Artificial Neural Networks and Deep Learning
Homework 2

Nicola Dean	Marco Fasanella	Raffaello Fornasiere	Christian Spano
10674826	10617541	10790353	10823764
nicola.dean @mail.polimi.it	marco.fasanella @mail.polimi.it	raffaello.fornasiere @mail.polimi.it	christian.spano @mail.polimi.it

1 Introduction

In this homework our approach has been to predict the target variables at a future time point based on the learning from previous time points.
To keep the problem tractable, learning methods used use data from a fixed length window in the past as an explicit input. the range of possible solutions was limited compared to the previous task, in this case the tests done included RNN, Resnet, LSTM and 1D CNN models.
In order to ease the development of our best solution, we thought it would have been useful to create an ad-hoc helper.
As a matter of fact, a lot of code is often repeated or many lines of code are needed just to plot or to get insightful results (such as the confusion matrix or the accuracy). In particular it has been useful to build sequences and data augmentation (dealing with randomness and noise).

2 Preprocessing

When it comes to TimeSeries classification/forecasting, preprocessing is a crucial step to grant optimal results. In this section will be described all the preprocessing techniques we tried or discarded.

2.1 Normalization

After visualizing the data, we noticed that samples has different scale factor based on targets. To avoid advantaging some of the classes we decided to normalize the dataset.

2.1.1 Per Column

As first approach, we normalized the dataset by **FULL columns**

- **MinMax** Does not work properly; When applied, the validation accuracy dropped to 33
- **Mean / Std** This technique does not improve the accuracy on vanilla models.

2.1.2 Per Sample

The next method we tried, was to normalize each sample (TimeSeries) independently, to completly remove the different scales of the targets.

- **MinMax** Improve performances but not enough, at least on vanilla models
- **Mean / Std** This was the **Game Changer** for us, it improves the performances on Vanilla models from 63% to 67% on the Vanilla BiLstm

2.2 Augmentation

In the Homework1 challenge, we noticed the importance of augmentation on this field and so we decided to give it a try also on time series. We tried both "by hand" and Library approach with the following results.

2.2.1 Numpy

Using the function *np.random.normal* it is possible to generate random noise with a certain mean std and shape. To obtain augmented samples we copied the dataset multiple time, and then, for each time serie, we calculated it's standard deviation and applied the augmentation like follow:

$$x = x + np.random.normal(0, std, x.shape) * w$$

Figure 1: With x be a single time series, w be a weight of our choice

The result of augmentation was immediate and bring us a solid 68% using the vanilla BiLstm (that was performing only 67% without augmentation)

2.2.2 TSUNG/Librery

RAFFAELLO

2.3 Seasonal + Trend preprocess

Christian

2.4 Expanding Window size

2.5 Adding New Features

Christian

3 Vanilla Models

We first approached a Vanilla model in order to understand if we could easily tackle the classification

problem with a 1D Convolutional Neuronal Network.
The best configuration we found consisted in 3 convolutions followed by a Dropout and 2 Dense layers:

This model unfortunately arrived at a result of maximum 0.50 in terms of Accuracy, reached with a grid search on filters,kernels and dropout.
We then moved to a Vanilla LSTM and Bidirectional-LSTM, but with poor results.

4 Net Concatenations

Now that our Preprocessing was giving promising results and the possible changes to vanilla models was not giving results better than 68.0% we decided to try combining LSTM and CNN into a single network.

4.1 CNN + LSTM

At first we combined an CNN with a LSTM thinking to use the CNN as feature extractor for the LSTM. This was giving us the same results of BiLstm + Norm + Augm.

4.2 LSTM + CNN

We also tried using LSTM to analyze the time series and then CNN as feature extractor for the analyzed data but the result was the same of BiLstm also in this case.

5 Inspired By Existing Models

5.1 InceptionNet

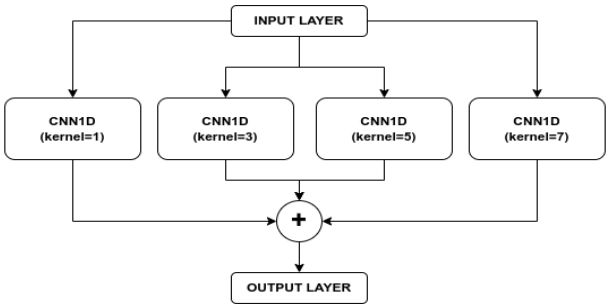

Figure 2: CNN1D Inception Like Net

5.2 Resnet

5.2.1 Resnet-CNN1D

Resnet Model works by adding a big number of layer blocks alternated by some skip layers to avoid vanishing gradient descent. We replicated this model in keras using Conv1D instead of the original Conv2D but unfortunatly it has not improved performances and we were still stucked to 68%.

5.2.2 Resnet-LSTM

A second step we had done was to replace the CNN1D with some LSTM like in figure3, This improved the performances by 0.9% respect to our best model so far, bringing the accuracy on codalab to 68.9%.

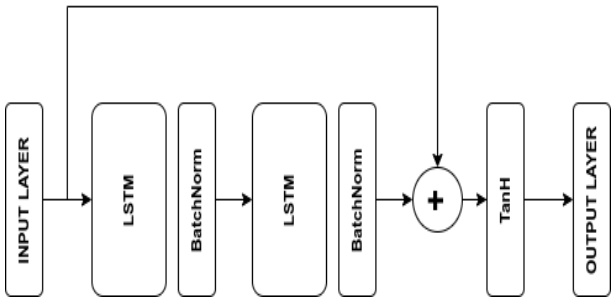

Figure 3: Resnet Like LSTM Net

6 Heterogeneus Layers

Not seeing results by concatenating network we thought to create a network of Hybrid layers of LSTM and CNN

6.1 LSTM + CNN

By inspiration on the last laboratories on Time Series forecasting, we tried to create multiple layer of LSTM + CNN obtaining an improvement in performances from 68% to 69% in respect to the vanilla BiLstm.

6.2 CNN + LSTM

We also tried to do the opposite obtaining slightly better results (70% on submission).

6.3 CNN + DENSE

6.4 ALTRI DI RAFFAELLO

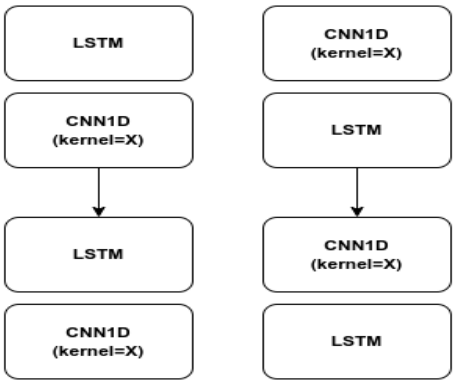

Figure 4: LSTM-CNN and CNN-LSTM model schematics

7 Our best Model

Unexpectedly bla bla bla

8 Conclusion