Katholieke
Universiteit
Leuven

**Department of
Computer Science**

COMPUTER VISION PROJECT 2019

In the name of deep learning

Breyne Emile (r0629298)
Pauwels Rik (r0622635)

Academic year 2018–2019

Contents

1 Introduction

In this assignment we explore how artificial neural networks can be used in computer vision. Artificial neural networks are able to detect patterns in large datasets so when these are used in the right way they should be able to detect, differentiate and label objects in images as humans would (or even better!), without having to implement hand-made complex feature extractors etc.

2 Auto-encoder

2.1 PCA vs Auto-encoder

Auto-encoders are flexible machine learning models that can consist of many layers using different activation functions. The aim of an auto-encoder is to learn the best representation possible for a given set of data using less variables than the original input so it is no surprise that they can mimic a much simpler PCA.
The auto-encoder architecture is as follows: one input layer followed by a fully connected hidden layer consisting of only n neurons (with n the dimensionality of the PCA) and a fully connected decoder layer to reconstruct the data. All the activation functions are linear because those are the only transformations that take place in the creation of a PCA plot.
When using the loss function 'mean squared error' the model will learn to encode and later decode the data in such a way that the reconstruction error is minimal. The projected points will now lay in subspace X and we can say that projecting all the points onto this subspace X minimizes the error.
The PCA algorithm will in each step calculate the axis for which the variance is maximal and will use all these axes to span its subspace. All the axes must by orthogonal and together they form a subspace that will maximize the subspace variance when the data points are projected onto it.
It is provable that maximizing the variance and minimizing the mean square error will produce the same subspace so the results of the auto-encoder and the PCA will be similar but not entirely the same: nothing stops the auto-encoder from using a non-orthonormal basis but the subspace spanned by this basis will be the same as the subspace that the PCA became.
Another difference could emerge if we allow non-linear activation functions in the auto-encoder. This addition of non-linearity in the model enables the model to learn more complex transformations, while a regular PCA is restricted to linear matrix transformations.

2.2 Non-linear and convolutional

Images are a combination of complex structures that can be broken down into combinations of less complex structures and structures of structures. Luckily for us there is a network that is pretty good in recognizing those simple structures: a convolutional neural network. Famous image-classification models like VGG19 and GoogLeNet use the power of CNNs to make very accurate image classifiers. This is done by stacking them on top of each other, the first CNN recognizes for example horizontal and vertical edges while the second CNN can combine those detected structures to detect for example squares. CNNs higher in the infrastructure are then for example able to detect entire dogs or persons by searching for pawns or hands. Those networks are provably very accurate but often too large to implement and train on a simple laptop.

Auto-encoders can provide another feature apart from classification namely image compression. If the hidden layer in between the encoder layers and the decoder layers is smaller than the original input, then the data will be compressed since it must fit in the neurons presented in the smaller layer. This compression is lossy but if the loss is small enough then it wont really matter since it will still be clear what is on the picture. Minimization of this reconstruction loss is the optimalization problem that we wish to solve when training the model.

2.3 Design of the model

We implemented the model below. It has 9 CNN layers to encode the data and 9 CNN layers to decode it. We use 3 MaxPooling layers and keep 32 filters in the hidden layer so the original image (144*144*3) will be reduced by a factor 6 to 18*18*32.

The model we made has 121.555 parameters which is a lot, but we kept it as small as possible because

we noticed that with more parameters, the model tends to over-fit and with less parameters the model has a bigger reconstruction error.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 144, 144, 3)	0
conv2d_1 (Conv2D)	(None, 144, 144, 16)	448
conv2d_2 (Conv2D)	(None, 144, 144, 16)	2320
conv2d_3 (Conv2D)	(None, 144, 144, 16)	2320
max_pooling2d_1 (MaxPooling2	(None, 72, 72, 16)	0
conv2d_4 (Conv2D)	(None, 72, 72, 32)	4640
conv2d_5 (Conv2D)	(None, 72, 72, 32)	9248
conv2d_6 (Conv2D)	(None, 72, 72, 32)	9248
max_pooling2d_2 (MaxPooling2	(None, 36, 36, 32)	0
conv2d_7 (Conv2D)	(None, 36, 36, 32)	9248
conv2d_8 (Conv2D)	(None, 36, 36, 32)	9248
conv2d_9 (Conv2D)	(None, 36, 36, 32)	9248
max_pooling2d_3 (MaxPooling2	(None, 18, 18, 32)	0

conv2d_10 (Conv2D)	(None, 18, 18, 32)	9248
conv2d_11 (Conv2D)	(None, 18, 18, 32)	9248
conv2d_12 (Conv2D)	(None, 18, 18, 32)	9248
up_sampling2d_1 (UpSampling2	(None, 36, 36, 32)	0
conv2d_13 (Conv2D)	(None, 36, 36, 32)	9248
conv2d_14 (Conv2D)	(None, 36, 36, 32)	9248
conv2d_15 (Conv2D)	(None, 36, 36, 32)	9248
up_sampling2d_2 (UpSampling2	(None, 72, 72, 32)	0
conv2d_16 (Conv2D)	(None, 72, 72, 16)	4624
conv2d_17 (Conv2D)	(None, 72, 72, 16)	2320
conv2d_18 (Conv2D)	(None, 72, 72, 16)	2320
up_sampling2d_3 (UpSampling2	(None, 144, 144, 16)	0
conv2d_19 (Conv2D)	(None, 144, 144, 3)	435

Figure 1: Textual representation of the encoder (left) and the decoder (right)

We used the mean square error loss function. This loss function, which tries to minimize the mean square error of the predicted data, is a function that works well in regression problems and is fast. The speed of the loss function is especially important when lots of training is necessary like it was in this question.

We trained the model on the training images that the given filter-code returned. We did not train on the test data neither did we use the other training data present in the large database. This way we were able to evaluate our model independently of the training data. The plot below shows that the model keeps learning until the end but the learning rate is now so low that training the model further wont result in big improvements.

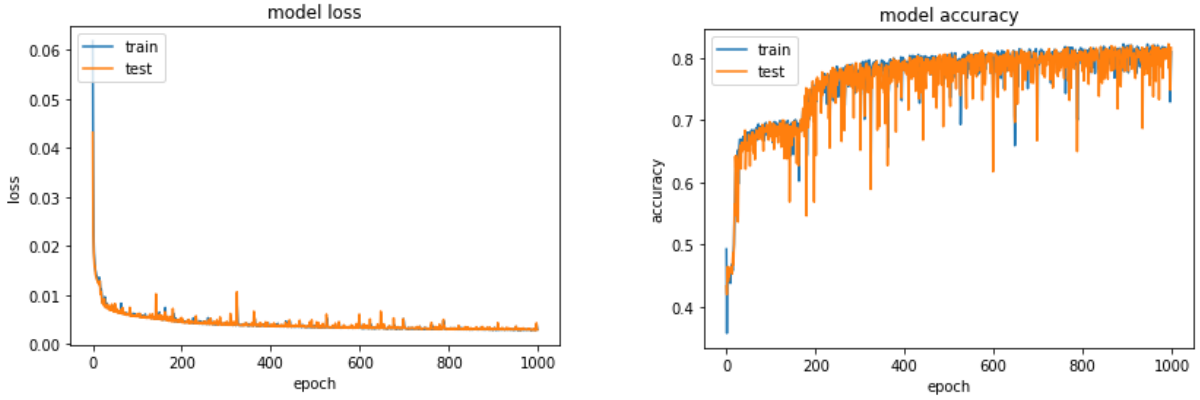

Figure 2: The loss and accuracy of the auto-encoder during training. Minimal loss = 0.002, maximum accuracy = 83%

Figure 3 is a plot of the result of the model. The first image is the original image (144x144x3) and the second one is the image after compressing it 6 times to 18x18x32 and restoring it back to 144x144x3. It is clear that we lost some details but the general representation is still quite similar to the original picture. The last picture is a picture that is compressed to 58x58x3 using the given filtercode and while it is obvious that you can still gain some detail by smoothing this picture and making it 144x144x3 the level of detail will not exceed the level of detail our auto-encoder achieves.

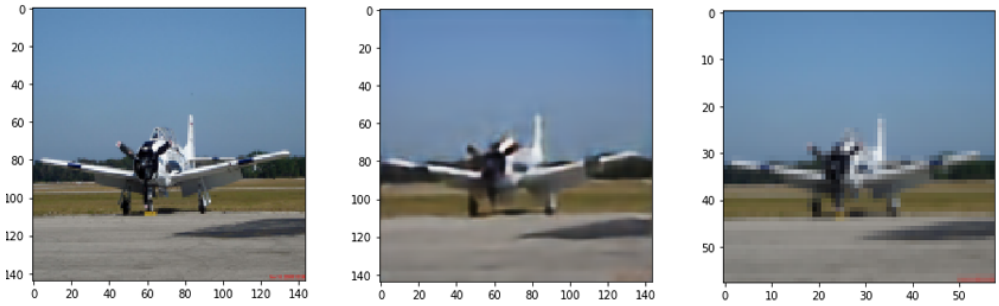

Figure 3: Left an original picture out of the dataset, middle the reconstructed picture, right the compressed picture using the obvious approach

3 Classification

3.1 Design of the model

For the classification task we used the encoder part out of the auto-encoder and extended it with two fully connected layers of 2048 neurons and 5 separate fully connected layers of 1 neuron together with the sigmoid activation function. The big fully connected layers are excellent in learning what features (recognized by the CNNs) are important in the classification process and the small fully connected layers will output the certainty of the model about whether the picture is of that class. To counter overfitting we also added two dropout layers right after the two big dense layers.

An image can be classified in multiple classes (it can contain a dog and a bird) so we opted for 5 one-class logistic learners instead of one multi-class logistic learner. We choose this because the second one is hard to implement on this kind of data and will in the end work in a similar way.

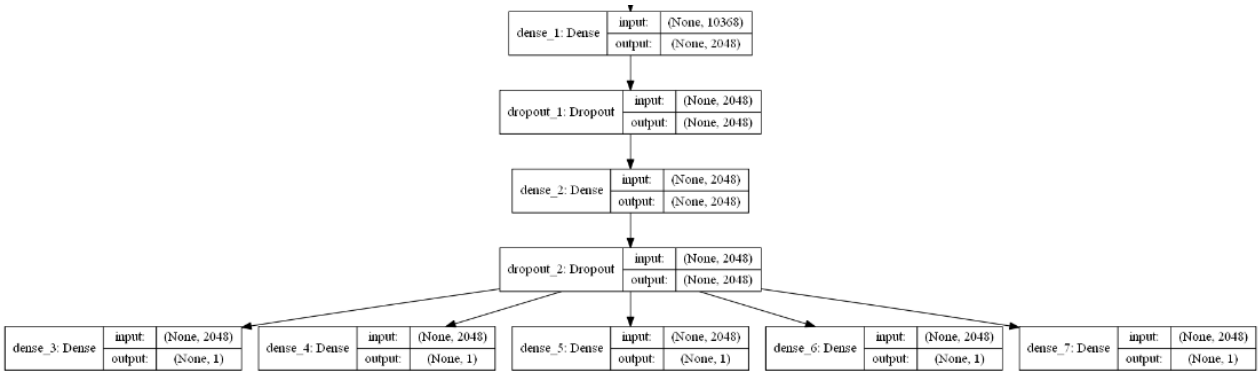

Figure 4: Schematic representation of the prediction model extension

3.2 Activation and loss function

We used the non-linear activation function ReLu because it is fast and because it sets all the negative values to 0. A feature can be present or not but not 'negatively present' so all the negative values are mapped to 0. This way we dont have to worry about the negative part and we can make our model stronger. Another thing to note is that we tried to produce more training data by mirroring the given images and adding them to the original training data set. This data augmentation technique helped our model train longer before overfitting on the training data since there is more training data available for the learner.
For this learner we tried to maximize the log-likelihood by using the log function binary_crossentropy. We chose this function because logistic regression learners use a sigmoid activation function and this works well with crossentropy loss functions. After training on the training data, our model classified all the validation images and in 53.1% of the time the model gave the highest chance to a label that was present in the picture (for example 'dog').

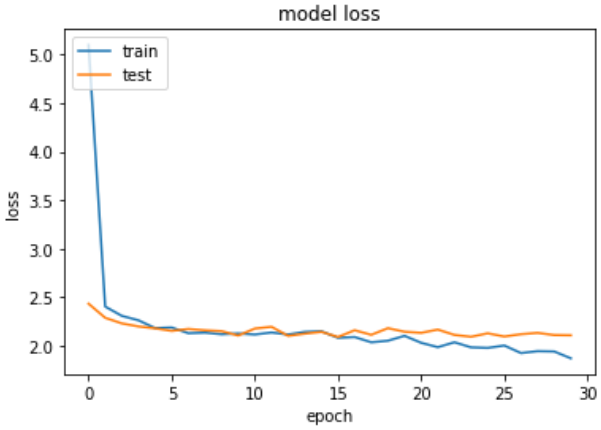

Figure 5: Training and validation loss during training

The predictions the model made show that the model definitely learned something in the training process. The pictures below show some of the results and you can see that the model scores good on clear pictures and bad on pictures where the object is not entirely visible or too small to detect all of its features.

Figure 6: Predictions the model made

3.3 The fully trainable model

The previously defined model still uses the weights of the auto-encoder which might not be optimal for the classification problem. The auto-encoder is for example able to somewhat recognize trees because it needs to encode them but there is no need for a 'tree structure' it wants to decide if there is a chair in the picture. We trained the model from scratch and the results (shown in Figure 7) were surprisingly not better. Now the label that received the highest score was in 51% of the cases the correct one.

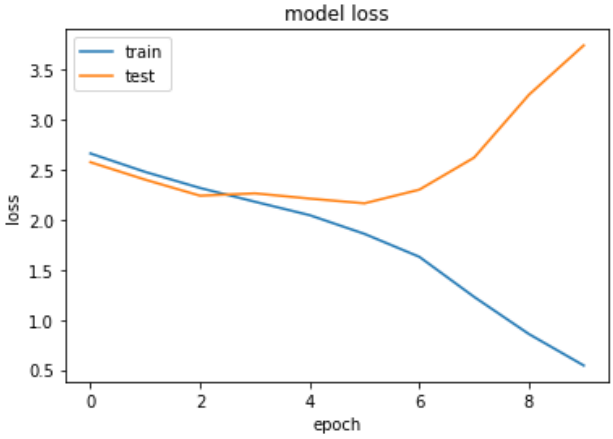

Figure 7: Loss of the model with parameters trained from scratch

The model clearly over-fits the data pretty fast because there are 25 million parameters to train and only 1470*2 training images. This is probably the reason why this classifier preforms a bit worst than the previous one. Another thing to highlight is that the model is now more 'certain' in its results, many (but not all) prediction values are way higher than in the previous model but that does not make them more right. For example the model is 90% certain that there is a car in the first picture but there is none.

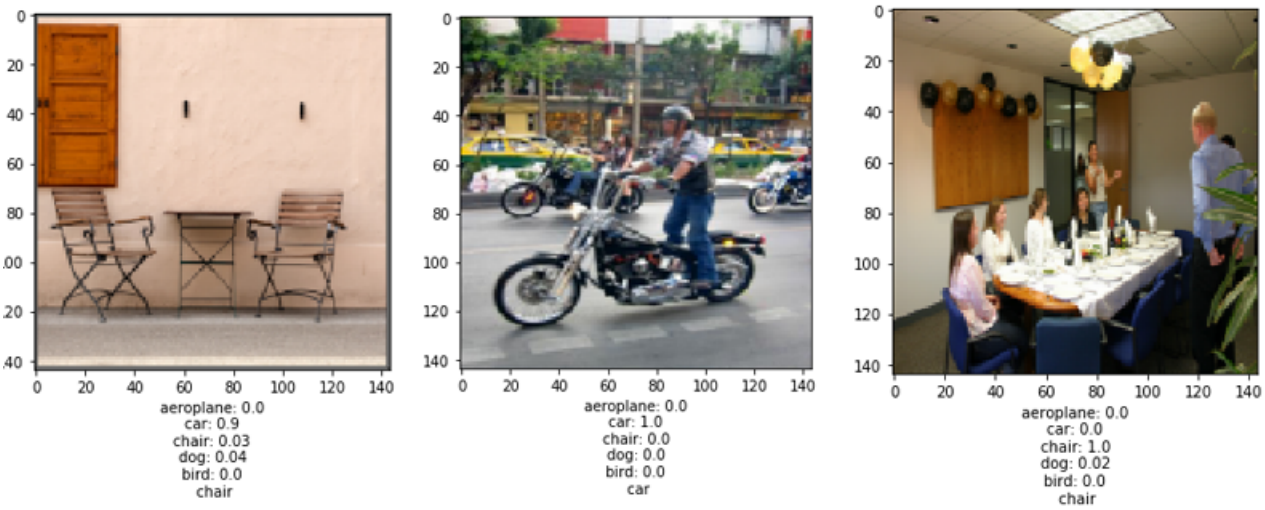

Figure 8: Predictions of the model with parameters trained from scratch

4 Segmentation

4.1 Approach

The goal of this part is to obtain a segmentation network who is capable of classifying each pixel in a foreground or background class.

Training data The PASCAL VOC dataset provides a semantic segmentation ground truth for a subset of the samples. This means that all the pixels belonging to an object are classified between 20 different classes (person, bird, car, etc.). The semantic (class) segmentation problem can be seen as a continuation of the classification problem. However, for this task we consider only 2 classes: foreground and background. For this purpose, we altered the given ground truth images to allow for binary classification.
As the dataset has shrunk a lot in comparison to the previous tasks we first fed all the possible classes to train and evaluate the network. We also kept the 3 color channels and fed them directly to the network (after resizing). As this produced only mediocre results. We decided to limit the data again to the 5 classes used before ('aeroplane', 'car', 'chair', 'dog', 'bird'). That reduced the size of the training dataset to 143 images, which isn't enough. To try to remedy over-fitting due to little available data, we augmented the data with a random generator. We also only used grayscale images in this part, as to reduce the dimensionality of the input by a factor of 3.

4.2 The network Architecture

We built a network based on the approach detailed in Ronneberger et al.'s paper ([RFB15]). The UNET model is a proven architecture for image segmentation. It follows the structure of an encoder-decoder: first downsampling to find relevant features, then upsampling to match the same image dimensions as the input.

UNET architecture The UNET was developed by O. Ronneberger, P.Fischer and T. Brox for Bio Medical Image Segmentation [RFB15]. The architecture contains two paths. First path is the contraction path (also called as the encoder) which is used to capture the context in the image. The encoder is just a traditional stack of convolutional and max pooling layers. The second path is the symmetric expanding path (also called as the decoder) which is used to enable precise localization using transposed convolutions. Thus it is an end-to-end fully convolutional network (FCN), i.e. it only contains Convolutional layers and does not contain any Dense layer because of which it can accept image of any size. (In our experiment we did resize all the images to 256x256 however).

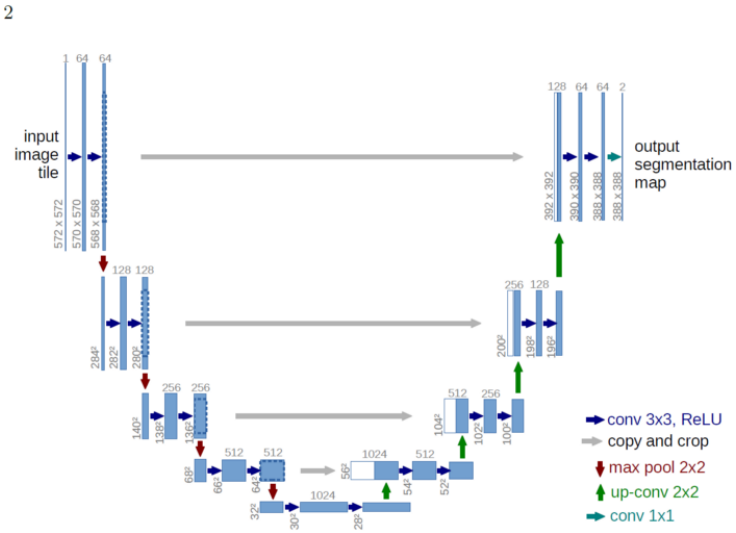

Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

We chose the UNET architecture over other ones because it achieved far greater performance on the ISBI challenge 2015 than other popular networks, and it was recommended by various other papers.
We did mention that this model was initially designed for image segmentation in the biomedical world. The logic behind it should also be applicable to other real-world images. Our research question could be formulated as how well does this architecture work on a more global classification task.

Other considerations

4.3 Metrics

The standard metric keras uses to evaluate the predictions of a model is simply counting the percentage of pixels in the image which were correctly classified. 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 cases (ie. where the class is not present). In this case we would wrongly think our model is doing well, even if it classifies all pixels as negative.

The "best" way to measure the accuracy of a segmentation depends to a large extent on the concrete problem. A simple metric could be the Volume Error:

$$VE = \frac{V1 - V2}{V1 + V2}$$

The problem with this one is that it does not take local deviation into account. A segmentation that has the wrong shape and localization could still have a little volume error.

Dice Score Overlap ratio measures are metrics that apply to many situations. Unlike volume error, they are sensitive to misplacement of the segmentation label, but they are relatively insensitive to volumetric under- and over-estimations. One example is the Jaccard similarity index or the Intersection over Union (IoU) metric.

$$IoU = \frac{target \cap prediction}{target \cup prediction}$$

Or in case of binary classification:

$$JI = \frac{TP}{TN + FN + FP}$$

The current most popular overlap metric is the Dice Score. It is relatively similar to the Jaccard Index, but counts True positives (TP) and True negative (TN) twice. Dice Score:

$$DSC = \frac{2 * TP}{2 * TN + FN + FP}$$

Other possible metrics are Recall, Precision,...

4.4 loss function

These metrics give a more accurate idea of the overall accuracy for an entire image. In order to train the neural network, we need a loss function: this is a real-valued number which has to be minimized. The current loss function we used for the experiments is the binary cross-entropy loss function. It is widely used as it is computationally efficient (easily differentiable). Some research has been done regarding the use of the dice score in the loss minimization ([Fid+17]). The average Dice Score could easily be used. It will however be necessary to adapt it to take the thresholding into account, as the networks outputs probabilities, while the dice score is computed based on a binary classification (0 or 1). A good idea would be to incorporate the learning of the optimal threshold into the network. The biggest advantage is of course that it wouldn't suffer from class imbalance anymore. Others argue that the gradients of the dice score are ugly, i.e. they could in some cases blow up to huge values, so the training would become unstable (to be verified). We didn't train a network this way as training takes a lot of computing time unfortunately.

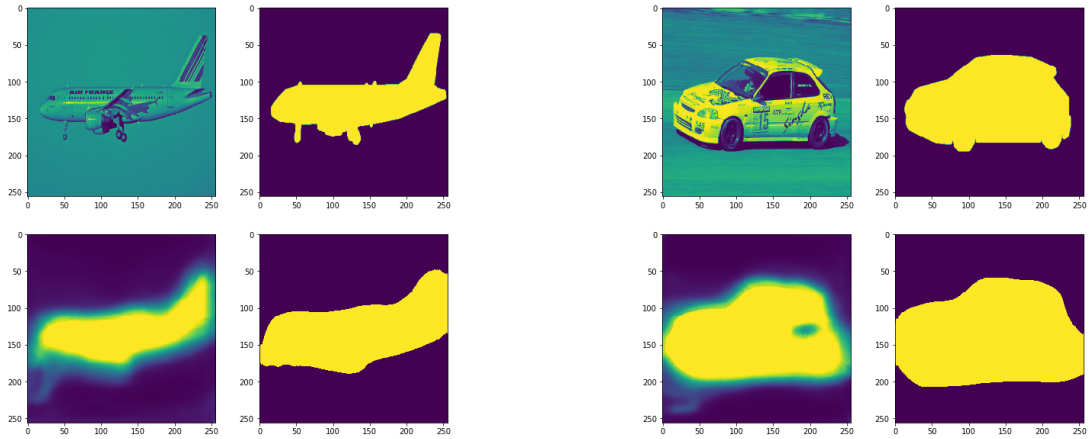

Figure 10: Results of using the segmentation network on the validation dataset. 2 upper pictures: initial image and ground truth. 2 lower pictures: network output (probabilities and thresholded)

Threshold We discussed the accuracy metrics in the section above. These metrics are computed based on binary input images: all pixels are either positive (1) or negative (0). However, our network produces probabilities. We have to threshold the output and assign them to one of the 2 classes based on these probabilities. As there are a lot more negative pixels then positive ones (in the ground truth), we expect that the probabilities will be biased downwards (i.e. the threshold will be lower than 0.5).
So we should try to find the optimal threshold to maximize our accuracy. Here, we can make efficient use of the Dice Score. We thresholded the images with multiple possible thresholds between 0.2 and 0.5 and retained the one that gave the best average Dice Score as a result (0.2).

4.5 Results

We trained our model on the 5 classes in grayscale for about 70 epochs in batches of 2. Bigger batch size would probably have been a bit better to average the gradients but wasn't possible due to limited memory (as we used a relatively big network).

Grayscale (5 classes) The training data here consisted of 143 training images and 150 validation images. The learning curve (training loss and validation loss) is plotted in 9.

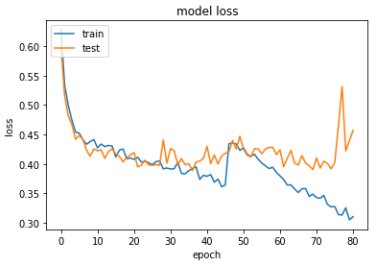

Figure 9: This plot plots the cross-entropy loss on the training data (blue) and the validation data (orange). We can clearly see that the model tends to over-fit at the end.

We can clearly see that the model learns steadily and begins to overfit around epoch 35. Some results are given in 10 and 11

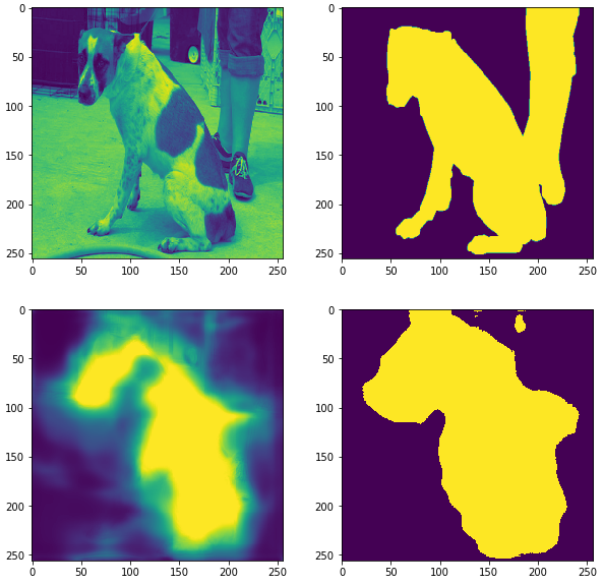

Figure 11: Results of using the segmentation network on the validation dataset. 2 upper pictures: initial image and ground truth. 2 lower pictures: network output (probabilities and thresholded)

In the upper left corner is given the initial image, and left from it the segmentation ground truth. Underneath are plotted the results of our network. To the left: the probabilities as outputted from the network. To the right: result after thresholding. We can clearly see that the networks does find the approximative location of the foreground objects, but it's still quite distorted. This is probably due to the limited dataset, and insufficient training. The average accurracy (cross-entropy) on the validation data at the end is 0.81. We also computed the Dice Score : 0.53. Globally, the results are satisfactory, but there is still room for improvement.

References

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Ed. by Nassir Navab et al. Cham: Springer International Publishing, 2015, pp. 234–241. ISBN: 978-3-319-24574-4.

[Fid+17] Lucas Fidon et al. "Generalised Wasserstein Dice Score for Imbalanced Multi-class Segmentation using Holistic Convolutional Networks". In: *CoRR* abs/1707.00478 (2017). arXiv: 1707.00478. URL: http://arxiv.org/abs/1707.00478.