Comeputer Vision Project : In the name of deep learning

Isabel Montón Quesada
Colin Comey

Contents

1 Introduction

1.1 Overview

This project consists of 3 parts and is concerned with training deep neural networks for image classification and image segmentation. Deep neural networks are at the forefront for solving modern complex computer vision tasks. This is primarily due to their ability to generalise over extremely large data sets which can provide excellent scalable and adaptable characteristics.

1.2 Part 1: Data

Part 1 deals with obtaining the dataset, and splitting it into training, validation and test sets. The dataset which was used was the PASCAL VOC-2009 dataset which contains over 7000 images of 20 specific object classes. In order to reduce the complexity of the task, the neural networks were trained using fewer classes, in this case two. The remaining classes are "chair" and "car".

1.3 Part 2 : Model network architectures

Part 2 of the project introduces auto-encoders and convolutional neural networks. Similar to PCA, the primary motivation for using autoencoders is to achieve a dimensionality reduction. They are a type of feedforward neural network whereby an image is fed into the network and the same image is then reconstructed as well as possible at the output. They consist of 3 components: the encoder, the code and the decoder. The idea behind it is that hidden layers in the middle consist of fewer nodes than the input and output layers, thus by using this layer to then represent the image a dimensionality reduction is achieved. This is also known as latent space representation. A huge advantage of using autencoders is that they are unsupervised and self-labelling. [1]

Convolutional neural networks is at the cutting edge for current image problems due to their accuracy and efficiency. They consist mainly of an input layer, multiple convolution and pooling layers and multiple fully connected layers. If the task is multi-class classification a soft max layer can be used as the final layer.

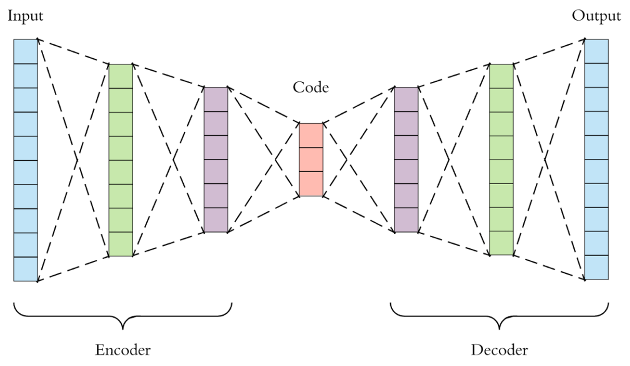


Figure 1: Auto-encoder image. [2]

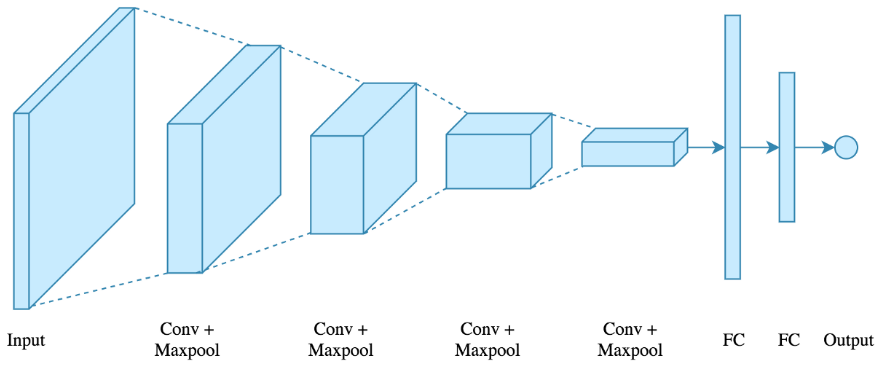


Figure 2: CNN architecture image. [2]

In the convolutional layer, a filter / kernel is passed over the image to obtain a feature map. Pooling layers are then used to down sample the obtained feature map and also combat overfitting. The output of the final pooling layer is flattened and used as the input to the first fully connected layer.

1.4 Part 3 : Classification

In part 3 the designed network architectures are put to the test and used for image classification. During this section a couple different classifiers were designed. The first of which took the trained encoder layers of the auto-encoder and added a fully connected layer together with a soft max layer for classification. The weights of the encoder layers were frozen before training. This means that during training only the weights of the fully connected and soft max layer would be updated.
The second model used exactly the same architecture, however in this instance random weights were assigned in advance of training.

1.5 Part 4 : Segmentation

Part 4 is concerned with constructing a neural network for pixel-wise classification as opposed to image classification. In other words a '1' will be assigned to the pixels of an image that are deemed to contain an object, and a '0' is assigned otherwise. This operation results in image segmentation.

2 Auto-encoders

2.1 PCA vs auto-encoder

Principal Component Analysis (PCA) is a statistical technique which aims dimensionality reduction to increase the computational efficiency while retaining most of the information. As a difference with an auto-encoder, PCA can only perform linear transformations, whilst auto-encoders can have non-linear encoders. Both techniques are directed to analyze data, reducing the features observed, and make predictive models.

To mimic a linear PCA, the auto-encoder should be a linear auto-encoder with one-hidden fully connected layer neural network and with the loss function of the *mean squared error*. However, the hidden layer of the auto-encoder will not be exactly the same as the eigenvectors from the PCA technique, because PCA uses an orthogonal transformation, to transform a set of correlated variables into a set of linearly uncorrelated variables (finding the direction along which the variance is maximum). However, both methods minimize the reconstruction error and can yield the same result. [3]

2.2 Non-linear and convolutional

2.2.1 Building the training, validation and test sets

As stated earlier, to reduce the complexity of the task only two object classes were used. All files in the root folders containing the labels "chair" and "car" were read into a arrays. The validation data was then randomly split into 80% validation data and 20% test data, which was kept for testing the models, thanks to *train_test_split* module of scikit-learn. Therefore, there were 710 images for training, 550 images for validating and 138 images for testing.

2.2.2 Constructing the auto-encoder

The auto-encoder consisted in 3 convolutional layers.

The encoding part compresses the data and results in the encoding layer which is going to be used for reconstructing the images. The encoder was formed by:

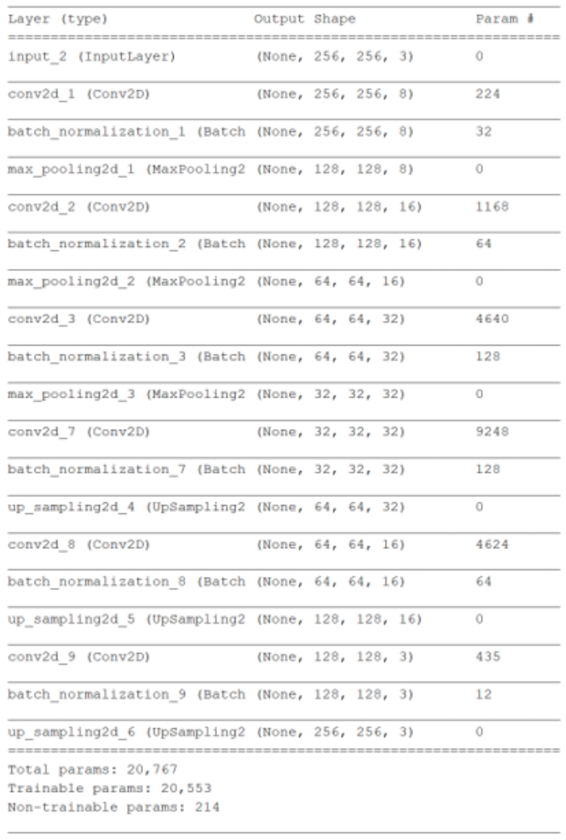
- First layer: 8 filters of kernel 3x3, followed by a batch normalisation layer and a downsampling (max pooling layer).
- Second layer: 16 filters of kernel 3x3, followed by a batch normalisation layer and a downsampling.
- Third layer: 32 filters of kernel 3x3, followed by a batch normalisation layer.

This last layer was downsampled, and is referred to as the code. It is the highest level of feature extraction the occurs in the auto encoder and is encoded in relatively few dimensions, in this case (32,32,32) compared to the input data of (256,256,3).

The decoder was formed by:

- First layer: 32 filters of kernel 3x3, followed by an upsampling layer.
- Second layer: 16 filters of kernel 3x3, followed by an upsampling layer.
- Third layer: 3 filters of kernel 3x3 followed by an upsampling layer.

The primary reason for performing batch normalisation was that it allowed for a faster learning rate by normalizing the input layer (adjusting and scaling the activations).



```
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 256, 256, 3)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 256, 256, 8)       224
_________________________________________________________________
batch_normalization_1 (Batch (None, 256, 256, 8)       32
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 128, 128, 8)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 128, 128, 16)      1168
_________________________________________________________________
batch_normalization_2 (Batch (None, 128, 128, 16)      64
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 64, 64, 16)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 64, 64, 32)        4640
_________________________________________________________________
batch_normalization_3 (Batch (None, 64, 64, 32)        128
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 32, 32, 32)        9248
_________________________________________________________________
batch_normalization_7 (Batch (None, 32, 32, 32)        128
_________________________________________________________________
up_sampling2d_4 (UpSampling2 (None, 64, 64, 32)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 64, 64, 16)        4624
_________________________________________________________________
batch_normalization_8 (Batch (None, 64, 64, 16)        64
_________________________________________________________________
up_sampling2d_5 (UpSampling2 (None, 128, 128, 16)      0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 128, 128, 3)       435
_________________________________________________________________
batch_normalization_9 (Batch (None, 128, 128, 3)       12
_________________________________________________________________
up_sampling2d_6 (UpSampling2 (None, 256, 256, 3)       0
=================================================================
Total params: 20,767
Trainable params: 20,553
Non-trainable params: 214
_________________________________________________________________
```

Figure 3: Summary of the autoencoder.

2.2.3 Training the autoencoder

The autoencoder was compiled using the *Adam* optimisation algorithm and *mean squared error* loss function. The *Adam* gradient descent is essentially an extension of the stochastic gradient descent algorithm. Keras uses the following default parameters: learning rate = 0.001, beta$_1$ $= 0.9, beta_2 = 0.999, epsilon = 1e - 08, decay = 0.0$.

The loss function was selected to maximize the likelihood between the input and the output images. A benefit from using this function is that it has consistency: as the number of samples increases, the estimation improves. Moreover, this function is appropriate for regression predictive modelling problems. [4]

In order to train the network, the training images were set as both the input and target (label) layer and the error was minimised between the two. The training was performed over 50 epochs. A validation test was used in order to ensure that overfitting was not occurring. The performance of the sets was compared using the accuracy metric.

2.2.4 Autoencoder results and performance

As displayed in the figure below, the validation loss was never increasing continuesly, which implies that overfitting did not occur. It can be seen that both plots even off at a loss of less than 0.01. This indicates that the model was generalising well, and that the reconstruction error would be relatively low.

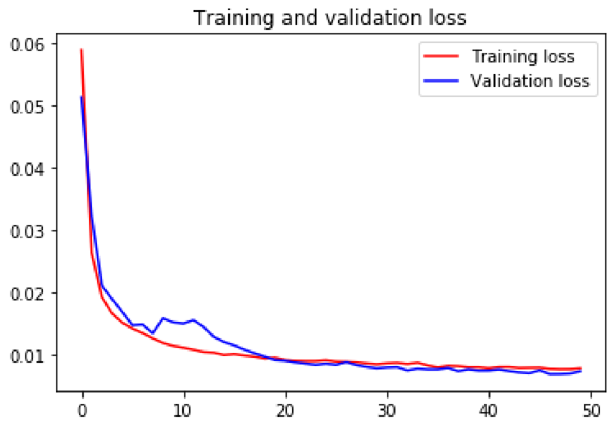


Figure 4: Plot showing the training and validation loss during training of the autoencoder.

When reconstructing the images the model performed reasonably well. It was extremely clear what objects were in the images which indicated that high level feature extraction occurred.

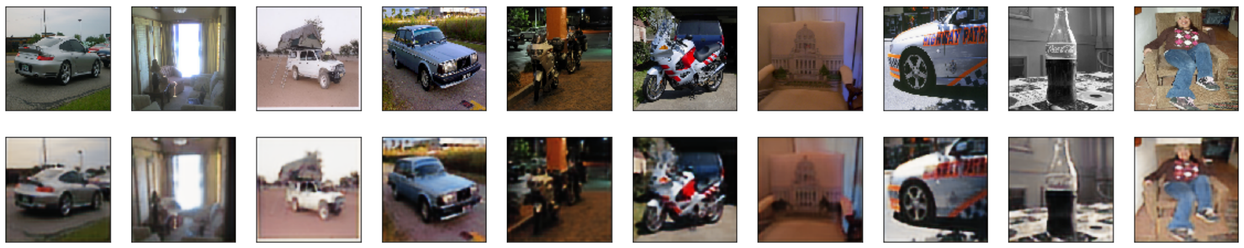
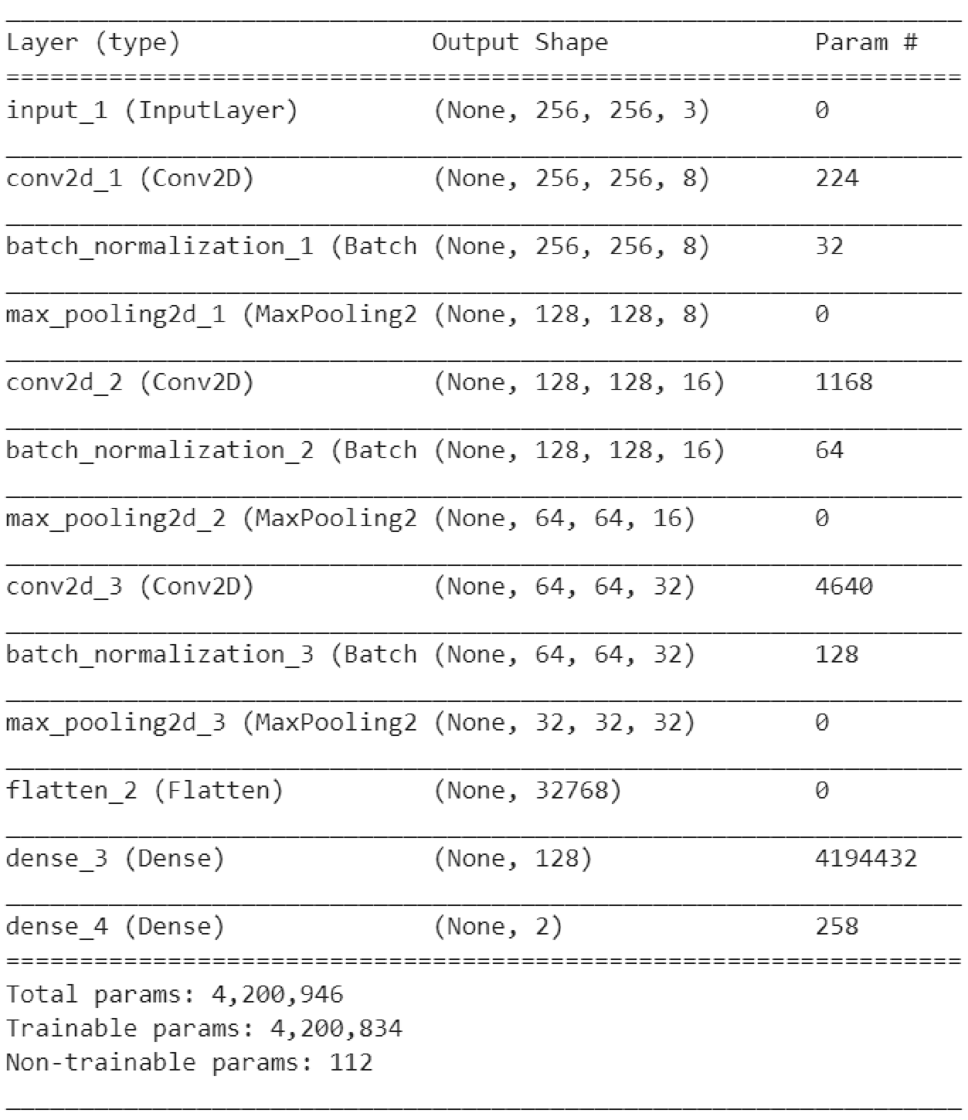


Figure 5: Reconstructed images from the autoencoder.

3 Classification

3.1 Constructing the classifiers

Two classification models were used. Both models used the same encoder architecture as the decoder part of the autoencoder was essentially stripped away and a fully connected layer and soft max layer were added in its place. The fully connected layer contained 128 neurons and the soft max layer contained 2, as it was a binary classification task. The weights of the second model were initialised randomly with the kernel initializer from Keras *glorot_uniform* prior to training, whereas the weights of the first were not. Both networks followed the architectures in the figures below.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         (None, 256, 256, 3)       0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 256, 256, 8)       224       
_________________________________________________________________
batch_normalization_1 (Batch (None, 256, 256, 8)       32        
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 128, 128, 8)       0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 128, 128, 16)      1168      
_________________________________________________________________
batch_normalization_2 (Batch (None, 128, 128, 16)      64        
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 64, 64, 16)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 64, 64, 32)        4640      
_________________________________________________________________
batch_normalization_3 (Batch (None, 64, 64, 32)        128       
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 32, 32, 32)        0         
_________________________________________________________________
flatten_2 (Flatten)          (None, 32768)             0         
_________________________________________________________________
dense_3 (Dense)              (None, 128)               4194432   
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 258       
=================================================================
Total params: 4,200,946
Trainable params: 4,200,834
Non-trainable params: 112
_________________________________________________________________
```

Figure 6: Summary of the layer architecture in the classification models 1 and 2.

The optimizer used was the *stochastic gradient descent*. In this case, as it was a problem of classification, the loss function that was used was the *binary_crossentropy* function. Cross-entropy is used to estimate the difference between the training data a predicted and estimated probability distributions. As the classification problem consisted on predicting the likelihood of the input image to belong to class “car” or class “chair”, that is, a binary classification problem, the loss function stated was used and thus, the log likelihood (which takes into account the uncertainty of the estimation based on its variation considering the actual label) was maximized. [4]

The non-linearity used for the encoder was the ReLu non-linearity, with the purpose of introducing non-linearities to layers that have being only using linear operations (convolutions), by changing all the negative activations to zero. Moreover, it is really fast to evaluate. However, for the fully connected layer, the non-linearity used was the *softmax* non-linearity, which tells the probability of the classes to be true. The *sigmoid* does the same, but only for binary class problems (so in this case it could be also have been used). [5]

3.2 Classifier results and performance

Once training was complete, the classifiers were tested on the 710 images and the 138 test images. Model 1 predicted 99.15% and 75.72% in the training and testing sets, respectively. Model 2 predicted 99.15% and 76.45% of the training and testing sets images, respectively.

The model that predicted best and resulted in a better outcome was the second one. As our neural network was trained using an stochastic optimization algorithm (stochastic gradient descent: *Adam*) which used randomness for selecting a starting point for the start of the training, the use of small random values for the initialization, improved the performance of the model as it can be observed in the accuracy parameters of both models (in model 2 slightly better). So, the evaluation was only made in the second model, as it was the best one between the two models. [6]

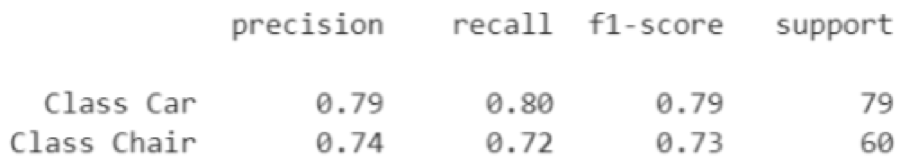
The precision, recall and F1 scores for model 2 are displayed in figure 7.

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

$$F1 = 2 * \frac{precision * recall}{precision + recall} \tag{3}$$

Precision is an important metric when there is a high cost associated with false positives (FP) and recall is a better metric to use when there is a high cost associated with false negatives (FN). F1 is a good metric to consider if one wishes to strike a balance between the two. The model performed slightly better on the images which contained cars, however it must be taken into consideration that it was trained on more car images.



```
             precision    recall  f1-score   support

   Class Car       0.79      0.80      0.79        79
 Class Chair       0.74      0.72      0.73        60
```

Figure 7: Precission, recall and F1 scores for model 2.

As a result of the testing, 105 labels were predicted correctly opposing to 33 labels predicted incorrectly (76% correct, 24% incorrect).

4 Segmentation

4.1 Loading and binarizing the data

The aim of the segmentation task was to implement a CNN for pixel-wise classification in order to perform semantic segmentation of images. For that purpose, the images corresponding to "car" and "chair" from the SegmentationClass folder of the dataset were used. Again, the validation set was split randomly in validating (80%) and test (20%) images. Therefore, 132 training images, 84 validating images and 22 test images were obtained.

To obtain binary images, split in foreground and background, all the objects available in the image were coloured as red and the background as black. This resulted in an easier implementation of the methods utilised.

4.2 Construction of the segmentation CNN

Further, the segmentation CNN was built. The U-net architecture or fully connected convolutional network (FCN) was used. As it can be observed in figure 8, the U-net architecture is formed by a contracting part and a expansive part, more or less symmetrical. In between those parts, there are skip connections referred in the code as concatenations, meant to improve the reconstruction of the images by combining high resolution features and spatial information. The contracting part is formed by a convolutional network followed by ReLu non-linearities and max pooling. The expansive part is formed by upsamplings and convolutions. [7]

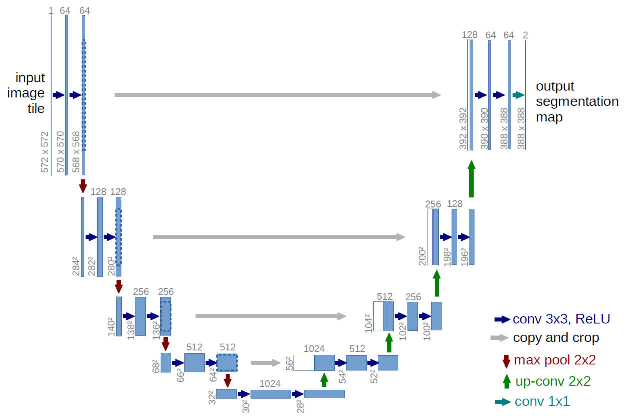


Figure 8: U-net Architecture. [7]

4.3 Results and performance

To compile the model, the *Adam* optimizer and the *binary_crossentropy* loss function were again used. The model was trained using the training data as input and the segmentation data as labels. As it can be observed, the loss function decreased as epochs increased, respectively. Ten of the predicted images are shown in figure 10.

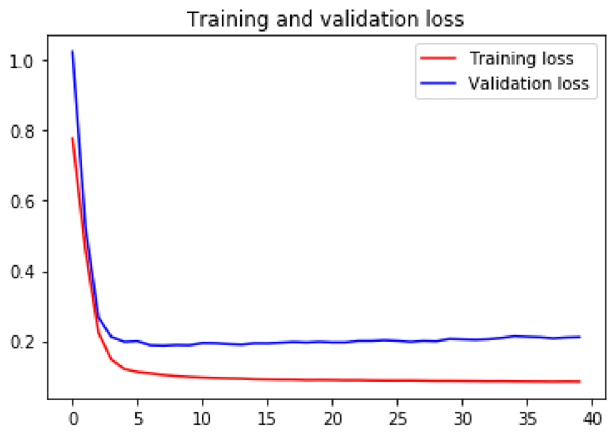


Figure 9: Loss from training and validating set.

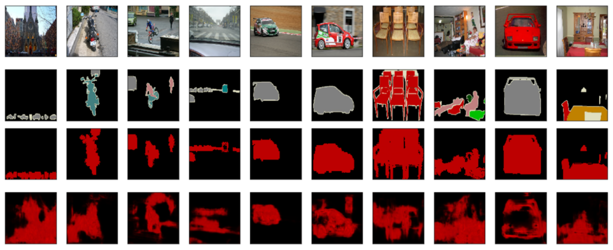
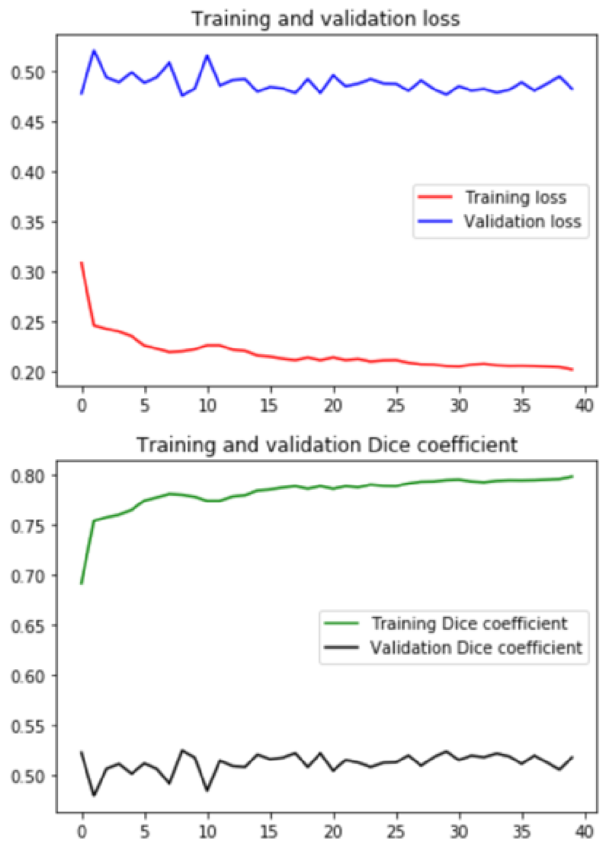


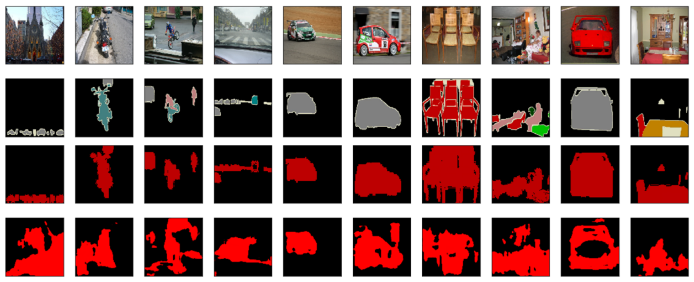
Figure 10: Recontructed images. *The first row corresponds to the original image, the second one to the image from SegmentationClass folder, the third to the binary image used as label and the last one, the outcome of the segmentation CNN.*

However, a better metric to evaluate the performance of the segmentation CNN could be the intersection over union (IoU) method. It evaluates the area of overlapping divided by the area of union of the predicted and ground truth segmented images. In this way, it can be known which percentage of the predicted segmented foreground matches the ground truth segmented foreground, being the 100% a perfect match. To prove that, a metric for evaluating the IoU, the *Dice score* coefficient was computed. This metric was used too as the loss function for the compilation of the model (1-*DiceScore*). [8]

$$DiceScore = 2 * \frac{|true \cap predicted|}{|true| + |predicted|} \tag{4}$$



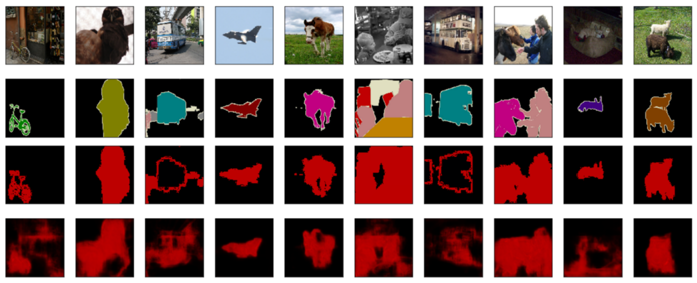
(a) Dice loss and Dice coefficient from training and validating set.



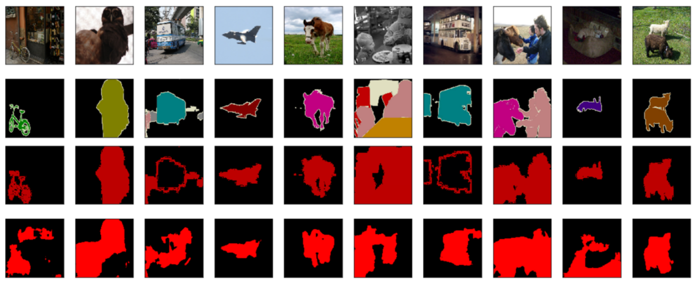
(b) Reconstruction with Dice loss

Figure 11: Results when using the Dice score coefficient.

The training and validating sets didn't achieve a very high Dice score, although it was around 50%. When all the data of the SegmentationClass was used, the performance results improved, something that can be observed in the reconstructed segmentation images. The advantage respect to using the previous loss function was that in this case, a more opaque and firm segmentation was performed.



(a) Reconstruction with *binary_crossentropy* loss.



(b) Reconstruction with Dice loss

Figure 12: Results when using the Dice score coefficient with all the SegmentationClass images.

Another measured for evaluating the IoU is the *Jaccard* coefficient. However, the *Dice* coefficient seems to be more intuitive as it can be seen as the percentage of overlap between the ground truth and the predicted result. [8]

$$Jaccard = \frac{|true \cap predicted|}{|true| + |predicted| - |true \cap predicted|} \tag{5}$$

Segmentation methods with high precision and reproducibility are very important in medical context as, for example, in tumour resection surgeries. When more accurate is the outcome, more healthy tissue can be preserved. It is also important in determine brain deterioration diseases, as the quantity of white and grey matter is important in this pathologies to have an early diagnosis. [9]

Together with the Dice coefficient and the *Jaccard* index, other metrics to evaluate image segmentation tasks are: the “sensitivity and recall” metric, which measures the portion of positive pixels in the ground truth that are also identified as positive in the prediction image, the “specificity” (analogous to the previous one), which measures the portion of negative pixels (background) in the same ways as in the previous evaluation method, among others. [9]

5 Conclusion

As an overall view, the results of the first two parts were very good, however, in the segmentation part the outcome was worse when utilising only two classes. Although sometimes an intuitive form could be deduced from the predicted images, in most of the cases, the segmentation between foreground and background was not good. This could be resolved by augmenting data, or trying other architectures, as well as implementing *Jaccard* coefficient as a metric to evaluate the segmentation CNN.

References

[1] M. Stewart, "Comprehensive introduction to autoencoders," https://towardsdatascience.com/generating-images-with-autoencoders-77fd3a8dd368, 2019.

[2] A. Dertat, "Applied deep learning - part 4: Convolutional neural networks," https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2mages-with-autoencoders-7 2017.

[3] O. Cohen, "Pca vs autoencoders," https://towardsdatascience.com/pca-vs-autoencoders-1ba08362f450, 2018.

[4] S. A. Czepiel, "Maximum likelihood estimation of logistic regression models: Theory and implementation," https://czep.net/stat/mlelr.pdf.

[5] J. Yang, "Relu and softmax activation functions," https://github.com/Kulbear/deep-learning-nano-foundation/wiki/ReLU-and-Softmax-Activation-Functions, 2017.

[6] J. Brownlee, "Why initialize a neural network with random weights?" https://machinelearningmastery.com/why-initialize-a-neural-network-with-random-weights/, 2018.

[7] S. Karagiannakos, "Semantic segmentation in the era of neural networks," https://sergioskar.github.io/Semantic_Segmentation/, 2019.

[8] A. Rosebrock, "Intersection over union (iou) for object detection," https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/, 2016.

[9] A. H. Abdel Aziz Taha, "Metrics for evaluating 3d medical image segmentation: analysis, selection, and tool," https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4533825/, 2015.

[10] J. Jordan, "Evaluating image segmentation models." https://www.jeremyjordan.me/evaluating-image-segmentation-models/, 2018.

[11] M. Babaee, "A deep convolutional neural network for background subtraction," https://arxiv.org/pdf/1702.01731.pdf, 2017.

[12] G. Shperber, "Background removal with deep learning," https://towardsdatascience.com/background-removal-with-deep-learning-c4f2104b3157, 2017.

[13] A. Karpathy, "A recipe for training neural networks," https://karpathy.github.io/2019/04/25/recipe/, 2019.

[14] P. Dwivedi, "Using tensorflow object detection to do pixel wise classification," https://towardsdatascience.com/using-tensorflow-object-detection-to-do-pixel-wise-classification-702bf2605182, 2018.

[15] A. Rosebrock, “Keras conv2d and convolutional layers,” https://www.pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/, 2018.

[16] J. Brownlee, “How to improve deep learning performance,” https://machinelearningmastery.com/improve-deep-learning-performance/, 2016.

[17] “Usage of loss functions,” https://keras.io/losses/.

[18] T. Dehaene, “Variational autoencoders for new fruits with keras and pytorch,” https://becominghuman.ai/variational-autoencoders-for-new-fruits-with-keras-and-pytorch-6d0cfc4eeabd, 2018.

[19] A. Sharma, “Implementing autoencoders in keras: Tutorial,” https://www.datacamp.com/community/tutorials/autoencoder-keras-tutorial, 2018.

[20] S. Saha, “A comprehensive guide to convolutional neural networks—the eli5 way,” https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, 2018.

[21] “Chapter 4. fully connected deep networks,” https://www.oreilly.com/library/view/tensorflow-for-deep/9781491980446/ch04.html, 2018.