Boat detection and classification on the ARGOS dataset

HW #2 ML Course 2018/2019 - MINR

Leonardo Sarra

14 December 2018

1 Introduction

ARGOS (Automatic & Remote GrandCanal Observation System) is a system that was developed in order to monitor the traffic of boats in Venice.
It is able to classify boats and identify critical situations and even illicit behaviors.
Those problems are common in Venice and can be resolved using machine learning, which could help the maintainers of ARGOS in keeping the system reliable over time.
This homework will discuss a simple implementation that is able to detect a particular class from images and classify them using the ARGOS dataset, furthermore, the performances of the given implementation will be discussed and analyzed.

2 Dataset

The dataset used to solve the detection and classification problem it's the one used by the ARGOS system, developed by the University of Rome Sapienza, it contains a collection of images, taken in Venice, depicting a particular boat.
The boats contained in the dataset belong to the following categories:

- People transport (Lancia, alilaguna, and vaporetto).
- General transport (Motobarca, mototopo and "raccolta rifiuti".
- Pleasure craft (Barchino, topa, and patanella)

- Rowing transport (Gondola and caorlina)
- Public utility (Police, firefighters, and ambulance)

This dataset is provided with a default separation in training and test set, the first is organized in folders (each for every type of boat) and the second, instead, make use of a ground truth file containing a map where filenames are associated with the type of the images.
Sadly these two datasets are not exactly consistent and so some changes, made at runtime, were required in order to use them without occurring in any problem later in the classification and validation phase.
The testing dataset, in particular, contains also some images with partial boats and some containing only water, these will come in handy later on for testing purposes.
All those images help us in training our classifier and so improving the performance of our implementation
Later one we will see the changes in the metrics when the way the feature is computed changes.

3 Tecnologies

In order to implement the learning algorithm, the parser and the validator for performance I decided to use the Python language.
In particular, I used Keras (which uses the Tensorflow framework) to extract features using a pre-trained neural network.
On top of that I used the implementation provided by scikit-learn for SVM which was very useful to learn from the extracted features, scikit-learn was also used to compute the confusion matrix given the results provided by the classifier.
And finally, I used pandas, seaborn and plotlib to render the confusion matrix as a nice looking figure.

3.1 Keras

Keras is a high-level API for Tensorflow which is extremely useful when building deep learning applications.
For this problem, I used the models provided by Keras and the pre-trained weights of ImageNet for feature extraction. The models used are the followings:

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
VGG16	528MB	0.713	0.901	138,357,544	23
VGG19	549MB	0.713	0.900	143,667,240	26

3.2 Neural networks for image classification

Neural networks are simple system made of multiple interconnected processing elements which process information by their dynamic state response to external inputs.
Neural networks are typically organized in layers, made up of a number of interconnected “nodes” containing an “activation function”. The input is given using the “input layer”, which communicates to one or more “hidden layers” where the actual processing is done via a system of weighted “connections”. The hidden layers then link to an “output layer” where the answer is given as output.
Neural networks are very useful for finding complex or numerous patterns that are hard to a human to extract and teach the machine to recognize.
In particular one of the problems in which the neural networks shine is the images analysis.
For example, a neural network can be used to assign a class to an input image, choosing from a fixed amount of possibilities.
This is not a trivial work because of the many factors that make this problem hard to resolve, like differences in viewpoint, scale or illumination, regarding the objects, there is the need to take into consideration the intra-class and the background clutter, and the possibility of having some sort of occlusion or deformation.

4 How to use

The project is available in my GitHub repo[1] and requires Tensorflow, Python 3.6.7 and other dependencies, that are written in "requirements.txt", to be installed.
It's is important to place the ARGOS testing and training set in the right location. After the requirements are met In order to start the project, you have to execute main.py.
Take into account that the execution time may vary depending on the hardware in which the program is run.
To use every feature of the project, you have to pass some parameters, whose meaning will be explained later.

```
usage: main.py [-h]
               [mode] [network] [split] [inputclass] [svmkernel]
               [default_testing]

Customization options using args

positional arguments:
  mode             Mode of operation that can be detection or classification
                   [detection,classification]
  network          Network architecture to do feature extraction [vgg16,
                   vgg19]
  split            Number of splits in k-fold cross validation (default is 3)
  inputclass       Class to analyze when doing detection (default is Water)
  svmkernel        Kernel to be used for SVM (default is the linear kernel)
  default_testing  Use default ARGOS testing to test the classifier[True,False]
```

5 Data loading

The data loader is the component used to load and organize the data from the dataset.
It is object oriented and it creates an object "BoatPhoto" for each unique photo contained in the dataset, this object is made of four variables:

[1]https://github.com/LithiumSR/venice_boat_ml

- “boatType”: A string that represents the class of the image, it can be a specific boat or an image containing only water.
- “filename”: The name of the image file.
- “features”: An array of arrays containing features extracted using the neural network.
- “testingSet”: A boolean value that represents when an image comes from the ARGOS default training or testing set.

The loader supports both the ARGOS training and testing set, in the first dataset the images are classified using the name of which there are on, in the second, instead, a ground truth file is used to know the type of the image are joined together by the loader, a new testing and training set will be computed by the validator that will be seen later on. Considering that there are some differences between both datasets I decided to consider “Snapshot acqua” class of the testing set as the “Water” of the training set and the “Mototopo corto” class of the testing set became a “Mototopo” class, moreover the ground truth files received some minor tweaks regarding the spelling of the classes to be consistent with the training set.

The data loader comes with two modes of operation which let the user decide how the type of the image is decided:

- Detection mode: This mode requires an input of a specific class, doing so the set will contain images whose type is equal to the input class or not-class (e.g: If I give as an input ”Mototopo” I will receive a set of BoatPhoto whose type is “Mototopo” and “NotMototopo”). In case no class is given as input the type of the images will be “Water” and “NotWater”.
- Classification mode: In this mode, each image has a type that represents the specific type of boat that is contained in the photo. For this mode ”Snapshot barca singola” and ”Snapshot barca multipla” of the testing set are ignored.

5.1 Features extraction

The DataLoader takes care of the feature extraction using a pre-trained neural network.
In particular, it supports VGG network architecture.
This network architecture is characterized by its simplicity, using only $3 * 3$ convolutional layers stacked on top of each other in increasing depth. Reducing volume size is handled by max pooling. Two fully-connected layers, each with 4,096 nodes are then followed by a softmax classifier.
There are two variants of VGG, one with 16 weight layers called VGG16 and one with 19 weight layers named VGG19.

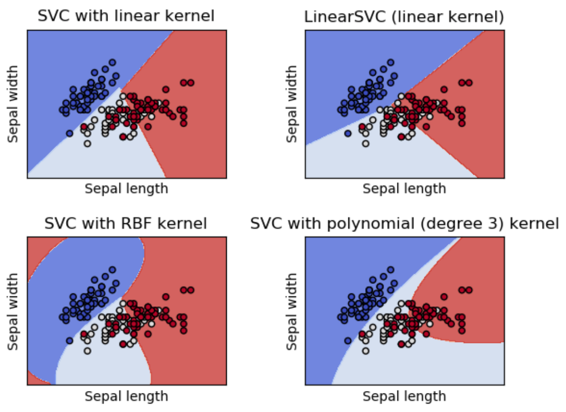
The VGG architecture is quite slow to train and its weight are quite large (in terms of disk/bandwidth).
In order to extract the features of the boat I used pre-computed weights of the VGG network on the ImageNet images database.
The ImageNet project is a large visual database designed for use in visual object recognition software research.
Doing so I was able to skip the training of the network using the images of the boats and still get good results regarding my detection/classification problem.
It is important to notice that in order to improve the quality of the features extracted the image was subject to preprocessing and to keep the computational time low it was resized to 224x224.
The features that are obtained using the neural networks are a multidimensional matrix which will be later "flatten" to fit our SVM classifier.

Listing 1: Example of feature extraction

```
from keras.preprocessing import image
from keras.applications.vgg16 import VGG16
from keras.applications.vgg19 import VGG19
import numpy as np
def extractFeature( file,network):
    img = image.load_img( file, target_size=(224, 224)
    img_data = image.img_to_array(img)
    img_data = np.expand_dims(img_data, axis=0)
    if self.network == VGG16:
              from keras.applications.vgg16 import preprocess_input
              img_data = preprocess_input(img_data)
             else:
            from keras.applications.vgg19 import preprocess_input
          img_data = preprocess_input(img_data)
     vgg16_feature = self.model.predict(img_data)
```

6 Data Learning

The Data Learner object contains a simple SVC implementation that will be used as a supervised classifier.
It takes as input the dataset on which the training will happen, it also takes as input the name of the kernel that has to be used (default is "linear").
If the input kernel is linear, the learner will use an optimized SVM for linear SVMs named LinearSVC provided by scikit-learn whose training time is roughly $O(m * n)$. This is better than using the linear kernel on the default SVC implementation of scikit-learn because it performs much worse than the LinearSVC, the time required is roughly $O(m^2 * n)$ and $O(m^3 * n)$. After the learning phase is completed the SVM classifier can be used to classify unknown instances of images.
The linear kernel can be used only when the features are linearly separable if that is not possible other kernels are needed resulting in higher complexity and so computational time. The saved time can be used to train with more examples or perform more exhaustive validation to optimize parameters. Later on, we will check the performance of a linear kernel and an RBF kernel (with default parameters) on the detection and classification problem.



6.1 SVM

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection.
The advantages that characterize SVM classifiers are:

- Effective in high dimensional space, even when the number of dimensions is greater than the number of samples.
- Memory efficiency, thanks to the use of a subset of training points in the decision function, called support vectors.
- Versatility provided by the different kernels, some of which are provided by default by some libraries, and the ability to create custom ones.
- It works really well with a clear margin of separations.

The disadvantages of support vector machines include:

- SVMs does not directly provide probability. It has to use five-fold cross-validation that is rather expansive on performance.
- If the number of features is much higher than the number of samples, it is very important to choose the kernel function wisely or you risk to go in an over-fitting situation.
- It doesn't perform well when classes are overlapping.

7 Data classification and validation

In this case, the classification of unknown instances is way easier when compared to what was presented in the report "Malware detection and classification".
To classify an instance you have to have an instance of the Boat Learner and call its classifier like that:

```
prediction = learner.classifier.predict(features) //returns predicted class
```

At this point, we have everything that is needed to make some tests regarding the performance of the implementation.
The validation is a crucial job, it is needed in order to understand if the algorithm is working and how well the predictive model will perform in practice; it has to be done preventing overfitting (models closely explain a training data set, but fail to generalize) and selection bias.
For this reason, I decided to use K-fold cross-validation, a technique that partition a sample of data into complementary subsets, performing the analysis on the training set, and validating the analysis on the validation/testing set.
Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining $k-1$ subsamples are used as training data.
This technique involves k rounds, at each round the retained set is different and so also the remaining $k-1$ subsets used for training. In the end, the results will be calculated using the average of the metrics found at each of the k iterations.

The validator that was developed has two modes of operation:

- Classification mode: It validates the performance for the classification problem using k-fold cross-validation on a set that is computed by joining the ARGOS testing and training set. It offers the possibility to use the ARGOS training set to train SVM and the testing set to validate it, this option was made available to test the performance of the implementation against the results provided in the ARGOS paper.
- Detection mode: It validates the performance for the detection problem using k-fold cross-validation on a set obtained by joining ARGOS testing and training set. Like in the classification mode, this one too

has the option to use the ARGOS training set to train the SVM and the testing one to validate the classifier.

Regarding the detection mode I decided to compute the following metrics:

- Accuracy is a metrics representing $\frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions}$
- Precision is the fraction of retrieved documents that are relevant to the query. It can be expressed like $\frac{|\{Relevant\ Documents\}| \cap |\{Retrived\ Documents\}|}{|\{Retrived\ Documents\}|}$.
  If there are only two classes it can be useful to use another definition: $\frac{true\ positives}{true\ positives + false\ positives}$.
  In the case of the detection problem, the precision represents how many files are correctly classified as a particular type among those I considered as having that type.
- Recall is the fraction of the relevant documents that are successfully retrieved $\frac{|\{Relevant\ Documents\}| \cap |\{Retrived\ Documents\}|}{|\{Relevant\ Documents\}|}$. If there are only two classes it can be expressed like $\frac{true\ positives}{true\ positives + false\ negatives}$.
  In case of the detection problem, it represents how many entries with a particular type were detected among those in the test set.
- False positive rate is expressed like the ratio between the number of negative events wrongly categorized as positive (false positives) and the total number of actual negative events $\frac{False\ positives}{False\ positives + true\ negatives}$.

It is important not to use only the accuracy metric because, in case of imbalanced datasets, it can provide a false sense of correctness.
In the classification mode the precision, recall, confusion and the f1-score are computed.
The confusion matrix, also known as error matrix, is a specific table layout that allows visualization of the performance of an algorithm; each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa).
This makes very easy to check when the implementation is confusion two classes.
F1 score, instead, is the harmonic average of the precision and recall that conveys the balance between the precision and the recall.

8 Results

Using the objects above mentioned I tested the performance of the feature extraction using a particular neural network and the SVM classifier with a specific kernel.
Those tests were made considering both a scenario with k-fold cross-validation and one where the ARGOS sets are not merged or shuffled.

The first problem to be analyzed were the detection problem (Water - Not-Water) using the VGG16 network for feature extraction and SVC with a linear kernel, whose performance were as follow:

Metric	4-Fold	Default test set
Accuracy	0.99%	0.97%
Precision	0.98%	0.99%
Recall	0.97%	0.87%
FPR	0.004%	0.002%

Those results were promising, we got a low percentage of false negative and false positive.
Almost every image was correctly classified, so the implementation is able to easily distinguish the images of boats from images containing only water. When the specific default test set was used the metrics changed a bit, there was a rise of false negative but nothing to worry about, it probably was caused by the distribution of the entries of the default ARGOS.

After VGG16 I then decided to check the metrics in the case I use VGG19, which were as follow:

Metric	4-Fold	Default test set
Accuracy	0.99%	0.96%
Precision	0.98%	0.99%
Recall	0.97%	0.86%
FPR	0.004%	0.002%

Given those metrics it is clear that for this particular detection problem changing the network architecture from VGG16 to VGG19 doesn't lead to any relevant changes.
Considering that there were no relevant changes I then tried to change the

kernel from linear to RBF expecting the metrics to change.
Those are the results when using the RBF kernel on a detection problem with VGG16 and VGG19:

Metric	VGG16 4-Fold	VGG16 - Default	VGG19 4-Fold	VGG19 - Default
Accuracy	0.87%	0.84%	0.87%	0.84%
Precision	0.37%	0.26%	0.38%	0.27%
Recall	0.99%	0.99%	0.99%	0.99%
FPR	0.13%	0.16%	0.13%	0.16%

The results were mediocre, the RBF is underperforming when compared to the linear kernel, it is susceptible to classify negative instances as positive.

RBF wasn't a good choice from the start because it is not ideal in a case when the number of features is higher than the number of observations, it is still possible to make it perform better (and reach the performance of the linear kernel) by customizing its parameters:

- The gamma parameter defines how far the influence of a single training example reaches, it can be seen as the inverse of the radius of influence of samples selected by the model as support vectors.
- The C parameter, instead, trades off correct classification of training examples against maximization of the decision functions margin.

If we are interested in getting similar results to the linear kernel case using RBF we have to set gamma = inf.
For this problem, the linear kernel is better suited because the problem is linear separable and because the optimization problem for a linear kernel is much faster.

I also tried the classification mode, ignoring, as said before, the following classes: "Snapshot barca singola" and "Snapshot barca multipla".
As before I tried running the implementation using k-fold first and the default testing set later.
The results followed the trend that was already seen in the detection problem, results are good and almost every type of boat is classified correctly.
The classification performs in a similar way both on VGG16 and VGG19, the first performing slightly better.

The confusion matrix in the default testing set highlights some minor issues when classifying "Patanella", which is often confused as a "Barchino", and the "Motopo" that is confused with "Lanciamaggioredi10metriMarrone" when VGG19 is used.
Those issues are not visible when using k-fold, this is probably due to the fact that the default set is not particularly balanced, resulting in worse performance.
I then tried using the RBF kernel on the classification problem with VGG16 and the metrics followed a similar trend to what we have seen for the detection resulting in mediocre performance in classifying almost every type of boat.
In particular the results showed that in case a particular boat type has a low number of entries in the dataset, it is usually confused with another type of boat more easily than what was saw from the results using the linear kernel, so the only classes that were not confused are the "Water" and "VaporettoACTV" one.
At this point, it's clear that using RBF, in this detection/classification problem, without tweaking its parameters, is not preferable considering the time complexity and the mediocre results and that there not much difference between VGG16's performance and VGG19's one when those architectures are applied to this problem.

Class	precision	recall	f1-score	support
Polizia	0.35	0.40	0.38	15
Patanella	0.64	0.81	0.71	74
VaporettoACTV	1.00	1.00	1.00	325
Cacciapesca	0.00	0.00	0.00	0
Sanpierota	0.00	0.00	0.00	0
Barchino	0.68	0.25	0.37	51
Ambulanza	0.80	0.73	0.76	22
Gondola	0.50	0.33	0.40	3
Raccoltarifiuti	0.77	0.89	0.83	19
Motopontonerettangolare	1.00	0.67	0.80	3
Lanciafino10mMarrone	0.86	0.87	0.87	125
Caorlina	0.00	0.00	0.00	0
Lanciafino10mBianca	0.87	0.89	0.88	217
Mototopo	0.93	0.95	0.94	284
Alilaguna	1.00	0.89	0.94	19
Sandoloaremi	1.00	0.33	0.50	3
Lanciafino10m	0.00	0.00	0.00	7
Topa	0.60	0.31	0.41	29
Lanciamaggioredi10mMarrone	0.00	0.00	0.00	0
Motobarca	0.55	0.56	0.55	59
VigilidelFuoco	0.00	0.00	0.00	0
MotoscafoACTV	1.00	1.00	1.00	1
Lanciamaggioredi10mBianca	0.00	0.00	0.00	6
Water	0.95	0.99	0.97	420
avg / total	0.88	0.88	0.88	1682

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.

Table 1: Results by validation on the default test set using VGG19 and linear kernel for the classification problem

class	precision	recall	f1-score	support
Polizia	0.73	0.47	0.57	86
Patanella	0.72	0.78	0.75	353
VaporettoACTV	0.99	1.00	1.00	1274
Cacciapesca	0.57	0.24	0.33	34
Sanpierota	0.00	0.00	0.00	5
Barchino	0.59	0.60	0.59	163
Ambulanza	0.84	0.86	0.85	107
Gondola	0.86	0.70	0.78	27
Raccoltarifiuti	0.89	0.78	0.83	113
Motopontonerettangolare	1.00	0.90	0.95	21
Lanciafino10mMarrone	0.90	0.92	0.91	480
Caorlina	0.00	0.00	0.00	1
Lanciafino10mBianca	0.84	0.93	0.89	701
Mototopo	0.92	0.95	0.94	1162
Alilaguna	0.99	0.96	0.98	132
Sandoloaremi	0.60	0.38	0.46	16
Lanciafino10m	0.33	0.14	0.20	29
Topa	0.71	0.55	0.62	107
Lanciamaggioredi10mMarrone	0.00	0.00	0.00	5
Motobarca	0.68	0.57	0.62	274
VigilidelFuoco	0.00	0.00	0.00	12
MotoscafoACTV	1.00	0.58	0.74	12
Lanciamaggioredi10mBianca	0.00	0.00	0.00	15
Water	0.96	0.99	0.98	1327
avg / total	0.89	0.90	0.89	6456

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.

Table 2: Results by 4-Fold cross-validation on the full dataset using VGG19 and linear kernel for the classification problem

	precision	recall	f1-score	support
Motobarca	0.56	0.54	0.55	59
Lanciafino10mBianca	0.88	0.88	0.88	217
Mototopo	0.93	0.96	0.94	284
Sandoloaremi	0.50	0.33	0.40	3
Alilaguna	1.00	0.95	0.97	19
Raccoltarifiuti	0.81	0.89	0.85	19
Lanciamaggioredi10mMarrone	0.00	0.00	0.00	0
Cacciapesca	0.00	0.00	0.00	0
Polizia	0.50	0.47	0.48	15
Topa	0.59	0.34	0.43	29
Patanella	0.66	0.82	0.73	74
Sanpierota	0.00	0.00	0.00	0
Barchino	0.71	0.29	0.42	51
Gondola	0.50	0.33	0.40	3
Lanciamaggioredi10mBianca	0.00	0.00	0.00	6
VaporettoACTV	1.00	1.00	1.00	325
MotoscafoACTV	1.00	1.00	1.00	1
Lanciafino10mMarrone	0.87	0.90	0.88	125
Motopontonerettangolare	1.00	0.67	0.80	3
Water	0.95	0.99	0.97	420
Caorlina	0.00	0.00	0.00	0
Ambulanza	0.89	0.77	0.83	22
VigilidelFuoco	0.00	0.00	0.00	0
Lanciafino10m	0.00	0.00	0.00	7
avg / total	0.88	0.89	0.88	1682

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.

Table 3: Results by validation on the default test set using VGG16 and linear kernel for the classification problem

	precision	recall	f1-score	support
Motobarca	0.68	0.59	0.64	274
Lanciafino10mBianca	0.86	0.94	0.90	701
Mototopo	0.91	0.95	0.93	1162
Sandoloaremi	0.58	0.44	0.50	16
Alilaguna	0.98	0.94	0.96	132
Raccoltarifiuti	0.88	0.81	0.84	113
Lanciamaggioredi10mMarrone	0.00	0.00	0.00	5
Cacciapesca	0.47	0.26	0.34	34
Polizia	0.69	0.53	0.60	86
Topa	0.70	0.52	0.60	107
Patanella	0.74	0.78	0.76	353
Sanpierota	0.00	0.00	0.00	5
Barchino	0.65	0.61	0.63	163
Gondola	0.90	0.67	0.77	27
Lanciamaggioredi10mBianca	0.33	0.07	0.11	15
VaporettoACTV	1.00	1.00	1.00	1274
MotoscafoACTV	1.00	0.75	0.86	12
Lanciafino10mMarrone	0.90	0.93	0.91	480
Motopontonerettangolare	1.00	0.86	0.92	21
Water	0.96	0.99	0.98	1327
Caorlina	0.00	0.00	0.00	1
Ambulanza	0.87	0.84	0.85	107
VigilidelFuoco	0.17	0.08	0.11	12
Lanciafino10m	0.45	0.17	0.25	29
avg / total	0.89	0.90	0.89	6456

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.

Table 4: Results by 4-Fold cross validation using VGG16 and linear kernel

	precision	recall	f1-score	support
Sandoloaremi	0.00	0.00	0.00	3
Raccoltarifiuti	0.00	0.00	0.00	19
Sanpierota	0.00	0.00	0.00	0
Ambulanza	0.00	0.00	0.00	22
Mototopo	0.70	0.76	0.73	284
Motobarca	0.14	0.02	0.03	59
VigilidelFuoco	0.00	0.00	0.00	0
VaporettoACTV	0.99	0.95	0.97	325
Topa	0.00	0.00	0.00	29
Lanciamaggioredi10mMarrone	0.00	0.00	0.00	0
Barchino	0.75	0.06	0.11	51
Alilaguna	0.40	0.89	0.55	19
Patanella	0.16	0.50	0.24	74
Gondola	0.00	0.00	0.00	3
Motopontonerettangolare	0.00	0.00	0.00	3
Lanciafino10mMarrone	0.51	0.29	0.37	125
Lanciafino10m	0.00	0.00	0.00	7
Cacciapesca	0.00	0.00	0.00	0
Lanciafino10mBianca	0.53	0.88	0.66	217
MotoscafoACTV	0.00	0.00	0.00	1
Lanciamaggioredi10mBianca	0.00	0.00	0.00	6
Polizia	0.00	0.00	0.00	15
Water	1.00	0.82	0.90	420
Caorlina	0.00	0.00	0.00	0
avg / total	0.70	0.69	0.67	1682

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.

Table 5: Results by validation on the default test set using VGG16 and RBF kernel for the classification problem

	precision	recall	f1-score	support
Sandoloaremi	0.00	0.00	0.00	16
Raccoltarifiuti	0.00	0.00	0.00	113
Sanpierota	0.00	0.00	0.00	5
Ambulanza	0.00	0.00	0.00	107
Mototopo	0.73	0.70	0.72	1162
Motobarca	0.24	0.38	0.29	274
VigilidelFuoco	0.00	0.00	0.00	12
VaporettoACTV	0.99	0.95	0.97	1274
Topa	0.00	0.00	0.00	107
Lanciamaggioredi10mMarrone	0.00	0.00	0.00	5
Barchino	0.47	0.09	0.15	163
Alilaguna	0.00	0.00	0.00	132
Patanella	0.22	0.47	0.30	353
Gondola	0.00	0.00	0.00	27
Motopontonerettangolare	0.00	0.00	0.00	21
Lanciafino10mMarrone	0.58	0.59	0.58	480
Lanciafino10m	0.00	0.00	0.00	29
Cacciapesca	0.00	0.00	0.00	34
Lanciafino10mBianca	0.45	0.75	0.57	701
MotoscafoACTV	0.00	0.00	0.00	12
Lanciamaggioredi10mBianca	0.00	0.00	0.00	15
Polizia	0.00	0.00	0.00	86
Water	0.98	0.91	0.94	1327
Caorlina	0.00	0.00	0.00	1
avg / total	0.66	0.67	0.65	6456

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.

Table 6: Results by 4-Fold cross-validation on the full dataset using VGG16 and RBF kernel for the classification problem

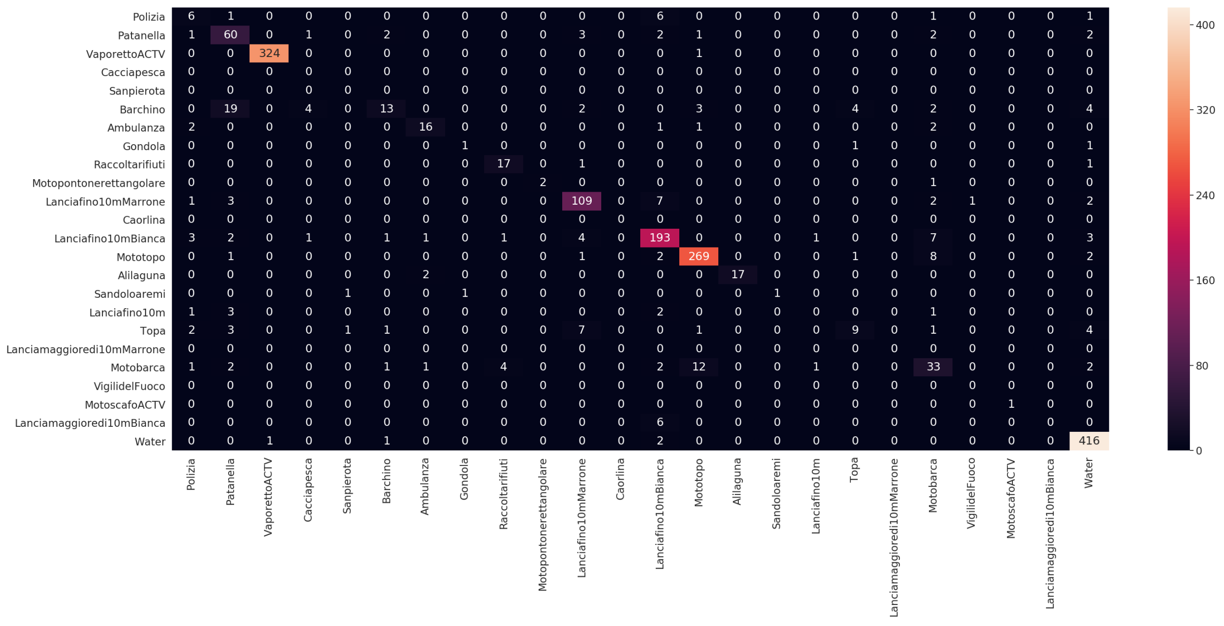


Figure 1: Confusion matrix of the classification problem using VGG19 and a linear kernel on the default test set.

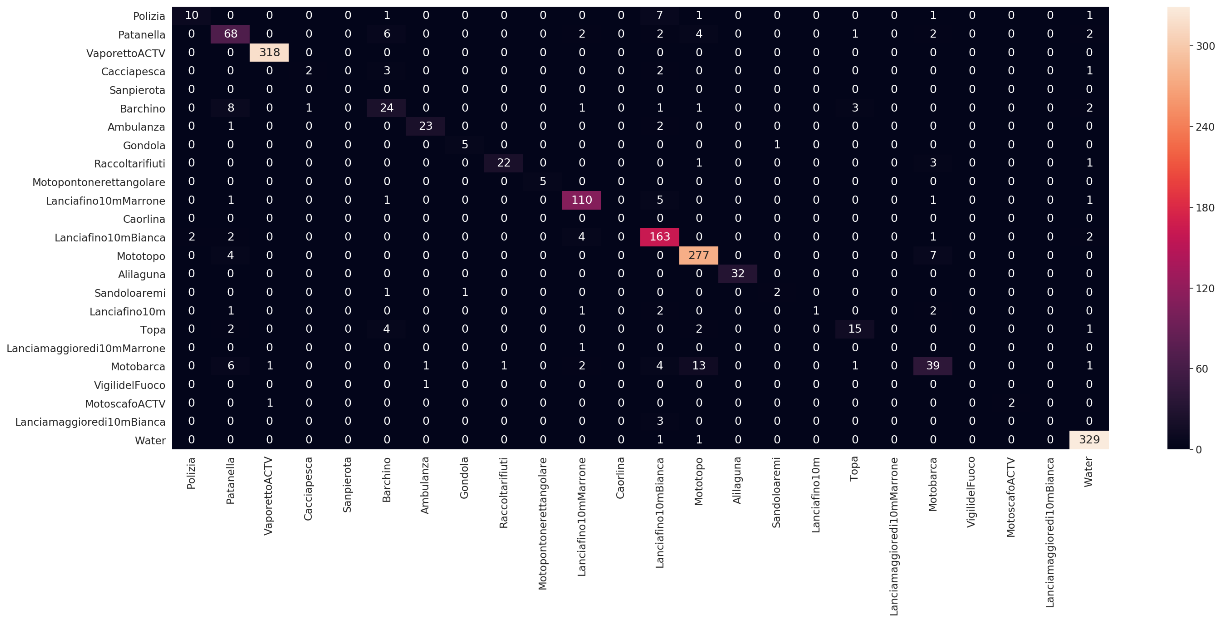


Figure 2: Confusion matrix of the classification problem using VGG19 and a linear kernel on a 4-fold of the dataset.

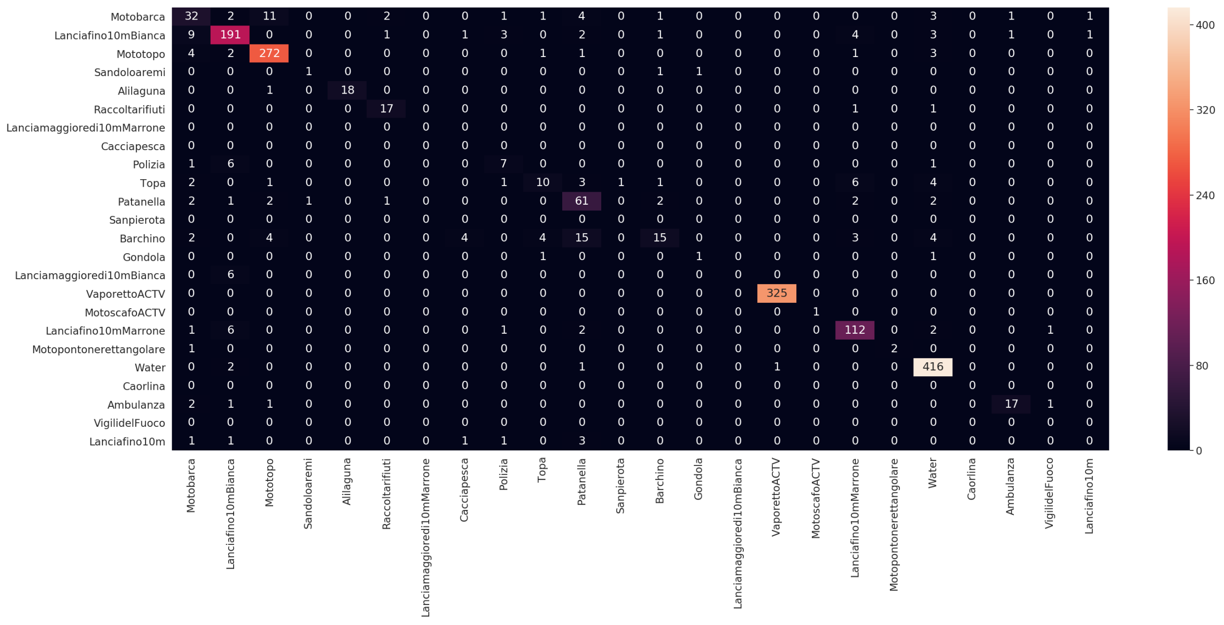


Figure 3: Confusion matrix of the classification problem using VGG16 and a linear kernel on the default test set.

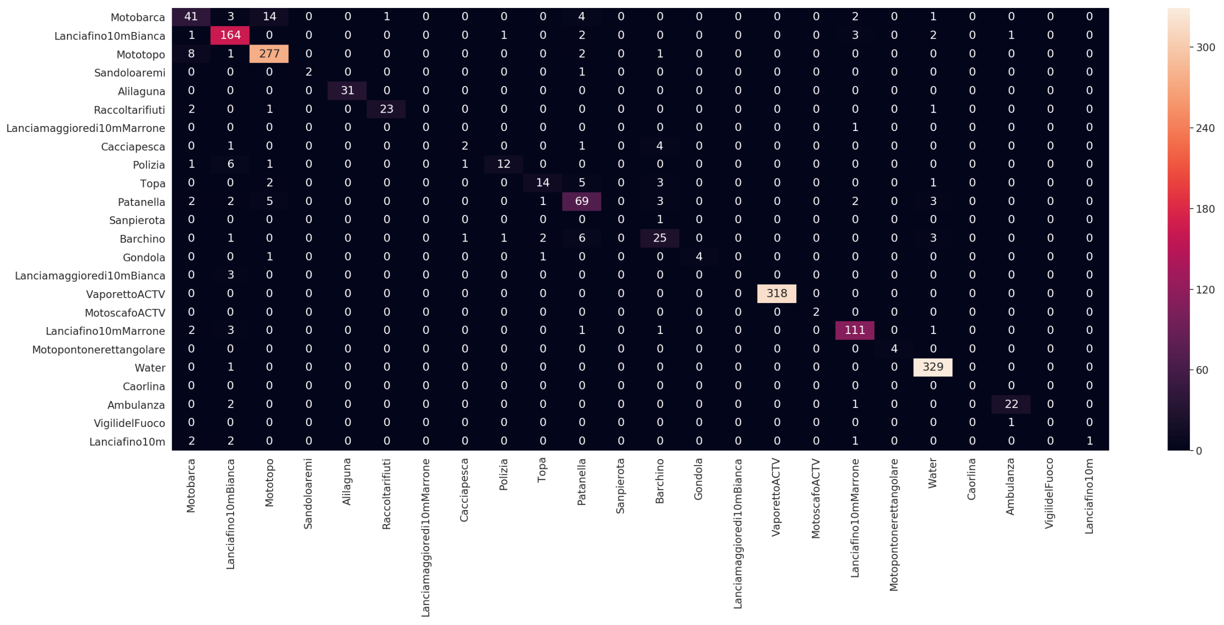


Figure 4: Confusion matrix of the classification problem using VGG16 and a linear kernel on a 4-fold of the dataset.

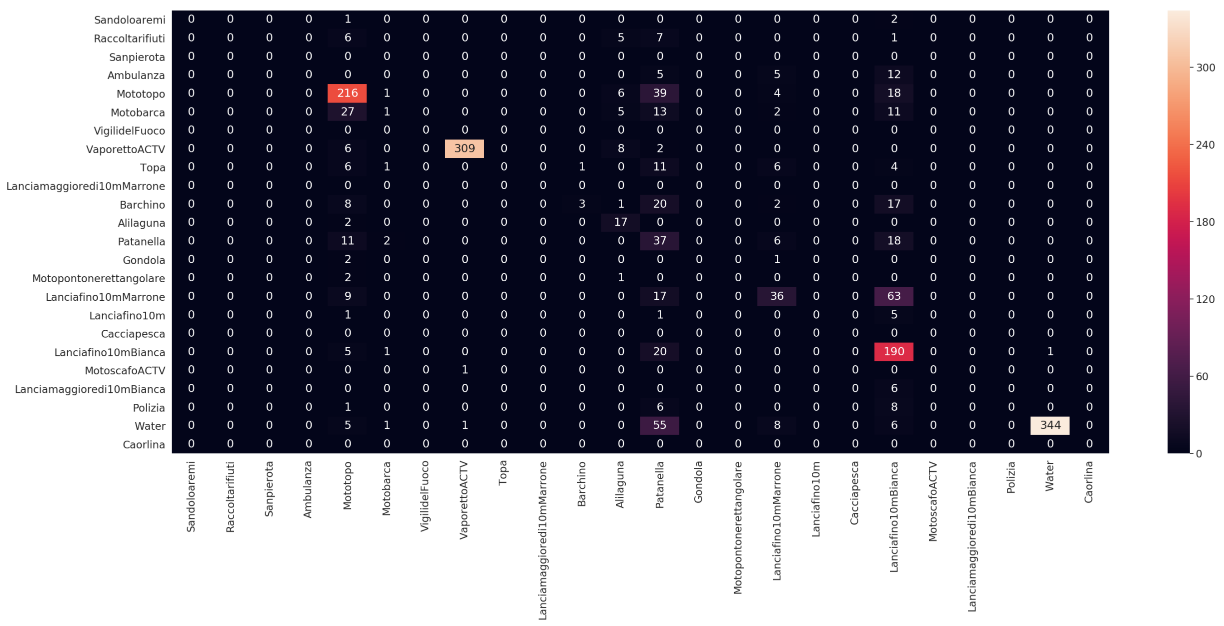


Figure 5: Confusion matrix of the classification problem using VGG16 and the RBF kernel on the default test set.

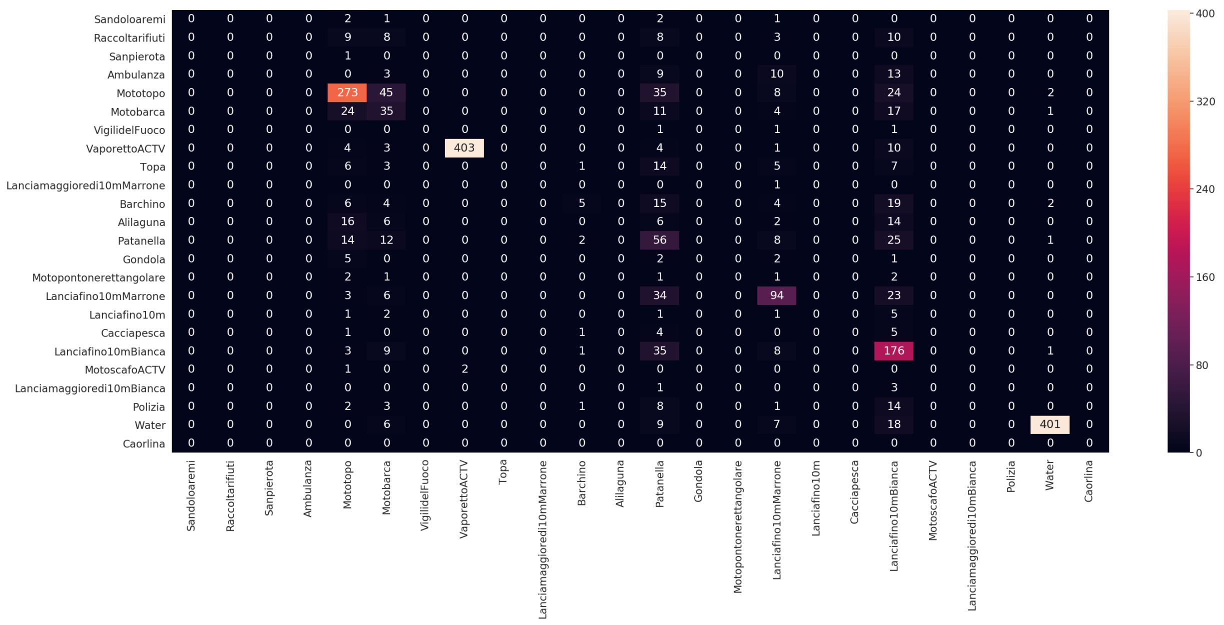


Figure 6: Confusion matrix of the classification problem using VGG16 and the RBF kernel on a 4-fold of the dataset.

References

[1] ARGOS-Venice Boat Classification - Domenico D. Bloisi, Luca Iocchi, Andrea Pennisi, Luigi Tombolini
`http://www.dis.uniroma1.it/~bloisi/papers/bloisi-vrs2015-draft.pdf`

[2] Scikit-learn documentation,
`https://scikit-learn.org/stable/documentation.html`

[3] ARGOS dataset - MarDCT,
`http://www.dis.uniroma1.it/~labrococo/MAR/`