RDFIA: Homework 1-a,b,c

GALMICHE Nathan PIRIOU Victor

October 14, 2022

Abstract

The goal is to produce an algorithm that is able to classify images from the dataset 15-Scenes. This algorithm belongs to the set of traditional methods that were constituting the state-of-the-art in computer vision until the end of the 2000s, before the deep learning era.

Training The training of the algorithm consists of:

1. Decomposing each of the images (of the datasets) into patches.
2. Encoding each of these patches using the SIFT algorithm.
3. Creating a visual dictionary, *i.e.* a list a recurrent patterns present in our dataset.
4. Encoding all the images of the dataset. What we have is an image of size Width × Height. What we want is an output (or an "encoding") vector whose size is equal to the number of class we can possibly find. What we have is an image of size Width × Height. What we want is an output (or an "encoding") vector whose size is equal to the number of class we can possibly find. This dimensionality reduction is done in three steps: First, from each image we deduce a set of descriptors based on the magnitude and the orientation of the gradients of the images, which is a relevant encoding since what characterizes the most an object are its contours. Second, each of the SIFT descriptors is encoded such that it is expressed in function of one or more words of the dictionary. Third, all the SIFT descriptors of each image are aggregated into a compact representation of the image, based on the frequency of detection of the features that can possibly appear in the image.
5. Training a classifier such that it learns the boundaries that separate the classes between each other.

Inference The classification of an image consists of:

1. Encoding the image in the same way that is described above.
2. Using a SVM classifier to assign an image to a class.

1 SIFT descriptor

1. **Show that kernels M_x and M_y are separable, i.e. that they can be written $M_x = h_y h_x^\top$ and $M_y = h_x h_y^\top$ with h_x and h_y two vectors of size 3 to determine.**

   We have

$$\frac{1}{2}\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \cdot \frac{1}{2}\begin{bmatrix} -1 & 0 & 1 \end{bmatrix} = \frac{1}{4}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = M_x,$$

   and

$$\frac{1}{2}\begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \cdot \frac{1}{2}\begin{bmatrix} 1 & 2 & 1 \end{bmatrix} = \frac{1}{4}\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} = M_y,$$

so $M_x = h_y h_x^\top$ and $M_y = h_x h_y^\top$ where $h_x = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$ and $h_y = \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$.

2. **Why is it useful to separate this convolution kernel?**
   The convolution product is associative.

   So, $I_x = I * M_x = I * (h_y * h_x^\top) = (I * h_y) * h_x^\top$ and $I_y = I * M_y = I * (h_x * h_y^\top) = (I * h_x) * h_y^\top$. This reduces the computational costs on an $N \times M$ image with a $m \times n$ filter from $\mathcal{O}(M \cdot N \cdot m \cdot n)$ down to $\mathcal{O}(M \cdot N \cdot (m+n))$. Here the complexity is deduced from the number of products involved in the convolution.

3. **What is the goal of the weighting by Gaussian mask?**
   In our case, we sample one 16×16 patch every 8 pixels. Hence, the sampled patches have some overlap. The weighting by a Gaussian mask gives a lower weight to the off-center pixels that belong to several samples.

   This weighting is also used because the center pixels are naturally more representative of the patch than the off-centers pixels.

4. **Explain the role of the discretization of the directions.**
   First of all, the discretization of the directions is needed because without a fixed number of directions we couldn't create the histogram. Second, this discretization increases the robustness to rotation. Indeed, if we slightly rotate a patch, we expect it to have the exact same "signature" vector than that of the original patch. The optimal trade-off between this robustness and the precision of the orientations depends on the context.

5. **Justify the interest of using the different post-processing steps.**
   If it is smaller than a threshold of 0.5, P_{enc} is set to a null vector and is immediately returned. This first thresholding is useful to desensitize the descriptor to the eventual noise of flat patches. Otherwise, the descriptor is normalized to have a unit euclidian norm to improve its robustness to global change of contrast (or "affine change in illumination") inside the image. Finally, values greater than 0.2 will be clamped to 0.2, and then the descriptor is L_2 normalized a second time. This final post-processing step allows the descriptor to deal with some non-affine change in illumination, which is desirable because two different patches that correspond to the same regions of a unique object must have exactly the same semantic representation even if the contrast is not the same in the two patches. In a nutshell, a patch just has to have a sufficiently high contrast in order to be meaningful.

6. **Explain why SIFT is a reasonable method to describe a patch of image when doing image analysis.**
   SIFT is not too computationally expensive and is a reasonable method to describe a patch with simple vectors between which it's very easy to compute a distance in order to compare them. Moreover, SIFT descriptors are robust to rotation, translation and illumination change. Finally, they just take into account the points of interest while ignoring the areas that don't provide significant information. This results in a sparse and spatial encoding.

7. **Interpret the results you got in this section.**
   It is hard to understand how a patch is built only by looking at the SIFT's descriptors. As we can see in this next image, even when the variations in the patch are not complex, it is hard to tell how the patch looks like only by peaks on the descriptor.

   Still, we can recognize breaks in the image that are constant in gradient modulus and orientation on the descriptor when peaks are of same size and evenly spaced.

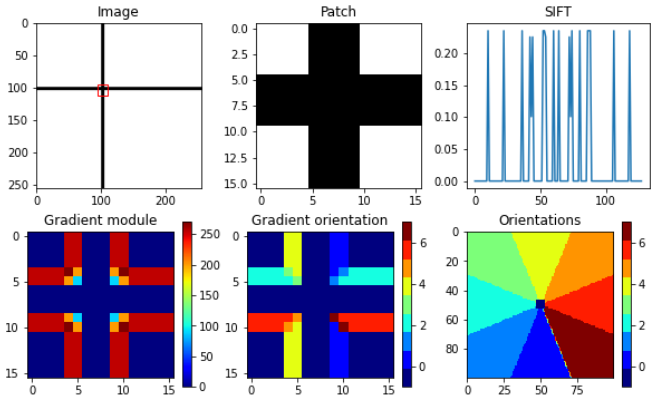
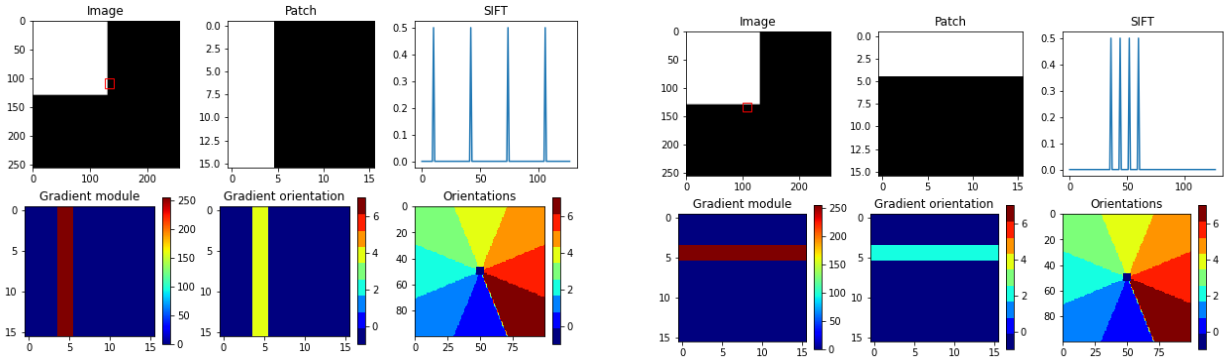


Figure (1) Graphs and SIFT descriptor obtained from a patch



(a) (b)

Figure (2) Example of patches containing a vertical break (a) and a horizontal break (b)

2 Visual dictionary

8. **Justify the need of a visual dictionary for our goal of image recognition that we are currently building.**
   We need to know what we can expect to recognize in our images. To do so, we have to do a clustering of all SIFT vectors to find words. Such a word is the descriptor that is the most representative of the descriptors belonging to a same class. Consequently, the visual dictionary is needed because the classification of a SIFT vector is done by assigning it to the corresponding class of the nearest word.

9. **Considering the points $\{x_i\}_{i=1..n}$ assigned to a cluster c, show that the cluster's center that minimize the dispersion is the barycenter (mean) of the points x_i:**

$$\text{Dispersion}(c) = \min_c \sum_i \|x_i - c\|_2^2$$

If the dispersion of the set of points belonging to the cluster c is minimized, then we have:

$$
\begin{aligned}
& \frac{\partial \text{Dispersion}(c)}{\partial c} = 0 \\
\Leftrightarrow \quad & \frac{\partial \sum_{i=1}^{n} \|x_i - c\|_2^2}{\partial c} = 0 \\
\Leftrightarrow \quad & \frac{\sum_{i=1}^{n} \partial((\sum_{k=1}^{n'} (x_{i,k} - c_k)^2)^{\frac{1}{2}})^2}{\partial c} = 0 \\
\Leftrightarrow \quad & \frac{\sum_{i=1}^{n} \partial(\sum_{k=1}^{n'} (x_{i,k} - c_k)^2)}{\partial c} = 0 \text{ where } n' \text{ is the number of components.} \\
\Leftrightarrow \quad & \frac{\sum_{i=1}^{n} \partial(x_{i,k} - c_k)^2}{\partial c_k} = 0 \\
\Leftrightarrow \quad & \sum_{i=1}^{n} -2(x_{i,k} - c_k) = 0 \\
\Leftrightarrow \quad & \sum_{i=1}^{n} x_{i,k} = c_k \sum_{i=1}^{n} \\
\Leftrightarrow \quad & \sum_{i=1}^{n} x_{i,k} = c_k \cdot n \\
\Leftrightarrow \quad & c_k = \frac{1}{n} \sum_{i=1}^{n} x_{i,k}
\end{aligned}
$$

10. **In practice, how to choose the "optimal" number of clusters?**
    First of all, we need to keep in mind that for a specific dataset there is not necessarily a unique and obvious number of possible clusters. For instance, the clustering can be hierarchical; the eventual overlap between two clusters can be defined as a cluster, etc. It is also important to mention that generally we also want the number of cluster to be as low as possible because it's computationally expensive to deal with a high number of clusters.

    To choose the optimal number of clusters, there are two possibilities: either the number of cluster is known in advance, either we need to determine it experimentally. To do so, several techniques exist. Among the most commonly used, there are: the elbow method, the silhouette coefficient and the Gap statistic. Roughly speaking, the optimal number of clusters is the smallest one such that if we increase it, we don't obtain significant decrease of the intra-cluster distances.

11. **Why do we create a visual dictionary from the SIFTs and not directly on the patches of raw image pixels?**
    The clustering must be robust to small changes and perturbations applied on the image. Therefore, if we use directly the patches of raw image pixels, the clustering would be very unstable. On the contrary, a clustering done in the SIFT domain would be much more stable since the SIFT descriptors are invariant to such transformations, as we mentioned above.

    Moreover, the dimensionality of the space in which our vectors (describing our patches) are embedded has to be as low as possible in order to minimize the computational cost.

12. **Comment the results you get.**
    Results are satisfying as the images in the clusters shown seem very similar for the most part. However, when using a low number of clusters, some groups of images show pretty different images that have in common only a general scheme (grids for example). This happens much less when we use a big number of clusters.

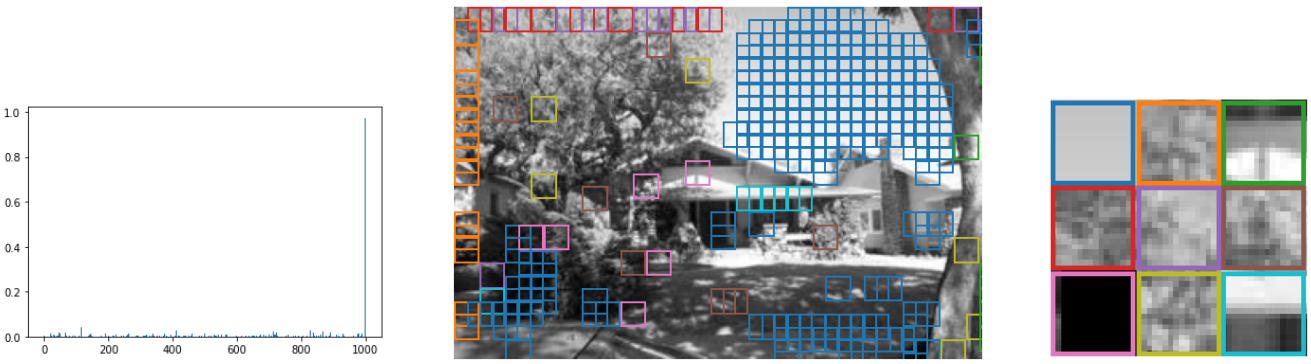
Figure (3) List of patches linked under a same word.

3 Bag of Words

13. **Concretely, what does the vector z represent of the image?**
z is an encoding of the image obtained by dividing, for each possible word, the number of times it was detected divided by the total number of patches of the image. In other words, z is a representation of the image based on the frequency of detection of the features that can possibly appear in the image.

14. **Show and discuss the visual results you got.**

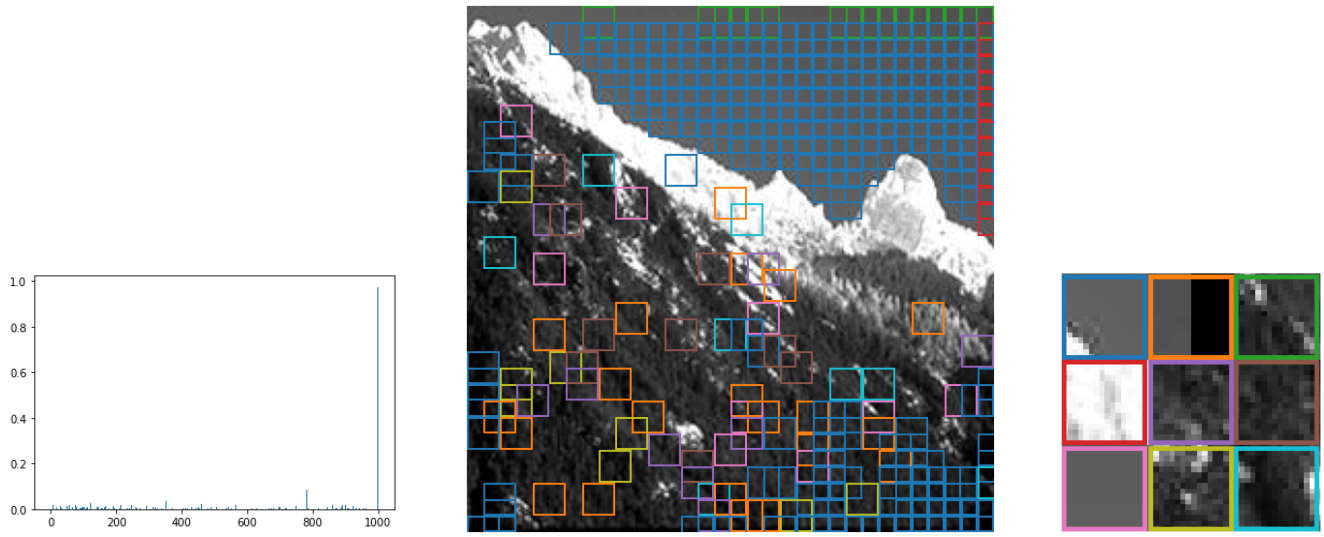
It appears our results are not perfect which is probably due to the fact that we had to limit the number of images used to reduce computation time. However, we can still analyse them.



(a) (b) (c)

Figure (4) (a) Repartition of the patches in the words, (b) an image and its most recognized words and (c) some patches

The first thing we note is that there almost always is a word that is widely found in the image. This word always corresponds to the plain patch. In most pictures we can find a plain wall, the sky or even an under/overexposed part of the image that is of constant color. As SIFT uses the gradient, we can expect those patches to be described the same way. This word corresponds to the null word we added for this purpose.



(a) (b) (c)

Figure (5) (a) Repartition of the patches in the words, (b) an image and its most recognized words and (c) some patches

As for other words, they seem to be evenly distributed. Even though some words seem pretty similar to the human eye, they are different when considering their SIFTs. As such, patches among a word are similar. Moreover, we experimented with 1000 clusters so we could expect to not find 1000 very different words.

We also note that some words are defined by the borders of the image. Those words can be then found inside the image when there is a vertical break in the patch.

15. **What is the interest of the nearest-neighbors encoding? What other encoding could we use (and why)?**
Nearest-neighbors encoding allows to assign each patch to a single word of the dictionary and then group comparable descriptors. The advantages of this encoding are the ease with which it can be implemented, and its robustness to small transformations applied to the descriptors.

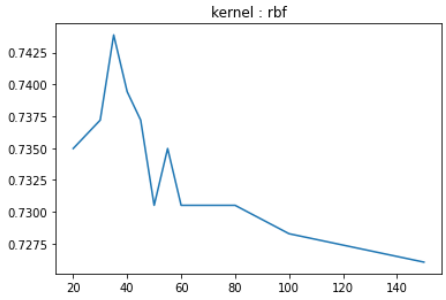
A more flexible solution would consist of doing a soft-assignment. As we saw in the fourth class of Matthieu Cord, several strategies exist (plausibility, uncertainty; cf slides 35 and 36). They are all based on the application of a kernel (typically an exponential one) on the distance between the descriptor and every centroid. Another way of doing a soft-assignment is to sparsely encode the descriptors (cf. slides 38 and 39).

16. **What the interest of the sum pooling? What other pooling could we use (and why)?**
The sum pooling keeps a lot of information in the image as we keep information on every feature in the image. Alternatively, we could do a max-pooling to just keep the feature that is the most present in the image. This would attenuate the noise effect.

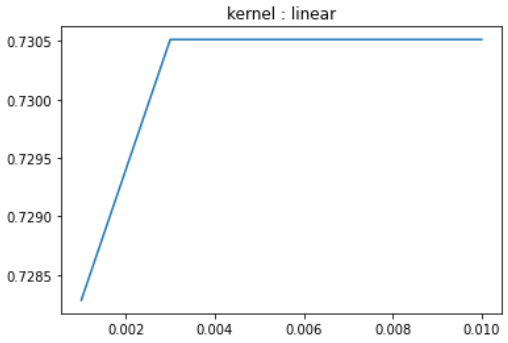
17. **What is the interest of the L_2 normalization? What other normalization could we use (and why)?**
The L_2 normalization is useful to make sure that the encoding of an image doesn't depend on the number of features present in it. We could also use the L_1 norm to limit the impact of the big values on the normalization.

4 SVM classifier

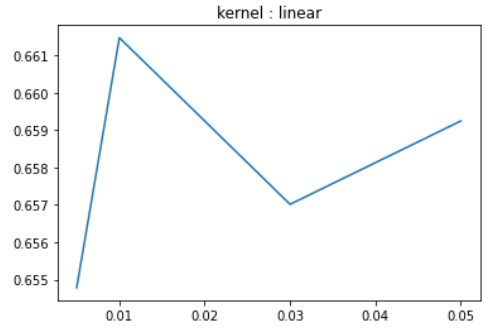
18. **Discuss the results, plot for each hyperparameters a graph with the accuracy in the y-axis.**



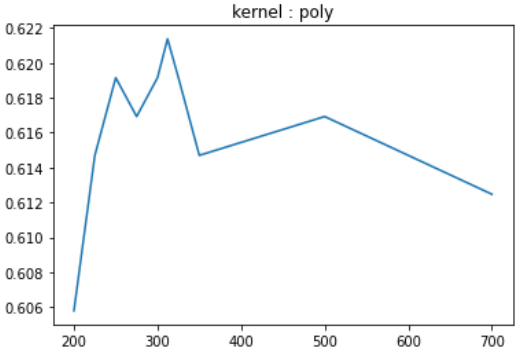
Accuracies obtained with the RBF kernel depending of C



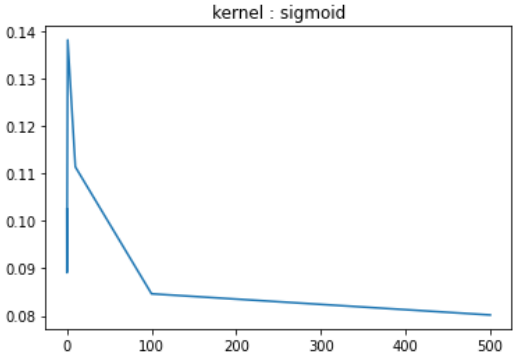
Accuracies obtained with the linear kernel depending of C



Accuracies obtained with the linear kernel but with a one vs all strategy, depending of C



Accuracies obtained with the polynomial kernel depending of C



Accuracies obtained with the Sigmoid kernel depending of C

First we tried the different kernel types. We found that for this data, the sigmoid kernel does not work at all. Whatever we set for the other hyperparameters, the score on the validation set does not go beyond 0.14.

The polynomial kernel is more interesting with scores on the validation set reaching 0.62. It is optimal when C is equal to about 312. The following graph was generated using the 'scale' kernel coefficient, which shows similar results as the 'auto' coefficient.

The second best kernel is the linear kernel using which reaches a score of 0.73. Changing C has pretty little impact as the score does not change with C greater or equal to 0.003.

Finally, the best kernel is the Radial Basis Function (RBF) kernel as its scores reach a bit more than 0.74.

All these scores were generated with the *one vs one* strategy. When we try to use the *one vs all* strategy with linear kernel, we find that the results really depend on the starting points which are chosen randomly.

19. **Explain the effect of each hyperparameter.**

The regularization parameter λ (or "C") It determines the trade-off between increasing the margin size and ensuring that the points lie on the correct side of the margin. In other words, it controls the trade off between achieving a low training error and a low testing error which is related to the ability to generalize to unseen data.

Kernel type Another hyperparameter is the kernel type. Kernels allow to transform the input space such that it becomes the most separable as possible after this transformation. Instead of computing the dot product $\langle \phi(X), \phi(W') \rangle$ of each pair of data points projected in a higher dimensional space, we just use a less costly function $K = \langle \phi(X), \phi(W') \rangle$ that is called a kernel. In *scikit-learn*, four kernels are available:

(a) The RBF is defined as $K(X, W') = \exp\left(-\gamma \cdot \|X - W'\|^2\right)$ where $\gamma > 0$ defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'. If is too large, the radius of the area of influence only includes the support vector itself. Otherwise, it is very small, the boundary is not complex enough.

(b) The linear kernel is defined as $K(X, W') = \langle X \cdot W' \rangle$. This kernel is pretty popular as it is very effective with linearly separable data.

(c) The Sigmoid kernel is defined as $K(X, W') = \tanh\left(\langle X, W' \rangle + r\right)$. It is the first kernel to use when we don't really know what is the data like as it is the most efficient with non linearly separable data.

(d) The polynomial kernel is defined as $K(X, W') = \left(\gamma \cdot \langle X, W' \rangle + r\right)^d$ where $\gamma > 0$. Intuitively, the polynomial kernel looks not only at the given features of input samples to determine their similarity, but also combinations of these. d is the degree while r is a free parameter trading off the influence of higher-order versus lower-order terms in the polynomial. The effect of γ is the same as the RBF kernel's γ.

Each kernel type is more adapted for certain data types. For example, as we saw in question 18, a Sigmoid kernel does not work for our data. In our case, the most efficient was the RBF kernel.

20. **Why the validation set is needed in addition of the test set ?**
The validation set is used to optimize the hyperparameters. We need this new set because the scores on the training set are already biased by the learning. Moreover, tuning the hyperparameters would bias the scores we get on the test set which is not desirable if we want to compare models.

RDFIA: Homework 2-a,b,c,d

GALMICHE Nathan PIRIOU Victor

November 17, 2022

Abstract

Since the ImageNet 2012 challenge, deep learning is the go-to technology for numerous and various prediction or estimation problems. When used right, deep learning-based methods are often more effective than the traditional ones that have been developed for decades to reach their efficiency (*e.g.* the SVM+BoW approach studied in the previous practical works).

However useful or pleasant to use they can be, understanding how these technologies work is quite challenging.

Deep Learning is a subset of Machine Learning and is based on artificial neural networks. The goal of such networks is to learn some parameters that best approximate a mapping between an input data and a desired output. These networks are modelled as acyclic weighted oriented graphs composed of simplified models of neurons aggregated into successive *layers* that are connected by learnable weights. Each layer allows the learnt function to be potentially more non-linear and then more complex. The architecture of the network depends on the data we want to process and the results we are looking for. For example, in this practical work we learned that convolutional neural networks are very well-suited to classify images.

Interestingly, the *Universal Approximation Theorem* states that any continuous function can be modelled by a sufficiently large single-layer perceptron. There are, however, a few difficulties with the use of an extremely wide and shallow network. The main issue is that these very wide and shallow networks are excellent at memorization, but not so good at generalization. Unlike deep neural networks that are good at generalization and able to capture the hierarchical nature of the world.

For the function to be learned, the network needs to be trained to understand the data. During the training, the possible error an Artificial Neural Network makes, when predicting the expected outcome, is measured by a *loss function* that needs to be differentiable. This function allows the problem of learning the weights of these networks to be cast as an optimization problem that seeks to be solved by iterative gradient descent methods.

It's important to mention that despite myriad of efforts, the ability of the learning to converge to a good local minima, of the possibly non-convex loss function, has never been proven.

1 Theoretical foundation

1. ★ **What are the train, val and test sets used for?**
   The train set is the part of the dataset we use to compute the optimal weights of the neural network.

   The evaluation set is used to evaluate different configurations of hyperparameters (number of layers, number of neurons, etc). It also allows to watch the model's efficiency on less biased data.

   The test set is especially dedicated to obtain an unbiased evaluation of the final model, that is done on unseen data. It allows to evaluate our final model.

2. **What is the influence of the number of examples N ?**
   The bigger N is, the better the capacity of the model to generalize will be. This means that with few examples, the model will be trained to stay close to these and won't be effective on new examples. If N is very big, the model will be trained to recognize all of them and it is more likely that new examples will be close to the ones already used for training.

3. **Why is it important to add activation functions between linear transformations?**
   An artificial neural network composed of L layers that can been seen as a combination of non-linear function (applied element-wise) compositions and matrix multiplications:

   $$f(x) := g^L(W^L g^{L-1}(W^{L-1}...g^1(W^1 x)...)),$$

   where $W^L = \{w^l_{i,j}\}$ is the weight between the j-th node in layer $l-1$ and the i-th node in layer l.

   Without such non-linear functions, f would be a succession of linear transformations, *i.e.* a linear transformation.

4. ★ **What are the sizes n_x, n_h, n_y in the figure 1?**
   In figure 1:

   - n_x is the number of neurons of the first layer which is 2.
   - n_h is the number of neurons of the hidden layer which is 4.
   - n_y is the number of neurons of the output layer which is 2. Generally, the way we have to interpret this number isn't always the same. In our case, it represents the number of class since we use the SoftMax function.

   In practice, how are these sizes chosen? In practice, n_x and n_y are respectively equal to the input data size and target size. In order to determine the hyper-parameter n_h, we need to test different values and see which one works the best. The optimal value depends on the data and the problem we want to solve.

5. **What do the vectors $\hat{y}$ and y represent ? What is the difference between these two quantities?**
   y is a one-hot encoding of the groundtruth whereas $\hat{y}$ is the output vector of the network and it represents the probability distribution of the class. The goal is to make $\hat{y}$ as close as y as possible.

6. **Why use a SoftMax function as the output activation function?**
   We use it to make the sum of our outputs equal to one. This is a condition that needs to be satisfied for the output to be a probability distribution.

7. **Write the mathematical equations allowing to perform the *forward* pass of the neural network, i.e. allowing to successively produce $\tilde{h}, h, \tilde{y}$ and $\hat{y}$ starting at x.**

   $$\begin{aligned}
   \tilde{h} &= W_h x + b_h \\
   h &= \tanh(\tilde{h}) \\
   \tilde{y} &= W_y h + b_y \\
   \hat{y} &= \text{SoftMax}(\tilde{y})
   \end{aligned}$$

There are many ways to do the forward. Here, we chose one that is different than that of *PyTorch*. That's why in questions 16/17 we don't obtain exactly the same results as the ones written in the backpropagation cheatsheet that uses the same convention as *PyTorch*.

8. **During training, we try to minimize the loss function. For cross entropy and squared error, how must the $\hat{y}_i$ vary to decrease the global loss function $\mathcal{L}$?**
Mathematically, if $\mathcal{L}$ is the cross entropy then we have:

$$\frac{\partial l(y,\hat{y})}{\partial \hat{y}_i} = \frac{\partial(-y_i \cdot \log(\hat{y}_i) - (1-y_i)\cdot\log(1-\hat{y}_i))}{\partial \hat{y}_i} = -y_i \cdot \frac{\partial \log(\hat{y}_i)}{\partial \hat{y}_i} - (1-y_i)\cdot\frac{\partial \log(1-\hat{y}_i)}{\partial \hat{y}_i} = -\frac{y_i}{\hat{y}_i} + \frac{1-y_i}{1-\hat{y}_i},$$

and, if it's the mean squared error we have:

$$\frac{\partial l(y,\hat{y})}{\partial \hat{y}_i} = \frac{\partial(\sum_i (y_i-\hat{y}_i)^2)}{\partial \hat{y}_i} = \frac{\partial (y_i-\hat{y}_i)^2}{\partial \hat{y}_i} = -2\cdot(y_i-\hat{y}_i).$$

In both cases, in order to decrease the global loss function $\mathcal{L}$, $\hat{y}_i^{(t)}$ must vary such that it moves toward $\hat{y}_i^{(t+1)}$ defined by:

$$\hat{y}_i^{(t+1)} = \hat{y}_i^{(t)} - \gamma \cdot \frac{\partial l(y,\hat{y})}{\partial \hat{y}_i},$$

where γ is the learning rate.

Finally:

- if $\hat{y}_i > y_i$ then $\frac{\partial l(y,\hat{y})}{\partial \hat{y}_i} > 0$ and $y_i < \hat{y}_i^{(t+1)} < \hat{y}_i^{(t)}$;
- else $\hat{y}_i < y_i$ then $\frac{\partial l(y,\hat{y})}{\partial \hat{y}_i} < 0$ and $y_i > \hat{y}_i^{(t+1)} > \hat{y}_i^{(t)}$;

so $\hat{y}_i$ varies towards y_i.

9. **How are these functions better suited to classification or regression tasks?**

Cross entropy loss is better for classification. Minimizing the cross entropy loss is equivalent to maximum likelihood estimation (MLE), under the label independence assumption. In MLE,

$$\begin{aligned}
f^* &= \arg\max_f \prod_{i=1}^{n_x} \Pr\left(\hat{y}_i = y_i \mid f, x_i\right) \\
&= \arg\max_f \sum_{i=1}^{n_x} \log\left[\Pr\left(\hat{y}_i = y_i \mid f, x_i\right)\right] \\
&= \arg\max_f \sum_{i=1}^{n_x}\sum_{j=1}^{n_y} \mathbb{1}_{y_i=y_j} \log\left[\Pr\left(\hat{y}_i = y_i \mid f, x_i\right)\right] \\
&= \arg\max_f \sum_{i=1}^{n_x}\sum_{j=1}^{n_y} \Pr\left(y_i = y_j \mid f, x_i\right) \log\left[\Pr\left(\hat{y}_i = y_i\right)\right] \\
&= \arg\max_f \sum_{i=1}^{n_x} -\mathrm{H}\left(Pr\left(y_i \mid f, x_i\right), \Pr\left(\hat{y}_i = y_i \mid f, x_i\right)\right) \\
&= \arg\min_f \sum_{i=1}^{n_x} \mathrm{H}\left(\Pr\left(y_i \mid f, x_i\right), \Pr\left(\hat{y}_i = y_i \mid f, x_i\right)\right) \text{ where } \mathrm{H}(y_i,\hat{y}_i) = -\sum y_i \log \hat{y}_i \text{ is the binary cross-entropy.}
\end{aligned}$$

Mean Squared Error loss is better for regression. Using this loss amounts to suppose that the samples are independent and normally distributed, which are good assumptions for regression.

$$
\begin{aligned}
f^* &= \arg\max_f \prod_{i=1}^{n_x} \Pr\left(\hat{y}_i = y_i \mid f, x_i\right) \\
&= \arg\max_f \sum_{i=1}^{n_x} \log\left[\Pr\left(\hat{y}_i = y_i \mid f, x_i\right)\right] \\
&= \arg\max_f \sum_{i=1}^{n_x} \log\left[\mathcal{N}\left(y_i \mid f\left(x_i\right), \sigma^2\right)\right] \\
&= \arg\max_f \sum_{i=1}^{n_x} \log\left[\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{\left(y_i - f\left(x_i\right)\right)^2}{2\sigma^2}}\right] \\
&= \arg\max_f \sum_{i=1}^{n_x} \left[\log\frac{1}{\sigma\sqrt{2\pi}} - \frac{1}{2\sigma^2}\left(y_i - f\left(x_i\right)\right)^2\right]^2 \\
&= \arg\max_f n_x \log\frac{1}{\sigma\sqrt{2\pi}} - \sum_{i=1}^{n_x} \frac{1}{2\sigma^2}\left(y_i - f\left(x_i\right)\right)^2 \\
&= \arg\max_f -\sum_{i=1}^{n_x}\left(y_i - f\left(x_i\right)\right)^2 \\
&= \arg\min_f \sum_{i=1}^{n_x}\left(y_i - f\left(x_i\right)\right)^2 \text{ where } f\left(x_i\right) = \hat{y}_i.
\end{aligned}
$$

10. **What seem to be the advantages and disadvantages of the various variants of gradient descent between the classic, mini-batch stochastic and online stochastic versions? Which one seems the most reasonable to use in the general case?**

    Classic gradient descent It's the more stable variation but it's the one that needs the most epochs for the convergence of the learning.

    Online stochastic gradient descent The most unstable but it's the one that needs the smallest number of epochs to converge.

    Mini-batch stochastic gradient descent It's a trade-off between the classic and the online stochastic gradient descent. It's the one that is the most reasonable to use in the general case.

11. ★ **What is the influence of the learning rate η on learning?**
    If η is lower than the optimal learning rate, it will take more epochs for the learning to converge. If it's higher, it will take less epochs at the cost of a risk of divergence as the point of convergence can be skipped by a step being too large. In short, η influences the learning speed but it has to be set cautiously.

12. ★ **Compare the complexity (depending on the number of layers in the network) of calculating the gradients of the *loss* with respect to the parameters, using the naive approach and the *backprop* algorithm.**
    The naive approach consists of directly computing the gradient with respect to each weight individually so its complexity is $\mathcal{O}(n!)$ where n is the number of layers.

    The backpropagation algorithm works by computing the gradient of the loss function with respect to each weight by the chain rule, computing the gradient one layer at a time, iterating backward from the last layer to avoid redundant calculations of intermediate terms in the chain rule so its complexity is $\mathcal{O}(n)$ which is much smaller than that of the naive approach.

13. **What criteria must the network architecture meet to allow such an optimization procedure ?**
    The network must be modelled as acyclic oriented graphs where each of its function are derivable.

14. **The function SoftMax and the *loss* of *cross-entropy* are often used together and their gradient is very simple. Show that the *loss* can be simplified by:**

$$\ell = -\sum_i y_i \tilde{y}_i + \log\left(\sum_i e^{\tilde{y}_i}\right).$$

Let's $\tilde{y}_i$ be the output of the network:

$$\hat{y}_i = \frac{\exp(\tilde{y}_i)}{\sum_{i=1}^{n_y} \exp(\tilde{y}_i)},$$

hence the cross-entropy loss is defined by:

$$\begin{aligned}
\ell(y, \hat{y}) &= -\sum_i y_i \log\left(\frac{e^{\tilde{y}_i}}{\sum_j e^{\tilde{y}_j}}\right) \\
&= -\sum_i y_i \left(\log\left(e^{\tilde{y}_i}\right) - \log\left(\sum_j e^{\tilde{y}_j}\right)\right) \\
&= -\sum_i y_i \left(\tilde{y}_i - \log\left(\sum_j e^{\tilde{y}_j}\right)\right) \\
&= -\sum_i y_i \tilde{y}_i + \sum_i y_i \log\left(\sum_j e^{\tilde{y}_j}\right)
\end{aligned}$$

Because the y_i are probabilities we have $\sum_i y_i = 1$ so we finally obtain:

$$\ell(y, \hat{y}) = -\sum_i y_i \tilde{y}_i + \log\left(\sum_i e^{\tilde{y}_i}\right).$$

15. **Write the gradient of the *loss (cross-entropy)* relative to the intermediate output $\tilde{y}$.**

$$\begin{aligned}
\frac{\partial \ell}{\partial \tilde{y}_i} &= \frac{\partial\left(-\sum_j y_j \tilde{y}_j + \log\left(\sum_k e^{\tilde{y}_k}\right)\right)}{\partial \tilde{y}_i} \\
&= -\frac{\partial\left(\sum_j y_j \tilde{y}_j\right)}{\partial \tilde{y}_i} + \frac{\partial\left(\sum_k e^{\tilde{y}_k}\right)}{\partial \tilde{y}_i} \cdot \frac{1}{\sum_k e^{y_k}} \\
&= -y_i + \frac{e^{\tilde{y}_i}}{\sum_k e^{\tilde{y}_k}} \\
&= \text{SoftMax}\left(\tilde{y}_i\right) - y_i \\
&= \hat{y}_i - y_i
\end{aligned}$$

hence :

$$\nabla_{\tilde{y}} \ell = \begin{bmatrix} \frac{\partial \ell}{\partial \tilde{y}_1} \\ \vdots \\ \frac{\partial \ell}{\partial \tilde{y}_{n_y}} \end{bmatrix} = \begin{bmatrix} \widehat{y}_1 - y_1 \\ \vdots \\ \widehat{y}_{n_y} - y_{n_y} \end{bmatrix} = \widehat{y} - y.$$

16. **Using the *backpropagation*, write the gradient of the loss with respect to the weights of the output layer $\nabla_{W_y} \ell$. Note that writing this gradient uses $\nabla_{\tilde{y}} \ell$. Do the same for $\nabla_{b_y} \ell$.**

Gradient of the loss with respect to the weights of the output layer $\nabla_{W_y}\ell$:

$$\nabla_{W_y}\ell = \begin{bmatrix} \frac{\partial \ell}{\partial W_{y,11}} & \cdots & \frac{\partial \ell}{\partial W_{y,1n_h}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \ell}{\partial W_{y,n_y1}} & \cdots & \frac{\partial \ell}{W_{y,n_yn_h}} \end{bmatrix} = \begin{bmatrix} \sum_k \frac{\partial \ell}{\partial \tilde{y}_k}\frac{\partial \tilde{y}_k}{\partial W_{y,i,j}}\Big|_{i,j=1,1} & \cdots & \sum_k \frac{\partial \ell}{\partial \tilde{y}_k}\frac{\partial \tilde{y}_k}{\partial W_{y,i,j}}\Big|_{i,j=1,n_h} \\ \vdots & \ddots & \vdots \\ \sum_k \frac{\partial \ell}{\partial \tilde{y}_k}\frac{\partial \tilde{y}_k}{\partial W_{y,i,j}}\Big|_{i,j=n_y,1} & \cdots & \sum_k \frac{\partial \ell}{\partial \tilde{y}_k}\frac{\partial \tilde{y}_k}{\partial W_{y,i,j}}\Big|_{i,j=n_y,n_h} \end{bmatrix},$$

where :

$$\frac{\partial \tilde{y}_k}{\partial W_{y,i,j}} = \frac{\partial\left(\sum_{\ell=1}^{n_h} W_{y,k,\ell}\cdot h_l + b_{y,k}\right)}{\partial W_{y,i,j}} = \frac{\partial(W_{y,i,j}\cdot h_j + b_{y,k})}{\partial W_{y,i,j}} = h_j.$$

Finally, we get :

$$\nabla_{W_y}\ell = \begin{bmatrix} \frac{\partial l}{\partial \tilde{y}_1}\cdot h_1 & \cdots & \frac{\partial l}{\partial \tilde{y}_1}\cdot h_{n_h} \\ \vdots & \ddots & \vdots \\ \frac{\partial l}{\partial \tilde{y}_{n_y}}\cdot h_1 & \cdots & \frac{\partial l}{\partial \tilde{y}_{n_y}}\cdot h_{n_h} \end{bmatrix} = \nabla_{\tilde{y}}\ell \cdot h^T.$$

Gradient of the loss with respect to the weights of the output layer $\nabla_{b_y}\ell$: The same way, we get :

$$\frac{\partial \tilde{y}_k}{\partial b_{y,i}} = \frac{\partial\left(\sum_{\ell=1}^{n_h} W_{y,k,\ell}\cdot h_l + b_{y,k}\right)}{\partial b_{y,i}} = \frac{\partial\left(\sum_{\ell=1}^{n_h} W_{y,i,j}\cdot h_l + b_{y,k}\right)}{\partial b_{y,i}} = \frac{\partial(b_{y,i})}{\partial b_{y,i}} = 1.$$

It follows that :

$$\frac{\partial \ell}{\partial b_{y,i}} = \sum_k \frac{\partial \ell}{\partial \tilde{y}_k}\frac{\partial \tilde{y}_k}{\partial b_{y,i}} = \frac{\partial \ell}{\partial \tilde{y}_i}.$$

Finally :

$$\nabla_{b_y}\ell = \nabla_{\tilde{y}}\ell.$$

17. ★ **Compute other gradients: $\nabla_{\tilde{h}}\ell, \nabla_{W_h}\ell, \nabla_{b_h}\ell$.**
Gradient$\nabla_{\tilde{h}}\ell$

$$\nabla_{\tilde{h}}\boldsymbol{\ell} = \begin{bmatrix} \frac{\partial \ell}{\partial \tilde{h}_1} \\ \vdots \\ \frac{\partial l}{\partial \tilde{h}_{n_h}} \end{bmatrix} = \begin{bmatrix} \sum_k \frac{\partial \ell}{\partial h_k}\frac{\partial h_k}{\partial \tilde{h}_i}\Big|_{i=1} \\ \vdots \\ \sum_k \frac{\partial l}{\partial h_k}\frac{\partial h_k}{\partial \tilde{h}_i}\Big|_{i=n_h} \end{bmatrix},$$

where:

- $\frac{\partial h_k}{\partial \tilde{h}_i} = \frac{\partial(\tanh(\tilde{h}_k))}{\partial \tilde{h}_i} = \frac{\partial(\tanh(\tilde{h}_k))}{\partial \tilde{h}_i} = \begin{cases} 1-\tanh^2\left(\tilde{h}_k\right) = 1-\tanh^2\left(\tanh^{-1}(h_k)\right) = 1-h_i^2 & \text{if } k=i \\ 0 & \text{otherwise} \end{cases}$,
- $\frac{\partial \ell}{\partial h_i} = \sum_k \frac{\partial \ell}{\partial \tilde{y}_k}\frac{\partial \tilde{y}_k}{\partial h_i} = \sum_k \frac{\partial \ell}{\partial \tilde{y}_k}\frac{\partial\left(\sum_{\ell=1}^{n_h} W_{y,k,\ell}\cdot h_\ell + b_{y,k}\right)}{\partial h_i} = \sum_k \frac{\partial \ell}{\partial \tilde{y}_k}\frac{\partial(W_{y,k,i}\cdot h_i + b_{y,k})}{\partial h_i} = \sum_k \frac{\partial \ell}{\partial \tilde{y}_k}W_{y,k,i}.$

Finally, we get:

$$\nabla_{\tilde{h}}\ell = \begin{bmatrix} \frac{\partial \ell}{\partial \tilde{h}_1} \\ \vdots \\ \frac{\partial \ell}{\partial \tilde{h}_{n_h}} \end{bmatrix} = \begin{bmatrix} \left(1-h_1^2\right)\sum_k \frac{\partial \ell}{\partial \tilde{y}_k}\cdot W_{y,k,1} \\ \vdots \\ \left(1-h_{n_h}{}^2\right)\sum_k \frac{\partial \ell}{\partial \tilde{y}_k}\cdot W_{y,k,n_h} \end{bmatrix} = W_y^T\cdot\nabla_{\tilde{y}}\ell \odot \left(1-h^2\right).$$

Gradient $\nabla_{W_h}\ell$

$$\nabla_{W_h}\ell = \begin{bmatrix} \frac{\partial \ell}{\partial W_{h,1,1}} & \cdots & \frac{\partial \ell}{\partial W_{h,1,n_x}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \ell}{\partial W_{h,n_h,1}} & \cdots & \frac{\partial \ell}{W_{h,n_h,n_x}} \end{bmatrix} = \begin{bmatrix} \sum_k \frac{\partial \ell}{\partial \tilde{h}_k}\frac{\partial \tilde{h}_k}{\partial W_{h,i,j}}\Big|_{i,j=1,1} & \cdots & \sum_k \frac{\partial \ell}{\partial \tilde{h}_k}\frac{\partial \tilde{h}_k}{\partial W_{h,i,j}}\Big|_{i,j=1,n_x} \\ \vdots & \ddots & \vdots \\ \sum_k \frac{\partial \ell}{\partial \tilde{h}_k}\frac{\partial \tilde{h}_k}{\partial W_{h,i,j}}\Big|_{i,j=n_h,1} & \cdots & \sum_k \frac{\partial \ell}{\partial \tilde{h}_k}\frac{\partial \tilde{h}_k}{\partial W_{h,i,j}}\Big|_{i,j=n_h,n_x} \end{bmatrix}$$

where $\sum_k \frac{\partial \ell}{\partial \tilde{h}_k}\frac{\partial \tilde{h}_k}{\partial W_{h,i,j}} = \sum_k \frac{\partial \ell}{\partial \tilde{h}_k}\frac{\partial\left(\sum_{\ell=1}^{n_x} W_{h,k,\ell}\cdot x_\ell + b_{h,k}\right)}{\partial W_{h,i,j}} = \sum_k \frac{\partial \ell}{\partial \tilde{h}_k}\frac{\partial(W_{h,k,j}\cdot x_j + b_{h,k})}{\partial W_{h,i,j}} = \frac{\partial \ell}{\partial \tilde{h}_i}\frac{\partial(W_{h,i,j}\cdot x_j + b_{h,i})}{\partial W_{h,i,j}} = \frac{\partial \ell}{\partial \tilde{h}_i}\cdot x_j.$

Finally, we get:

$$\nabla_{W_h}\ell = \nabla_{\tilde{h}}\ell \cdot x^T.$$

Gradient $\nabla_{b_h}\ell$

$$\nabla_{b_h}\ell = \begin{bmatrix} \frac{\partial \ell}{\partial b_{h,1}} \\ \vdots \\ \frac{\partial \ell}{\partial b_{h,n_h}} \end{bmatrix} = \begin{bmatrix} \sum_k \frac{\partial l}{\partial \tilde{h}_k}\frac{\partial \tilde{h}_k}{\partial b_{h,i}}\Big|_{i=1} \\ \vdots \\ \sum_k \frac{\partial l}{\partial \tilde{h}_k}\frac{\partial \tilde{h}_k}{\partial b_{h,i}}\Big|_{i=n_h} \end{bmatrix}$$

where $\sum_k \frac{\partial \ell}{\partial \tilde{h}_k}\frac{\partial \tilde{h}_k}{\partial b_{h,i}} = \sum_k \frac{\partial \ell}{\partial \tilde{h}_k}\frac{\partial\left(\sum_{\ell=1}^{n_x} W_{h,k,\ell}\cdot x_\ell + b_{h,k}\right)}{\partial b_{h,i}} = \frac{\partial \ell}{\partial \tilde{h}_i}\frac{\partial\left(\sum_{\ell=1}^{n_x} W_{h,i,\ell}\cdot x_\ell + b_{h,i}\right)}{\partial b_{h,i}} = \frac{\partial \ell}{\partial \tilde{h}_i}\frac{\partial\left(\sum_{\ell=1}^{n_x} W_{h,i,\ell}\cdot x_\ell + b_{h,i}\right)}{\partial b_{h,i}} = \frac{\partial \ell}{\partial \tilde{h}_i}.$

Finally, we get:

$$\nabla_{b_h}\ell = \nabla_{\tilde{h}}\ell.$$

2 Implementation

2.1 Neural networks

1. **Discuss and analyze your experiments following the implementation. Provide pertinent figures showing the evolution of the loss; effects of different batch size / learning rate, etc.**

We experimented on the neural networks by training on a dataset made of two classes, with the first encircling the second. We trained four different networks, each made differently. They should be very similar as the architectures are the same and the only difference is the way we implemented them. We manually implemented everything in the first one, we used PyTorch's autograd for the backward pass in the second one, we replaced the forward pass by PyTorch's torch.nn in the third one and finally in the fourth one we simplified the SGD with PyTorch's optimizer. Here is how they compare.
We can see that they all behave a little differently during training. However it is important to first note that the results are very similar as all four networks have a final accuracy of around 94%, which was expected since the architectures are the same. The aspect of the curves slightly differs from one implementation to the other because of the random initialization of the weights.
In the experiments above we used a learning rate of 0.07. This number can be increased a little bit to make our models converge faster. However, if we set it too high there will be a risk of divergence. As we can see in the figure

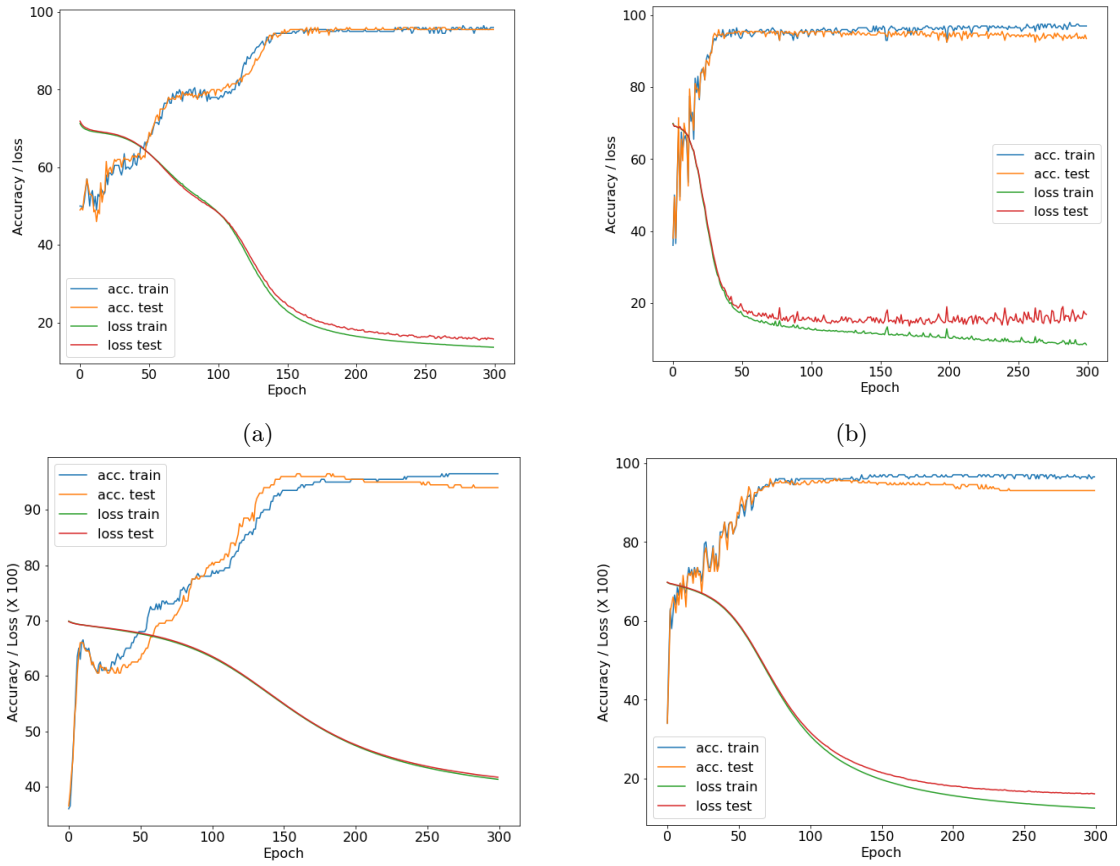


Figure (1) Graphs illustrating the evolution of the loss and the accuracy during the training. (a) was a fully implemented neural network, (b) was a network using autograd, (c) used autograd and torch.nn and (d) used autograd, torch.nn and optimizer.

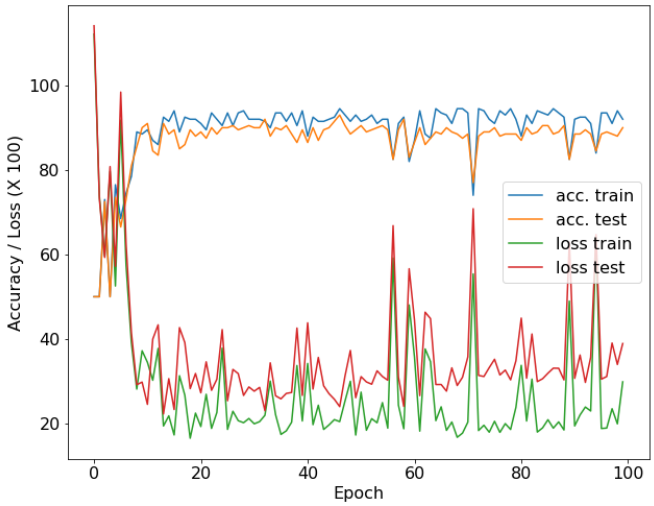
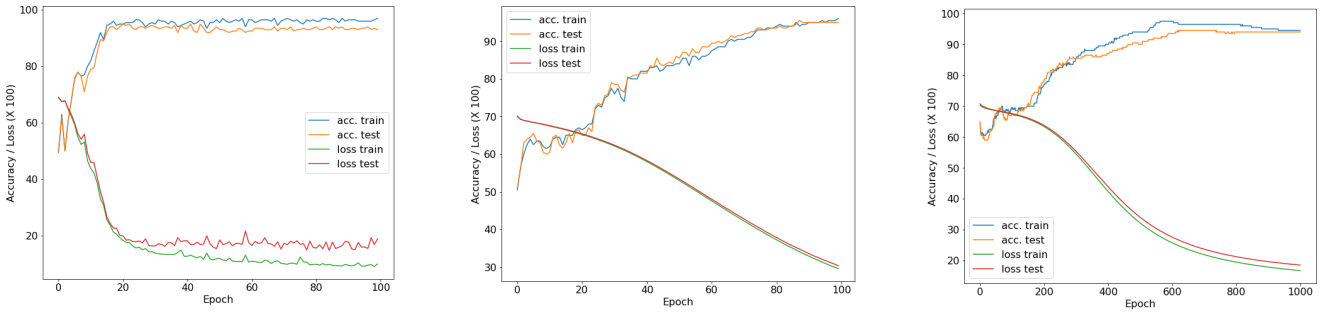


Figure (2) Evolution of the loss and accuracy of a neural network trained with a learning rate of 2.

2, even if the network reaches pretty good scores on the test set, we can not know if it is optimal as it is very unstable.

Finally, we experimented on the batch size. It allows to choose the number of data the networks look at before

updating its weights. On figure 3, we can see the results we obtained when using a small batch size, a reasonable one (which we used for the previous trainings) and a large one.



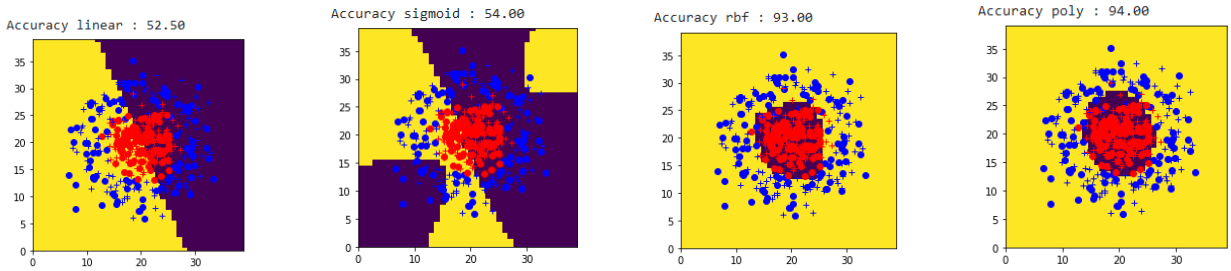
(a) (b) (c)

Figure (3) Evolution of the loss and accuracy of a neural network trained with a batch size of 1 (a), 10 (b) and 50 (c)

We see that using a batch of 50 takes very long to converge and using a batch of 1 is very fast. This was expected because with 50 data per batch, the weights are updated only a few times per epoch. A lot of epochs are thus required in order to update the weights enough time for it to converge.
On the other hand, we see that using a batch size of 1 results in pretty unstable losses and accuracies. This is also expected because the weights are updated with every data which means that the updates are very poorly generalized. The update fits only the data seen and not the rest. Even though using a low batch size can make the training faster as we said, it is also possible that computing the gradient on only one example does not give a good estimation of the gradient thus resulting in divergence or a slower convergence.
In the end, using a batch size of 10 seems like a good trade-off in order for the model to generalize well while still converging fast enough.

2.2 SVM

We wanted to predict the classes of the data from the test set of the circles dataset. We tried different kernel types and the first thing we note is that the linear and sigmoid kernels are not able to predict the classes. This was expected since one class circles the other. It is thus impossible to separate the classes linearly or with a sigmoid function. Also the polynomial kernel is only effective when its degree is even. Otherwise, the results are similar to the sigmoid function as it is not able to circle data.



(a) (b) (c) (d)

Figure (4) Separation of the data using different kernels. (a) linear kernel with C=0.01, (b) sigmoïd kernel with C=0.1, (c) RBF kernel with C=0.01 and (d) polynomial kernel of degree 2 and with C=100.

We will focus on both the RBF kernel and the polynomial kernel with a degree of 2.

```
[0.001, 0.01, 0.1, 1, 10, 100]
scores rbf : [0.95, 0.95, 0.95, 0.94, 0.935, 0.93]
scores polynomial : [0.535, 0.745, 0.955, 0.955, 0.945, 0.94]
```

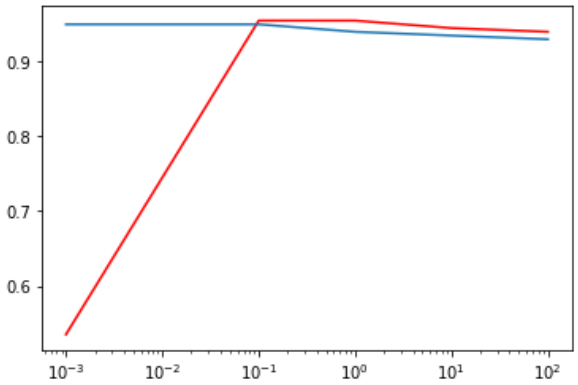


Figure (5) Scores obtained with different values of C. red=poly, blue=rbf.

We can see that both kernel types are very effective and reach a precision up to 95.5% for the polynomial kernel and 95% for the RBF kernel.
As the data are already pretty well separated, the influence of C is lesser. We can see it with the RBF that stays effective whatever the value of C. The polynomial kernel still catches some of the outlying values when the value of C is lower and is less effective.

3 Introduction to convolutional networks

1. **Considering a single convolution filter of padding p, stride s and kernel size k, for an input of size $x \times y \times z$ what will be the output size?**
   The output size will be $x' \times y' \times z'$ where:

   - $x' = \left\lfloor \frac{x-(k-1)-1+2p}{s} \right\rfloor + 1 = \left\lfloor \frac{x-k+2p}{s} \right\rfloor + 1$
   - $y' = \left\lfloor \frac{y-(k-1)-1+2p}{s} \right\rfloor + 1 = \left\lfloor \frac{y-k+2p}{s} \right\rfloor + 1$

   How much weight is there to learn ?
   There will be $(k \times k) \times z$ weights to learn.

   How much weight would it have taken to learn if a fully-connected layer were to produce an output of the same size ?
   If a fully-connected layer were to produce an output of the same size, it would have needed $(x \times y \times z) \times 2 \times (x' \times y' \times z)$ because there are two learnable parameters per connection.

2. ★ **What are the advantages of convolution over fully-connected layers ?**
   Using fully-connected layers to process an image wouldn't be a good idea because fully-connected layers ignore the topology of the input data so they are not robust and invariant to small distortions of the image. In addition, the number of weights grows largely with the size of the input image.

   Whereas convolution layers don't have these problems. Indeed, they use shared weights, which allow them to be translation equivariant (*i.e.* translating the input to a convolutional layer will result in translating the output) and to require a dramatically reduced number of parameters. In other words, by outputting 2D feature maps that keep spatial information, they provide a more stable representation of the images.

   What is its main limit ?
   The attention process is done locally, requiring a big number of layers in order to obtain a global assessment of the image.

3. ★ **Why do we use spatial pooling?**
   After each convolutional layer we use spatial pooling in order to add a small amount of translational invariance. This is interesting because in multiple fields such as object recognition where an object could be distorted or more or less big (making its features farther apart), spatial pooling allows to account for multiple locations at the same time making the model resistant to such changes. It's also useful to reduce the size of the feature maps without needing any parameter.

4. ★ **Suppose we try to compute the output of a classical convolutional network (for example the one in Figure 2) for an input image larger than the initially planned size (224×224 in the example). Can we (without modifying the image) use all or part of the layers of the network on this image ?**
   The convolutional as well as the pooling layers can be applied on any size. On the contrary, the input size of a fully-connected layers depends on its number of connexions so it's fixed. Consequently, this larger image can only be processed by the convolutional network until the last layer preceding the first fully-connected layer.

5. **Show that we can analyze fully-connected layers as particular convolutions.**
   The output of a neuron in a fully-connected layer is simply the sum of a linear combination of the elements of the receptive field and a bias. Therefore, the linear combination can be replaced by the result of a convolution applied on the receptive field with a filter whose size is equal to the number of element of the receptive field.

6. **Suppose that we therefore replace fully-connected by their equivalent in convolutions, answer again the question 4. If we can calculate the output, what is its shape and interest ?**
   It still is not possible to use all of the layers of the network on this image. Indeed, the SoftMax function takes as input as many variables as there are classes. However, since the input image is larger than the initially planned size, instead of just having variables, we still have feature maps, of size $(r+1) \times (c+1)$ where a and b are respectively the number of rows and columns added. The interest is that we can apply a Global Average Pooling (GAP) on these feature maps in order to reduce each of these maps to its average value. Once we do that, we are able to process images larger than the initially planned size. This is a much smarter approach than the approach that consists of sliding a window on the whole image to detect the object we are interested in. Moreover, this technique using GAP can be used to do localize an object in an image in a weakly supervised manner.

7. **We call the *receptive field* of a neuron the set of pixels of the image on which the output of this neuron depends. What are the sizes of the receptive fields of the neurons of the first and second convolutional layers ?**
   The size of the receptive fields of the neurons of the first convolutional layer is k_1 where k_1 is the kernel size of the first layer. The size of the second one is $(k_1 + (k_2 - 1) \times s_1)^2$ where k_2 is the kernel size of the second layer, s_1 the stride of the first layer and s_2 the stride of the second one. If $k_1 = k_2 = 3$ and $s_1 = 1$ then the size of the first receptive field is 9 and that of the second one is 25.

   Can you imagine what happens to the deeper layers ? How to interpret it ?
   The deeper the layer is, the larger its receptive field is and the more global and then abstract features it's able to detect. For instance, elementary local features detected in the first layers can be lines whereas the more global and abstract features detected in the last layers can be faces or cars.

4 Training from *scratch* of the model

8. **For convolutions, we want to keep the same spatial dimensions at the output as at the input. What padding and stride values are needed ?**
   We have to solve $\left\lfloor \frac{x-k+2p}{s} \right\rfloor + 1 = x$ and $\left\lfloor \frac{y-k+2p}{s} \right\rfloor + 1 = y$ for p and s. Since we don't want to reduce the spatial dimensions, we can choose the smallest possible stride size, which is 1. Now, given the fact that $x - k + 2p$ and $y - k + 2p$ are integers, these equations are easily solvable and we get $p = \frac{k-1}{2}$.

9. **For max poolings, we want to reduce the spatial dimensions by a factor of 2. What padding and stride values are needed ?**
   We want to reduce the spatial dimensions by a factor of 2 while minimizing the loss of information so we don't

want any padding. Let's choose it equal to 0. Now, we have to solve $\left\lfloor \frac{x-k}{s} \right\rfloor + 1 = \frac{x}{2}$ and $\left\lfloor \frac{y-k}{s} \right\rfloor + 1 = \frac{y}{2}$. The solutions are $s = k = 2$.

10. ★ **For each layer, indicate the output size and the number of weights to learn. Comment on this repartition.**
The input size of the dataset is $32 \times 32 \times 3$. Besides, it's assumed that:

    - every filter/neuron has a bias to learn;
    - the spatial dimensions of the output are the same as that of the input.

layer	**description**	**output size**	**number of weights to learn**
conv1	32 convolutions 5×5, followed by ReLU	$32 \times 32 \times 32$	$32 \times [(5 \times 5 \times 3) + 1] = 2,432$
pool1	max-pooling 2×2	$16 \times 16 \times 32$	0
conv2	64 convolutions 5×5, followed by ReLU	$16 \times 16 \times 64$	$64 \times [(5 \times 5 \times 32) + 1] = 51,264$
pool2	max-pooling 2×2	$8 \times 8 \times 64$	0
conv3	64 convolutions 5×5, followed by ReLU	$8 \times 8 \times 64$	$64 \times [(5 \times 5 \times 64) + 1] = 102,464$
pool3	max-pooling 2×2	$4 \times 4 \times 64$	0
fc4	fully-connected, 1000 output neurons, followed by ReLU	1000	$1000 \times [(4 \times 4 \times 64) + 1] = 1,025,000$
fc5	fully-connected, 10 output neurons, followed by SoftMax	10	$10 \times (1000 + 1) = 10,010$

11. **What is the total number of weights to learn ? Compare that to the number of examples.**
We have 50,000 examples and a model with 1,191,170 parameters. The fact that the number of weights to learn is much bigger than the number of training examples is positively correlated to the risk of overfitting.

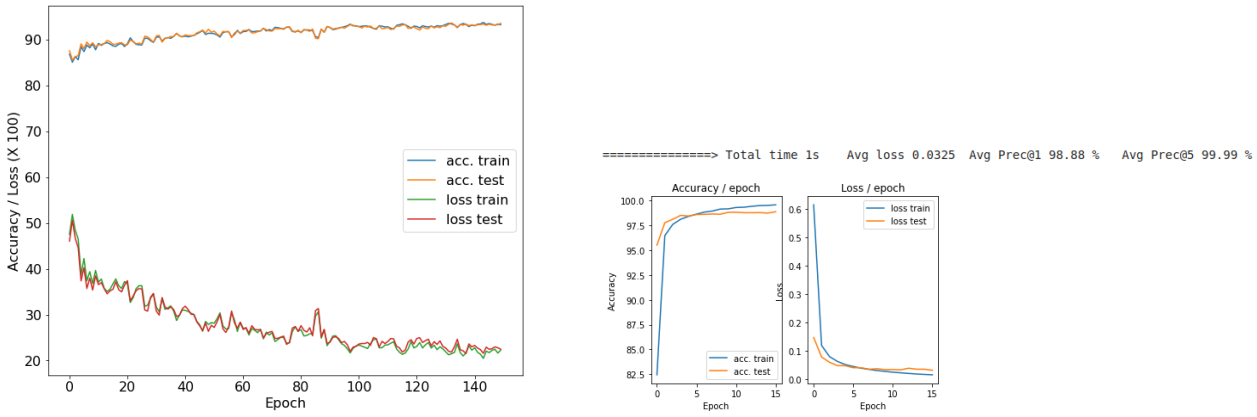
12. **Compare the number of parameters to learn with that of the BoW and SVM approach.**
The BoW+SVM approach requires one SVM for each of the 10 classes we have and each each of these SVM has a number of learnable parameters equal to the number of words from which the input data are encoded. This number was equal to 1000 in the last practical work so in total this approach mainly requires 10000 parameters which is more than 1000 times smaller than the total number of parameter of the convolutional network approach. However, the latter has the advantages to allow an end-to-end learning (of features) whereas the Bow+SVM approach relies on handcrafted features and each of its steps (extraction of features, clustering, SVM classification) need to be processed separately, which is tedious.

13. **Read and test the code provided.**
It takes only a few epochs for the convolutional neural network to stabilize and reach much higher scores than the network from part 1. The fully connected layers network reached scores up to 94.9% while the convolutional network reached 98.9%.

14. ★ **In the provided code, what is the major difference between the way to calculate loss and accuracy in train and in test (other than the difference in data) ?**
The same functions are called to calculate the loss and accuracy in train and in test. The main difference is the moment at which they are called. In fact, the loss and accuracy in train are generated before test because during the training the weights are modified. The loss in train is used to calculate the gradients and optimize the network's weights while the loss in test is simply used to evaluate the model. We thus generate the loss in test after we generate loss in train because the loss in test allows to evaluate how effectively were the weights modified.

15. **Modify the code to use the CIFAR-10 dataset and implement the architecture requested above.**

    We can see that our convolutional model is not very effective as it reaches a maximum accuracy of around



(a) (b)

Figure (6) Scores reached by the fully connected layers network (a) and the convolutional network (b).

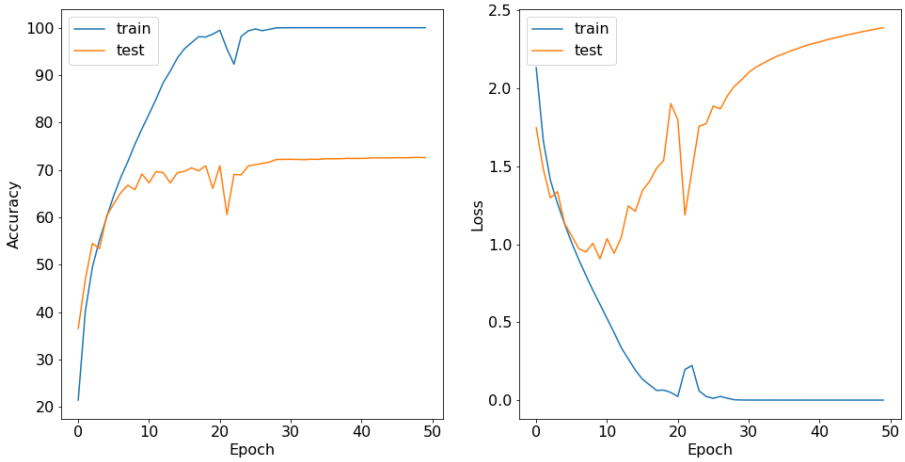


Figure (7) Evolution of the loss and the accuracy during the training of the convolutional network on the CIFAR10 dataset.

70%. The accuracy does not improve because the network manages to be perfect on the training set and thus can't learn anymore. There are two reasons that could explain why the network manages to be so perfect on the training set and not on the test set. The first one is that the data might be too "easy" in the sense the objects are too identifiable on the training images and they are too different from one class to the other. The second reason is simply that there is not enough data in the training set. The consequence is that the model will have few examples for each class and it will not be able to generalize when seeing the data from the test set.

16. ★ **What are the effects of the *learning rate* and of the *batch-size* ?**
The smaller the batch-size, the more we can parallelize! When we lower the batch size, we reduce even more the network's ability to generalize. As a result, the loss is even more unstable and the accuracy stays pretty bad. Even on the training's loss we see that the network struggles to generalize and we see the loss increase. When we set it too high, We see that the model takes way too long to converge. Moreover, it generalizes too much and accuracies reached are not really interesting.
As for the learning rate, and as we discussed earlier in this report, a big learning rate keeps the network from converging so we need to not set it too high in order for the training to be fast enough while still converging.

17. **What is the error at the start of the first epoch, in train and test ? How can you interpret this ?**
At the start of the first epoch, the accuracy is around 10%. This was expected since there are 10 classes so if they are picked randomly there is one chance out of 10 that it will be correct which is 10%.

18. ★ **Interpret the results. What's wrong ? What is this phenomenon ?**

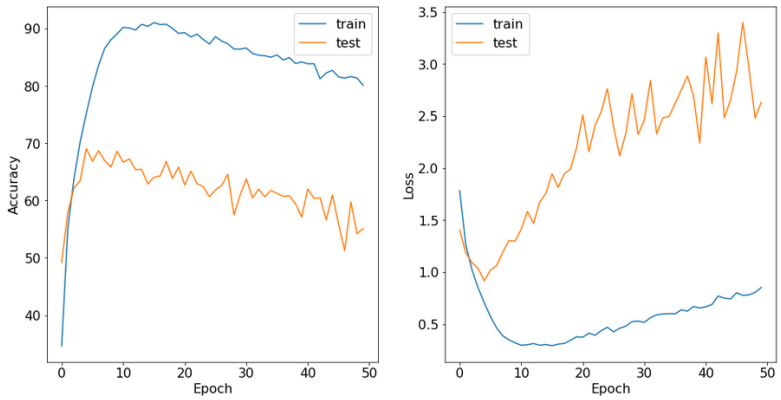


Figure (8) Evolution of the loss and accuracy during a training with a batch size set to 32.

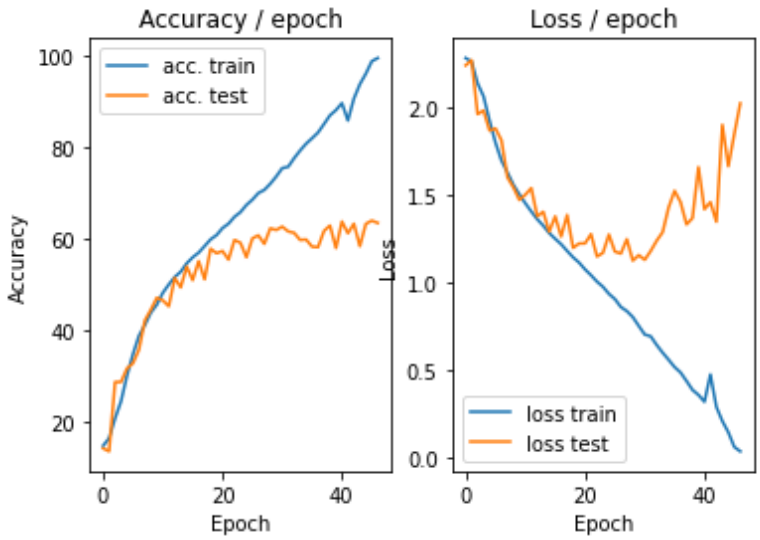


Figure (9) Evolution of the loss and accuracy during a training with a batch size set to 512.

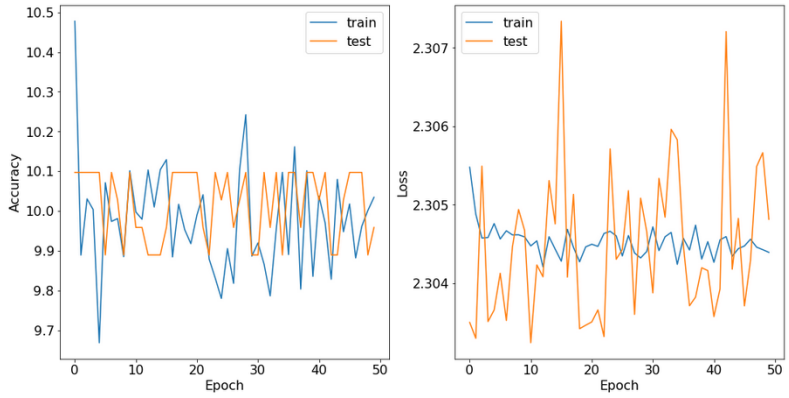


Figure (10) Evolution of the loss and accuracy during a training with a learning rate of 1.

As we said above, the results are not great. We mentioned that the network achieves perfect accuracy on the training set. Also the loss on the test set starts to increase around the tenth epoch. This means that our network overfits on the training set. It is too precise on the the images from this set and it is not able to generalize on the other images such as the ones in the test set.

5 Results improvements

★ For this part, in the end of session report, indicate the methods you have successfully implemented and for each one explain in one sentence its principle and why it improves learning.

19. **Describe your experimental results.**
The first effect of normalization we notice is that the network converges faster. We can also see an improvement in the accuracies as the network now reaches an accuracy of 75.8% compared to the previous 70%. These improvements can be explained because we 'evened' the data. In fact, the learning could be disturbed by data with extreme values in its calculations. Normalizing allows to smooth these values for more meaningful processing.
However the data sets stay similar and the learning follows a similar path as before, resulting again in overfitting.

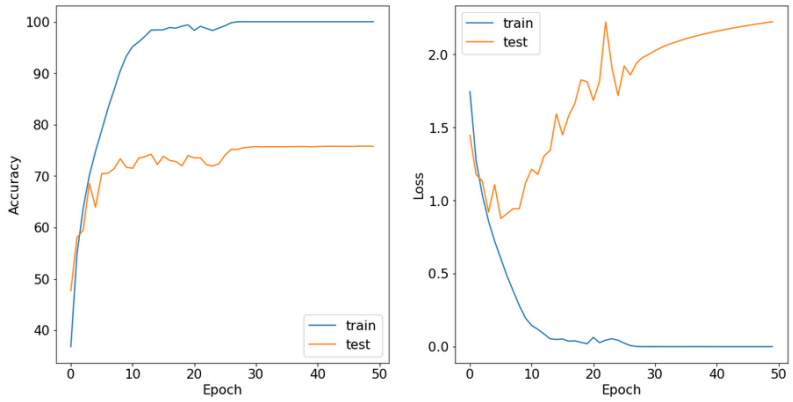


Figure (11) Performance of the network when we normalize the data.

20. **Why only calculate the average image on the training examples and normalize the validation examples with the same image ?**
First, it's important to note that the normalization of the validation examples is needed because they need to be in the same condition that the normalized examples on which the network is trained.

Computing the average image from the test or validation set would result in a bias so we use the average image of the training set. It makes sense to do that because we assumed that the distribution of the training examples is representative of that of the test set.

21. **BONUS : There are other normalization schemes that can be more efficient like ZCA normalization. Try other methods, explain the differences and compare them to the one requested.**
Among the other existing normalization schemes, we can mention Principal Component Analysis (PCA) and Zero-phase Component Analysis (ZCA).

PCA In order to normalize images with PCA, we first need to pre-process the dataset such that each pixel has a zero mean and unit variance. However, in natural images a pixel is far from being independent from its nearby pixels. Consequently, decorrelating the pixels by projecting them into the eigenbasis and keeping only the top few eigenvectors would completely destroy the natural spatial-correlation.

ZCA ZCA is a kind of whitening transformation, *i.e.* a linear transformation that transforms a vector of random variables with a known covariance matrix into a set of new variables whose covariance is the identity matrix. Therefore, if we pre-process the images with ZCA, only the linear dependencies between pixels will be removed. This implies that the pre-processed images will still be efficiently recognizable by the convolutional network since it uses local attention in the first layers.

22. **Describe your experimental results and compare them to previous results.**
Again, our results are improving. Adding crops and horizontal symmetries allows to reach an accuracy of

about 80%. The main reason for this improvement is the adding of new data. The more a network is given data to be trained on, the better it will be. Here we add data by modifying the one we already have which proves to be effective. Of course, having more data to train on means that the training will take longer and we in fact see that the model converges later than before (around the twentieth epoch against the tenth fro the previous training). It is not as obvious as before but the model overfits again starting around the twentieth epoch. This is because the network does not manage to be perfect on the training set, simply because there is more data which might even be harder for the model since some of the added data is flipped.

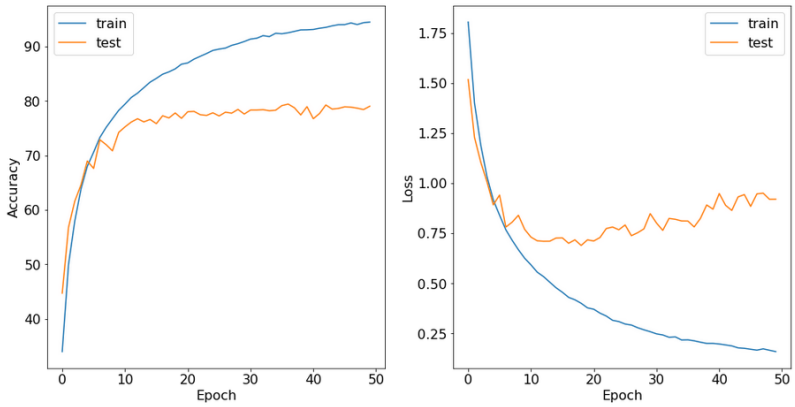


Figure (12) Network's perfromance when trained with cropping and flipping augmentations.

23. **Does this horizontal symmetry approach seems usable on all types of images ?**
No because this horizontal flipping can change the meaning of the images.

In what cases can it be or not be ?
For instance, if we want to train a network to detect all biological cells in an image then this approach is relevant because a flipped cell still looks like a cell.

On the contrary, if we want to train a network to classify the letter present in an image, into one of the 26 classes of the possible letters, using this horizontal symmetry approach would worsen the result. For example, a flipped image containing the letter "M" is an image that seems to contain a "W".

24. **What limits do you see in this type of data increase by transformation of the dataset ?**
Identifying an optimal data augmentation strategy is difficult.

Some more sophisticated data augmentation strategies exist but they are tricky to implement because for instance they require the use of Generative Adversarial Networks.

If the dataset is already large, the learning of the augmented one will require additional time and power.

25. **Bonus : Other data augmentation methods are possible. Find out which ones and test some.**
Among the numerous data augmentation methods available in PyTorch, we can mention:

- **ColorJitter** Randomly change the brightness, contrast, saturation and hue of an image; The problem with changing an image's hue is that it changes objects' color which is one of the aspects that allow to identify them. That is what happened in our experiment, explaining the poor performance.
The intensity and the contrast usually are very interesting to change in order to augment our data. In fact, an object captured in different configurations usually appears in different intensities and contrasts on the images. In our case, we suspect that the images are too small and changes in intensity and contrast are too powerful on the few pixels of the images. As a result, the network performs better than when we modify hue but much worse than when we only crop and flip.
- **RandomAffine** Random affine transformation of the image keeping center invariant; Much like modifying the hue of an image, affine transformations can be dangerous for object recognition. An object can be identifiable depending on its orientation. It is thus no surprise to see our model perform pretty badly with this augmentation.

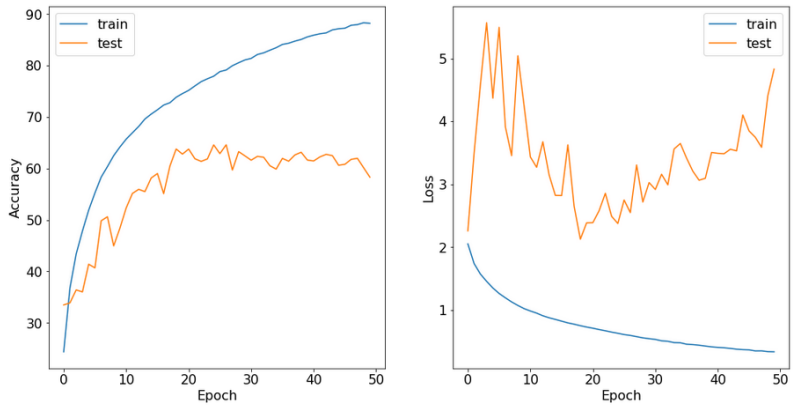


Figure (13) Network's performance when trained with a modification of intensity and hue.

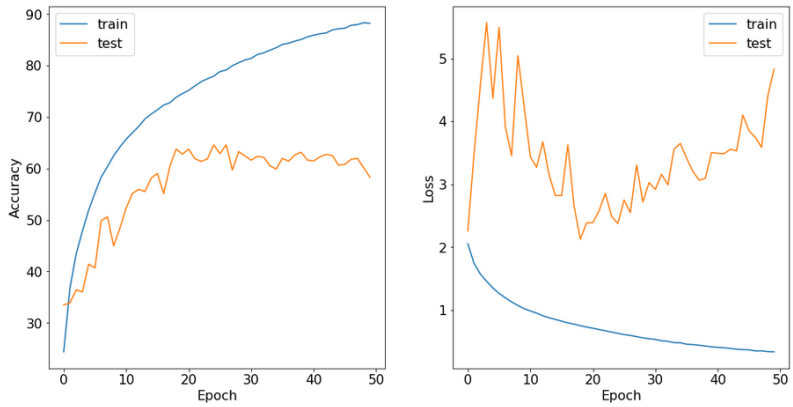


Figure (14) Network's performance when trained with a modification of intensity and contrast.

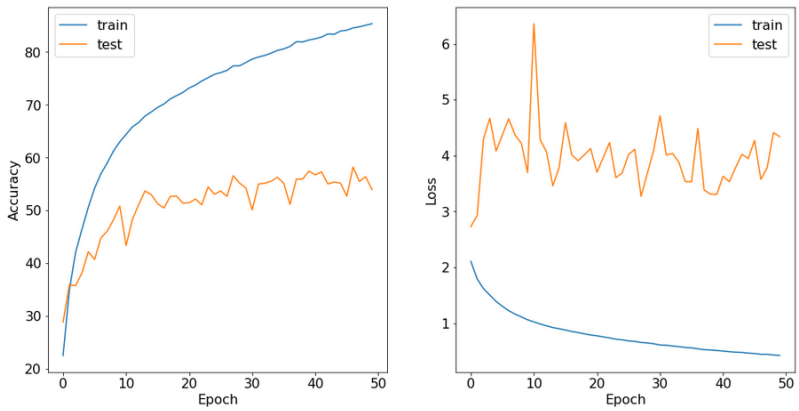


Figure (15) Network's performance when trained with a affine transformation augmentation.

- **RandomPerspective** Performs a random perspective transformation of the given image with a given probability. This augmentation is interesting because objects often look different when we look at them from another perspective. Simulating this by modifying the original images can help the network generalize better. In fact this augmentation is actually the best one we tested in this question. The best scores on the test set are similar to what we obtained with only the crop and flip augmentations.
- **RandomRotation** Rotate the image by angle; It can already be done in the RandomAffine augmentation mentionned above.
- **RandomVerticalFlip** Vertically flip the given image randomly with a given probability; It is similar to the horizontal symmetry we already performed.

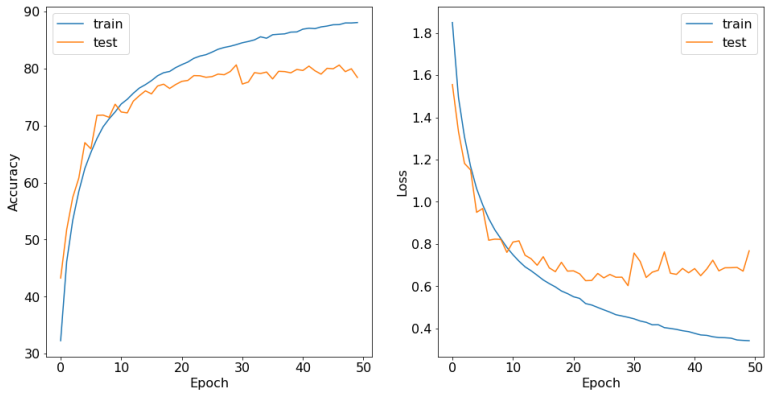


Figure (16) Network's performance when trained with a perspective trnaqformation augmentation.

- **GaussianBlur** Blurs image with randomly chosen Gaussian blur.
  This augmentation could be pretty effective. In fact, whatever the way we acquire the image, there is always a chance that the quality will be poor. Often, this degradation in quality takes the form of a blur on the image. Thus, applying this augmentation makes the network more robust to poor quality images. However in our case, the quality of the image is probably pretty good and similar in both the training and test set. This means that adding blurring makes the network generalize too much and it is less effective than without blurring (around 74% max accuracy).

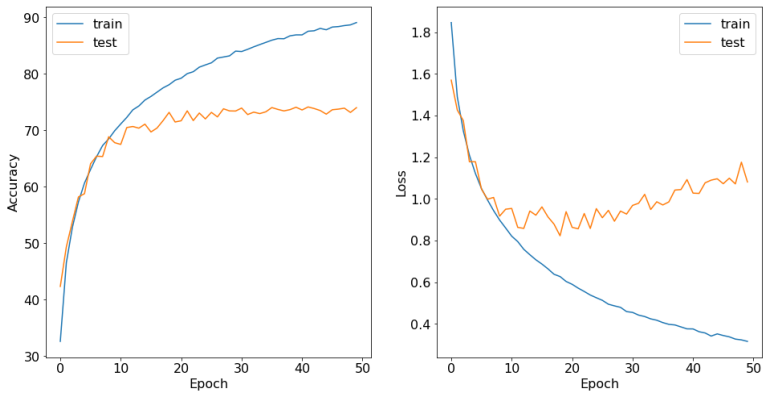


Figure (17) Network's performance when trained with a blurring augmentation.

The relevance of each of these transformation depends not only on the data but also on the intended application. In our case, we won't use any of them as they do not really improve our results and they consequently increase the training time.

26. **Describe your experimental results and compare them to previous results, including learning stability.**
    The learning rate scheduler slightly improves our results. Refining the learning rate over time allowed to gain 1% in accuracy on the test set. The other thing we notice after adding it is that the loss and accuracy curves are smoother. In figure 12, the loss and accuracy curves on the training set are not as smooth as in figure 18. For the test set, we can see more peaks in figure 12 than in figure 18.

27. **Why does this method improve learning ?**
    When we use a constant learning rate with this network, the algorithm tends toward a good minimum. However, it keeps wandering around a minima without being able to completely reach it. Actually, it's the fact that the learning rate is too big to avoid divergence that prevents it from going deeper in the landscape of the loss function. Besides, it allows to have an initially large learning rate to accelerate training and escaping spurious local minima that we may encounter at the beginning of the learning.

    An initially large learning rate may also help the network from memorizing noisy data while decaying the

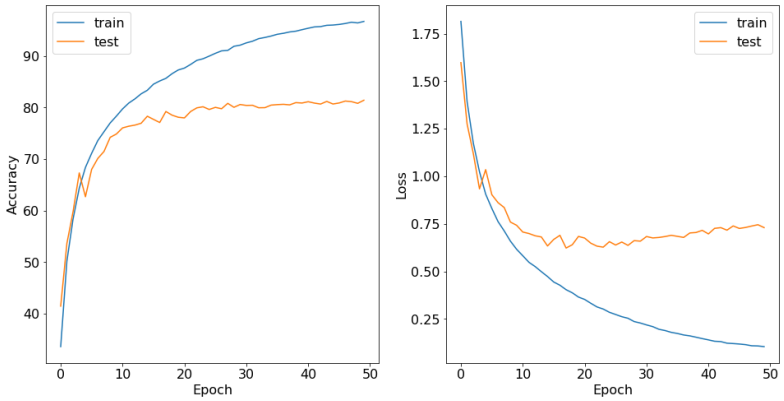


Figure (18) Network's performance with a learning rate scheduler

learning rate improves the learning of complex patterns (*cf.* the paper "How does learning rate decay help modern neural networks?" by Kaichao You *et al.*, published in 2019).

28. **BONUS : Many other variants of SGD exist and many *learning rate* planning strategies exist. Which ones ? Test some of them.**

Given the fact that the update of the SGD algorithm is based on an approximation of the true gradient, there can be a lot of oscillations that can cause the convergence to be very slow. To attenuate this problem, instead of using the vanilla SGD:

$$w \leftarrow w - \eta \frac{\partial \mathcal{L}(x,y)}{\partial w},$$

we can use SGD with momentum:

$$v_{\text{mom}}^{t+1} = \beta_1 \cdot v_{\text{mom}}^{t} + (1-\beta_1) \cdot \frac{\partial \mathcal{L}(x,y)}{\partial w},$$

$$w \leftarrow w - \eta \cdot v_{\text{mom}}^{t+1},$$

where $v_0 = 0$ and β is typically equal to 0.9.

Among the other common existing learning rate planning strategies, we can mention the:

- **step decay** $\eta_t = \eta_0 \times r^{\frac{t}{t_u}}$;
- **inverse (time-based) decay** $\eta_t = \frac{\eta_0}{1+r \cdot t}$;

where η_0, t_u, r are hyper-parameters and t is the iteration number.

There also exist methods like Root Mean Square propagation optimzer (RMSprop) or Adam that can adaptively the learning rates, and even do so per parameter.
We experimented with the step decay which proved to decrease too fast when we set the learning rate to 0.1 and the step size to 0.002. Exponential decay proves to be way more adapted to the way the learning evolves as it adapts to the fast evolution at the start and slow at the end.

RMSProp The update of this optimizer is:

$$v_{i,rms}^{t+1} = \beta_2 \cdot v_i^t + (1-\beta_2) \cdot \left(\frac{\partial \mathcal{L}(x,y)}{\partial w} \odot \frac{\partial \mathcal{L}(x,y)}{\partial w}\right),$$

$$w_i \leftarrow w_i - \frac{\eta}{\sqrt{v_i^{t+1}}} \cdot \frac{\partial \mathcal{L}(x,y)}{\partial w_i}.$$

The more a parameter makes the estimate of the cost function oscillate, the more its update is penalized. In our experiments, we find that this optimizer is less efficient than the SGD.

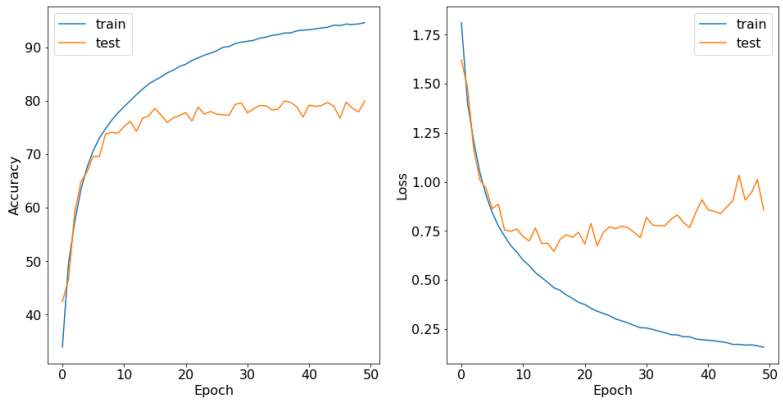


Figure (19) Network's performance when scheduling the learning rate with a step decay.

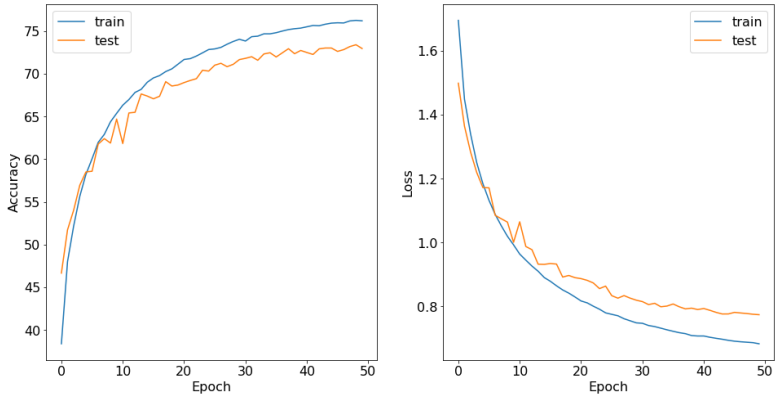


Figure (20) Network's performance when optimizing with the RMSprop method.

Adam The update of this optimize is given by:

$$v^{t+1}_{i,\text{corrected mom}} = \frac{v^{t+1}_{i,mom}}{1-\beta_1^t},\ v^{t+1}_{i,\text{corrected rms}} = \frac{v^{t+1}_{i,rms}}{1-\beta_2^t}$$

$$w_i \leftarrow w_i - \eta \cdot \frac{v^{t+1}_{i,\text{corrected mom}}}{\sqrt{v^{t+1}_{i,\text{corrected rms}}}+\varepsilon} \cdot \frac{\partial \mathcal{L}(x,y)}{\partial w_i}.$$

where ε is a small number to avoid division by zero.

We can see that it's a kind of a fusion of the momentum and RMSProp optimizers. The bias corrections are just here to adjust for a slow startup when estimating momentum and a second moment because they are initialized to zero.
In our experiments, Adam achieves very similar results to the SGD with an accuracy on the test set almost reaching 81%.

However, SGD, SGD with momemtum, RMSProp and Adam have a common drawback. Indeed, when there mutltiple dimensions, the loss can reach a minimum along one but not the others. This means that the gradient will be equal to 0 and the descent will stop. A type of optimisation based on Newton's method solves this problem by using the following update :

$$w \leftarrow w - [H_{\mathcal{L}}(X,Y)]^{-1} \cdot \frac{\partial \mathcal{L}(X,Y)}{\partial w},$$

where HL(X, Y) is the Hessian matrix (of the loss function) which intuitively describes the local curvature of the loss function. The update is more efficient and it does not require any hyper parameter such as the learning rate. The problem of this method is that it very complex to compute the Hessian matrix and its

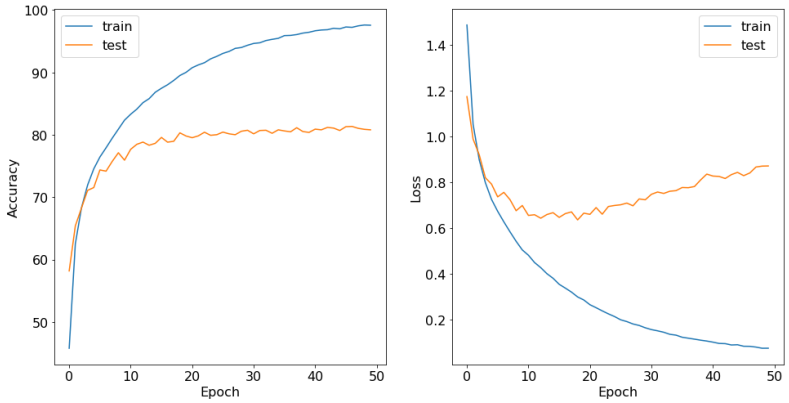


Figure (21) Network's performance when optimizing with the Adam method.

inverse so it is hardly usable in our context of deep learning. However, some methods have been developed to get close to copy this one in a less costly fashion, such as L-BFGS.

29. **Describe your experimental results and compare them to previous results.** Adding the dropout

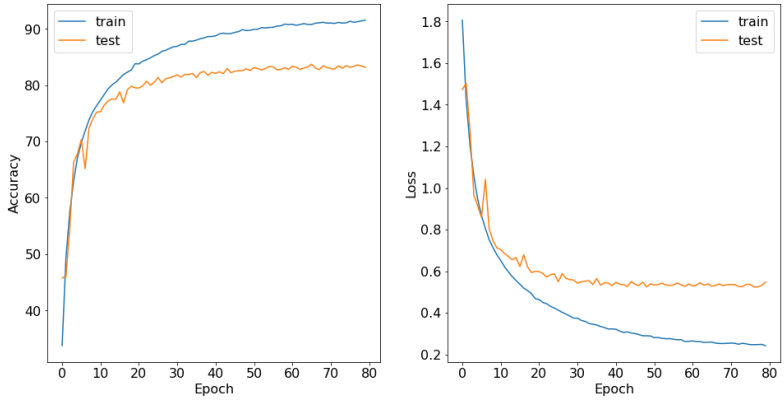


Figure (22) Network's performance with a dropout layer added.

allows us to improve even more the accuracy on the test set. In fact we reach a score of around 82%. We also find that overfitting is not as high as it was before, both the losses and accuracies converge now.

30. **What is regularization in general ?**
Regularization is to make results more simply explainable. Generally we use for two different purposes:

    (a) Limiting the complexity of a machine learning model in order to prevent overfitting and increase its ability to generalize. This limitation is done by removing the least impactful variables (*e.g.* dropout).

    (b) Finding solutions to ill-posed problems (*e.g.* to impose sparsity in compressed sensing).

    In both cases, these constraints are often defined by the addition of a penalization term.

31. **Research and "discuss" possible interpretations of the effect of dropout on the behavior of a network using it ?**
By preventing feature co-adaptation (feature only helpful when other specific features present), the Dropout technique significantly improves generalization and therefore reduces overtifitting (especially when there are huge fully connected layers). One can interpret a neural network with dropout as an average of many neural networks.

    We will also understand in a future class of RDFIA that:

    - a neural network with dropout can be precisely interpreted as a variational Bayesian approximation "with particular prior and approximate posterior";

- this interpretation offers an explanation to some of dropout's key properties, such as its robustness to over-fitting.

32. **What is the influence of the hyperparameter of this layer ?**
This hyper-parameter corresponds to the probability of a neuron of this layer to be disabled during a pass. If it's too big then the network will underfit the data and will be unstable. If it's too small the potential of dropout to correct overfitting will not be exploited enough.

33. **What is the difference in behavior of the dropout layer between training and test ?**
During the training neurons are randomly disabled whereas when we want to test the network we have to use all the neurons. Furthermore, in PyTorch the outputs are scaled by a factor of $\frac{1}{1-p}$. This allows us to make sure the outputted numbers of the layers in train and in test have approximatively the same scale even if the number of neurons from which they depend is not the same.

34. **Describe your experimental results and compare them to previous results.**
Finally, adding batch normalization brings the score to a maximum of 85%. This is effective because similarly

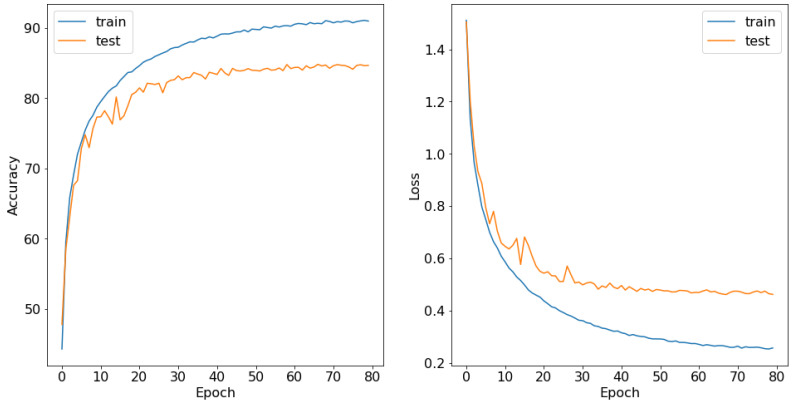


Figure (23) Network's performance with a batch normalization added.

to the impactful normalization we did earlier in this work, outputs of the layers are not renormalized and it is possible to find extreme values among those.
Also, batch normalization is interesting because the training is then less sensible to poor parameter tuning. It also acts as a regularization technique as the extreme values are attenuated by the normalization.

RDFIA: Homework 3-a,b,c,d

GALMICHE Nathan PIRIOU Victor

February 20, 2023

Abstract

The performance of neural networks has been improving for a decade now. For instance, the field of image synthesis was revolutionised by the advent of Generative Adversarial Networks (GAN) whose goal is to learn the joint distribution of the network's inputs and outputs. More generally, GAN have fundamentally allowed new ways of approaching problems. However, the deployment of other deep learning based systems is still hampered by certain problems:

- **their lack of interpretability.** The problem of the lack of interpretability of these networks is waiting to be addressed, especially in the field of medical diagnostic assistance. One way to make them more interpretable is to generate maps allowing to visualise the degree of involvement of the pixels in the classification.
- **their vulnerability to adversarial attacks.** Indeed, they can be misled during classification by adding modifications to the image that are imperceptible to the naked eye, making these systems vulnerable to attacks.
- **the lack of annotated data, time and computational resources to train them.** Transfer learning is a solution to this problem. It allows to exploit the features learned by a network on another dataset. Another solution to this problem is domain adaptation which allows to train a network with little or no annotated data. This technique only needs the annotated data of another dataset having a similar domain. The lack of annotated data can also be partly filled by annotated data synthesis via GAN. By reducing the data synthesis problem to a fight between a generator and a discriminator, they allow, among other things, to generate much more realistic images than before. Indeed, the use of a discriminator makes it possible to get rid of the problem of certain similarity measures such as the least squares error, which sometimes leads to unrealistic examples.

These practical works allowed us to discover the way the different methods studied work, to think about the different ways of implementing these approaches and to become aware of their limits.

1 Transfer Learning through feature extraction from a CNN

1.1 VGG16 Architecture

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).**
   VGG16 has 3 fully-connected layers.

   The first one is applied on a tensor of size $7 \times 7 \times 512$ and has 4096 neurons. Hence, it has $(7 \times 7 \times 512) \times 4096 + 4096$ bias $= 102\ 764\ 544$ parameters.

   The second one is applied on a tensor of size $1 \times 1 \times 4096$ and has 4096 neurons. Hence, it has $(1 \times 1 \times 4096) \times 4096 + 4096$ bias $= 16\ 781\ 312$ parameters.

The third one is applied on a tensor of size $1 \times 1 \times 4096$ and has 1000 neurons. Hence, it has $(1 \times 1 \times 4096) \times 1000 + 1000$ bias $= 4\ 097\ 000$ parameters.

In total, these three fully-connected layers have 123 642 856 parameters.

2. **★What is the output size of the last layer of VGG16?**
   The output size is equal to the number of classes, *i.e.* 1000.

   What does it correspond to?
   In theory, the output of the last layer is the Softmax function applied on the last fully-connected layer. The Softmax function just normalises the feature vector such that it becomes a probability distribution over the classes. It does not change the size of the vector on which it is applied.

   In practice, the last layer of our pre-trained model is the last fully-connected network.

3. **BONUS : Apply the network on several images of your choice and comment on the results.**
   We first applied the network on an image from the COCO dataset. As we obtained a lot of false positives

Figure (1) An image from the COCO dataset whose images features more complex scenes than the ones of ImageNet.

```
Top 15:
     umbrella : 11.321854
     cash machine, cash dispenser, automated teller machine, automatic teller machine, automated teller, automatic teller, ATM : 11.201432
     limousine, limo : 11.009268
     fur coat : 10.813289
     trench coat : 10.593885
     groom, bridegroom : 10.484614
     suit, suit of clothes : 10.425961
     abaya : 10.124124
     television, television system : 9.912357
     pay-phone, pay-station : 9.819627
     crutch : 9.752721
     ashcan, trash can, garbage can, wastebin, ash bin, ash-bin, ashbin, dustbin, trash barrel, trash bin : 9.668255
     shoe shop, shoe-shop, shoe store : 9.350463
     stage : 9.226672
     jean, blue jean, denim : 8.938109
```

Figure (2) Top-15 predictions of the image coming from the COCO dataset.

(*e.g.* the detected limousine, umbrella, etc...). We also obtained many false negative. For instance, the car and the table are not present in this top-15 predictions whereas these are important parts of the image. These unsatisfactory results were expected because the ImageNet dataset, from which our network was pre-trained, contains images centered around the object whereas our more complex image from the COCO dataset is not like that. Moreover, we had to resize it in order to feed it to the network, which is a naive approach as we saw in the sixth lecture.

We also applied the network on an image that contains a brown bear and the results were very good since the real class was detected as the most probable one and because other classes that were the most predicted are semantically very close to the real one.

Figure (3) An image similar to the ones that ImageNet contains in the sense that this image mainly contains one object which is highly visible in the foreground.

```
Top 15:
     brown bear, bruin, Ursus arctos : 25.984428
     American black bear, black bear, Ursus americanus, Euarctos americanus : 19.949831
     sloth bear, Melursus ursinus, Ursus ursinus : 16.747196
     ice bear, polar bear, Ursus Maritimus, Thalarctos maritimus : 16.668316
     otter : 14.589978
     hyena, hyaena : 14.416616
     weasel : 14.29308
     wild boar, boar, Sus scrofa : 14.236489
     marmot : 14.190776
     Airedale, Airedale terrier : 14.034263
     hog, pig, grunter, squealer, Sus scrofa : 13.807155
     baboon : 13.610524
     mink : 13.268329
     king penguin, Aptenodytes patagonica : 13.245407
     chimpanzee, chimp, Pan troglodytes : 12.981028
```

Figure (4) Top-15 predictions of the image containg the brown bear.

Figure (5) A cropped frame of a cartoon movie that features a duck.

This time the results are not so good because to most probable class is the wrong one. Indeed, the duck was not recognised by the network. However, the latter detected a pelican which is a class that is semantically very close to the real one. These bad results are explained by the fact that ImageNet contains realistic images and not ones taken from cartoon.

```
Top 15:
     vacuum, vacuum cleaner : 12.652191
     swing : 10.946797
     broom : 10.844319
     crutch : 10.776963
     golfcart, golf cart : 10.711409
     swab, swob, mop : 10.512167
     laptop, laptop computer : 10.256154
     lawn mower, mower : 9.111918
     shopping basket : 9.070771
     electric guitar : 8.9348
     paddle, boat paddle : 8.907834
     pelican : 8.709079
     shield, buckler : 8.438528
     shoe shop, shoe-shop, shoe store : 8.3623295
     motor scooter, scooter : 8.320831
```

Figure (6) Top-15 predictions of the image of a duck taken from a cartoon.

4. **BONUS : Visualize several activation maps obtained after the first convolutional layer. How can we interpret them?**

   In sum, all these features contained by these maps are complementary between each other.

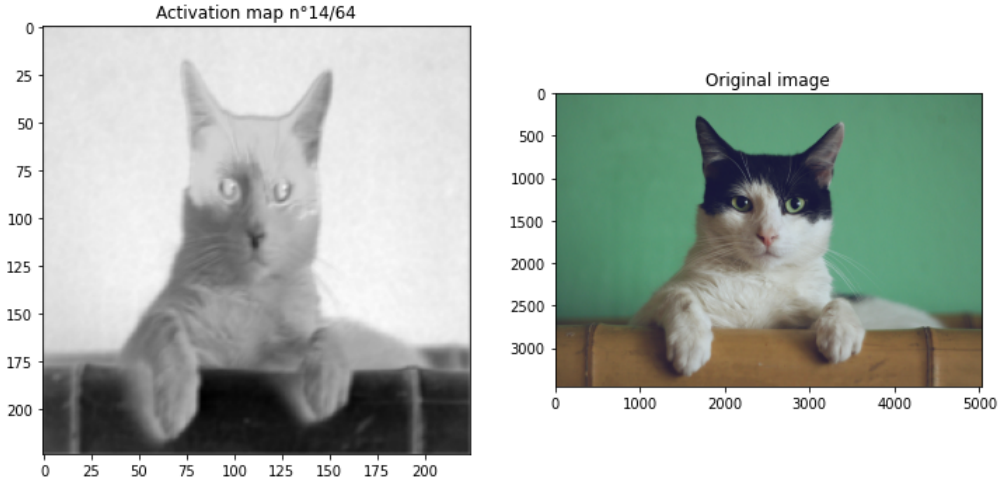


Figure (7) This filter seems to strongly react to green area.

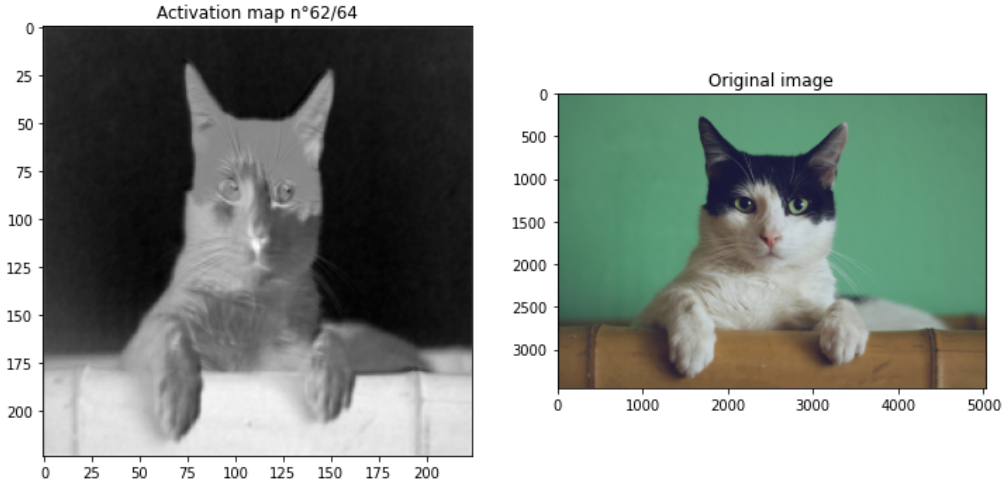


Figure (8) This one seems to strongly react to brown area.

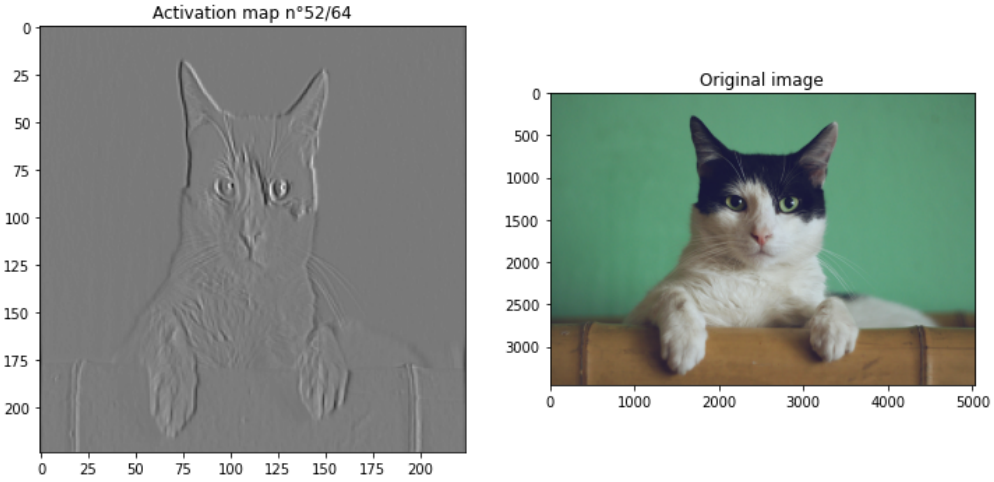


Figure (9) This one seems to act like a high-pass filter because it extracts the contours.

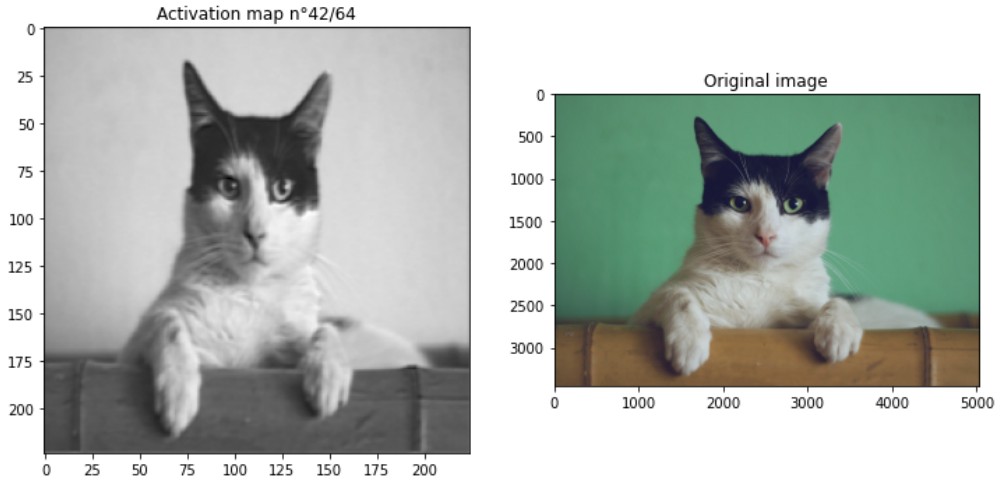


Figure (10) This one seems to act like a low-pass filter because it blurs the image.

1.2 Transfer Learning with VGG16 on 15 Scene

5. ★ **Why not directly train VGG16 on 15 Scene?**
   There are two reasons for that.

   Firstly, the number of neurons of the last layer of VGG16 is not equal to the number of classes of the 15 Scene dataset.

   Secondly, if we directly train VGG16 on 15 Scene there is a risk of overfiting. Indeed, VGG16 has many parameters because its architecture was designed so that it can be train on the huge ImageNet dataset containing more than 14 million images. Although they are organised into 21,841 subcategories that have 500 images on average, these subcategories can be considered as sub-trees of 27 high-level categories. Hence, we can roughly say that in ImageNet each class has much more images than the 15 Scene dataset that contains 200-400 images per category.

6. ★ **How can pre-training on ImageNet help classification for 15 Scene?**
   ImageNet is a very diversified dataset. Therefore, the first layers of a convolutional network trained on this dataset can be seen as an extractor of high-level features that allows to make the classes more linearly separable. In our case, the fact that the pre-trained model is frozen during the training on the 15 Scene dataset reduces the risk of overfitting.

7. **What limits can you see with feature extraction?**
   If the classification task of the new problem differs too much from the one that the pre-trained network was

trained to solve then the extracted features will not be the most relevant ones, especially after the last layers of the network.

Additionally, the domain of the data from which we want to extract features needs to be similar to the one of the data from which the pre-trained model was trained.

8. **What is the impact of the layer at which the features are extracted?**
   The deeper this layer is the more abstract are the extracted features. It may seem counter-intuitive but ablation studies (*cf.* course 4) show that the deeper the layer from which the features are extracted is, the better the classification results we obtain.

9. **The images from 15 Scene are black and white, but VGG16 requires RGB images. How can we get around this problem?**
   For each image, we just have to stack two copies on top of the original image in order to obtain images that have three identical channels.

10. **Rather than training an independent classifier, is it possible to just use the neural network? Explain.**
    Yes it is possible but we need to change the number of neurons of the third fully-connected layer in order to make it equal to the number of classes (15) of the dataset we use.

11. **For every improvement that you test, explain your reasoning and comment on the obtained results.**
    First, we trained the multi-class SVM classifier on the training set with $C = 1$ using the *deep features* previously extracted and we obtained an accuracy of 0.887772 on the test set. Then, we changed the layer at which the features are extracted while fine-tuning the C parameter. Here are the results we obtained:

	Representation size	**Best accuracy**	**Best C value**
1st conv. layer	3211264	unknown	unknown
2nd conv. layer	1605632	unknown	unknown
3rd conv. layer	802816	unknown	unknown
4th conv. layer	401408	unknown	unknown
5th conv. layer	100352	0.898	2.100
6th conv. layer	25088	0.900	1.350
1st FC layer	4096	0.889	1.450
2nd FC layer	4096	0.889	1.450
3rd FC layer	1000	0.874	2.450

The bigger the representation size, the more the computational cost (time and memory) of the SVM's training is increased. It may seem counter-intuitive but, as we saw in class, the deeper the layer at which is extracted the features is the better will be the results obtained with the SVM. However, in practice this is not always verified. Here, the fact that the best accuracy is not obtained at the last layer may be explained by the high dissimilarity between the Scene-15 and the ImageNet datasets. The resource freely allocated by Google Colab are not sufficient to train the SVM on such big embeddings.

We also tried to use Resnet instead of VGG16 because it is known to be more efficient by itself. However, when we cut the architecture at the last convolution layer, just before the fully connected one, and applied a SVM on the features obtained, we find that the best score obtained is not better than VGG16. This can be explained by the fact that resnet has a very deep architecture and the features extracted by the model are thus very abstract.

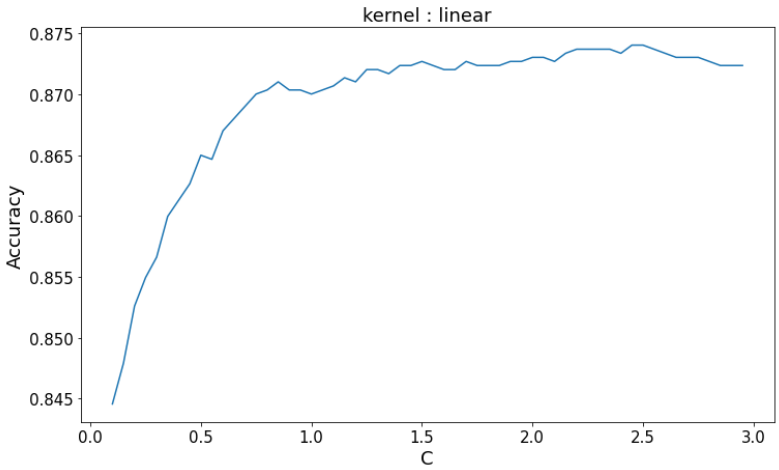


Figure (11) Evolution of the precision depending of C when transferring the features extracted from the last convolution layer of a ResNet50.

2 Visualizing Neural Networks

2.1 Saliency Map

1. ★ **Show and interpret the obtained results.**

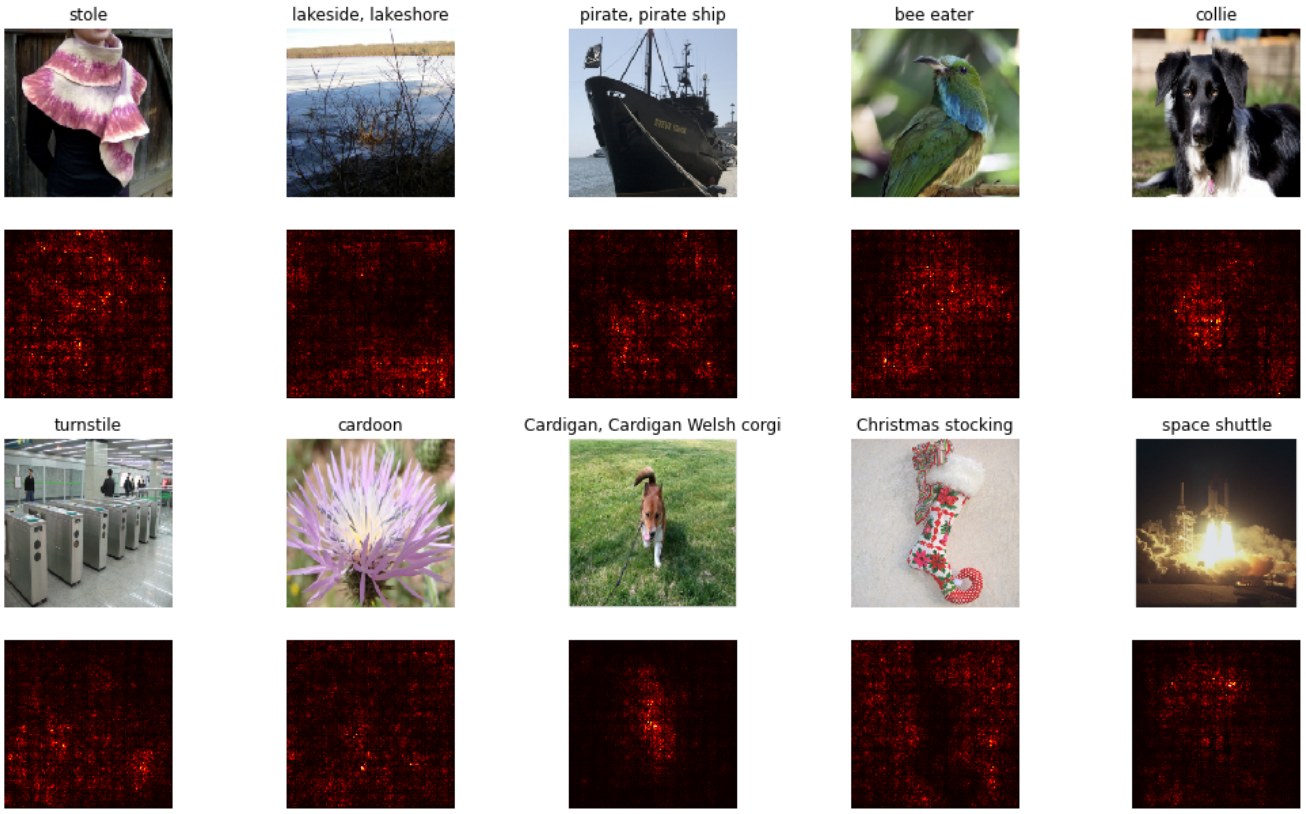


Figure (12) SqueezeNet saliency maps of various images from ImageNet.

We obtain different kinds of results. Some of the saliency maps are exactly the way we expect them to be. For example, the most impactful pixels for the Cardigan are all stacked in the center of the image which is where the dog is on the image. Also for the Collie which takes almost all of the image, we see that the important pixels are mainly where the face of the dog is on the image. Since the face is where most of the features specific to this species are, we expected this saliency map.

Some other saliency maps are a bit more surprising, such as the turnstile's. We see that the important pixels are more spread on the map. This still makes sense since there are more than one turnstile on the image and they take almost all of it. We can still wonder about the patch on the bottom right which seems to be on the floor. This could be due to a phenomenon that is very obvious in the Christmas stocking's saliency map. We see that almost none of the actual stocking's pixels are important to recognise it. The pixels that are important are the background's, to the point where the complementary shape of the stocking appears. This is a problem in the dataset that is probably linked to the fact that Christmas stockings are often portrayed against a plain wall. Therefore, during training, the model has associated Christmas stocking with the plain background instead of the stocking itself. This was probably done because it is easier for the network to learn to recognise this simple background than the appearance of a stoking. To solve this issue, we would have to add images with plain background in the other classes and add diversity to the Christmas stocking class.

2. **Discuss the limits of this technique of visualizing the impact of different pixels.**
These activation maps probably also differ from the real activation maps of the network because they are a bit different from the ones we can naturally expect. This is mainly explained by the limiting assumption that is implicitly done by the approximation of the linear function: the independence between pixels. Another limit is that only the localisation of these pixels is taken into account while their intensity is completely ignored.

3. **Can this technique be used for a different purpose than interpreting the network?**
Yes, they can be useful for detection and segmentation tasks because the features that allow the model to recognize are generally on the object. The activated pixels allow to localise the object. More precisely, this could also be used to localise certain parts of the objects. If we notice that the network uses certain features of the object, we can use these maps to localise them on other images.

4. **BONUS: Test with a different network, for example VGG16, and comment.**

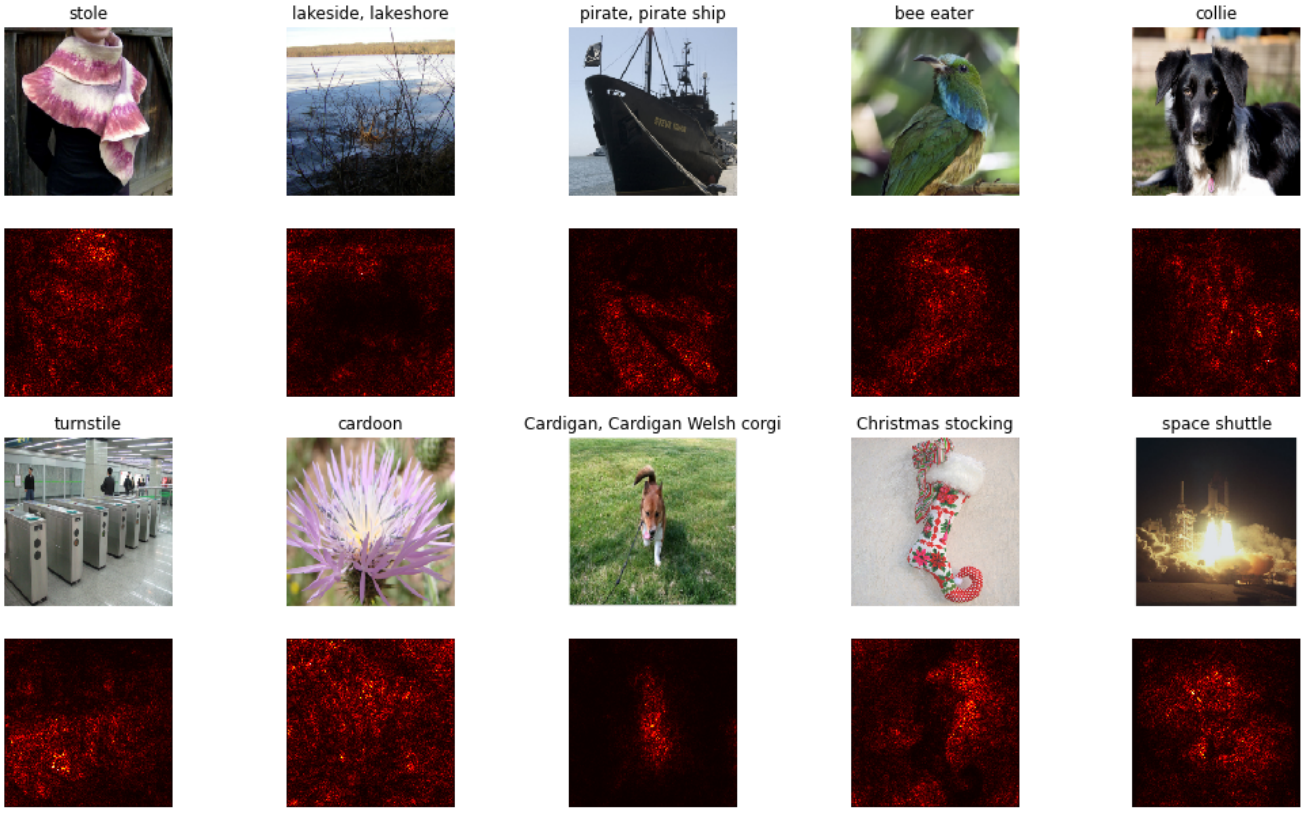


Figure (13) VGG16 saliency maps of various images from ImageNet.

We can see that VGG16 is a lot more precise than SqueezeNet. This was expected because it has more and bigger layers. The saliency maps that were already precise with SqueezeNet are even better here. For example, we can even recognise the Cardigan's tail in the map. The maps where the main pixels were spread still are but with less outliers. For example, the turnstiles' map makes more sense as there are more important pixels

on the actual objects and less stacks on the ground.
However, we still find the same phenomenon with the Christmas stocking, which is not surprising since the problem probably comes from the dataset, as explained in question 1.

2.2 Adversarial Examples

5. ★ **Show and interpret the obtained results.**

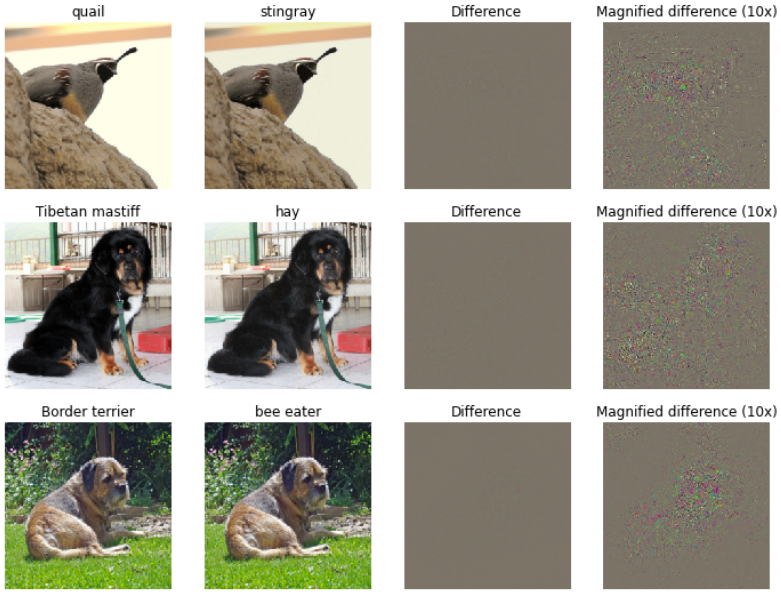


Figure (14) Original images and modified images where the model is fooled.

At first glance, it looks like the images have not been modified and the model is still fooled. When taking a closer look we can see small changes, in intensity for example. Even the difference image has to be magnified in order to make the modifications visible.
Even though the modifications are optimised to fool the model as fast as possible, it is interesting to note that it happens in only a few iterations.

6. **In practice, what consequences can this method have when using convolutional neural networks?**
This is an hindrance to the deployment of convolutional neural networks in real life. Indeed, since such networks can be adversarially attacked, we cannot guarantee that in real life these networks will have performance similar to the ones they had in test. Among the applications that are affected by this problem we can mention autonomous driving since people can easily change the input of the network whereas we cannot afford wrong predictions.

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).**
Here we perform a backpropagation so we assume that we have access to the trained model, which is not always the case. Another issue with this approach is the number of model queries required to compute the gradient of a solution within the high-dimensional image space.

Among the existing methods, we can cite the one that [1] Phoenix Williams and Ke Li proposed in 2022. It is a novel attack that uses gradient-free optimisation by evolving a series of overlapping transparent shapes (triangles, rectangles, etc) using an evolutionary algorithm. Their method only requires the query feedback returned by the network and has shown an ability to outperform the other *black-box* attacks method that rely on the estimation of the gradient.

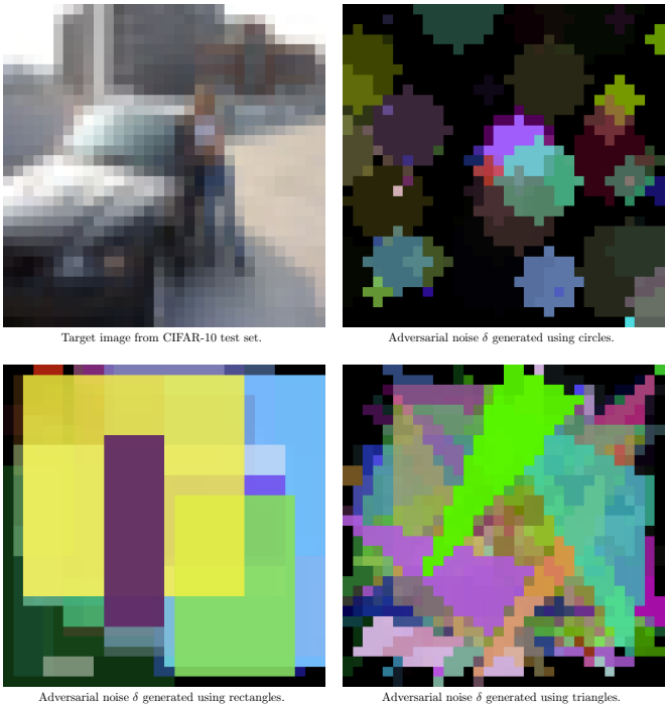


Figure (15) Example of magnified adversarial noise δ generated using 50 shapes for the top left target image [1].

The same year, Dongbin Na *et. al* proposed another simple but powerful *black-box* attack method [2]. Their methods takes as input two images: a source image x_{src} and a target image x_{adv} that are semantically different. What we want is to classify the source image in y_{trg}, the class of the target image. To do so, two steps are required. First, we use an auto-encoder (made of an encoder and a decoder G) in order to respectively extract from the source image and the target image two latent vectors w_{src} and w_{trg}. Second, we modify w_{trg} such that the distance between w_{src} and w_{trg} is minimised while the class $F\left(G\left(w_{trg}\right)\right)$ predicted by the network F is unchanged. Once this objective is reached, the modified vector w_{trg} is named w_{adv}. Formally, the objective is:

$$\underset{w_{adv}}{\text{minimize}} \quad D\left(w_{src}, w_{adv}\right) \quad \text{s. t.} \quad F\left(G\left(w_{adv}\right)\right) = y_{trg}.$$

Finally, the adversarial example is the generated image $G(w_{trg})$.

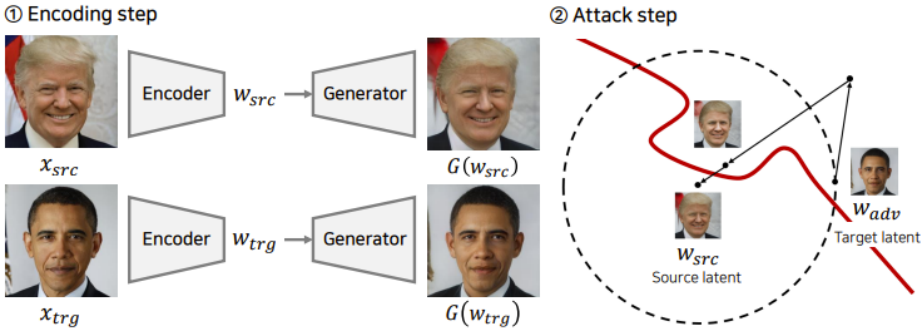
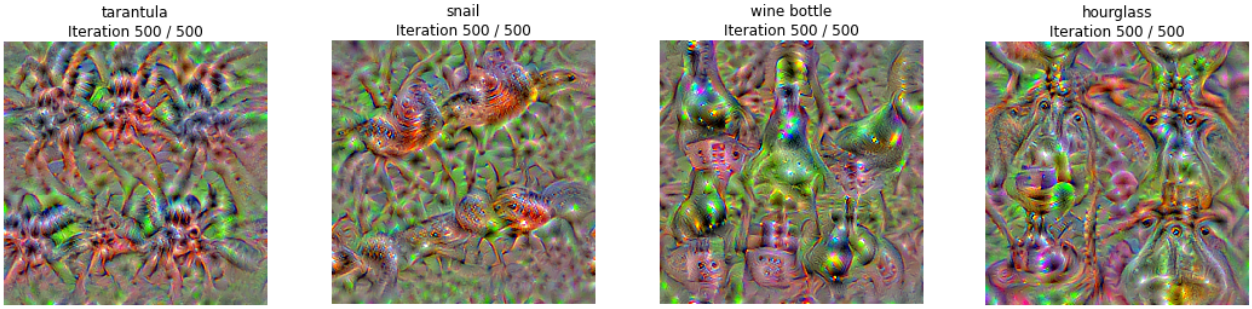


Figure (16) An illustration of the method proposed in [2].

However, the authors noticed that sometimes the adversarial examples that the methods creates look too different from the original images.

2.3 Class Visualization

8. ★ **Show and interpret the obtained results.**



(a) (b) (c) (d)

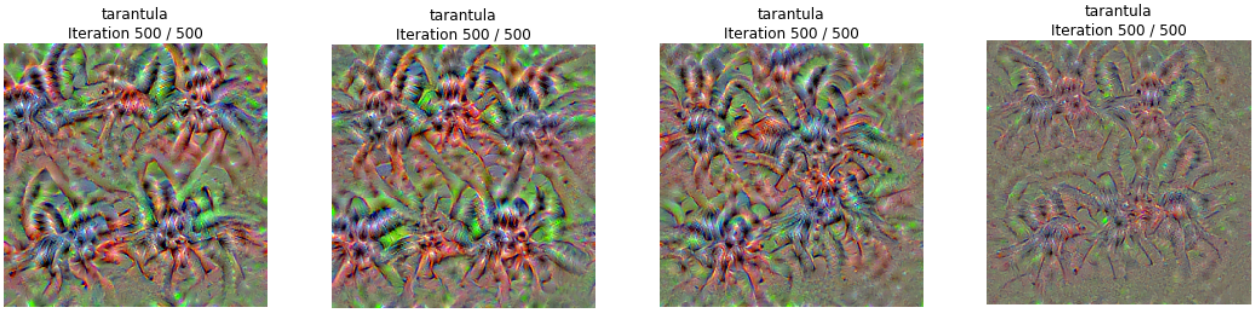
Figure (17) Different examples of visual patterns producing class predictions.

It is interesting to note that it is hard for the human eye to recognise the class that was reproduced. The images are not realistic but we can still see some of the features. For example, if we look closely at the tarantula image, we can see the 8 characteristic legs as well as kind of bodies.

Depending on the class, we see that the colors make more or less sense. For example, the hourglass is represented with many colors mixed. This is because this object does not really have a characteristic color, whereas the wine bottle is mainly represented in green which is its basic color. We can also note the touches of green around the tarantulas and snails that are probably caused by the fact that theses animals are often portrayed on grass or leaves.

We also see that on all of the above examples, the object is represented many times on the image. This is because the model has been trained on many different images for each class and the position of the object on these varies. The model is thus trained to find the object anywhere on the image and in different positions and sizes.

9. **Try to vary the number of iterations and the learning rate as well as the regularisation weight.**

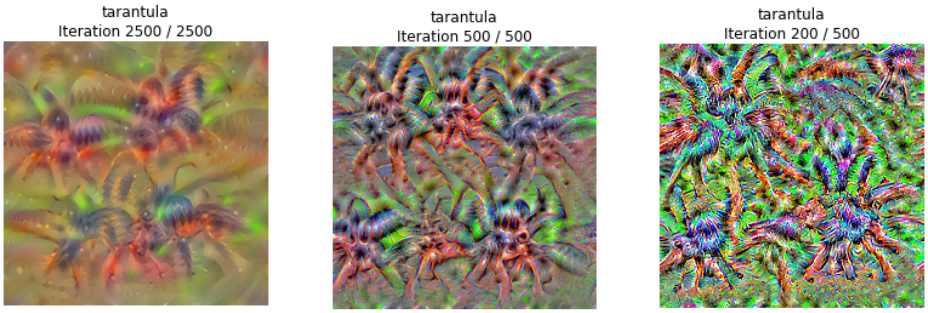


(a) (b) (c) (d)

Figure (18) Tarantula class generated from random noise with the regularisation weight set to (a) 10^{-7}, (b) 10^{-3}, (c) 10^{-1} and (d) 1.

By letting the regularisation weight vary we see in Figure 18 that having a big weight really limits the apparition of new details. In fact, as the regularisation penalises the image's norm, if it is high we expect it to limit the intensities. Here, one is too big of a weight since it is hard to recognise the tarantula shapes. However, setting it to 0.01 make the appearing shapes a bit more intense and we start to recognise them. If we want to really recognise the class's features, we can use a regularisation weight of 10^{-3}. We note the reducing even more the weight does not really change the appearance of the generated image. However, we can expect the difference to reside in the fact that we are more likely to use this method on real images instead

of random noise. Then the more we will reduce this weight, the less the original image will be altered. If we want to visually modify the class of the image, it is better to keep the weight big enough.



(a) (b) (c)

Figure (19) Tarantula class generated from random noise with the learning rate set to (a) 1, (b) 5 and (c) 10.

Obviously, the first impact we note is that the more we reduce the learning rate, the more iterations it takes to obtain a meaningful result.
Also, by comparing the three images we see that the images with a bigger learning rate have much more contrast between the colors. This is because the features are generated much more strongly at each step. A small learning rate induces that the image is not modified by a lot so only small changes in intensities are applied. As a result, the different colors are mixing more than if the intensities changed by a lot every step.
Another effect is that bigger learning rate seem to generate more features. We can see in figure 18.c that there are shapes everywhere in the image, while in figure 18.a there are zones with no shapes.

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.**

    As we can see in figure 19, the visual interest of generating real class features on an image is limited. The image is altered while the objects of interest are modified. For a human, it only makes it harder to recognise and understand the scene.
    However, it is interesting to do this when the idea is to show the modified image to a model. In fact, features making it classified in the hay class are amplified and it improves the chances for the model to get the guess the right class.

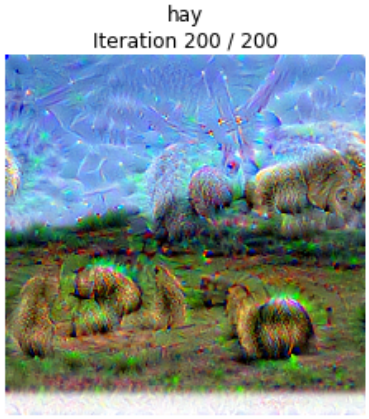
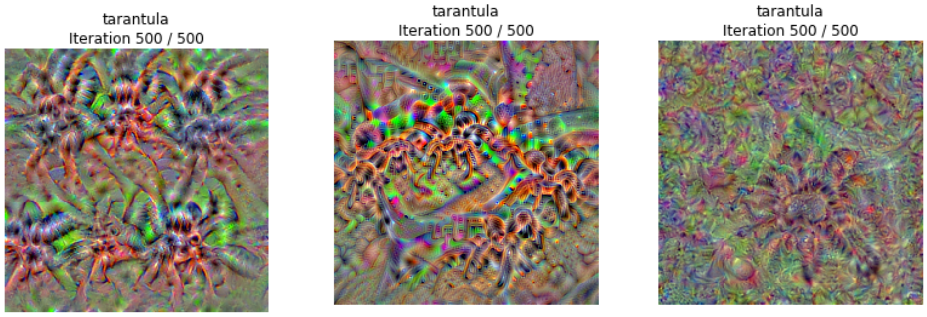


Figure (20) Image from the hay class with hay features generated.

11. **BONUS: Test with another network, VGG16, for example, and comment on the results.**

    As we expected, results are very different from one model to the other. For example, VGG16 seems to have better taken into account the backgrounds of the tarantula images as we can see the to of the image showing



(a) (b) (c)

Figure (21) Tarantula class generated from random noise with (a) SqueezeNet, (b) VGG16, (c) ResNet.

different features than the rest.
On all three images, we can recognise tarantula shapes. However their sizes, number and position are different. ResNet's tarantula is one big animal with a very faithful shape. VGG16's are more than one and much smaller, they also are much harder to recognise as we can mainly see the features of the legs. SqueezeNet's are also more than one but this time pretty big. The main diference with the other two is that the spiders take all of the image.
Another important difference is that it took much longer to generate the images with VGG16 and ResNet, which is normal since SqueezeNet is made to be compact.

3 Domain Adaptation

3.1 Practice

1. **If you keep the network with the three parts (green, blue, pink) but didn't use the GRL, what would happen ?**
   If we remove the GRL layer, the features will be domain dependant so the label predictor will be less successful on the source domain.
   Consequently, the learnt features will be more specific to the target domain and then not optimised for the label predictor.

2. **Why does the performance on the source dataset may degrade a bit ?**

   The performance will be degraded because the network will learn to generalize more. The features it recognises will be less precise on the source domain in order to envelop the target domain.

3. **Discuss the influence of the value of the negative number used to reverse the gradient in the GRL.**
   The gradient needs to be reversed just before that because we want the feature extractor to produce features such that the domain classifier cannot discriminate the domain. The bigger λ is, the more the weights of the feature extractor will be updated such that it similarly encodes the features of the two domains. However, if λ is too high there will be a risk of divergence.

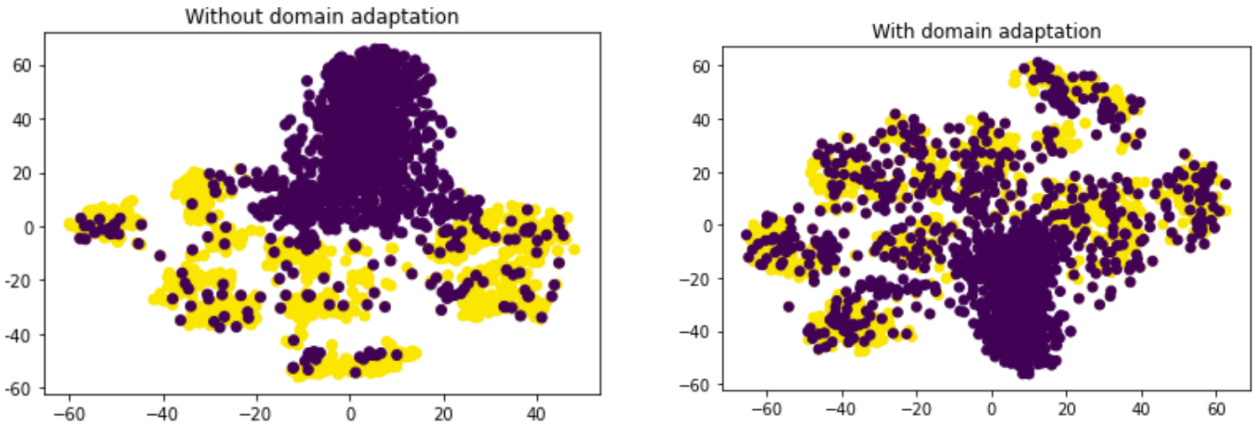
4. **Another common method in domain adaptation is pseudo-labeling. Investigate what it is and describe it in your own words.**
   When used in domain adaptation, data labelling consists of training a network in a supervised fashion with labeled and unlabeled data simultaneously. For a given unlabeled data, its associated groundtruth or *pseudo-label* is the class which has maximum predicted probability by the network. The overall loss is given by:

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

   where:

- n and n' are respectively the number of mini-batch in labeled data and unlabeled data;
- f_i^m and y_i^m are respectively the prediction and the label associated to the $m's$ sample;
- $f_i'^m$ and $y_i'^m$ are respectively the prediction and the pseudo-label associated to the $m's$ sample;
- $\alpha(t)$ is a weighting term which increases during the optimisation process to take into account the fact that the pseudo-labels become more and more reliable during the iterations. Its increasingness is progressive in order to avoid poor local minima.

3.2 Coding results



(a) (b)

Figure (22) Visualisations, with t-SNE, of the latent spaces respectively obtained (a) without and (b) with domain adaptation.

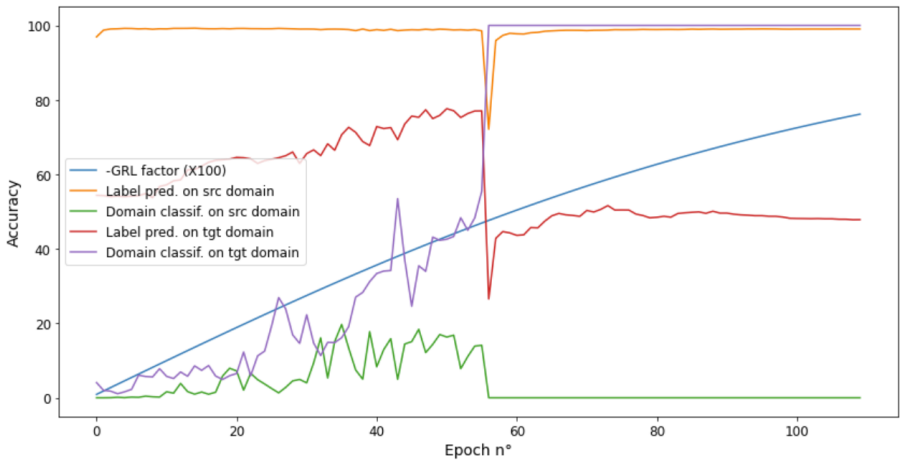


Figure (23) Evolution of the accuracies obtained during the training as well as that of the GRL factor. We expected the accuracy of the domain classifier to be high in the begining and around 50% at the end of the training but it is apparently not the case despite the good accuracy (0.78) that we obtained.

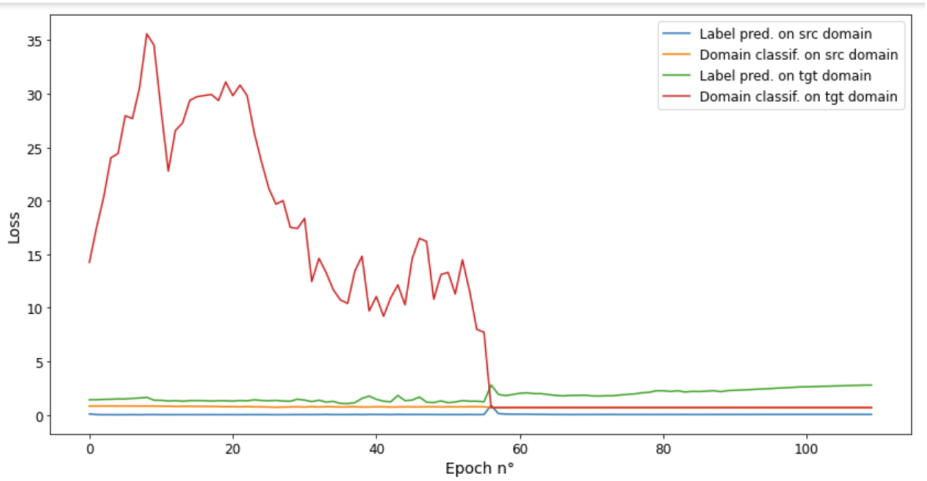


Figure (24) Evolution of the different losses obtained during the training.

4 Generative Adversarial Networks

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

$$\max_G \mathbb{E}_{\mathbf{z}\sim P(\mathbf{z})} [\log D(G(\mathbf{z}))] \tag{6}$$

$$\max_D \mathbb{E}_{\mathbf{x}^*\in \mathcal{D}\text{ata}} [\log D(\mathbf{x}^*)] + \mathbb{E}_{\mathbf{z}\sim P(\mathbf{z})} [\log (1 - D(G(\mathbf{z})))] \tag{7}$$

We know that $D(x)$ represents the probability that the image x is real according to the discriminator D.

Consequently, Equation (6) represents what the generator G seeks to obtain: weights optimised such that it synthesise only images that the discriminator will think are real.

Equation (7) represents what the discriminator D seeks to obtain: weights optimised such that for every image taken as input, it can determine if this image is real or not, without ever being wrong. The first term tests if it recognises real images and the second tests if it can recognise fake ones.

(b) **Ideally, what should the generator G transform the distribution $P(\mathbf{z})$ to?**
Ideally, the generator G should transform the distribution $P(\mathbf{z})$ to the distribution of real data $P(X)$.

(c) **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:

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

The problem with this equation is that it causes the gradient to vanish. In fact, the last layer of the discriminator is a Sigmoid and when we compute the gradient of this loss, we obtain a result proportional to the Sigmoid. This is a problem because, at the start of the training, the generated images are not realistic and the discriminator easily determines that the image is not real so $D(G(x))$ is close to one and therefore the gradient is close to 0.

(d) **Comment on the training of of the GAN with the default settings (progress of the generations, the loss, stability, image diversity, etc.)**
The training worked pretty well. We can see some images that are faithful to the real numbers. As it is often the case with GANs, some of the images don't resemble any number. But this is necessary if we want diversity in the images. In fact, the images generated are varied thanks to the initialisation that is generally random, and then transformed into a realistic image by the generator which has its limits.

Therefore, to have diversity, we need to have bad images too.

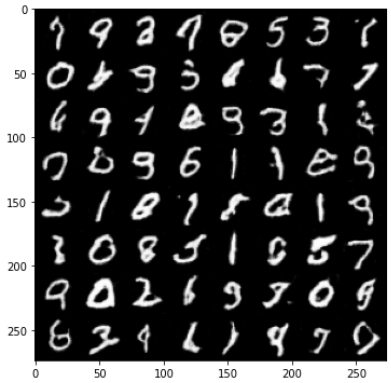


Figure (25) **Numbers generated by the GAN**.

The evolution of the losses is pretty interesting. The loss of the generator decreased at the beginning but both losses evolution can be described as chaotic. This is an effect of the adversarial aspect of the method. Both models are optimised to try to be better than the other which makes the losses fluctuate. An important aspect of the losses evolution is the peaks we regularly encounter over the iterations. This can be explained by the fact that the generator starts to produce always the same type of image. Once the discriminator recognizes it, the generator's loss peaks. On the next iteration, it will have been optimized to produce different kinds of images but the discriminator will only look for the previous type. In turn, the discriminator's loss will peak. We often encounter this phenomenon in the following experiences.

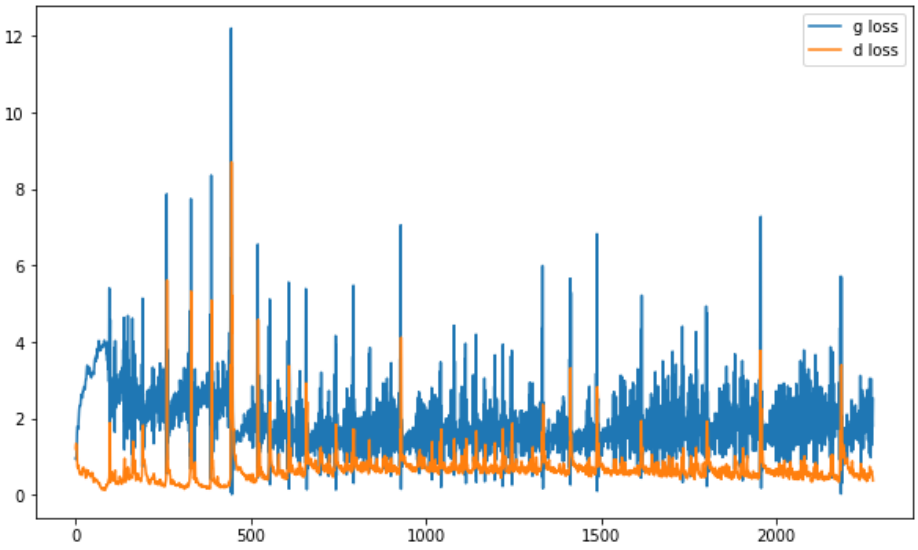
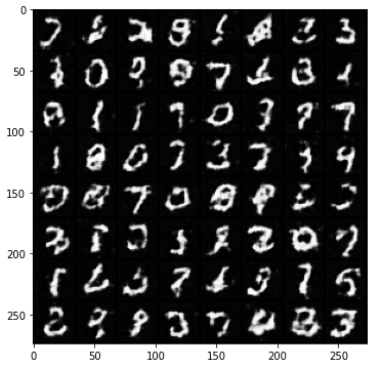


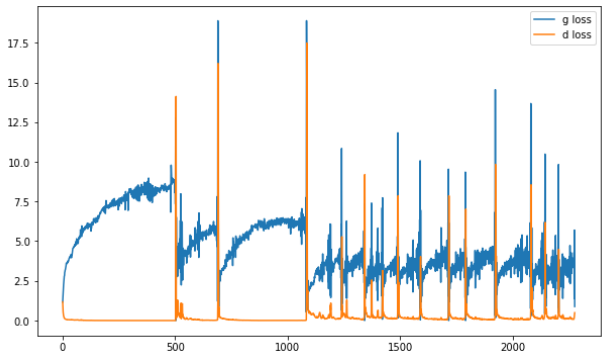
Figure (26) Losses (depending on the epoch n°) obtained during the GAN training.

(e) **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.**

- When we significantly decrease ngf, we see that the generator's loss struggles to stabilize and decrease. Periodically, the loss decreases by a big step and then starts to increase again.



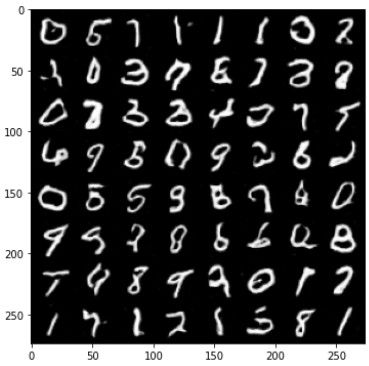
(a) Numbers generated by the GAN.



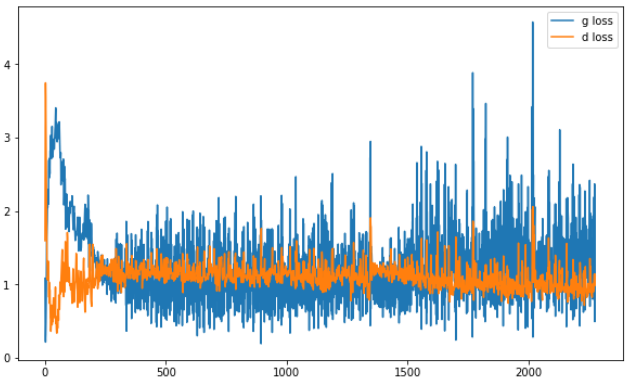
(b) Losses evolution during training.

Figure (27) GAN training with the parameter ngf (number of channels before the generator's last layer) set to 4.

- However, if we increase ngf, we see that the losses reach values we expected them to. We note that they are much less stable than with the default settings.
  Visually, the numbers look realistic and we could even argue that they are better than with the default settings.



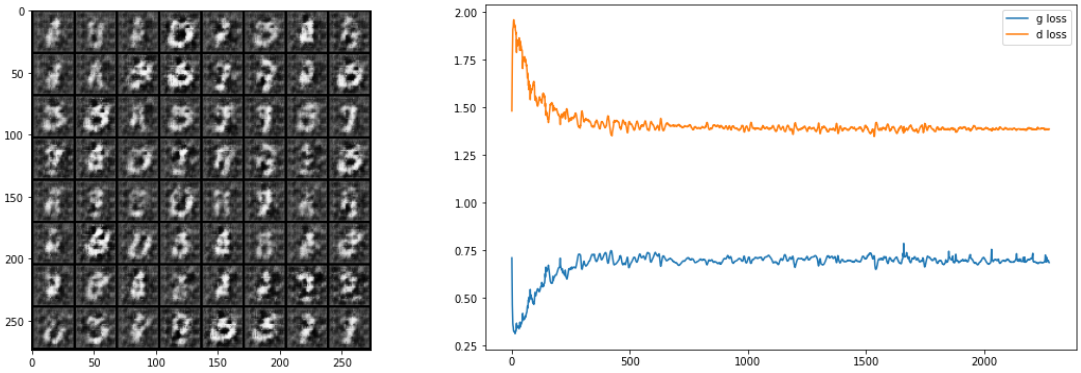
(a) Numbers generated by the GAN.



(b) Losses evolution during training.

Figure (28) GAN training with the parameter ngf (number of channels before the generator's last layer) set to 256.

- When we significantly decrease ndf, we see the training works very poorly. The discriminator's loss does not manage to decrease enough and it means that it will not recognize images efficiently. This thus means that the generator will not be trained properly and we obtain these bad generated numbers.

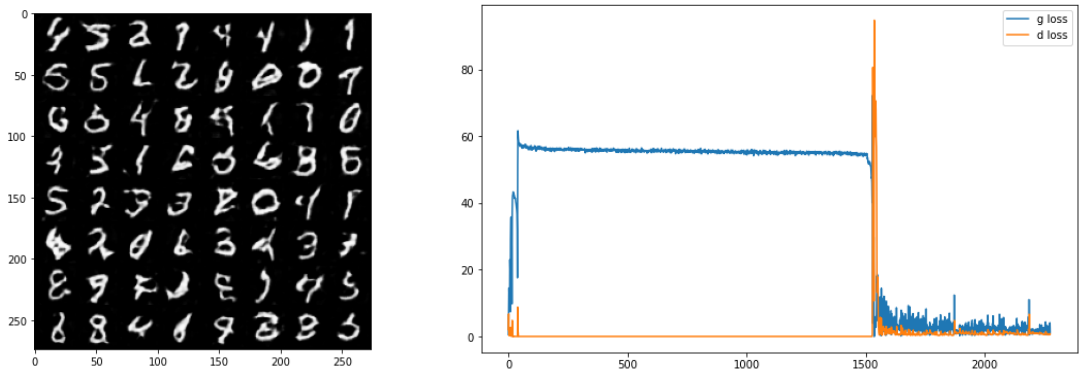


(a) Numbers generated by the GAN.

(b) Losses evolution during training.

Figure (29) GAN training with the parameter ndf (number of channels before the discriminator's last layer) set to 4.

- However when we set ndf to a big number, we see that the discriminator is so efficiently trained that it takes 1500 iterations to make a mistake. This slows the training down but the generator ends up by catching up and the training ends normally.

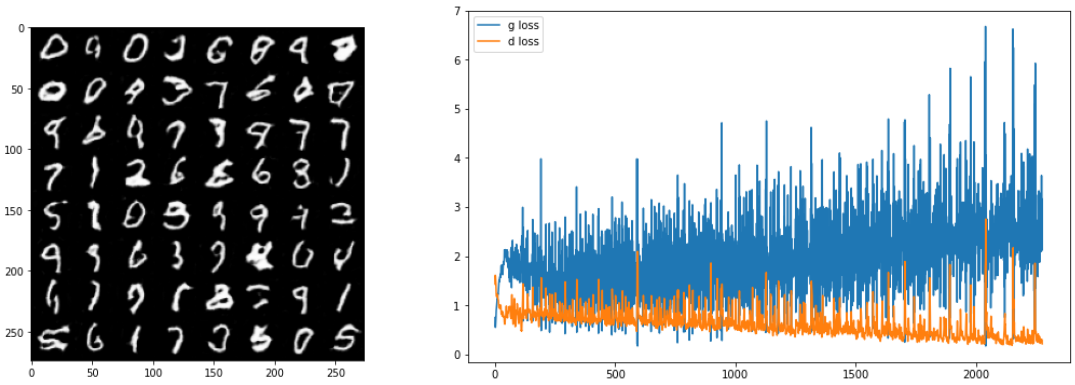


(a) Numbers generated by the GAN.

(b) Losses evolution during training.

Figure (30) GAN training with the parameter ndf (number of channels before the discriminator's last layer) set to 256.

- By initialising the models' weights to their default setting, we see that the training is much less stable. The result does not look much worse but we note that the generator's loss tends to increase over the iterations.

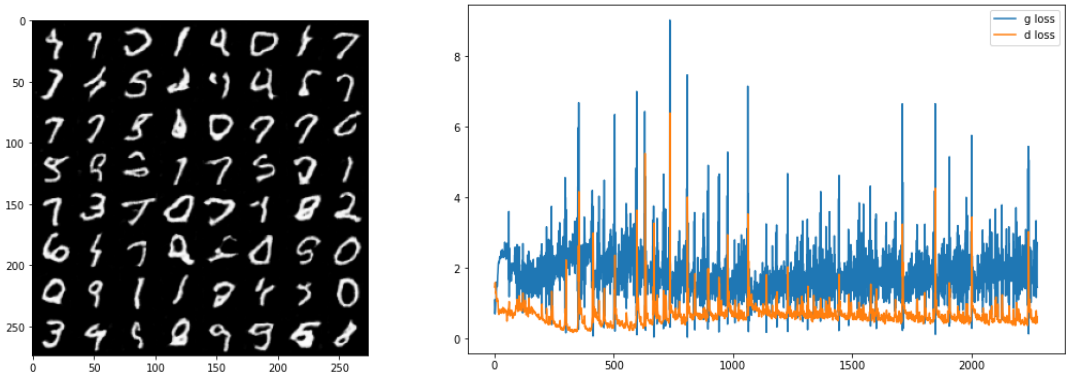


(a) Numbers generated by the GAN.

(b) Losses evolution during training.

Figure (31) GAN training with the models' weights initialised with Pytorch default parameters.

- It seems that the gradient did not vanish at the beginning of the training. We see that the results are similar which was expected since the true loss and the adapted loss are equivalent.

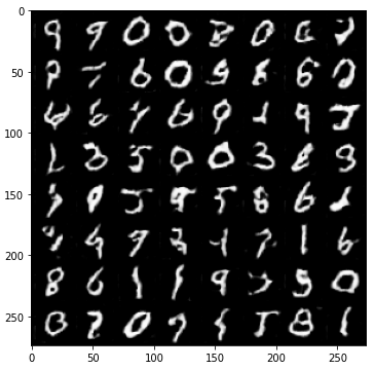


(a) Numbers generated by the GAN.

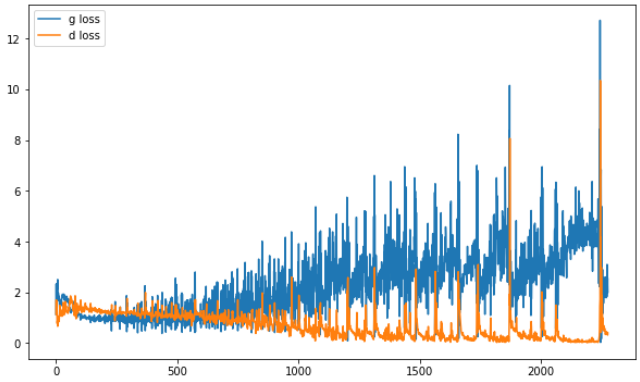
(b) Losses evolution during training.

Figure (32) GAN training with the true theoretical loss for the generator's optimisation.

- Decreasing the generator's learning rate, we note that its loss starts to increase over the iterations. This is because the discriminator learns faster than the generator which means it gets better more efficient over the iterations.



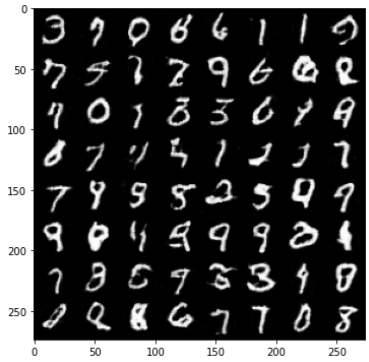
(a) Numbers generated by the GAN.



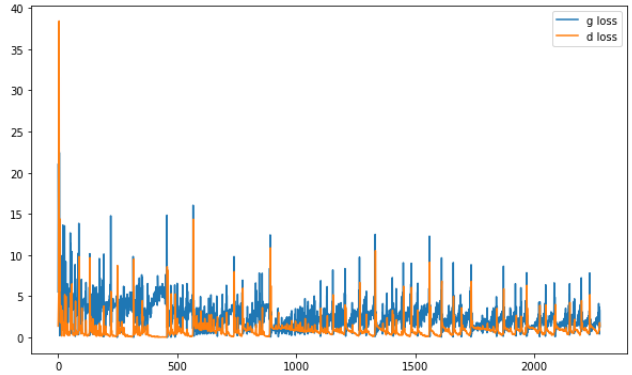
(b) Losses evolution during training.

Figure (33) GAN training with the learning rate for the generator set to 0.05.

- However, when we decrease the discriminator's learning rate, we don't see this same phenomenon. The losses simply seem to be much less stable.



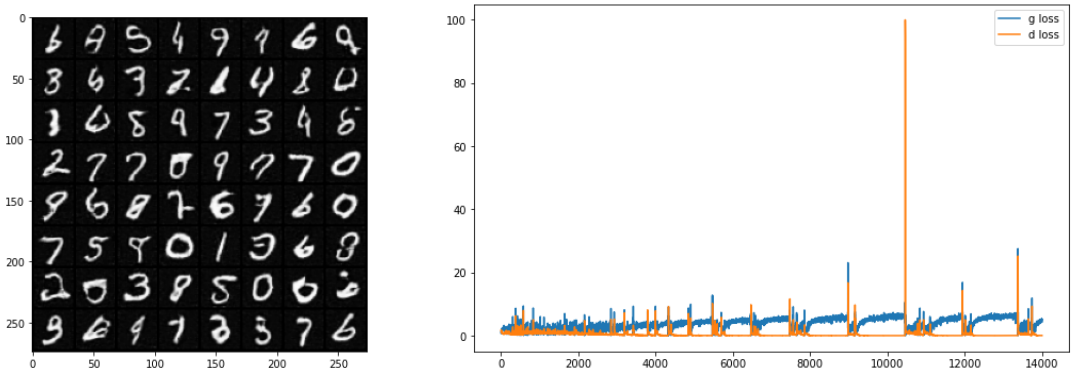
(a) Numbers generated by the GAN.



(b) Losses evolution during training

Figure (34) GAN training with the learning rate for the discriminator set to 0.02.

- When we increase the number of epochs, the result don't seem to improve. In fact, we see through the losses that the training enters a cycle where the generator's loss increases more and more every cycle.

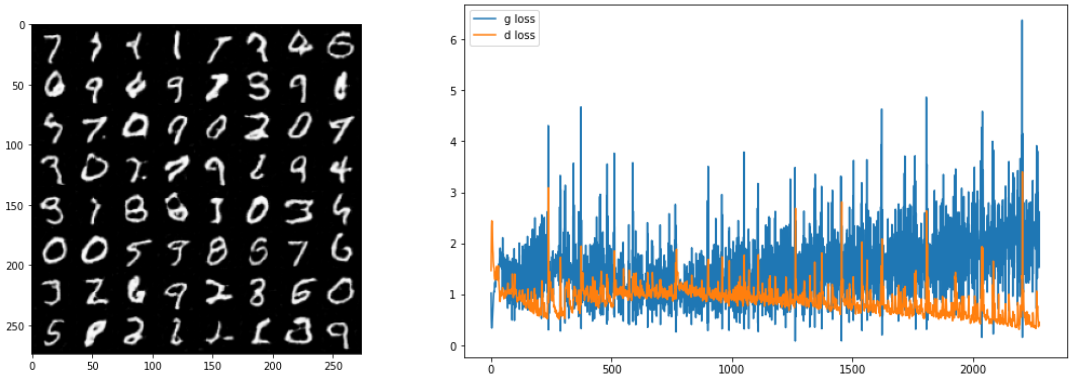


(a) Numbers generated by the GAN.

(b) Losses evolution during training.

Figure (35) GAN training after 30 epochs.

- Using a very small initialisation of size 10 produces pretty good results. Peaks even seem to be smaller that with a size of 100 showing better stability. However we see that the discriminator's loss decreases over the iterations compared to the generator. This might be because it is entering 'Collapse mode'. It means that the generator produces a limited variety of images that are recognized better and better.

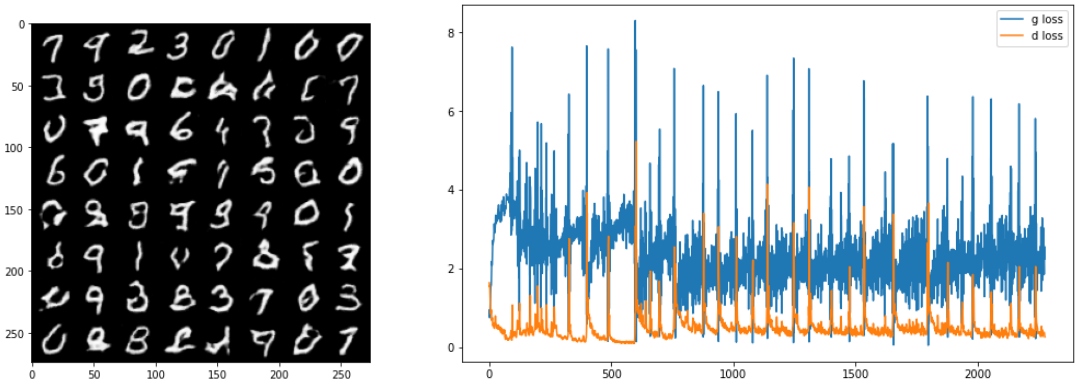


(a) Numbers generated by the GAN.

(b) Losses evolution during training.

Figure (36) GAN training with the parameter n_z (the latent size) set to 10.

- When setting n_z very high, we note that the losses are very unstable. There are more and bigger peaks. However, we don't encounter the collapsing, just like with $n_z = 100$ because we have a bigger variety of initialisation.

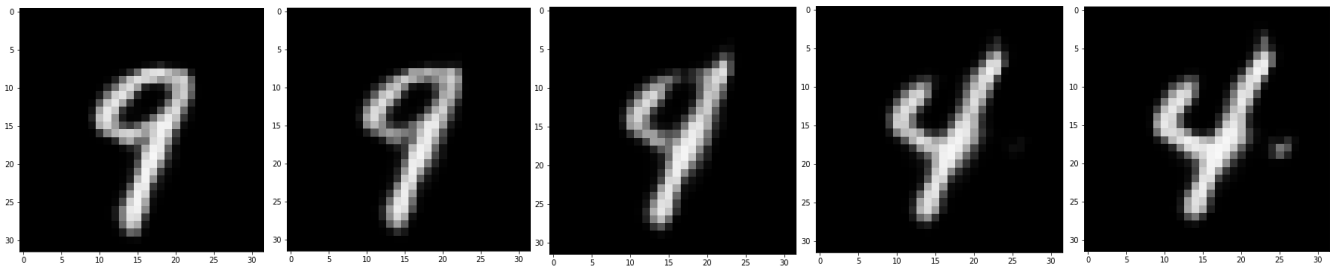


(a) Numbers generated by the GAN.

(b) Losses evolution during training

Figure (37) GAN training with the parameter n_z (the latent size) set to 1000.

- As we can see in Figure 38, the arithmetic operations that we can perform in the latent space allow a control of the generated image. We can make numbers voluntarily ambiguous or control a little bit the shape by making it tend toward another number.

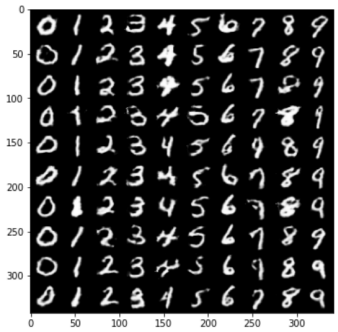


$\alpha = 0.00$ $\alpha = 0.25$ $\alpha = 0.50$ $\alpha = 0.75$ $\alpha = 1.00$

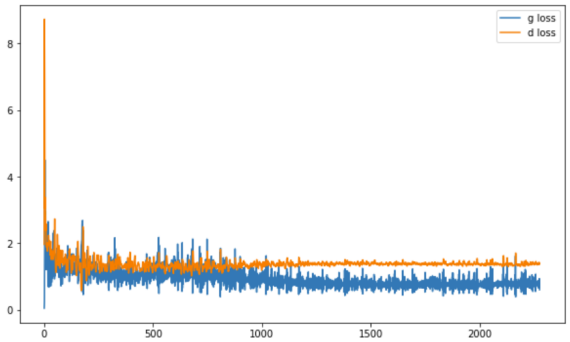
Figure (38) Images generated from the latent vector $\alpha \cdot z_1 + (1 - \alpha) \cdot z_2$ where z_1 is the latent vector of a 9 while z_2 is that of a 4.

4.1 BONUS : Conditional Generative Adversarial Networks

(f) **Comment on your experiences with the conditional DCGAN.**
For all of these experiments we kept $ngf = 64$.

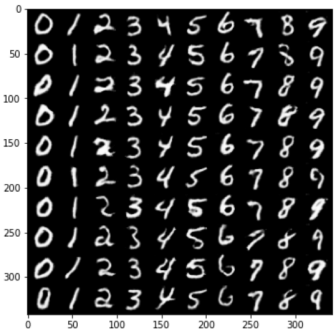


(a) Numbers generated by the GAN.

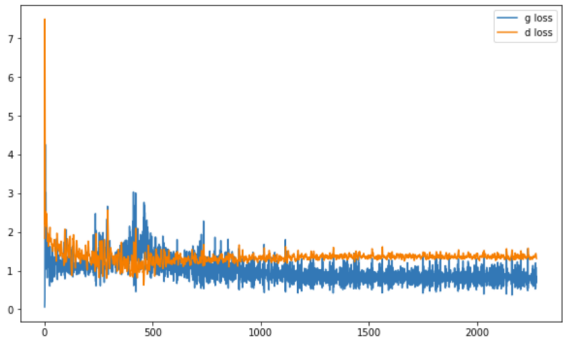


(b) Losses evolution during training

Figure (39) **GAN training with the parameter n_z (the latent size) set to 10.** In (a) each column corresponds to a class while each row corresponds to a particular style. We are able to identify the digits but the intra-class variability of the style is too high.

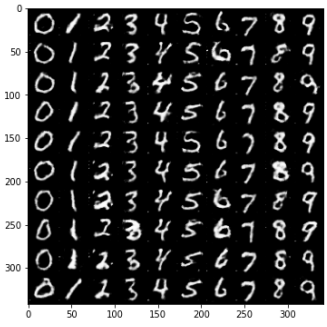


(a) Numbers generated by the GAN.

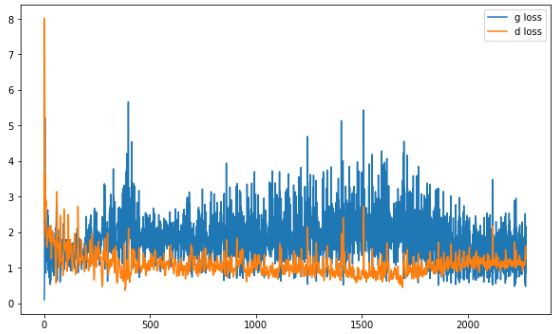


(b) Losses evolution during training

Figure (40) **GAN training with the parameters n_z (the latent size) and ngf respectively set to 100 and 64.** The quality of the images are better than with $n_z = 10$ or $n_z = 1000$.



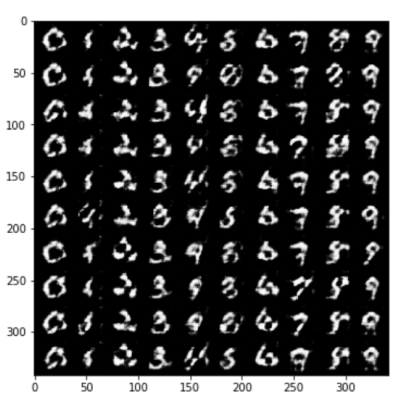
(a) Numbers generated by the GAN.



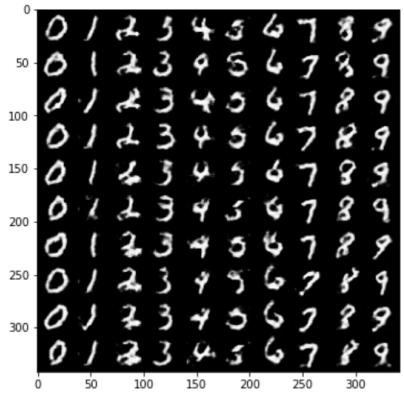
(b) Losses evolution during training

Figure (41) **GAN training with the parameter n_z (the latent size) set to 1000.** We can notice that the quality of the generations is bad and that the loss of the generator is higher than before. Here, the intra-variability of style is also too high.

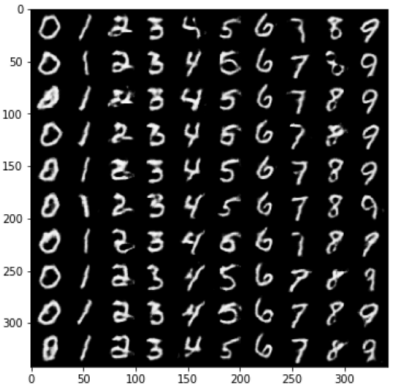
Just like in the unconditional case, we see that using a big n_z makes the losses much less stable. This is because the input can vary too much and the models struggle to adapt. However we notice a slight tendency of the numbers to be more varied which we expected. On the other hand, using a small n_z, while more stable, gives poor results even though we don't have mode collapse this time. With too few possibilities, inputs from one number to the other might be too close and they can thus look like mixes of different numbers.



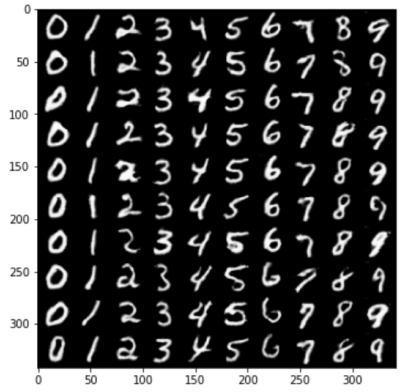
(a) Numbers generated at iteration n°669.



(b) Numbers generated at iteration 1138.



(c) Numbers generated at iteration 1607.



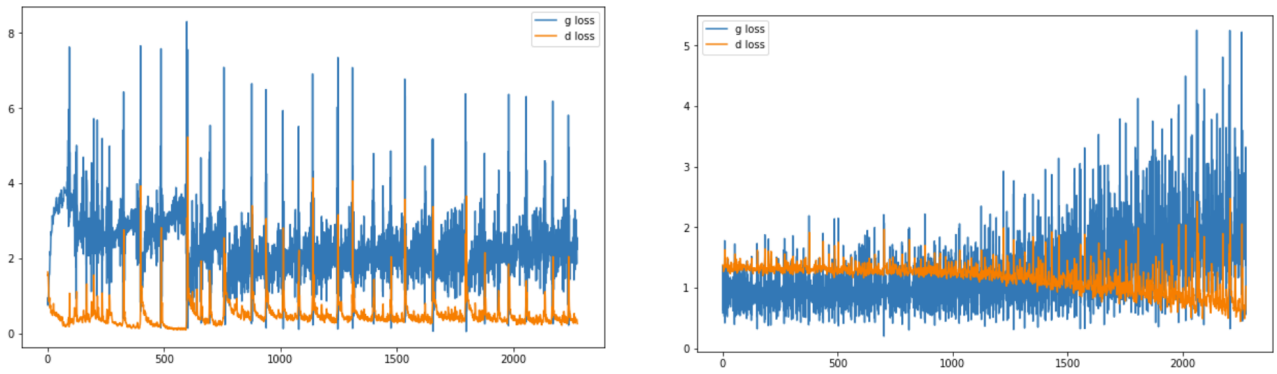
(d) Numbers generated at iteration 2276.

Figure (42) Evolution of the quality of the generated images.

We notice that during the learning, numbers look very different from the start. They are separated at the beginning and the generator learns to make each class without taking into account the others.

(g) **Could we remove the vector y from the input of the discriminator (so having $cD(\mathbf{x})$ instead of $cD(\mathbf{x}, \mathbf{y})$)?**
No, if we do that we would obtain an error because the input of the discriminator needs to be fed with a tensor of fixed size. Modifying the architecture of the discriminator such that it only takes the image $\mathbf{x}$ as input would not be a good idea either because we do not only need to control the realism of the generated image but we also need make sure the generated image corresponds to the label $\mathbf{y}$.

(h) **Was your training more or less successful than the unconditional case ? Why ?**
The training was more successful than the unconditional case because the generated images were more realistic. Indeed, in the unconditional mode the network had to find by itself the clusters corresponding of the classes and therefore the plausible images whereas in the conditional case it is easier for the network to learn how to generate a realistic image of a given class.



(a) (b)

Figure (43) Losses evolution of the (a) unconditional GAN and (b) conditional GAN.

(i) **Test the code at the end. Each column corresponds to a unique noise vector z. What could z be interpreted as here ?**
The vector z contains the information about the style and the background of the image that will be generated from z. It does not contain the class. We can use it to choose a "font" for the numbers we want to generate.

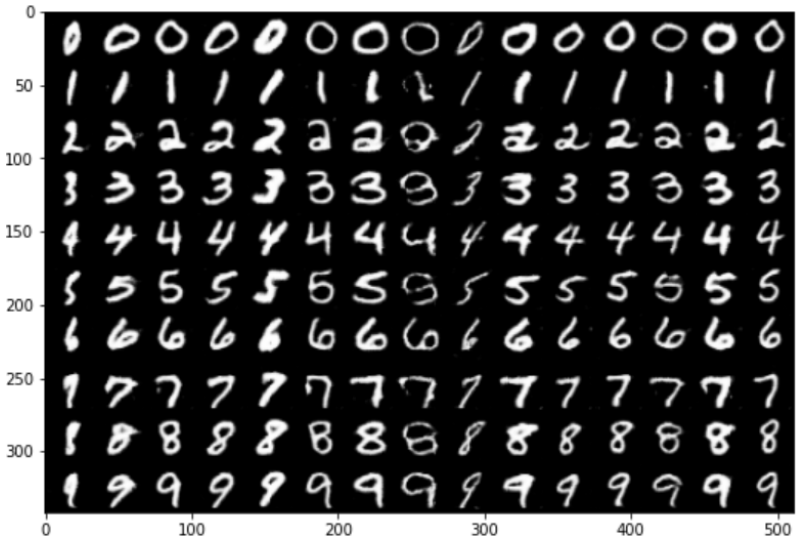


Figure (44) Results obtained by executing the last cell. Here $n_z = 100$.

References

[1] Phoenix Williams and Ke Li. *Art-Attack: Black-Box Adversarial Attack via Evolutionary Art*. 2022. DOI: 10.48550/ARXIV.2203.04405. URL: https://arxiv.org/abs/2203.04405.

[2] Dongbin Na, Sangwoo Ji, and Jong Kim. *Unrestricted Black-box Adversarial Attack Using GAN with Limited Queries*. 2022. DOI: 10.48550/ARXIV.2208.11613. URL: https://arxiv.org/abs/2208.11613.

RDFIA: Homework 4-a,b,c (Bayesian deep learning)

GALMICHE Nathan PIRIOU Victor

February 13, 2023

Abstract

We have studied in previous lectures and practical works many ways to recognize shapes from images. We learned that there are technologies, such as convolutional neural networks, that are able to recognize objects with very high accuracy. The problem is that there are numbers of issues that come with those. For instance, neural networks are not perfectly interpretable. Also, in some use cases, it is desirable to know how confident the model is about its predictions as they may involve human lives. We thus studied a way to have a better understanding of our tools functioning and performances by looking at Bayesian models that estimate the whole posterior distribution whose variance represents the epistemic uncertainty whereas classic neural networks only output the mode of this distribution.

We first started by studying the Bayesian Linear regression. It is a good way to get introduced to them as it represents a very simple case where we can compute the posterior distribution thanks to the existence of its closed form.

During this practical work, this Bayesian framework was also applied on neural networks to do classification. The problem is that the solution is not a closed form anymore and we thus have to estimate it. We did it in different ways. A first way was variational inference which estimates the posterior distribution of the parameters by optimizing the model using a gradient descent. A second way was the use of the Laplace approximation that consists of approximating the posterior by a normal distribution. A third way was by using the Monte-Carlo dropout which involves sampling different parameter configurations where we ignore some of the weights at test time in order to obtain an approximation of the posterior distribution by averaging all of the outputs.

Finally, we applied uncertainty quantification methods to failure prediction and out-of-distribution detection. In order to better predict the uncertainty, we looked at different criteria, other than the most voted class. They include the variation ratio, the mutual information and the entropy.

1 Bayesian Linear Regression

1.1 Linear Basis function model

Question 1.2 : Recall closed form of the posterior distribution in linear case. Then, code and visualize posterior sampling. What can you observe?
The closed form of the posterior distribution in linear is:

$$p(\mathrm{w} \mid \mathrm{X}, \mathrm{Y}) = \mathcal{N}(\mathrm{w} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}),$$

with the mean of the parameters' distribution defined by:

$$\boldsymbol{\mu} = \beta \boldsymbol{\Sigma} \boldsymbol{\Phi}^T \mathrm{Y},$$

and the inverse of the co-variance matrix of the parameters' distribution defined by:

$$\boldsymbol{\Sigma}^{-1} = \alpha \boldsymbol{I} + \beta \boldsymbol{\Phi}^T \boldsymbol{\Phi},$$

where:

- $\mathrm{Y} \in \mathbb{R}^{N \times K}$ is the ground truth. Here N is the number of training examples while K is the number of dimensions of the ground truth,
- $\Phi \in \mathbb{R}^{N \times (p+1)}$ is the matrix containing the data of dimensionality p and the bias,
- $\mathbf{W} \in \mathbb{R}^{(p+1) \times K}$ is the matrix containing the parameters,
- α is the parameter controlling the prior $p(\mathrm{w} \mid \alpha) = \mathcal{N}\left(\mathrm{w}; 0, \alpha^{-1} I\right)$,
- $\beta = \frac{1}{2\sigma^2}$ is the parameter controlling the likelihood $p\left(\mathrm{y}_i \mid \mathrm{x}_i, \mathrm{w}\right) = \mathcal{N}\left(\Phi_i^T \mathrm{w}, \beta^{-1}\right)$.

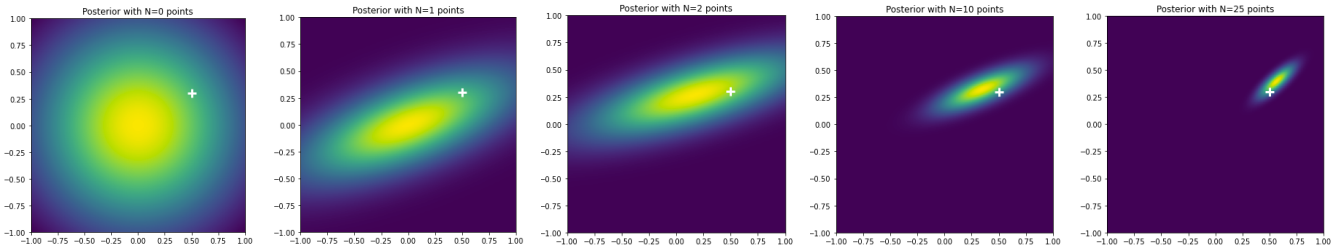


Figure (1) Posterior samplings with different number of points.

In Figure 1, the more data points we have the more the variance of the parameters' distribution is reduced and the closer the estimate gets to the ground truth (represented in white). In other words, the more data points we have the more the aleatoric uncertainty over the parameters is reduced.

Question 1.3 : Recall and code closed form of the predictive distribution in linear case.
The closed form of the predictive distribution in linear case is defined by:

$$p\left(y \mid x^*, \mathcal{D}, \alpha, \beta\right) = \mathcal{N}\left(y; \mu^T \boldsymbol{\Phi}\left(\mathrm{x}^*\right), \frac{1}{\beta} + \boldsymbol{\Phi}\left(\mathrm{x}^*\right)^T \boldsymbol{\Sigma} \boldsymbol{\Phi}\left(\mathrm{x}^*\right)\right),$$

where $\mathcal{D}$ is the dataset and x^* a new input.

Question 1.4 : Based on previously defined `f_pred()`, predict on the test dataset. Then visualize results using `plot_results()` defined at the beginning of the notebook.

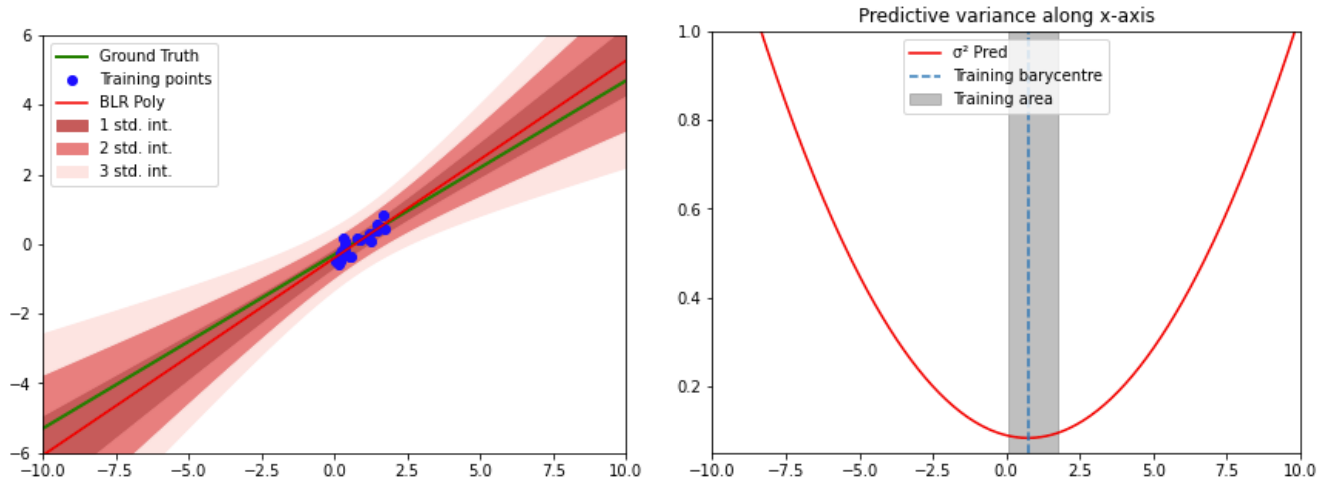


Figure (2) Visualization of the predictive distribution of a test dataset.

Question 1.5 : Analyse these results. Why predictive variance increases far from training distribution? Prove it analytically in the case where $\alpha = 0$ and $\beta = 1$.
We can notice that:

- the further we are from the data, the bigger the variance is,
- the variance is minimal at the barycenter of the training points.

Let's prove it analytically in the case where $\alpha = 0$ and $\beta = 1$.

We have:

$$\begin{aligned}\Sigma^{-1} &= \alpha \boldsymbol{I} + \beta \Phi^T \Phi \\ &= \Phi^T \Phi \\ &= \begin{pmatrix} 1 & \dots & 1 \\ x_1 & \dots & x_N \end{pmatrix} \cdot \begin{pmatrix} 1 & x_1 \\ \dots & \dots \\ 1 & x_N \end{pmatrix} = \begin{pmatrix} N & \sum_i x_i \\ \sum_i x_i & \sum_i x_i^2 \end{pmatrix}.\end{aligned}$$

This implies that:

$$\Sigma = \frac{1}{det(\Sigma^{-1})} \begin{pmatrix} \sum_i x_i^2 & -\sum_i x_i \\ -\sum_i x_i & N \end{pmatrix}.$$

Then, the variance of a new data point x^* is obtained by:

$$\begin{aligned}\Phi\left(\mathrm{x}^*\right)^T \Sigma \Phi\left(\mathrm{x}^*\right) &= \frac{1}{det(\Sigma^{-1})} \begin{pmatrix} 1 & x^* \end{pmatrix} \cdot \begin{pmatrix} \sum_i x_i^2 & -\sum_i x_i \\ -\sum_i x_i & N \end{pmatrix} \cdot \begin{pmatrix} 1 \\ x^* \end{pmatrix} \\ &= \frac{1}{det(\Sigma^{-1})} \cdot \begin{pmatrix} \sum_i x_i^2 - x^* \sum_i x_i & -\sum_i x_i + N x^* \end{pmatrix} \cdot \begin{pmatrix} 1 \\ x^* \end{pmatrix} \\ &= \frac{1}{det(\Sigma^{-1})} \cdot \Big(\sum_i x_i^2 - 2x^* \sum_i x_i + N (x^*)^2 \Big) \\ &= \frac{1}{det(\Sigma^{-1})} \cdot \Big(\sum_i (x_i - x^*)^2 \Big).\end{aligned}$$

As the determinant is constant, it is now clear that the further we are from the training examples the bigger $(x_i - x^*)^2$ is and the bigger the variance is.

Bonus Question : What happens when applying Bayesian Linear Regression on the following dataset?

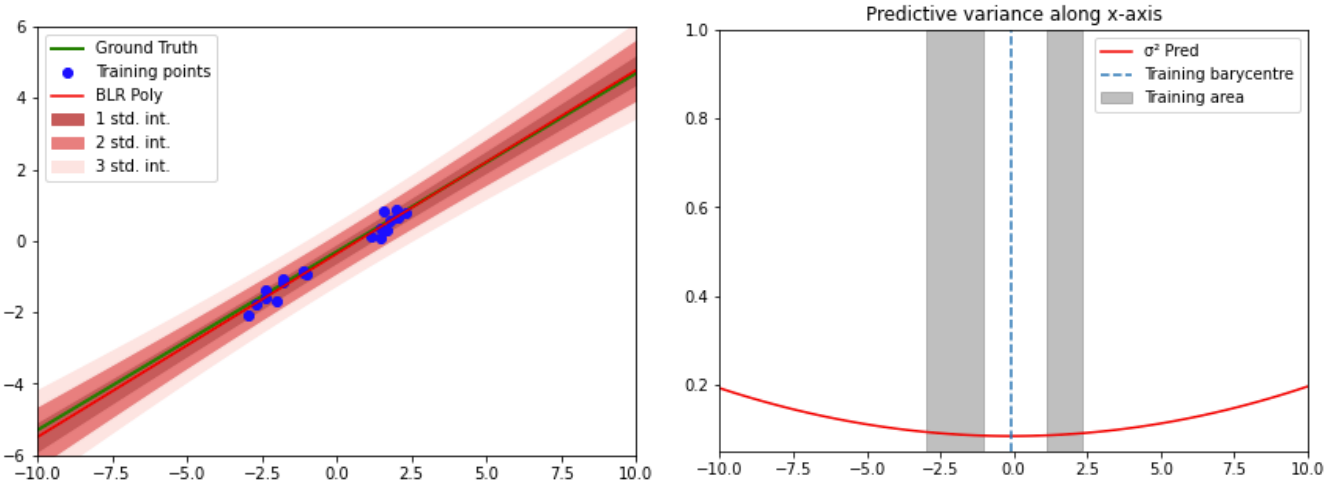


Figure (3) Visualization of the predictive distribution of a test dataset.

On Figure 3 we can see that the predictive variance is still minimized at the barycenter of the training points. This fact is not desirable this time. Indeed, we clearly have two separated clusters of points so the variance should be minimized around them and bigger the further we get away from these clusters.

1.2 Non Linear Models

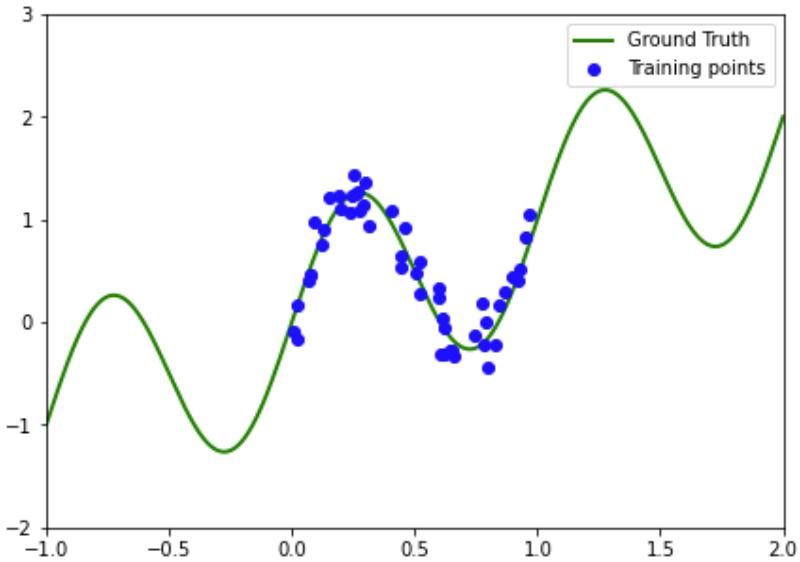


Figure (4) Dataset of an increasing sinusoidal curve.

1.2.1 Polynomial basis functions

Question 2.2 : Code and visualize results on sinusoidal dataset using polynomial basis functions. What can you say about the predictive variance?

On Figure 5, the predictive variance gets bigger as we move away from the training points. This is expected, however, it looks like the model does not capture the periodicity of the sinusoidal ground truth because the model has no

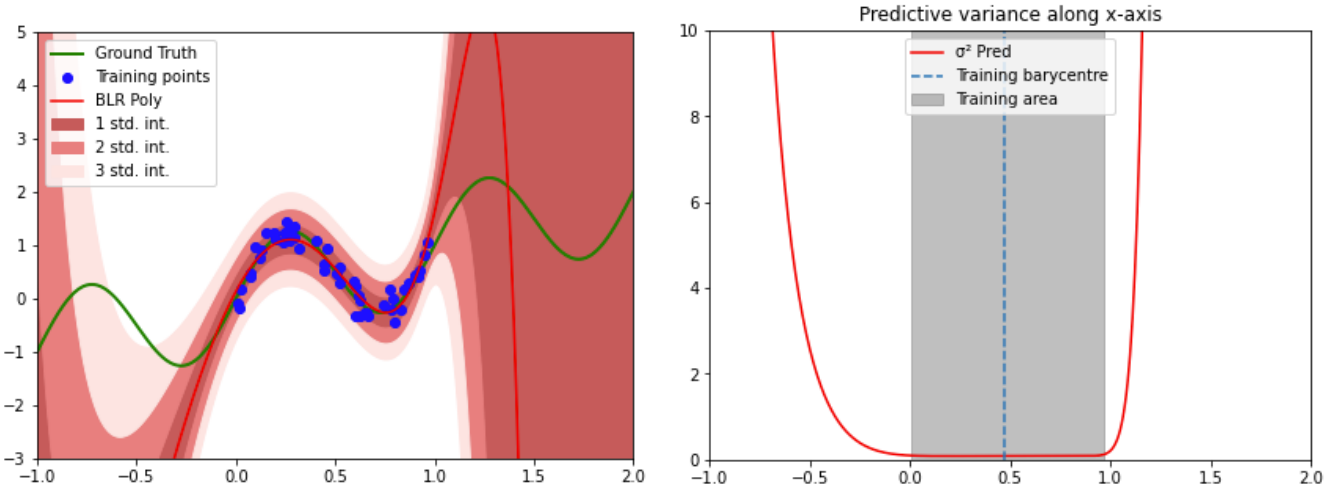


Figure (5) Visualization of the predictive distribution of a test dataset.

data to know which values to predict. It thus predicts a continuation of the first data that do not give information on the previous and next periods. We thus notice that the ground truth gets very far from the prediction to the point where the variance does not even include it which is very bad if we were to predict in this range.

1.2.2 Gaussian basis functions

Question 2.4 : Code and visualize results on sinusoidal dataset using Gaussian basis functions. What can you say this time about the predictive variance?

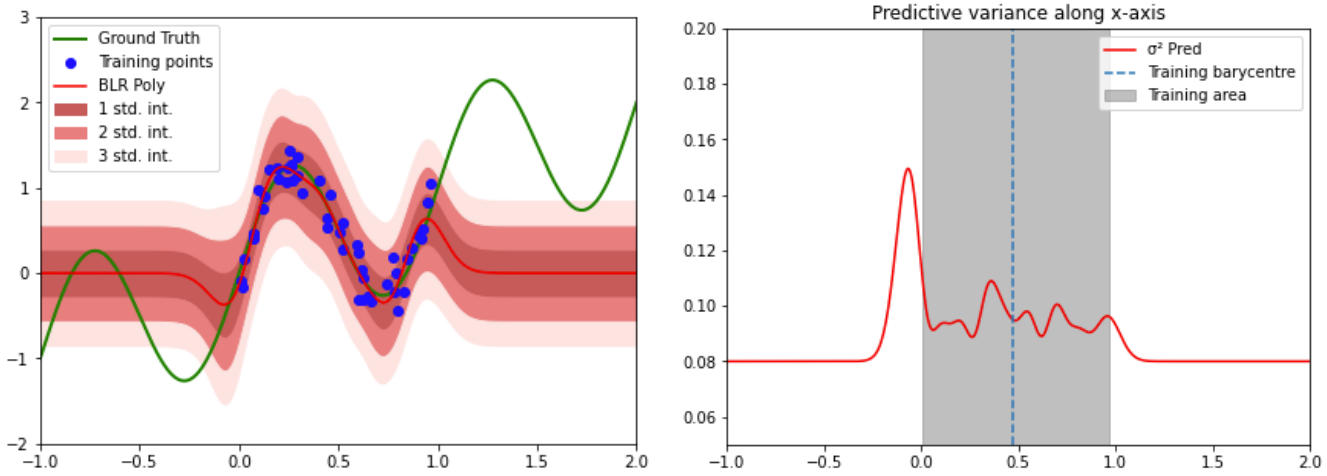


Figure (6) Visualization of the predictive distribution of a test dataset.

Figure 6 shows that compared to polynomial basis functions the use of the Gaussian basis functions is associated to a smaller variance and a constant prediction in the area where we are far from the training points. It fits very well to the data with the variance fluctuating depending on the concentration of the data. However this time, in the zones where we do not have any data, we see that the variance does not really increase and keeps a pretty low value while the prediction is constant. This not what we want because we see that we are getting really far from the ground truth.

Question 2.5 : Explain why in regions far from training distribution, the predictive variance converges to this value when using localized basis functions such as Gaussians.
This is explained by the fact that the epistemic uncertainty $\Phi(\mathrm{x}^*)^T \Sigma \Phi(\mathrm{x}^*)$ converges to 0 so $\sigma^2_{\text{pred}} = f(\mathrm{x}^*) =$

$\beta^{-1} + \Phi(\mathrm{x}^*)^T \Sigma \Phi(\mathrm{x}^*)$ becomes $\sigma^2_{\text{pred}} = f(\mathrm{x}^*) = \beta^{-1}$ and is therefore reduced to the aleatoric uncertainty. In our case, $\beta^{-1} = 0.08$ which fits the graph we obtained.

2 Approximate inference

2.1 Bayesian Logistic Regression

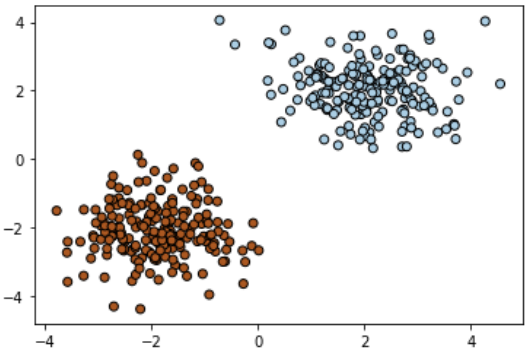


Figure (7) Visualization of the used dataset that is linearly separable.

Question 1.1 : Analyze the results provided by previous plot. Looking at $p(\mathrm{y} = 1|\boldsymbol{x}, \boldsymbol{w}_{\mathbf{MAP}})$, what can you say about points far from train distribution?
We can see on Figure 8 that the uncertainty does not increase when we are far from the training data. In fact, as we obtain a point wise estimate of the parameters, it is only able to say in which class each point belongs and thus cannot take into account the amount of information we originally had around.

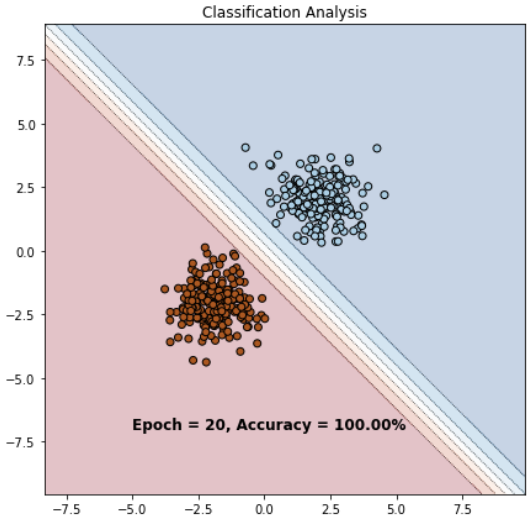


Figure (8) Decision boundary obtained after having trained a Bayesian logistic regression using the plug-in approximation.

Question 1.2 : Analyze the results provided by previous plot. Compared to previous MAP estimate, how does the predictive distribution behave?

We can see on Figure 9 that this time the uncertainty increases when we are far from the training data. Computing the Hessian matrix allows to obtain an evaluation of the variance at each point and we are thus able to show some sort of uncertainty. This gaussian approximation is not perfect since it is only done at the mode of the distribution. Consequently, the global properties of the distribution are ignored.

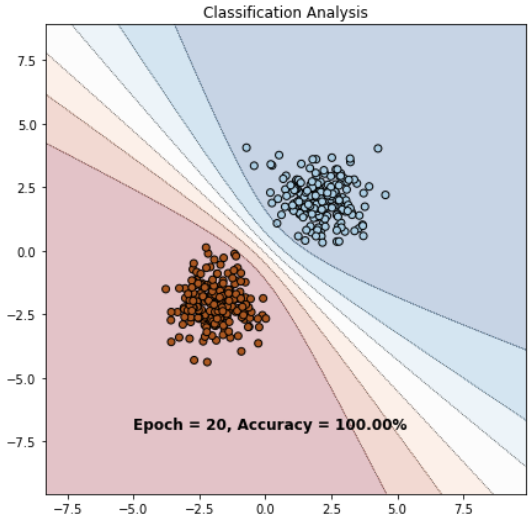


Figure (9) Decision boundary obtained after having trained a Bayesian logistic regression using the Laplace approximation.

Question 1.3 : Analyze the results provided by previous plot. Compared to previous MAP estimate, how does the predictive distribution behave?

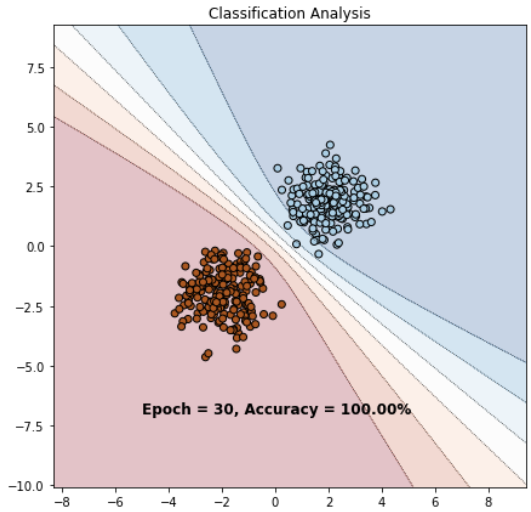


Figure (10) Decision boundary obtained after having trained a variational logistic regression.

This technique allows to estimate the posterior distribution that cannot be evaluated analytically. This means that we are able to evaluate for each point which class to put it in and how certain we are. We obviously find similar results as the previous technique because the data is pretty simple but the estimation is correct.

2.2 Bayesian Neural Networks

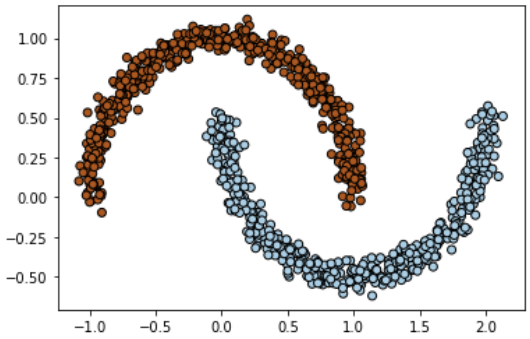


Figure (11) Visualization of the two moons' dataset whose classes are not linearly separable.

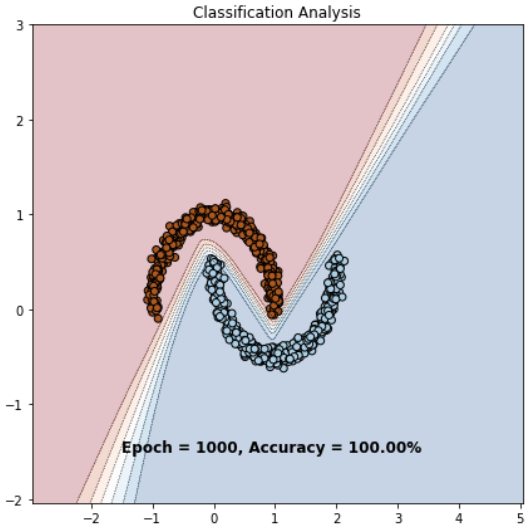
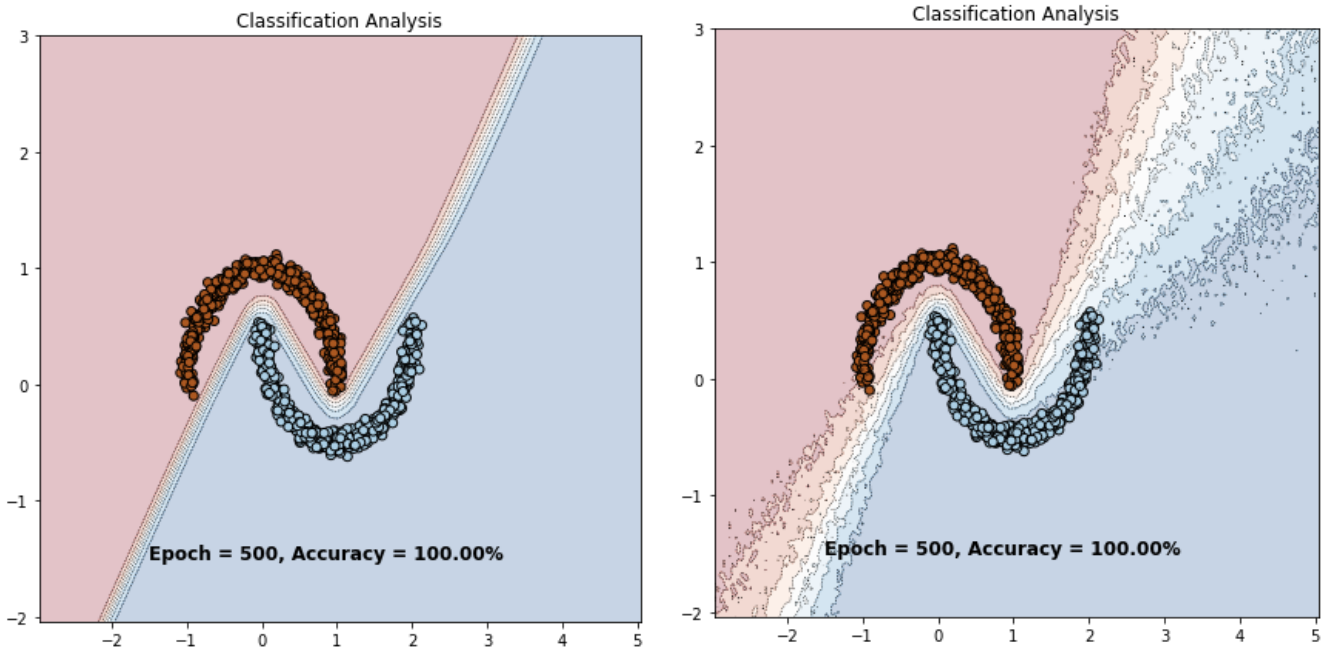


Figure (12) Decision boundary obtained after having trained a variational MLP.

2.2.1 Monte Carlo Dropout

Question 2.1 : Again, analyze the results showed on plot. What is the benefit of MC Dropout variational inference over Bayesian Logistic Regression with variational inference?
First we notice that the Bayesian Logistic Regression shows good results as we see that the uncertainty increases when we get further from the data. It is still precise where there is data.



(a) Without Monte Carlo Dropout

(b) With Monte Carlo Dropout

Figure (13) Decision boundaries obtained after having trained a MLP with dropout layers.

However using Monte Carlo dropout gives us very different results this time. The first thing we notice is that the limits between the classes is not as straight. We see that there are some irregularities that come from the dropout. Those are not very important but we also notice that the uncertainty is much more important with this technique as some is introduced right by the data that are at the extreme. This could seem interesting since the model cannot be precise when we are far from any data but when the data is separable like this, we should not need to introduce uncertainty close to the data. Here, we see that the extreme ends of the moons are less certain than the rest, which is not needed.

The rate of the dropout can also be modified to lessen or heighten this effect.

3 Applications of uncertainty

3.1 Monte-Carlo Dropout on MNIST

Question 1.1 : What can you say about the images themselfs. How do the histograms along them helps to explain failure cases? Finally, how do probabilities distribution of random images compare to the previous top uncertain images?

In general in this dataset, the number we are looking at in the image is pretty big and fully contrasted so it is clear. However when looking at the worst images, it can be hard even for our human brain to understand what number does it represent. We thus expect the model to struggle. This is very visible in histograms when you have different classes with similar probabilities. Also, when looking more into the details of each classes, we look at histograms showing the confidence of every prediction for the class. We notice that when the model struggles, the histogram is either pretty balanced which means it is sometimes confident and sometimes not, or the histogram is dominant on the lower confidences.

This is very different than the good images where we notice that there is only one bar on each histogram : only one class predicted, always the same level of confidence (very high for the right class, very low for the wrong ones).

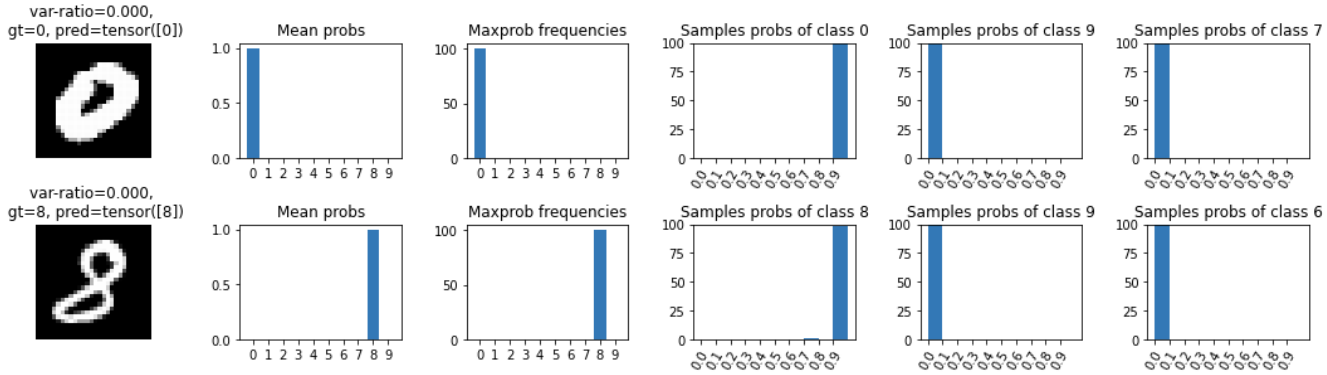


Figure (14) Random images with their var-ratios value.

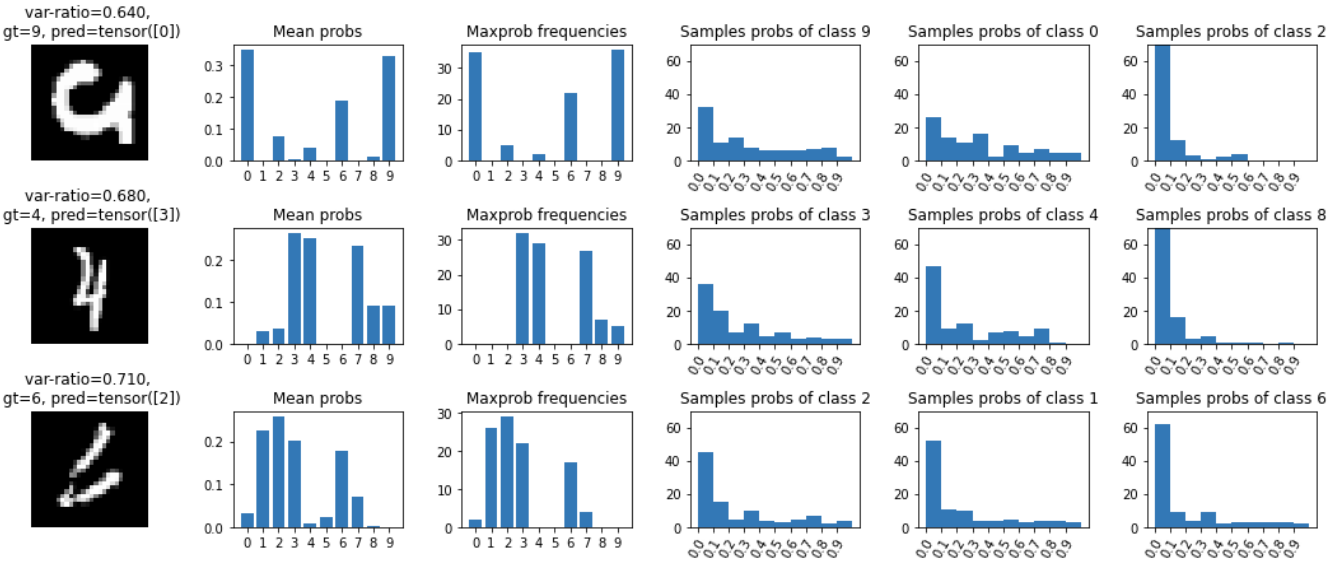


Figure (15) Top-3 most uncertain images along with their var-ratios value.

4 Failure prediction

Question 2.1 : Compare the precision-recall curves of each method along with their AUPR values. Why did we use AUPR metric instead of standard AUROC?

Globally, we can see on Figure 16 that the area under the curve of ConfidNet is much more big than that of the other methods. More precisely, ConfidNet has both an higher precision and a higher recall values. In sum, it is much more successful than the MCP and MCPDropout methods.

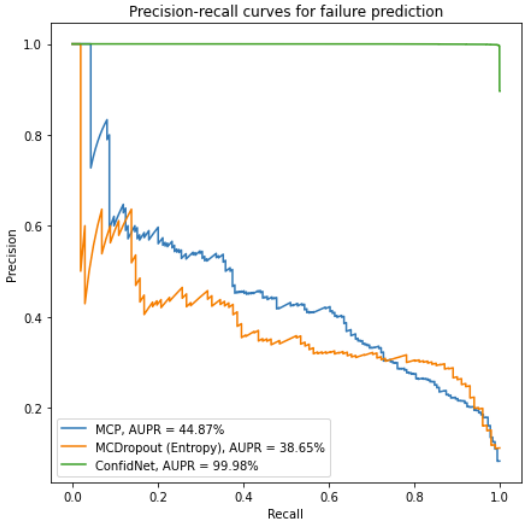


Figure (16) Precision-recall curves along with their AUPR values for failure prediction.

The AUPR metric does not make use of the True Negatives at all. Hence, in our case the AUPR metric is more suitable than the standard AUROC because the classes are imbalanced. The AUROC metric would ignore a lot of errors, which is not what we want.

5 Out-of-distribution detection

Question 3.1 : Compare the precision-recall curves of each OOD method along with their AUPR values. Which method perform best and why?

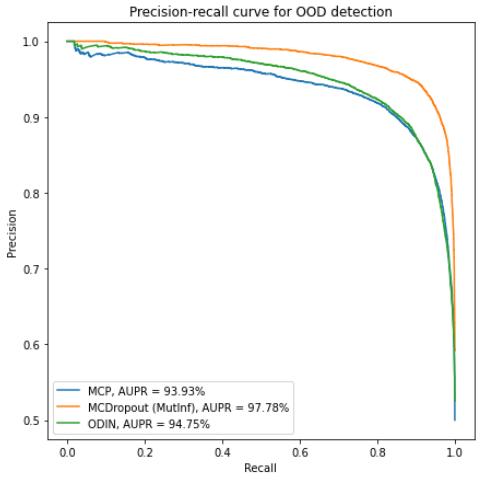


Figure (17) Precision-recall curves along with their AUPR values for out-of-distribution detection.

The best method is the MCdropout which is expected because we get more information thanks to the sampling. The ODIN idea improves a little bit the score of the simple MPC but it is still worst than the MCDropout.