**Data Augmentation in Neural Networks**

In application for images and time series

Jiyon Park – MatrNr. 11182049  
Felix Mayer – MatrNr. 11137598

|  |  |  |  |
| --- | --- | --- | --- |
|  | Auditors | Prof. Dr. Daniel Gaida |  |
|  |  | Prof. Dr. Wolfgang Konen |  |

Gummersbach, July 4, 2023



Abstract

This article states the application of data augmentation techniques for images and time series data in neural network models. For data augmentation in images, methods such as flip, rotation are performed based on the library of TensorFlow. The utility of data augmentation was visualized by the demonstration of graphs, noting the improvements of training have been achieved by the augmentation of training data. In the case of time series data, techniques such as jittering, rotation/flipping, scaling, magnitude warping, permutation, and slicing were investigated to augment the limited real event datasets. By combining these augmentation methods and training a STGCN with LSTM layers, significant enhancements in forecasting accuracy were observed. Data augmentation emerged as a valuable approach for improving model performance, preventing overfitting, and enabling better generalization to unseen data in both image and time series domains.

Table of Content

[List of Figures 4](#_Toc139329611)

[1 Introduction 5](#_Toc139329612)

[2 Data Augmentation for Images 6](#_Toc139329613)

[2.1 Background 6](#_Toc139329614)

[2.2 Dataset 6](#_Toc139329615)

[2.3 Data augmentation 6](#_Toc139329616)

[2.3.1 Data augmentation method 6](#_Toc139329617)

[2.3.2 Visualizing the utility of data augmentation. 8](#_Toc139329618)

[3 Data Augmentation for Time Series 10](#_Toc139329619)

[3.1 Background 10](#_Toc139329620)

[3.2 Dataset 10](#_Toc139329621)

[3.3 Data Selection and Preparation 10](#_Toc139329622)

[3.4 Model 11](#_Toc139329623)

[3.5 Data Augmentation 11](#_Toc139329624)

[3.5.1 Jittering 12](#_Toc139329625)

[3.5.2 Rotation / Flipping 12](#_Toc139329626)

[3.5.3 Scaling 13](#_Toc139329627)

[3.5.4 Magnitude Warping 13](#_Toc139329628)

[3.5.5 Permutation 14](#_Toc139329629)

[3.5.6 Slicing 14](#_Toc139329630)

[3.6 Deployment in a real model 15](#_Toc139329631)

[3.6.1 Integration into data pre-processing 15](#_Toc139329632)

[3.6.2 Without Data Augmentation 16](#_Toc139329633)

[3.6.3 Scaling 16](#_Toc139329634)

[3.6.4 Jittering 17](#_Toc139329635)

[3.6.5 Vertical Flipping 17](#_Toc139329636)

[3.6.6 Combination of the Methods 18](#_Toc139329637)

[4 Conclusion 19](#_Toc139329638)

List of Figures

[Figure 1: Caputer of Fashion-MNIST and its classes [4] [5] 6](#_Toc139329639)

[Figure 2: Example of data augmentation using Keras preproducing layer[2] 7](#_Toc139329640)

[Figure 3: Augmented images using tf.image 7](#_Toc139329641)

[Figure 4: tf.image random image operations and example[2] 8](#_Toc139329642)

[Figure 5: Graph of training and validation accuracy and loss at training data 80% 8](#_Toc139329643)

[Figure 6: Code used for data augmentation[2] 9](#_Toc139329644)

[Figure 7: Compare of reduced and Augmented data 9](#_Toc139329645)

[Figure 8: 50epochs and 100epochs of the augmentation of the 1% training data 9](#_Toc139329646)

[Figure 9: Speed measurement points [7] 10](#_Toc139329647)

[Figure 10: Structure of the neural network 11](#_Toc139329648)

[Figure 11: Internal structure of the STGCN [8] 11](#_Toc139329649)

[Figure 12: Example time series (original) [5] 12](#_Toc139329650)

[Figure 13: Example time series with jittering [9] 12](#_Toc139329651)

[Figure 14: Example time series with rotation / flipping [9] 13](#_Toc139329652)

[Figure 15: Example time series with scaling [9] 13](#_Toc139329653)

[Figure 16: Example time series with magnitude warping [9] 14](#_Toc139329654)

[Figure 17: Example time series with permutation [9] 14](#_Toc139329655)

[Figure 18: Example time series with slicing [9] 15](#_Toc139329656)

[Figure 19: Flow of data pre-processing 15](#_Toc139329657)

[Figure 20: Without data augmentation (training results left; forecasting right) 16](#_Toc139329658)

[Figure 21: Real results for training with Scaling (training results left; forecasting right) 17](#_Toc139329659)

[Figure 22: Real results for training with jittering (training results left; forecasting right) 17](#_Toc139329660)

[Figure 23: Real results for training with flipping (training results left; forecasting right) 18](#_Toc139329661)

[Figure 24: Real results for training with combined DA methods (training results left; forecasting right) 18](#_Toc139329662)

# Introduction

Data augmentation is a powerful technique in machine learning that aims to address the challenges posed by limited datasets. This involves generating additional training samples by applying various random based transformations to existing data, while preserving the inherent features and labels. By increasing the diversity and quantity of training data, data augmentation can improve model performance by preventing overfitting and enhancing generalization to unseen data. Along with the fundamental studies of data augmentation applied in image processing tasks, its application to other domains, such as time series data, is gaining increasing attention.

This article explores the utilization of data augmentation techniques for both image and time series data in neural network models. The purpose is to demonstrate the effectiveness of data augmentation by visualizing the improvement of the performance of neural networks though it is trained with limited datasets. Specifically, the article focuses on two distinct datasets: Fashion-MNIST, a popular dataset used for image classification, and PeMSD7, a dataset containing speed measurements in a time series format.

# Data Augmentation for Images

## Background

In order to achieve the quality of the neural network (NN) a sufficient amount of data is needed. It requires a large amount of tagged data to train the NN models to avoid overfitting. [1] Data augmentation, which is a technique to increase the diversity the training set by applying random (but realistic) transformations, can be inferred as one kind of technology to overcome overfitting. [2][3] The idea was to demonstrate the utility of data augmentation.

## Dataset

One main purpose of the Fashion-MNIST dataset is the classification of clothing. [4] Fashion-MNIST is a dataset of Zalando's article images consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. [5] The labels are an array of integers, ranging from 0 to 9. These correspond to the class of clothing the image represents. [4]



Figure 1: Caputer of Fashion-MNIST and its classes [4] [5]

## Data augmentation

### Data augmentation method

For data augmentation for images, there are mainly two options that can be suggested. One is using the Keras preproducing layers, and the other is using the tf.images.

Keras preproducing layers such as tf.keras.layers.RandomFlip and tf.keras.layers.RandomRotation are an example. Based on this, creating some pre-processing layers and applying them repeatedly to the same image would look like this [2].

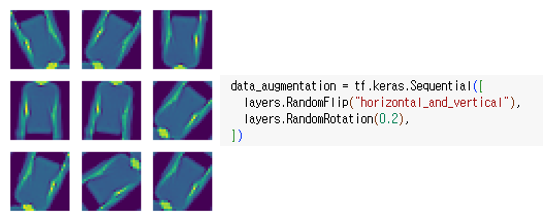


Figure 2: Example of data augmentation using Keras preproducing layer[2]

Also, Keras layers such as tf.keras.layers.RandomContrast, tf.keras.layers.RandomCrop, tf.keras.layers.RandomZoom and a variety preprocessing layers can be used for data augmentation [2].

Using tf.image gives the opportuinity of a finer control and allows to write individual data augmentation pipelines or layers using tf.data and tf.images. The representative modification that can be made are Flip, Grayscale, Saturate, Change Brightness, Crop, Rotate [2].

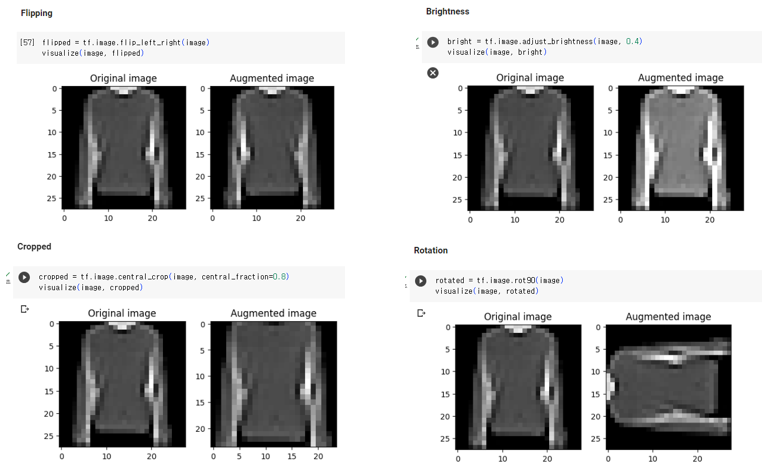


Figure 3: Augmented images using tf.image

For the Fashion-MNIST dataset, the grayscale and saturation are not appropriate because the dataset is consisted with grayscale images.

Also, applying random transformations using the tf.image API is possible which can further help generalize and expand the dataset. The tf. Image provides eight random image operations [2].



Figure 4: tf.image random image operations and example[2]

### Visualizing the utility of data augmentation.

The reduction of the training data is one way to visualise the need for data augmentation. Below graphs shows the training and validation accuracy and the training and validation loss when the training data is 80% and the validation data is 20% of the dataset.

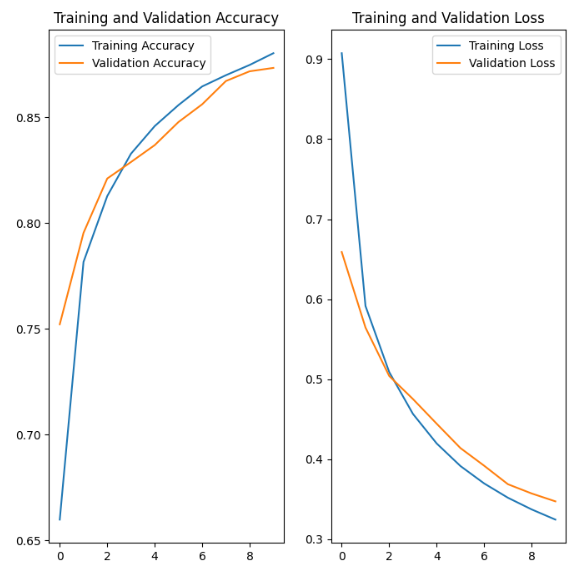


Figure 5: Graph of training and validation accuracy and loss at training data 80%

Reducing the amount of data used and using it to form data augmentation would allow the utility of data augmentation to be visualized. Data augmentation is provided by layers.RandomFlip and layers.RandomRotation.



Figure 6: Code used for data augmentation[2]

Shown by the below graph, where the training data is reduced to 40%, 20%, 10%, 1%, and comparing it to the graph where the reduced data was used for augmentation, we can see that the less data there is, the more augmentation is needed.

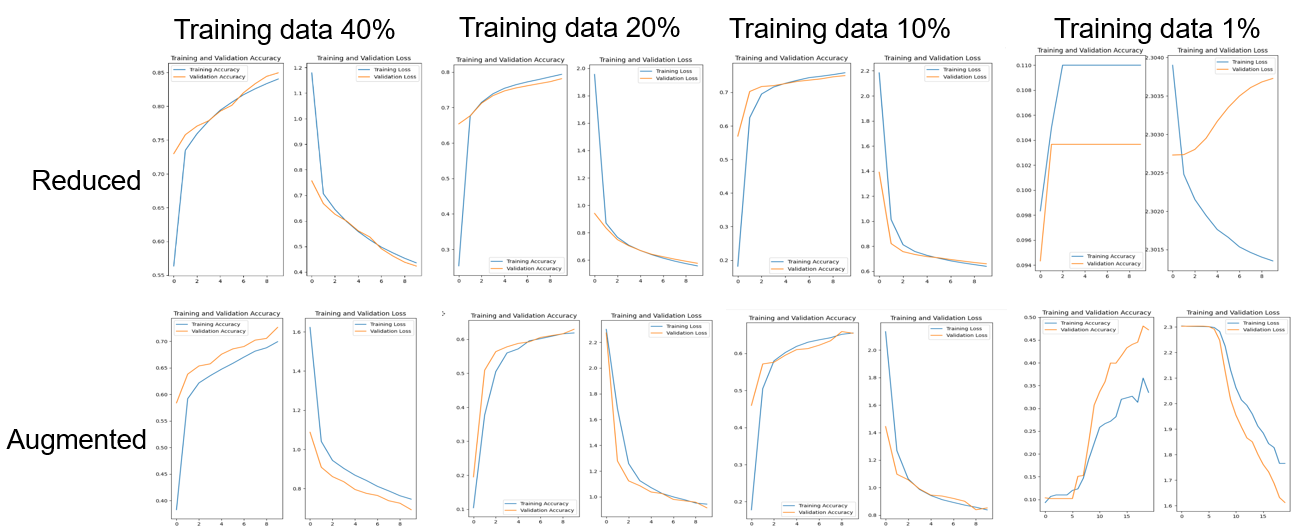


Figure 7: Compare of reduced and Augmented data

If we increase the epochs of the graph using the 1% training data used for data augmentation, it becomes more obvious to see that data augmentation is util. In contrast to the graph where only 1% of the data was used, the extended graph shows a higher degree of accuracy. Thus, we can conceptualize the utility of data augmentation

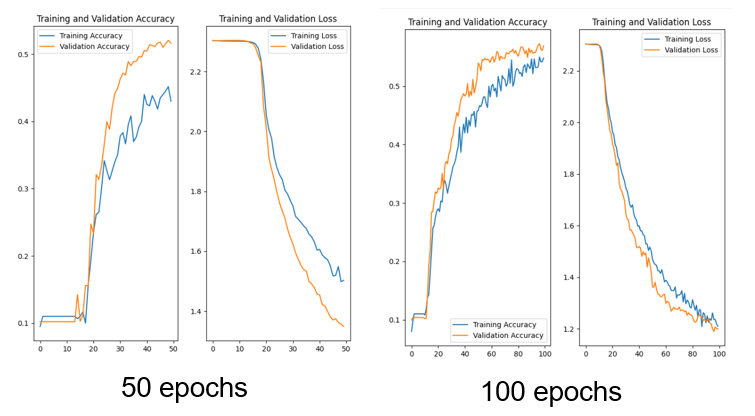


Figure 8: 50epochs and 100epochs of the augmentation of the 1% training data

# Data Augmentation for Time Series

## Background

For the quality of a neural network, the quality as well as the quantity of the underlying data set is crucial. This can be a particular challenge for time series, since these describe often processes, which are not repeatable or incomplete to a certain extent. An example for such an application is the prediction of earthquakes. [6] It is possible to generate data through simulations and experiments, but data sets from real events are only available to a limited extent. In order to still obtain a useful model, the data set can be extended by data augmentation as shown in this section.

## Dataset

The PeMSD7 dataset is used for the following experiments. This dataset consists of 228 speed measurement points on various expressways in District 7 of California, USA. Measurements were taken exclusively on weekdays by the California Department of Transportation in the period from May to June 2012. The average speed was measured in miles per hour (mph) at five-minute intervals. [7]

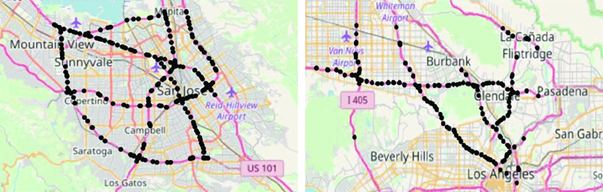


Figure 9: Speed measurement points [8]

The previously described data can be found in the *PeMSD7\_V\_228.csv* file of the dataset. Additionally, it includes the file *PeMSD7\_W\_228.csv*, which contains the distance between stations in miles in a 228 x 228 matrix. This can be used later in the process to create a weighting matrix.

## Data Selection and Preparation

Before the model can be applied to the data set, the data must first be prepared.

For this purpose, the data set is divided into a training, a validation and a test dataset. Then, the mean value and the standard deviation of the training data are determined for each parameter (measuring point). With the help of these parameters, all three datasets can then be normalized as shown in (1).

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

In this example, to show the effect of data augmentation, the number of measurement points was reduced to four and the number of data points was also strongly reduced to 144 each.

## Model

The model used is a neural network consisting of a spatio-temporal graph convolutional network (STGCN) and a long short-term memory (LSTM) layer. [7] The model is from a Keras example and has not been modified except for hyperparameters, termination conditions and an adjustment of the data structure. The last one was broken in the example due to an update of the dataset. The change was committed to the official GitHub repository as a bug fix as part of this project and has already been merged.



Figure 10: Structure of the neural network

The STGCN is made of two spatio-temporal convolutional blocks (ST-Conv block) and one deeply connected (dense) layer as output layer. Each ST-Conv block contains two temporally controlled convolutional layers and a spatial graph convolutional layer in between.

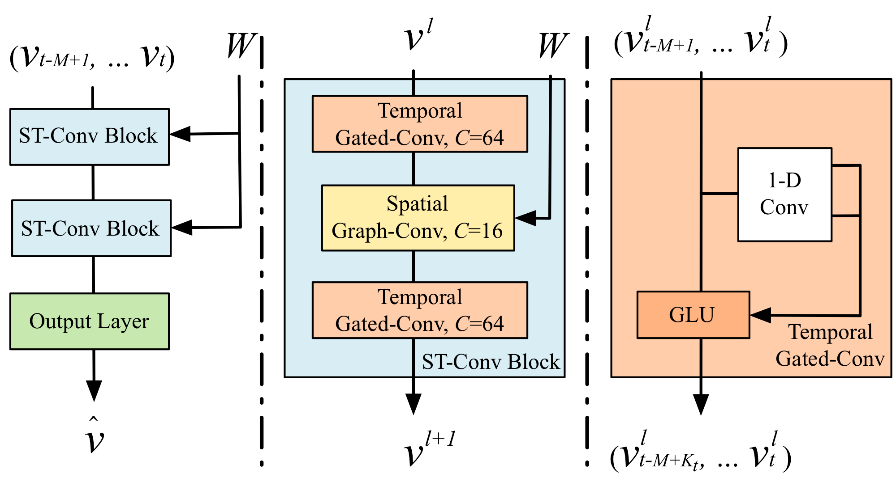


Figure 11: Internal structure of the STGCN [9]

Afterwards, the data output from the output layer of the STGCN is passed into a Keras LSTM layer with Rectified Linear Unit (ReLU) activation function. Before passing through another dense layer as the final output layer. [9]

## Data Augmentation

While data augmentation is a standard procedure when training neural networks for image processing, caution is required when using it for time series. Augmentation can cause the loss of information in the data set, such as temporal relationships. In the following, different techniques for data augmentation are presented using the time series shown in Figure 14 as an example.

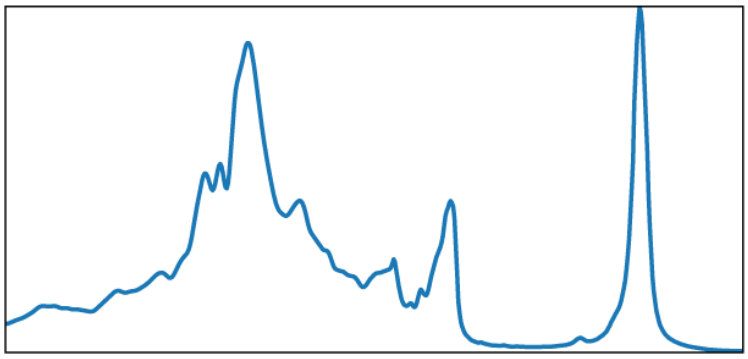


Figure 12: Example time series (original) [5]

### Jittering

Applying jittering to time series adds noise to the data, which allows the model to become robust to small fluctuations and improve its generalizability. The amount and type of jittering are hyperparameters and should be chosen based on the characteristics of the time series and the specific problem. [10]

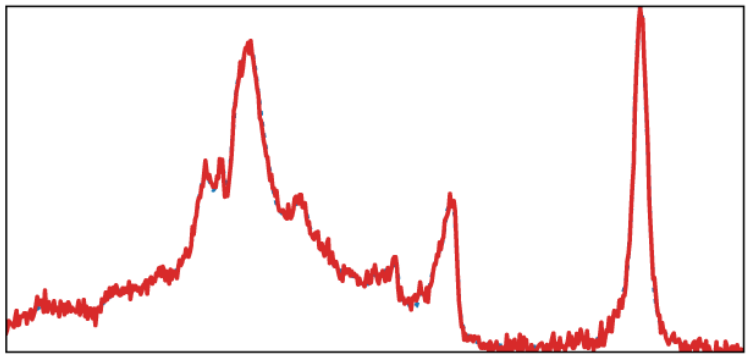


Figure 13: Example time series with jittering [10]

### Rotation / Flipping

Rotation and flipping are techniques used to transform time series by changing their orientation or direction. Vertical rotation, for example, refers to the process of rotating a time series around a reference axis. This rotation changes the vertical position of the data points while the temporal dimension remains unchanged. It is similar to the rotation of a graph around the x-axis. [10]

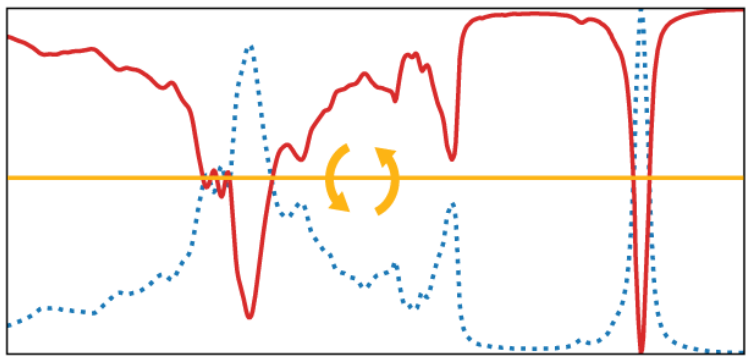


Figure 14: Example time series with rotation / flipping [10]

### Scaling

Scaling artificially creates additional training samples by rescaling the original time series data. Amplitude scaling multiplies the values of a time series by a constant factor. This technique adjusts the amplitude of the data while preserving its shape and temporal characteristics. By scaling the amplitude, the time series can represent variations in different orders of magnitude, which can help the model learn patterns and correlations.

Scaling can help increase the diversity and variability of the training data set, reducing the risk of overfitting and improving the model's ability to generalize to unseen data. [10]

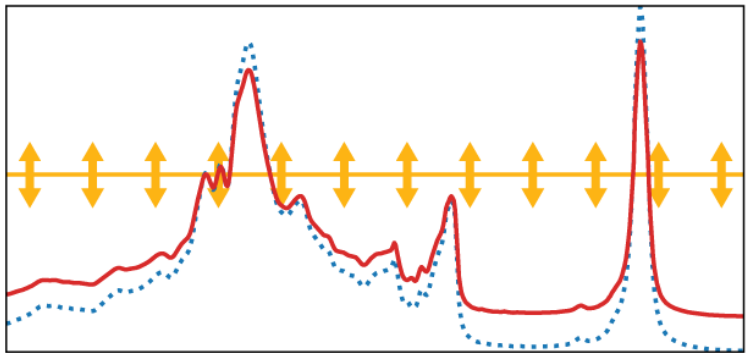


Figure 15: Example time series with scaling [10]

### Magnitude Warping

In magnitude warping, the original time series data is multiplied by a cubic spline with a defined number of nodes. The position of the nodes is chosen by a normal distribution and can be configured by the standard deviation and the number of nodes using the hyperparameters.

Magnitude warping increases the generalization of the model by randomly amplifying and reducing random regions of the training data. [10]

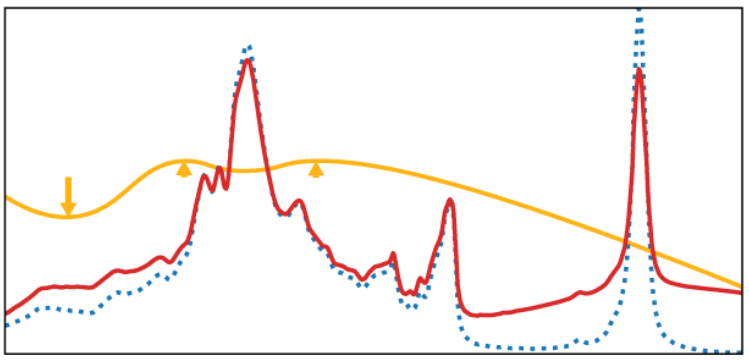


Figure 16: Example time series with magnitude warping [10]

### Permutation

In contrast to the previous methods, permutation is a temporal transformation. With this method, the original dataset is divided into areas, which are then randomly rearranged. It can be a number of areas defined as hyperparameters or a random number.

The permutation creates new patterns from the existing data. This can lead to better generalization and robustness, but can also easily lead to a degradation of the model. [10]

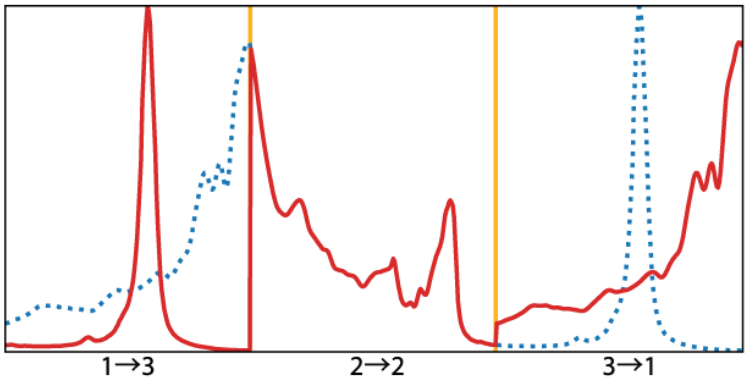


Figure 17: Example time series with permutation [10]

### Slicing

In slicing, or window slicing, the data set is extended by a window of a defined or random width selected from the original data. The position of the window has an additional effect on the model. For example, a random position, a moving or rolling window with and without overlapping can be used. [10]

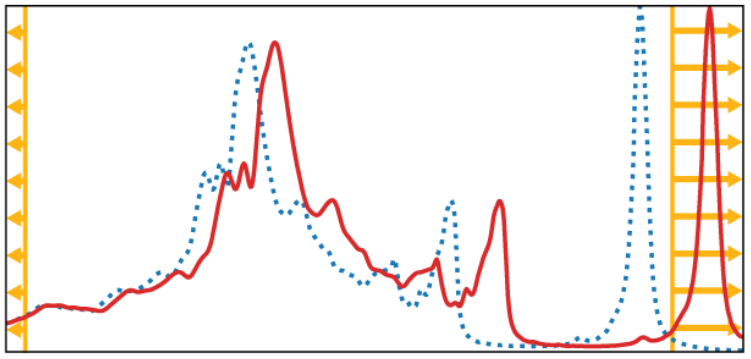


Figure 18: Example time series with slicing [10]

## Deployment in a real model

The previous section introduced different possibilities for data augmentation. In the following, some of the methods are applied to the model described in section 3.4.

### Integration into data pre-processing

The data set is first loaded, the parts to be processed are selected, and afterwards the data set is split into three parts. On one of these three parts, the training data set, the data augmentation is applied. It is important that it is never applied to the validation or test data.

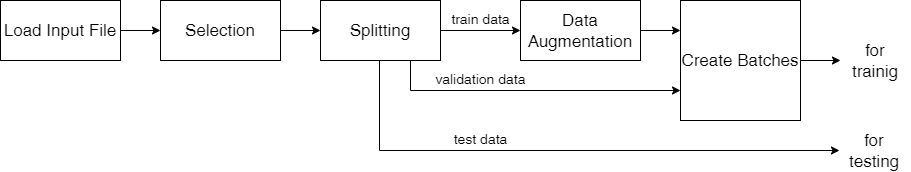


Figure 19: Flow of data pre-processing

Different data augmentation functions can and often must be used simultaneously. These can start from the original real data or be built up cascaded. As in the following code excerpt both variants are represented. For example, the *flipping* and *scaling* functions are applied to the real data in *train\_array*, while the *jitter* function is applied to the entire augmented dataset in *da\_array*.

1. # copy real training data

2. da\_array = train\_array

3.

4. # use DA function vertical flipping

5. da\_array = np.append(da\_array, flipping(train\_array), 0)

6.

7. # use DA function scaling

8. da\_array = np.append(da\_array, scaling(train\_array, 0.5), 0)

9.

10. # use DA function jitter (add noise)

11. da\_array = np.append(da\_array, jitter(da\_array, 0.2), 0)

12.

13. # use augmented data as training data

14. train\_array = da\_array

### Without Data Augmentation

Without data augmentation, the training of the model shows a strong overfitting. In addition, there is a very strong difference between the real test dataset and the forecast.

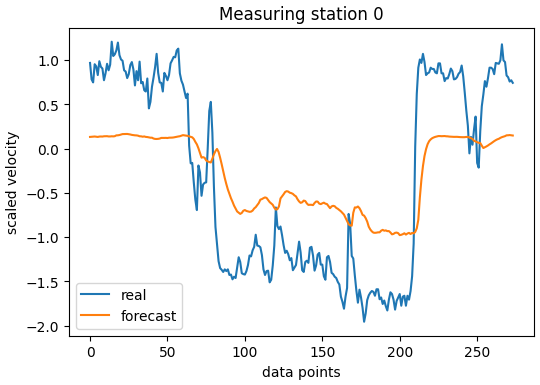
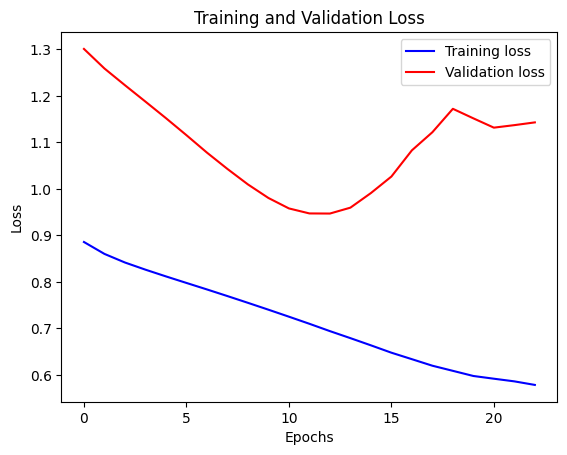


Figure 20: Without data augmentation (training results left; forecasting right)

### Scaling

The scaling function is executed three times in the implementation with different hyperparameters. This increases the size of the data set by a factor of three and leads to a better fit of the model, but cannot prevent the overfitting.

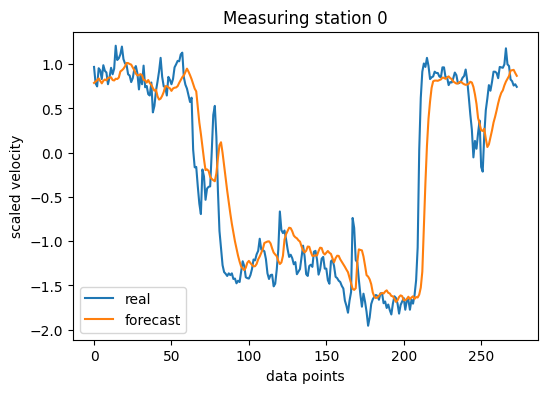
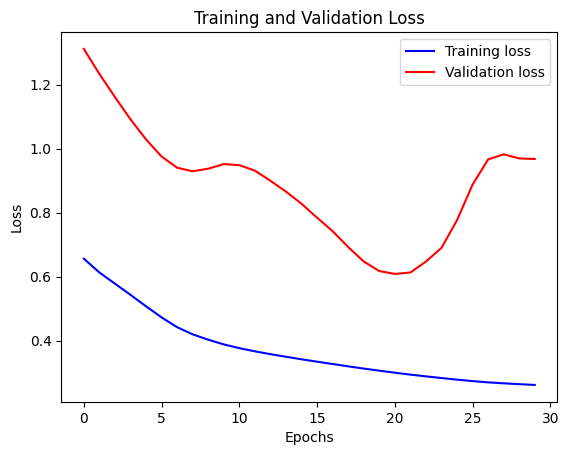


Figure 21: Real results for training with Scaling (training results left; forecasting right)

### Jittering

The jitter function without a further data augmentation does not bring a clear improvement in the training. When using noise with a low amplitude, no change is visible, while with a high amplitude the model performs even worse.

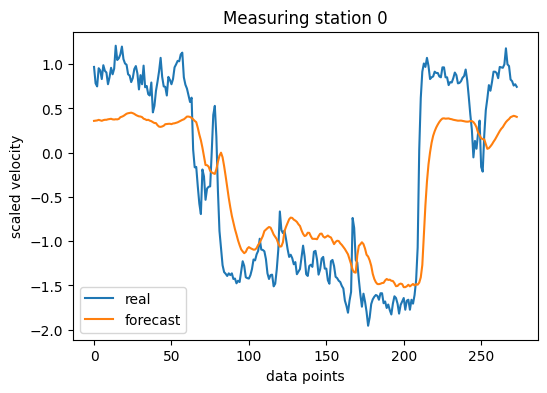
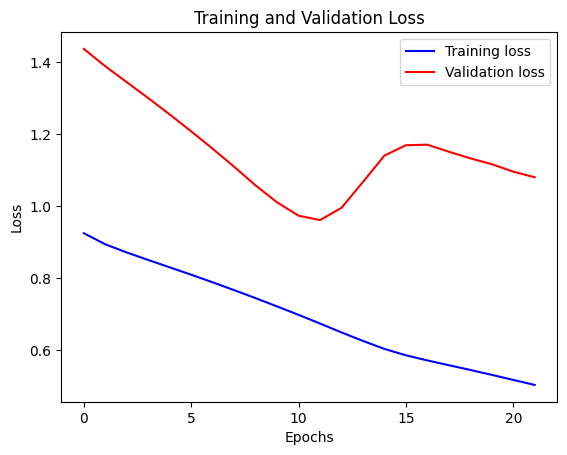


Figure 22: Real results for training with jittering (training results left; forecasting right)

### Vertical Flipping

The vertical flip brings a small improvement during training and compared to the jitter the forecast improves in the areas with high amplitude. It is also noticeable that there is a significantly increased overfitting, which occurs reproducibly.

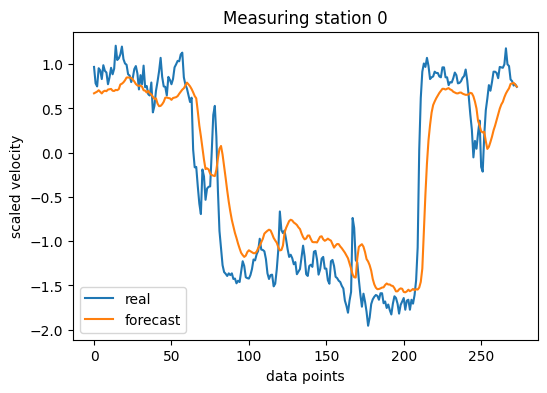
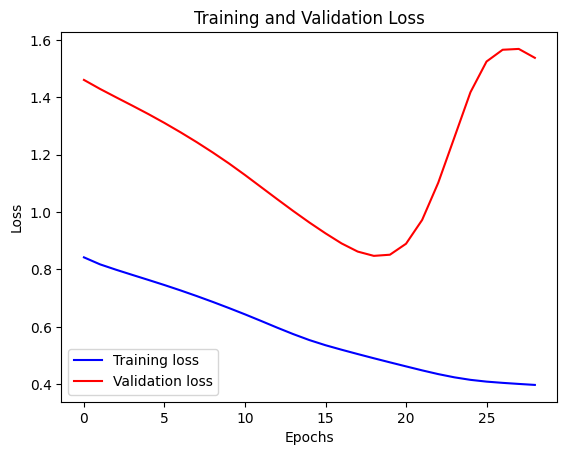


Figure 23: Real results for training with flipping (training results left; forecasting right)

### Combination of the Methods

By combining the methods described above, a significant improvement of the model can be achieved. The forecast is very similar to the real curve, but shows a slight shift in time.

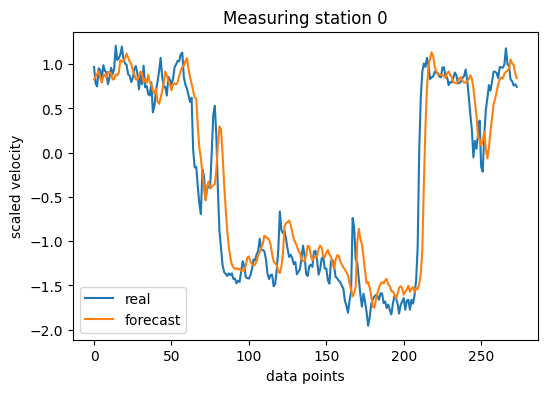
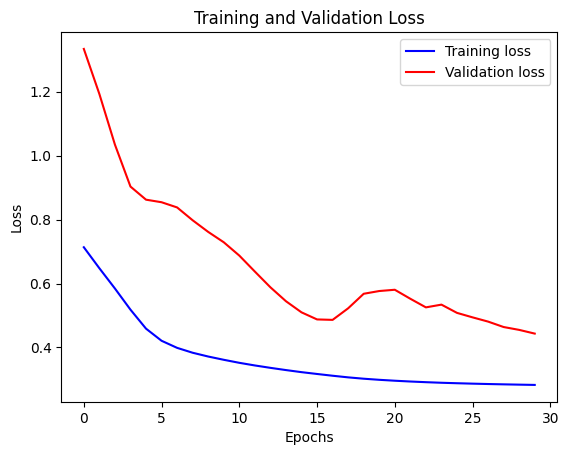


Figure 24: Real results for training with combined DA methods (training results left; forecasting right)

# Conclusion

Data augmentation techniques play a crucial role in improving the quality and performance of neural network models for both image and time series data.

For image data, data augmentation helps overcome overfitting by increasing the diversity and the amount of the training set. Data augmentation can be made by using Keras in preproducing layers and tf.images. Techniques such as random flip, rotation are shown to augment the data according to the libraries of TensorFlow. The utility of data augmentation was visualized based on graphs depicting training and validation accuracy and loss. It demonstrated the fact that as the amount of training data decreased, the need for augmentation increased. By applying data augmentation techniques, the accuracy of the model improved, showcasing the effectiveness of this approach.

In the context of time series data, data augmentation presents unique challenges due to the limited availability of real event datasets. However, various techniques were explored to augment time series data, including jittering, rotation/flipping, scaling, magnitude warping, permutation, and slicing. Each technique offers its own benefits and considerations, and caution must be exercised to preserve temporal relationships and avoid information loss. The PeMSD7 dataset, which captures traffic speed on expressways, served as an example for time series data augmentation. Through the application of different augmentation techniques and the training of a STGCN with LSTM layers, significant improvements in forecasting accuracy were achieved. The combined augmentation methods demonstrated the most promising results, closely aligning the forecasted curves with the real data.

In conclusion, data augmentation techniques provide valuable tools for enhancing the quality, generalizability, and robustness of neural network models. By increasing the diversity and size of the training data, data augmentation helps prevent overfitting, improve model performance, and enable better generalization to unseen data. However, it is essential to carefully select and apply augmentation techniques based on the specific characteristics and requirements of the data domain. 

References

[1] J. Lemley, S. Bazrafkan, and P. Corcoran, “Smart Augmentation Learning an Optimal Data Augmentation Strategy,” 2017. [Online]. Available: https://​arxiv.org​/​pdf/​1703.08383

[2] tensorflow, *Data Augmentation.* [Online]. Available: https://​www.tensorflow.org​/​tutorials/​images/​data\_​augmentation (accessed: Jul. 3 2023).

[3] J. Shijie, W. Ping, J. Peiyi, and H. Siping, “Research on data augmentation for image classification based on convolution neural networks,” in *2017 Chinese Automation Congress (CAC)*, Jinan, 2017, pp. 4165–4170.

[4] tensorflow, *TensorFlow: Image Classification.* [Online]. Available: https://​www.tensorflow.org​/​tutorials/​keras/​classification (accessed: Jul. 3 2023).

[5] tensorflow, *Fashion MNIST Dataset.* [Online]. Available: https://​www.tensorflow.org​/​datasets/​catalog/​fashion\_​mnist (accessed: Jul. 3 2023).

[6] W. Zhu, S. M. Mousavi, and G. C. Beroza, “Seismic signal augmentation to improve generalization of deep neural networks,” in *Advances in Geophysics, Machine Learning in Geosciences*: Elsevier, 2020, pp. 151–177.

[7] Arash Khodadadi, *Traffic forecasting using graph neural networks and LSTM.* [Online]. Available: https://​keras.io​/​examples/​timeseries/​timeseries\_​traffic\_​forecasting/​ (accessed: Jun. 26 2023).

[8] Papers With Code, *PeMSD7.* [Online]. Available: https://​paperswithcode.com​/​dataset/​pemsd7 (accessed: Jun. 26 2023).

[9] B. Yu, H. Yin, and Z. Zhu, “Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting,” 2017. [Online]. Available: https://​arxiv.org​/​pdf/​1709.04875

[10] B. K. Iwana and S. Uchida, “An empirical survey of data augmentation for time series classification with neural networks,” *PloS one*, vol. 16, no. 7, 2021, doi: 10.1371/journal.pone.0254841.