Machine learning Assignment5

Chiara Saporetti

December 2020

Abstract This assignment is divided in two parts:
Task A aims at implementing in MATLAB a simple autoencoder.
Task B addresses more complex cases and aims at getting acquainted with Matlab's pretrained convolutional neural networks and their uses.

1 Task A

1.1 Introduction

The autoencoder is an unsupervised artificial neural network that firstly compressed and encodes data and then reconstructs it back from the reduced representation to a representation that is as close to the original input as possible. By design, it reduces data dimensions by learning how to ignore the noise in the data. Its structure is shown in Figure 1.
In this assignment the autoencoder is a multi-layer perceptron neural network

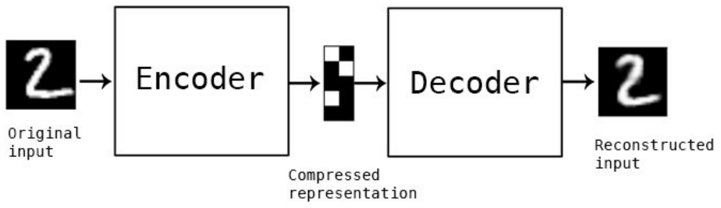
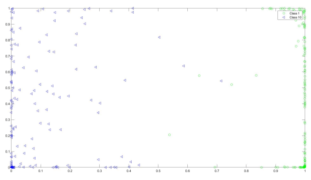


Figure 1: Autoencoder model

with one input layer, one hidden layer and one output layer.
It is trained using the same pattern as both the input and the target: it implements an identity mapping: input = target. So it is an unsupervised problem, specifically a self-supervised problem.
The number of hidden units should be smaller than the number of inputs and outputs since it learns a more compact version of the input. The pattern of activations in the hidden layer is the representation of the problem: what we desire is that similar patterns have similar representations. The number of


(a) Digits 1 and 0

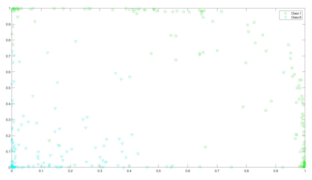
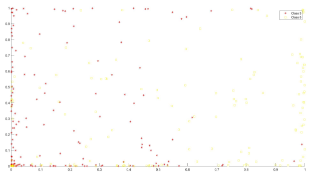

(b) Digits 1 and 8

Figure 2: Digits with different appearance


(a) Digits 3 and 5

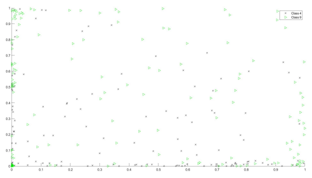

(b) Digits 4 and 9

Figure 3: Digits with same appearance

hidden layers of this autoencoder is nh = 2 because the 2 activations are used as axis of the activation space to plot the data.

2 Matlab code

2.0.1 main.m

The code loads the MNIST dataset lines relative to two chosen digits. It shuffles the examples and extracts subsets, by finally merging the datasets.
It then trains an autoencoder with the dataset by using the Matlab function trainAutoencoder() and obtains the compressed representation of the data by using the Matlab function encode(). In the end it generates the plots by using the given function plotcl() and saves the plotted results as images.

2.1 Results

Hereby are shown the results obtained. Each point represents a digit, and the same digits have the same color and shape. Figure 2 shows two couples of digits with different appearance: they are linearly separable. Figure 3 shows two couples of digits with similar appearance, and the results are more mixed than in Figure 2, so they're not linearly separable.

3 Task B

3.1 Convolutional neural networks

A Convolutional Neural Network is a Deep Learning algorithm which can take an image in input and assign importance (learnable weights and biases) to various aspects/objects in the image. The required pre-processing is much lower if compared to other classification algorithms since, with enough training, CNNs have the ability to efficiently learn the necessary filters. [6]
CNNs vary in structure but have some common layers:

- **Convolutional layer**: The layer's parameters consist of a set of learnable filters K. Each filter shifts over the whole image, every time performing a matrix multiplication operation between K and the portion P of the image over which it is hovering (Figure 4). In this way it produces a 2-dimensional activation map of that filter.

  The objective of the Convolution Operation is to first extract Low-Level features such as edges, color, gradient orientation, etc. With added convolutional layers, the architecture adapts to the High-Level features as well. [1]

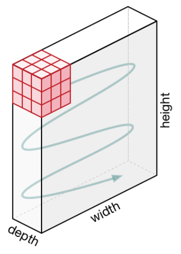


Figure 4: Kernel shifting on image

- **Pooling layer**. It is a form of non-linear down-sampling useful for extracting dominant features which are rotational and positional invariant.
  There are two types of Pooling: Max Pooling (that returns the maximum value from the portion P of the image covered by the Kernel) and Average Pooling (that returns the average of all the values from P) (Figure 5).
  The pooling layer serves to progressively reduce the spatial size of the representation, the memory footprint and the amount of computation in the network, and hence to also control overfitting.

- **ReLu**: the name is the abbreviation of rectified linear unit, which applies the non-saturating activation function f(x)=max(0,x). It removes negative values from an activation map by setting them to zero. It increases the nonlinear properties of the decision function and of the overall network without affecting the receptive fields of the convolution layer.
  Note: other functions may be used too, but ReLu is the fastest.

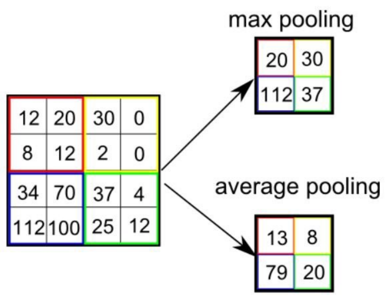


Figure 5: The two kinds of pooling

- **Fully connected layer**: the high-level reasoning in the neural network. Neurons in a fully connected layer have connections to all activations in the previous layer.
- **Loss layer**: it specifies how training penalizes the deviation between the predicted (output) and true labels and is normally the final layer of a neural network. Various loss functions may be used and usually Softmax loss is used for predicting a single class of K mutually exclusive classes.

3.2 The pretrained neural networks

Pretrained networks have different characteristics that matter when choosing one to apply to a problem. The most important characteristics are network accuracy, speed, and size. Choosing a network is generally a tradeoff between these characteristics. Here are the convolutional neural networks used in the following tasks. They are all trained on the ImageNet database and classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals.[4]

- **GoogLeNet** is 22 layers deep and has an image input size of 224-by-224.
- **AlexNet** is 8 layers deep and has an image input size of 227-by-227.
- **ResNet-18** is 18 layers deep and has an image input size of 224-by-224.

3.3 The datasets

- **ImageNet** is a large visual database designed for use in visual object recognition software research. More than 14 million images have been hand-annotated by the project to indicate what objects are pictured and in at least one million of the images, bounding boxes are also provided.
- **Merchdata** is a small data set containing 75 images of MathWorks merchandise, belonging to five different classes (cap, cube, playing cards, screwdriver, and torch). This data can be used to quickly try transfer learning and image classification. The images are of size 227-by-227-by-3.

3.4 Goals when using a pretrained neural network

The goals, when using a pre-trained neural networks, can be:

- **classification**: use them as black boxes to classify new images.
- **feature extraction**: use them by reading the activation values from a layer other than the output one. In this way all the previous layers are used as feature extractors. The activations from an intermediate layer can be used as an input for a final classifier, e.g., a shallow multi-layer perceptron.
- **transfer learning**: use them as the initialization to learn a new (but similar) task.

3.5 Deep learning Matlab demo

This task consisted in following Matlab's tutorial on deep learning [2]. The first step in this task is to understand how to use deep learning to identify objects on a live webcam by using only 10 lines of Matlab code. The neural network used for this goal is the AlexNet and the camera is simply the computer's camera.

3.5.1 Results

Some of the results I obtained are shown in Figures 6 to 9.

3.6 Use pretrained Convolutional Neural Network

This task is about using a pretrained deep neural network as a starting point to learn a new task [5]. The neural nework is the GoogLeNet and we use it by giving it some new images as inputs to see if it can recognize them.

3.6.1 Results

In order to try it I downloaded some images from Google, resized them to match the desired input size, and used the classify() function of the neural network on them. Some of the results I obtained are shown in Figures from 10 to 15.

It also possible to display the top five predicted labels and their associated probabilities as a histogram: since many categories are similar, it is common to consider the top-five accuracy when evaluating networks.

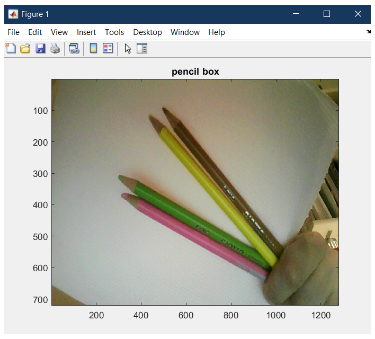


Figure 6: Pencils

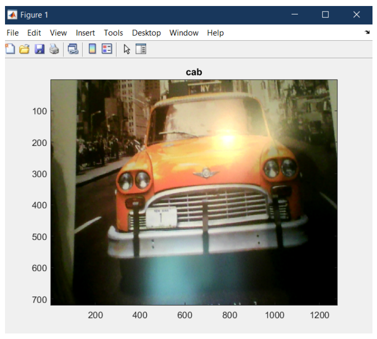


Figure 7: NY cab

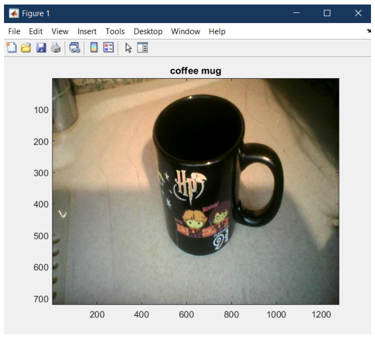


Figure 8: Cup

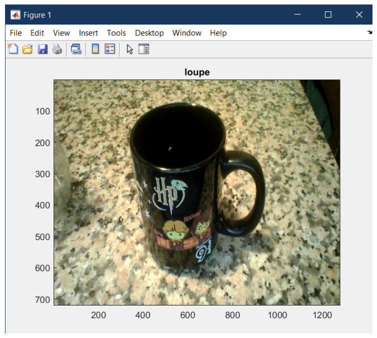


Figure 9: Cup on granite

3.7 Transfer learning via feature extraction

This task consists in extracting learned image features from a pretrained CNN and using those features to train an image classifier. This is done by following Matlab's tutorial [3].
The dataset is the MerchData dataset, that is split to have 55 training images and 20 validation images. The neural network is the ResNet-18 network.
The feature extracted from the training images at the selected internal layer of the ResNet-18 layer are used as predictor variables to fit a multiclass support vector machines (SVM).

3.7.1 Results

In Figure 16 is reported a classification of four different random images from the MerchData dataset.

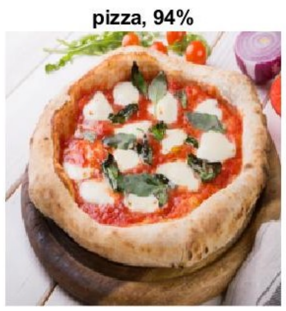


Figure 10: Pizza

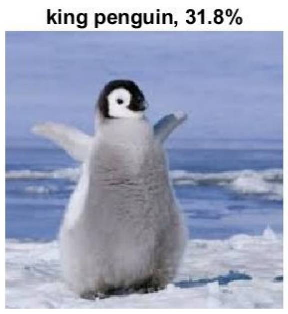


Figure 11: Penguin

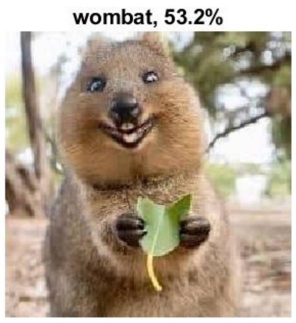


Figure 12: Quokka

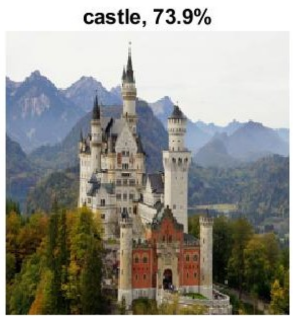


Figure 13: Castle

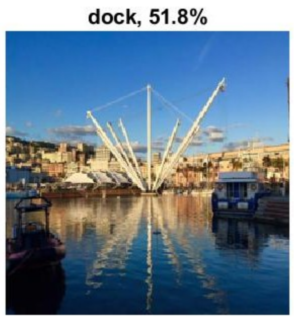


Figure 14: Genova dock

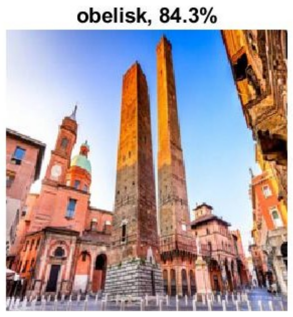


Figure 15: Bologna towers

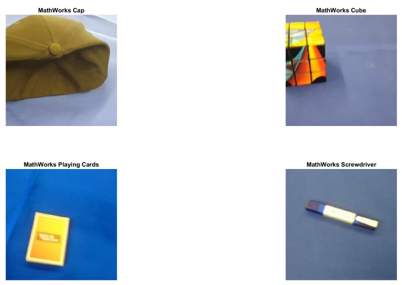


Figure 16: Predictions

3.8 Transfer learning via retraining

This task consists in using transfer learning to retrain a convolutional neural network to classify a new set of images. This was done by following Matlab's tutorial [1]. The dataset is the MerchData dataset and the CNN is the GoogLeNet.

- The main idea is to replace the last layers of the CNN with new ones that adapt to our problem. In fact, the last two layers, (loss3-classifier and output in GoogLeNet), contain information on how to combine the extracted features into class probabilities, a loss value and predicted labels. The new last layer we'll specify is a fully connected layer with the number of outputs equal to the desired number of classes in the new data set.
- Optionally, it's possible to "freeze" the weights of earlier layers in the network by setting their learning rates to zero: this means that their gradient won't be updated and this can significantly speed up network training. If the new data set is small, it can also prevent those layers from overfitting to the new data set. Here I've frozen the first 10 layers. In general, it's desirable to have fast learning in the new layers, slower learning in the middle layers, and no learning in the earlier, frozen layers.
- Then it's a good approach to use an augmented image datastore to resize the training images and optionally perform additional operations to perform on the training images: flipping, translating, scaling. Data augmentation helps prevent the network from overfitting.
- You also need to specify the number of epochs to train for, the mini-batch size and validation data.
- At last, you can actually train the CNN.

References

[1] *https://en.wikipedia.org/wiki/Convolutional$_n$eural$_n$etwork.*

[2] *https://it.mathworks.com/help/deeplearning/gs/try-deep-learning-in-10-lines-of-matlab-code.html.*

[3] *https://it.mathworks.com/help/deeplearning/ug/classify-image-using-googlenet.html.*

[4] *https://it.mathworks.com/help/deeplearning/ug/data-sets-for-deep-learning.html.*

[5] *https://it.mathworks.com/help/deeplearning/ug/pretrained-convolutional-neural-networks.html.*

[6] *https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.*