Timage – A Robust Time Series Classification Pipeline

Marc Wenninger[1], Sebastian P. Bayerl[2], Jochen Schmidt[1], and Korbinian Riedhammer[2]

[1] Rosenheim Technical University of Applied Sciences, Dep. of Computer Science,
[2] Technische Hochschule Nürnberg Georg Simon Ohm, Dep. of Computer Science
{marc.wenninger,jochen.schmidt}@th-rosenheim.de
{sebastian.bayerl,korbinian.riedhammer}@th-nuernberg.de

Abstract. Time series are series of values ordered by time. This kind of data can be found in many real world settings. Classifying time series is a difficult task and an active area of research. This paper investigates the use of transfer learning in Deep Neural Networks and a 2D representation of time series known as Recurrence Plots. In order to utilize the research done in the area of image classification, where Deep Neural Networks have achieved very good results, we use a Residual Neural Networks architecture known as ResNet. As preprocessing of time series is a major part of every time series classification pipeline, the method proposed simplifies this step and requires only few parameters. For the first time we propose a method for multi time series classification: Training a single network to classify all datasets in the archive with one network. We are among the first to evaluate the method on the latest 2018 release of the UCR archive, a well established time series classification benchmarking dataset.

Keywords: Deep Neural Networks, Transfer Learning, Time Series Classification

1 Introduction

Time series are N-dimensional signals ordered by time resulting in $N+1$ dimensions, hence all real world data recorded over time are a time series. The sequence of data points allows one to identify higher features based on the combination of multiple data points in respect to the order. All time series classification approaches try to identify these higher features to separate the classes based on them. Time series classification is a hard problem that is not yet fully understood and numerous attempts have been made in the past to create generic and domain specific classification methods. Because of the diverse domains where time series are present, the research and methods are diverse as well. Many proposed methods for time series classification are actually classification pipelines: Multi-stage processes combining preprocessing steps and the actual classification method that separates the classes. Preprocessing is used to provide more easily

separable features, e.g., by reducing dimensionality, identifying regions of interest, or simply reducing length, always with the goal to support the classification algorithm.

During preprocessing, domain specific knowledge plays a vital role, as domain knowledge allows for identifying and removing noisy features and boost class-unique features by transferring knowledge from related research. A generic classification pipeline needs to overcome the lack of domain knowledge. The authors of the UCR data set came to the conclusion that most probably there will never be one classification pipeline that rules them all, but every pipeline will have its advantages and disadvantages in the specific domain [3]. The reason for this lies within the differences of where higher features are located in the data. The importance of adaptable feature extraction is underlined by the success of classification pipelines that focus on feature extraction such as Weasel (Word ExtrAction for time SEriescLassification) [18] and BOSS (Bag-of-SFA-Symbols) [17], which are able to adapt the feature extraction to different domains. These methods preprocess the time series by encoding sliding windows into strings based on a similarity measure, followed by a classification using 1-nearest-neighbor. A nearly preprocess-less classification pipeline was proposed by [13] using a Convolutional Neural Network (CNN) with a Long Short-Term Memory (LSTM) path. [8] proposes a pipeline where time series are encoded as Reccurence Plot (RP) images, which are then classified using a CNN. Encoding the time series of a 1D signal into a 2D image introduces new texture features, that allows one to utilize research efforts from image classification on time series classification. RP in combination with a nearest neighbor classifier are used by [15]. The encoding of time series into images has also been used for sound classification, but instead of only using RP, [16] also evaluated the use of spectrogram images in combination with a CNN architecture that takes both image representations as the network input. [4] evaluate the use of Spectrogram, Mel-Frequency Cepstral Coefficients (MFCC), and RP for the classification of environmental sound recordings using the AlexNet and GoogLeNet image recognition networks. [10] presented results of an extensive experiment on feeding time series directly into 9 different well known neural network architectures such as MLP, FCN and ResNet. Their research shows that ResNet outperforms the others on the UCR archive. The same author also showed that CNNs can benefit from transfer learning by reusing network weights [11].

Our contribution is a robust time series classification pipeline using transfer learning on ResNet with RP called *timage*. The ResNet network architecture is kept unchanged apart from adapting the input and output size of the network to fit the dataset. As a starting point for transfer learning we reuse the network weights from the ImageNet challenge [9] and further experimented with different strategies to obtain better weights. We present a Single Classifier (SC) for each dataset in the UCR archive as done by all published models and are among the first to evaluate our method on the latest 2018 UCR release. For the first time we present an All Classifier (AC), a single classification model trained on all datasets in the archive at once.

2 Method

2.1 Approach

The main idea of this paper is to use existing knowledge in the form of pretrained image recognition networks to classify time series. In order to take advantage of these neural networks that were trained on huge amounts of image data, time series must be converted to images. This can be achieved by representations used for qualitative analysis like Recurrence Plots (RP), Spectrograms, Gramian Angular Summation/Difference Fields (GASF/GADF), and Markov Transition Fields (MTF) [19, 14, 2]. For our experiments we focus on RP for time series representation as proposed in [15, 16].

Transfer Learning One of the main concepts exploited in this paper is called transfer learning. The general assumption behind it is that something learned for one problem can be used to improve generalization for another problem. It is assumed that many factors which explain variations in the first problem are also relevant for a similar problem. Simply speaking, knowledge created by solving the first problem is used and applied to solve another problem [7]. This is especially useful when only a small amount of training data is available. In the case of the UCR archive, this means that even the sets with only very little training samples can be properly learned by a neural network which usually needs huge amounts of training data to converge.

As we wanted to explore the concept of transfer learning even further, we tried different strategies to obtain new weights. The first approach was ultimately the most successful one using the original weights obtained by training the networks on the ImageNet dataset and then use these weights for our networks. A second method to obtain better weights for transfer learning was to train an AC system and then use these weights in training SC systems. Another approach was training an AC not to separate all different classes but to separate datasets. For this, the UCR archive data was relabeled to only have one label per dataset. This means the network is trained to separate the datasets and not the classes within each dataset. This approach in itself worked reasonably well but using these weighs for training AC or SC systems did not improve overall accuracy. An interesting observation that could be made was that AC and SC systems trained on weights obtained by training with the RPs converged faster than systems using the original ResNet weights.

Reccurrence Plots The motivation for RP is that recurrence is a property of many natural and real world systems. Situations or states that have been observed once are often followed by similar states. RP can be used to visualize these recurrences [14]. RP plot the recurrence matrix that is formally defined by equation (1), where N is the number of measured points $\boldsymbol{x_i}$, ϵ a threshold and Θ the Heaviside or step function where $\Theta(x) = 0$ if $x \leq 0$ and $\Theta(x) = 1$ otherwise.

A proper distance measure has to be chosen for $||.||$, such as Euclidean distance or cosine distance.

$$R_{i,j}(\epsilon) = \Theta(\epsilon - ||\boldsymbol{x_i} - \boldsymbol{x_j}||), i, j = 1, ..., N \tag{1}$$

For states that are in an ϵ-neighborhood, the notion in (2) can be used:

$$\boldsymbol{x_i} \approx \boldsymbol{x_j} \iff R_{i,j} \equiv 1 \tag{2}$$

The RP is then generated by binarizing the recurrence matrix using a threshold [14]. These plots are particularly useful to humans in qualitative assessment because patterns can be quickly identified visually. For the use in image classification, more data in the form of more discriminable values in between might be desireable. This is the main reason for using a modification of recurrence plots where no thresholding is performed to get more information encoded into the image. Input data usually comes in the form of feature vectors, and the pairwise distances between those are computed, which creates the distance matrix $D_{i,j}$:

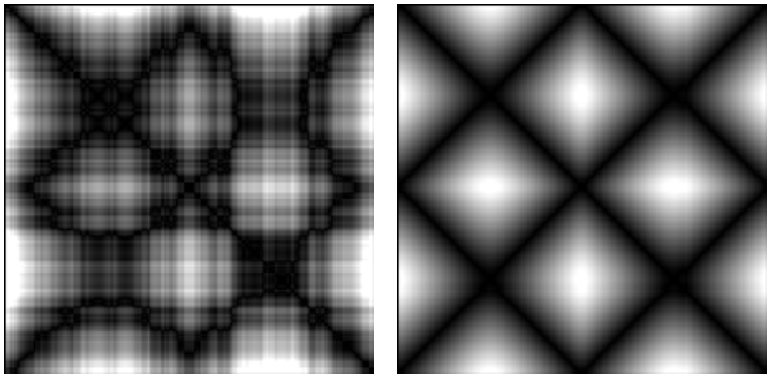
$$D_{i,j} = ||\boldsymbol{x} - \boldsymbol{y}|| \quad . \tag{3}$$

These pairwise distances are then plotted. This modification of the RP is also known as *unthresholded recurrence plot* or, maybe more appropriate, *distance plots* as described in (3) [14]. We chose to use Euclidean distance for our plots. After calculating the distances we apply some normalization by using a Min-Max scaler and a threshold cut-off at three times the standard deviation of all distances in the distance matrix. (4) represents the plots used in this paper best:

$$D_{i,j} = D_{i,j}(d \leq 3\sigma) = \begin{cases} 3\sigma & d \geq 3\sigma \\ d & d < 3\sigma \end{cases} \quad . \tag{4}$$

The resulting plots show a gray-shaded pattern opposed to classical RP that have a simple black and white pattern. Two samples taken from UCR archives Adiac dataset can be found in Fig. 1.

Compared to methods like WEASEL or BOSS, our proposed method does not put much emphasis on extracting the best hyper-parameters for each dataset. The only parameter required in its configuration is the image size. By using the unthresholded RPs we got rid of the ϵ parameter used in traditional RPs which makes configuration easier and also leads to a plot that contains more information. By using Euclidean distance by default and not trying to determine the best distance measure for each dataset we reduce configuration even more. The individual datasets in the UCR archive are treated equally and no individual adaption is performed. Within every experiment, the dataset is preprocessed in the same way, and the same resolution is used for all images in order to be as generic as possible. It is reasonable to assume that this will lead to information loss, but through this inter-class normalization we gain more easily comparable images of equal size. To compensate for the in part heavily skewed datasets, we used class weights during training.



(a) Class A from the Adiac dataset (b) Class B from the Adiac dataset

Fig. 1: Distance Plots of two classes from the Adiac dataset, where classes represent the outlines of unicellular algae taken from images.

2.2 Experimental Setup

Experiments were run on a HP Server with 1 TB of RAM, two Intel Xeon E5-2650(28 cores @ 2.60 GHz), and NVIDIA Tesla V100 graphics cards. The server was running Ubuntu 16.04 LTS. To improve reproducibility, Docker containers were used during training. Docker encapsulates the runtime environment for the experiments. That way the environment is easy to recreate and version, as the runtime environment is simply described in structured text files.[3] All neural networks used in this paper were implemented using the Keras API with a TensorFlow back-end [5, 1]. Keras also has a ready-to-go implementation of a Deep Residual Network (DSN) with 50 layers that will be referred to as ResNet-50 in this paper. Also a DSN with 152 layer was used.[4] Its implementation is based on code published on github.[5] It will be referred to as ResNet-152. The networks were trained with categorical cross entropy as the loss function, which is widely used for multi class problems, and used a Stochastic Gradient Descent (SGD) optimizer. The results were evaluated using the models' accuracy.

3 Data

The datasets used in this paper are all taken from the UCR Time Series Archive that was first introduced in 2002 and since then has been used in more than one thousand publications. The archive got an update in 2018, expanding it from 85 to 128 datasets. The new datasets on average have a higher number of training samples and also contain sets with variable length time series to

[3] `https://www.docker.com/`

[4] `https://github.com/flyyufelix/cnn_finetune/blob/master/resnet_152.py`

[5] `https://gist.github.com/previtus/c1a8604a4a07de680d5fb05cebfdf893`

represent many real-world problems [6], ranging from the movement of insect wings to the energy consumption profile of electronic devices. Most datasets in the archive have already been Z-score normalized. For the sets that were not yet normalized, Z-score normalization was applied. The datasets are split in fixed train and test sets. The authors of the archive argue that the fixed train/test splits lead to more reproducible results and make publications and methods easier to compare. The sizes of the individual datasets vary significantly and in some of the sets, the class distribution is strongly skewed and class distributions of test and train set also differ. For example, the Chinatown train set consists of only 20 samples for 2 classes with 10 samples each, but the test set contains 95 samples of one class and 250 of the second class.

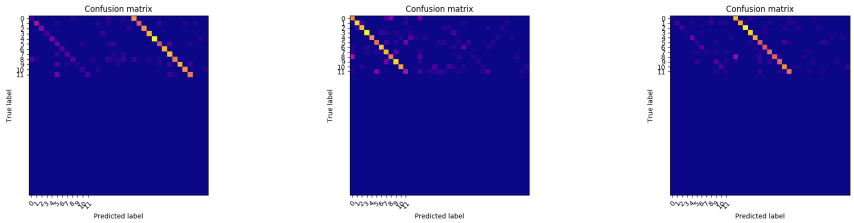
Another dataset that has at least indirectly been used in this paper is ImageNet, an image database containing millions of pictures[6]. The publishers of the dataset hold an annual competition, the Large Scale Visual Recognition Challenge (ILSVRC). The challenge was won with a Deep Residual network as proposed in [9]. The network together with weights obtained by training on the ImageNet dataset was published and is widely used in transfer learning applications.[7] [8] [9]

4 Experiments

To the best of our knowledge, there have been no publications using the complete UCR 2018 archive at the time of writing of this paper. Some results on the new dataset were only recently presented by [12]. Therefore, we present our results on the UCR 2018 archive separated into two tables. The datasets already contained in the 2015 archive are compared with results from relevant publications and the new datasets are compared to the Dynamic Time Warping (DTW) published on the archives website[10, 11].

In total we ran 3942 experiments during our work on this paper. Detailed results of all the experiments performed can not be properly presented nor discussed. We want to primarily present two methods. They will be referred to as All Classifier (AC) and Single Classifier (SC). The AC was trained on all images from the UCR archive at once with its 1118 overall classes, i.e. the AC network has 1118 targets using one-hot encoding. The SC is the classical approach to train one DNN per dataset. The AC was evaluated in the same way as the SC, using only one dataset at a time, resulting in individual accuracy for each dataset. Final results are presented for gray-scale images with a resolution of 224x224, the resolution used in the original ResNet publication. We experimented with different higher resolutions up to 512x512, but this did not

[6] `http://image-net.org/`
[7] `https://keras.io/applications/#resnet50`
[8] `https://www.mathworks.com/help/deeplearning/ref/resnet50.html`
[9] `https://pytorch.org/docs/0.4.0/_modules/torchvision/models/resnet.html`
[10] `https://www.cs.ucr.edu/~eamonn/time_series_data_2018`
[11] Online results at `https://github.com/patientzero/timage-icann2019`



(a) CricketX (b) CricketY (c) CricketZ

Fig. 2: Confusion Matrix for AC on the Cricket datasets, which are accelerometer data taken from actors performing Cricket gestures.

improve classification accuracy significantly. For some data in the UCR archive this means length reduction, performed using average pooling after calculating the RP matrix. We also tried non gray-scale pictures with three colors channels by mapping the distance matrix to a color scheme, which lead to overall worse results. Detailed results on the 2015 datasets are shown in Table 2, results on 2018 datasets in Table 3.

The AC did exceed our expectations of it being nothing but an interesting idea by far. Results for some datasets are very good and come close to state of the art systems and results from the SC, but for some datasets it completely fails, being worse than the default rate. The default rate is defined by the accuracy a classifier would achieve by always predicting the most probable class. We need to keep in mind that using a single classifier for all datasets within the UCR archive increases the complexity dramatically since it requires separating the datasets as well. Maybe hyper-parameter tuning with special regard to the huge amount of classes might lead to better overall results.

Detailed results confirm some of the things we know about recurrence plots and the conclusions that can be drawn about a time series by looking at the plot. Patterns found in the images by the DNN will probably be too similar to be differentiated properly. An in-depth look at misclassifications revealed that some of the very poor results were almost completely misclassified within a single other dataset. An example of such a plot can be found in Fig. 2. The datasets are very similar in general. An accuracy of 64% could be reached with both AC classifiers, which is not particularly good but still better than the baseline (DTW) published by the authors. The AC completely fails to classify anything correctly within CricketX and CricketZ. Both datasets were almost exclusively classified as classes belonging to the CricketY dataset. A probable explanation for this can be found in shuffling and batching of training data. Shuffling the training data before feeding it in batches to the network may lead in case of very similar classes to over prioritizing earlier seen classes, thus failing to learn the classes later seen. In other random states or in a different order the (mis-)classification results might be reversed.

Table 1: Histograms of relative accuracy changes of SC models compared to Weasel and LSTM on (a) UCR 2015 and baseline (DTW) on (b) UCR 2018.

Rel. Diff. Accu Range		ResNet-50		ResNet-152	
		Weasel	LSTM	Weasel	LSTM
	< -0.80	0	0	0	0
>= -0.80	< -0.40	1	0	1	0
>= -0.40	< -0.20	2	0	2	1
>= -0.20	< -0.10	6	2	5	0
>= -0.10	< -0.05	7	1	8	0
>= -0.05	< -0.01	20	3	14	6
>= -0.01	< 0.00	5	1	7	1
0.00	0.00	5	7	5	4
> 0.00	<= 0.01	6	7	9	9
> 0.01	<= 0.05	12	23	13	19
> 0.05	<= 0.10	7	15	11	22
> 0.10	<= 0.20	7	14	5	16
> 0.20	<= 0.40	5	10	4	6
> 0.40	<= 0.80	2	2	1	1
> 0.80		0	0	0	0

(a)

Rel. Diff. Accu Range		ResNet-50	ResNet-152
		DTW	DTW
	< -0.80	2	3
>= -0.80	< -0.40	1	2
>= -0.40	< -0.20	8	5
>= -0.20	< -0.10	9	11
>= -0.10	< -0.05	3	4
>= -0.05	< -0.01	3	2
>= -0.01	< 0.00	0	0
0.00	0.00	0	0
> 0.00	<= 0.01	2	1
> 0.01	<= 0.05	3	2
> 0.05	<= 0.10	2	5
> 0.10	<= 0.20	4	0
> 0.20	<= 0.40	3	5
> 0.40	<= 0.80	3	3
> 0.80		0	0

(b)

Since there has been more research done on SC classifiers on the UCR archive we also present results in detail. Table 1 shows the relative accuracy differences of the SC timage compared to state of the art systems. Table 1(a) shows the differences for the UCR 2015 archive compared to LSTM and WEASLE [13, 18] and Table 1(b) for the new datasets in the UCR 2018 archive compared to the baseline. For the UCR 2015 *timage* outperforms WEASEL using ResNet-50 for 41 datasets, and using ResNet-152 for 37 datasets. Same accuracy is achieved for 5 datasets for both ResNet-50 and ResNet-152. Compared to the LSTM, *timage* performs better for 7 datasets using ResNet-50 and for 8 datasets using ResNet-152. Equal results were achieved in 7 cases using ResNet-50 and 4 using ResNet-152. For 18 datasets relative classification accuracy was still comparable and only within a 5% relative difference using ResNet-50 compared to WEASEL and 22 for ResNet-152. Respectively for LSTM, 30 are within the 5% range using ResNet-50 and 28 using ResNet-152. For 8 datasets from the UCR 2015 archive, timage could achieve the best classification accuracy so far as shown in Table 2. Comparing the results for the UCR 2018 archive with the baseline, timage outperforms it in 26 out of 41 cases using ResNet-50 and 27 using ResNet-152. For 5 datasets results were within the 5% range using ResNet-50 and 3 using ResNet-152.

The results that can be achieved with the SC are comparable to state of the art systems as shown by Table 2. The method is able to classify all datasets well with the exception of ShapeletSim and TwoPatterns, where it is still better than the default rate.

Table 2: Accuracy of experiments on UCR 2015 archive. Best results are highlighted.

			SC		AC	
Dataset	**WEASEL**	**F-t LSTM-FCN**	**ResNet50**	**ResNet152**	**ResNet50**	**ResNet152**
Adiac	0.8312	**0.8849**	0.8440	0.8235	0.7238	0.6982
ArrowHead	0.8571	**0.9029**	0.8857	0.8629	0.8000	0.7486
Beef	0.7667	**0.9330**	0.7333	0.7667	0.8333	0.7000
BeetleFly	0.9500	**1.0000**	0.9000	0.9000	0.2000	0.2500
BirdChicken	0.8000	**1.0000**	0.9000	**1.0000**	0.1500	0.0500
Car	0.8667	**0.9670**	0.8833	0.9000	0.8500	0.8167
CBF	0.9833	**1.0000**	0.9044	0.9911	0.9478	0.9100
ChlorineConcentration	0.7526	**1.0000**	0.7844	0.7852	0.7237	0.7109
CinCECGTorso	**0.9935**	0.9094	0.8913	0.8580	0.9080	0.8971
Coffee	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
Computers	0.6560	**0.8600**	0.7400	0.6960	0.2960	0.2080
CricketX	0.7641	**0.8256**	0.7359	0.7333	0.1385	0.1564
CricketY	0.7897	**0.8256**	0.7359	0.7538	0.6462	0.6410
CricketZ	0.7872	**0.8257**	0.7282	0.7615	0.0897	0.0923
DiatomSizeReduction	0.8856	**0.9771**	0.9379	0.9346	0.8693	0.7941
DistalPhalanxOutlineAgeGroup	0.7698	**0.8600**	0.7842	0.7842	0.0216	0.0432
DistalPhalanxOutlineCorrect	0.7790	**0.8217**	0.8152	0.8080	0.2428	0.1014
DistalPhalanxTW	0.6763	**0.8100**	0.7194	0.6835	0.0432	0.0144
Earthquakes	0.7482	**0.8261**	0.7770	0.7986	0.6331	0.6475
ECG200	0.8500	0.9200	0.8700	**0.9400**	0.7900	0.7900
ECG5000	**0.9482**	0.9478	0.9458	0.9442	0.9358	0.9307
ECGFiveDays	**1.0000**	0.9942	0.8165	0.9024	0.9477	0.8269
ElectricDevices	0.7329	**0.7633**	0.7313	0.7297	0.6956	0.7181
FaceAll	0.7870	**0.9680**	0.7497	0.7828	0.7793	0.7704
FaceFour	**1.0000**	0.9772	0.7273	0.9091	0.8182	0.7273
FacesUCR	0.9522	**0.9898**	0.7771	0.8566	0.8371	0.8215
FiftyWords	**0.8110**	0.8066	0.7978	0.7868	0.7407	0.7253
Fish	0.9657	**0.9886**	0.9771	0.9771	0.9371	0.9257
FordA	0.9727	**0.9733**	0.9386	0.9235	0.6295	0.6667
FordB	0.8321	**0.9186**	0.8222	0.8074	0.5432	0.4926
GunPoint	**1.0000**	**1.0000**	**1.0000**	0.9933	0.9867	0.9867
Ham	0.6571	**0.8000**	0.7905	0.7429	0.7333	0.6952
HandOutlines	0.9487	0.8870	**0.9514**	0.9297	0.9162	0.9216
Haptics	0.3864	**0.5584**	0.5162	0.4968	0.4643	0.4351
Herring	0.6563	**0.7188**	0.6875	0.6563	0.5781	0.6094
InlineSkate	**0.6127**	0.5000	0.4036	0.4309	0.3055	0.3655
InsectWingbeatSound	0.6404	**0.6696**	0.6177	0.6212	0.5944	0.5894
ItalyPowerDemand	0.9514	**0.9699**	0.9602	0.9602	0.9475	0.9397
LargeKitchenAppliances	0.6827	**0.9200**	0.8080	0.7573	0.6267	0.4987
Lightning2	0.5574	0.8197	**0.8525**	**0.8525**	0.3279	0.2295
Lightning7	0.7123	**0.9178**	0.8493	0.7945	0.3014	0.3425
Mallat	0.9655	**0.9834**	0.9326	0.9441	0.9365	0.9032
Meat	0.9167	**1.0000**	**1.0000**	0.9833	0.8500	0.8833
MedicalImages	0.7408	**0.8066**	0.7868	0.7803	0.7276	0.7184
MiddlePhalanxOutlineAgeGroup	0.6039	**0.8150**	0.6364	0.6234	0.0000	0.0000
MiddlePhalanxOutlineCorrect	0.8076	0.8333	0.8522	**0.8591**	0.1959	0.1134
MiddlePhalanxTW	0.5390	**0.6466**	0.6234	0.5909	0.0195	0.0260
MoteStrain	0.9353	**0.9569**	0.8403	0.8866	0.4968	0.5719
NonInvasiveFetalECGThorax1	0.9288	**0.9657**	0.9405	0.9405	0.9196	0.9288
NonInvasiveFetalECGThorax2	0.9415	**0.9613**	0.9522	0.9547	0.9328	0.9405
OliveOil	**0.9333**	**0.9333**	**0.9333**	0.8667	0.5333	0.5000
OSULeaf	0.8967	**0.9959**	0.8678	0.8512	0.7355	0.6488
PhalangesOutlinesCorrect	0.8170	0.8392	0.8578	**0.8590**	0.1643	0.2960
Phoneme	0.3265	**0.3602**	0.2410	0.2416	0.2152	0.1930
Plane	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.9810
ProximalPhalanxOutlineAgeGroup	0.8488	**0.8878**	**0.8878**	0.8732	0.0049	0.0000
ProximalPhalanxOutlineCorrect	0.8969	**0.9313**	0.9244	0.9278	0.3814	0.3883
ProximalPhalanxTW	0.8098	**0.8275**	0.8049	0.8195	0.0634	0.0293
RefrigerationDevices	0.5387	**0.5947**	0.5520	0.5440	0.4480	0.4267
ScreenType	0.5467	**0.7073**	0.4640	0.4720	0.2880	0.2853
ShapeletSim	**1.0000**	**1.0000**	0.5556	0.6333	0.4500	0.6667
ShapesAll	**0.9183**	0.9150	0.8850	0.8600	0.7600	0.7400
SmallKitchenAppliances	0.7893	**0.8133**	0.7333	0.6880	0.6587	0.6853
SonyAIBORobotSurface1	0.8236	**0.9967**	0.8802	0.9601	0.7953	0.7488
SonyAIBORobotSurface2	0.9349	**0.9822**	0.8143	0.8458	0.7650	0.7870
StarLightCurves	0.9773	0.9763	**0.9806**	**0.9806**	0.9709	0.9677
Strawberry	0.9757	**0.9864**	0.9811	0.9838	0.9676	0.9622
SwedishLeaf	0.9664	**0.9840**	0.9680	0.9616	0.9312	0.9232
Symbols	0.9618	**0.9849**	0.9719	0.9668	0.7739	0.8000
SyntheticControl	0.9933	**1.0000**	0.7133	0.6700	0.6767	0.6767
ToeSegmentation1	0.9474	**0.9912**	0.9167	0.9211	0.5570	0.5263
ToeSegmentation2	0.9077	**0.9462**	0.9385	0.8846	0.4077	0.4692
Trace	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
TwoLeadECG	0.9982	**1.0000**	0.9895	0.9956	0.9816	0.9974
TwoPatterns	0.9898	**0.9973**	0.5157	0.5145	0.5028	0.4903
UWaveGestureLibraryAll	0.9503	**0.9609**	0.9375	0.9436	0.9137	0.9112
UWaveGestureLibraryX	0.8096	**0.8498**	0.7074	0.7004	0.4738	0.4693
UWaveGestureLibraryY	0.7247	**0.7661**	0.7513	0.7133	0.5419	0.5034
UWaveGestureLibraryZ	0.7700	**0.7993**	0.7289	0.7052	0.4913	0.4740
Wafer	**1.0000**	**1.0000**	0.9977	0.9966	0.9971	0.9942
Wine	0.8519	**0.8890**	0.6667	0.8333	0.5000	0.6667
WordSynonyms	**0.7241**	0.6991	0.6850	0.6771	0.6348	0.6176
Worms	0.8052	0.6851	0.8182	**0.8442**	0.4156	0.3506
WormsTwoClass	0.8052	0.8066	**0.8961**	0.8442	0.2857	0.2727
Yoga	0.9090	**0.9163**	0.9000	0.8827	0.8730	0.8713

Table 3: Accuracy of experiments on new entries of UCR 2018 archive. Best results are highlighted.

		SC		AC	
Dataset	**DTW**	**ResNet50**	**ResNet152**	**ResNet50**	**ResNet152**
ACSF1	0.6400	0.7800	**0.7900**	0.5700	0.5300
AllGestureWiimoteX	**0.7157**	0.4943	0.5200	0.3757	0.3829
AllGestureWiimoteY	**0.7286**	0.6000	0.5629	0.4500	0.4586
AllGestureWiimoteZ	0.6429	**0.6514**	0.5871	0.5014	0.4943
BME	0.9000	**1.0000**	**1.0000**	0.9200	0.9533
Chinatown	**0.9565**	0.7246	0.7565	0.0058	0.0087
Crop	0.6652	**0.7559**	0.7532	0.7405	0.7377
DodgerLoopDay	0.5000	0.4875	**0.5125**	0.4000	0.3375
DodgerLoopGame	**0.8768**	0.6812	0.6812	0.0000	0.0870
DodgerLoopWeekend	0.9493	0.9420	**0.9710**	0.1667	0.0435
EOGHorizontalSignal	**0.5028**	0.2790	0.2652	0.1160	0.1326
EOGVerticalSignal	**0.4475**	0.2238	0.2569	0.1547	0.1215
EthanolLevel	0.2760	**0.8400**	0.7100	0.7100	0.7100
FreezerRegularTrain	0.8989	0.9961	**0.9975**	0.9221	0.9565
FreezerSmallTrain	0.7533	**0.9793**	0.9354	0.0667	0.0368
Fungi	0.8387	0.8871	**0.9194**	0.8763	0.8548
GestureMidAirD1	0.5692	0.5308	**0.6000**	0.4077	0.4231
GestureMidAirD2	0.6077	**0.6231**	0.5538	0.4923	0.4846
GestureMidAirD3	0.3231	0.4154	**0.4615**	0.3923	0.2231
GesturePebbleZ1	0.7907	**0.9186**	**0.9186**	0.4070	0.3547
GesturePebbleZ2	0.6709	**0.8544**	0.8354	0.0506	0.0949
GunPointAgeSpan	0.9177	**0.9873**	0.9842	0.1582	0.0886
GunPointMaleVersusFemale	**0.9968**	0.9937	0.9937	0.2247	0.2658
GunPointOldVersusYoung	0.8381	**0.9810**	**0.9810**	0.1873	0.2190
HouseTwenty	**0.9244**	0.8319	0.8403	0.6807	0.6218
InsectEPGRegularTrain	0.8715	**0.9719**	0.9639	0.8273	0.6988
InsectEPGSmallTrain	0.7349	**0.9438**	0.8795	0.1004	0.2088
MelbournePedestrian	**0.7906**	0.3604	0.3563	0.3298	0.3282
MixedShapesRegularTrain	0.8416	**0.9645**	0.9509	0.8470	0.8276
MixedShapesSmallTrain	0.7798	**0.9027**	0.8635	0.0247	0.0049
PickupGestureWiimoteZ	0.6600	0.7400	**0.8000**	0.4800	0.4800
PigAirwayPressure	0.1058	0.1442	**0.1683**	0.0913	0.1106
PigArtPressure	0.2452	0.3510	**0.5288**	0.1635	0.2548
PigCVP	0.1538	0.4279	**0.5288**	0.3462	0.3221
PLAID	**0.8399**	0.8231	0.8119	0.7877	0.7858
PowerCons	0.8778	0.9389	**0.9722**	0.9333	0.9000
Rock	0.6000	0.7800	**0.8400**	0.7400	0.6000
SemgHandGenderCh2	**0.8017**	0.7867	0.7767	0.1900	0.2117
SemgHandMovementCh2	**0.5844**	0.5244	0.5267	0.1422	0.1911
SemgHandSubjectCh2	**0.7267**	0.6644	0.6889	0.3133	0.2578
ShakeGestureWiimoteZ	0.8600	0.8800	**0.9400**	0.5400	0.6400
SmoothSubspace	0.8267	**0.9933**	0.9867	0.9800	0.9533
UMD	**0.9931**	0.8264	0.7847	0.6944	0.6528

5 Conclusion

Especially in the light of the huge diversity of datasets in the UCR archive, the robustness of using transfer learning on ResNet to classify time series by using RP is surprising. A number of changes in the preprocessing step like increasing the resolution or using a color scheme did not help the DNN to separate the classes more accurately compared to our initial experiments. Different strategies to obtain weights for transfer learning did not lead to significant improvements but had impact on the time networks needed to converge. The classification pipeline presented generally simplifies what needs to be done to detect patterns and correctly classify time series.

The analysis of confusion matrices for datasets that performed extremely bad with the AC show that some of them almost exclusively misclassified within one other dataset. They do not fail the task of separating classes within one dataset, but fail on the harder problem of an AC, that is separating not only classes within a dataset, but to separate very similar datasets as well. This supports some of the findings regarding RP for time series, and the conclusions regarding its properties, that can be drawn from looking at the respective RP: Similar time series produce very similar patterns [14].[12] Thus failing to always boost more easily separable features.

The nature of the UCR archive simplifies the classification task, because due to mostly fixed length and nearly perfect alignment of time series, some difficulties, like segmentation, faced in real world scenarios are removed. In the case of the UCR archive all data is pre-segmented, which in itself is a very hard task to begin with. Some of the harder tasks are included in the UCR 2018 archive where some datasets contain time series of different length and overall classification results are not as good as for the UCR 2015 archive. In the future it will be interesting to adapt and evaluate *timage* to real world scenarios where, for example, a sliding window is used.

Acknowledgement

This work is supported by a research grant of the Bayerisches Staatsministerium für Bildung und Kultus, Wissenschaft und Kunst and further by the Bayerische Wissenschaftsforum (BayWISS).

[12] `http://www.recurrence-plot.tk/`

References

1. Abadi, M., Agarwal, A., Barham, P.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015), https://www.tensorflow.org/
2. Alessio, S.M.: Digital Signal Processing and Spectral Analysis for Scientists. Springer International Publishing (2016)
3. Bagnall, A., Lines, J., Bostrom, A., Large, J., Keogh, E.: The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. Data Mining and Knowledge Discovery **31**(3), 606–660 (May 2017)
4. Boddapati, V., Petef, A., Rasmusson, J., Lundberg, L.: Classifying environmental sounds using image recognition networks. Procedia Computer Science **112**, 2048–2056 (2017)
5. Chollet, F., et al.: Keras (2015), https://keras.io
6. Dau, H.A., Keogh, E., Kamgar, K., Yeh, C.C.M., Zhu, Y., Gharghabi, S., Ratanamahatana, C.A., Yanping, Hu, B., Begum, N., Bagnall, A., Mueen, A., Batista, G.: The ucr time series classification archive (October 2018), https://www.cs.ucr.edu/~eamonn/time_series_data_2018/
7. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016)
8. Hatami, N., Gavet, Y., Debayle, J.: Classification of time-series images using deep convolutional neural networks. Proc. SPIE 2018 (2017)
9. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. CoRR (2015)
10. Ismail Fawaz, H., Forestier, G., Weber, J., Idoumghar, L., Muller, P.A.: Deep learning for time series classification: a review (2018), https://arxiv.org/abs/1809.04356
11. Ismail Fawaz, H., Forestier, G., Weber, J., Idoumghar, L., Muller, P.A.: Transfer learning for time series classification. In: IEEE International Conference on Big Data. pp. 1367–1376 (2018)
12. Karim, F., Majumdar, S., Darabi, H.: Insights into lstm fully convolutional networks for time series classification (2019), https://arxiv.org/abs/1902.10756
13. Karim, F., Majumdar, S., Darabi, H., Chen, S.: Lstm fully convolutional networks for time series classification. IEEE Access **PP** (09 2017)
14. Marwan, N., Carmenromano, M., Thiel, M., Kurths, J.: Recurrence plots for the analysis of complex systems. Physics Reports **438**(5-6), 237–329 (jan 2007)
15. Michael, T., Spiegel, S., Albayrak, S.: Time series classification using compressed recurrence plots (09 2015)
16. Park, T., Lee, T.: Musical instrument sound classification with deep convolutional neural network using feature fusion approach. CoRR **abs/1512.07370** (2015)
17. Schäfer, P.: The boss is concerned with time series classification in the presence of noise. Data Mining and Knowledge Discovery **29**(6), 1505–1530 (Nov 2015)
18. Schäfer, P., Leser, U.: Fast and accurate time series classification with WEASEL. CoRR **abs/1701.07681** (2017), http://arxiv.org/abs/1701.07681
19. Wang, Z., Oates, T.: Imaging time-series to improve classification and imputation. CoRR **abs/1506.00327** (2015), http://arxiv.org/abs/1506.00327