Practical Assignment (24 + 12 Points)

Multimedia Retrieval (Basic Concepts) – SS20

Data Analysis and Visualization Group
University of Konstanz

Prof. Dr. Daniel A. Keim
Philipp Meschenmoser, Matthias Kraus, Thilo Spinner

Submission deadline: **Wed., 8 July 2020, 08:00**
Upload of ZIP (Code, Model, PDF) in ILIAS as Group Submission

1 Introduction

The aim of this assignment is to develop a content-based image retrieval pipeline. In contrast to manual feature extraction techniques, this assignment will be based on the **autoencoder** deep learning architecture to automatically extract features from the dataset. An autoencoder is an unsupervised deep learning model, consiting of an **encoder** and a **decoder** network. As shown in Figure 1, the encoder decreases the dimensionality of the input up to the layer with the fewest neurons, called **latent space**. The decoder then tries to reconstruct the input from this low-dimensional representation. This way, the latent space forms a bottleneck, which forces the autoencoder to learn a compression representation of the data.
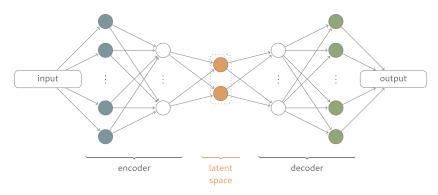

Figure 1: The abstract architecture of an autoencoder. The encoder compresses the input to a low-dimensional representation in the latent space, the decoder tries to reconstruct the output from the latent activation.

2 General Information

- You have to form groups of **2 persons**. Please get in touch with one of the supervisors if you cannot find a partner.

- You should finish the project in **two weeks**, and present your solution in the next exercise which will be on **Wed., 1st July 2020** in the regular slot. After that you will have one more week to improve your implemenation and tune your pipeline based on the feedback from the presentation. The final submission deadline is **Wed, 8th July 2020, 8:00.**

- The practical assignment will be introduced in the exercise slot on **Wed., 17th June 2020**. Each task will be descriped in detail and you can ask questions on things that are unclear. Furthermore, we will offer a helping session in the exercise slot on **Wed., 24th June 2020**.

- You may only use **standard functionalities** in your code, **unless external libraries are explicitly allowed** in the exercise description. Such external packages are marked as *pip:<package name>*. If you are unsure about if you can use some library, get in touch with the supervisors. We can only support programs written in **Python**. You have to provide additional documentation on where to find your code and how it works. Missing or incomplete documentation will lead to point reduction.

- For this practical assignment you can achieve **24 points** plus **12 bonus points** for additional work. The bonus task is optional; the achieved points are added to your overall exercise points without affecting the denominator.

- Please upload your results to ILIAS as a **ZIP file**, containing the trained model as HDF5 file, the Python code, as well as a PDF file with the report of each task (results, images, descriptions). Upload only one file per group in the form of an **ILIAS group submission**.

- Make sure to read the full assignment sheet before starting with your work. You might have to iterate the model building, training and evaluation steps until you reach a satisfying performance. For example, after creating an initial architecture for your autoencoder (Section 3.2), you should train the model (Section 3.3), and check the intermediate results (Section 4). From these check you will most likely detect issues, which bring you back to the model building process. Therefore, the tasks are not strictly sequential. Instead, they serve as an outline for the general procedure.

3 Feature Encoding (16 Points)

In this exercise we will create and train an autoencoder to find a low-dimensional representation of the dataset images. For this, you can either use the Tensor-Flow (pip:tensorflow) or the PyTorch (pip:pytorch) deep learning framework. The following explanations are based on TensorFlow. However, there are many learning resources and blog posts for both frameworks available on the internet.

3.1 Dataset Preparation (1/16 P)

We will use the CIFAR10 dataset. It consists of small images of 32×32 pixels from 10 different classes. Due to its simplicity, the CIFAR10 dataset will keep the training times for our network relatively low, while still being sufficient to showcase the full image retrieval pipeline.

1. **Getting the dataset** – You can either use the example CIFAR10 dataset of Keras or download the dataset from the University of Toronto.

2. **Loading the dataset** – Load the dataset into Numpy (pip:numpy) arrays. We recommend organizing the data as shown in Table 1.

3. **Loading the dataset** – Convert the pixel values from `uint8` numbers in the range $[0, 255]$ to `float32` numbers in the range $[0, 1]$.

4. **Checking the results** – Verify that the data shows the following properties:

   - The value range of the image arrays should be $[0, 1]$.
   - The dtype of the image arrays should be `float32`.
   - The shape of the arrays should be as shown in Table 1.

Variable	Description	Shape
`x_train`	The images of the training dataset.	$(50000, 32, 32, 3)$
`y_train`	The labels of the training dataset.	$(50000, 1)$
`x_test`	The images of the testing dataset.	$(10000, 32, 32, 3)$
`y_test`	The labels of the testing dataset.	$(10000, 1)$

Table 1: Organizing the training and testing datasets as Numpy arrays.

What to submit? Submit the Python code for this exercise, as well as a short description of your approach and the results in the PDF document.

3.2 Building the Autoencoder (11/16 P)

Search for resources on how to build an autoencoder in the deep learning framework of your choice (TensorFlow or PyTorch). Use the same image as input and output of the model. You can start with a simple architecture. As soon as you have a working model (i.e., the network trains) you can further improve the complexity of your architecture. As the final result, you should present a convolutional deep neural network, i.e., a network including multiple convolutional layers for feature extraction in the encoder. **The latent layer should feature**

an output capacity of 10 neurons, resulting in a ten-dimensional feature encoding for each image. The following hints should help you in the process of building the autoencoder.

- Start with a simple architecture, e.g., two dense layers for both encoder and decoder. When training goes well, increase the complexity of the architecture step by step.

- To later encode the features, you will need to execute only the encoder part of your network. Therefore, it will help to directly modularize the autoencoder accordingly. In Keras (as part of TensorFlow 2.x) this can be done by creating three models: the encoder model, the decoder model, and the full autoencoder as the composition of the encoder and decoder models.

- After each modification to your model, train it for a few epochs to detect possible issues (see Section 3.3).

- If the model trains well, check the reconstruction results and analyze the distributions of the latent activations (see Section 4). This will help you to refine the hyperparameter settings (layer sizes, parameter for convolutions, or activation functions) of your model.

What to submit? Provide a description on how you model evolved in the iterative model building and refinement process in the PDF file. Add the Python code of the final model. Also submit a visual representation of the final models architecture (e.g., by using TensorFlow's model_to_dot function) in the PDF file.

3.3   Training the Auto-Encoder   (4/16 P)

Training a model can be a time-consuming task. To monitor the training process, log training metrics (loss and accuracy) using a monitoring system like TensorBoard. In Keras, this can easily done by using the TensorBoard callback. Please include screenshots of the plots visualizing the metrics for the final training of your model, as shown in Figure 2. The following hints should help you in minimizing the effort for training.

- Do not fully train a model that has obvious issues or is not finished yet. After starting the training, observe loss and accuracy metrics over the first few epochs using a monitoring system like TensorBoard. Figure 2 shows, how a healty training process looks like. Be sure to check both training and testing accuracy and loss. If the training accuracy still increases, but the testing accuracy drops, this is a sign of overfitting.

- Save data whenever possible. There is nothing more annoying than having trained a model for hours to realize that the result was never saved. Keras makes model logging easy by providing callback mechanisms that are executed at certain steps during training. Using a ModelCheckpoint you can easily save the full model configuration, including trained weights and model architecture.

- **Only fully train your model if you are confident that it will result in a reasonable performance. The final training might take several hours until a convergence of accuracy and loss is reached.**
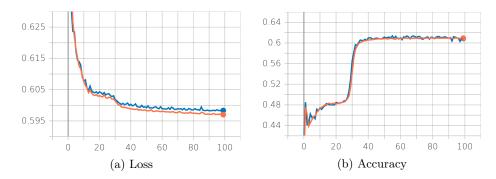

(a) Loss

(b) Accuracy

Figure 2: Metrics logged in TensorBoard.

What to submit? Add the plot of the performance metrics to your PDF report and describe how the training process went for the final model. Submit the trained model as HDF5 file.

4 Sanity Check (4 Points)

There are many sources of errors when building and training a deep learning model. Therefore, checking the results is essential to finally create a working model that solves the task sufficiently well. In this exercise, we will evaluate the results of our model. Note, that you will need to execute this step not only for you final model, but also during the iterative model building and refinement process. For the visualizations in this exercise, you can use the matplotlib package (pip:matplotlib). Furthermore, for data representation, Numpy (pip:numpy) and Pandas (pip:pandas) are allowed. For data processing, such as normalization or pairwise distance computation, you can use ScikitLearn (pip:scikit-learn).

What to submit? For each sub-task, add the results to your PDF report. This includes a description of how the results were achieved, as well as the plots. Submit the Python code for each task.

4.1 Visualizing Reconstruction Results (1/4 P)

The final goal of an autoencoder is to reconstruct the input image as well as possible. Therefore, examining the reconstruction results is vital to assess the quality of our model asides of abstract metrics like accuracy and loss. Plot the first ten images of the testing dataset together with their reconstruction result, as shown in Figure 3. The relatively bad reconstruction quality comes from the high compression in the latent space down to ten dimensions. However, as we will see, this is already sufficient to encode the features for data querying.

Figure 3: Reconstruction results for the first 10 images of the test dataset.

4.2 Distribution Analysis (1/4 P)

To allow for meaningful distance measures in the latent space, the value distribution of the latent activations should look somehow "well-behaved". Therefore, we will examine the distribution of the latent activations over the testing dataset. Use the encoder part of your network to encode 50 random images from the test dataset. Visualize the results using seaborn's (pip:seaborn) pairplot function. The result should look similar to the plot shown in Figure 4.

4.3 Projecting Results (2/4 P)

Finally, we want to make sure that our autoencoder has learnt something meaningful. Assuming that two images of the same dataset class will likely share more features than two images of different classes, we use the class information of the
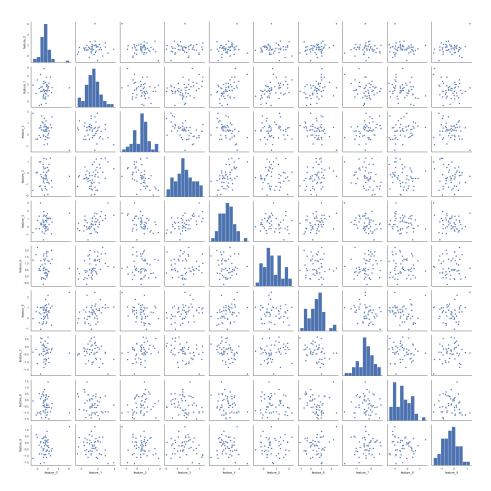
Figure 4: SPLOM of the latent values of the test dataset, including the distribution of each dimension.

dataset to assess the autoencoders performance. Project the encodings for the images of the test dataset to 2D using UMAP (pip:umap-learn). Visualize the results in a scatterplot, coloring the dots according to the class information of the data entity. You should at least see a rough seperation of the different classes, as shown in Figure 5.
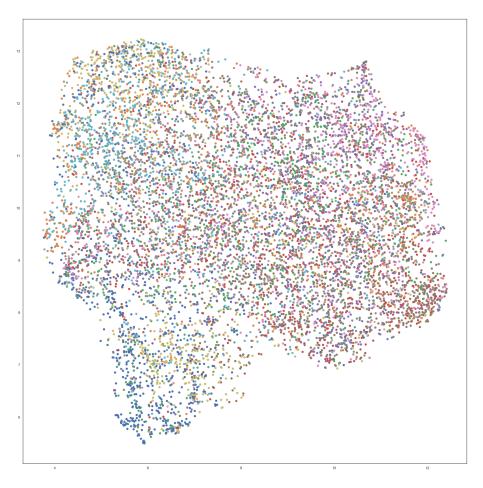
Figure 5: UMAP-projected latent values of the test dataset, colored by class label.

5 Data Querying   (4 Points)

Finally, we want to use our encoder to query for similar images in the dataset. Select 20 arbitrary query images from the test dataset and encode them using your encoder model, resulting in the ten-dimensional encodings $q_1, \ldots, q_{20}$. Then encode all 60.000 images from the combined test and training datasets, resulting in the ten-dimensional encodings $d_1, \ldots, d_{60.000}$. For each query encoding q_n, compute the pairwise distance to all dataset encodings d_m. Sort the distances ascendingly, and plot the ten most similar images for each of the query images. The result for each query image should look similar to the plot in Figure 6.

Figure 6: The ten most similar images to the query image in the top row.

Use at least three different distance measures and compare the results.

What to submit? Describe your approach in the PDF report, including the resulting visualizations. Submit the Python code.

6 Bonus: Improving Stability (12 Points)

So far, we are only training our model with the same image as in- and output. However, an autoencoder can achieve more! By distorting our input data, we can teach the model to "correct" the image for us. For example, we could add noise to our input image and train the autoencoder to still restore the original image. Thereby, the model learns to distinguish relevant features from irrelevant features that are not needed or have to be discarded for the reconstruction. Besides reducing noise, we can also train the autoencoder to recognize input images that were transformed. For this exercise, you can use any image processing library of your choice.

6.1 Pre-processing (6/4 P)

Implement two generator functions yielding distorted variations of an image. The first function should generate images from the training dataset, containing an arbitrary amount of additive gaussian noise. The second function should return images from the training dataset where an arbitrary projective transformation has been applied. The images should maintain their size, i.e., 32×32 pixels. You can set background pixels that do not contain any image data after the transformation to black $(0, 0, 0)$.

What to submit? Submit the python code for the image transformation functions. Add examples for the distorted images to your PDF report and describe your approach and the results.

6.2 Re-training (4/4 P)

Continue the training of your already existing model while mixing in a certain amount of distorted images. It is important that you do not start the training process from scratch, since the network might not be able to pick up the more complex features without pre-training. Log performance metrics and verify that your model trains well. With increasing accuracy you can mix in more distorted images or increase the complexity of the transformations that are applied to the inputs.

What to submit? Add the plots of the performance metrics during training process to the PDF report. Describe, how you mixed in the transformed images and how this affected the loss/accuracy of the model during training. Submit the final model as HDF5 file.

6.3 Querying Modified Data (2/4 P)

Evaluate the robustness of the old model against the retrained one. How do both models respond to transformed images? Are the most similar results also visually similar?

What to submit? Add the results to your PDF report, including a description of what you experienced during the evaluation of the improved model. Compare the old model to the new one and describe how their performance differs. Provide the Python code for the image querying.