TP 3 (a), (b), (c) et (d) : Reconnaissance des formes pour l'analyse et l'interprétation d'images

Réalisé par :

Gaël Marec et Léo Hein

Table des matières

1 3-a - Transfer Learning through feature extraction from a CNN.

1.1 VGG16 architecture

(1) : Knowing that the weights of the fully-connected layers account for the majority of the parameters in a model, we can estimate the number of parameters in VGG16.

- fully-connected 1 : $7 \times 7 \times 512 \sim 1.0 \times 10^8$
- fully-connected 2 : $4096 \times 4096 \sim 1.6 \times 10^7$
- fully-connected 3 : $4096 \times 1000 \sim 0.4^7$

The number of parameters of the VGG16 is approximately 120 million .

(2) : As VGG16 is trained on the ImageNet database, containing images categorized in 1000 classes, its output layer is composed of 1000 units. Thanks to the softmax layer, the output layer can be seen as a probability distribution. Each one of these units represents the probability of belonging to the corresponding class, the sum of all being equal to 1.

(3) : In order to test the network, we chose four specific pictures which are shown below. The great white shark and the sombrero are two classes existing in ImageNet while the surgeon mask and Flash McQueen aren't, which should give insights on how the network is associating the classes. We will also show decreasingly the resulting top 5 classes that the network associated to each picture.

```
great white shark, white shark, man-eater, man-eating shark, Carcharodon carcharias
tiger shark, Galeocerdo cuvieri
hammerhead, hammerhead shark
sturgeon
dugong, Dugong dugon
```

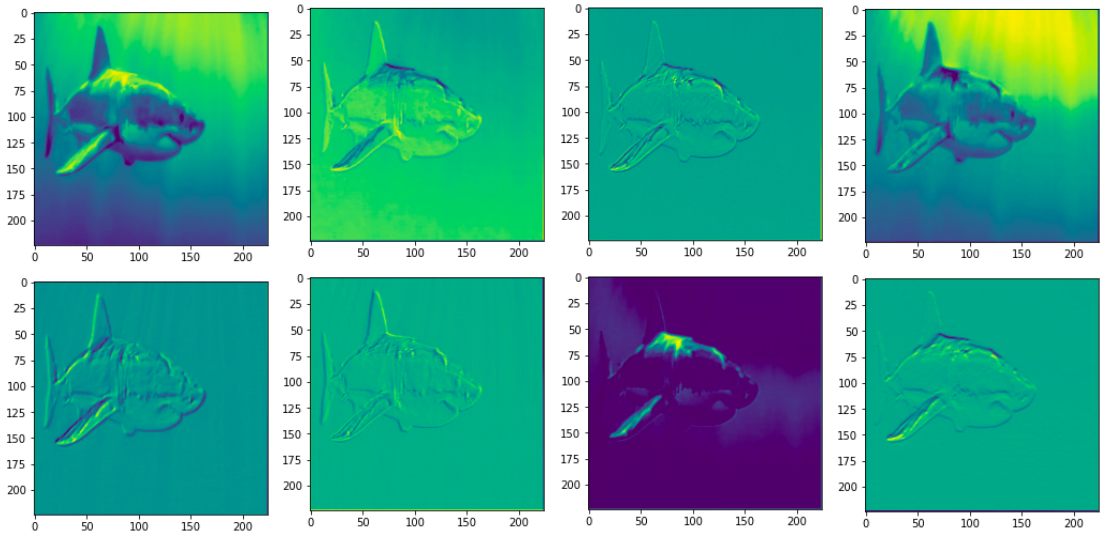
```
sombrero
cowboy hat, ten-gallon hat
drum, membranophone, tympan
gong, tam-tam
hamper
```

```
diaper, nappy, napkin
mailbag, postbag
purse
pencil box, pencil case
wool, woolen, woollen
```

```
crash helmet
sports car, sport car
car wheel
racer, race car, racing car
convertible
```

The network correctly classified the two first pictures as expected. We can see that the other guesses for the sombrero are about drums, probably because the shape and color are quite similar. Because the network can't classify correctly the two other pictures, it is interesting to see how it tries to associate them to particular classes. Most of the top 5 guesses for the surgeon mask revolve around objects with similar matter or made of tissue, the first one being a diaper. The results for Flash McQueen are very interesting as even if the network find similarities with cars or racing cars, the first guess is crash helmet. We can imagine that the network assimilates the eyes of Flash McQueen as a helmet visor.

(4) : In this question we show 8 randomly selected feature maps for the great white shark picture extracted just after the first convolutional layer.



These feature maps are globally quite different from each other, even if some of them present some similarities. Usually, the first convolutional layers in a CNN try to distinguish very general and variate features in an image such as color contrast or outlines. We can see that four of these features particularly focused on the shark outlines while others such as the fourth or seventh ones seem to focus on the picture colour contrast. The deeper we extract the features in a CNN, the more specific they tend to be.

1.2 Transfer learning with VGG on 15 Scene

1.2.1 Approach

(5) : Since the VGG network is already trained on the image-net dataset, one can use those trained parameters for a different task and add minimal training steps rather than totally re-train the network. This is the main benefit of transfer learning : the training phase is much quicker and the network reaches even results faster. Also, 15-scenes does not contain a lot of examples per class in comparison with ImageNet, meaning that VGG may not be able to reach good results on 15-scenes by lack of data. Using transfer learning enables the network to take maximal advantage of the target examples by using them to fine-tune the network rather than whole training it.

(6) : As we saw in question (4), the first extracted features in a CNN are quite general and could be used to classify a lot of different pictures from different classes, as most "natural" pictures (from ImageNet and 15-Scenes) share very similar features one not necessarily see as an exterior user (corners, edges, colors...). Then, even if the classes in ImageNet and 15-Scenes are not the same, using general features trained on the ImageNet database in order to classify 15-scenes pictures could help reducing training time and we would reach at least even performances using smaller datasets. Also, because the network has already been trained on the source dataset, one can use the whole target dataset to fine-tune the network rather than training it entirely. Intuitively, the network has more target data to focus on and the resulting performances should be better. However one must be aware of the overfitting risk.

(7) : Transfer learning may be inefficient and irrelevant in several cases. If the pre-learning and final dataset are too independent, it can causes worse performances than training from scratch and we then talk about negative learning. It can intuitively be understood with features extraction. Indeed, if the source and target datasets are too different, the features extracted from the source dataset, even in the first layers, may be irrelevant regarding the target dataset, as we would obtain very different low-level features by training from scratch on the target dataset. Besides and more generally, it can be a tough and demanding process to know at which optimal depth one should extract the features of the pre-trained network to use it on the target domain and tasks.

1.2.2 Feature Extraction with VGG16

(8) : As we discussed in the previous subsections, the deeper the features are extracted in the network, the more specific they tend to be. One should be able to evaluate at each level the pre-trained features should be extracted depending on the similarity between the source and target datasets. Intuitively, if the datasets and tasks are very similar, one should extract the features at a deep level in the network and vice versa.

(9) : An image in black and white can be seen as an RGB image when all coordinates are equal. Then we can simply duplicate three times our black and white image to get an RGB image. Another idea but more complicated is to colorize the 15-Scene dataset with a neural network to get a RGB dataset.

1.2.3 SVM classifiers learning

(10) : After the feature extraction step, the only requirement is to train a classifier. It could be done with fully-connected layers, a support vector machine or even global average pooling. However, this classifier must be adapted to the target task. In this practical, the target task is to classify the 15-scene data while the VGG16 architecture is designed to train the ImageNet dataset with a final classifier layer of size 1000. That's why we cannot use the neural network as it is. However, with slight modifications in the classifier layers, one can directly use the network as a classifier. Yet, one must be careful to freeze the parameter of the feature extraction part of the network or to adapt the learning rate so the use of transfer learning makes sense.

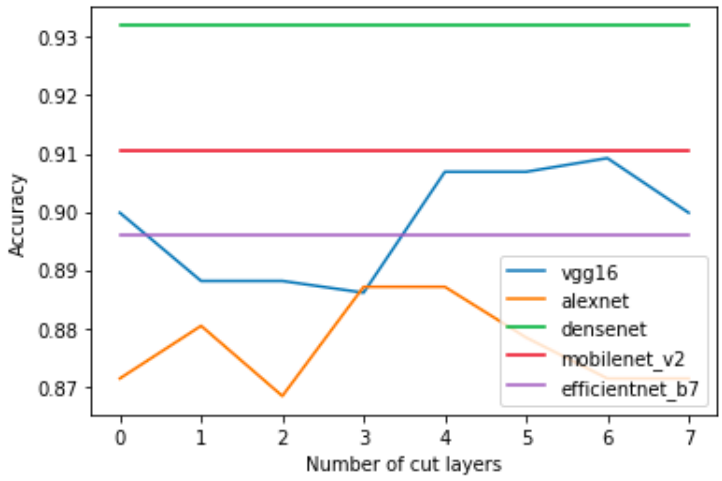
1.2.4 Going further

(11) : We then tried several implementations in order to improve the performances.

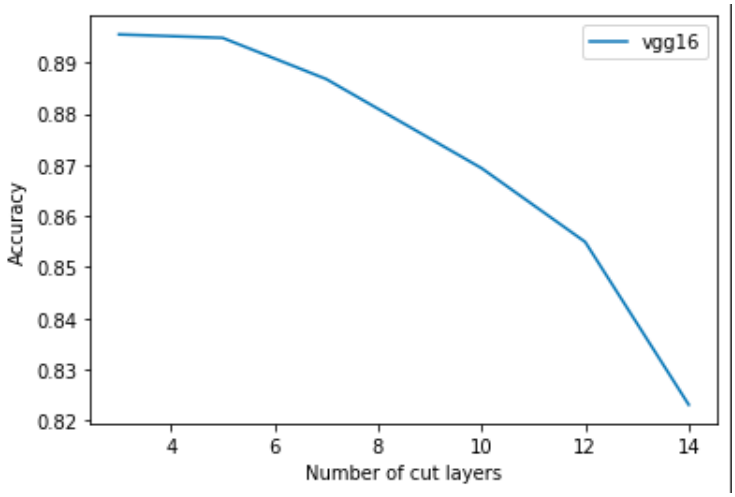
To begin with, we changed the layer at which the features are extracted in the classifier only. The deeper we extract the features in the network, the more specific the features tend to be. At the same time, we also tried several other pre-trained networks : AlexNet, Densenet, MobileNet-v2 and EfficientNet-b7.

AlexNet is the anchor point of VGG net, the architectures are similar but AlexNet uses bigger receptive fields and is less deep, which implies worse performances than VGG. DenseNets are convolutional neural networks that utilises dense connections between layers, through dense blocks, where all layers are connected (with matching feature-map sizes) directly with each other. MobileNetV2 is a convolutional neural network that seeks to perform well on mobile devices. It is based on an inverted residual structure where the residual connections are between the bottleneck layers. EfficientNet is a convolutional neural network architecture and scaling method that uniformly scales all dimensions of depth, width and resolution using a compound coefficient.

The graphic below shows the results of this experiment. The best performances are obtained with DenseNet whereas, as expected, AlexNet is the only network showing worse performances than VGG16. The difference in performance between the architectures is caused by the "quality" and the distribution of the features extracted by each net. Besides, only VGG16 and AlexNet see their performances changing with the number of cut layers in their respective classifier part. For VGG16, the best performances are obtained when only a few layers are left in the classifier part (4, 5 and 6 cut layers), but a conjecture about the relation between depth and performance is hard to guess with this graph only.



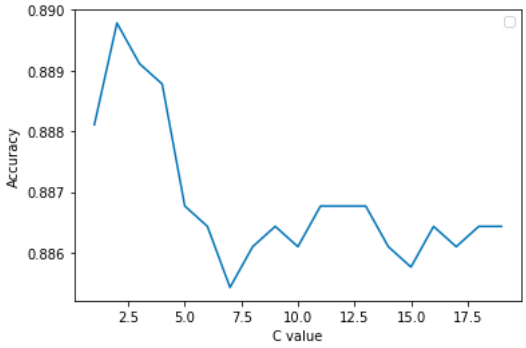
Next we focus on the VGG16 architecture and cut the classifier entirely with additional layers in the features extractor part of the network. We extracted the features repeatedly after each convolutional block starting from the end of the network to see if there are differences in the classification performance, given the specificity of the extracted features the classifier is given in input. The graph below shows the resulting performances. Each point corresponds to an extraction after one more convolutional block, which can be composed of 2 or 3 layers (conv2d, ReLU, pooling).

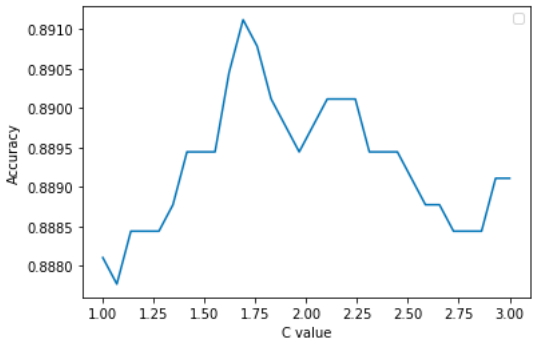


The performances decrease with the number of cut layers in our neural network, meaning that the target task (classifying 15-scenes pictures) can easily take advantage of the most specific extracted features from our source domain (ImageNet), by the VGG16 architecture.

We then replaced the last layer of VGG16 with a new fully-connected layer but couldn't get rid of memory errors while trying tuning it.

We then tuned the C value in order to achieve better performances. We reach an accuracy of 0.891 with a C value of 1.68, as illustrated by the two following graphs.

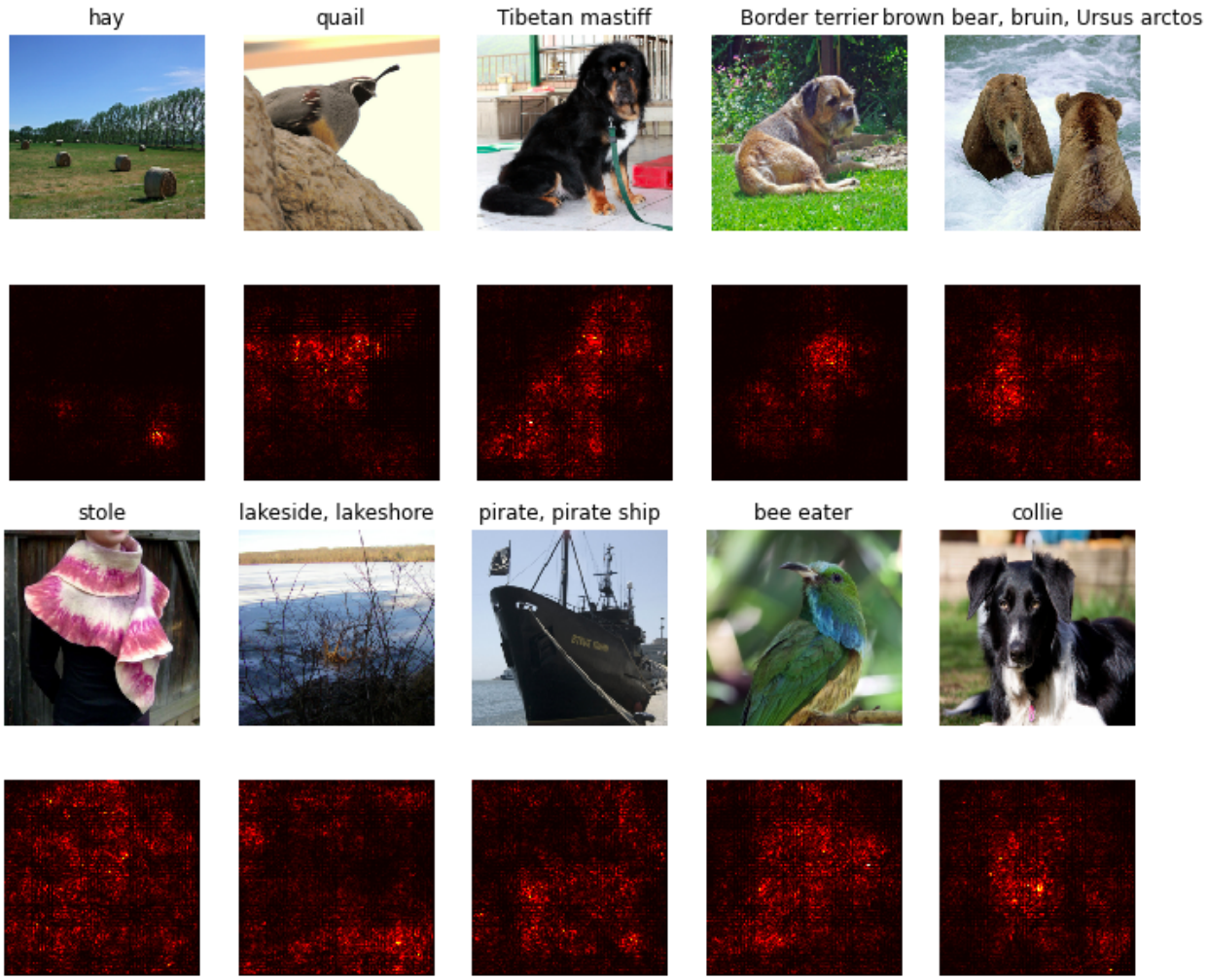




2 3.b - Visualizing Neural Networks.

2.1 Saliency Map

(1) A saliency map is a map which indicates the important zones of an image. In our case of studying a CNN, this corresponds to the most impactful pixels for predicting the correct class. We show below the resulting saliency maps for a few different input examples.



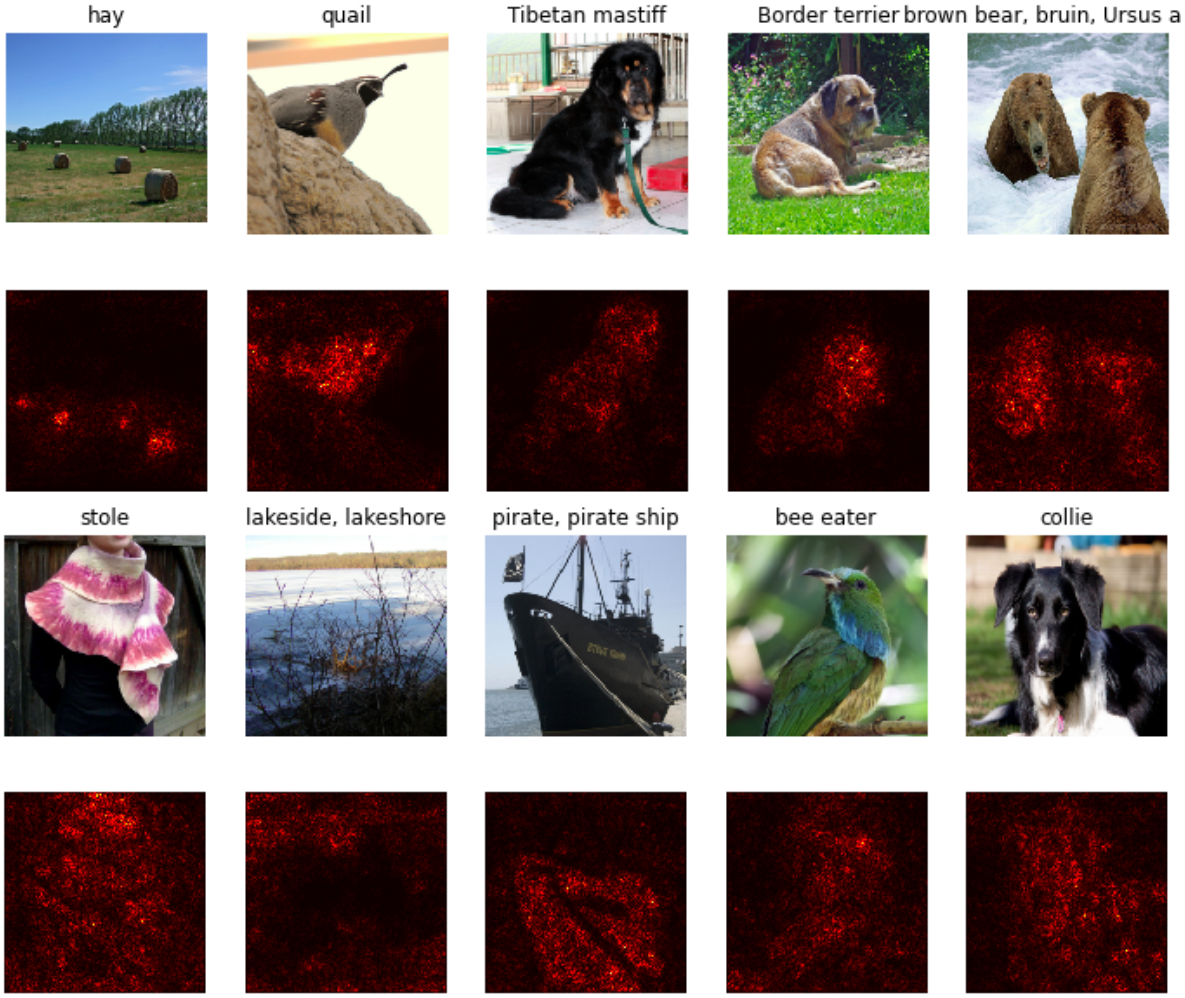
The red pixels on the saliency maps are the more impactful for the correct classification. Most of the maps are relevant considering the animal or object to classify but one can observe that the number of impactful pixels differs a lot. For example, the red pixels on the hay saliency map are way more clustered that the ones on the stole map, even when taking into account the size of the object on the picture. With most of these saliency maps, one could guess a very vague shape of the objects on the initial picture.

(2) Even if the saliency maps are very useful to recognize the important regions of a picture by giving a visual characterization of the input, it also gather several limitations. Recent research papers demonstrate that pixel attribution methods can be

very fragile to perturbations, can be highly unreliable and can be insensitive to model and data, which all are undesirable properties. More generally, the main limitation of saliency maps is their evaluation, making it difficult to know whether an explanation is correct or not. Besides, more intuitively, it may seem more relevant to try to reason in terms of regions impact rather than pixel impact in order to consider the classified object in the picture as a whole, with the spatial relations between its pixels, or at least as a minimal set of regions. When we try to distinguish a dog from a cat, we do not take into account the pixels separately, but try to compare parts of them, just as CNNs do with features. Using global average pooling on the extracted features of a CNN is one way to get region-based class activation maps.

(3) Even if its main purpose is to interpret the network, saliency map can also be seen as an instance of image segmentation. One can imagine that with little additional work on regions delimitation, it would be possible to distinguish the boundaries of the classified object.

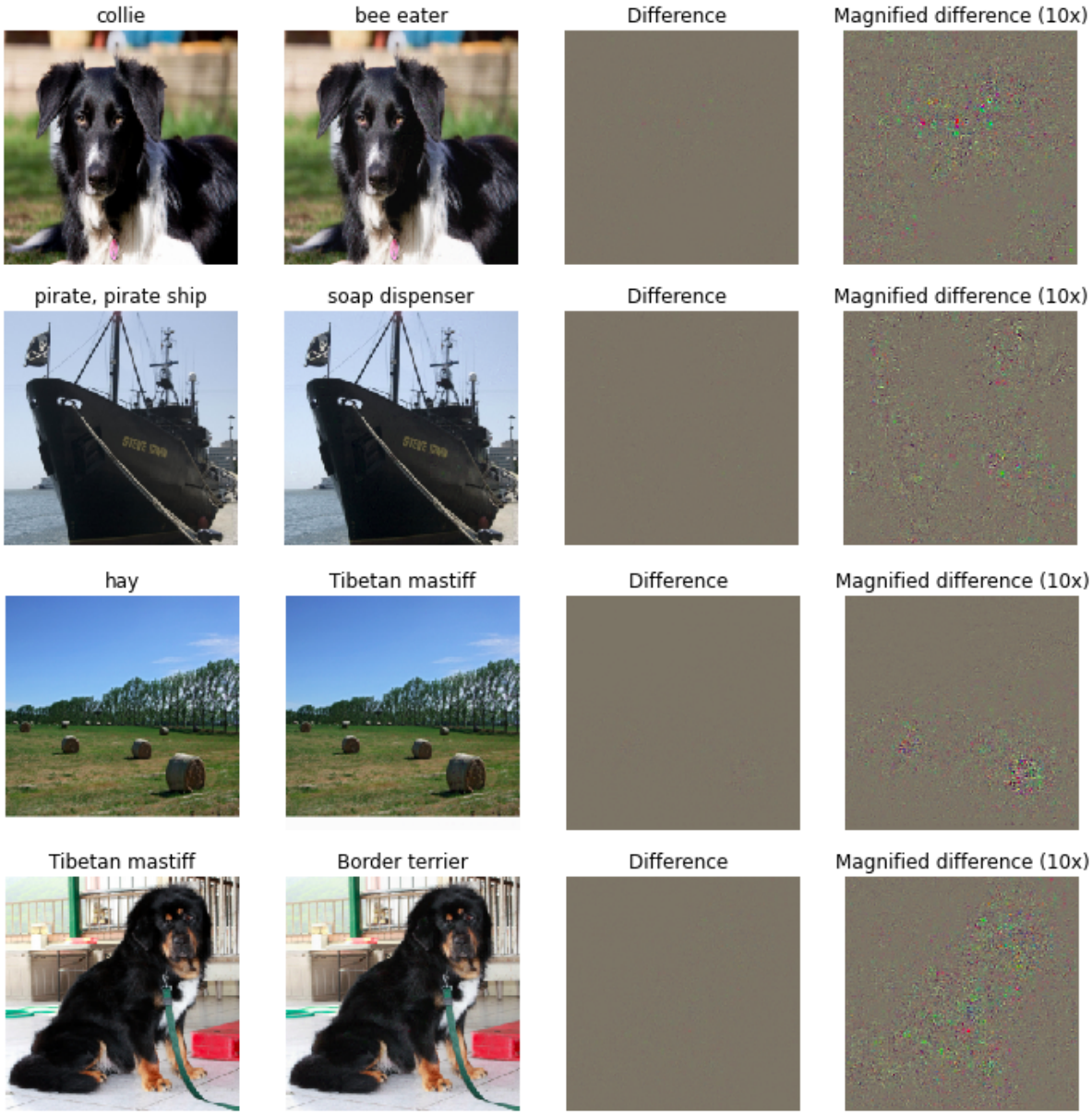
(4) We now apply the same method of the question (1) with a different pre-trained network : the vgg16 architecture, in order to discuss the observed differences between the two architectures behaviours.



With the VGG16 architecture, the objects are way better delimited on most of the saliency map, especially the pirate ship. These saliency maps are globally easier to interpret.

2.2 Adversarial Examples

(5) We show on two examples the obtained results of the implemented method that fools our model when classifying input images by slightly modifying them.



The differences are nearly impossible to distinguish with human eyes even if the source and target classes are very dissimilar, one shall heighten the pixel differences by a factor of 10 in order to see it clearly. Interestingly, the most modified pixels tend to be the most discriminating ones of the source class, as it is clearly shown on the last two examples. It seems the behaviour of the fooling process is more focused on

making disappear the evidences of the source class than making up ones for the target class.

(6) Using fooling images on CNNs in order to cause miss-classification in the "human sense" can bring huge issues to a lot of different applications. For example, autonomous vehicles may be very troubled and cause disasters if one succeed in giving false information through fooling examples to the vehicle environment recognition system. For example, Tencent's Keen Security Lab showed they were able to manipulate a Tesla Model S into switching lanes so that it drives directly into oncoming traffic. CNNs security and robustness to attacks raises a lot of concerns in the research community.

(7) With this method, the "quality" of the obtained adversarial example may not be satisfying as there is no control on the created perturbations. We can reduce adversarial examples generation to this minimizing problem :

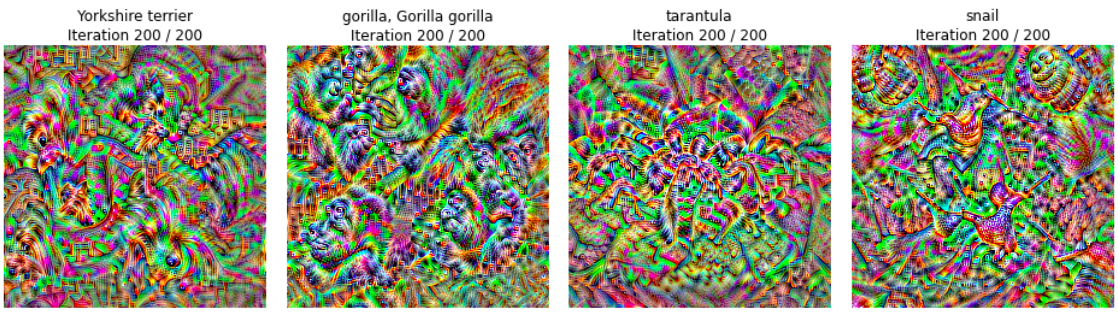
$$arg\min_{r} J(f(x+r), y) + QualityPenalty(r)$$

Where J is the loss function measuring the distance between outputs of the model. f represents the model and y is the target output. r is the perturbation and x+r is the adversarial example. The QualityPenalty function can for example be the $L_0, L_1 or L_\infty$ norms, which gives different resulting perturbations and aim at improving their quality.

As a trendy subject in the research community, a lot of improvements are made on the generation. For example, an article (Zhao and al., 2018) states that the created perturbations are often unnatural, not semantically meaningful, and not applicable to complicated domains such as language, and use GANs to make them more natural. Another one (Jaeckle and al., 2021) presents a method using Graph Neural Networks to deal with the following observation : "Generative models often fail to match the performance of iterative optimization-based methods on finding minimal perturbations."

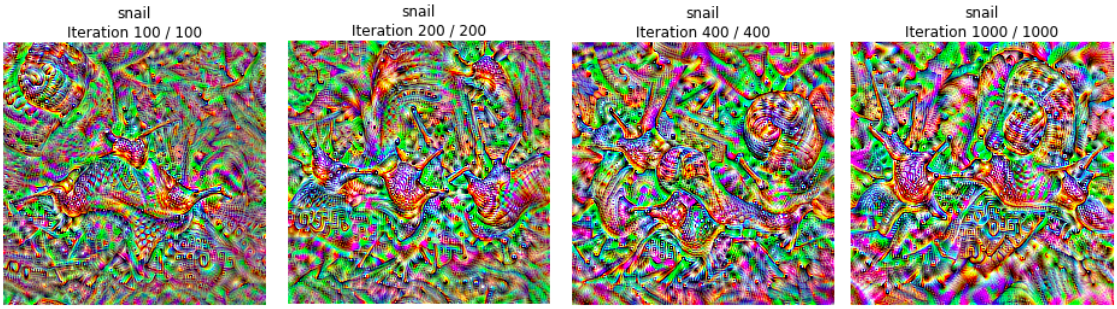
2.3 Class Visualization

(8) We show the results for four different classes with the implemented hyperparameters values. For each target class, this technique aims at creating corresponding discriminant patterns from noise. By looking closely, one can distinguish characteristics features of the target class : long hair and snouts on the first example, gorilla faces and hands on the second, spider legs on the third and snail antennas and shell on the last one.

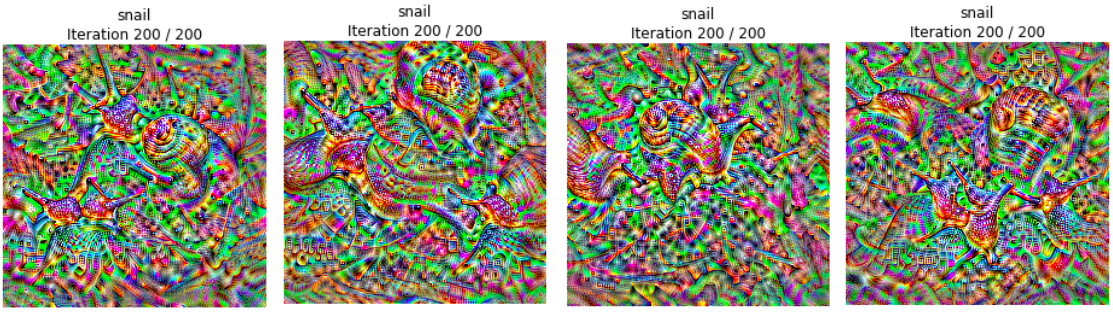


(9) We now vary the number of iterations, the learning rate and the regularization weight each one at a time to see how influential are these hyperparameters.

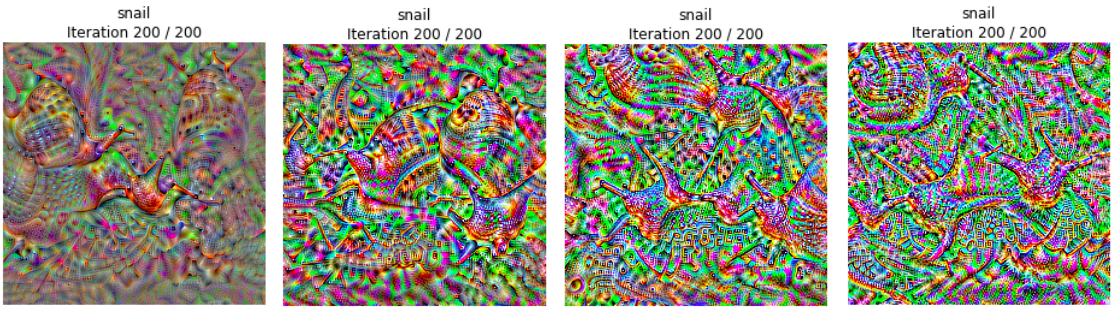
We start by varying the number of iterations (100-200-400-1000).



We continue with the regularization weight (1e-4, 1e-3, 1e-2, 1e-1).

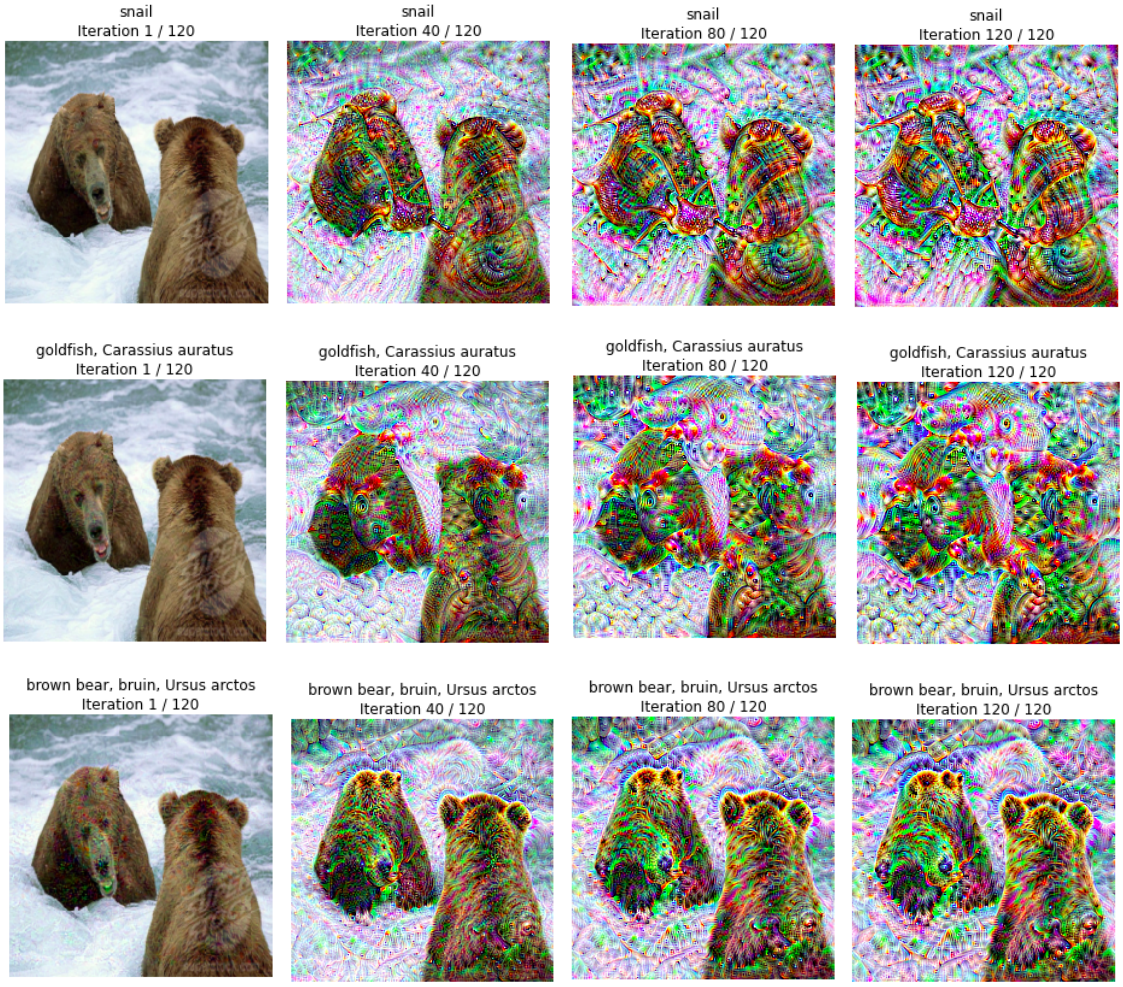


We end with the learning rate (1-5-10-20).



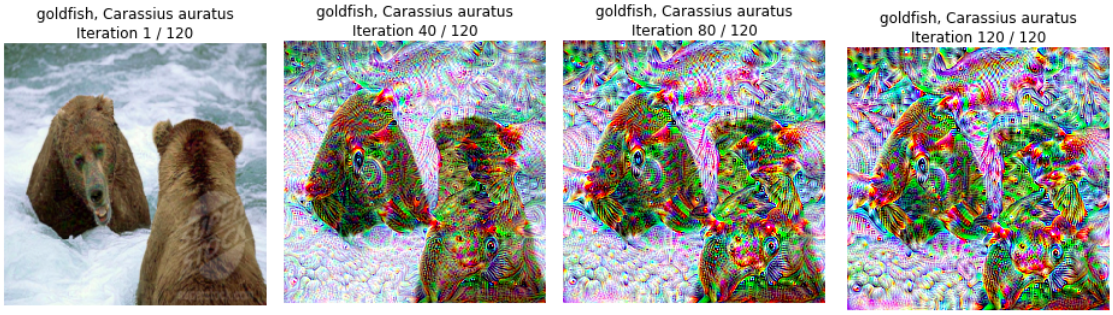
It is very hard to conclude from these different pictures. The learning rate seems to have the biggest influence on the results and a learning rate of 1 may seem a bit small to converge properly in less than 200 iterations. In order to compare more properly, we noted the iteration after which the picture is classified as the target class for each one of these hyperparameter sets. Obviously, the iteration number doesn't change the switching iteration. For the regularization weight, the switching iterations are respectively 12, 9, 15 and 16, meaning that a weight of around 1e-3 seems optimal. For the learning rate, we respectively obtain 35, 9, 8, 13, meaning that a learning rate of approximately 10 seems optimal.

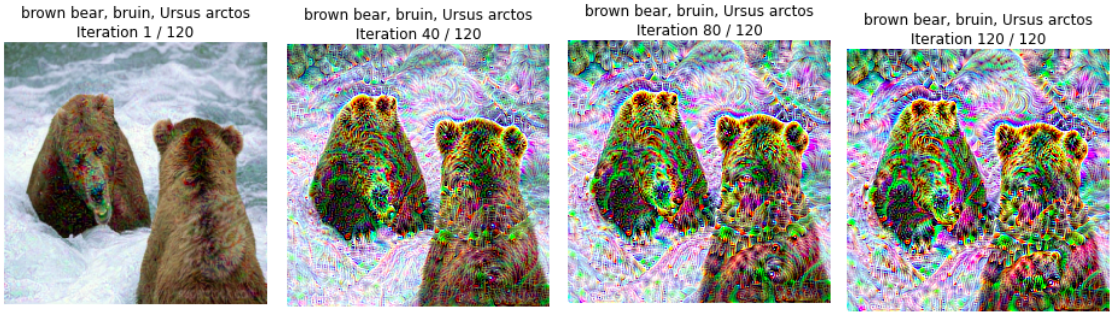
(10) We now use an image from ImageNet as the source image instead of a random image and observe the evolution of the picture with different target classes (snail, goldfish, brown bear).



We can observe characteristics of the target class after enough iterations. For example, we can clearly distinguish a fish in the second experiment. We also can see that when the target and source classes are the same, the process tend to highlight what might be the discriminant features of this class. Indeed, in the third experiment, the edges, ears and particularly the snout of the brown bear are "enlightened", meaning that the process is trying to emphasize them in order to increases the prediction score of this class.

(11) Finally, we test this technique with another architecture, the used model is this time the VGG16 Net. On contrary to saliency maps, the model change doesn't impact that much the results. The detected patterns remain nearly the same for both the target classes.




brown bear, bruin, Ursus arctos
Iteration 1 / 120
brown bear, bruin, Ursus arctos
Iteration 40 / 120
brown bear, bruin, Ursus arctos
Iteration 80 / 120
brown bear, bruin, Ursus arctos
Iteration 120 / 120

3 3.c - Domain Adaptation

3.1 Practice

1) : As the article explains, Gradient reversal ensures that the feature distributions over the two domains are made similar (as indistinguishable as possible for the domain classifier). If we remove the GRL we can have different representations for different domains because there is no more opposition in the learning of green and pink neural networks.

2) : In the DANN network, we add the domain classification branch which allows us to generalize the naive approach to other domains (here MNIST-M). But the abstraction of the domain for feature extraction is not perfect so we lose performance when we test only on MNIST. We have learned to classify in a much more general way but if we compare to a network that has been specifically designed to classify MNIST and only MNIST it is normal that the performance is degraded.

3) : The λ parameter controls the trade-off between the objectives of the pink and green networks (which determine the feature extraction). In practice we gradually change it from 0 to 1 with the schedule : $\lambda_p = \frac{2}{1+\exp(-10p)} - 1$.

4) : It is not always easy to collect labelled data. Then we can apply a semi-supervised learning method such as *pseudo-labeling*. The idea is the following, we have a certain number of labeled data (X_1, Y_1) and other unlabeled data X_2. We learn on (X_1, Y_1) in order to obtain pseudo-labels Y_2 on X_2. Then we find ourselves in a supervised learning problem $((X_1, X_2), (Y_1, Y_2))$.

4 3.d - Generative Adversarial Networks

4.1 Generative Adversarial Networks

4.1.1 General Principle

1) :

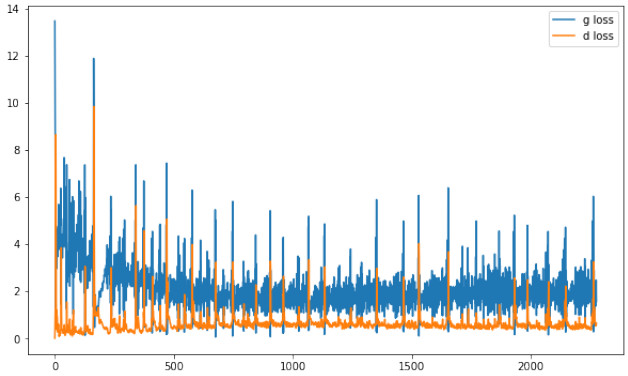
- The equation (6) corresponds to get the best generator, whose the images which are generated by are never detected as fake by the discriminator. If we only train with equation (6), the generator will just learn to dupe a bad discriminator, but we (humans) are not bad discriminators when it comes human faces. So we understand why it would be not satisfying in most cases.
- The equation (7) corresponds to get the best discriminator, which can always determine what images are fake. But if we only train with equation (7), generated images will be obviously fake and the discriminator learned won't be interesting.

2) : Ideally we want the generator to look like $\mathcal{P}(X)$ because that's exactly how the real data are generated.

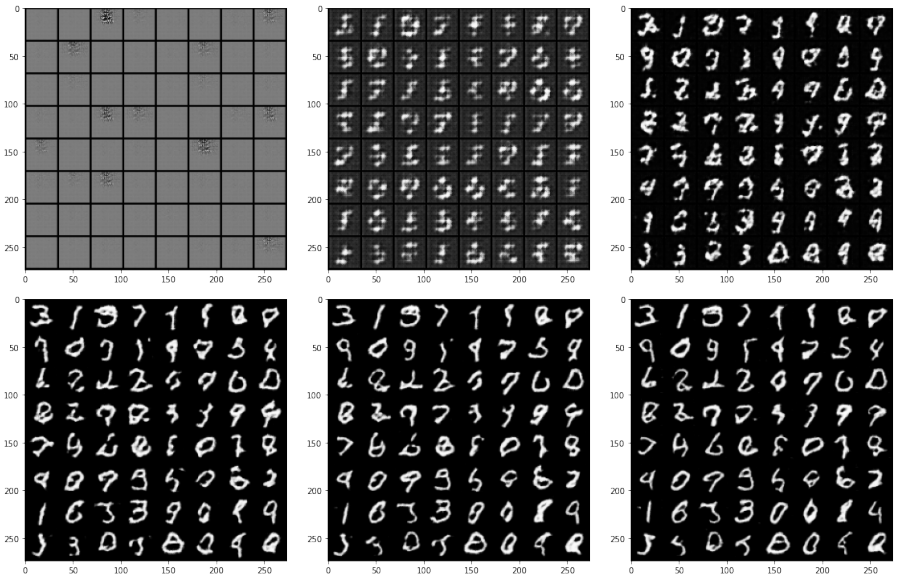
3) : Rather than training G to minimize $\log(1 - D(G(z))$ (the true equation should be $\min_G \log(1 - D(G(z)))$ the author proposes to maximize $\log D(g(z))$. That give the same results but provides stronger gradients in learning.

4.1.2 Implementation

Default settings
4) :



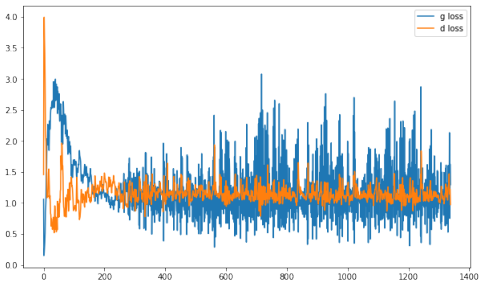
g-loss and d-loss against number of iterations



Generated image at respectively 0, 200, 500, 1100, 1600 and 2000 iterations

We observe that the losses globally decrease during all the training even if they seem to decrease much more slowly after 800 iterations. The two losses seem to be highly correlated, which is not surprising given their definition. The algorithm is quite stable despite the appearance of some loss peaks (both in G and D because when the generator becomes worse, the classifier also becomes worse and vice versa). Some of numbers are not clearly identifiable at the end of the training, others can be identified by humans without any ambiguity. We notice that some numbers change during the learning process, for example, a number that looks like a 4 becomes a 7 at the end of the learning process.

5)
ngf>>ndf :



loss for ngf=256 and ndf=32

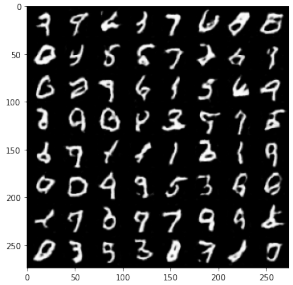
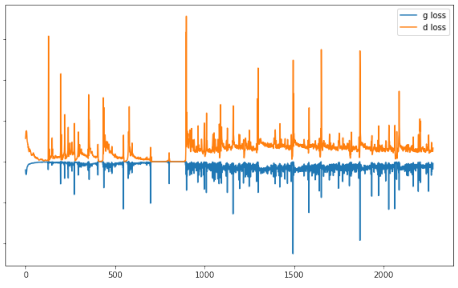
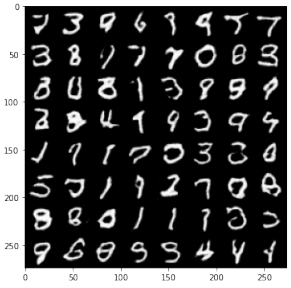


image generated at 1300 iterations

We notice a less good stability of the algortihm but we converge all the same towards networks of qualities equivalent to the default settings. Moreover we notice that d-loss is greater and g-loss is smaller than in default settings.



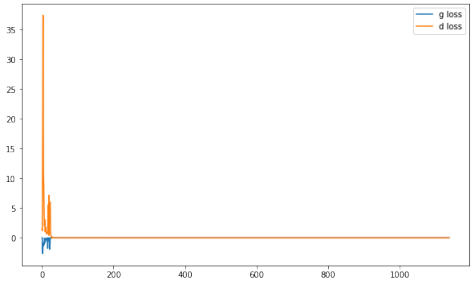


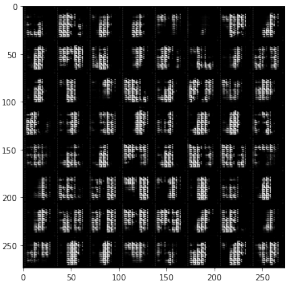
training with the "true" loss :

Image generated at 2000 iterations.

The shape of losses is symetric because they are almost the opposite of each other mathematically. The algorithm manage to give good results despite of the theoretically weakness of the gradient.

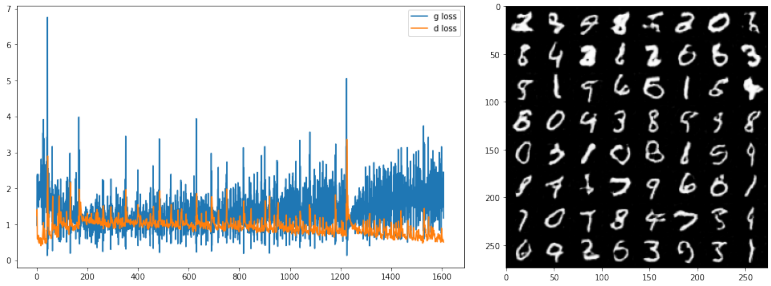
Modified Learning rate :



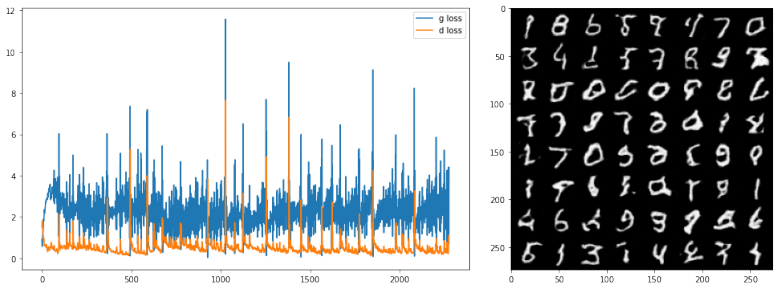


We multiplied the both learning rates by 100, the results is a convergence of gradient in a local optimum. We can't recognize any number on the generated image.

Decrease n_z to 10 :



Increase n_z to 1000 :



When we increase n_z to 1000, the learning process need more iterations to make G able to generate a good fake MNIST. However, the bigger the latent space dimension is, the larger is the variety of fake generated images.

4.2 Questions

6 8)

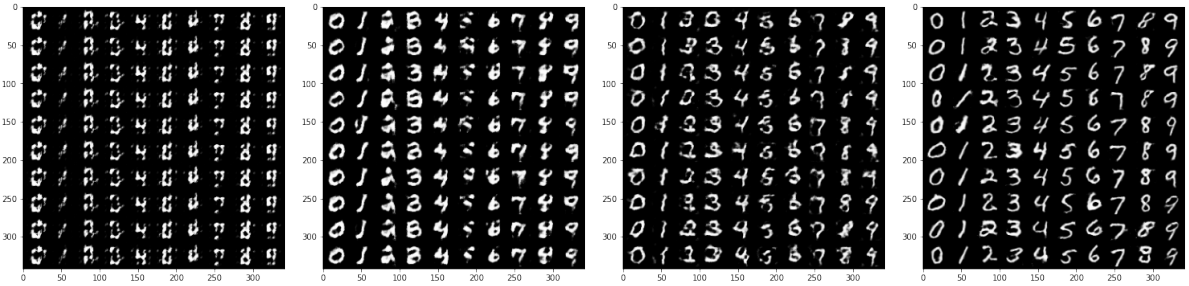
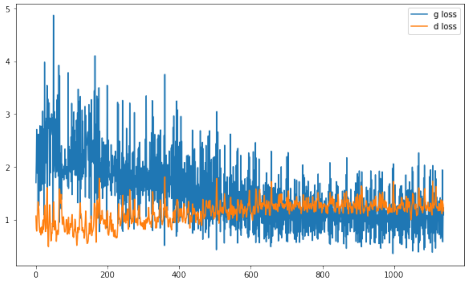


image generated for respectively, 1, 200, 400 and 1100 iterations.

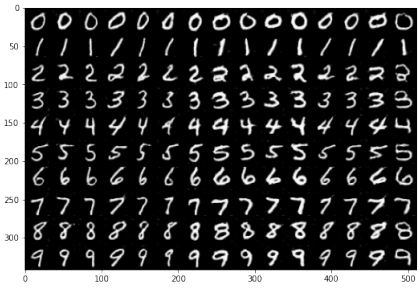


Compared to GAN, DCGAN learns faster and better. After 1000 iterations, there is no ambiguity about the numbers generated. Moreover, no errors are observed on the

generations associated with the labels. The good results of DCGAN are not surprising since we learn with additional data, the Y.

7) If we remove the vector y from the input of the discriminator, so the generator will learn to trick the discriminator independently of the value of y. So, it works but it is the same as unconditional GAN.

9)



We see that each column represents the same numbers but differently, it is the interest of choosing a random vector z. Now that we have a data generator, it's almost as if we had a infinite dataset where each z corresponds to a single datum. We finally managed to simulate the law of $\mathcal{P}(X, Y)$