RDFIA part 2

Raouf Toukal 21213774
Keryan Chelouche 28607835

November 2023

1 Transfer Learning

1.1 Context

In this report, we explore the utilization of transfer learning, a method that allows us to apply the power of a pre-trained, large-scale neural network to a variety of tasks. Specifically, we use the VGG network, initially trained on the comprehensive ImageNet dataset, for feature extraction in the context of the 15 Scene dataset. This approach showcases the versatility of transfer learning, enabling us to leverage the sophisticated learning of a massive network like VGG for distinct and specialized tasks, thereby solving a range of challenges with minimal additional training.

1.2 Questions and Answers

1.2.1 Section 1 – VGG16 Architecture

Question 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

Answer: We will ignore the convolution part of our network and focusing only in our fully connected part of the network after flattening our image in the last pooling layer we will have a vector of size
$7 \times 7 \times 512 = 25\,088$
the first layer have 4096 output so we will have
$(25\,088 + 1) \times 4096 = 102\,764\,544$ param.
the second layer we have also 4096 output so we will have
$(4096 + 1) \times 4096 = 16\,781\,312$ param
the last layer have 1000 output so we will have
$(4096 + 1) \times 1000 = 4\,097\,000$ param
in total we will have $123\,642\,856$ parameters

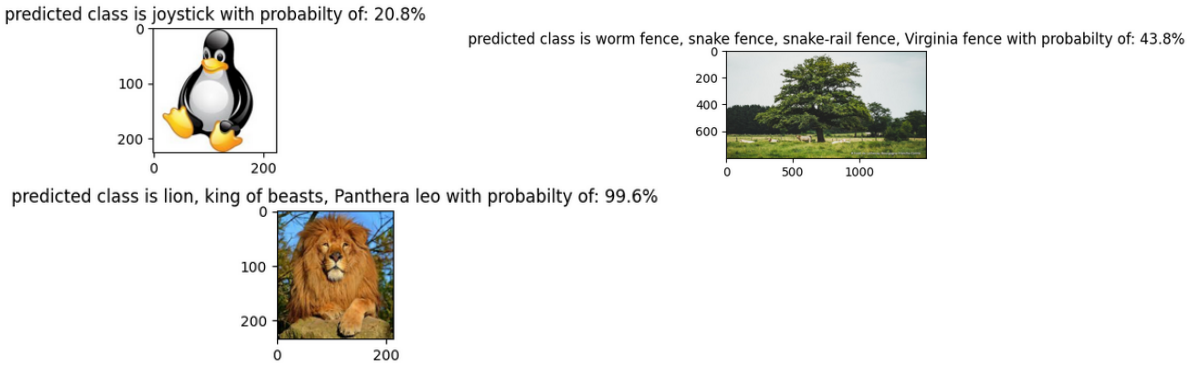
Question 2: What is the output size of the last layer of VGG16? What does it correspond to?

Answer: the size of last layer of VGG16 is 1000, This corresponds to the number of class scores for the ImageNet competition. and it correspond to the logits, that are unnormalized scores that the neural network predicts for each class. They can be any real number, not just between 0 and 1. we can apply the argmax and predict the class without the softmax in inference but in learning we combine softmax to have probability distribution and a CE.

Question 3 (Bonus): Apply the network on several images of your choice and comment on the results.

Answer : We observe mixed results with models trained on ImageNet, especially when they encounter image domains that differ from those typically found in ImageNet. For instance, the models tend to perform poorly with images like "Linux logos" or "Minions," which are not common in the ImageNet database. On the other hand, they yield better results with images of categories similar to those in ImageNet, such as "tigers" and "lions."

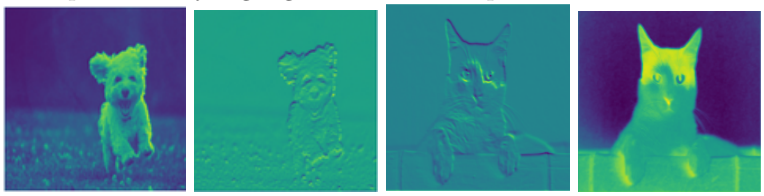
However, even within domains that resemble those in ImageNet, the models sometimes struggle with more complex topology. A prime example of this is with images of "trees." Despite being a category represented in ImageNet, the complexity of such images can lead to inaccuracies in model predictions, as illustrated below.



Question 4 (Bonus): Visualize several activation maps obtained after the first convolutional layer. How can we interpret them

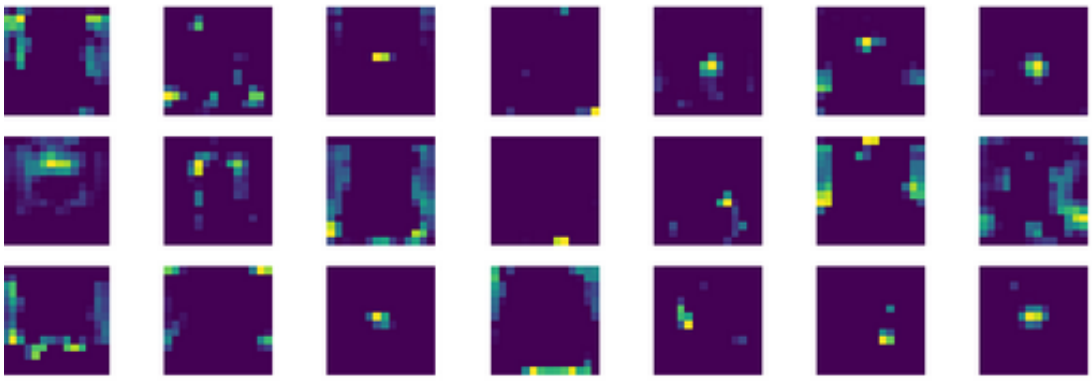
Answer : We observe that each filter in the model creates a unique activation map, each highlighting different characteristics of the input image. For example, in the first map below, the "hot" areas correspond to the dog in the image. In the second map, the focus is on the lines and curves that form the dog's shape, including facial features and the ground. The third map is similar to the second, but the subject of the image is a cat. Finally, in the fourth map, the "hot" areas are predominantly where the cat and the couch it's sitting on are located, with the cat's head being

more prominently highlighted than other parts.



The features of the second layer predominantly contain maps with cold (less active) areas, and only a few with hot (more active) zones, as illustrated by the examples below. This observation is due to the fact that our new feature maps are more robust than the first layer, as they combine features, making them more specialized. For instance, to extract lines or edges, the entire image is represented in cold (inactive) zones, with only the lines or edges highlighted in hot (active) zones. This contrasts with the first layer, where the entire image was predominantly in hot zones.

For the last convolutional layer, the feature maps become highly specialized to the point where they highlight only a small portion of the image as hot (active) zones and targeted, leaving the majority of the image in cold (inactive) areas.



1.2.2 Section 2 – Transfer Learning with VGG16 on 15 Scene

Question 5 : Why not directly train VGG16 on 15 Scene?

Answer : Opting not to train VGG16 directly on the 15 Scene dataset is primarily influenced by the dataset's limited size. VGG16 is a complex architecture with a vast capacity, originally trained on the extensive ImageNet dataset. Deploying such a powerful model on the relatively small 15 Scene dataset could result in overfitting, as the model may adapt too closely to the limited training data, learning idiosyncratic details that do not generalize well to unseen images. Additionally, the computational resources required to train VGG16 from scratch are substantial. Training deep neural networks involves significant computational power and time, potentially incurring high costs and resource consumption that may not be justifiable for smaller datasets. By leveraging a pre-trained VGG16, one can utilize the rich feature representations that the network has already learned from the diverse and extensive ImageNet dataset. These features are often generalizable across various image recognition and can provide a strong foundational knowledge, enabling the model to learn the new task with greater efficiency and less risk of overfitting.

Question 6: How can pre-training on ImageNet help classification for 15 Scene?

Answer : Citing our model's training on a vast dataset like ImageNet, its feature extraction capability stands out for its performance. This proficiency is attributed to several key factors. Firstly, the model has learned a rich representation of visual features, ranging from basic elements like edges to more complex structures relevant to various objects and scenes. Furthermore, this versatility shines when applied to scene recognition tasks like those in the 15 Scene dataset. Additionally, this approach significantly reduces the risk of overfitting, as the model leverages pre-learned features, promoting effective generalization and resilience to dataset noise

Question 7: What limits can you see with feature extraction?

Answer : If our base model was trained on data from a different domain than the one we are using for our target, this would result in a "domain gap". Some features in our images might not be captured because, during the training of VGG, it did not take them into account, and therefore its filters ignore these features. Additionally, the features extracted might not be relevant for our classification task.

Question 8: What is the impact of the layer at which the features are extracted?

Answer : In a convolutional neural network, each layer processes a different level of abstraction. The initial layers focus on simple features like edges, colors, and textures. As we progress to deeper layers, the network identifies more complex and abstract elements, such as patterns and object components. The final layers achieve the highest level of abstraction and are closely linked to specific tasks for which the network was designed, such as recognizing specific classes in image classification.
Therefore, when choosing shallower layers, we may fail to capture information that could be relevant later, leading to underfitting. Conversely, opting for very deep layers can result in capturing overly specialized information. If our model was trained on a dataset vastly different from ours, this could lead to poor results due to a mismatch in feature relevance, and there's also a risk of overfitting.

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

Answer : We duplicate our channel until we have three.

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

Answer : Yes, we can convert our network to a fully convolutional one by removing the entire fully connected (dense) portion of our old network and replacing it with a new convolutional network layer tailored to our specific problem.

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

Answer : Let's begin by studying the impact of the depth of the features extracted on the accuracy and the execution time.

Until now, we extracted after the (4) ReLU layer in the classifier. We will try extracting earlier : after the (1) ReLu layer of the classifier. We will also try to extract during the convolutional part of the network : At the very end (30) MaxPool, and before some MaxPool layer : after the (29) and (22) ReLU layer of the features.

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

Figure 1: VGG 16 architecture.

The optimal results in our tests were achieved with features extracted after the classifier's first ReLU layer, reaching an accuracy of 90.92% in just 39.30 seconds which is higher than the performance obtained with features extracted after the second ReLU layer. In contrast, extracting features even earlier in the network, before the convolutional section's end, not only reduced accuracy but also significantly increased execution time to 193.26 seconds. Moreover, extracting at the (22) ReLU layer led to a system crash due to RAM limitations, highlighting the computational challenges of handling data from earlier network layers. However, extracting at the end of the feature network still give good results: 89.54% in 45 seconds.

Those computational difficulties are expected because when extracting before all of the MaxPool layers, the number of features is way bigger : 4096 after the first ReLU in the classifier versus $7 \times 7 \times 512 = 25088$ at the end of the features network, $14 \times 14 \times 512 = 100352$ for (29) ReLU features layer or even $28 \times 28 \times 512 = 401408$ for the (22) ReLU features layer.

Layer	Accuracy (%)	Execution Time (seconds)
(4) ReLU classifier	88.61	39.82
(1) ReLU classifier	90.92	39.30
(30) ReLU features	89.55	45.07
(29) ReLU features	87.54	193.26
(22) ReLU features	N/A	N/A

Table 1: Depth of extraction comparison

Let's see if some methods of dimensionality reduction could help reduce the execution time without sacrificing accuracy. We are using Primary Components Analysis (PCA) on the 1st ReLU classifier layer and the (29) ReLU features layer.

Layer	PCA (100 features)	Accuracy (%)	Execution Time (seconds)
(1) ReLU classifier	No	90.92	39.30
(1) ReLU classifier	Yes	90.08	37.17
(29) ReLU features	No	87.54	193.26
(29) ReLU features	Yes	86.57	54.81

Table 2: PCA comparison

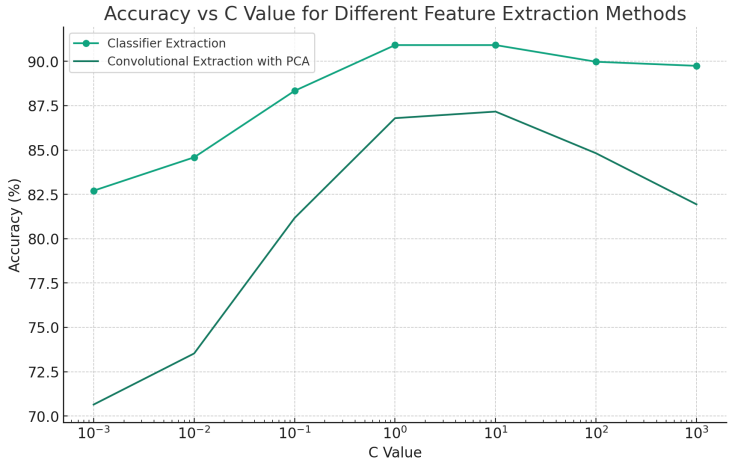
We see in both case a drop of less than a percent in accuracy. However, if the decreases of execution time is only 2 seconds for the classification layer, the execution time of the feature extraction at the convolution part decreases to a quarter of what is was originally : from 193.26 to 54.81 seconds.

PCA doesn't really seems to be worth it if we extract features in the classifier but seems almost mandatory if we extract features in the convolutional part of the network.

Finally, let's try to optimise to find the optimal C to push our accuracy as high as possible. Let's also see if this optimal C is the same regardless of the depth of the extraction.

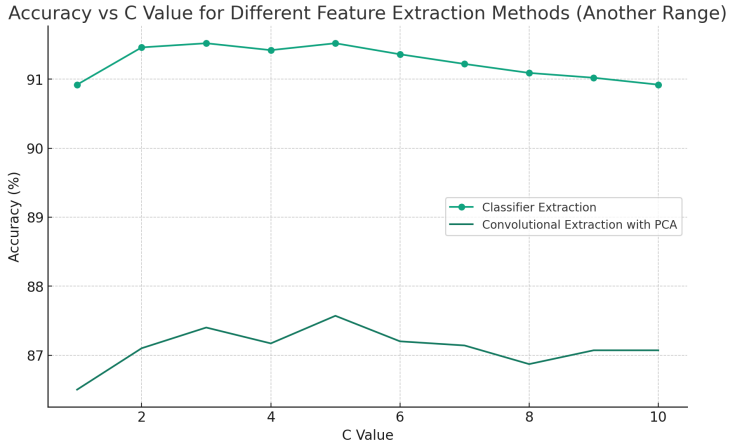
We begin by a first experiment using a vast array of values of C to see what seems to work the best.

For both methods the best C seems to be between 1 and 10. We get very similar performance with C = 1 and 10 in both methods (identical for the classifier extraction), but slightly better results with 10 for the Convolutional extraction. We can also see that the value of C has more impact on the performance of the convolutional extraction with PCA : The difference of accuracy between the worst and best C being 15% (71%



to 86%) for the convolutional method but only 8.5% (82.5% to 91%) for the classifier method.

Let's now test different values of C between 1 and 10 to find an optimum.



C = 5 seems to be optimal for both with a final accuracy of 91.52% for the Classifier extraction and 87.57% for the Convolutional extraction

2 Visualizing Neural Networks

2.1 Context

Neural networks, notably in the realm of computer vision, are frequently perceived as 'black boxes' due to their complex and often opaque decision-making processes. This lack of transparency makes it challenging to interpret and explain the results they produce. Addressing this issue, the field of neural network visualization has emerged as a vital area of research, offering insights into the inner workings of these networks. Our project focuses on employing and evaluating several recent visualization techniques specifically designed for convolutional neural networks (CNNs). This will enhance our understanding of how these networks process visual data in image recognition and analysis, thereby providing clearer insights into the mechanisms of artificial intelligence.

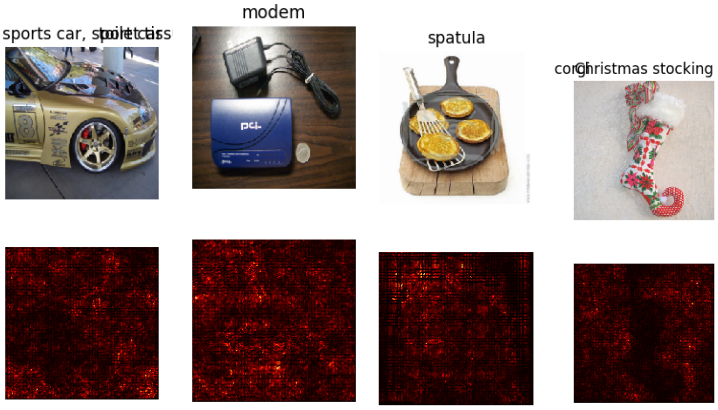
2.2 Questions and Answers

2.2.1 Section 1 - Saliency Map

Question 1 Show and Interpret the Obtained Results

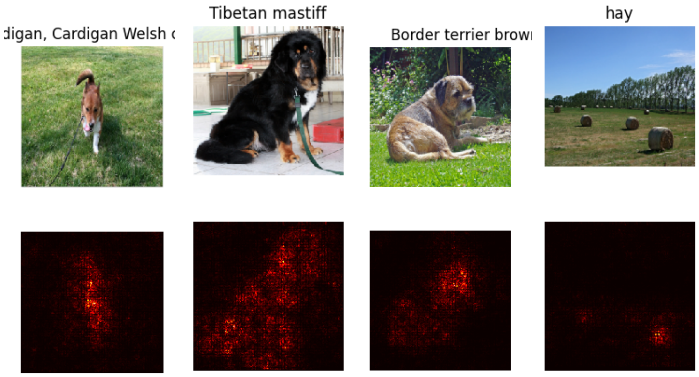
Answer We notice that our model is not very efficient for object detection. This inefficiency is evident in some images where our saliency map does not perform well. The saliency maps are expected to highlight the regions of interest in the images that are crucial for object detection. However, in our case, these maps do not consistently identify the relevant areas, indicating potential issues with either the model's training or its inherent architecture.

To illustrate this, consider the following images and their saliency maps:



We can observe that across the three saliency maps, the focus appears to be entirely random and does not concentrate on the object itself for the

3 first saliency map. And for the last saliency map related to Christmas stockings, it is evident that our model's prediction was influenced by the surrounding environment rather than the actual structure of the stockings



We note that for the first three saliency maps, the objects are detected through their own characteristics, which is a positive sign. However, for the fourth saliency map, while some objects are correctly highlighted, smaller or more distant elements in the image are not detected, which indicates a limitation in our model's effectiveness.

we can that our model is biased.

Question 2 Discuss the limits of this technique of visualizing the impact of different pixels

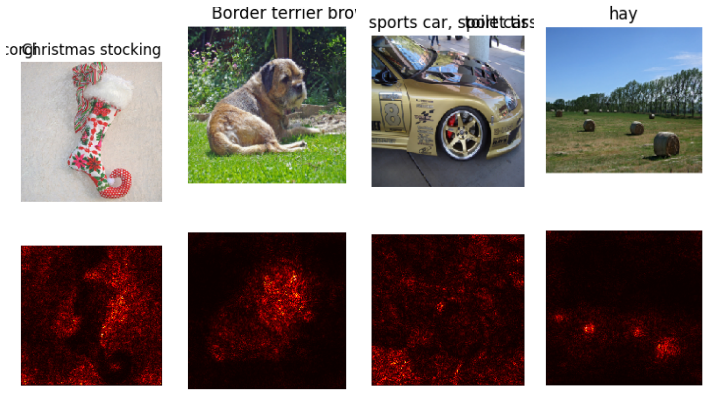
Answer Saliency maps, a tool in deep learning, have several limitations. Firstly, **Subjective Interpretation** can be an issue; different individuals might interpret the same map differently. For instance, one might consider a feature important, while another sees it as irrelevant. Secondly, **Issues with Complex Images** arise; cluttered or intricate images can lead to complex and hard-to-interpret maps. Thirdly, **Lack of Causal Relationships** means that while maps show influential areas, they don't explain the 'why'. **Noise Sensitivity** is another limitation, where noisy data can lead to misleading maps. Lastly, **Gradient Evaporation** in deep networks can result in less informative maps, as the gradient signal diminishes during backpropagation.

Question 3: Can this technique be used for a different purpose than interpreting the network?

Answer : If our model detects objects based on their shape, we can use saliency maps with a bit of smoothing for segmentation.

Question 4 (Bonus): Test with a different network, for example VGG16, and comment.

Answer : We observe that the intensity of our saliency maps is greater compared to the first model, and VGG16 performs a bit better than the initial model. This suggests that the VGG16 architecture is more effective in highlighting relevant features in the images, leading to more pronounced and potentially more accurate saliency maps.

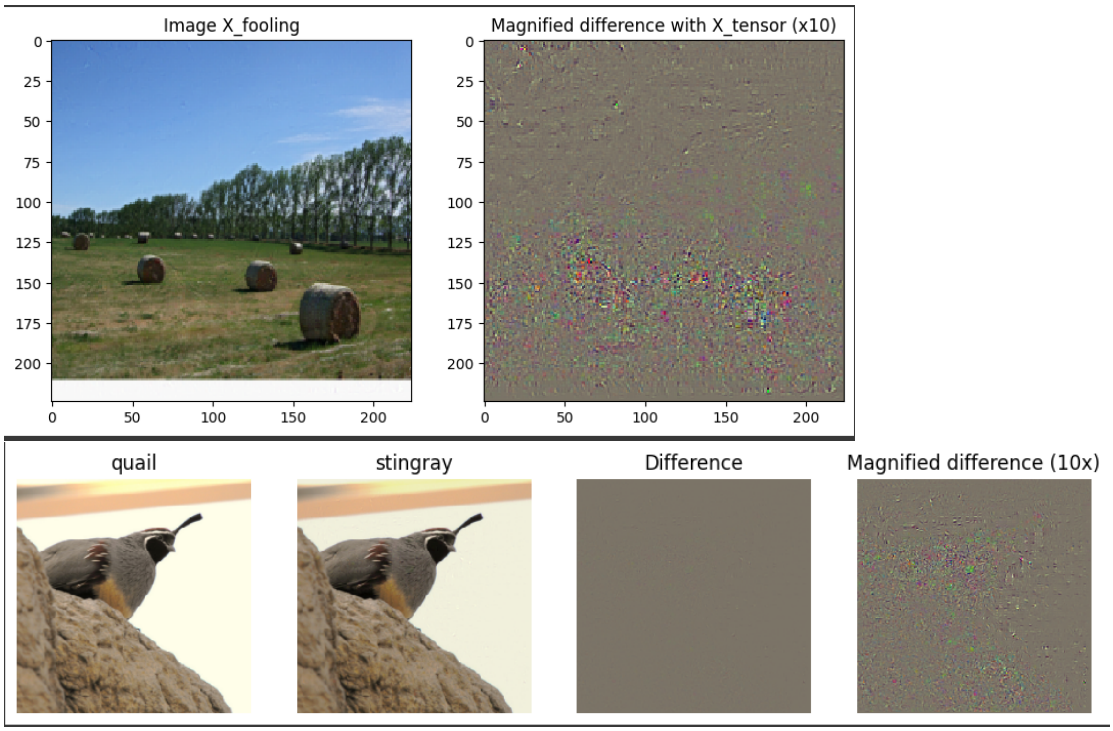


We can also note that in the saliency map of the sports car photo, some details like the road are discernible, albeit with some noise. This represents a slight improvement over the first model, indicating that VGG16, while still not perfect, is better at capturing and highlighting relevant features in the image, albeit with some residual noise.
For the detection of distant objects ("hay") in our images, we observe that VGG16 detects more such objects compared to the first model. However, objects that are too far away and thus appear small in the image are still challenging to detect accurately. Despite this limitation, VGG16 demonstrates an improvement over the initial model in terms of detecting and highlighting these distant objects.
we have also the same problem for detection of christmas stockings.

2.2.2 Section 2 – Adversarial Examples

Question 5: Show and interpret the obtained results.

Answer : We can observe that the image generated by this method is not visibly different from the original image. This technique shifts our image, initially classified as class 'i', towards the boundary of class 'j' through gradient ascent. Notably, the subtle changes are usually concentrated in areas that contributed most to the initial classification as object 'i'. This is logical because minimal alterations are more effective when applied to the most influential regions for the detection of the original object.



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

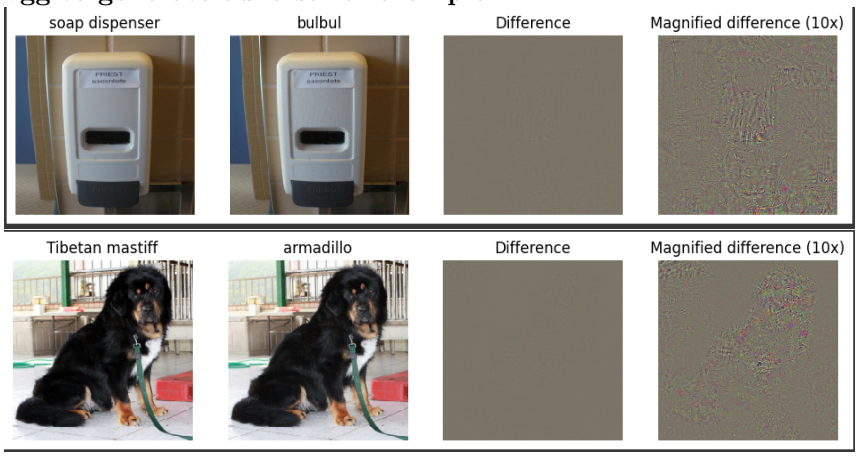
Answer : In practice, the use of adversarial examples in convolutional neural networks (CNNs) can have significant consequences. These examples can exploit vulnerabilities in CNNs, leading to misclassifications. This is particularly concerning in critical applications like autonomous vehicles, healthcare, and security systems, where reliability is paramount. Adversarial attacks can deceive systems into making incorrect decisions, posing safety and security risks. Thus, understanding and addressing these vulnerabilities is crucial for ensuring the robustness and trustworthiness of AI systems in real-world scenarios.

Question 7 (Bonus): Discuss the limits of this naive way to construct adversarial images. Can you propose some alternative or modified ways? (You can base these on recent research).

Answer : For a given image and a target class 'j', generating an adversarial image using a naive approach will always result in the same altered image. This implies a lack of generalization across different possible examples, indicating a limitation in the method's ability to produce varied and broadly effective adversarial examples under different class 'j' and same image.

we will use the "growing sphere" method for generating adversarial images, we start from an initial image in a space of dimension 'd' (height * width * channels). We gradually generate a sphere around our image, and if a generated point is classified as class 'j', we stop the process. Then, we take these generated points as 'g' and modify their attributes as much as possible to resemble the attributes of our base image while still being classified as class 'j'. This approach allows generate adversarial example with some random factor.

vgg to generate adversarial example



We observe that image generation with VGG is more precise than with our initial model because VGG extracts patterns from objects more effectively. This enhanced pattern recognition ability of VGG leads to more accurate and detailed representations in the generated images, highlighting its superiority in capturing the nuances of object features compared to the initial model.

2.2.3 Section 3 – Class Visualization

Question 8: Show and interpret the obtained results.

Answer : We notice that with each iteration, patterns are added to our image to make it more detectable as class 'i'. The changes are primarily applied to the areas that most influence the image. For instance, in the case of the 'hay' image, when trying to maximize the classification as 'snail', we initially modified the hay. After several iterations, the entire image was altered, giving more importance to changing the hay objects to snail-like features, as these hay objects were the most influential in our image according to our first experiment.

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

Answer : As we increase the number of iterations, the image transforms into a new one with more refined and clearer patterns. Reducing the regularization term takes the original image more into account, meaning we add patterns to maximize class 'i' while considering the topology of the original image. Conversely, increasing the regularization term generates an image that focuses solely on patterns, neglecting the original image. This balance between pattern maximization and adherence to the original image's structure is crucial in class visualization techniques.

Question 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.

Answer : As previously explained, in the case of the 'hay' and 'snail' example, we modify the areas of the image that contribute most to its prediction. In the maximization process for the same class, the strategy remains the same: it starts with the most important areas and then moves to the lesser ones. From our example, we notice that it refines the 'hay' (hot zones corresponding to areas contributing the most) in its own way to maximize pattern detection and also creates new features in the cooler zones.

Question 11 (Bonus): Test with another network, VGG16, for example, and comment on the results.

Answer : We observe that with VGG, for the same number of iterations as our base model, we get a more precise image with added patterns and more significant modifications. This is because VGG captures patterns more effectively. However, with a large number of iterations, since VGG captures more patterns than the initial model, it tends to over-alter the image more than the first model.

3 Domain Adaptation

3.1 Context

In real-world scenarios, obtaining cleanly labeled data can be a significant challenge, often impeded by the extensive time and resources required for dataset creation. The ability to apply training from a domain with accessible, high-quality labeled data to perform effectively in another domain, where such data may not be as readily available, is critically important. This process, known as domain adaptation, is essential for enhancing the flexibility and effectiveness of machine learning models in various and often unpredictable environments. In this project, we aim to implement domain adaptation techniques using the Domain-Adversarial Neural Network (DANN) approach. Our focus will be on adapting from the MNIST dataset, which consists of cleanly labeled grayscale images of handwritten digits, to the MNIST-M dataset, characterized by its

unlabeled, colored, and more complex images. The key objective is to generate domain-agnostic features that enable effective performance on MNIST-M without the need for extensive labeling whithout dropping in performance on MNIST.

3.2 Questions and Answers

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

Answer : our 2 domains in the embedding we learn will be differentiable and so the feature extraction network will no longer be able to create features such that the discriminator will not be able to differentiate.

Question 2: Why does the performance on the source dataset may degrade a bit ?

Answer : the performance on the source may degrade because the model is being adjusted to generalize to the target domain. This process often involves finding a compromise between the source domain specificity and the target domain generalization. As a result, the model might lose some of its ability to perform optimally on the source domain due to this shift in focus towards learning more domain-invariant features that perform well across both domains.

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

Answer : The Gradient Reversal Layer (GRL) typically starts with a small gradient reversal scalar (the value by which the gradients are multiplied during the backpropagation) and gradually increases it to 1 during training. This approach is based on the idea that at the beginning of training, we want the model to focus on learning good features for the classification task without being too constrained by the need to be domain-invariant. As the model starts to learn and fit to the source domain, we increase the scalar to force the model to focus more on learning features that cannot be used to distinguish between the source and target domains, thereby promoting the development of domain-invariant features. The gradual increase allows the model to stabilize and adjust before the full adversarial effect is enforced.

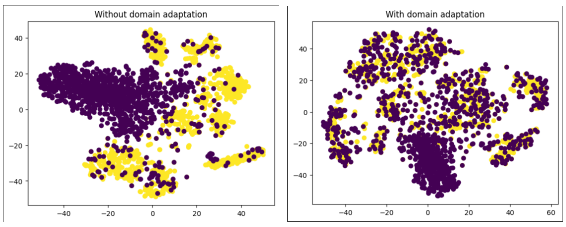
Question 4: Another common method in domain adaptation is pseudo-labeling. Investigate what it is and describe it in your own words.

Answer :

- Feature Alignment : this method try to aligns the source and target domain feature distributions using techniques like Maximum Mean Discrepancy (MMD) minimizes the distance between the mean feature embeddings of both domains or Correlation Alignment (CORAL) aligns their covariances.

- Co-Training : this method train two models on source domain data, using them to generate pseudo-labels for target domain data, and iteratively refining the models with these labels. This method leverages confident predictions to improve the models' performance on the target domain through a process of cross-training and feedback.
- Domain-Invariant Feature Learning : this method train a model to learn features that are effective across both source and target domains. This method aims to reduce the model's sensitivity to domain-specific variations.

3.3 Experiment Analysis

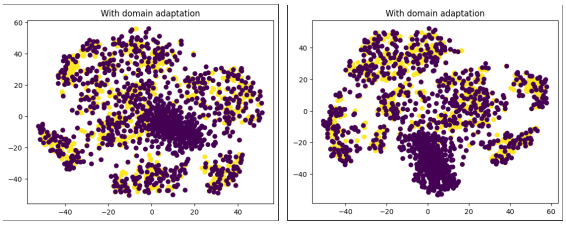


(a) without UDA (b) with UDA

Figure 2: epochs =20

UDA	src Class loss/acc	src Domain loss/acc	trg Class loss/acc	trg Domain loss/acc
with	0.04424 / 98.58%	0.55958 / 100.0%	1.12631 / 75.41%	1.00475 / 0.0%
without	0.02592 / 99.16%	0	1.51877 / 54.07%	0

The UDA method performs better because our model is more robust to domain gap

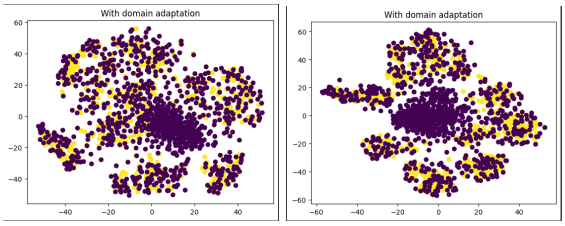


(a) epochs 50 (b) epochs 20

Figure 3: UDA

epochs	src Class loss/acc	src Domain loss/acc	trg Class loss/acc	trg Domain loss/acc
20	0.04424 / 98.58%	0.55958 / 100.0%	1.12631 / 75.41%	1.00475 / 0.0%
50	0.06751 / 97.9%	0.53374 / 100.0%	0.93523 / 78.56	1.03806 / 0.0%

Increasing the number of epochs give us a more efficient model in target data because our CNN merge the two domain more efficiently than with only 20 epochs. However, we lose some performance in source data because of domain merge (reverse gradient)



(a) UDA schedulers provided (b) UDA schedulers per step

schedulers	src Class loss/acc	src Domain loss/acc	trg Class loss/acc	trg Domain loss/acc
provided	0.06751 / 97.9%	0.53374 / 100.0%	0.93523 / 78.56	1.03806 / 0.0%
step	0.04467 / 98.82%	0.50238 / 100.0%	0.85702 / 80.41%	1.0097 / 0.0%

With this method of scheduling we increase even further the performance of our model on the source and target data

Our final performance is then **80.41% on target data** (with 98.82% on source). This is slightly less than the paper results (81.49 % on target data). The cause could be the use of batch normalisation, which we didn't implement (Problem with trigger CUDA and lack of time)

4 Generative Adversarial Networks

4.1 Context

Generative Adversarial Networks (GANs) have marked a significant milestone in the realm of generative vision models, being the first of their kind with a proven ability to deceive human perception. In our project, we aim to delve into the intricacies of GANs by implementing its two critical components: the Generator and the Discriminator. Initially, our focus will be on exploring the unique aspects of training GANs. Following this, we will engage in a comparative analysis, juxtaposing the traditional, unconditional GAN with its more advanced

counterpart, the Deep Convolutional GAN (DCGAN). This comparison aims to shed light on the evolution and enhanced capabilities of GAN architectures in the field of generative modeling.

4.2 Question and Answers

4.2.1 Section 1 – Generative Adversarial Networks

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

Answer : Equation 6 : by trying to optimise the generation of our images such as the discriminator it will confuse them with real ones.
because if $D(G(z)) = 0$ $log(D(G(z))) = -\infty$
and we want to maximize it so we try to push $D(G(z))$ to 1 and we will have $log(D(G(z)))$ close as possible to 0

Equation 7 :
the discriminator tries to differentiate between real and fake images
$log(x) < 0$ if $x < 1$ so
we try to make $D(x^*)$ closer to 1
we try to make $1 - D(G(z))$ closer to 1 equivalent $D(G(z))$ to 0

if we use only equation 6 :
our discriminator doesn't improve, it will be restricted to mediocre results. Consequently, as the discriminator stalls, our generator's development is also limited, lacking the challenge needed to produce high-quality images. The generator easily surpasses the unimproved discriminator

if we use only equation 7 :
The generator's performance is stagnating because the focus is solely on optimizing the discriminator. As a result, the generated images are mediocre. With just a few iterations, the discriminator becomes proficient at distinguishing between real and generated images, leading to its progress plateauing without significant improvement.

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

Answer : As close as possible to $P(Data)$

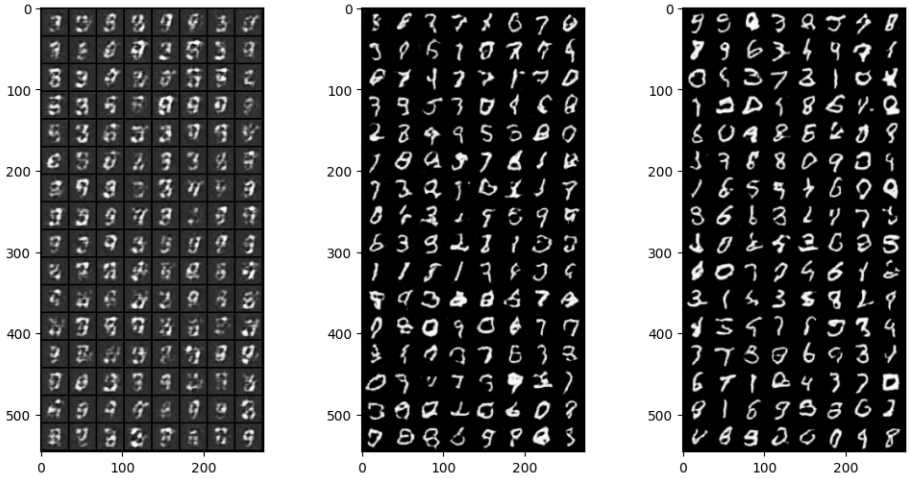
explanation :
When updating the discriminator, we keep the generator fixed and aim to maximize Formula 7, striving to differentiate between real and generated images as much as possible. Thus, we choose the discriminator that most

effectively distinguishes between P(Z) and P(Data). Conversely, when updating the generator with a fixed discriminator, we optimize Formula 6, which is designed to deceive the discriminator. Here, the goal is to select a generator that brings P(Z) closer to P(Data).

Question 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 ?

Answer : $\min_G E_{z\sim p(z)}[\log(1 - D(G(z)))]$

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



(a) iterations = 200 (b) iterations = 1138 (c) End of training

Figure 5: GAN training progress

After only 200 iterations, we can see that the generated image are still very noisy, with the background being gray, not black. Some digits are almost recognisable, especially 9, 8 and 3. Most image are illegible however.

After 1000 iterations, the image is clearer, with a sharply defined black background. More image are recognisable and with more diversity : All digits except 2 seems to be represented. However, most of them are still really rough with a lot of completely unrecognisable digit.

At the end of training, we can see that our generator was able to create a large diversity of digits : from 0 to 9 all digits are able to be generated. However, from this example, some digits seems to be more rare than others : there is almost no 2. Some generation are also still unrecognizable.

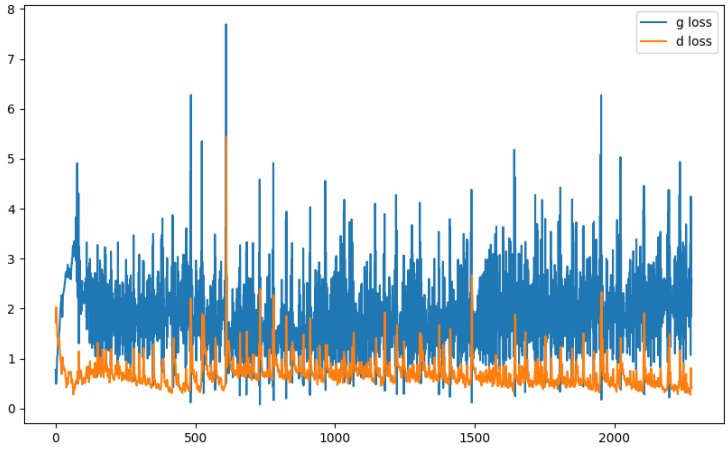


Figure 6: GAN loss

We can see that both loss are extremely noisy, especially the Generator loss. At the beginning, the discriminator loss quickly decreases, while the generator loss quickly increases.

Question 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.

Answer : Let's see what happen when we generate from the results of the interpolation between two noise vector z1 and z2.

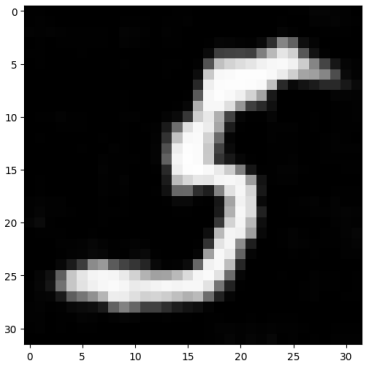


Figure 7: Fake image generated from z1

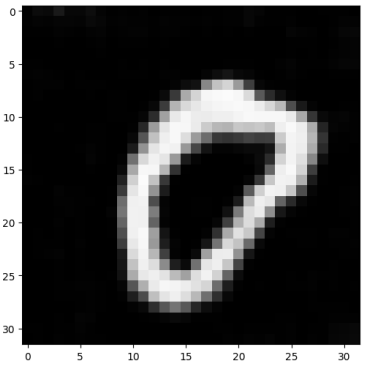


Figure 8: Fake image generated from z2

We can see that the Image 1 progressively morph into Image 2. That shows us that distance in the latent z space is equivalent to a "semantic

Figure 9: Linear Interpolation between z1 and z2

distance" in the X (Image) world, i.e. close z vectors generate similar image.

Let's see the influence of the number of latent dimension (nz).

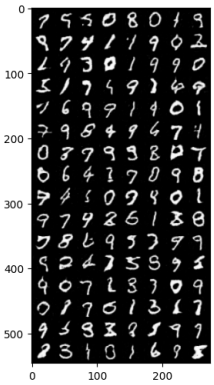


Figure 10: nz = 10

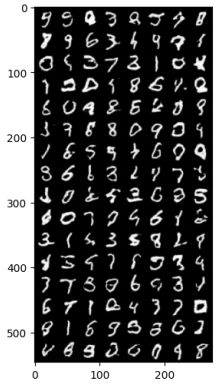


Figure 11: nz = 100

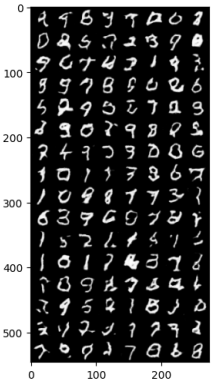


Figure 12: nz = 1000

Surprisingly, the better looking results seems to be with nz = 10. There is less unrecognisable digits than with nz = 1000 and even nz = 100. However, there is less diversity between generation of the same digits. With nz = 10, all 9 looks the same, which is not the case with nz = 100 or 1000.

We still get good result with a smaller nz, let's try to push it to the extreme, with nz = 3 :

As expected, the result look a bit rough. But at least half of the digits are still recognisable. The diversity problem is even worse though, for exemple, almost every 3 looks identical.

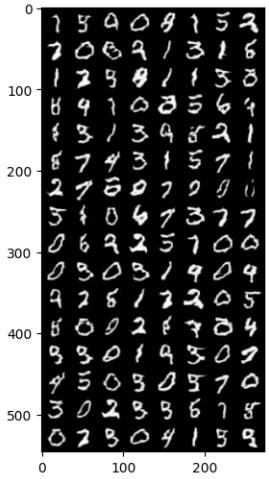


Figure 13: nz = 3

Finally, let's try on another dataset : CIFAR10. For simplicity, we will grey-scale the image. CIFAR10 is a much more complex dataset than MNIST so let's go to 20 epoch.

The results are quite good, however the generated images seems less sharp, with less contrast and more blury than the real ones.

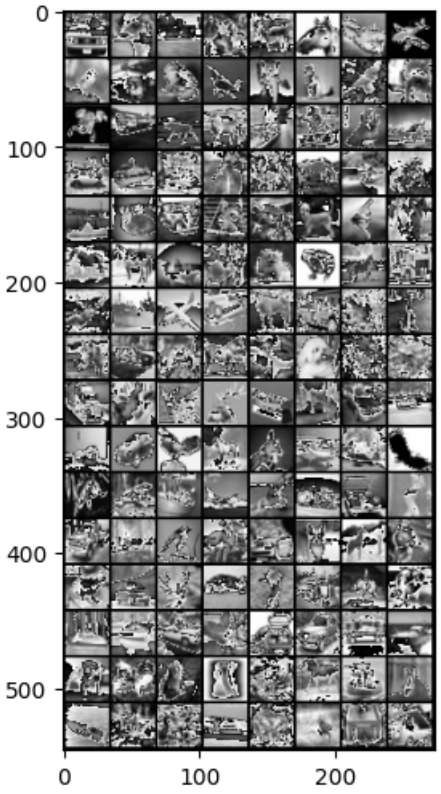


Figure 14: Real Image

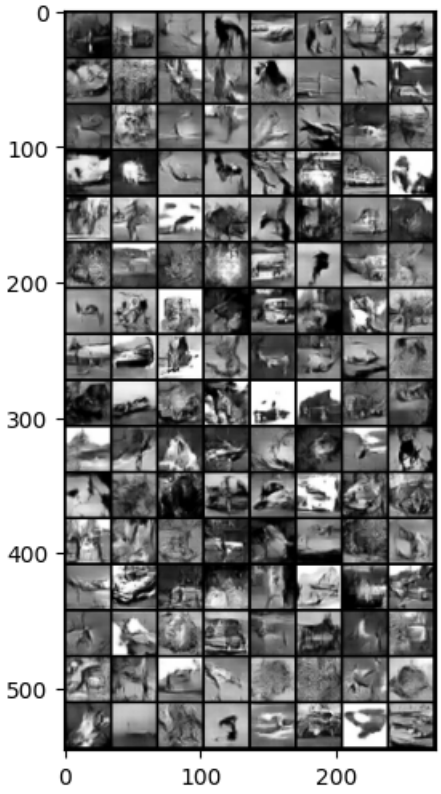


Figure 15: Generated Image

4.2.2 Section 2 – Conditional Generative Adversarial Networks

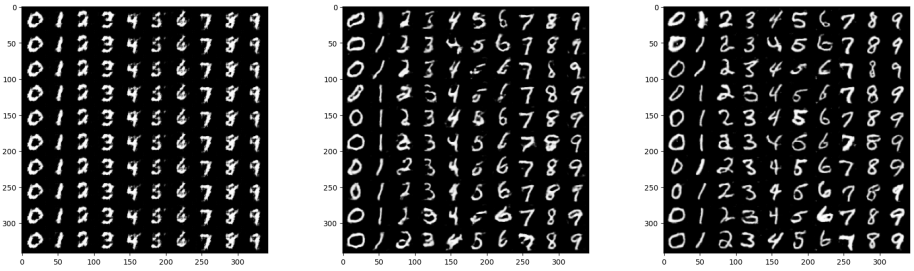
Question 6: Comment on your experiences with the conditional DCGAN.

Answer : After 200 iterations, the background is already clearly black and well defined. Every digit except maybe 2 is already recognisable but still a little rough. However there is a clear lack of diversity between generation of the same digit. They all look the same.

After 1138 iterations, there is more diversity between different generation of the same digit. Also, the digits are not longer blurry. Some of them are still rough though, notably some 2, 4, 5 and 7.

At the end of training, every digit is recognisable with diversity between generations of the same digit.

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



(a) iterations = 200 (b) iterations = 1138 (c) End of training

Figure 16: cGAN training progress

Answer : No, we can't. Without y, the discriminator can only judge the authenticity of the image x (i.e., real or fake) and cannot assess whether x correctly corresponds to a specific condition. This change essentially transforms the Conditional GAN into a traditional GAN.

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

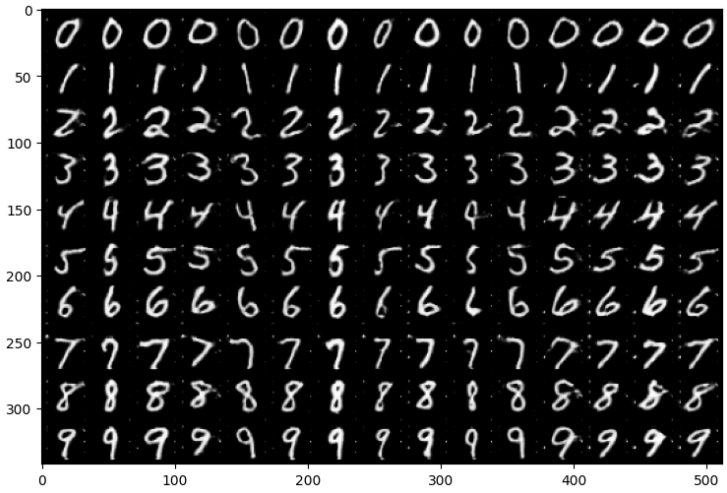
Answer : Our training seems more successful than the unconditional case. Just by looking at the image, in the conditional case, every image is recognisable while in the unconditional case, some were not. Moreover, examining the discriminator's final scores reveals a significant difference: in the conditional case, the accuracy is 56% for real images and 33% for generated ones. In contrast, the unconditional case shows a higher accuracy of 88% for real images and 27% for generated ones. Given our objective for the generator to successfully deceive the discriminator by the end of the training, the conditional approach demonstrates superior performance.

The reason could be more controlled and relevant data generation due to conditioning on specific attributes. This not only improves the learning dynamics between the generator and discriminator but also leads to richer feature representations and a more stable training process. cGANs are particularly beneficial for applications requiring targeted data generation (as MNIST is), as they better capture the nuances necessary for such tasks and reduce issues like mode collapse, resulting in more diverse and higher-quality outputs aligned with specific conditions or classes.

	Unconditional Case	**Conditional Case**
Real Image	88%	56%
Fake Image	27%	33%

Table 3: Comparison of Discriminator's Final Scores

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



Answer : We can see that digit on the same columns looks similarly written, almost as if drawn by the same person. For exemple, all number on column 14 are very chunky while number on columns 8 are very thin. We could then almost interpret z as a "style" of drawing numbers.