Classification of Fishing Activities using ResNet Neural Network from AIS Data

Adriano Mancini[1], Alessandro Galdelli[1], Mirko Simoni[1], and Lorenzo Medici[1]

[1]Università Politecnica delle Marche, Ancona, Italia

Abstract—**Given the really high number of fishing activities taking part all around the world, including the Mediterranean Sea as well, it is a really time-consuming and complex task the one of supervising and keeping in check all the vessel working every day and night.**

This means that too many time illegal activities such as the fishing in protected area or closed seasons could deeply damage the maritime life of the seas without the authorities ever been able to notice it, until it's too late

Thus, we exploited the advantages of a classification neural network rearranged and adapted for the specif task at hand, making it recognise several fishing activities from each other with acceptable performances.

I. Introduction

In the past decade the efficiency of neural networks applied for deep learning tasks has seen quite the improvement, and now such architectures are able to perform problem solving tasks like never before.

In our particular case of study, we're are now able to code neural networks able to forecast ship trajectory in the ocean, given a certain amount of reliable input data in form of time series, and assess what type of activity the ship is performing.

This is very important for the monitoring and identification of possible ship's illegal activity, such as fishing in restricted area or during closed seasons, which is very complex given the high number of vessel in the sea every day.

The data used are gathered from the Automatic Identification System (AIS), which is devised for recording and identifying all the different fishing sessions in the form of "pings".

We call "pings" the information created and sent by the ship electronic equipment to land's antennas every 5 minutes during navigation, from when the boat leaves the harbour until it comes back to it.

Those "pings" end up forming a very large dataset holding "raw" information about ship traveling such as speed, course, GPS position, harbour of departure and arrival, etc.

All of this is collected as ".csv" files by the provider, astrapaging.com [1], which our university bought the data from.

Our university also worked in collaboration with the National Research Council (CNR) in order to recognise all the different fishing sessions starting from the data bought from the provider.

Through consultations with CNR's experts, they managed to label all the sessions present in the ".csv" files holding 2017 fishing activities in the Mediterranean, making a distinction between 6 main fishing gear used by ships, since the number of other techniques used was neglectable.

Since it takes fat too much time for humans to track every ship in the ocean at all times, thanks to neural networks architectures it is instead possible to let learning algorithms elaborate ship routes, once they have been trained on a certain amount of known fishing activities, thus succeeding in forecasting ship activities with high enough reliability.

In this paper, we will discuss the reasons inspiring our work, the state of the art neural network chose for our purposes,the approach to dataset analysis, the methods applied to reshape the data, the experiments run on the network, the results and their meaning and possible applications of our results for future interests in forecasting networks.

II. Motivations and goals

Many of the recorded sessions were deemed unusable because of the absence of a number high enough of pings recorded between the start and the end of them.

This is linked to the likely misconduct of the sailors who, by turning off the electronic device on board during the fishing activity, made impossible to reconstruct the ship's movement and gear used.

For this reason, we decided to only study those sessions with 80% of the number of pings recorded out of the total number expected from the duration of the session, deeming the remaining sessions not fitting for classification and thus filtering those away from the dataset.

Another problem we had with the dataset was the high number of fishing sessions labeled as Single Boat Bottom Otter Trawls (OTB) type.

Since the OTBs sessions were hundreds of times more then the average of all other 5 types of gears, we needed to balance the dataset by randomly eliminate OTBs sessions, until reaching a number of the same order of magnitude of the other types of fishing sessions.

Otherwise, the network would have resulted as strongly biased in recognising only the OTB sessions as such.

Once solved these issues we had to study several neural network architecture to understand which one applied better for our classification tasks.

On the paper [2] is conducted a profound study and comparison between the latest state of the art neural networks designed for deep learning task.

Reading the given results, it appeared as the most suitable choice to select the ResNet network in order to elaborate our time series, since it proved to perform really well on all sorts of dataset of different nature.

This was true especially for Multi-Class time series forecasting, which is the type of data structure we had to use given that we had to perform the classification of sessions with more then one variable to elaborate (talking about speed, course, latitude,...).

Thus, we utilise the architecture shared on this GitHub repository which is suited for handling such task Our goals consist in:

- ”cleaning” the ”.csv” files from unusable data, making the network work with lighter datasets which are filtered and balanced
- constructing the dataset from ”.csv” files through Matlab scripts in order to randomize Excel’s rows choice for ”test” and ”training” data-set creation
- testing network performances on different data included in the time series
- reaching a certain accuracy and precision from the scripts simulating the forecasting of ship routes

III. Data manipulation

We mainly worked with three .csv files:

- One holding in each row a single fishing session performed by a particular ship, showing in each column all relative details such as date and time of activity, ID of the ship, start and end of the pings counted, harbour of arrival and departure, etc.
- Another manufactured through expert consulting, who recognized every single session making a distinction between every different type of fishing, labeling every session accordingly.
- The last one containing data such as speed, course, latitude, longitude and other sailing properties of the ships during their sessions recorded every 5 minutes.

We utilised the software ”pgAdmin 4” [3] and a python script in order to reshape our first 2 tables loaded in a PostgreSQL type of database on order to execute queries and eventually link the tables to a Matlab [4] function, which we’ll explain in the next section.

In particular we used ”PostGIS”, an open source, freely available, and fairly OGC compliant spatial database extender for the PostgreSQL Database Management System.

This extension adds spatial functions such as distance, area, union, intersection, and specialty geometry data types to the database.

In our case was especially useful to represent GPS’s position of ships as a geometry data type.

we performed one query to filter away sessions lacking enough number of pings in the first table, which is the information sent every 5 minutes to create each row of the file.

Afterwards, we also eliminated enough random sessions from the second file so that all the fishing types of the ground truth (the labels) were taken in similar number, before creating the train and test data-set for the neural network.

In the fist case we used a query which calculates the ratio between the number of pings actually recorded by the ship’s electronic equipment and the number expected, given the time of the session.

```sql
SELECT *
FROM public."Session_2017"
WHERE (endid-startid)/((((enddata-startdata)*1440)
    +((date_part('hour',(endtime-starttime)))*60)+(
    date_part('minute',(endtime-starttime))))/5)>
    0.7
;
```

Listing 1. Query cutting away every session with less then 79% of pings

For the second objective instead we performed few simple lines of code displayed in , given the much more simplicity of the task.

```python
import pandas as pd
from google.colab import files
from google.colab import drive
drive.mount('/content/gdrive')

%cd "/content/gdrive/My Drive/database_barche"
sgear = pd.read_csv("sessions_per_gear.csv", sep=';'
    )

sgear_balanced = sgear.drop(sgear[sgear.GEAR=='OTB'
    ].sample(frac=.985).index)

sgear_balanced.to_csv('gear_per_session_balanced.csv
    ')
```

Listing 2. Script operating balancing of OTBs sessions

Afterwards, we wrote a query on pgAdmin to cut away all sessions from the first table with a value under the column ”joined” different then 0, since they were previously established as ”bad” session in other works of studying.

```sql
SELECT *
FROM public."Session_2017"
WHERE joined = 0;
```

Lastly, we changed the values within the second table, merging 2 types of fishing into a single label, since they were actually the same kind of fishing.

```sql
UPDATE public."GroundT4"
SET gear=LL
WHERE gear = LLS AND gear = LLD;
```

From here, we joined the table with the sessions and the one holding the labels by the ID of the ships and the ID of the session, creating the dataset that is going to be used in Matlab.

Through a Matlab script, rearranged starting from the one created by the PhD student Alessandro Galdelli [**?**], we created a cycle for every session, we extrapolated the interested data from each ping cross referenced with the third table holding the actual measures and created a new data structure shaped as a time series.

Time series is a data structure created in order to make neural network perform classification tasks.

The idea is that the neural network is trained on 80% of the dataset selected randomly, meaning it learns how to distinguish one session from the other by detecting patterns along all the pings forming the different sessions.

By associating distinct patterns to the corresponding labels, the network is later tested on the remaining 20% of the dataset, trying to guess correctly the labels of the sessions given what the network has previously learned on the other dataset.

We can record wheter the network guesses are correct or not thanks to the labels present in the testing dataset.

In order to perform all this tasks, the time series are structured in 4 fields:

- trainlabels = vector of type double filled with a randomly selected 80% of the labels extrapolated from the joined table's column
- train = vector of type cell, each representing one of those 80% of sessions selected randomly, holding the interested data on separate rows, depending on the test running in each case (either speed, course, latitude, longitude, even derived measures such as acceleration,etc)
- testlabels = the remaining 20% of labels with the same data structure as trainlabels
- test = the remaining 20% of sessions with the same data structure as train

The Matlab scripts, copied in the appendix, shows the specifics steps in order to craft a Multi-class time series with the measures acceleration and course extracted from the table with the raw data.

Basically, it establishes a connection to the pgAdmin[3] server where we created the previous mentioned tables.

This way we could associate the first table to a Matlab variable and perform a query within the code.

It instantiate a variable with all the rows of sessions belonging to a specific ship each cycle.

Then through another cycle, it perform the query ”PostgreSQL” that changes each loop to select the data of a specific session defined by all its pings recorded in the relative table.

In this case, every yth session of the ith ship will be analysed through its acceleration and course, where acceleration is derived from the column speed.

Once stored the result of the query in the variable ”raw” it reshapes the columns in rows, it converts the string values of the measures in double type and transforms the speed from knots to km/h.

The next cycle calculates the acceleration of the ship from each speed.

Once we do this for each ship, we change the values of the column holding the 5 different labels of fishing type from string to double, otherwise the network can't process them, replacing the gear labels with numbers from 1 to 5 using ”for” loops.

The dataset is finally transformed in a Multi-class time series through the random partitioner that splits it in the training and testing sets of labels and data.

The .mat file that comes out of the function will be imported in the run time application Colab of google drive, where the ResNet network is also imported from GitHub.

This is the environment where the network will perform the classification of our time series.

IV. Data classification

The main characteristic of ResNets is the shortcut residual connection between consecutive convolutional layers.

The network is composed of three residual blocks followed by a GAP layer and a final softmax classifier whose number of neurons is equal to the number of classes in a dataset.

Each residual block is first composed of three convolutions whose output is added to the residual block's input and then fed to the next layer.

The number of filters for all convolutions is fixed to 64, with the ReLU activation function that is preceded by a batch normalization operation.

In each residual block, the filter's length is set to 8, 5 and 3 respectively for the first, second and third convolution.[2]

The ResNet architecture is shown in Figure 1.

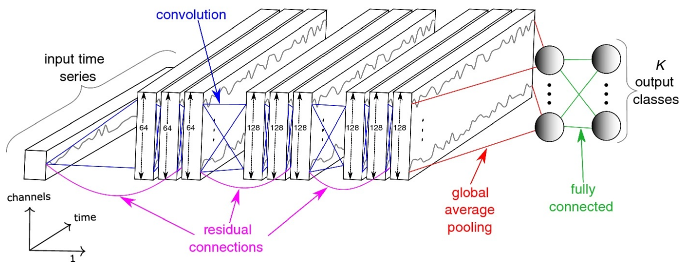

Figure 1. The Residual Network's architecture for time series classification

The entire network is built on Python libraries and on the imported GitHub repository above mentioned.

The structure and functioning of it is deeply explained on this READ ME. For the sake of our work we're gonna describe only the parts of the code that we altered in order to make it work on the Colab environment with our datasets.

We started by importing the GitHub repository, then we install all the requirement mentioned in the previous READ ME and proceed to alter the files accordingly to our tasks:

```
!git clone https://github.com/hfawaz/dl-4-tsc.git

%cd /content/dl-4-tsc/

!mkdir archives

!pip install tensorflow==2.1
!pip install tensorflow-gpu==2.1
!pip install -r /content/dl-4-tsc/utils/pip-
    requirements.txt
!pip3 install git+https://www.github.com/keras-team/
    keras-contrib.git

!python3 main.py mts_archive SC985 resnet _itr_8
```

Listing 3. Code wrote on Colab environment to import necessaries libraries and run the main code of the github repository

Secondly, we had to swap the function responsible for adjusting the cell's dimension in the dataset.

This was because the part of the code designed to work with multi-class time series accepts only same length time series as input.

The default function within the ”utils” of the repository is a cubic interpolation function, which is far too precise and heavy on the elaboration time given our specific datasets.

Thus, we decided to write a different function that simply fills the gaps of the smaller time series's cells with enough zeros to make them all equally long.

We tested filling the time series from the start and from the end of their vectors, which produced the same network accuracy and precision at the end of the runs, ensuring the order of such operation doesn't affects our results.

The code of the zero padding can be found in the appendix at the end of the paper.

Once we successfully changed the code, we ran the ResNet architecture on our dataset with a number high enough of iterations to make the results of the elaboration reliable.

The runs (with 6 or 8 iterations) usually take between 2 and 3 hours, so we tested only the following datasets on the network:

- SC985 = Multi-class time series dataset with 985 cells of speed and course's sessions
- SCLL985 = Same as before, but also with latitude and longitude
- CLL985 = Same as before but without speed
- AC985 = Same as before but with acceleration and course

```
# change this directory for your machine
root_dir = '/content/dl-4-tsc'
```

Listing 4. Directory holding the file main.py in order to run the code

```
MTS_DATASET_NAMES = ['SC985', 'SCLL985', 'CLL985', '
    AC985']
```

Listing 5. Constant declared within the utils.py file holding the names of the datasets used

The goal of using these files during our studying was to test how the network behaved when given different datasets.

We wanted to understand if the network would perform differently once changed the the type measures inside the time series, on the assumption that every class influenced differently the accuracy of the network.

V. Results

The performances of the ResNet have been all collected as measurements indicating the following properties of the network working with Multi-Class classification:

- Precision = Ratio calculated as the sum of true positives across all classes divided by the sum of true positives and false positives across all classes.

$$\sum_{n=1}^{k} \frac{TP_k}{TP_k + FP_k} \tag{1}$$

- Accuracy = Ratio calculated as the sum of true positive and true negative and all the other

$$\sum_{n=1}^{k} \frac{TP_k + TN_k}{TP_k + TN_k + FP_k + FN_k} \tag{2}$$

- Recall = Ratio calculated as the sum of true positives across all classes divided by the sum of true positives and false negatives across all classes.

$$\sum_{n=1}^{k} \frac{TP_k}{TP_k + FN_k} \tag{3}$$

- Duration = Duration of the run in seconds

The parameters were adopted in order to assess the efficiency of the network and the quality of the datasets, given the fact that we struggled to raise the accuracy of the network over 33% as long as our data were poorly manipulated into the data structure of the time series.

Once we optimized that phase, we managed to collect acceptable results as shown in the following table:

Dataset name	Precision	Accuracy	Recall	Duration
AC985	0.99027	0.98477	0.98537	5566.03
CLL985	0.96121	0.95939	0.95461	9928.22
SCLL985	0.98104	0.98477	0.98042	10293.64
SC985	0.98516	0.97969	0.97242	2859.45

Table I
ResNet results

VI. Conclusion

Deep learning networks keep showing more promising results every year in several field of application, including the classification of complex activities performed in the real world.

We have demonstrated a small example of how an human time-consuming work can be replaced by a much faster and self-sufficient computer-based system (CBS), saving the time of experts and professionals workers from repetitive tasks, thus possibly redirecting their efforts on other works in their field.

Also, we wanted to show trough the transformation from AIS data into time series structure how any records of activities can be easily adapted for classification purposes, paving the way for any other scientific study concerned with similar issues.

Above all, the systems scale well even with heavier datasets to elaborate and can be adopted on many different studies of various nature.

VII. Future developments

It is not hard to think that soon all the work we performed on this activity can be included within an user friendly software of some sort, allowing also inexperienced researchers and scientists in the fields of computer engineering to solve problems of classifications or forecast, depending on their specific tasks and necessities.

For example, it is possible to adopt all the above mentioned technology and codes within a web application, allowing people on the internet from all around the world to share their achievements and progress with people of different academic backgrounds.

This would allow them to skip the complicated computer system to which they would be agnostic, while performing better works of research and study thanks to these powerful architectures.

VIII. Appendix

```matlab
function out = CreazioneMTS

conn = database('postgres','postgres','0000','Vendor','PostgreSQL','Server','localhost','PortNumber',5432);


ss = readtable("C:\Users\Asus\Desktop\DEEP LEARNING\PROGETTO\Matlab Ale\Session_JOINED_Gear(3).csv");
count = 0;

mmsi = unique(ss(:,1),'stable');
mts = struct;
count = 1;
dat = [];
t=[];


     for i=1:size(mmsi,1)
         tab = ss(ss.mmsi==mmsi{i,1},:); %Prendo tutte le sessioni della barca iesima

          for y=1:size(tab,1)

               sql=['select id, speed, course from public."Raw2" where mmsi = ' num2str(tab{y,1}) ' and id between ' num2str(tab{y,3})...
               ' and ' num2str(tab{y,4}) ' order by id'];

               raw = select(conn,sql);   %seleziono tutti id singola sessione di una barca
               dat1= table2array(raw(:,2:3))';   % trasposta colonne  course z in righe per metterle in train dopo
               dat1= str2double(dat1(:,:));     % converto da stringhe in double

               dat1(1,:) = dat1(1,:) * 1.852 ;   %trasformare nodi in km/h (*1.85),

                    for q=1:size(dat1,1)
                        dat1(1,q) = (abs(dat1(1,q+1) - dat1(1,q)))/0.083334; %rapporto incrementale vel2 - vel1 diviso tempo
                    end


               dat{1,count}=dat1;      %creo cella per le righe di accelerazione e course
               count = count+1;
          end

     end


sel1=ss.gear=="TBB";
sel2=ss.gear=="LL";
sel3=ss.gear=="OTB";
sel4=ss.gear=="PS";
sel5=ss.gear=="PTM";


for i=1:size(ss)
    if (sel1(i,1) == 1)
        ss{i,9} = {[1]};
    end
end

for i=1:size(ss)
    if (sel2(i,1) == 1)
        ss{i,9} = {[2]};
    end
end

for i=1:size(ss)
    if (sel3(i,1) == 1)
        ss{i,9} = {[3]};
    end
end

for i=1:size(ss)
    if (sel4(i,1) == 1)
        ss{i,9} = {[4]};
    end
end

for i=1:size(ss)
    if (sel5(i,1) == 1)
        ss{i,9} = {[5]};
    end
end


ss.gear = cell2mat(ss.gear);


%Cross validation (train: 80%, test: 20%)
rng('default');
cv = cvpartition(size(dat',1),'HoldOut',0.2);
idx = cv.test;
mts.trainlabels = ss.gear(~idx');
mts.train=dat(1,~idx');
mts.testlabels = ss.gear(idx');
mts.test=dat(1,idx');
% Separate to training and test data
out = mts;
end
```

Listing 6. Matlab file creating a Multi-Class time series with acceleration and course as data to be classified within a 80-20 structure division for training and testing set

```python
def transform_0pad(x, n_var, max_length):
      n = x.shape[0]

    # the new set in ucr form np array
      ucr_x = np.zeros((n, max_length, n_var), dtype=np.float64)

    # loop through each time series
      for i in range(n):
          mts = x[i] #prendo iesima cella
          curr_length = mts.shape[1]  #prendo num colonne iesima cella
          idx = np.array(range(curr_length))  #crea array grande quanto curr_lenght
          idx_new = np.linspace(0, idx.max(), max_length) #prendo un numero di max_lenght di intervalli da 0 fino a idx.max()
          for j in range(n_var):
              ts = mts[j]
              new_ts = np.pad(ts, (max_length-curr_length, 0), 'constant')
              ucr_x[i, :, j] = new_ts

      return ucr_x
```

Listing 7. Function operating the padding of the time series in order to fill the begin of each cell with zeros

```python
def transform_mts_to_ucr_format():
    mts_root_dir = '/content/dl-4-tsc/archives/mts_archive/'
    mts_out_dir = '/content/dl-4-tsc/archives/mts_archive/'
    for dataset_name in MTS_DATASET_NAMES:

        out_dir = mts_out_dir + dataset_name + '/'

        # if create_directory(out_dir) is None:
        #     print('Already_done')

    a = loadmat(mts_root_dir + dataset_name + '/' + dataset_name + '.mat')
    a = a['mts']
    a = a[0, 0]

    dt = a.dtype.names
    dt = list(dt)

    for i in range(len(dt)):
        if dt[i] == 'train':
            x_train = a[i].reshape(max(a[i].shape))
        elif dt[i] == 'test':
            x_test = a[i].reshape(max(a[i].shape))
        elif dt[i] == 'trainlabels':
            y_train = a[i].reshape(max(a[i].shape))
        elif dt[i] == 'testlabels':
            y_test = a[i].reshape(max(a[i].shape))

    # x_train = a[1][0]
    # y_train = a[0][:,0]
    # x_test = a[3][0]
    # y_test = a[2][:,0]

    n_var = x_train[0].shape[0]

    max_length = get_func_length(x_train, x_test, func=max)
    min_length = get_func_length(x_train, x_test, func=min)

    print(dataset_name, 'max', max_length, 'min', min_length)
    print()

    x_train = transform_0pad(x_train, n_var, max_length)
    x_test = transform_0pad(x_test, n_var, max_length)



    # save them
    np.save(out_dir + 'x_train.npy', x_train)
    np.save(out_dir + 'y_train.npy', y_train)
    np.save(out_dir + 'x_test.npy', x_test)
    np.save(out_dir + 'y_test.npy', y_test)

    print('Done')
```

Listing 8. Function where our function is called to perform the padding of zeros within the data structure of the time series

REFERENCES

[1] “http://www.astrapaging.com/.”

[2] J. W. L. I. P.-A. M. Hassan Ismail Fawaz, Germain Forestier, “Deep learning for time series classification: a review,” 2019.

[3] “https://www.pgadmin.org/.”

[4] “https://www.mathworks.com/products/matlab.html.”

Listings