Report challenge 2

Time series classification for ANN and Deep Learning

Francesca Forbicini
Vincenzo Greco
Nicolò Fontana

21 December 2022

1 Introduction

The aim of the challenge was to perform times series classification over a 12-classes extremely imbalanced dataset.

2 Evolution of the network

2.1 From first attempts...

We started building our own networks using the ones given us during the exercise lessons as reference. The first models were: a simple LSTM, a bidirectional LSTM, a fully connected network, and a mix of LSTM layers with Dense ones put either in series or in parallel. Despite the variety all of them performed badly. It is worth noticing that, even in theory an LSTM network would have performed worse than a perceptron in this case since the provided window was too narrow to emphasize the memory property and also, usually, a memory-based approach is more suited for prediction tasks rather than classification ones (as the one we were facing).

2.2 ...to convolution...

So, once we noticed that the size of the window was fixed (always 36 points for each sample) and that the whole input was known at training and, as consequence, there was no need to exploit any memory, we moved to a 1D convolutional network, which allowed us to reach meaningful results in terms of accuracy. Since the convolutional network yielded the best result even in the previous challenge we decided to try to pass the input as a 1x36 image with 6 channels and use 2D convolution, but this led to no significant improvement, so we stuck with the 1D one. At this point, we started to intensively tune the hyperparameters and find the right preprocessing and data augmentation reaching what would have been our best model and result.

2.2.1 ...with a little deja vu...

We tried to reuse also another kind of architecture that helped us in the previous challenge. In particular, we implemented a model using EfficientNet, adapting it to one dimension rather than two. The performance dropped. We believe that was due to the complexity of the model that was overwhelming with respect to the simplicity of the dataset. Anyway, one important factor was also missing; indeed transfer learning usually leverages on the fact that the transferred network, having been trained on huge and extensive dataset, is able to generalize, giving to the top custom network the easier task to specialize that knowledge to the specific task that it is trained for. This main aspect is founded on the assumption that the specific task belongs to the same conceptual category of the more general one of the feature extractor; assumption that is very likely to be absent in this case.

2.3 ...and beyond

Not satisfied with the achieved result (the model was performing 0.75 on validation and 0.72 on testing) we decided to shift to different approaches. Each one of us tackled one them to have a broader search space.

2.3.1 Transformers

We tried to use a transformer, substituting the embedding layer with a linear one, since the input was composed of continuous values. We tried to use different numbers of heads and layers, but, unfortunately, it performed worse than our best model and due to small remaining time and the complesxity we preferred to focus more on the autoencoder approach.

2.3.2 Multiple classifiers

We tried to isolate those classes which were making our dataset imbalanced by training dedicated classifiers. In particular, we tried two paths. First, we trained two classifiers between classes 4-9 and 11-3 (since many samples of the first class of each pair were misclassified as of the second class) combined with a 12 classifier. Second, we used 5 classifiers: an initial one for 12-class task, and then 2 pairs of 1vsALL classifier and 11-class classifiers in those cases where the intial one returned 3 or 9 as result (if the 1vsALL confirmed the initial prediction over 12 classes that would have been the result, otherwise the 11-class classifier would have predicted which one of the other classes the sample was belonging to). None of both, albeit with different magnitudes, performed well enough.

2.3.3 Autoencoders

We tried also to train 12 autoencoders, one for each class, with the idea that when a new sample had to be predicted it would have been passed through all of them, and the best autoencoder to reconstruct it would have given us the predicted class (since it was trained only on samples of that class so it would have seemed natural for it to accurately reconstruct only samples belonging to its class). The main problem we feared to face with this approach was to obtain autoencoders good enough to allow the needed distinction, instead we the exact opposite problem: indeed every autoencoder (regardless from the class it was trained on) was able to perfectly encode and decode every class. In particular with the exception of class 9 where the predictions were slightly visibly off any other class was matched with unnatural precision. In addition, all of this was happening even with the simplest autoencoder possible (taking 1 sample of 36x6=216 input, encoding it in 1 central neuron, and having an output of again 216 units). Sadly, even if the result was astonishing we couldn't use it for our scope, because since there was no difference among the performance of all 12 it was impossible to understand which class the new sample was belonging to.

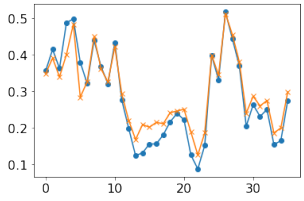

Figure 1: Autoencoder of class 0 predicts class 0

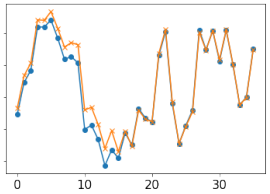

Figure 2: Autoencoder of class 0 predicts class 1

3 Preprocessing and data augmentation

3.1 Worthy ideas

3.1.1 Gaussian noise

It seemed to us the more natural type of preprocessing for a time series since it preserves the order of the points in the samples while adding some diversity to them. Like in the previous challenge we to decided to embed it into the network using the apposite Keras layer at the beginning of our model. Indeed it revealed itself a valuable preprocessing and thus we kept it in every model we built.

3.1.2 Scaling

We found the RobustScaler provided by scikit-learn to be the best one when using classifiers, while the MinMaxScaler allowed us to obtain the incredible result of the perfect 1-neuron autoencoder. Unfortunately, both of them, if used on the other approach (Robust on autoencoders and MinMax on classifiers) made the performance drop.

3.1.3 Imbalanced dataset

Both oversampling and undersampling, through the use of the random samplers from the imblearn library, provided a good measure against the overfitting, but without any other relevant change in the performance of the models they were applied to. Also, in some cases, they even reduced the performance so we decided to use them only when this was not happening.

3.1.4 Feature selection

Fixed the model we tried various combinations of features starting from the 6 given to us, the only subset

which seemed to us to have an impact (in detail: reducing the overfitting of the model) was when the feature 0 was dropped, so we decided to use it in the final model

3.2 Discarded ideas

3.2.1 Scaling

The one scaler which performed badly both using the classifiers and the autoencoders was the Standard one, so we resorted to other ones (Robust and MinMax as said above)

3.2.2 Window size change

We tried to reduce the window size to extract more information from a single sample, but we didn't achieved any particular result. On the other hand we dropped by the principle the idea of enlarging it because we didn't know how the samples were generated so we couldn't have known if the dataset given to us had a particular order of the samples (other than the obvious one of the class in ascending order) and then how we could have concatenated the samples without adding some unwanted bias.

3.2.3 Zeroing values

We opted also to put to zero some parts of the samples to reduce the noise of each time series to search for patterns otherwise unnoticed. But also this attempt gave us any significant outcome.

3.2.4 Shifting values

Another approach we experimented with was to shift the points in each sample forward and backward, duplicating the ones which exited the window on the other end of the sample (e.g. forwarding the TS of 3 points would have made the last 3 points to be concatenated at the front of the sample instead of at the end). The main result obtained from this was useless to the aim of improving performance, but it has been extremely helpful in giving us valuable insights on the actual nature of the sample. Indeed doing this any performance from the classifier was lost giving the confirmation that actually there was a time link between the points in each sample, so that those were actually time series.

3.2.5 Class weight

While under and oversampling gave us mixed results, the application of class weights to put more emphasis on the less represented classes yielded even more deteriorated performance than the worst cases with sampling. So it was promptly abandoned.

4 Conclusions

What we were asked to deal with in this challenge made us realize even more the importance of data in deep learning, indeed their scarcity led the network to just forget the existence of the less represented classes. Furthermore, not knowing the dataset gave us little to no way to try to approach the problem in other ways. For example, in the previous challenge we had the possibility to exploit many different types of data augmentation (e.g. even discard almost half of the image) since the images had no particular orientation or area of concentration of meaningful information. From this, we can understand how important it is sometimes to know the environment we are working with. Furthermore, we may have some hypotheses on how the data were collected or, at least in our opinion, generated. Due to the performance of our autoencoder, we were wondering if the dataset was generated by an algorithm or sampled by observing a highly controlled environment. This is because, in our understanding, a single central neuron of an autoencoder would not have enough bits to be able to store the entropy of data coming from an extremely noisy environment such as the real world.