ANN2DL Homework 2 - Report

Leonardo Breda - MTM - 10695861
Pablo Giaccaglia - CS - 10626428
Alessandro La Conca - CS - 10652084

December 2022

Abstract

In this homework, we were asked to perform a times series classification task. Starting from canonical data exploration, we tried 2 different solutions to tackle the problem: time series-based and vision-based approaches. In our final model, we combine both approaches to achieve the best result. Our code can be found on this notebook.

1 Data Loading and Exploration

The dataset that has been given to us consists of 2429 time series samples, each composed of 36 timestamps of 6 features, these data were saved in a `.npy` file. Each time series sample is associated with a class label. Overall the dataset contains data from 12 classes.

The first thing we did was a canonical data exploration. We first plotted all the samples for each feature: we immediately noticed the presence of a wide range of values. Secondly, we analyzed the distribution of samples among the classes. As shown in Figure 2 the dataset is highly unbalanced: 5 classes have less than 100 samples and class 9 has more than 700 samples.

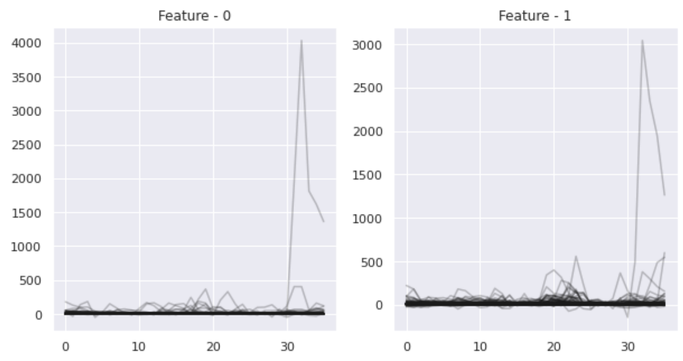

Figure 1: Samples from the first and second features, we enlight the wide range of values

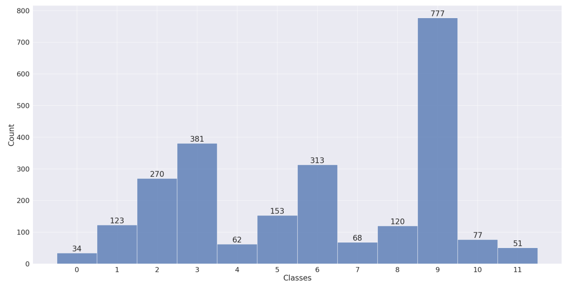

Figure 2: Class distribution

In order to deal with more neural network-friendly data we performed some pre-processing. For our first attempts we employed the SciKit-learn[3]'s RobustScaler in a sample-wise manner: taking into account all the features we scaled each sample to the IQR. For the final ensemble model, we applied 2 different sample-wise pre-processings: data was normalized in the range 0-255 for the vision-based model and both normalized and standardized for the 1D-convolution-based model. To handle class unbalance and to improve the robustness of the network we performed data augmentation. For each training sample, 2 additional samples were generated by randomly applying a transformation among noise addition, constant scaling, magnitude warping, time warping, window slicing, and window warping. We observed that combining augmentations led to worse results than applying just one.

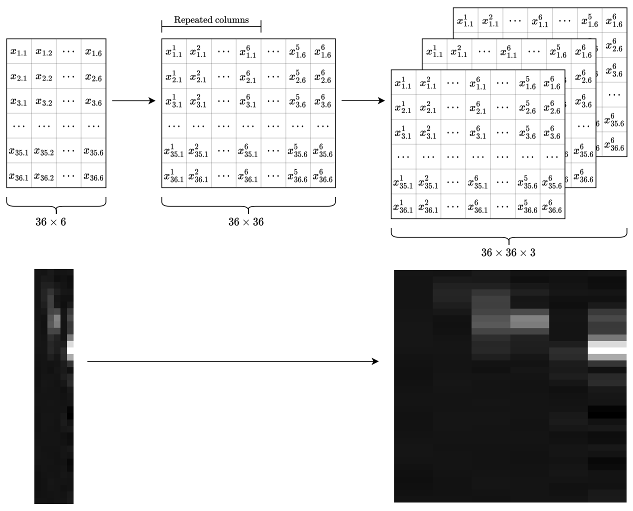

Figure 3: Time series sample seen as an image is first stretched and then repeated 3 times along the channel axis

Then samples were manipulated with two different approaches: on one hand, we decided to use the time series without modifying the structure of the sample, that is of shape 36×6, on the other hand, we tackled the problem in a more vision-like way. To do this we reshaped the time series by repeating each feature 6 times, stacking them side by side. In this way, we obtained a 36×36 image. We then stacked 3 copies of this along the channel axis to get a 3-channel RGB - like image. The process is illustrated in Figure 3.

Finally a stratified train-validation-test split was performed to effectively conduct model training, selection, and test evaluation.

2 First attempts

2.1 Time series-based approaches

Our first trial was implementing a model based on Bidirectional LSTM. We stacked 3 Bi-LSTM layers with 64 units, followed by a dense layer and a soft-max classifier layer. The training was performed employing Focal Loss.

Our second attempt relies on a 1D convolution-based model. We stacked 6 convolutional layers interspersed with max poolings. After the feature extraction, we placed a dropout and a soft-max classifier layer.

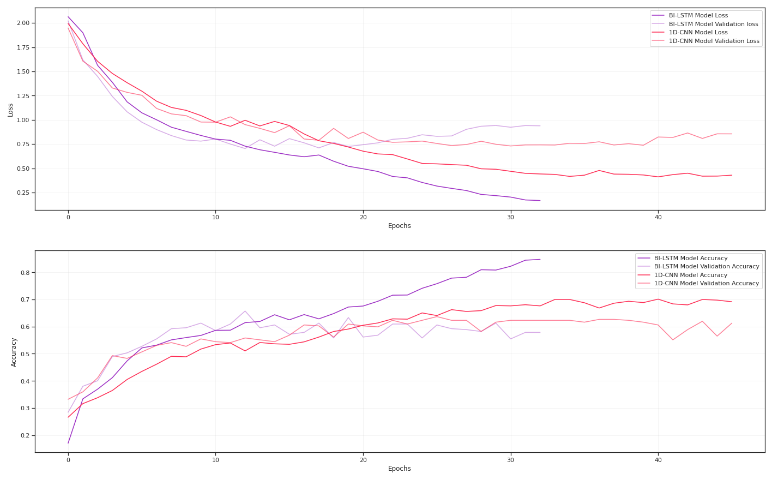

Figure 4: Comparison of Loss and Accuracy between the BI-LSTM model and the 1D-CNN one

In both trainings we used a Cyclical learning rate scheduler with a triangular policy. In Figure 4 we can see the comparison between the losses and the accuracy of the 2 models, we can observe that both models achieved similar performance with the LSTM one overfitting earlier.

2.2 More into Experiments

In this section, we present some sub-optimal solutions we explored during the development of the final model.

Transformer based models We performed transformer experiments both with time series-based and vision-based approaches. Our best times series-based transformer model embeds the times series before using Attention Blocks, following the technique described in the Time2Vec paper [1], which decomposes the time series into sinusoidal components. We achieved $\approx 64.1\%$ test set accuracy by training the model with normalization and data augmentation. Another vision-based approach was based on the paper "Patches are all you need"[4], using 12 ConvMixer blocks. With this architecture we reached $\approx 68\%$ test set accuracy.

GAN augmentation We unsuccessfully used Generative Adversarial Networks for data augmentation. The vision-based GAN could not learn useful features while the time series-based GAN seemed promising but required too many hours to train.

3 Final Model

The final model is composed of an ensemble of a vision-based model and a time series-based CNN model.

The former model's feature extraction part was built testing the ones of Keras Applications. After trying various combinations of backbones, such as EfficientNet and ResNet, and classification networks, we obtained the best performance using ConvNeXtLarge [2]. We loaded the feature extraction part of the model with the weights learned from the ImageNet dataset and experimented with various numbers of layers to freeze during training.

We found the best value for this hyperparameter to be 80 out of 295 backbone layers. This means that we froze the first 80 layers of the model. The assumption is the same as the last homework: that the first layers of the backbone were trained on a rich dataset to perform the extraction of general low-level features, thus the weights are already suitable for our classification task and any further training on our data could lead to worse performance. On the other hand, the remaining part of the backbone has a more specialized role in the feature extraction part, so it is more suited to be a starting point for the search for effective weights to extract the most complex features characterizing our dataset. In order to obtain better performance from the backbone we resized the input image to 224x224 pixels. For the classification section, we used a Dropout layer, a ReLU dense layer with 256 neurons, and a SoftMax dense layer with 12 neurons for the classification task (Figure 8).

The latter model's feature extraction part is composed by three 1D Convolutional layers followed by a Global Average Pooling layer and then a classification section equal to the one for the vision-based model is placed.

The two models have been trained separately and then merged into a single model using an average layer. The time-based model achieved a test accuracy of $\approx 73.7\%$ while the vision-based model achieved a test accuracy of $\approx 77.0\%$. The ensemble, which averages the 2 model's class scores, achieved a test accuracy of $\approx 78.4\%$. The increase in accuracy is most likely due to the two models having different peculiarities and strengths and to the fact that they are seeing the problem from different points of view.

We can notice from the confusion matrices that the time series-based model has higher accuracy in classes 4,5 and 7 (Figure 5) compared with the larger vision-based model (Figure 6) which has higher or similar accuracy on the other classes. That's another probable reason why by averaging the prediction we get an increased accuracy (Figure 7)

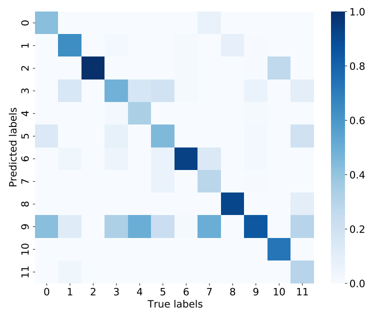

Figure 5: Time-series based model's confusion matrix

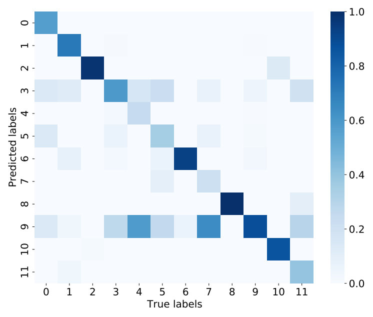

Figure 6: Vision-based model's confusion matrix

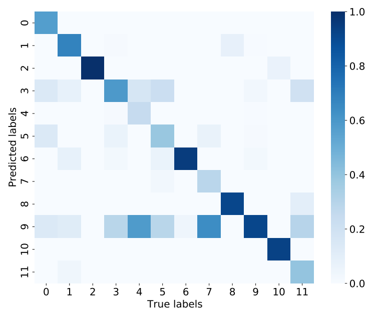

Figure 7: Two models' ensemble confusion matrix

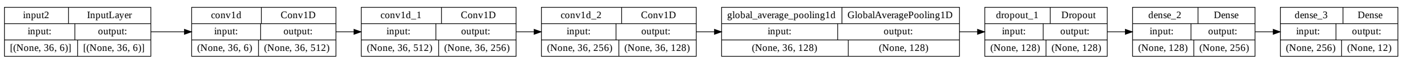
input2 | InputLayer | input: [(None, 36, 6)] | output: [(None, 36, 6)]
conv1d | Conv1D | input: (None, 36, 6) | output: (None, 36, 512)
conv1d_1 | Conv1D | input: (None, 36, 512) | output: (None, 36, 256)
conv1d_2 | Conv1D | input: (None, 36, 256) | output: (None, 36, 128)
global_average_pooling1d | GlobalAveragePooling1D | input: (None, 36, 128) | output: (None, 128)
dropout_1 | Dropout | input: (None, 128) | output: (None, 128)
dense_2 | Dense | input: (None, 128) | output: (None, 256)
dense_3 | Dense | input: (None, 256) | output: (None, 12)

Figure 8: Time series-based model plot

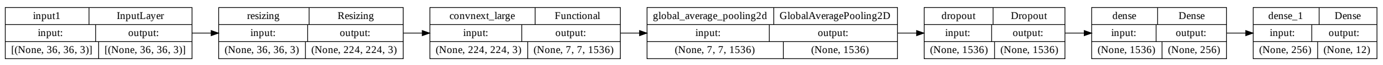
input1 | InputLayer | input: [(None, 36, 36, 3)] | output: [(None, 36, 36, 3)]
resizing | Resizing | input: (None, 36, 36, 3) | output: (None, 224, 224, 3)
convnext_large | Functional | input: (None, 224, 224, 3) | output: (None, 7, 7, 1536)
global_average_pooling2d | GlobalAveragePooling2D | input: (None, 7, 7, 1536) | output: (None, 1536)
dropout | Dropout | input: (None, 1536) | output: (None, 1536)
dense | Dense | input: (None, 1536) | output: (None, 256)
dense_1 | Dense | input: (None, 256) | output: (None, 12)

Figure 9: Vision-based model plot

Another improvement we made was using Test Time Augmentation. We noticed that while TTA decreased accuracy by $\approx 0.2\%$ and recall (previously $\approx 65.7\%$) by $\approx 0.2\%$ it improved precision (previously $\approx 81.6\%$) by $\approx 1.5\%$ and the F1 score (previously $\approx 70.6\%$) by $\approx 0.1\%$. These performance variations led to substantial improvements in model testing on private data.

Finally a last improvement we made to achieve the final accuracy was to train the model on all the available data before testing it against the private dataset.

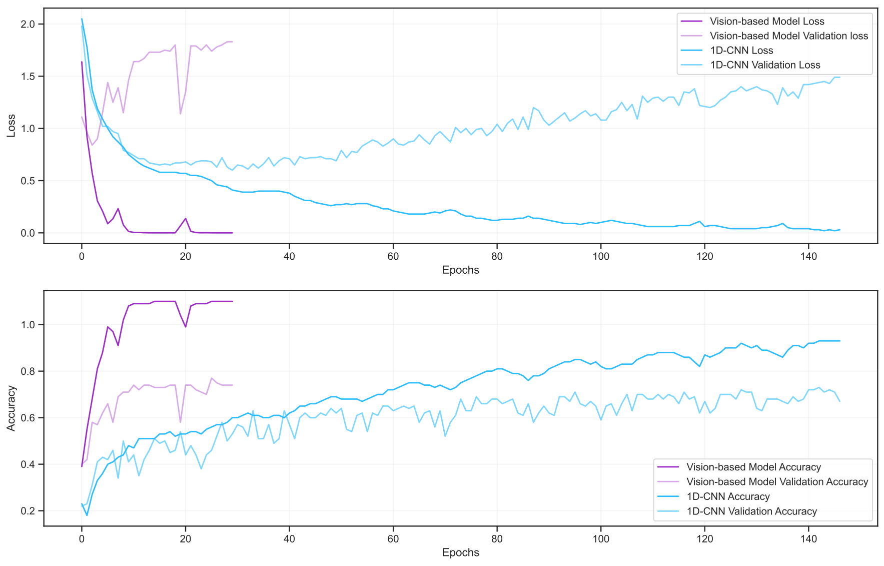

Figure 10: Final Models Loss and Accuracy

References

[1] Seyed Mehran Kazemi et al. "Time2vec: Learning a vector representation of time". In: *arXiv preprint arXiv:1907.05321* (2019).

[2] Zhuang Liu. *A ConvNet for the 2020s*.

[3] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[4] Asher Trockman and J. Zico Kolter. *Patches Are All You Need?* 2022. DOI: `10.48550/ARXIV.2201.09792`. URL: `https://arxiv.org/abs/2201.09792`.