2 ACML Homework: Convolutional Autoencoders

Disclaimer: This is a new assignment, that was constructed by three very smart people. However, this will be its first run, and you are its beta-testers, so there might be parts that leave room for improvement, and if you find such a part: Please let us know!

An autoencoder (AE) is a neural network that is trained to copy its input to its output. It has one or more hidden layers **h** that each describe a code used to represent the input. The network may be viewed as consisting of two parts: an encoder $h = f(\mathbf{x})$ that produces the code and a decoder that produces a reconstruction of the data $r = g(\mathbf{h})$.

The encoder is tasked with finding a (usually) reduced representation of the data, extracting the most prominent features of the original data and representing the input in terms of these features in a way the Decoder can understand. The Decoder learns to read these codes and regenerate the data from them. The entire AE aims to minimize a loss-function while reconstructing. In their simplest form encoder and decoder are feedforward neural networks. When the inputs are images, it makes sense to use convolutional neural networks (CNNs) instead, obtaining a convolutional autoencoder (CAE).

A CAE uses the convolutional filters to extract features. CAEs learn to encode the input as a combination of autonomously learned signals and then to reconstruct the input from this encoding. CAEs learn in what can be seen as an unsupervised learning setup, since they don't require labels and instead aim to reconstruct the input. The output is evaluated by comparing the reconstructed image by the original one, using a Mean Square Error (MSE) cost funtion.

Data

In this lab we will be using the CIFAR-10 dataset that you can find in the link below:

- CIFAR-10 dataset (https://www.cs.toronto.edu/~kriz/cifar.html).

The CIFAR-10 dataset consists of 60 000 thousand color images of dimensions 32 × 32. Images can be of any of 10 categories, and each image may only be of one such category, although for this lab the category of the images is largely irrelevant. Here are some examples of what to expect:

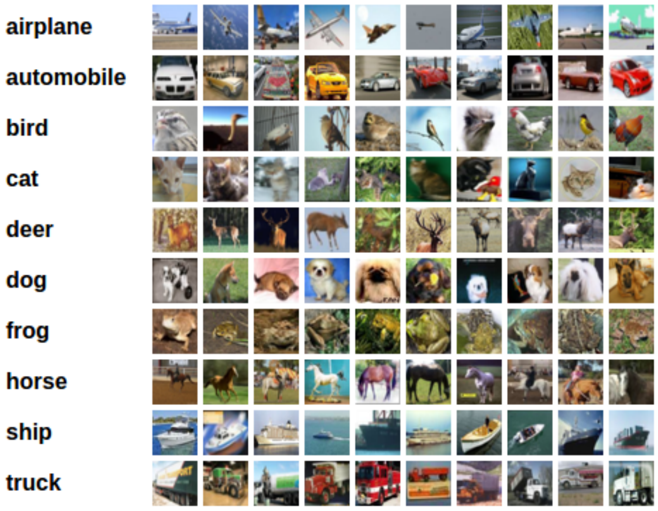


Figure 1: CIFAR-10 dataset samples.

Links are provided to download the dataset in a format already prepared for python or Matlab. However, since this dataset is so common, there is likely a binding in your preferred machine learning toolkit that will download the data for you in a format ready to be used. For instance:

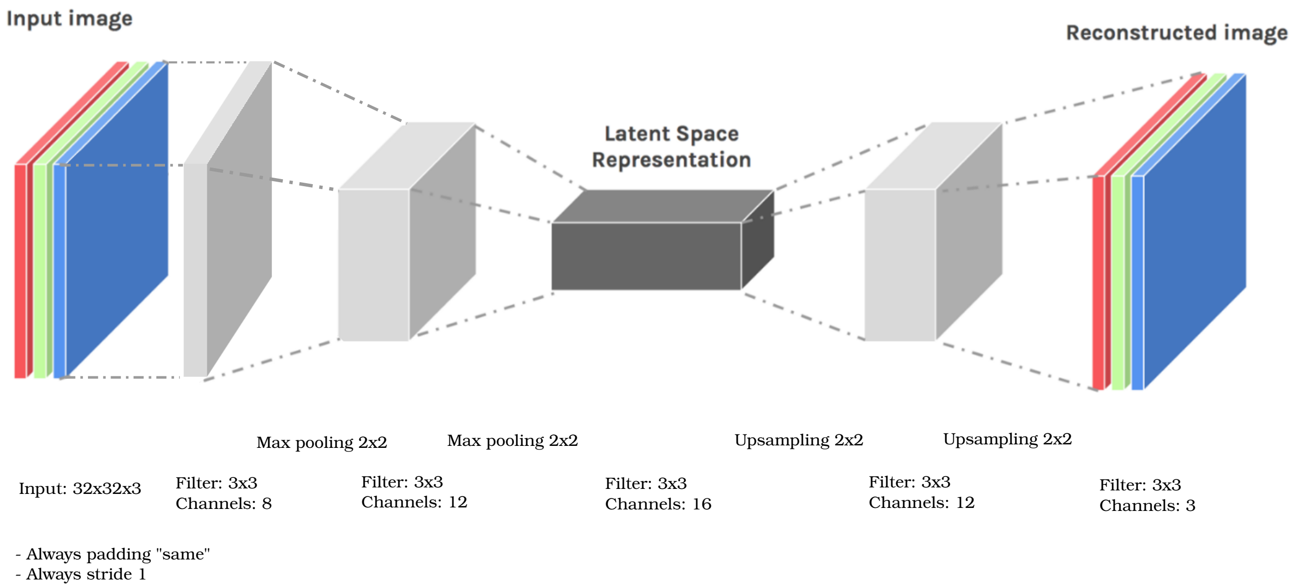
- Keras – https://keras.io/api/datasets/cifar10/.

- Matlab – Look up the `helperCIFAR10Data` function.
- Pytorch – Look up the `torchvision.datasets.CIFAR10` class.

Reconstruction

In a first step, you will use a Deep Learning library to construct a convolutional autoencoder that takes a image from the CIFAR-10 dataset as input, produces a representation in latent space, and attempts to reconstruct the original image as precisely as possible. The only data the autoencoder requires is the colored image, which will act as both the input and the output to compare against.

Build the following network architecture using the DL library of your choice:



The above network consists of a total of 9 layers. The input images (of size 32 × 32 × 3) are fed into a convolutional layer with filter size 3 × 3 and 8 channels (or dimensions). This is followed by a max pooling layer which downsamples the image with a 2 × 2 filter, which effectively reduces the size of the image by 4 (two in each dimension). Then, another layer of filter size 3 × 3 and 12 channels follows.

Some points to bear in mind:

- Padding is ”same” throughout the entirety of the network. In other words, padding is 1 for convolutional layers, and 0 for pooling or upsampling layers.
- Stride is always 1.
- Use the images as both the input and the desired output of your network.
- As a suggestion, the rectified linear unit (ReLU) is a simple and efficient activation function which usually works well with convolutional layers.
- As another suggestion, the error function ”means square error” or ”binary cross-entropy” usually works well for these type of workloads.

Exercises

1. Divide your dataset into training (80%), validation (10%) and test (10%). Normalize the data.
2. Implement the autoencoder network specified above. Run the training for at least 10 epochs, and plot the evolution of the error with epochs. Report also the test error.

The size of a 2-D convolutional layer representation can be calculated as the output of that layer by employing the following formula:

$$(\frac{W - K + 2P}{S} + 1)^2 \cdot C \tag{1}$$

where:

- W – Input volume.
- K – Kernel size.
- P – Padding.
- S – Stride.
- C – Number of channels.

For instance, the first convolutional layer has an input volume of 32×32 ($W = 32$), a kernel size of 3×3 ($K = 3$), a padding (P) of 1, a stride (S) of 1 and 8 channels (C). Therefore, the size of the first convolutional layer representation can be calculated as follows:

$$(\frac{32 - 3 + 2 \cdot 1}{1} + 1)^2 \cdot 8 = 8192 \tag{2}$$

Exercises

1. What is the size of the latent space representation of the above network?
2. Try other architectures (e.g. fewer intermediate layers, different number of channels, filter sizes or stride and padding configurations) to answer questions such as: What is the impact of those in the reconstruction error after training? Is there a correlation between the size of the latent space representation and the error?

Colorization

You should now have an autoencoder capable of encoding and reconstructing a colored image into and from a contractive latent space. Given that the autoencoder decodes from a lower-dimensional representation, the encoder must capture regularities in the input data. These regularities can be leveraged in more interesting applications such as colorizing black-and-white images.

One aspect that makes autoencoders so powerful as feature extractors is that they learn in an unsupervised setting: No labeled data is required and as such any collection of images at hand can be used. However, that does not mean that the same exact tensor is required as input and target output. It is possible to add an additional preprocessing step to the input layer that impairs the image in some way and still let the CAE reconstruct the unimpaired original. Generally, this is useful if there is a map $f : \mathbf{X} \to \mathbf{Y}$ that is hard to compute but for which the inverse f^{-1} can be computed easily. A classical application for such a mapping is the removal of noise from an image. In this assignment, the addition of colors to black and white images is investigated. As such, our f^{-1} is the grayscaling operation.

Exercises

1. Adapt your network from the previous part such that it learns to reconstruct colors by feeding in grayscale images but predicting all RGB channels. As a starting point, use the hyperparameters (including the network architecture) that you identified to yield the best performance in Exercise 3.2.
2. Report on your results and reason about potential shortcomings of your network. What aspects of the architecture/hyperparameters/optimization could be improved upon to fit the model more adequately to this application? Try out some ideas.

(Hint) A neat trick is to not predict the full color image, but only its chrominance - the proportion of the image determining the colors but not the luminance. By predicting the chrominance, we relieve the model of also reconstructing the details (such as contours) that we already have in the grayscale image. The predicted chrominance can then be merged with the luminance captured in grayscale to reconstruct the full image.