RDFIA - TME 3

Lucrezia Tosato & Marie Diez

Contents

1 Transfer Learning

1.1 Section 1 – VGG16 Architecture

1.1.1 Supervised dataset

1. Vgg16 have k=3, so we can compute the number of parameters as follow :

64*(3*3*3+1)	1792
64*(3*3*64+1)	36928
128*(3*3*64+1)	73856
128*(3*3*128+1)	147584
256*(3*3*128+1)	295168
256*(3*3*256+1)	590080
256*(3*3*256+1)	590080
512*(3*3*256+1)	1180160
512*(3*3*512+1)	2359808
512*(3*3*512+1)	2359808
512*(3*3*512+1)	2359808
512*(3*3*512+1)	2359808
512*(3*3*512+1)	2359808
4096*(1*1*7*7*512+1)	102764544
4096*(1*1*4096+1)	16781312
1000*(1*1*4096+1)	4097000
	138357544

Figure 1: Number of parameters

We can check the result with :

```
summary(vgg16, (3, 224, 224), device="cpu")
```

We obtain the same result : 138,357,544

2. It is of size 1×1×1000, which corresponds to a vector on which each position can be understood as the probability that that input image is in one of the 1000 ImageNet classes.

3. Applying the network on several images of our choice we notice that the network works properly, in the Fig.3 we got as a result "Egyptian cat" with 10.28565%, in the Fig.2, we got "alp " with 14.491319% probability, in Fig.4 we got "Sealyham terrier" 11.342503% in Fig.5 we got "Three-toed sloth, ai, Bradypustridactylus" 19.192665% probability.

Figure 2: Alps

Figure 3: Egyptian cat

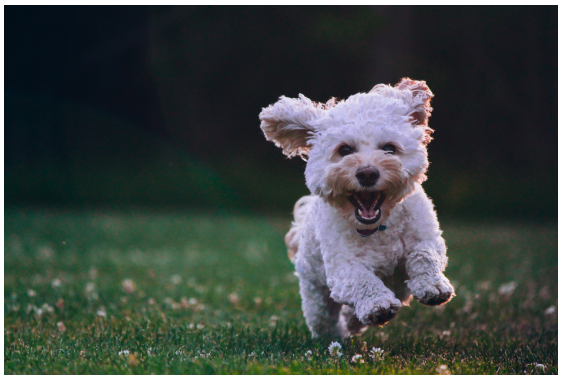


Figure 4: Sealyham terrier

Figure 5: three-toed sloth, ai, Bradypus tridactylus

4. Visualizing several activation maps obtained after the first convolutional layer we can interpret the result as the zones where our network is focusing in the fist layer, we know that the first layers learn more generic info than the last ones, in conclusion the activation map in the first layer points out the more general information that the network will learn about a specific image.

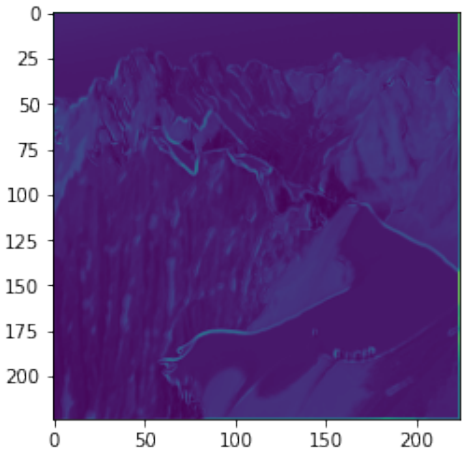


Figure 6: Alps

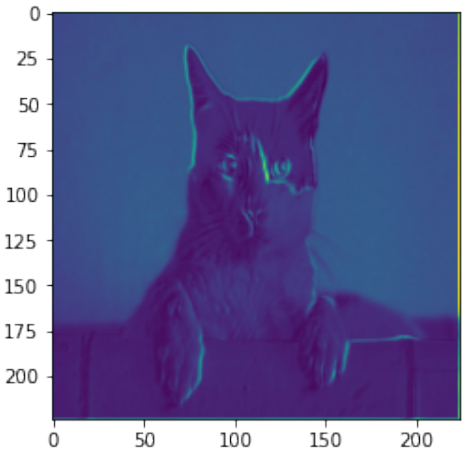


Figure 7: Egyptian cat

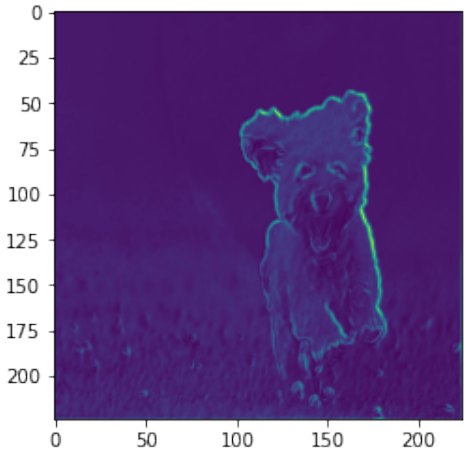


Figure 8: Sealyham terrier

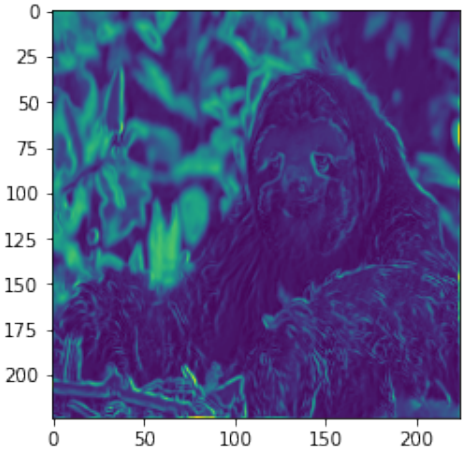


Figure 9: three-toed sloth, ai, Bradypus tridactylus

If we check the activation map at the end of the network we will obtain the region where the network is focusing for less generic informations :

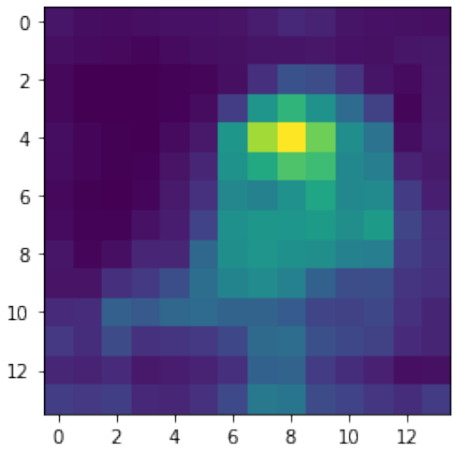


Figure 10: activation map for hree-toed sloth, ai, Bradypustridactylus towards the end of the network

1.2 Section 2 - Transfer Learning with VGG16 on 15 Scene

1.2.1 Approach

5. Because of the risk of overfitting. VGG16 is a rather complex network that was made to be trained on ImageNet, which has more than 14 million images. If it is trained directly on 15 Scene, which has a much smaller number of images, we would probably be overfitting for the task at hand.

6. With prior learning on ImageNet, we can teach the network to identify relevant features to differentiate real-world images, at different levels of abstraction. Then, we can exploit this ability to generate useful features by taking some of the first trained layers of VGG16, responsible for this feature extraction, and using their output as feature vectors for our classification task with 15 Scenes.

7. For the extracted features to make sense, the picture domain on which the original network was learnt must be similar to the domain of the network on which we are using this transferred learning. It implies that we must encounter difficulties while transitioning from a real-world images domain to a game-like images domain or an abstract-art paintings domain, for example.
Furthermore, the tasks must be related in some way so that the features collected in the first task are relevant in the second. In our example, both tasks are to do classification over real-world photos, which are sufficiently similar for us to use transfer learning.

1.2.2 Feature Extraction with VGG16

8. It determines the level of "abstraction" of the features extracted.

9. By stacking each black and white image from 15 Scenes on top of each other, we can go from 1 channel to 3 channels. In this approach, VGG16 will be able to accept them as input.

1.2.3 Apprentissage de classifieurs SVM

10. Yes. Instead of learning an independent SVM classifier we could just stack over the extracted layers of VGG16 one fully connected layer, which would have 15 neurons and be followed by a Softmax operation, for example. This way we would have as final output a tensor representing a vector of probabilities for each one of the 15 classes of our dataset.

1.2.4 Going further

11.

- Resize of image with transforms.Compose for cut at relu7 we have an accuracy of 0.888107
- Instead of resize we can use centerCrop for cut at relu7 we have an accuracy of 0.905863, we will keep centerCrop transform for the next.
- Change the layer at which the features are extracted. What is the importance of the depth of this layer? What is the representation size and what does this change?
  We can try to cut earlier in the network, for example instead of cutting at relu 7 we can cut 3 layers before, we obtain an accuracy of 0.918928. The more earlier we cut in the network the more generalized will be our features. If our data are lot different from imageNet it can be better to cut earlier, the more the data look like imageNet the later we can cut.
- Try other available pre-trained networks. What are the differences between these networks? We can try AlexNet with cut at 1 layers before the end, we obtain an accuracy of : 0.882412 We can try AlexNet with cut at 2 layers before the end, we obtain an accuracy of : 0.870352
- Tune the parameter C to improve performance For C=[1, 5, 10, 20] we have respectively for vgg16 network cut at 5 layers before the classifier : [0.91892797319933, 0.9195979899497487, 0.9199329983249581, 0.9185929648241206] We can see that the parameter C doesn't have an important influence on the accuracy.

- Instead of training an SVM, replace the last layer of VGG16 with a new fully-connected layer and continue to train the network on 15 Scene, we get better performance with the FC layer with 5 epoch we get accuracy above 90%.

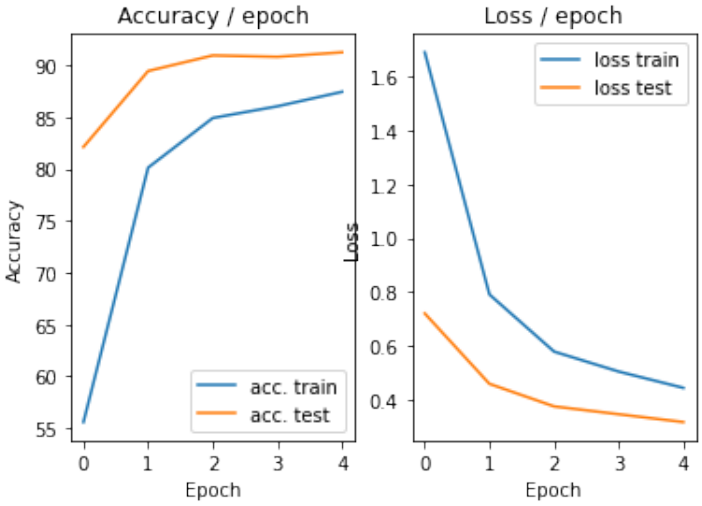


Figure 11: Performance with FC layer for classification

- Look into methods for dimensionality reduction before classification and their impact on performance and execution time. Wa can do a PCA for the dimensionality reduction. We chose to keep 90% of total variance, so we keep on 4096 dimension only 614.

2 Visualization

2.1 Saliency Map

1.

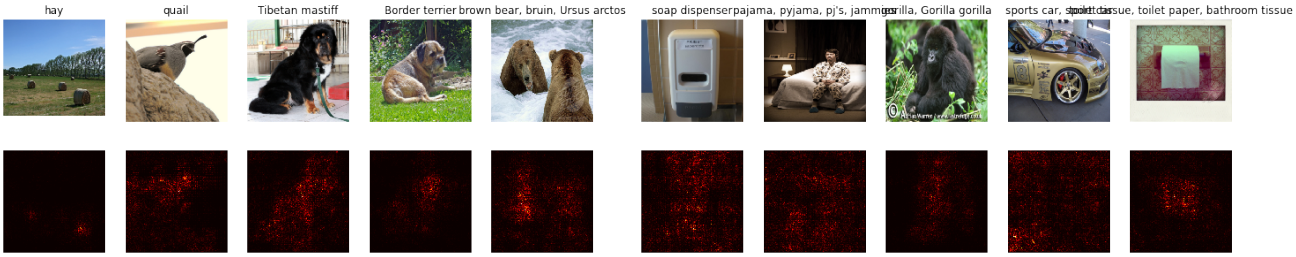


Figure 12

Figure 13

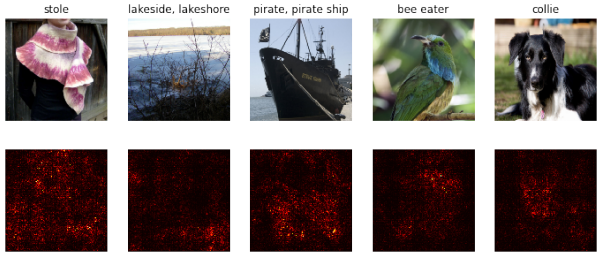


Figure 14

We can see on our results that the important zones of an image corresponding to the most impactful

pixels for predicting the correct class. For example we can see for the hay that the activations are mainly located on haystacks, as for the dog where we have activation mainly only in the shape of the dog, the rest of the image is not important to the classification. Moreover we can note that the activation seems more important for the head of the animal for the border terrier for example.

2. The limit that we can think of is that the network can lean a lot of bias in function of the context, for example for the pyjama example almost all the room is activated, because the network can learn the link between the context of the room and the pyjama.

3. This technique can be used to object localisation for example in an image in addition to interpreting the network. This activation can also give information about the more important localisation of object, in case of an animal it can give an approximation localisation of the face, this can be use as an initialisation for other work such as face detection.

4. With vgg16 we obtain :

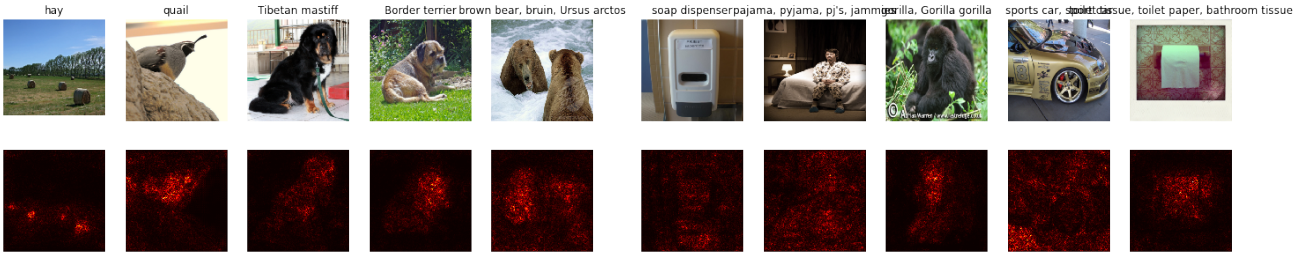


Figure 15

Figure 16

Figure 17

We can see that the result are a little different in some cases. In particular VGG16 is more precise on the detections, for example for the hay we can really see an improvement to located the haystacks. There is still the previous limit, for the pyjama the network still learn bias of the context but it seems a bit more performant and seems to learn less bias.

2.2 Adversarial Examples

5.

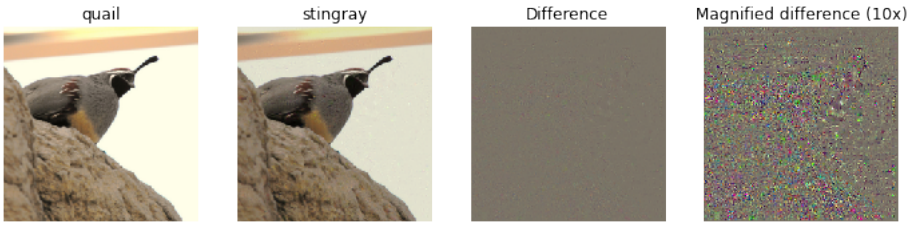


Figure 18: fool

Figure 19: fool

We can see that the network is fooled, after the transformation of the image following the TP guide, we can see that the network predict as we want for the first case a stingray and not more a quail and also in the second example where the network see a vase and not more a gorilla as we wanted. Moreover the 2 images look visually like the same, but as we can see on the magnified difference, our modification leads the network to predict our target class.

6. In real life if the image change just a little, this can lead to predict the wrong class. This can be dangerous for self driving cars, if the pixel of stop color change a little bit the network can badly interpret the class and therefore do a bad decision.

7. We can try to fool the network by creating adversarial image following the paper [1], we obtain :

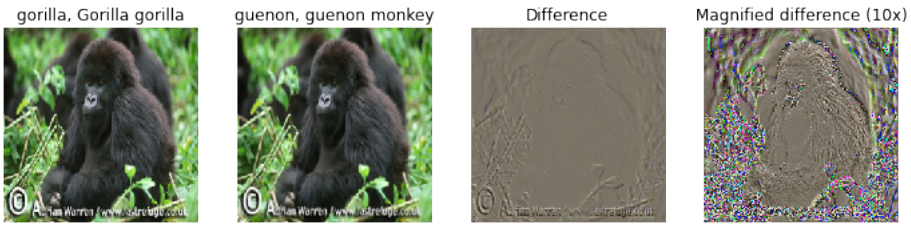


Figure 20: fool

The network is not really fool, he still see a kind of monkey, we didn't find why exactly.

2.3 Class Visualization

8.

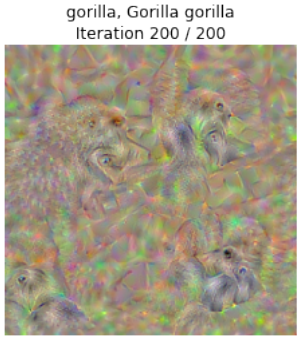


Figure 21: gorilla

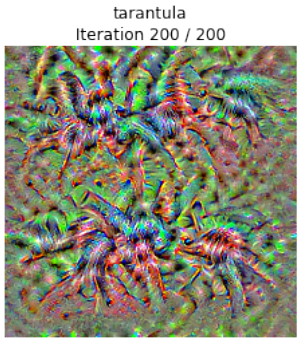


Figure 22: tarantula

In previous section we wanted to modifying the image as little as possible such that it becomes misclassified by the network visually similar as our image but sufficiently different to fool the network, now we want from a noisy image to modify this image such as to maximize the score for class i. To improve the quality of the results, we propose to use a certain number of regularization techniques. We can indeed see a still of gorilla in Figure 21 and tarantula in Figure 22.

9. After changing some parameters the results are expressed in the Figure below. With a smaller learning rate the image evolves more slowly and then we have a reduced richness of details and patterns when comparing Figure 23 with Figure 22. When increasing the regularization weight we can observe a reduced intensity of details as well, as shown by Figure 24, as we penalize more the increases on the L2 norm of our image. Finally, when only increasing the number of iterations from 200 to 400 we see even more intense patterns and details as represented by Figure 25, as we allowed more steps towards making the image more and more prone to activate our network for that given label ("tarantula" here).

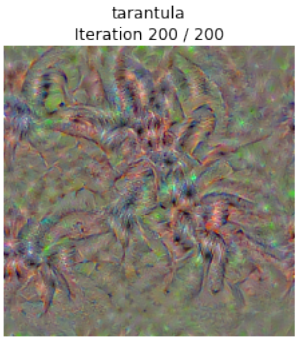


Figure 23: Changed the learning rate from 5 to 1

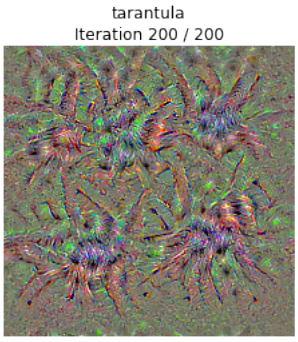


Figure 24: Changed the regularization weight from $10^{-}3$ to 1

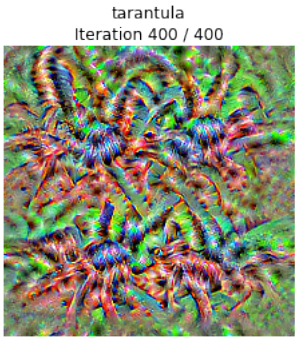


Figure 25: Changed the number of iterations from 200 to 400

10. Starting from an ImageNet image of the dog the result is shown with Figure 28. We can see that the process manages indeed to identify patterns in the image that are relevant for the classification and proceeds intensifying these patterns. The approach proceeds adding more other patterns across the image as well, near the initial relevant zones but not only, giving us a glimpse on how the initial image could be adapted to excite even more the network in hands to return the given label - one of the interests of this approach.
This method can also be used to make the network more certain about the choice of a given label, which can be useful in some cases (when dealing with thresholds for example), or to understand better what in that image is exciting more the network and what could be done to activate it even more, or less, depending on the use case.

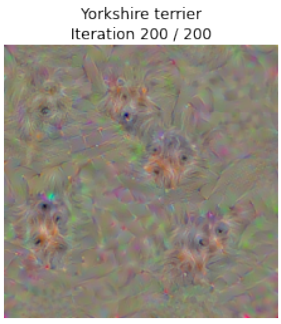
Figure 26: First iteration

Figure 27: Final iteration

Figure 28: Classes visualization - starting from an ImageNet image of a dog

11.

- squeezenet model Starting from :

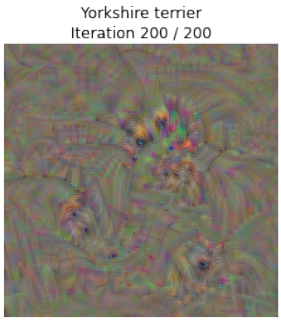


(a) Noisy, Final iteration

(b) ImageNet, Final iteration

Figure 29

- VGG model Starting from :

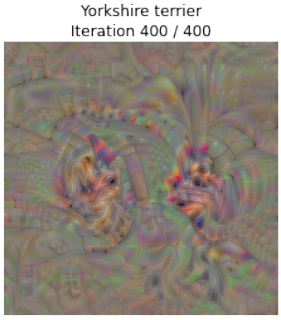


(a) Noisy, Final iteration

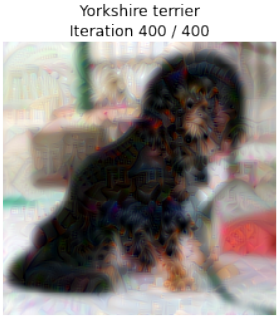
(b) ImageNet, Final iteration

Figure 30

We can notice that the computation time and the convergence is lot more longer than with the squeezenet. Indeed the SqueerNet network is designed to be very compact and efficient all while allowing good results. So with VGG-16 need to use more iterations, but the result is not that good :



(a) Final iteration



(b) Final iteration

Figure 31

3 Domain Adaptation

1. If we keep the network with the three parts (green, blue, pink) but not using the GRL, the features will not be domain independent. Since, we will keep decrease the loss, making the network dependent on the domain.

2. With domain adaptation we have :

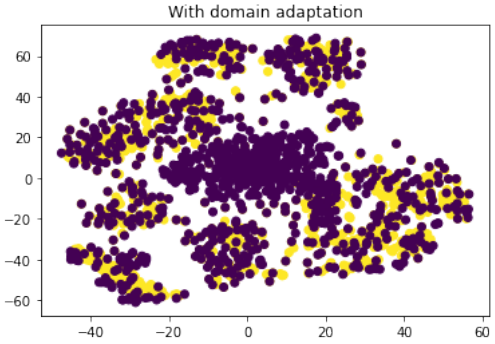


Figure 32: With domain adaptation, negative GRL

Without domain adaptation we have :

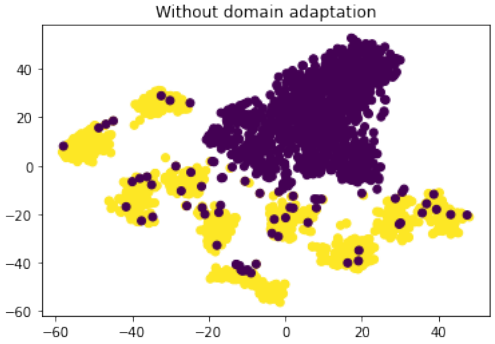


Figure 33: Without domain adaptation, negative GRL

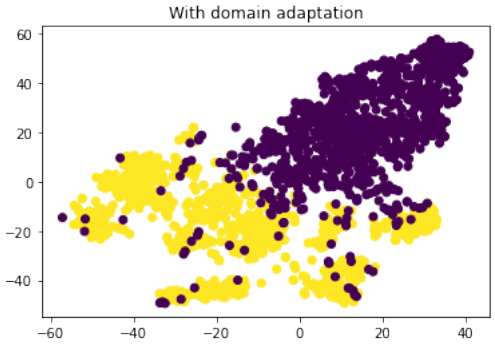


Figure 34: Without domain adaptation, positive GRL

We can see that the domain adaptation try to mix the data representation of both domain. Without the adaptation or with positive GRL the domain will be distinct as the Figure 33 and Figure 34. In

this case the domain have an importance, so for domain edge data can be misunderstood and therefore misclassified.

3.

The influence of the value of the negative number used to invert the gradient in the GRL is explained as follows. The domain classifier is connected to the feature extractor via a gradient inversion layer that multiplies the gradient by a certain negative constant used during backpropagation-based training. Otherwise, training proceeds in a standard way and minimises the loss of label prediction (for source examples) and the loss of domain classification (for all samples). Gradient inversion ensures that the feature distributions over the two domains are made similar (as indistinguishable as possible for the domain classifier), resulting in domain invariant features. As we can see below, the more the negative is high the more the domain will be mix because the more the model will be penalize to distinguish the domain.

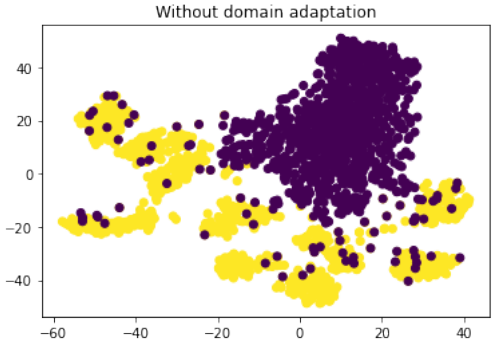


Figure 35: Without domain adaptation

For $\lambda = -1$ we have :

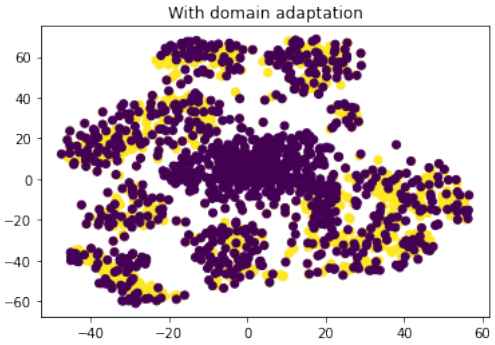


Figure 36: With domain adaptation

For $\lambda = -0.1$ we have :

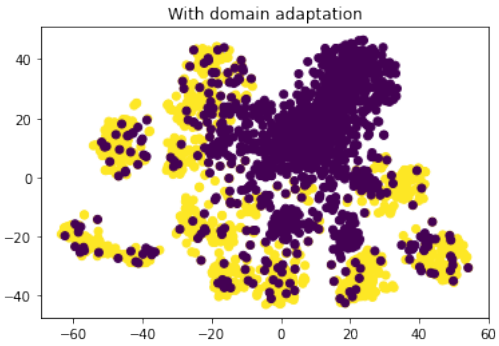


Figure 37: With domain adaptation

4. To increase a model's performance, the pseudo-labeling method employs a tiny collection of labeled data with a huge amount of unlabeled data. The technique itself is extremely simple, consisting of four basic phases that are repeated iteratively:

1. Train the model on a set of labeled data.
2. Apply the trained model to a batch of unlabeled data to predict labels.
3. Calculate the loss on unlabeled data using the predicted labels.
4. Back-propagate labeled loss with unlabeled loss.

In each batch, pseudo-labeling trains the network with both labeled and unlabeled data. This means that the training loop does the following for each batch of labeled and unlabeled data:

1. The labeled loss is calculated using a single forward pass on the labeled batch.
2. One forward pass on the unlabeled batch to forecast the unlabeled batch's "pseudo labels".
3. Calculate the unlabeled loss using this "pseudo label."

The overall loss function is as follows:

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

Or in simpler words the Loss per Batch is given by the labeled loss +weight*unlabeled loss.
The weight (alpha) parameter is used to limit the impact of unlabeled data to the total loss. Furthermore, the weight changes over time (epochs) and is gradually increased during training. When the classifier's performance is poor, the model can focus more on the labeled data at first. The weight grows as the model's performance improves over time (epochs), and the unlabeled loss has a greater impact on the total loss.

4 Generative Adversarial Networks

4.1 Generative Adversarial Networks

1. With equation 6 we are trying to get the best generator G that fools the discriminator D. That is so because maximizing the given expression means finding a G that gives a maximum D(G(z)), so, a value as close as possible of 1 (value for a real image).

With equation 7 we are now trying to find the best discriminator D that gives the right labels for each input (real or fake). That is so because in the first part of the expression we try to maximize the average probability of real image $\mathbb{E}_{\mathrm{x}\in\mathcal{D}\mathrm{ata}}\ [\log D\,(\mathrm{x})]$, which means get it as close as possible $\mathrm{D}(x^*)$ of 1 (value for real images). In the second part we maximize the average probability for fake image

$\mathbb{E}_{z\sim\in\mathcal{P}_{(\ddagger)}}$ [log(1 − D (G(z)))], minimizing then $D(G(z))$ to a value as close as possible of 0 (label for fake images).

Since the training of GANs is like a game of G against D, if we use only one of the equations we will be improving only one of them against the other. So, the cooperative learning shall not happen: if only G is improved it will eventually be able to always fool D with relatively simple images (that may not even be realistic since D is weak), in the opposite case D will improve and eventually always win against the static G separating well fake images from the real ones.

2. Ideally, the Generator G should be able to finally transform the the distribution P(z) in a distribution very close to $P(X)$ the one of a real image, capable of fooling the Discriminator D.

3. The true equation should be:

$$\min_{\text{G}} E_{z\sim P(z)}[log(1 - D(G(z)))]$$

Because where the generator seeks to maximize the probability of real images, instead of minimizing the probability of a fake image. This avoids generator saturation through a more stable weight update mechanism.

4. With the default settings in the training of the GAN we can see that the loss have a periodical trend (Fig. 38), they keep increasing and decreasing, this is due to the fact that both the Discriminator and the Generator try to improve, trying to do better than the other. As anticipated after 1000 iteration the GAN start producing results very realistic, from that moment on it keeps improving, but not that much. After 2344 iterations, we have quite good results, even if some numbers are clearly still not recognizable as number (Fig. 52 and 39).

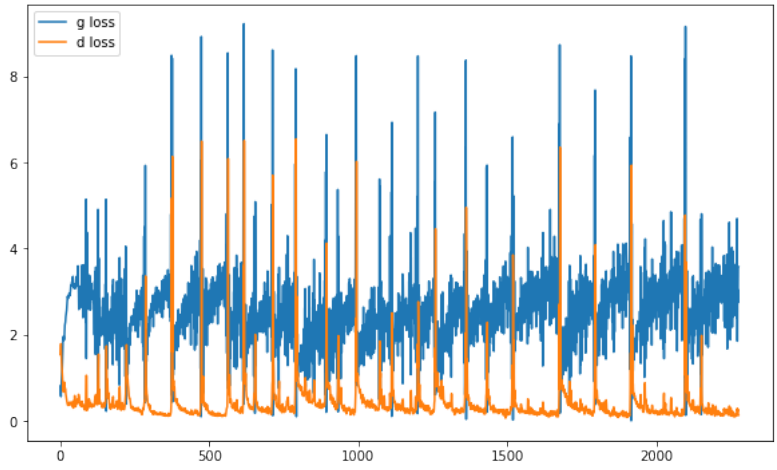


Figure 38: Losses

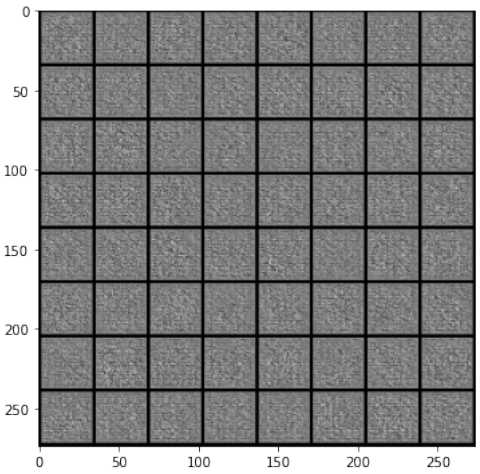


Figure 39: Starting iteration

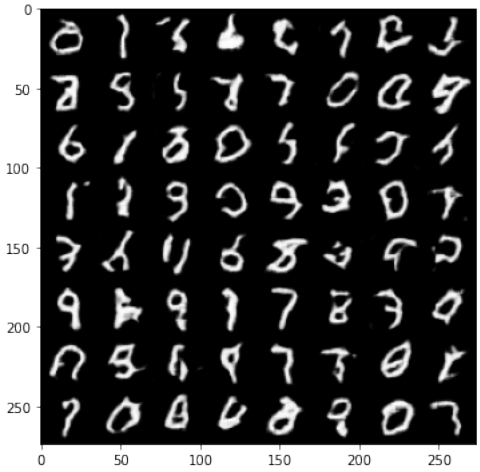


Figure 40: Result with default settings after 2344 iterations

5. Experiences with suggestions :

Decreasing nz to 10

Changing the number of dimensions of the latent space from 100 to 10 (setting the variable "nz" to 10 in the code).

we observe that we obtain bigger oscillations in the loss than with nz=100, and the final images (Fig. 41) seems to be little less realistic than the previous one, which makes sense since the initial input "z" from the latent space to the generator is less fine grained now, more gross, which translates as more struggle for the GAN to learn details.

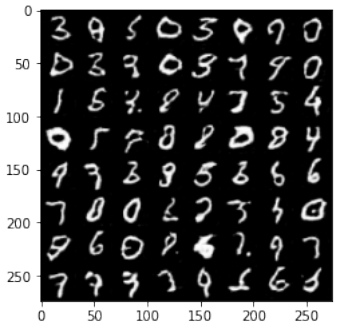


Figure 41: Result with nz=10 settings after 2344 iterations

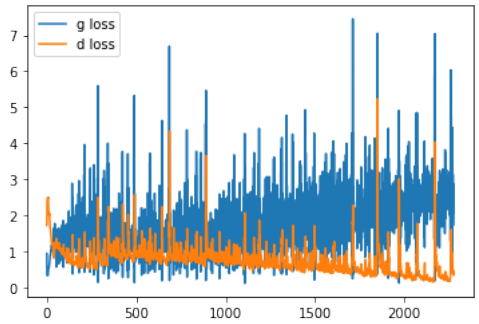


Figure 42: Trend loss result with nz=10 settings after 2344 iterations

Increasinz nz to 1000

Changing the number of dimensions of the latent space from 100 to 1000 (setting the variable "nz" to 1000 in the code).

we observe that the loss has less oscillation than with nz=10, and seems to be a little more stable than with nz=100. The final images (Fig. 43) are little more realistic than the previous one, which makes sense since the initial input "z" from the latent space to the generator is more fine grained now, more gross, which translates as more facility for the GAN to learn details.

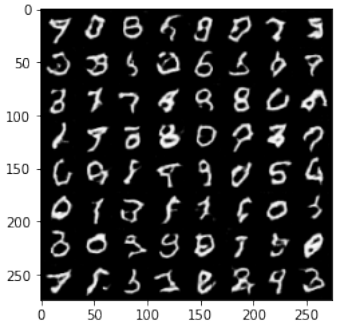


Figure 43: Result with nz=1000 settings after 2344 iterations

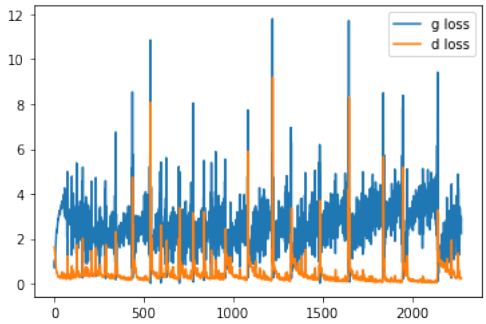


Figure 44: Trend loss result with nz=1000 settings after 2344 iterations

Increasing the epochs to 30

Changing the number of epochs from 5 to 30 (setting the variable "nb_epochs" to 30 in the code and the rest is default parameters) we observe the final trend (Fig. 46) and the final result (Fig. 45). We can observe that the game between discriminator and generator keeps going on, we seems to have a repeated pattern after our 5 epochs, it don't seems necessary to increase to much the number of epochs.

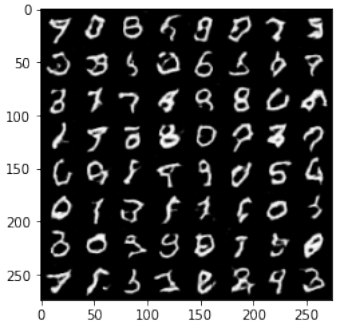


Figure 45: Final result with 30 epochs

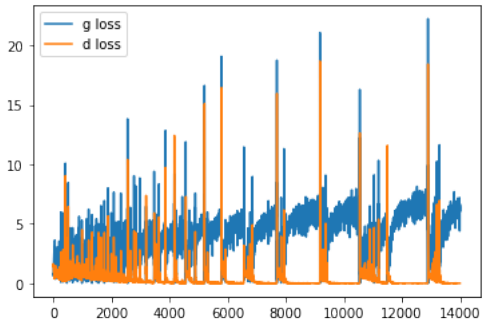


Figure 46: Trend loss with 30 epochs

Increase of the learning rate

Increasing strongly the learning rate provides very poors results, indeed with too high learning rate we jump over the "optimal gan" solution, without ever reaching it.

From lr_d = 0.0002, lr_g = 0.0005 to lr_d = 0.02, lr_g = 0.05 :

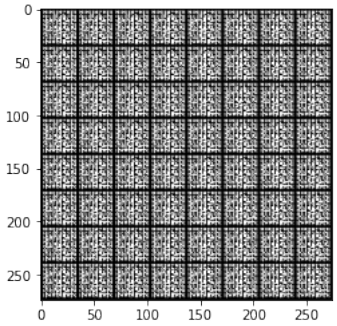


Figure 47: Final result with with increasing of the lr

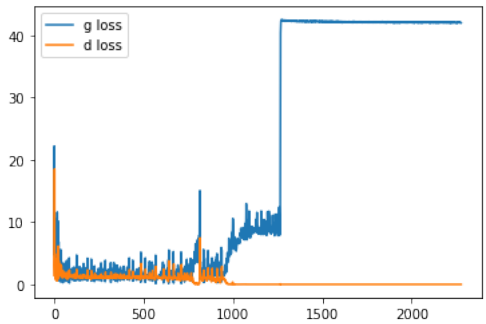


Figure 48: Trend loss with increasing of the lr

Decreasing of the learning rate
Decreasing strongly the learning rate provides very poors results, indeed with too low learning rate we never reach the "optimal gan" solution.

From lr_d = 0.0002, lr_g = 0.0005 to lr_d = 0.000002, lr_g = 0.000005 :

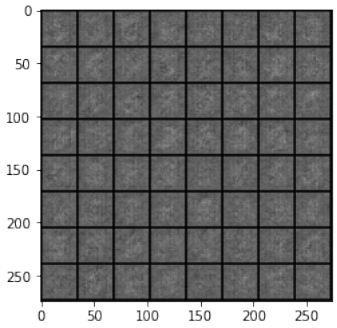


Figure 49: Final result with with decreasing of the lr

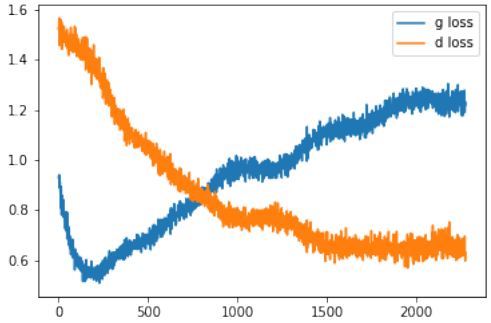


Figure 50: Trend loss with decreasing of the lr

Increasing significantly ngf to 256
Increasing significantly ngf provides less good results than with the default parameters (ngf=32), with bigger oscillations. Indeed with a high ngf we create mores features map and then a lot more parameters to learn, so it's more difficult for the network to learn and can increase the risk of overfitting.

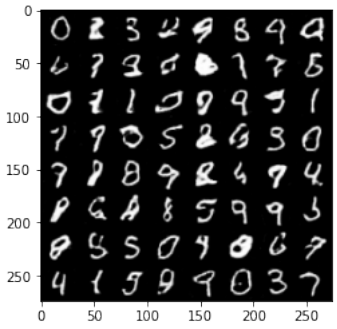


Figure 51: Final result with with increasing of ngf

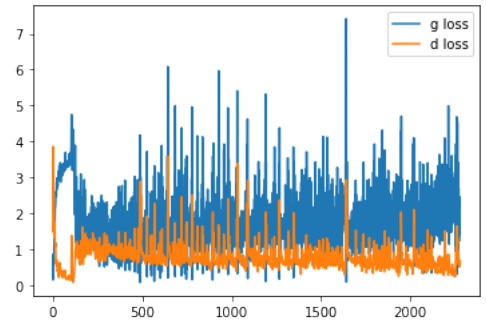


Figure 52: Trend loss with increasing of ngf

4.2 BONUS : Conditional Generative Adversarial Networks

6. With the cDCGAN during the training process it was possible to have a glimpse on how powerful deep convolutional GANs can be: by the 400th step we can already see the MINIST numbers and tell apart each one of them from the others. Also, it is important to point out the diversity of variations and shapes the network manages to generate (Fig. 53 and 54).

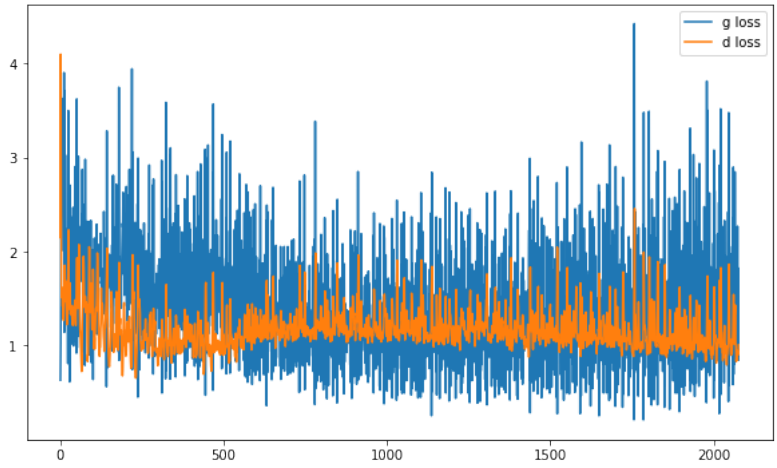


Figure 53: Trend with DCGAN

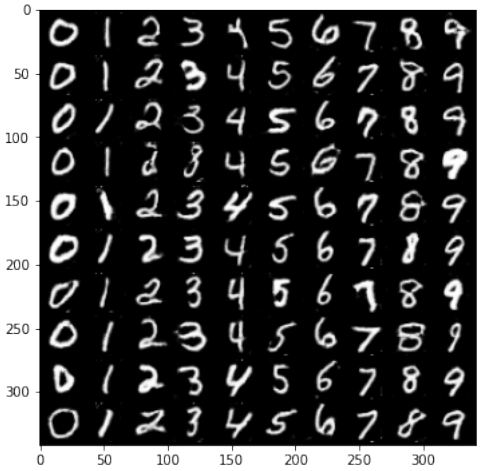


Figure 54: Final result with DCGAN

7. If we remove the vector y from the input of the discriminator we would have an error, since the "concatenation and classification" part of the discriminator's architecture expects to receive a tensor with a given number of channels , which would not happen with only the channels from x.

8. The training with the conditional GAN was more successful with respect to the unconditional GAN with default parameters, since with MNIST unconditional GAN we produce random digits, while with a conditional MNIST GAN we specify which digit the GAN should generate.
The training is indeed more stable than with unconditional gan, we have bigger oscillations but no more high pic, we viually can see a huge improvement of the generations.

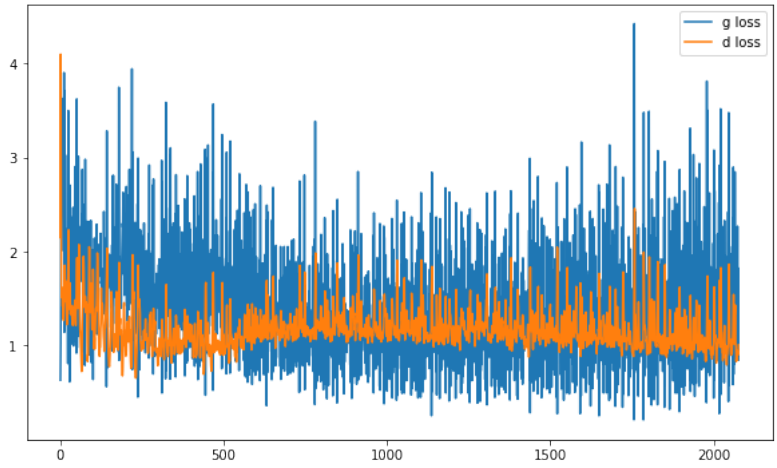


Figure 55: Trend with DCGAN

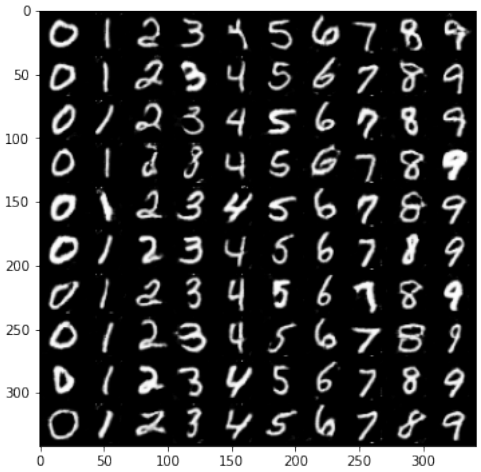


Figure 56: Final result with DCGAN

9. Z could be interpreted as "The background and style of the image must be contained within the generating code (the “noise”), so some they must have some numeric realization. But that does not mean that it is straightforward to find the relationship between noise and particular backgrounds or styles.

5 Bibliography

[1] R. Alaifari, G. S. Alberti, and T. Gauksson, “ADef: an Iterative Algorithm to Construct Adversarial Deformations,” arXiv:1804.07729 [cs, stat], Apr. 2018, Accessed: Dec. 08, 2021. [Online]. Available: http://arxiv.org/abs/1804.07729