Deep Learning Project 2

Task 1: CNNs

Dataset:

The used dataset is the EuroSAT dataset from Huggingface, it contains colored pictures (3 channels) of landscapes from the Sentinel-2 satellite with a resolution of 64x64. The dataset has the following 10 classes: Annual Crop', 'Forest', 'Herbaceous Vegetation', 'Highway', 'Industrial', 'Pasture', 'Permanent Crop', 'Residential', 'River', 'Sea/Lake'. The dataset is easily accessible via the Huggingface API and can easily be transformed into a PyTorch dataset. The different models in this exercise will be trained to predict the class of an unseen picture.

For computational reasons not the full dataset is used, but just the first 4000 instances. The data is then prepared by ensuring a predefined picture size of 64 and split into a train, test, and validation dataset. The training dataset contains 80% of the data and the test and validation datasets each 10%. The models will be trained with a batch size of 32. In the following picture are some instances of the dataset displayed:

Filtering single pictures of the dataset:

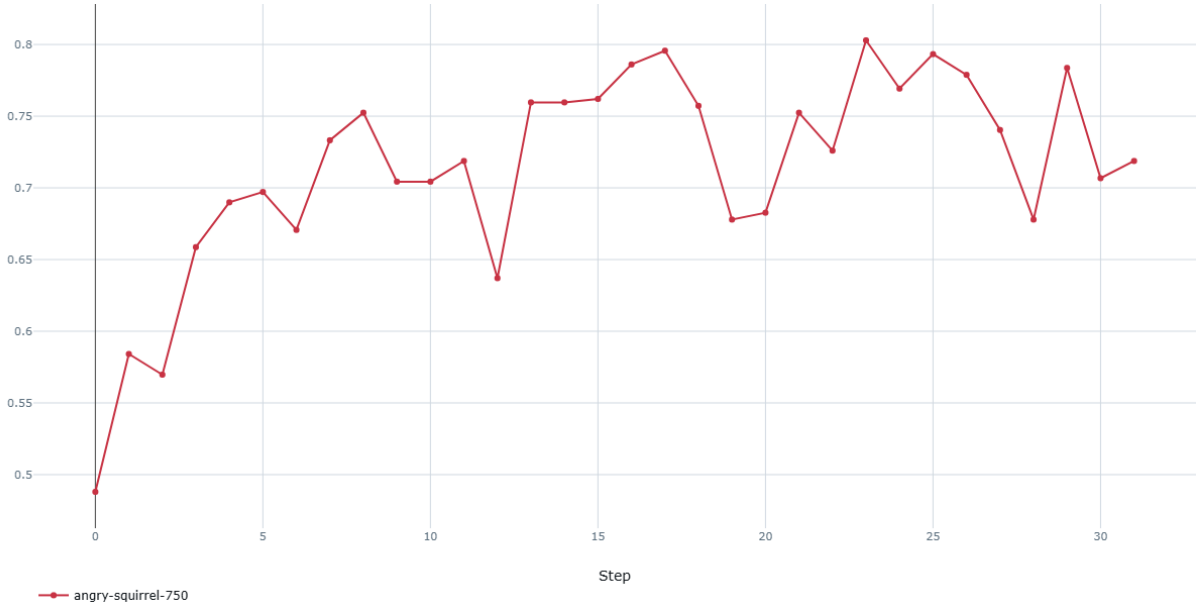
For filtering are three different kernels applied, the gaussian, an edge sharpening, and an edge detection kernel. All of them produce a one-channel result from a 3-channel input. On the output pictures, the different effects of the kernels are visible, as the Gaussian filter reduces the structures in the pictures, while both others emphasize them.

Results of the simple CNN:

The simple CNN is designed with one skip connection, incorporated inside of a residual block, inspired by the resnet. The residual block itself has two convolutional layers where the input

is passed through and at the end the inputs to the outputs of the two layers for the forward pass. In case the number of channels is changed during both convolutions, an additional 1x1 convolution will be applied to ensure the inputs and outputs have the same dimension for adding them. Between all layers, the ReLU activation and batch normalization are applied. The model itself then uses the first convolutional layer and then passes the outputs into the residual block. After that two more convolutional layers follow, which reduce the spatial dimensions to 1x1 and 128 channels. Between all layers, batch normalization is applied and the tanh activation function is used.

The model is trained using the Adams algorithm with a 0.0001 learning rate, a batch size of 32 and the crossentropy lossfunction. The number of epochs was 32 in the first try and resulted in a validation accuracy of 71,8%:

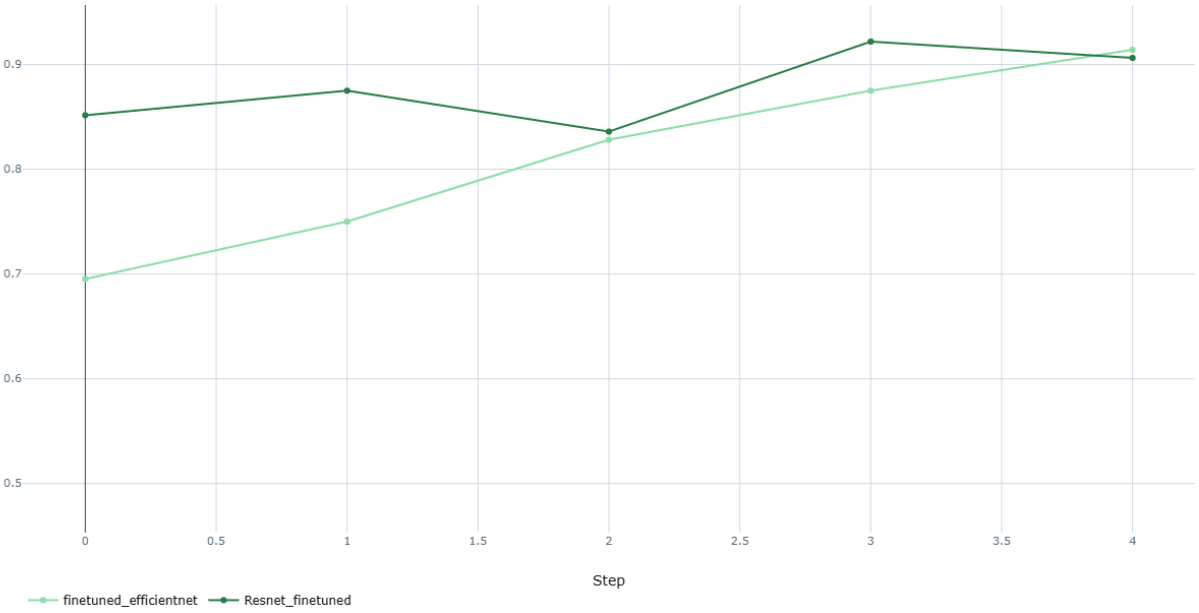


The oscillation in the trajectory at the end indicates that the model starts to overfit. This could mean, that the model either is too complex or there is not enough training data, to train this model from scratch.

Finetuning models:

The chosen models for finetuning are the resnet50 by Microsoft, with 50 is the number of residual blocks, and the efficientnet-b0 by Google. Both models are easily accessible via the huggingface API. As both models are trained for the ILSVRC their input size is 224x224 and their output size is 1000, which is misaligned with the 10 classes we have. That's why the models are loaded with random weights for their last layer and only 10 outputs and the images are upscaled to the size of 224x224. The resnet-50 then has ca. 23,5 Mio parameters and the efficientnet-b0 has ca. 4 Mio parameters.

Both models are finetuned also, with the Adams algorithm with a learning rate of 0.0001, a batch size of 32 and the cross entropy loss function. The number of epochs is reduced to 5 as the high parameter count leads to long training times. The following graph shows the trajectory of the validation accuracy for both models.



The Resnet has in the end an accuracy score of 90,6% and the Efficientnet of 91,4%.

Both networks show a way stronger ability to classify the different images, even with a way reduced number of epochs. Another aspect is that the models already start at a higher validation accuracy than the self-designed model, showing that the first epoch already yields a good score, showing that the models already learned some image classification. Even though these models are as complex as the self-designed models, they yield way better scores on even a smaller amount of data, while not seeming to overfit the data. That shows the power of transfer learning when we have a smaller dataset.

YOLO and semantic segmentation

The YOLO models is easily accessible via the PyTorch modelhub. In this Project I used the yolov5 model for object detection on 3 different images. As the model is already trained no further finetuning is necessary and the pictures can be plugged into the model directly, even without further preprocessing, as the model takes images of different shapes. The results of the model are very good, various bounding boxes could be generated around objects in this picture, all of them assigned with a confidence score.

You can see, that the mouth which was detected as cell phone, so a wrong bounding box, has a low confidence score, showing that non-max-suppression could help here to increase the accuracy of object detection.

The Segmentation model also is easily accessible via the PyTorch hub. Contrary to the YOLO model, here some preprocessing is necessary, as the model needs an input size of at least 224x224. The segmentation model can detect 21 classes of which one is a background class, so the model has 21 output channels, of which each one represents one class, and we check where the maximum is at each pixel, to determine the predicted class. This results in a colored map like this one:

Here the blue part in the picture is a detected human and the green parts around him are dogs. The black rest is the background class, where no other class was predicted.

Task 2: RNNs

Dataset

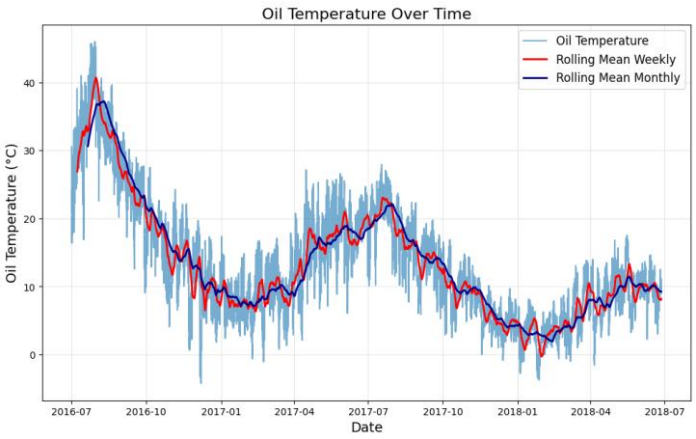
The dataset is the Electricity Transformer Dataset, which contains data from different electricity transformers in China over a two-year period of approximately two years. The data is logged every 15 minutes, resulting in 4*24 data points per day. The used data set is the small version, which means it only contains the data of 2 stations. On this data set time series forecasting is performed for the target variable Oil Temperature (OT) with additional input features:

Field	**date**	**HUFL**	**HULL**	**MUFL**	**MULL**	**LUFL**	**LULL**	**OT**
Description	The recorded **date**	**H**igh **U**seFul **L**oad	**H**igh **U**seLess **L**oad	**M**iddle **U**seFul **L**oad	**M**iddle **U**seLess **L**oad	**L**ow **U**seFul **L**oad	**L**ow **U**seLess **L**oad	**O**il **T**emperature (target

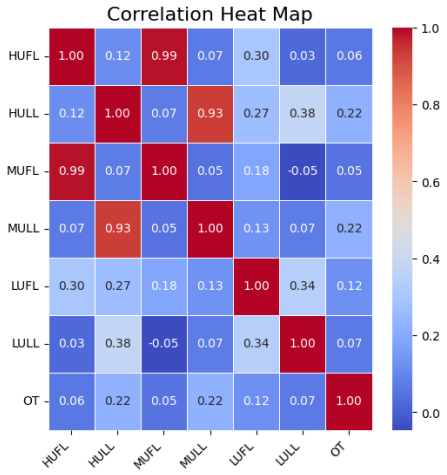
The summary statistics for the different features are:

	HUFL	HULL	MUFL	MULL	LUFL	LULL	OT
count	69680.000000	69680.000000	69680.000000	69680.000000	69680.000000	69680.000000	69680.000000
mean	7.413993	2.261418	4.322226	0.896509	3.082888	0.857907	13.320642
std	7.082928	2.041293	6.829189	1.807239	1.174063	0.600440	8.564817
min	-23.242001	-5.693000	-26.367001	-6.041000	-1.188000	-1.371000	-4.221000
25%	5.827000	0.804000	3.305000	-0.284000	2.315000	0.670000	6.964000
50%	8.841000	2.210000	6.005000	0.959000	2.833000	0.975000	11.396000
75%	11.788000	3.684000	8.635000	2.203000	3.655000	1.218000	18.079000
max	24.180000	10.315000	18.087000	7.853000	8.498000	3.046000	46.007000

The target variable has a very high volatility throughout all observations, already indicated by the standard deviation, which is the highest among all features. The following picture shows the trajectory of the target variable along with weekly and monthly rolling means.

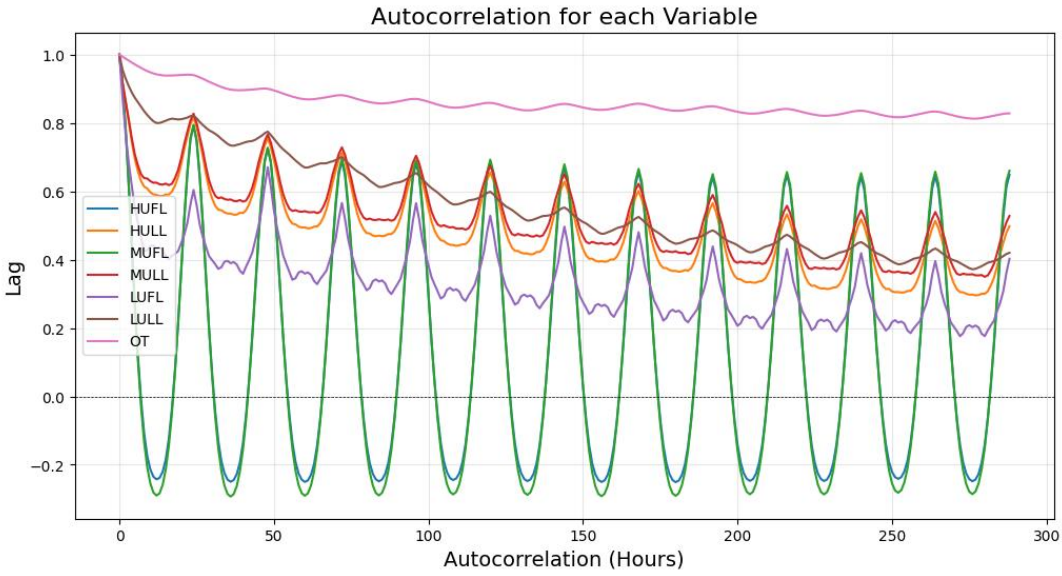


The plot confirms the high volatility of the oil temperature and reveals a yearly seasonality, with a peak in July every year. In addition, the figure reveals an overall downward trend in the oil temperatures. To get a first idea, how the oil temperature is dependent on the other variables I also plotted a correlation matrix for all the variables and the oil temperature:



The figure reveals that some of the independent variables are intercorrelated really strong, the HUFL and the MUFL even have a correlation of 0.99, which is so high that these two variables could be linear dependent resulting in inefficient usage of capacity for neural networks. Another detail that comes to my eye is that there is almost no negative correlation between any of these variables, just one pair shows a correlation slightly under 0. To investigate the correlation between the target variable and the others, the last row or column should be analyzed. There it shows that HULL and MULL have the highest correlation with the Oil Temperature with 0.22. This is still a quite low correlation, however the correlation coefficient measures just linear dependencies, so there still could be another form of dependency between these variables.

As we are dealing with time series data analyzing the time dependency between the variables is crucial. Therefore, the autocorrelation for each variable is computed for hourly lags up to 12*24 = 288 hours, so 12 days.



The plot shows how high the timely dependence of all the variables is. It shows that the Oil-Temperature is highly correlated to its lagged observations, even up to very long time ago. Another very significant pattern is the autocorrelation of the independent variables, which exhibit notable daily seasonality, with a peak after every 24 hours. MUFL and HUFL even show negative autocorrelation in the valleys of their autocorrelation plot.

With the knowledge of the exploratory data analysis, the models will be trained by chunks of the independent variables, with a sequence length of 24, meaning that the last 24 observations of the independent variables are used to predict the oil temperature of the next day. For training the dataset is split into a training and test data set with 80% of the dataset as train dataset and 20% as test. In addition all models are trained using a batch size of 32.

Selfdesigned RNN models

First, we designed a baseline model, that just uses the oil temperature of the current day to predict the oil temperature of the next day. All following models will be evaluated against this model.

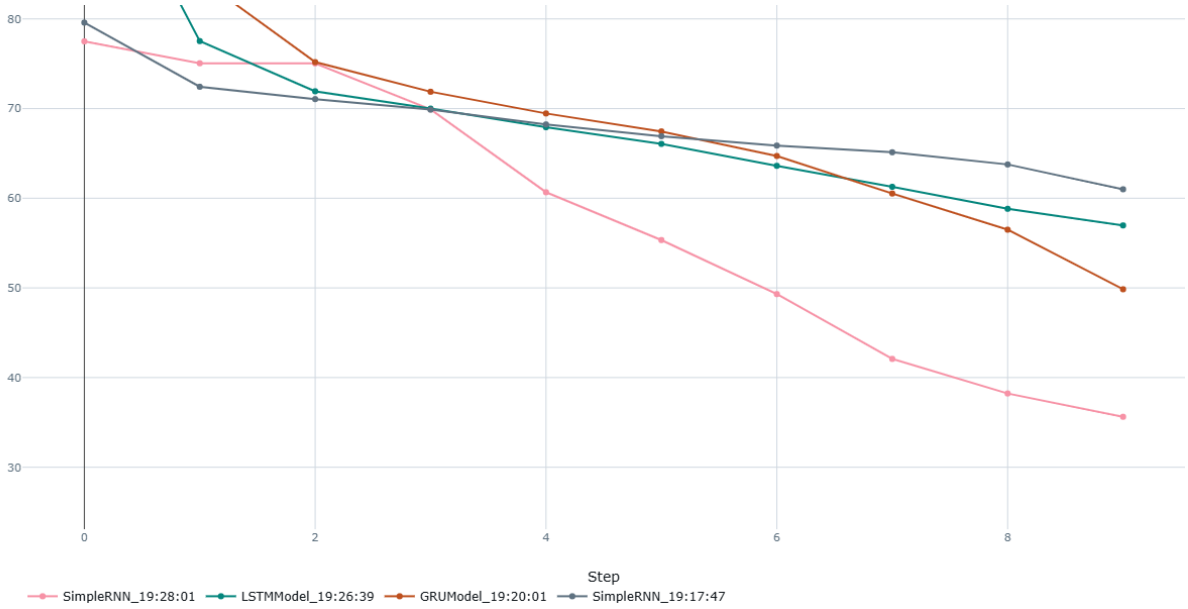
The first self-designed model is a simple RNN with one recurrent layer. It takes the whole chunk of input data, so it takes 6 features, each with a sequence length of 24. The size of the hidden layer is set to 64, meaning that the hidden state vector at each timepoint has a size of 64.

The next self-designed model is a Gated Recurrent Unit (GRU), which is implemented manually, without the corresponding PyTorch function. The unit is defined by the reset and update gate, as well as the candidate hidden state, which all share the same hidden size, which is set here to 32. This model takes the same inputs as the simple RNN.

An extended version of the GRU is the Long-Short-Term-Memory network (LSTM), which uses three gates to control the information flow between states and has two different kinds of hidden states. This model is implemented using the corresponding PyTorch function with a hidden size of 32 and one layer, meaning that there is just one LSTM unit stacked.

The last model uses unlike the other models more than on unit. It is a stacked RNN unsing more than one recurrent unit, stacked over each other. To implement this, the same model class like in the simple RNN can be used, just by specifying the the num_layers parameter to bigger than one, in our case to three, meaning that there are three Recurrent units used. The rest of the models parameters are set equal to the simple RNN.

All of the models are trained and validated using the MSE loss function, which is a common loss function for regression tasks, as it is perfect for continuous data. For training time reduction, the models are trained with 10 epochs. The models are trained with the Adams algorithm, with a learning rate of 0.0001. The results of the model training are depicted in the following figure, by showing the trajectory of the validation MSE:



The pink trajectory represents the stacked RNN, while the gray one represents the RNN with just one layer. The stacked RNN yields the best training results, with the steepest decrease of all models, over the epochs compared to the other models, where the steepness of the descent decreased after the first 3 epochs. In addition, the plot indicates that complex models yield better results, by showing that the GRU and LSTM have better scores than just a plain RNN model. Also, the still descending validation MSE suggests that more training epochs could yield better results for all models, but especially the more complex ones.

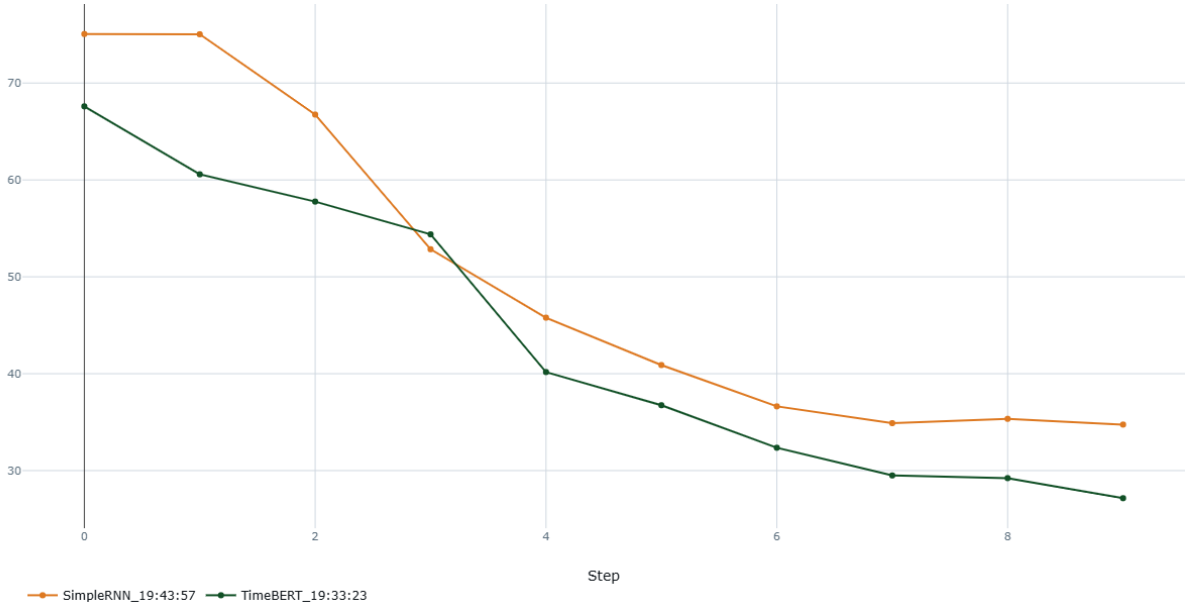
Finetuning and Hyperparameter optimization

In the next steps it is tried to further enhance the prediction performance by using a pretrained model and by doing a hyperparameter optimization on the RNN model.

The pretrained model used is the BERT model, which will be rewritten to analyze time series data. BERT is a transformer-based model, designed for language tasks, which can be applied to this context as it is very strong at recognizing sequential patterns. To apply the model, we use the BertModel class from Huggingface to access the pretrained model and provide it with a suitable config. We define the BertModel using a configuration where the hidden size, representing the embedding size the model will process, is set to hidden_dim. The number of attention heads is set to 4, and the number of layers in the model is defined as 2. The intermediate layer size in the feed-forward network is configured as hidden_dim * 4, ensuring adequate capacity for capturing complex patterns. The maximum context length, which corresponds to the input sequence length, is set to sequence_length. Additionally, the layers are configured with a 10% dropout rate for both the hidden states and attention probabilities to mitigate overfitting during training. To make the data suitable for the Bert and to generate the embeddings a custom encoding and decoding layer are added to the bert model, which are both just linear layers transforming the inputs into dimension 128 and decoding the outputs into dimension one for regression.

For the hyperparameter optimization the optuna package is employed. The best hyperparameters are searched for the stacked RNN, including the best number of layers, the best hidden size and the best dropout rate. The hidden_size is search in a range between 23 and 128, the number of layers is searched in a range between 1 and 8 and the dropout rate is searched between 0.0 and 0.5. The hyperparameters are optimized with 16 trials by the optuna package, which uses Tree-structured Parzen Estimator to find the best parameters, which is a Bayesian optimization algorithm that decreases to overall number of trials needed for finding the best hyperparameters. After the optimization the following hyperparameters came out: {'hidden_size': 105, 'num_layers': 3, 'dropout': 0.2970317684831742}.

Both models have been trained with the same training parameters as the other models, which results in the following figure:



The overall validation performance is better than for the other two models, indicated by lower ending validation MSE values. In addition, both models show a steep descent in the validation MSE, suggesting that more epochs could further enhance the results.

Overall model comparison

So far, the models have been evaluated with the same metric as they got trained. However, there are more and even better metrics to evaluate Regression models. Therefore all of the priorly discussed models are evaluated not only with the MSE, but also with the MAE, SMAP and R^2-Score.

The MAE Measures the average absolute difference between the true values and the predicted values. It provides an intuitive sense of how far predictions are from the actual values. It is used, because it is interpretable, and penalizes errors proportionally to their magnitude.

$$MAE = \frac{1}{n}\sum_{1=1}^{n}|y_i - \hat{y}i|$$

The SMAPE normalizes the error as a percentage, making it scale-independent. It symmetrically accounts for over- and under-predictions by dividing by the average of the actual and predicted values. Therefore, its particularly useful for datasets where absolute values vary significantly or are on different scales.

$$SMAPE = \frac{100\%}{n} \sum_{i=1}^{n} \frac{|y_i - \hat{y}_i|}{(|y_i| + |\hat{y}_i|)/2}$$

The R^2- Score shows how much of the variance in the data can be explained by the model. The numerator of the score measures the model's prediction error, and the denominator measures the total variance in the data. R^2 ranges: R2=1: Perfect fit. R2=0: Predictions are as good as predicting the mean. R^2 < 0 Predictions are worse than predicting the mean. The R^2 Score is easy to communicate and independent of the data magnitudes.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

The result table is the following:

Metric	Baseline	Simple RNN	GRU	LSTM	Stacked RNN	Time Bert	Optim-RNN
MSE	144.74	58.51	47.83	54.39	33.27	25.46	32.04
MSE	9.29	5.78	5.06	5.60	4.09	3.70	4.13
SMAPE	71.75	48.08	42.79	46.46	36.28	34.08	36.48
R^2	-0.986	0.19	0.34	0.25	0.54	0.65	0.56

The results show that the baseline model is far outperformed by every model, showing that the other models have a bit of predictive power. However, the scores of the baseline model are very bad, indicating that the model performs worse than just predicting the mean value of the data. The table also shows that the fine-tuned Bert Model outperforms the other models by far, showing the power of transfer learning and maybe even of transformers. The results also show that the hyperparameter optimized RNN could outperform the stacked RNN but by far not to the extent that the Bert does it.

Conclusions and Discussion

Using more complex models could enhance the predictive performance by far. Also using pretrained models helped, as the datasets size is not large enough to train very large models from scratch. However, the number of epochs and the sequence length was decreased, to reduce computation time as the training is done just on a common Laptop CPU. Enhancing the sequence-length, increasing the number of epochs and incorporating lagged predictions of the target variable would enhance the predictive performance of all models and would allow more complex models.