Lazare Plisson-Arcos

CE6146 Introduction to Deep Learning

Assignment 2

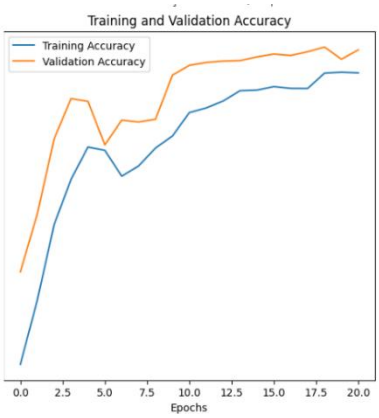
I will be using google Colab for this assignment.

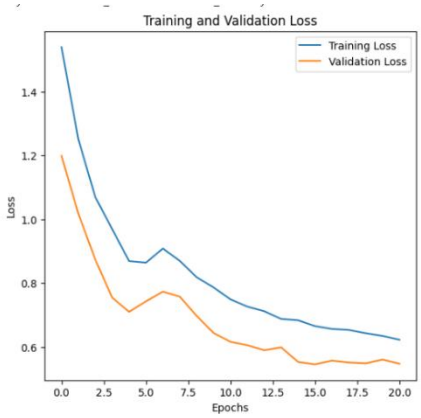
Task 1: Human Activity Classification with LSTM

Explanation of the chosen architecture, hyperparameters, and observations from the learning curves:

- At the start my architecture was 2 LSTM layers with 64 units each. The more layers and units there is, the more complex pattern you can capture but it has drawback. It also increases the model's capacity and may require more data. Because my PC is not very fast, I reduced to 32 units to keep the model complexity and to be reasonably computational efficient.
  Our architecture can capture complex pattern in the data but it's not so much, so it avoids overfitting. LSTM layers were chosen because we need to learn long-term dependencies.
- I also reduced the number of epochs to 30, so the result appears faster.
- I used dropout layers after each LSTM layer to prevent overfitting. I used a high dropout rate of 0,5.
- I'm using ReLU because it is a simple and easy to use activation function, it's very efficient.
- The output size is the same as the number of activities, so it's a multi-class classification.
- A test size of 20% is a common practice and is good for the size of our data.
- I defined the random state to an arbitrary number (11) so I will get the same split of data every time I run the code.
- A batch size of 64 is a commonly used as a starting point.
- I used an early stopping and a ReduceLROnPlateau callbacks to ensure that the model does not overfit and adjusts the learning rate based on the validation loss. Also, the early stopping allows me to not lose time if the results are already stabilized.
- I used Adam because it's very popular, converges faster and has great results (in particular for sparse gradient)
- I used sparse cross entropy because it's efficient for multi-class classification.

The learning curves indicate effective learning from the training data, because the accuracy is steadily rising, and the loss is continuously declining.
There is a small gap between training and validation accuracy and loss that could indicate an overfitting. But the gap is narrow and consistent, so if it exists, it's small and consistent.



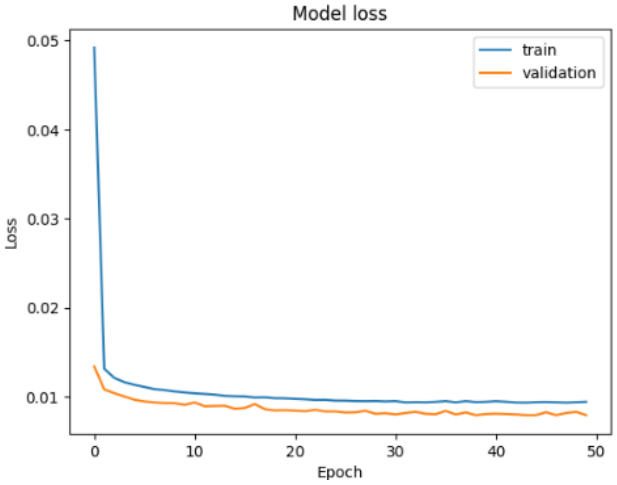


Task 2: Temperature Series Forecasting with GRU

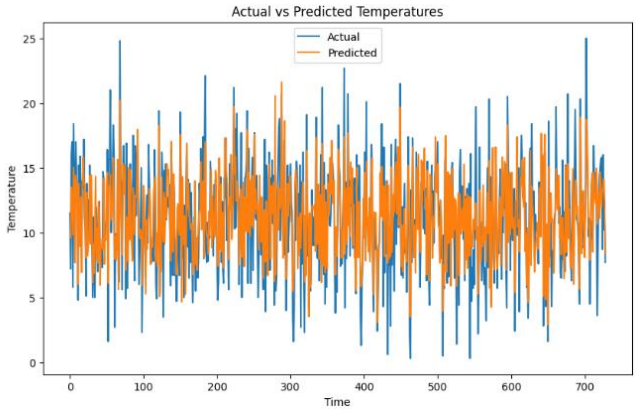
Explanation of the chosen architecture, hyperparameters, and observations from the learning curves:

- This time the computing was fast on my PC. Thus, I kept the GRU neural network with a single layer of 50 units. GRU are used because they capture long-term dependencies well.
- Because the computing was fast, I set the number of epochs of 50. But I still implemented an early stopping callback to avoid overfitting (I used a patience of 10).
- I use a dense layer with a single output unit because it's a regression tasks aimed at predicting a continuous value, the temperature.
- A test size of 20% is a common practice and is good for the size of our data.
- A batch size of 72 is a commonly used as a starting point.
- I defined the random state to an arbitrary number (11) so I will get the same split of data every time I run the code.
- Since the task is regression, I used the RMSE root mean squared error for the loss function during training, which is a common choice for predicting numerical values. Also, it is sensible to outlier, which is useful to detect significant deviation that could represent temperature estimation mistake.
- We continue with the optimizer Adam for the same reason as before (because it's very popular, converges faster and has great results).

The training and validation loss curves are reducing steadily and converging. The training and the validation reach a plateau with only a small gap in between them. These are good indicators that the model is learning and generalizing well without overfitting.



This plot shows the actual temperatures compared to those predicted by the model over time. The 2 are overlapping. It means that the model's predictions closely follow the actual temperature evolution. Thus, the model is performing well.

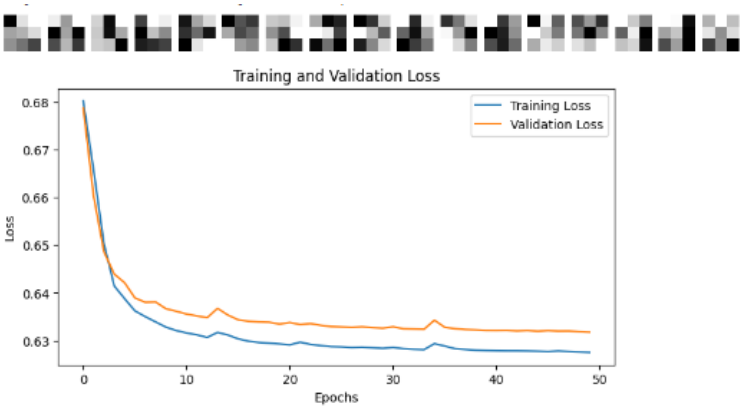


Task 3: Unsupervised Feature Learning with Autoencoders

Explanation of the chosen architecture, hyperparameters, and observations from the learning curves:

- I kept the autoencoder neural network with 64 dimensions. The result was better that way. Autoencoders are effective for unsupervised feature learning because they can learn a compressed representation of the input data, capturing the most salient features.
- I tried multiple epochs, and even thought the computation was pretty slow, I opted for 50 epochs because the result was improving drastically with the number of epochs.
- The autoencoder uses convolutional layers, because they are particularly suited for image data due to their ability to capture spatial hierarchies and features. A test size of 20% is a common practice and is good for the size of our data.
- We use the activation function ReLU in the convolutional layers because it is efficient and also really good at avoiding issue like vanishing gradient problem.
- We use a test of 20%, because it is enough and don't reduce the training dataset size too much.
- A batch size of 32, which is a commonly used starting point.
- I defined the random state to an arbitrary number (11) so I will get the same data split every time I run the code.
- The loss function used is binary cross-entropy, which is suitable for reconstruction error measurement in autoencoders.
- We continue with the optimizer Adam for the same reason as before (because it's very popular, converges faster and has great results). It's also good with those sparse gradients.

Here are the visualization weights of the first encoder layer.



The training and validation loss curves show a sharp decrease at the start, followed by a stable and slower descent. They converge closely and maintain a small and consistent gap throughout the epochs. This suggests an effective learning and generalization, with no apparent signs of overfitting or underfitting.

The top row displays the original faces, while the bottom row shows the reconstructions after processing by the autoencoder. Although some details are lost—a consequence expected given the encoding dimension of 64—a smoothing effect is evident, signifying that finer details have been averaged out. Nevertheless, key facial features such as glasses, eyes, noses, and mouths are well preserved. This suggests that an encoding dimension of 64 is sufficient to retain the essential information for reconstructing these aspects of the faces.