RDFIA Practical work 3

Giulia Prosio and Alexander Hölzl

DATE

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

Summary

The goal of this first practical work is to overcome the over-fitting problem that we commonly encounter when training our model on a subset that has not enough data.
Because of this the model would be too specialized on the training-set data and would not perform well when using it on new data.

The idea that we want to implement to solve this problem is to train the model on a much larger set of data from a bigger available data set similar to the one we are analysing.
This way the model will develop strong *feature extraction* tools that we can use to train the classification network.
Once we have developed the CNN, we can use it with our original dataset to have good results without falling into over-fitting.

Question 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
Only considering the fully-connected layers of the VGG16 architecture we consider its last three layers which have:

- fc1: in: 25088, out: 4096 -> 102760448 parameters
- fc2: in: 4096, out: 4096 -> 16777216 parameters
- fc3: in: 4096, out: 1000 -> 4096000 parameters

So just considering the fully connected layers we have circa 123633664 parameters.

Question 2

What is the output size of the last layer of VGG16? What does it correspond to?
The output size of the last layer of VGG16 is *1000*, it represents the list of the thousand classes from ImageNet that the images can be classified to.

Question 3 - bonus

Apply the network on several images of your choice and comment on the results

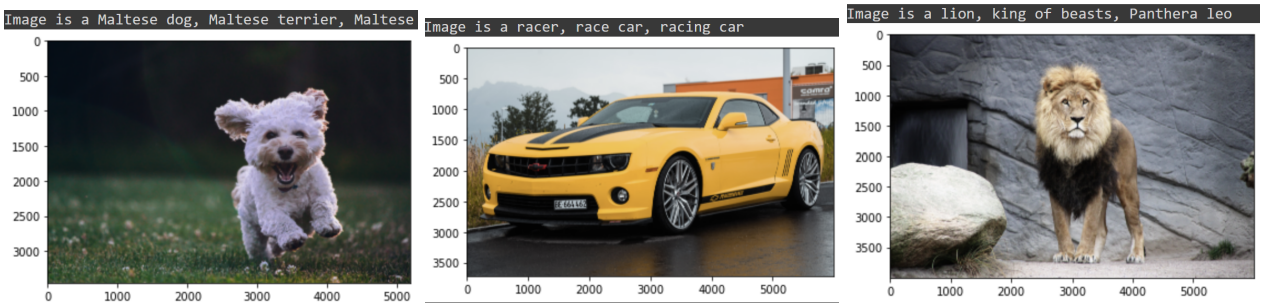
We have input the pictures shown in figure 1 to the CNN build using the Transfer Learning technique through feature extraction.
The three pictures are taken from the internet, and represent a lion, a maltese dog and a race car.
As we can see from the labels on top of each image, the prediction given by the CNN for each picture was not only correct, but even very specific.

Question 5

Why not directly train VGG16 on 15 Scene?
We do not train it directly on *15Scene* dataset because it has too few data and using it to train the model would mean incurring in over-fitting.

Question 6

How can pre-training on ImageNet help classification for 15 Scene?
It helps the classification because the dataset has much more available and usable data and the two data sets have similar data, so we are able to build a strong network that can then be used for feature



(a) Maltese dog (b) Race car (c) Lion

Figure 1: Image Classification with CNN using Transfer Learning

extraction in *15 Scene* dataset even though it does not contain a lot of data.
This way we are able to successfully classify the dataset without incurring in over-fitting.

Question 7

What limits can you see with feature extraction?
The main problem with the feature extraction approach with pre-trained models is that if the domains of the two dataset are very different, the extracted features in the first dataset cannot be used to perform feature extraction in the second dataset.

Another problem is the high cost of building these feature extraction models, as they learn on a huge amount of data, and sometimes it is not worth the effort but it may suffice to fine-tune the hyperparameters of the model on the original smaller dataset.

Question 8

What is the impact of the layer at which the features are extracted?
The deeper the layer, the more specific features can be extracted.

Question 9

The images from 15 Scene are black and white, but VGG16 requires RGB images. How can we get around this problem?
Since the images from the *15 Scene* dataset are black and white, their pixels are encoded with a value between 0 and 255 in only one channel.
To overcome this problem, since VGG16 requires RGB images, we can copy three times the black-and-white channel of the original images to have the three *red, green, blue* required channels.

Question 10

Rather than training an independent classifier, is it possible to just use the neural network? Explain
Yes it is possible, it may suffice to add another final fully connected layer to the neural network to perform the classification task.

Question 11

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

The changes we tested in our model are the following:

- *change the layer at which the features are extracted*
  To observe the differences made by this change we compared the *accuracy* value of the svm model.
  The original layer set as the one to extract features was '-2', which means the second starting from the end of the network architecture.
  Its Accuracy is 88.509213.
  We then changed the layer to '-4', so the fourth layer counting from the end of the network architecture. This means that we removed other final layers from the model. The Accuracy obtained here is 90.284757.
  This increase in the accuracy value is explained by the fact that we train the model on a different dataset. Removing further layers from the end of the network can help the model to be more general and so obtain an increase in its performances.
  We also put the feature extraction layer at '-7' and observed an Accuracy value of 90.251256, so degrading a bit from the '-4' solution.
  This is due to the fact that we are removing too many layers from the model and losing its ability to classify.

- *tune the parameter C*
  The parameter C in SVM represents the degree of misclassification that we want to avoid during training of the SVM.
  The original value was set at C=1.0, to which corresponded an Accuracy level of 88.509213.
  We observed that when increasing the value of C, the increase of the Accuracy value would be insignificant. This means that the data was already split in such a way that even if the margins of the hyperplane are narrower the misclassification value does not change.
  On the other hand, as expected, if we decrease the value of C - here we used C=0.0000007, the Accuracy value decreases, here to 78.827471, as the number of accepted misclassifications increases.

3-b: Visualizing Neural Networks

Summary

The goal of this practical work is to study some techniques recently proposed to study the behaviour of CNNs, in order to better understand the reasoning behind the decisions made by the machine. They all use the gradient of an input image with respect to an output class.

Saliency Map

Question 1

Show and interpret the obtained results.
The first technique we study is the Saliency Map, which outputs a map highlighting the parts of the image on which the neural network pays most attention.

As we can observe from the shown images in figure 2, what the saliency map does is to give interpretation to the pixels composing an image, and in particular show if they are taken into account by the NN or not.

Question 2

Discuss the limits of this technique of visualizing the impact of different pixels.
A limit of this technique is that it takes into account the singular pixels but oftentimes it is important also to study the relationship between the pixels to have an accurate classification process.

Question 3

Can the Saliency Map technique be used for a different purpose than interpreting the network?
The Saliency Map is a technique widely used in Medical Image Analysis in order to support the doctor's decision making concerning the state of health of a patient.
It can be used to understand how the network arrives at its decision which can give the medical personnel further information needed to validate the diagnosis.

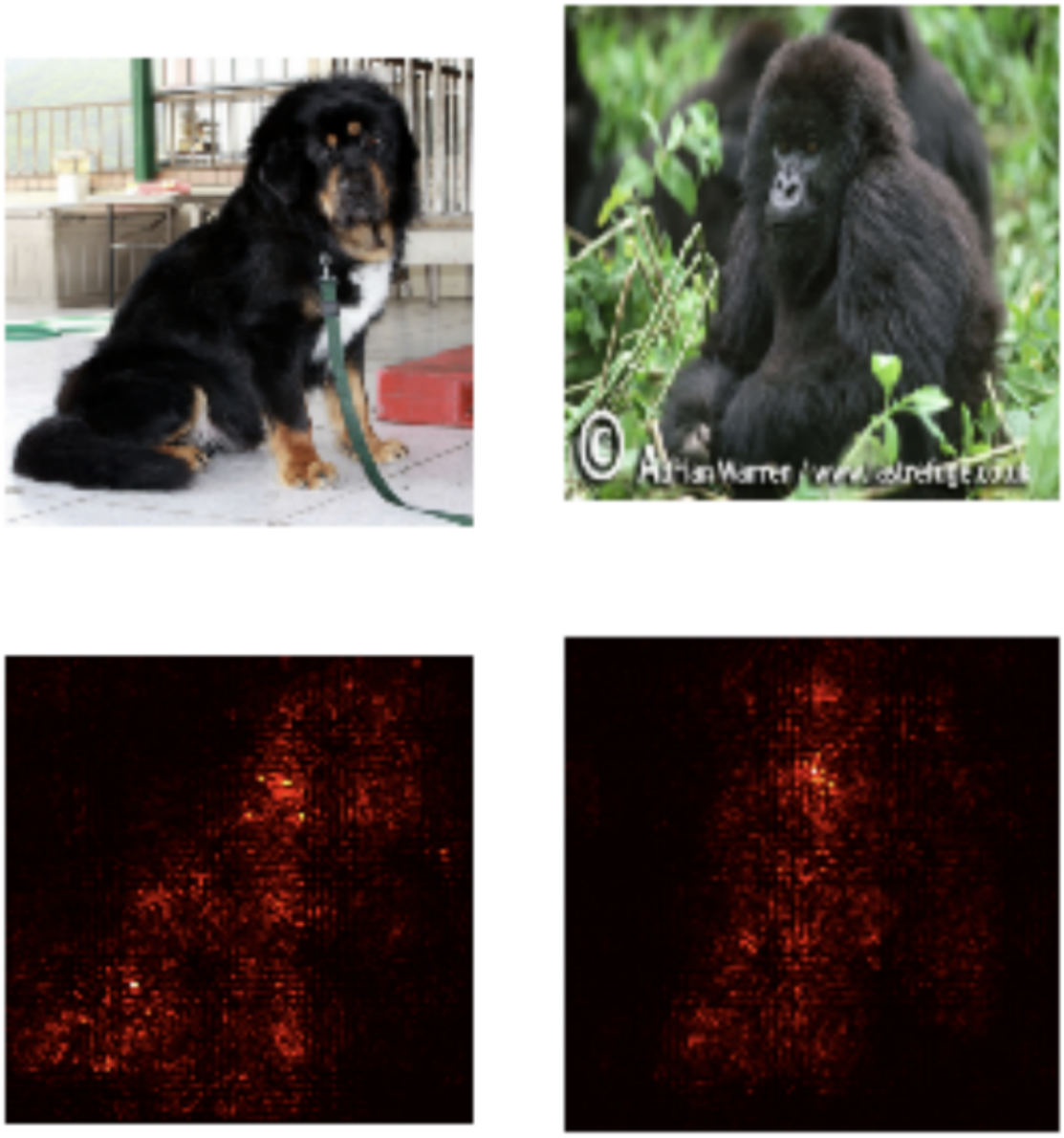
Adversarial Examples

Question 5

Show and interpret the obtained results
Adversarial examples are *fooling examples*. The purpose of using this technique is to change an image that was correctly classified by the neural network as little as possible so that it gets misclassified.

What was proposed to do throughout this section was to consider an image correctly classified in a class i and modify it - not the network - so that it is going to be classified in a class j.

As we can see from figure 3, by adding noise in the picture of a correctly classified quail, we are able to fool the neural network into classifying it as a stingray.



(a) tibetan mastiff (b) gorilla

Figure 2: Examples of produced Saliency Maps

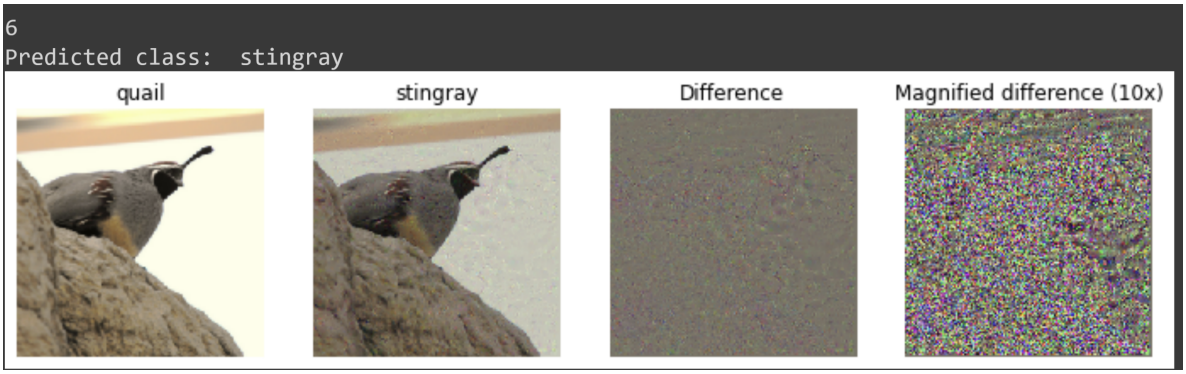
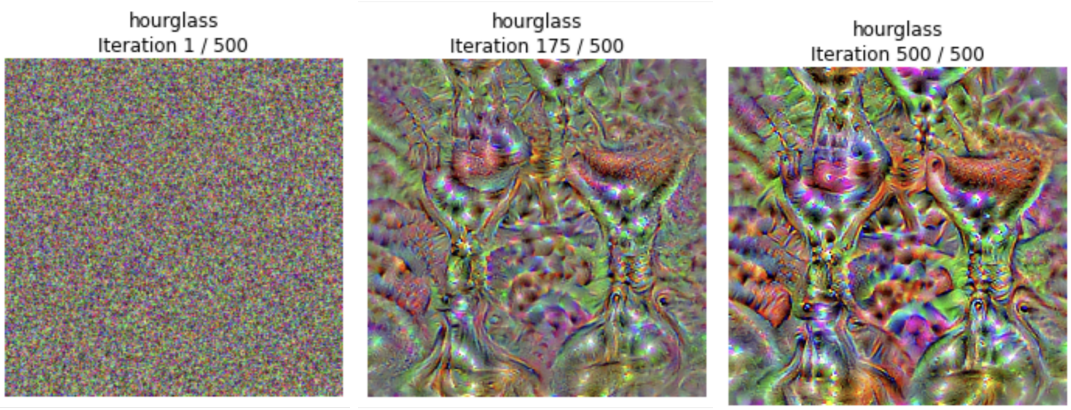


Figure 3: Adversarial Example: from quail to stingray



(a) hourglass iteration 1 (b) hourglass iteration 175 (c) hourglass iteration 500

Figure 4: Class Visualization from random noise to hourglass

Question 6

In practice, what consequences can this method have when using convolutional neural networks?
We can use the information given by this technique to make the NN more robust to little changes and avoid misclassification even if the domain of the image changes slightly.
As the network can be used to perform tasks that might endanger human life is not carried out correctly - such as traffic sign recognition in autonomous vehicles - it is important be able to understand what might lead such networks to misclassify images.

Class Visualization

Question 8

Show and interpret the obtained results
The last analysis technique studied here is Class Visualisation. The idea of this technique is to visualize the type of patterns likely to produce a particular class prediction.

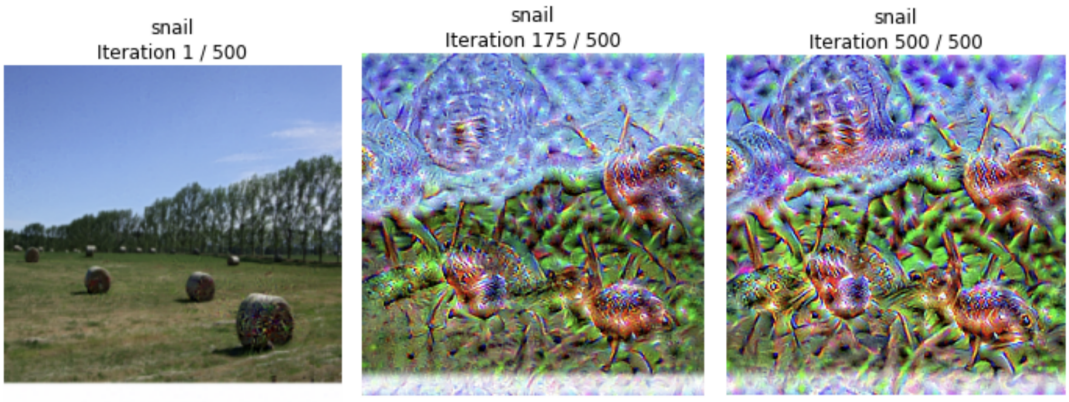
We have used this technique in the project in two different ways: first taking an image composed by random noise and transforming it to visualize the patterns that would lead the network to classify it as "hourglass".
As shown in figure 4.

The second exercise consists in starting from an image and enhacing the patterns that would lead the network to wrongly classify it. Extract of the iterative process shown in figure 5.

Question 9

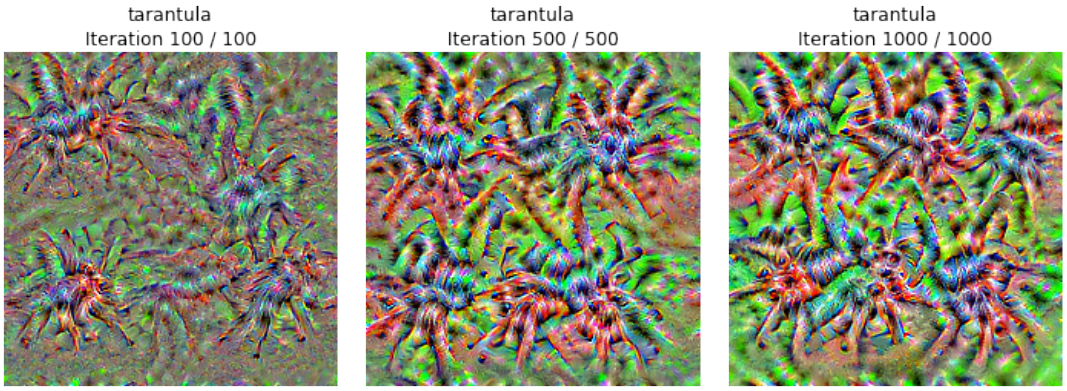
Try to vary the number of iterations and the learning rate as well as the regularization weight

- Changing the number of iterations to obtain the pattern that the network would search for to classify the image as a tarantula. Examples seen in figure 6.
  As we can observe increasing the number of iterations will produce more specific patterns in output.
- Changing the learning rate of the network wanting the patterns to classify an image as gorilla. Examples seen in 7.
  As we can observe a too big or too small learning rate can produce in output a not well defined



(a) hail as snail iteration 1 (b) hail as snail iteration 1 iteration 175 (c) hail as snail iteration 500

Figure 5: Class Visualization from hail miss-classified as snail



(a) 100 iterations (b) 500 iterations (c) 1000 iterations

Figure 6: Changing nb of iterations of tarantula class visualization

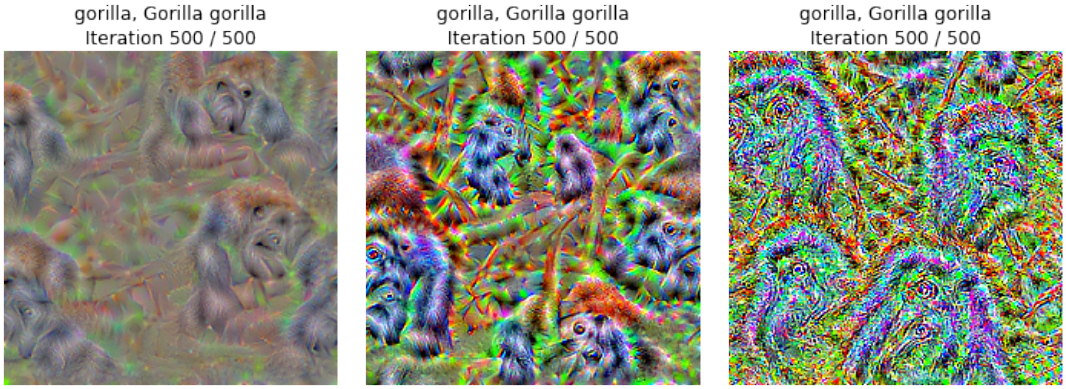
picture, as the jumps taken from one iteration and the other are either too small, requiring a bigger number of iterations to produce a valid output, or too big, with the risk of missing out important information.

- Changing the regularization weight of the network wanting the patterns to classify an image as hourglass. Examples seen in 8.

Question 10

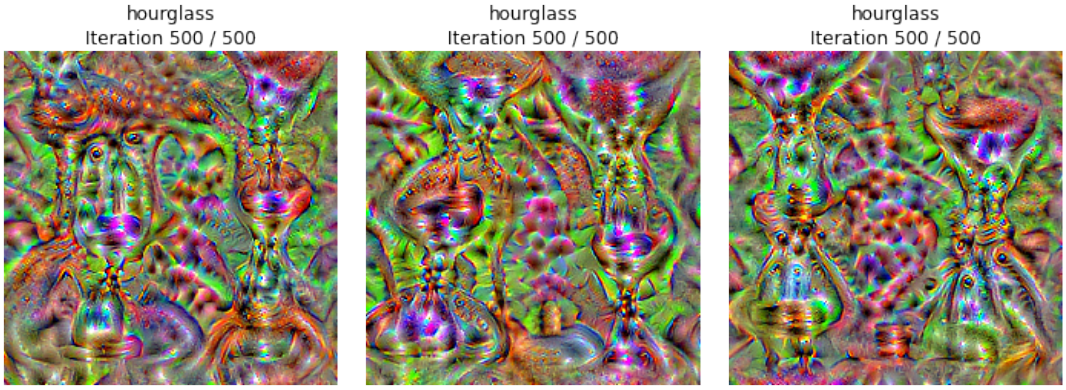
Try to use an image from ImageNet as the source image instead of a random image (parameter init_img). You can use the real class as the target class. Comment on the interest of doing this.

For this question we used the picture of a Border Collie, taken from the dataset ImageNet.
As we can observe from figure 9, after 500 iteration of the class visualization network we can still recognize the dog, but its features have been exaggerated in the direction for the network to classify it as Border Collie.



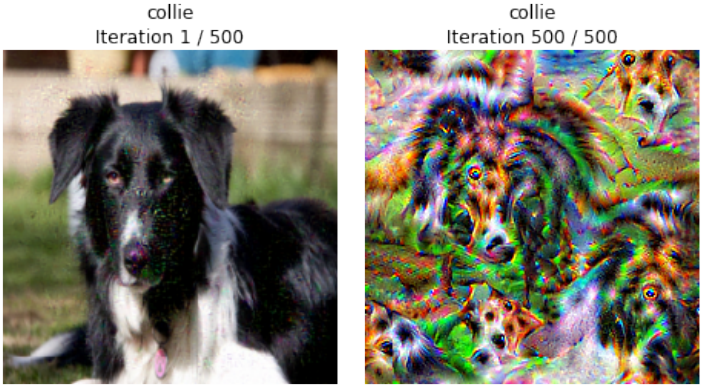
(a) learning rate 0.5 (b) learning rate 5 (c) learning rate 50

Figure 7: Changing learning rate of gorilla class visualization



(a) regularization weight e-2 (b) regularization weight e-3 (c) regularization weight e-4

Figure 8: Changing regularization weight of hourglass class visualization



(a) original picture (b) picture after 500 iterations

Figure 9: class visualization of border collie dog from ImageNet

3-c: Domain Adaptation

Summary

A big problem when training neural network is the difficulty of obtaining correctly labeled data. Domain adaptation tries to approach this problem by training on a labeled source set while simultaneously also being able to perform well on a similar target set of unlabeled data. The network consists of three parts, a feature extractor, a label predictor and a domain classifier. The feature extractor, as the name suggests extracts features from the input images, and then label predictor is trained to predict the labels of the labeled set. The most important part of the network, which allows the processing of unlabeled data is the domain classifier - its job is to say if a sample belongs to the labeled source domain or the unlabeled target domain. In order to do the classification for both sets we want the feature extractor to learn features that generalize well which means that the domain classifier can't distinguish between the two sets. That also means we want to maximize the loss with regard to the domain classifier. This is done by reversing the gradients in the gradient reversal layer which sits between the feature extractor and the domain classifier.

The goal of this exercise was to implement this DANN network, as described above.

Question 1

If you keep the network with the three parts (green, blue, pink) but didn't use the GRL, what would happen?
The domain classifier would always be able to distinguish between the two domains and the feature extraction is not forced to learn features that generalize well for both domains and that means there will be no performance gain on the labeled data.

Question 2

Why does the performance on the source dataset may degrade a bit?
Because the features are more general and so they lose a bit of specific adaptation to the source dataset.

Question 3

Discuss the influence of the value of the negative number used to reverse the gradient in the GRL.
Results with GRL factor set to -1 fix

```
[SOURCE] Class loss/acc: 2.3588 / 10.28%, Domain loss/acc: 0.31326 / 100.0%
[TARGET] Class loss/acc: 2.3588 / 10.28%, Domain loss/acc: 1.31326 / 0.0%
```

As can be seen, contrary to the intuition one might have, just reversing the gradients in a straightforward way by multiplying them with -1 is not sufficient, a more complex approach needs to be taken. As can be seen the approach given in the tutorial, which uses a varying factor given by $\frac{0.01}{(\frac{1+\alpha*e}{\text{epochs}})^{\beta}}$ yields much better results.

```
[SOURCE] Class loss/acc: 1.49108 / 97.28%, Domain loss/acc: 0.43089 / 100.0%
[TARGET] Class loss/acc: 1.7485 / 71.22%, Domain loss/acc: 0.97696 / 0.0%
```

It is interesting to note that the domain's loss is very high for both cases, as one might expect.
There is a big difference in performance in regards to performance when predicting labels.
For the case where the GRL factor is set to -1 only a accuracy of about 10% can be obtained, which means that the performance of the network is on par with just guessing.

Question 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 technique that works on the following steps:

1. train the model on a batch of labeled data
2. use this model to predict labels for unlabeled data
3. use the predicted labels to calculate the loss on unlabeled data
4. combine labeled loss with unlabeled loss and backpropagate

3-d: Generative Adversarial Networks

GAN

Summary

The aim of this last section of the Project Work was to study GANs, which focuses not on the classification of images but on their recreation.
To do this, the Generative Adversarial Networks make use of two neural networks: the Generator, which generates images from a random distribution of noise trying to replicate a particular distribution and the Discriminator, another neural network whose aim is to distinguish real images from those generated by the Generator.
By optimizing both networks we are able to produce false images that are indistinguishable from the source set.

Question 1

What would happen if we only used one of the two objective functions to optimize Generator and Discriminator?

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

The Generator tries to minimize this function while the Discriminator tries to maximize it.

While the Generator is trained, it samples random data and produces an output out of it.
The output then goes through the Discriminator, which will classify it as "fake" or "real".
The Generator's loss function is then calculated from the Discriminator's classification, so that it is rewarded if it is able to fool the Discriminator and penalized if not.

On the other hand for each wrongly classified image the Discriminator gets a penalty.

Using only one of the two optimisation functions would lead to an inequality situation where the output obtained from Gan would not be optimal: if we use only the generator optimisation function we try to maximise D(G(z)), thus always having 1 - the discriminator that classifies the image as real. Since the discriminator is not penalised when it misclassifies, it thinks that the approach it is following is the correct one and therefore continues to consider false images as real, leading to incorrect output.

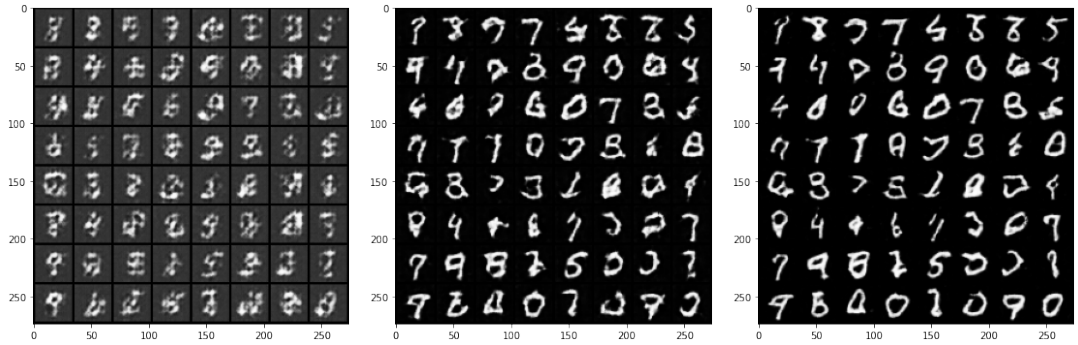
If we use only the optimisation function of the discriminator, we want on the other hand to minimise the number of misclassifications. If in this way the discriminator becomes very good at distinguishing between true and false images, the generator does not improve, it will not be able to carry forward better images than those produced randomly.

Question 2

Ideally, what should the generator G transform the distribution P(z) to ?
With GAN we have a dataset of real pictures, and the objective of the Generator is to generate in output images following that same distribution.
The purpose of the Generator is to transform the fixed distribution of the random vector **z** into something similar to the real dataset.

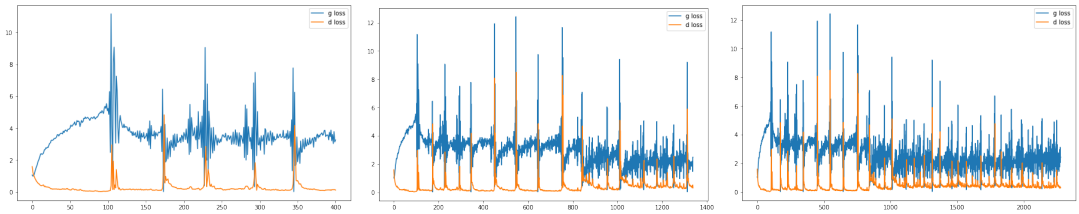
Question 3

Remark that the equation for the optimization of the generator G is not directly derived from the total optimization equation proposed. This is justified by the authors to obtain more stable training and avoid the saturation of gradients. What should the "true"



(a) iteration 400 (b) iteration 1338 (c) iteration 2776

Figure 10: iterations of GAN for MNIST dataset



(a) loss iteration 400 (b) loss iteration 1338 (c) loss iteration 2776

Figure 11: loss iterations of GAN for MNIST dataset

equation be here ?
The authors presented a slightly different version of the loss function for optimization of G, where it maximizes the log of the discriminator probabilities.
To put it in a more practical way, it meas looking at the problem using a different perspective: instead of minimizing the probabilities for an image to be fake, the generator tries to maximize the probabilities for it to be classified as real.

The true equation should be:

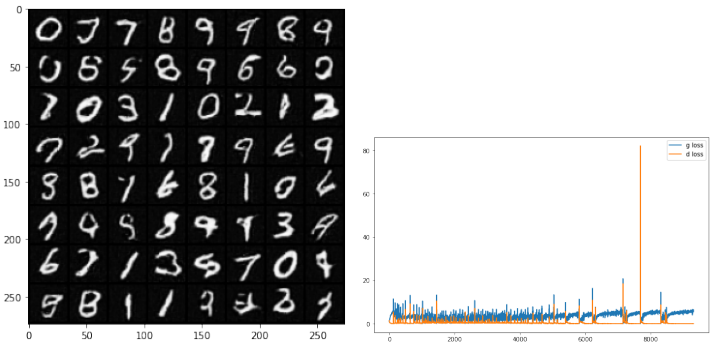
$$\max_G E_{\boldsymbol{z}\sim P(\boldsymbol{z})}[\log(1 - D(G(z))]$$

Question 4

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

The figures 10 and 11 respectively are the output of the GAN model after different iterations and the corresponding generator and discriminator losses.
As we can see the output of the network starts being more realistic after at least a thousand batches.

As we can observe, even if the loss functions are very noisy, is that there is a trend when increasing the number of iterations so that the discriminator's loss function increases and the generator's decreases. This is coherent with the GAN model as when the images generated become more realistic, the discriminator is penalized more often for misclassifying them (and the generator rewarded for the same reason).



(a) iteration 400 (b) iteration 1338

Figure 12: Results of training the GAN for a much bigger number of iterations

Question 5

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

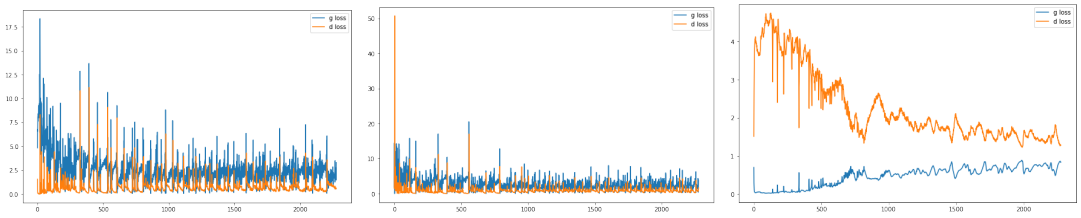
As can be seen in figure 12 training the GAN for a much longer amount of time results in a visual improvement of the generated images, even though it is still noticeable that these are not real images. In this case we trained it for twice as many epochs, 20 instead of 10, as compared to figure 10.

In the figures 13 15 we can observe different results obtained by changing the learning rate of the Discriminator and the Generator in the model.

As can be seen, using a learning rate that is too low for both the generator and the discriminator produces incomprehensible results, far from the starting dataset, as at each iteration the network learns too little for it to be able to produce satisfactory results in the number of iterations considered.

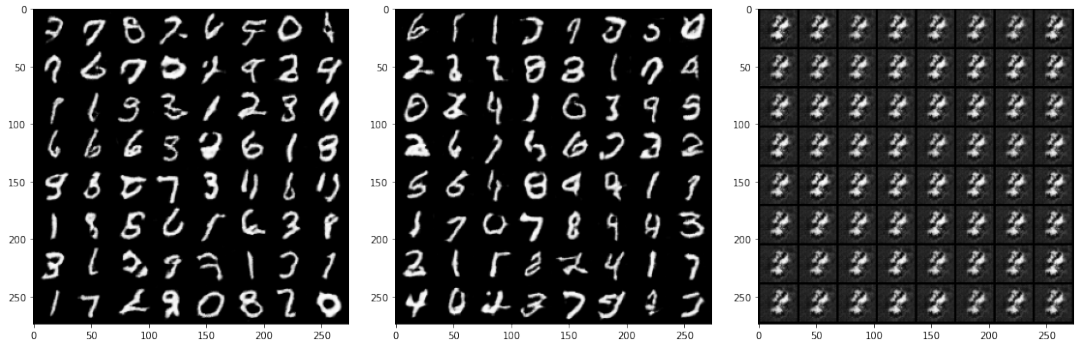
On the other hand increasing the learning rate by a decimal point produces results slightly better than the ones produces with the original settings but the difference is so little that it makes no real sense to change the value.

We can see that a better result can be obtain by simply increasing the number of iterations (figure 12



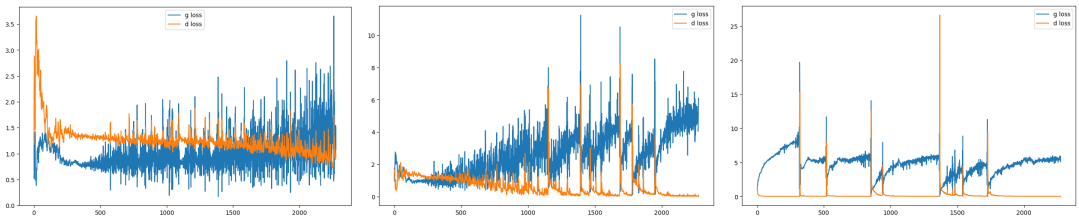
(a) lr discriminator: 0.002 (b) lr discriminator: 0.02 (c) lr discriminator: 0.00002

Figure 13: Comparing performance using different learning rates for the discriminator and a learning rate of 0.0005 for the generator



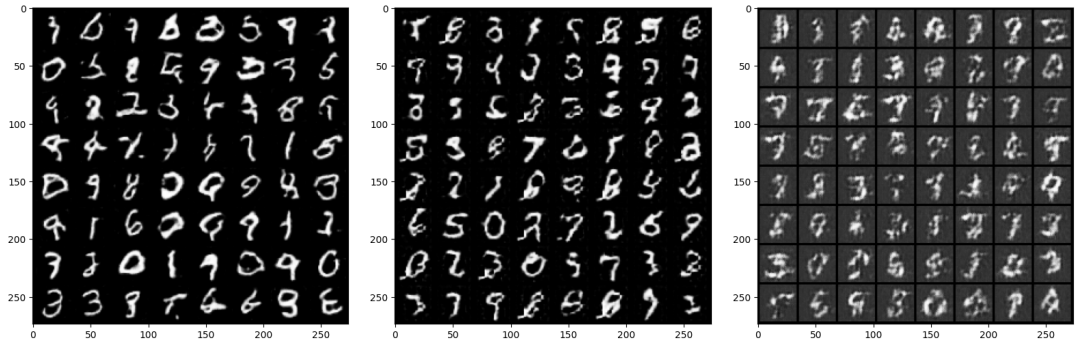
(a) lr discriminator: 0.002 (b) lr discriminator: 0.02 (c) lr discriminator: 0.00002

Figure 14: Comparing performance using different learning rates for the discriminator and a learning rate of 0.0005 for the generator



(a) lr generator: 0.005 (b) lr generator: 0.05 (c) lr generator: 0.00005

Figure 15: Comparing performance using different learning rates for the discriminator and a learning rate of 0.0005 for the generator



(a) lr generator: 0.005 (b) lr generator: 0.05 (c) lr generator: 0.00005

Figure 16: Comparing results using different learning rates for the discriminator and a learning rate of 0.0005 for the generator