Group assignment:
Crafting and learning features

Computer Vision (H02A5a)

The goal of this assignment is to explore more advanced techniques for constructing features that better describe objects of interest and perform a few tasks using these features. This assignment will be delivered in *groups of 4, randomly assigned by your TA's*. Start from the provided template notebook on Google Colab. The final notebook should be exported (as .ipynb) and uploaded to toledo (one per group) by **Tuesday 7 April 23:59**. Make sure that it is well documented and runs from start-to-end (Runtime → Run all) without errors.

1 Overview

In this assignment you are a group of computer vision experts that have been invited to ECCV 2020 to do a tutorial about "Feature representations, then and now". To prepare the tutorial you are asked to create a Google Colab notebook that can be easily executed by the participants. Your target audience is: (master) students who want to get a first hands-on introduction to the techniques that you apply.
In this assignment you will build a small dataset, construct and learn features and use the features for face recognition and finding similar faces. We discriminate two types of features, handcrafted features and features learned from data. Handcrafted features are properties that can be algorithmically extracted from the image (similar to previous assignment). In this assignment you will experiment with Histogram of Oriented Gradients (HOG) or Scale Invariant Feature Transform (SIFT) feature descriptors. The features learned from data are features that are learned by finding patterns in multiple images. You will explore Principal Component Analysis (PCA) in the context of face detection, so-called eigenfaces, and leverage pre-trained weights from a Convolutional Neural Network (CNN) using the Keras library.

More practical, this assignment is broken down into four main parts:

1. construct your own dataset of faces (Sect. 3);
2. build some feature representations using handcrafted and non-handcrafted techniques (Sect. 4);
3. use and compare the feature representations in context of classification of faces and assessing similarity between faces (Sect. 5);
4. discussion (sect. 6).

2 Preparation: learning to use a machine learning library (TensorFlow/Keras)

In this assignment you will be using the TensorFlow/Keras [Ten; Cho+15] library for building a deep learning network. Keras is a high-level neural networks API written in Python. Since the TensorFlow 2.0 release, Keras has been integrated more tightly into TensorFlow. While there are many good libraries available for building deep learning models, we recommend Keras as it is easy to use and still offers flexibility to make more complicated models at a later stage. Familiarize yourself with TensorFlow/Keras using for instance this tutorial, created by François Chollet.

3 Build a faces dataset

Use the provided template notebook (see first paragraph of assignment) to construct a dataset. The notebook contains a cell that downloads the VGG Face Dataset [PVZ15]. Search across this dataset for your favorite celebrities to build a small dataset that you will use throughout assignment[1]:

- 20 training and 10 testing images of person A
- 20 training and 10 testing images of person B
- 10 testing images of person C, who looks similar to person A
- 10 testing images of person D, who looks similar to person B

Make sure that persons C and D share some characteristics with person A and B, respectively. You can do this based on your own judgement, it can be quite broad e.g. both male, same hair type, both wearing glasses, etc. Use OpenCV to detect the faces in the images and extract the faces (some of this code is already provided in the template notebook). Make sure that you have some interesting differences between the faces of person A and B and within each group. Having variability will make it easier to evaluate the representativeness of the feature representations that you will build in the next section. Make sure to plot a nice overview of all the faces used, use labels/titles and explain why you chose these individuals.

4 Build feature representations

Once you know what defines an object or what allows you to differentiate between two objects, it becomes much easier to detect an object in an image. Learning a robust (same features are extracted from the same object in different conditions) and discriminative (different image objects can be easily separated from each other in feature space) feature representation is key to perform automated object detection effectively. Two big classes of feature representation methods can be identified:

[1] We realize that this is a simplified setup but good enough to reach the goals of this assignment.

- Handcrafted features (Sect. 4.1)
- Features learned from data (Sect. 4.2)

You will try building both types of features on the faces dataset that you have built in Sect. 3. Remember to discuss the design choices and limitations of each of the feature representations that you build.

4.1 Handcrafted features

Handcrafted feature descriptors are extracted from the image itself, using a specific deterministic algorithm. The algorithms used are determined by the feature engineer and might differ from task to task. To assess if a handcrafted feature is discriminative we compute it for a few images, we then compare the match score (e.g. euclidean distance between the feature representations) and observe if this matches our intuition/domain knowledge. Handcrafted features were commonly used in traditional machine learning approaches and we still see them today in complicated machine learning tasks e.g. doing machine learning on a manifold mesh.

First, you will compute features from pixel intensities in order to learn a representation of local structures present in an image. Ideally, such a (local) feature descriptor will hold the robust and discriminative property and will additionally be invariant to e.g. illumination, pose, scale, to allow you to detect objects of interest in virtually any given image using that descriptor. You should **build one handcrafted feature descriptor** from one of these two state-of-the-art handcrafted image feature descriptors:

- Histogram of Oriented Gradients (more details can be found here)
- Scale Invariant Feature Transform (see course book SB 4.1.2)

Reflect carefully about how local these descriptors should be for the object that you want to detect and how to make your descriptor behave well in different circumstances (reflect on which variations can make your images/faces look different), how does it compare to your previous grabbing task in the individual assignment? Do you need specific pre-processing steps before computing these feature descriptors on your images (which ones and why)? Second, detecting an object of interest in a new image involves matching the local descriptors to the image, this allows you to detect which regions have structures that correspond well to the ones present in your feature descriptor. As a distance metric you could for instance calculate the euclidean distance between the feature outputs and the feature outputs of your object(s) of interest at every position in the image.

4.2 Learning features from data

An alternative approach to handcrafting features that capture desired object properties is to learn these features from training data. In this case, the algorithm looks for patterns in the data that are informative. Be careful, you can only use the training images for this step!

4.2.1 Unsupervised learning: PCA

First, you will apply an unsupervised learning approach called Principal Components Analysis (PCA) to find the different components of variation present in your faces training set. Think about this problem mathematically:

- How to convert my image (you can work in color if you like) dataset to a 2D matrix?
- Can you exploit the dimensionality of this data matrix to make your computations more effective?
- Mean subtraction or not?
- Shall we use eigenvalue or singular value decomposition?
- How many non-zero eigenvalues/singular values should we have?

Visualize the reconstructions of one face using gradually more eigenfaces. Attempt to choose an *optimal* number p of principal components such that the dimensionality of the feature space is reduced but still informative (e.g. based on the average reconstruction loss of your training images). Reflect on the design choices that you made and how this might influence the performance of the features that you have learned. Did you need special pre-processing to make this work (which one and why)?

Visualise the faces in the principal component space. Use the eigenfaces learned on your training images, project the test images onto the same p-components and visualize the new faces on a face-feature plot of the first two principal components like in Fig. 1.

4.2.2 Transfer learning

Deep learning is everywhere these days, in this assignment you will start by applying it in its simplest form. Your goal is not to train a neural network from scratch (don't worry you can show these scratching skills in the final project) but rather, to leverage the *face-features* from a pre-trained Convolutional Neural Network. There are plenty of papers on Face Recognition and a few interesting open sources implementations are available. We recommend that you use the keras-vggface package, spend some time to understand how this model has been built (VGG Face Descriptor) and make sure that you can run the network on the images from your faces dataset (what pre-processing do you need?).

Visualize your feature representations to gain more insight. To do this, use the high dimensional data visualisation tool t-SNE[2] [MH08]), an example is shown in figure 2.

5 Exploit feature representations

Now it's time to use the three feature representations that you have constructed in Sect. 4.1, 4.2.1 and 4.2.2 and start to make inference on unseen images. In this step, you will use the

[2]Scikit-learn has an implementation.

test images from the faces dataset that you built previously. Don't forget to apply the same pre-processing pipeline for each feature representation as you did before! Your results will evaluate each feature representation for both classification and identification. Feel free to use your favorite classification algorithm and distance metric, you can use existing functions from open source Python libraries (e.g. scikit-learn) or implement them yourself. Make sure that you discuss each method and describe why it is appropriate for each task.

5.1 Classification

In classification it is your goal to assign a class to an unseen data example. In this exercise you will train a binary classifier that can recognise person A (class 0) and person B (class 1). Discuss how you build this classifier for each of your feature representations. Then run predictions on your test set and compute the mean accuracy. How well does the model predict on test images of person C and D? Can you explain why and how the three feature representations behave differently?

5.2 Identification

In an identification setup the goal is to compute similarity scores between pairs of data examples and use them to identify new images. In this exercise you will compute the feature representation of every image and create a data space using your training data, give each of the data points a label based on the person they contain. Next try to identify the test images using k-means clustering. What value did you use for k? Was the identification of person A and B successful for all descriptors? How did images of person C and D get labeled?

5.3 Impress your TA's

Impress your TA's by doing any of the following (choose one or come up with your own trick; explain in your notebook why and how you do so):

- Pick one feature representation and try to improve your classification/identification results (explain in your notebook why and how you do so).
- Modify/add test images to understand what *breaks* your classifier (e.g. apply additional variations to your test images such as different lighting, rotations, partial obstruction of the faces or person wearing glasses etc), explain what is happening and why it is happening.

6 Discussion

Summarize what you have learned in this assignment. Discuss qualitative differences between the different feature representations that you constructed on your (small) faces training set. Why would one work better than the other? What would you do better/more if you would have plenty of time to spend on this step? Discuss the results on the test images for the two tasks. Imagine that there is a company that wants to purchase your face detection prototype system to install in their offices as a means of authenticating employees. Would it be easy to fool this system? What would you advise the company? What are the current limitations of your prototype and what would you do next to make your system better?

7 Submission

Export your Google Colab notebook as a .ipynb file and submit it (one per group) to Toledo by **Tuesday 7 April 23:59**. Make sure that your notebook is **self-contained** and **running smoothly**. Test this out yourselves first, from someone else's Google account! Walk us through all steps of your code. Treat your notebook as a tutorial for students who need to get a first hands-on introduction to the techniques that you apply. Provide strong arguments for the design choices that you made and ensure that no additional actions are required for the TA's to execute all code inside it. Make use of the *Group assignment* forum/discussion board on Toledo if you have any questions.

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.

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

[PVZ15] Omkar M. Parkhi, Andrea Vedaldi, and Andrew Zisserman. "Deep Face Recognition". In: Section 3 (2015), pp. 41.1–41.12. DOI: 10.5244/c.29.41.

[Ten] Google Tensorflow. *Tensorflow*. URL: www.tensorflow.org (visited on 05/22/2016).

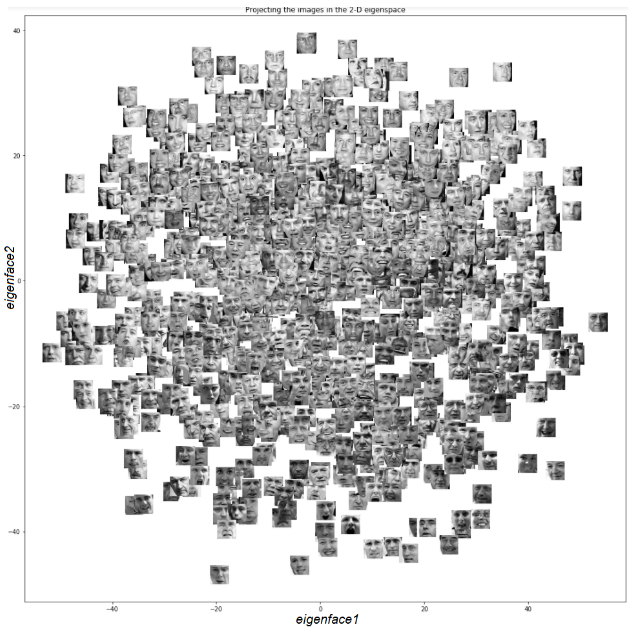

Figure 1: The face-feature plot (example from `https://sandipanweb.wordpress.com/2018/01/06/eigenfaces-and-a-simple-face-detector-with-pca-svd-in-python/`.

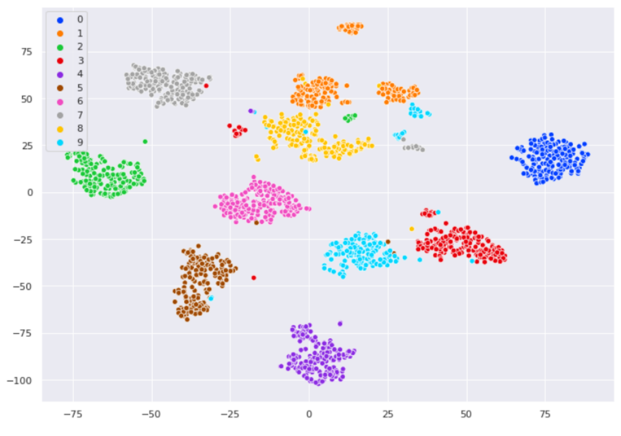

Figure 2: T-SNE plot (example from `https://towardsdatascience.com/t-sne-python-example-1ded9953f26`.