RDFIA: Homework 3-a,b,c,d

GALMICHE Nathan PIRIOU Victor

February 20, 2023

Abstract

The performance of neural networks has been improving for a decade now. For instance, the field of image synthesis was revolutionised by the advent of Generative Adversarial Networks (GAN) whose goal is to learn the joint distribution of the network's inputs and outputs. More generally, GAN have fundamentally allowed new ways of approaching problems. However, the deployment of other deep learning based systems is still hampered by certain problems:

- **their lack of interpretability.** The problem of the lack of interpretability of these networks is waiting to be addressed, especially in the field of medical diagnostic assistance. One way to make them more interpretable is to generate maps allowing to visualise the degree of involvement of the pixels in the classification.
- **their vulnerability to adversarial attacks.** Indeed, they can be misled during classification by adding modifications to the image that are imperceptible to the naked eye, making these systems vulnerable to attacks.
- **the lack of annotated data, time and computational resources to train them.** Transfer learning is a solution to this problem. It allows to exploit the features learned by a network on another dataset. Another solution to this problem is domain adaptation which allows to train a network with little or no annotated data. This technique only needs the annotated data of another dataset having a similar domain. The lack of annotated data can also be partly filled by annotated data synthesis via GAN. By reducing the data synthesis problem to a fight between a generator and a discriminator, they allow, among other things, to generate much more realistic images than before. Indeed, the use of a discriminator makes it possible to get rid of the problem of certain similarity measures such as the least squares error, which sometimes leads to unrealistic examples.

These practical works allowed us to discover the way the different methods studied work, to think about the different ways of implementing these approaches and to become aware of their limits.

1 Transfer Learning through feature extraction from a CNN

1.1 VGG16 Architecture

1. ★ **Knowing that the fully-connected layers account for the majority of the parameters in a model, give an estimate on the number of parameters of VGG16 (using the sizes given in Figure 1).**
   VGG16 has 3 fully-connected layers.

   The first one is applied on a tensor of size $7 \times 7 \times 512$ and has 4096 neurons. Hence, it has $(7 \times 7 \times 512) \times 4096 + 4096$ bias $= 102\ 764\ 544$ parameters.

   The second one is applied on a tensor of size $1 \times 1 \times 4096$ and has 4096 neurons. Hence, it has $(1 \times 1 \times 4096) \times 4096 + 4096$ bias $= 16\ 781\ 312$ parameters.

The third one is applied on a tensor of size $1 \times 1 \times 4096$ and has 1000 neurons. Hence, it has $(1 \times 1 \times 4096) \times 1000 + 1000$ bias $= 4\ 097\ 000$ parameters.

In total, these three fully-connected layers have 123 642 856 parameters.

2. **★What is the output size of the last layer of VGG16?**
   The output size is equal to the number of classes, *i.e.* 1000.

   What does it correspond to?
   In theory, the output of the last layer is the Softmax function applied on the last fully-connected layer. The Softmax function just normalises the feature vector such that it becomes a probability distribution over the classes. It does not change the size of the vector on which it is applied.

   In practice, the last layer of our pre-trained model is the last fully-connected network.

3. **BONUS : Apply the network on several images of your choice and comment on the results.**
   We first applied the network on an image from the COCO dataset. As we obtained a lot of false positives

Figure (1) An image from the COCO dataset whose images features more complex scenes than the ones of ImageNet.

```
Top 15:
     umbrella : 11.321854
     cash machine, cash dispenser, automated teller machine, automatic teller machine, automated teller, automatic teller, ATM : 11.201432
     limousine, limo : 11.009268
     fur coat : 10.813289
     trench coat : 10.593885
     groom, bridegroom : 10.484614
     suit, suit of clothes : 10.425961
     abaya : 10.124124
     television, television system : 9.912357
     pay-phone, pay-station : 9.819627
     crutch : 9.752721
     ashcan, trash can, garbage can, wastebin, ash bin, ash-bin, ashbin, dustbin, trash barrel, trash bin : 9.668255
     shoe shop, shoe-shop, shoe store : 9.350463
     stage : 9.226672
     jean, blue jean, denim : 8.938109
```

Figure (2) Top-15 predictions of the image coming from the COCO dataset.

(*e.g.* the detected limousine, umbrella, etc...). We also obtained many false negative. For instance, the car and the table are not present in this top-15 predictions whereas these are important parts of the image. These unsatisfactory results were expected because the ImageNet dataset, from which our network was pre-trained, contains images centered around the object whereas our more complex image from the COCO dataset is not like that. Moreover, we had to resize it in order to feed it to the network, which is a naive approach as we saw in the sixth lecture.

We also applied the network on an image that contains a brown bear and the results were very good since the real class was detected as the most probable one and because other classes that were the most predicted are semantically very close to the real one.

Figure (3) An image similar to the ones that ImageNet contains in the sense that this image mainly contains one object which is highly visible in the foreground.

```
Top 15:
     brown bear, bruin, Ursus arctos : 25.984428
     American black bear, black bear, Ursus americanus, Euarctos americanus : 19.949831
     sloth bear, Melursus ursinus, Ursus ursinus : 16.747196
     ice bear, polar bear, Ursus Maritimus, Thalarctos maritimus : 16.668316
     otter : 14.589978
     hyena, hyaena : 14.416616
     weasel : 14.29308
     wild boar, boar, Sus scrofa : 14.236489
     marmot : 14.190776
     Airedale, Airedale terrier : 14.034263
     hog, pig, grunter, squealer, Sus scrofa : 13.807155
     baboon : 13.610524
     mink : 13.268329
     king penguin, Aptenodytes patagonica : 13.245407
     chimpanzee, chimp, Pan troglodytes : 12.981028
```

Figure (4) Top-15 predictions of the image containg the brown bear.

Figure (5) A cropped frame of a cartoon movie that features a duck.

This time the results are not so good because to most probable class is the wrong one. Indeed, the duck was not recognised by the network. However, the latter detected a pelican which is a class that is semantically very close to the real one. These bad results are explained by the fact that ImageNet contains realistic images and not ones taken from cartoon.

```
Top 15:
     vacuum, vacuum cleaner : 12.652191
     swing : 10.946797
     broom : 10.844319
     crutch : 10.776963
     golfcart, golf cart : 10.711409
     swab, swob, mop : 10.512167
     laptop, laptop computer : 10.256154
     lawn mower, mower : 9.111918
     shopping basket : 9.070771
     electric guitar : 8.9348
     paddle, boat paddle : 8.907834
     pelican : 8.709079
     shield, buckler : 8.438528
     shoe shop, shoe-shop, shoe store : 8.3623295
     motor scooter, scooter : 8.320831
```

Figure (6) Top-15 predictions of the image of a duck taken from a cartoon.

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

   In sum, all these features contained by these maps are complementary between each other.

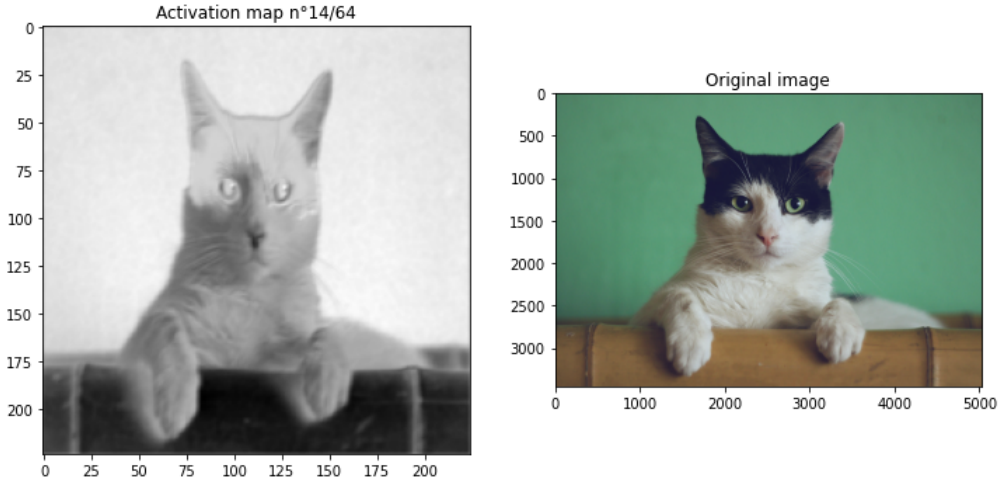


Figure (7) This filter seems to strongly react to green area.

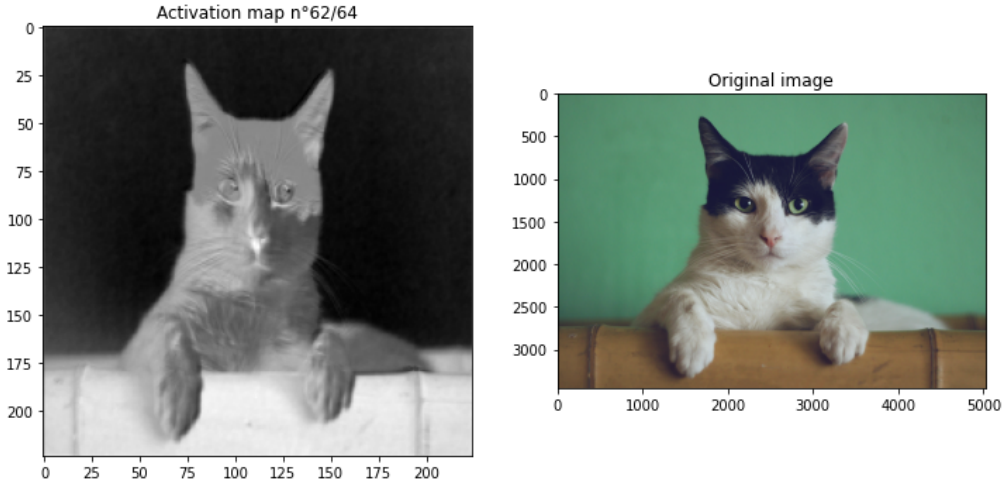


Figure (8) This one seems to strongly react to brown area.

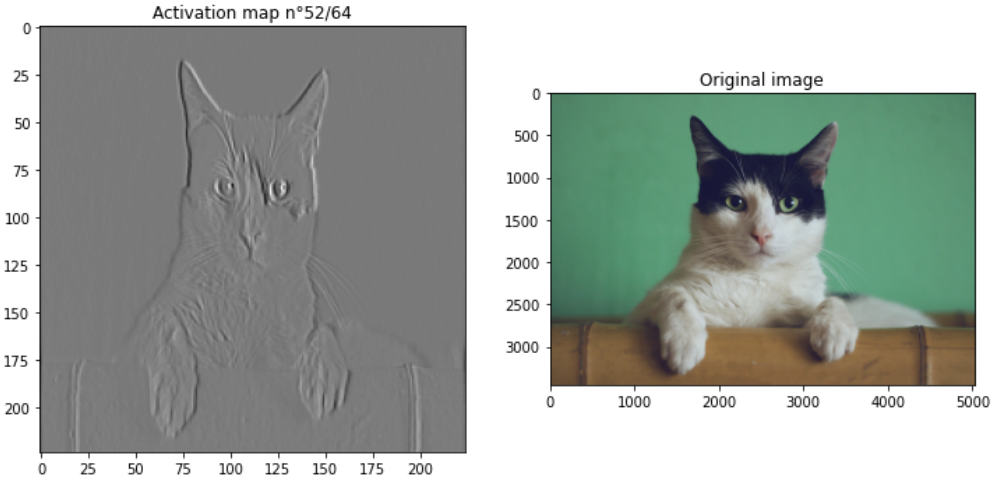


Figure (9) This one seems to act like a high-pass filter because it extracts the contours.

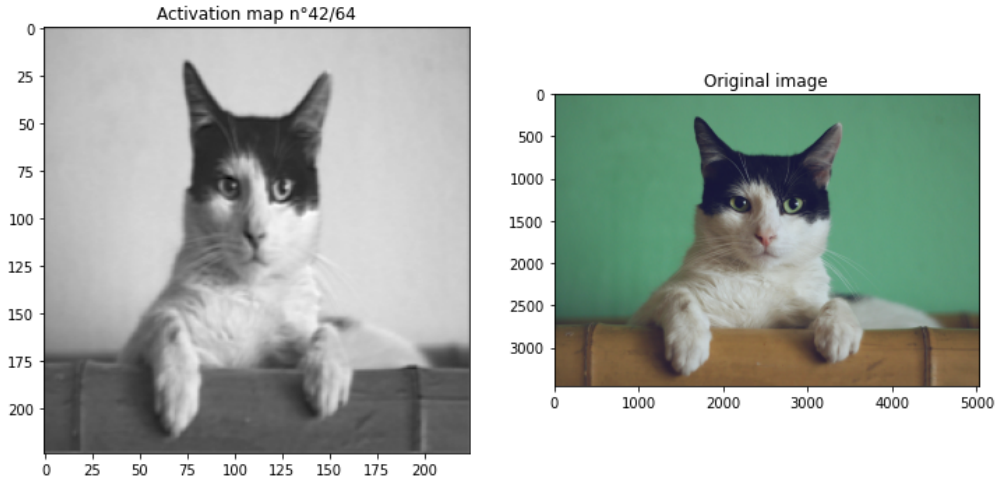


Figure (10) This one seems to act like a low-pass filter because it blurs the image.

1.2 Transfer Learning with VGG16 on 15 Scene

5. ★ **Why not directly train VGG16 on 15 Scene?**
   There are two reasons for that.

   Firstly, the number of neurons of the last layer of VGG16 is not equal to the number of classes of the 15 Scene dataset.

   Secondly, if we directly train VGG16 on 15 Scene there is a risk of overfiting. Indeed, VGG16 has many parameters because its architecture was designed so that it can be train on the huge ImageNet dataset containing more than 14 million images. Although they are organised into 21,841 subcategories that have 500 images on average, these subcategories can be considered as sub-trees of 27 high-level categories. Hence, we can roughly say that in ImageNet each class has much more images than the 15 Scene dataset that contains 200-400 images per category.

6. ★ **How can pre-training on ImageNet help classification for 15 Scene?**
   ImageNet is a very diversified dataset. Therefore, the first layers of a convolutional network trained on this dataset can be seen as an extractor of high-level features that allows to make the classes more linearly separable. In our case, the fact that the pre-trained model is frozen during the training on the 15 Scene dataset reduces the risk of overfitting.

7. **What limits can you see with feature extraction?**
   If the classification task of the new problem differs too much from the one that the pre-trained network was

trained to solve then the extracted features will not be the most relevant ones, especially after the last layers of the network.

Additionally, the domain of the data from which we want to extract features needs to be similar to the one of the data from which the pre-trained model was trained.

8. **What is the impact of the layer at which the features are extracted?**
   The deeper this layer is the more abstract are the extracted features. It may seem counter-intuitive but ablation studies (*cf.* course 4) show that the deeper the layer from which the features are extracted is, the better the classification results we obtain.

9. **The images from 15 Scene are black and white, but VGG16 requires RGB images. How can we get around this problem?**
   For each image, we just have to stack two copies on top of the original image in order to obtain images that have three identical channels.

10. **Rather than training an independent classifier, is it possible to just use the neural network? Explain.**
    Yes it is possible but we need to change the number of neurons of the third fully-connected layer in order to make it equal to the number of classes (15) of the dataset we use.

11. **For every improvement that you test, explain your reasoning and comment on the obtained results.**
    First, we trained the multi-class SVM classifier on the training set with $C = 1$ using the *deep features* previously extracted and we obtained an accuracy of 0.887772 on the test set. Then, we changed the layer at which the features are extracted while fine-tuning the C parameter. Here are the results we obtained:

	Representation size	**Best accuracy**	**Best C value**
1st conv. layer	3211264	unknown	unknown
2nd conv. layer	1605632	unknown	unknown
3rd conv. layer	802816	unknown	unknown
4th conv. layer	401408	unknown	unknown
5th conv. layer	100352	0.898	2.100
6th conv. layer	25088	0.900	1.350
1st FC layer	4096	0.889	1.450
2nd FC layer	4096	0.889	1.450
3rd FC layer	1000	0.874	2.450

The bigger the representation size, the more the computational cost (time and memory) of the SVM's training is increased. It may seem counter-intuitive but, as we saw in class, the deeper the layer at which is extracted the features is the better will be the results obtained with the SVM. However, in practice this is not always verified. Here, the fact that the best accuracy is not obtained at the last layer may be explained by the high dissimilarity between the Scene-15 and the ImageNet datasets. The resource freely allocated by Google Colab are not sufficient to train the SVM on such big embeddings.

We also tried to use Resnet instead of VGG16 because it is known to be more efficient by itself. However, when we cut the architecture at the last convolution layer, just before the fully connected one, and applied a SVM on the features obtained, we find that the best score obtained is not better than VGG16. This can be explained by the fact that resnet has a very deep architecture and the features extracted by the model are thus very abstract.

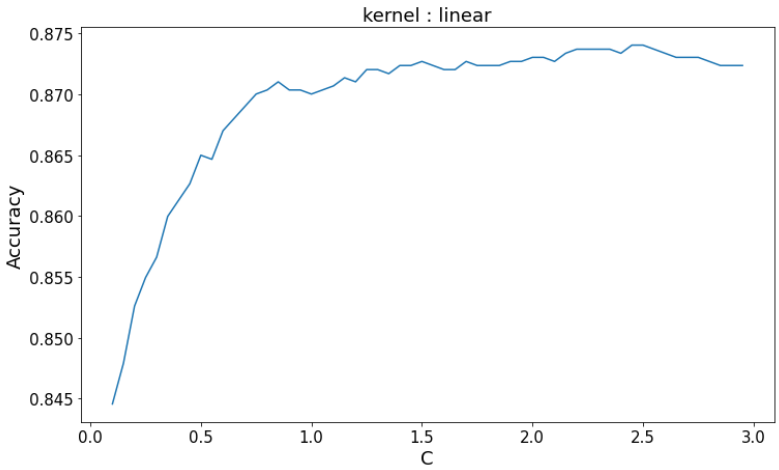


Figure (11) Evolution of the precision depending of C when transferring the features extracted from the last convolution layer of a ResNet50.

2 Visualizing Neural Networks

2.1 Saliency Map

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

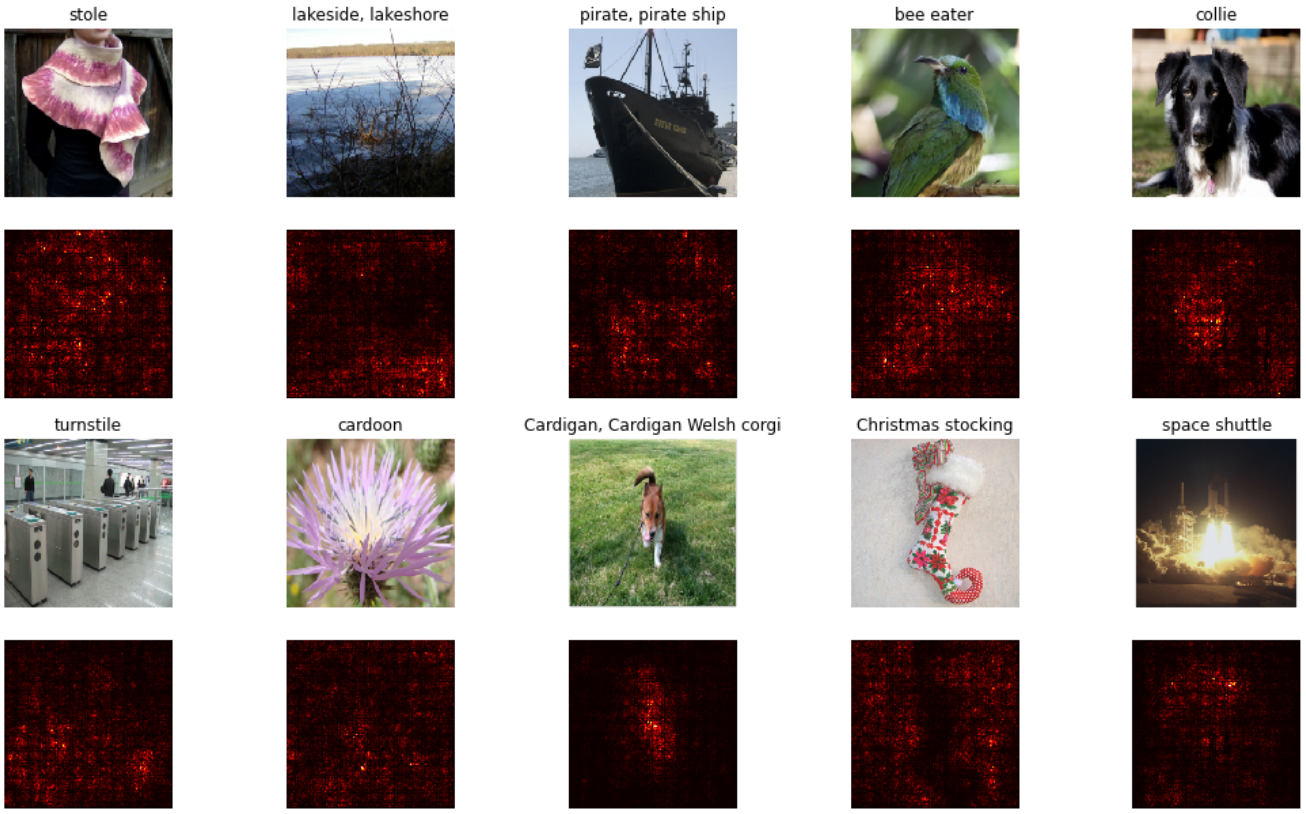


Figure (12) SqueezeNet saliency maps of various images from ImageNet.

We obtain different kinds of results. Some of the saliency maps are exactly the way we expect them to be. For example, the most impactful pixels for the Cardigan are all stacked in the center of the image which is where the dog is on the image. Also for the Collie which takes almost all of the image, we see that the important pixels are mainly where the face of the dog is on the image. Since the face is where most of the features specific to this species are, we expected this saliency map.

Some other saliency maps are a bit more surprising, such as the turnstile's. We see that the important pixels are more spread on the map. This still makes sense since there are more than one turnstile on the image and they take almost all of it. We can still wonder about the patch on the bottom right which seems to be on the floor. This could be due to a phenomenon that is very obvious in the Christmas stocking's saliency map. We see that almost none of the actual stocking's pixels are important to recognise it. The pixels that are important are the background's, to the point where the complementary shape of the stocking appears. This is a problem in the dataset that is probably linked to the fact that Christmas stockings are often portrayed against a plain wall. Therefore, during training, the model has associated Christmas stocking with the plain background instead of the stocking itself. This was probably done because it is easier for the network to learn to recognise this simple background than the appearance of a stoking. To solve this issue, we would have to add images with plain background in the other classes and add diversity to the Christmas stocking class.

2. **Discuss the limits of this technique of visualizing the impact of different pixels.**
These activation maps probably also differ from the real activation maps of the network because they are a bit different from the ones we can naturally expect. This is mainly explained by the limiting assumption that is implicitly done by the approximation of the linear function: the independence between pixels. Another limit is that only the localisation of these pixels is taken into account while their intensity is completely ignored.

3. **Can this technique be used for a different purpose than interpreting the network?**
Yes, they can be useful for detection and segmentation tasks because the features that allow the model to recognize are generally on the object. The activated pixels allow to localise the object. More precisely, this could also be used to localise certain parts of the objects. If we notice that the network uses certain features of the object, we can use these maps to localise them on other images.

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

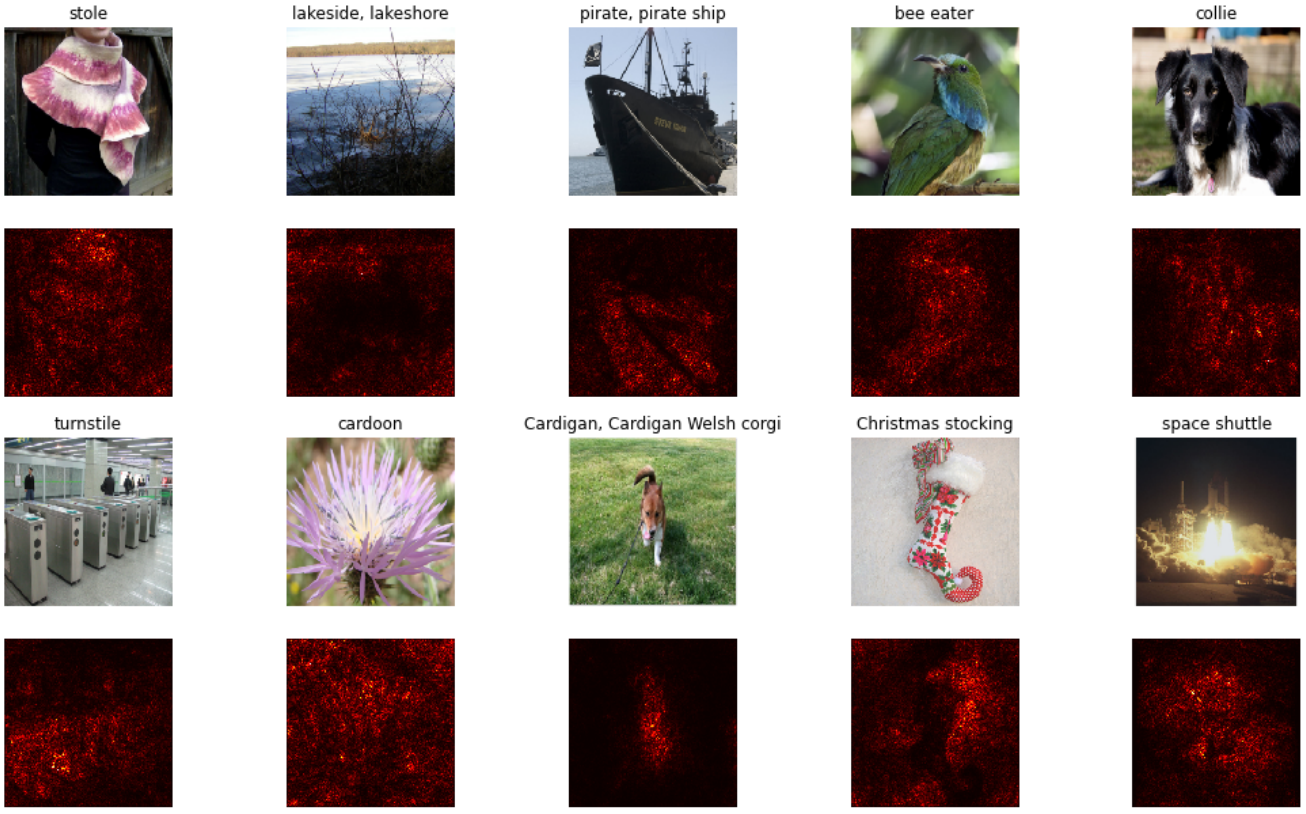


Figure (13) VGG16 saliency maps of various images from ImageNet.

We can see that VGG16 is a lot more precise than SqueezeNet. This was expected because it has more and bigger layers. The saliency maps that were already precise with SqueezeNet are even better here. For example, we can even recognise the Cardigan's tail in the map. The maps where the main pixels were spread still are but with less outliers. For example, the turnstiles' map makes more sense as there are more important pixels

on the actual objects and less stacks on the ground.
However, we still find the same phenomenon with the Christmas stocking, which is not surprising since the problem probably comes from the dataset, as explained in question 1.

2.2 Adversarial Examples

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

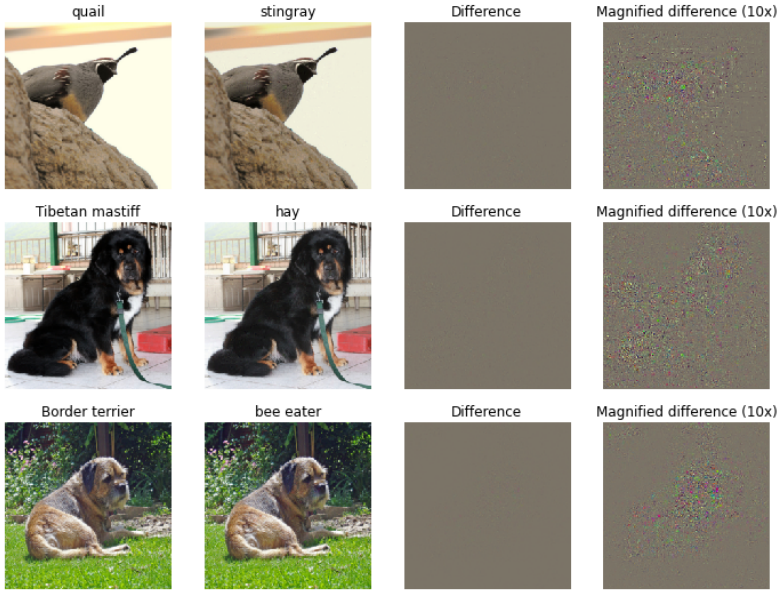


Figure (14) Original images and modified images where the model is fooled.

At first glance, it looks like the images have not been modified and the model is still fooled. When taking a closer look we can see small changes, in intensity for example. Even the difference image has to be magnified in order to make the modifications visible.
Even though the modifications are optimised to fool the model as fast as possible, it is interesting to note that it happens in only a few iterations.

6. **In practice, what consequences can this method have when using convolutional neural networks?**
This is an hindrance to the deployment of convolutional neural networks in real life. Indeed, since such networks can be adversarially attacked, we cannot guarantee that in real life these networks will have performance similar to the ones they had in test. Among the applications that are affected by this problem we can mention autonomous driving since people can easily change the input of the network whereas we cannot afford wrong predictions.

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).**
Here we perform a backpropagation so we assume that we have access to the trained model, which is not always the case. Another issue with this approach is the number of model queries required to compute the gradient of a solution within the high-dimensional image space.

   Among the existing methods, we can cite the one that [1] Phoenix Williams and Ke Li proposed in 2022. It is a novel attack that uses gradient-free optimisation by evolving a series of overlapping transparent shapes (triangles, rectangles, etc) using an evolutionary algorithm. Their method only requires the query feedback returned by the network and has shown an ability to outperform the other *black-box* attacks method that rely on the estimation of the gradient.

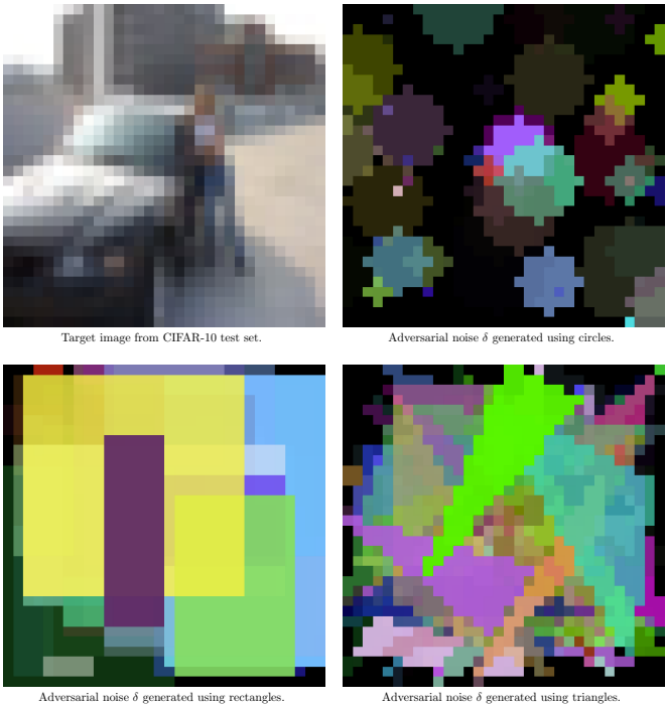


Figure (15) Example of magnified adversarial noise δ generated using 50 shapes for the top left target image [1].

The same year, Dongbin Na *et. al* proposed another simple but powerful *black-box* attack method [2]. Their methods takes as input two images: a source image x_{src} and a target image x_{adv} that are semantically different. What we want is to classify the source image in y_{trg}, the class of the target image. To do so, two steps are required. First, we use an auto-encoder (made of an encoder and a decoder G) in order to respectively extract from the source image and the target image two latent vectors w_{src} and w_{trg}. Second, we modify w_{trg} such that the distance between w_{src} and w_{trg} is minimised while the class $F\left(G\left(w_{trg}\right)\right)$ predicted by the network F is unchanged. Once this objective is reached, the modified vector w_{trg} is named w_{adv}. Formally, the objective is:

$$\underset{w_{adv}}{\text{minimize}} \quad D\left(w_{src}, w_{adv}\right) \quad \text{s. t.} \quad F\left(G\left(w_{adv}\right)\right) = y_{trg}.$$

Finally, the adversarial example is the generated image $G(w_{trg})$.

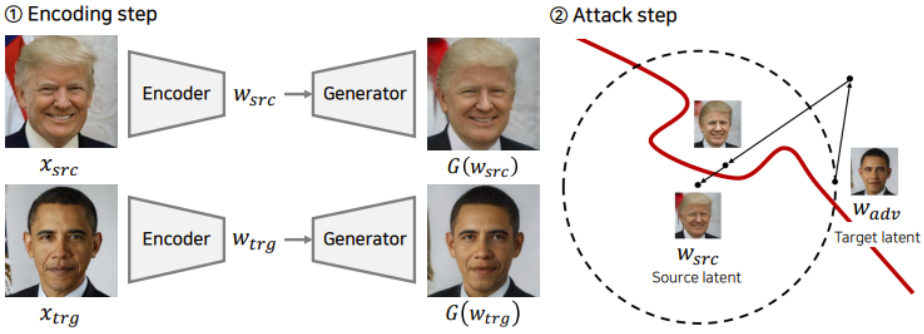
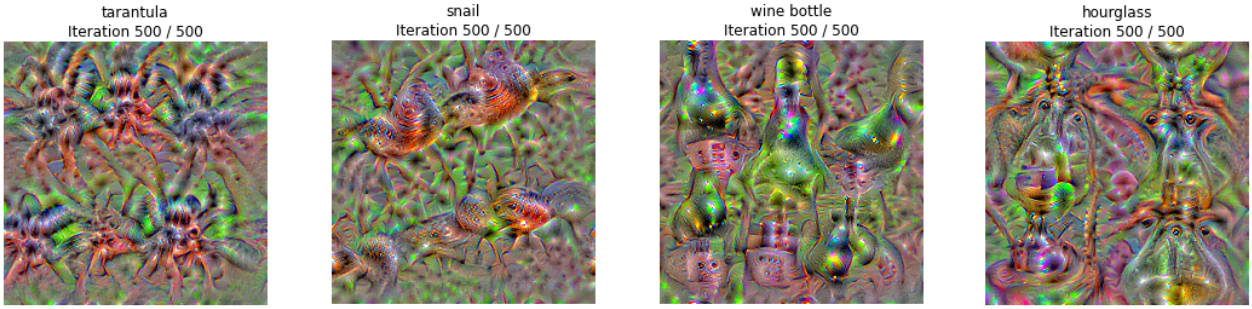


Figure (16) An illustration of the method proposed in [2].

However, the authors noticed that sometimes the adversarial examples that the methods creates look too different from the original images.

2.3 Class Visualization

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



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

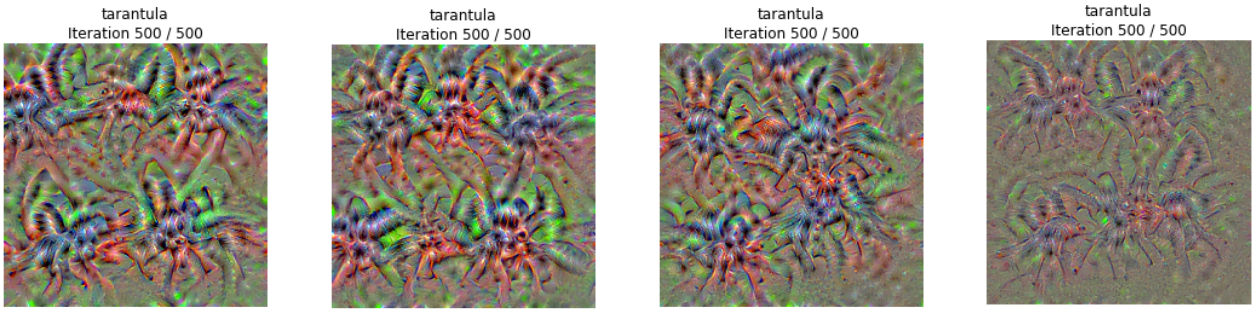
Figure (17) Different examples of visual patterns producing class predictions.

It is interesting to note that it is hard for the human eye to recognise the class that was reproduced. The images are not realistic but we can still see some of the features. For example, if we look closely at the tarantula image, we can see the 8 characteristic legs as well as kind of bodies.

Depending on the class, we see that the colors make more or less sense. For example, the hourglass is represented with many colors mixed. This is because this object does not really have a characteristic color, whereas the wine bottle is mainly represented in green which is its basic color. We can also note the touches of green around the tarantulas and snails that are probably caused by the fact that theses animals are often portrayed on grass or leaves.

We also see that on all of the above examples, the object is represented many times on the image. This is because the model has been trained on many different images for each class and the position of the object on these varies. The model is thus trained to find the object anywhere on the image and in different positions and sizes.

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

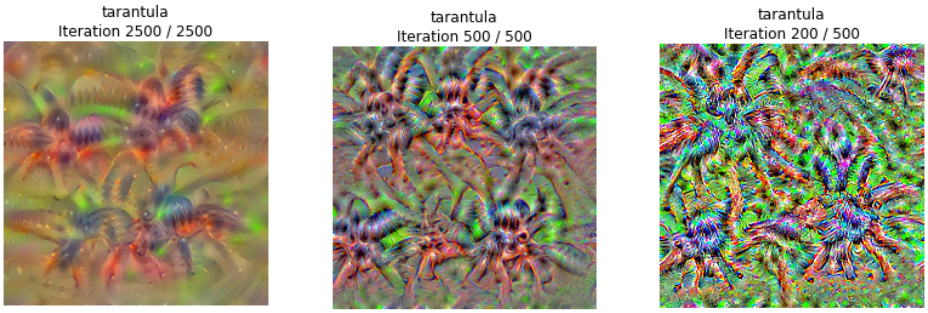


(a) (b) (c) (d)

Figure (18) Tarantula class generated from random noise with the regularisation weight set to (a) 10^{-7}, (b) 10^{-3}, (c) 10^{-1} and (d) 1.

By letting the regularisation weight vary we see in Figure 18 that having a big weight really limits the apparition of new details. In fact, as the regularisation penalises the image's norm, if it is high we expect it to limit the intensities. Here, one is too big of a weight since it is hard to recognise the tarantula shapes. However, setting it to 0.01 make the appearing shapes a bit more intense and we start to recognise them. If we want to really recognise the class's features, we can use a regularisation weight of 10^{-3}. We note the reducing even more the weight does not really change the appearance of the generated image. However, we can expect the difference to reside in the fact that we are more likely to use this method on real images instead

of random noise. Then the more we will reduce this weight, the less the original image will be altered. If we want to visually modify the class of the image, it is better to keep the weight big enough.



(a) (b) (c)

Figure (19) Tarantula class generated from random noise with the learning rate set to (a) 1, (b) 5 and (c) 10.

Obviously, the first impact we note is that the more we reduce the learning rate, the more iterations it takes to obtain a meaningful result.
Also, by comparing the three images we see that the images with a bigger learning rate have much more contrast between the colors. This is because the features are generated much more strongly at each step. A small learning rate induces that the image is not modified by a lot so only small changes in intensities are applied. As a result, the different colors are mixing more than if the intensities changed by a lot every step.
Another effect is that bigger learning rate seem to generate more features. We can see in figure 18.c that there are shapes everywhere in the image, while in figure 18.a there are zones with no shapes.

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.**

    As we can see in figure 19, the visual interest of generating real class features on an image is limited. The image is altered while the objects of interest are modified. For a human, it only makes it harder to recognise and understand the scene.
    However, it is interesting to do this when the idea is to show the modified image to a model. In fact, features making it classified in the hay class are amplified and it improves the chances for the model to get the guess the right class.

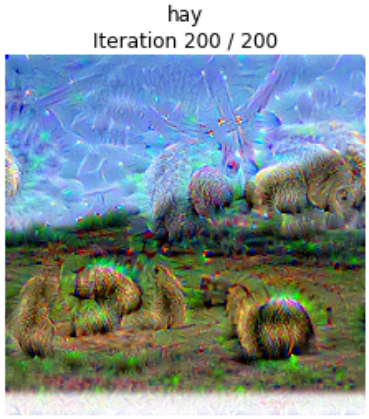
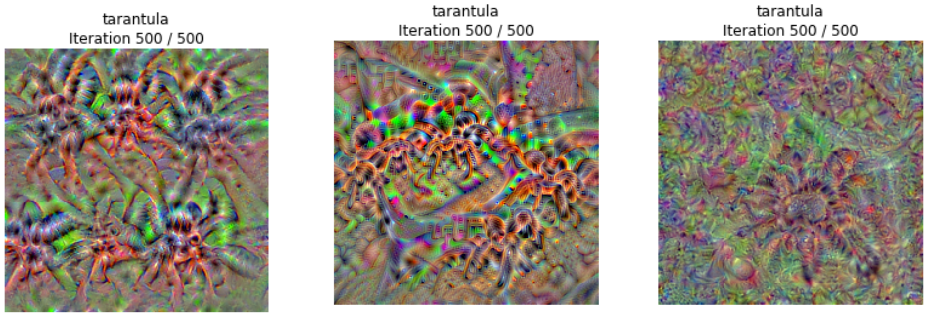


Figure (20) Image from the hay class with hay features generated.

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

    As we expected, results are very different from one model to the other. For example, VGG16 seems to have better taken into account the backgrounds of the tarantula images as we can see the to of the image showing



(a) (b) (c)

Figure (21) Tarantula class generated from random noise with (a) SqueezeNet, (b) VGG16, (c) ResNet.

different features than the rest.
On all three images, we can recognise tarantula shapes. However their sizes, number and position are different. ResNet's tarantula is one big animal with a very faithful shape. VGG16's are more than one and much smaller, they also are much harder to recognise as we can mainly see the features of the legs. SqueezeNet's are also more than one but this time pretty big. The main diference with the other two is that the spiders take all of the image.
Another important difference is that it took much longer to generate the images with VGG16 and ResNet, which is normal since SqueezeNet is made to be compact.

3 Domain Adaptation

3.1 Practice

1. **If you keep the network with the three parts (green, blue, pink) but didn't use the GRL, what would happen ?**
   If we remove the GRL layer, the features will be domain dependant so the label predictor will be less successful on the source domain.
   Consequently, the learnt features will be more specific to the target domain and then not optimised for the label predictor.

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

   The performance will be degraded because the network will learn to generalize more. The features it recognises will be less precise on the source domain in order to envelop the target domain.

3. **Discuss the influence of the value of the negative number used to reverse the gradient in the GRL.**
   The gradient needs to be reversed just before that because we want the feature extractor to produce features such that the domain classifier cannot discriminate the domain. The bigger λ is, the more the weights of the feature extractor will be updated such that it similarly encodes the features of the two domains. However, if λ is too high there will be a risk of divergence.

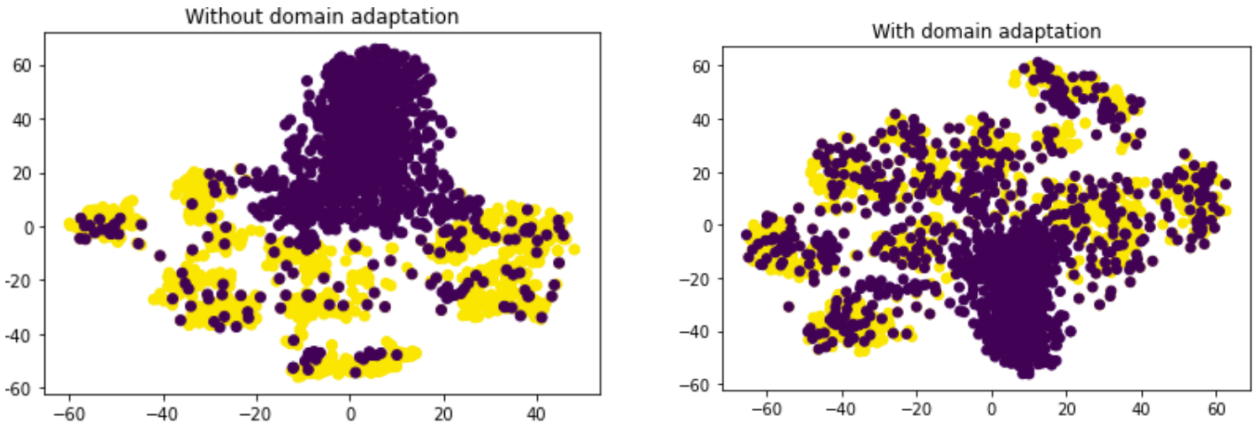
4. **Another common method in domain adaptation is pseudo-labeling. Investigate what it is and describe it in your own words.**
   When used in domain adaptation, data labelling consists of training a network in a supervised fashion with labeled and unlabeled data simultaneously. For a given unlabeled data, its associated groundtruth or *pseudo-label* is the class which has maximum predicted probability by the network. The overall loss is given by:

$$L = \frac{1}{n}\sum_{m=1}^{n}\sum_{i=1}^{C} L\left(y_i^m, f_i^m\right) + \alpha(t)\frac{1}{n'}\sum_{m=1}^{n'}\sum_{i=1}^{C} L\left(y_i'^m, f_i'^m\right),$$

   where:

- n and n' are respectively the number of mini-batch in labeled data and unlabeled data;
- f_i^m and y_i^m are respectively the prediction and the label associated to the $m's$ sample;
- $f_i'^m$ and $y_i'^m$ are respectively the prediction and the pseudo-label associated to the $m's$ sample;
- $\alpha(t)$ is a weighting term which increases during the optimisation process to take into account the fact that the pseudo-labels become more and more reliable during the iterations. Its increasingness is progressive in order to avoid poor local minima.

3.2 Coding results



(a) (b)

Figure (22) Visualisations, with t-SNE, of the latent spaces respectively obtained (a) without and (b) with domain adaptation.

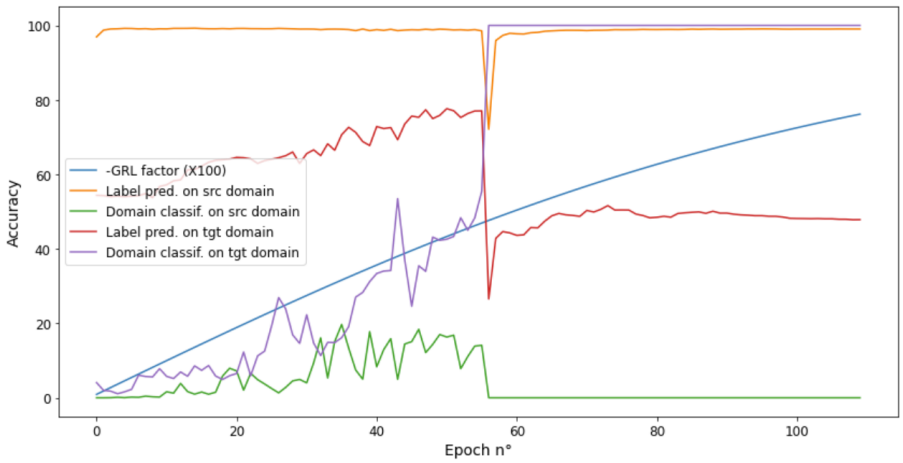


Figure (23) Evolution of the accuracies obtained during the training as well as that of the GRL factor. We expected the accuracy of the domain classifier to be high in the begining and around 50% at the end of the training but it is apparently not the case despite the good accuracy (0.78) that we obtained.

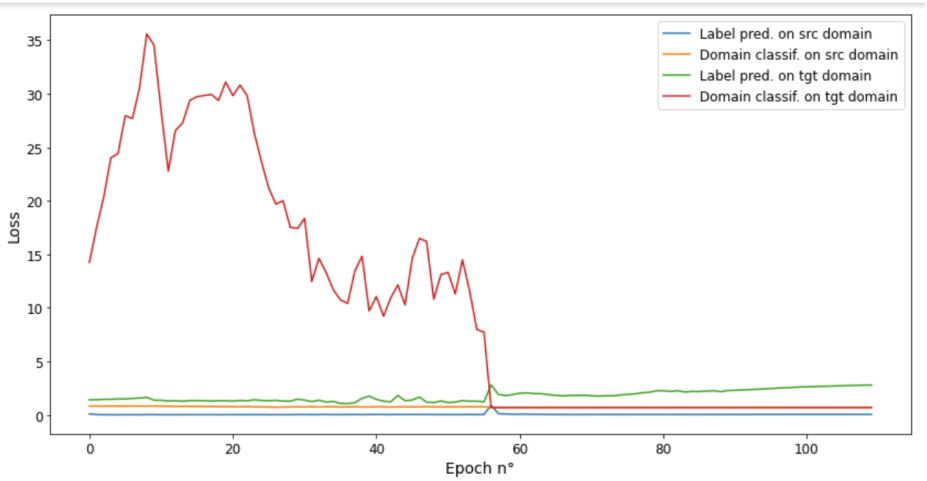


Figure (24) Evolution of the different losses obtained during the training.

4 Generative Adversarial Networks

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

$$\max_G \mathbb{E}_{\mathbf{z}\sim P(\mathbf{z})} [\log D(G(\mathbf{z}))] \tag{6}$$

$$\max_D \mathbb{E}_{\mathbf{x}^*\in \mathcal{D}\text{ata}} [\log D(\mathbf{x}^*)] + \mathbb{E}_{\mathbf{z}\sim P(\mathbf{z})} [\log (1 - D(G(\mathbf{z})))] \tag{7}$$

We know that $D(x)$ represents the probability that the image x is real according to the discriminator D.

Consequently, Equation (6) represents what the generator G seeks to obtain: weights optimised such that it synthesise only images that the discriminator will think are real.

Equation (7) represents what the discriminator D seeks to obtain: weights optimised such that for every image taken as input, it can determine if this image is real or not, without ever being wrong. The first term tests if it recognises real images and the second tests if it can recognise fake ones.

(b) **Ideally, what should the generator G transform the distribution $P(\mathbf{z})$ to?**
Ideally, the generator G should transform the distribution $P(\mathbf{z})$ to the distribution of real data $P(X)$.

(c) **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 ?**
The true equation should be:

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

The problem with this equation is that it causes the gradient to vanish. In fact, the last layer of the discriminator is a Sigmoid and when we compute the gradient of this loss, we obtain a result proportional to the Sigmoid. This is a problem because, at the start of the training, the generated images are not realistic and the discriminator easily determines that the image is not real so $D(G(x))$ is close to one and therefore the gradient is close to 0.

(d) **Comment on the training of of the GAN with the default settings (progress of the generations, the loss, stability, image diversity, etc.)**
The training worked pretty well. We can see some images that are faithful to the real numbers. As it is often the case with GANs, some of the images don't resemble any number. But this is necessary if we want diversity in the images. In fact, the images generated are varied thanks to the initialisation that is generally random, and then transformed into a realistic image by the generator which has its limits.

Therefore, to have diversity, we need to have bad images too.

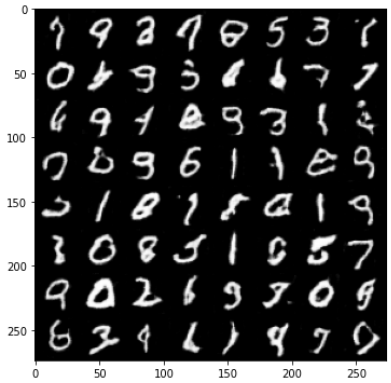


Figure (25) **Numbers generated by the GAN**.

The evolution of the losses is pretty interesting. The loss of the generator decreased at the beginning but both losses evolution can be described as chaotic. This is an effect of the adversarial aspect of the method. Both models are optimised to try to be better than the other which makes the losses fluctuate. An important aspect of the losses evolution is the peaks we regularly encounter over the iterations. This can be explained by the fact that the generator starts to produce always the same type of image. Once the discriminator recognizes it, the generator's loss peaks. On the next iteration, it will have been optimized to produce different kinds of images but the discriminator will only look for the previous type. In turn, the discriminator's loss will peak. We often encounter this phenomenon in the following experiences.

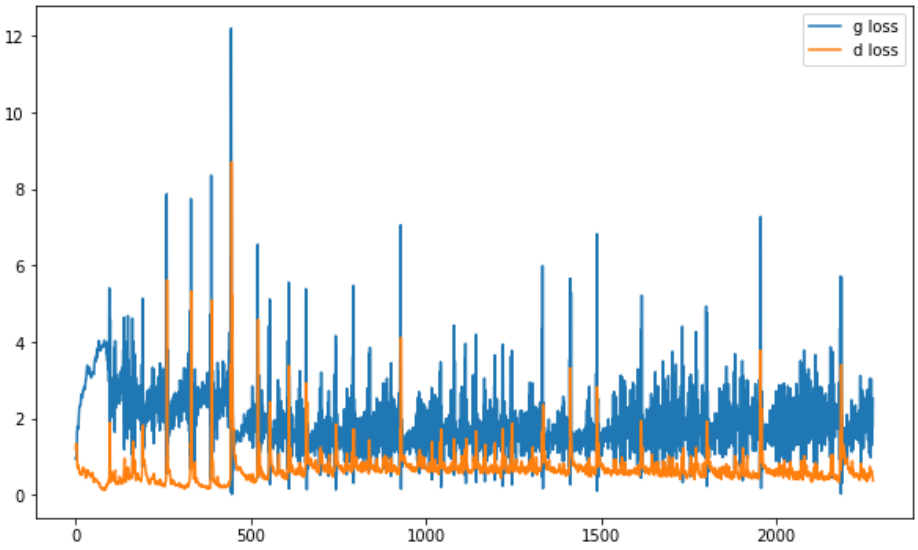
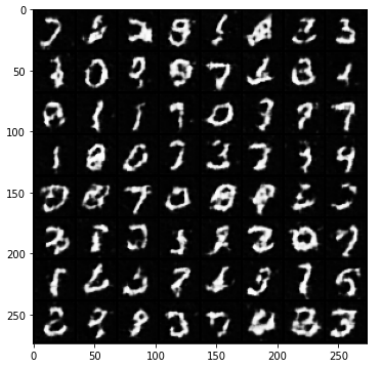


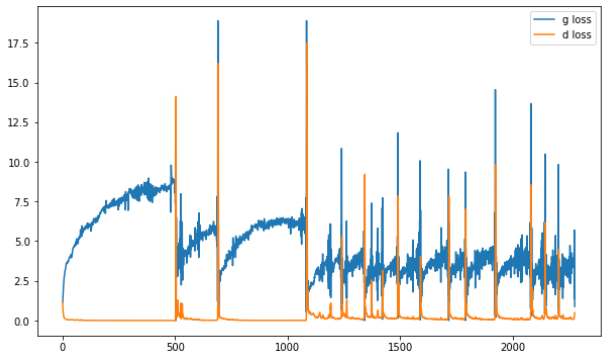
Figure (26) Losses (depending on the epoch n°) obtained during the GAN training.

(e) **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.**

- When we significantly decrease ngf, we see that the generator's loss struggles to stabilize and decrease. Periodically, the loss decreases by a big step and then starts to increase again.



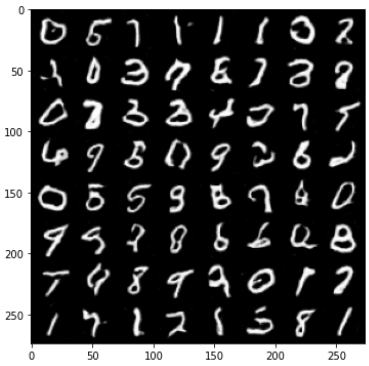
(a) Numbers generated by the GAN.



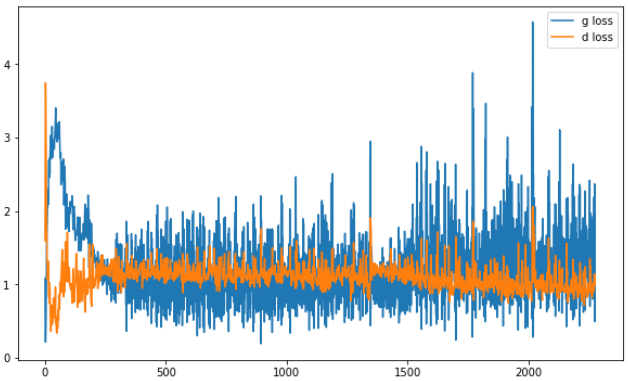
(b) Losses evolution during training.

Figure (27) GAN training with the parameter ngf (number of channels before the generator's last layer) set to 4.

- However, if we increase ngf, we see that the losses reach values we expected them to. We note that they are much less stable than with the default settings.
  Visually, the numbers look realistic and we could even argue that they are better than with the default settings.



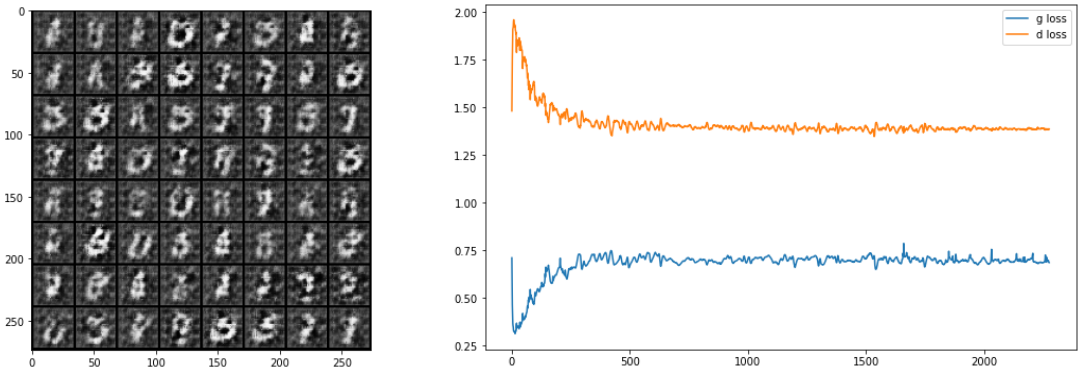
(a) Numbers generated by the GAN.



(b) Losses evolution during training.

Figure (28) GAN training with the parameter ngf (number of channels before the generator's last layer) set to 256.

- When we significantly decrease ndf, we see the training works very poorly. The discriminator's loss does not manage to decrease enough and it means that it will not recognize images efficiently. This thus means that the generator will not be trained properly and we obtain these bad generated numbers.

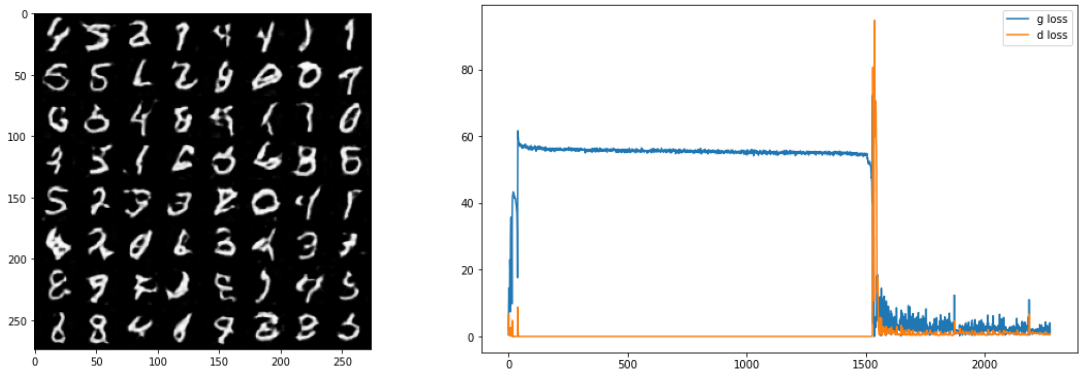


(a) Numbers generated by the GAN.

(b) Losses evolution during training.

Figure (29) GAN training with the parameter ndf (number of channels before the discriminator's last layer) set to 4.

- However when we set ndf to a big number, we see that the discriminator is so efficiently trained that it takes 1500 iterations to make a mistake. This slows the training down but the generator ends up by catching up and the training ends normally.

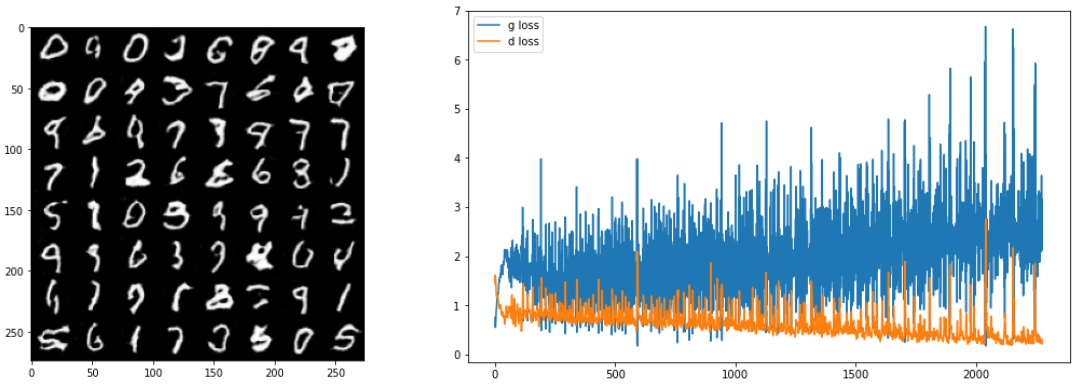


(a) Numbers generated by the GAN.

(b) Losses evolution during training.

Figure (30) GAN training with the parameter ndf (number of channels before the discriminator's last layer) set to 256.

- By initialising the models' weights to their default setting, we see that the training is much less stable. The result does not look much worse but we note that the generator's loss tends to increase over the iterations.

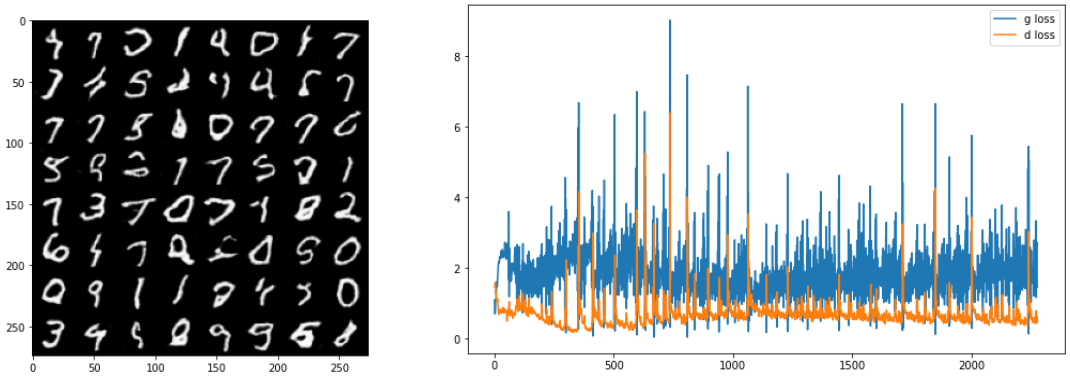


(a) Numbers generated by the GAN.

(b) Losses evolution during training.

Figure (31) GAN training with the models' weights initialised with Pytorch default parameters.

- It seems that the gradient did not vanish at the beginning of the training. We see that the results are similar which was expected since the true loss and the adapted loss are equivalent.

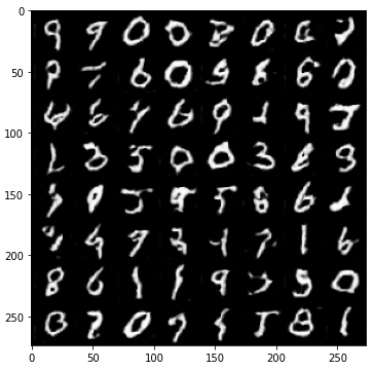


(a) Numbers generated by the GAN.

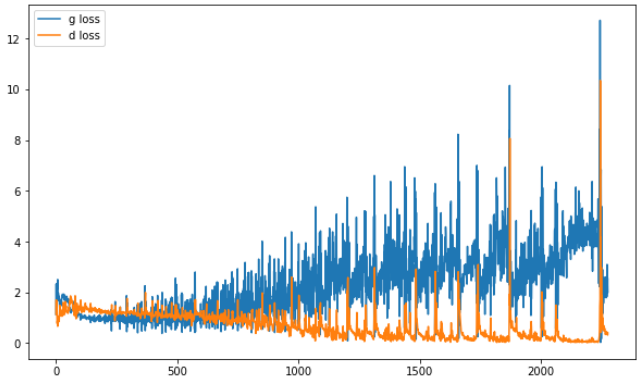
(b) Losses evolution during training.

Figure (32) GAN training with the true theoretical loss for the generator's optimisation.

- Decreasing the generator's learning rate, we note that its loss starts to increase over the iterations. This is because the discriminator learns faster than the generator which means it gets better more efficient over the iterations.



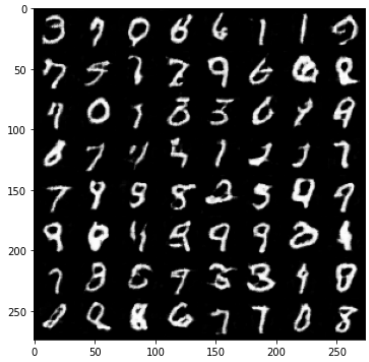
(a) Numbers generated by the GAN.



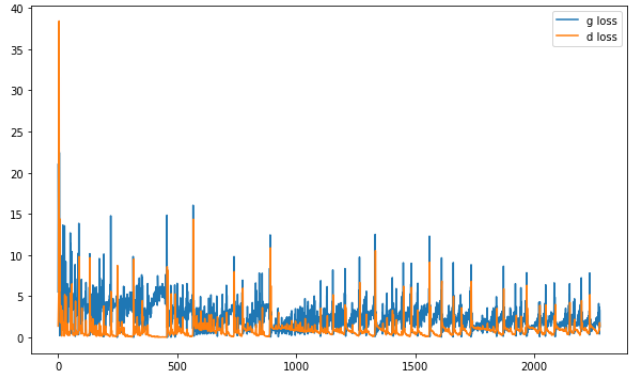
(b) Losses evolution during training.

Figure (33) GAN training with the learning rate for the generator set to 0.05.

- However, when we decrease the discriminator's learning rate, we don't see this same phenomenon. The losses simply seem to be much less stable.



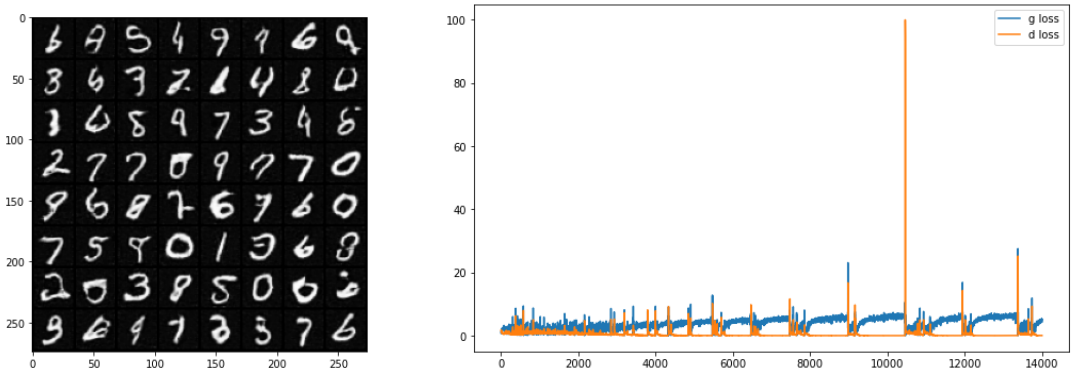
(a) Numbers generated by the GAN.



(b) Losses evolution during training

Figure (34) GAN training with the learning rate for the discriminator set to 0.02.

- When we increase the number of epochs, the result don't seem to improve. In fact, we see through the losses that the training enters a cycle where the generator's loss increases more and more every cycle.

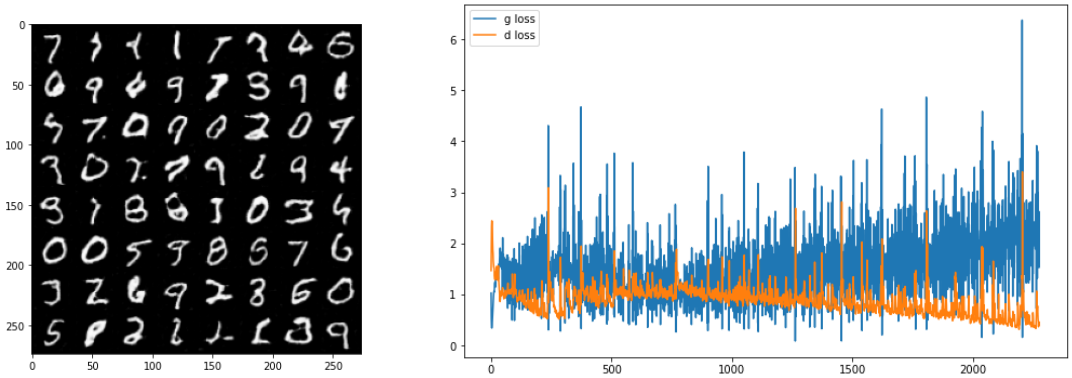


(a) Numbers generated by the GAN.

(b) Losses evolution during training.

Figure (35) GAN training after 30 epochs.

- Using a very small initialisation of size 10 produces pretty good results. Peaks even seem to be smaller that with a size of 100 showing better stability. However we see that the discriminator's loss decreases over the iterations compared to the generator. This might be because it is entering 'Collapse mode'. It means that the generator produces a limited variety of images that are recognized better and better.

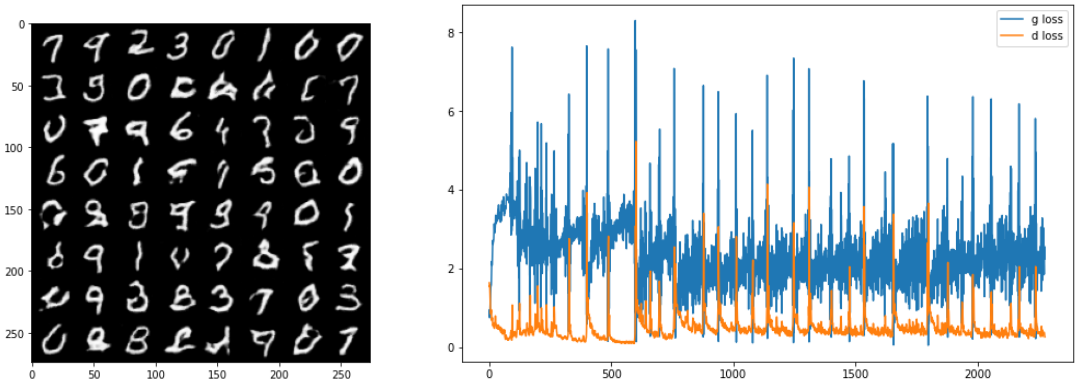


(a) Numbers generated by the GAN.

(b) Losses evolution during training.

Figure (36) GAN training with the parameter n_z (the latent size) set to 10.

- When setting n_z very high, we note that the losses are very unstable. There are more and bigger peaks. However, we don't encounter the collapsing, just like with $n_z = 100$ because we have a bigger variety of initialisation.

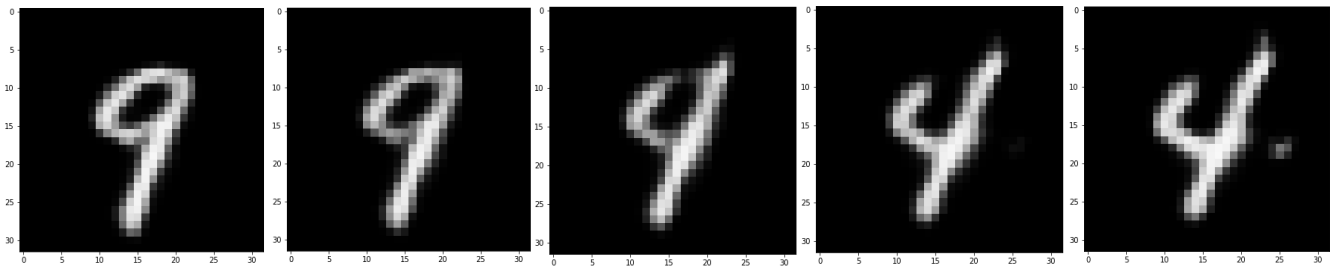


(a) Numbers generated by the GAN.

(b) Losses evolution during training

Figure (37) GAN training with the parameter n_z (the latent size) set to 1000.

- As we can see in Figure 38, the arithmetic operations that we can perform in the latent space allow a control of the generated image. We can make numbers voluntarily ambiguous or control a little bit the shape by making it tend toward another number.

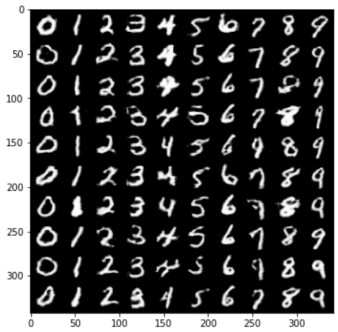


$\alpha = 0.00$ $\alpha = 0.25$ $\alpha = 0.50$ $\alpha = 0.75$ $\alpha = 1.00$

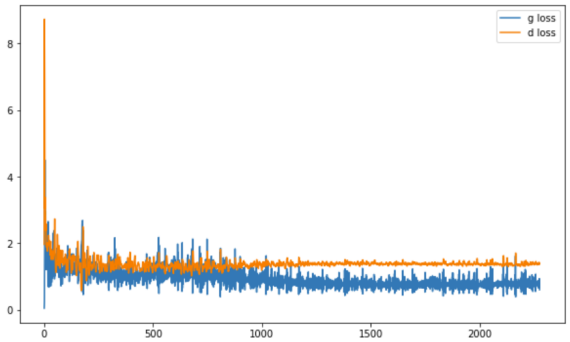
Figure (38) Images generated from the latent vector $\alpha \cdot z_1 + (1 - \alpha) \cdot z_2$ where z_1 is the latent vector of a 9 while z_2 is that of a 4.

4.1 BONUS : Conditional Generative Adversarial Networks

(f) **Comment on your experiences with the conditional DCGAN.**
For all of these experiments we kept $ngf = 64$.

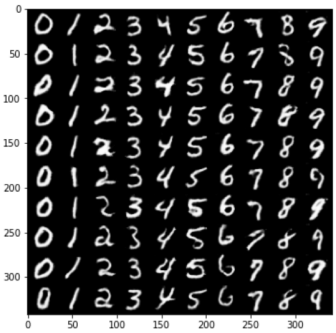


(a) Numbers generated by the GAN.

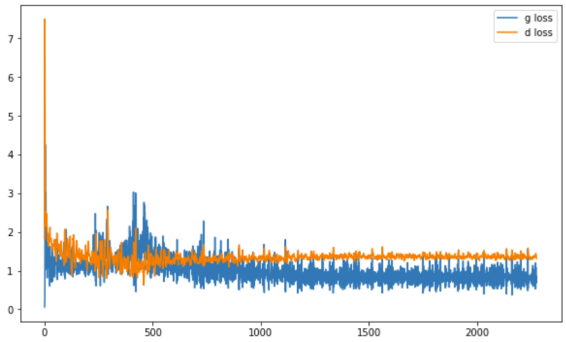


(b) Losses evolution during training

Figure (39) **GAN training with the parameter n_z (the latent size) set to 10.** In (a) each column corresponds to a class while each row corresponds to a particular style. We are able to identify the digits but the intra-class variability of the style is too high.

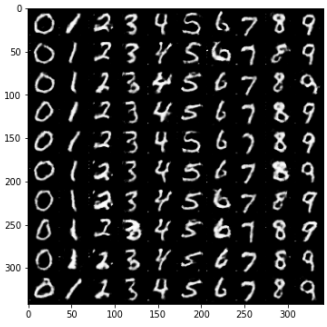


(a) Numbers generated by the GAN.

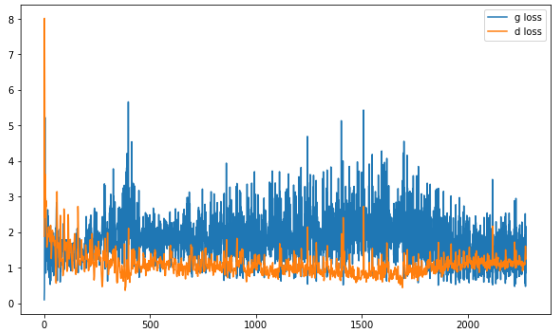


(b) Losses evolution during training

Figure (40) **GAN training with the parameters n_z (the latent size) and ngf respectively set to 100 and 64.** The quality of the images are better than with $n_z = 10$ or $n_z = 1000$.



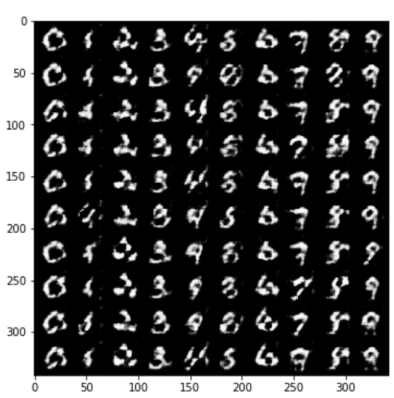
(a) Numbers generated by the GAN.



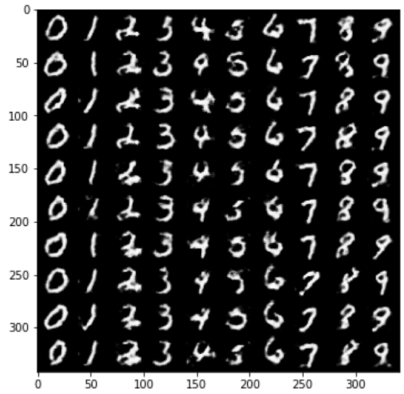
(b) Losses evolution during training

Figure (41) **GAN training with the parameter n_z (the latent size) set to 1000.** We can notice that the quality of the generations is bad and that the loss of the generator is higher than before. Here, the intra-variability of style is also too high.

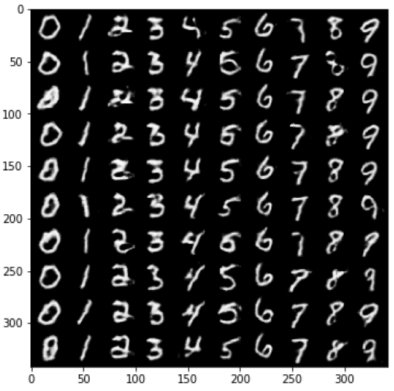
Just like in the unconditional case, we see that using a big n_z makes the losses much less stable. This is because the input can vary too much and the models struggle to adapt. However we notice a slight tendency of the numbers to be more varied which we expected. On the other hand, using a small n_z, while more stable, gives poor results even though we don't have mode collapse this time. With too few possibilities, inputs from one number to the other might be too close and they can thus look like mixes of different numbers.



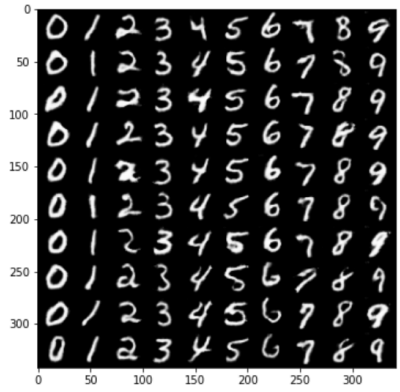
(a) Numbers generated at iteration n°669.



(b) Numbers generated at iteration 1138.



(c) Numbers generated at iteration 1607.



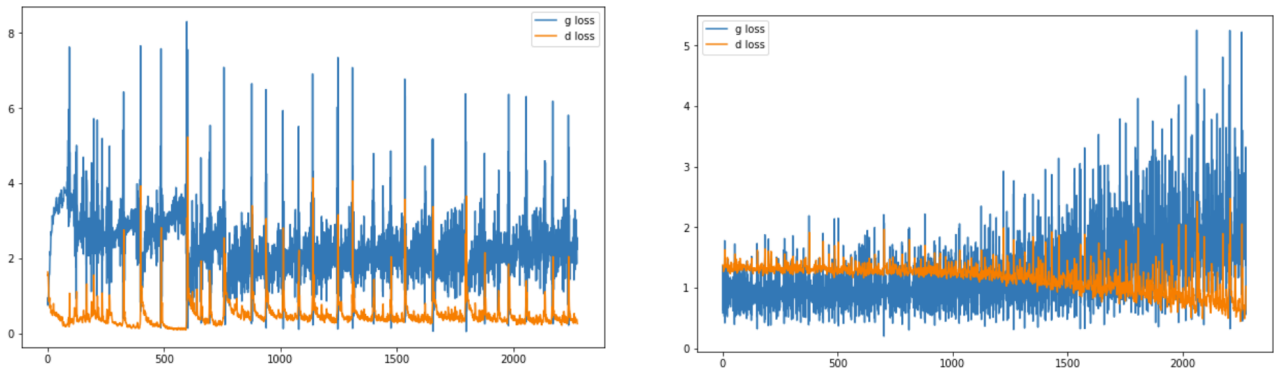
(d) Numbers generated at iteration 2276.

Figure (42) Evolution of the quality of the generated images.

We notice that during the learning, numbers look very different from the start. They are separated at the beginning and the generator learns to make each class without taking into account the others.

(g) **Could we remove the vector y from the input of the discriminator (so having $cD(\mathbf{x})$ instead of $cD(\mathbf{x}, \mathbf{y})$)?**
No, if we do that we would obtain an error because the input of the discriminator needs to be fed with a tensor of fixed size. Modifying the architecture of the discriminator such that it only takes the image $\mathbf{x}$ as input would not be a good idea either because we do not only need to control the realism of the generated image but we also need make sure the generated image corresponds to the label $\mathbf{y}$.

(h) **Was your training more or less successful than the unconditional case ? Why ?**
The training was more successful than the unconditional case because the generated images were more realistic. Indeed, in the unconditional mode the network had to find by itself the clusters corresponding of the classes and therefore the plausible images whereas in the conditional case it is easier for the network to learn how to generate a realistic image of a given class.



(a) (b)

Figure (43) Losses evolution of the (a) unconditional GAN and (b) conditional GAN.

(i) **Test the code at the end. Each column corresponds to a unique noise vector z. What could z be interpreted as here ?**
The vector z contains the information about the style and the background of the image that will be generated from z. It does not contain the class. We can use it to choose a "font" for the numbers we want to generate.

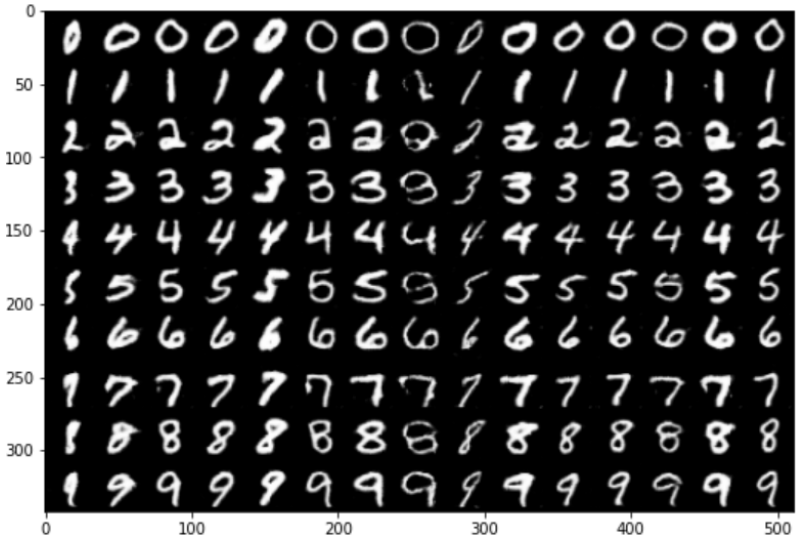


Figure (44) Results obtained by executing the last cell. Here $n_z = 100$.

References

[1] Phoenix Williams and Ke Li. *Art-Attack: Black-Box Adversarial Attack via Evolutionary Art*. 2022. DOI: 10.48550/ARXIV.2203.04405. URL: https://arxiv.org/abs/2203.04405.

[2] Dongbin Na, Sangwoo Ji, and Jong Kim. *Unrestricted Black-box Adversarial Attack Using GAN with Limited Queries*. 2022. DOI: 10.48550/ARXIV.2208.11613. URL: https://arxiv.org/abs/2208.11613.