Computer Vision Project Report: In the name of Deep learning

Faustine GINOUX and Guillaume LAMINE
Master in Artificial Intelligence

June 18th, 2019

1 Introduction

The PASCAL VOC-2009 dataset consists of color images of various scenes with different object classes. For the auto-encoder (Section 2) and the classification (Section 3), only a subset of the images is used. After trying out with sheep and cows, then with birds and cats, we finally opted for boats and cats. Sheep and cows were abandoned as the images are too similar and for both classes, not a lot are available. Bird were put aside as the images are not always of good quality, and birds are sometimes hard to spot. Although there are less images of boats than cats, this selection proved to be the most successful. The final training set consists of 170 images of boats and 266 images of cats, resized to 128x128 pixels (in color, so total shape is 128x128x3 for the three RGB color channels). Similarly, the validation set contains 155 reshaped images of boats and 277 of cats. Some example of images are displayed in Figure 1.

Figure 1: **Example of images of the two classes (from the training set).** These images are "good" images. On some images, boats and cats are less obvious.

2 Auto-encoders

2.1 PCA vs Auto-encoder

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation seeking the directions of maximum variance in the data. It converts a set of observations of possibly correlated variables (x) into a set of values of linearly uncorrelated variables ($z = E^T x$) called **principal components** (PCs). The first PC accounts for the most variability in the dataset as possible, the last one for the least. It can be used to simplify datasets by reducing them to lower dimensions, while minimizing the information loss, which can be an effective technique against the curse of dimensionality. PCA uses the eigenvectors corresponding to the q largest eigenvalues of the covariance matrix of the mean-centered dataset to reconstruct the data in q dimensions ($\tilde{x} = Ez$, and q smaller than the original dimension). The eigenvalue is a measure of the variance in the dataset that can be explained by the direction of the corresponding eigenvector. PCA is most effective if the variance is concentrated within a few PCs, which is not exactly the case in our data set (Figure 2).

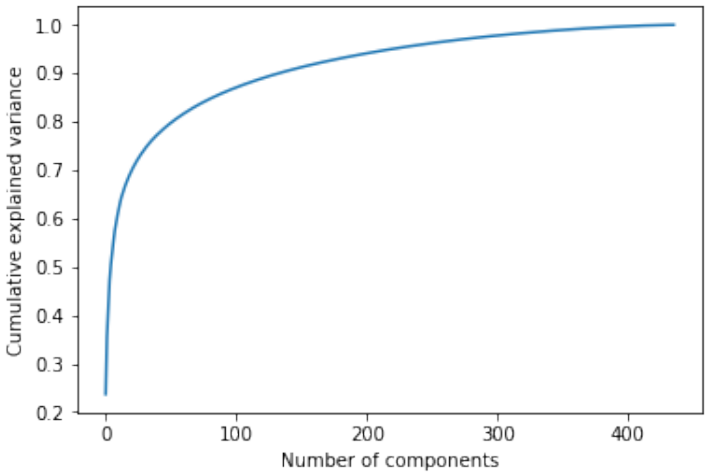


Figure 2: **Cumulative explained variance ratio.** 50% of the variance is explained by 5 PCs, 75% by 33 PCs, 90% by 134 PCs, 95% by 221 PCs, 99% by 355 PCs and 100% by all 436 PCs.

An auto-encoder neural network is an unsupervised learning algorithm that applies backpropagation and attempts to replicate its input at its output (also considered as **self-supervised** learning). Thus, the size of its input will be the same as the size of its output. However, such as PCA, it can be used to reduce the dimensionality of the dataset. When the number of neurons in the hidden layer is less than the size of the input, the auto-encoder learns a compressed representation of the input. The auto-encoder is comprised of an encoder followed by a decoder. The encoder maps an input to a hidden representation, which can be useful for extracting features from data, by learning a low-dimensional representation of the data. The decoder attempts to reverse this mapping to reconstruct the original input.

An auto-encoder with linear activation function and a single hidden fully-connected layer can approximate PCA. There is a linear relationship between the data and the learned representation, and the objective functions of PCA and of the linear auto-encoder (LAE) are the same, minimizing the reconstruction error (**mean-squared error**, MSE). The inputs x are projected to a lower dimension by $z = f(Wx)$ and reconstructed in the higher dimension by $\tilde{x} = g(Vz)$. The goal is to minimize the reconstruction error $\sum(x - \tilde{x})^2$, which yields $\sum(x - VWx)^2$ if f and g are linear. The optimal solution is thus PCA.

Additionally, if the LAE hidden layer has a size q, the weights of the LAE span the same vector subspace as the one spanned by the first q components of PCA (Figure 3), and the output of the LAE is an orthogonal projection onto this subspace. Note however that the LAE weights are different from the PCs ($W \neq E^T$ and $V \neq E$). In general, the weights are not orthogonal, but the PCs can be recovered using Singular Value Decomposition[1].

From visual evaluation (Figure 4) and from reconstruction error comparison (Table 1), PCA is better than LAE to reconstruct the images of this dataset. In theory, non-linear autoencoders are better at reconstructing the original dataset than PCA when q is small, but the error converges as q increases. For very large data sets this difference is larger, meaning that the same accuracy can be achieved with less neurons than components for PCA and hence a smaller data set. An advantage of PCA is that we can choose the number of PCs based on the amount of variability they explain; with AE, we don't know how many neurons to set in the hidden layer.

2.2 Non-linear and convolutional auto-encoder

One strength of auto-encoders is that it is not limited to linear transformation. Instead of linear activation functions, non-linear functions such as *tanh* or *sigmoid* can be used. As the inputs of the auto-encoder are images, it makes sense to use convolutional layers. Convolutional Neural Networks (CNNs) are deep neural networks particularly efficient for image analysis. They differ from fully-connected net-

[1] *From Principal Subspaces to Principal Components with Linear Autoencoders*, E. Plaut, 2018.

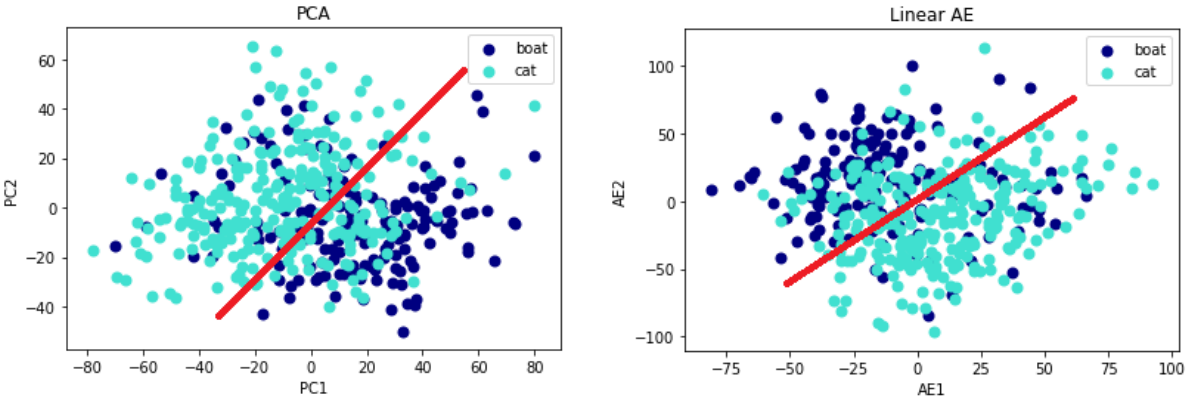


Figure 3: **Projection of the images in two dimensions.** Two PCs were used for PCA and a LAE with two hidden neurons. We can notice that the space spanned is similar, with a distribution approximately symmetrical around the red axis. The weight values are different, as $PC1 \neq AE1$ and $PC2 \neq AE2$. Also, contrary to PCA, the neurons of LAE have no particular meaning in their numbering. Here, $AE1$ seems to be more similar to $PC2$ and $AE2$ to $PC1$, hence the symmetrical aspect of the projections.

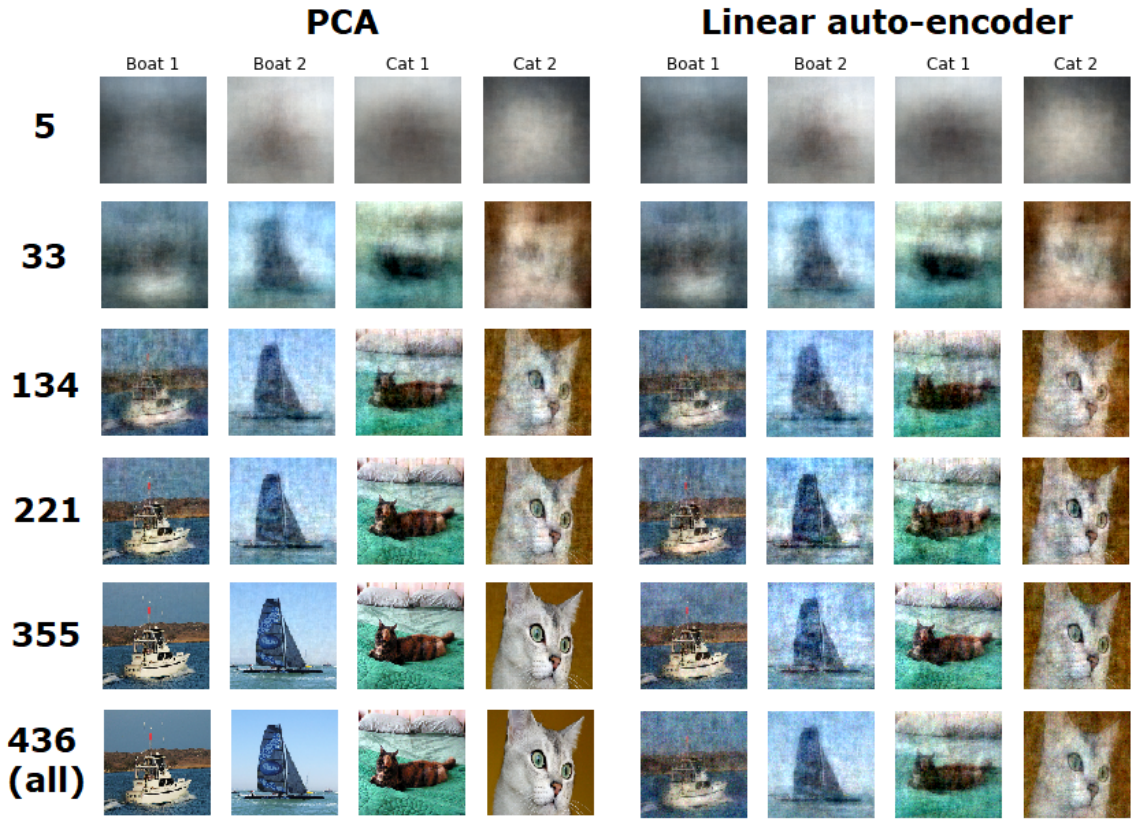


Figure 4: **Comparison of the reconstruction of images by PCA and LAE.** The number on the left indicates the number of principal components (PCA) and hidden neurons (LAE). PCA: the more components are added, the lower the reconstruction error is; for 436 PCs - all of them - the original images are reconstructed. For LAE, adding more neurons decreases the error but not as well as adding more PCs in PCA. For LAE, 50 epochs and a batch size of 32 are used for training.

works by exploiting local connectivity, which can prevent overfitting if no regularization is performed. Convolutional layers take advantage of the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns. For images, this closely resembles the biological process of the visual cortex: the first layer of the network learns filters for capturing "primitive" blob and edge features,

PCs / neurons	PCA error	LAE error
5	3.58	3.64
33	1.80	1.96
134	0.73	1.14
221	0.36	1.26
255	0.07	0.84
436	$10^{-13} \approx 0$	0.79

Table 1: **Reconstruction error (MSE, values in %) for PCA and LAE in function of the number of principal components and hidden neurons.** For a small number of PCs/neurons, the reconstruction error is similar. As the number of PCs increases, the error decreases drastically, which is not the case when the number of neurons increases.

which are then processed by deeper network layers combining the early features to form higher-level image features that are better suited for recognition tasks, as they have a richer image representation. A convolutional auto-encoder (CAE) learns to encode the input in a set of simple features and then tries to reconstruct the input from them. The main difference between CAEs and CNNs is that CNNs are trained end-to-end to learn filters and combine features with the aim of classifying their input (supervised learning). Instead, CAEs are trained only to learn filters able to extract features that can be used to reconstruct the input. Rather than manually engineering the convolutional filters used in a CNN, the CAE lets the model learn the optimal filters that minimize the reconstruction error (MSE). Once the filters to encode the images are optimized, the coding variables can be used to train a classifier (see section 3).

CAEs with two to four convolutional layers in the encoder were tried, yielding respectively features of size 32x32x8, 16x16x8 and 8x8x8 in the lower dimension (original space was 128x128x3). This is equivalent to a dimension division by 6, 24, and 96 respectively. Their reconstruction errors are compared and summarized in Table 2. The larger the encoding dimension is, the better the reconstruction: less information is lost. This is the opposite of classical CNNs, for which the deeper the network is, the better the output (although the output generally is classification, not reconstruction). A CAE with two convolutional layers of size 16 and 8 was kept and further analyzed, as it is a good trade-off between the minimization of the reconstruction error and computation time (Figure 5).

The reconstruction is much better with a CAE than a LAE. One reason is that the convolutions exploit the 3D structure of the images. In sandard auto-encoders, the images must be passed as one single vector and the network is built following the constraint on the number of inputs. This introduces redundancy in the parameters, forcing each feature to be global and span the entire visual field[2], while CAEs do not and allow local features to be learned.

Number of conv. layers	Layer dimension	Code dimension	Reconstruction error
2	64 - 8	8192	0.64
2	32 - 8	8192	0.66
2	**16 - 8**	**8192**	**0.74**
3	64 - 32 - 8	2048	0.96
3	64 - 16 - 8	2048	1.07
3	32 - 16 - 8	2048	1.12
3	16 - 8 - 8	2048	1.13
4	64 - 32 - 16 - 8	512	1.56

Table 2: **Reconstruction error (%) for different CAE architectures.** All CAEs were trained for 50 epochs with a batch size of 32. All filters had a size 3x3. Filters of size 5x5 and 7x7 were tried as well, but the computation time increased without observing any significant improvement in the reconstruction results. The convolutional layers have a ReLU activation function, except the last decoding one which takes a *sigmoid* function. The model optimizer is *adam*.

[2] *Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction*, J. Masci, U. Meier, D. Ciresan, and J. Schmidhuber, 2011.

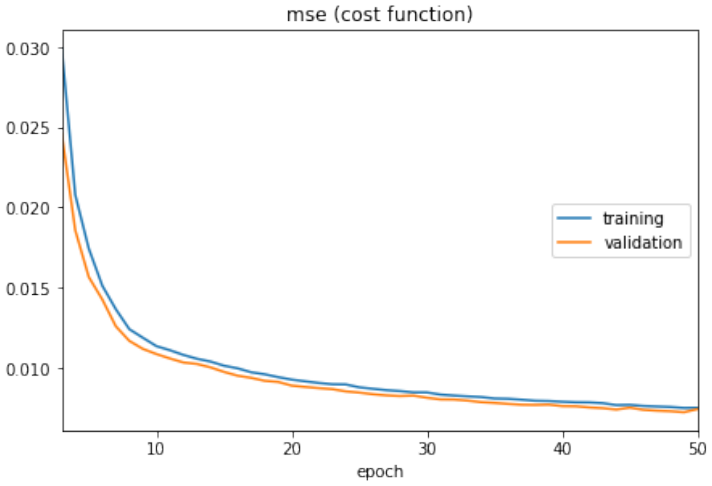


Figure 5: **Evolution of the reconstruction error (MSE) of the training and validation set.** The two errors converge to 0.007, showing that the auto-encoder model does not overfit the training data.

As convolutional layers are used, **ReLU** activation function performs the best. For the last convolutional layer in the decoder, different activation functions were tried: *linear*, *sigmoid*, *tanh*, *relu*, *softmax* and all yield similar reconstruction errors except *softmax* (Figure 6), and *sigmoid* was kept. Different optimizers were tried as well (Figure 7), and *adam* was kept. All optimizers have quite good performance, exept *sgd* which converges slowly and reconstructs the images in grayscale. The *rmsprop* has a similar reconstruction error as *adam*, but the validation reconstruction oscillates more (in fact, *adam* is *rmsprop* with a momentum). The *addagrad* optimizer was slightly less good than *adam*, and quite surprisingly, *adadelta* was less good than *adagrad*.

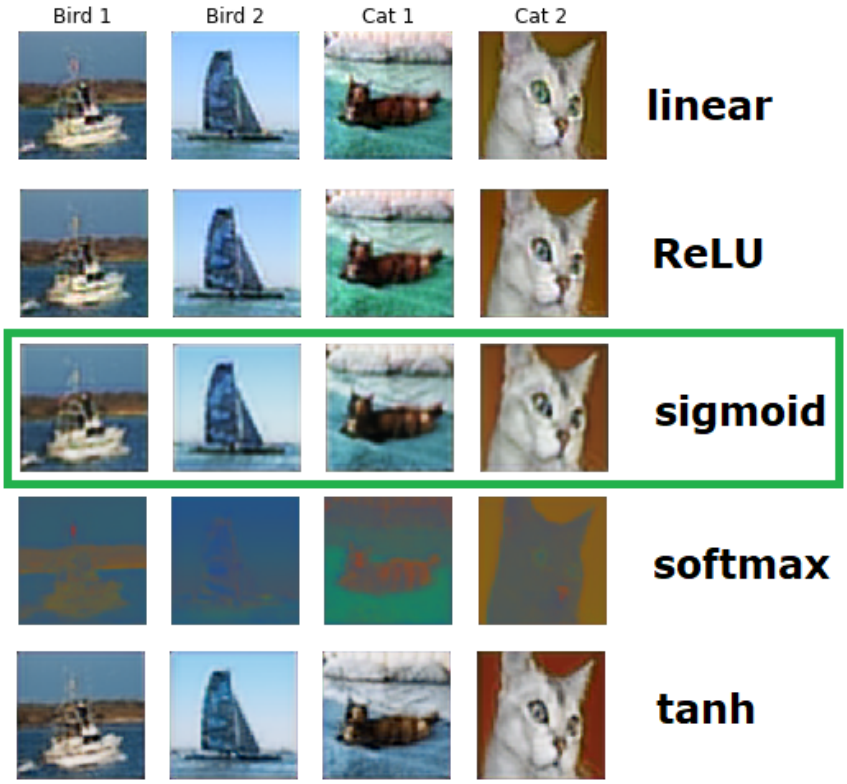


Figure 6: **Comparison of the reconstruction of images with different activation function in the last decoder layer.**

Figure 7: **Comparison of the reconstruction of images with different optimizers.**

Different regularization techniques were tried such as adding a dropout layer or L1 penalty. Using 25% and 40% dropout, the training reconstruction error was similar but the validation reconstruction error increased. Adding L1 regularization prevented the model from learning anything, resulting in all black or gray reconstructed images, with a regularization constant of 10^{-6} or above. With smaller regularization constants, the reconstruction error was of similar range as without regularization, but the computation time was increased. Regularization is not needed as the model is not overfitting (the training and validation errors converge, Figure 5).

3 Classification

Note that the models are evaluated based on the validation accuracy. A better set-up should be to divided the data into a training set, a validation set, and an additional test set on which the performance should be evaluated. Moreover, the accuracy is calculate for each label separately. The probabilities that are output are rounded, i.e. 0 if the probability is smaller than 0.5 (no object of the class is identified), and 1 if the probability is larger or equal to 0.5 (at least one object of this class is detected). The possible outputs are thus [0, 0], [0, 1], [1, 0] and [1, 1]. If the prediction is for example [1, 1] and the true labels are [1, 0], it will yield an accuracy of 50% and not 0%.

The model used for classification is the auto-encoder from section 2 with two convolutional layers in the encoder of size 16 and 8, with ReLU activation functions in all layers except the last decoder layer with *sigmoid* activation function (Figure 8, left). The model optimizer is *adam*, and the code di-

mension is 32x32x8 (8192) for inputs images of 128x128 pixels. First, the auto-encoder is trained in an self-supervised fashion as previously. Then, the encoder part is frozen and the encoding variables (Figure 9, left) are used as inputs of a fully-connected layer with two outputs (Figure 8, right). This layer uses a *sigmoid* activation function and the whole classifier model is trained with **binary cross-entropy** as loss function. Note that minimizing the binary cross-entropy corresponds to maximizing the log-likelihood. These choices are made so that the outputs of the network are probabilities that the images contain a boat (output 1) and a cat (output 2), with the possibility to have **multi-labels output**. If we assume that either boats or cats are present, we can use a *softmax* activation function in the dense output layer and use **categorical cross-entropy** loss function. In the latter case, the sum of the probabilities (outputs) of all classes must be equal to 1. For both models, the resulting accuracies and losses are of similar range (Figure 10). We can see that during the training of the whole classifier, the accuracy on the training set almost reaches 100% and its loss decreases towards 0, whereas the validation set accuracy remains around 85% and its loss keeps increasing. Thus, the final classifier is overfitting the training set.

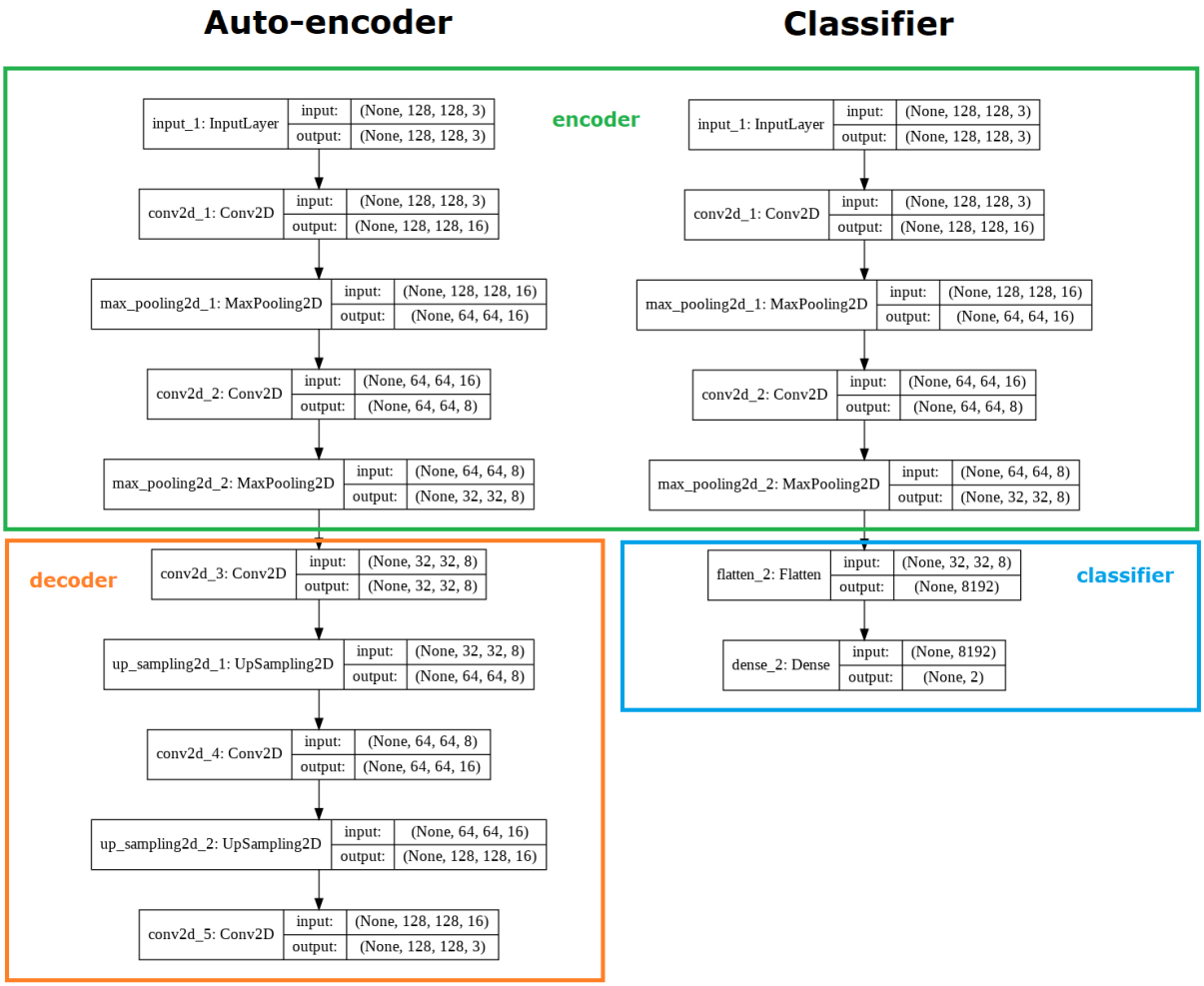


Figure 8: **Auto-encoder and classifier architectures.** First, the whole auto-encoder is trained in a self-supervised manner on the training set. Then, the encoder with the weights resulting from the previous training is frozen, i.e. they won't be updated during the training of the whole classifier. The whole classifier is constructed by flattening the output layer of the encoder and adding a dense (i.e. fully-connected) layer with two outputs, one for each class. These outputs are probabilities that the input image contains respectively boats and cats, and as they are not mutually exclusive, their sum might add up to more than 1.

Fine-tuning was performed on the classifier with *sigmoid* activation function and binary cross-entropy loss function: the layers corresponding to the encoder were unfrozen, and the whole classifier was trained again (Figure 9, middle). The accuracy on the training set reaches 100% and its loss decreases to 0, which makes the classifier perfect on the training data. However, fine-tuning does not increases the

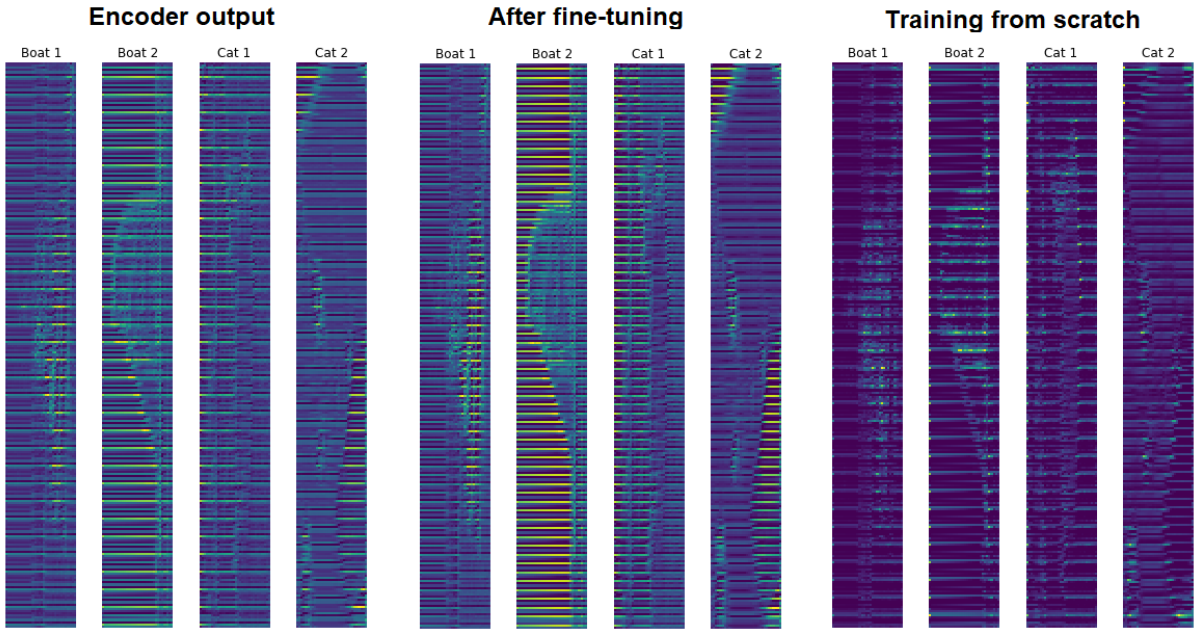


Figure 9: **Low-dimensional representations of the four images by the encoders.** Left: encoder resulting from the self-supervised training of the auto-encoder. These remain the same after the supervised training of the classifier, since the encoder layers are frozen. Middle: the encoder layers are unfrozen and the whole classifier is trained again. The same patterns are observed but the high weights are strengthened and low weights are weakened. Right: a classifier with the same architecture but random initial weights is trained. Some similarity in the shape of the weight distribution is observed, but weight values are different.

performance on the validation set, which remains around 85%, and we still observe overfitting (Figure 11).

Using the same classifier architecture but initialized with random weights, a new model is trained from scratch, i.e. we do not use the weights from the auto-encoder (Figure 9, right). The same results as with previous models are obtained: 100% accuracy on the training set, about 85% on the validation set. We can notice that the model converges faster, that 100% accuracy on the training set is obtained within 50 epochs, and that the accuracy on the validation set is more stable (Figure 12).

A model using the weights of an auto-encoder with the same architecture but trained in a greedy layer-wise manner was efficient to avoid the increase in the loss of the validation set, but did not improve its accuracy (Figure 13).

Two deeper CNNs were tried without improving the validation set accuracy. Adding activity regularization L1 and/or L2 to the original classifier based on the encoder, as well as adding dropout and/or batch normalization layers, did not increase the model performance either. Merging the training and validation data and splitting them into 80% training and 20% validation sets did not make any significant difference. A new model architecture with a smaller encoding representation (1024 variables) was tried as well and did not yield better results.

Finally, data augmentation technique was tried, with different parameters (Figure 14). This removed the overfitting by decreasing the training set accuracy, but did not yield accuracies above 80-85% for both the training and validation sets. We do not really observe any improvement during the learning of the classifier.

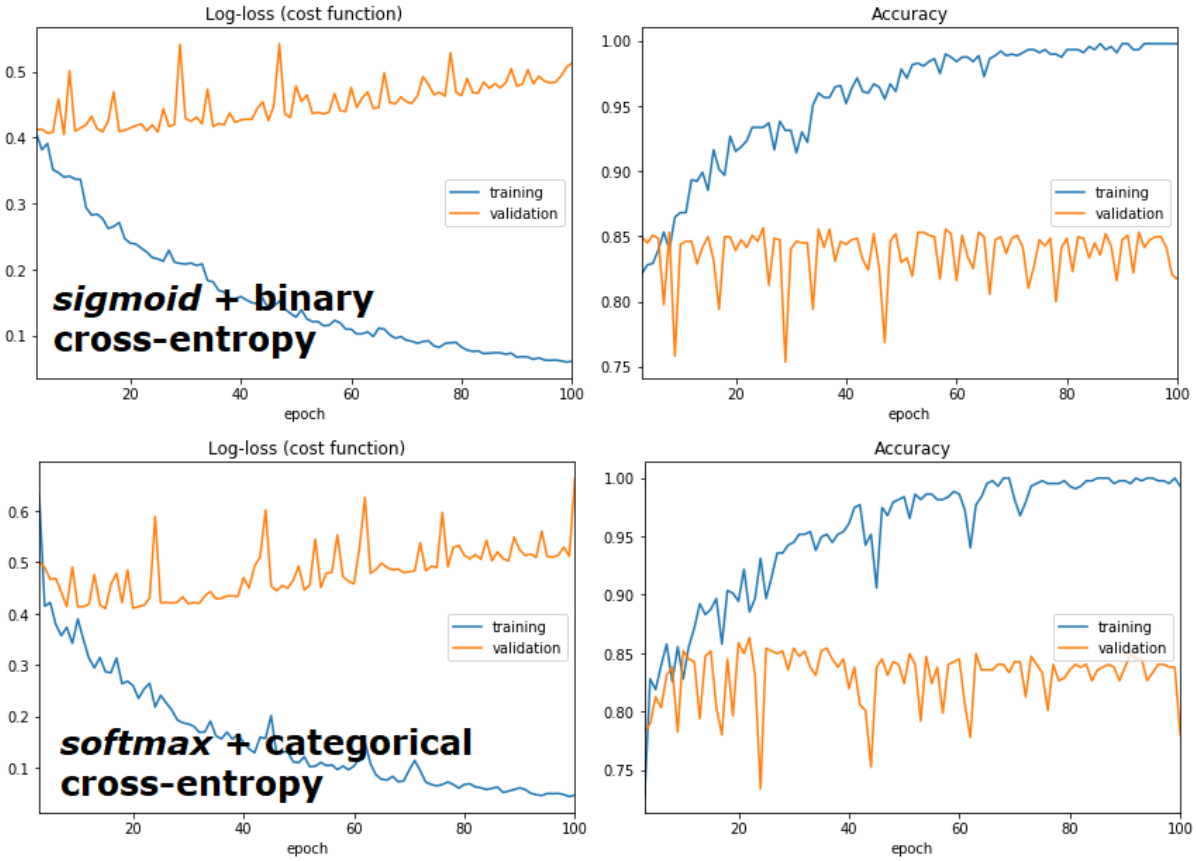


Figure 10: **Comparison of the loss and accuracy of two models. Top:** we do not assume that each image contains one class only, and boat(s) **and** cat(s) can be present at the same time. Thus a *sigmoid* activation function is used in the fully-connected layer, and a binary cross-entropy loss to train the model. **Bottom:** we assume that each picture only contains one class, boat(s) **or** cat(s). Thus a *softmax* activation function is used in the fully-connected layer, and a categorical cross-entropy loss to train the model. The resulting accuracies are similar, and both models overfit the training data. The models were trained for 100 epochs with a batch size of 32.

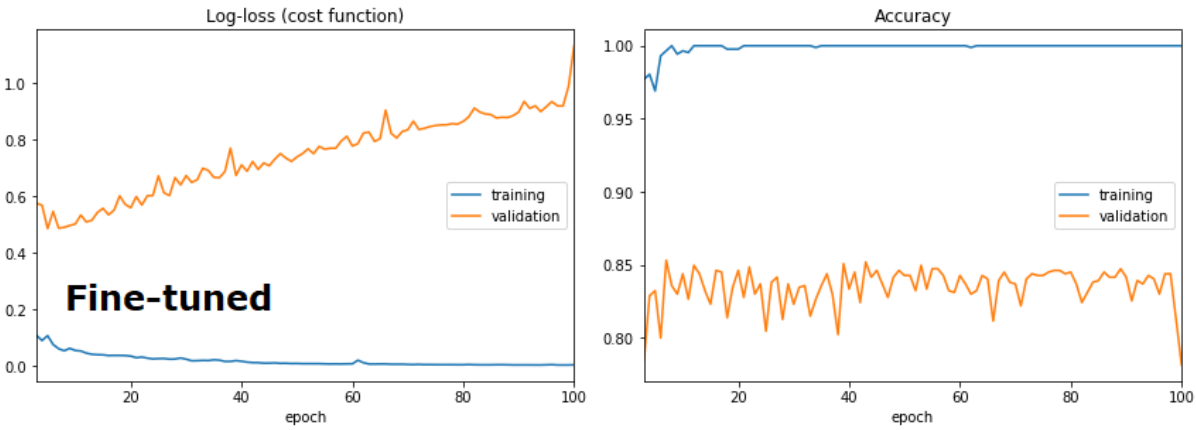


Figure 11: **Fine-tuned model.** The previously frozen layers are unfrozen and the whole classifier is trained again. The fine-tuning improves the loss and accuracy on the training set but does not influence the results on the validation set.

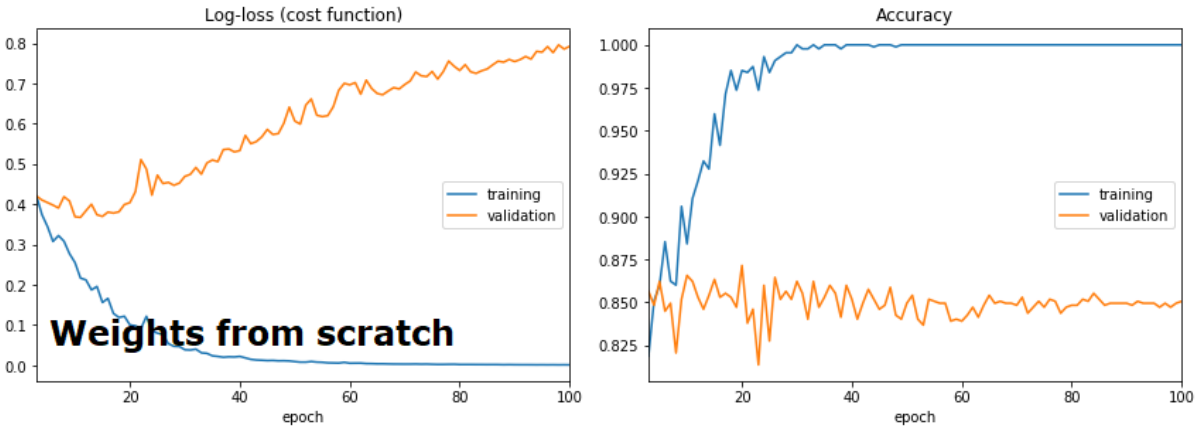


Figure 12: **Model with randomly initialized weights.** All parameters are trained from scratch. We obtain the same accuracies as using the weights extracted from the auto-encoder. We can observe that the model converges faster and that the accuracy on the validation set is more stable.

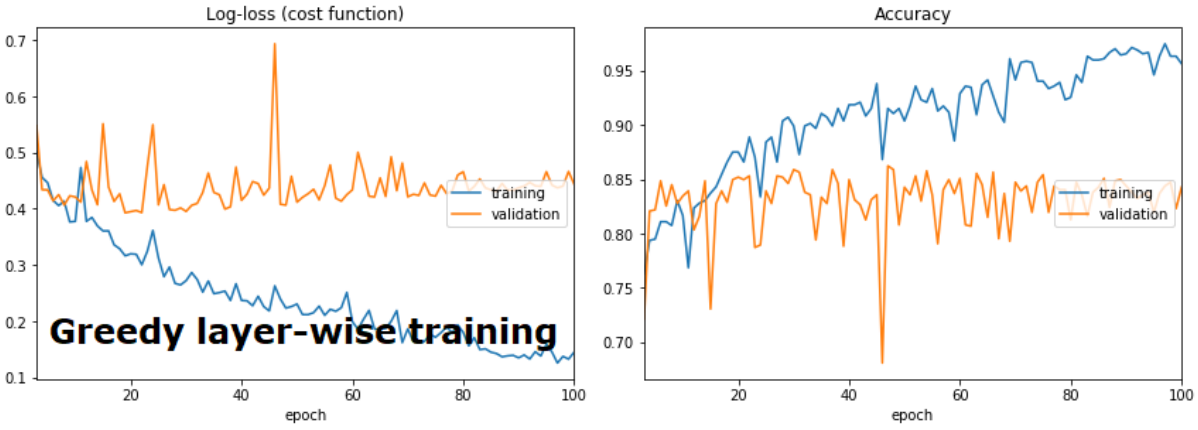


Figure 13: **Model with greedy layer-wise trained auto-encoder.** The first auto-encoder of one layer is trained. Then, using the output of the encoder part, the second auto-encoder is trained. The two encoders layers are frozen and a fully-connected layer with two outputs is added to train the whole classifier. We can notice that we do not observe an increase in the loss on the validation set, however the accuracy remains around 85%. Also, the accuracy on the training set does not reach 100% within 100 epochs.

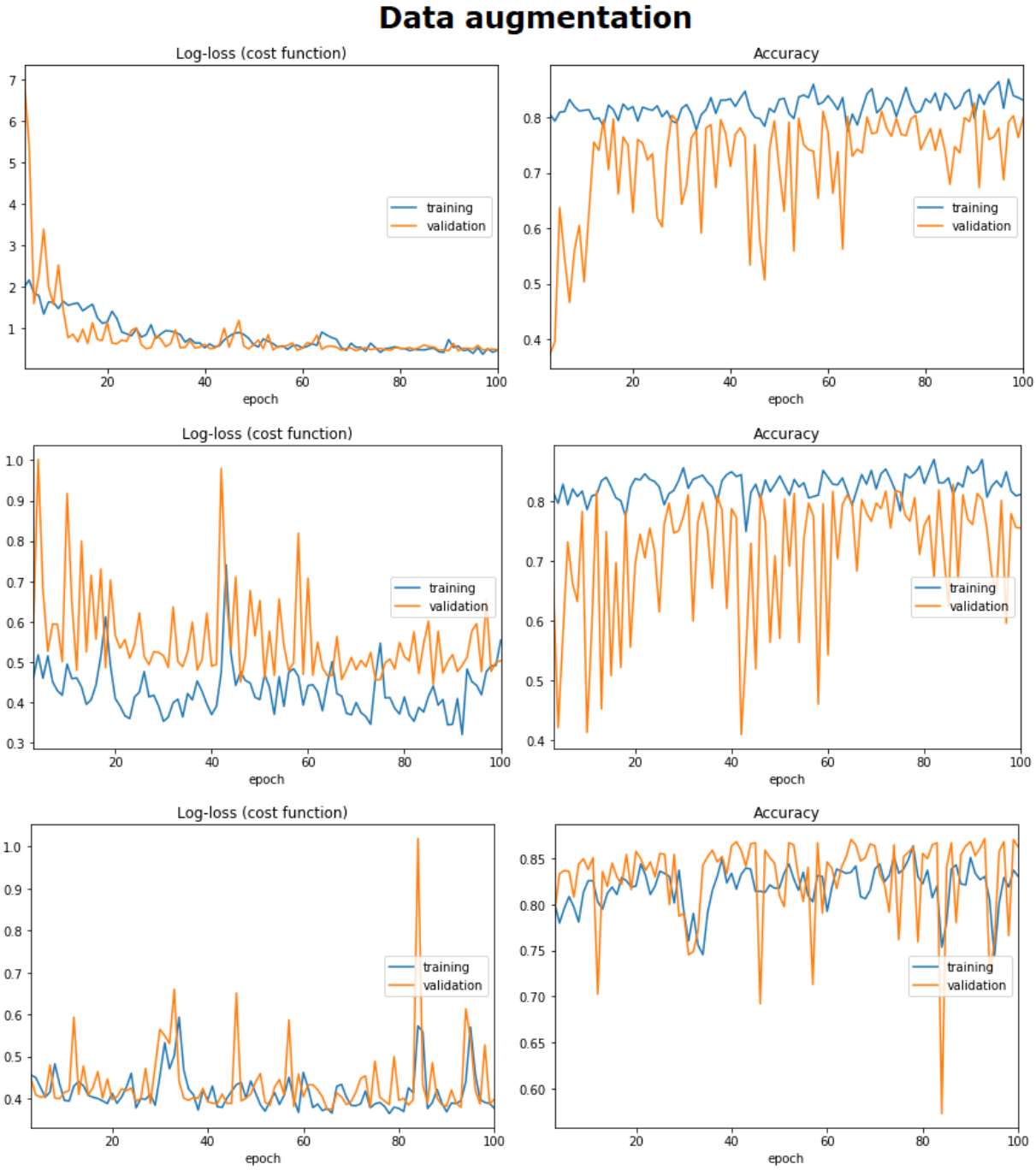


Figure 14: **Training of the model with data augmentation.** All parameters are trained from scratch. We obtain the same accuracies as using the weights extracted from the auto-encoder. We can observe that the model converges faster and that the accuracy on the validation set is more stable.

4 Segmentation

In this section, the problem of segmentation of images is tackled. The task can be seen as a classification among the different classes, similarly to Section 3. However, now the classification is done for each pixel, depending on the local features around the pixel. Hence, the last CNN architecture must be modified. The approach taken here was to reproduce a known architecture that has proven to have good performances for segmentation : a fully convolutional network. More specifically, we implemented the FCN 8s architecture.

4.1 The processing of the data

Before building the network, a few remarks concerning the data are necessary.

Firstly, in order to assure a good amount of data, the whole segmentation dataset from Pascal VOC-2009 is used. Secondly, the task of segmentation was here reduced to only a binary segmentation, for simplification. As the segmentation labels consist of rgb images, the data was processed so that all the background pixels were put to zero and all the classes pixels were put to one.

Thirdly, the FCN 8s architecture is based on the vgg16 pre-trained network. This network takes as input 224 by 224 pixels images. The whole dataset was then resized accordingly.

Finally, note that the FCN architecture takes a one hot encoding as output labelling. In the binary case, it was simply made duplicate and inverse the previously processed data.

The Figure 15 shows the input images, the initial segmentation labels and the post processing labels.

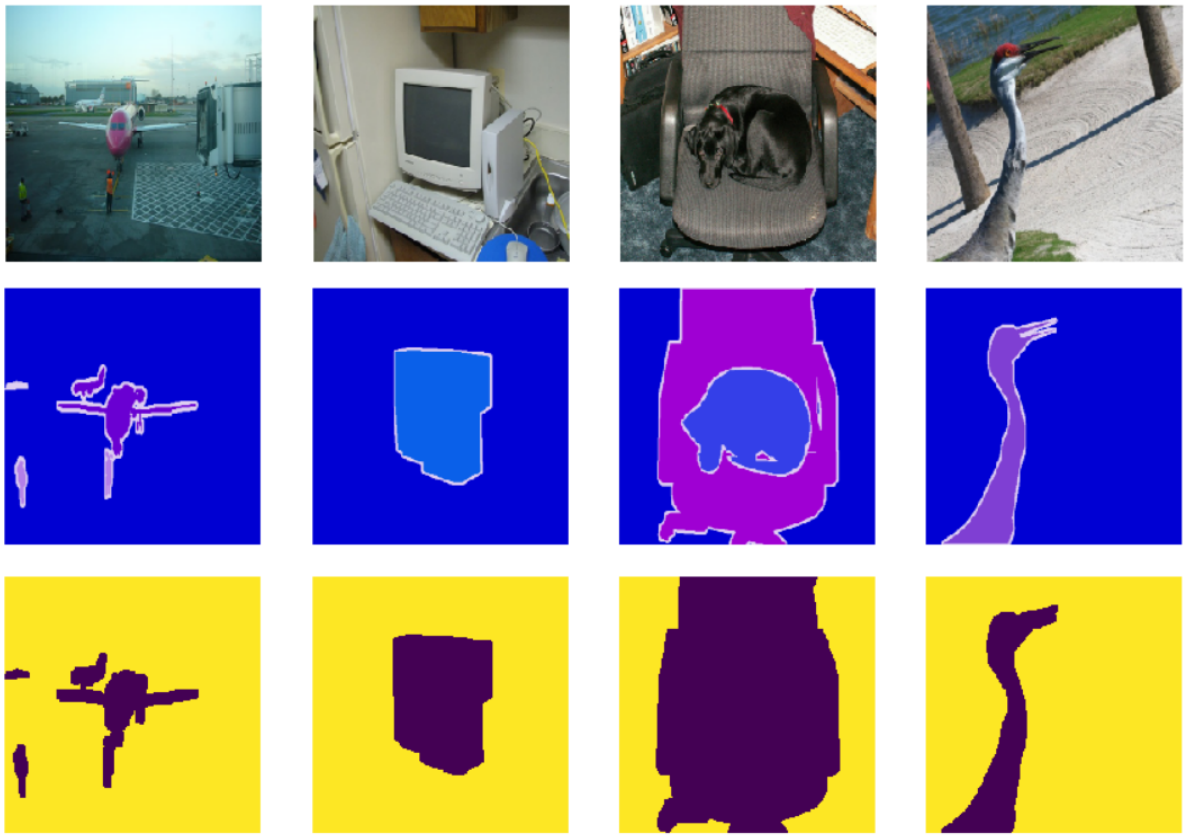


Figure 15: Segmentation dataset

4.2 FCN 8s architecture

The full architecture of the FCN8s is shown on Figure 16. In our case, the FCN takes advantage of the VGG16 structure. The pretrained convolutional layers of the VGG16 are used as the encoder part of the FCN to extract features. The FCN architecture is built on top of it.

Note that an other type of architecture could have been chosen. An other really known one that has even better performances is the U-Net. That one also can be implemented on top of a pre trained net.

4.3 Performance metrics

In the problem of segmentation, new metrics for the evaluation of performances are necessary. The output predictions can be seen as a window of same size as the input image but with a channels depth equal to the number of classes. Each channel layer is equivalent to a binary mask (one hot encoding) for each of the classes. The classic accuracy measure will compute the following value :

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

This is equivalent to pixel accuracy. However, this metric can sometimes provide misleading results when the class representation is small within the image, as the measure will be biased in mainly reporting how well you identify negative case (ie. where the class is not present) [3].

Two other performance measures are commonly used for segmentation : the Dice score (also called the F1 score) and the Intersection over Union (IoU - also called the Jaccard index). Their equations are respectively :

$$Dice = \frac{2 \cdot |X \cap Y|}{|X| + |Y|} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN} \tag{2}$$

and

$$IoU = \frac{|X \cap Y|}{|X \cup Y|} = \frac{TP}{TP + FP + FN} \tag{3}$$

As both can be arguably good measures of similarity over sets, we have implemented both for evaluating the model.

The results for different number of epochs are shown on Table 3. The class 0 is the background and the class 1 is the objects.

Metrics	**50 epochs**	**75 epochs**	**100 epochs**	**200 epochs**
IoU_0	0.580	0.812	0.811	0.809
IoU_1	0.779	0.574	0.566	0.568
$Dice_0$	0.735	0.896	0.896	0.895
$Dice_1$	0.876	0.729	0.723	0.724
mean Dice score	0.805	0.813	0.809	0.809
mean IoU	0.680	0.693	0.689	0.688

Table 3: Performance metrics for the segmentation task - Class 0 is the background, class 1 is the objects.

Before commenting the results, an important note is to be made. We did not use a loss function to directly optimize the Dice score or the IoU. Instead, the typical `binary_crossentropy` for classification was used. The risk of this choice is as said earlier, to be biased in the direction of background which is the most common class. Moreover, it would get even worse in a case of mutliclass as each class would count less pixels.

The reason is the loss function must be differentiable which is not the case for the Dice score and the IoU. In order to really use them, different papers have approximated it by continuous functions. However, the results are often less stable and gradients are more complicated.

The results of Table 3 show that the IoU and Dice are the best for 75 epochs. It would mean that between 75 and 100 epochs, the model begins to overfit. Note that it seems to confirm the difference between cross entropy and Dice or IoU. Looking at Figure 17, the model seems to overfit the cross entropy earlier around 30 epochs.

Furthermore, the results in IoU are even better that the result found in other papers (here : Long, mean IoU = 62.7 on multiclass segmentation). This is mainly due to the fact that the task was made easier because only binary.

[3] https://www.jeremyjordan.me/evaluating-image-segmentation-models/

Finally, IoU and Dice could possibly give better results, even if it is not always guaranteed. The advantage is to cope with the natural imbalance in the segmentation task. However, this could also be tackled by weighing the class losses compared to the background.

4.4 Results

First, by monitoring the accuracy and the loss during the 100 epochs simulation on Figure 17, we can see that the optimization on the training set produces quite unstable performances on the validation set. However, generally, the error and accuracy on the test set seem to converge. The loss even rise again which states the beginning of overfitting. Nevertheless, we must remind that is normally not supposed to be really showing the final result in IoU or Dice.

Some example results are displayed on Figures 18 to 22. On them, we can compare the differences across epochs. Even if the loss function is not exactly appropriate, the results are quite homogeneous. An interesting behaviour of the results is the squared segmentation of the predictions. This should need more research but a possible reason would be the deconvolution layers that are not yet tuned enough.

Finally, note that a further interesting step would be to generalize to see if we can reach as good performances and also what the model is predicting for each object.

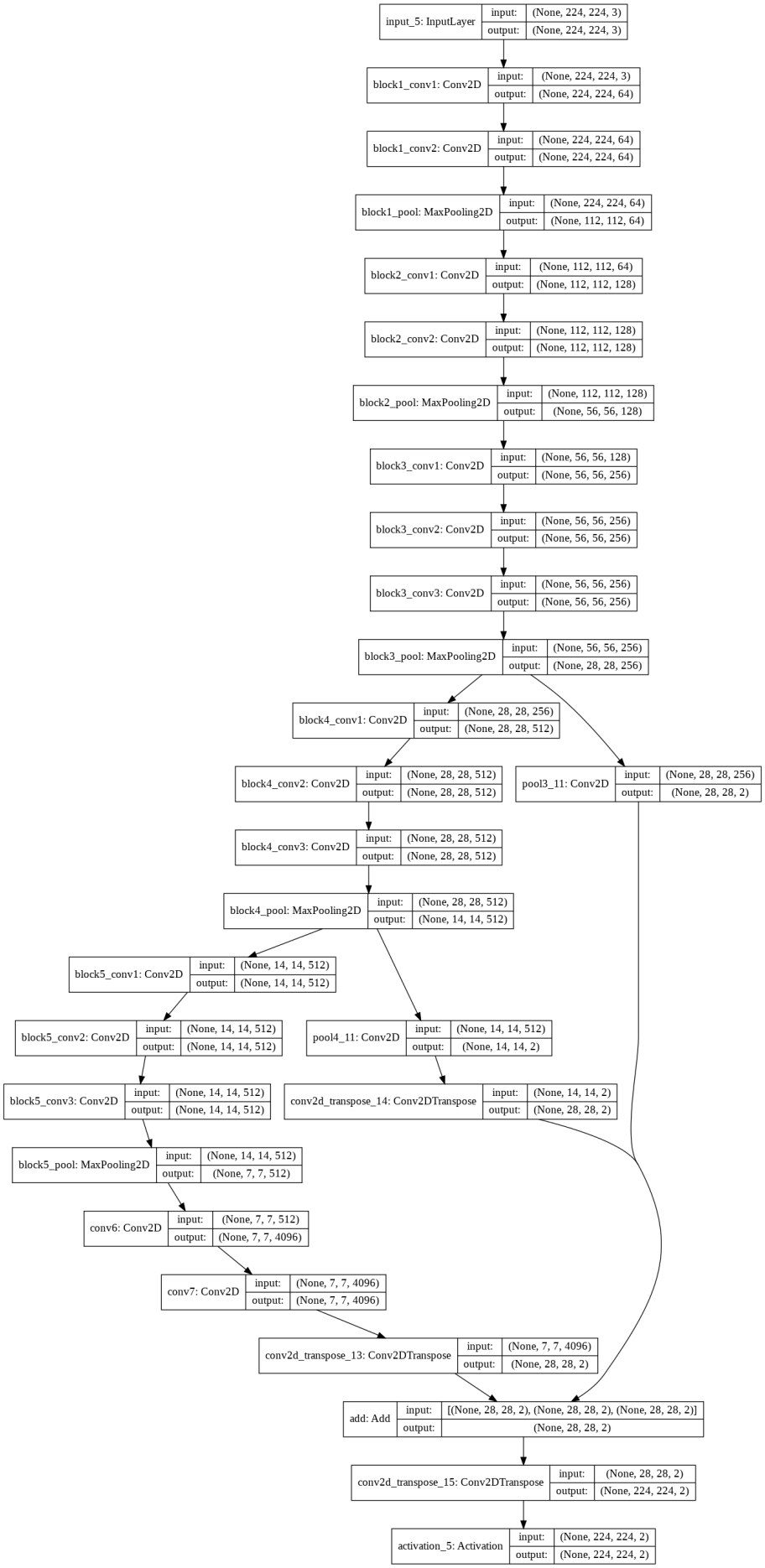
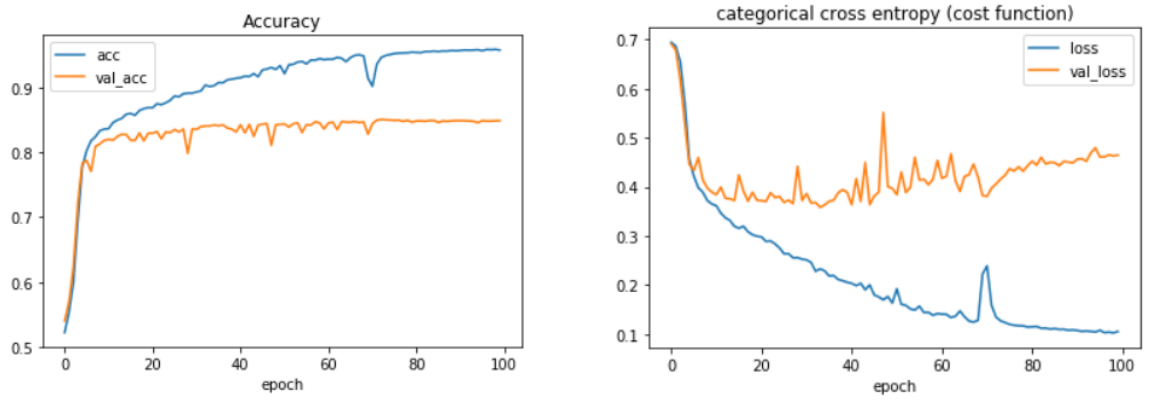


Figure 16: **FCN8s architecture**



(a) (b)

Figure 17

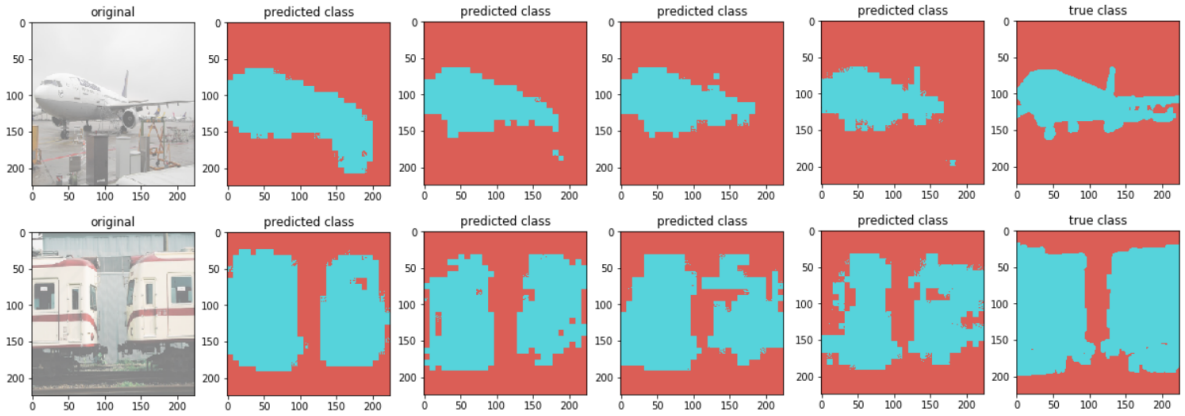


Figure 18: Results of the segmentation. From left to right, 50, 75, 100 and 200 epochs for the predicted images.

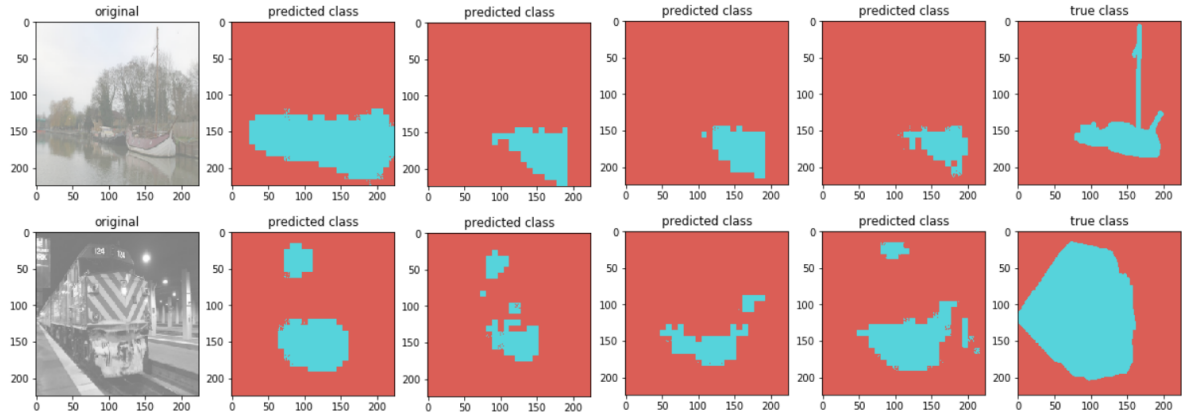


Figure 19: Results of the segmentation. From left to right, 50, 75, 100 and 200 epochs for the predicted images.

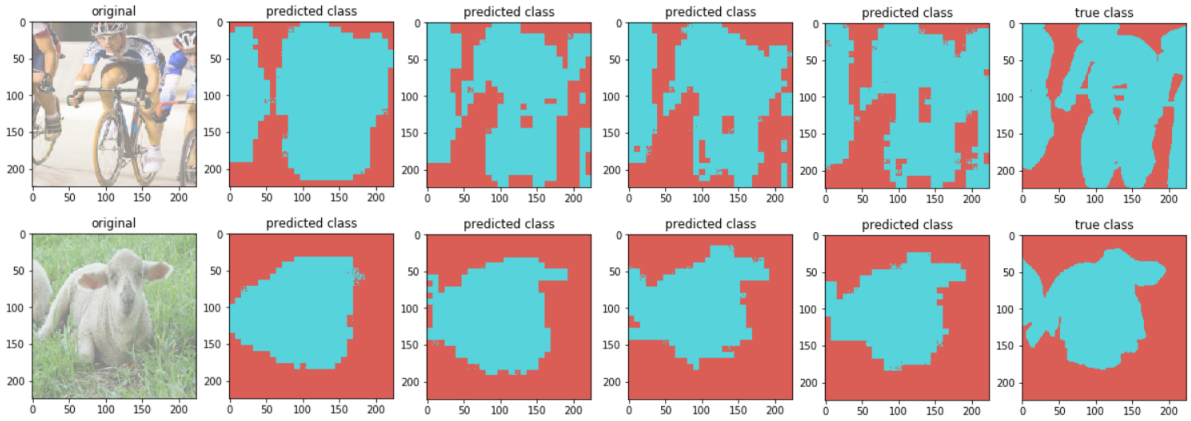


Figure 20: Results of the segmentation. From left to right, 50, 75, 100 and 200 epochs for the predicted images.

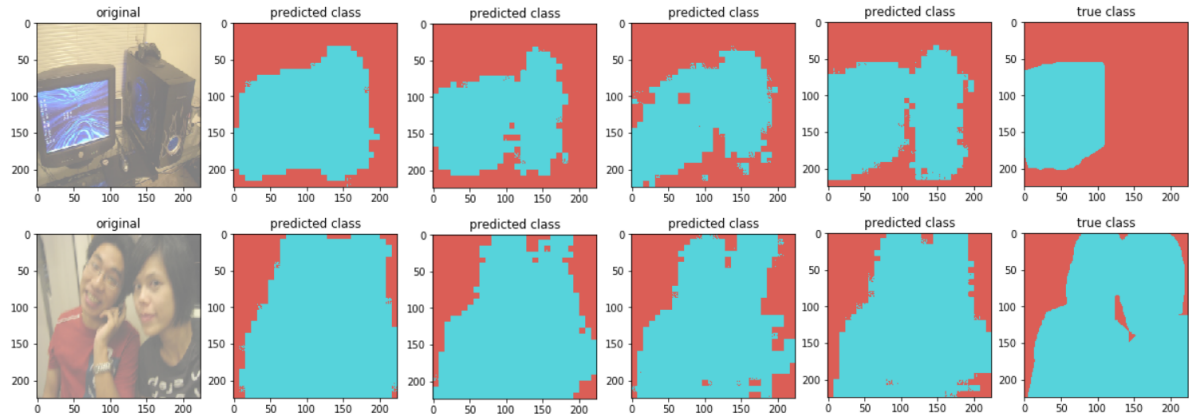


Figure 21: Results of the segmentation. From left to right, 50, 75, 100 and 200 epochs for the predicted images.

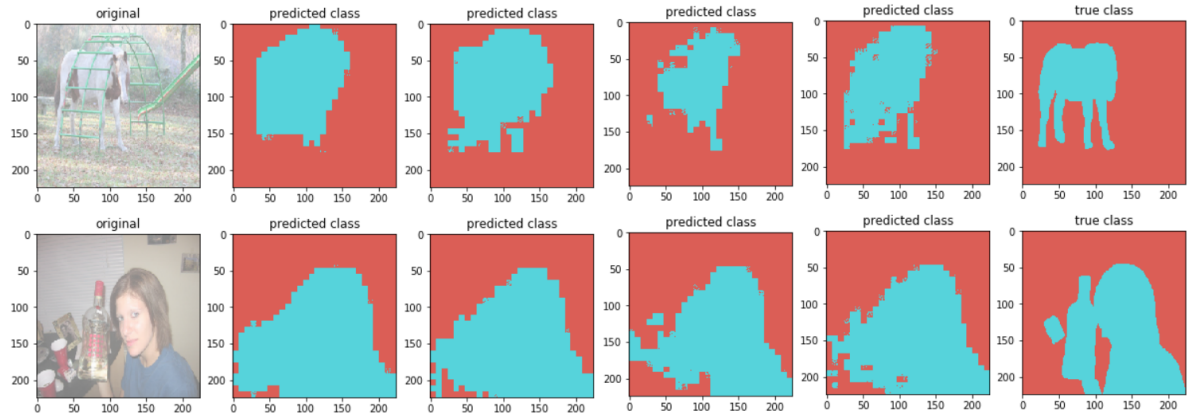


Figure 22: Results of the segmentation. From left to right, 50, 75, 100 and 200 epochs for the predicted images.

5 Conclusion

The reconstruction of images is very accurate, especially using a non-linear convolutional auto-encoder. Using a pre-trained auto-encoder for the classification of boats and cats images yields an accuracy of about 85% on the validation set, which is better than random, but which we did not manage to improve despite trying several methods to reduce the overfitting. In general, we noticed that about 41% of the boats of the validation set and 23% of the cats were misclassified. One solution could be to generate more boat images by data augmentation, to balance the difference in the number of images (155 boats vs 277 cats). Another reason why boats perform less well is that some images are very different from each other and boats are not always distinguishable (e.g. selfies on a canoe). One could manually select only boat images that are photographed from the outside, such as the ones used as example throughout the report. Finally, a better solution could be to use weights of a pre-trained network such as ImageNet.

Concerning the segmentation, only one model was implemented, the FCN 8s. It reaches satisfying results that are comparable to the results in the literature. There are even above because the task here was only binary segmentation. The evaluation of performances was done with both IoU and Dice score but not directly optimized with the loss function. There are ways to go further. First, we could implement a loss function directly optimizing the Dice score or IoU. Moreover, there is also the squared board behaviour on the predictions that needs further research. Secondly, we can obviously generalize the problem to the multiclass segmentation. Finally, an other architecture known for performing better could be implemented like the U-Net.