Faculté des Sciences et Ingénierie - Sorbonne université

Master Informatique parcours - Données Apprentissage et Connaissances

RDFIA

Report on practical sessions

Section 2 : Deep learning applications

Done by:
BENHADDAD Sabrina
BENSIDHOUM Azzedine

December 2023

Contents

List of Figures

Summary

In our previous practical sessions, we have delved into the world of Convolutional Neural Networks (CNNs), which have revolutionized the field of computer vision. Known for their effectiveness in image-related tasks such as object classification and image recognition, CNNs have not only markedly enhanced the performance of classical neural networks in image processing but have also inspired a plethora of applications and techniques that harness their unique architecture.

Building upon this foundation, our next focus is transfer learning, a technique that addresses the challenges of computational intensity and the need for large datasets in training CNNs. By emulating the way the human brain capitalizes on past experiences, transfer learning repurposes a model pre-trained on a comprehensive dataset like ImageNet for new tasks. This approach leverages the learned features of the pre-trained model, adapting it to new contexts by training specific layers, such as the classification layer. In this report, we will dive into the intricacies of transfer learning, starting with leveraging features from the renowned VGG16 network, trained on ImageNet, for a 15-Scene classification task. Additionally, we will examine other state-of-the-art architectures such as AlexNet, assessing the impact of transfer learning on each.

Another critical dimension of working with visual data is understanding and interpreting the decisions made by CNNs. Unraveling the reasons behind a CNN's predictions is crucial, not just for enhancing model performance but also for uncovering potential biases. Techniques like saliency maps, adversarial examples, and class visualization offer profound insights into the inner workings of CNNs in processing and classifying images. These methods can be instrumental in identifying biases, adding an essential layer of transparency to these powerful models. In this exploration, we will delve into these inter-

pretability techniques, analyzing their outcomes and significance.

Further, we will explore the concept of domain adaptation, which is pivotal when models trained in one domain, such as grayscale images of handwritten digits, are adapted to a related but distinct unsupervised domain, like color images of handwritten digits. An effective approach in this realm is the use of Domain Adversarial Neural Networks (DANNs). These networks are adept at recognizing the domain of an input by simultaneously training a discriminator to predict the domain label and a classifier for the target task. The essence of DANNs lies in their ability to blur the distinctions between source and target domains, thereby enhancing adaptability and performance in new domains. This technique exemplifies the seamless transition and utilization of learned features across varied domains.

Lastly, we will venture into the exciting domain of Generative Adversarial Networks (GANs), renowned for their ability to generate synthetic images that are nearly indistinguishable from real ones. To deepen our understanding of adversarial loss and explore transpose convolutions, we will engage in training both a GAN and a Conditional GAN (cGAN) on the MNIST dataset. This exercise will not only enable us to generate novel handwritten digits but also allow us to evaluate the performances and stability of these models through various experiments. Our journey through this report will provide a comprehensive understanding of the capabilities and limitations of these advanced neural network architectures.

Chapter 1

Transfer Learning through feature extraction from a CNN

Transfer learning is a technique in machine learning that involves reusing a model developed for one task for a different, yet related task. This method is highly beneficial in deep learning, as it saves the computational cost of training large networks from scratch by utilizing previously acquired knowledge.

The main objective of this practical session is to become more familiar with the well-known and commonly-used deep CNN network : **VGG16**, originally introduced in 2015 by Simonyan and Zissernan.
Our focus will be on understanding the architecture and functionality of this influential network.

We will focus on transfer learning's practical applications, specifically on methods to extract key features from pre-trained networks.

Finally, we will apply these extracted features in a practical scenario using SVM classification. This will illustrate how deep features from networks like VGG16 are effectively used for classification, showcasing the adaptability of transfer learning in deep learning.

1.1 VGG16 Architecture

The VGG16 network, a classification model, was originally trained on the extensive ImageNet dataset, which includes over a million images across 1000 classes. Its architecture, as shown in the figure 1.1 comprises five distinct blocks, each featuring:

- Two or three convolutional layers that maintain spatial dimensions.
- A pooling layer, reducing the spatial dimensions by half.

As we progress through these blocks, the network doubles the number of feature maps. Concluding the architecture are three fully-connected layers.

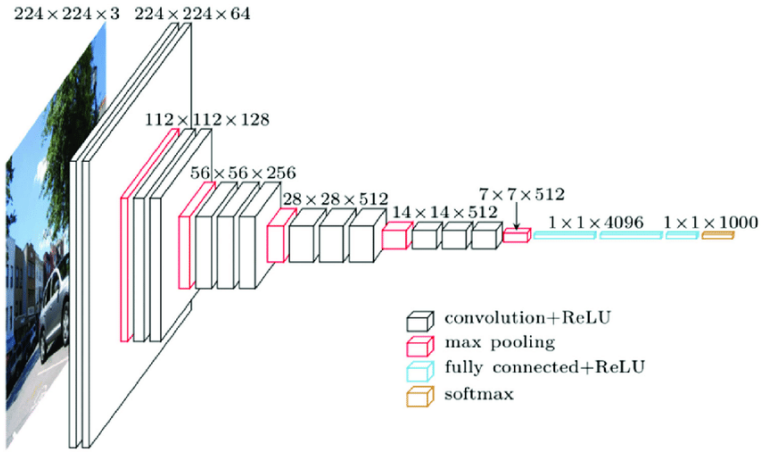

Figure 1.1: VGG16 Network

In this section, we will be utilizing PyTorch's pre-trained VGG16 model. This model requires input images to be resized to 224×224 pixels and normalized for effective processing. Our goal is to explore and understand this architecture, particularly focusing on how it classifies images into one of 1000 categories. This exploration will provide valuable insights into the workings of a deep CNN network in practical applications.

1.1.1 Question 1

Knowing that the last three fully-connected layers account for the majority of the parameters in a model, we will provide an estimate of the total number of parameters in VGG16.
To do so, we will multiply the size of its input by the size of its output and add the number of bias terms.

- **First fully connected layer** - $[7 \times 7 \times 512] \times [1 \times 1 \times 4096] + 4096 = 102.764.544$
- **Second fully connected layer** - $[1 \times 1 \times 4096] \times [1 \times 1 \times 4096] + 4096 = 16.781.312$
- **Third fully connected layer** - $[1 \times 1 \times 4096] \times [1 \times 1 \times 1000] + 1000 = 4.097.000$

If we sum the parameters of each layer, we find around **123 million** parameters. It accounts for the majority of VGG16 which has approximately 138 million parameters.

The presence of numerous parameters in a model can potentially decrease its performance and increase the risk of overfitting. This raises an important question of whether replacing the last fully connected layers improve the model's performance while simultaneously reducing its complexity.

1.1.2 Question 2

The output size of the last layer of VGG16 is a vector of **1000** elements. It corresponds to the predicted probabilities for 1000 different classes from the ImageNet dataset, each probability obtained after applying a *Softmax* function.

1.1.3 Bonus - Question 3

In this question, we aim to assess the pre-trained VGG16 model using a variety of images, each with different levels of complexity. Our selection includes common subjects like cats and dogs, alongside diverse ones such as a car, a kiwi, and a fox. We will also test the model with a complex image of a human. This diverse range of images is chosen to understand how VGG16 performs across varying visual data complexities.

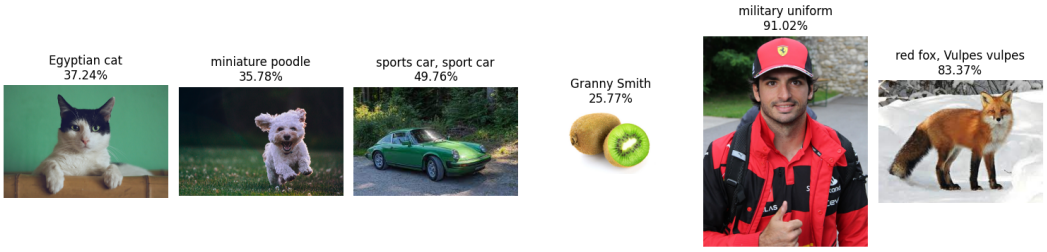

Figure 1.2: The predictions of VGG16 applied to several images

In Figure 1.2, we observe various prediction outcomes. Firstly, images like the red fox and the sports car have been accurately predicted with high scores, suggesting that the model has effectively recognized their features and matched them to the correct classes. This accuracy may be due to the presence of similar images in the training dataset.

The image of the cat has been identified correctly but with a lower score, while the model makes errors when classifying the dog and the kiwi. This indicates that the model perceives similarities with other images it has encountered before.

Finally, regarding the image of the man, even though he is not in military uniform, the model still manages to recognize the concept of a uniform with a high confidence of 91%. This could suggest a significant representation of such images in the training data, allowing the model to strongly associate the image with this category. However, it is important to note that predicting human subjects may present different challenges compared to inanimate objects or animals due to the greater variability in appearance and posture.

1.1.4 Bonus - Question 4

Visualizing the feature maps from convolutional layers can offer valuable insights into the representations a model has learned for a given input at a certain layer.

In this task, we will visualize multiple activation maps obtained after the first convolutional layer.

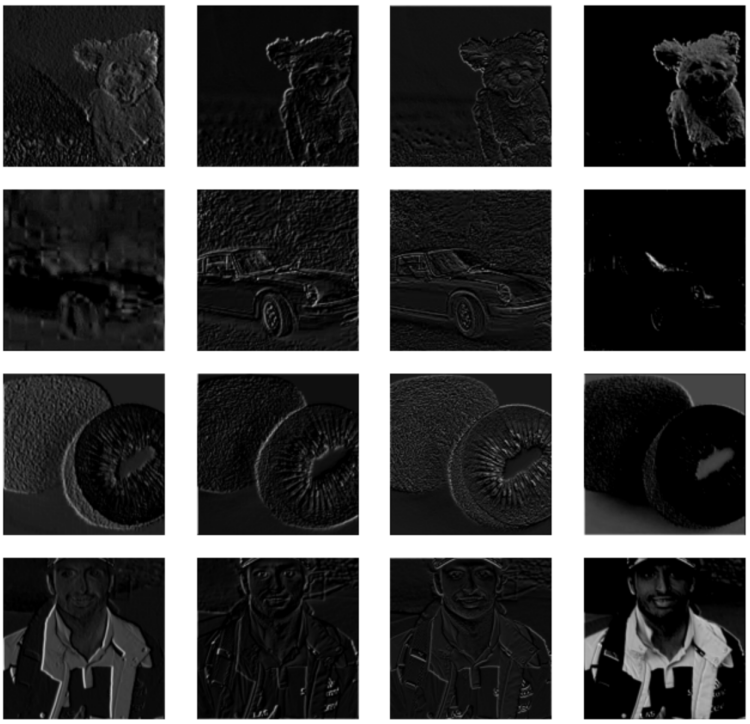

Figure 1.3: The feature maps of our images after the first convolutional layer of the VGG16 architecture

It can be observed that the initial feature maps, which are in proximity to the input, tend to capture a greater level of image detail.

Although this outcome may appear counterintuitive, it is, in fact, expected. The objective of VGG16 is classification, and as we delve deeper into the model, it transforms the original image features into more abstract concepts that are useful for classification. While these deeper feature maps may be challenging for humans to interpret, in the end, they enable the model to recognize the corresponding class.

1.2 Transfer Learning with VGG16 on 15 Scene

In this section, we'll outline our approach, which involves two steps: feature extraction using a pre-trained neural network and training a classification network.
We'll use the VGG16 network, originally designed for image classification, to extract

features from images. These features will be used to classify images from the 15 Scene dataset using an SVM.

1.2.1 Question 5

We don't train VGG16 directly on the 15 Scene dataset because transfer learning is a more efficient approach. Indeed, VGG16 was initially trained on ImageNet, a much larger dataset than 15 Scene. Given the significantly large number of parameters in VGG16, training it from scratch would be especially costly and likely lead to overfitting.
The preferred strategy is to exploit transfer learning techniques, meaning leveraging a pretrained model and adapting it with minimal changes. This approach will not only yield better performance and results but also save computational time and resources.

1.2.2 Question 6

Pre-training VGG16 on ImageNet benefits the classification of the 15 Scene dataset in several ways.
First, it saves considerable computational resources and time. This is because training a large model like VGG16 from scratch requires a significant amount of computational power. By using a model that has already learned to identify a wide range of features on ImageNet, which is a much larger and more diverse dataset than 15 Scene, much of the heavy lifting in terms of basic feature identification such as corners for example has already been done.
Secondly, the model's performance is likely to be improved. ImageNet includes a vast array of images and as a result, the VGG16 model pre-trained on this dataset has learned to recognize a wide variety of features, from basic ones like edges and textures to more complex patterns. When this pre-trained model is applied to the 15 Scene dataset, it can effectively utilize its already acquired knowledge to identify relevant features in these new images.

1.2.3 Question 7

Although feature extraction offers numerous advantages in terms of computational time, resource efficiency, and performance enhancement, this technique indeed has some limitations that can be summarized as follows.
If the source and target domains are different, the performance of the new model may not be improved and could even be relatively weak. This is because transfer learning primarily works well when the source and target domains are similar. If the model encounters new

and different images than those in the source dataset, it may struggle to accurately detect the features, which can lead to poor performance.

1.2.4 Question 8

The choice of layer for feature extraction in deep learning models significantly impacts their effectiveness. Early layers extract basic, general features like edges and textures (low-level features), while deeper layers identify more complex objects, tailoring to specific problems (high-level features).
The depth of feature extraction depends on the similarity of the source and target domains. Extracting from early layers requires additional training for complex object recognition, while deeper layers are better suited for domains with similar datasets, providing more specialized feature recognition. In cases with similar source and target domains, using high-level features from deeper layers can be significantly advantageous.

1.2.5 Question 9

The images from 15 Scene are black and white, but VGG16 requires RGB images. To address this issue, we can duplicate the 15 Scene images three times - one for each channel. In other words, we convert the grayscale images to three-channel images by duplicating the single grayscale channel across the three channels (R, G, B). This way, each channel will contain the same grayscale information, allowing the VGG16 model, which expects RGB inputs, to process the image.

1.2.6 Question 10

It's entirely feasible to use only neural networks instead of employing an independent classifier. However, this requires adaptation. The VGG16 classifier has an output size of 1000, while the 15 Scene dataset requires an output of size 15. To address this, one would load the pre-trained model and freeze the weights before the final layers where feature extraction occurs. Then, change the output vector size to 15 classes instead of 1000, and train these last classification layers on the 15 Scene dataset.
In summary, we can use neural networks like VGG16 for classification without separate classifiers by adapting the network's final layers. For instance, with the 15 Scene dataset, we adjust VGG16's output layer to match the dataset's 15 classes, retain the final layers, and use the pre-trained earlier layers for feature extraction.

1.2.7 Going further

In this section, we will focus on several experiments conducted to evaluate our transfer learning model. To do so, we introduce some modifications to the classification scheme created to observe their effects on performance.

Trying other available pre-trained networks

To initiate our experiments, we will utilize alternative pre-trained networks and assess their performance. In addition to VGG16, we will evaluate the *AlexNet* network.

AlexNet : AlexNet is one of the first deep neural networks, presented in 2012. It is pretty simple compared to the others. It introduced the idea of using ReLU as an activation layer, which helped with the vanishing gradient problem. It has over 60M parameters.
These two networks have in common the fact that they are variants of CNN architectures.
Table 1.1 shows the results of evaluating each architecture on the 15 Scene dataset. As

Model	Accuracy
VGG16ReLU7	**88.58%**
AlexNet	80%

Table 1.1: Accuracy of AlexNet and VGG16 pre-trained models on the 15 Scene dataset with Transfer Learning

expected, AlexNet is less performant than VGG16, even if it's less expensive in terms of execution time.

Changing the layer at which the features are extracted

Now that we have explored other pre-trained models, we are in a position to evaluate the impact of the layer from which features are extracted in the VGG16 model.

Note that to achieve this goal, we have slightly modified the vgg16relu7 class, renaming it to vgg16_modified. This updated class accepts a layer parameter, specifying the particular layer at which features will be extracted.

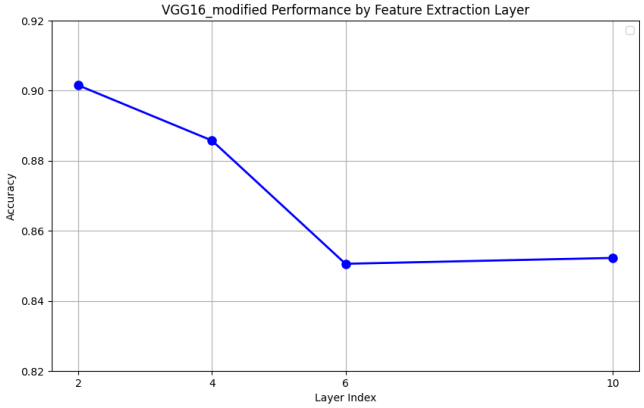

Figure 1.4: Accuracy of the Pre-trained VGG16_modified Model as a Function of the Feature Extraction Layer

Figure 1.4 illustrates the impact of the feature extraction layer on the model's accuracy. It is evident that generally, the deeper the layer from which features are extracted, the better the performance. This suggests that for this dataset, extracting high-level features (from deeper layers) is more advantageous.

The depth at which features are extracted is crucial, as it significantly influences the classification results. Extracting low-level features for this dataset means overlooking the general concepts learned by the pre-trained model, leading to a decrease in performance.

Tuning the C parameter

In this section, we explore the impact of the SVM's C parameter.

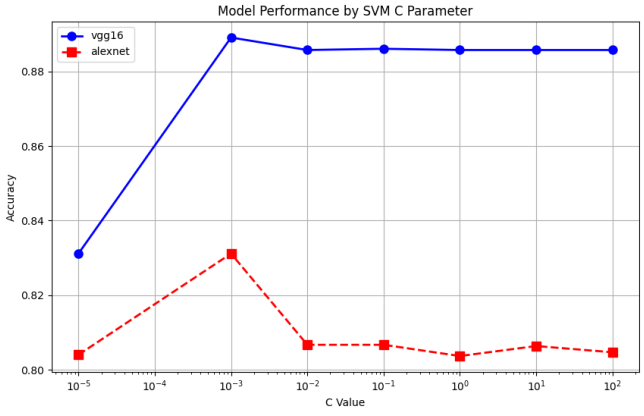

Figure 1.5: Accuracy of the Pre-trained VGG16_modified Model as a Function of the SVM C Parameter

We observe that the optimal value of C for both the AlexNet and VGG16 models is 10^{-3}. At this value, the VGG16_modified model achieves an accuracy greater than 90%,

while the AlexNet model achieves 84% accuracy.

Dimensionality reduction

Dimensionality reduction techniques have proven useful for capturing important information and reducing computational time.

We apply the PCA method to our data. The results are shown in Figure 1.6.

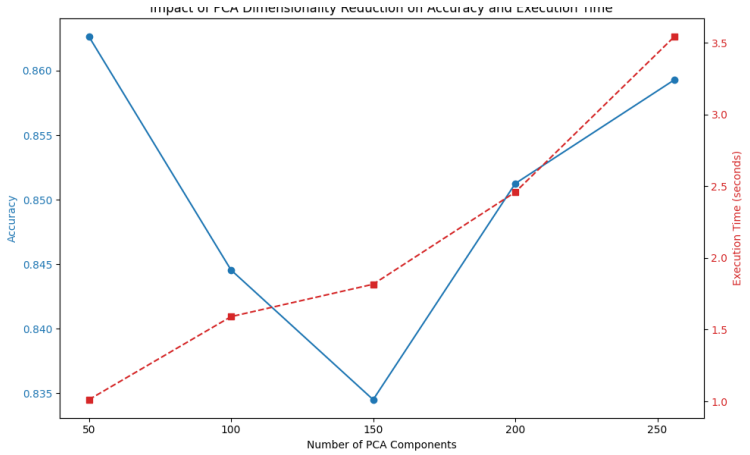

Figure 1.6: Impact of PCA Dimensionality Reduction on Accuracy and Execution Time

We observe that the execution time decreases when we reduce the number of components. However, this is of limited benefit since accuracy drops significantly with reduced dimensions. For this task, we conclude that it is preferable not to use dimensionality reduction methods, even though they speed up computational time.

Chapter 2

Visualizing Neural Networks

In the fast-evolving field of deep learning, understanding Convolutional Neural Networks (CNNs) is crucial for technological and AI advancements. This work emphasizes neural network visualization as a means to interpret and analyze CNN behavior. Instead of the traditional approach of modifying or training the network, this study innovatively applies gradient-based techniques to see how CNNs react to varying inputs, using a pre-trained network and altering input images to investigate the network's response indirectly.
Three core visualization methods are spotlighted to facilitate this analysis.

Saliency Maps : These maps offer insight into the CNN's internal mechanics, highlighting which pixels influence image processing.

Adversarial Examples (Fooling Samples): This approach reveals how slight image perturbations can mislead classifications, highlighting the networks' sensitivity.

Class Visualization: This technique reveals the network's recognition patterns and features for different classes.

Each visualization method offers a distinct perspective into CNNs, aiding in the decryption of their image processing and responses to a variety of inputs, including the figure 2.1 that represents the specific images being tested in our study. This enhances our comprehension of CNN functionality.

Figure 2.1: 24 ImageNet's examples

2.1 Saliency Map

2.1.1 Question 1

The model demonstrates strong performance in image classification, effectively focusing on pixels crucial to the target classes. Its capability to prioritize relevant features and sensitivity to spatial arrangements is evident, particularly in scenes where it emphasizes spatially prominent objects, like the nearest hay bale in a hay image (as shown in Figure 2.2). In animal identification tasks, the model accurately classifies species by emphasizing facial features such as eyes, ears, and beaks. However, it sometimes integrates contextual elements into its decision-making process, which, while generally beneficial, can lead to misclassifications under varying contexts.

One challenge the model faces is in differentiating between relevant and irrelevant features, especially in complex backgrounds. For instance, it may focus on the ground in a quail image or snow in a bear image, potentially leading to misclassifications when overly reliant on context. This issue, highlighted in Figure 2.2, is exemplified by its failure to correctly identify critical elements in a Christmas stocking image, suggesting the need for more diverse training data.

Overall, the model shows a keen attention to detail, but there is a risk of overfitting to specific patterns, limiting its ability to generalize to new, unseen images. This limitation is particularly crucial for objects in various orientations or styles that the model may not have encountered during training. Figure 2.2 visually presents these findings, illustrating both the strengths and limitations of the model in image classification.

2.1.2 Question 2

Saliency maps come with several limitations in interpreting neural network decisions:

Linear Approximation of Non-linear Models: Saliency maps often represent a linear approximation to the inherently non-linear nature of neural networks, which may not capture the full complexity of their decision-making processes.

Potential for Misinterpretation: The simplification inherent in saliency maps can lead to misinterpretations about what the model has learned and how it makes decisions.

Focus on Individual Pixels: Saliency maps emphasize the importance of individual pixels without considering the spatial hierarchies and relationships that are critical to understanding CNNs, missing the broader context.

Combining RGB Channels: The practice of combining RGB channels to create a single saliency map can conceal the distinct contributions of each channel, complicating the interpretation of the model's behavior.

2.1.3 Question 3

Saliency maps are versatile tools for more than just interpreting neural networks. They provide valuable insights for model debugging, improvement, and bias detection.

Model Debugging and Improvement: Saliency maps can pinpoint why models make incorrect predictions. For example, if a model trained on animal recognition focuses more on the background, saliency maps can reveal this flaw. This insight can help in augmenting training data to help the model better distinguish between relevant and irrelevant features, and modifying the model's architecture, possibly by incorporating attention mechanisms, to focus on important areas.

Bias Detection: These maps are instrumental in uncovering biases, especially concerning sensitive attributes like race or gender. They are critical in areas like hiring or medical diagnosis, where biased decisions can be harmful. Saliency maps can uncover if decisions are based on biased or sensitive features, suggesting a need for feature adjustment or removal, and aid in developing fairness-aware training techniques to minimize the influence of sensitive features on the model's output.

In summary, saliency maps serve multiple purposes, from identifying biases and debugging models to improving their accuracy and fairness. They are key in understanding where a model may be erring and guiding efforts to enhance its performance.

2.1.4 Bonus - Question 4

It's crucial to recognize that saliency maps are model-dependent; those generated with VGG16 exhibit marked differences from those by SqueezeNet. The VGG16 model demonstrates an advanced level of training, distinctly focusing on the pivotal features of objects. For instance, in the quail image, VGG16 effectively distinguishes the bird from its background, precisely highlighting the quail's body and head, and even capturing finer details like straw clumps. This enhanced attention to critical features is also evident in other examples, such as lacquer items, pirate ships, and swans.
In these cases, VGG16 accurately interprets reflections in the water, a notable improvement over the previous model's performance. Yet, VGG16 is not without its shortcomings. This is apparent in certain images, like the Christmas sock, where it exhibits similar

misidentification issues as the previous model. These observations are clearly illustrated in Figure 2.3, which showcases the saliency maps obtained using VGG16.

2.2 Adversarial Examples

2.2.1 Question 5

The generation of Adversarial Examples (Fooling Images) for an image classification model reveals significant insights, as shown in the two-panel display of the figure 2.4

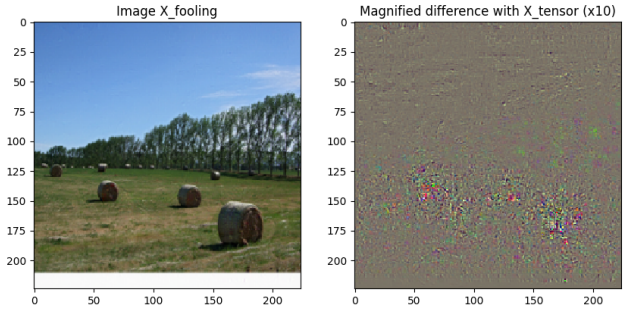

Figure 2.4: First results of the Adversarial Examples method

In the left panel, a seemingly typical rural scene with hay bales is presented. However, subtle manipulations have been applied to this image, effectively fooling the classifier into misidentification.

The right panel offers a magnified comparison between the original and the manipulated image, with the perturbations amplified tenfold for clarity. A detailed analysis reveals that these alterations are concentrated around the hay bales, suggesting that the adversarial algorithm specifically targeted these areas. These focused modifications indicate that the algorithm has identified the hay bales as key features influencing the classifier's decisions. Such strategic perturbations, while imperceptible to humans, significantly impair the AI's recognition capabilities.

These results highlight the susceptibility of image classification models to Adversarial Examples — images meticulously engineered to exploit weaknesses in the models. Even minor adversarial adjustments, as shown in the figure 2.5 can substantially degrade the

classifier's accuracy, posing serious risks in real-world scenarios where these models are deployed for critical decision-making tasks.

The findings underscore the urgent need for developing more resilient machine learning models that can resist manipulations by Adversarial Examples. They also stress the importance of understanding and preparing for the types of attacks that models may face, particularly in sectors where security and precision are paramount.

2.2.2 Question 6

Adversarial examples in CNNs have significant repercussions in their application and development:

Model Vulnerability and Security Risks: Adversarial examples expose CNNs to security vulnerabilities due to their sensitivity to minor input modifications. This poses substantial risks in critical areas like facial recognition and autonomous driving.

Robustness Evaluation and Model Generalization: They necessitate reevaluating neural network robustness and underscore the challenges in achieving effective generalization across diverse scenarios.

Defensive Techniques and Increased Awareness: The rise of adversarial examples has sparked the development of defensive strategies and raised awareness about neural network limitations and vulnerabilities.

In practical applications, adversarial examples pose significant risks to CNNs, especially in critical fields like autonomous vehicles, medical diagnostics, and secure authentication systems. The ability of these examples to induce errors in models, undetectable by humans, underscores the vital need for developing more robust and dependable deep learning models.

2.2.3 Bonus - Question 7

Limits of the Naive Approach

- *Overfitting to Specific Models:* These methods tend to overfit to certain models or datasets, diminishing their effectiveness across different architectures (Papernot et al., 2016).
- *Detectability:* Simple perturbations created by naive approaches are more easily identified and countered by defenses like input preprocessing or adversarial training (Goodfellow et al., 2014).

- *Lack of Realism:* Naive adversarial images often fail to retain realistic image qualities, reducing their practicality in real-world applications (Sharif et al., 2016).

Alternative Approaches

- *Transferable Adversarial Examples:* Focusing on creating adversarial examples that retain their effectiveness across various models, addressing the overfitting problem (Liu et al., 2016).
- *Generative Adversarial Networks (GANs):* Employing GANs to produce more complex and less detectable adversarial examples (Xiao et al., 2018).
- *Robust Optimization Techniques:* Implementing robust optimization during model training to naturally resist naive adversarial attacks (Madry et al., 2017).

2.3 Class Visualization

2.3.1 Question 8

The progression of images offers a glimpse into how a neural network's perception of a specific concept, such as a tarantula, evolves during optimization. Starting with random noise, the optimization algorithm gradually refines the input to maximize neuron activation related to the concept. Initially, the image is just unstructured noise, but as optimization advances, distinct features emerge, representing the network's learned associations with the concept.

These visual features grow increasingly defined, gradually taking on the likeness of a tarantula's legs, body, and form. The optimization process effectively asks the network to identify input patterns that most convincingly represent a tarantula. Through iterative adjustments, the network zeroes in on these key features.

By the end of the process, the image transforms from noise into a visualization rich with tarantula-like elements. This final image reflects the network's interpretation of the most significant features for recognizing the concept, after many rounds of optimization.

This evolution offers deep insights into the neural network's information processing and highlights the visual patterns it deems crucial for concept recognition, revealing the inner workings of its learning process.

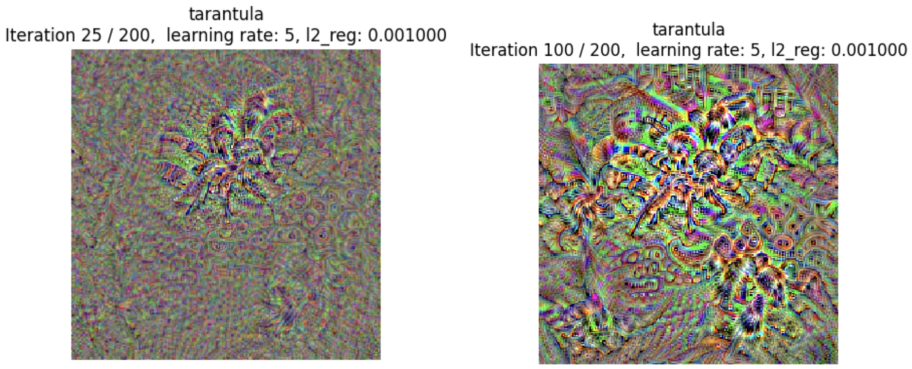

Figure 2.6: First results of the generated image maximizing the score of a class, subject to a certain number of regularizations

2.3.2 Question 9

When analyzing the effects of varying parameters, key observations emerge:

Number of Iterations

The number of iterations profoundly affects the clarity and complexity of the network's visual features. More iterations allow for finer adjustments, yielding a detailed and intricate representation of the target class's features. As iterations increase, the network more precisely tailors the image to the learned features of the class, enhancing the depiction of characteristic shapes and textures, especially noticeable in the case of tarantulas.

Learning Rate

The learning rate significantly impacts the optimization's speed and stability. A higher rate accelerates convergence but may introduce instability, while a lower rate ensures gradual, stable changes, potentially leading to a more accurate depiction of the target class. However, slower rates might necessitate more iterations for convergence. This observation highlights the trade-off between convergence speed and image quality, emphasizing the need for a balanced approach in activation maximization experiments.

L2 Regularization (l2 reg)

L2 regularization is crucial in moderating the complexity and interpretability of learned features. Higher regularization weights smooth out the image, possibly reflecting average class features more accurately, while lower weights permit detailed, albeit potentially more abstract, visualizations. Thus, L2 regularization governs the balance between simplicity and complexity in the generated images, significantly influencing the visualizations' appearance and utility in activation maximization.

In conclusion, the interplay of iteration count, learning rate, and L2 regularization weight is vital for fine-tuning the network's output, with each parameter playing a pivotal role in the quality and representativeness of the resulting visualizations.

2.3.3 Question 10

Starting the optimization process with an ImageNet image belonging to the target class, such as 'hay,' offers several advantages:

- **Class Reinforcement:** The process strengthens the features the network already associates with the target class, enhancing our understanding of what the network deems indicative of that class.
- **Subtler Modifications:** Using an image from the target class leads to more subtle changes than starting with random noise or unrelated images. This results in a less distorted and clearer image, aiding interpretability.
- **Model Interpretability:** This approach sheds light on how the model refines predictions on real-world data, offering insights into how it weighs and prioritizes certain features.
- **Identifying Model Limitations:** Observing the model's alterations on an authentic class image can reveal its tendencies to exaggerate specific features, pinpointing potential limitations or biases in the model's perception of the class.

The provided images illustrate the transformation from the original ImageNet picture of a hay scene to the optimized version, accentuating 'hay' characteristics. The final image becomes richer in texture and patterns, with hay bales more pronounced, potentially reflecting the network's learned exaggerations for the 'hay' concept.

2.3.4 Bonus - Question 11

The variations in the images demonstrate the distinct architectural traits and feature extraction capabilities of SqueezeNet and VGG16. Both networks were subjected to the same number of iterations, learning rate, and L2 regularization, revealing how their structures differently interpret the 'hay' class:

- **SqueezeNet (First Image):** After 200 iterations, the image from SqueezeNet shows an accentuation of hay features while largely maintaining the original scene's structure. SqueezeNet's compact architecture likely concentrates on refining localized features indicative of 'hay,' enhancing existing elements without significantly altering the overall image.

- **VGG16 (Second Image):** VGG16, known for its depth, processes a broader range of features. The resulting image appears more abstract, with features dispersed across the scene. Its larger capacity and depth enable VGG16 to integrate a wider context, leading to a more elaborate transformation of the original image.

This approach, focusing on the actual class for activation, sheds light on how each model amplifies hay bale features to enhance their recognizability. This comparison is insightful for understanding the networks' feature processing methods and their interpretations of the 'hay' class. It also reveals insights into each model's robustness and predictive behavior, particularly when encountering new images containing hay.

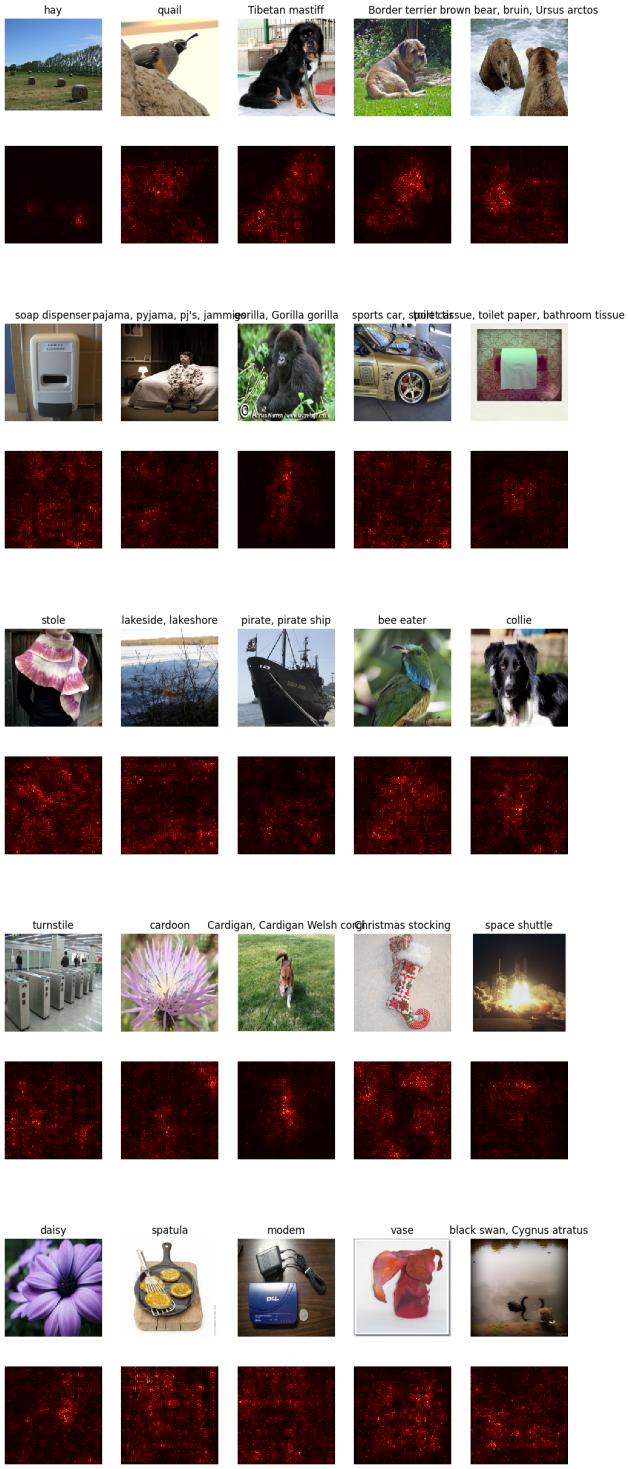

Figure 2.2: Figure representing the saliency maps of images of the ImageNet dataset with the SqueezeNet architecture

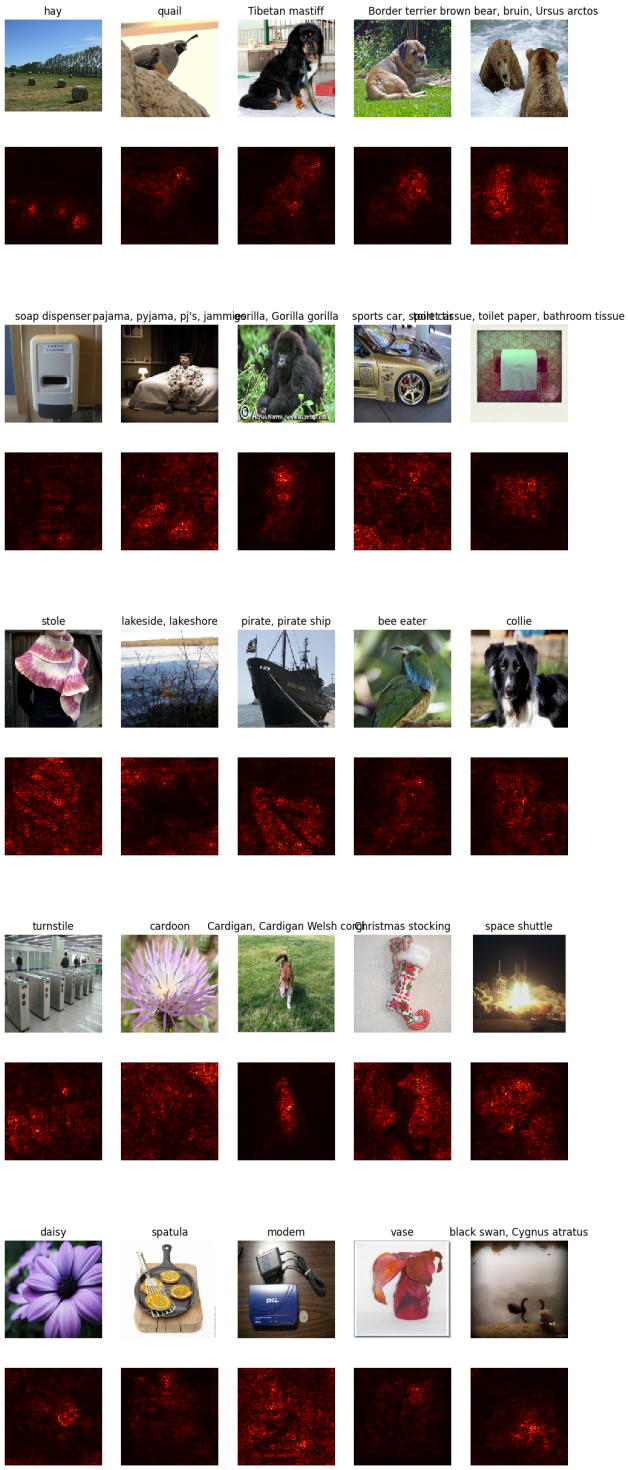

Figure 2.3: Figure representing the saliency maps of images of the ImageNet dataset with the VGG16 architecture

Figure 2.5: Figure representing the original correctly classified images and their modified example in fooling the SqueezeNet network

Chapter 3

Domain Adaptation

Domain adaptation is a field within computer vision where the objective is to train a neural network on a source dataset and achieve high accuracy on a target dataset, which significantly differs from the source. The purpose of this practical session is to gain a deeper understanding of domain adaptation and its applications.

To this end, we will construct a DANN model (Domain Adversarial Neural Network). This model will be trained on a labeled *MNIST* dataset and subsequently evaluated on an unlabeled *MNIST-M* dataset, as illustrated in Figure 3.1.

Figure 3.1: Example of samples taken from both MNIST and MNIST-M

3.1 DANN and the GRL layer

As mentioned earlier, this section is dedicated to comprehending and experimenting with the DANN (Domain Adversarial Neural Network) model. Let's begin by examining its architecture, which is depicted in Figure 3.2.

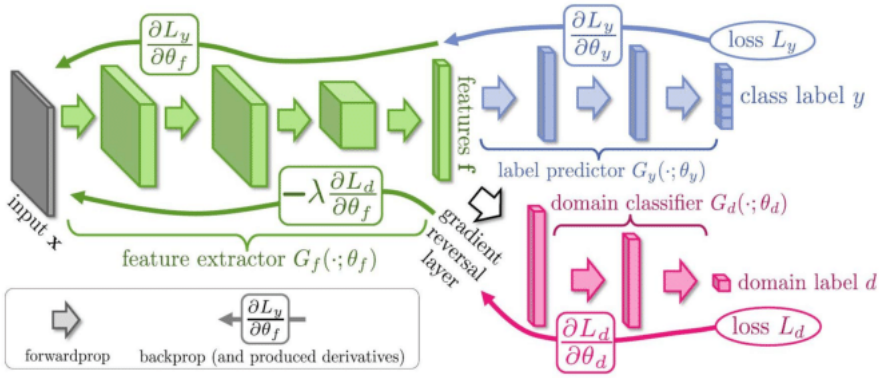

Figure 3.2: DANN Architecture

In this architecture, the green layers are the convolutional neural network (ConvNet), primarily focused on learning robust feature representations, for the classifier, whose layers are in blue. Beyond just features extraction, our aim is to ensure domain independence of these representations. For instance, the digit '7' should be consistently represented regardless of its background color. To do so, the network incorporates a domain classifier branch (pink layers), which determines whether the ConvNet's output originates from the source or target domain.

The key to this functionality is the Gradient Reversal Layer (GRL). The GRL, situated between the domain classifier and the feature extractor, reverses the gradient flow, encouraging the ConvNet to produce domain-agnostic features. This means that despite different backgrounds (like in MNIST and MNIST-M datasets), the network focuses only on the digit information, making features consistent across domains.

Experiments and results

We implement a Domain Adverserial Neural Network (DANN) and get 77% accuracy on the target domain. The results are good, but they are still not as good as the results obtained in the original article of the model (81%). So, in order to improve our model and get closer performances to the original paper, we tried using more epochs - 50 - and testing different standardization. The results are presented in Table 3.1. It is observed that

Model	Accuracy
Original DANN	76.73%
Improved (20 epochs, BatchNorm2D,BatchNorm1D, Dropout)	$\approx 73\%$
Improved (50 epochs, normalization)	81.4%
Improved (50 epochs, BatchNorm2D,BatchNorm1D, Dropout)	**82.43%**

Table 3.1: Accuracies of the DANN with different experiments

increasing the number of epochs significantly enhances performance, aligning it with the benchmarks set in the referenced article. Additionally, the incorporation of BatchNorm2d into the convolutional layers, alongside BatchNorm1d in both the label and domain classifiers, and the use of dropout, further improves performance, achieving an accuracy of 82.43%. These standardization techniques contribute to stabilizing the optimization process, which likely accounts for the observed improvements in accuracy.

3.2 Practice

3.2.1 Question 1

The Gradient Reversal Layer (GRL) is designed to reverse the gradient between the domain classifier (in pink) and the feature extractor (in green). It aids in aligning the feature distributions of the source and target domains and enhances the model's generalization performance by encouraging the feature extractor to learn domain-invariant features. Removing it will likely lead to poor generalization performance on the target domain.

If we retain the network with all three parts (green, blue, pink) but do not use the GRL, the model's performance might improve on the source domain. However, this would cause the model to become overly specialized to the source domain, impairing its ability to generalize to the target domain.

This is precisely what occurred in the example we tested: omitting the GRL layer led to poor generalization, with an 54% accuracy in the test phase, and an 99% accuracy in the train phase.

3.2.2 Question 2

Performance on the source dataset may slightly degrade when using a Domain Adversarial Neural Network (DANN) approach, due to several factors. The main goal of DANN is to learn features invariant across both source and target domains. This focus can sometimes lead to the underrepresentation of domain-specific features crucial for the source dataset, potentially decreasing its performance. Additionally, the regularization effect of learning domain-invariant features, while beneficial for generalizing to the target domain, might restrict the model's ability to fit closely to the source data. Incorporating the Gradient Reversal Layer (GRL) could further cause a slight performance decline on the source dataset as the model learns features from the target domain, making it less specific to the source. However, this trade-off often results in better generalization and improved performance on the target dataset.

3.2.3 Question 3

The value of the negative number used in the Gradient Reversal Layer (GRL) plays a crucial role in the network's training dynamics and overall performance. This value, which reverses the gradient during backpropagation, effectively determines the balance between feature discriminability and domain invariance.

- A **higher value** of this parameter shifts the focus more towards domain invariance. The GRL then strongly reverses the gradients, encouraging the feature extractor

to learn domain-invariant features. This enhances the model's generalization across different domains, but might reduce its discriminative power for specific tasks within each domain.

- Conversely, a **lower value** of this parameter tends to preserve more discriminative features specific to the source domain. With a weaker influence on the gradients, the feature extractor may learn more domain-specific features. This approach can lead to improved performance on the source domain, but it might compromise generalization to the target domain and be less effective in aligning the distributions of the source and target domains.

Practically, the magnitude of this parameter is often scheduled throughout the training process to optimize the balance between learning domain-specific and domain-invariant features.

3.2.4 Question 4

Pseudo-labeling is a semi-supervised learning technique frequently employed in domain adaptation. It is particularly useful in scenarios where a source domain has abundant labeled data, but the target domain has little or no labeled data. This method is well illustrated in the accompanying figure 3.3, which outlines the general concept.

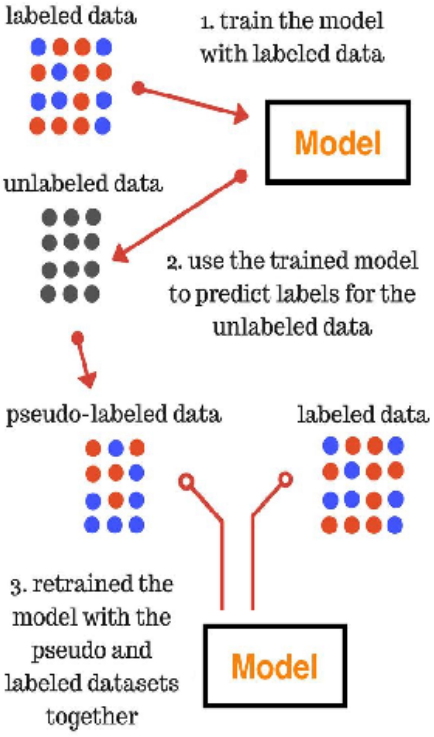

Figure 3.3: Pseudo labeling schemas

The core idea of pseudo-labeling is to use a model trained on a labeled source dataset to generate labels for an unlabeled target dataset. These generated labels are known as pseudo-labels. The process of pseudo-labeling involves the following steps :

- Training the model on the source domain.
- Using the trained model to predict labels for unlabeled data from the target domain, thereby creating pseudo-labels.
- Retraining or continuing the training of the model on a combination of labeled source data and unlabeled target data, using the pseudo-labels as ground truth for the target domain.

The training process begins with the model learning exclusively from the labeled data in the source domain. Once the model reaches a certain level of performance, it is employed to predict labels for the unlabeled data in the target domain. These pseudo-labels are subsequently utilized, often along with the original labeled data, to further train the model.
This method features an iterative refinement mechanism. As the model's accuracy improves, so does the quality of the pseudo-labels it generates. Future training phases incorporate these enhanced labels, contributing to the gradual increase in the model's accuracy.
In summary, pseudo-labeling bridges the gap between different domains by leveraging the model's own predictions to create additional training data, thus facilitating more effective domain adaptation.

Chapter 4

Generative Adversarial Networks

In recent years, the field of computer vision has experienced a paradigm shift, moving away from traditional classification models that predominantly focus on modeling the conditional probability distribution $P(Y|X)$. This shift has been significantly influenced by the advent of Generative Adversarial Networks (GANs), which have introduced new methodologies in data analysis and image processing.

First introduced in 2014, GANs mark a substantial deviation from classic generative models that emphasize the joint distribution $P(X, Y)$. Their approach, a cornerstone in unsupervised learning, minimizes reliance on labeled datasets, underscoring their adaptability and efficiency.

This report aims to offer an in-depth analysis of Generative Adversarial Networks. We will explore the structural nuances of GANs, illuminating their transformative role in image generation, completion, and editing. Our discussion will encompass not only the fundamental principles of GANs, but also extend to Conditional GANs (cGANs). These advanced iterations of GANs integrate extra input variables, enhancing the control and specificity of the image generation process. This allows for the creation of contextually relevant and personalized visual content, showcasing the remarkable versatility of these networks.

Upon concluding this report, we will acquire a thorough understanding of GANs' operational mechanisms and their profound implications in artificial intelligence, especially in computer vision.

4.1 Generative Adversarial Networks

A Generative Adversarial Network (GAN) involves two key components: the generator G and the discriminator D. The generator function $G(z)$, where $z \sim P(z)$, creates images from a random noise vector z. The discriminator D assesses these images against real samples, trying to differentiate real data x^* from generated images $\tilde{x}$. The training process is adversarial: G aims to maximize $\log D(G(z))$, while D aims to maximize $\log D(x^*) + \log(1 - D(G(z)))$, leading to a competitive improvement of both networks.

4.1.1 Question 1

The formula $\max \mathbb{E}_{z\sim P(z)}[\log D(G(z))]$ represents the generator's goal in a Generative Adversarial Network (GAN) to create images from noise vectors z, which are convincingly realistic enough to be classified as real by the discriminator D. It seeks to maximize the expected likelihood $\log D(G(z))$ across all samples of z, indicating the generator's effectiveness in producing images that closely mimic real data.

The formula $\max \mathbb{E}_{x^*\in\text{Data}}[\log D(x^*)] + \mathbb{E}_{z\sim P(z)}[\log(1 - D(G(z)))]$ encapsulates the discriminator's objective in a Generative Adversarial Network (GAN). The first term, $\mathbb{E}_{x^*\in\text{Data}}[\log D(x^*)]$, represents the goal of maximizing the probability that the discriminator D correctly identifies real images x^* from the dataset as real. The second term, $\mathbb{E}_{z\sim P(z)}[\log(1-D(G(z)))]$, reflects the need to maximize the probability of correctly identifying fake images produced by the generator G as fake. Overall, the discriminator aims to distinguish accurately between real and generated images, enhancing its ability to detect generated images from the generator.

4.1.2 Question 2

The generator G in a Generative Adversarial Network should transform the input noise distribution $P(z)$ into a distribution that closely mimics $P(X)$, the distribution of real data, effectively generating synthetic data that is indistinguishable from actual data.

4.1.3 Question 3

To address gradient saturation issues in GAN training, the *true* equation for the generator's objective is modified from the original GAN formulation to $\max_G \mathbb{E}_{z\sim P(z)}[\log D(G(z))]$. This approach focuses on maximizing the probability of the generator's outputs being classified as real by the discriminator, providing stronger gradients for more stable and effective training.

Our implementation focuses on a modified DCGAN architecture tailored for generating 32×32 pixel images. By adjusting various hyperparameters, experimenting with different initializations, and modifying network training losses, we aim to understand the intricacies of GANs, including their potential and limitations. From generating digits to exploring interpolations between noise vectors and beyond, this exploration is poised to deepen our comprehension of generative models and their application to diverse datasets, including the likes of celeba or CIFAR-10, and potentially scaling up to 64×64 images.

4.1.4 Architecture of the networks

Through hands-on experimentation guided by the instructions in the provided notebook, one is expected to witness the GAN's ability to produce increasingly realistic images, an endeavor that not only enriches our understanding but also pushes the boundaries of what's possible in generative AI.

4.1.5 Question 4

The training of the Generative Adversarial Network (GAN) initiates with image generation from a baseline of noise, progressively refining towards more distinct visuals. Early generations yield rather blurred images with figures that are barely perceptible. Advancing through several hundred iterations, we observe a darkening background and a solidification of the digits' outlines. At the 600-iteration mark, the lines of the digits become significantly sharper.

In terms of the loss function, the generator initially faces challenges in creating images that convincingly resemble real ones. As the training advances, the generator begins to produce images with increasing realism, as indicated by its loss.

However, the stability of the algorithm appears to be suboptimal, evidenced by frequent peaks in the loss curves, suggesting variable performance throughout the training process.

Diversity of the generated images also presents as an area of concern. An analysis of a sample of 65 images reveals a distribution where the digit '0' appears 8 times, '1' appears 4 times, '2' is missing entirely, '3' appears 6 times, '4' once, '5' and '6' each appear 8 times, '7' appears 7 times, '8' thrice, and '9' stands out with 16 instances. Moreover, 5 images feature digits that are unrecognizable, highlighting challenges with the diversity and accuracy of generated images.

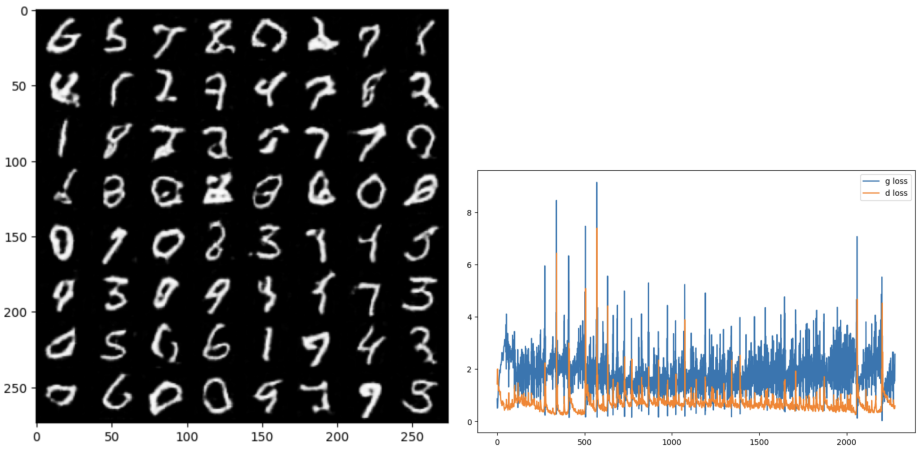

Figure 4.1: Results of GAN training

4.1.6 Question 5

In this section, we will conduct a series of experiments with the Deep Convolutional Generative Adversarial Network (DCGAN) architecture. These experimentations are designed to deepen our understanding of the DCGAN's functionality, highlighting both its strengths and limitations. Through this exploratory approach, we aim to gain comprehensive insights into the practical aspects of the DCGAN, observing how it performs under various conditions and scenarios, and identifying areas where improvements might be needed.

Decreasing ndf

Decreasing `ndf` significantly and observing a decline in the discriminator's performance in a Generative Adversarial Network (GAN) typically results in easier but lower-quality image generation by the generator. While this might initially ease the generator's training, it ultimately leads to poorer overall results, as the generator fails to improve adequately without effective adversarial feedback from the discriminator. This imbalance often results in less realistic and diverse outputs, underscoring the importance of maintaining a balanced adversarial dynamic between the generator and discriminator.

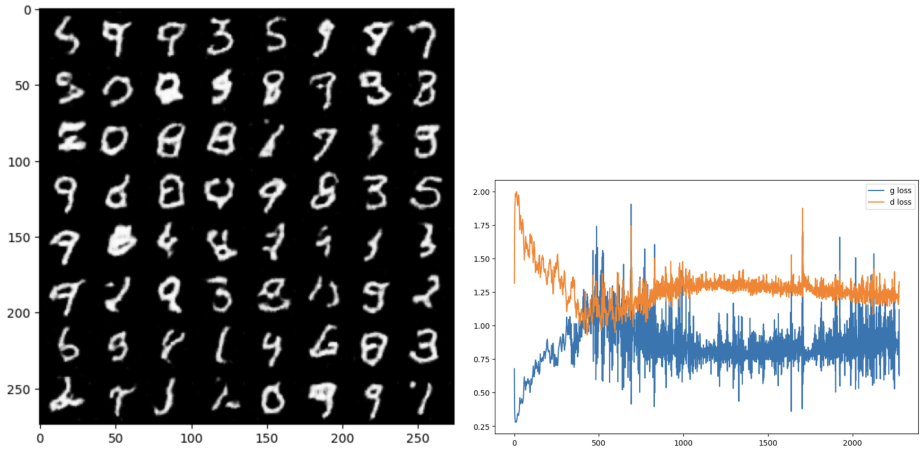

Figure 4.2: Results of GAN training with reduced ndf

Pytorch default initialization

Custom weight initialization appears to confer greater stability in GAN training, leading to a steady discriminator learning curve and preventing the generator from being dominated. Conversely, the default PyTorch initialization may lead to training instability and suboptimal generator performance.

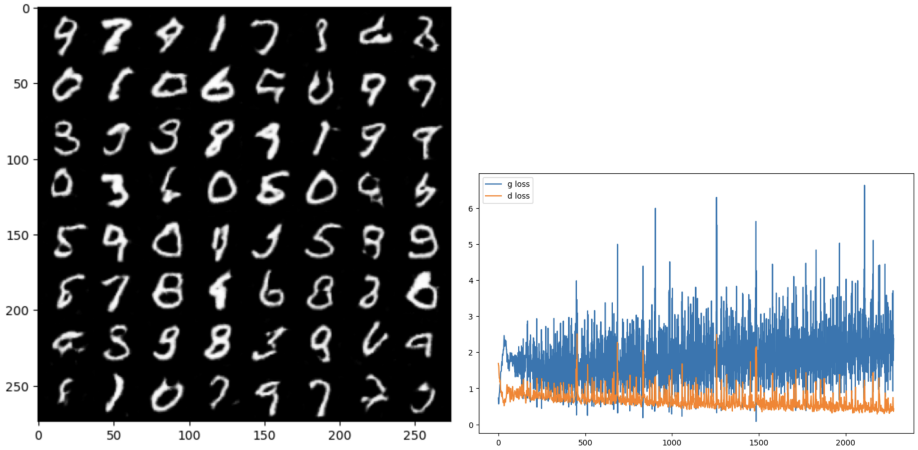

Figure 4.3: Results of GAN training with Pytorch's default initialisation

Slightly increasing the learning rate of the generator

Increasing the generator's learning rate slightly induces more volatile training dynamics, as reflected by a higher generator loss, indicating challenges in fooling the discriminator effectively.

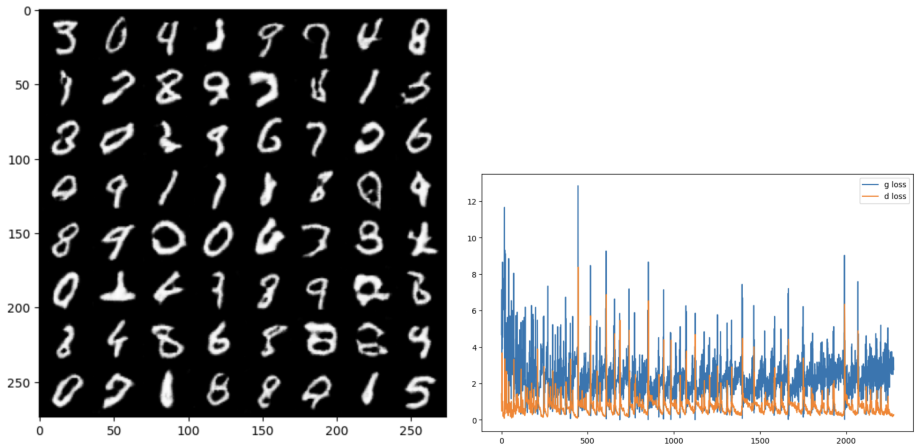

Figure 4.4: Results of GAN training by slightly increasing the learning rate of the generator

Learning for longer

In GAN training, more epochs *do not* guarantee improved results and may lead to overfitting or mode collapse.

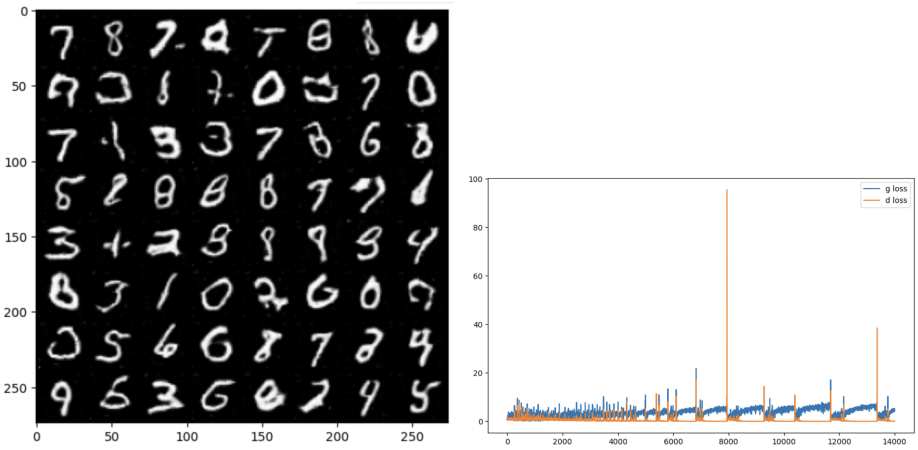

Figure 4.5: Results of GAN training with more epochs

Increasing significantly nz

Increasing the noise vector size from 100 to 1000 dimensions did not enhance the generated image quality, indicating that a larger noise input space does not inherently lead to improved GAN outputs. This outcome suggests the necessity for a refined generator architecture or enhanced training procedure tuning. Determining the optimal noise vector dimensionality requires empirical experimentation, aiming to balance model complexity with practical performance.

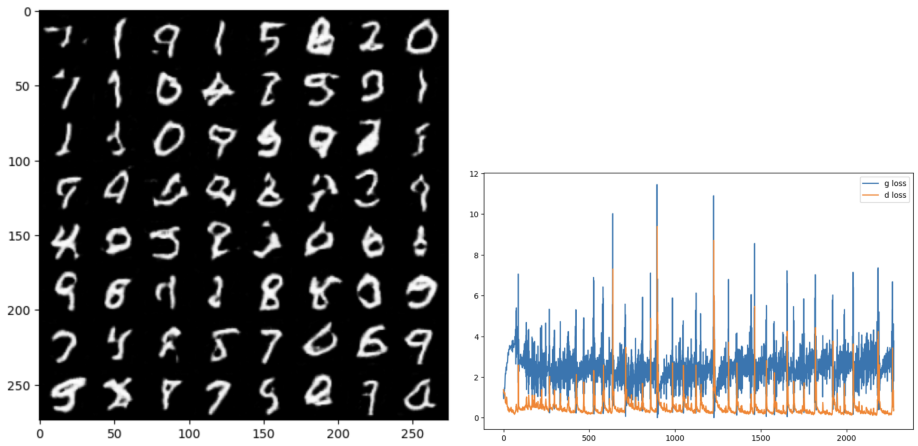

Figure 4.6: Results of GAN training with increased nz

4.2 Bonus - Conditional Generative Adversarial Networks

Generative Adversarial Networks (GANs) can be extended to conditional models if both the generator and the discriminator are conditioned by an additional factor, y as suggested by Mirza & Osindero (2014). For instance, in the context of MNIST, this condition could

be the class label incorporated into the networks.
Similarly, in facial image generation, attributes like wearing glasses or hair color can be included. The condition might also be in the form of text or another image.
Figure 4.7 illustrates the basic concept of a conditional GAN.

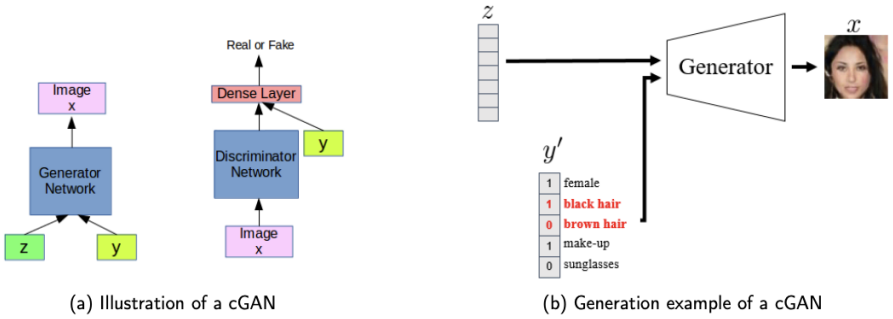

Figure 4.7: Conditional GAN (cGAN) concept

Conditional GANs (cGANs) allow for more directed and controlled generation processes, making them a versatile tool in various applications including image completion, style transfer, and more.

In our implementation, we will focus on a specific variant of cGANs known as cDCGAN (conditional Deep Convolutional GAN). The generator in our network, denoted as $cG(z, y)$, will generate an image from a noise vector z and an attribute vector y associated with the image $x^* \in$ Data. The discriminator, denoted as $cD(x, y)$, aims to differentiate between real images x^* and generated images $\tilde{x}$.

The architecture of both the generator and the discriminator will be designed to effectively fuse the label information with the input data, facilitating the generation of images corresponding to specific labels. Our approach includes branching and concatenation techniques, as detailed in Figure 4.8, to integrate label information into the image generation and classification process.

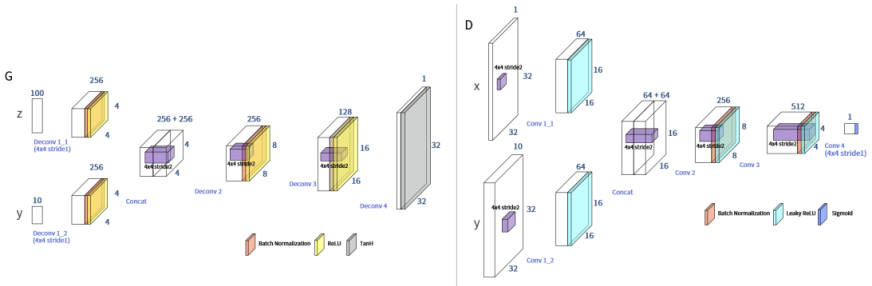

Figure 4.8: The cDCGAN to implement

4.2.1 Question 6

Conditional GANs (cGANs) specialize in generating data tailored to specific conditions. In this case, the condition is the generation of particular digits, as denoted by number 'y', in the MNIST dataset. Figure 4.13 showcases the progression of digit generation using cGANs. Notably, the digits produced by cGANs demonstrate superior quality compared to those generated by standard GANs. This improvement can be attributed to the additional conditioning information supplied to the cGAN, which likely facilitates the generator's learning of the training data's distinct characteristics, resulting in more realistic outputs.

The hypothesis here is that conditioning on the class label enables the cGAN to generate more authentic images of specific digits, as it gains a refined understanding of each digit's appearance. Furthermore, the diversity in the representation of the generated digits is more pronounced, possibly due to the cGAN's access to expanded information. This results in a more varied and consequently more engaging set of generated data.

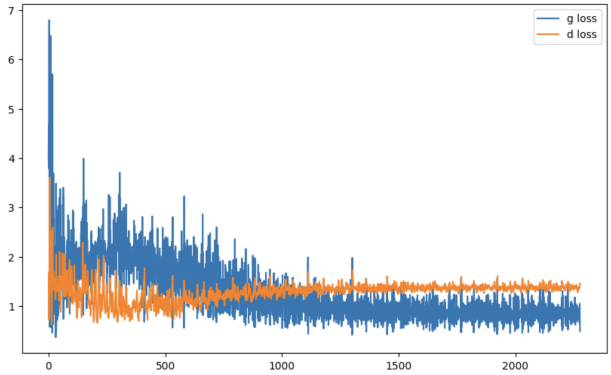

Figure 4.14: Figure representing the evolution of the loss of the cGAN with default parameters

Figure 4.14 presents the evolution of the loss during training. The loss curve of the cGAN is observed to be more stable, characterized by fewer significant spikes and steadier convergence, indicating a more consistent learning process

4.2.2 Question 7

Removing the conditioning vector y from the discriminator's input in a Conditional Generative Adversarial Network (cGAN) significantly alters its functionality. The vector y provides essential conditioning information, enabling the network to generate data specific to a particular condition or class. Without access to y, the cGAN loses its ability to tailor the generation process to specific attributes or conditions. Consequently, the network reverts to functioning like a standard GAN, lacking the capacity to produce conditioned, targeted outputs. Therefore, omitting the vector y from the discriminator in a

cGAN setup is generally not advisable, as it negates the primary advantage of conditional generation.

4.2.3 Question 8

As observed in the previous sections, the evolution of the loss in the Conditional GAN (cGAN) framework is considerably steadier and more stable compared to the unconditional case, resulting in improved outcomes. This enhancement can be attributed to the additional information provided by the conditioning vector y in the cGAN. This vector effectively guides the learning process, allowing the network to develop more accurate and representative models of the numbers. By incorporating specific conditions or attributes, the cGAN is able to focus its learning more effectively, leading to a more structured and stable convergence and superior generation of targeted outputs.

4.2.4 Question 9

In this section, we aim to analyze how variations in the noise vector z influence the diversity of data generated by the Conditional GAN. Figure 4.15 displays these results, illustrating that different noise vectors (represented by each column) lead to notably diverse outputs in the generated numbers. This diversity suggests that the choice of the noise vector z plays a crucial role in 'controlling' the variation in the generated data. In the context of generating handwritten digits, for instance, z can be seen as a tool for introducing variety in handwriting styles. By altering z, we effectively navigate through different representations of the numbers, showcasing the Conditional GAN's ability to produce a wide range of distinct yet plausible outcomes based on the same conditioning input.

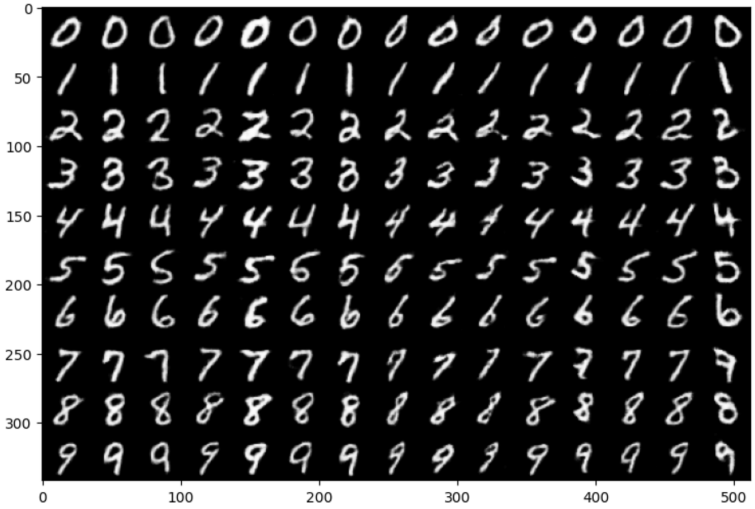

Figure 4.15: Figure representing the result of the variation of the noise vector z

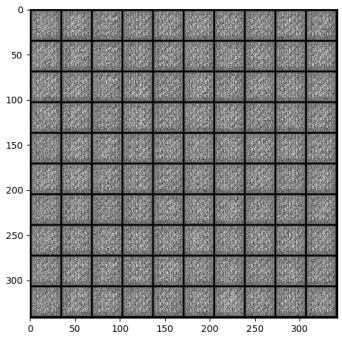

Figure 4.9: Iteration 0

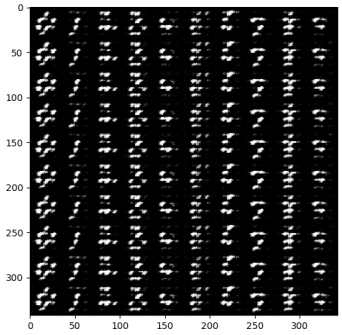

Figure 4.10: Iteration 200

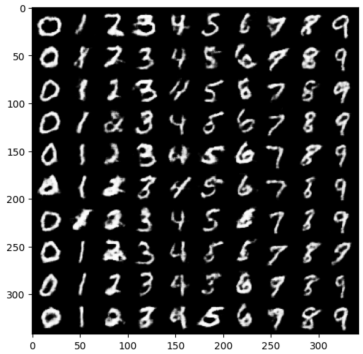

Figure 4.11: Iteration 1000

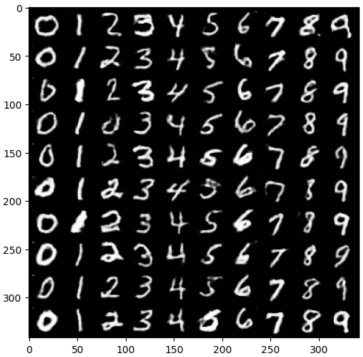

Figure 4.12: Iteration 2000

Figure 4.13: Figure representing the result of the DCGAN with the default settings