POLITECNICO DI TORINO

MASTER's Degree in MECHATRONIC ENGINEERING

MASTER's Degree Thesis

Data augmentation for neural network generalization optimization for indoor localization using infrared sensors

Supervisors

Prof. MIHAI TEODOR LAZARESCU

Prof. LUCIANO LAVAGNO

Candidate

MUNISKHON ABDURASHITOVA

APRIL 2022

Summary

Data augmentation is a technique for increasing the size of a training dataset artificially by producing modified versions of the original dataset. The ability of Neural Network models to generalize what they have learned to new data can be improved by training neural network models on more data, and the augmentation techniques can create variations of the datasets that can improve the ability of the fit models to generalize what they have learned to new unseen data. In this work, we will look at improving model inference generalization robustness by employing data augmentation techniques when training neural networks. There are a variety of strategies for generating augmented data depending on the problem at hand. The most widely used methods are primarily based on computer vision. In our situation, we are looking into human indoor localization in a constrained environment. Experimental data is time-series data taken with a low-resolution infrared(IR) camera (4x4 pixels). Well-known augmentation methodologies like Cropping, Flipping, Padding, Rotation, or Scaling may not work well on this occasion. We refer to systematic errors present both in instruments and in the experimental environment, such as ambient temperature, changing illuminations, and the use of electronic gadgets that produce heat. Different sorts of noise can be used to simulate all of these occurrences, which will be our strategy for generating augmented data. Our experiment explores Gaussian white noise and pink noise models at different amounts to improve the model inference generalization ability. Firstly, the model baseline obtained without augmentation techniques is generated and compared against the model results obtained with augmented data. Then we calculate considerable portions of IR sensor signal variation between a human detection and human absence as a starting noise amount for augmentation. The model is trained on one set and tested on three other sets. The generalization quality is calculated as the sum of all sets' mean square errors(MSE). Based on the experiment results, noise amplitude is modulated to improve generalization performance. All sets show various characteristics in the noise amplitude under investigation, with one showing a significant improvement and the others showing no improvement. When we add up MSE measurements, big gains in one or two sets may outweigh losses in other sets, making it difficult to determine which model

generalization is better. After obtaining the overall sum of MSEs, we extract the areas with decreased MSE sum values relative to the baseline to avoid "misleading" points. We should examine the extracted areas and conclude that the model has improved generalization for all sets if at least three sets show improvements in these intervals. The white noise amplitude ranges between [0; 0.6], consisting of 10 percent of the signal variation between person presence and person absence is investigated to check the model inference generalization. After analyzing the results, it is found that the model's overall behavior is best in the range [0; 0.3] showing improvements of at least three sets. For pink noise, reasonable amplitudes are higher than white noise because it represents small frequencies for high amplitudes and higher frequencies for small noise levels. Explored amplitude range is between [0; 3], giving the most promising noise range of [0;1], improving the overall model generalization. Finally, the combination of two noises is investigated simultaneously, giving the best model inference generalization in the white noise amplitude range [0; 0.2] and pink noise amplitude range [0; 1].

Acknowledgements

ACKNOWLEDGMENTS

Table of Contents

List of Tables

List of Figures

Acronyms

AI
artificial intelligence

NN
neural networks

IR
infrared

ANN
artificial neural network

RNN
recurrent neural network

NLP
natural language processing

LSTM
long short-term memory network

CNN
convolutional neural network

MSE
mean square error

VAE
variational autoencoder

GAN

generative adversarial network

Chapter 1

Introduction

1.1 Neural network training

Modern technology has advanced so far in our innovative world that we are all connected to artificial intelligence in many aspects of our life. Nowadays, artificial intelligence and machine learning are prevalent and in-demand topics among many scientists, researchers, and analysts. It is frequently debated among people in many other spheres like business, medicine, society, and industry. Creating intelligent systems or computers that can learn and train themselves, not requiring explicit programming or human interaction, is the main goal in this field. Systems deploy different machine learning techniques based on consideration and conscious reasoning on the problem, available data type, and size.

One of the methods used in machine learning, which describes data using the interconnection of neurons, is Neural Networks (NN). Each neuron receives a signal, analyses it, and then sends signals to other neurons it is connected to. Both the inputs and outputs are provided during training. The network processes the input data, generates output data, and compares its outputs to the desired outputs. Errors between desired and generated outputs are subsequently propagated back through the system, causing the weights that regulate the network to be adjusted. As the weights are gradually changed, this process repeats again.

The "training set" is the set of data used for the NN training. The same set of data is processed numerous times throughout the training of a network as the connection weights are improved, reducing the inference errors. Low training errors do not guarantee the quality of the NN model because the model must generalize well for new data sets, never seen by the model.

Based on the evolution of the error during training (training loss) and during validation (validation loss), the NN model can be overfitting or underfitting, both leading to poor generalization quality. Overfitting occurs when a NN model fails to

fit effectively on previously unseen data. Underfitting behavior represents a model that can neither model the training dataset nor generalize to the new dataset.

The NN generalization capacity is mainly determined by the network's system complexity and training. Application of regularization techniques like dropout, transfer learning, and batch normalization can overcome poor generalization of the model.

Another way to improve the model performance is by improving input data quality by employing augmentation techniques because insufficient input data during the "training" phase does not allow the model to learn adequately. Many different methods exist to generate augmented data based on the problem under consideration. Most popular techniques are mainly based on image recognition. In our case, we are exploring human indoor localization in a restricted area. Experimental data is time-series data collected by means of a very low resolution (4×4 pixel) infrared camera. In this scenario, the changes in the environment like ambient temperature, different illuminations, use of electronic devices which produce heat are systematic factors affecting collected experimental data. All these phenomena can be modeled by different types of noise, which will be our method to generate augmented data. We will discover appropriate noise levels by producing augmented data to increase input data size, which in turn improves model generalization.

1.2 Data augmentation for the improvement of neural networks performance

Data augmentation techniques artificially generate different versions of a real dataset to increase its size. Many practices have shown an increase in the accuracy of machine learning models after applying them. Although data augmentation can be utilized in various fields, it is most typically used in computer vision. According to an experiment, a deep learning model after image augmentation performs better in training loss & accuracy and validation loss & accuracy than a deep learning model without augmentation for the image classification task.

The following are some of the most frequent image data augmentation techniques: Cropping, Flipping, Padding, Rotation, Translation, Affine transformation, Scaling, Brightness, Contrast, Saturation, Color enhancement. Another technique of generating synthetic data is noise. Augmented sets are generated by adding different amounts of noise to the original data.

A common type of noise is white noise. It is a stochastic signal with the same intensity at all frequencies, resulting in a constant power spectral density. A white noise signal samples can be time-ordered or structured in one or more spatial dimensions.

Another type of noise that can be applied to our problem is the pink noise, which

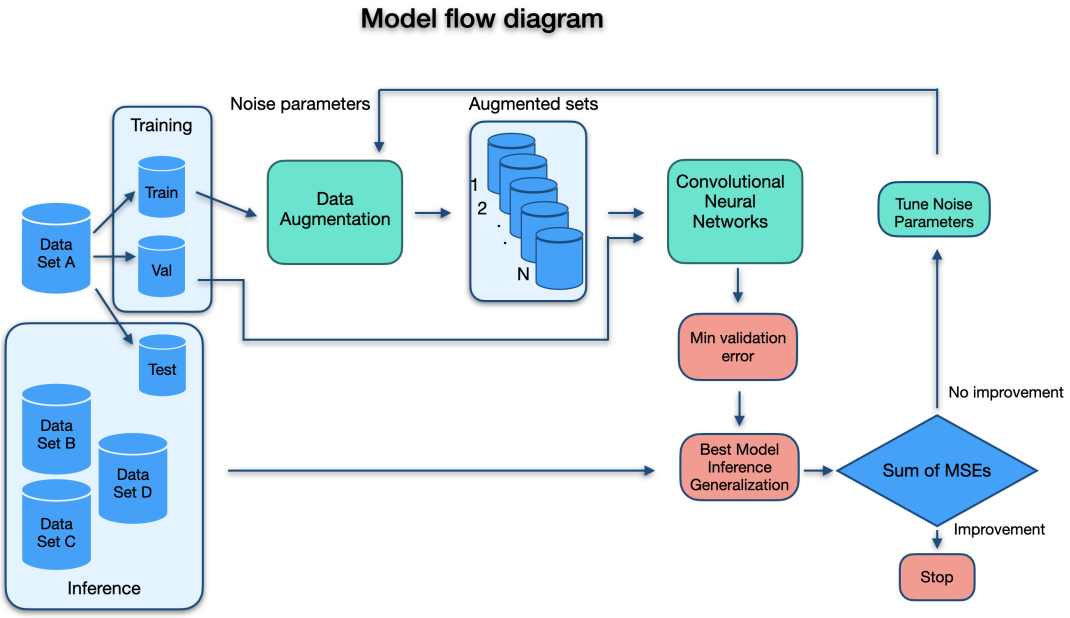

Figure 1.1: Noise parameter optimization flow for data augmentation based on the NN generalization capability.

emulates the long-term drifts in the system. It has amplitude at high frequencies, and increasing as the frequency decreases.

As shown in Figure 1.1, we start with four different sets of experimental data collected in various conditions like different ambient temperatures, walking at a different speed, and in different environments. The NN model is trained on one set and tested on the other three sets. Since the conditions for collecting the data are constrained in terms of time, the training data size is small.

We increase training data size by generating augmented sets with the addition of noise of different parameters. The purpose is to obtain higher generalization quality of the model because the same model is tested on three other sets. Noise parameters are tuned based on the model generalization capability, until the best one is found. All sets show various characteristics in the noise amplitude under investigation, with one showing a large improvement and the others showing no improvement. When we add up MSE measurements, big gains in one or two sets may outweigh losses in other sets, making it difficult to determine which model generalization is better. After obtaining the overall sum of MSEs, we extract the areas with decreased MSE sum values relative to the baseline to avoid "misleading" points. We should examine the extracted areas for each set separately. We can conclude that the model has improved generalization for all sets if at least three sets show improvements in these intervals. Assume that fewer than three sets demonstrate significant improvements

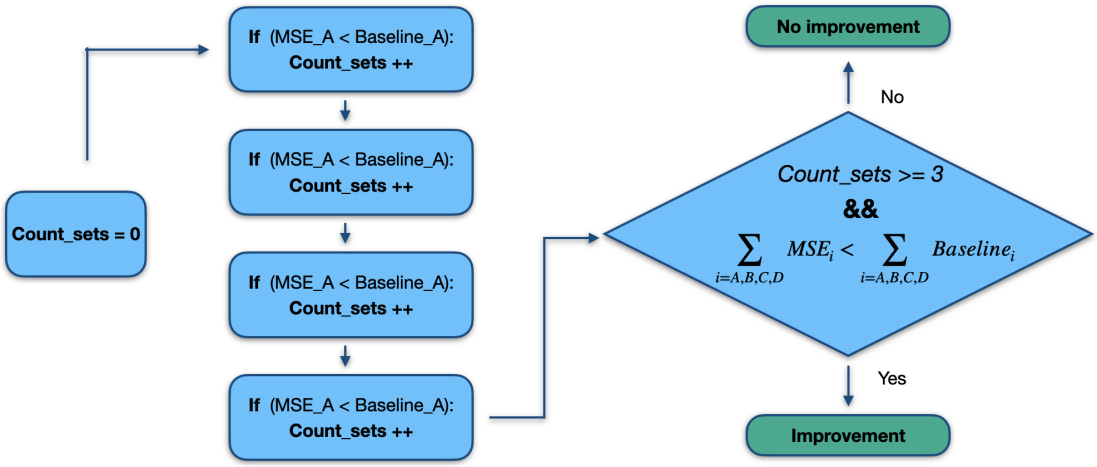

Figure 1.2: Flow of the model generalization capability inspection.

over the dismal outcomes of other sets. These noise intervals aren't deemed a noise range with greater generalization quality in that scenario. Considering this phenomenon we perform one more checking condition, that is described in Figure 1.2, to clarify the actual noise amounts giving better inference generalization.

Chapter 2

Neural network architectures and data augmentation techniques

2.1 Neural network architectures

An Artificial Neural Network (ANN) is a data processing paradigm modeled after the biological human brain. ANNs, like human brain, learn by doing. An ANN is tuned for a specific purpose through a learning process, such as pattern recognition or data classification. Although there are numerous neural network architectures, we discuss essential ones in this text that are as follows: feed-forward neural networks, recurrent neural networks, convolutional neural networks and long short term memory neural networks.

The feed-forward network is made up of perceptrons with three different types of layers: input layers, hidden layers, and output layers as shown in Figure 2.1. The signal from the preceding layer is multiplied by a weight, added to a bias, and sent via an activation function during each connection. Backpropagation is used in feed-forward networks to iteratively adjust the parameters until they achieve the desired performance.

The vanishing gradient problem, which occurs when networks are too long for valuable information to be backpropagated across the network, is one issue with deep feed-forward neural networks. The signal that updates the parameters gradually fades as it passes through the network until the front weights of the network are not altered or used. A Residual Network (shown in Figure 2.2) uses skip connections to propagate signals across a 'jumped' layer to solve this problem. By using less prone connections to vanishing gradients, the problem of disappearing

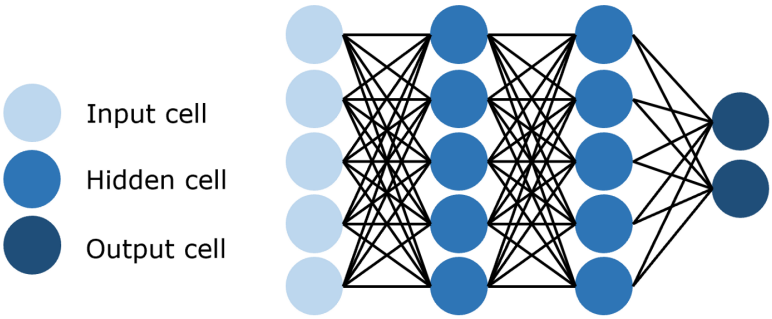

Figure 2.1: The feed forward neural networks general structure [1].

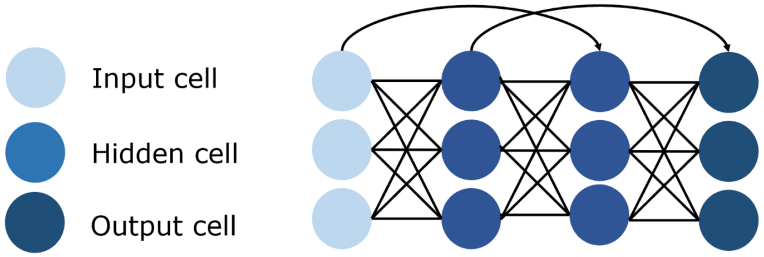

Figure 2.2: The residual networks general structure [1].

gradients is reduced. As the network learns the feature space, it learns to recover skipped layers over time, but it is more efficient in training because it is less subject to vanishing gradients and needs to explore less feature space.

A recurrent neural network(RNN) is a specific network with loops that recurs over themselves, hence the name "recurrent." RNNs use reasoning from previous training to make better, more educated judgements about upcoming events since they allow knowledge to be stored in the network. It accomplishes this by using past forecasts as 'context cues'. It is commonly used for ordinal or temporal problems, such as language translation, natural language processing (NLP), speech recognition, and image captioning; they are incorporated into popular applications such as Siri, voice search, and Google Translate. Like feedforward and convolutional neural networks, recurrent neural networks utilize training data to learn. They are distinguished by their "memory" as they take information from prior inputs to influence the current input and output. While traditional deep neural networks assume that inputs and outputs are independent of each other, the output of recurrent neural networks depend on the prior elements within the sequence. While future events would also be helpful in determining the output of a given sequence, unidirectional recurrent neural networks cannot account for these events in their predictions [2]. In Figure 2.3, we can see the general structure of this NN architecture.

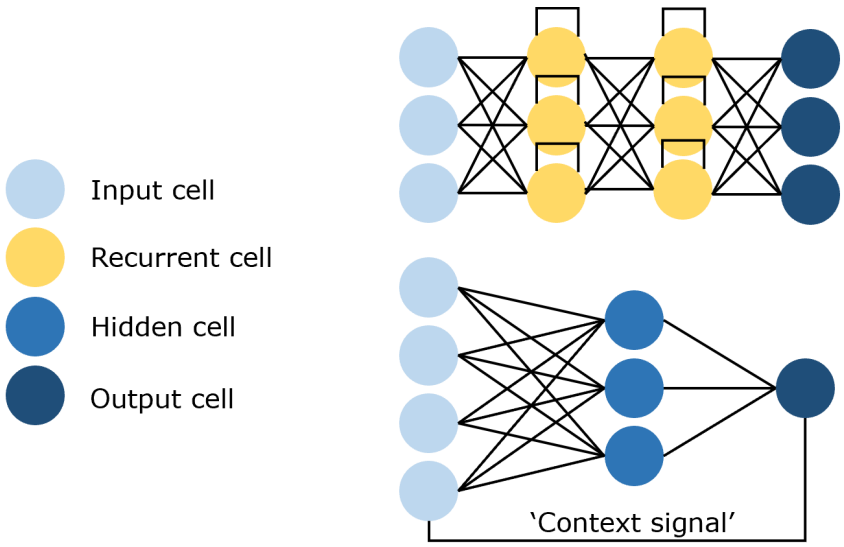

Figure 2.3: The recurrent neural networks general structure [1].

Recurrent neural networks (RNN) have a relatively narrow range of contextual information in practice. As a particular input is cycled around the network's connections, its influence (backpropagated error) on the hidden layer (and hence on the network's output) either explodes exponentially or decays to zero. A Long Short-Term Memory Network, or LSTM, is the answer to this vanishing gradient problem. This RNN architecture is tailored to solve the vanishing gradient problem by incorporating memory blocks into the topology. These blocks are similar to computer memory chips. They each contain many recurrently connected memory cells and three gates (input, output, and forget, equivalents of write, read, and reset). Because the network can only communicate with cells through each gate, the gates learn to open and close wisely to avoid exploding or vanishing gradients while simultaneously propagating important information via "constant error carousels" and rejecting useless memory content. Where standard RNNs fail to learn the presence of time lags larger than five to ten time steps between input events and target signals, LSTM, with general structure described as in Figure 2.4, is not affected and can learn to connect time lags even 1,000 time steps by enforcing a useful constant error flow [1].

A recurrent neural network with a sparsely connected hidden layer is called an echo state network (typically, a one-percent connectivity). The connectivity and weights of neurons are given at random, with no regard for layer or neuron differences (skip connections). The network learns the weights of output neurons such that it can make and reproduce specified temporal patterns. The idea for this network stems from the fact that, while being nonlinear, the synapse connections are the only weights that change during training, and so the error function can be differentiated

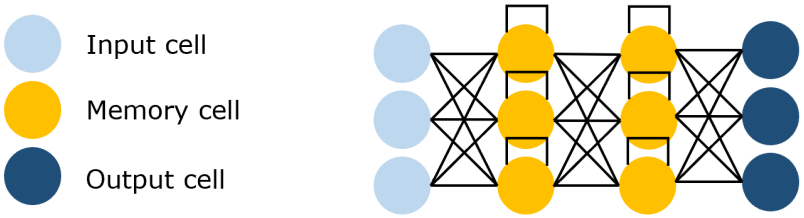

Figure 2.4: The long short term memory neural networks general structure [1].

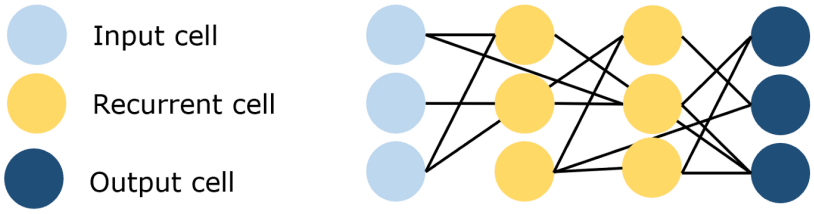

Figure 2.5: General structure of the echo state networks [1].

into a linear system. We can see in Figure 2.5 the structure of this network type.

Image processing is one of the popular practices in machine learning problems. Images have very high dimensionality and size. Training a standard feed-forward network to recognize them would necessitate hundreds of thousands of input neurons, which, aside from being prohibitively expensive computationally, can lead to a slew of issues related to the curse of dimensionality in neural networks. The Convolutional Neural Network (CNN) solves this problem by using convolutional and pooling layers to reduce an image dimensionality. Convolutional layers can highlight crucial areas of an image and pass each of them along because they are trainable but have fewer parameters than a standard hidden layer. The last few layers of a CNN are usually hidden layers that process the condensed picture information. Figure 2.6 describes the general structure of the CNN.

A Generative Adversarial Network (GAN) is a specialized sort of network that is made up of two networks: a discriminator and a generator. It is used to produce graphics. The discriminator job is to tell whether an image came from the dataset or was created by the generator. The generator job is to create convincing images that the discriminator cannot tell whether they are real or not. These two enemies compete over time, with rigorous regulation. Each one is driving to succeed in improving the other. Consequently, we will have a well-trained generator that can produce a realistic-looking image. The discriminator is a convolutional neural network to increase its accuracy in distinguishing real/fake images. At the same time, the generator is a deconvolutional neural network to reduce the discriminator

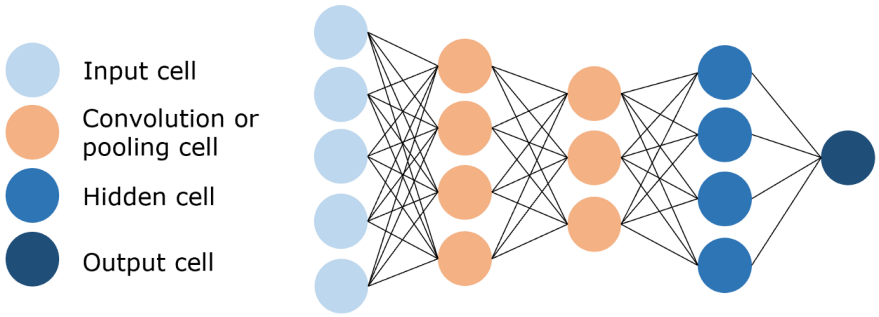

Figure 2.6: The convolutional neural networks general structure [1].

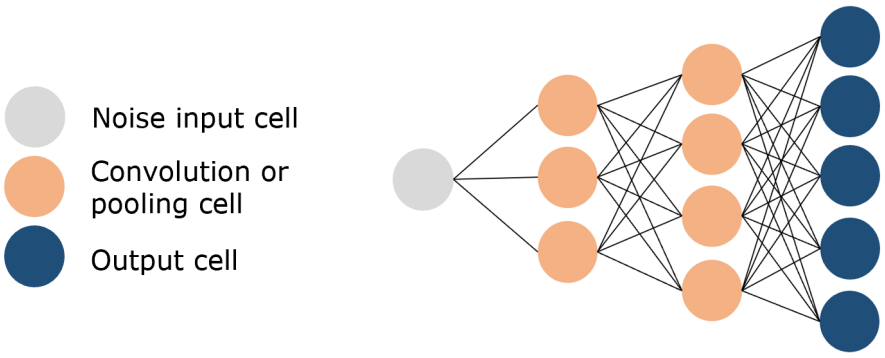

Figure 2.7: General structure of the generative adversarial networks [1].

performance.

An autoencoder basic concept is to take in high-dimensional input, compress it into highly informative and low-dimensional data, and then project the compressed data onto a new space. Dimensionality reduction, picture compression, denoising data, feature extraction, image synthesis, and recommendation systems are just a few autoencoder uses. It can be used as an unsupervised or supervised method, and it can provide a lot of information about the data. Convolutional layers can be used to replace hidden cells while processing photos as described in Figure 2.8.

A variational autoencoder (VAE) learns the probability distribution parameters representing the data. In contrast, an autoencoder learns a compressed representation of an input, which could be images or text sequences, for example, by compressing the input and then decompressing it to match the original input. Rather than just learning a function to describe the data, it acquires a more thorough and nuanced understanding of the data by sampling from the distribution and creating fresh input data samples. In this way, it is similar to a GAN in that it is entirely

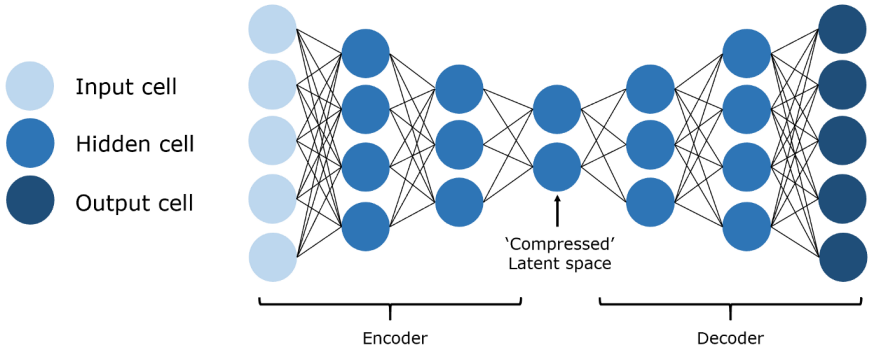

Figure 2.8: General structure of the auto encoder [1].

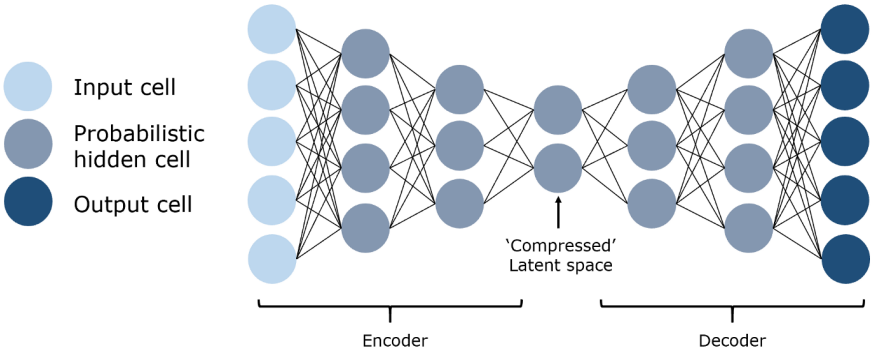

Figure 2.9: General structure of the variational auto encoder [1].

generative. Using a radial basis function, a VAE employs a probabilistic hidden cell that calculates the difference between the test case and the cell mean.

2.2 Augmented data generation

Data augmentation refers to a set of strategies for creating new training samples from existing ones by introducing random variations and disturbances, ensuring that the data is not destroyed. Our purpose is to boost the model generalizability using data augmentation. We are considering two types of noise: white noise and pink noise. First, we discuss the generation of augmented sets with the addition of Gaussian white noise to original data. We are exploring human indoor localization in a restricted area because it is trendy in the fields like energy management, health monitoring, and security. The purpose is to explore cheap but efficient techniques for indoor localization because for outdoors person tracking, there exists GPS technology that can quickly determine a person location through the wearable

tag or cell phone. At home or inside a room, a person is not always carrying a phone; that is why we should search approaches for tagless localization. Four sets of experimental data are collected in a 3x3 meter room for a short period. A 4x4 pixel Omron D6T-44L-06 thermopile infrared sensor is installed on the ceiling of a room, and the person reference location is collected with an ultrasound-based tag of the Marvelmind Starter Set HW v4.9 [3]. Each tuple of experimental data has 18 elements: 16 pixels of the infrared sensor (IR) plus the X and Y coordinates of the person representing the label.

The first set of experimental data is used for the model training in a 60/20/20 ratio representing training/validation/test sets. The other three sets are used only for inference because our purpose is to improve the model generalization, which refers to model performance for unseen data. Since the amount of training data is small, we generate synthetic data by adding Gaussian white noise. Noise has a normal distribution with zero mean and finite variance

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}. \tag{2.1}$$

Function *f(x)* is the probability density, σ is the standard deviation, μ is the mean. Mean determines how far the sample average of the data is expected to differ from the actual mean, whereas standard deviation assesses the variability (or dispersion) between individual data values and the mean. We consider nonzero variance, and zero mean to describe white noise parameters. Since the collected data is time-series data, we should keep the order unchanged. To combine original and augmented sets, we refer to each set as a batch and put all the batches sequentially, one on top of another, so that scenario fractions are not destroyed. Overall, 29 augmented are generated, resulting in a 30 times increase of original training data. Validation and test sets are not changed. The critical point is determining the amount of noise improving model generalization. A reasonable range for noise could be related to the signal variations of the sensor detecting a person and no person. Inside the sensor, 16 pixels values are translated into temperature values representing a person presence with a higher temperature.

We start by analyzing the sensor single-pixel variations. The highest point means person presence, while the lowest point refers to the absence of a person. The difference between these two magnitudes will be the basis for searching noise amounts. In Figure 2.10, we can see the sensor one-pixel output variations. The deviation between max and min points is approximately 5 °C. We investigate noise levels at different ratios of signal magnitude obtained from deviation like $\frac{1}{25}$, $\frac{1}{20}$, and $\frac{1}{10}$ which could be reasonably small noise amounts. Another type of noise, pink noise (1/f noise), is used to generate augmented data for emulating long-term drifts in the system. It represents a frequency spectrum where the power spectral density (power per frequency interval) is inversely proportional to the signal frequency.

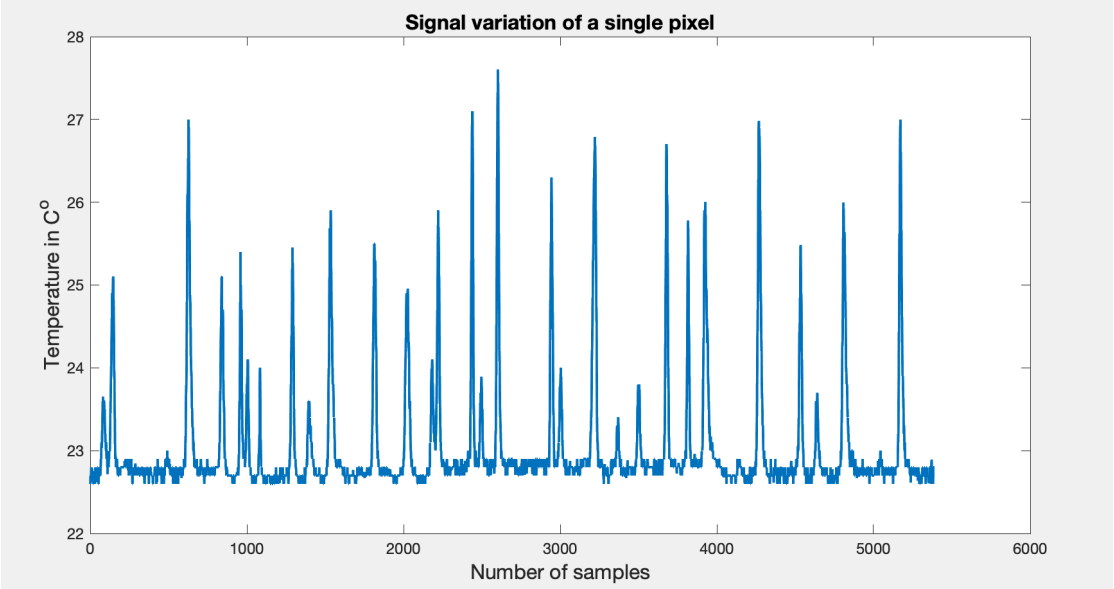

Figure 2.10: IR sensor 1 pixel output signal variation collected at 5 Hz for 18 minutes. The horizontal axis represents original training data set samples. The vertical axis represents the corresponding temperature variation detected by a single pixel.

Each octave interval in pink noise delivers the same amount of noise energy. The power spectral density of pink noise is represented as follows:

$$S(f) \propto \frac{1}{f} \tag{2.2}$$

We use the same techniques that we have used for white noise for deciding noise levels of pink noise. Independent exploration of two noises to improve model generalization quality gives an insight into using a combination of noises to generate augmented sets. Naturally, all systems undergo different perturbations simultaneously at the same time. The third method of augmented data generation is combining white and pink noises. This section mainly focused on the different augmentation techniques we applied to our problem without explicitly giving noise levels. Reasonable noise levels will be determined after performing many trainings. We will discuss reasonable noise levels in the following sections.

2.3 NN training with augmented data

We use the CNN architecture in our problem because the IR sensor captures a room pictures dynamically with 16 pixels. Overall data is the time series organization

of each frame which generates a "movie". Once the appropriate architecture for the model is decided, next step is working on the model input data. The neural network can be seen as a function that accepts some input and outputs a response. It has a domain, just like all other functions. To ensure that the values we wish to provide to the neural net are in the domain, we must first normalize them. If the arguments are out of the domain, the outcome is not guaranteed to be appropriate, as it is with all functions. The actual behavior of the NN on arguments outside of the domain is determined by the neural net implementation. However, if the arguments are not within the domain, the result is worthless. Except for the labels, all the input data are rescaled between [0; 1]. Both the original IR sensor data and augmented sets are normalized and fit the model input domain. The next step is feeding the input to our model with CNN architecture. Let us discuss what kind of layers are used in our model in detail.

In convolutional neural networks, the principal building elements are convolutional layers. Convolution refers to a process of applying a filter to an input data to produce an activation. When the same filter is applied to input multiple times, a feature map is created, displaying the positions and strength of a recognized feature in input, such as an image. The capacity of convolutional neural networks to learn many filters in parallel, particularly to a training dataset, under the restrictions of a specific predictive modeling problem, such as image classification, is its unique feature. As a result, exact traits appear on input photographs that can be identified everywhere.The CNN is a neural network model created for working with two-dimensional picture data. At the same time, it can also be utilized with one-dimensional and three-dimensional data. The convolutional layer, which gives the network its name, is at the heart of the convolutional neural network. This layer performs a process known as "convolution." A convolution is a linear process in a convolutional neural network that involves the multiplication of a set of weights with the input, similar to a standard neural network. The multiplication is done between an array of input data and a two-dimensional array of weights, called a filter or a kernel, because the approach was created for two-dimensional input. The filter size is more compacts than the input, and the dot product is used to multiply a filter-sized patch of the input with the filter. A dot product is the element-wise multiplication of the input and filter filter-sized patch, which is then summed, always yielding a single value. The procedure is often referred to as the "scalar product" because it produces a single value. It is intentional to use a filter that is smaller than the input because it facilitates the same filter to be multiplied by the input array several times at different points on the input. From left to right, top to bottom, the filter is applied systematically to each overlapping section or filter-sized patch of the incoming data. This notion of applying the same filter to an image systematically is a great one. Suppose the filter is designed to detect a specific feature in the input. In that case, applying it systematically throughout the entire image gives the filter the chance to find that

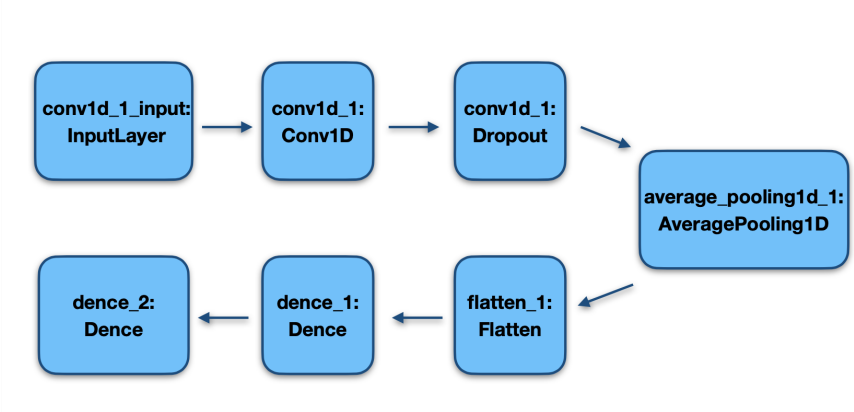

Figure 2.11: Plot of the experimental CNN model layers as a graph.

feature anywhere in the image. This property is known as translation invariance, and it refers to the general interest in whether a feature exists rather than where it exists.

The CNN architecture does not always guarantee appropriate model fit. In order to avoid overfitting, we add to our model some regularization techniques. A simple and powerful regularization method for neural networks and deep learning models is a dropout. Dropout is a technique where randomly selected neurons are ignored during training. They are "dropped out" randomly as described in Figure 2.12. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass, and any weight updates are not applied to the neuron on the backward pass [4]. As a neural network learns, neuron weights settle into their context within the network. Weights of neurons are tuned for specific features providing some specialization. Neighboring neurons rely on this specialization, which, if taken too far, can result in a fragile model too specialized to the training data. This reliance on context for a neuron during training is complex co-adaptation.

Other neurons will have to take an action and manage the representation required to make predictions for missing neurons if neurons are randomly dropped out of the network during training. As a result, the network learns numerous distinct internal representations becoming less sensitive to the weights of individual neurons. In other words we can say that, the network is better equipped to generalize and is less prone to overfit the training data. Convolutional layers summarize the existence of features in an input image in a convolutional neural network. The placement of the features in the input is sensitive to the location of the features in the output feature maps, which is an issue. Downsampling the feature maps is one way to deal with this sensitivity. This has the effect of making the downsampled feature maps more resistant to changes in the position of the feature in the image, which is referred to as "local translation invariance" in technical terms. By summarizing the existence of features in portions of the feature map, pooling layers provide a method for

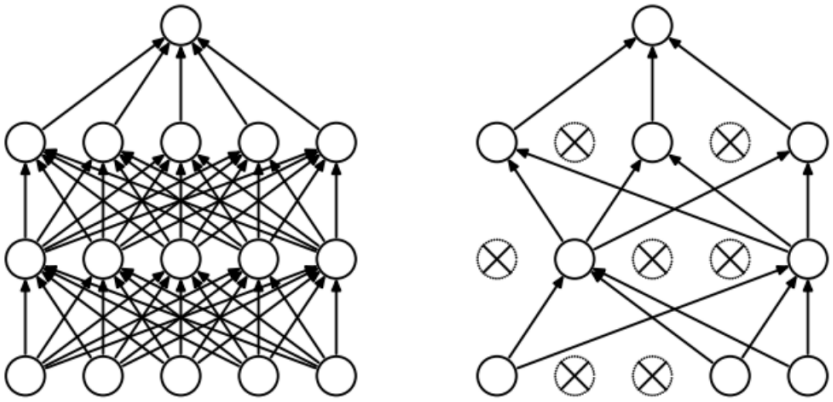

Figure 2.12: Neural net internal structure change after applying dropout [5].

downsampling feature maps. Average pooling and max pooling are two prominent pooling approaches that recap the average feature existence. A convolutional neural network convolutional layers apply learned filters to input images systematically to build feature maps which summarize the presence of those features in the input. Convolutional layers are particularly successful, and stacking them in deep models allows layers adjacent to the input to learn low-level characteristics (e.g., lines) while layers farther in the model learn high-order or more abstract features, such as forms or individual objects. The feature map output of convolutional layers has the drawback of recording the exact position of features in the input. This means that even tiny changes in the feature position in the input image will result in a different feature map. Re-cropping, rotation, shifting, and other techniques can cause this problem. In this case downsampling is a typical signal processing technique for overcoming the issue. This is when a reduced resolution version of an input signal is made, keeping the main or critical structural features but removing the fine detail that may not be as valuable to the task. A pooling layer is a more robust and standard method of downsampling. After the convolutional layer, a new pooling layer is introduced, specifically after applying a nonlinearity (e.g., ReLU) to the feature maps produced by a convolutional layer.The activation function goal is to introduce non-linearity into a neuron output. We know that neurons in a neural network work in accordance with their weight, bias, and activation function. We would change the weights and biases of the neurons in a neural network based on the output error. Back-propagation is the term for this procedure. Because the gradients are supplied simultaneously with the error to update the weights and biases, activation functions enable back-propagation. Without an activation function, a neural network is just a linear regression model. The activation function transforms the input in a non-linear way, allowing it to learn and accomplish more

complex tasks. In Table 2.1 we can see the most used activation functions.

A popular strategy for arranging layers within a convolutional neural network is to add a pooling layer after the convolutional layer. This pattern could be repeated one or more times in a particular model. The pooling layer works on each feature map separately to build a new set of pooled feature maps with the same number of features. Pooling is similar to applying a filter to feature maps in that it entails selecting a pooling procedure. The pooling operation or filter is usually smaller than the feature map. The following are two standard pooling functions:

- Average pooling calculates the average value for each patch on the feature map.
- Max pooling calculates the largest value for each patch of the feature map.

A final version of the features detected in the input results from using a pooling layer and creating downsampled or pooled feature maps. They are helpful because small changes in the feature location in the input that the convolutional layer detects will result in a pooled feature map with the feature in the same place. The model invariance to local translation is a feature added via pooling. When creating deep learning neural network models, weight initialization is a critical design decision. Weight initialization used to be done with small random integers, but over the last decade, more specific heuristics have been created that take variables like the type of activation function being used and the number of inputs to the node [4]. The stochastic gradient descent optimization approach can be used to train neural network models more effectively with these more personalized heuristics. In creating a neural network model, weight initialization is an essential factor. In neural networks, nodes are made up of weighted parameters used to produce a

Table 2.1: Examples of activation functions, operating either element-wise or vector-wise, depending on the function

ReLU	$f(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$
Softmax	$f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}} i = 1, ..., J$
tanh	$f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$
sigmoid	$f(x) = \frac{1}{(1 + e^{-x})}$

weighted total of the inputs using this formula:

$$\text{output} = f_{\text{activation}} \left(\sum_{\#\text{neurons}} input_i + bias \right) \tag{2.3}$$

Stochastic gradient descent is an optimization approach for fitting neural network models that incrementally modify the network weights in order to minimize a loss function, resulting in weights for the mode that may make compelling predictions. This optimization procedure requires a beginning point in the space of possible weight values to begin the optimization process. Weight initialization is a process that involves setting the weights of a neural network to small random values that serve as the beginning point for optimization (learning or optimization). A neural network is initialized with a distinct set of weights each time, which results in a different optimization process starting point and, possibly, a different end set of weights with varied performance characteristics. We cannot set all weights to 0.0 since the optimization technique causes some asymmetry in the error gradient, making it impossible to search effectively.

The `compile()` function is used to specify the loss function, optimizer, and metrics for a pre-built model, all of which will be detailed later. These are crucial characteristics of a neural network final predictions. Let us start discussing the function of loss that is set inside the compile function. On a high level, the model established in the starter notebook will learn how to identify particular images as digits, and it will do so by generating a forecast, determining how far off its prediction was from the true answer, and then updating itself to predict those types of digits better. The loss function is the part of the model that calculates the distance between a prediction and the correct answer. Different types of loss functions will be required for different types of models. The loss function for a situation like this, where the outputs to our model are probabilities, would have to be substantially different from the loss function for a model trying to forecast something like price in dollars. This model loss function is sparse categorical cross-entropy, which is suitable for multi-class classification situations such as this one. A significant loss will result in our scenario if the model predicts that an image has just a tiny chance of being its accurate label. After initializing the model weights training is repeated several times to reduce the value of loss function. In Table 2.2 we can see the most employed loss functions in practice.

Another way to put it is that you are trying to minimize the loss when you train a model. If loss measures how far a forecast is from the correct answer, and a higher loss indicates a more erroneous prediction, minimizing loss is a measurable way of measuring how well a model performs. As previously stated, updating the mathematical parameters of the nodes of a network based on how effective those parameters were in categorizing a picture is an important aspect of

training neural networks. Backpropagation is a technique in which neural networks adjust parameters to improve the model using a mathematical tool called gradient descent. The intricacies of those terms are outside the scope of this article, but the optimizer parameter of the model is essential to grasp what the essential notebook is doing. The compile() function offers a method for speeding up and improving the backpropagation process. The "adam" optimizer is a widely used optimizer that performs well in this situation. In this algorithm 1 we can see the full procedure with the sequence of instruction that performs the adam optimizer.

The metrics to evaluate the model are specified by the compile() function. Accuracy is a valuable but flawed statistic for assessing model performance. It should be used with caution when used alone. Many hyperparameters have to be tuned to have a robust convolutional neural network that will be able to accurately classify images. One of the most important hyperparameters is the batch size, which is the number of images used to train a single forward and backward pass [6]. The number of times the model sees all of the training data is specified by the epoch parameter in the fit() function. We want the model to see all of the training data several times because one pass may not be enough for the model to change its weights sufficiently when computing weighted sums to enhance prediction power significantly. Every machine learning algorithm has a set of basic parameters that can be tweaked to improve accuracy. During the fitting phase, we create a machine learning model by running an algorithm on data for which you know the target variable, also known as "labeled" data. The correctness of the results is then determined by comparing them to actual, observed values of the target variable. Then we use that data to tweak the algorithm standard settings to reduce error and improve its accuracy in detecting patterns and relationships between the rest of its features and the target. We repeat this procedure until the algorithm discovers the best settings for producing valid, practical, and usable insights for our real-world business challenge. The outcomes produced by our model will not be

Table 2.2: y is the output of the network, N is the batch size multiplied by the number of outputs (e.g. pixels), C is the number of classes and $\hat{y}$ is the correct output.

MSE / L2 Loss / Quadratic Loss	$\frac{\sum_{i=1}^{N} (y_i - \hat{y}_i)^2}{N}$
(Binary) Cross Entropy (average reduction on higher dimensions)	$\frac{\sum_{i=1}^{N} \sum_{j=1}^{C} \hat{y}_i \log (y_{i,j})}{N}$
Categorical Cross Entropy (sum reduction on higher dimensions)	$-\sum_{i=1}^{N} \hat{y}_i + \log \left(\sum_{i=1}^{N} \sum_{j=1}^{C} y_{i,j} \right)$

Algorithm 1 Adam optimizer algorithm. All operations are element-wise, even powers. Good values for the constants are $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$. ϵ is needed to guarantee numerical stability.

procedure ADAM($\alpha, \beta_1, \beta_2, f, \theta_0$)
  ▷ α is the stepsize
  ▷ $\beta_1, \beta_2 \in [0, 1)$ are the exponential decay rates for the moment estimates
  ▷ $f(\theta)$ is the objective function to optimize
  ▷ θ_0 is the initial vector of parameters which will be optimized
  ▷ Initialization
  $m_0 \leftarrow 0$ ▷ First moment estimate vector set to 0
  $v_0 \leftarrow 0$ ▷ Second moment estimate vector set to 0
  $t \leftarrow 0$ ▷ Timestep set to 0
  ▷ Execution
  while θ_t not converged **do**
    $t \leftarrow t + 1$ ▷ Update timestep
    ▷ Gradients are computed w.r.t the parameters to optimize
    ▷ using the value of the objective function
    ▷ at the previous timestep
    $g_t \leftarrow \nabla_\theta f(\theta_{t-1})$
    ▷ Update of first-moment and second-moment estimates using
    ▷ previous value and new gradients, biased
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$
    ▷ Bias-correction of estimates
    $\hat{m}_t \leftarrow \dfrac{m_t}{1 - \beta_1^t}$
    $\hat{v}_t \leftarrow \dfrac{v_t}{1 - \beta_2^t}$
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \dfrac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$ ▷ Update parameters
  end while
  return θ_t ▷ Optimized parameters are returned
end procedure

accurate enough to be useful for actual decision-making if it does not fit your data appropriately. Hyperparameters in a correctly fitted model capture the complicated interactions between known factors and the target variable, allowing the model to identify useful insights and generate accurate predictions. Fitting is an automated process that ensures our NN models have the particular parameters that are most suited to solving our specific real-world problem with high accuracy.

Chapter 3

Experimental results

3.1 Augmentation for optimal generalization

The model architecture and input data are designed based on the state-of-the-art. Next, we explore the augmentation parameters that improve best the neural network generalization performance. For each combination of parameters, the neural network model is trained 30 times with the same input data because the training process is stochastic. Each time model weights are initialized with random variables resulting in different output values for each training. Out of these 30 independent trainings, the best one is chosen based on the generalization ability. To determine the generalization capability of the neural network model, we use four sets of experimental data, denoted as A, B, C, D. They are collected in different days, for different movement trajectories, and in different environmental conditions. As a metric for the generalization capability is calculated the sum of the mean square error (MSE) for each of the four sets, A, B, C, and D, as follows:

$$\text{Overall\_Performance} = MSE_{\text{A}} + MSE_{\text{B}} + MSE_{\text{C}} + MSE_{\text{D}}. \tag{3.1}$$

We consider only the training with the lowest Overall_Performance out of the 30 that we do for each parameter set. First, we generate a baseline against which to track the generalization improvement. The baseline is obtained by training the model with original data without augmented sets, and resulting values are given in the table 3.1. Our aim at generating augmented sets is to acquire improved generalization compared to the model training without augmentation. As we previously discussed, noise amplitude is related to the signal amplitude, i.e., between the signal when a person is detected and when there is no person. The signal change is calculated based on the variance of the sensor pixel outputs. Figure 2.10 shows the maximum signal variance around 5 °C, because the corresponding IR sensor pixel values are converted into temperature by the sensor. We explore thus the behavior of the model with

noise amplitudes a fraction of the maximum signal, starting from 1 %, and increasing until the effect of the noise is detrimental for the generalization (at around 12 %).

3.2 White noise augmentation generalization

For the white noise, the standard deviation parameter allows modulating the noise amplitude. That is why the first experiment trains the model with the following values {0.01, 0.11, 0.21, 0.31, 0.41, 0.51, 0.61}. Table 3.1 represents the output of the model for each white noise parameter. For each white noise amplitude in the range {0.01, 0.11, 0.21, 0.31, 0.41, 0.51, 0.61}, based on the best training giving the smallest Overall_Performance, obtained MSE metrics for sets A, B, C and D are given in the table 3.1.

Table 3.1: Model training results at different white noise amplitudes are compared to the baseline. For each noise parameter corresponding MSE metrics for sets A, B, C, D are shown.

Noise Amplitude	MSE_A	MSE_B	MSE_C	MSE_D
Baseline	0.001565	0.040742	0.113454	0.080686
0.01	0.001904	0.037636	0.033084	0.062054
0.11	0.002187	0.055132	0.053950	0.068099
0.21	0.001810	0.052376	0.054861	0.075765
0.31	0.001577	0.055950	0.075206	0.090898
0.41	0.002161	0.070060	0.051307	0.077119
0.51	0.002728	0.068208	0.040365	0.085448
0.61	0.002427	0.076800	0.035568	0.093364

Now we can compare the model results trained on augmented data with our baseline. Let us represent the model results in a graphical way so that we can easily extract the features. Figure 3.1 shows that for set A, the model behavior fluctuates in an interval. For smaller values of the noise amplitude, the model output seems to have smaller MSE values, while for higher noise values, MSE is also increasing. Overall results show that model training with augmented sets does not improve with respect to the baseline. The model shows a more coherent behavior for the set B. The MSE is steadily rising for increasing amounts of noise, showing improvements in a neighborhood of 0.01 noise standard deviation. Set C is the only one fully improved by the training with augmented data. It has smaller MSE values compared to baseline over the whole noise range. Finally, set D expresses

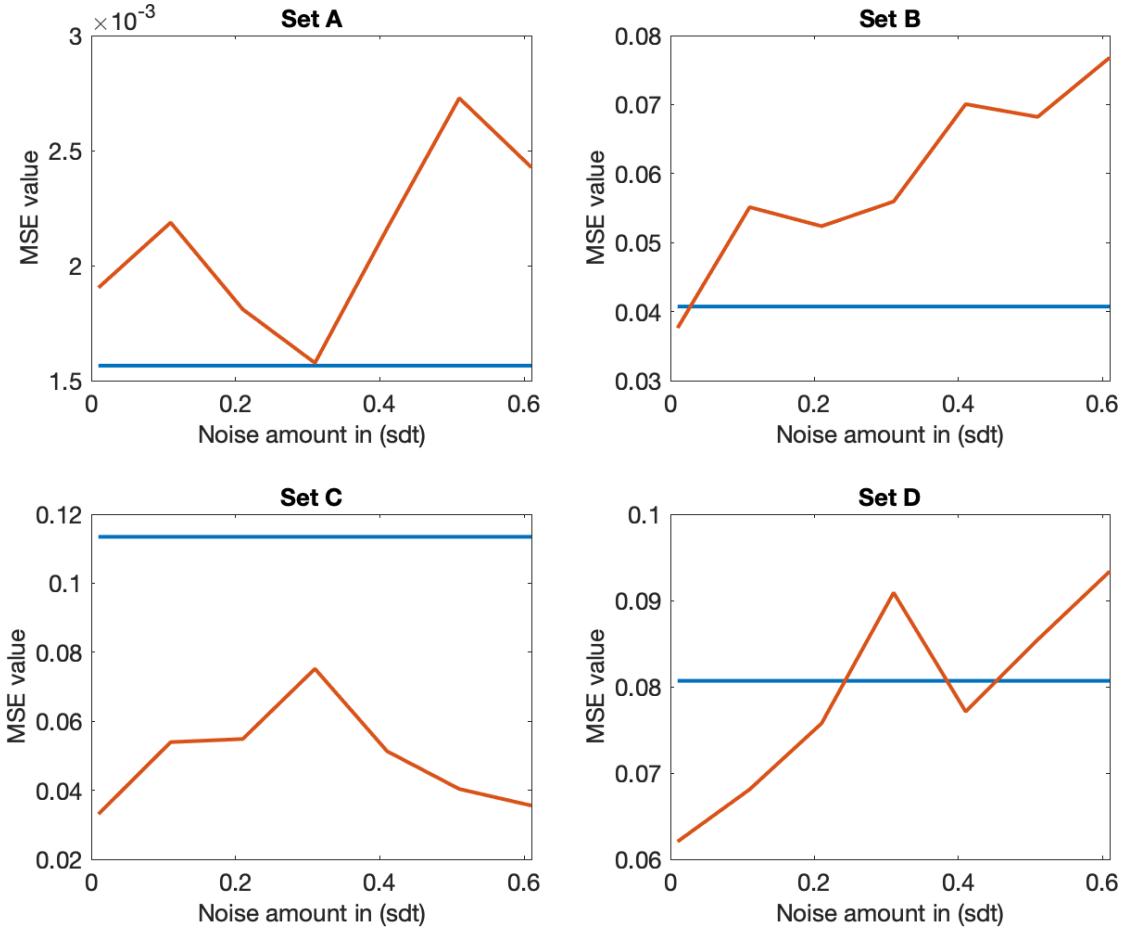

Figure 3.1: Model output MSE values(red line) obtained by augmentation with white noise amplitude range [0; 0.6], for data sets A, B, C and D. Blue line represents the model MSE baseline obtained without augmented data.

better model improvement up to noise standard deviation 0.5 except for the peak around a point 0.3 and increasing MSE for higher noise standard deviations.

Since we are interested in overall model generalization quality, we should merge all independent results of the four sets according to the formula. Summing the results of all sets can generate inappropriate decisions as described in the Figure 1.2. In the noise amplitude under the exploration, all sets show different characteristics, one showing very significant improvement while others without any improvements. When we sum MSE metrics, significant improvements of one or two sets could exceed losses in other sets, so it could be a misleading point to decide as a better model generalization. To avoid these kinds of "misleading" points after generating the overall sum of MSEs, we extract the areas with smaller MSE sum values with

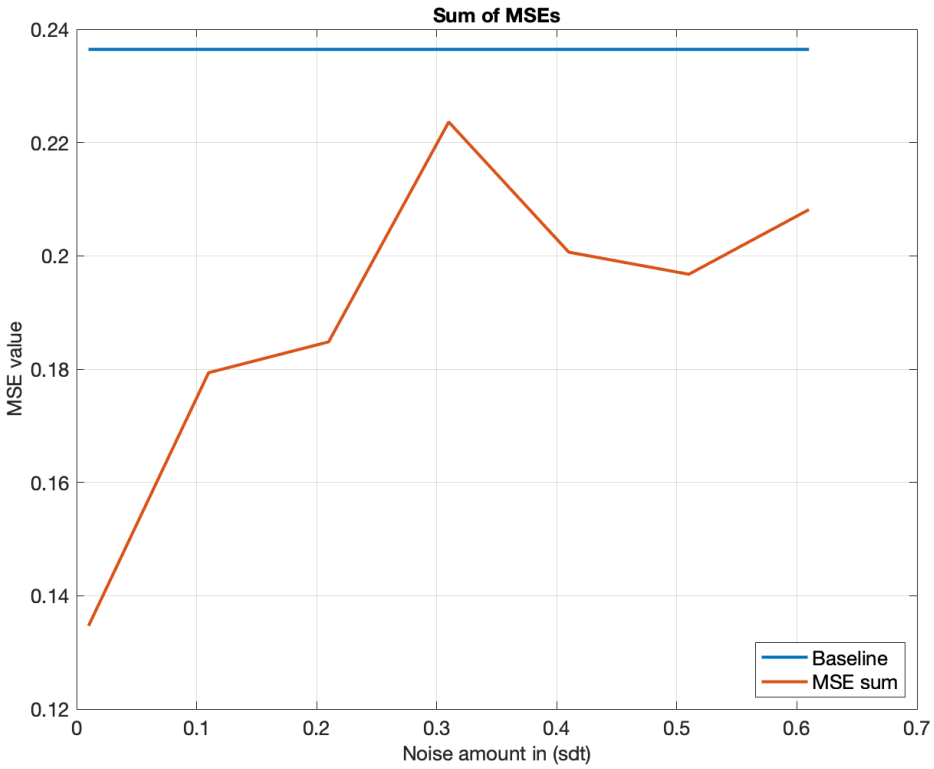

Figure 3.2: The model overall inference generalization obtained by white noise augmentation.

respect to the baseline. We should further inspect extracted areas for each set independently. If at least three sets have improvements in these intervals, we can decide that the model has improved generalization for all sets. Suppose less than three sets show more extensive improvements that surpass the unsatisfactory results of other sets. In that case, these noise intervals are not considered as a noise range with better generalization quality. Going back to our results in the figure 3.2, we have overall model results for all sets. It is showing improvement in a total interval of investigated noise standard deviation. However, it is not enough to decide on the improved generalization. We go back to independent plots of sets and search for the areas with more than two sets showing smaller MSEs than the baseline. It gives the range roughly around zero.

We perform another experiment in the same interval, but with a higher resolution. White noise standard deviations, which are directly proportional to white noise amplitudes, are chosen as follows: {0.01 0.06 0.11 0.16 0.21 0.26 0.31 0.36 0.41 0.46 0.51 0.56 0.61}. With experiment 1, we have completed one loop described in the flow in Figure 1.1. Changing the noise amplitude ranges corresponds to block Tune Noise Parameters. When we analyze the results for every 30 trainings best

model is chosen near to 200 epochs, which means that the model was still able to learn. According to the second experiment, model training with specified noise parameters (by performing 30 trainings for each noise value and choosing the best one) generates the following results given in Table 3.2. Since the resolution of the given interval is higher than the previous experiment, we can precisely extract the sections of the range with better model generalization. Let us visually analyze the model output with graphs to determine the most promising areas in Figure 3.3.

Mean square error loss function values for set A show smoother behavior in this experiment. MSE value is slightly higher than the baseline at the beginning of the noise range. Then it gradually decreases below the baseline, demonstrating model improvement until around 0.3. After that, it continuously increases with increasing noise amplitude. It is evident that more significant noise amounts does not help the training generate improvements for set A. Analyzing the plot of set B, we can say that set B is less resistant to noisy data showing slight improvement of the model for small amounts of noise approximately up to 0.05. After that point, the model MSE value starts to increase gradually as for set A. Set C has total improvement in

Table 3.2: Result of the model trainings at different white noise amplitudes. For each noise parameter corresponding MSE metrics for sets A, B, C, D and best training are shown.

Noise Amplitude	MSE_A	MSE_B	MSE_C	MSE_D
Baseline	0.001565	0.040742	0.113454	0.080686
0.01	0.0016	0.039653	0.049996	0.065936
0.06	0.001485	0.040832	0.041435	0.062892
0.11	0.001424	0.04568	0.037765	0.063757
0.16	0.001463	0.050361	0.043676	0.069003
0.21	0.001434	0.05065	0.05417	0.068153
0.26	0.001488	0.066244	0.060627	0.062341
0.31	0.001564	0.067676	0.066316	0.06029
0.36	0.001637	0.093435	0.070558	0.05868
0.41	0.001805	0.099847	0.057439	0.056119
0.46	0.001938	0.117797	0.050876	0.071523
0.51	0.002039	0.142877	0.072843	0.084473
0.56	0.002214	0.141222	0.076763	0.087068
0.61	0.002502	0.196399	0.072507	0.137806

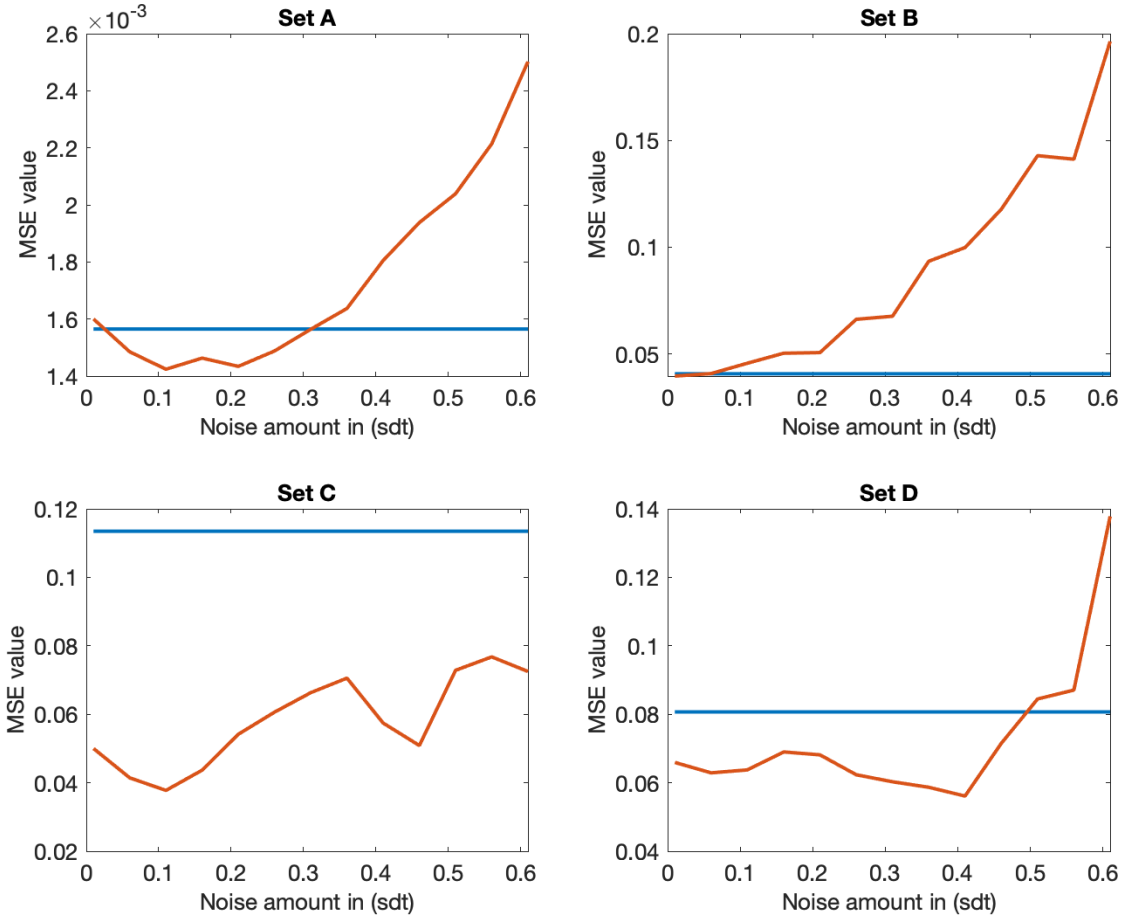

Figure 3.3: The second experiment. Model output MSE values(red line) with augmented data generated in white noise amplitude range [0; 0.6], for data sets A, B, C and D. Blue line represents the model MSE baseline obtained without augmented data.

this range, as in the previous experiment. Considering the graph of set D, we can say that we could generate more stable and understandable behavior with respect to the previous experiment. It shows model improvement in a noise range of [0; 0.5], which is the same as the initial results. After noise amplitude 0.5, the MSE starts to increase abruptly.

Now we can generate a plot to check model overall improvement as we decided according to (3.1). The sum of the MSE values for all sets is represented on the vertical axis in Figure 3.4, while on the horizontal axis, noise amplitudes are described. Overall model behavior shows a gradual MSE increase for increasing amounts of noise amplitude. Up to 0.45, MSE is below the baseline, which means the model has overall improvement. However, Figure 3.4 is not enough factor to

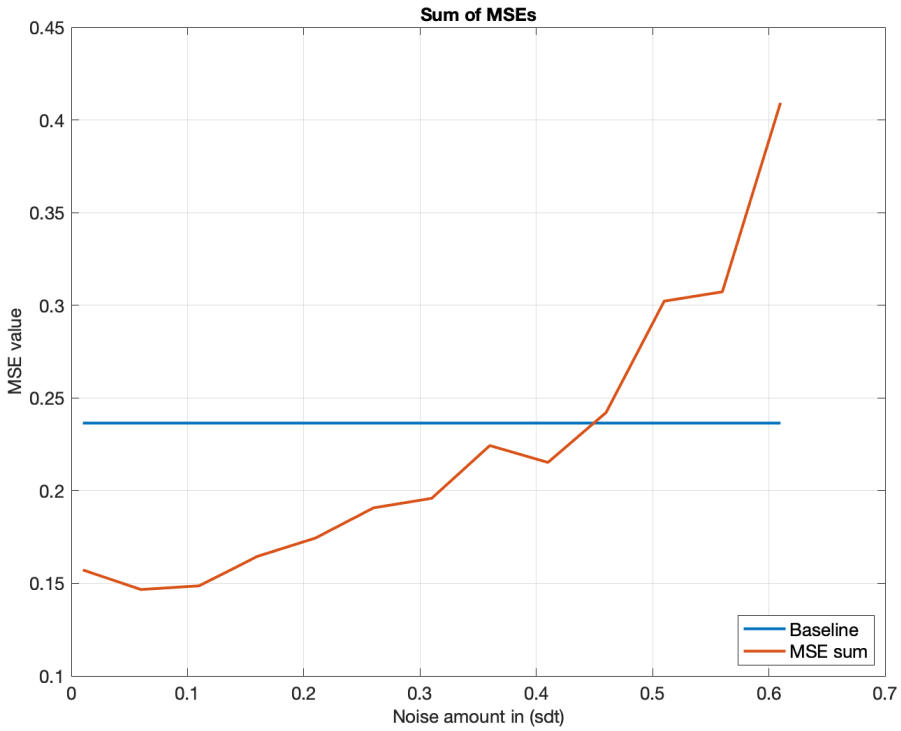

Figure 3.4: The model overall inference generalization obtained by white noise augmentation (the second experiment).

decide on model generalization quality because we should further check the plots of each set to avoid points in which improvements of one or two sets are big so that it surpasses the destructive results of other sets. At the upper end of the noise interval, the overall model characteristic shows higher MSE values than the baseline, which means the model has poor performance.

The model overall generalization quality is determined both by the improved MSE sum of all sets and individual improvements of at least three sets. Based on Figure 3.4, it is obvious that MSE sum is below the baseline in a noise range of [0; 0.45]. In this range sets C and D have total improvement. Fields of improvement for sets A and D are smaller than [0; 0.45]. Set A starts to go above the baseline after point 0.3, which means that we should reduce our initial range from [0; 0.45] to [0; 0.3] to have at least three sets with individual improvements. For set B, noise amounts in the range [0; 0.3] could result in MSE values above and below the baseline. However, we can accept this range as a reasonable interval giving better model generalization quality since at least sets A, C, D are performing adequately. The last experiment is accomplished to precisely determine the noise amplitude. All the sets have simultaneous improvement concerning the corresponding baselines and improved overall model generalization. We will not explicitly state the resulting

table and plots here. The noise amplitude in the interval [0; 0.046] gives the best model inference generalization quality.

3.3 Pink noise augmentation generalization

The model is characterized by adequate input data and architecture and trained with augmented data generated by white Gaussian noise. We will explore pink noise ranges that provide higher performance as a second method to generate augmented sets. First, we create a baseline from which we may track the generalization improvement. The baseline is obtained by training the model without augmented data. Our goal in creating augmented sets is to achieve better generalization of the model. The first intuition on noise parameter is related to the signal variance between a person detection and without detection, as indicated in previous sections. The signal deviation is derived using the maximum and minimum sensor one-pixel output variance values obtained from the Figure 2.10. All 16 pixels of the IR sensor are transformed into temperature inside the sensor. We look at how the model behaves when the noise amplitude is up to 50 % of the signal level because the power of the pink noise decreases with the frequency. Thus, we use noise amplitudes in range [0; 3]. Because training is a stochastic process, we select

Table 3.3: Result of the model trainings at different pink noise amplitudes. For each noise parameter corresponding MSE metrics for sets A, B, C, D and best training are shown.

Noise Amplitude	MSE_A	MSE_B	MSE_C	MSE_D
Baseline	0.001565	0.040742	0.113454	0.080686
0.01	0.001538	0.039735	0.046079	0.061516
0.34	0.001481	0.038605	0.04483	0.059618
0.67	0.001427	0.037736	0.048638	0.050147
1.00	0.001577	0.043275	0.042183	0.04185
1.33	0.00185	0.047579	0.044584	0.041498
1.66	0.002317	0.043262	0.033977	0.042545
1.99	0.002867	0.043579	0.032207	0.042899
2.32	0.003487	0.041271	0.02802	0.044242
2.65	0.004355	0.048448	0.020107	0.047483
2.98	0.004683	0.047419	0.027579	0.045834

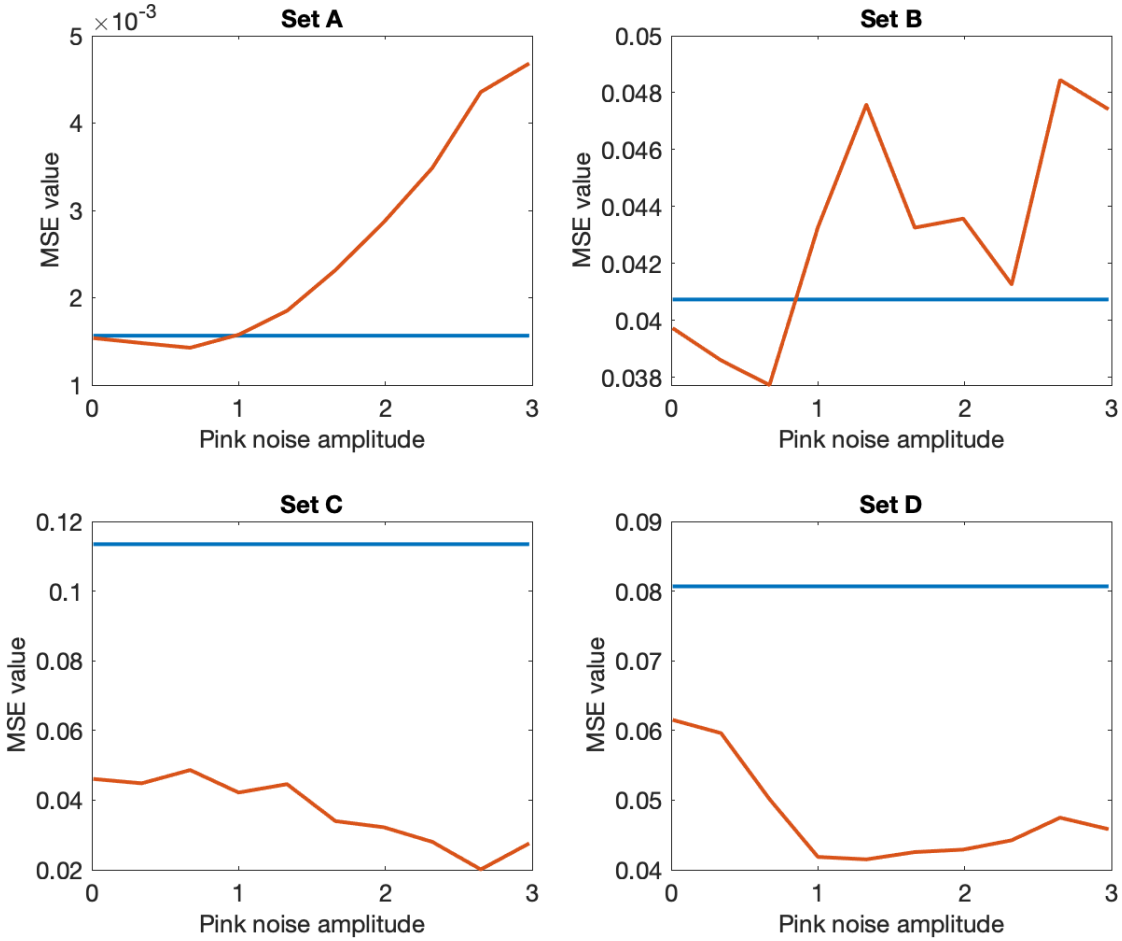

Figure 3.5: Model output MSE values(red line) with augmented data generated in pink noise amplitude range [0; 3], for data sets A, B, C and D. Blue line represents the model MSE baseline obtained without augmented data.

the best of 30 trainings using unchanged data and parameters. Since we have four sets of experimental data collected under different conditions with different environments and ambient temperatures, the same criteria are used to evaluate the model overall generalization quality calculated using (3.1). We analyze the model behavior trained with augmented sets generated based on pink noise amplitude in this range: {0.01 0.34 0.67 1.00 1.33 1.66 1.99 2.32 2.65 2.98}. For every noise amplitude in the range, the model is trained 30 times and based on the minimum MSE sum of all sets, the best model is chosen. Table 3.3 reports the best results of trainings, shown in Figure 3.5 and Figure 3.6. For set A, model behavior shows a monotonic increase as the noise amount increases. In an interval [0;1], MSE is below the baseline, which means the model can obtain better results with augmented

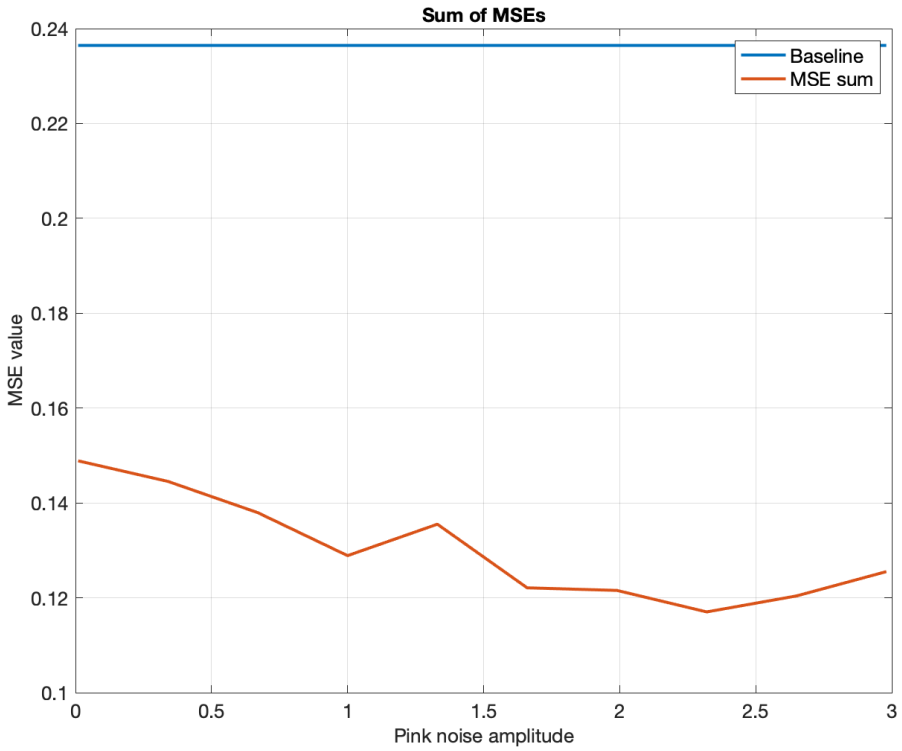

Figure 3.6: The model overall inference generalization obtained by pink noise augmentation.

data. Noise values greater than one seem to reduce the original data quality so that the model output is increasing very high above the baseline. Considering pink noise, set B has more improved characteristics than white noise. The model output is below the baseline until the noise amplitude one and starts to fluctuate with higher MSE values above the baseline. The plots of sets C and D have almost similar behavior. Both of them are below the corresponding baselines in full range, even showing better generalizations for higher noise amplitudes.

After analyzing all sets independently, overall model generalization quality is evaluated based on Figure 3.6. General behavior describes decreasing output which is always below the baseline showing total improvement of the model. We should exclude from this range the pink noise amplitude intervals in which one or two sets of improvement overcomes other sets MSEs that are above the baseline. Going back to Figure 3.5 it becomes clear that in a range [0;1], the model has the most improved area with all individual sets improvements. Only the set A results at the beginning of the noise interval could be a bit confusing. We can clarify it with a further experiment in which the range [0; 1] is explored with high resolution obtained by dividing the range into several intervals. Pink noise amplitudes greater than 1 show the model

improvement for sets C and D, which overcome the terrible results of sets A and B. However, it does not satisfy the stated criteria for good generalization. We performed one more experiment to check the model behavior for the pink noise amplitude range in [0; 1] to double-check our previous results. We will not expressly state the results and plots here, but the results prove that the interval [0; 1] is indeed a range giving the model the best generalization quality. After determining the reasonable intervals of white and pink noise amplitudes, we can further continue our trainings to explore the model characteristics with the combination of white and pink noise amplitudes.

3.4 Combined noise augmentation generalization

In general, all the systems are simultaneously affected by different kinds of noises. We have seen the model behavior that is affected by white and pink noises separately.

To better model real-world settings, we explore the effects of combinations of white and pink noises. We use the results of our previous training. We choose the interval [0; 0.2] for white noise ranges, which is proven as a maximum noise amount improving the model overall generalization. While for the pink noise possible range is proven as [0; 1]. We use white and pink noise amplitudes as two parameters describing the model MSE represented in 3D plots. Let us divide the chosen intervals of white and pink noises as follows: white noise amplitude ranges are selected as {0.001 0.025 0.050 0.075 0.100 0.125 0.150 0.175 0.200} and pink noise amplitudes are considered as {0.01 0.21 0.41 0.61 0.81 1.01 1.21 9.41 1.61}. The model is trained 30 times and out of 30 trainings, the best model which shows the best model generalization is chosen.

In Figure 3.7, plots of the model MSE for sets A and B are described. Overall, set A behavior replicates the model characteristics of white and pink noises explored individually. The model shows almost stable characteristics over the white noise amplitude axes. While the pink noise amplitudes significantly affect the overall results, the pink noise amplitude range of [0; 1] shows total improvement of the model results regardless of the white noise magnitude added on top of it. In the area close to the origin of the noise amplitudes model, MSE slightly increases to reach the baseline except for the point [0.001; 0.01], corresponding white and pink noise amplitudes. The point [0.001; 0.01] shows MSE that is slightly greater than the baseline. Pink noise amplitude values greater than one show results more remarkable than the baseline referring model worse generalization in this area. In the same Figure 3.7, we can analyze the model plots for set B. The model results for set B do not show smooth behavior. The results fluctuate both for white and pink noise amplitudes. In the diagonal direction close to the origin, there is a small area in which we can observe the model improvement with smaller MSE values than the baseline. It corresponds to the pink noise amplitude range of [0;1] and

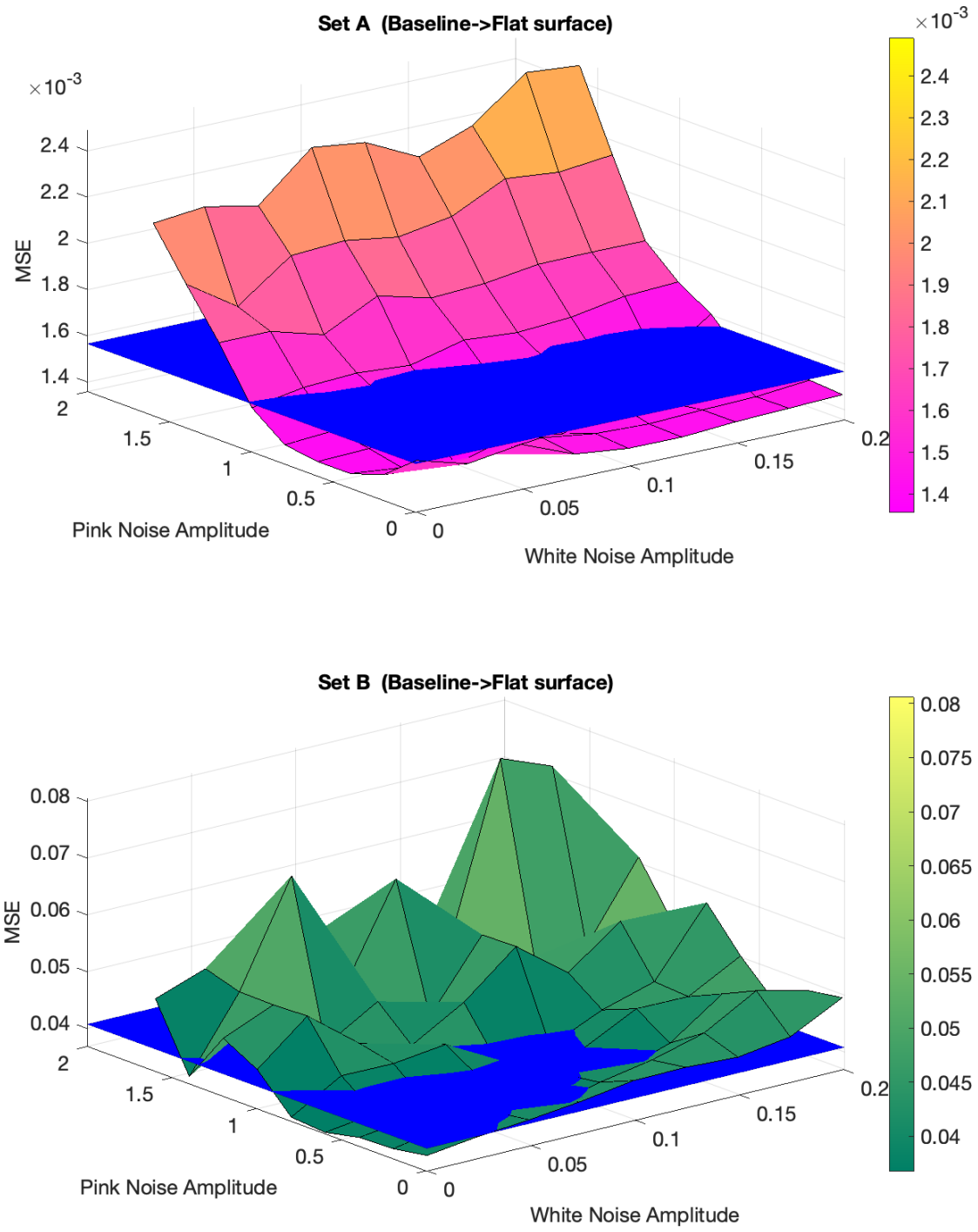

Figure 3.7: Model MSE value(obtained by the combination of white and pink noise amplitudes) compared to the baseline.

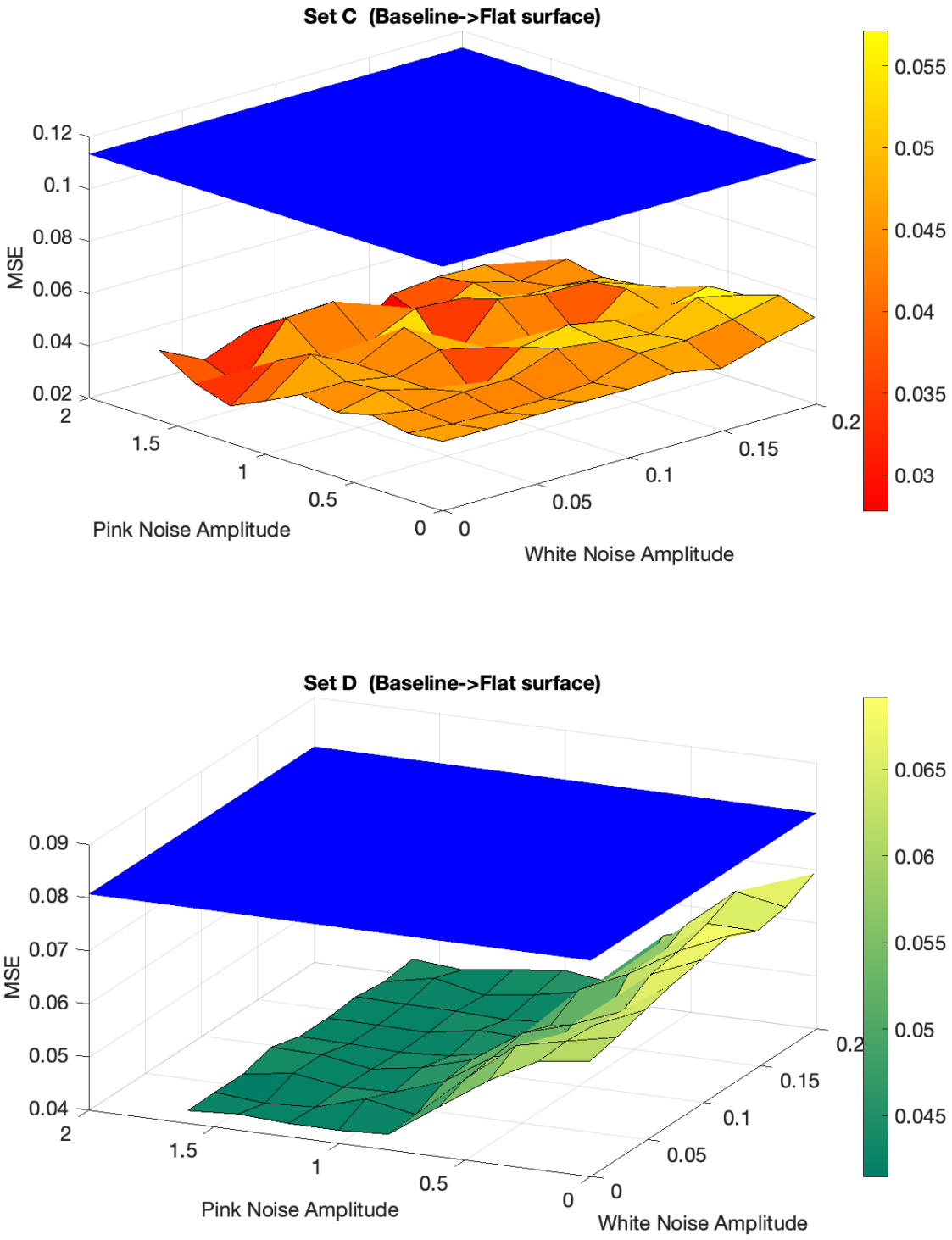

Figure 3.8: Model MSE value(obtained by the combination of white and pink noise amplitudes) compared to the baseline.

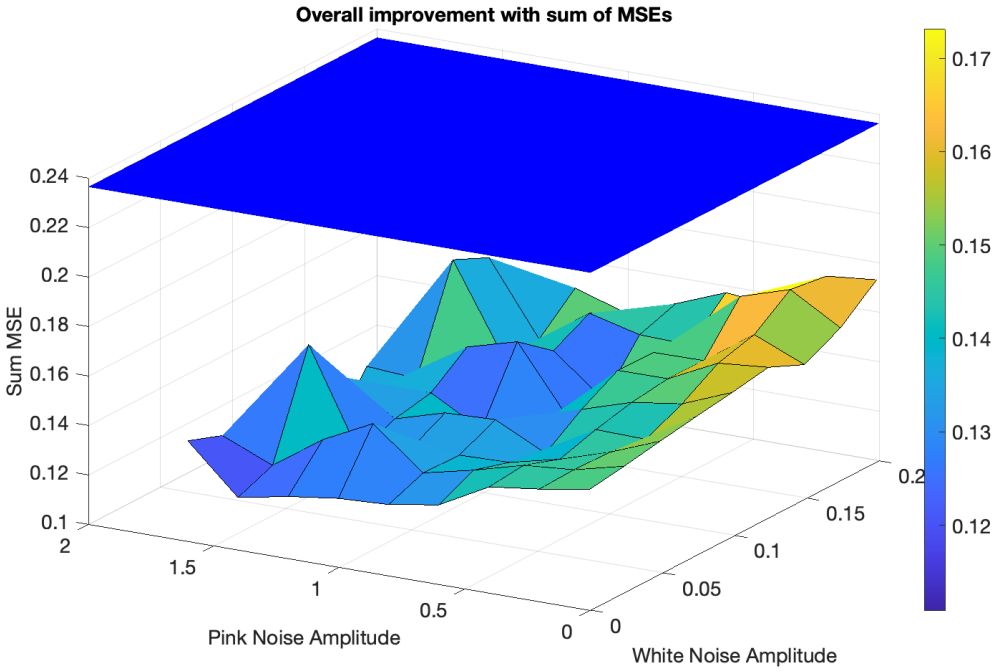

Figure 3.9: Comparison of the model overall generalization(obtained by the combination of white and pink noise amplitudes) with the baseline. Flat surface represents overall baseline.

white noise amplitude interval of about [0; 0.15]. There are three more points [0.001; 1.41], [0.05; 1.61] and [0.1; 1.61] which are below the baseline giving the model improvement. Only the point [0.001; 1.41] is visible in the Figure 3.7 while others are invisible due to the covering surfaces. From the plots, we can conclude for set B that all rising edges of the plot correspond to either bigger white noise amplitude or higher pink noise amplitude or the combination of the two.Regarding the plots of the sets C and D described in Figure 3.8, we can say that the model has improvement in the full range of white and pink noises for both sets. For set D, overall model behavior for white noise shows smooth characteristics, while for pink noise until the point [0.001; 1], the model MSE increases for decreasing amounts of the pink noise amplitude, trying to reach the baseline. Finally, let us analyze the model overall generalization ability by combining the individual results of all sets. Overall results are obtained as the sum of MSEs of four sets using (3.1) and the baseline is calculated as the sum of individual baselines

$$OverallBaseline = Baseline_{\mathrm{A}} + Baseline_{\mathrm{B}} + Baseline_{\mathrm{C}} + Baseline_{\mathrm{D}}. \qquad (3.2)$$

Figure 3.9 describes the model overall generalization ability considering all the

sets. The flat blue surface represents the sum of the baselines of individual sets, and the irregular surface is obtained as the sum of all independent sets MSEs. The rough surface has several local minimum points, which are considered points of best generalization because the smallest MSE below the baseline is the best improvement point. In Figure 3.9, all local minimum points are not visible due to overlapping other parts of the surface. We change the representation of the rough surface in another way to enable analyzing the promising areas. We calculate the difference between the baseline and the irregular surface using this formula:

$$\begin{aligned} NewSurface = Baseline_{\mathrm{A}} + Baseline_{\mathrm{B}} + Baseline_{\mathrm{C}} + Baseline_{\mathrm{D}} - \\ (MSE_{\mathrm{A}} + MSE_{\mathrm{B}} + MSE_{\mathrm{C}} + MSE_{\mathrm{D}})\,. \end{aligned} \tag{3.3}$$

The longest distance (or the most prominent difference) is the most promising point due to the smallest MSE sum. After calculating the difference for all the surface points, we generate another surface in which calculated deviations are placed on the original coordinate surface of the X and Y axes. As a result, we obtain

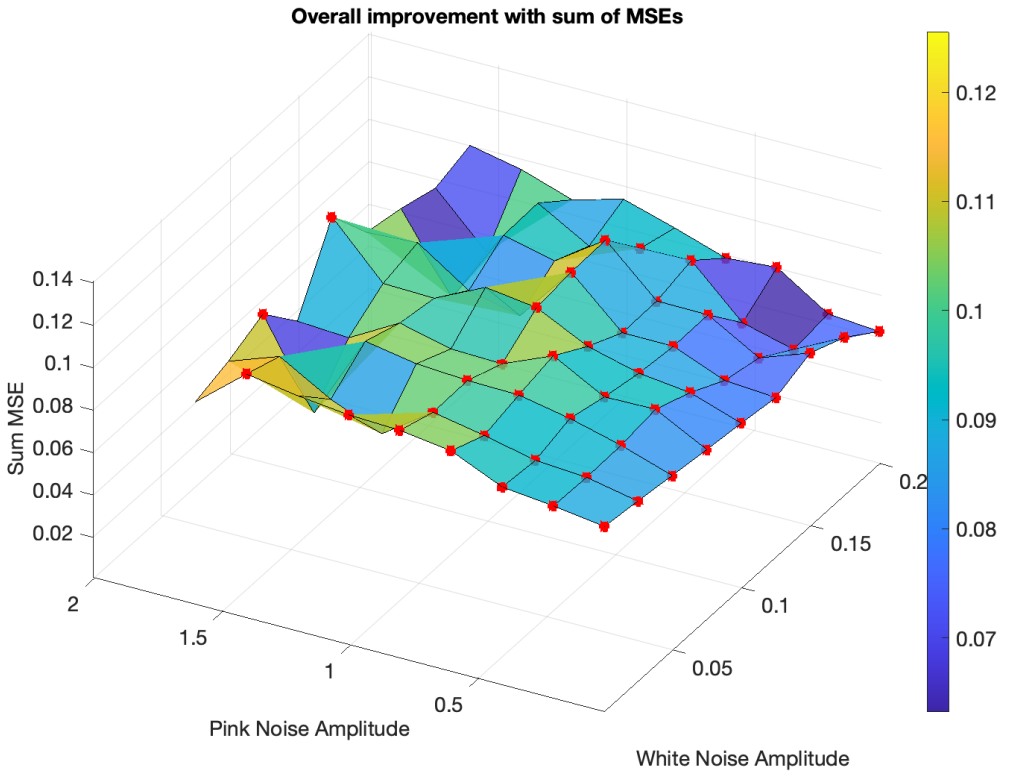

Figure 3.10: Inverted representation of the model overall MSE sum to analyze the areas with improved generalization.

another surface in Figure 3.10 where local maximum points refer to the model best generalization. The next step is to check each local maximum point representing better model generalization and exclude false local maxima in which one or two sets of improvements exceed the terrible results of the other sets. Referring to the criteria that we decided to analyze better generalization points, we check individual improvements of at least three sets giving overall model generalization. The surface is represented by a 2D matrix calculated with respect to two coordinates, X and Y, referring to corresponding white and pink noise amplitudes. Since each element of the surface matrix is obtained as the sum of each individual set MSE, we introduce a variable *check_matrix* with the same size as the MSE surface matrix to track the overall generalization. *Check_matrix* is initially initialized to zero. For each element of the MSE surface matrix, we count (with another variable *count_sets*) the independent improvements of sets. If the count of individual improvements is greater than or equal to three, then the corresponding element of *check_matrix* is converted to one. After performing the same operation for each element of the MSE surface matrix, we end up with *check_matrix* filled with either zero or ones. Zero means no improvement, while one means improvement points. In Figure 3.10, we can see the plots of the MSE surface matrix highlighted with star symbols, representing the points with improved generalization, obtained from a result examination stated below.

```
    check_Matrix = zeros(9,9);
for i = 1:9
    for j = 1:9
        count_sets = 0;
        if baseline_A-mse_A(i,j)>0
            count_sets = count_sets+1;
        end
        if baseline_B-mse_B(i,j)>0
            count_sets = count_sets+1;
        end
        if baseline_C-mse_C(i,j)>0
            count_sets = count_sets+1;
        end
        if baseline_D-mse_D(i,j)>0
            count_sets=count_sets+1;
        end
        if (count_sets>=3) && (baseline_A-mse_A(i,j)+baseline_B-
        mse_B(i,j)+baseline_C-mse_C(i,j)+baseline_D-mse_D(i,j))>0
            check_Matrix(i,j)=1;
        end
    end
end
```

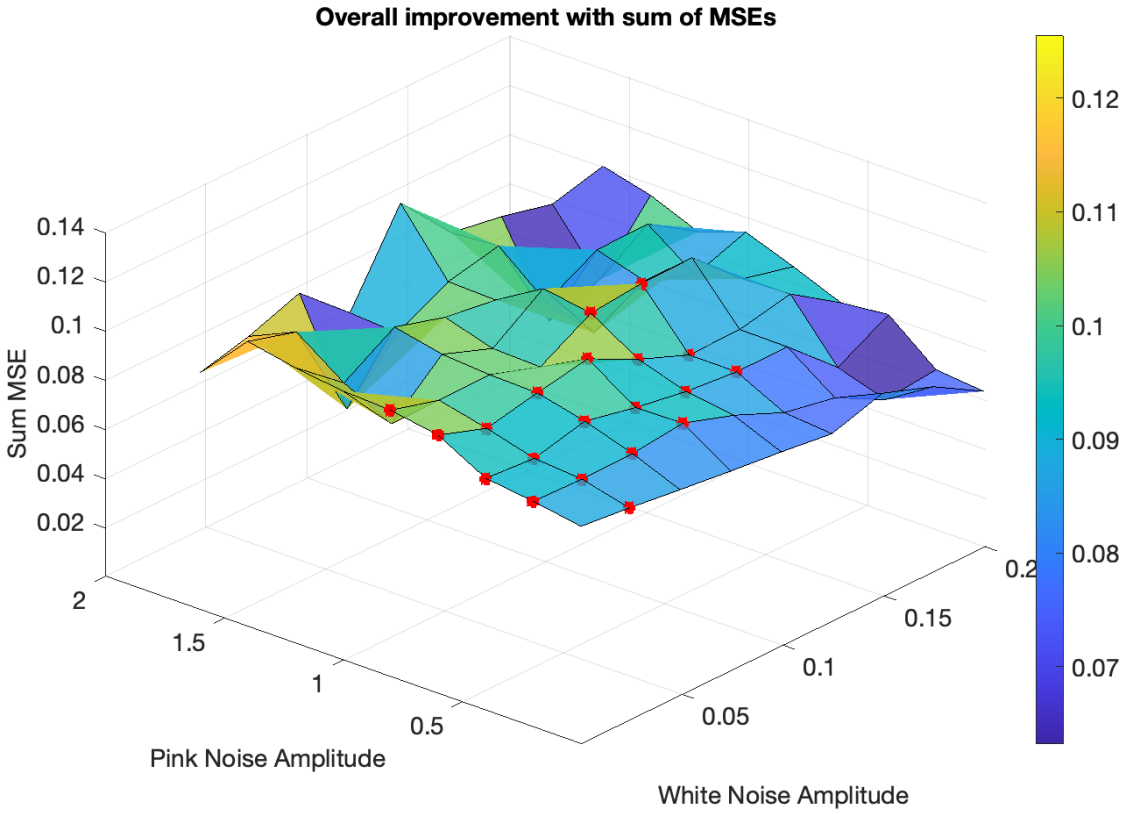

Figure 3.11: Comparison of the model overall generalization with the baseline.

In this work, all the plots are generated in Matlab, and the examination process of determining better generalization points is also done on the same platform. From the Figure 3.10, it is evident that the overall model generalization is highly affected by the pink noise amplitude ranges. The area covered with white noise amplitude range [0; 0.2] and pink noise amplitude range [0; 1] shows the model improvement by induced noise levels. For the pink noise ranges greater than one model shows rare improvement points for small amounts of white noise specifically at points [0.001; 1.01] , [0.001; 1.41], [0.05; 1.61] and [0.1; 1.61].

To further determine the areas of the surface in which all the sets show individual improvements, we set the condition $count\_sets = 4$) in the code and evaluate the most promising areas as represented in Figure 3.11. Finally, we can conclude that by considering the model overall generalization concerning the combination of white and pink noise amplitudes, the diagonal section of the surface considered in white noise intervals of [0;0.15] and pink noise of [0; 1], gives the model best improvement points.

Chapter 4

Conclusion

The application of machine learning techniques in many spheres is making ease the solution of challenging parts of the problem. Different machine learning algorithms are applied to many fields like healthcare, transportation, security, and industry. The neural networks, the set of algorithms used in machine learning problems, are considered one of the many used techniques. In most cases, NN generates satisfying results, whereas, in some cases, the model poor performance could be improved by applying augmentation or regularization techniques. This work has explored the data augmentation techniques to improve model performance by artificially increasing input data size. Augmentation techniques are prevalent in computer vision, and they are mainly applied to images. Random rotation, zoom, cropping, flipping, and brightness adjustment are the most widespread techniques. Since we are analyzing human indoor localization in a small 3x3m area using a 4x4 pixel IR sensor, the resolution of each frame generated by the sensor is relatively low. The corresponding person location with X and Y coordinates for each sensor frame is also collected. Person tracking is performed for 30 minutes at 5 Hz overall, generating experimental data of around 9000 tuples consisting of 16 IR sensor values and corresponding X and Y coordinates. We have collected four different sets of experimental data generated in different conditions with different ambient temperatures. The first set of experiments is used to train the network, and the rest three sets are used to evaluate the model inference. For improving the overall model inference, data augmentation by noise is employed because the sensor data could be affected by the environmental noise. The first type of noise, white noise, is a common type of noise that is everywhere around us, and the second one, the pink noise, is used to emulate the drifts in the system. We have explored different white and pink noise levels to obtain good results. The white noise amplitude ranges between [0; 0.6], consisting of 10 percent of the signal variation between person presence and person absence, are investigated to check the model inference generalization. In Figure 3.3, we can see that the model overall behavior is best in the range [0; 0.3]

showing improvements of at least three sets. For pink noise, reasonable amplitudes are higher than white noise because, for high amplitude, it gives small frequency, and for small noise amplitudes, higher frequency levels. Explored amplitude range is between [0; 3], giving the most promising noise range of [0;1],as described in Figure 3.5, improving the overall model generalization. Finally, the combination of two noises is investigated simultaneously, representing the model characteristics in 3D for white and pink noise amplitudes. From the Figure 3.10, the model improvement by induced noise levels is visible in the area covered by white noise amplitude range [0; 0.2] and pink noise amplitude range [0; 1]. The diagonal part of the diagnosed surface provides the model with the best improvement points.

Bibliography

[1] Andre Ye. *11 Essential Neural Network Architectures, Visualized & Explained*. https://medium.com/analytics-vidhya/11-essential-neural-network-architectures-visualized-explained-7fc7da3486d8. 2020 (cit. on pp. 6–10).

[2] IBM cloud education. *Recurrent Neural Networks*. https://www.ibm.com/cloud/learn/recurrent-neural-networks. 2020 (cit. on p. 6).

[3] Osama Bin Tariq, Mihai Teodor Lazarescu, and Luciano Lavagno. «Neural networks for indoor person tracking with infrared sensors». In: *IEEE Sensors Letters* 5.1 (2021), pp. 1–4 (cit. on p. 11).

[4] Jason Brownlee. *Dropout Regularization in Deep Learning Models With Keras*. https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/. 2016 (cit. on pp. 14, 16).

[5] Amar Budhiraja. *Dropout in (Deep) Machine learning*. https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5. 2020 (cit. on p. 15).

[6] Ibrahem Kandel and Mauro Castelli. «The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset». In: *ICT Express* 6.4 (2020), pp. 312–315. ISSN: 2405-9595. DOI: https://doi.org/10.1016/j.icte.2020.04.010. URL: https://www.sciencedirect.com/science/article/pii/S2405959519303455 (cit. on p. 18).