COMP6223 - Computer Vision
Coursework 3

Azzino Tommy & Drago Matteo

Introduction

All the code for this coursework has been written in MATLAB.
In all our algorithms we used `imageDatastore` in order to speed up images retrieval and manipulation. In particular, we decided to use 25% of the training set for validation: in this way we tested our different algorithms multiple times and in different configuration before categorizing the test set. The split of the training set, where possible, has been done with `splitEachLabel`.

Run 1

As first approach we had to develop a classifier which use as features *tiny images*. This particular representation consists on few simple steps: initially we resize the image to a 16x16 matrix, then we build a vector of 256 pixels composed of the consecutive rows of the new image.
After that, in order to improve classification performances, we implemented two different types of normalization to the vector, which in the following we refer to as $\mathbf{x}$:

- **standard normalization**: $\bar{\mathbf{x}} = \frac{\mathbf{x}-\mu}{\sigma}$ where μ is the mean of $\mathbf{x}$ and σ is its standard deviation
- **unit length normalization**: $\widehat{\mathbf{x}} = \frac{\bar{\mathbf{x}}}{\|\bar{\mathbf{x}}\|}$

In this way we obtained a set of unit length vectors of zero mean.
The idea behind the classifier is that one vector representing an image of a specific class will likely be similar to other vectors of the same category. So, once we perform this operation for all the training set, we can determine the category of each image of the validation set using the **k-nearest-neighbour** classifier: this means that we evaluate the distance between the tiny image to validate and all the vectors from the training set; then, we pick the k nearest vectors (of which we know the class) and we classify our image with the most represented class in the neighbourhood. In case we have two classes most represented we can implement different choice polices, in our case we used the first classes returned by MATLAB.
In order to find better performances, we tried to tune the value of k for the several measures of distance available in `vl_alldist2`, in particular:

$$\mathbf{L_{INF}} = max|X - Y| \qquad \mathbf{L_2} = sum(X - Y)^2 \qquad \mathbf{L_1} = sum|X - Y| \tag{1}$$

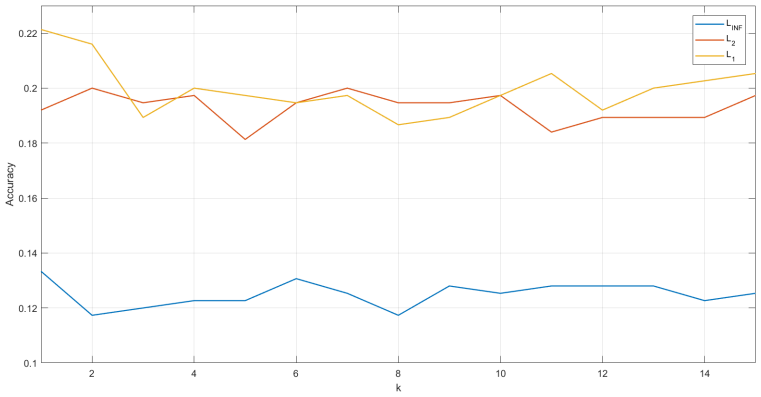

Figura 1: Performances of different distance measures in function of k

As we can see from Figure 1, while L_{INF} can't reach even the 14% of accuracy, L_2 and L_1 can reach up to 20% and beyond. It's interesting to highlight that the latter perform better when we use either just

the nearest neighbour or we increase the number of neighbours beyond 10, while L_2 presents its maximum around 7 or 8. In conclusion, considering its stable behaviour on different runs that we performed, we decided to implement our classification using L_2 with k = 7. Of course, given the simplicity of these features, we couldn't expect more than 20% of accuracy. On the following sections we will implement different features extraction methods to improve these performances.

Run 3

For this last approach we decided to implement **transfer learning**: this means that in order to extract significant features from our images we used layers of a pre-trained neural network. So, to complete our classification architecture, we re-trained just the last layers used for classification on our training set, without doing backpropagation on the previous layers. Surely one advantage of this solution is that allowed us to save a considerable amount of computational time for the training of all the layers of the network.
For this particular problem we decided to use the convolutional neural network (CNN) `alexnet`, winner of the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC2012). At the time, it takes an entire week to train the complete network with the competition dataset (1.2M of images).