2^{st} Challenge AN2DL

CorrectHorseBatteryStaple
Andrea Bonifacio 10655065
Lorenzo Ravasi 10680017
Sun Yixuan 10819426

In this report we intend to describe the steps we made during the 2^{st} challenge of the AN2DL Course. It will be covered the structure of the dataset, the first steps we made, what we did right, what we did wrong and how we decided to finally tackle the problem.

Dataset

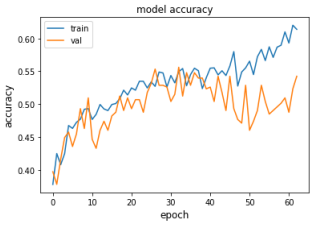
Our dataset consist of a series of time-series, each one made of 36 time steps with 6 features per time step. It was presented to us as two NumPy arrays, one with the collection of time series, called `x.npy` and the other with the labels assigned to each time series, with number from 0 to 12. We do not know what these time series represent, so we decided to follow a standard approach, based on what we learned during the lab sessions and tips we retained from the previous challenge. We opted for a $90/10$ split for training and validation. The data is not homogeneous, we have some classes that have almost 800 samples, and other with less than 80. This created a lot of problems in building our models, because every model we tried wasn't able to classificate the less common class (called 'Breathe'), and we really weren't able to find a solution to manage this kind of imbalance. We thought about data augmentation, but as we were working with time series, we weren't so sure of how to do it. There were some third party libraries, but the documentation wasn't so clear on how to make it work on NumPy arrays, so we quickly stopped pursuing this idea and opted for `Dropout` layers to fight overfitting.

Creating the Neural Network

We started by following the approach seen in the time series classification lab, so we created a Recurren Neural Network (RNN), using the LSTM layer. This time it wasn't so clear what to do, we could have tried a RNN-based approach or a CNN one. We also tried a network composed of `Dense` layers, each one wrapped in a `TimeDistributed` layer. But it probably falls under the definition of RNN. Here we illustrate our two proceedings:

CNN Approach

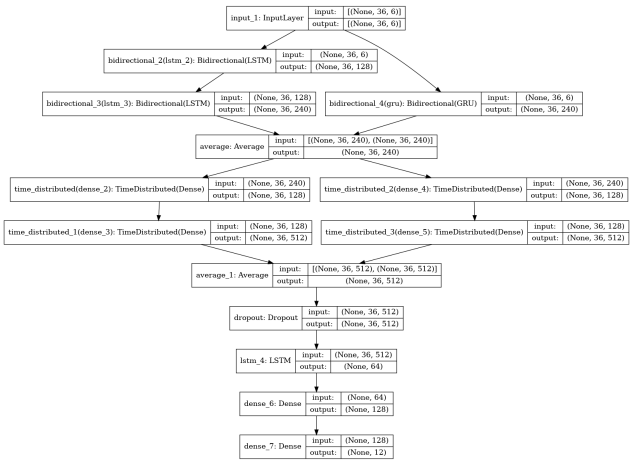
First we tried with a simple CNN made of a stack of three convolutional layers. This basic design clearly didn't work well, our validation accuracy topped out around $\approx 50\%$. We tried rescaling our values with respect to the mean and standard deviation, but we didn't saw any significant improvement. At some point during the challenge, in the Google Drive folder where we found the dataset, in the Logbook document appeared an hint, which suggested to use a model similar to ResNet50. We tried to use the pre-trained ResNet50 from Keras, but we couldn't adapt our dataset to the input required, so we quickly abandoned it, in favor of LSTM models.



RNN Approach

We tried with a simple LSTM, such as the one we used in the lab. In that case our accuracy was around $\approx 60\%$, so we decided to follow this approach, hoping that there was room for improvement. Even without knowing what our dataset represent, we realized that `Bidirectional` models worked a little better, so one of the first change we made was the implementation of various bidirectional layers. We then tried rescaling our dataset with respect to the mean and standard deviation, which also netted us a little gain in accuracy. We didn't reach the same gain when trying to rescale with respect to minimum and maximum valuse of the dataset. At this

point, we started thinkering with every RNN options we found on Keras. We tried wrapping `Dense` layers in a `TimeDistributed` layer, which works by running through a Dense layer every time step of the series.



From the last challenge, we learned how to use the `Average` layer, which, as the name suggests, works by averaging the results of two or more different layers. Since computational power wasn't a problem it was useful to run two models in parallel and then averaging their results in the output, trying to balance advantages and disadvantages typical of each model. We tried to take it a step further and put averaging layers in the middle of the model, the idea that paved the way to our best performing model, as shown in the image. We averaged between two LSTM layers and a GRU layer, and then averaged between two different Dense layers wrapped in a TimeDistributed layer. For the final training we tried to add some dropouts even inside the LSTM layer, which helped a bit avoiding overfitting, but made the model much more slow. Using this model and rescaling our dataset we were able to reach an accuracy of $\approx 69\%$.

What we did right

It's quite difficult this time to say what were our best decisions. Frankly, we hoped to score a little better than under 70%, but this time was more challenging. One of the thing that we think was really helpful was the use of `Bidirectional` layers. Since we had no idea what our data represented, we couldn't be certain about the reason why going bidirectional was working. Maybe reading the sequences both ways was needed to extract useful features. Or maybe not, but that's speculation, since what those numbers represented is obscure to us. The only certain data that we have is that the same model, without bidirectional layers, in average scored worse. The other improvement that we were able to pinpoint to an exact action we took, was the rescaling of values with respect to their mean and standard deviation. We ran some simulation with and without rescaling and consistenly saw an overall gain in accuracy, while scaling with respect to the minimum and maximum values didn't gave us the same results, always a bit worse. Clearly we are talking about small differences, but we needed every little improvement this time to reach a good result.

What we did wrong

Seeing our final results, it's safe to say that we didn't found a really efficient way to tackle this problem. One of the hints that we should have explored more, was the idea of using a ResNet50 model. We tried recreating it thanks to snippets of code found online, but, given our dataset and the short time frame, we couldn't get it to work properly. It kept giving us errors, so we opted for the RNN approach. Another hint that we probably haven't explored enough was the idea of using attention. We had two choices, using `Attention` layers, as given by the Keras framework, or trying a third party library called `keras-attention-mechanism`. The first option seemed quite a bit tricky to implement, since it needed an input given by two different layers, but we didn't manage to make it work properly. We opted for the library, which gave us a simpler implementation, it created an attention layer which needed only one input. It wasn't really clear how it avoided the necessity of two input layer, and probably that's the reason it only worsened our model. We thought that maybe it was built for something else.

Conclusion

It is not that easy to draw some conclusions from this challenge. We started thinking that, like the previous one, we needed to follow what we saw during the course and that would be more than enough. This time it

was more tricky. We didn't found such an obstacle in not knowing what our dataset represented. After all, we could have deduced a lot from trial and error. Our dataset was prone to bidirectional analysis from what we experienced, otherwise we would have seen worse results. This didn't affect our decisional process, just opened up more possibilities. Our final model sure has a lot of complexity, and don't even try to find a balance between computational consumption and accuracy. But we opted for a model that included a lot of the things we saw during the course. Last time we tried a lot of different ideas that worked somewhat in a similar way, but this challenge we tried many possibilities, that often lead to unpleasant results. We didn't talk about all the time we wrote a model from scratch, only to see it reaching an accuracy of 30%. It was definitely a challenge which helped us learn what not to do this time, and how tricky time series can be.