Project: In the name of deep learning

Computer Vision (H02A5a)

April 15, 2019

1 Introduction

In this project, we enter the world of computer vision as it currently is, with deep learning taking the cake. First, we introduce the data you will be using and the possibilities/instruments you have to elaborate each of the following sections. The first goal will be to transition from linear PCA (Assignment 3) to non-linear (convolutional) auto-encoders. The second goal will be to use the learned, abstract latent space directly for classification, and train your first *end-to-end* neural network. The final goal will be to translate image classification into a pixel-wise classification task for segmentation purposes.
We have designed this project such that you should be able to run all computations on your laptop (no need for special hardware). We want you to submit your final report together with your Python code (you can use Jupyter notebook if you like) as one single ZIP-file **at least two days before your *exam***. During your exam, you have the opportunity to present (in any format you like) your work in 15 min, followed by 15 min of discussion.

1.1 Data

You will be using the PASCAL VOC-2009 dataset [Eve+10] for this project. This dataset consists of colour images of various scenes with different object classes (e.g. animal: *bird, cat, ...*; vehicle: *aeroplane, bicycle, ...*), totalling 20 classes (Figure 1). The dataset has been used as a benchmark for classification, detection and segmentation challenges and for comparing results of individual research (more details can be found here). Start by downloading and unpacking the training/validation dataset (900 MB TAR-file) from this link onto your local computer. Make yourself a bit acquainted with the dataset by looking at the description and plotting a few images.
Next, you will build a training and validation set that you will use for this project. You can start from the example Python script, provided together with the project description, to **filter the PASCAL VOC-2009 images** to optionally: 1) use less images, 2) resample (e.g. downsample to fixed size) and 3) make the classification and segmentation tasks simpler with fewer classes (e.g. binary=two-class). This will result in lower complexity and hopefully faster training on your laptop. Have a look at the dataset description to decide which classes you want to keep (e.g. [*aeroplane*, *car*, *chair*, *dog*, *bird*] will yield 1.489 training and 1.470 validation images). We do not expect excellent performance so don't worry if your hardware

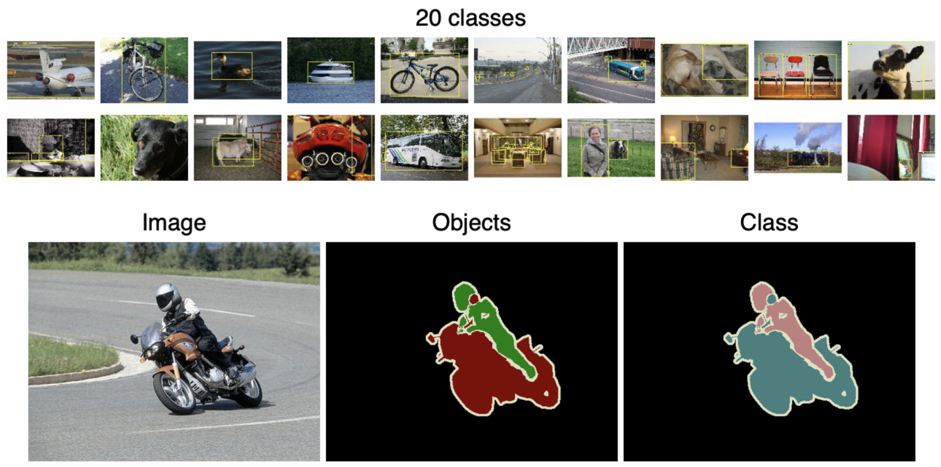

Figure 1: PASCAL VOC-2009 dataset [Eve+10] (source).

does not allow training big datasets. **We expect that you will use the same training and validation set for all tasks in this project, and that you clearly define it in the report.** Please explain clearly if you deviate from this approach (in this case, you should first describe your dataset for each task).

1.2 Deep learning

We build further on Assignment 4 (see Sect. 1.1) and advise you to use Keras [Cho+15] as the library for building your (deep) neural networks. As for deep learning, there are many aspects to take care of but most of them are out of the scope for this course. Therefore, unless explicitly requested, we do not expect, although we don't discourage, you to explore different optimizers (e.g. SGD, ADAM), regularization mechanisms (e.g. weight decay, L2, data augmentation), non-linearities (e.g. ReLU, sigmoid), architectures (e.g. U-Net, DeepMedic, ResNet), etc. **The main goal of this project is to understand how deep learning can be used in computer vision applications, and not to explore all the intricacies of deep learning.**

2 Auto-encoders

Go back to Assignment 3 and familiarize yourself (again) with PCA. Then go ahead with this section, knowing that the ideas remain but the encoding/decoding can be made more complex.

2.1 PCA vs auto-encoder

An auto-encoder is typically considered and depicted as an encoder-decoder structure shown in Figure 2. As a first task, consider an auto-encoder that mimics linear PCA. Explain how you can do this and what the differences are. In terms of a neural network the auto-encoder will contain an input layer, one hidden layer and an output layer. It is optional to practically experiment, but we ask you to elaborate *theoretically* on the expected differences in results when PCA is framed into a neural network. We do not expect any mathematical proves, but rather want you to show us your understandings.

2.2 Non-linear and convolutional

When working with images, it has been shown that convolutional neural networks (CNNs) have superior performance compared to fully-connected variants (hint: the latter is what you should have considered for mimicking linear PCA). You can find a nice introduction to CNNs here.
Your task is to implement a self-made CNN to build a non-linear and convolutional auto-encoder, which is deeper than the PCA-mimicking auto-encoder. Think of what loss function you will need to use (a reconstruction loss). As an extra, you may visualize your latent space (using techniques to visualize high-dimensional data such as t-SNE[MH08]) and/or try to improve your existing representations (you are free to test ways to improve any of the above representations or come up with new ones). When you feel your resulting auto-encoder has a good trade-off between the number of latent variables in the lowest layer ("Code" layer in Figure 2) and the reconstruction error, you can move on to the next section. Don't go too far with these experiments. We advise you to use the standard architecture and training parameters available in Keras (e.g. optimizer, regularization, non-linearity) and only experiment a bit with the number of coding variables. Report on your strategy: your loss function, the

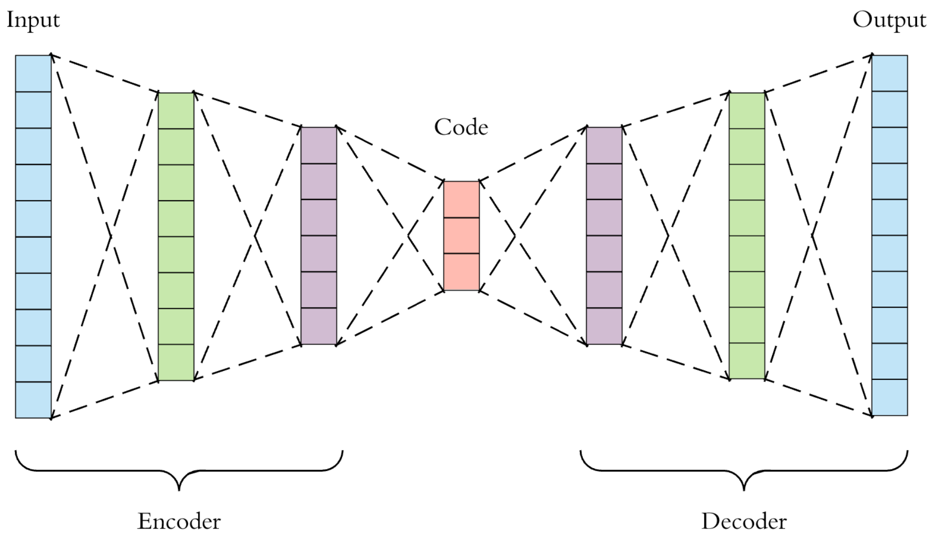

Figure 2: Typical visualization of auto-encoders or encoder-decoder networks (source).

architecture, how you trained the network, the number of coding variables, reconstruction results, etc.

3 Classification

The auto-encoder which you have built now represents an image in a number of coding variables (maybe still spatially distributed as a *feature map*), which is (hopefully) relatively small compared to the number of pixels. The question remains whether these variables contain semantic information, hence can be used to classify images into distinct classes.

3.1 Build an object classification network

Take the coding variables of your auto-encoder and train a classifier. You can consider different classifiers, but we advise you to stick to simple logistic regression. You should train two networks. First, simply take the encoding part of your auto-encoder, *freeze* these layers and add a fully-connected layer at the end (a technique called fine-tuning that you already applied in Assignment 4). Think about the non-linearity you will have to use in your final layer and which loss function to use (hint: typically you want to maximize the log likelihood). Second, you will use the same network architecture but train all parameters from scratch (aka randomly initialize each parameter at start of training; hint: this is a simple flag in Keras). Describe what happens to the classification performance when comparing both networks. You can further experiment with larger and more complex architectures, optionally taking pre-trained networks from the Keras library and (again) fine-tuning them to this classification problem. Practical note on computation time: please make sure to filter the dataset appropriately as described in Sect.1.1.

3.2 Things to reflect on

Further report on why your choice of non-linearity, why to maximize the log-likelihood and why your choice of loss function. You can optionally experiment by going back to your auto-encoder and simultaneously train it to reconstruct and classify. Explain how you do this and what the results are. It might be interesting to visualize the learned coding space and compare this with what you had before. You are also free to do other things and report on those.

4 Segmentation

In this final section, you will have to reformulate your CNN for image classification into a CNN for pixel-wise classification. As such, each pixel gets a label of belonging to a certain object or not and the result is a segmented image. Optionally, for this purpose **you can group all available objects in an image into one single *foreground* class** and do binary image segmentation.

4.1 Build a binary segmentation network

Your task is to make a segmentation CNN to extract the foreground from an image. You can think off the CNN that you have built for classification as an operation that takes a NxN input and processes this into a 1x1 output. Due to the convolutional and layer-wise processing, the latent variables in deeper layers get to see more context and the learned *features* are/can be more complex (due to the successive non-linearities). In the final layer, the resulting features are classified linearly using logistic regression. Now, imagine that you are classifying the central pixel of the image (a 1x1 output) based on its surroundings (a NxN input). In order to label every pixel in the image, we can reformulate one NxN image as being N^2 pixel-wise examples, each having a different pixel in the center that gets classified based on a surrounding window [Cir+12]. Later, more parallel implementations closely follow the encoder-decoder structure (Figure 2) by using convolutional layers to encode and upsampling/deconvolutional layers to decode directly into pixel-wise predictions [LSD15]. The loss function which you should have used for the classification task is then applied to and averaged across *all* the pixels. From this, one of the most popular networks for image segmentation was derived by adding so-called *skip connections* from encoding layers to their corresponding decoding layers at the same scale [RFB15] to enhance learning in earlier layers and improve the reconstruction of feature maps at a higher resolution from feature maps at a lower resolution.
You main task is simple: build a segmentation CNN of your choice (with the loss function you applied before) and visualize the training and validation learning curves. As you will notice, now the loss or other standard metrics like accuracy are less meaningful. Find better metrics and make sure to visualize your segmented images in a nice way.

4.2 Things to reflect on

It is truly yours to implement and report on the results of any segmentation strategy you like. We want you to think on different ways to evaluate a segmentation. For example, the *Dice score* is an important *image overlap* metric, especially in (medical) image segmentation tasks. Which other metrics are commonly used? Related to this, are there other loss functions which you could implement to directly optimize the Dice score? What is an advantage compared to the loss function you used before? Again, you are free to experiment even further.

5 Submission

In the report it should be clear how you simplified the PASCAL VOC-2009 dataset into your training/validation set. For every step (i.e. auto-encoder, classification, segmentation) elaborate on your approach. You are free to make it your own project, but make sure to show your understandings and answer the questions posed throughout this text. Please submit your final report together with your Python code (you can use Jupyter notebook if you like) as one single ZIP-file **at least two days before your *exam***. During your exam, you have the opportunity to present (in any format you like) your work in 15 min, followed by 15 min of discussion.

References

[MH08] Laurens van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE". In: *Journal of machine learning research* 9.Nov (2008), pp. 2579–2605.

[Eve+10] Mark Everingham et al. "The pascal visual object classes (voc) challenge". In: *International journal of computer vision* 88.2 (2010), pp. 303–338.

[Cir+12] Dc Ciresan et al. "Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images". In: *Nips* (2012), pp. 1–9. URL: https://papers.nips.cc/paper/4741-deep-neural-networks-segment-neuronal-membranes-in-electron-microscopy-images.pdf.

[Cho+15] François Chollet et al. *Keras*. 2015.

[LSD15] J Long, E Shelhamer, and T Darrell. "Fully convolutional networks for semantic segmentation". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 3431–3440. ISSN: 1063-6919. DOI: 10.1109/CVPR.2015.7298965. arXiv: 1411.4038.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *Miccai* (2015), pp. 234–241. ISSN: 16113349. DOI: 10.1007/978-3-319-24574-4_28. arXiv: 1505.04597.