SCIENCES
SORBONNE
UNIVERSITÉ

Deep Learning Practical Work 3

SIHAMDI Mostefa, BOUSBA Abdellah

UE RDFIA 2021-2022, Encadrants : Arthur Douillard, Alexandre Rame, Asya Grechka

M2 DAC

I. Transfer Learning through feature extraction from a CNN :

A. VGG16 Architecture:

- Question 1 :

The number of parameters of a fully connected layer is the input dimension multiplied by the output dimension so:

- First fully connected layer : $7 \times 7 \times 512 \times 4096 + 4096\ bias = \ 102764544$
- Second fully connected layer : $4096 \times 4096 + 4096\ bias = \ 16781312$
- Third fully connected layer : $4096 \times 1000 + 1000\ bias = \ 4097000$

By summing all the above, we get 123633664 parameters. Which takes the majority of the parameters of the VGG16 model (138 mil).

- Question 2 :

The size of the output is 1000 which corresponds to the probability distribution of belonging to each class.

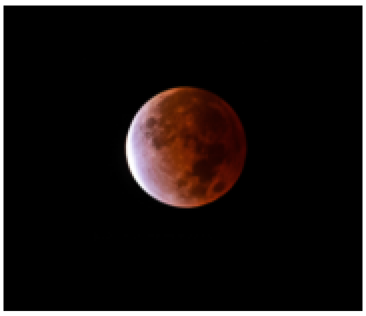
- Question 3 :

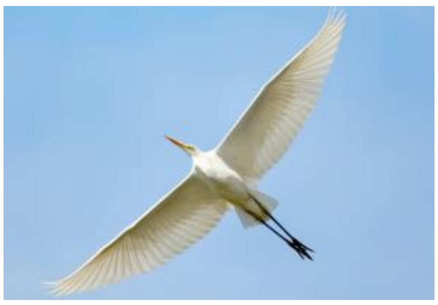
For each image we have show the predicted class :

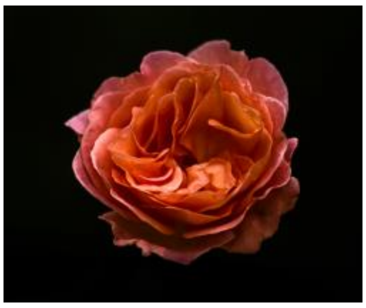
Egyptian cat, acc = 68.97%

Palace, acc = 71.55%

bee, acc = 91.22%


Ping-pong ball, acc = 21.25%


spoonbill, acc = 35.35%


coral fungus, acc : 61.33%

We can notice that the cat, palace and bee are correctly predicted with an acceptable score, for the spoonbill it is predicted correctly but with low probability which can be explained by the fact that in the image the bird form is clear but the type of bird is hard to detect. On the other hand for the moon and the flower the prediction is wrong which is logical because the similarity between the moon form (round) and a ping pong ball, on top of that the color is close to red, for the flower the flower the angle of the picture shows curves that are close to a coral fungus.

- Question 4 :

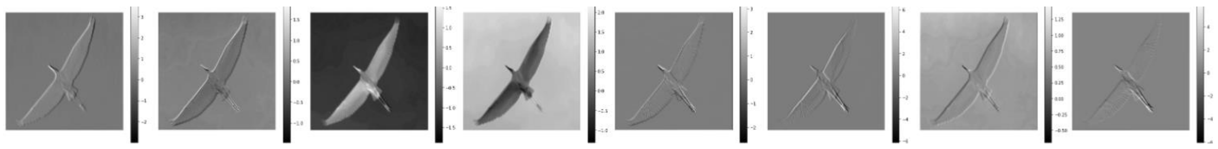


Figure 1 - (bird) 8 samples of the first conv layer

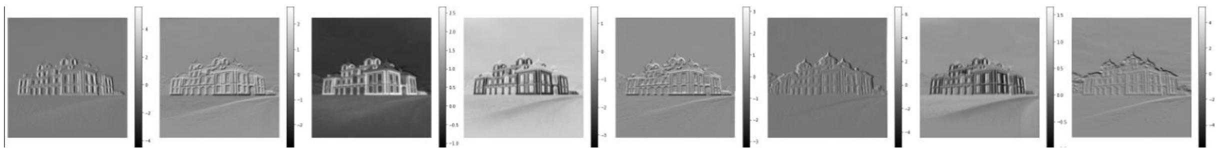


Figure 2 - (palace) 8 samples of the first conv layer

We take 8 samples of the first convolutional layer maps, we can see that the filter captures the most important characteristics each time. For example, the third and fourth map detects color, the eighth captures shades and the others capture levels of contour. We can also notice that each map captures same thing with different images.

B. Transfer Learning with VGG16 on 15 Scene:

- Question 5 :

We can't train VGG16 directly on 15 Scene because: First, the number of output classes that is hardcoded to classify 1000 classes. Also, fixing this issue won't be sufficient due to the small size of the dataset that may cause an overfitting.

- Question 6 :

The pre-trained model on imageNet can be used to extract global features of images (contours, edges..) and we can keep the same weights because this step is the same for all tasks and we only need to train the last part for classification.

- Question 7 :

The limits of feature extraction are that if the model is trained on a certain type of images and encounters a new type of image, the features will be hard if not impossible to be detected for example change in luminosity or saturation…etc

- Question 8 :

The layer we extract features from becomes higher in level the further we go, the first layers extract some contour or shapes but for the last layer it can extract window of a house or an eye in the face or a wheel on a car..etc

- Question 9 :

VGG16 expects RGB images, to convert our black and white to RGB we can simply duplicate the image 3 times on each channel making R = G = B.

- Question 10 :

Yes, instead of the SVM we can use a fully connected layer with an output size of 15 classes at the end of the VGG network instead of the one with 1000 classes output. To avoid changing the weights of the previous layers we should freeze them, this way we only train the classification part

- Question 11 :
    - Change the layer at which the features are extracted: we calculate the accuracy and execution time for different levels of extraction:

	Accuracy	Time (s)
VGG16 relu7	90.08%	81.8 s
VGG16 relu6	**91.69%**	**80.9 s**
VGG16 no classifier	90.94%	109.3 s
VGG16 less features	90.26%	392.6 s

We can notice that removing another ReLU layer returns better results and is a little bit faster but as we remove the FC layers we get more input data for the SVM so it takes longer to train and the accuracy doesn't improve much.

    - Trying other pre-trained networks:

	Accuracy	Time (s)
VGG16	91.69%	80.9 s
GoogleNet	91.21%	30.1 s
ResNet18	91.27%	24.0 s

Other networks are faster but the VGG accuracy is slightly better.

    - Tuning C parameter:

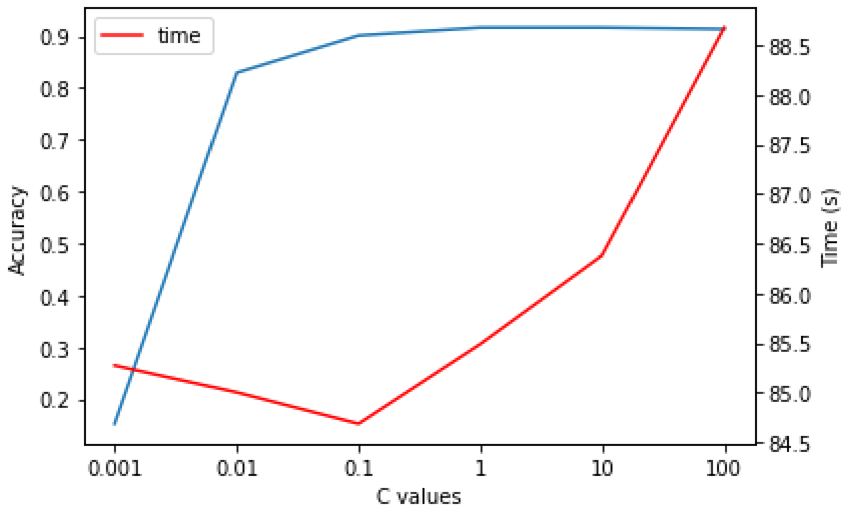


Figure 3 - accuracy and time for different C values VGG16

The execution time is higher the bigger is C, but the accuracy is at its best when C equals to 1.0

- <u>Dimension reduction (PCA):</u>

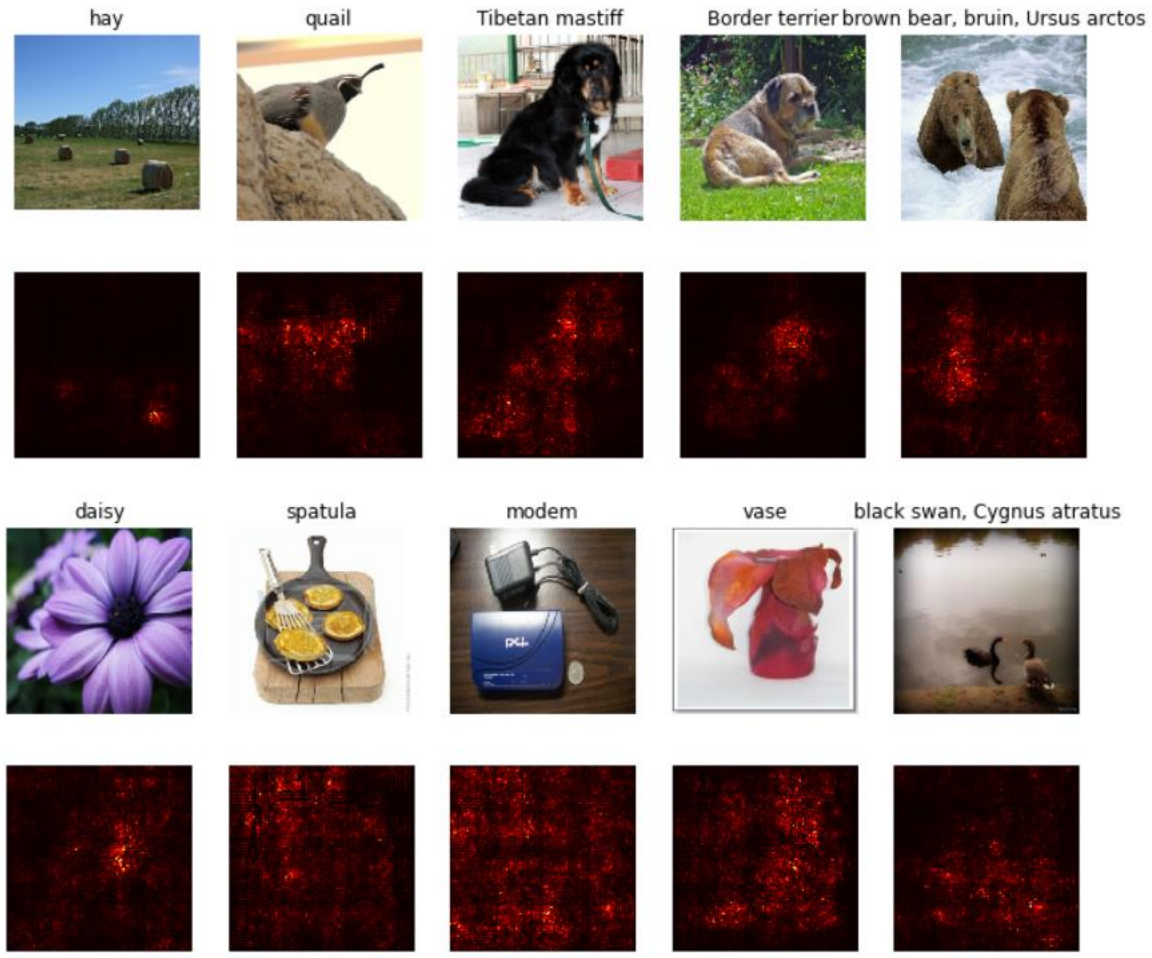
After using the PCA for reducing the dimension the score doesn't improve but the execution time is faster, so we can use this method to obtain faster results but it can't help us improve the accuracy.

II. Visualizing Neural Networks:

A. Saliency Map:

- Question 1 :

After execution, we obtain the Saliency Map of our images.



we notice that the most important pixels (in red) generally correspond to the pixels of the object, For the image of the hay, for example, we notice that the most important pixels are on the haystacks, also for the image that represents a dog the model seems to have focused on the area where the dog is.

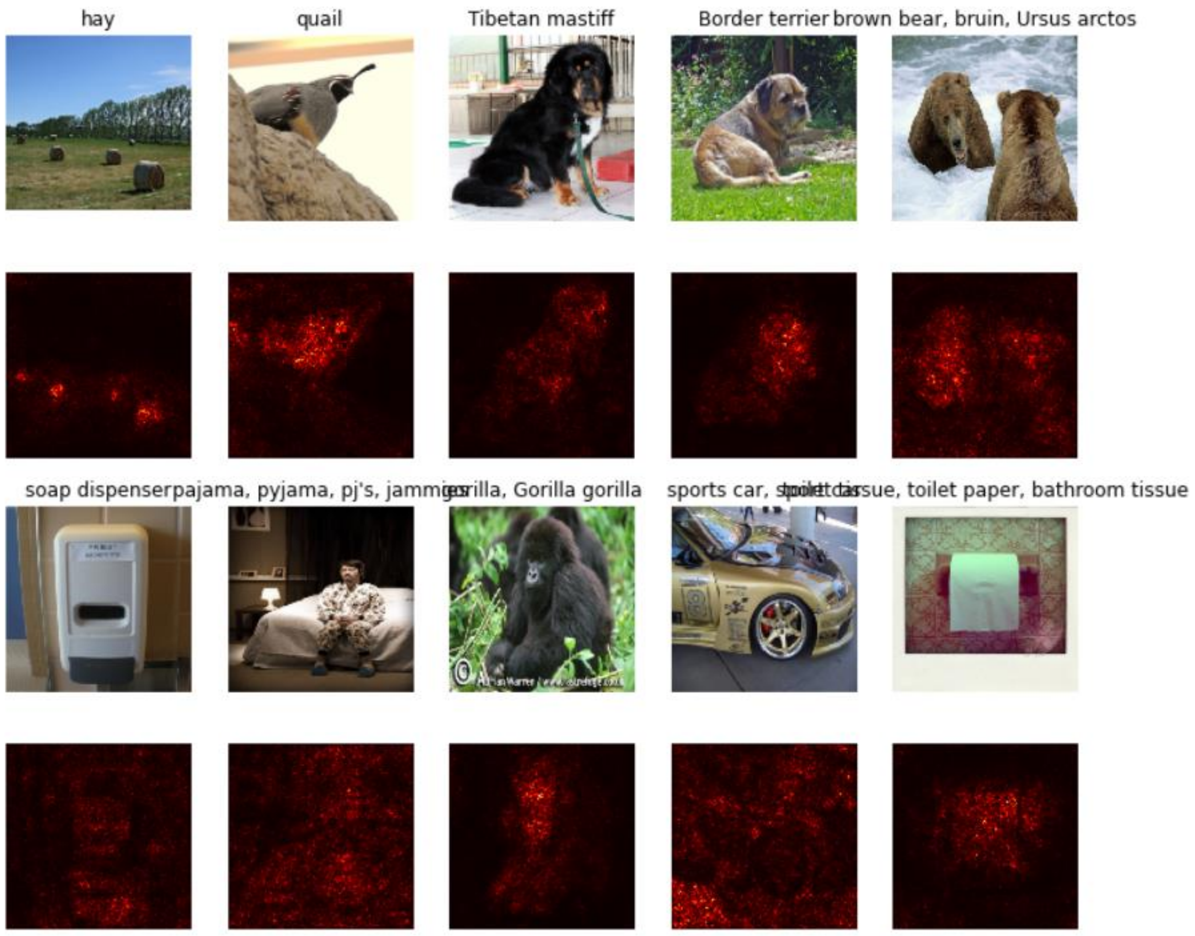
- Question 2 :

the limits of this technique are: Indicates the importance of several pixels instead of each pixel for identification, Does not determine all areas that are important.

- Question 3 :

Yes we can use this technique for a different purpose than interpreting the network,for example image segmentation, object localization

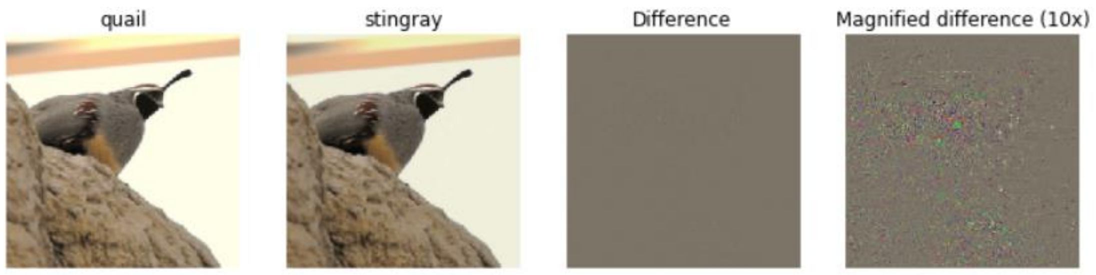
- Question 4 :


hay
quail
Tibetan mastiff
Border terrier
brown bear, bruin, Ursus arctos
soap dispenser
pajama, pyjama, pj's, jammies
gorilla, Gorilla gorilla
sports car, sport car
toilet tissue, toilet paper, bathroom tissue

We notice that the VGG16 model is more correct than the squeezenet model, for example the image of hay, we notice that there are important pixels that participates in the predicition, we find them in the results of vgg16 but not in the results of squeezenet

B. Adversarial Examples:

- Question 5 :



we add a small vector of noise to fool the neural network. on the example above. We start with the image correctly classified as "quail". After adding noise, the neural network recognized the image as "stingray" while keeping a similar image, the right figures show the modified pixels.

- Question 6 :

In practice, the use of these neural networks is not so trustful, especially if the model is accessible, because a neural network can be fooled into believing that it is another class even if the image is well classified by a human eye.

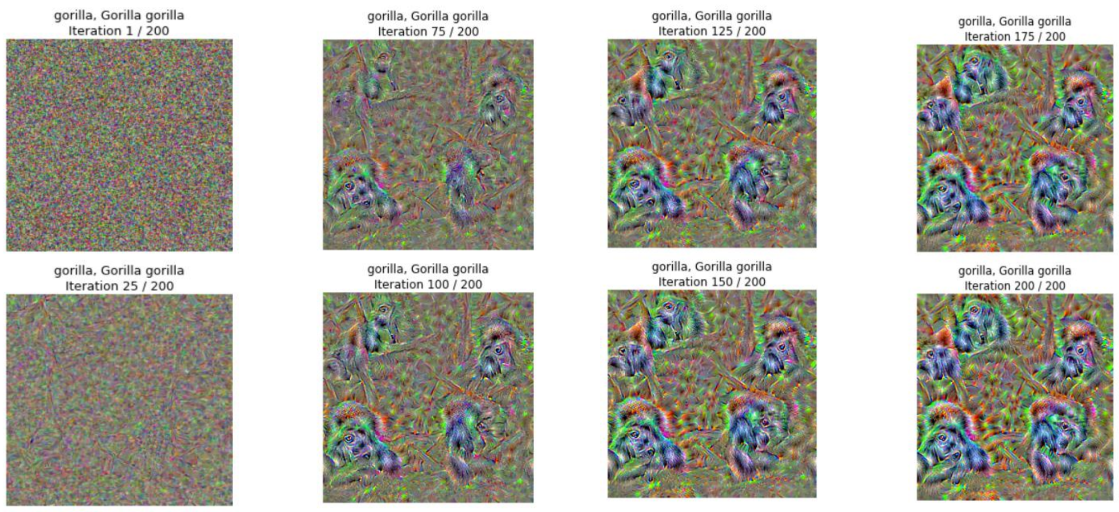
- Question 7 :

One of the limitations of this method is that the attack here depends on the learned parameters of the model and correctly classified images. One way to prevent it would be to not give access to them.

another alternative is described in this article,this technique is called ”black-box attacks”, the principle of this approach is to apply the ascending gradient to a familiar model, trained to have classifications similar to those of the target, and attack with the resulting image.

C. Class Visualization :

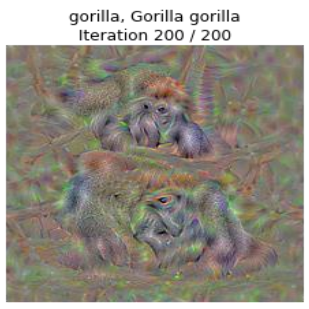
- Question 8:



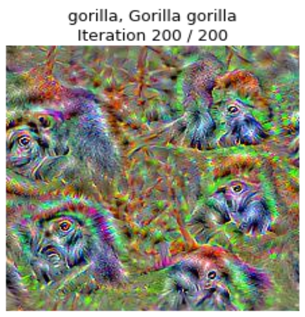
From an image containing only random noise, we try to compute an image maximizing the score of a class. In the examples above we try to maximize the score of the class Gorilla, we notice that we get gorilla patterns at the end.

- Question 9:

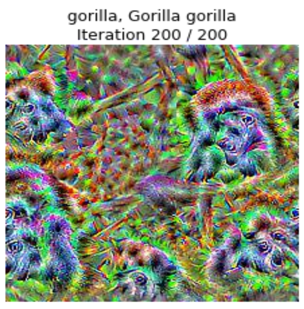
  we try to vary the number of iterations, learning rate and the regularization weight.



Learning rate=1

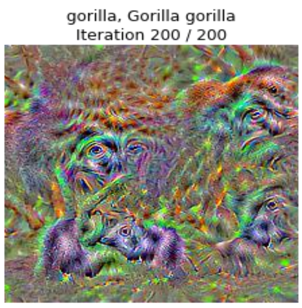


Learning rate=5

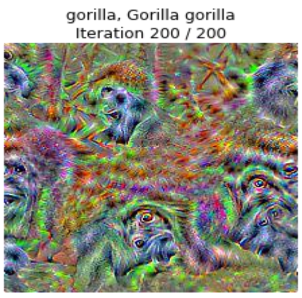


Learning rate=10

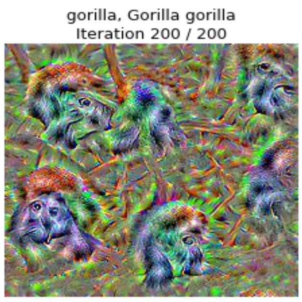
Decreasing the learning rate allows us to refine the image a little more, we can recognize better the image



regularization weight=0.1

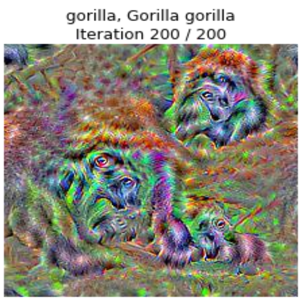


regularization weight=0.01

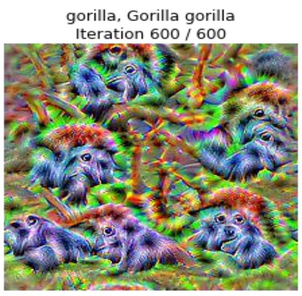


regularization weight=0.0001

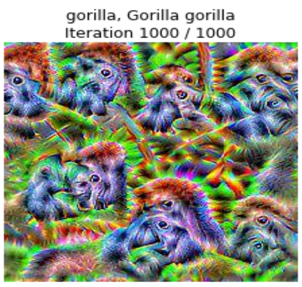
The variation of the regularization weight does not affect the obtained results much. However, we notice that for a very small weight the image contours are not modified, and the gorilla bodies are better delimited.



number of iterations=200



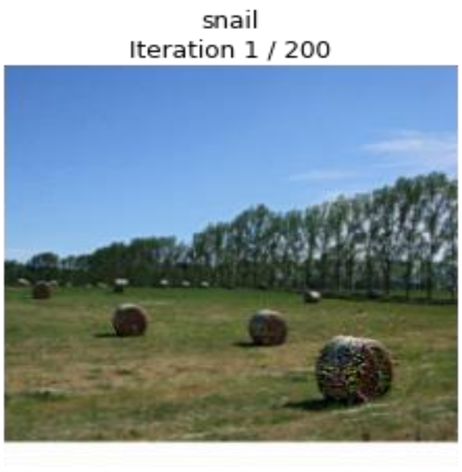
number of iterations=600



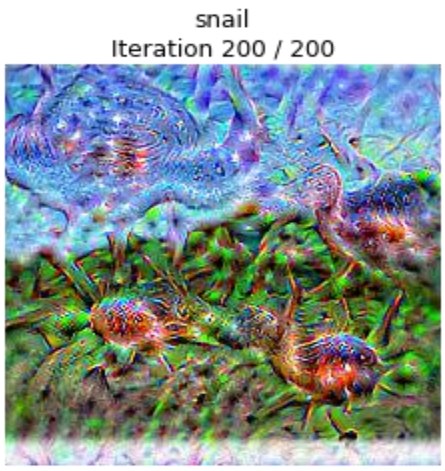
number of iterations=1000

We notice that for a very large number of iterations all the image is modified, and we notice more detail on the image than with a smaller number of iterations.

- Question 10:



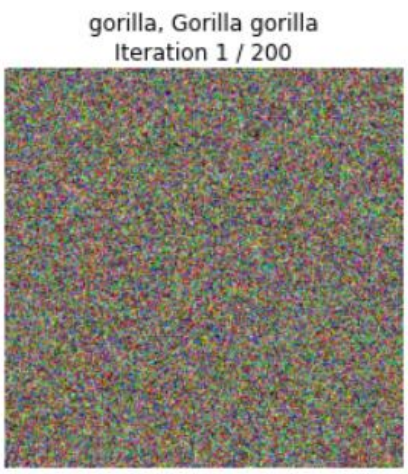
Input image

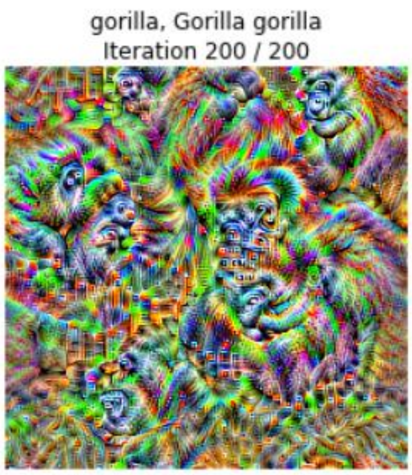


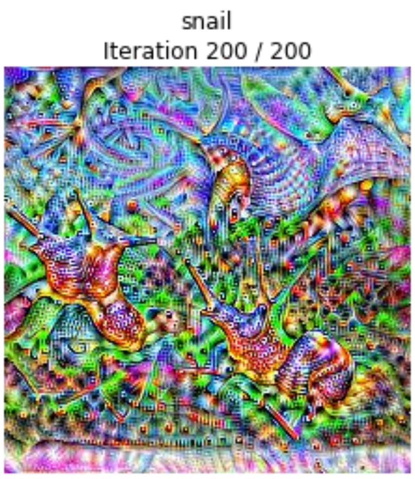
output image

We do the same treatments as for the previous example by replacing the starting image by an image from Image Net. We notice that the attention is focused on the snail objects since this is what allows to classify the image. Maximizing the image class score improves the quality of the results and makes it easy to recognize the class of the image.

- Question 11:







we use the same parameters used in the previous questions, we notice that with the vgg16 model we find quickly (number of iteration equal to 200) the details of the image (shape, contours), he structures the texture of the image, but at given time it makes the details of the image unreadable. also, we notice that he adds new colors that we don't find it with squeezenet

III. Domain Adaptation:

A. DANN and the GRL layer:

- Data:

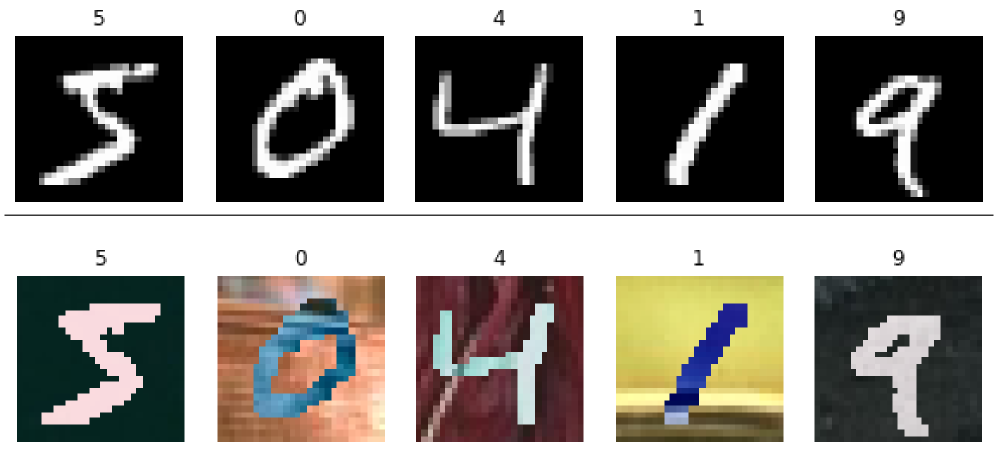
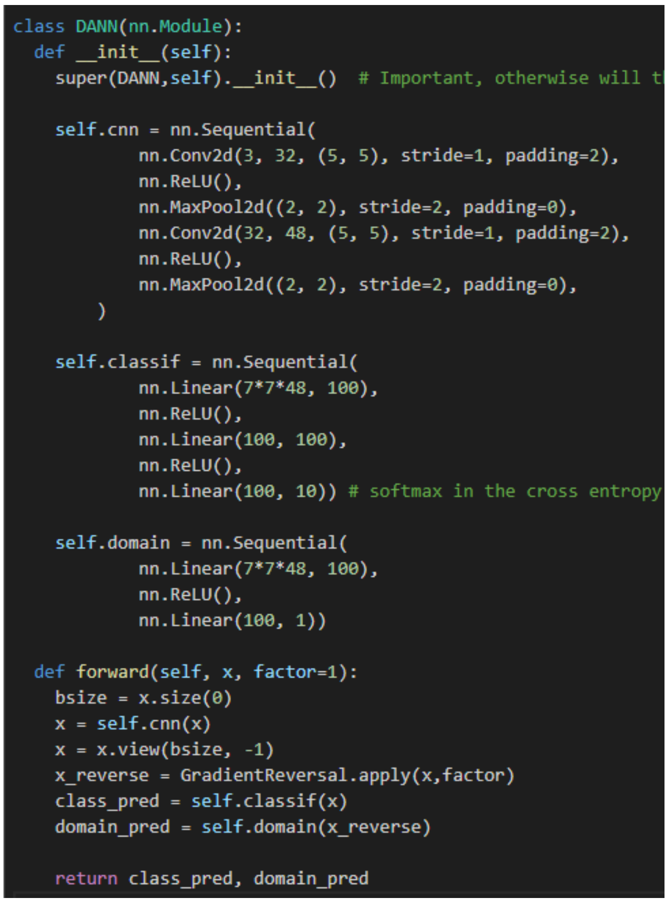


Figure 4 - MNIST vs MNIST-M Data

- Model:



```
class DANN(nn.Module):
  def __init__(self):
    super(DANN,self).__init__()  # Important, otherwise will t

    self.cnn = nn.Sequential(
            nn.Conv2d(3, 32, (5, 5), stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d((2, 2), stride=2, padding=0),
            nn.Conv2d(32, 48, (5, 5), stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d((2, 2), stride=2, padding=0),
        )

    self.classif = nn.Sequential(
            nn.Linear(7*7*48, 100),
            nn.ReLU(),
            nn.Linear(100, 100),
            nn.ReLU(),
            nn.Linear(100, 10)) # softmax in the cross entropy

    self.domain = nn.Sequential(
            nn.Linear(7*7*48, 100),
            nn.ReLU(),
            nn.Linear(100, 1))

  def forward(self, x, factor=1):
    bsize = x.size(0)
    x = self.cnn(x)
    x = x.view(bsize, -1)
    x_reverse = GradientReversal.apply(x,factor)
    class_pred = self.classif(x)
    domain_pred = self.domain(x_reverse)

    return class_pred, domain_pred
```

B. Practice:

- Question 1:

The model will lose the domain-agnostic property and will be able to discriminate both domains as shown on the figure below, we obtain perfect accuracy on the domain classifier, but we get the same bad results for the target data (mnist-m) 46.37% accuracy.

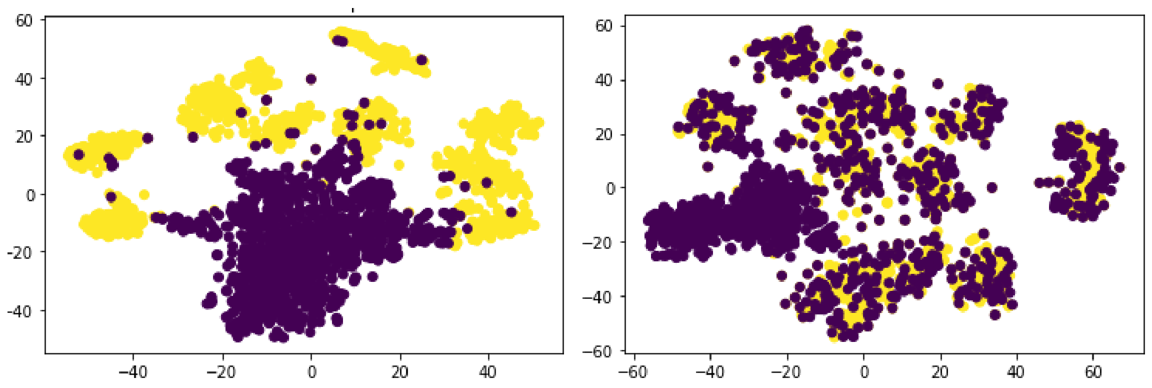


Figure 5 - TNSE embeddings without GRL vs with GRL (yellow for source and purple for target)

- Question 2:

The accuracy for the source data without domain adaptation is 99.02% vs 98.82% after the adaptation and this is because the model learns more general features of that makes him less biased by the source data and thus slightly less accurate on them, but it is worth it as long as we can improve the accuracy on the target data.

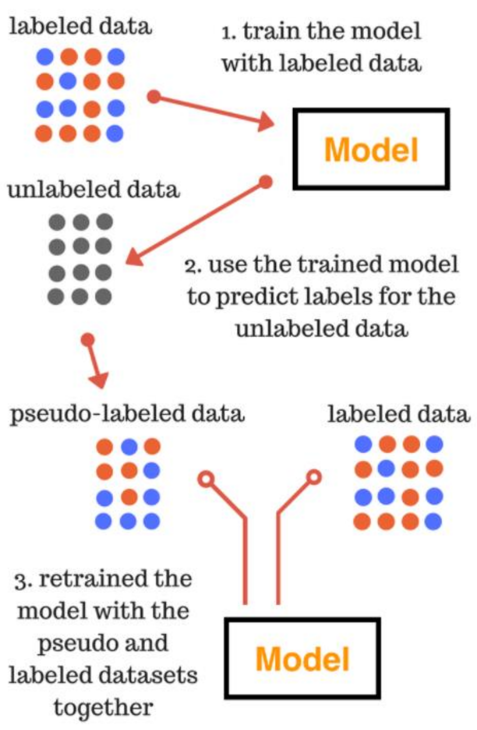
- Question 3:

The value of the factor is very important, first it must be negative, second if its value is very low the penalization will be weak and so the model will differentiate between the two domains as seen in question 1. On the other hand, if it is very high it may cause instability while training and mess up the classification accuracy.

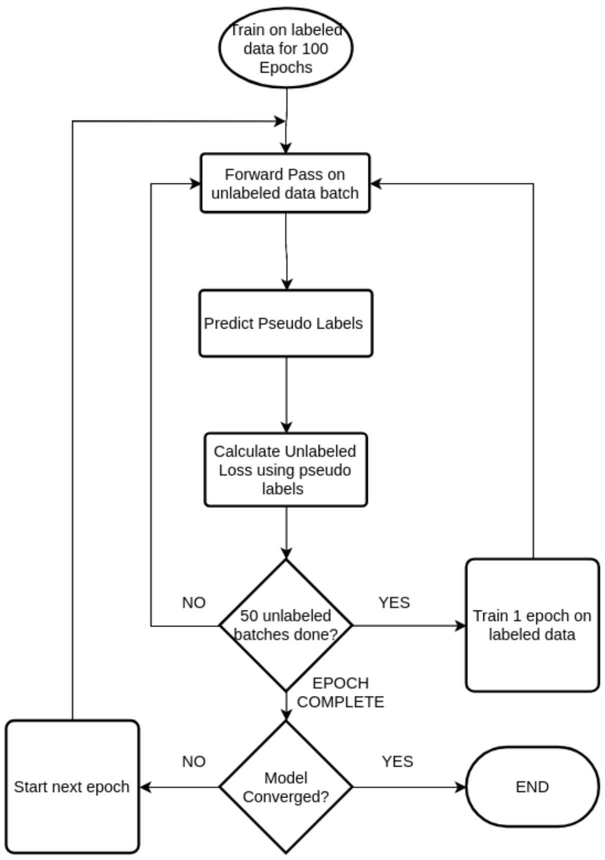
- Question 4:

Pseudo labelling is the process of adding confident predicted test data to the training of our model as follows:

- Build a model using source data
- Predict labels for target data
- Use confidence threshold to select confident predictions only
- Build new model using combined labelled source and target data


labeled data
1. train the model with labeled data
Model
unlabeled data
2. use the trained model to predict labels for the unlabeled data
pseudo-labeled data
labeled data
3. retrained the model with the pseudo and labeled datasets together
Model

We can also instead of using the combined data to build the new model to calculate the loss and add it to the original loss and backpropagate or starting from a pretrained model of the labelled data (after 100 epochs) we alternate between source and target data with 50 to one ratio for example as shown on the figure below.


Train on labeled data for 100 Epochs
Forward Pass on unlabeled data batch
Predict Pseudo Labels
Calculate Unlabeled Loss using pseudo labels
NO
50 unlabeled batches done?
YES
Train 1 epoch on labeled data
EPOCH COMPLETE
NO
Model Converged?
YES
Start next epoch
END

IV. Generative Adversarial Networks:

A. Generative Adversarial Networks:

- Question 1:

For equation (6) the generator G tries to maximize the probability that the discriminator D classify the generated image as a real image D(G(z)). As for equation (7) the discriminator have to first maximize the probability to distinguish between real and fake images from the dataset and second to minimize the probability to classify a generated image as a real image (thus maximize 1 – D(G(z)))

If we only use equation (6) the discriminator won't be able to learn and if we use equation (7) only the generator won't learn so we will get bad generated images. The game can't be played with only one player doing all the work.

- Question 2:

Ideally, the generator would transform the distribution P(z) to the distribution of the original data P(data) because we want the generated images to be as close as possible to real images

- Question 3:

The real equation should be:

$$\min_G E_{\mathrm{z} \sim \mathrm{P(z)}} \; [log\ (1 \; - \; D(G(z)))]$$

- Question 4:

Using default parameters, we obtain the following generated images:

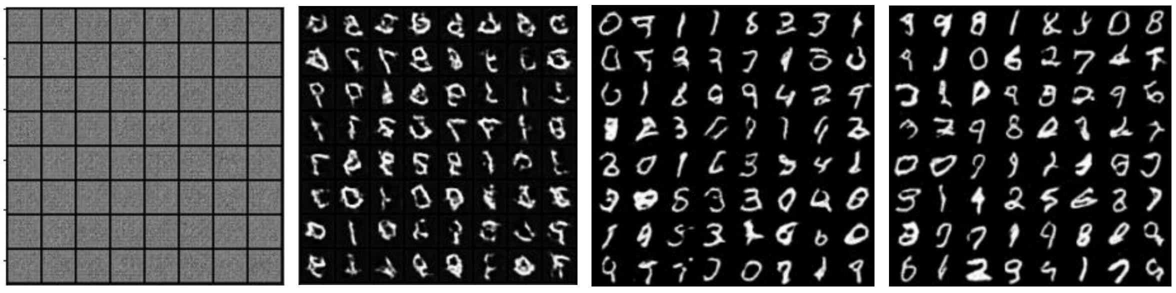


Figure 6 - Generated images on first, 400th, 1000th and 2000th epoch

We can see that from noise we can get some realistic results. The numbers start to form around the 400th iteration but looks a bit off. It gets better starting from the 1000th iteration and the numbers and get more and more realistic the further we train, and a variety of numbers are generated with different handwriting styles.

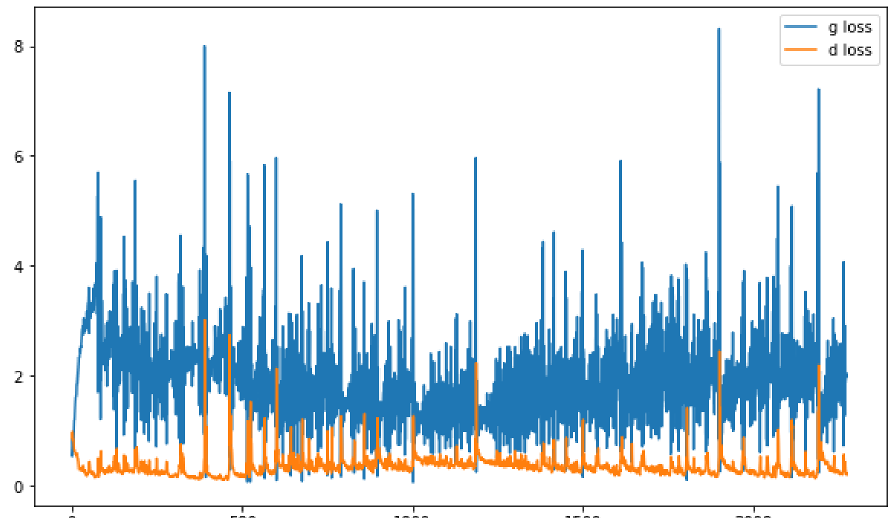
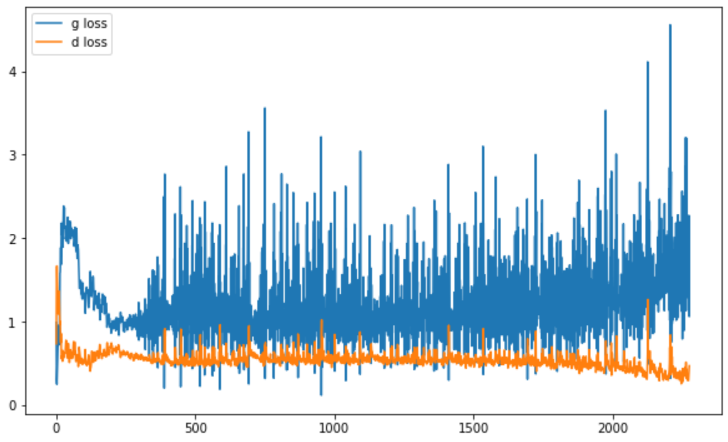


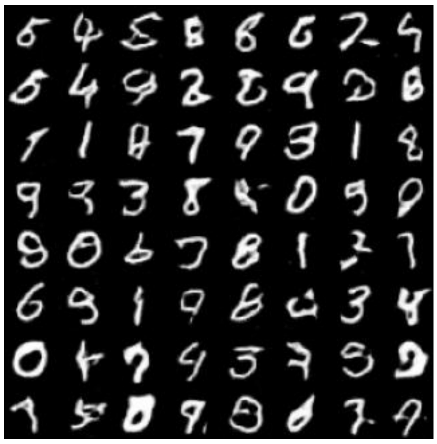
Figure 7 - Generator and Discriminator loss on each epoch

We notice that the discriminator loss decreases a lot at the first epochs and that’s because he finds it easy to separate real images from fake ones but the further, we train the harder it gets for him, and the loss slowly increase. As for the generator we can say the opposite and we can also notice that the loss in both cases is not stable which is logical because of the adversarial property of the GANs, the moment the generator learns how to fool the discriminator its loss gets lower and it’s also true the other way around.

- Question 5:

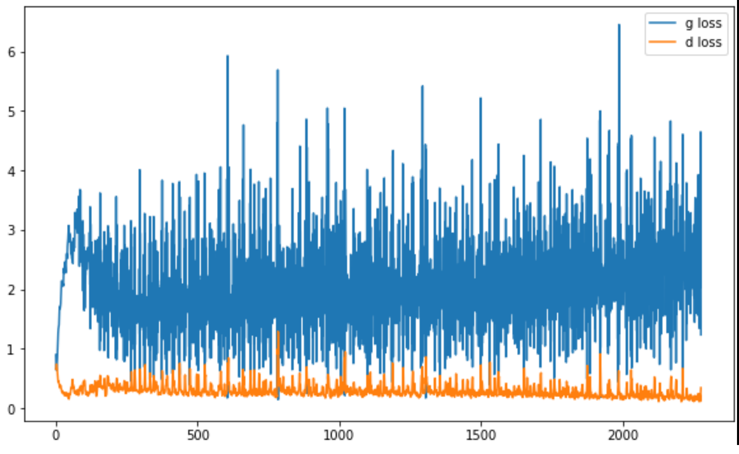
  - <u>Increasing ngf:</u> ngf = 128

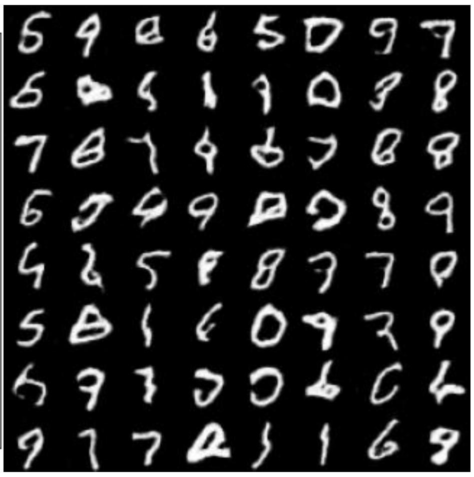




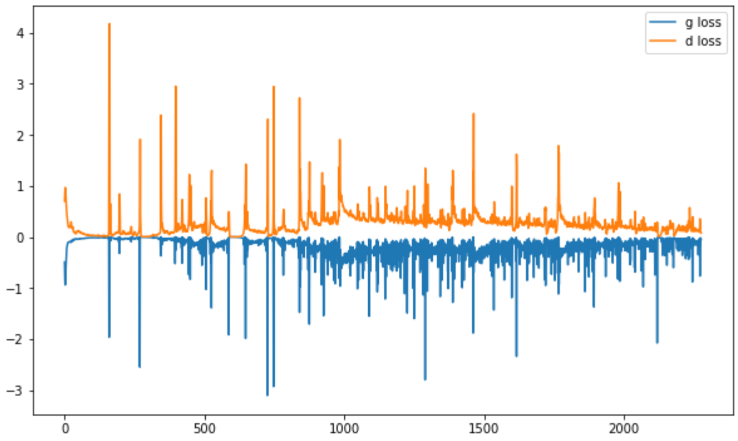
The generator loss is more instable and higher when we increased the ngf, as for the genrated images their quality is slightly worst. The most noticible change is the execution time that is longer with the increse of ngf.

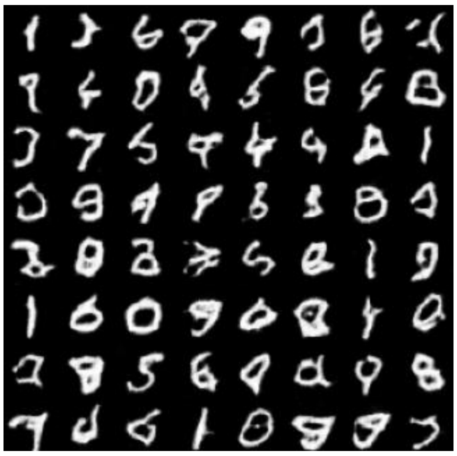
- Using default pytorch init:





- Using the true loss of generator:

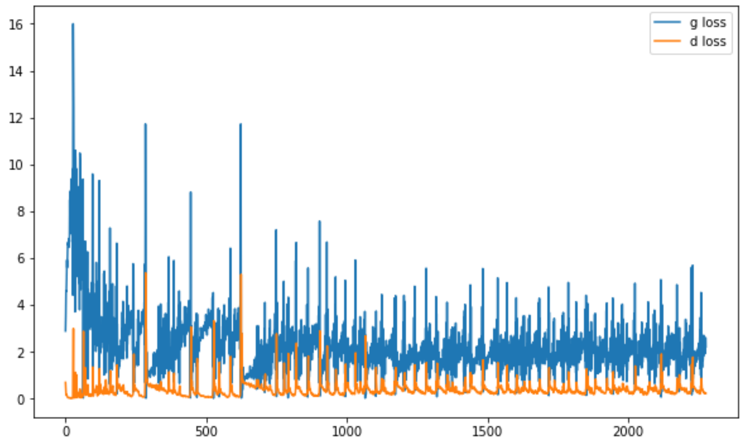


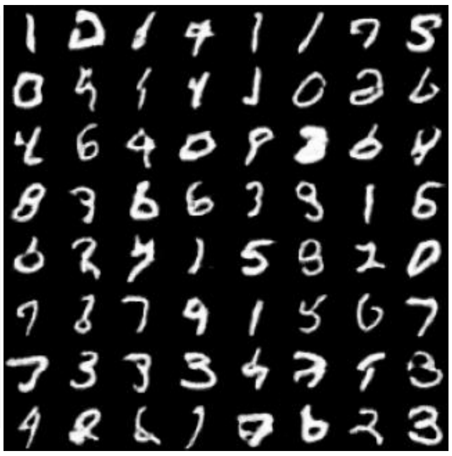


To use the true generator loss we have to go in an ascending way so we just take the negatif loss of 1 – D(G(z)) to do that, we can see that the generated images are worst than before and the loss is almost the mirror of the discriminator.

- Changing the learning rate:

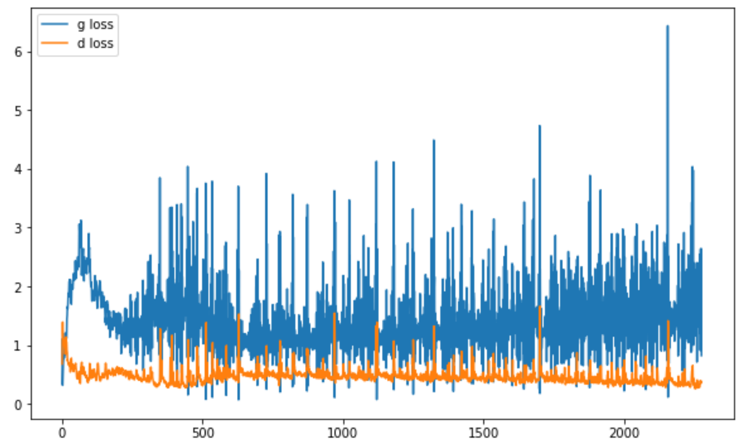
d_loss : 0.001 , g_loss : 0.0005

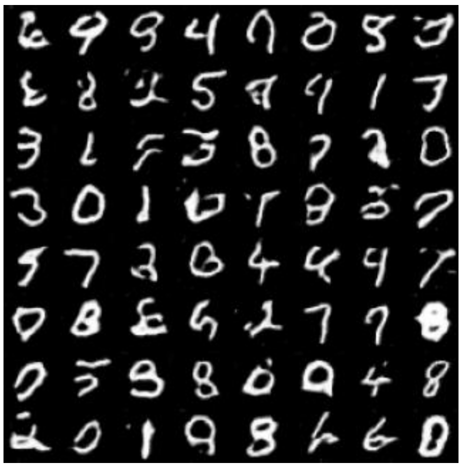




With these parameters the effects on the generator aren't very noticeable aside from the fact that the loss in the first iterations is higher than before, the discriminator has a loss that is more instable, but the generated images are more or less of a same quality as before.

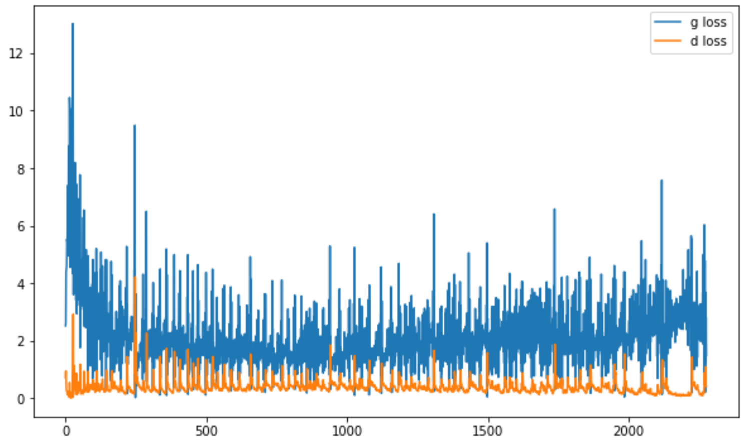
d_loss : 0.0002 , g_loss : 0.001

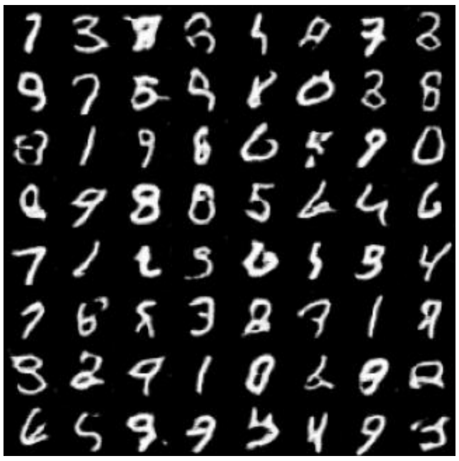




Using 0.001 as generator loss means that it converges faster and thus tries to fool the discriminator faster that isn't learning fast enough, so we think that the generated images are real, but they are not. This can be seen on the images, the numbers are not quite realistic.

d_loss : 0.001 , g_loss : 0.001

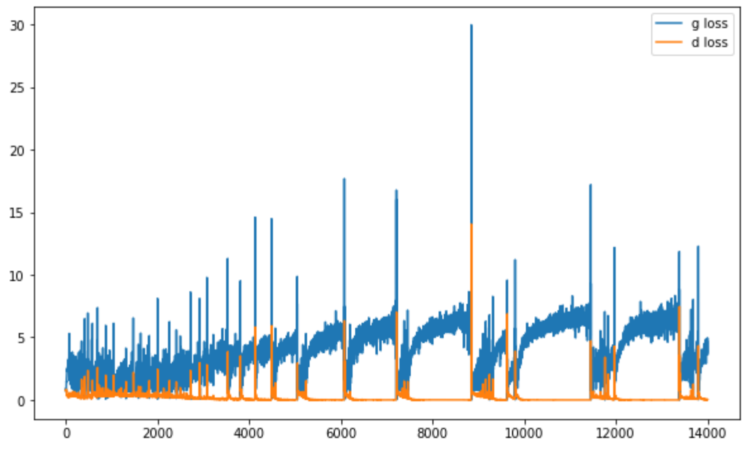




We can see that the convergence is more stable, but the generator is faster to converge than the discriminator, so the generated images are acceptable but still not as good as the default settings.

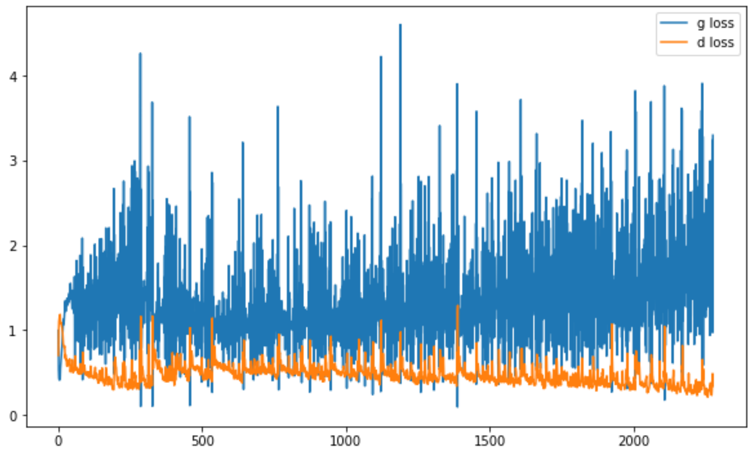
- Learning for longer: 30 epochs

Learning longer doesn't always mean better as we can see both g_loss and d_loss became very instable starting from the 4000th iteration, we can notice some big spikes and the generated images contains noise around the numbers.



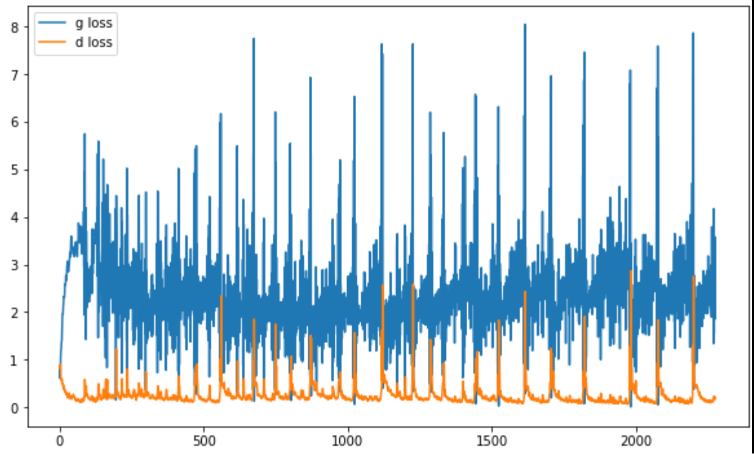
- <u>Changing the nz value:</u>

nz = 10



For nz = 10 the loss is similare than the default parameters if not more stable

nz = 1000



For nz = 1000 the loss reaches higher values and seems to be less stable.

B. Conditional Generative Adversarial Networks:

- Question 6 :

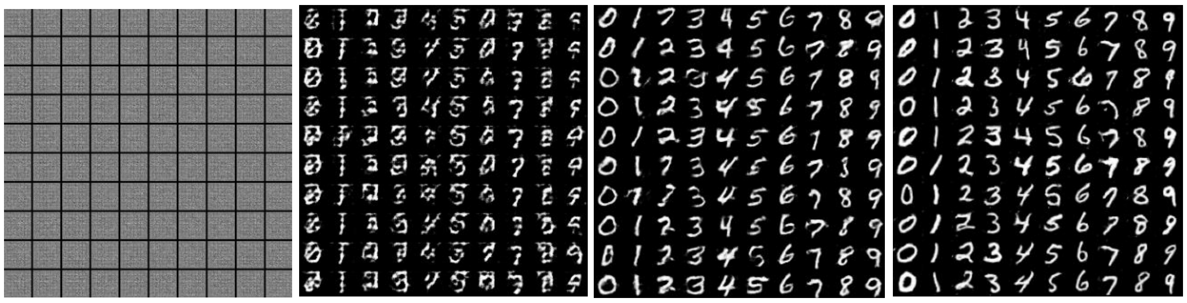


Figure 8 - Generated images with condition on first, 400th, 1000th and 2000th epoch

The generated images are acceptable starting from the 1000th iteration and almost perfect on the 2000th.

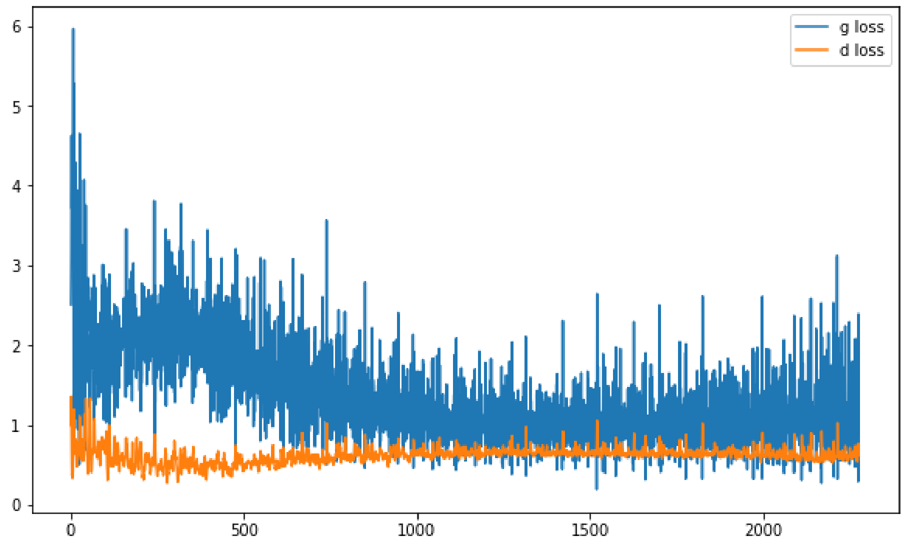


Figure 9 - Generator and Discriminator loss on each epoch

The loss is more stable than before especially for the generator and is much lower starting from 800-1000 epoch

- Question 7 :

No, the information is a must for the discriminator so that he will be able to tell if the generator is generating the correct image of that label y.

- Question 8 :

Using conditional GANs produces better results, and the loss of both the generator and discriminator is lower, but it takes more time to execute with 3 minutes per epoch for conditional GAN and less than one minute for normal GAN. The improvement is due to the label y where we are restricting the model to generate the correct number, so we get more precise images

- Question 9 :

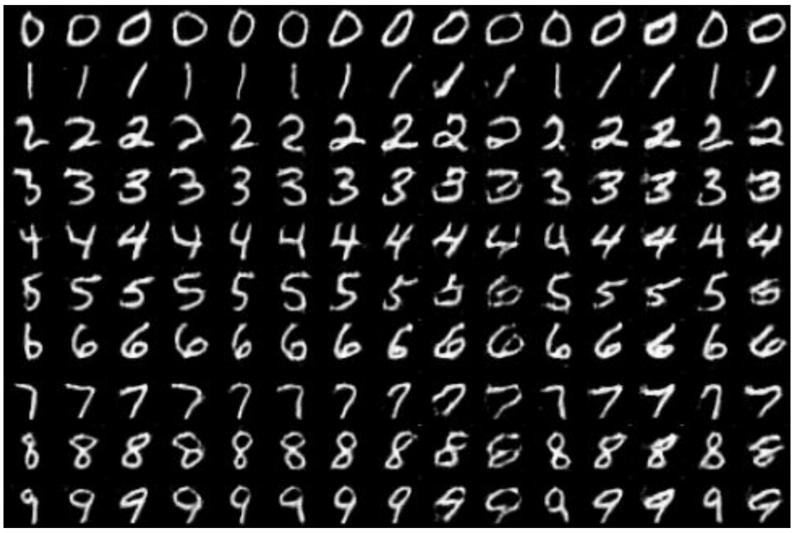


Figure 10 - generated images for 15 different noise starting points

We can notice that for each different noise we get the same number but in a different handwriting style so the z here can be considered as the style the numbers are writing in.