In the name of deep learning

Eric Massip Cano[1] **and Manuel Palacios Hurtado**[2]

[1]eric.massipcano@student.kuleuven.be
[2]manuel.palacioshurtado@student.kuleuven.be

ABSTRACT

In this document we analyze the value of Convolutional Neural Networks for three topics in image processing, feature extraction, classification and segmentation and we describe theoretically the main differences between Principal Component Analysis and Neural Networks. The data used in this report is a subset of a well-known dataset, PASCAL VOC-2009[1], which has been used to train different architectures of neural networks and check their performance. In feature extraction we have tested different codification factors, such as, 6, 12 and 24 achieving a Mean Squared Error of 0.01. In image classification, we could demonstrate that the Deep CNN performs widely better than a feature extraction architecture with a fully connected layer. Finally, in the segmentation section we obtained a promising pixel-wise classifier with a Dice coefficient value of 0.68.

Introduction

Deep learning is shown to be the best technique for image processing, especially for feature extraction, classification and segmentation. Especially, in this document we are developing different Neural Networks for each topic mentioned before and we are analyzing the results. In the first section, we define what Principal Component Analysis and Neural Networks are as well as their main similarities and differences. In the second section, we implement a CNN as an autoencoder with different layers to get different coding variable sizes and perform the reconstruction. In the third section, we compare some image classifiers, freezing the autoencoder layers and adding a fully connected layer, retraining the complete autoencoder or using a deep CNN. In addition, we propose as future work the use of a pretrained model, such as AlexNet. In the forth section, we deploy a pixel-wise classifier in gray scale for different types of classes.

Principal Component Analysis and Neural Networks

In this section a short definition of Principal Component Analysis and Neural Network architectures is made and we summarize the main differences and similarities. PCA is an orthogonal transformation to extract linearity uncorrelated variables from a correlated set of data, where there is a number of components, k, and a number of original variables, n, that follow the condition $k <= n$[2]. A neural network is a black box model composed by complex layers that generates a non-linear output based on certain input. In the literature, we notice that there is a wide amount of architectures, each one suits a different application. Therefore, a NN could simulate the PCA analysis and extract the eigenvectors and eigenvalues, the architecture is formed by an input layer and an output layer with a linear activation function. Even though they could be used as feature extraction technique, there are few differences, that are analyzed in[3].

Auto-encoder

This section introduces our approach to build and train an auto-encoder, which goal is to reduce the amount of features of an input image through the encoder, and then reconstruct it again through the decoder. First, the dataset characteristics and preprocessing is presented. Secondly, the network architecture and the loss functions used are explained. Finally, the training and evaluation performance measurements are analyzed.

Dataset

The available dataset is the PASCAL VOC-2009 dataset[1]. It is a standard dataset for image segmentation and classification. It has 20 different classes such as *dog*, *bird* or *car*. Our preprocessed dataset has precisely 657 number of images in the training set and 644 number of images in the validation set. The images belong to the classes *cat*, *motorbike* and *bottle*. All the pictures have been normalized to limit the value of the weights and resized to 128x128

because is an optimal trade-off between resolution and size of the neural network. A sample image from each of the classes is shown in Figure 1.

Figure 1. Random samples of a *cat*, *motorbike* and *bottle* from the training set.

Network

A CNN with an encoder/decoder structure is implemented and its architecture and the loss functions used are explained in this section. The first part (i.e. encoder) has the following architecture. The input layer has a size of 128x128x3, because the input images are 128x128 and there is one channel for each color (i.e. RGB). The following layers are a sequence of convolutional layers with (3,3) filters and padding *same* and max-pooling layers with a (2,2) window and *ReLU* activations. The amount of filters in every convolutional is 256, 128, 64, 32, 16, 8, respectively. The rest of the network is part of the decoder, it is the inverse of the encoder. There are six convolutional layers with (3,3) filters and padding *same* in the decoder. Each of them is followed by an upsampling layer with a (2,2) window and all have *ReLu* activations, except from the last one. They have 16, 32, 64, 128 and 256 filters, respectively. The last convolutional layer has 3 (3,3) filters and a *sigmoid* activation function.

Loss functions

The convolutional neural network has been trained to minimize the *mean squared error*, but we have also experimented with the *mean absolute error* function. Both loss functions are described below:

- **Mean Squared Error (MSE)**: The MSE is the average sum of squared differences between two arrays.

$$MSE = \frac{1}{N}\sum^{N}(x-\hat{x})^2 \tag{1}$$

- **Mean Absolute Error (MAE)**: The MAE is the average sum of absolute differences between two arrays.

$$MAE = \frac{1}{N}\sum^{N}|x-\hat{x}| \tag{2}$$

Training the Network

The network is trained for 60 epochs with a batch size of 32. The training is done in a Google Colab notebook with the GPU runtime activated. As a result, the training of one single epoch takes less than 5 seconds. The optimizer used for the training is *rmsprop*. The loss function chosen to be minimized is *mse*. We also tried the loss function *mae* but the results were almost equal.

In order to compare the performance with different numbers of coding variables, 3 different networks were trained. The first net has 2048 coding variables between the encoder and the decoder. The second reduces one convolutional and max-pooling layers at each side of the auto-encoder, so it has 4096 coding variables. The third reduces two convolutional and max-pooling layers at each side of the auto-encoder, so it has 8192 coding variables. The evolution of the training and validation of the first and deepest (i.e. the one with 2048 coding variables) network is shown on Figure 2.

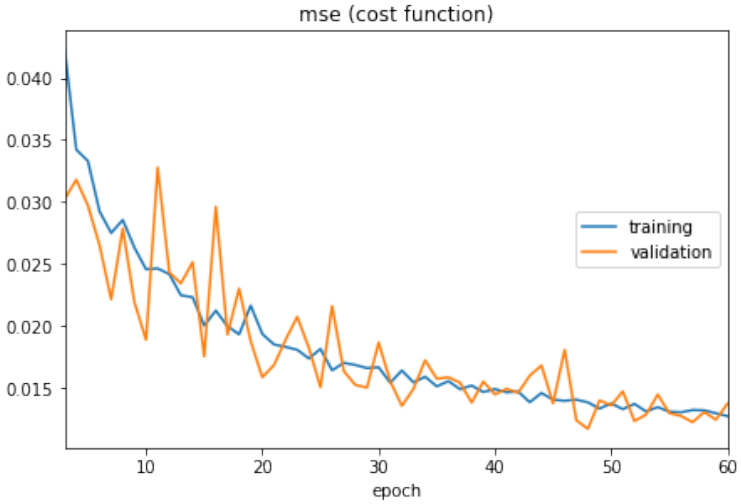


Figure 2. Evolution of the loss function training the deepest auto-encoder with 2048 coding variables.

Coding variables	MSE	MAE
2048	0.04	0.15
4096	0.02	0.10
8192	**0.01**	**0.08**

Table 1. Comparison of the performance of the 3 implemented networks (i.e. 8192, 4096 and 2048 coding variables) in terms of MSE and MAE.

Results

This section compares the results of the three networks previously introduced in terms of the *mean squared error* and the *mean absolute error*. Table 1 shows the performance of the 3 presented networks in terms of MSE and MAE.

The reason why the MSE is always smaller than the MAE is because of the quadratic term. The difference values when measuring the loss will always be smaller than or equal to 1. In the case of calculating the MSE, each one of those values is squared which makes the value even smaller. That quadratic term does not appear in the measuring of the MAE so it will always be higher than or equal to the MSE.

The narrower network but with more coding variables (i.e. 8196) is the one which performs better. However, the performance of the other two is not bad in terms of MSE and MAE. Figure 3, one descriptive example per class of a reconstructed image by every network.

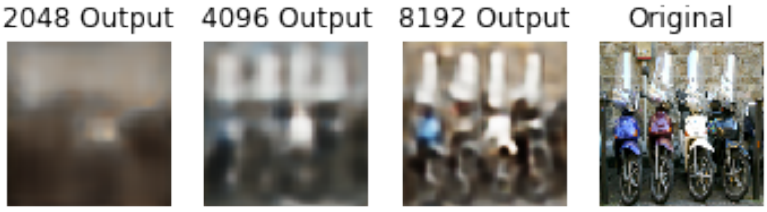
Classifier

In this section we define different neural network structures for image classification with multi-label images. The dataset for training is the same as the autoencoder, however, some of the samples of the validation set were shifted into the training set to improve the training accuracy and have more samples to train. First, the encoder structure was modified to an image classification structure. Secondly, we retrained the complete model of the encoder. Finally, we developed a simple CNN with integrated techniques to avoid overfitting, such as, normalization or regularization.

Frozen encoder

One of the most famous techniques for image classification is the combination of a feature extraction architecture with a simple neural network or fully connected layer. In this case, the first structure is a pretrained model with non-trainable parameters and 8192 coding variables. The second part is a dense layer with 24579 trainable parameters and a sigmoid activation function. In fact, the final model uses a sigmoid activation function and binary cross-entropy as loss function. This choice is fundamentally based on the fact that the dataset includes images with different labels instead of just one class per image, where we could have used a softmax activation function and categorical-cross entropy as loss function.

(a) Cat



(b) Motorbike

(c) Bottle

Figure 3. Reconstruction of validation images for class *cat*, *motorbike* and *bottle* by the the 3 proposed networks (i.e. 2048, 4096 and 8192 coding variables, respectively).

This model is overfitted as it is shown in the difference between the training log-loss function and the validation log-loss function. As we can observe in 2, the gap between these two values is considerably great and it is increasing with the number of epochs. Another interesting result is the accuracy of the complete model, which represents that only 40% of the validation images are classified correctly.

Encoder

The second model that we propose in this report is reusing the encoder structure mentioned in the previous section. This model has 419,043 trainable parameters and 3 sets of layers, convolutional, relu and maxpooling. In order to reinitialize the architecture, the initial weights and bias are set to random and zeros respectively, hence, the network starts the training from scratch.

As it is shown in 2 we have the same situation as the frozen encoder, the model is overfitting. Furthermore, the validation log-loss value is even greater due to the amount of trainable parameters and unlimited increase of the network weights during the training. In summary, we can say that this architecture is not optimal for classification. However, the performance for encoding techniques is excellent.

Deep CNN

As a solution for the overfitting problem, we propose a deep convolutional neural network with batch normalization layers, dropout layers and kernel regularization in some of the convolutional layers. This structure combines 5 set of layers formed by a convolutional, batch normalization, activation, max-pooling and dropout layer. In particular, the activation functions follow the same principle as in the previous section, *ReLu* in the hidden layer to increase the non-linearities and *sigmoid* in the output layer for the multi-label classification task. Additionally, we add

early-stopping to stop the training if the performance does not improve after a certain number of epochs, moreover, the best model is saved during the training. The accuracy is better than the previous ones, even though it is still overfitting and there are some peaks, as it is shown in 4.

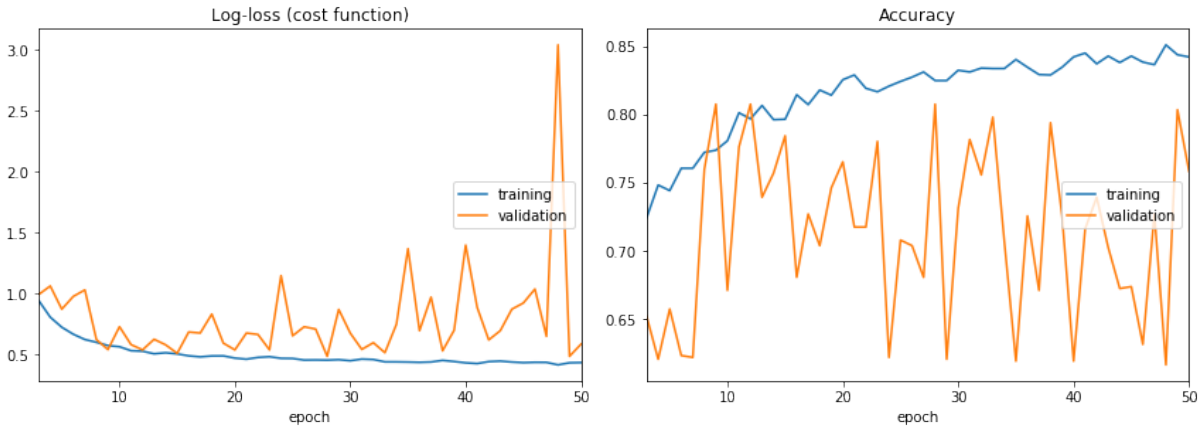


Figure 4. Performance of the simple classifier

Classifier	Validation accuracy	Training log-loss	Validation log-loss
Frozen encoder	0.4139	0.136	0.781
Encoder	0.4672	0.000	2.693
Simple classifier	0.6721	0.466	1.033

Table 2. Comparison of the performance of the classifiers

Pre-trained model

As it is shown in the previous sections, the values of accuracy are not as we expected. However, in some articles[4] they achieve 85.6% of accuracy using the same dataset and a pre-trained model, such as AlexNet. Therefore, one of the future proposals for this project could be the use of this type of models. The process to follow is to add a small dense layer for classification, while the pre-trained weights of the model are freezed, as we did in .

Segmentation

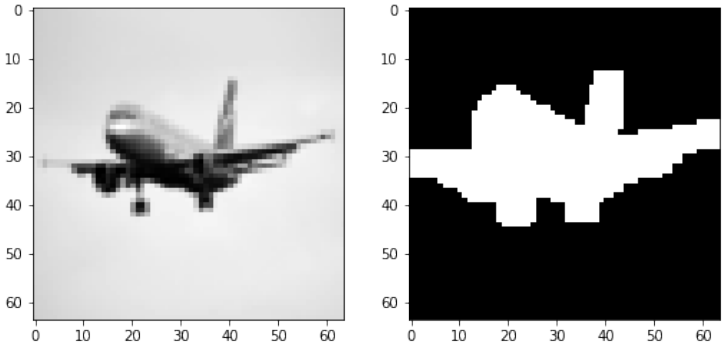
This section presents our approach to build a binary segmentation network. The idea is that, given an input image, the network is able to label correctly the pixels which belong to an object. The result is a binary mask image with the foreground parts of the image in white and the background in black.

Data Preprocessing

The amount of segmented samples in the dataset is 1499. This amount is much more reduced than the amount available for classification, so we choose not to filter for classes but to keep all of them for segmentation. Consequently, the training and validation sets include 749 and 750 images, respectively. Each one of these images is resized to a shape of 64x64 and converted to grayscale. Contrary to the previous sections, this time the training labels are the binary masks of the training feature images. The pixels which belong to an object appearing in every training feature image are set to 1, while the rest of pixels are set to 0. This gives this idea of foreground and background class. There is an example of one of these training feature-label pairs shown in Figure 5. The validation set images go through the exact same process.

Network Architecture

A very similar encoder-decoder network to the one used in the first section is built here for image segmentation. The first part (i.e. encoder) has the following architecture, the input layer has a size of 4096 (i.e. 64x64) and just one color channel because the images were converted to grayscale. The following layers are a sequence of convolutional



(a) Feature image. **(b)** Label image.

Figure 5. Example of one feature image from the training set and its corresponding label.

layers with (3,3) filters and padding *same* and max-pooling layers with a (2,2) window and *ReLu* activations. The first, second, third and fourth hidden layers have 256, 128, 64 and 32 filters, respectively. There is a Dropout layer between the third convolution and pooling layers with a rate of 0.5 in order to prevent overfitting and increase training speed, as shown in[5]. The rest of the network is part of the decoder, it is the inverse of the encoder. There are four convolutional layers with (3,3) filters and padding *same* in the decoder. Each of them is followed by an upsampling layer with a (2,2) window and all have *ReLu* activations, except from the last one. They have 32, 64 and 128 filters, respectively. The last convolutional layer has just 3 (3,3) filters and a *sigmoid* activation function.

Evaluation metrics

There are two kind of metrics used to evaluate the performance of the trained networks, the Dice Coefficient and the Intersection over Union.

Dice Coefficient

The Dice Coefficient measures the overlap between the pixels of two images, essentially the predicted image and the target image. It is calculated by multiplying the intersection of common pixels between both images by a factor of 2, then dividing that term by the sum of the pixels of both images. A Dice score of 1 means there is a perfect overlap, while a Dice score of 0 means there is no overlap at all.

$$Dice = \frac{2\,|target \cap prediction|}{|target| + |prediction|} = \frac{2TP}{2TP + FP + FN} \tag{3}$$

Intersection over Union

The IoU metric, also called the Jaccard index, measures the overlapping percentage between the predicted mask and the target mask. It is calculated dividing the intersection of common pixels between the predicted and target masks by the total number of pixels present across both masks. An IoU of 1 means there is a perfect overlap, while an IoU of 0 means there is no overlap at all.

$$IoU = \frac{target \cap prediction}{target \cup prediction} = \frac{TP}{TP + FP + FN} = \frac{Dice}{2 - Dice} \tag{4}$$

Both metrics above are very similar, but there is a small difference which is worth mentioning, apart from the evident fact that the Dice Coefficient weights the true positives higher. When these two instances are used to calculate the average score over two sets, the IoU tends to penalize more the single misclassifications. For example, imagine there are two classifiers A and B, A performs better than B most of the time but A performs much worse than B in some specific instances, then IoU will probably prefer B while Dice will prefer A.

Training The Network

The network is trained for 40 epochs with a batch size of 32. The training is done in a Colab notebook with the GPU runtime activated. As a result, the training of one single epoch takes less than 4 seconds. The optimizer used for the training is *adam*.

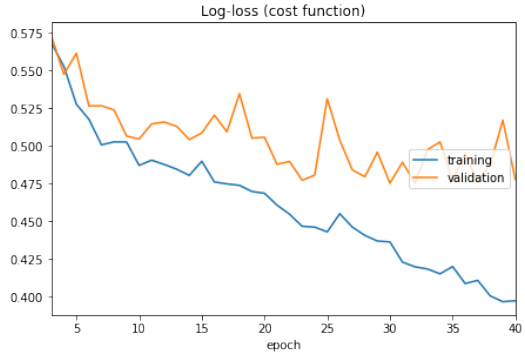


Figure 6. Evolution of the training and validation loss using binary cross-entropy.

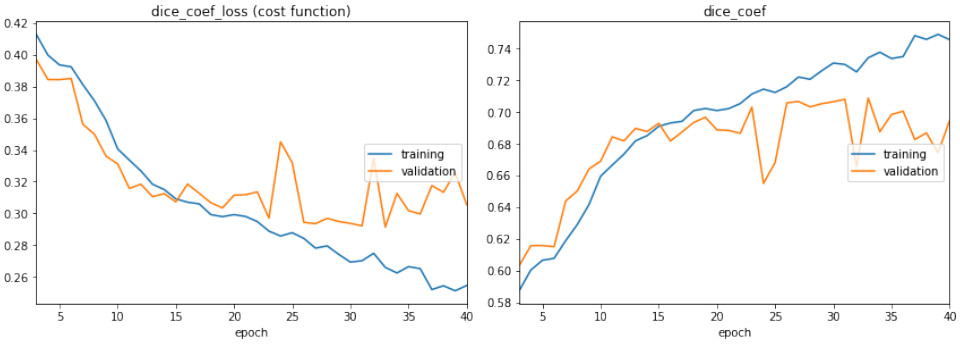


Figure 7. Evolution of the loss and the Dice coefficient for training and validation.

Two networks are trained with different loss functions. The performance of the training for both of them is compared and analyzed in the following sections.

Binary Cross-Entropy

The first network loss function used is *binary cross-entropy*. This loss function is chosen because the goal is to know what is the likelihood for every pixel to be activated. Every neuron in the output layer corresponds to one pixel of the output image and each of them has a *sigmoid* activation function, so each neuron will have a value between 0 and 1 with the likelihood that that pixel is activated. In other words, the neurons (i.e. pixels) more likely to be fired will have a value closer to 1, while the neurons less likely to be fired will have a value closer to 0.

Dice Coefficient

The second network minimizes the Dice loss during training directly. It is a loss function, not a coefficient anymore, so the function to be minimized becomes $DiceLoss = 1 - DiceCoefficient$. The advantage of using the Dice loss over binary cross-entropy is that it helps when there is class imbalance. The amount of pixels on the foreground class fluctuates a lot so there is usually a big imbalance. The Dice loss helps in this matter.

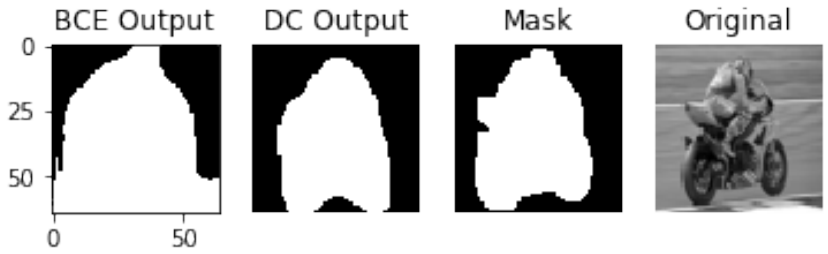
Results

This section compares the results of the two networks previously introduced using the evaluation metrics previously described.

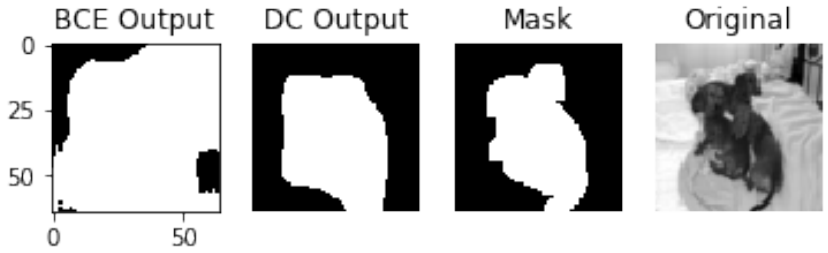
As it can be seen on Table 3, the network using the Dice Loss performs significantly better in terms of Dice Coefficient and Intersection over Union metrics. Figure 8 shows a comparison between the predicted segmented

Network Loss	Dice Coefficient	Intersection over Union
Binary Cross-Entropy	0.55	0.39
Dice Loss	**0.68**	**0.54**

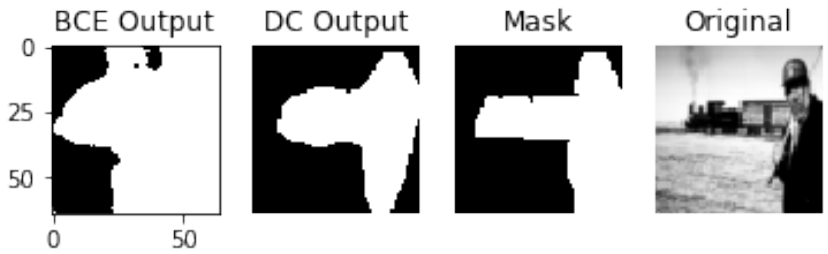
Table 3. Comparison of the performance of the presented neural network with two different loss functions.



(a) Motorbike.



(b) Dogs.



(c) Man and a train.

Figure 8. Examples of segmented images with binary cross-entropy (BCE) and Dice Loss (DC).

images using both available loss functions against their correspondent target image.

It seems like the Dice Loss trained network tends to clean up the images and draws better boundaries. At the bottom center of Figure 8a, it is noticeable the detailed boundary that marks the space between the motorbike wheels. Similarly, the tiny space between the dogs can slightly be appreciated as well as the tail on Figure 8b. Also, a very good example of the clean up power of the Dice Loss is shown on Figure 8c.

Conclusion

This report includes some examples of feature extraction, classification and segmentation using the images of the dataset PASCAL VOC-2009. The obtained result are successfully positives with feature extraction and segmentation, however, in the case of image classification, it is shown that the models are overfitting and is essentially necessary to use a pretrained model to extract a value of accuracy higher than 70%.

References

1. Everingham, M. *et al.* The PASCAL Visual Object Classes (VOC) Challenge. Tech. Rep.
2. Salkind, N. Principal Components Analysis. In *Encyclopedia of Research Design*, vol. 1, 1–6, DOI: 10.4135/9781412961288.n334 (SAGE Publications, Inc., 2455 Teller Road, Thousand Oaks California 91320 United States, 2012).
3. Jiménez, R. *et al.* Dimensionality Reduction in Data Mining Using Artificial Neural Networks. *Methodology* **5**, 26–34, DOI: 10.1027/1614-2241.5.1.26 (2009).
4. Shetty, S. Application of Convolutional Neural Network for Image Classification on Pascal VOC Challenge 2012 dataset. (2016). 1607.03785.
5. Srivastava, N., Hinton, G., Krizhevsky, A. & Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Tech. Rep. (2014).