RDFIA Homework
Part II

Argentieri Gabriele
Zattoni Francesco

January 13, 2026

Introduction

In this second report, we explore advanced deep learning techniques beyond basic classification. We begin by adapting the VGG16 architecture to new tasks using Transfer Learning, followed by an analysis of model interpretability and robustness through saliency maps and adversarial attacks. The study then extends to Unsupervised Domain Adaptation to handle distribution shifts, and concludes with the training of Generative Adversarial Networks (DCGANs) to synthesize realistic images and disentangle latent representations.

2-a: Transfer Learning

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 (using the sizes given in Figure 1).**

   Transitioning from the last max-pooling layer ($7 \times 7 \times 512$, flattened to 25088 inputs) to the first FC layer (4096) requires a matrix of size $25088 \times 4096 = 102.760.448$.

   Then this number gains an additional $4096 \times 4096 = 16.777.216$ weights from the second layer and $4096 \times 1000 = 4.096.000$ from the last one.

   The total number of parameters of VGG16 is not far from the amount accounted by the FC layers (123.633.664 parameters). In fact, these layers contribute for around $92,26\%$ of the total 134 millions parameters declared in the original paper of the model.

2. ★ **What is the output size of the last layer of VGG16? What does it correspond to?**

The output size is a vector of dimension 1,000 to which the Softmax activation function is applied, normalizing the output logits such that the sum of all class probabilities equals one, thus corresponding to a probability distribution over the 1,000 classes of ImageNet.

3. **★ Apply the network on several images of your choice and comment on the results.**

   The selected images for the exercise are:

Figure 1: Input images of a koala, a cat, a dog and a deer.

The VGG16 model's predictions are not very accurate: it correctly identified the 'cat', 'dog', and 'koala'; however, the breeds of the cat and the dog are incorrect, classifying them as 'egyptian cat' and 'lakeland terrier'. Also, the fourth image of the deer is classified as a dog, in particular an 'Ibizan hound'.

- **What is the role of the ImageNet normalization?**
  Normalization was used during the training of ImageNet to make the optimization smoother by giving it inputs from a smaller range and centered around zero. It is also required on the examples used during the test phase to ensure consistency, since the weights were tuned on data with certain statistical properties and it is used to see data of that type.
- **Why setting the model to eval mode?**
  The reason is that VGG uses dropout, which behaves differently at training and test time, so the `eval()` function is needed to switch to the evaluation phase.

4. **Bonus: Visualize several activation maps obtained after the first convolutional layer. How can we interpret them?**

   In Figure 2, the activation maps from the first convolutional layer show the different features learned by the model. For instance, some filters detect edges, others certain textures or even uniform regions of a certain color, as we can see in the two activation maps that highlight the green background of the cat image.

   Usually, we expect the first layer to learn low-level features, but actually some activation maps show more complex features, such as details of the subject in the foreground.

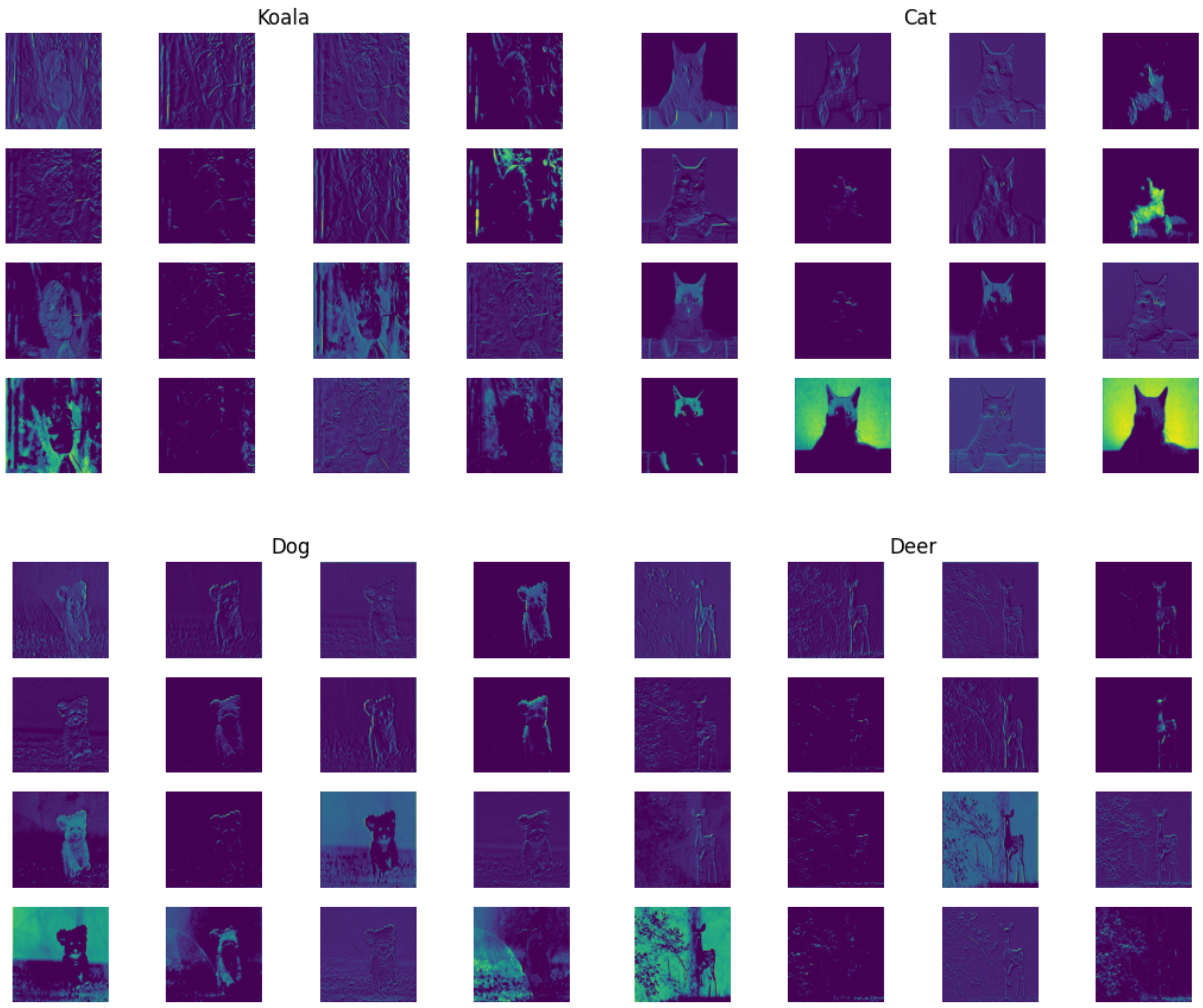

Figure 2: First 16 activation maps for each input.

5. ★ **Why not directly train VGG16 on 15 Scene?**

   Direct training of VGG16 on a small dataset such as *15 Scene* would lead to it memorizing the entire training set and consequently to overfit to it. It could even be disruptive, because it would update the weights' values learned on ImageNet in such a way that the model could forget what it learned in the first place, thus not being able anymore to classify correctly on more general domains.

6. ★ **How can pre-training on ImageNet help classification for 15 Scene?**

   The main idea behind using a pre-trained model is exploiting its general features learned in a broader domain, like ImageNet, to do classification on a much more restricted one. The effectiveness of this approach is proven by the activation maps shown in the previous question, where universal features learned on ImageNet, like edges in certain orientations, colors or textures, are useful for the target domain.

7. **What limits can you see with feature extraction?**

   As seen in our notebook output ("Features size: 25088"), extracting features from the final convolutional/pooling layers results in massive feature vectors ($512 \times 7 \times 7$). Training a classifier on such high-dimensional data with a small dataset can lead to overfittting. Also, the features are "frozen" and were learned for a specific task: while powerful, they are not optimized for scene recognition. And the network cannot adjust its weights to learn patterns unique to the target *15 Scene* dataset.

8. **What is the impact of the layer at which the features are extracted?**

   Early layers capture generic, low-level features like edges, colors, and textures. These are highly transferable to almost any visual task. Instead later layers capture high-level, semantic information (parts of objects or even objects). These are more specific to the original training data (ImageNet). With a different dataset, these specific features might not transfer as well.

9. **The images from 15 Scene are black and white, but VGG16 requires RGB images. How can we get around this problem?**

   To solve this problem it's possible to modify the input image adding two more channels by concatenating the same grayscale original image. In this way, the original input is not changed, but it is compliant to the model's requisites.

10. **Rather than training an independent classifier, is it possible to just use the neural network? Explain.**

    Yes, it is possible to use just the neural network. The feature extractor outputs a feature tensor in some feature latent space depending on the feature's dimensionality. Test features can be classified by checking the similarity of training set features, whose labels are known.

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

    The base case is a VGG16 without the final prediction head replaced by a linear SVM classifier, with regularization hyperparameter $C = 1.0$, that obtains an accuracy of 88.57%.

    - **Change the layer at which the features are extracted:** we checked the influence of the extraction layer on VGG16 pre-trained model in 2 different tests. Both extractions output features of high dimensionality (100352) causing a dramatic usage of RAM. Indeed, an additional test was made on features from layer -8, but their too high dimensionality lead to a session crash. The features extracted from the last layer of CNN part contain more semantic information than in previous layers, that can be beneficial in the case of 15 Scene dataset. The improvement to the base case can be explained by the fact that additional linear mappings to compress and transform linear data are not useful for our scenario.

  - **from last layer of CNN** leading to an accuracy of 88.74%.
  - **from layer -4 of CNN** leading to an accuracy of 88.17%.
- **Try other available pre-trained networks:** we tested 2 CNN pre-trained models by using the same feature extraction approach, i.e. before prediction head:
  - **ResNet18:** accuracy of 88.01%.
  - **AlexNet:** accuracy of 85.66%.
- **Tune the parameter C** that controls the regularization in SVM loss. The chosen values are from 10^{-3} to 10^{2} in a log scale and the accuracies are visible in figure 3. A proper regularization causes softer margins between classes preventing overfitting and in our case seems quite beneficial, since C is inversely proportional to the amount of regularization.

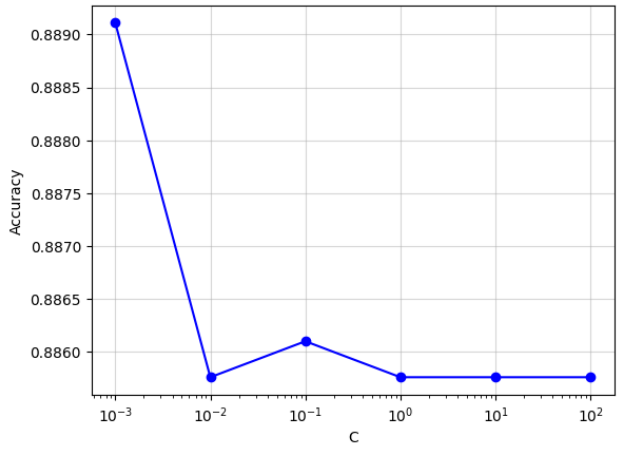

Figure 3: SVM accuracy with different Cs.

- **Replace the last layer of VGG16 with a new fully-connected layer:** new prediction head trained for 20 epochs optimized by SGD with momentum set to 0.9.
  - **Frozen backbone:** only new head's weights are updated gaining an accuracy of 82.21%.
  - **Fine-tuning:** backbone's weights are updated too but with a smaller learning rate of 10^{-4} leading to an accuracy of 80.63%. In this case, with such a restricted target domain fine-tune the whole network can decrease generalization capabilities. In general, SVM gets better results probably because the number of epochs is not large enough or because SVM's linear separation
- **Dimensionality reduction:** applied a PCA reduction to a dimensionality of 100 from the tensor of size 4096 before the prediction head.
  - **Without PCA** accuracy of 88.57% in 35s.

– **With PCA** accuracy of 81.34% in 35s. Probably the data compression removed some features beneficial for a correct classification without any time gaining, since most of the execution time was taken by the data loading and not by the SVM training.

2-b: Visualizing Neural Networks

1. **★ Show and interpret the obtained results.**

   The saliency maps computed from SqueezeNet can explain the model prediction and, in particular, which pixels contributed the most for the final classification. As we can notice, the majority of the pixels that belongs to the objects had an influence on the outcome, especially in animals' faces, like in the last two images. We could deduce that the model has learned to recognize faces, in the animal classification scenario.

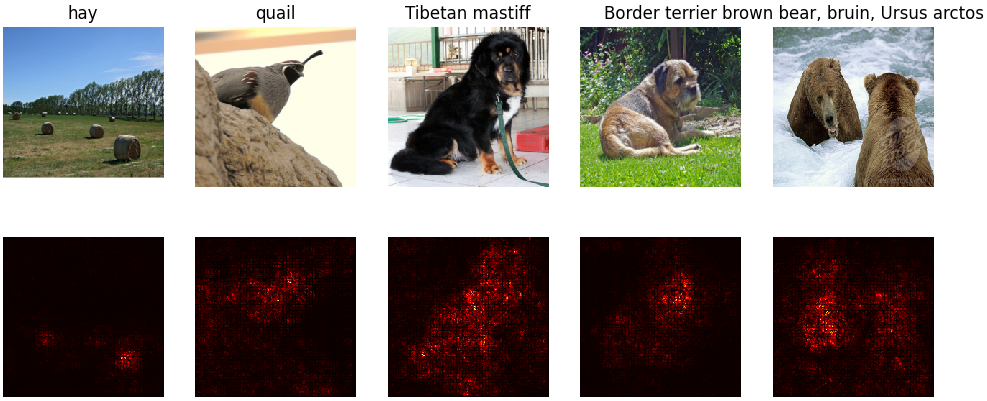

Figure 4: Saliency maps computed on 5 different test images with SqueezeNet.

2. **Discuss the limits of this technique of visualizing the impact of different pixels.**

   Each step of this technique can lead to limits on the results:

   - **Model approximation:** the paper proposed to approximate the model, composed by many layers and many non-linear transformations to a simple linear transformation.
   - **Ignoring color information:** if the model was trained on a RGB image, then every input channel can have an influence on the prediction, that is ignored by taking the maximum value of the 3 channels.

3. **Can this technique be used for a different purpose than interpreting the network?**

   The idea behind this technique is indeed applied to generate adversarial examples by modifying slightly the pixels of an image to get a misclassification. For each pixel, the

amount of change is proportional to the influence of the pixel itself on the wrong class, by computing the gradient $g = \frac{\partial \tilde{y}_j}{\partial x}$. If we took the maximum value of the 3 channels, we would have a saliency map for the wrong class j.

4. **Test with a different network, for example VGG16, and comment.**

   The saliency maps computed from a pre-trained VGG16 model, in figure 5[!hs], show some differences compared to the saliency maps in figure 4: the heat maps have larger values in magnitude, noticeable by very bright pixels in key regions of the images, and the silhouettes are much more precise, clearly delimiting the identified predicted objects. For instance, the Tibetan mastiff's saliency map in SqueezeNet had some active pixels on the right that do not belong to the dog. These results can be explained by looking at the number of parameters in the two models: VGG16 has 100 times more weights than SqueezeNet, allowing finer internal representation of the object.

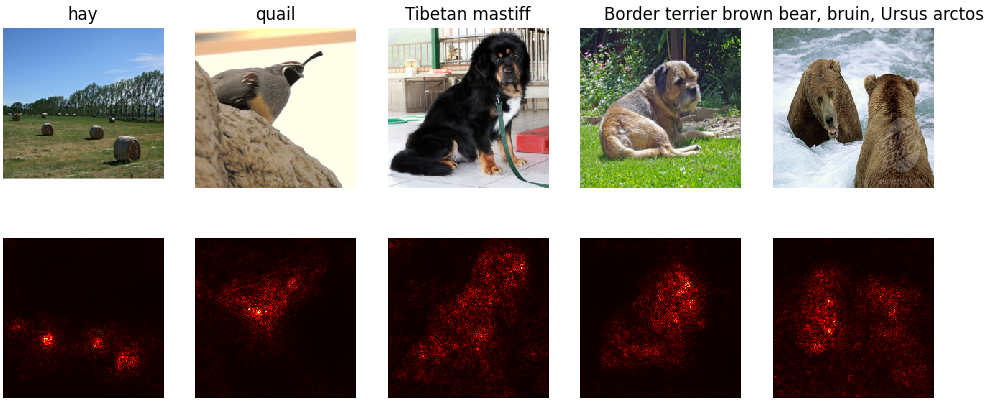

Figure 5: Saliency maps computed on 5 different test images with VGG16.

5. ★ **Show and interpret the obtained results.**

   As shown in Figure 6, adversarial examples are created by modifying a correctly classified input image only slightly (almost imperceptibly).

   - **Gradient-Based Direction:** This effect is achieved by iteratively updating pixel values in the direction of the gradient that increases the score of a target (incorrect) class.
   - **Pixels Influence:** The pixels modified with the greatest magnitude are those most influential for the target classification; these changes effectively "push" the image across the network's decision boundary.

6. **In practice, what consequences can this method have when using convolutional neural networks?**

   This method highlights a critical vulnerability in CNNs: while they may appear robust on standard datasets, issues arise when we apply targeted, non-random perturbations.

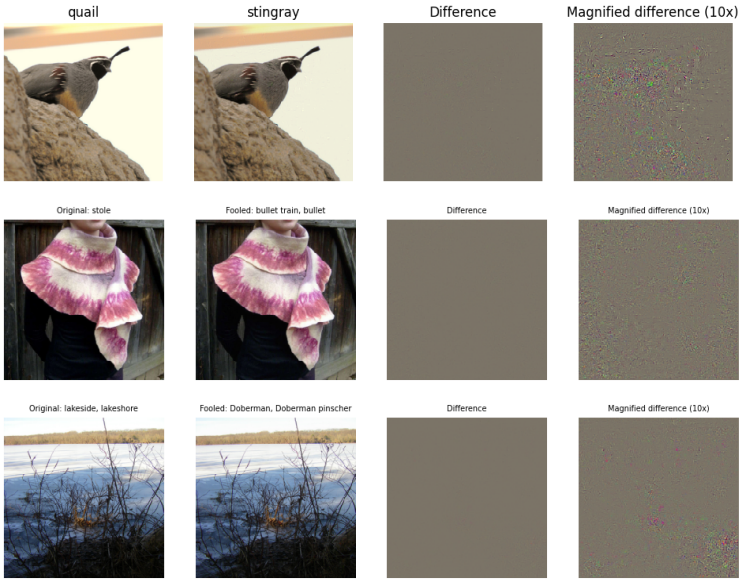

Figure 6: Adversarial examples (fooling images)

The main consequence is an alarming security risk in real applications. Since the adversarial modifications are imperceptible to humans, malicious actors could manipulate inputs (such as traffic signs for autonomous vehicles or images for facial recognition systems) to deceive the model without detection. This demonstrates that high accuracy on a test set does not guarantee robustness against malicious attacks.

7. **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).**

   The main limits of this technique are:

   - **Pixel Value Overflow:** Simply adding values to raw pixel could lead to go beyond the valid range of intensity.
   - **Computational Inefficiency:** The iterative nature of the method could make it significantly slow if the initial image is far from convergence.

   To address these limits, research has introduced more robust and efficient alternatives:

   - **Fast Gradient Sign Method (FGSM):** Proposed by Goodfellow et al. (2014), this is a one-step method that uses the sign of the gradient to ensure every pixel is changed by a fixed small amount ϵ. It is defined as $x_{adv} = x + \epsilon \cdot \text{sign}(\nabla_x)$.
   - **DeepFool:** An iterative approach that calculates the minimal distance required to push an image across the model's decision boundary. It produces much smaller, more imperceptible perturbations than Szegedy proposed method.

8. ★ **Show and interpret the obtained results.**

   Starting from random noise and iteratively updating the image based on the gradient $\nabla_x \mathcal{L}$, the process extracts the fundamental patterns that the model associates with a specific class. The following observations can be made from the results in figure 7:

   - **Emergence of Features:** The optimization reveals "canonical" patterns, such as animal heads, eyes, or specific textures, which are the most influential for model prediction. For instance, we can notice the hourglass symmetry and the complex fur patterns of a Yorkshire Terrier.
   - **Spatial and Frequency Correlation:** The density and positioning of these patterns are directly related to their frequency within the ImageNet training dataset.
   - **Necessity of Regularization:** Without the application of L_2 norm regularization and periodic blurring, the image would be dominated by high-frequency noise that is mathematically optimal for the score but semantically meaningless to humans.

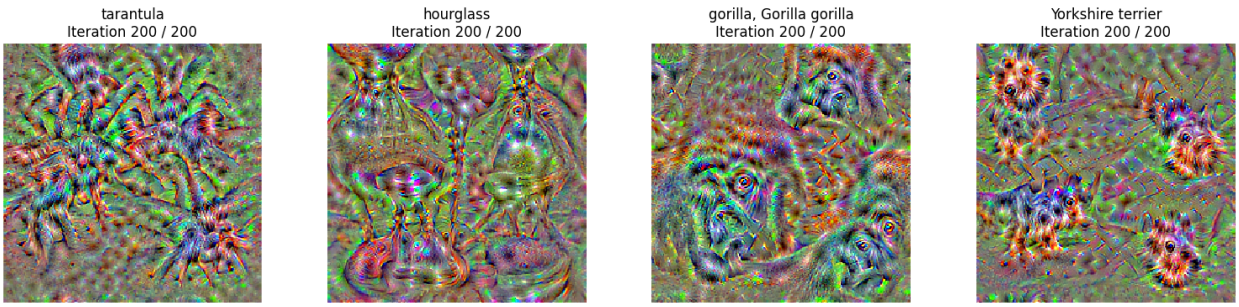

Figure 7: Visualization of 4 different classes.

9. **Try to vary the number of iterations and the learning rate as well as the regularization weight.**

   Following the experiments conducted on the Yorkshire Terrier class, we observed the following behaviors:

   - **Number of Iterations:** Increasing the number of iterations seems to improve the quality of the generated image. At 300 iterations, the features are still emerging from the noise. By increasing the count to 700, the results is sharper and has more recognizable features (e.g., the texture of the fur and the eyes of the dog become clearly visible).
   - **Learning Rate:** In the second experiment we test the impact of the step size η.
     - With a **low learning rate (3)**, the image remains blurry after 200 iterations because the updates are too small to drive the pixel values far enough from the initial noise.

  - A **moderate learning rate (7)** offers the best trade-off, producing clear patterns without excessive noise.
  - A **high learning rate (10)** accelerates convergence but introduces high-frequency noise and saturation, making the image look harsh and overly contrasted.

- **L2 Regularization (λ):** The regularization term $-\lambda||x||_2$ plays a crucial role in suppressing noise and keeping pixel values within a natural range.
  - **High regularization ($\lambda = 1$):** The penalty is too strong. The optimization prioritizes minimizing the norm $||x||_2$ over maximizing the class score, resulting in a mostly grey, featureless image.
  - **Balanced regularization ($\lambda = 10^{-5}$):** This value produces the most visually appealing result. It allows the features to emerge while preventing pixel values from exploding.
  - **Low regularization ($\lambda = 10^{-7}$):** The penalty is negligible. The pixels grow without bound to maximize the score, resulting in an image dominated by high-contrast, psychedelic colors and noise, losing the natural appearance of the texture.

Figure 8: Test with growing number of iterations

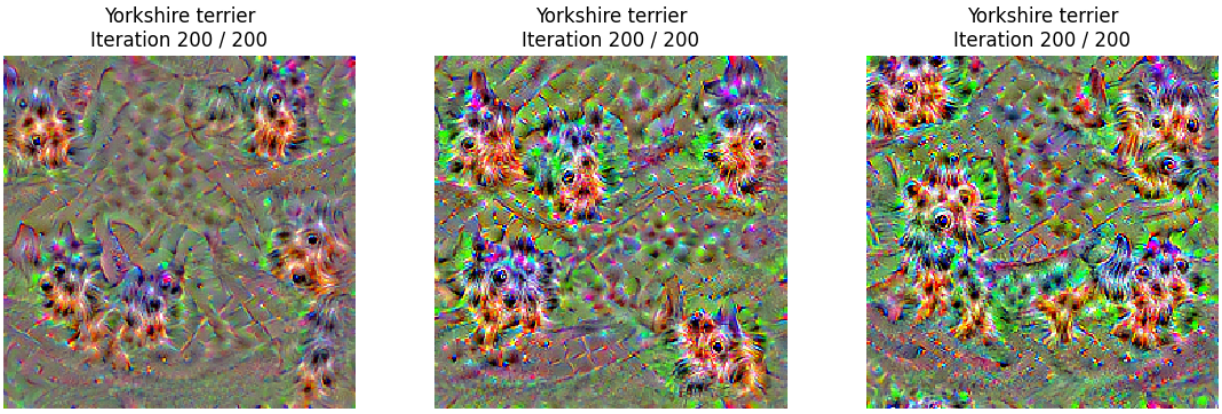

Figure 9: Test with learning rate 3, 7, 10

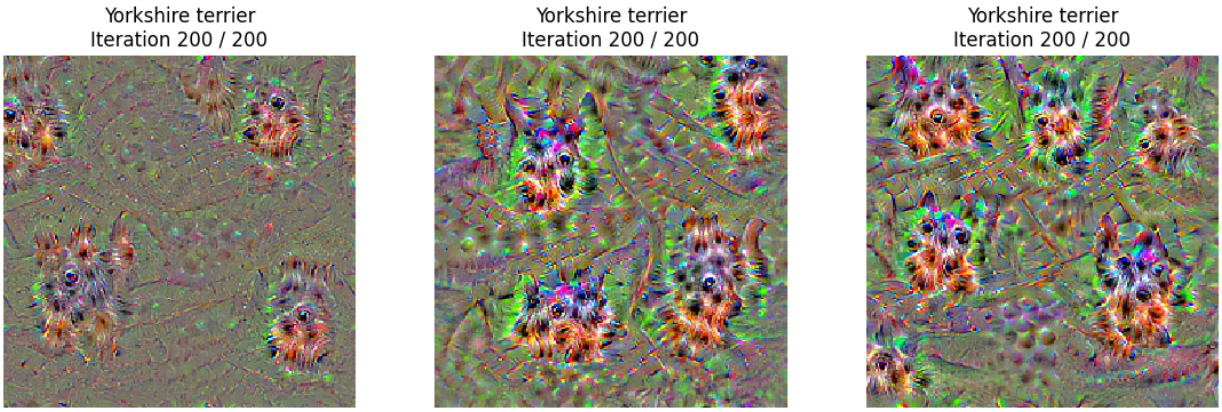

Figure 10: Test with regularization weight 1, 1e-5, 1e-7

10. **Try to use an image from ImageNet as the source image instead of a random image (parameter `init_img`). You can use the real class as the target class. Comment on the interest of doing this.**

    Using an image from ImageNet as the source instead of random noise yields significantly more structured and interpretable visualizations. For every iteration, it updates the image but keeping some details and the overall structure of the starting image. For example, in figure 11 the pan with the pancakes became a ship seen from the front or the duck in the right bottom corner became a sea lion.

    In the second test, shown in figure 12, it enhanced the details of the already present object or it added patterns typical of that class. For instance, it added new pirate ships near the starting one, which is still clearly visible and it added hays all over the last image.

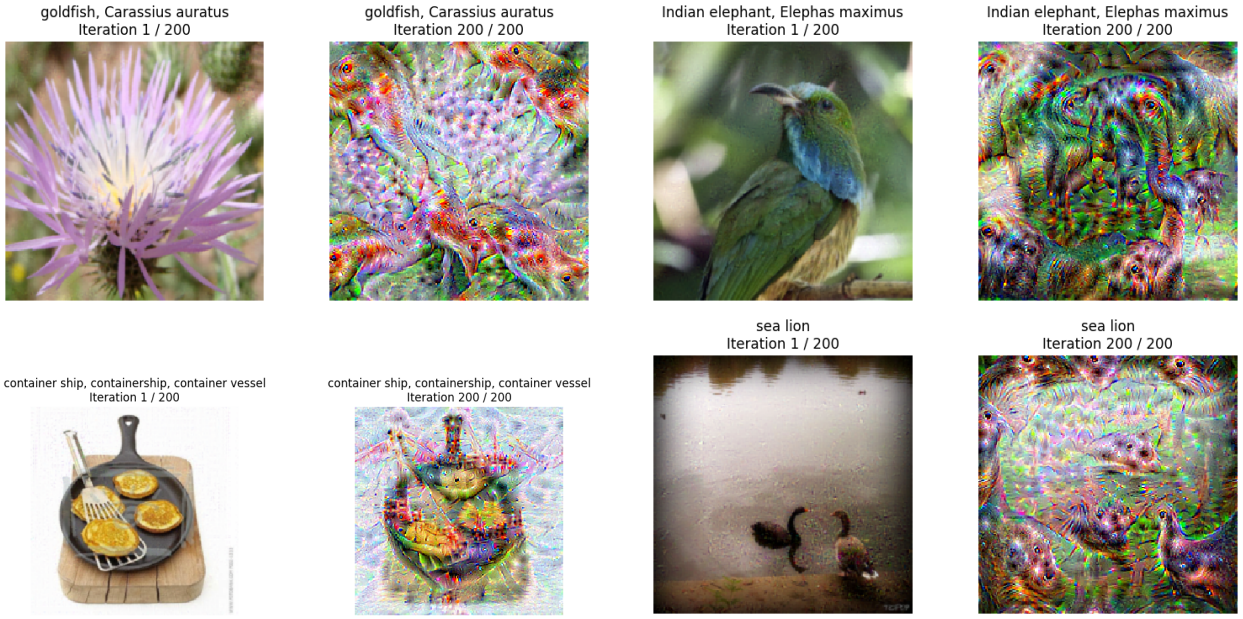

Figure 11: Visualization of 4 different classes from a random image taken from ImageNet.

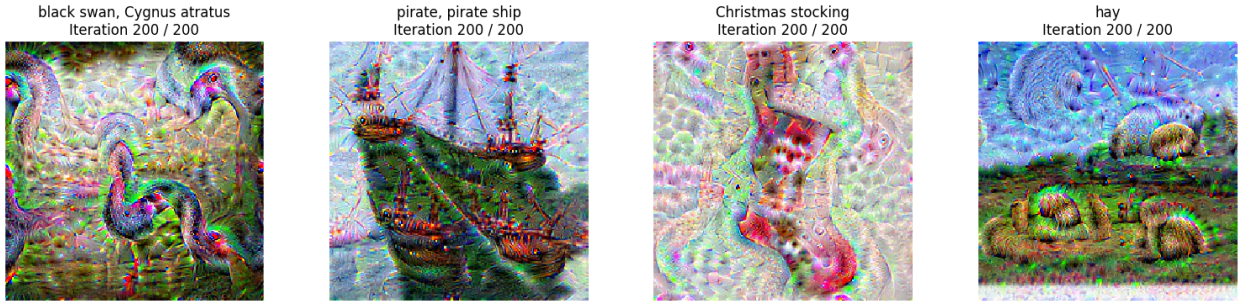

Figure 12: Visualization of 4 different classes from an image of that class.

11. **Test with another network, VGG16, for example, and comment on the results.**

    Testing with a deeper architecture like VGG16, the class visualizations, in figure 13, show significant differences compared to SqueezeNet's results.

    - **Feature Complexity**: VGG16 visualizations contain more intricate, repetitive, and higher-level geometric patterns, like the complex eyes and legs in the tarantula or the distinct textures of the Yorkshire terrier.
    - **Robustness**: It produces visualizations that are more visually "dense" and resistant to noise, making very hard to distinguish different instances of the class, as it was possible to do in figure 7.

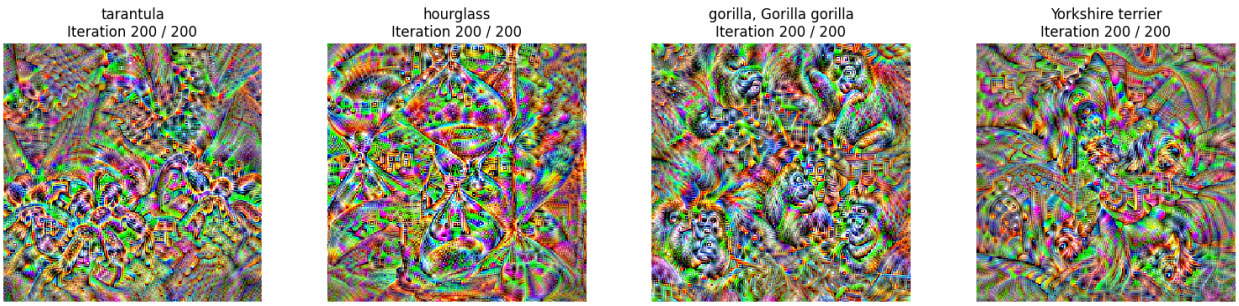

Figure 13: Visualization of 4 different classes with VGG16.

2-c: Domain Adaptation

1. **If you keep the network with the three parts (green, blue, pink) but didn't use the GRL, what would happen?**

   If we did not utilize the GRL, the network would behave as usual, minimizing both the label prediction loss and the domain classification loss. Thus the feature extractor would learn to produce features that are easily distinguishable by domain, leading to a separation of the source from the target distributions instead of aligning them. This contradicts the goal of Domain Adaptation, which requires domain-invariant features.

2. **Why does the performance on the source dataset may degrade a bit?**

   Performance on the source dataset may degrade because the model is forced to ignore discriminative features that are specific to the source domain in order to satisfy domain-invariance. Ideally, the model should only use features shared across domains; if the source task relied heavily on source-specific details, ignoring them hurts the performance. This is a perfect example of the specificity-generalization tradeoff that can be also observed in overfitting models. Instead of descending toward a local minimum, the optimization process now is guided to a saddle point. At this point, the loss is minimized with respect to the task classifier's parameters but maximized with respect to the domain discriminator's parameters.

3. **Discuss the influence of the value of the negative number used to reverse the gradient in the GRL.**

   The parameter λ controls the trade-off between the influence of the label predictor and that of the domain classifier.

   As suggested by the authors of the original paper, λ should not be fixed: at the start of training, in fact, the domain classifier is untrained and produces noisy gradients and backpropagating these noisy signals can have a bad influence on feature learning.

   A lower λ suppresses noisy signals from the domain classifier at the early stages of the training procedure, while an higher λ gives more importance to the domain classifier, thus steering towards shared features across the two domains.

   Therefore, λ is typically initialized at 0 and gradually increased to 1 to allow the domain classifier to converge before it starts influencing the feature extractor.

4. **Another common method in domain adaptation is pseudo-labeling. Investigate what it is and describe it in your own words.**

   Pseudo-labeling is a semi-supervised learning strategy where the model utilizes its own high-confidence predictions to label the unlabeled target data. The core idea is that the model effectively "teaches itself" about the target domain.The process is typically structured in the following steps:

   (a) **Initial Training:** A classifier is first trained using the available labeled data from the source domain.

   (b) **Prediction (Inference):** This trained model is then used to classify all the unlabeled data points belonging to the target domain.

   (c) **Selection and Labeling:** The predictions where the model exhibits high confidence (e.g., those exceeding a set probability threshold, such as $P > 0.9$) are selected. The predicted class is then assigned as the "pseudo-label" for that target sample.

   (d) **Retraining (Self-Training):** The model is retrained or fine-tuned using the combination of the original labeled source data and the newly created pseudo-labeled target data.

   By treating high-confidence predictions as temporary ground truth, the model adapts its feature representation and decision boundaries to better fit the target domain's structure. The main risk associated with pseudo-labeling is error propagation (or confirmation bias): if the model confidently assigns the wrong class to a target sample during the initial prediction phase, the subsequent retraining step will reinforce this incorrect label, potentially degrading the model's overall performance.

2-de: Generative Adversarial Networks

1. **Interpret the equations (6) and (7). What would happen if we only used one of the two?**

   If we only used one of the two equations, the result would be incomplete. If the discriminator was not optimized, the generator would not be encouraged to produce images resembling real data, since it would be trivial to fool an untrained discriminator. Vice versa, if the generator was fixed or trivial, the discriminator would easily distinguish fake images from real ones, preventing any meaningful learning process.

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

   It should transform $P(z)$ to $P(X)$, so the distribution of real data from where we sampled $Data$.

3. **Remark that the equation (6) is not directly derived from the equation 5. This is justified by the authors to obtain more stable training and avoid the saturation of gradients. What should the "true" equation be here?**

   As mentioned in the original paper, the gradient for G derived from equation (5) would not be sufficiently efficient especially in the earlier epochs of the training, because D could recognize artificial images with high confidence since G is not yet good enough to generate convincing results. This effect is also visible in figure 14: when the discriminator outputs with high confidence a value close to zero, the true formula derived from equation (5) would have a very small gradient, leading to very slow learning; instead, equation (6) is very steep, making learning in the earlier epochs much faster.

   The true equation should be:

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

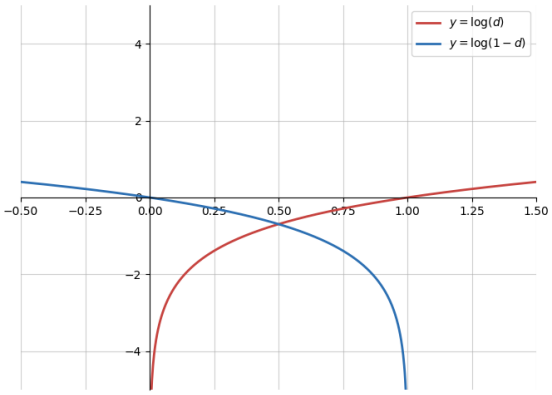

Figure 14: Comparison of equation (5) and (6), with $d = D(G(z))$

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

   GAN's training with the default settings (5 epochs, $ngf = 32$, $ndf = 32$, $n_z = 100$) lead to the following results in figure 15. The generated test sample has a large variety of digits that resemble the real ones, even if most of them are clearly fake. The loss plot has an oscillating nature caused by the alternating update of the two models. Discriminator's final average scores are 0.75 for real images and 0.02 for fake ones.

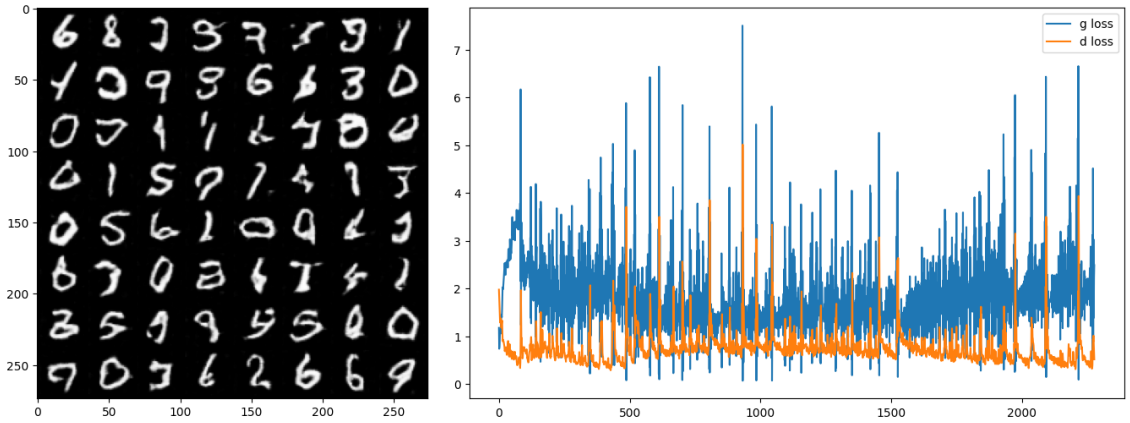

Figure 15: Generator's sample and training loss.

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.**

   - **Reduce or Increase ndf or ngf:** by tuning the hyperparameter that controls the hidden complexity of one of the two models, the training is not as balanced as the default one:
     - $ngf = 64$: discriminator's loss is higher and its scores are 0.67 and 0.16, for real and fake images, even if the generated images are not better than before, as shown in figure 16.
     - $ngf = 8$: generator's loss is much higher than discriminator's one that approaches immediately 0. D's scores are are 0.95 and 0.08, for real and fake images, indeed the generated images are worse (figure 17).
     - $ndf = 64$: loss plot (figure 18 is similar to the one in previous test, but with lower values, since, the two models are both more complex. Indeed, generated digits are more realistic than before.
     - $ndf = 8$: now the discriminator loss is higher than L_G and for G it is much easier to fool D with unrealistic images, as the ones in figure 19.

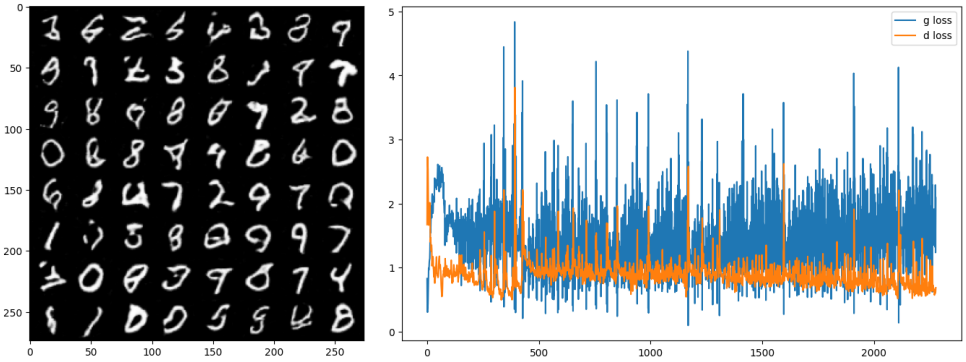

Figure 16: Generator's sample and training loss with $ngf = 64$.

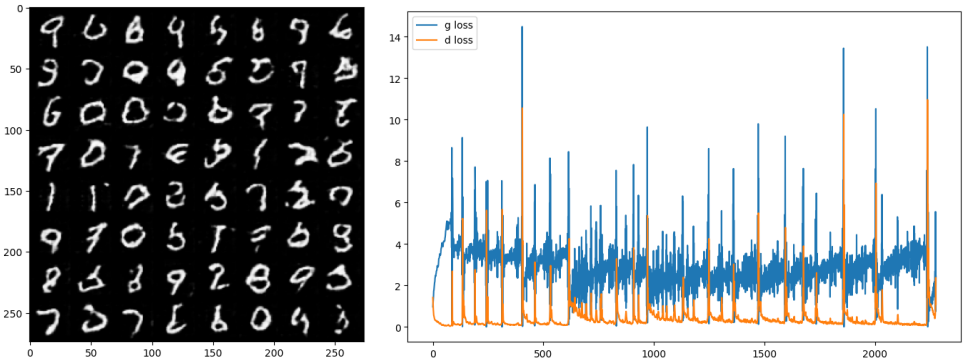

Figure 17: Generator's sample and training loss with $ngf = 8$.

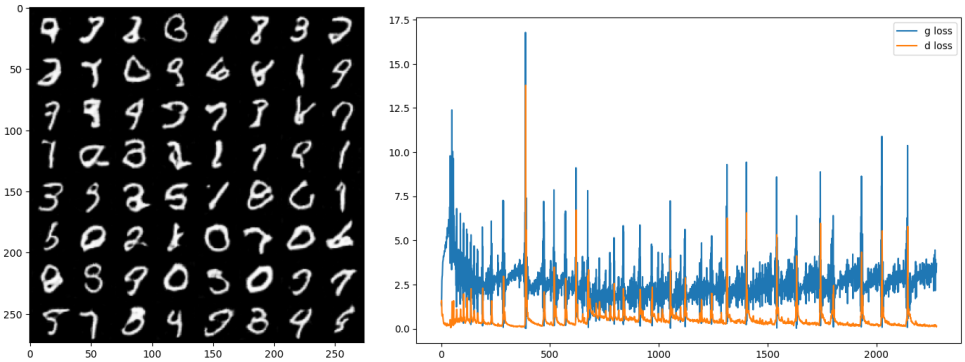

Figure 18: Generator's sample and training loss with $ndf = 64$.

- **Replace the custom weight initialization with `pytorch`'s default one:** By adopting the default weight initialization the generated image seem quite similar to the custom initialization test, but the training G's loss is much more unstable, with very frequent peaks in figure 20.

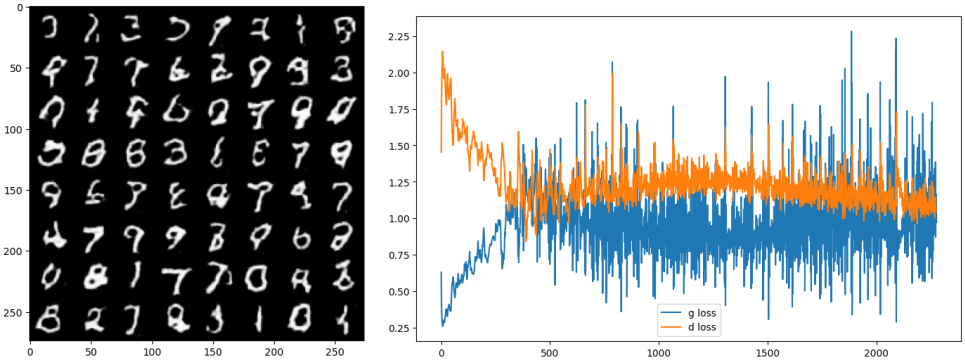

Figure 19: Generator's sample and training loss with $ndf = 8$.

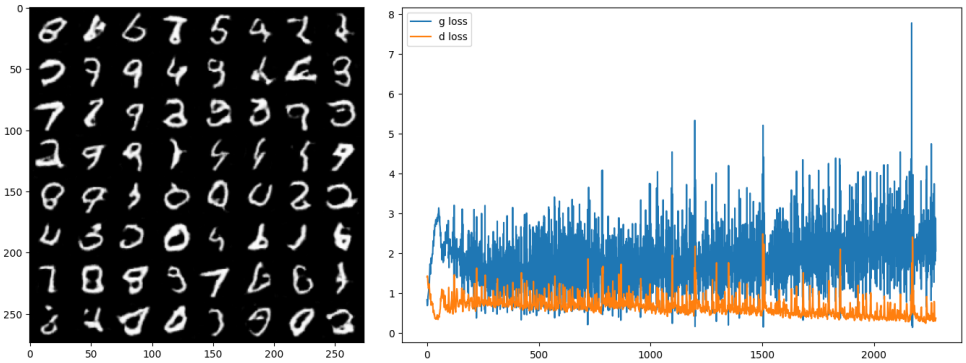

Figure 20: Generator's sample and training loss with default weight initialization.

- **Learn for longer:** Figure 21 shows the effects of longer training. The generated digits may seem slightly better than the ones generated by default training settings, but considering the duration of the training we expected more realistic results. This outcome can be explained by the loss progression: D's loss is close to 0 for most of the iterations, except for some sudden spikes. A loss so small would cause a vanishing gradient problem for G that stops improving.

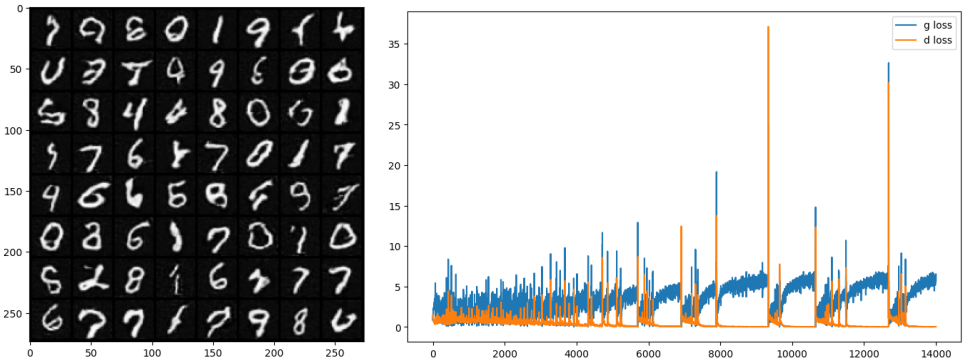

Figure 21: Generator's sample and training loss with a longer training.

- **Reduce or increase significantly n_z:** the test with $n_z = 10$, in figure 22, obtains better results than with $n_z = 1000$, in figure 23. This difference could be due to the fact that the generator has to map random points from the latent space to realistic digits and this mapping could be too difficult to learn if n_z's dimensionality is too large.

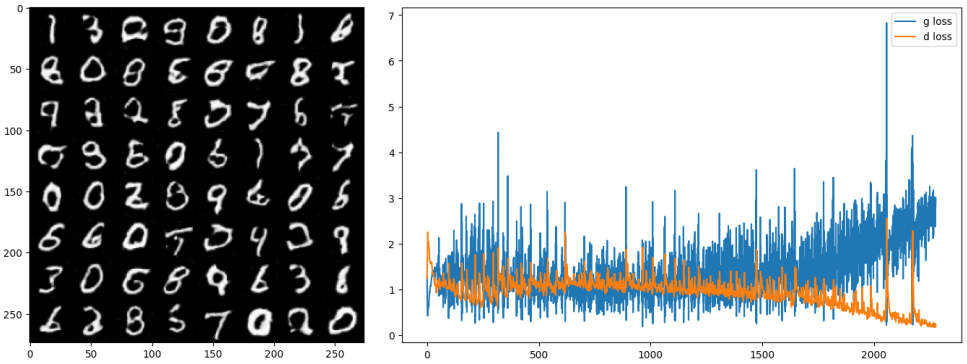

Figure 22: Generator's sample and training loss with $n_z = 10$.

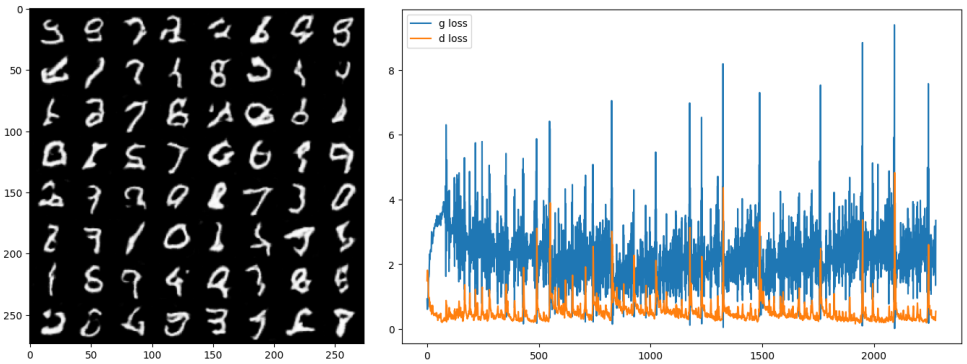

Figure 23: Generator's sample and training loss with $n_z = 1000$.

- **Using a learned GAN, take 2 noise vectors z_1 and z_2 and generate the images corresponding to several linear interpolations $\alpha z_1 + (1-\alpha)z_2; \alpha \in [0;1]$:**

  Figure 24 demonstrates the results of linear interpolation in the latent space Z. We selected pairs of random noise vectors, z_1 and z_2, and generated images for 11 steps of α ranging from 0.0 to 1.0.

  As we traverse the path between two vectors, the generated digits undergo a smooth transformation. For instance, in the first row, a "6" transforms into a "9" by gradually opening the bottom loop.

  The fact that the images in the middle remain coherent and look like plausible handwritten digits indicates that the Generator has learned a continuous mapping

from the latent space to the data space. If the model had simply memorized discrete examples (overfitting), the intermediate points in the latent space would likely correspond to noise or unrealistic artifacts rather than smooth transitions.

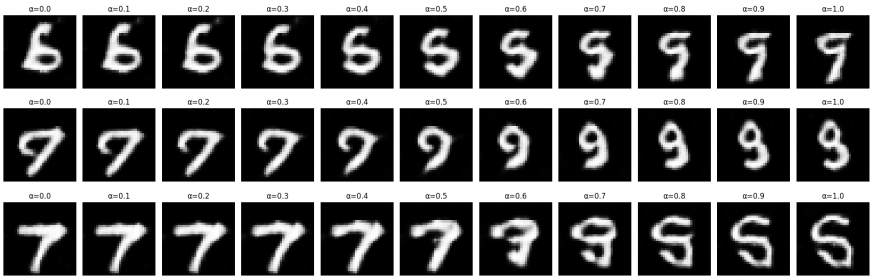

Figure 24: 11 linear interpolations between 2 noise vectors.

- **Try to generate 64×64 images by adding another block in your models (resize your real data accordingly when defining the dataloader):**

  Compared to the standard 32×32 baseline, the training appears more volatile, which is expected given the increased complexity of the image space (which contains 4 times the number of pixels).

  The Generator loss (blue curve) exhibits significant oscillations and sharp spikes throughout the training process. This behavior indicates that the discriminator occasionally becomes much stronger at identifying artifacts in the higher-resolution images, forcing the generator to adjust its parameters drastically to recover.

  Despite this increased volatility, the losses do not diverge. The Discriminator loss (orange curve) stabilizes around 1.0, and the Generator loss, while noisy, maintains a consistent average without collapsing to zero or exploding to infinity. This suggests that the model successfully found an equilibrium, demonstrating that the DCGAN architecture is robust enough to handle the upscaling to 64×64.

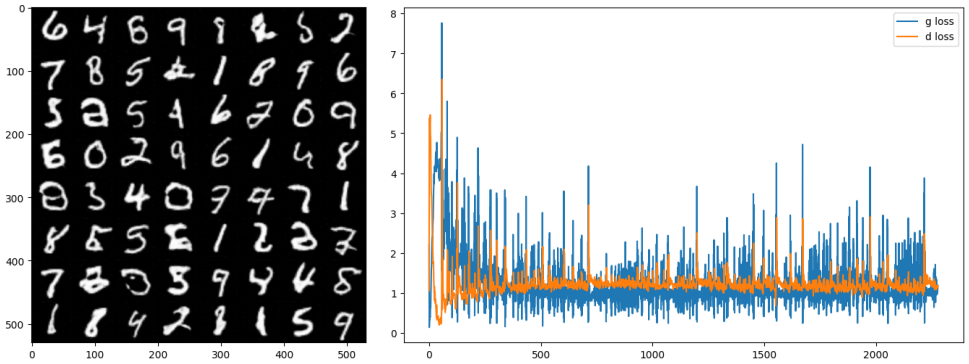

Figure 25: Generator's sample and training loss with resolution 64×64.

- **Try another dataset, CIFAR-10, for example:** figure 26.

- **Try another dataset, CIFAR-10, for example:**

  Transitioning from MNIST to CIFAR-10 introduces a significant increase in complexity, as the model must now handle three color channels (RGB) and a much higher intra-class variance compared to the rigid structure of handwritten digits.

  The results in Figure 26 reflect this difficulty. While the generated samples successfully capture the general color distributions of natural images, such as blue gradients representing the sky or green patches for vegetation, they fail to form distinct, sharp objects. The images appear more like vague textures or "blobs" rather than recognizable entities like cars or animals.

  The loss plot indicates that the training remains stable without divergence, with the discriminator and generator losses oscillating within a reasonable range.

  However, the lack of visual sharpness suggests that the current DCGAN architecture, or the limited training time, struggles to capture the intricate high-frequency details and structural dependencies required for realistic natural image synthesis.

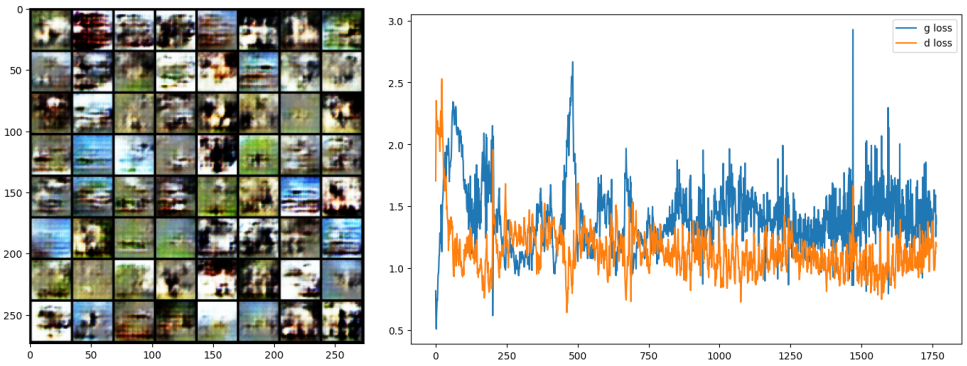

Figure 26: Generator's sample and training loss with CIFAR dataset.

- **Replace the training loss of the generation with the "true" loss derived from the original equation (see question 3):** figure 27.

  The results in Figure 27 clearly shows why the theoretical minimax loss is rarely used in practice. As predicted, the model fails to learn any meaningful patterns, resulting in generated images that are merely uniform noise.

  The loss plot reveals the underlying cause: the discriminator's loss drops to zero almost immediately, implying that it perfectly distinguishes real images from the untrained generator's noise.

  Mathematically, because the discriminator is highly confident (outputs close to 0 for fake data), the term $\log(1 - D(G(z)))$ saturates, causing the gradients back-propagated to the generator to vanish. Without a sufficient gradient signal to guide its updates, the generator cannot improve, leading to visual results like the one obtained.

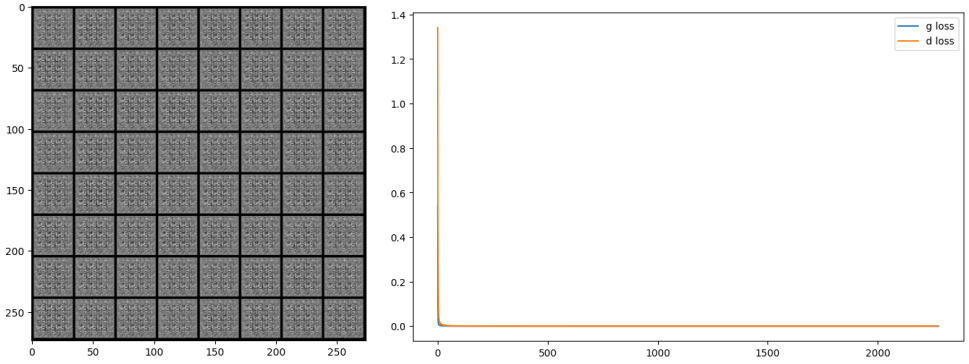

Figure 27: Generator's sample and training loss using "true" loss derived from the original equation.

- **Change the learning rate of both models:**

  The experiments visualized in Figure 28. highlight the critical sensitivity of GANs to the balance between the generator and discriminator learning rates.

  In the first scenario (top row), where the generator's learning rate is 10 times larger than the discriminator's, the training is characterized by high volatility. The loss curves oscillate violently, indicating that the generator is making large updates that destabilize the discriminator. However, despite this instability, the model manages to converge to a state where recognizable digits are produced, suggesting that an aggressive generator can still drive the learning process, even if inefficiently.

  In contrast, the second scenario (bottom row), where the generator optimizes 10 times slower than the discriminator, results in total failure. The discriminator overpowers the generator almost immediately, driving its loss to near zero. In this state, the discriminator perfectly distinguishes real from fake samples, providing no meaningful gradient gradients for the slow-moving generator to learn from. Consequently, the generator gets stuck in a local minimum, producing only uniform noise without ever capturing the data distribution.

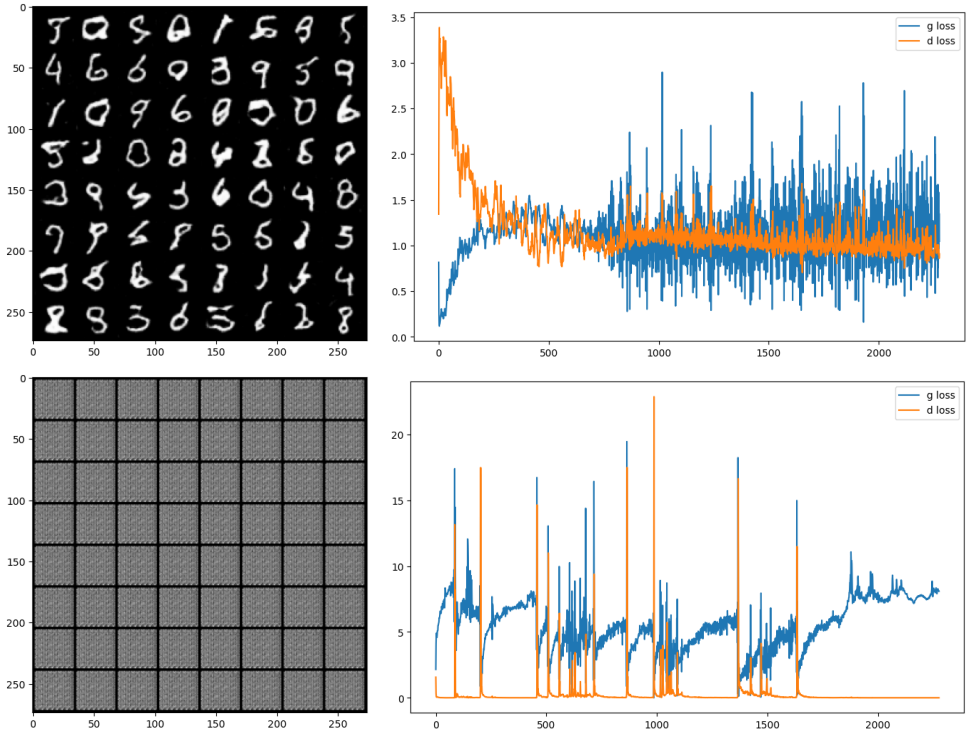

Figure 28: Generator's sample and training loss with generator's learning rate, respectively, 10 times larger and 10 times smaller than discriminator's learning rate.

6. **Comment on your experiences with the conditional DCGAN.**

   The training of the conditional DCGAN, as shown in Figure 25, appears remarkably stable compared to the unconditional counterpart. The generator loss (blue) decreases rapidly in the first 500 iterations and then settles into a stable oscillation around 1.0, while the discriminator loss (orange) remains consistently bounded. This behavior suggests a balanced situation where neither network overpowers the other, avoiding the vanishing gradient problem and reaching an equilibrium more efficiently than in the unconditional experiments.

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

   No, we cannot remove the vector y from the input of the discriminator if we want the generator to respect the class labels. The discriminator in a cGAN needs to model the joint probability $P(x, y)$; it must judge not only if the image x is real, but if it is the correct image for the given label y. If we removed y and inputted only x, the discriminator would merely check if the image looks like a real digit. The generator would then produce realistic digits to fool the discriminator but would have no incentive to produce the specific digit requested by the input label (e.g., generating a "3" when asked for a "7").

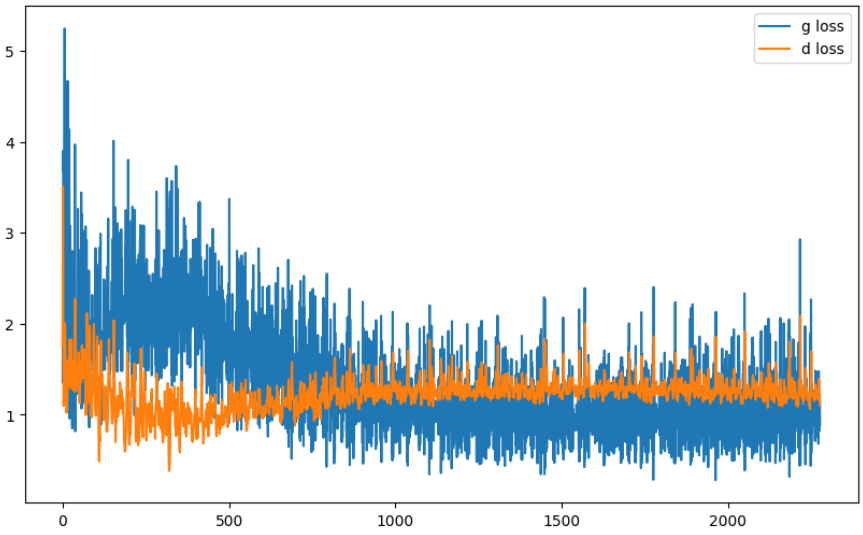

Figure 29: Training loss of DCGAN.

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

   The training was generally more successful and stable than the unconditional case. This improvement occurs because providing the class label y reduces the complexity of the mapping problem. In the unconditional case, the generator must map the latent space Z to the entire multimodal distribution of all digits simultaneously.

   In the conditional case, the problem is partitioned: the generator only needs to learn the distribution of a specific digit given y. The label acts as a strong guide for the gradients, constraining the search space and reducing the risk of mode collapse.

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

   In the generated grid, where rows differ by class y and columns share the same noise vector z, z can be interpreted as the representation of **style** or **handwriting characteristics** (e.g., stroke thickness, orientation...). Since the label y dictates the which digit is generated, z encodes the variations within that content.

   This is evident because a column sharing the same z tends to exhibit consistent stylistic features across different digits, showing that the network has learned to decouple style from content.

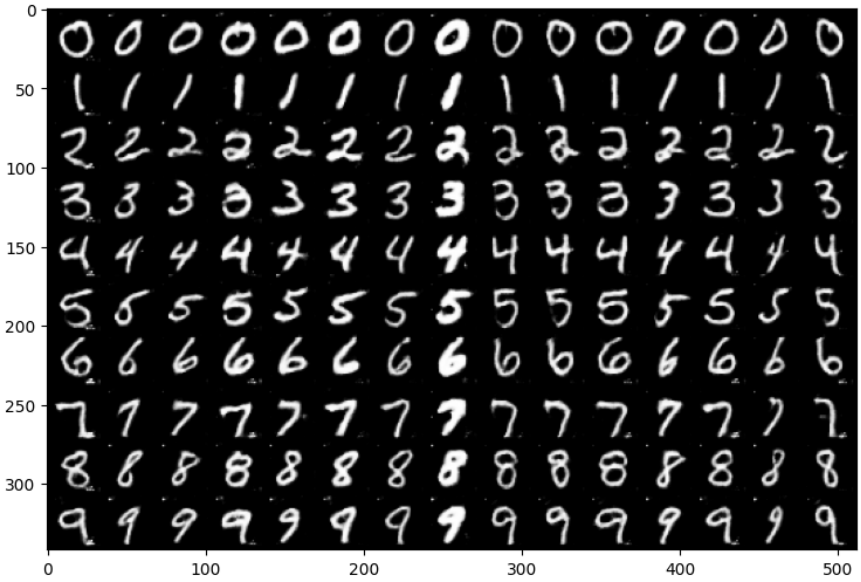

Figure 30: DCGAN testing