Venice Boat Classification

HW2 - ML Sapienza

Giuliano Abruzzo
1712313

2018

Contents

1 Introduction

The goal of the second Machine Learning homework is to define an image classification problem for Venice boats, implement a classifier for the chosen problem and evaluate the results. I decided to analyze the problem of classifying n $<$ 24 specific classes of boats, with an approach based on CNN.

2 Dataset

For this homework, we are going to use the **MarDCT dataset** (Maritime Detection, Classification, and Tracking dataset). This dataset is composed of two files .rar: *sc5-2013-Mar-Apr.rar* that contains the training set and *sc5-2013-Mar-Apr-Test-20130412* that contains the test set.

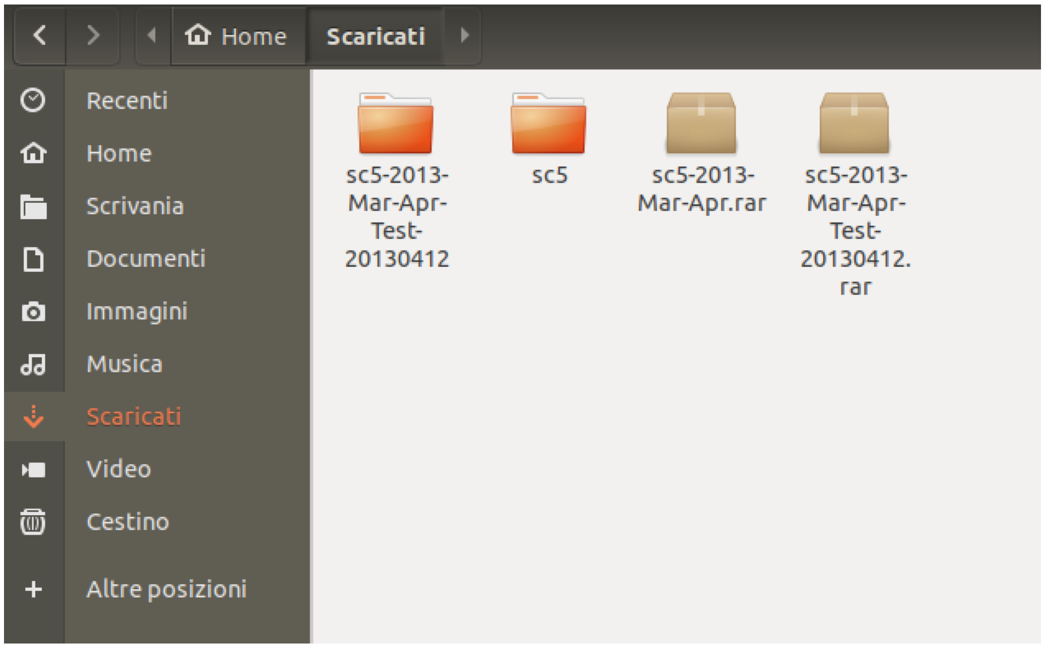


Figure 1: MarDCT data set

2.1 Training Set

The **training set** contains images from 24 different categories of boats navigating in the City of Venice (Italy). The *sc5-2013-Mar-Apr.rar* file contains a folder for each category. The jpeg files inside the folders are named according to the date, hour, and system track number. The folder *"Water"* contains false positives. The different **categories** are:

1. Alilaguna
2. Ambulanza
3. Barchino
4. Cacciapesca
5. Caorlina
6. Gondola
7. Lanciafino10m
8. Lanciafino10mBianca
9. Lanciafino10mMarrone
10. Lanciamaggioredi10m
11. Lanciamaggioredi10mBianca
12. Lanciamaggioredi10mMarrone
13. Motobarca
14. Motopontonerettangolare
15. MotoscafoACTV
16. Mototopo
17. Patanella
18. Polizia
19. Raccoltarifiuti
20. Sandoloaremi
21. Sanpierota
22. Topa
23. VaporettoACTV
24. VigilidelFuoco

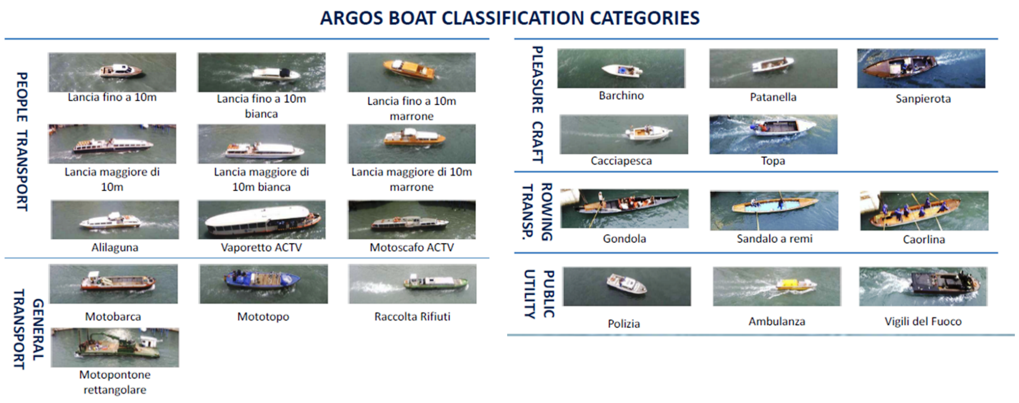
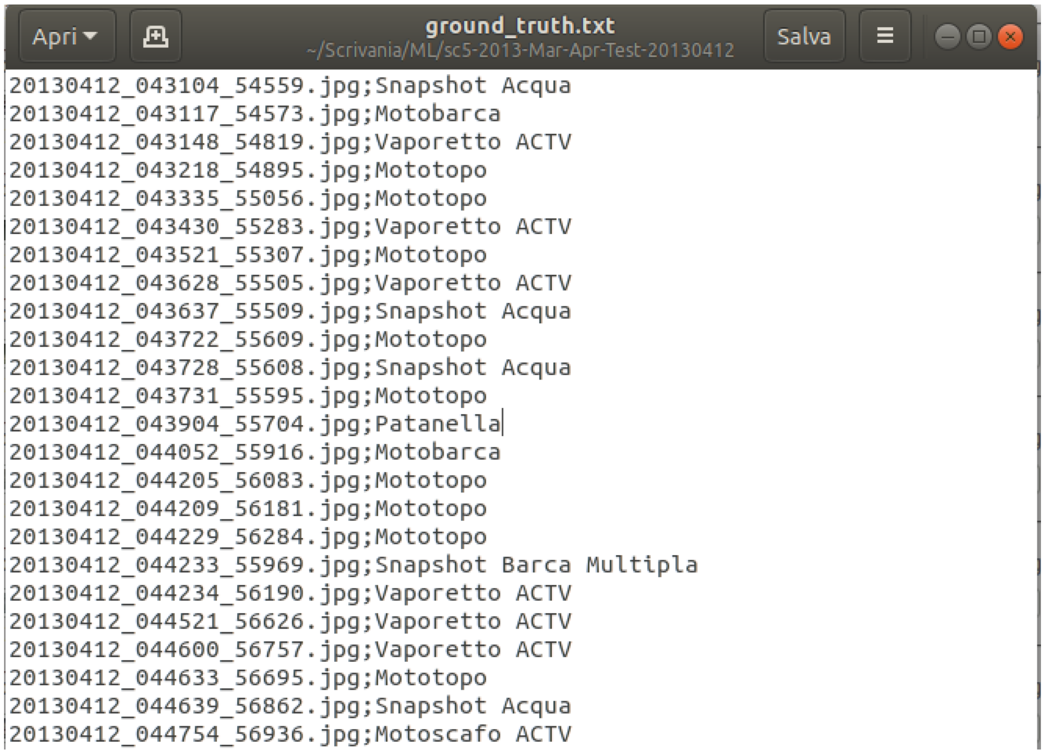


Figure 2: Categories of boats

2.2 Test Set

The **Test Set** contains images with ground-truth annotations in the format: *image-name;category* for example *20130412_044827_56976.jpg;Vaporetto ACTV*. The annotations are contained in the file ground_truth.txt.



```
20130412_043104_54559.jpg;Snapshot Acqua
20130412_043117_54573.jpg;Motobarca
20130412_043148_54819.jpg;Vaporetto ACTV
20130412_043218_54895.jpg;Mototopo
20130412_043335_55056.jpg;Mototopo
20130412_043430_55283.jpg;Vaporetto ACTV
20130412_043521_55307.jpg;Mototopo
20130412_043628_55505.jpg;Vaporetto ACTV
20130412_043637_55509.jpg;Snapshot Acqua
20130412_043722_55609.jpg;Mototopo
20130412_043728_55608.jpg;Snapshot Acqua
20130412_043731_55595.jpg;Mototopo
20130412_043904_55704.jpg;Patanella
20130412_044052_55916.jpg;Motobarca
20130412_044205_56083.jpg;Mototopo
20130412_044209_56181.jpg;Mototopo
20130412_044229_56284.jpg;Mototopo
20130412_044233_55969.jpg;Snapshot Barca Multipla
20130412_044234_56190.jpg;Vaporetto ACTV
20130412_044521_56626.jpg;Vaporetto ACTV
20130412_044600_56757.jpg;Vaporetto ACTV
20130412_044633_56695.jpg;Mototopo
20130412_044639_56862.jpg;Snapshot Acqua
20130412_044754_56936.jpg;Motoscafo ACTV
```

Figure 3: ground_truth.txt file

2.3 Reduced Dataset

Since I decided to classify only n (<24) specific classes of boats (ex. *Lanciafino10mBianca*, *Lanciafino10mMarrone*, *Mototopo*, *VaporettoACTV*), then the script implemented in python begins by **restricting** the original dataset, in fact, from the **sc5 folder,** the script will extract only the folders of the n classes of boats chosen and will place them in the folder called *ReducedTraining*. The same happens for the Test set folder where the file *ground_truth.txt* is opened and all the photos considered useful (excluding snapshots and photos belonging to specific classes of boats not considered by the problem) are placed in the folders of their classes in the folder *ReducedTest*.

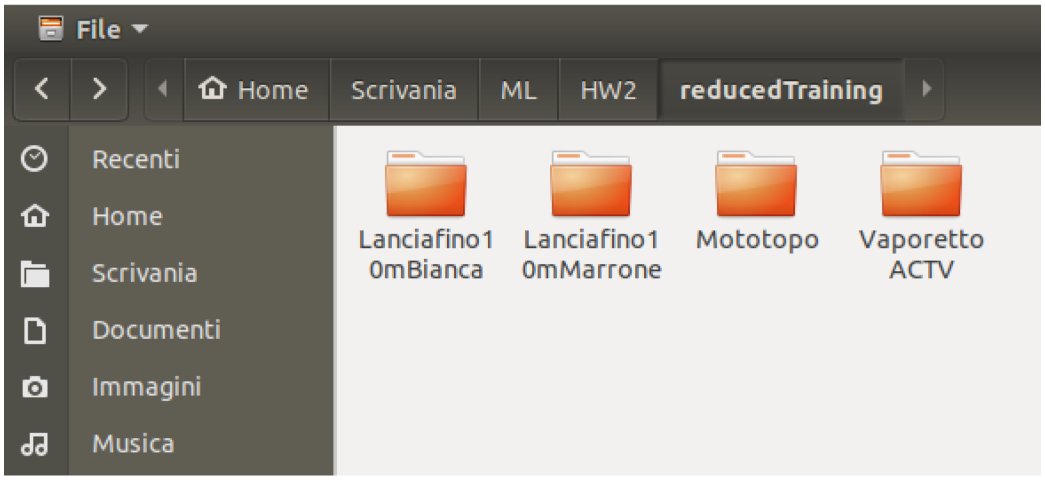


Figure 4: ReducedTraining Set folder

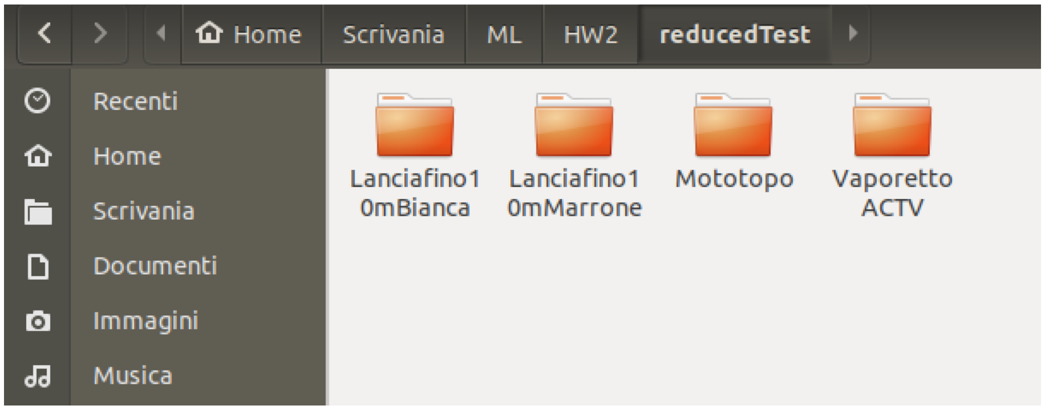


Figure 5: ReducedTest Set folder

3 CNN

The **CNN** (Convolutional Neural Network) used for this problem was created through the *Keras* python library, a high-level neural network API capable of running on top of *TensorFlow, CNTK, or Theano*. A common CNN is composed by: convolutional layer, pooling layer, and fully-connected layer. The **convolutional layer** applies a convolution operation to an input, the **pooling layer** performs a down-sampling of the input through a nonlinear function and the **fully-connected layer** is where classification happens. The CNN has a loss function that needs to be minimized to provide a good classification of the input.

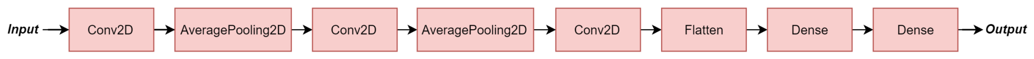
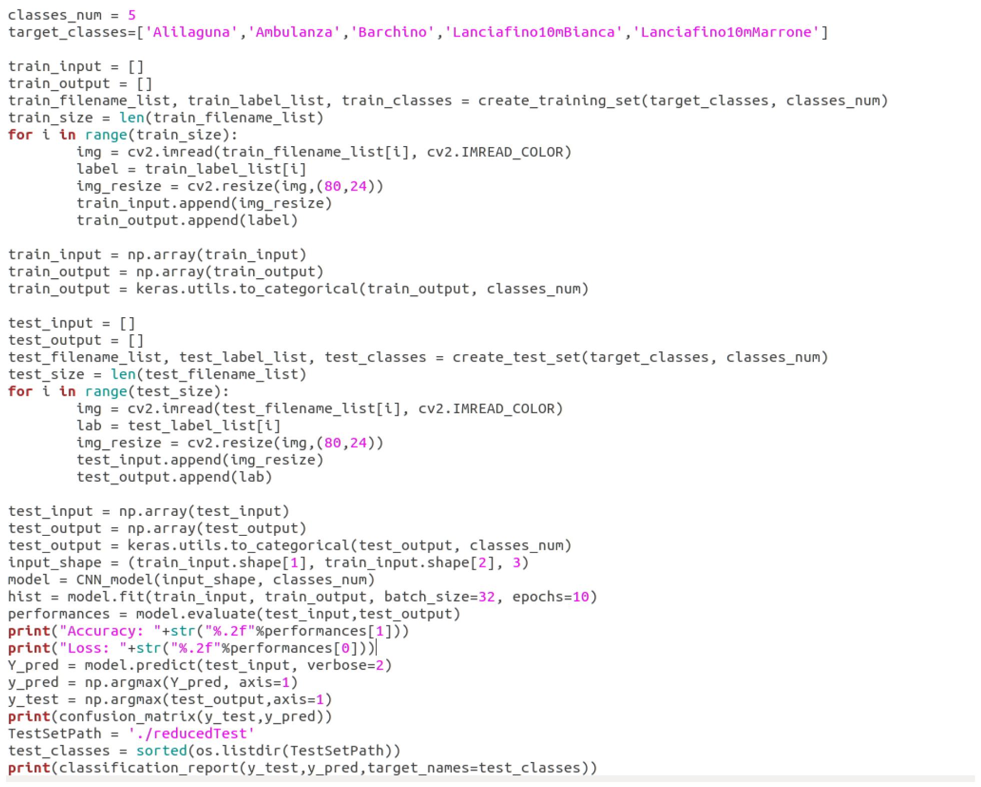


Figure 6: CNN model

The **CNN model** used for this problem is a sequential one, where at the beginning we have a Convolutional Layer 2D with 6 filters, created through the use of **Conv2D** function of keras, with *"tanh"* as activation function (hyperbolic tangent function) and with images size 1/10 of the original ones. The next layer added is a Pooling Layer 2D used for pooling operation on the results of the previous Convolutional Layer, this pooling layer was created through the use of **AveragePooling2D** and will halve the input in both spatial dimension. After two more layers equal to these two already described are added (Conv2D and Pooling), but with the difference that the second Conv2D layer will have 16 filters instead of the 6 filters of the first one. We add another Conv2D layer with 128 filters and a Flatten layer. In the end, there will be two Dense Layer, used for creating a fully-connected layer such that the first one Dense layer is connected with the set of nodes obtained from the Flatten layer with an activation function of *"tanh"* and the second one that has an activation function of *"softmax"*. This model is then compiled using **ADAM**, an algorithm for first-order gradient-based optimization of stochastic objective functions, as optimizer and with categorical cross-entropy as loss function and the metrics parameter is the accuracy. in the python script the model is constructed by calling the function: *CNN_model()*.

4 Code analysis

Once finished creating the two datasets through the functions: *create_training_set()* and *create_test_set()*, we need to fit the training dataset into the model of **CNN**. In order to do this, it is necessary to transform image datasets into **numpy** arrays. To do this we will use two functions, one from *openCV* and the other from *Keras*: *imread* and *to.categorical*. The first function, *imread*, is used to read the images of the dataset. These once read and resized will be inserted into a numpy array. The second function *to_categorical* will be used for converting the class vector to a binary matrix representation of the input,in fact this function performs one-hot encoding of the input. The results of the two function will then be used by the **fit** function of the *keras* library which will train our CNN model. These two functions are also used for the dataset test, but their results are used by the **predict** function of *keras*. Finally, the **performance** of our CNN model is evaluated.



```
classes_num = 5
target_classes=['Alilaguna','Ambulanza','Barchino','Lanciafino10mBianca','Lanciafino10mMarrone']

train_input = []
train_output = []
train_filename_list, train_label_list, train_classes = create_training_set(target_classes, classes_num)
train_size = len(train_filename_list)
for i in range(train_size):
        img = cv2.imread(train_filename_list[i], cv2.IMREAD_COLOR)
        label = train_label_list[i]
        img_resize = cv2.resize(img,(80,24))
        train_input.append(img_resize)
        train_output.append(label)

train_input = np.array(train_input)
train_output = np.array(train_output)
train_output = keras.utils.to_categorical(train_output, classes_num)

test_input = []
test_output = []
test_filename_list, test_label_list, test_classes = create_test_set(target_classes, classes_num)
test_size = len(test_filename_list)
for i in range(test_size):
        img = cv2.imread(test_filename_list[i], cv2.IMREAD_COLOR)
        lab = test_label_list[i]
        img_resize = cv2.resize(img,(80,24))
        test_input.append(img_resize)
        test_output.append(lab)

test_input = np.array(test_input)
test_output = np.array(test_output)
test_output = keras.utils.to_categorical(test_output, classes_num)
input_shape = (train_input.shape[1], train_input.shape[2], 3)
model = CNN_model(input_shape, classes_num)
hist = model.fit(train_input, train_output, batch_size=32, epochs=10)
performances = model.evaluate(test_input,test_output)
print("Accuracy: "+str("%.2f"%performances[1]))
print("Loss: "+str("%.2f"%performances[0]))
Y_pred = model.predict(test_input, verbose=2)
y_pred = np.argmax(Y_pred, axis=1)
y_test = np.argmax(test_output,axis=1)
print(confusion_matrix(y_test,y_pred))
TestSetPath = './reducedTest'
test_classes = sorted(os.listdir(TestSetPath))
print(classification_report(y_test,y_pred,target_names=test_classes))
```

Figure 7: Code

5 Performance Evalutation

We will perform a comparative analysis of our model in different scenarios. For each scenario, we will see the average loss, average accuracy, the confusion matrix and a table that contains precision, recall, and f1-score for each class of boat.

5.1 5 specific classes of boats

In this case the model uses only 5 classes of specific boats: [*'Alilaguna', 'Ambulanza', 'Lanciafino10mMarrone','Patanella','VaporettoACTV'*].

- **AVG Test Accuracy**: 0.89
- **AVG Test Loss**: 0.49

The confusion matrix is:

	Ambulanza	***Patanella***	***VaporettoACTV***	***Alilaguna***	***L.f.10mM.***
Ambulanza	**14**	**4**	**1**	**1**	**2**
Patanella	**7**	**48**	**0**	**0**	**19**
VaporettoACTV	**3**	**2**	**318**	**1**	**1**
Alilaguna	**3**	**0**	**1**	**14**	**1**
L.f.10mM.	**4**	**10**	**1**	**1**	**109**

The performace values are:

	Precision	***Recall***	***f1-score***
Ambulanza	**0.45**	**0.64**	**0.53**
Patanella	**0.75**	**0.65**	**0.70**
VaporettoACTV.	**0.99**	**0.98**	**0.98**
Alilaguna	**0.82**	**0.74**	**0.78**
L.f.10mM.	**0.83**	**0.87**	**0.85**

As we can see from the average values of accuracy and loss and from the values of recall and precision of the various classes we have a very good result.

5.2 5 different specific class of boats

In this case the model uses 5 different classes of specific boats: [*'Barchino', 'Lanciafino10mBianca', 'Motobarca','Mototopo','Raccoltarifiuti'*].

- **AVG Test Accuracy**: 0.74
- **AVG Test Loss**: 0.77

The confusion matrix is:

	L.f.10mB.	***RaccoltaRifiuti***	***Motobarca***	***Barchino***	***Mototopo***
L.f.10mB.	**188**	**5**	**3**	**5**	**16**
RaccoltaRifiuti	**0**	**10**	**3**	**0**	**6**
Motobarca	**8**	**1**	**15**	**3**	**32**
Barchino	**11**	**2**	**2**	**18**	**18**
Mototopo	**19**	**9**	**14**	**3**	**229**

The performace values are:

	Precision	***Recall***	***f1-score***
L.f.10mB.	**0.83**	**0.87**	**0.85**
RaccoltaRifiuti	**0.37**	**0.53**	**0.43**
Motobarca	**0.41**	**0.25**	**0.31**
Barchino	**0.62**	**0.35**	**0.45**
Mototopo	**0.76**	**0.84**	**0.80**

So in this scenario, we have a lower average accuracy value and a higher average loss value but still acceptable.

5.3 10 specific classes of boats

The last scenario that we will see is the scenario of 10 specific classes of boats: [*'Alilaguna', 'Ambulanza', 'Barchino', 'Lanciafino10mBianca', 'Lanciafino10mMarrone', 'Motobarca', 'Mototopo', 'Patanella', 'Raccoltarifiuti', 'VaporettoACTV'*].

- **AVG Test Accuracy**: 0.73
- **AVG Test Loss**: 0.92

The confusion matrix is:

	Ambulanza	*L.f.10mB.*	*Raccoltarifiuti*	*Patanella*	*Motobarca*	*VaporettoACTV*	*Barchino*	*Alilaguna*	*Mototopo*	*L.f.10mM.*
Ambulanza	7	5	0	0	1	1	2	1	3	2
L.f.10mB.	17	147	4	7	3	1	5	2	10	21
Raccoltarifiuti	0	0	9	1	2	0	0	0	7	0
Patanella	0	8	1	44	3	0	6	1	8	3
Motobarca	2	2	2	0	20	1	3	1	24	4
VaporettoACTV	4	2	0	0	0	313	0	2	2	2
Barchino	0	9	1	24	0	0	6	1	7	3
Alilaguna	1	1	1	0	1	1	0	12	1	1
Mototopo	1	8	6	3	25	1	5	3	215	7
L.f.10mM.	4	7	2	1	1	2	0	1	10	97

The performace values are:

	Precision	*Recall*	*f1-score*
Ambulanza	0.19	0.32	0.24
L.f.10mB.	0.78	0.68	0.72
Raccoltarifiuti	0.35	0.47	0.40
Patanella	0.55	0.59	0.57
Motobarca	0.36	0.34	0.35
VaporettoACTV	0.98	0.96	0.97
Barchino	0.33	0.22	0.20
Alilaguna	0.52	0.63	0.57
Mototopo	0.75	0.78	0.77
L.f.10mM.	0.69	0.78	0.73

Seeing the values of average accuracy, average loss and performance values we can say that our model also considering the increase in the number of classes continues to provide good results.