Computer Vision: In the name of deep learning

David Schotmans & Jesse Smits

June 2019

Abstract

In this paper we will discuss 3 different learning tasks regarding image processing. For each task we use the VOC2009 dataset and deep neural networks. The first task is the create a convolutional autoencoder to reduce the size of the original images and to represent them in a low dimension feature space. The following task is the build a fully connected classification network on top off the encoding network learned in the previous task. The last task is to create a segmentation network. For this task we consulted existing architectures and adapted them to perform optimally for our dataset.

1 Introduction

This report discusses the project for Computer Vision. There is an accompanying notebook that explains the different tasks in a similar way as this report, but also generates results for these tasks (e.g. classification, segmentation).

2 PCA vs Autoencoder

Linear PCA uses a linear mapping for dimensionality reduction, therefore it can be compared to an autoencoder that uses linear functions. Using a neural network, this translates into linear activation functions in the encoder and decoder layers. This kind of autoencoder approximates linear PCA as it reduces the dimensionality of the input using a linear mapping (similarly to linear PCA). Dimensionality reduction can result into latent feature representations, which can be used for classification. However, an autoencoder with linear activations cannot learn non-linear feature representations. Linear PCA is relatively fast and is more easily to interpret as it consists only of linear mappings.

An autoencoder is more flexible in the kind of mappings that can be learned. An autoencoder is a neural network which can use either linear or non-linear activation functions. By having non-linear activations, we can learn more expressive features by having a so-called deeper architecture. An autoencoder with multiple layers in the encoder and decoder parts combined with non-linear

activations can potentially learn a latent representation that is more complex, compared to linear PCA. Having more than 1 hidden layer in the encoder and decoder parts will not be useful if we use only linear activation functions because it will remain linear when combined together.

By using a neural network, we can also train on large-scale data, as we can use techniques such as stochastic gradient descent. PCA cannot not handle large-scale data very well, as it operates on all data at the same time. Non-linear feature representations may have a lower number of dimensions as the mapping from input to latent representation and vice versa can be theoretically any non-linear function. Linear feature representations are limited by linear mappings and therefore may not be able to compress the input data as much as non-linear mappings can.

3 Dataset definition

For the different learning tasks we chose to work with 5 classes (aeroplane, car, chair, dog, bird). For the segmentation network we also considered a dataset with only 2 classes (aeroplane, car) to compare the performance. The images from the VOC2009 dataset will be down-sampled to the size 224x224. This format will be used in all 3 neural networks. The validation set is split into a smaller validation set and a separate test set. This split is obtained using a fixed seed, so that we get the same sets every time we run the script.

4 Convolutional Autoencoder

In this section, the results of the convolutional autoencoder will be discussed.

4.1 Autoencoder architecture

Figure 1 shows the architecture of the convolutional autoencoder. It consists of an encoder and a decoder network. We have experimented with several configurations for the layers in each of these networks.

The encoder reduces the inputs through 3 convolutional layers combined with max-pooling layers. The latent representation has a size of 28x28x32. The size of this representation is about 16% of the size of the original input, which is a significant reduction.

The decoder reconstructs the original input from the latent representation by using up-sampling layers combined with convolutional layers.

We use relu activation functions in all layers, except the last one. This relu function helps to reduce the problem of vanishing gradients. The kernel size of the convolutional layers in both the encoder and the decoder networks is 3x3,

which is a typical size to be used.

The last layer uses a sigmoid activation function. As the input image has RGB values ranging from 0 to 1, we use a sigmoid function to ensure that the predicted output values are also within this range.

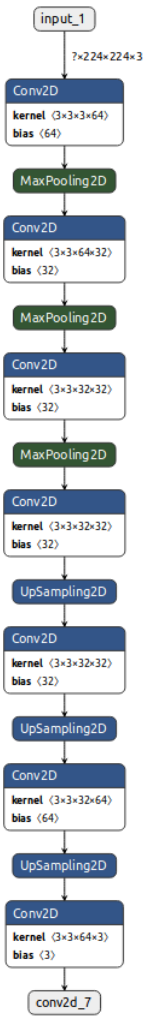
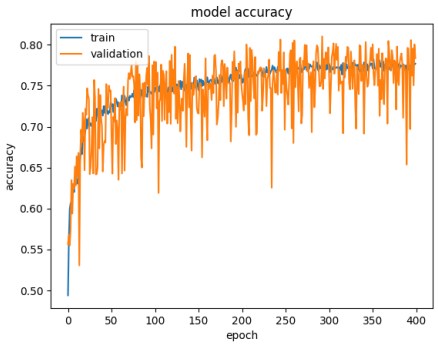


Figure 1: Architecture of the convolutional autoencoder.

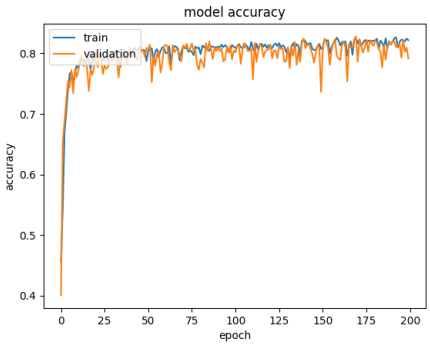
4.2 Loss functions

We have experimented with several loss functions: mean absolute error (MAE) and mean squared error (MSE). Figure 14 shows the progress of training and

validation accuracy when training with the MSE loss function. The MAE loss function results in a slightly better accuracy compared to the MSE function. However, when visually inspecting some test images, we observed that the reconstructed images from the MSE model retained slightly more details than that of the MAE model.

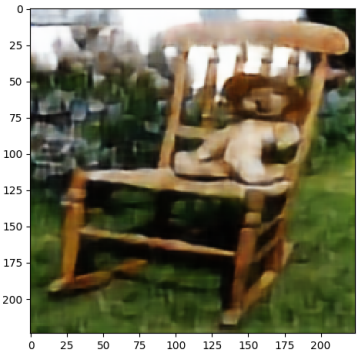


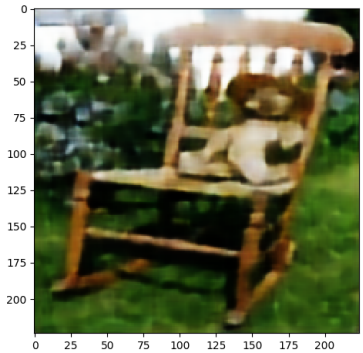
(a) Accuracy on training and validation data using the MSE model.



(b) Accuracy on training and validation data using the MAE model.

By using the MSE loss function, we can measure how close the reconstructed image is to the original image. The accuracy on training and validation data is about 80%. The original input is compressed by a factor of 6.25. This is a good trade-off between number of coding variables (which is 28x28x32) and reconstruction error.



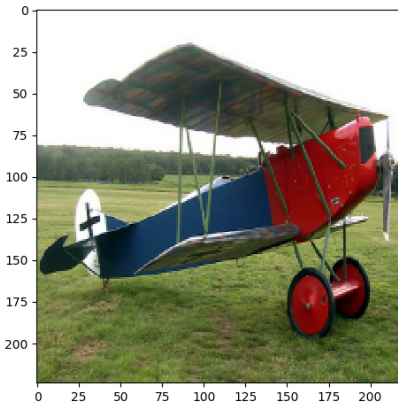


(a) Reconstruction using the MSE model. (b) Reconstruction using the MAE model.

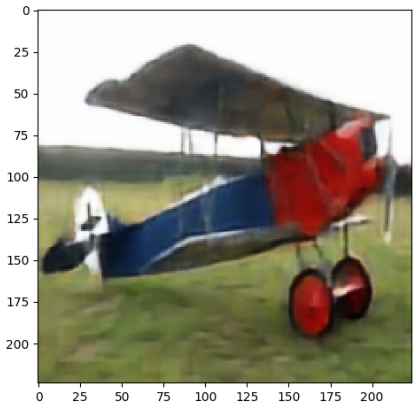
loss function	**model accuracy**	**model loss**
Mean Squared Error	0.7835	0.0037
Mean Absolute Error	0.7913	0.0362

4.3 Results

This section shows some results of the convolutional autoencoder. It compares original images and their reconstructions. The model that was used to generate this reconstructed images was trained with the MSE loss function.

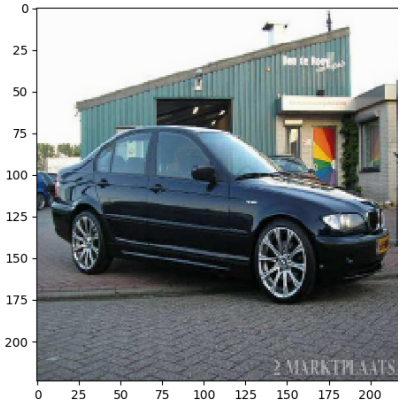


(a) Original image of an aeroplane

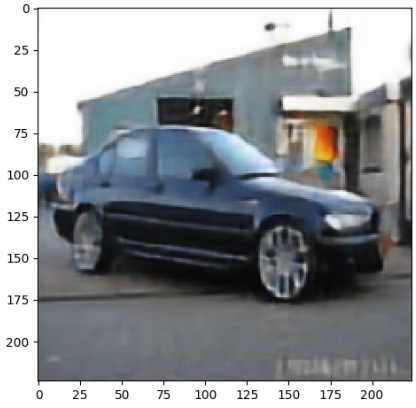


(b) Reconstructed image of an aeroplane

Figure 4: Original and reconstructed images of an aeroplane.

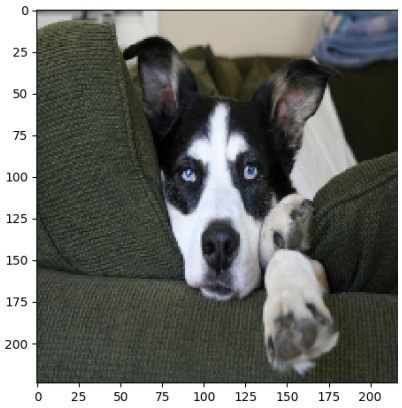


(a) Original image of a car

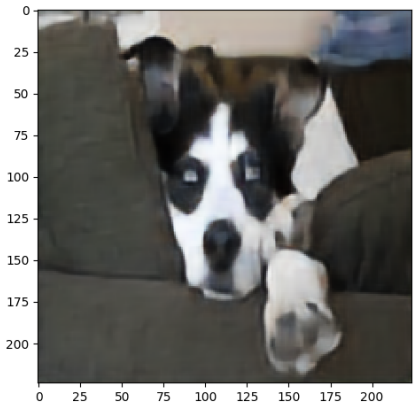


(b) Reconstructed image of a car

Figure 5: Original and reconstructed images of a car.

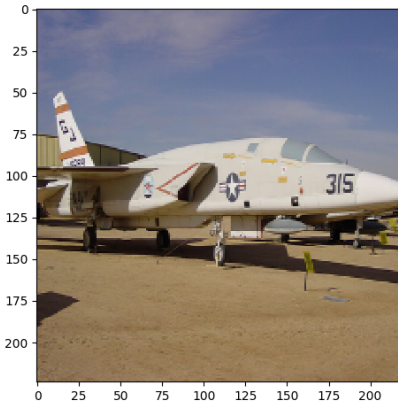


(a) Original image of a dog

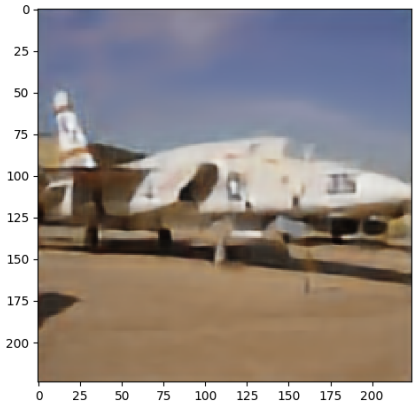


(b) Reconstructed image of a dog

Figure 6: Original and reconstructed images of a dog.

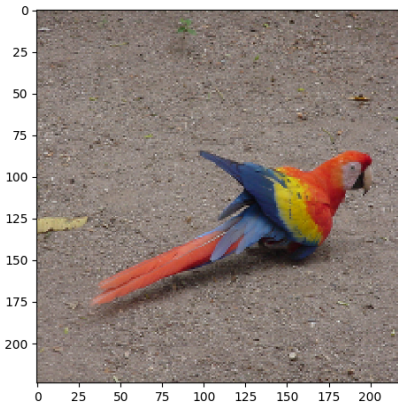


(a) Original image of an aeroplane

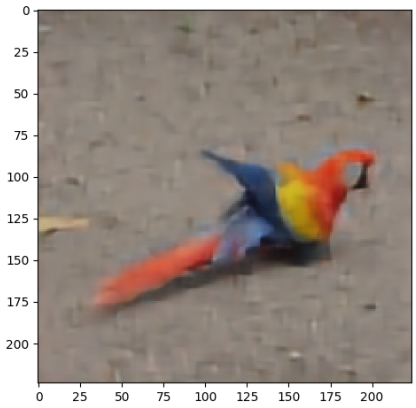


(b) Reconstructed image of an aeroplane

Figure 7: Original and reconstructed images of an aeroplane.

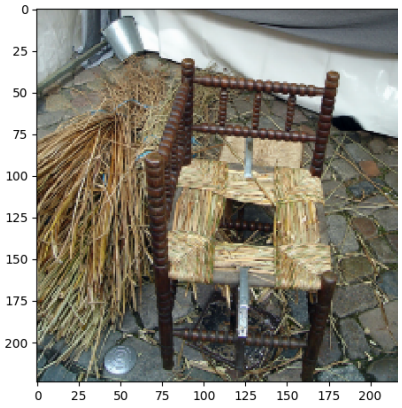


(a) Original image of a bird.

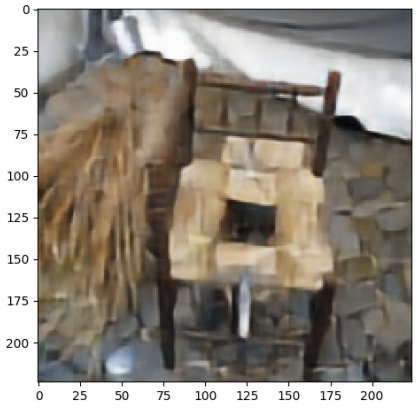


(b) Reconstructed image of a bird

Figure 8: Original and reconstructed images of a bird.

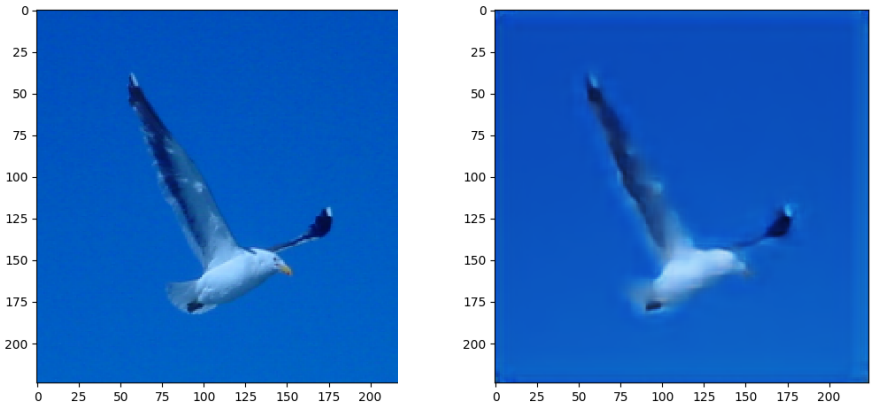


(a) Original image of a chair



(b) Reconstructed image of a chair

Figure 9: Original and reconstructed images of a chair.



(a) Original image of a bird (b) Reconstructed image of a bird

Figure 10: Original and reconstructed images of a bird.

5 Classification

5.1 Fine-tuning using encoder network

To perform classification on our dataset, we created a neural network using the encoder part of the autoencoder of section 4. We added a fine-tuning network on top of this base network by using fully connected layers. Figure 11 shows the architecture of our classification model.

As non-linearity function, we use a relu activation function in the fully connected layers. To reduce the risk of overfitting, we introduce several dropout layers. The final classification layer uses a sigmoid activation function. We use a sigmoid function as it does not define a probability distribution over the possible classes. This is important as a given image may have more than 1 label (e.g. bird and car in the same picture). Softmax activation is not appropriate as it does define a probability distribution over the classes. This would imply that whenever the probability of a car increases, then the probability of all other classes decreases as a necessary consequence. However, when a car and a dog are in the same picture, we want both the probabilities for a dog and a car to be high.

The fully connected layers shrink gradually in size, from 512 to 128. This allows the network to learn a mapping from images to classes in a more hierarchical manner. A single fully connected layer would introduce a potentially too complex mapping from features to classes, which would be prone to overfitting.

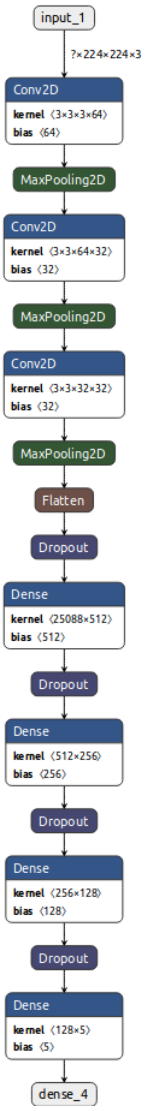


Figure 11: Architecture of the classification model.

We have experimented with two loss functions: binary and categorical crossentropy. The crossentropy loss function is also called the negative log likelyhood. By minimizing the crossentropy, we can maximize the log likelyhood. Using categorical crossentropy, we can use an arbitrary number of classes. With binary crossentropy, we use only 2 classes. The categorical crossentropy resulted in a lower training and validation classification accuracy, compared to binary crossentropy. However, when evaluated on a test set, the model trained using the categorical crossentropy performs better than the one trained with binary

crossentropy.

Figure 12 shows the evolution of the classification accuracy using binary crossentropy. And figure 13 shows the classification accuracy evolution using categorical crossentropy and training from scratch. While the model trained with a freezed encoding network (meaning pre-trained as in section 4, the accuracy on training and validation set is relatively high. However, it does not perform well on a separate test set. The accuracy on this test set was only 0.53.

Training the model from scratch and with categorical crossentropy results in a reduced training and validation accuracy. However, it does perform relatively better on the same test set, where it obtains a test accuracy of about 0.63. Thus, training from scratch may help optimizing the parameters in the encoding network more towards the classification task.

loss function	**mode**	**model accuracy**	**test accuracy**
Binary crossentropy	freezed	0.79	0.53
Categorical crossentropy	from scratch	0.61	0.63

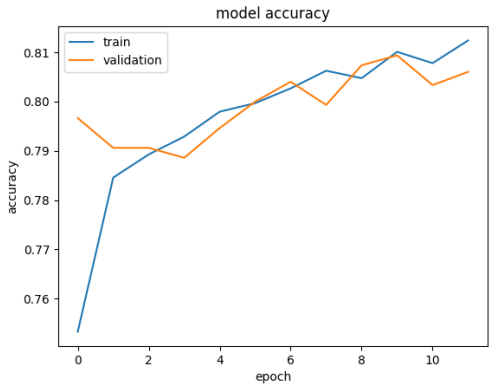


Figure 12: Progress of classification accuracy on training and validation set using binary crossentropy.

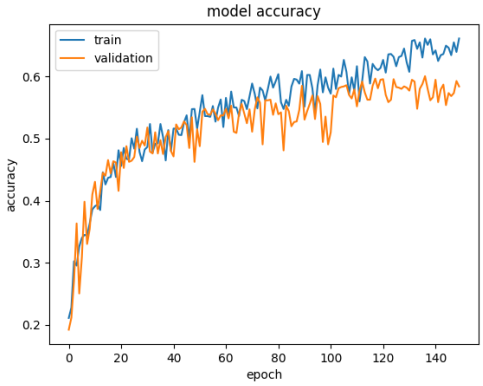


Figure 13: Progress of classification accuracy on training and validation set using categorical crossentropy and training from scratch.

5.1.1 Why do we maximize the log likelyhood?

The training set only represents a sample of the true population of all images (within the 5 defined classes) and their labels. We want to learn a mapping from these images to the labels in such a way that it is closest to the true mapping of the population. Maximizing the log likelyhood gives us a way to estimate the parameters in such a way that the learned mapping is close to the unknown population mapping.

5.2 Fine-tuning using pre-trained VGGNET-16

As an alternative approach, we used the VGG-16 architecture with pre-trained weights. We removed the classification layer of VGG-16 and replaced them with our own fine-tuning network.

The performance of this alternative classification network is similar to the one defined in section 5.1. The performance on the test set was, however, significantly worse.

5.3 Training from scratch

As a last option, we decided to train the full network defined in 5.1 from scratch. This means that the encoder layers will be trained in combination with the fully connected classification layers. However, this also resulted in a similar performance on the training and validation set.

5.4 Data augmentation

Since the dataset is relatively limited in terms of the number of images per class, we use a technique called data augmentation. This creates variations of the images present in the training and validation sets by shifting, rotating, stretching and flipping the images. This way we provide the model with more examples of the different classes and in doing so, we expect the model to be more resilient to changes in orientation of the objects.

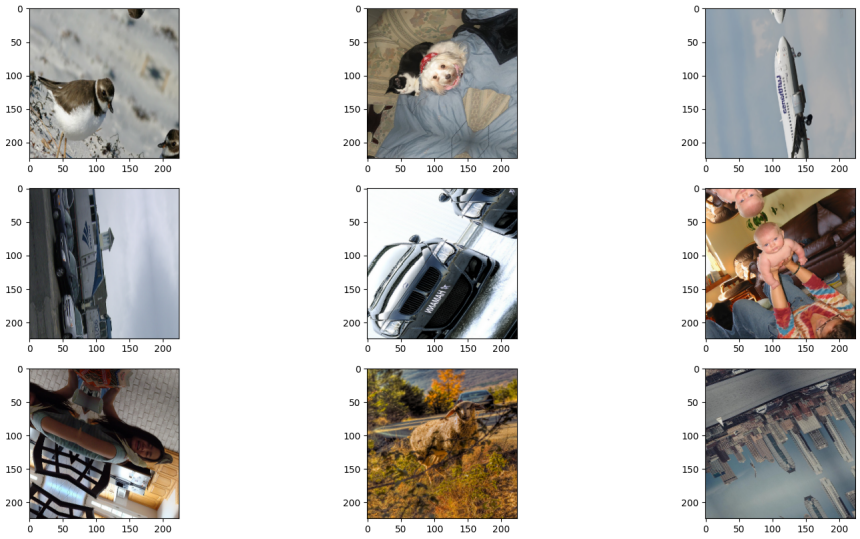


Figure 14: Training images generated by data augmentation.

5.5 Results

Figure 15 and figure 16 show label predictions for test images using a model trained using binary crossentropy and a model trained using categorical crossentropy. As can be seen from the figures, the model with categorical crossentropy classifies the test images correctly, while the model with binary crossentropy has more difficulty classifying them (e.g. it misclassifies the birds in figure 15).

(a) Predictions (binary crossentropy) (b) Predictions (categorical crossentropy)

Figure 15: Shows the predictions of the models trained with binary crossentropy (left) and categorical crossentropy (right).

(a) Predictions (binary crossentropy) (b) Predictions (categorical crossentropy)

Figure 16: Shows the predictions of the models trained with binary crossentropy (left) and categorical crossentropy (right).

6 Segmentation

To create a segmentation network, we used several typical architectures, such as UNet, Fully Convolutional Network (FCN) and Semantic Segmentation Network (SemNet).

6.1 Data preparation

The provided segmentation masks of the Pascal VOC dataset were given in RGB format (i.e. a colormap was applied). However, we need the pixels in every mask image to reflect the class to which it belongs. To achieve this, we used the source code from `https://github.com/tensorflow/models/blob/master/research/deeplab/datasets/download_and_convert_voc2012.sh`. This file removes the colormap in the ground truth annotations and results into segmentation masks where every pixel is labeled with a number between 0 and 20.

(a) Segmentation mask with a colormap (b) Segmentation map without a colormap

6.2 Experiments with different architectures

6.2.1 UNet architecture

The first architecture we have used to do segmentation is the UNet architecture. This architecture gets it name from the U-shape, as can be seen in figure 18.

2

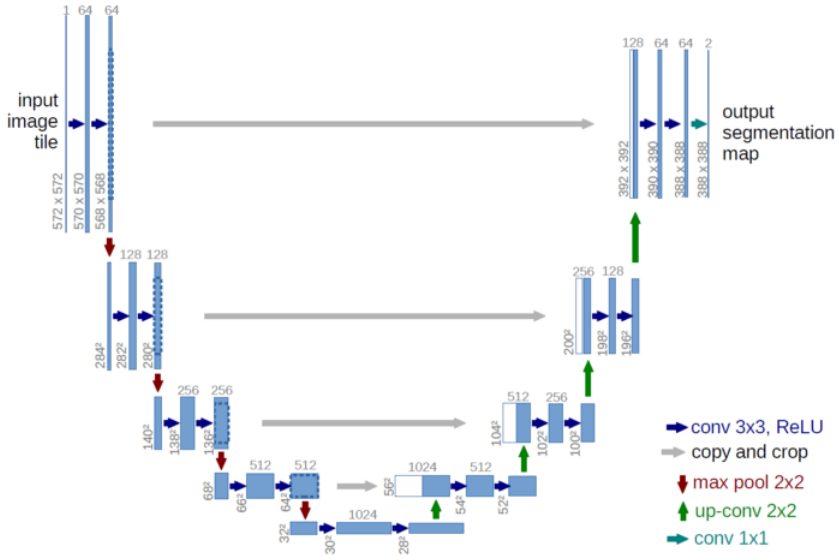


Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows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

Figure 18: Representation of the Unet architecture.

This architecture uses skip connections which propagates information about the encoding process to the decoding process of the fully convolutional neural network. In figure 18 this corresponds to the horizontal gray arrows connecting layers from the encoder to layers in the decoder.
We use the dice coefficient as a loss function. This measures the overlap between the ground truth mask and the predicted mask.

6.2.2 UNet Results

After having trained several models with the UNet architecture, we could not obtain a good result in terms of segmentation masks. The predicted masks were often very inaccurate or the background class was predicted for the majority of the pixels. In addition, the UNet architecture is mainly used for segmentation of medical images, whereas our data set consists of general images.

6.2.3 SegNet Architecture

An alternative architecture is the SegNet architecture. This architecture also consists of a encoder and decoder part. This architecture is given by figure 19.

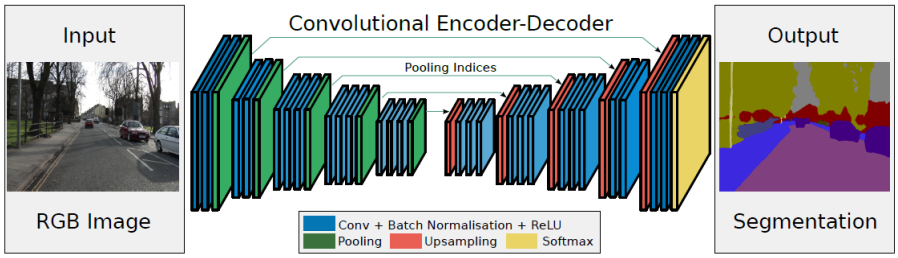
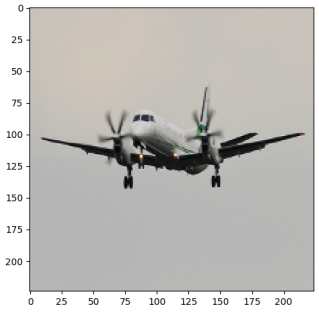


Figure 19: Representation of the SegNet architecture.

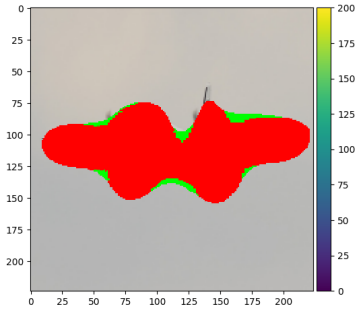
We made several adjustments to this architecture. We do not use skip connections in this architecture. In addition, we use a sigmoid activation function, instead of a softmax function, in the final classification layer. For the internal layers, we use relu activations (to reduce the vanishing gradient problem). After each convolutional layer, we have a batch normalization layer. These batch normalizations are used to speedup the learning process.

6.2.4 SegNet results

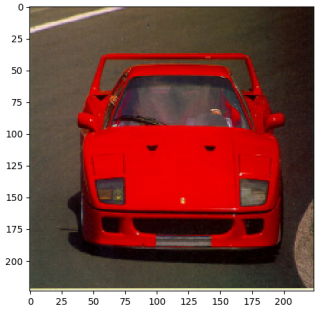
This architecture also did not result into good segmentation results when using 5 different classes. However, when we use only 2 different classes (aeroplane and car), we do get reasonable segmentation results.
The following figures show some good segmentation results using only two classes. The colors indicate the corresponding class, with red = aeroplane and green = car.



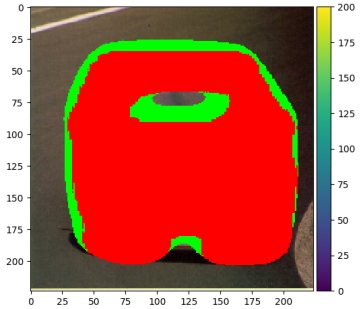
(a) Original image of an aeroplane



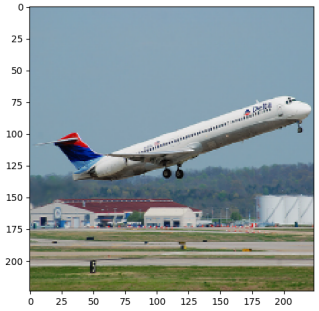
(b) Segmented image of an aeroplane



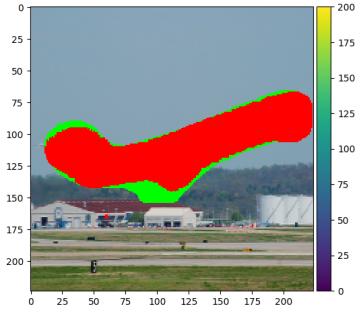
(a) Original image of a car



(b) Segmented image of a car



(a) Original image of an aeroplane



(b) Segmented image of an aeroplane

6.3 Discussion of the results

The two previous architectures did not result into good segmentation results. We discuss some of the potential reasons for these results.

6.3.1 Small dataset

The segmentation dataset is significantly smaller than the classification dataset. Every class has a limited number of examples and the intra-class variance between the different images in every class is quite large (i.e. a lot of different perspectives of the same object).

6.3.2 Bias of the dice coefficient

$$\frac{2|X \cap Y|}{|X| + |Y|}$$

The dice coefficient measures the ratio of the intersection and the union of inputs and outputs. However, our representation for a ground truth segmentation mask is a numpy array of shape `image_width` x `image_height` x `num_classes`. Figure

23 shows this representation. As can be seen from this figure, it is a very sparse representation (i.e. containing a lot of zeros).

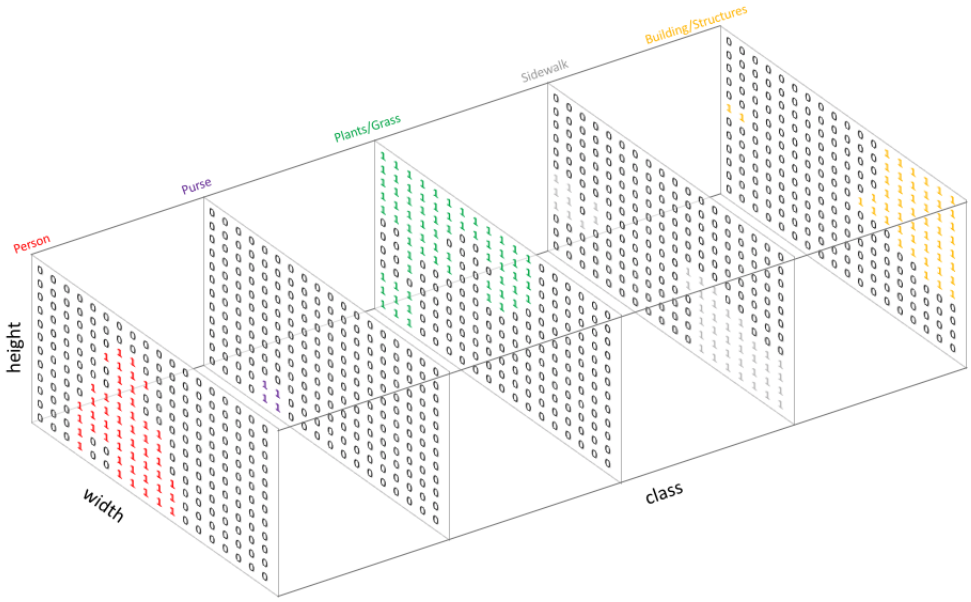
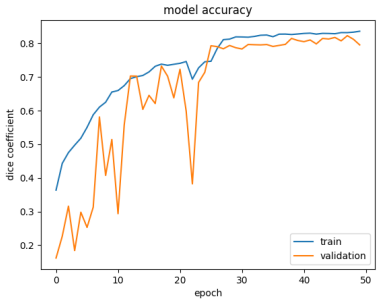
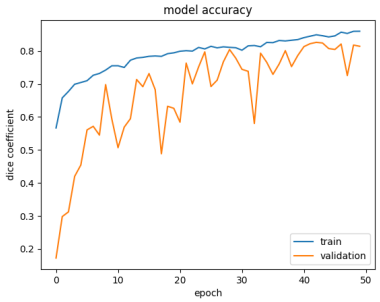


Figure 23: Segmentation mask representation as one-hot encoded shape.

Training the SegNet resulted often in a relatively high dice coefficient (around 0.7-.075). This can be seen in figure 24a and figure 24b. However, a high dice coefficient does not necessarily mean that the segmentation performance is equally good. Because of the sparsity of the representation, the network often focuses too much on the background class (which is coded as 0), as this is the majority class in almost every mask.



(a) Evolution of dice coefficient (training with 5 different classes)



(b) Evolution of dice coefficient (training with only 2 different classes)

6.3.3 Potential solution for the bias

We argue that introducing an argmax layer in the network (right after the final classification layer) will lead to better segmentation results. This argmax layer will reduce the representation of figure 23 to a shape of dimension `image_width` x `image_height`. This new representation can be seen in figure 25. The sparsity will be reduced significantly, which allows the network to focus more on other classes than the background class. However, the argmax layer cannot be used in combination with backpropagation for training the network, as argmax is not differentiable. There exist workarounds/approximations, but all are mostly defined in tensorflow. We have created our project entirely in Keras, which makes these workarounds very difficult to integrate.

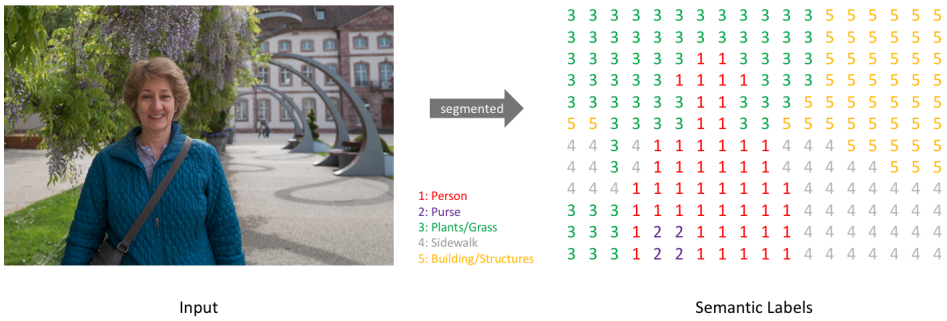


Figure 25: Reduced segmentation mask prediction.

6.4 Loss functions

Several loss functions can be used for image segmentation tasks. The following list shows a few popular loss functions:

- Dice coefficient
- Pixel-wise cross-entropy
- Focal Loss

The dice coefficient is the loss function we have used in our networks. This coefficient relates directly to what a good image segmentation should look like. It calculates the intersection over union of the ground truth mask and the predicted mask. When this coefficient is close to 1, we expect a relatively good segmentation. However, as we have seen in the previous sections, it is highly affected by the sparsity of the solution.

The pixel-wise cross-entropy tries to optimize every pixel independently, whereas the dice coefficient looks more at the overall segmentation result.

The focal loss function should be used when there is a high class imbalance in the dataset. However, our dataset did not show any sign of a high class imbalance.

7 Conclusion

In this project, we learned new ways to perform dimensionality reduction through the use of autoencoders. They can build feature representations, which can be used in a classification task. Convolutional layers play an important role in reducing the dimensionality of the input, while keeping the number of parameters relatively low. In addition, we learned some important differences and similarities between linear PCA and autoencoders.

In the part 2, we have learned how to fine-tune an existing network for use with classification. The loss function that is used during training is relatively important to obtain better results. We could compare a fine-tuned network with a network trained from scratch. And we experienced that training from scratch could sometimes be beneficial in terms of optimizing the parameters towards the given objective or task (in this case classification).

In the last part, we have been introduced to semantic segmentation. We have learned new representations and trade-offs when creating a segmentation network. Training such a network and obtaining good results has not been trivial and it gave us better insights into what did work well (and why) and what did not work well.