RDFIA - TME3

Modern Elements of Computer Vision

ZHAO Yunfei

Andrew Caunes

January 16, 2022

In this homework we aim at programming and testing modern elements of computer vision including Transfer learning, Neural Network Visualisation, Domain Adaptation and Generative Adversarial Networks.

Contents

List of Figures

Chapter 1

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

1.1 VGG16 Architecture

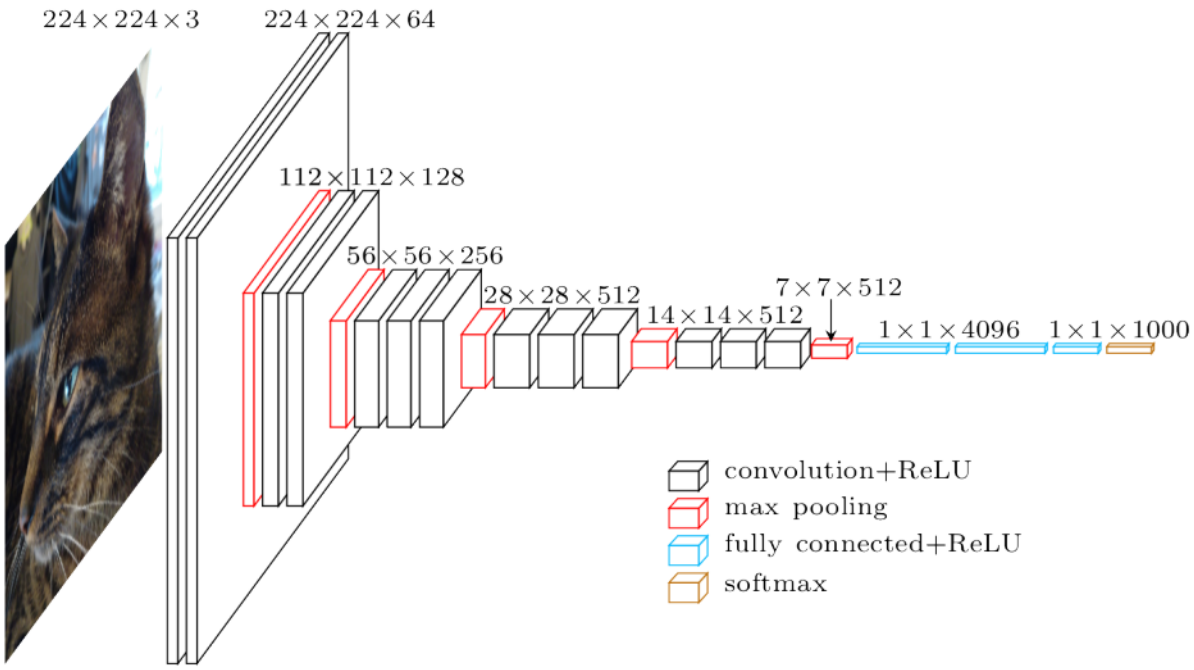


Figure 1.1: VGG16 Network.

Question 1.1: ★ Knowing that the fully-connected layers account for the majority of the parameters in a model, give an estimate on the number of parameters of VGG16 (using the sizes given in Figure 1.1).

From the VGG16 [2] network structure 1.1 and VGG16 parameters table 4.8 we can calculate the number of parameters of VGG16, as we know that the fully-connected layers account for the majority of the parameters. The total number of parameters is more than

$$(25088 + 1) \times 4096 + (4096 + 1) \times 4096 + (4096 + 1) \times 1000 = 123642856$$

So it is at 10^8 and the 138,423,208 parameters.

Question 1.2: ★ What is the output size of the last layer of VGG16? What does it correspond to?

The last layer of VGG16 is a FCN with output size 1000 and it is corresponding to the likelihood that an input image is of the 1000 classes provided by ImageNet e.g. the first value of this tensor correspond to the likelihood that this image is of classes of index 1.

Question 1.3: Bonus : Apply the network on several images of your choice and comment on the results.

We take two images and put them directly in VGG. On Fig. 1.2, we can the two images which are a cat (left panel) and a photo of myself (right panel). The trained labeled the cat as "**Egyptian cat**" and my photo as a "**wig**", which is absurd. This show that whenever the data is far from the train set, the results quickly deteriorate, especially when the sample doesn't belong to any predefined class. As there are Dropouts in FCN layers, if the mode of CNN is not on **evaluation mode**, results can be random too.

(a) Image of a cat.

(b) Image of me.

Figure 1.2: Test VGG on some images

Question 1.4: Bonus : Visualize several activation maps obtained after the first convolutional layer. How can we interpret them?

We show the feature maps of layers after the first convolutional layer. The image we use here is a an image of a dog running on grass. The feature map from convolutional layer 0 and layer 28 is shown respectively in Fig. 1.3 and Fig. 1.4 where each little figure correspond to an output of a channel. Thus, as layer is going deeper, the channel number increases and feature map size decreases, the presentation is more difficult to understand. They are high level features which are extracted by convolutional layer from an image.

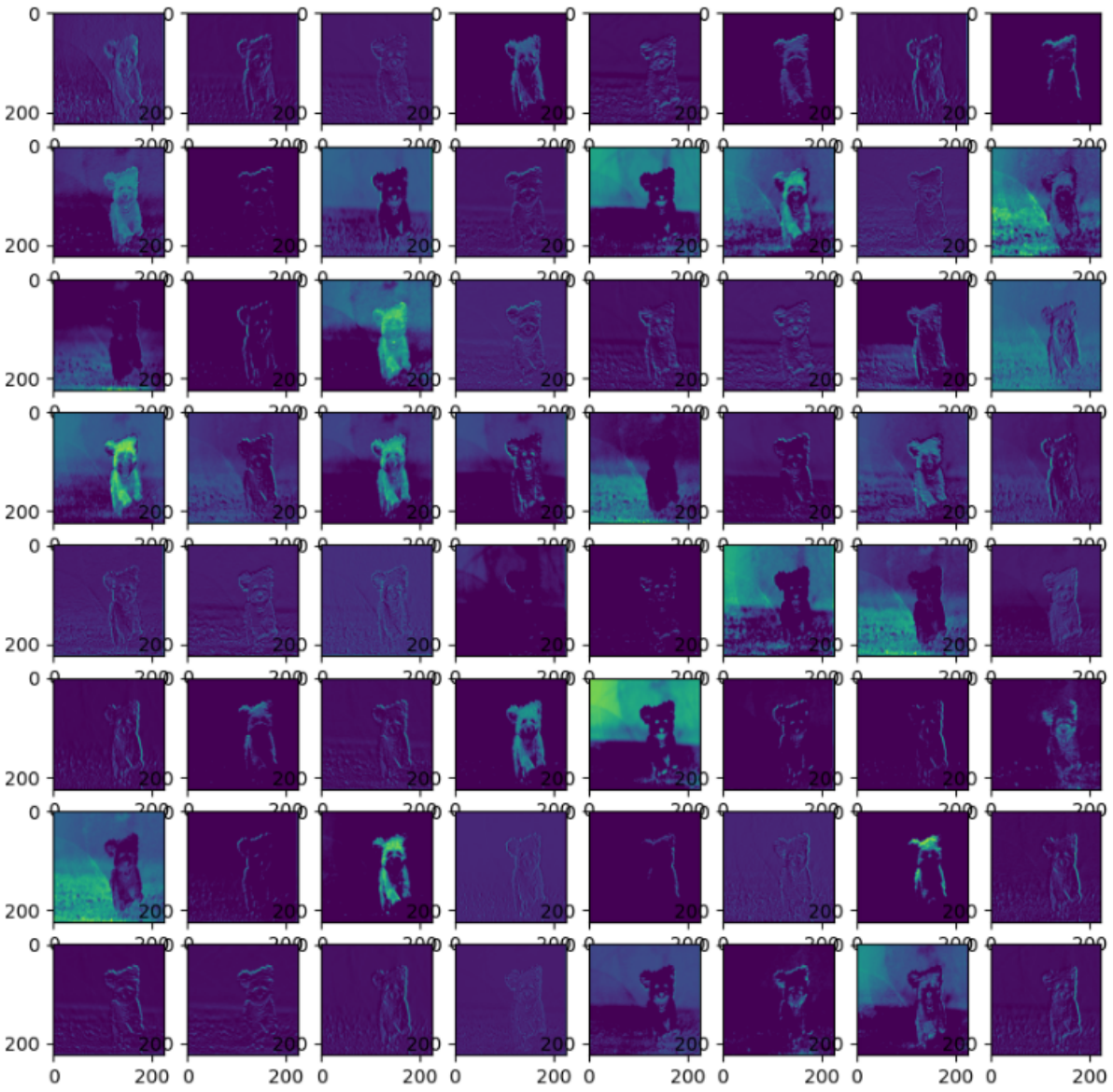


Figure 1.3: Feature map of VGG16 convolutional layer 0.

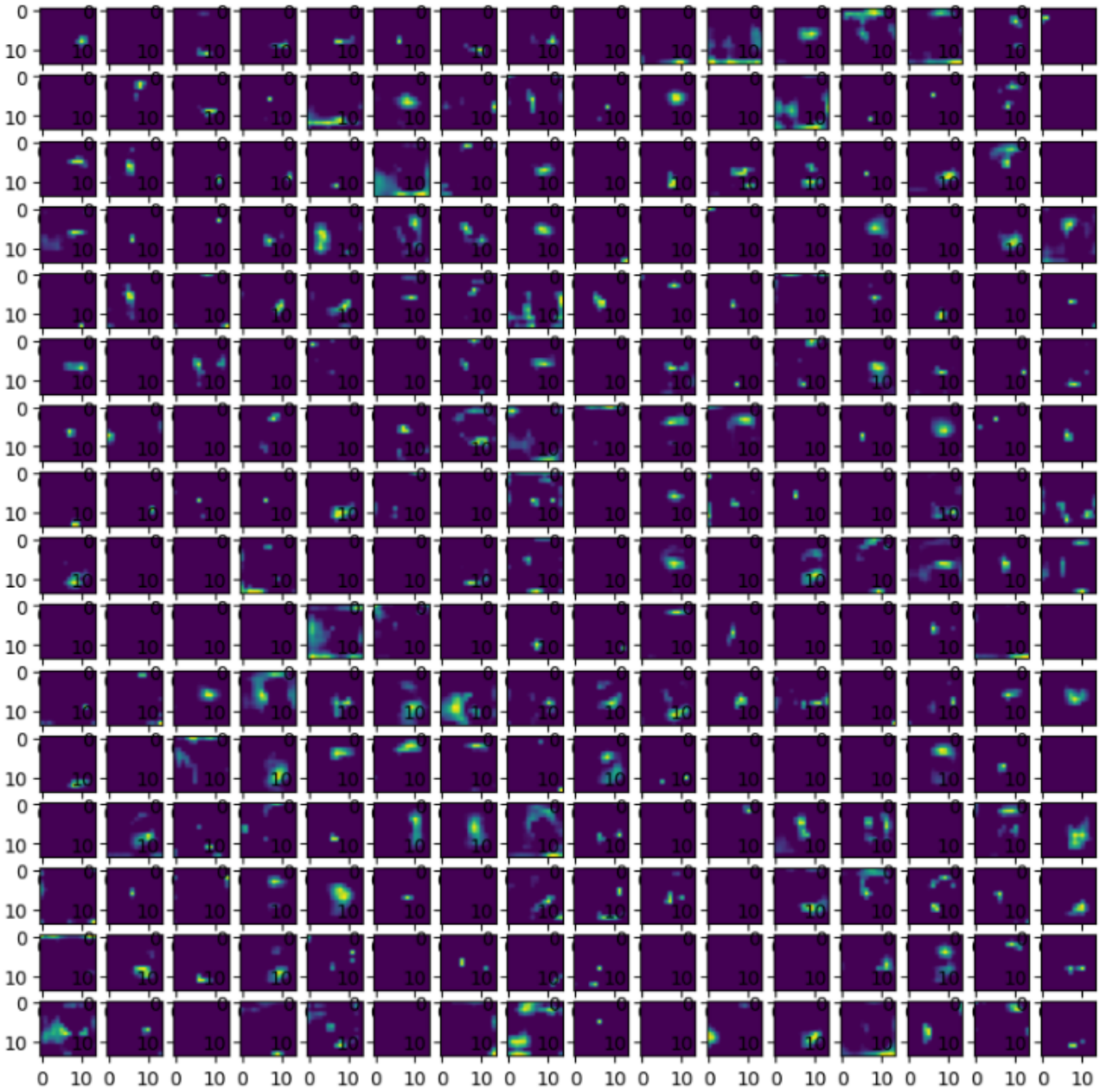


Figure 1.4: Feature map of VGG16 convolutional layer 28.

1.2 Transfer Learning with VGG16 on 15 Scene

Question 1.5: ★Why not directly train VGG16 on 15 Scene?

Firstly, we are in the case that of similar domain and different task of transfer learning. The class we have in 15 Scene is different of which in ImageNet and even the class number is far from which in ImageNet. Secondly, we do not have enough data to train a large model like VGG16 in 15 Scene which may lead too over-fitting. So we use convolutional to extract features and than train our classification layers.

Question 1.6: ★ How can pre-training on ImageNet help classification for 15 Scene?

CNN pre-training on ImageNet can extract features from an image in source data and as in our case source data and target data share the same domain, the features which should be extracted for classification are similar. So we can use the same convolutional layers to extract these features to do the classification for 15 Scene.

Question 1.7: What limits can you see with feature extraction?

First of all, if source data and target data are not in the same domain, features from CNN trained on source data may not adapted on target data. Then, the performance of model strongly depends on trained model. Further more, the format of image (channel number, data type, size, etc.) will also influence the feature extraction process.

Question 1.8: What is the impact of the layer at which the features are extracted?

Features from higher layer have more abstract representations and features from lower layer have more explainable features. For example in image processing, we tend to use features from lower layers which represent the features of corners, edges, etc of objects in an image, while in speech processing, we tend to use the last few layers of CNN which may represent the relation of worlds in a phrase.

Question 1.9: The images from 15 Scene are black and white, but VGG16 requires RGB images. How can we get around this problem?

We extend the one channel black and white image to a three channels image by duplicate the channel we have.

Question 1.10: Rather than training an independent classifier, is it possible to just use the neural network? Explain.

Yes, it is possible. We can just add a new small deep FCN on top of the origin CNN which is used to extract features, then we train the new CNN. But for this one, we may need more data.

Question 1.11: For every improvement that you test, explain your reasoning and comment on the obtained results.

• 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?
• Tune the parameter C to improve performance.
• Try other available pre-trained networks. What are the differences between these networks?
• 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 (with or without propagating the gradients to the rest of the network).
• Look into methods for dimensionality reduction before classification and their impact on performance and execution time.

• The shallower layer has more features and from the schema, we can see that for each

reduction of dimension, the feature map is reduced 2×2 times and channel numbers is doubled so the total features number is doubled. While the features are of lower level

• Using the features from the last second layer from the VGG16, the C parameters does not change significantly the result, while with very large C, SVM may fail to converge.

• We also tried RestNet. This CNN has nearly the same parameters number while it has deeper convolutional layer.

- Use VGG16 to extract 4096 dimensional vector + SVM:

  Train time:184.92653601500024

  Test time:0.10871751900049276

  Accuracy = 0.864992

- Use RestNet to extract 4096 dimensional vector + SVM:

  Train time:134.97819928599984s

  Test time:0.11067713699958404s

  Accuracy = 0.889447

• We change last layer of VGG16 to a FCN with output size 15 and we freeze the convolutional layer weights. With longer training time, we achieve better accuracy as show in figure **??**. But this method can easily lead to over-fitting and we need more data normally.

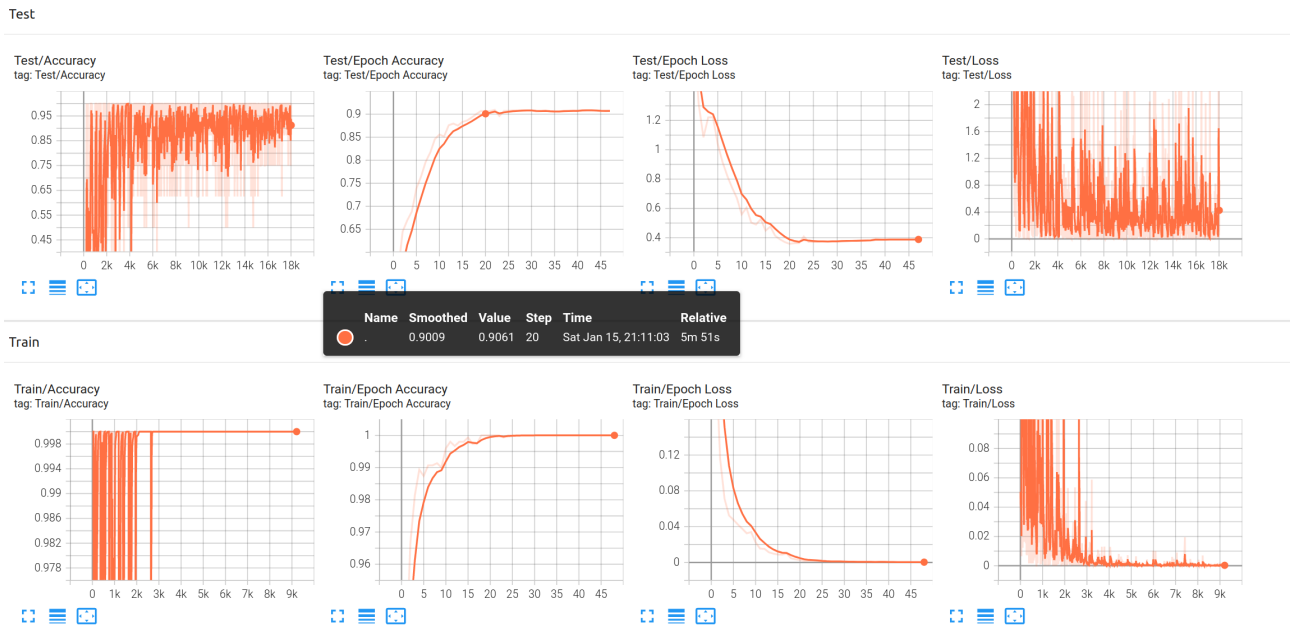


Figure 1.5: VGG with new FCN for classification layer

• Using method like PCA to reduce feature dimension before pass it to SVM can reduce training time significantly.

Chapter 2

3-b. Visualizing Neural Networks

In this practical, we will study the various techniques "recently" published in order to study the behavior of convolutional neural networks. The common principle of these approaches is to use the gradient of the input image with respect to an output class.

2.1 Saliency Map

Question 2.1: ★ Show and interpret the obtained results.

Saliency maps shows the impact of every pixel on the score of the correct class. Pixel with high gradient are lighted, that is to say, these points make bigger contribution to make the right decision. In the cases of daisy vase, black swan, the pixels on the object make more contribution to lead our network to make the right decision while for spatula, modem, the pixels from background also make some contributions.

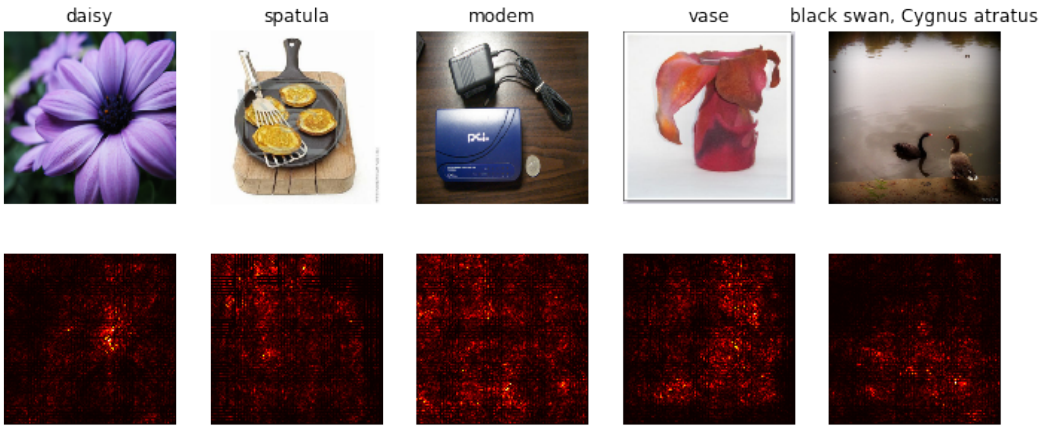


Figure 2.1: Saliency maps images from ImageNet validation set with SqueezeNet network

Question 2.2: Discuss the limits of this technique of visualizing the impact of different pixels.

The pixel attribution methods can be very fragile. Introducing small perturbations to an image, which still lead to the same prediction can lead to very different pixels being highlighted. This method only show a qualitative evaluation and it is difficult to know whether an explanation is correct. This method is also insensitive to model and data.

Question 2.3: Can this technique be used for a different purpose than interpreting the network?

We can also use the gradient to generate an image of a certain class or use the gradient to change the image to other classes.

Figure 2.2: Saliency maps images from ImageNet validation set with VGG16 network

Question 2.4: Bonus: Test with a different network, for example VGG16, and comment.

We can say that the Saliency maps from VGG16 on have highlighted points more concentrate on object in an image. But the difference is not huge and we conclude that this method of pixel analyse may not very sensitive to models.

2.2 Adversarial Examples

Question 2.5: ★ Show and interpret the obtained results.

The figure 2.3 is an example of adversarial attack by SqueezeNet with leaning rate 1 and 11 iterations. With only a little difference on original image, the fooling image successfully fools the neural network to classify an image of class quail to class stingray.

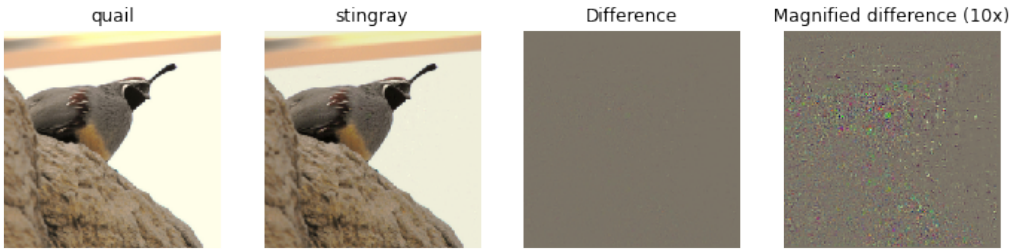


Figure 2.3: Fooling image generated by Adversarial Attack with SqueezeNet network.

Question 2.6: In practice, what consequences can this method have when using convolutional neural networks?

Most machine learning techniques were designed to work on specific problem sets in which the training and test data are generated from the same statistical distribution. This method shows that some kind of little noises on an image can change the distributions and lead to the mis-classification of a neural network. The data generate by adversarial attack may be arranged to exploit specific vulnerabilities and compromise the results.

Question 2.7: Bonus: Discuss the limits of this naive way to construct adversarial images. Can you propose some alternative or modified ways? (You can base these on recent research).

The method we use here is **Jacobian-based Saliency Map Attack (JSMA)**. It is computationally intensive as we need to do the back-propagation for several times. There are also **FastGradient Sign method (FGSM)**

2.3 Class Visualization

Question 2.8: ★ Show and interpret the obtained results.

Figure 2.4 is an example of four classes of class visualisation from a fake image generated from random signal with l2, jitter and 1d Gaussian filter regularization. From generated images, we can see the contour of object corresponding to the target class.

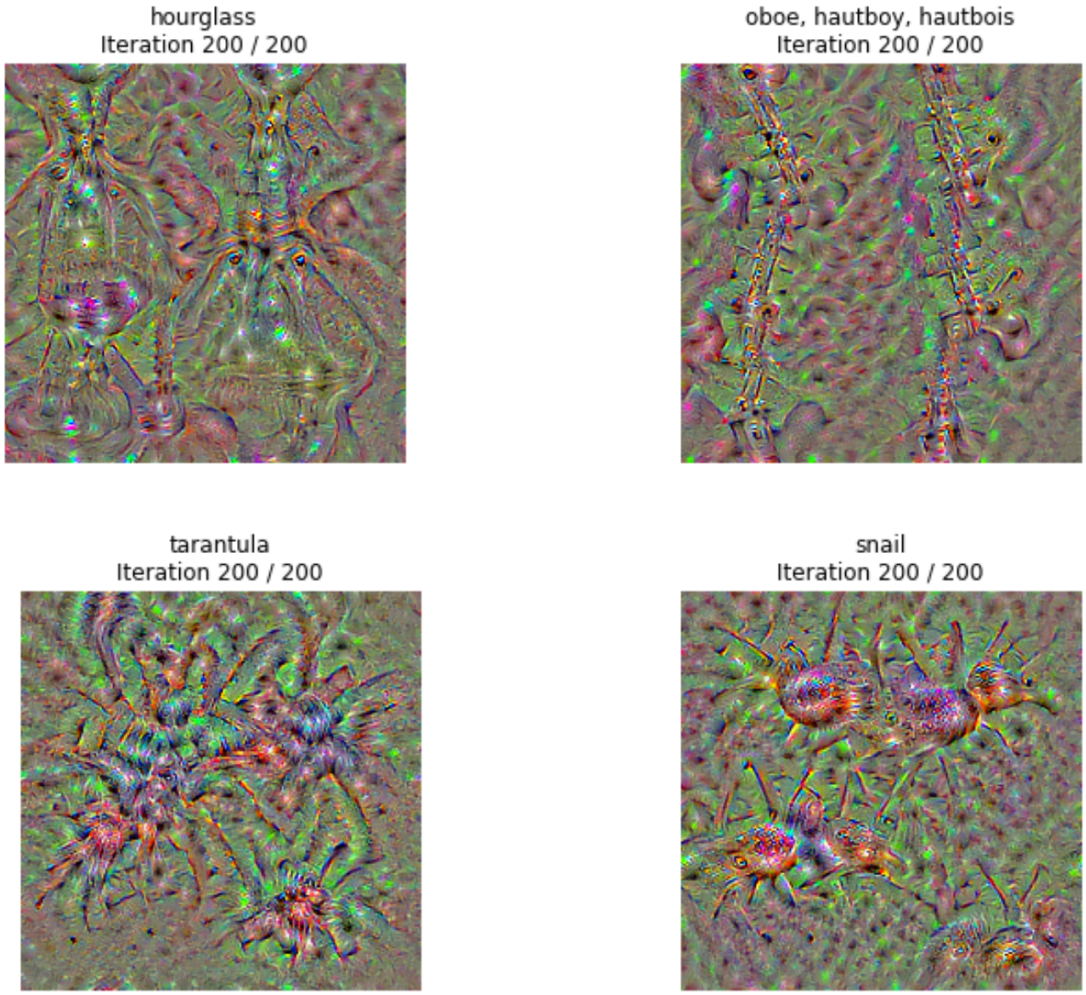


Figure 2.4: Class Visualization of four classes by SqueezeNet network.

Question 2.9: Try to vary the number of iterations and the learning rate as well as the regularization weight.

From figure 2.5, we are able to state that with high iteration, the generated image show more details of features corresponding to a class. As showed in figure 2.6, with a larger regularization term, the variation of pixel is smaller and with 200 iterations we could not have a great presentation. Given figure 2.7, we can conclude with lower learning rate, the process of generating an image is longer. It is like generating image with smaller iterations.

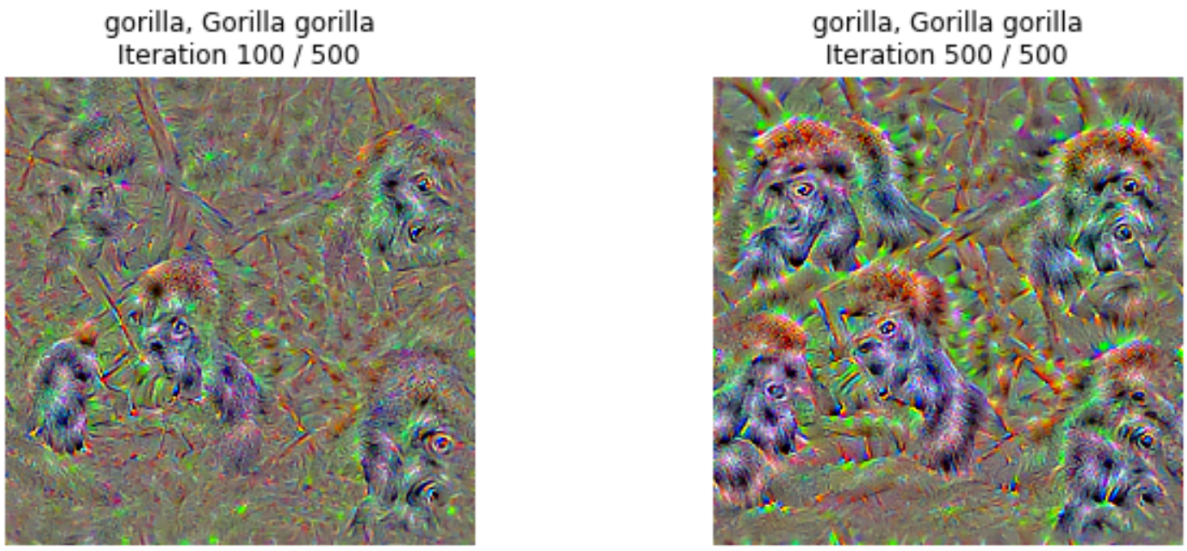


Figure 2.5: Comparison of class visualization with different iterations.

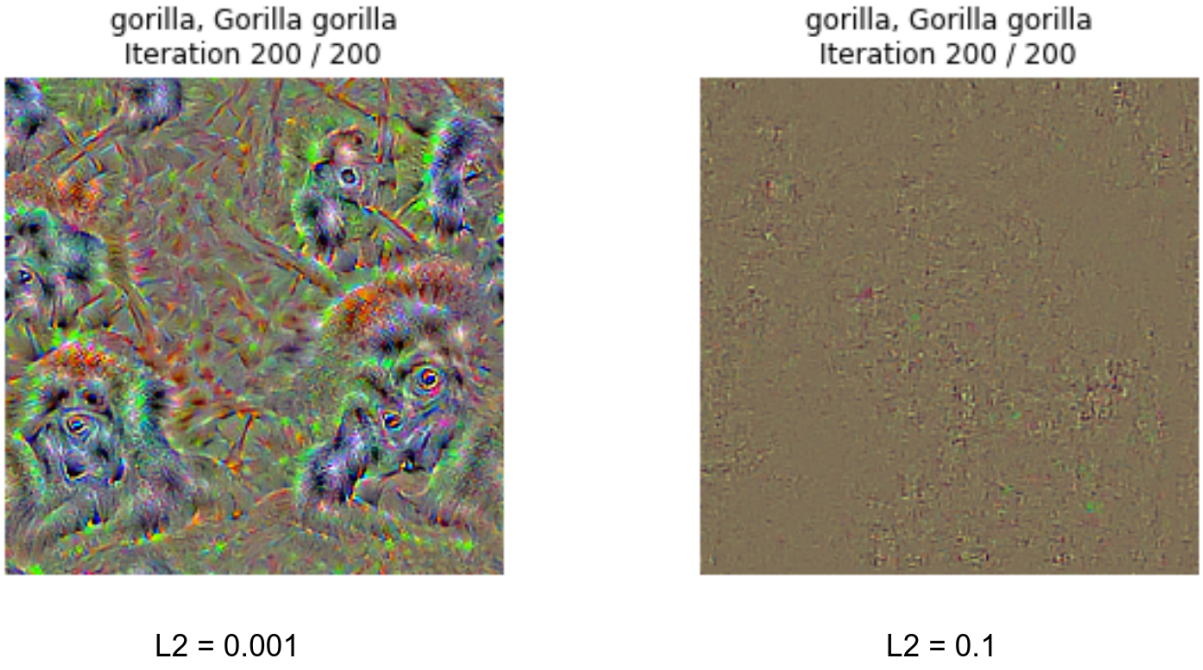


Figure 2.6: Comparison of class visualization with different L2 regularisation.

Question 2.10: Try to use an image from ImageNet as the source image instead of a random image (parameter init_img). You can use the real class as the target class. Comment on the interest of doing this.

With an initialisation, the process of generation will based on original image. If the CNN is well trained, it will concentrate more on the corresponding objects as shown in figure 2.8. This may help filter the background which is not corresponding to the chosen class. This method also given a better illustration of network visualisation as we can directly compared original image with the generated contour in one image.

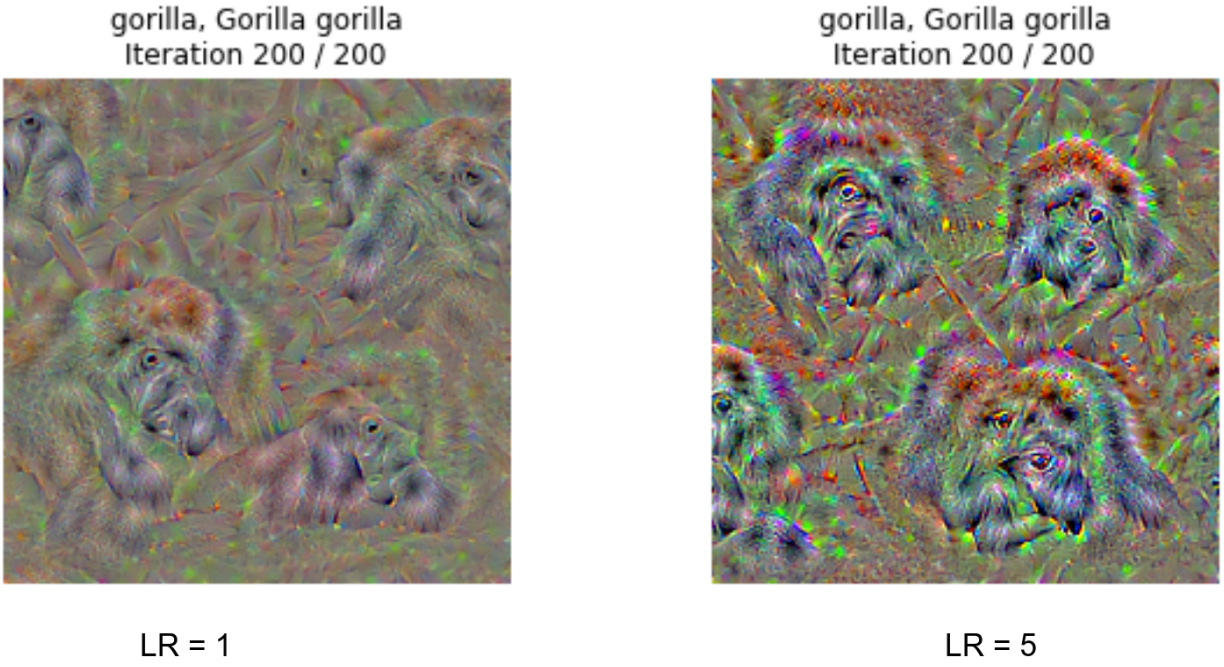


Figure 2.7: Comparison of class visualization with different learning rate.

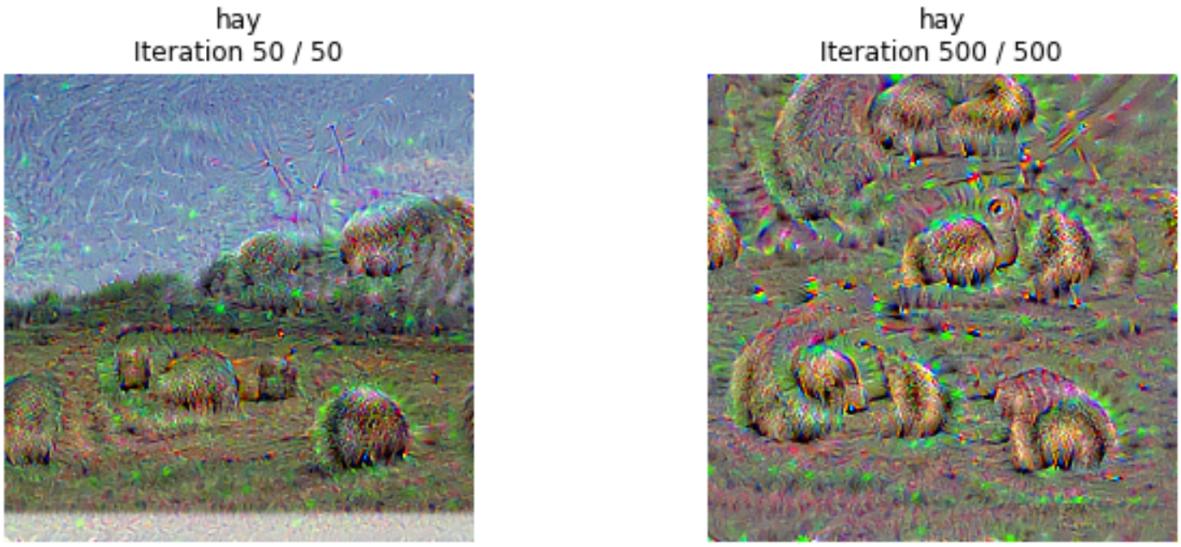


Figure 2.8: Generate an image of class hay from an image of class hay.

Question 2.11: Bonus: Test with another network, VGG16, for example, and comment on the results.

With VGG16, we find that running time is much more longer and the generated image is quite different as shown in figure 2.9. We would say that the visualisation is quite difficult and complicated as network become deeper and bigger.

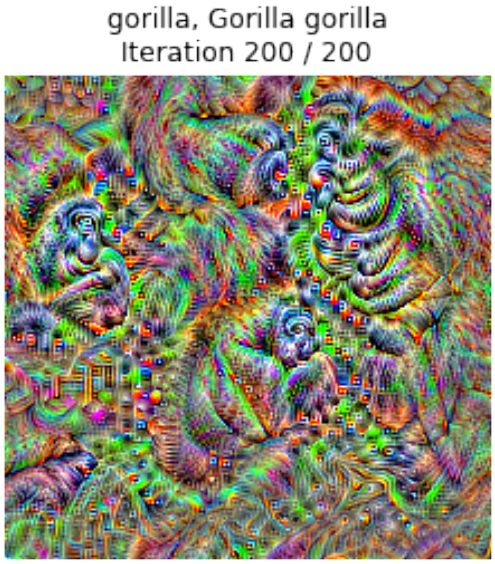


Figure 2.9: Generate an image of class gorilla from random signal with VGG16.

Chapter 3

3-c. Domain Adaptation

3.1 DANN and the GRL layer

We display our results here before answering the questions.
We trained a first naive network on the source dataset for 50 epochs and tested it on the target network to get a baseline. See the results in the following table.

	Test loss	Test Accuracy
Source dataset	1.47	98.95 %
Target dataset	1.93	53.0 %

We notice the performance on the source dataset is great, close to 99%, but the generalization is poor with only about 50% accuracy on the target dataset. We now train a new **DANN** network with an added domain classifier on 50 epochs and compare the results.

	Test loss	Test Accuracy
Source dataset	1.47	98.88 %
Target dataset	1.68	79.32 %

We see a significant improvement from about 50% to 80% in accuracy on the target dataset and pretty much the same performance for the source set as before, which is a clear demonstration of the effectiveness of the method.

See how the generated features look like for both the network in Fig. 3.1

Question 3.1: If you keep the network with the three parts (green, blue, pink) but didn't use the GRL, what would happen ?

If we keep the same network without a Gradient Reversal Layer, then the feature generator does not aim to fool the domain classifier anymore but to help it classifying the domains correctly.
This is the opposite of what we want because the generator will tend to separate the features for both domains and the label classifier will have a harder time generalizing to both domains.

Question 3.2: Why does the performance on the source dataset may degrade a bit ?

The performance on the source dataset might degrade because part of the full label classifier, now the feature generator, has a modified loss, which both accounts for the label classification error and for the domain classification success.
In other words, the feature generator doesn't focus only on the label classification of the source images anymore but it now has two goals which might conflict a little and reduce peak performance.

Question 3.3: Discuss the influence of the value of the negative number used to reverse the gradient in the GRL.

The value of the negative factor is what determines the balance between the two terms in the feature generator loss. If we set it to a high absolute value at first, it means that the generator will try to hard to confuse the domain classifier, at the cost of the label classification. This might be a problem because it might significantly slow down the learning, and we are not interested in a network that merges the domain perfectly but does a terrible job a classifying the labels.
Thus a more promising method is to set the factor to a low absolute value at first, therefore making the network prioritize label classification, and then increasing it slowly to smoothly orient the feature generator to merge the domains and fool the domain classifier. This is the approach proposed in the original paper [1].

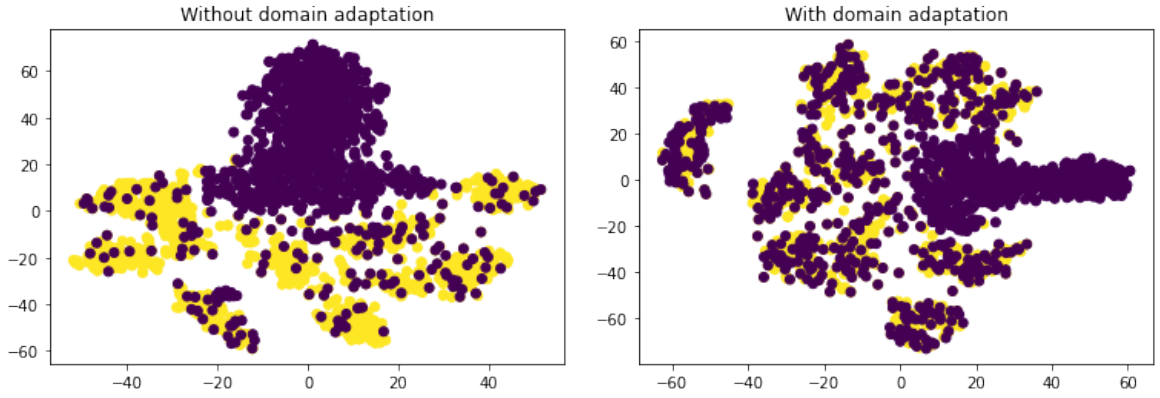


Figure 3.1: Features generated and projected in 2D for the Naive Network (Left Panel) and the DANN Network (Right Panel).

Question 3.4: Another common method in domain adaptation is pseudo-labeling. Investigate what it is and describe it in your own words.

Pseudo-labeling is a very intuitive method to deal with problems where a lot of unlabeled data is available but little labeled data. It consists of following these easy steps :

- Train the model on the labeled data
- Predict labels for the unlabeled data
- Train over all the now labeled data
- Repeat by predicting new labels for the unlabeled data

This kind of sounds like a sham of the likes of some perpetual energy machine, since we don't seem to add any new information to the dataset by following these steps. This however does work, and can lead to impressive results on some applications.
The reason why it works is very similar to the reason why K-means clustering works, as the distance from each predicted label to the centroids of the classes reduced at each step.
The labels end up being closer to the reality at each step and this can represent a significant increase in the final accuracy on a model.

Chapter 4

3-d. Generative Adversarial Networks

4.1 Generative Adversarial Networks

4.1.1 General Principle

Question 4.1: Interpret the equations (6) and (7). What would happen if we only used one of the two ?

- Equation (6) represents the objective of the generator G which is to maximize the probability that the discriminator D will classify it's output as a real image.
- Equation (7) represents the objective of the discriminator D which is to maximize the probability that it will correctly classify the real data as real and the generated data as fake.

If we only used (6), the discriminator would be terrible at predicting whether an image is real or not, and the generator would learn to produce exactly the kind of images that the discriminator wouldn't be able to recognize as fake. Thus the discriminator would end up classifying generated images as real data, probably with even more confidence than actual real data. The generator would outplay the discriminator.
On the other hand, if we only used (7), the generator would only generated random noise while the discriminator would easily learn to identify this output as fake, and real data as real. The discriminator would outplay the generator.

Question 4.2: Ideally, what should the generator G transform the distribution P(z) to ? That is to say G will generate fake images that is very close to real images

Ideally, the generator would perform a transformation on vectors $z \sim P(z)$ in order to output $G(z) \sim P_G$ with P_G a distribution as close as possible to P_{data}.

Question 4.3: Remark that the equation (6) is not directly derived from the equation 5. This is justified by the authors to obtain more stable training and avoid the saturation of gradients. What should the "true" equation be here ?

The "true" equation should be :

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

The paper specifies that in early learning, $D(G(z))$ is very close to 0 because the generated images are easy to discriminate, therefore the *log* term saturates, and it is easier to maximize equation (6).

Question 4.4: Comment on the training of the GAN with the default settings (progress of the generations, the loss, stability, image diversity, etc.

The training of the GAN is, as expected, somewhat unstable and hard to track, even though it works well.
The losses (Fig. 4.2) are not decreasing steadily and oscillate between themselves, as when the discriminator gets better, the generator loss increases, and conversely. We can clearly see how the discriminator loss goes down quickly making the generator loss increase, up until the generator finds a new way to fool the discriminator and reduces its own loss significantly, making the discriminator loss rise again.
Controlling the generation is the best way to keep track of the GAN's performance, and doing so reveals how chaotic the training really is : the results don't always improve after more epochs and sometimes get worse (see Fig. 4.1 where some artifacts appear late in the training and then go away. Notice also how the generation on the right, at 50 epochs, isn't as good as other results we obtained earlier in the training).
The image diversity is something that must be monitored as GANs tend to suffer from mode collapse, where the generator finds one way to fool the discriminator at one point and then overuses this trick to the point that it doesn't try to make various creations anymore but masters only one task (making only **1**'s for example.
Fortunately our network didn't suffer from it here.
The middle panel in Fig. 4.1 shows some manifestation of a problem where the generator keeps using the same design for all the **9's**, which eventually will leads to the discriminator reacting and punishing it.

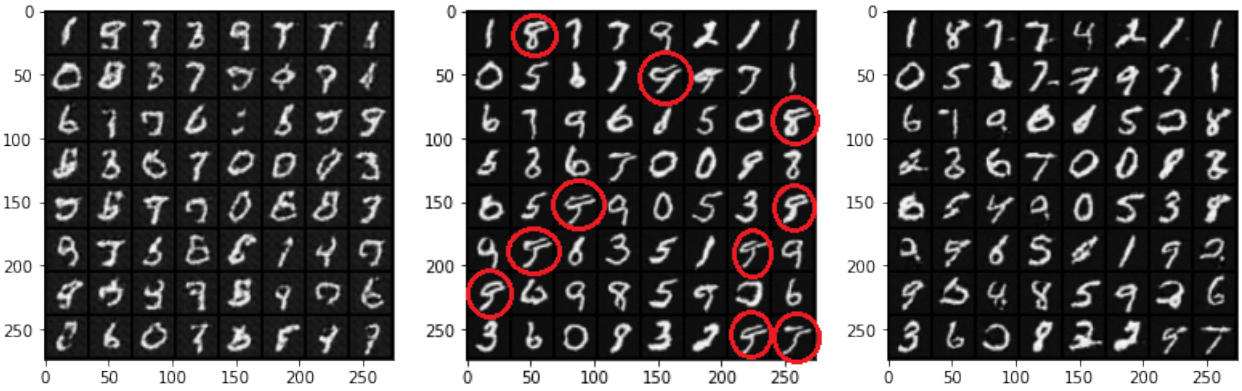


Figure 4.1: Output of GAN training after 200, 21000 and 23000 batches respectively.

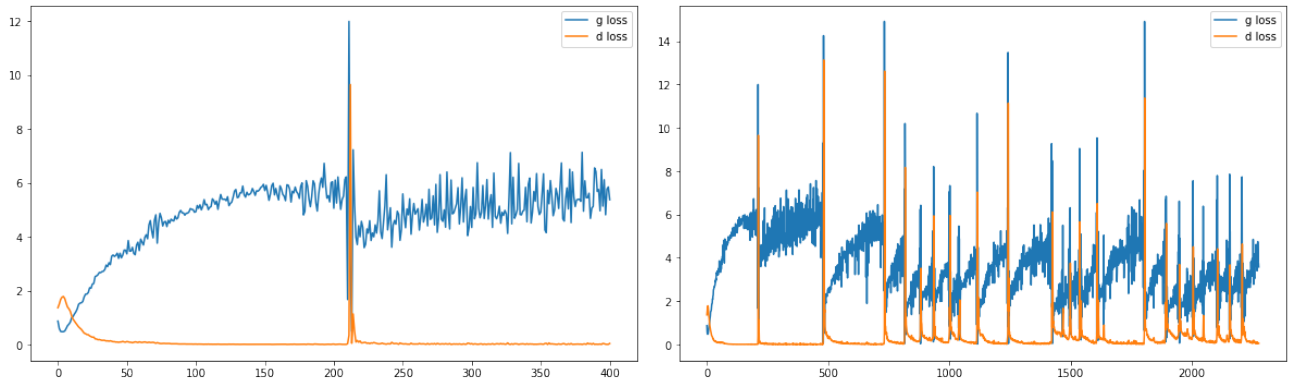


Figure 4.2: Loss of GAN training.

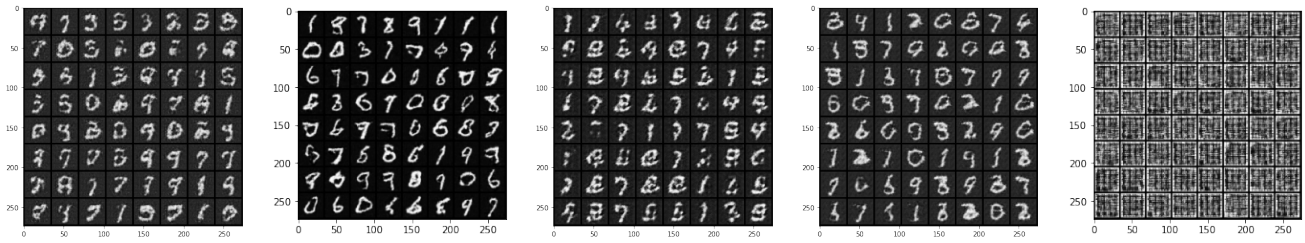
Question 4.5: Comment on the diverse experiences that you have performed with the suggestions above. In particular, comment on the stability on training, the losses, the diversity of generated images, etc.

We tried multiple variations on the training which can be seen in Fig. 4.3. Changing the size of the random vector z has some effect on the convergence rate, as can be seen after 1600 iterations for various nz, but isn't as much important later on. It seems like a smaller vector however tends to make the training easier but also to reduce the diversity of the examples.

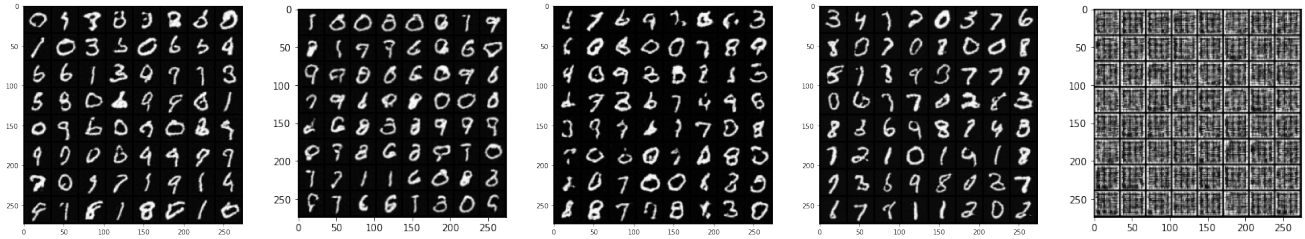
Greatly increasing the number of neurones in the generator network made the computation much slower, as one would expect, but the performance after a fixed number of iterations seems to be slightly superior.
Greatly increasing the same number in the discriminator however broke the training entirely as can be seen in 4.3 on the right-most panels.

Question 4.6: Comment on your experiences with the conditional DCGAN.

The training of the conditional GAN is straightforward as the generation steadily improves and is quick. The loss doesn't show the unstability we saw with the DCGAN. The results look promising after 1000 iterations and already above those of the DCGAN after 1800 iterations, as can be seen in Fig 4.5.

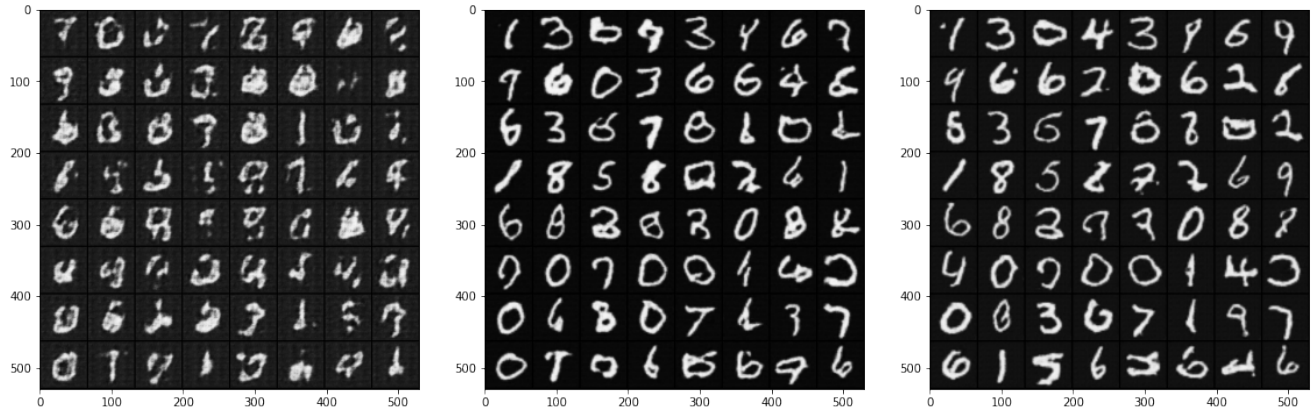


(a) 1600 it, nz=10 (b) 1600 it, nz=100 (c) 1600 it, nz=1000 (d) 1600 it, nz=10 (e) 1600 it, nz=100



(f) 4000 it, nz=10 (g) 4000 it, nz=100 (h) 4000 it, nz=1000 (i) 4000 it, nz=10 (j) 4000 it, nz=100

Figure 4.3: Output of GAN training with various hyperparameter tweak, after 1500 batches (top panels) and 4000 batches (bottom panels).

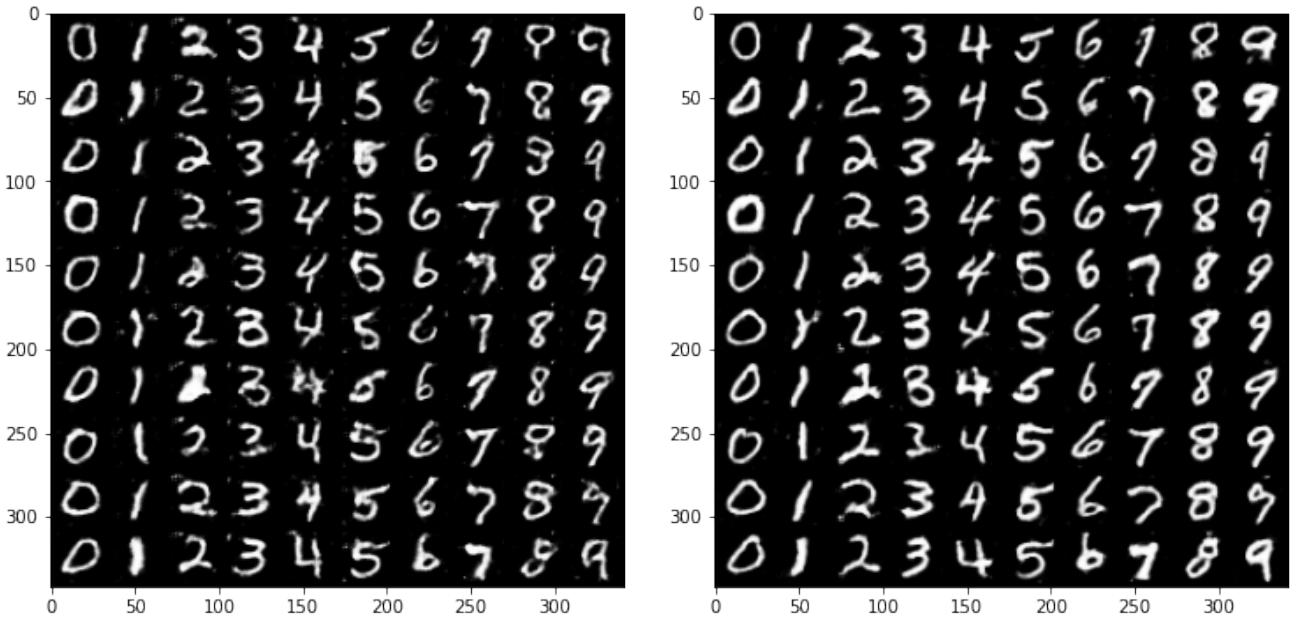


(a) 1600 it (b) 4000 it (c) 16000 it

Figure 4.4: Output of GAN training with 64x64 images. The training is slower but the results look similar to those with 32x32.

Question 4.7: Could we remove the vector y from the input of the discriminator (so having cD(x) instead of cD(x, y)) ?

Removing the vector y from the input of the discriminator would leave it unable to tell whether an image has the desired label it is supposed to, so there would be no information given to the generator on whether it created the right mode. Therefore the conditional GAN wouldn't go its job anymore of conditioning the generation.



(a) 1100 it

(b) 1800 it

Figure 4.5: Output of cDCGAN training.

Question 4.8: Was your training more or less successful than the unconditional case ? Why ?

The training of the conditional GAN is very efficient and straightforward, probably partly thanks to the well tuned hyper-parameters, but also thanks to the added information given to both networks. The generator does not only have guidance on whether its generation looks like numbers anymore but also on which number it is creating, which might help it to learn what each number looks like.
The losses are steadier and seem more stable than these of the DCGAN, as can be seen on Fig. 4.6.

Question 4.9: Test the code at the end. Each column corresponds to a unique noise vector z. What could z be interpreted as here ?

We observe in Fig. 4.7 that the vector z influences the style of the generation, with different kinds of writing and intensity.
This vector could be interpreted as different origins for the data, such as different writers.

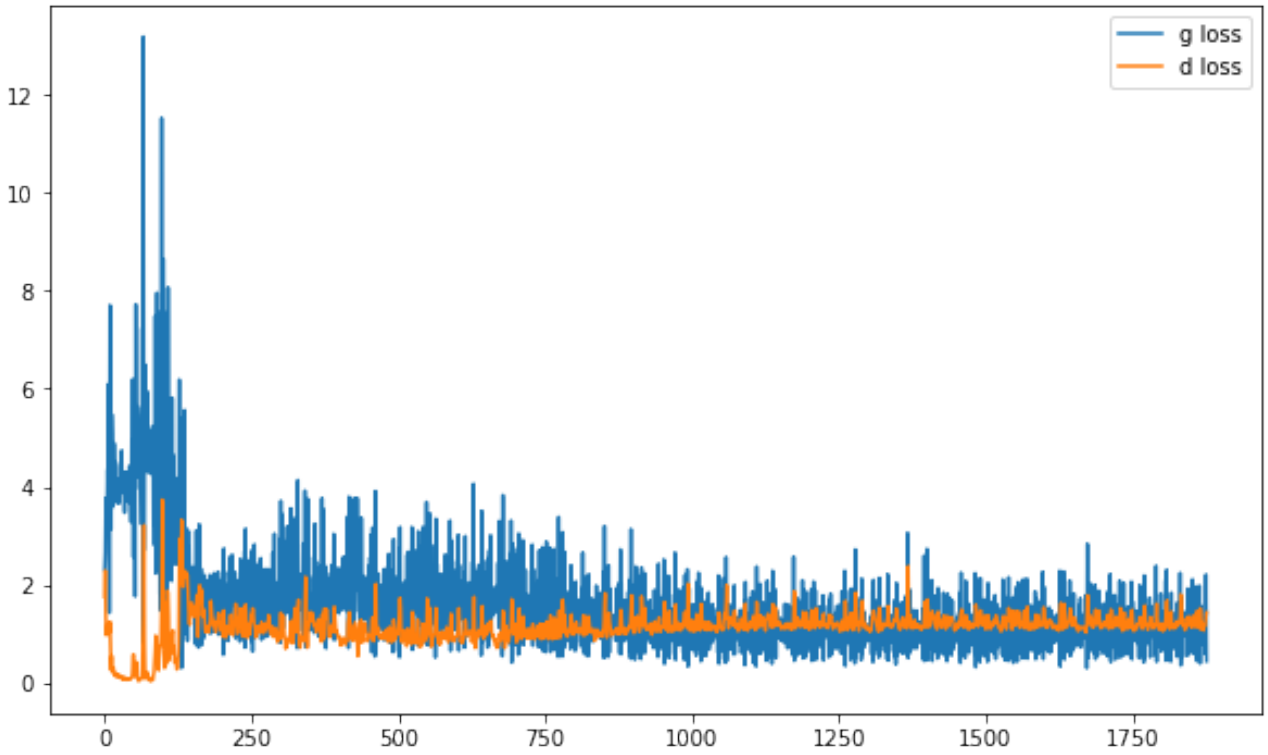


Figure 4.6: Loss of cDCGAN training after 1800 batches.

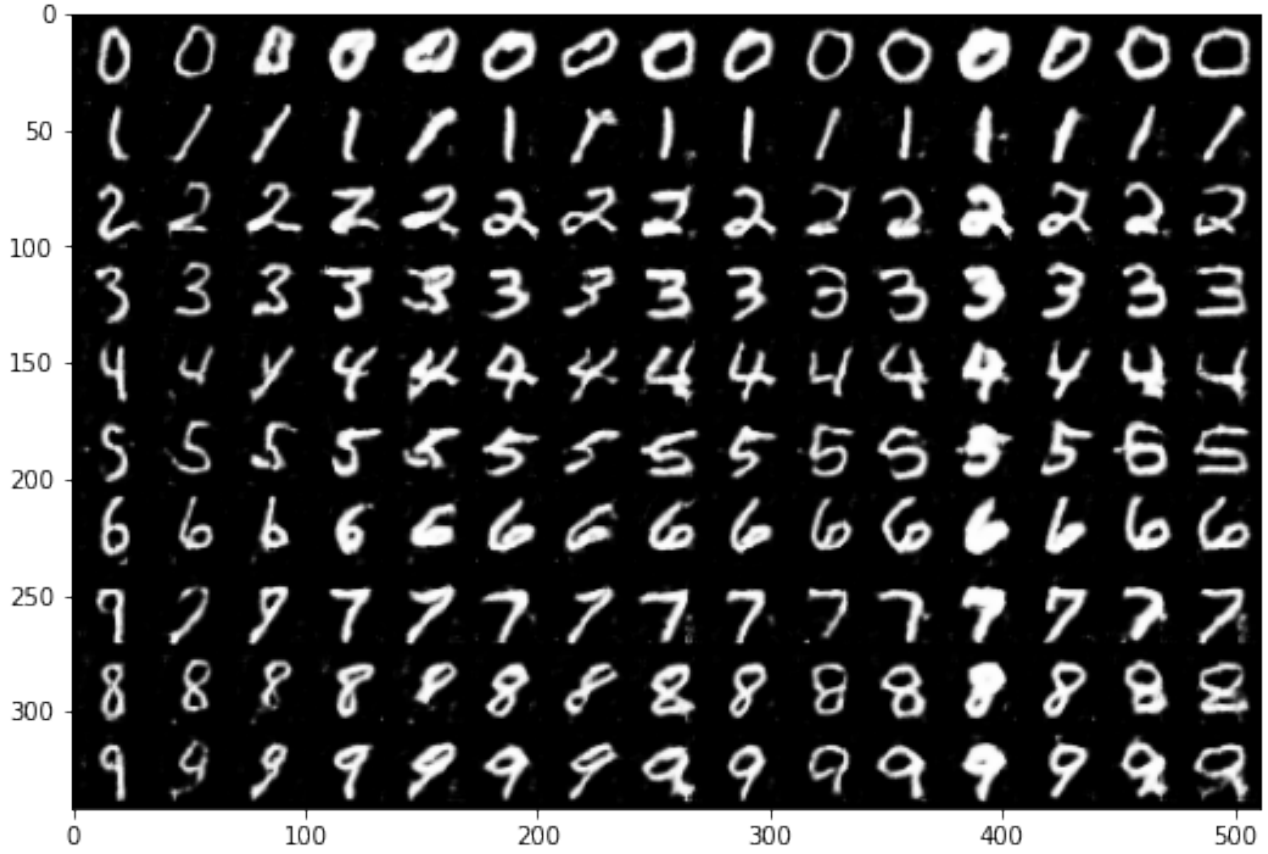


Figure 4.7: Various outputs of the cDCGAN for various vectors z.

4.2 Annex

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

Figure 4.8: VGG16 Network parameters details.

Bibliography

[1] Yaroslav Ganin and Victor Lempitsky. *Unsupervised Domain Adaptation by Backpropagation*. 2015. arXiv: 1409.7495 [stat.ML].

[2] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV].