Cake classification
Machine Learning

Alice Daldossi

Abstract

In this laboratory activity the goal is to solve an image classification problem: 1800 photos have to be classified between 15 desserts. First, I train a network using different types of low-level feature approach; then I make another network, but this time it takes the features from a pre-trained one instead of constructing them; the last network is built using transfer learning. I conclude studying the errors made.

1 Data

The data set contains 120 images for each of 15 kinds of dessert. For each class 100 images are in the training set and 20 in the test set. Usually the images have different resolutions, and so their sizes; for this reason as first step it is useful to put them in a standard form. I will work with pretty small images of 224×224 pixels in order to take not so much time in the training process.

2 Feature extraction and training

Imaging recognition is a very difficult process to be done, so feature extraction could be a valid choice to simplify it.

2.1 Low-level features

Script: `lowlevel_fe.py`

The low-level features approach intends to identify the basic properties that are commonly used to recognize visual stimuli. I tried color histogram (CH), edge direction histogram (EDH), gray-level co-occurrence matrix (CM) and rgb co-occurrence matrix (RCM) with and without hidden layers. A possible issue with low-level features approach is that all the colors will be described in the histogram, but most of them have nothing to do with the main subject (for example the colors of the plate or the table). A solution could be combining different low-level features in a single vector. All the results are summarized in the table 1 and 2.

I affirm that this report is the result of my own work and that I did not share any part of it with anyone else except the teacher.

	edh	ch	cm	rcm	edh+ch
Training accuracy	13.4 %	22.5 %	19.2 %	18.9 %	18.7 %
Test accuracy	10.0 %	19.0 %	**20.0 %**	16.7 %	17.3 %

Table 1: Comparison of the accuracies of the classifiers built with different low-level features but using always a learning rate of 0.0001, a batch size of 50, no hidden layers (so it is a linear classifier) and 5000 epochs.

	edh	ch	cm	rcm	edh+ch	rcm+ch	edh+ch+cm
Training accuracy	13.9 %	33.9 %	28.3 %	25.7 %	25.5 %	26.2 %	25.7 %
Test accuracy	11.3 %	**26.0 %**	24.7 %	21.0 %	21.7 %	21.0 %	22.0 %

Table 2: Comparison of the accuracies of the classifiers built with different low-level features but using always a learning rate of 0.0001, a batch size of 50, a single hidden layers of 64 neurons and 5000 epochs.

2.2 Neural features

Script: `neural_fe.py`

The accuracies obtained with the low-level features are not so satisfying, better results can be obtained taking advantage of the neural features computed using the `PVMLNet` convolutional neural network (CNN). This procedure starts removing the last layer from the pre-trained `PVMLNet` CNN. In this way the new last layer does not have any interpretations in terms of probability of belonging to a certain class, it is just a vector of numbers that encodes the presence of the patterns in the image, so I use it as the vector of features that trains a new classifier (linear MLP). The last hidden layer (-3) is empirically known as the best one to be substituted, but in my experiments the two before the last (-5) seems to be better, as written in table 3.
I am using 1024 vectors of features to represent 1050 images, so I can almost dedicate a feature to each image. For this reason, it is plausible to have a training accuracy of 100%.

	-3	-5	-8	-10
Training accuracy	100.0 %	100.0 %	100.0 %	100.00 %
Test accuracy	80.7 %	**89.7 %**	85.7 %	71.0 %

Table 3: Comparison of the accuracies of the classifiers built substituting different hidden layers of the original CNN, but always a learning rate of 0.0001, a batch size of 50, no hidden layers (so it is a linear classifier) and 5000 epochs.

2.3 Training

Script: `training.py`

Once extracted the features, I can finally train the network loading the data saved in the previous process (choosing between low-level or neural features) and then get the accuracies above discussed.

3 Transfer learning

Script: `transfer_learning.py`

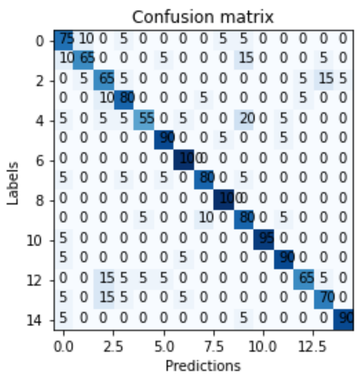
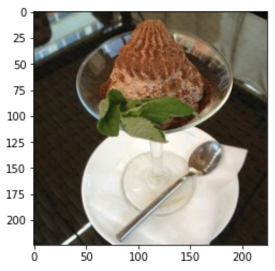
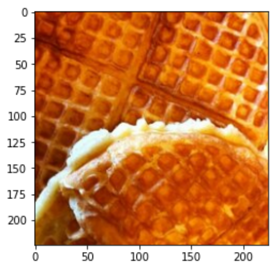


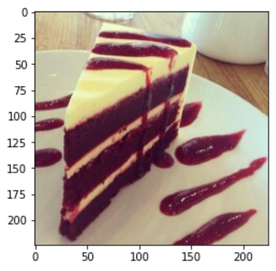
Figure 1: Confusion matrix of the MLP with neural features.



(a) An ice cream (9) classified as a chocolate mousse (4) with 100% of probability.



(b) Waffles (14) classified as an apple pie (0) with 99.9% of probability.



(c) A red velvet cake (12) classified as a carrot cake (2) with 99.9% of probability.

Figure 2: Pictures of errors with the highest probability of classification.

In order to get another good classifier, I can adapt the pre-trained network `PVMLNet` to my classification problem performing the so called *transfer learning*. In particular, this approach replaces the last layer (that in this case has 1000 neurons) of the `PVMLNet` with another one that has just 15 neurons (as the number of classes of the cake classification) and then trains the weights of only this new last layer.

4 Analysis

Script: `analysis.py`

It is time to analyze the errors. Let's consider the network built with neural features. In this case the confusion matrix C is the one in figure 1 and I can conclude that the most common error is the misclassification of class 4 (chocolate mousse) as class 9 (ice cream); although there are other 4 misclassifications that are quite frequent too, a couple of them are class 13 (tiramisu) instead of 2 (carrot cake) and 9 (ice cream) instead of 1 (cannoli). I notice that the value on the diagonal $C_{8,8}$ is equal to 100, this means that the class 8 (donuts) has always been correctly classified: any image of donuts has been recognized as class 8, but I cannot say that all the predictions 'donuts' are effectively of this class.
I wanted to see some examples of misclassified images, in figure 2 there are the 3 of them that have the highest probabilities of the assigned (but also wrong) class.