Homework 2 - Venice Boat Classification

Machine Learning 2018-19 - Sapienza

Luigi Russo 1699981

December 26, 2018

Contents

1 Introduction

This is my report for the second homework of Machine Learning course, about the Venice Boat Classification. First of all I will point out which have been the goals of this homework, starting of course from defining the specific classification problem (view section 1.1.1) that I have chosen to focus on. After defining the kinds of algorithms that I have implemented and used (view section 1.1.2), I will describe the dataset (view section 4) and how I have handled the preprocessing phase (view section 4.1). I will finally present the results of the evaluation phase (view section 6).

1.1 Learning goals

1.1.1 Classification problem

I have decided to focus on the problem of classifying **18** different classes of Venice boats. Given an instance of image, I want to be able to compute which is the correct type of boat of the input image. I have decided to use a pretrained model (Inception v3) in order to extract the features of each image: at this point, the feature vector is passed to some classifiers, such as SVM classifier. The model can be finally tested, computing the accuracy and it is possible to plot the confusion matrix.

Metrics The main goal of this homework is to correctly classify the highest number of input images, for this reason I will take into account both accuracy and precision metrics.

1.1.2 Many models

As in the first homework, also in this one I will give great importance to the comparison [1] between different models. Once the pretrained graph extracts the features vector of the input image, it is passed to some classifier that actually runs the last step of the classification. I have decided to compare these 6 different classifiers:

- Support Vector Machine (**SVM**)
- Extra Trees (**ET**)
- Random Forest (**RF**)

- K-Nearest Neighbor (**KNN**)
- Multi-Layer Perceptron (**ML**)
- Gaussian Binomial (**GNB**)

2 Tools used

I have used TensorFlow (Python 3.6), because of its simplicity and its performance. I did not want to reinvent the wheel, so I have used the well-known *Scikit-Learn*, *Numpy*, *Matplotlib* libraries to preprocess the dataset and evaluate the classifiers. In the next sections I will give further details on the precise methods used. I have provided the source-code as a collection of Python files (extension .py). It is possible to view the source-code in the attached zipped file (it is also present in this GitLab repository).

3 The pretrained model

Since I have decided to use a pretrained model to extract features, I have downloaded and reused a checkpoint of the Inception v3 model [2]. Inception-v3 is trained for the ImageNet Large Visual Recognition Challenge using the data from 2012, and has a 78.0% top-1 accuracy and a 93.9% top-5 accuracy over the 1000 classes of Imagenet. In particular I have used the checkpoint of 12/05/2015, saving it in the 'pretrained' folder. This file (extension .pb) cannot be directly used, but it has to be imported, when initializing the TensorFlow session; and this is possible thanks to some TensorFlow functions: *ParseFromString* and *import_graph_def*; the first one simply transforms a binary file into a complex graph object, instead the second one actually makes the graph available to the current TensorFlow session.

4 The dataset

Both training and test images are taken from the Sapienza MarDCT dataset, a wide database of images and videos containing data coming from multiple video sources (fixed, zooming, and Pan-Tilt-Zoom cameras) and from different scenarios: in particular I have used the images of 24 different categories of boats navigating in the italian city of Venice and extracted by the ARGOS

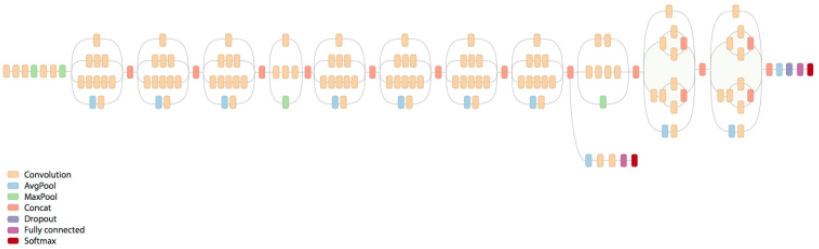

Figure 1: *Inception v3 architecture*

system. This dataset is already divided into training and test images: however, some preprocessing was needed in order to handle the different structure of the two sets of images and also because there were some multi-labeled images in the test set. In particular, the training set was organized in 24 folders, each containing a specific class of images; the name of each folder is the name of the class, i.e. the boat type; the class "water" actually contains a set of false positives images. The test set is provided, instead, as a unique (no folders!) collection of images; thanks to the file *ground_truth.txt*, however, it is possible to associate each file to a specific label. As already said, some images contain fragments of boats (these images are labeled as *partial*) or two or more boats of different types (multiple label). I have decided to simply skip these test images.

4.1 Preprocessing

First of all I have decided to limit the number of classes to 18 (out of the total 24), excluding all classes with less than 10 images. Figure 2 lists all the classes that I have included in the training phase. At the end of the preprocessing phase the dataset was made up by:

- **3810**[1] training images
- **1252**[2] test images

[1] cfr. figure 2 to see the number of images per label

[2] the number of test images is actually 1969, but as stated in section 4 I have filtered the images with partial or multiple labels and the false positives (i.e. *water* label)

4.1.1 Extract features

Once the pretrained graph was taken into main memory from disk, the images have been fed to a TensorFlow implementation of Inception v3, downloaded from the official repository, with the classification layer removed in order to produce a set of labeled feature vectors of 2048 floats: this *next-to-last* layer is a tensor called 'pool_3:0'. Inception is, indeed, a complex model (see figure 1) made up by a lot of submodules: namely the Inception modules. Each module implements a set of primitives (e.g. filtering, max pooling, etc.), so what I have done is simply remove the last layer of the network and collect the feature vectors. Once extracted, both the training and test features have been saved with pickle. In this way it has been possible to tune the classifiers, without repeating the feature extraction process, quite demanding in terms of resources and time.

Class	# samples
Alilaguna	113
Ambulanza	85
Barchino	112
Caorlina	1
Gondola	24
Lanciafino10m	22
Lanciafino10mBianca	484
Lanciafino10mMarrone	355
Lanciamaggioredi10mBianca	9
Lanciamaggioredi10mMarrone	5
Motobarca	215
Motopontonerettangolare	18
MotoscafoACTV	11
Mototopo	878
Patanella	279
Polizia	71
Raccoltarifiuti	94
Sandoloaremi	13
Sanpierota	5
Topa	78
VaporettoACTV	949
VigilidelFuoco	12
Water	907

Figure 2: *Boat classes from dataset MarDCT*

5 Classification

5.0.1 t-sne features

I have carried out dimensionality reduction on the 2048-D feature vector using t-distributed stochastic neighbor embedding (**t-SNE**) to transform the input features into a 2-D feature which is easy to visualize. The t-SNE is used as an informative step: if the same color (i.e. label) points are mostly clustered together there is a high chance that we could use the features to train a classifier with high accuracy. Figure 3 shows the result of this phase.

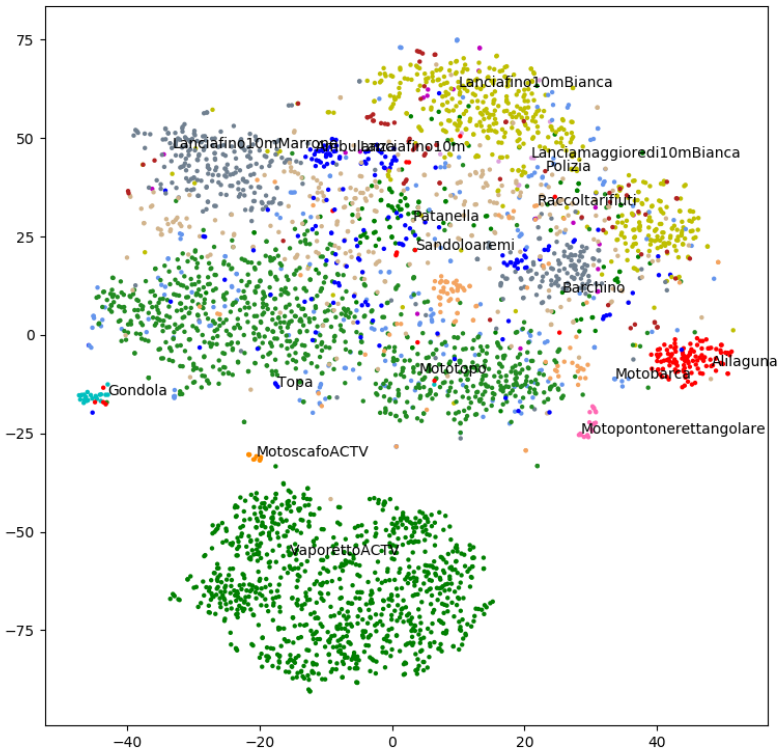

Figure 3: *t-SNE features extracted from Inception v3.*

5.1 Running the classifiers

As already stated in previous sections, the features of both training and test sets have been saved to disk: in this way running different classifiers does not require to load and extract features each time; moreover it has been possible to tune each classifier in a second moment. For each model (for the complete list see 1.1.2) I have run the proper Scikit-learn classifier. In most cases I have simply used the default configuration, but in some cases I have changed some parameter for tuning. For instance, as for SVM model, I have decided to use the LinearSVC classifier, that is similar to SVC with parameter kernel='linear', but implemented in terms of liblinear rather than libsvm, so it has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples. I have also set the *max_iter* parameter to 3000 (the default value is 1000) to improve by 0.1 both the accuracy and precision (see section 6). For a complete list of parameters passed to each classifier see the file *main.py*.

6 The results

I have computed the accuracy and the precision for each classifier, and I have finally plotted the confusion (and not normalized) matrices. They are shown in the following images. All the following results have been computed on the test set, meaning that the model was first trained on the training images and at the end both the accuracy and the precision have been computed on the test images. Remembering that:

$$\text{Accuracy} := \frac{|\text{true positives}|+|\text{true negatives}|}{|\text{instances}|}$$

$$\text{Precision} := \frac{|\text{true positives}|}{|\text{predicted positives}|}$$

I am going to show the results obtained for all the models: they are also plotted in figure 4.

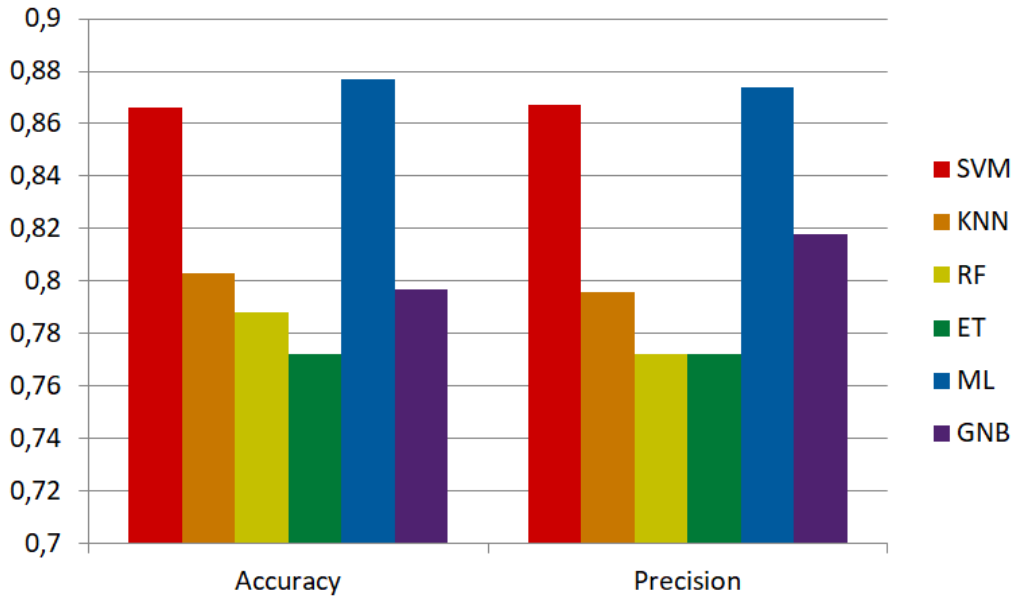

Figure 4: *Precision and accuracy of the classifiers*

6.1 Metrics

6.1.1 SVM

The accuracy was 86.6% and the precision was 86.7%

6.1.2 KNN

The accuracy was 80.3% and the precision was 79.6%

6.1.3 RF

The accuracy was 78.8% and the precision was 77.2%

6.1.4 ET

The accuracy was 77.2% and the precision was 77.2%

6.1.5 ML

The accuracy was 87.7% and the precision was 87.4%

6.1.6 GNB

The accuracy was 79.7% and the precision was 81.8%

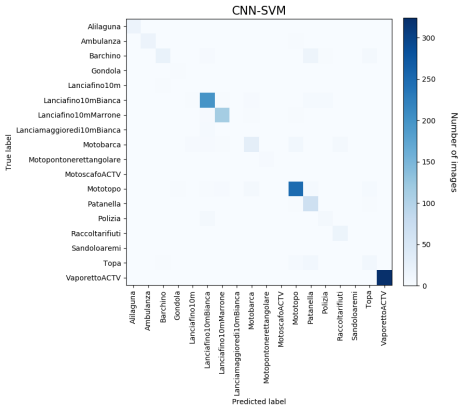

Figure 5: SVM confusion matrix

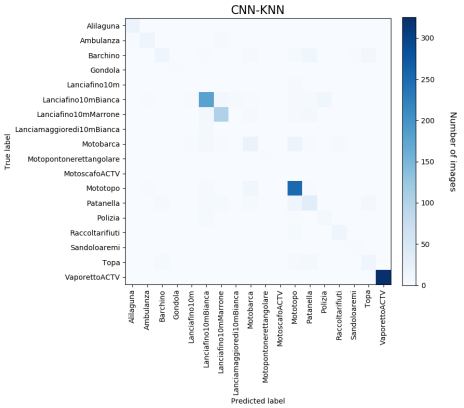

Figure 6: KNN confusion matrix

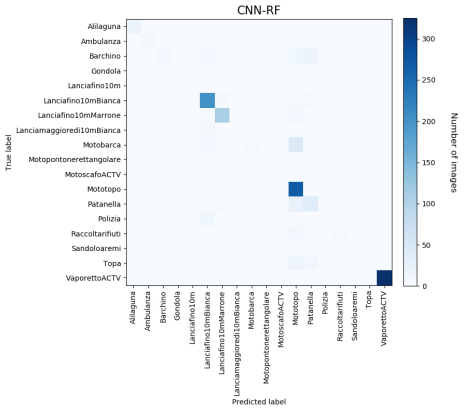

Figure 7: RF confusion matrix

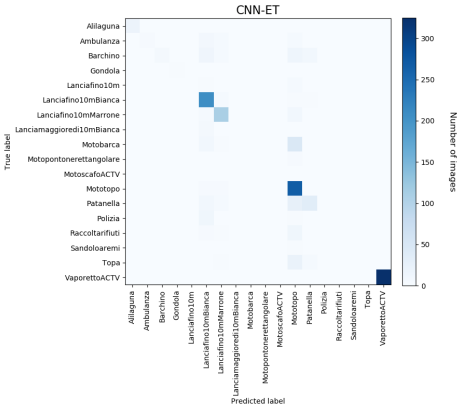

Figure 8: ET confusion matrix

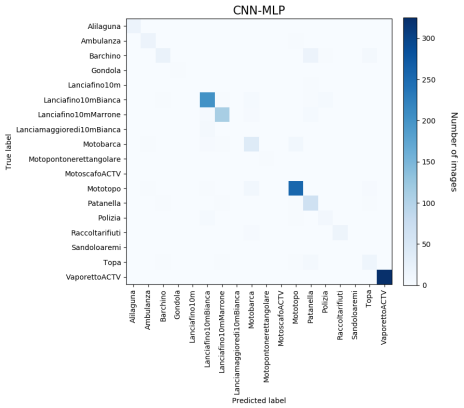

Figure 9: ML confusion matrix

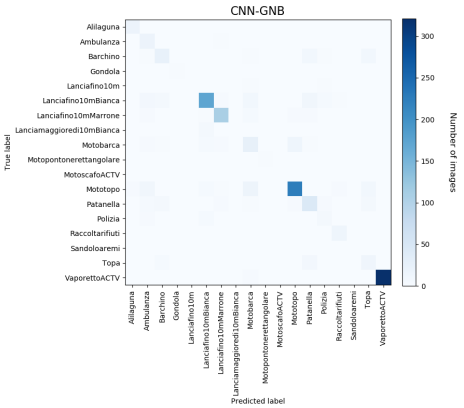

Figure 10: GNB confusion matrix

7 Conclusions

Figure 4 shows that Multi-Layer Perceptron (**MLP**) and Support Vector Machine (**SVM**) classifiers reach very good results in both accuracy and precision, with values higher than 0.86. The other classifiers, instead, reach levels of accuracy and precision under the 0.8 threshold. Generally speaking, every model is able to correctly classify the classes with lots of sample (e.g Vaporetto and Mototopo): and this fact strongly depends on the features extraction process, as can be seen in figure 3, where it is evident that some classes, such as Alilaguna, Gondola and Vaporetto, are clearly divided into clusters. So the major differences between the classifiers depend on the way they classify samples of classes with very low training images (e.g Sandaloaremi).

References

[1] Tom M. Mitchell *Machine Learning*.

[2] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna *Rethinking the Inception Architecture for Computer Vision*