Convolutional Autoencoders

ACML Assigment 2

Lavinia Pulcinella (i6233926) Enrique Barrueco (i6242336)

November 2020

Contents

1 Convolutional Autoencoders **1**

2 Image Reconstruction **1**
2.1 Model Performance 2
2.2 Different Architectures 3
2.3 Latent Space Dimension 3

3 Image Colorization **3**

A Disclaimer **4**

1 Convolutional Autoencoders

The present project is a Convolutional Autoencoder (CAE) implementation for image reconstruction based on the *pytorch* library. The goal is to first apply a number of convolutional layers to extract features from our image, after which we apply deconvolutional layers to upscale of our features to reconstruct the image. The `Cifar 10` Datasets consists in 60k observations and is available through pytorch using `torchvision.datasets.CIFAR10`.

Data was standardized using mean and standard deviation of each RGB channel. The default division of the dataset is 83-17 for training and testing respectively. Thus, in order to have a 80-10-10 division for training testing and validation we first concatenated the two datasets (i.e. training and testing) using the `torch.utils.data.ConcatDataset`, then performed an 80-20 division using `torch.utils.data.random_split` obtaining training and testing sets. The 20% set was then further divided into training and validation by splitting the previous set in half.

Training, testing ans validation sets are then wrapped by the `torch.utils.data.DataLoader` class which represents a Python iterable over the sets.

2 Image Reconstruction

The given model was then implemented using the `pytorch`'s `nn.Module class`. The encoding and decoding layers were structured based on the given architecture, for a total of 9 layers.

The high level idea of an autoencoder is to replicate the input as an output. To achieve this, the CAE is composed by an encoding part which aims at reducing the dimension of the input, the resulting reduced

representation (i.e. latent space representation) and a decoding part which "brings back" the original dimensions of the input.
In practice the compressed representation in between the encoding and decoding "parts" is the one holding key information about the input which is then exploited in order to reconstruct the image.

In pytorch, the encoder uses convolutional layers though the `conv2d` function for which one needs to specify the input number of channels (to that particular layer), the output channels, the kernel size (or filter size) which in the present architecture is always 3 and the padding and stride size used (always 1 for both in case of convolutions - padding is 0 for MaxPooling and Upsampling).

The decoder uses **transposed convolutional** layers via the `ConvTranspose2d` function.

2.1 Model Performance

The CIFAR 10 datasets includes image labels which are not of interest in this case (they would be in a classification problem). The reconstruction process was evaluated using the MSE (which will be what can be referred to as the reconstruction error of the network, it computes the square error at every pixel across our training data. We optimize this loss function (criterion) using the Adam optimizer.

The CAE was trained over 10 epochs using the `train()` function which iterates over each epoch. Each time the gradient is set back to zero using the `otpimizer.zero_grad()` function. The reconstruction over the training examples is achieved by outputting the CAE model. The MSE loss is backpropagated though the network using the `loss.backward()` function, and optimized though the `optimizer.step()` function.

After training the autoencoder for 10 epochs as the assignment indicates we get the following graph that represent the evolution of the training error.

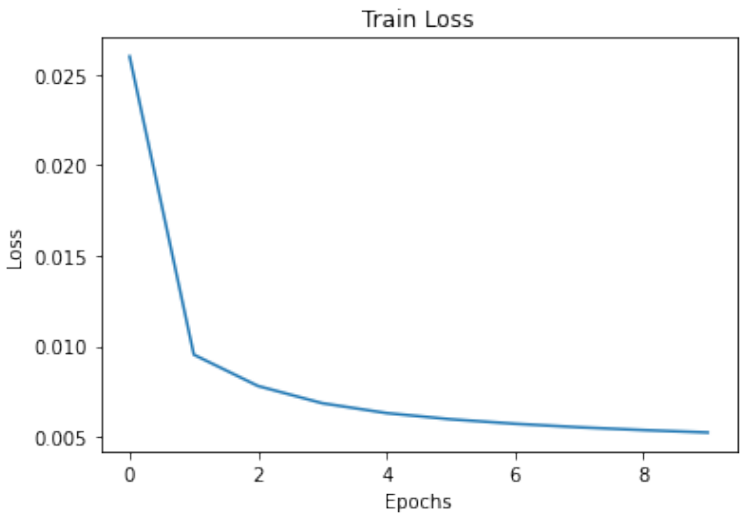

Figure 1: Train Loos of Autoencoder

To make sure that our model is not overfitting the training data we also tested the error on the test data, we obtain a very similar error of 0.005201, so our model seems to be valid, figure 3 shows an example of the reconstruction we achieved.

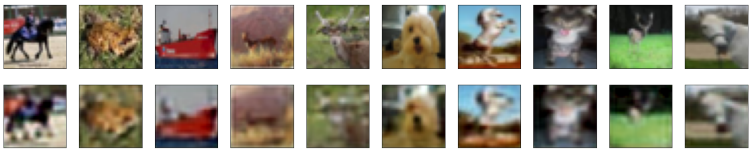

Figure 2: Reconstruction of Images by our Autoencoder

2.2 Different Architectures

Given our inexperience with these kind of models and libraries we were not able to successfully find a way to systematically test different architectures and obtain concrete and comparable results. However, we noticed that autoencoders with less convolutions produced a higher error but the reconstructed images were less blurry that the ones produced by our initial network, this could be due to images being less compressed and therefore less information being lost. We would also expect to overfit more if we used more layers of encoding.

2.3 Latent Space Dimension

The size of the latent space of the middle convolutional layer where the data is most compressed, is given by applying formula (1), with: (W, K, P, S, C) = (8, 3, 1, 1, 16), which results in a size of 1024. Following the arguments of the last point, with a bigger latent space we might obtain a smaller error but at the expense of over-fitting on the training data. Conversely, smaller latent spaces associated with simpler models tend to result in a bigger error.

3 Image Colorization

We encountered a number of problems when trying to convert images to grey-scale and use our auto-encoder to reconstruct colors using PyTorch, that is the reason why we decided to switch to Keras, for which we found more helpful online resources that allowed us to build a working model using our original architecture.

Finally, we were not able to get our network to colour greyscale images, we only managed to do it using the following GitHub Code[1].

We are including these results first as a reference for completeness but also because the above resource proved to be useful in - at least partially - understanding the colorization-task pipeline. The original plan was to use it as guidance for the colorization task. However, we ran out of time and we weren't able to replicate it by ourselves.

In theory, in order to colorize images, one needs to first convert the original RGB images into single gray-scale images. Thus the aim is to infer a full-colored image. In the assignment it was suggested to focus on chorminance and luminance of the images which would mean to work in the LAB colorspace (Lightness, A and B) which contains the same information as in RGB but it makes it easier to separate the lightness channel from the other two (which are referred to as just A and B). As for image reconstruction the model should first apply a number of convolutional layers to extract features from the input images and then "bring back" dimension to upscale the features.

[1] https://github.com/PacktPublishing/Advanced-Deep-Learning-with-Keras/blob/master/chapter3-autoencoders/colorization-autoencoder-cifar10-3.4.1.py

Figure 3: Coloring of Grey-Scale Images Using a Third Party Notebook

A Disclaimer

This was our first attempt in dealing with libraries such as PyTorch or Keras (or any ML libraries for that matter). Balancing both theory review and the actual implementation techniques (as well as other uni-related obligations) was hard and we spent quite a lot of time in understanding how the general pipeline works for both libraries. PyTorch was chosen first since it seemed easier to debug. However, there are more online resources using Keras and for colorization we "played around" with it in order to get a feeling of how the library works.