Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

TraClets: A trajectory representation and classification library

Ioannis Kontopoulos [*], Antonios Makris, Konstantinos Tserpes

Department of Informatics and Telematics, Harokopio University of Athens, Greece

A R T I C L E I N F O

Article history:
Received 15 July 2022
Received in revised form 14 December 2022
Accepted 3 January 2023

Keywords:
Trajectory classification
Deep learning
Convolutional neural networks
Computer vision

A B S T R A C T

Due to the advent of new mobile devices and tracking sensors in recent years, huge amounts of data are being produced every day. Therefore, novel methodologies need to emerge that dive through this vast sea of information and generate insights and meaningful information. To this end, researchers have developed several trajectory classification algorithms over the years that are able to annotate tracking data. Similarly, we propose a software that exploits image representations of trajectories, called TraClets, in order to classify trajectories in an intuitive to the human way, through computer vision techniques. The proposed software has been evaluated in three real-world datasets and is currently used in a live vessel tracking terrestrial base station.

Code metadata

Current code version	v01
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-22-00197
Legal Code License	MIT
Code versioning system used	git
Software code languages, tools, and services used	Python
Compilation requirements, operating environments & dependencies	numpy, pandas, opencv-python, bresenham, python 3.6 Interpreter
If available Link to developer documentation/manual	https://github.com/kontopoulos/TraClets
Support email for questions	kontopoulos@hua.gr, amakris@hua.gr

1. Motivation and significance

The sudden increase in mobile devices and tracking sensors has led to the creation of an abundance of data. These kind of data have shifted both researchers' and industries' attention alike towards algorithms and methodologies able to extract knowledge from a large pool of spatio-temporal data (e.g. trajectory classification). Trajectory classification is an important mining task as it can uncover information regarding potential animal migration patterns [1,2], the type and strength level of hurricanes [3] and illegal vessel activities at sea [4], thus improving overall surveillance and safety in various domains.

Several approaches for trajectory classification [4,5] try to exploit global features such as average speed, acceleration or the standard deviation of them. For instance, authors in [6] proposed a CNN model that has been trained on a set of invariant sequences extracted from the vessel trajectories. The features used are the distance between successive points, the speed over ground, acceleration, and the derivative of the vessel's course. Then, using a sliding-window the main trajectory is segmented into sub-trajectories and features are extracted for each sub-trajectory. Each sub-trajectory is classified and a majority voting scheme is used to assign a label to the whole trajectory. Duan et al. [7] also employ CNNs to extract features from the ship trajectories. They combine both kinematic and static information from vessel tracking data with an encoder–decoder Neural Network architecture and a classifier. The output of the classifier or the actual labels of the input trajectories is fed along with the latent features of each trajectory extracted by the encoder to a decoder that attempts to reconstruct the input trajectories. This semi-supervised approach allows using both labeled and unlabeled trajectories thus outperforming traditional supervised learning methods.

Other approaches try to apply trajectory partitioning [3] or find relevant sub-trajectories [8] in an attempt to identify more discriminative features. The context of the analysis is typically the physical world and the geography. Position and speed are

* Corresponding author.
E-mail addresses: kontopoulos@hua.gr (Ioannis Kontopoulos), amakris@hua.gr (Antonios Makris), tserpes@hua.gr (Konstantinos Tserpes).

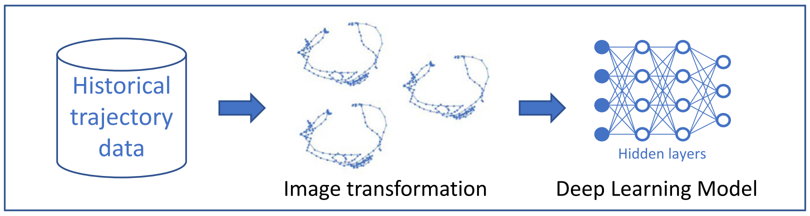


Fig. 1. System architecture.

the basic features in a possibly multi-dimensional space. However, experts rely heavily on the visualization of trajectories to manually classify trajectories that are of some significance. This provides an intuition to move the analysis in a different domain, leveraging computer vision approaches on classification. In computer vision, the most commonly used techniques include Convolutional Neural Networks (CNNs) [9,10]. Each layer of a CNN identifies a different feature of the image, including but not limited to shape and color. As an attempt to increase the performance of CNNs, deep learning emerged [11,12], increasing the complexity of the networks. One of the most common goals of such networks is to classify a set of images to a predefined set of labels of interest.

In this paper we present a software named TraClets, which provides support for the classification of trajectories. Most trajectory classification approaches found in the literature [5,13], require a pre-processing step that involves the understanding and analysis of data and the selection of features suitable only for the moving objects' trajectories to be classified. This means that features selected for a certain trajectory (e.g. cars) cannot be applied to other patterns as well (e.g. vessels) [14,15]. The proposed software skips entirely the aforementioned pre-processing step; the same technique for classifying an image (e.g. CNNs) can be applied for the classification of other trajectories since they are transformed into images. Therefore, TraClets yield a promising universal approach for the classification of moving objects' trajectories and have been used for the classification of vessel mobility patterns at sea [15,16].

2. Software description

2.1. Software architecture

TraClets is implemented in Python 3 using the Numpy and Keras frameworks, that provide efficient numerical routines for multi-dimensional data and neural network implementations, respectively.

Fig. 1 summarizes the pipeline of the proposed software. Initially, historical trajectories are collected and are annotated into specific mobility patterns, each pattern representing a different classification label. Then, trajectories of each label are transformed into images. Finally, these images are used to train a deep learning model.

2.2. Software functionalities

When applying machine learning applications, e.g., classification, a set of features is initially extracted that is suitable for the dataset at hand, and then, an algorithm exploits these features to perform the classification. The same principle also applies in the classification of spatio-temporal data and trajectories of moving objects. Therefore, to efficiently visualize and capture two key features that characterize the trajectory patterns of moving objects: i) the shape of the trajectory which indicates the way the object moves in space, and (ii) the speed that indicates how fast the object moves in space, image representations of trajectories are employed. Then, these images are fed to a Convolutional Neural Network (CNN) that performs the classification.

2.2.1. Image representation - TraClets

Trajectories of the similar moving objects (e.g. cars) tend to form similar patterns in terms of the way the object moves. However, the distance each moving object travels through space is different (e.g. a car in a highway travels greater distances compared to a car in an urban area). Therefore, the bounding box or the surveillance area in which the object moves needs to be normalized. To this end, the total distance of both the x and the y axis in which the object moves must be defined first. For this reason, both the total horizontal distance (Eq. (1)) and the total vertical distance the object has traveled are calculated (Eq. (2)) based on the minimum and maximum longitudes and latitudes respectively. The total horizontal distance is defined as:

$$d_x = longitude_{max} - longitude_{min} \tag{1}$$

and the total vertical distance the object has traveled is defined as:

$$d_y = latitude_{max} - latitude_{min} \tag{2}$$

Then, the distance that each position m has traveled from the minimum longitude and latitude can be calculated from the Eqs. (3) and (4) respectively as follows:

$$d(m_x) = longitude_m - longitude_{min} \tag{3}$$

and

$$d(m_y) = latitude_m - latitude_{min} \tag{4}$$

From Eqs. (1), (2) and (3), (4), the percentage of the total distance each position m has traveled so far can be calculated from the minimum coordinate in both x and y axes:

$$normalized(m_x) = d(m_x) \div d_x \tag{5}$$

and

$$normalized(m_y) = d(m_y) \div d_y \tag{6}$$

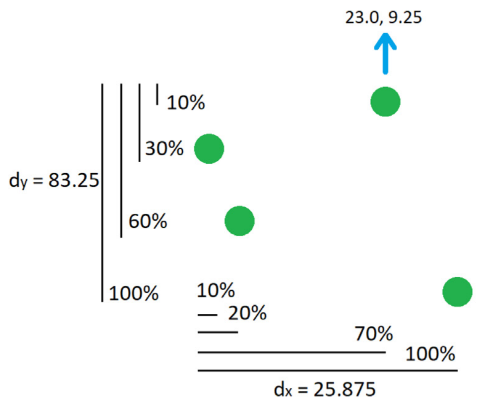
Given a predefined image size of $N \times N$, the exact position of m inside an image can be calculated as follows:

$$pixel_x = normalized(m_x) \times N \tag{7}$$

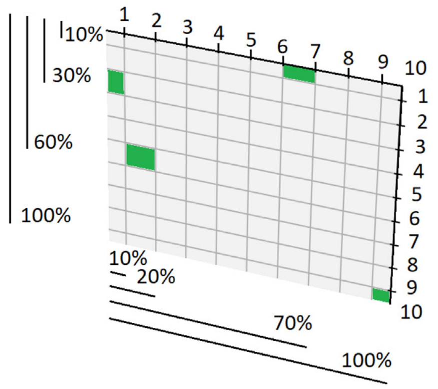
and

$$pixel_y = normalized(m_y) \times N \tag{8}$$

Therefore, each position is placed inside a normalized bounding box or a surveillance space of size $N \times N$ that is essentially an image representation. Fig. 2 demonstrates an example of the surveillance space normalization for $N = 10$. Each green circle corresponds to a position. The total horizontal distance is $d_x = 28.75 - 2.875 = 25.875$ while the total vertical distance is $d_y = 92.5 - 9.25 = 83.25$. Based on Eqs. (3) and (4), $d(m_x)$ of the upper right position ($longitude = 23.0$, $latitude = 9.25$) is equal to 20.125 and (m_y) is equal to 0. According to Eqs. (5) and (6), $normalized(m_x)$ is equal to 0.7 and $normalized(m_y)$ is equal to 0. Multiplying each normalized value by $N = 10$ results in $pixel_x = 7$ and $pixel_y = 0$. In order to fit boundary trajectory positions into



(a) Positions of the trajectory.



(b) Positions placed in a 10 × 10 raster.

Fig. 2. An example of space normalization.

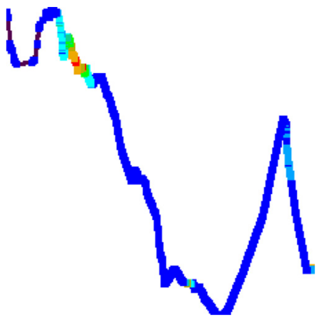


Fig. 3. An example of a TraClet.

the normalized surveillance space, pixel values smaller than N are transformed into 1 and pixel values larger than N are transformed into N. Therefore, $pixel_y$ is transformed into 1 and the final pixel position inside the 10×10 image is $x = 7$ and $y = 1$ as shown in Fig. 2.

To capture the changes in speed, previous research conducted on the classification of vessel trajectories [15–17] is taken into consideration, in which the maximum speed $max(speed)$ of the vessels was segmented to 11 equally sized increments with each speed increment corresponding to a different RGB color value in the final image. Let $S = [0, max(speed)]$ be the possible speed values at which the moving objects move. A speed increment $incr(speed)$ is defined as:

$$incr(speed) = max(speed) \div 11 \tag{9}$$

resulting in an arithmetic sequence such that:

$$\alpha_i = \alpha_1 + (i - 1) \times incr(speed) \tag{10}$$

where α_i is the ith term of the sequence, α_1 is the initial term of the sequence (in our case 0 m/s) and $incr(speed)$ is the common difference of successive members. Each pixel $p(pixel_x, pixel_y)$ that corresponds to a position m is colored with a different color value that corresponds to the ith term of the sequence depending on the speed the position has at the moment. Moreover, pixels that do not contain any positions are colored white. Finally, temporal successive pixels are connected to each other with lines using the Bresenham algorithm [18]. The final result is an image, called TraClet, that depicts the movement, and the speed of the trajectory. An example of a TraClet is illustrated in Fig. 3.

2.2.2. Deep learning for trajectory classification

This section describes the deep learning approach employed for mobility data classification from trajectory images, based on fine-tuning and Convolutional Neural Networks.

CNNs are able to learn deep features of trajectories such as shape and color (e.g., speed) automatically, thus eliminating hand-created feature extraction. As stated in [19], neurons in the CNN receive signals from other neurons in the local area in the preceding layer; thus, the CNN is able to capture more local spatial correlations. Furthermore, the weight-sharing characteristic in the connection of adjacent layers can significantly reduce the number of variables. The main disadvantage of deep learning approaches regarding image classification is that they require a large amount of data in order to perform accurate feature extraction and classification. In order to overcome this limitation, transfer learning is adopted. Transfer learning is a common and effective method, which aims at training a network with fewer samples, as the knowledge extracted by a pre-trained model is then reused and applied to the given task of interest. The intuition behind transfer learning is that generic features learned on a general large data set can be shared among seemingly disparate data sets. The learned features can be used to solve a different, but related task [20].

In general, there are two ways of transfer learning utilization in the context of deep learning [21]: (a) feature extraction [22] and (b) fine-tuning [23,24]. In feature extraction, representations learned from a pre-trained model are treated as an arbitrary feature extractor and employed in order to extract meaningful features from new samples. As the base convolutional network already contains generically useful features for classification, there is no need for retraining the entire model. In fine-tuning, the fully connected layers of the pre-trained model are replaced with a new set of fully connected layers. These new layers are trained on a given data set, and the weights of the top layers of the pre-trained model along with the newly-added layers are "fine-tuned" by means of backpropagation. Thus, the weights are tuned from generic feature maps to features associated specifically with the provided data set. Fine-tuned learning experiments have been presented to be much faster and more accurate in comparison with models trained from scratch [25].

More specifically, fine-tuning is a multi-step process which includes:

- remove the fully connected nodes at the end of the network (i.e., where the actual class label predictions are made);
- replace the fully connected nodes with freshly initialized ones;

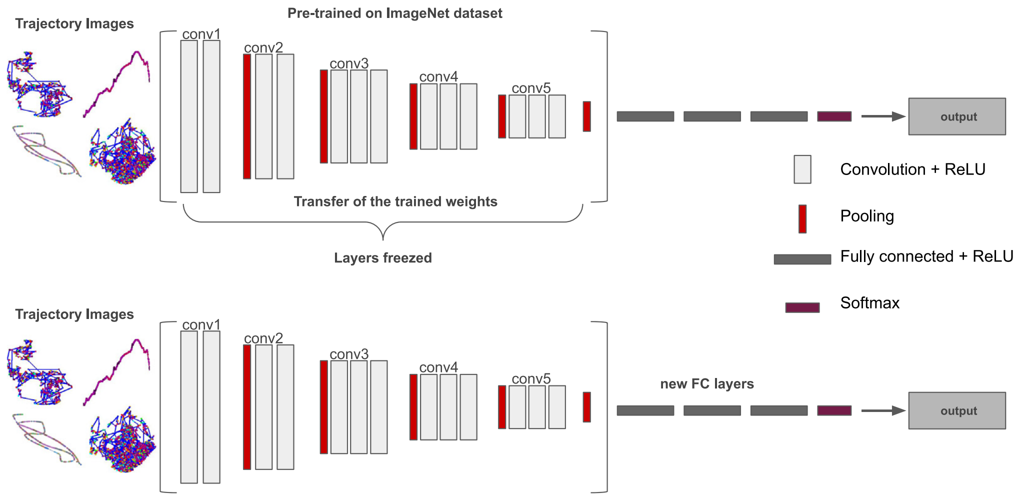


Fig. 4. Fine-tuning on the VGG16 network architecture.

- freeze earlier convolutional layers earlier in the network (ensuring that any previous robust features learned by the CNN are not destroyed);
- start training, but only train the fully connected layer heads;
- optionally unfreeze some/all of the convolutional layers in the network and perform a second pass of training.

The proposed deep learning approach provides an automated way to execute various pre-trained supervised deep learning models from Keras[1] (i.e. VGG16, ResNet50, InceptionV3, etc.), that are made available alongside pre-trained weights. Fig. 4 illustrates an example of the fine-tuning process on the VGG16 network. The weights are pre-trained on the ImageNet [26] dataset for all different models. The VGG16 model consists of 13 convolutional (*CONV*) and 3 fully connected (*FC*) layers. After each convolution layer, the pooling layer was introduced to carry out downsampling operations, which reduced the in-plane dimensionality of the feature maps. Downsample operations after convolutional layers introduce a translation invariance to small shifts and distortions and decrease the number of subsequent learnable parameters. Average pooling [27] is employed as the pooling strategy, which performs an extreme type of dimensionality reduction, where a tensor with dimensions $h \times w \times d$ is downsampled into a $1 \times 1 \times d$ array by simply taking the average of all the elements in each $h \times w$ feature map, whereas the depth of feature maps is retained. As a result, the number of learnable parameters is reduced, preventing overfitting. The output feature maps of the final layer are flattened and connected to the fully connected layers, in which every input is connected to every output by a learnable weight. The final set of layers which contain the *FC* layers along with the activation function is called the "head". The *FC* layers are truncated, and the final *pooling* layer is treated as a feature extractor. The body of the network, i.e., the weights of the convolution layers of the pre-trained network, are freezed such that only the *FC* head layer is trained. This is because the convolution layers had already learned discriminative filters and captured universal features like curves and edges; thus, these weights had to remain intact. Finally, the truncated layers are replaced by a new *FC* head layer, which is randomly initialized and placed on top of the original architecture (bottom of the figure). Then, the model is trained through a backpropagation process. In other words, the *FC* head layer is randomly initialized from scratch and focus on learning dataset-specific features. Early-layer features appeared more generic, whereas later features progressively become more specific to a task [28].

3. Illustrative examples

An illustrative example is provided in this section with a typical pipeline for creating trajectories into images and training a deep learning model with a given dataset. The dataset used for this example contains trajectories collected from a terrestrial receiver of vessel tracking data that covers the Saronic Gulf (Greece) including the port of Piraeus. The dataset provides information for 1229 unique vessels and contains 11, 769, 237 positional records in total. The vessels have been monitored for almost one and a half month period starting at February 18th, 2020 and ending at March 31th, 2020. Finally, the dataset contains three classes, one for each of the mobility patterns present in the trajectories, namely Anchored, Moored and Underway.

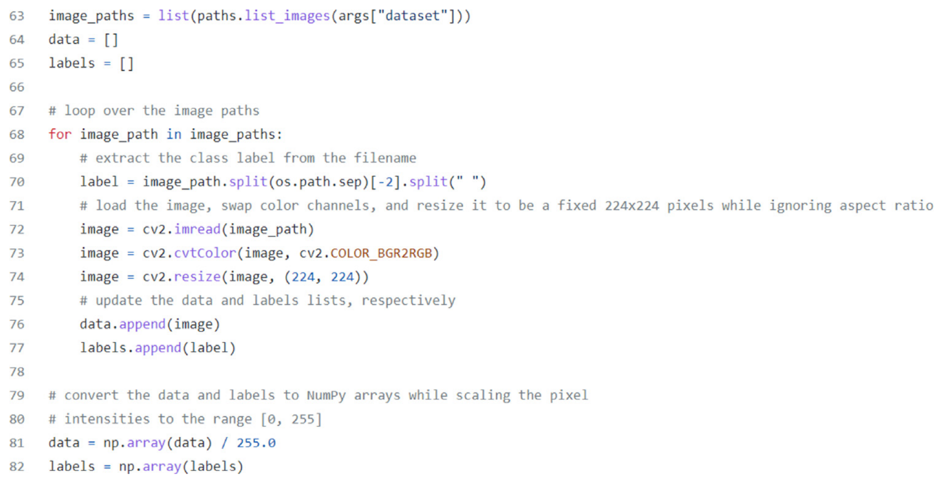
To transform trajectories into images, the user needs to run the following command:

```
python traclet.py --d --s
```

where *d* indicates the csv file containing the annotated trajectories, and *s* indicates the resolution of the resulting images. The result is a set of folders with each folder containing images of a different label. These images, i.e. TraClets, are then fed as input into the deep learning model for training. Before the model training process, a configuration file is provided, in which the user is able to configure several variables used for training as:

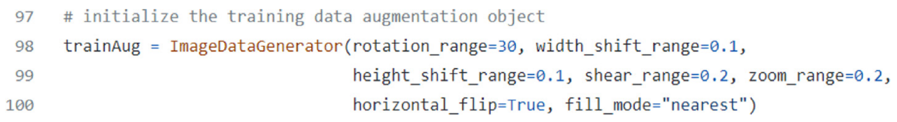
- *epochs* - an arbitrary cutoff, generally defined as "one pass over the entire dataset", used to separate training into distinct phases, which is useful for logging and periodic evaluation
- *batch_size* - the number of training examples in one forward–backward pass
- *test_size* - this parameter decides the size of the data that has to be split as the test dataset
- *dropout_keep_prob* - dropout is one of the most popular regularization techniques, which forces the weights in the network to receive only small values, making the distribution of weight values more regular. As a result, this technique is able to reduce overfitting on small training examples [29]
- *number_of_classes* - the number of classes

[1] Keras Applications, https://keras.io/api/applications/



```
63  image_paths = list(paths.list_images(args["dataset"]))
64  data = []
65  labels = []
66
67  # loop over the image paths
68  for image_path in image_paths:
69      # extract the class label from the filename
70      label = image_path.split(os.path.sep)[-2].split(" ")
71      # load the image, swap color channels, and resize it to be a fixed 224x224 pixels while ignoring aspect ratio
72      image = cv2.imread(image_path)
73      image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
74      image = cv2.resize(image, (224, 224))
75      # update the data and labels lists, respectively
76      data.append(image)
77      labels.append(label)
78
79  # convert the data and labels to NumPy arrays while scaling the pixel
80  # intensities to the range [0, 255]
81  data = np.array(data) / 255.0
82  labels = np.array(labels)
```

Fig. 5. Data processing.



```
 97  # initialize the training data augmentation object
 98  trainAug = ImageDataGenerator(rotation_range=30, width_shift_range=0.1,
 99                                height_shift_range=0.1, shear_range=0.2, zoom_range=0.2,
100                                horizontal_flip=True, fill_mode="nearest")
```

Fig. 6. Augmentation generator object.

- *dl_network* - the Keras model (VGG16, ResNet50, etc.)
- *activation_function* - the activation function is responsible for transforming the summed weighted input from the node into the activation of the node or output for that input
- *activation_function_output* - the activation function of the output layer
- *loss_function* - the loss function, a scalar value to be minimized during training of the model
- *optimizer* - optimizers are algorithms or methods used to change the attributes of the neural network such as weights and learning rate to reduce the losses

Finally, the user is able to train the selected deep learning model by providing three required command line arguments:

- *--d*, which points to the directory containing the images for training the image classifier
- *--m*, which saves the serialized image classifier after it has been trained
- *--c*, a configuration file which contains the variables used for training the model

using the following command:

```
python train_model.py --d --m --c
```

To load the data, we select all paths to the images in the *--dataset* directory as shown in Fig. 5, (Line 63). Then, for each *imagePath*, we extract the class label from the path (Line 70), load the image, preprocess it by converting to RGB channel ordering, and resizing it to 224×224 pixels so that it is ready for our Convolutional Neural Network (Lines 72–74). Afterwards, we update data and labels lists respectively (Lines 76–77). Finally, we scale pixel intensities to the range $[0, 1]$ and convert both data and labels to NumPy array format (Lines 81 and 82).

In addition, in order to prevent model overfitting, data augmentation is performed during training, leveraging several multiprocessing techniques. Specifically, the transformations employed included random rotation of the images (the maximum rotation angle was 30 degrees), horizontal flips, shearing, zooming, and small random noise perturbations. Data augmentation improves the generalization and enhances the learning capability of the model. The data augmentation generator object is illustrated in Fig. 6.

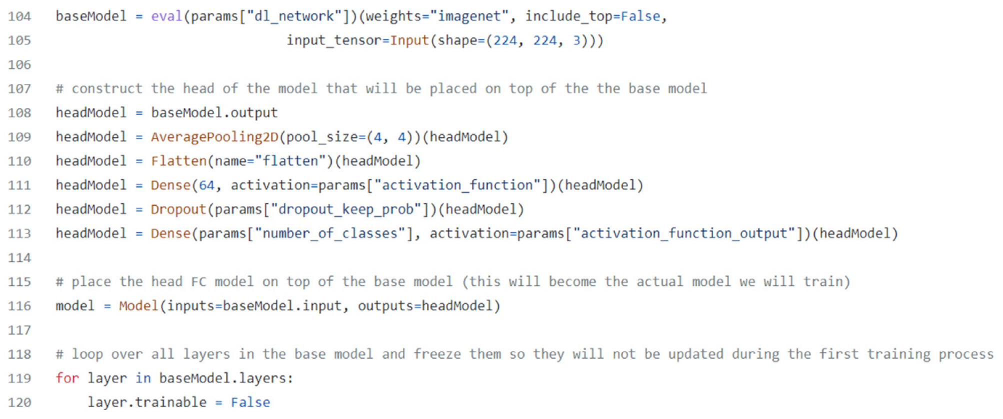
The fine-tuning process is illustrated in Fig. 7. Initially, the model's architecture is loaded (with pre-trained ImageNet weights), leaving off the *FC* layer head (Lines 104 and 105). From there, we define a new fully connected layer head (Lines 108–113). Then, the new *FC* layer head is placed on top of the base network (Line 116). Finally the CONV weights of the model are freezed such that only the *FC* layer head is trained (Lines 119–120); this completes our fine-tuning setup.

Then, the model is compiled with learning rate decay as illustrated in Fig. 8, (Lines 124–126). Learning rate decay is a technique for training neural networks. It starts training the network with a large learning rate and then slowly reducing/decaying it until local minima is obtained. Afterwards, the train process starts using the Keras' *fit_generator* method (Lines 132–138).

After the execution of the model, the *.model* file is created and saved along with the following plots:

- Confusion Matrix
- ROC curves
- ROC curves zoomed in view of the upper left corner
- Training/Validation - Accuracy/Loss

An example of the plots generated for a specific set of parameters in the configuration file is shown in Fig. 9. Fig. 9(a) presents the training and the validation accuracy and loss during the training of the model. It can be observed that both validation and training metrics are close to each other indicating that no significant deviations and overfitting occur when validating the model's performance. Fig. 9(b) visualizes the confusion matrix of the predicted labels of the proposed model. Finally, Fig. 9(c) illustrates the Receiver Operating Characteristic (ROC) curves of the model's classification performance and Fig. 9(d) zooms in on the ROC curves for better visualization.



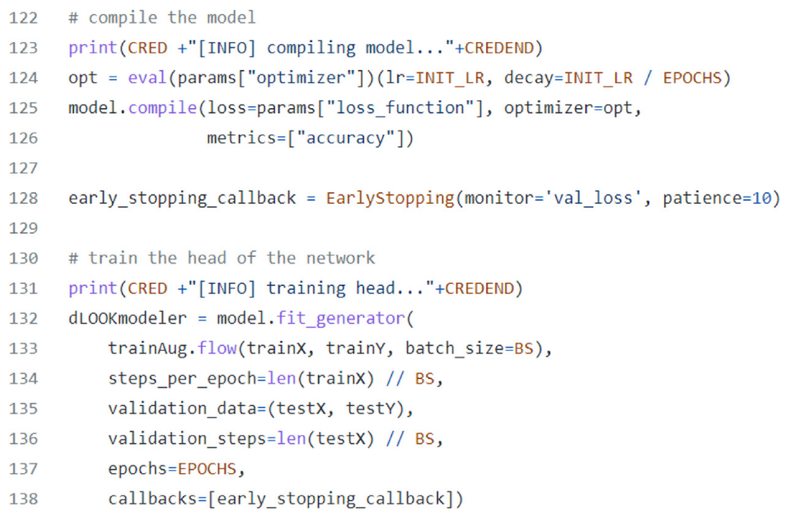
```
baseModel = eval(params["dl_network"])(weights="imagenet", include_top=False,
                            input_tensor=Input(shape=(224, 224, 3)))

# construct the head of the model that will be placed on top of the the base model
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(4, 4))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(64, activation=params["activation_function"])(headModel)
headModel = Dropout(params["dropout_keep_prob"])(headModel)
headModel = Dense(params["number_of_classes"], activation=params["activation_function_output"])(headModel)

# place the head FC model on top of the base model (this will become the actual model we will train)
model = Model(inputs=baseModel.input, outputs=headModel)

# loop over all layers in the base model and freeze them so they will not be updated during the first training process
for layer in baseModel.layers:
    layer.trainable = False
```

Fig. 7. Fine-tuning.



```
# compile the model
print(CRED +"[INFO] compiling model..."+CREDEND)
opt = eval(params["optimizer"])(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss=params["loss_function"], optimizer=opt,
              metrics=["accuracy"])

early_stopping_callback = EarlyStopping(monitor='val_loss', patience=10)

# train the head of the network
print(CRED +"[INFO] training head..."+CREDEND)
dLOOKmodeler = model.fit_generator(
    trainAug.flow(trainX, trainY, batch_size=BS),
    steps_per_epoch=len(trainX) // BS,
    validation_data=(testX, testY),
    validation_steps=len(testX) // BS,
    epochs=EPOCHS,
    callbacks=[early_stopping_callback])
```

Fig. 8. Compile and train the model.

4. Impact

While working with the analysis of spatio-temporal data, the process of feature engineering (i.e. transforming raw data into features suitable for the mobility patterns of different application scenarios) is a time-consuming and complex task. TraClets help data scientists in the fields of trajectory mining and surveillance by creating a trajectory representation format indicative of a moving object's behavior in space and time eliminating the need of feature engineering and extraction. Patterns formed by the trajectories of moving objects tend to be visually distinct, thus representing a trajectory as an image is an intuitive step for the classification of trajectories. This visual distinctiveness leads to an increased trajectory classification performance and for that reason TraClets have also been used in the industry for the classification of vessel activities at sea [16] and in real-time surveillance applications in the maritime domain [15]. Moreover, by skipping the time-consuming task of feature analysis, data scientists and surveillance authorities alike can focus on identifying behaviors of interest in sequences of already classified mobility patterns, allowing for a more comprehensive and accurate surveillance.

The advantages of TraClets are not limited to an increased classification performance and the elimination of feature engineering. The partitioning of trajectories often constitutes a first step when dealing with trajectory classification [3,30]. Such approaches often require input parameters which are hard to determine and that eventually have a significant impact on the partitioning results. As TraClets skip entirely this step, the need of arbitrary or empirical user-defined parameters is eliminated, making TraClets scalable and robust. Moreover, TraClets work for trajectory classification even when data are not transmitted at fixed intervals (e.g. hourly). This is in contrast to time-series methodologies [31] that are inherently unsuitable for such tasks and require data points at fixed time points.

Finally, despite the high classification accuracy provided by the proposed software, only the shape of the trajectory and the speed of the moving object is used in the image representation. The reason behind this is that when the acceleration was integrated into the images as a color coded Bresenham line, no significant performance gains were introduced. Nevertheless, a revised image representation is within the future plans that takes into account even more aspects of the trajectories and features of the positional data.

5. Conclusions

In this paper we present a novel and high-accuracy trajectory classification software called TraClets, in an attempt to provide an efficient and alternative way to treat the problem of trajectory classification. The proposed software employs several state-of-the-art deep learning models and creates a universal approach for the classification of trajectories. This universal approach can

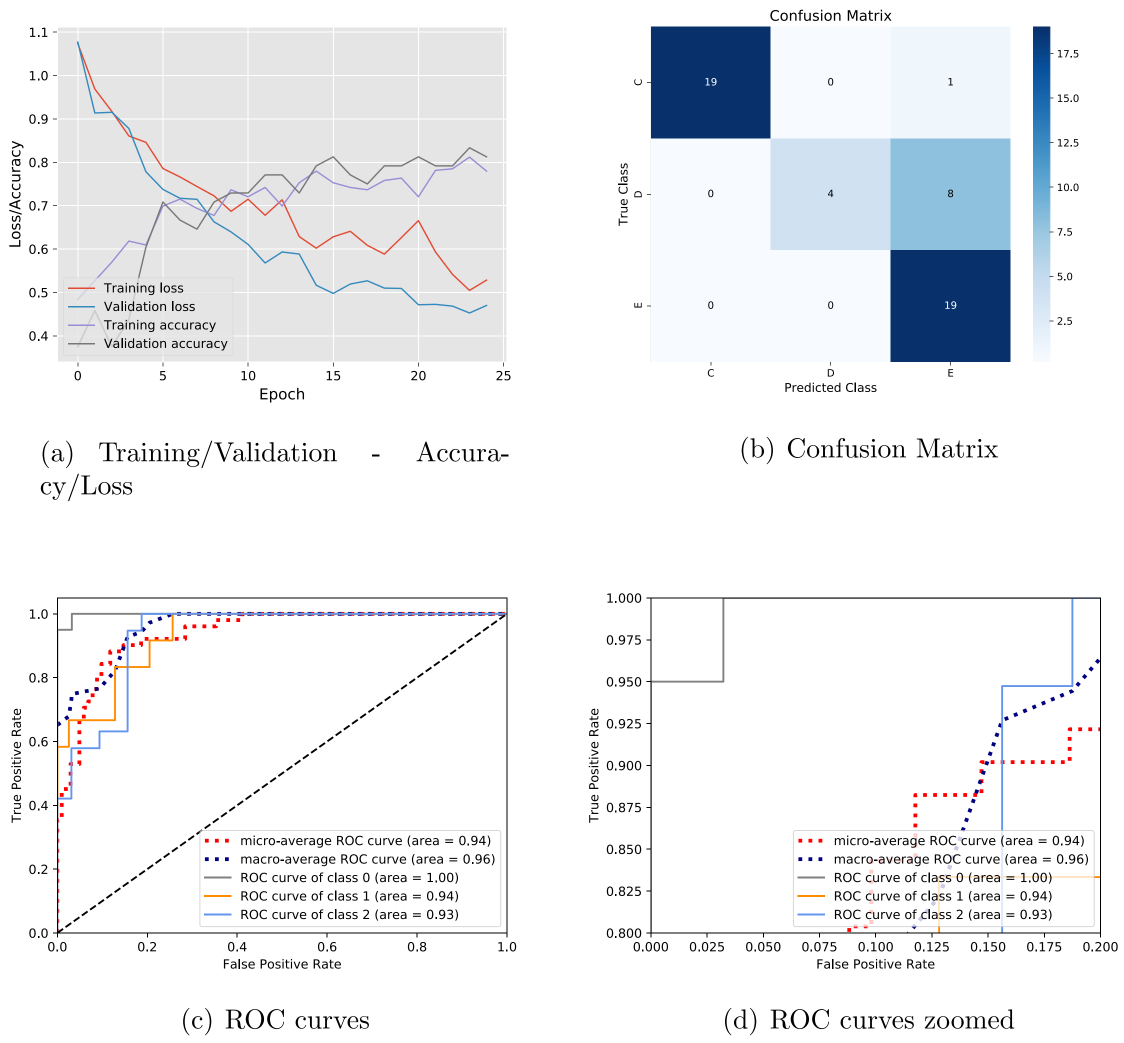


Fig. 9. Example of generated plots for a specific set of parameters.

be achieved through image representations of trajectories, that amplify the distinct visual difference that most of the moving objects' trajectories, if not all, have in three distinct domains.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work was supported by the SmartShip and MASTER Projects, Greece through the European Union's Horizon 2020 research and innovation programme under the Marie-Sklodowska Curie grant agreement no 823916 and 777695 respectively. The work reflects only the authors' view and that the EU Agency is not responsible for any use that may be made of the information it contains.

References

[1] Damiani ML, Issa H, Fotino G, Heurich M, Cagnacci F. Introducing 'presence' and 'stationarity index' to study partial migration patterns: an application of a spatio-temporal clustering technique. Int J Geogr Inf Sci 2016;30(5):907–28. http://dx.doi.org/10.1080/13658816.2015.1070267.

[2] De Groeve J, Van de Weghe N, Ranc N, Neutens T, Ometto L, Rota-Stabelli O, Cagnacci F. Extracting spatio-temporal patterns in animal trajectories: an ecological application of sequence analysis methods. Methods Ecol Evol 2016;7(3):369–79. http://dx.doi.org/10.1111/2041-210X.12453, arXiv:https://besjournals.onlinelibrary.wiley.com/doi/pdf/10.1111/2041-210X.12453.

[3] Lee J, Han J, Li X, Gonzalez H. *TraClass*: trajectory classification using hierarchical region-based and trajectory-based clustering. Proc VLDB Endow 2008;1(1):1081–94. http://dx.doi.org/10.14778/1453856.1453972, URL http://www.vldb.org/pvldb/vol1/1453972.pdf.

[4] Kontopoulos I, Chatzikokolakis K, Tserpes K, Zissis D. Classification of vessel activity in streaming data. In: Gascon-Samson J, Zhang K, Daudjee K, Kemme B, editors. DEBS '20: The 14th ACM International conference on distributed and event-based systems, montreal, Quebec, Canada, July 13-17, 2020. ACM; 2020, p. 153–64. http://dx.doi.org/10.1145/3401025.3401763.

[5] da Silva CL, Petry LM, Bogorny V. A survey and comparison of trajectory classification methods. In: 8th Brazilian conference on intelligent systems. IEEE; 2019, p. 788–93. http://dx.doi.org/10.1109/BRACIS.2019.00141.

[6] Arasteh S, Tayebi MA, Zohrevand Z, Glässer U, Shahir AY, Saeedi P, Wehn H. Fishing vessels activity detection from longitudinal AIS data.

In: Proceedings of the 28th International conference on advances in geographic information systems. 2020, p. 347–56.

[7] Duan H, Ma F, Miao L, Zhang C. A semi-supervised deep learning approach for vessel trajectory classification based on AIS data. Ocean Coast Manag 2022;218:106015.

[8] Ferrero CA, Alvares LO, Zalewski W, Bogorny V. MOVELETS: exploring relevant subtrajectories for robust trajectory classification. In: Haddad HM, Wainwright RL, Chbeir R, editors. Proceedings of the 33rd Annual ACM symposium on applied computing. ACM; 2018, p. 849–56. http://dx.doi.org/10.1145/3167132.3167225.

[9] Rawat W, Wang Z. Deep convolutional neural networks for image classification: A comprehensive review. Neural Comput 2017;29(9):2352–449. http://dx.doi.org/10.1162/neco_a_00990.

[10] Zhong S, Liu Y, Liu Y. Bilinear deep learning for image classification. In: Candan KS, Panchanathan S, Prabhakaran B, Sundaram H, Feng W, Sebe N, editors. Proceedings of the 19th International conference on multimedia 2011, Scottsdale, AZ, USA, November 28 - December 1, 2011. ACM; 2011, p. 343–52. http://dx.doi.org/10.1145/2072298.2072344.

[11] Wu J, Yu Y, Huang C, Yu K. Deep multiple instance learning for image classification and auto-annotation. In: IEEE Conference on computer vision and pattern recognition. CVPR 2015, Boston, MA, USA, June 7-12, 2015, IEEE Computer Society; 2015, p. 3460–9. http://dx.doi.org/10.1109/CVPR.2015.7298968.

[12] Makris A, Kontopoulos I, Psomakelis E, Tserpes K. Semi-supervised trajectory classification using convolutional auto-encoders. In: Proceedings of the 1st ACM SIGSPATIAL international workshop on animal movement ecology and human mobility. 2021, p. 27–32.

[13] Pipanmekaporn L, Kamonsantiroj S. A deep learning approach for fishing vessel classification from vms trajectories using recurrent neural networks. In: International conference on human interaction and emerging technologies. Springer; 2020, p. 135–41.

[14] de Souza EN, Boerder K, Matwin S, Worm B. Improving fishing pattern detection from satellite AIS using data mining and machine learning. ;rvtPLosO 2016;11(7):1–20. http://dx.doi.org/10.1371/journal.pone.0158248.

[15] Kontopoulos I, Makris A, Tserpes K. A deep learning streaming methodology for trajectory classification. ISPRS Int J Geo Inf 2021;10(4):250. http://dx.doi.org/10.3390/ijgi10040250.

[16] Kontopoulos I, Makris A, Zissis D, Tserpes K. A computer vision approach for trajectory classification. In: 22nd IEEE International conference on mobile data management. IEEE; 2021, p. 163–8. http://dx.doi.org/10.1109/MDM52706.2021.00034.

[17] Kontopoulos I, Makris A, Tserpes K, Bogorny V. TraClets: Harnessing the power of computer vision for trajectory classification. 2022, arXiv preprint arXiv:2205.13880.

[18] Gaol FL. Bresenham algorithm: Implementation and analysis in raster shape. J Comput 2013;8(1):69–78. http://dx.doi.org/10.4304/jcp.8.1.69-78.

[19] Chen R, Chen M, Li W, Wang J, Yao X. Mobility modes awareness from trajectories based on clustering and a convolutional neural network. ISPRS Int J Geo Inf 2019;8(5):208. http://dx.doi.org/10.3390/ijgi8050208.

[20] Pan SJ, Yang Q. A survey on transfer learning. IEEE Trans Knowl Data Eng 2009;22(10):1345–59.

[21] Yamashita R, Nishio M, Do RKG, Togashi K. Convolutional neural networks: an overview and application in radiology. Insights Imaging 2018;9(4):611–29.

[22] Chen Y, Jiang H, Li C, Jia X, Ghamisi P. Deep feature extraction and classification of hyperspectral images based on convolutional neural networks. IEEE Trans Geosci Remote Sens 2016;54(10):6232–51. http://dx.doi.org/10.1109/TGRS.2016.2584107.

[23] Bengio Y. Deep learning of representations for unsupervised and transfer learning. In: Guyon I, Dror G, Lemaire V, Taylor GW, Silver DL, editors. Unsupervised and transfer learning - workshop held at ICML 2011, Bellevue, Washington, USA, July 2, 2011. JMLR Proceedings, vol. 27, JMLR.org; 2012, p. 17–36, URL http://proceedings.mlr.press/v27/bengio12a.html.

[24] Donahue J, Jia Y, Vinyals O, Hoffman J, Zhang N, Tzeng E, Darrell T. DeCAF: A deep convolutional activation feature for generic visual recognition. In: Proceedings of the 31th international conference on machine learning. JMLR Workshop and Conference Proceedings, 32, JMLR.org; 2014, p. 647–55, URL http://proceedings.mlr.press/v32/donahue14.html.

[25] Mohanty SP, Hughes DP, Salathé M. Using deep learning for image-based plant disease detection. 2016, CoRR abs/1604.03169 arXiv:1604.03169.

[26] Deng J, Dong W, Socher R, Li L, Li K, Li F. ImageNet: A large-scale hierarchical image database. In: 2009 IEEE Computer society conference on computer vision and pattern recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA. IEEE Computer Society; 2009, p. 248–55. http://dx.doi.org/10.1109/CVPR.2009.5206848.

[27] Lin M, Chen Q, Yan S. Network in network. 2013, arXiv preprint arXiv:1312.4400.

[28] Zeiler MD, Fergus R. Visualizing and understanding convolutional networks. In: Fleet DJ, Pajdla T, Schiele B, Tuytelaars T, editors. Computer vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I. Lecture Notes in Computer Science, vol. 8689, Springer; 2014, p. 818–33. http://dx.doi.org/10.1007/978-3-319-10590-1_53.

[29] Hawkins DM. The problem of overfitting. J Chem Inf Comput Sci 2004;44(1):1–12.

[30] Lee J, Han J, Whang K. Trajectory clustering: a partition-and-group framework. In: Chan CY, Ooi BC, Zhou A, editors. Proceedings of the ACM SIGMOD international conference on management of data, Beijing, China, June 12-14, 2007. ACM; 2007, p. 593–604. http://dx.doi.org/10.1145/1247480.1247546.

[31] Fawaz HI, Forestier G, Weber J, Idoumghar L, Muller P. Deep learning for time series classification: a review. Data Min Knowl Discov 2019;33(4):917–63. http://dx.doi.org/10.1007/s10618-019-00619-1.