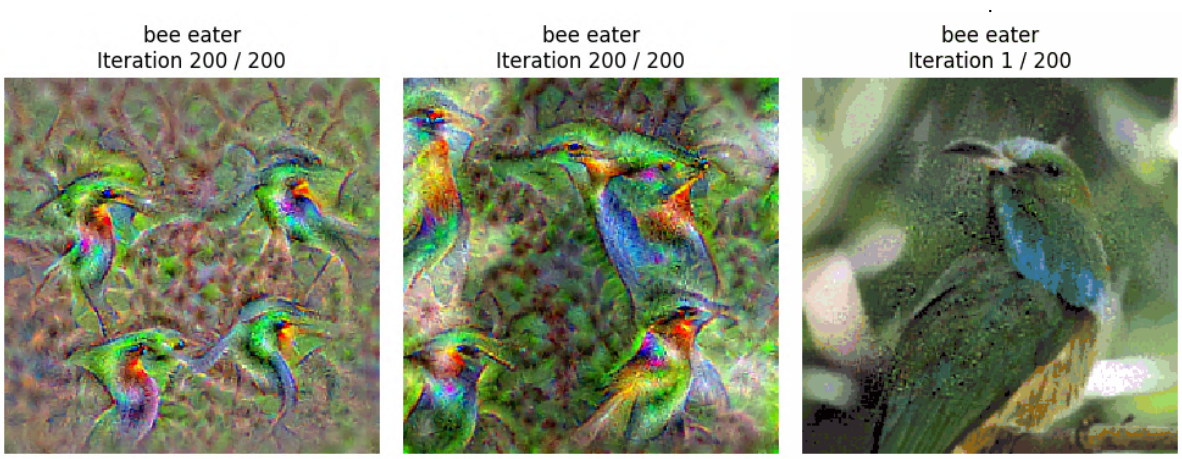
Our observation suggests that a lower learning rate allows for better convergence toward a global minimum, but it may also lead to the model getting trapped in a local minimum, resulting in the dreamlike aspect of the image.



(a) (b) (c)

Figure 2.11: Comparaison of the bee eater class visualization using SqueezeNet (a)

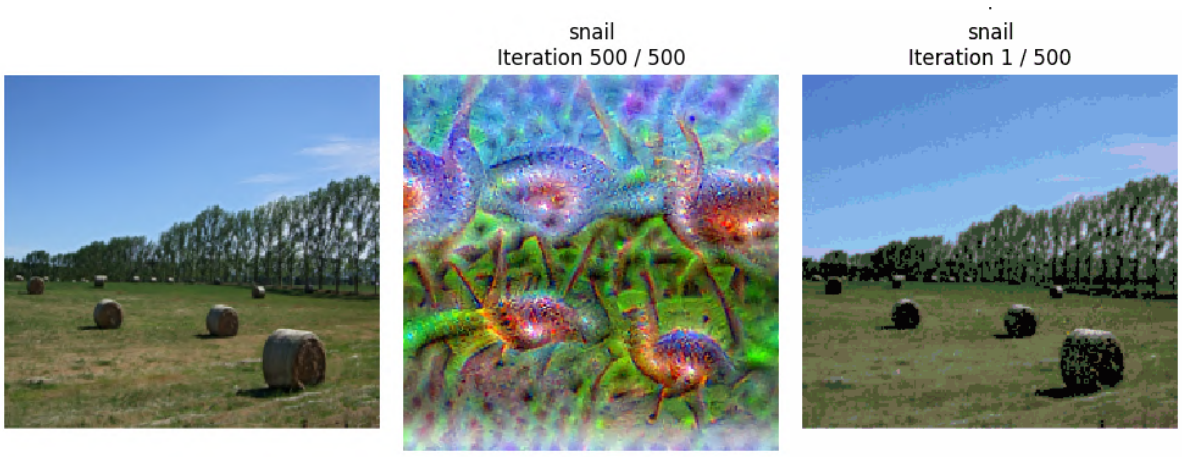
10. Try to use an image from ImageNet as the source image instead of a random image. You can use the real class as the target class. Comment on the interest of doing this. Initiating the process with an image from ImageNet is a valuable approach. It provides a meaningful starting point for gradient-based 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)

Figure 2.12: 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 high blurring frequency (2) and a sligthly lower regularization ($1e^{-5}$).

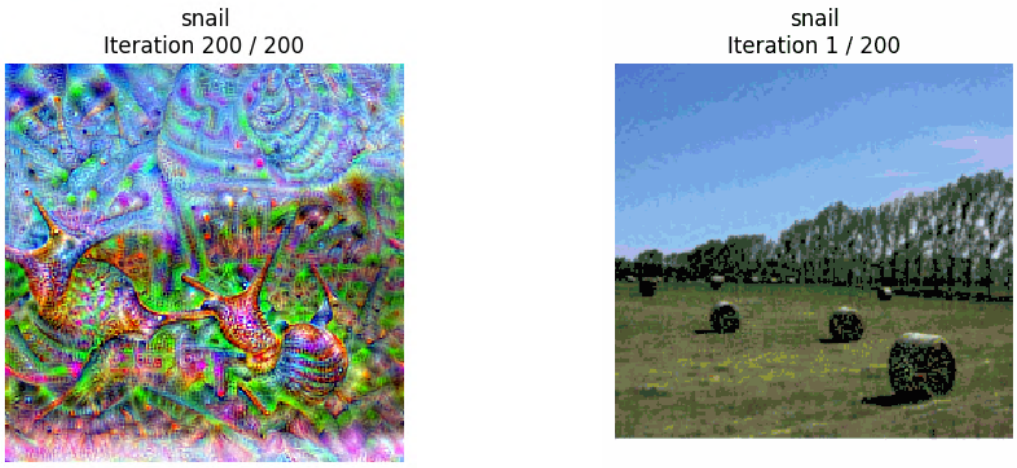
It pretty funny to create some objects from other objects. In Figure 2.13 we started from a image of hays to do a snail class visualization.



(a) Original image (b) Last iteration (c) Animation

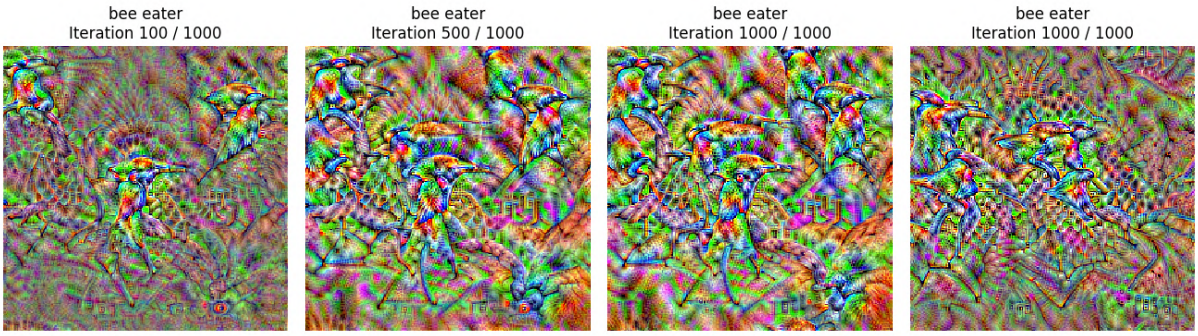
Figure 2.13: Class visualization using SqueezeNet with strong regularizations: Initiated from an initial hay image, aiming to maximize the score for the snail class.

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 2.14b to be more appealing. However, there are instances where regularization proves to be more challenging, as observed in Figure 2.15 or when comparing 2.16b and 2.16c.



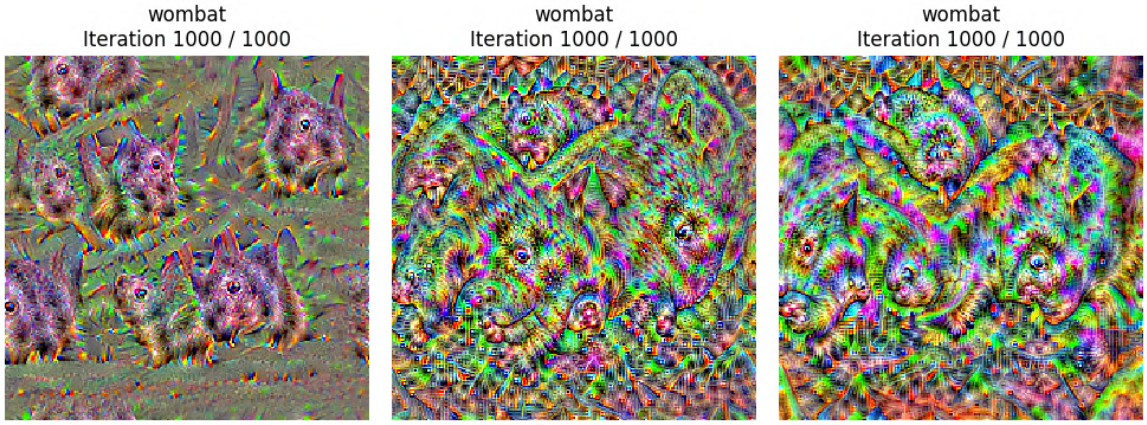
(a) Last iteration (b) Animation

Figure 2.14: Class visualization using VGG with high blurring frequency (2) and a sligthly lower regularization ($1e^{-5}$): started from a hay image to maximising the score for the snail class.



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

Figure 2.15: Comparaison of the bee eater class visualization using VGG with with high blurring frequency (2) and a sligthly lower regularization ($1e^{-5}$) (a) 200 iterations, (b) 500 iterations, (c) 1000 iterations and (d) at 1000 iterations but without having performed regularization.

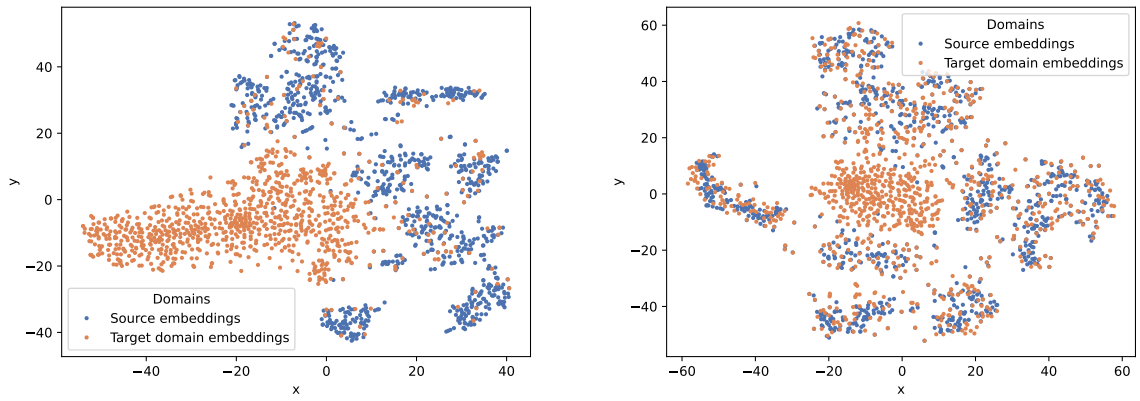


(a) (b) (c)

Figure 2.16: Class visualization of Wombat. (a) Using SqueezeNet for comparison with default parameters; (b) Using VGG with default parameters; (c) Using VGG with high blurring frequency (2) and a sligthly lower regularization ($1e^{-5}$).

Chapter 3

Domain Adaptation



(a) (b)

Figure 3.1: t-SNE visualization of the feature extractor latent space: (a) for a baseline network without domain adaptation training. (b) for a DANN (Domain-Adversarial Neural Network) after domain adaptation training.

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? The gradient of the domain classifier is directed in a way that aids it in distinguishing between features from the source and target domains. In the absence of the Gradient Reversal Layer (GRL), the gradient would be propagated to the feature extractor, encouraging it to assist the domain classifier in distinguishing between source and target domain features. This would result in a model that becomes more domain-specific rather than domain-generalized, as it would enable the domain classifier to excel at discriminating between source and target domains, contradicting the goal of domain adaptation.

To address this issue, we employ a simple technique: we reverse the sign of the gradient, causing it to move in the opposite direction. This reversal helps the feature extractor generate domain-agnostic features, aligning with the objective of achieving domain adaptation.

2. Why does the performance on the source dataset may degrade a bit? During our experimentation, we saw the source domain accuracy go from 99.07% to 98.64% and the target domain accuracy going from 53.52% to 78.85%. 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.

After 100 epochs, we a achived a target domain accuracy of 80%, keeping the source domain accuracy at 98.5%. The domain discriminator had an accuracy of 49%.

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.

The practical propose to use a modified version of the original DANN paper's scheduler for the GRL (Gradient Reversal Layer) value, defined as

Our version	DANN Paper version
$f : \mathbb{N} \to [0, 1]$	$g : \mathbb{N} \to [0, 1]$
$x \mapsto \frac{2}{(1+\exp(-2\frac{x}{n}))} - 1$	$x \mapsto \frac{2}{(1+\exp(-10\frac{x}{n}))} - 1$

where n represents the number of training iterations (number of images multiplied by the number of epochs). The ratio $\frac{x}{n}$ is denoted as p in the original paper, representing the training progress linearly changing from 0 to 1. The only difference between the two versions is the scalar value 2 or 10, which we can denote as c.

As an experiment, we attempted to vary c within the range of 1 to 9. The results are presented in Figure 3.2, which illustrates the accuracy of DANN on the target domain after 30 epochs. As is common in many adversarial training setups, it's worth noting that training can sometimes exhibit random failures or instability. We reran the experiment a second time, and it appears that higher values of c tend to introduce instability in the training process, resulting in more failures. Further experiments may be necessary to confirm this observation. We also attempted to use $c = 10$ as in the original paper, but the training still experienced failures. Additional tuning is required to gain a better understanding of this parameter.

As a baseline experiment, we also tried directly using -1 as the GRL factor. However, this approach resulted in poor performance, with only a 45% accuracy on the target domain after 20 epochs.

Please note that the performance and stability of adversarial training can be highly dependent on the specific dataset, model architecture, and other hyperparameters, so further experimentation and fine-tuning may be necessary to achieve optimal results.

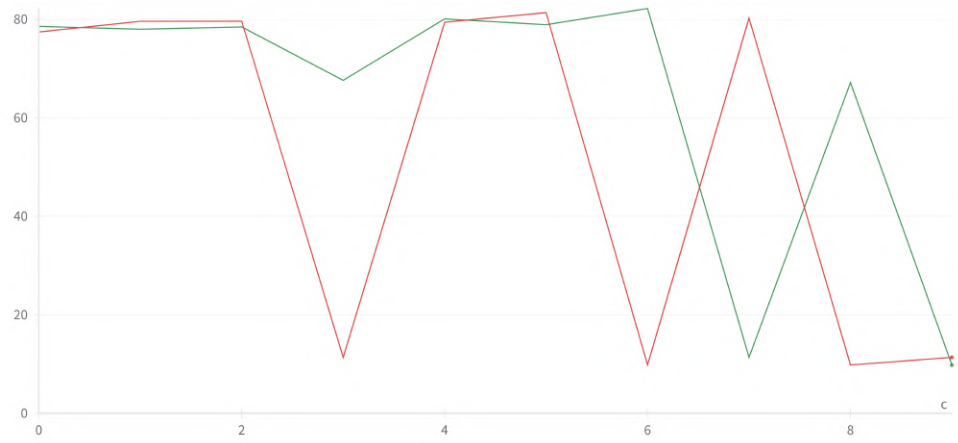


Figure 3.2: Final class accuracy on the target domain in function of c for two runs. An augmentation of c seem to create instability in training but further run might be needed.

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.

We attempted to implement a basic pseudo-labeling method, but unfortunately, it did not yield significant improvements. The accuracy remained relatively constant over 20 pseudo-labeling iterations. While further tuning may be necessary, we also suspect that the presence of both correct and incorrect labels may counterbalance each other, hindering the model's improvement.

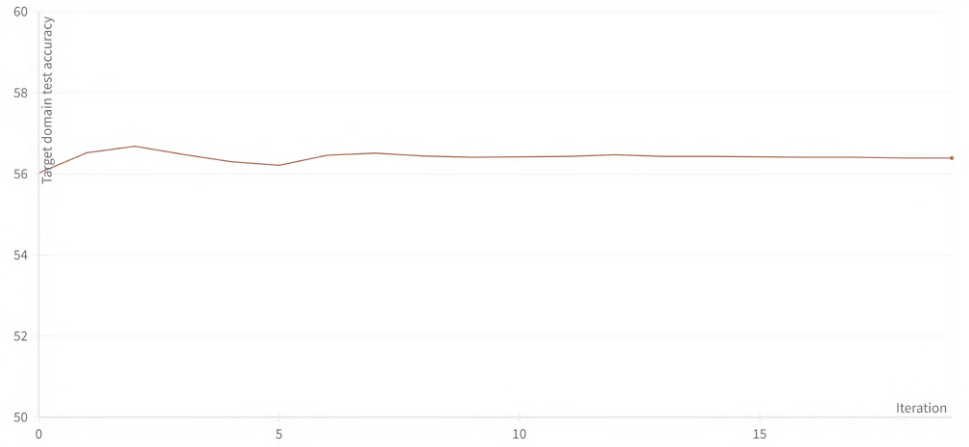


Figure 3.3: Accuracy on the target domain's test dataset at each pseudo-labeling iteration is being monitored. The model's improvement appears to be very limited, potentially because the balance between correct and incorrect labeling tends to average out.

Chapter 4

Generative Adversarial Networks

This exploration delves into the realm of generative models, specifically focusing on Generative Adversarial Networks (GANs) and their conditional variants. GANs have emerged as a groundbreaking concept in unsupervised learning that revolutionized image generation and manipulation. The objective is not only to comprehend the technical mechanisms of GANs but also to recognize their profound impact on image generation, editing, and completion. This exploration also highlights how these models are reshaping our approach to generative tasks within the field of machine learning in a broader context.

4.1 Generative Adversarial Networks

This section presents Generative Adversarial Networks (GANs), a deep learning framework that employs two neural networks, a generator, and a discriminator, in a competitive setup. It offers fundamental insights into how these networks collaborate to produce synthetic data instances that closely resemble real data. The focus is on understanding the architecture, the training procedure, and the core principles, laying the foundation for their various applications in areas such as image generation, style transfer, and beyond.

4.1.1 General principles

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

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

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

1. Interpret the equations 4.2 and 4.3. What would happen if we only used one of the two? Equation (4.2) describes the goal of the generator G in a GAN. The generator aims to produce images $\tilde{x} = G(z)$ that are indistinguishable from real images by the discriminator. The objective function for the generator maximizes the probability of the discriminator incorrectly classifying these generated images as real. In other words, the generator is trained to "fool" the discriminator. A higher value of $D(G(z))$ implies that the discriminator is more likely to mistake the generated image for a real one, indicating a better performance of the generator.

Equation (4.3) describes the goal of the discriminator D in a GAN. The discriminator's role is to distinguish between real images x^* from the dataset and fake images $\tilde{x}$ produced by the generator. The objective function for the discriminator is to maximize the sum of two terms: the probability of correctly identifying real images as real, and the probability of correctly identifying generated images as fake. The first term ($\mathbb{E}_{x^* \sim \mathcal{D}\text{ata}} [\log D(x^*)]$) encourages the discriminator to correctly classify real images as real, while the second term $\left(\mathbb{E}_{z \sim P(z)} [\log(1 - D(G(z)))]\right)$ pushes it to correctly identify the synthetic images as fake.

If we only used the generator's objective function (4.2) without the discriminator, the generator would not have a reliable way to measure how good its generated images are. It would lack the feedback mechanism that the discriminator provides, leading to poor-quality image generation. Conversely, if we only used the discriminator's objective function (4.3) without the generator, the discriminator would only learn to distinguish real images from a static set of fake images. It would not adapt to improving fake images, making it less effective over time as it would not be challenged by increasingly realistic fake images. A GAN relies entirely on the interplay between the generator and discriminator: the generator improves by trying to fool the discriminator, while the discriminator becomes better at distinguishing real from fake images, leading to a dynamic and effective learning process.

2. Ideally, what should the generator G transform the distribution $P(z)$ to? Ideally, the generator G in a GAN should transform the input noise distribution $P(z)$ into a distribution that closely resembles the real data distribution $P(X)$.

In a GAN, z is a vector sampled from a predefined noise distribution $P(z)$, which is typically a uniform or normal distribution. The role of the generator G is to map this noise vector z into a data point $\tilde{x} = G(z)$ that appears to have been drawn from the distribution of real data $P(X)$. The success of the generator is measured by how indistinguishable its output $\tilde{x}$ is from real data points when evaluated by the discriminator.

The end goal is for the distribution of the generated data $P_G(X)$ (where X is generated by $G(z)$ for $z \sim P(z)$) to be as close as possible to the real data distribution $P(X)$. When this happens, the discriminator should find it increasingly difficult to tell apart real data from the generated data, ideally reaching a point where it performs no better than random guessing, resulting in equilibrium, i.e. $\forall x,\ D(x) = 1/2$. This signifies that the generator has effectively learned to produce data that mimics the real data distribution.

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? In the concept of a GAN, the generator's goal is ideally defined as minimizing the likelihood that the discriminator correctly identifies its outputs as fake. Mathematically, this gives us the following "true" equation:

$$\min_G \mathbb{E}_{z \sim P(z)} \left[\log(1 - D(G(z)))\right]$$

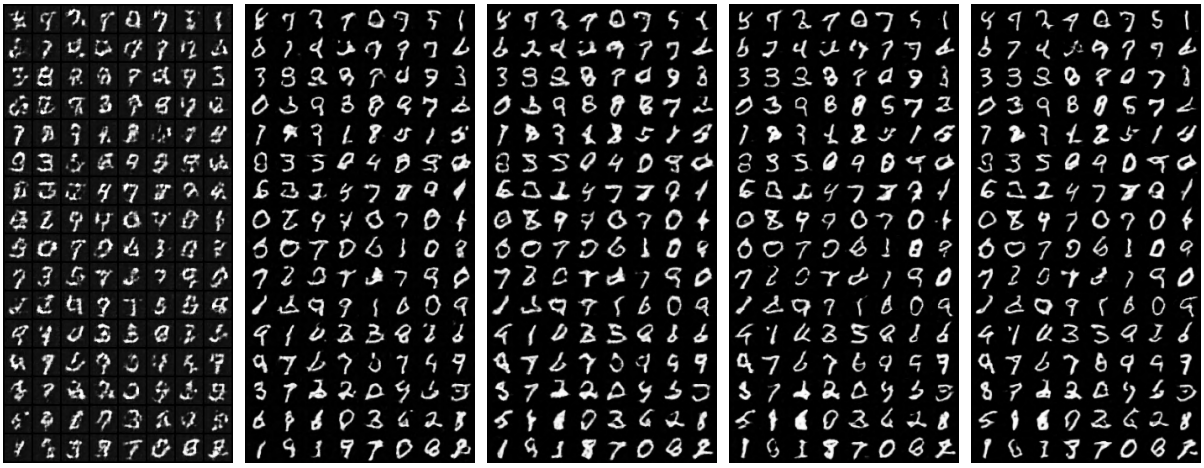
This objective aims for the generator to produce images that the discriminator will classify as fake (i.e., it tries to minimize the probability of the discriminator being correct).

4.1.2 Architecture of the networks

4. Comment on the training of the GAN with the default settings (progress of the generations, the loss, stability, image diversity, etc.). In our GAN training, we observed a notable progression in the quality of generated images as the training advanced. However, the loss values exhibit significant instability, oscillating across epochs, which is a common trait in GAN training, indicating their tendency not to converge in practice. This means that while training over 5 epochs, neither G nor D wins decisively, but D has a tendency to perform better at the game, i.e., it manages to identify the fake images, which in turn encourages G to produce better images.

While the generator does improve image quality over time, the generated results still appear noticeably synthetic and limited in diversity, primarily resembling digits such as 0, 3, 7, and 8. This observation raises concerns about a potential mode collapse issue, where the generator learns to produce only a subset of possible outputs that are sufficient to deceive the discriminator but lacks overall diversity.

For a detailed reference, please consult the training curves in Figure 5.8 in Appendix Section 5.2.1, where the 5-epoch mark corresponds to 2300 steps, and review the generated images in Figure 4.1.



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

Figure 4.1: Generated digits from a GAN during training using default settings after (a) 1 epoch, (b) 2 epochs, (c) 3 epochs, (d) 4 epochs and (e) 5 epochs.

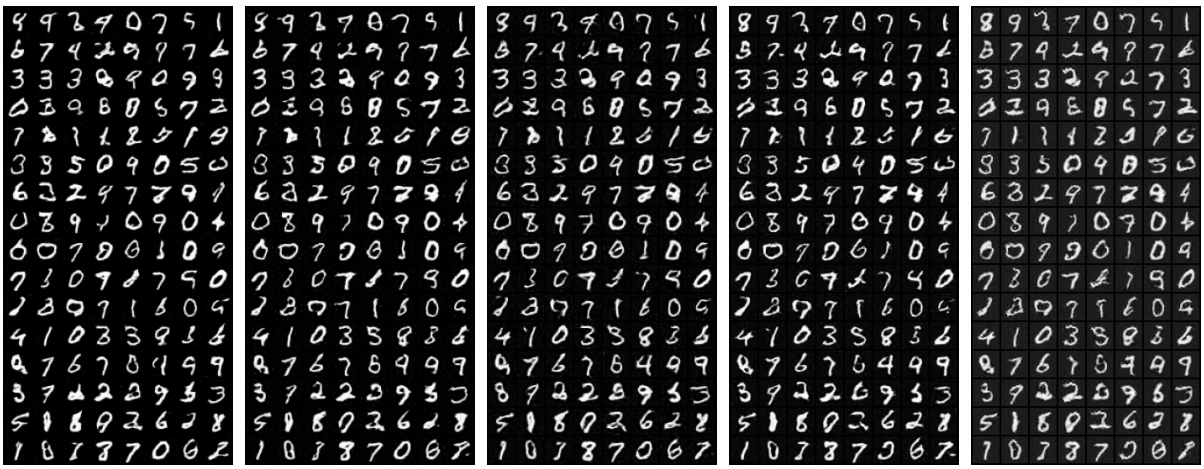
5. Comment on the diverse experiences that you have performed with the suggestions above. In particular, comment on the stability on training, the losses, the diversity of generated images, etc. To conduct our experiments, we developed a PyTorch-Ignite framework, which is included alongside this report for reference. This framework played a crucial role in ensuring the reproducibility and traceability of our experiments. It's worth noting that the curves presented in our results represent the number of total iterations, calculated as the number of epochs multiplied by the batch size. Consequently, it's normal to observe peaks in the curves, which indicate the start of a new epoch during training.

Longer epochs. In our experiments, we initially extended the training to 30 epochs. In deep learning, it's a common practice that more epochs can contribute to improved stability. Beyond the 10th epoch, we noticed consistent convergence in subsequent epochs. Further analysis of the training curves, depicted in Appendix Section 5.2.1, Figure 5.8, revealed that the discriminator's output, $D(x)$, approached 1, and the discriminator's output for generated images, $D(G(z))$, approached 0. This essentially means that the discriminator perfectly distinguished real images from fake images, indicating that the discriminator was winning the adversarial game, while the generator was losing, which is not the outcome we are looking for.

This situation prompted an intriguing observation when observing the generation process during training, as shown in Figure 4.2. We observed the evolution of the background in these images. Originally black, it gradually developed a form of noise. This phenomenon can be attributed to the generator's struggle to produce better digits. When it cannot improve the quality of the digits, it starts generating noise as a way to produce something different. If we were to continue training for even longer, we might end up with completely noisy images.

This observation underscores the delicate balance in GAN training. Increasing the number of epochs can be risky, as it can lead to the discriminator becoming proficient at distinguishing between real and fake images, causing the generator to resort to generating noise to try and deceive the discriminator.

In conclusion, it's important to note that as training progresses, the discriminator may start giving more and more random feedback, and the generator might train on uninformative feedback, resulting in a decrease in the quality of its generated outputs. As a result, convergence in GANs is often a temporary state rather than a stable and permanent condition.

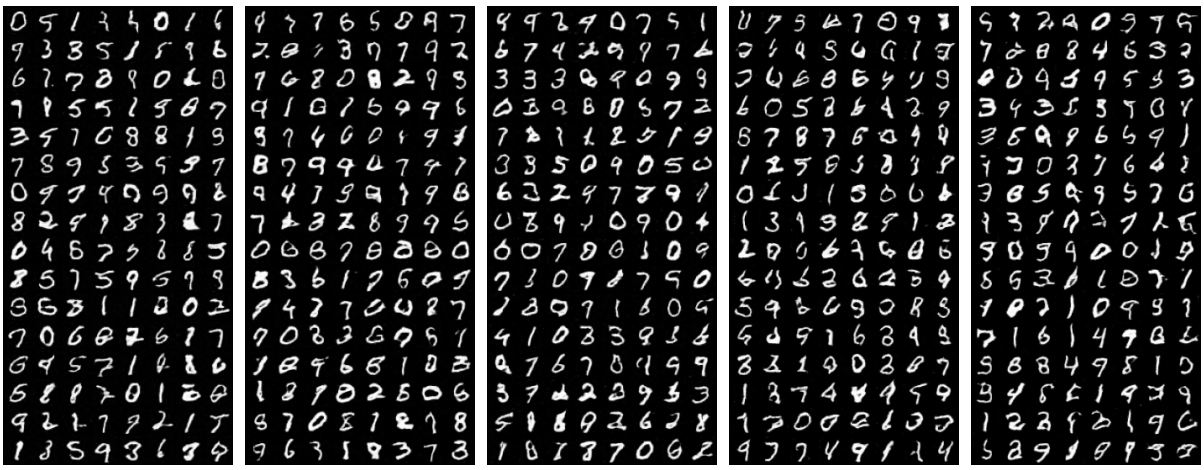


(a) (b) (c) (d) (e)

Figure 4.2: Generated images during training after (a) 10 epochs, (b) 15 epochs, (c) 20 epochs, (d) 25 epochs and (e) 30 epochs.

Influence of n_z. We conducted an investigation to explore the impact of varying the size of the noise vector (n_z), a parameter that plays a role in determining the diversity and complexity of the generated images. The results of this experiment are presented in Figure 4.3.

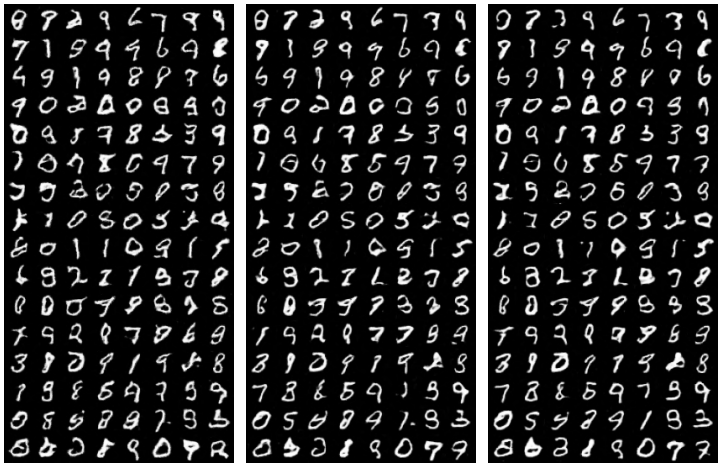
Analyzing only the losses and the discriminator's output values, there doesn't appear to be a significant difference between the various values of n_z. This suggests that the impact of n_z on the training process may not be fully reflected in these metrics alone. However, when we consider the generated images themselves, we observe more significant differences. Decreasing the value of n_z resulted in a reduction in the diversity of the generated outputs but allowed the model to produce good quality images more quickly. On the other hand, increasing the value of n_z enhanced diversity among the generated images but introduced greater complexity to the training process. These observations indicate that the choice of n_z has a more noticeable impact on the visual diversity and complexity of the generated images compared to its effect on training metrics like losses and discriminator outputs.



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

Figure 4.3: Generated images after being trained for 10 epochs with (a) $n_z = 10$ (b) $n_z = 50$ (c) $n_z = 100$ (d) $n_z = 500$ and (e) $n_z = 1000$.

Influence of weight initialization. Upon switching to PyTorch's default weight initialization, we observed that the discriminator managed to distinguish between fake and real images in just 6 epochs, as evidenced by the curves in Figure 5.9 in Appendix Section 5.2.2. Thus, the quality of the generated images was less convincing when using the default weight initialization compared to the results obtained with the optimized parameters. This outcome highlights the significant impact of customized weight initialization in GANs, as it affects both the training efficiency and the overall quality of the generated images.



(a) (b) (c)

Figure 4.4: Generated images during training with default PyTorch weight initialization after (a) 5 epochs, (b) 10 epochs and (c) 15 epochs.

Influence of the learning rate. We experimented with different combinations of learning rates and β_1 values in the Adam optimizer for both the discriminator and the generator. Specifically, we tried:

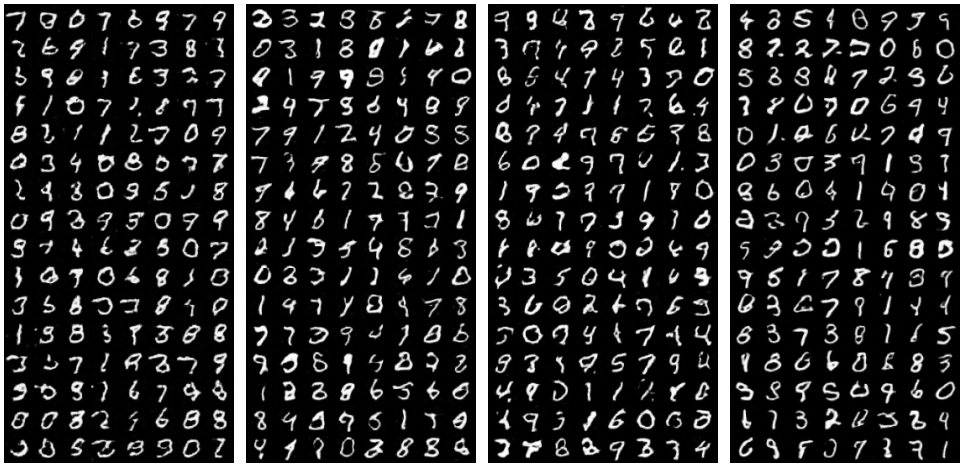
- Higher learning rate (0.001) for discriminator only.
- Higher learning rate (0.001) for discriminator only with a higher β_1 (0.9).
- Higher learning rate (0.001) with a higher β_1 (0.9) for both.
- Recommended learning rate with a higher β_1 (0.9) for both.

Our observations align with the findings of the authors, suggesting that increasing the learning rate and β_1 in Adam can make the GAN learn too quickly, which can negatively impact training stability and the quality of generated images. The training curves can be found in Figure 5.10 in Appendix Section 5.2.3.

Influence of ngf and ndf. When we increased the value of ndf to 128 while keeping ngf at 32, the discriminator consistently detected which image was real and which image was fake, regardless of the generator's efforts. Conversely, when we reduced ndf to 8 and left ngf at 32, we found ourselves in a situation where we were always close to equilibrium after the first epoch.

However, when we increased ngf to 128 while keeping ndf at 32, it did not help the generator generate better images. The discriminator still gained an advantage and "won" the game after 6 epochs. Similarly, when we reduced ngf to 8 and left ndf at 32, the discriminator consistently came out on top.

Interestingly, despite these differences in training dynamics, when looking at the generated images, there doesn't seem to be much discernible distinction to the human eye. We have provided generated images after 10 epochs for reference in Figure 4.5. The learning curves can be found in Appendix Section 5.2.4.

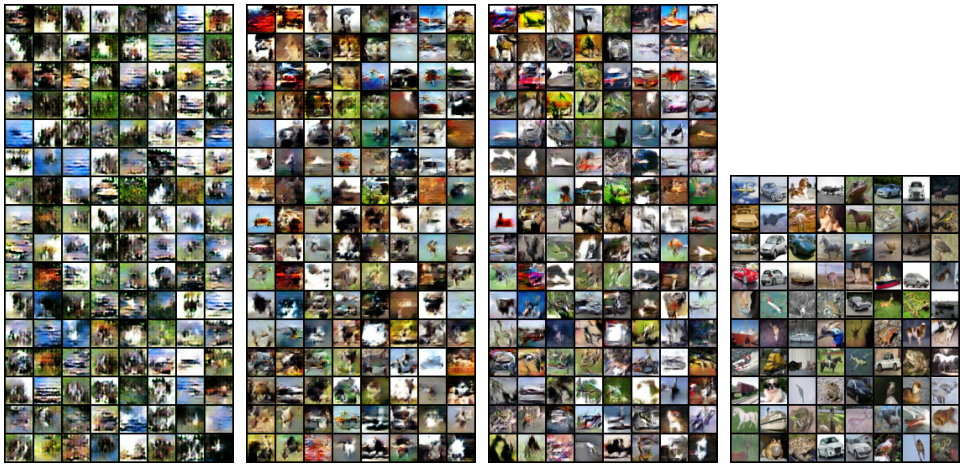


(a) (b) (c) (d)

Figure 4.5: Generated digits after 10 epochs with (a) $ngf = 8$, (b) $ngf = 128$, (c) $ndf = 8$ (d) $ndf = 128$.

Trying CIFAR-10 dataset. Unfortunately, due to Google Drive limitations, we were unable to perform experiments using CelebA. Instead, we conducted experiments on CIFAR-10, a dataset of more complex images. We trained a GAN over 15 epochs on 32×32 generated images. It's important to note that the generated images won't match the beauty of those in the dataset due to the reduced resolution. However, this experiment yielded interesting results.

One notable observation is that the generator had difficulty distinguishing the generated images from the real images, as indicated by the training curves in Figure 5.15 in Appendix Section 5.2.5. We have provided the generated images in Figure 4.6. While at first glance, these images may appear indistinct, they still exhibit characteristics that suggest they were generated from CIFAR-10. This implies that the generator has learned the underlying distribution of the images, even though the generated images themselves may not be visually appealing due to the limited resolution.



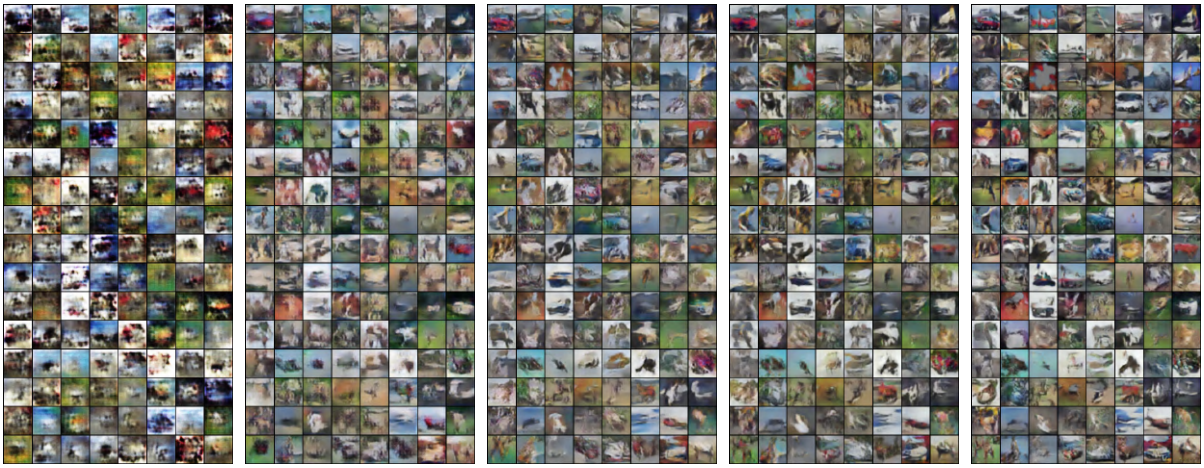
(a) (b) (c) (d)

Figure 4.6: Generated 32×32 CIFAR-10 images using default settings after (a) 5 epochs, (b) 10 epochs and (c) 15 epochs. (d) are examples of real 32×32 CIFAR-10 images.

Generating higher resolution images on CIFAR-10. To challenge our GAN further and explore its capabilities with higher resolution imagery, we extended our scope from generating 32×32 CIFAR-10 images to 64×64 images. This transition necessitated an additional block in our models to accommodate the increased detail and scale, resulting in longer training durations due to the greater complexity of handling larger image sizes.

In this experiment, we observed intriguing results in the training curves, displayed in Figure 5.16 in Appendix Section 5.2.6. Initially, during the first few epochs, the discriminator was highly proficient at distinguishing between the images generated by the generator, leading to $D(G(z)) \to 0$. However, as training progressed, G began producing higher-quality images, making it more challenging for D to differentiate between real and fake images. Nevertheless, significant instability is still evident in the training process.

The generated images from this experiment are displayed in Figure 4.7. To achieve good results, we need much more epochs than in MNIST or when we generated 32×32 CIFAR-10 images. We can see that at 5 epochs, the images are still blurry, and it's only after 15 epochs that they begin to exhibit some shapes characteristic of the classes in the CIFAR-10 dataset. Even after 30 epochs, they still lack the realism found in the original dataset. However, these shapes are more recognizable than those in the 32×32 generated images. For instance, the first image located at the top left corner vaguely resembles the red truck from the real images in Figure 4.8 (third row, second column).



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

Figure 4.7: Generated 64×64 CIFAR-10 images using default settings after (a) 5 epochs, (b) 10 epochs, (c) 15 epochs, (d) 20 epochs and (e) 30 epochs.

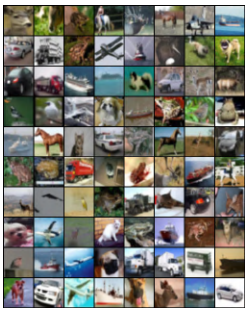


Figure 4.8: Examples of real 64×64 CIFAR-10 images.

Our experiments with GANs reaffirm their complexity in training. Balancing the generator and discriminator, adjusting hyperparameters, and the challenges introduced by image scaling highlight the intricate nature of GANs. Despite these challenges, GANs' ability to generate high-quality images showcases their significant potential, given the right conditions and resources.

4.2 Conditional Generative Adversarial Networks

This section delves into Conditional Generative Adversarial Networks (cGANs), an advanced variant of GANs. We explore how cGANs extend the GAN framework by introducing conditional variables, enabling more control

over the generated outputs. We examine how this added layer of control allows for targeted image generation, enhancing the utility of GANs in tasks like image-to-image translation and targeted style transfer. The focus is on understanding the structural differences from standard GANs and the implications of these changes in practical applications.

4.2.1 cDCGAN Architectures for MNIST

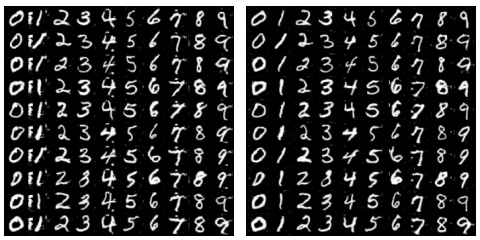
6. Comment on your experiences with the conditional DCGAN. Our experience with the conditional DCGAN showed a marked improvement in training stability compared to the unconditional model. Impressively, the network produced highly recognizable and differentiated images, which underscores the efficacy of conditioning the model, as displayed in Figure 4.9. This outcome suggests that incorporating conditional variables into the GAN architecture significantly enhances the model's ability to learn and generate targeted, high-quality images rapidly. However, it's worth noting that training takes much longer than a non-conditional GAN, which is a trade-off for the improved stability and control over image generation.

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

Figure 4.9: Generated digits from a cDCGAN during training using default settings after (a) 1 epoch, (b) 2 epochs, (c) 3 epochs, (d) 4 epochs and (e) 5 epochs.

Nonetheless, similar to previous GAN experiments, the conditional DCGAN encountered challenges when trained for longer epochs. Extending training to 30 epochs resulted in the same issues discussed in our previous experiments. Training curves are provided in Figure 5.17 in Appendix Section 5.2.7, and some generated images in Figure 4.10 reveal that the cDCGAN introduced noise in an attempt to outperform the discriminator.

Nevertheless, it suffers from the same problems as talked before, notably when using longer epochs. Using 30 epoches led into the same problems discussed in our previous experiments. Training curves can be found in Figure 5.17, and in Figure 4.10, we present some generated images that demonstrate the incorporation of noise, as previously described.



(a) (b)

Figure 4.10: Generated digits from a cDCGAN during training using default settings after (a) 25 epoch and (b) 30 epochs.

7. Could we remove the vector y from the input of the discriminator (so having $cD(x)$ instead of $cD(x, y)$)? Removing the vector y from the input of the discriminator in a conditional cGAN would significantly alter the network's nature and functionality. The discriminator's role would be reduced to assessing the realism of the images, without the capability to evaluate whether the generated images meet the specified conditions. Consequently, it would be unable to determine whether a generated digit image corresponds to the provided digit label. On the other hand, the generator would still receive y as input. However, with the discriminator no longer enforcing condition compliance, there would be less motivation for the generator to produce images that conform to the specific conditions. This would lead the network towards generating generic, realistic images that may not necessarily align with the intended conditional attributes.

In our experiment, we modified the code to employ an unconditional discriminator while keeping a conditional generator. Our observations supported our hypotheses: because the discriminator lacks conditioning,

the generator is not compelled to align with specific conditional inputs. Consequently, the latent space is not effectively conditioned, leading to a lack of consistent correspondence between specific conditions and the generated images. This was evident as the digits appeared to 'move' positions during training, as the discriminator did not impose any particular condition. We provide results in Figure 4.11

Additionally, we observed mode collapse, where each digit is only associated once, meaning that certain digits were not being adequately represented in the generated samples, leading to a lack of diversity in the output. This underscores the critical role of a conditional discriminator in a cGAN framework, as it ensures that the generated images align with the specified conditions and helps prevent mode collapse by encouraging diversity in the generated samples.

Figure 4.11: Generated digits from a cDCGAN trained for 10 epochs with the removal of the vector y from the input of the discriminator.

8. Was your training more or less successful than the unconditional case? Why? The training in the conditional case can be considered more successful compared to the unconditional one primarily because the introduction of the conditioning variable y allows the GAN to generate images that are conditioned on specific attributes or classes. This directed generation process can lead to a more controlled and diverse output. For example, in an unconditional GAN, without y, the generator has no guidance on what type of image to produce, which can sometimes result in mode collapse, where the generator produces very similar or even the same images repeatedly.

However, with the conditional GAN, each y corresponds to a different attribute or class label. This means that for each different y, the generator is tasked with producing an image that not only looks realistic but also matches the given condition. This helps to ensure that the generator produces a diverse set of images across different classes or attributes, effectively avoiding mode collapse. The discriminator, in turn, has to learn to judge not just the realism, but also the correctness of the conditioned output. This makes the training more robust and the model more versatile, as it learns a richer representation of the data.

9. Test the code at the end. Each column corresponds to a unique noise vector z. What could z be interpreted as here? In this context, the noise vector z can be interpreted as the source of randomness or variability that the generator G utilizes to produce different outputs. Each column in Figure 4.12 corresponds to a unique z, and since z is repeated for each class, it serves as a means to generate variations of images across different classes.

Essentially, z acts as a seed for the generative process. Different seeds are expected to yield different images, and with the repetition of z across classes, it's used to generate diverse variations of images corresponding to each class label provided in y_s. In summary, z introduces the stochastic element that, in conjunction with class labels, enables the generator to create a variety of images within the specified classes.

Figure 4.12: Influence of a unique noise vector z on our cGAN's generated images.

Chapter 5

Appendix

5.1 Transfer Learning

5.1.1 Activation maps

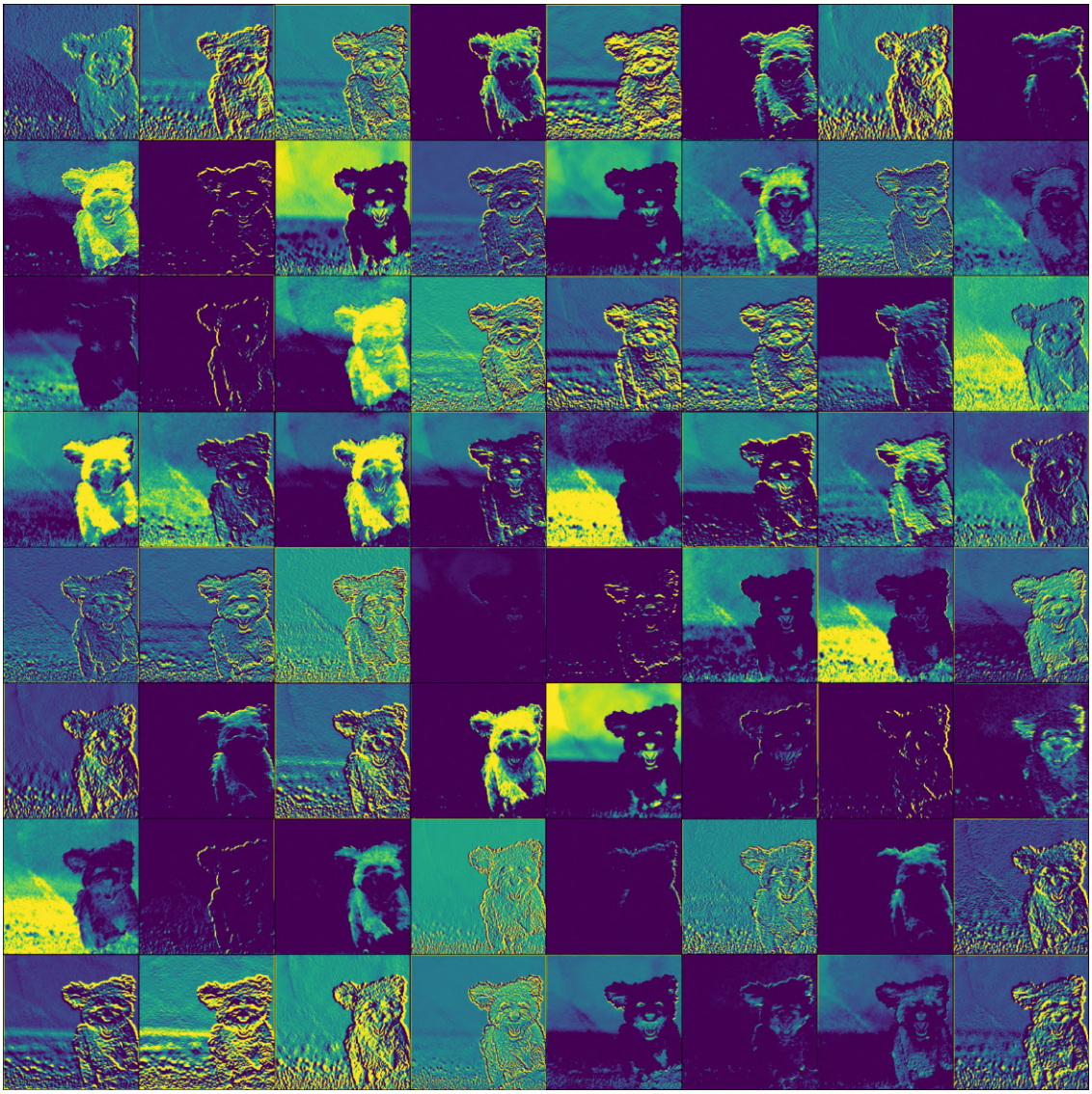


Figure 5.1: Activation maps obtanied after the first convolutional layer, on the Shih Tzu dog image.

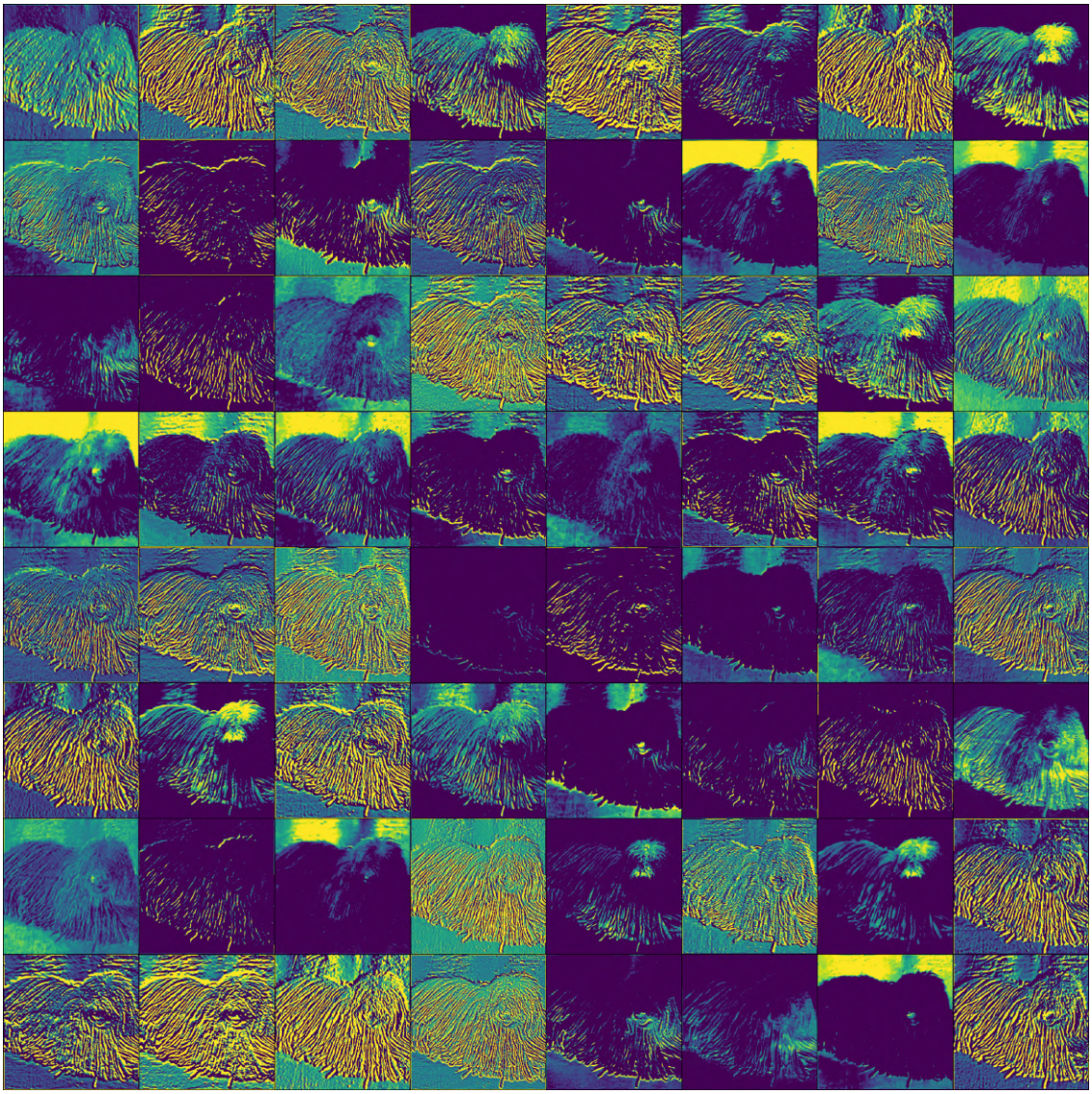


Figure 5.2: Activation maps obtanied after the first convolutional layer, on the Komondor dog image.

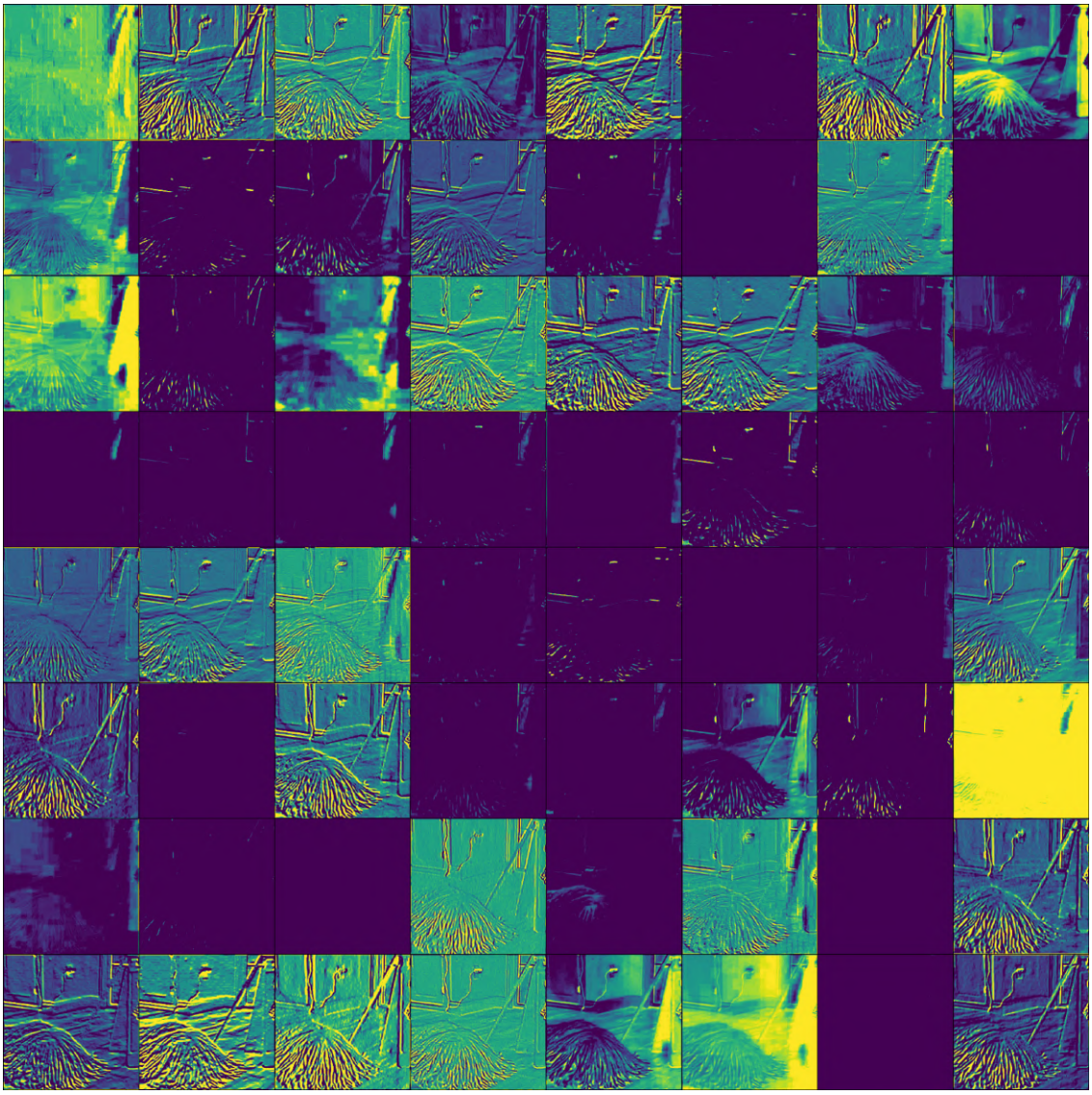


Figure 5.3: Activation maps obtanied after the first convolutional layer, on the "Komondor or mop?" image.

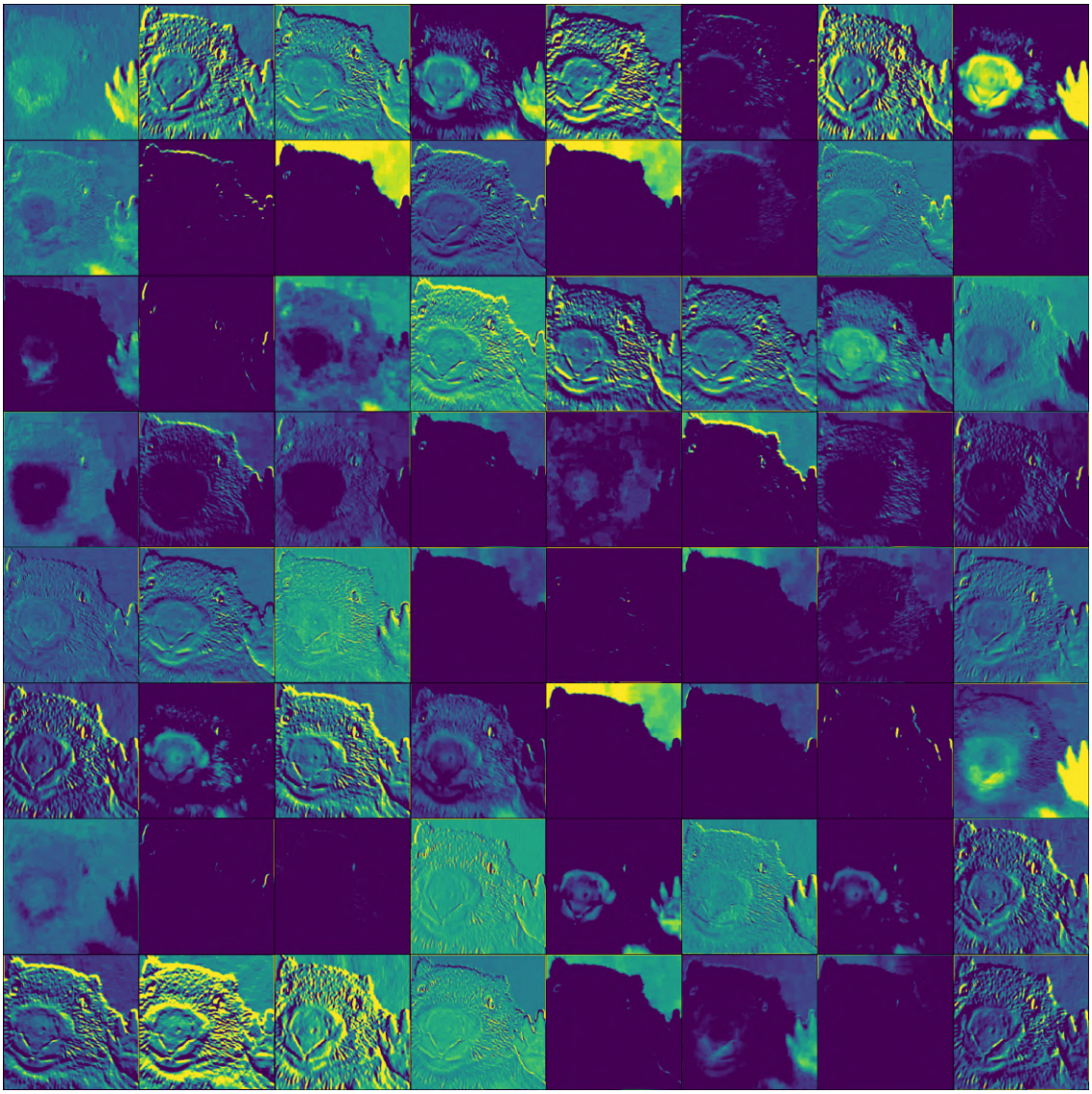


Figure 5.4: Activation maps obtanied after the first convolutional layer, on the wombat image.

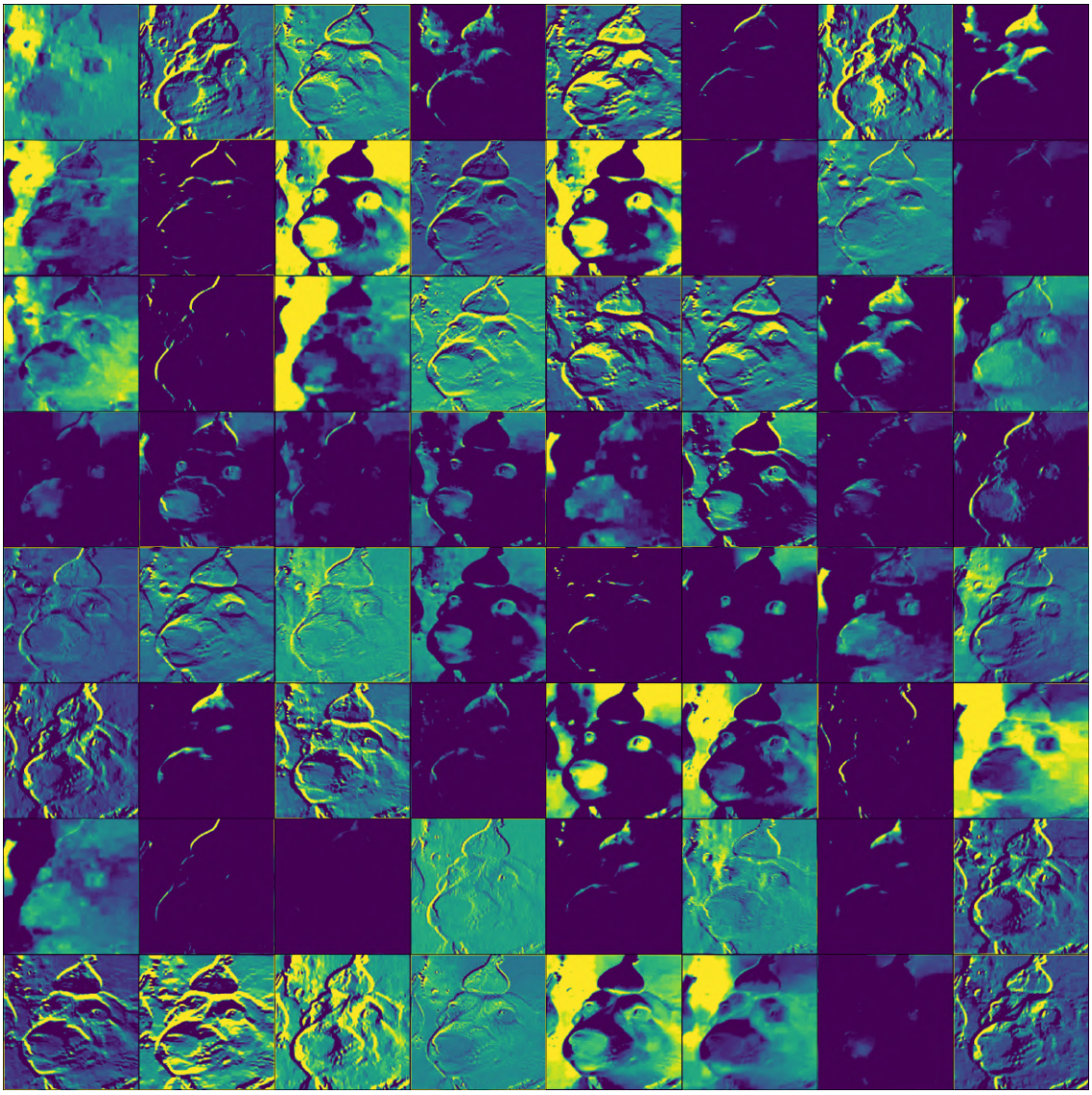


Figure 5.5: Activation maps obtanied after the first convolutional layer, on the dog (meme) image.

5.1.2 Going further: VisionTransformer

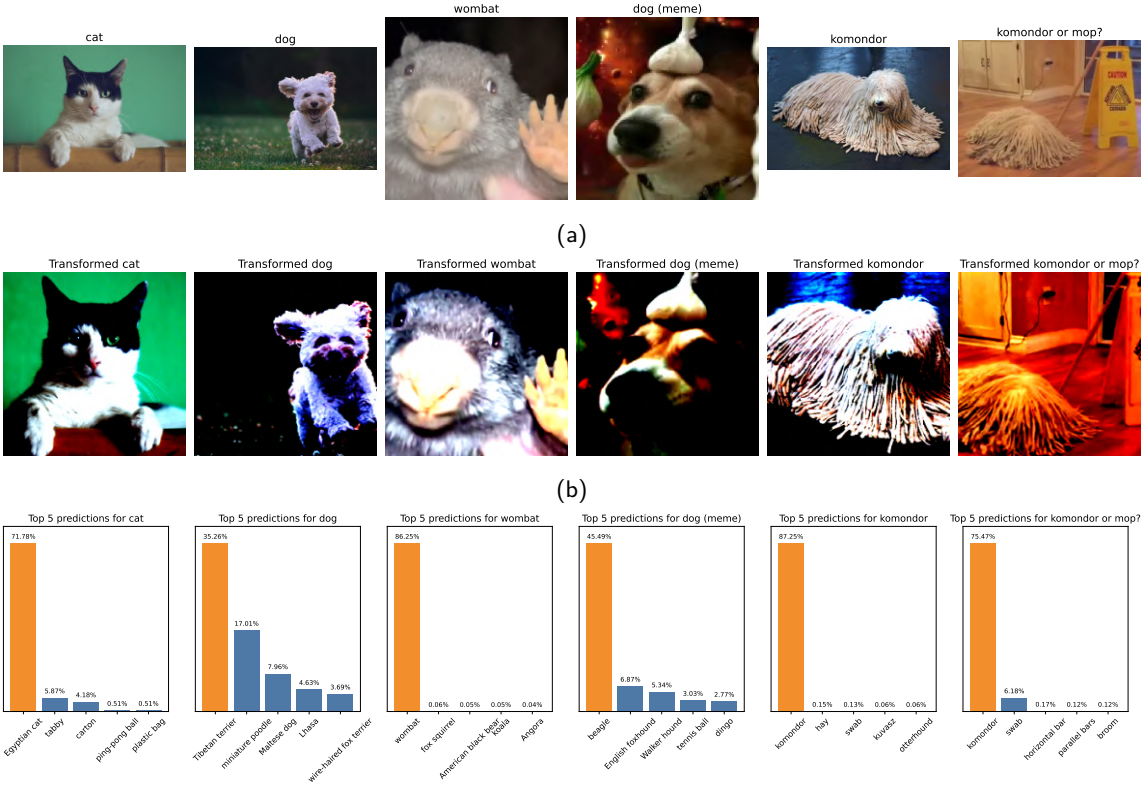


Figure 5.6: Performance evaluation of VisionTransformer pre-trained on ImageNet (IMAGENET1K_V1 weights): (a) Original images, (b) Transformed images normalized and resized to 224×224, and (c) Top 5 predicted classes.

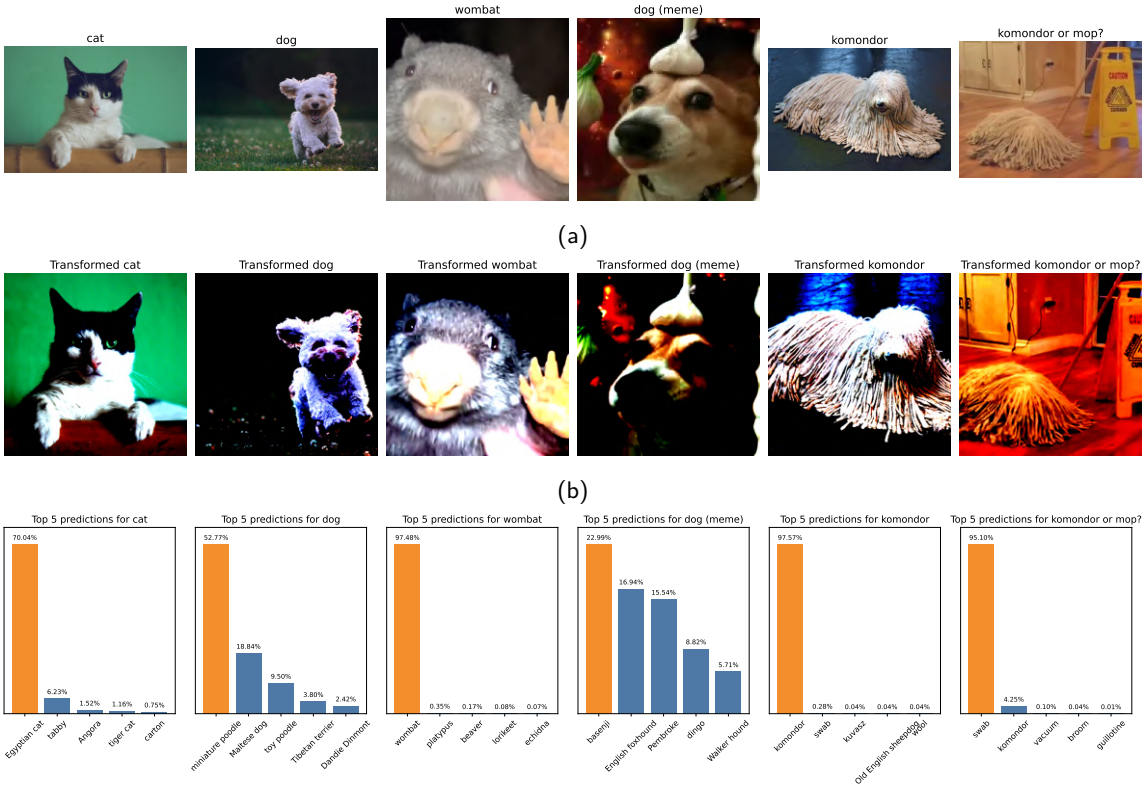


Figure 5.7: Performance evaluation of VisionTransformer pre-trained on ImageNet (IMAGENET1K_SWAG_E2E_V1 weights): (a) Original images, (b) Transformed images normalized and resized to 224×224, and (c) Top 5 predicted classes.

5.2 Generative Adversarial Networks

5.2.1 Longer epochs

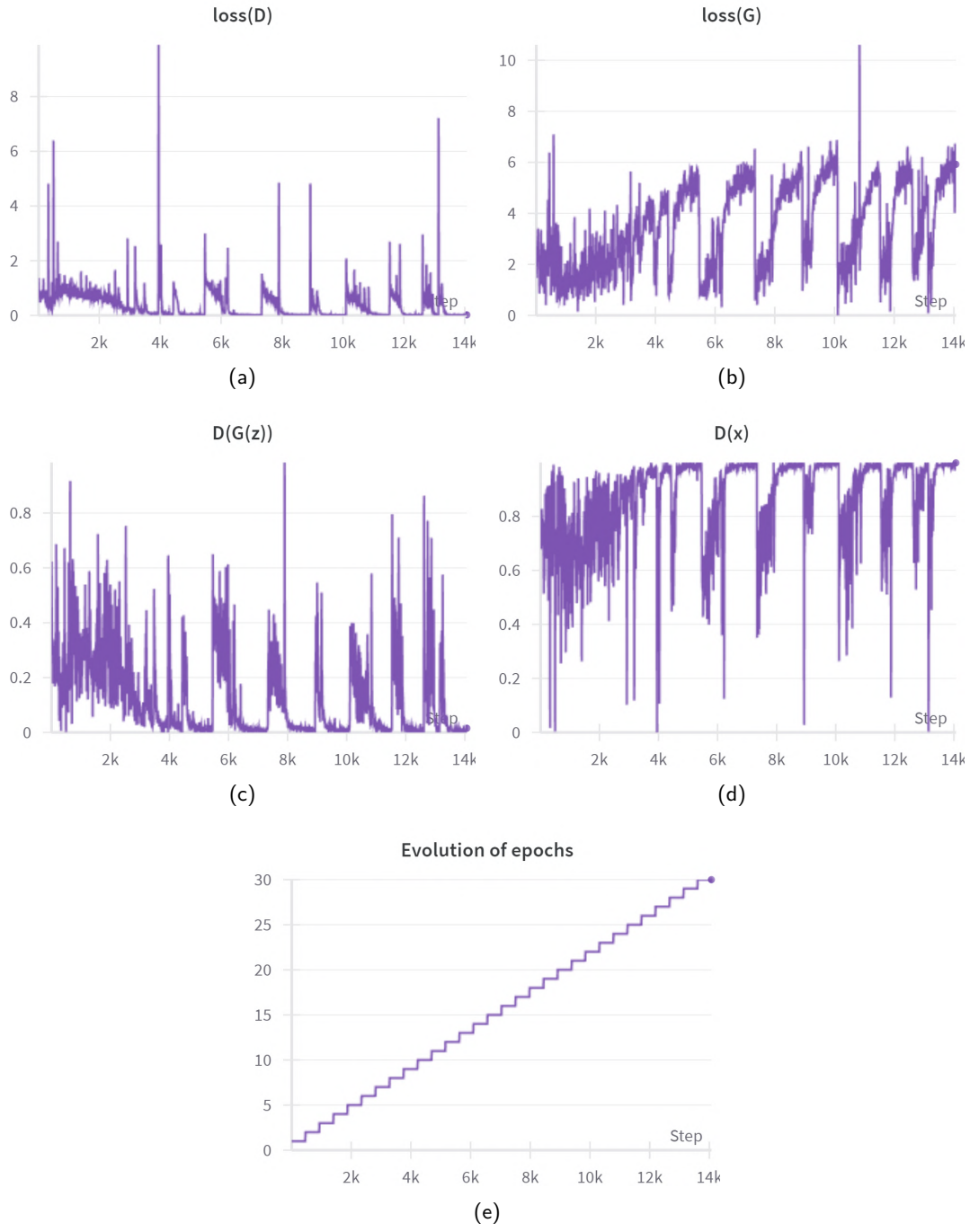


Figure 5.8: Training curves: (a) $loss(D)$, (b) $loss(G)$, (c) $D(G(z))$, (d) $D(x)$ and (e) the number of epochs by the number of iterations.

5.2.2 Influence of weight initialization.

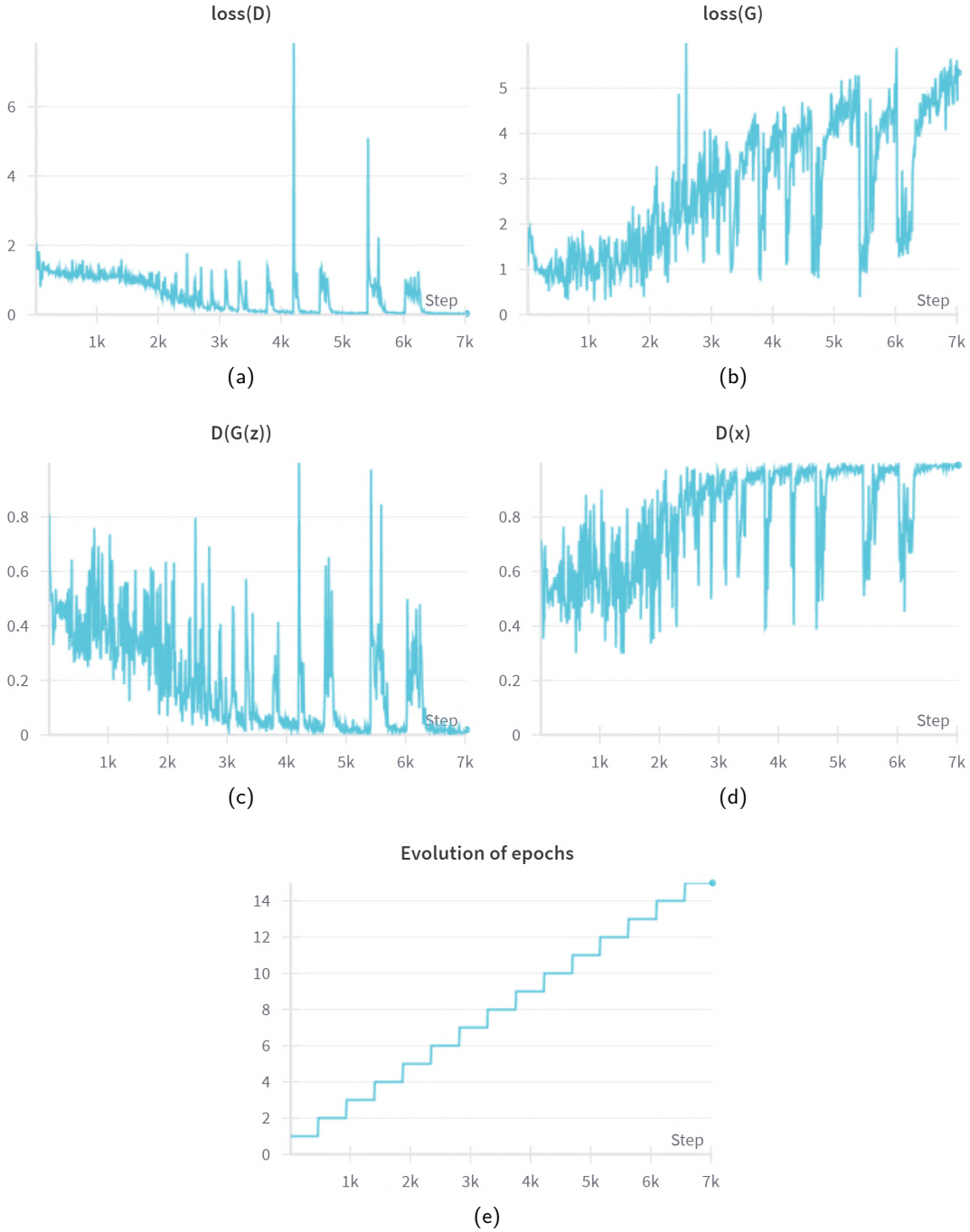


Figure 5.9: Training curves: (a) $loss(D)$, (b) $loss(G)$, (c) $D(G(z))$, (d) $D(x)$ and (e) the number of epochs by the number of iterations.

5.2.3 Influence of learning rate

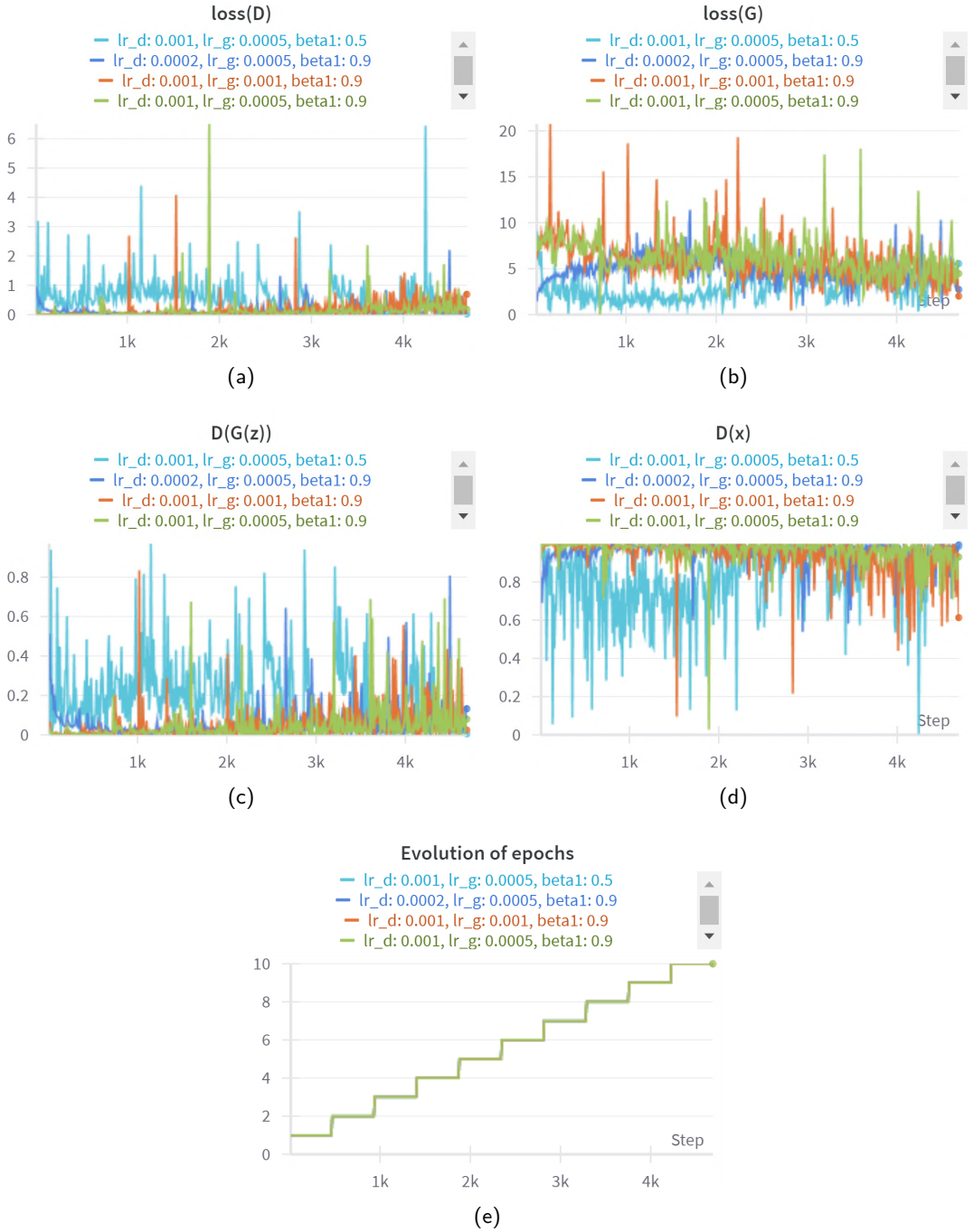


Figure 5.10: Training curves: (a) $loss(D)$, (b) $loss(G)$, (c) $D(G(z))$, (d) $D(x)$ and (e) the number of epochs by the number of iterations.

5.2.4 Influence of ngf and ndf

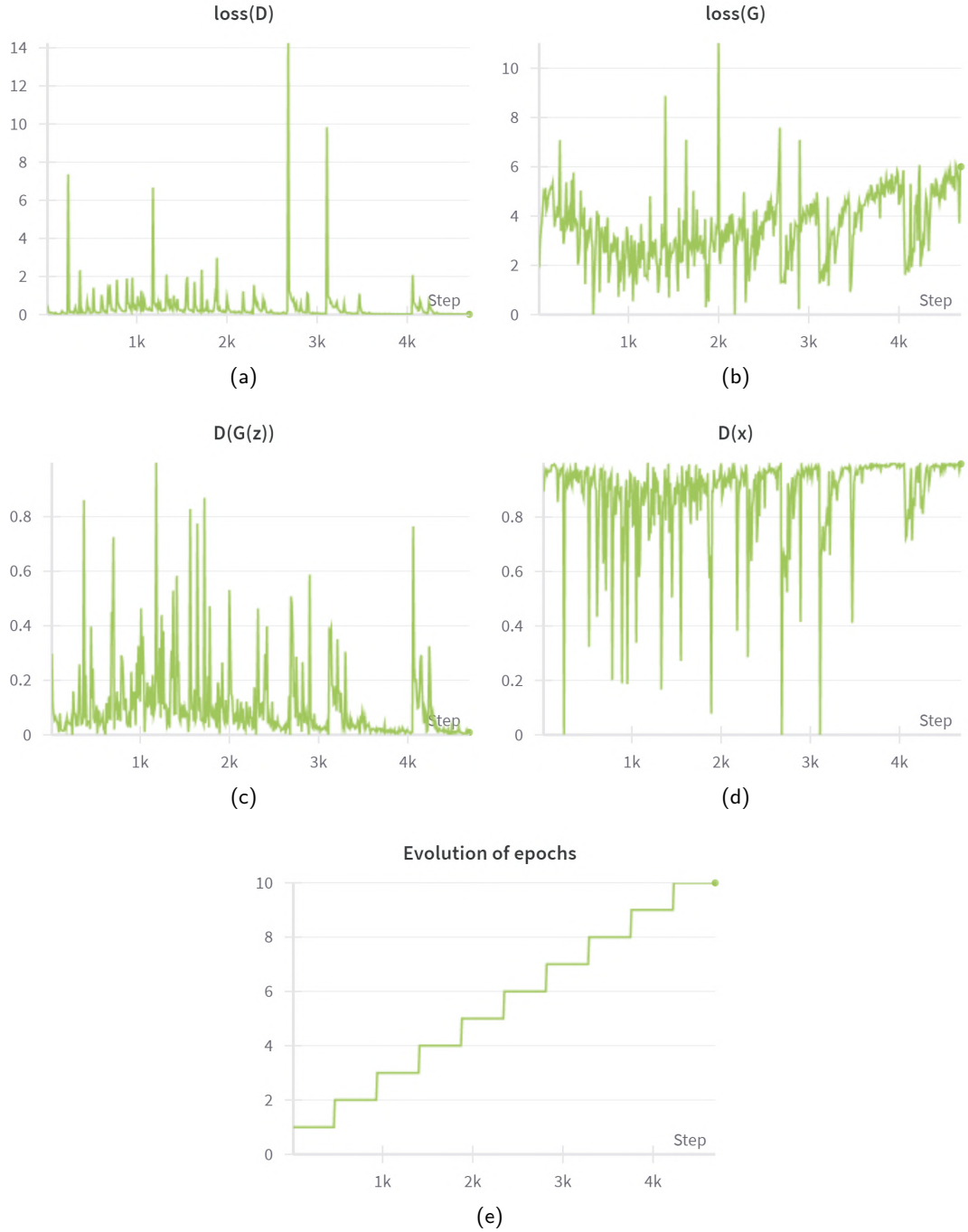


Figure 5.11: Training curves for $ngf = 8$: (a) $loss(D)$, (b) $loss(G)$, (c) $D(G(z))$, (d) $D(x)$ and (e) the number of epochs by the number of iterations.

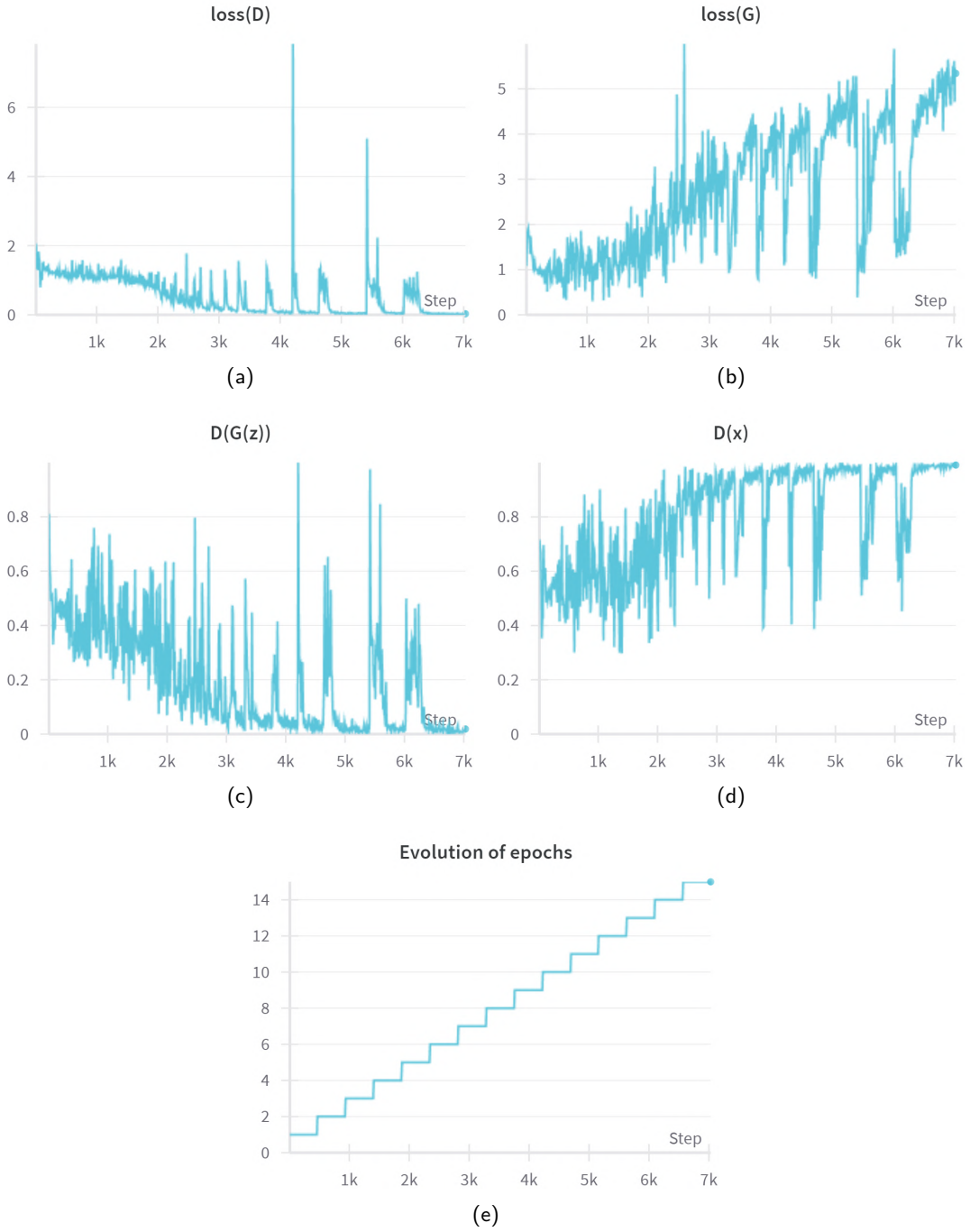


Figure 5.12: Training curves for $ngf = 128$: (a) $loss(D)$, (b) $loss(G)$, (c) $D(G(z))$, (d) $D(x)$ and (e) the number of epochs by the number of iterations.

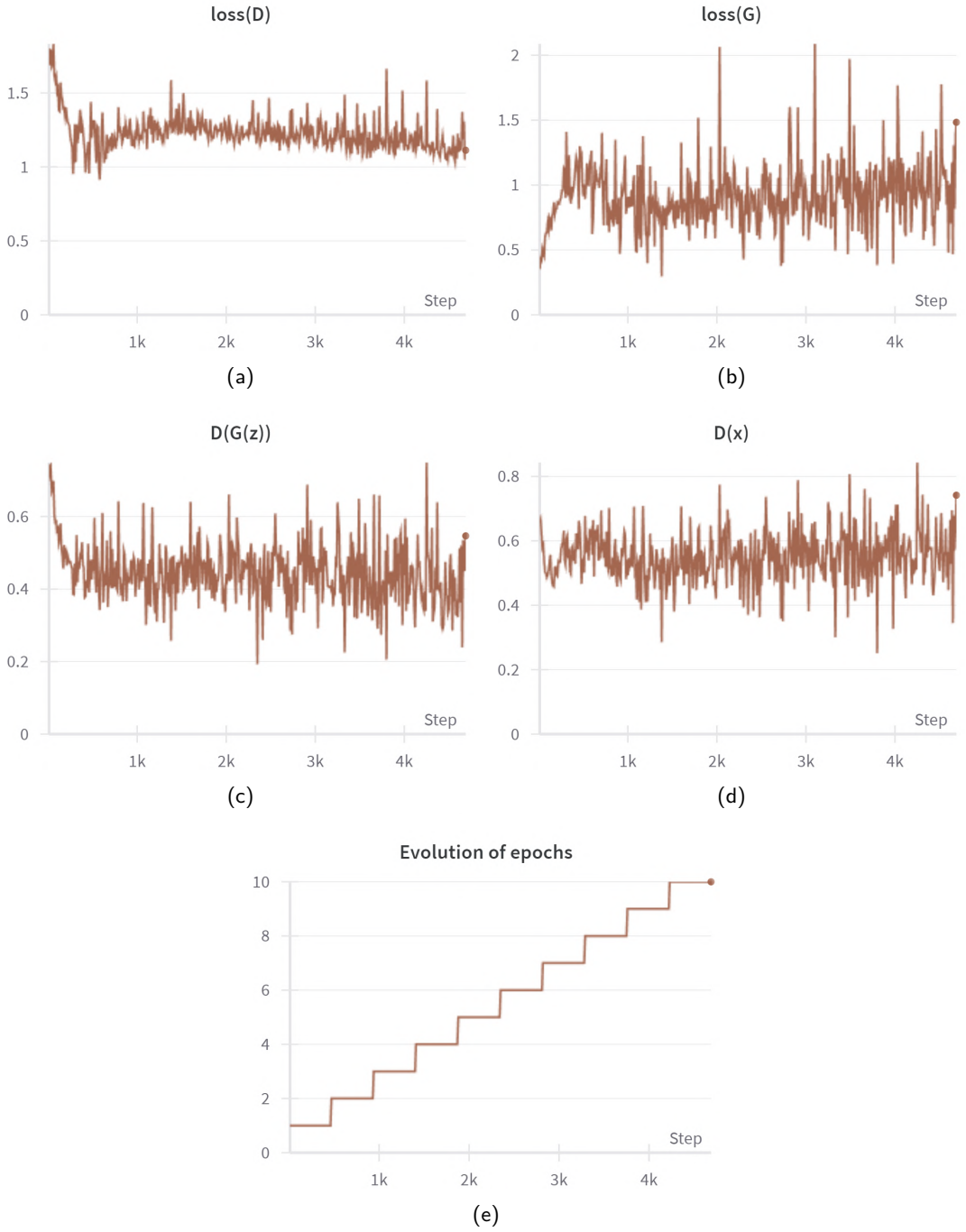


Figure 5.13: Training curves for $ndf = 8$: (a) $loss(D)$, (b) $loss(G)$, (c) $D(G(z))$, (d) $D(x)$ and (e) the number of epochs by the number of iterations.

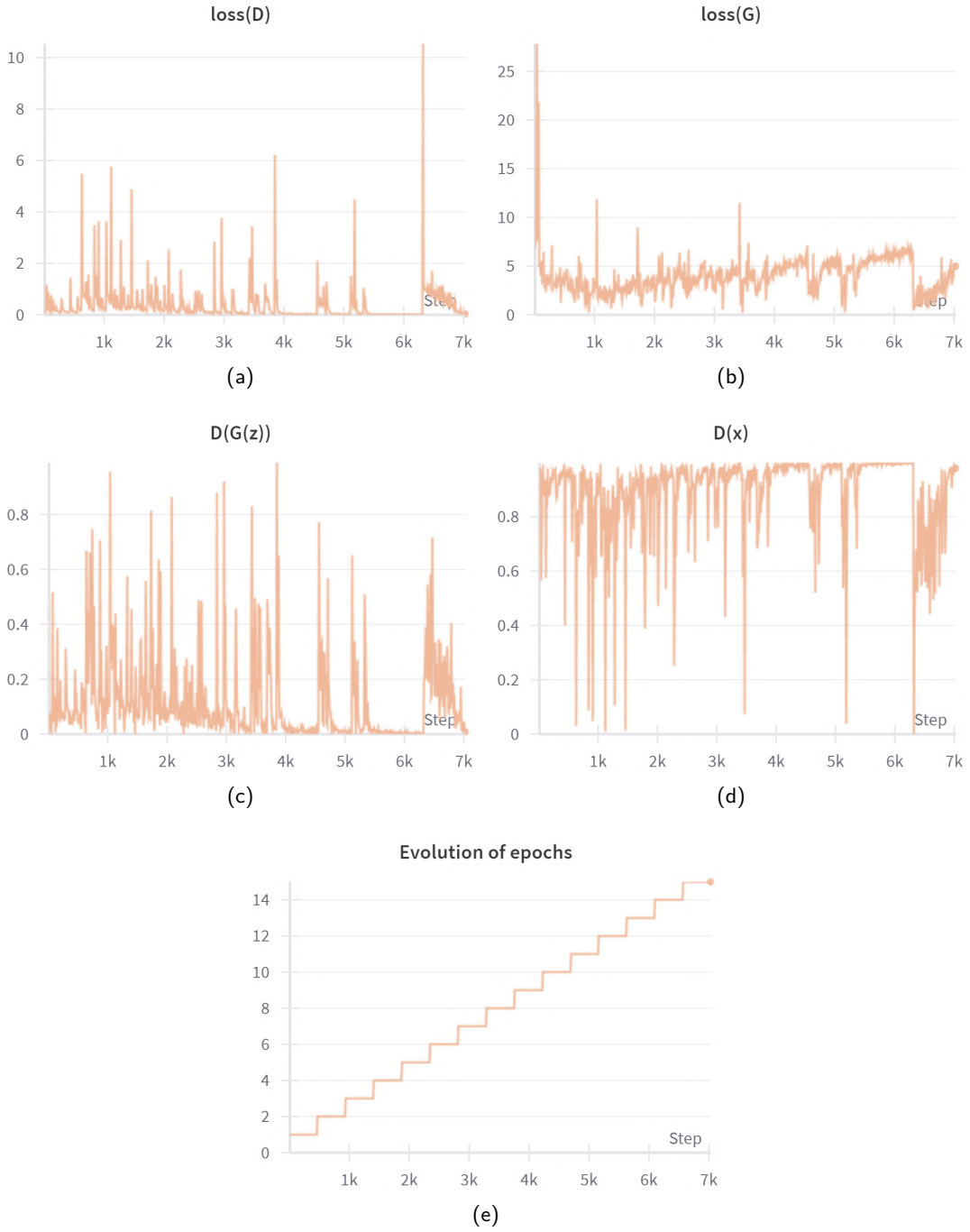


Figure 5.14: Training curves for $ndf = 128$: (a) $loss(D)$, (b) $loss(G)$, (c) $D(G(z))$, (d) $D(x)$ and (e) the number of epochs by the number of iterations.

5.2.5 Trying CIFAR-10 dataset

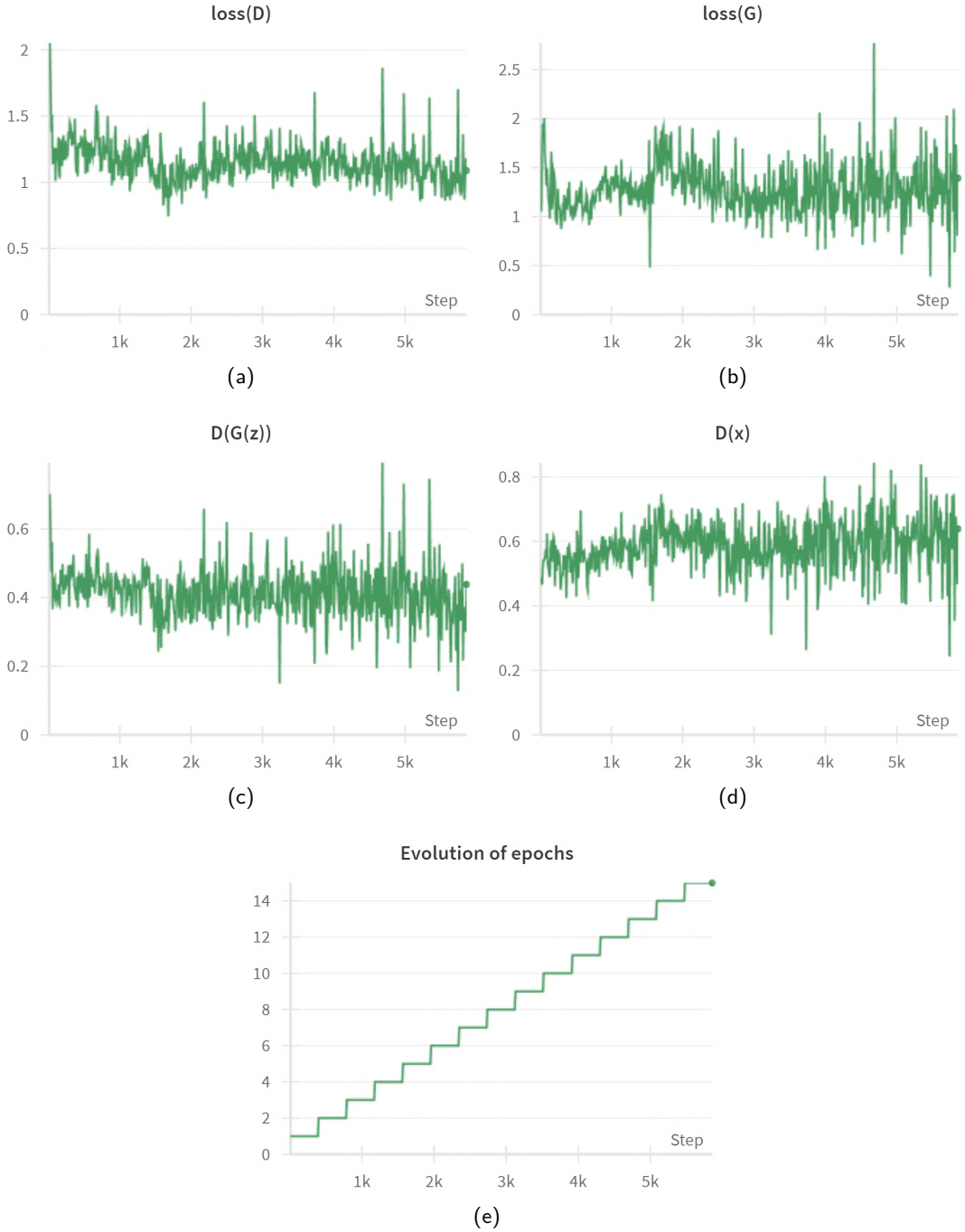


Figure 5.15: Training curves: (a) $loss(D)$, (b) $loss(G)$, (c) $D(G(z))$, (d) $D(x)$ and (e) the number of epochs by the number of iterations.

5.2.6 Generating higher resolution images on CIFAR-10

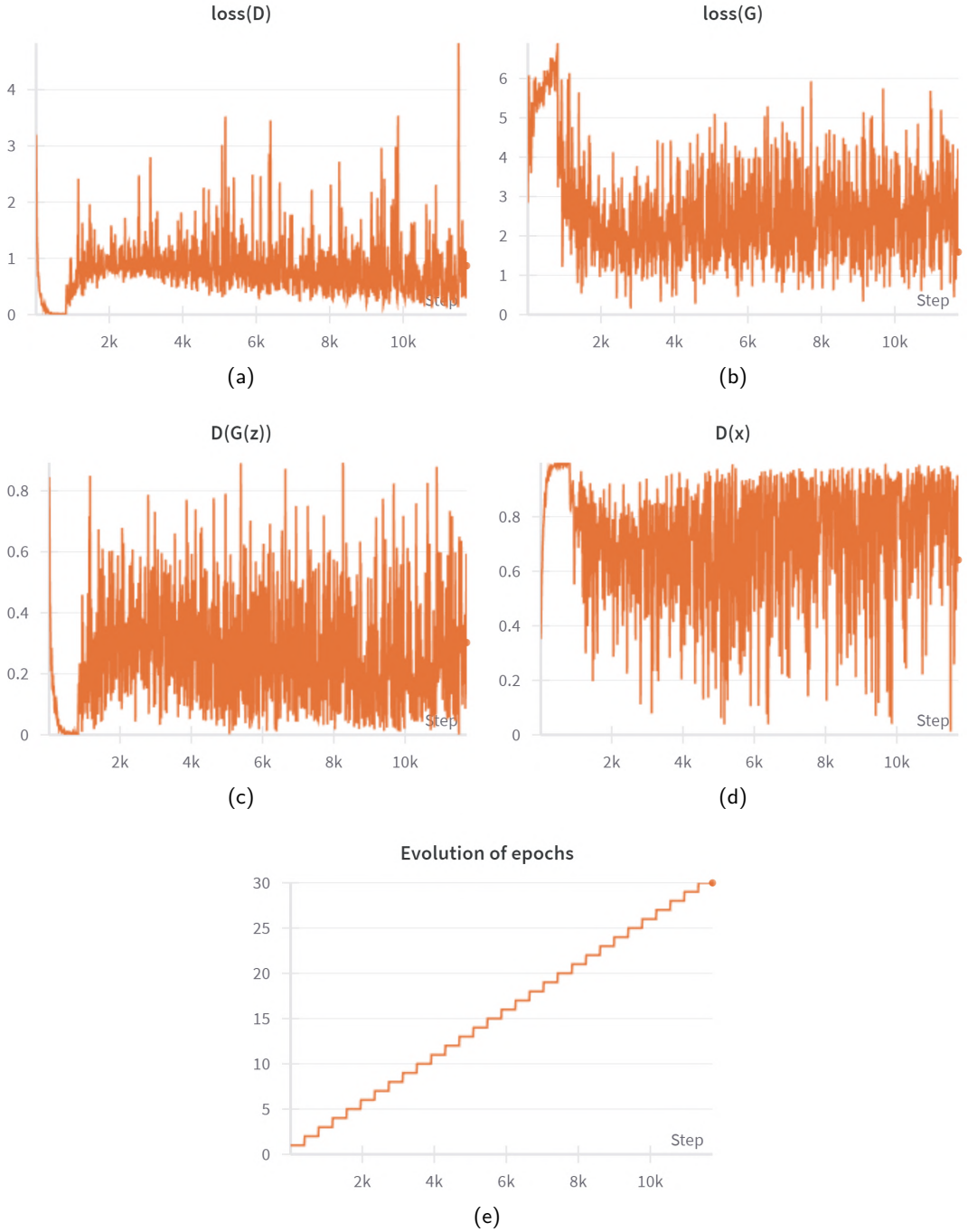


Figure 5.16: Training curves: (a) $loss(D)$, (b) $loss(G)$, (c) $D(G(z))$, (d) $D(x)$ and (e) the number of epochs by the number of iterations.

5.2.7 Conditional GAN

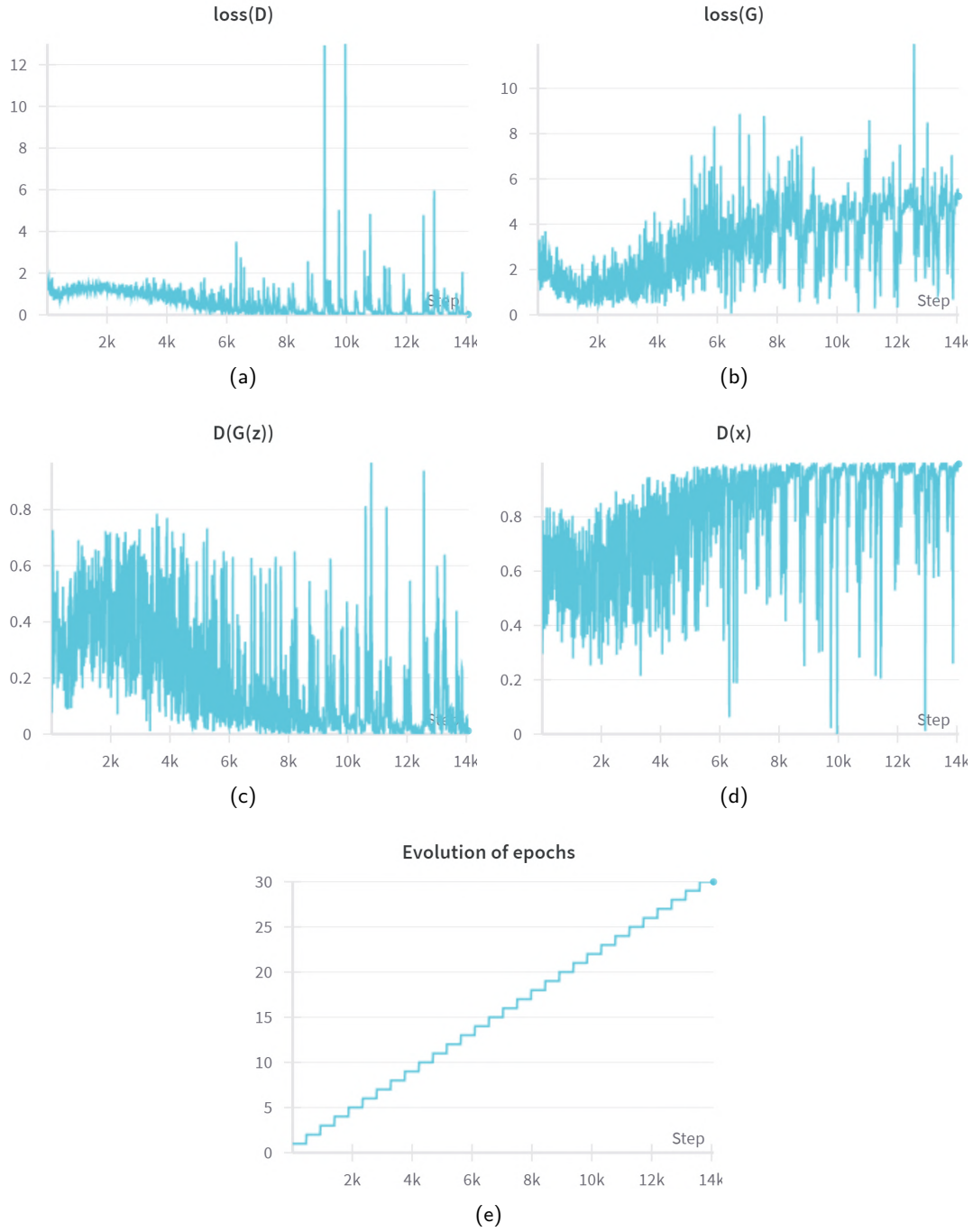


Figure 5.17: Training curves: (a) $loss(D)$, (b) $loss(G)$, (c) $D(G(z))$, (d) $D(x)$ and (e) the number of epochs by the number of iterations.

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.