Augmenta - Assignment

Argyrios Christodoulidis - argyrios.christodoulidis@gmail.com

June 29, 2023

1. Introduction

A dataset was given to me with simulated data from an experiment. I separated my work into 3 sections. The data analysis part where I perform summarization of the dataset, visualization, and evaluation. The second, which is the main part, where I perform the data preparation, the model building, training and evaluation. Finally, I include the model loading and inference code for testing the model with user interaction.

2. Dataset summary

The data consists of 3 features, the average pixel value, the ambient light sensor measurements, and the camera exposure time. The total measurements span 25000 points, and we can assume that the whole dataset is part of a time-series sequence. Since the last column units are in milliseconds, multiple and variable number of rows across the whole dataset should be grouped together to cover a single second.

From the statistics (output_stats.csv), the standard deviation for the first two columns is very high probably demonstrating that the measurements constantly change. The camera exposure time is generally low, and it does not vary much. Probably, the acquisition system is in such a way designed to minimize the camera exposure tie when the incoming photons from the environment reach a certain level. This is done to protect the sensor.

If we plot the measurements (output.png, output2.png), we can notice that the ambient sensor values constantly increase while the exposure time decreases. The signals also show that there are cycles. Generally, with each cycle the maximum exposure that is required to achieve saturation drops. This probably shows that the whole process is non stationary, it is not very efficient and the measurements when the pixel values are at maximum are not reliable.

3. ML model

The aim of this exercise was to create an end-to-end ML model to predict the exposure time given the sensor and the pixel values. We have relatively high number of data points, and generally the process of data generation for this experimental setting is cheap, so I choose data-driven methods that rely on neural networks to model sequential data.

For the pre-processing and the model training steps, I assumed a sliding window of a fixed size to sample the data. Then I performed a standard normalization technique (z-scaling) per column, and split the data to training, validation, and test sets. I tried 2 main architectures, a deep recurrent neural network and a long-short term memory model. For the implementation I used Keras library which can acts a front-end for tensorflow. In Keras the model building is abstracted at a high degree so we only have to set up the number of units, the fully connected or dense layer and its output, 1 single value that represents the exposure time in our case, the number of sequential layers, the type of loss function, the optimizer, a possible dropout for extra regularization, and finally the number of training epochs. As a first exploratory step I tried different settings for the parameters. Early stopping ensured regularization and termination when there was not significant model improving during training. For the best architecture, or the one that looked more promising, I considered the dropout and the number of units as hyperparameters, and I used Optuna toolkit to find the best values for the hyperparamters. Having the values I expanded the training to more epochs. To choose which model is the best, I monitored the training and valiation losses in plots. I also considered the training time, and if we can minimize the loss function. The final model was chosen as a single layer LSTM with dropout and specific number of units (LSTM_85_units_22_dropout_500_epochs). Overall, from the training plots we can observe a significant amount of underfitting (val loss under the train loss).

You can find the details and the implementation on the Google colab notebook (exposure_data_notebook.ipynb). I assumed that I have access to a cloud computing environment with a GPU to perform my computations.

4. Results, evaluation, and model deployment

For the performance I used standard metrics for sequence model evaluation, like the MAE, the MSE, the RMSE, and the R-squared. Quantitatively, the performance of the model is very high. The model was tested on the held out dataset from the original source of the data. The final step was to save/export the model and the scaler object to be able to reuse them offline from another location. At model_loading.py I load both the scaler object and the model, and through command line interface I allow the user to make predictions based on the model. Even though the model showed very nice performance in the colab environments, I performed some series of experiments and the prediction did not seem very good.