# Task Formulation

Create a review on the augmentation of image data associated with training neural  
networks, focusing on medical image data:  
- Introduce the concept of image data augmentation and its necessity in training neural networks.  
- Discuss the different types of image data augmentation techniques used in a general context.  
- Focus on challenges and considerations when augmenting medical images, such as preserving visual integrity and diagnostic quality of images.  
- Figure out which techniques are more suitable for medical image data.

Provide an overview of existing solutions for parallel tuning of hyperparameters in neural networks.

* Introduce Hyperparameter Optimization
* Examinate main approches
* Review modern platforms

Utilize processed insights to design augmented methods suitable for ultrasound image data.

Apply selected methods and create a mechanism for automatic and parallel verification of proposed experiments.

Compare the results of individual augmentation techniques and perform appropriate evaluation.

﻿﻿﻿

Prepare documentation according to the supervisor's instructions

# Analytical Chapter  
  
This chapter analyzes the basis of the thesis’s key points.  
  
First, we explain the idea behind the data augmentation technique and the importance of applying it when training neural networks.  
  
Secondly, identify peculiarities that we face by augmenting medical image data. provides an extensive overview of the existing approaches to augment image data.  
  
Finally, we evaluate which image data augmentation techniques are more appropriate for medical image data.  
  
  
  
## Data Augmentation  
  
Large amounts of data are necessary for training accurate, reliable neural networks, which is a well-known concept in Machine Learning [Cite AlexNet 2012].  
Since it is difficult and sometimes impossible due to the nature of the image to collect as many images as in ImageNet, we are forced to look for solutions with what we have.  
This is where the main idea of Data Augmentation comes from expanding the size of training samples by creating modified versions of available data.  
Despite having access to a large amount of data, one should pay attention to the importance of augmentation techniques. These methods can manipulate within image data space,  
creating alternative variants that account for possible invariances caused by shooting factors such as angle, distance [When To Warp]  
  
## Taxonomy of Data Augmentation in medical image data  
  
This section provides a taxonomy of data augmentation used in the medical field[image1].   
A diagram of a data processing process

Description automatically generated with medium confidence

Generally, we divide techniques into transforming existing data and creating artificial samples, as discussed in chapters 2.2 and 2.3, respectively.

2.2 Transformation of existing data

The first type of augmentation is based on direct changes to the input data, which allows the simulating of a wider range of scenarios that the model may encounter in real-world applications.

2.2.1 Affine transforms

This geometric transformation preserves lines and parallelism, but not necessarily distances and angles. It includes translation, reflection, scale, rotation, shear, and cropping transforms. All of them can be represented as matrices and hence can be combined to get composed transforms. This type is very common and is practically always a member of the data augmentation pipeline, due to ease of implementation and significant impact on model robustness.

2.2.2 Erasing transforms

In this kind of transform, we replace a random section of the Region of Interest(ROI) with random noise. It forces neural networks to avoid simplified detection patterns and originally helps to address issues with occlusions in RGB images – problems that are typically absent in medical images [Random Erasing.pdf]. This method is similar to Cutout, which uses a constant value, usually zero, instead of random noise. Additionally, this is a parameter-free technique, so there is no need to tune. Examples []

2.2.3 Elastic transforms

This type of augmentation involves modifying the original image through random displacement of pixels along a grid defined on the image, maintaining a smooth transition by chosen interpolation. Since they do not preserve collinearity and aspect ratio, like do-it affine transforms, they are able to create local shape variations. Based on it, it can simulate the breathing or equivalent processes inherent in medical images[cite <https://www.sciencedirect.com/science/article/pii/S0169260718315955>]. Examples []

2.2.4 Pixel-level

This type of transformation brings invariance to the pixel level of the image by changing pixel values. Since typically images in the medical domain are grayscale, changing in color space like RGB images is less meaningful. However, it could be helpful for distortions caused by retrieving images (image protocols, scanners, etc.). Common methods are blurring, color jitter, noise injection(Gaussian, Salt-and-pepper), histogram equalization, intensity normalization, sharpening, color shifting, contrast

Examples []

3. Parallel Hyperparameter Tuning

[https://wires.onlinelibrary.wiley.com/doi/epdf/10.1002/widm.1484]

Hyperparameter tuning is a fundamental aspect of machine learning that involves selecting a set of optimal hyperparameters for a learning algorithm. The optimal combination of hyperparameters extremely influences the model's behavior and performance, particularly in neural networks where configurations such as learning rate, count of hidden layers, and number of neurons in the hidden layers can сonsiderably affect model outcomes.

Manually attempting to identify optimal settings is time-consuming and inefficient. To address these challenges, parallel hyperparameter tuning was developed. The term “parallel” refers to the concurrent execution of multiple tuning processes. Instead of testing one combination of hyperparameters at a time, this approach distributes different sets across several computational units, such as GPUs, TPUs, and NPUs.

3.2 Main Approaches to Hyperparameter Tuning

A variety of approaches were designed to investigate the expansive hyperparameter space. In this section, we provide a brief overview of the most commonly utilized methods, their advantages, limitations, and working principles.

Grid Search: This is the simplest approach, which systematically examines the defined range of hypermeters. Its main advantages are simplicity and comprehensive exploration, which ensures that every combination listed will be evaluated. However, due to this exhaustive coverage, the number of combinations will grow exponentially with adding each new hyperparameter, making it very efficient in large spaces. Example[]

Random Search: This method selects a set of hyperparameters independently for each value based on a pre-defined distribution, usually uniform. It can sometimes find near-optimal configurations more quickly than Grid Search in higher-dimensional hyperparameter optimization settings[https://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf]. However, its unpredictability is also a disadvantage, as it might overlook significant combinations.

Bayesian Optimization: This technique is based on modeling the performance of configuration by constructing a probability model of the objective function and using it to select the most promising sets of hyperparameters to evaluate the true objective function. Gaussian process (GP) is commonly employed to model this objective function; it provides an estimate of the performance of hyperparameters and confidence in that estimate, which is essential for effective search space management. The evaluation is based on model outputs so poor assumptions may lead to suboptimal solutions[cite **Hyperparameter Optimization for Machine Learning Models Based on Bayesian Optimization**].

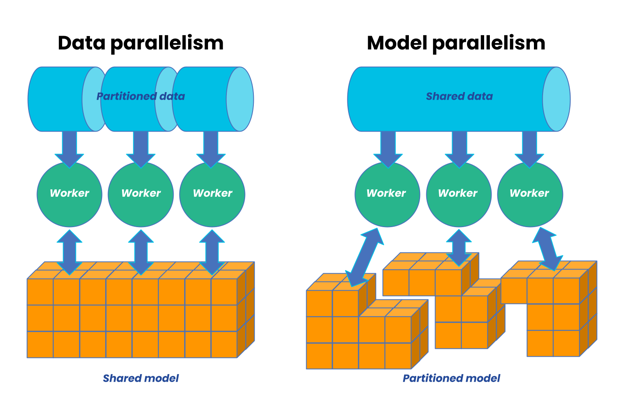
Genetic Algorithms: Inspired by natural selection, genetic algorithms treat each hyperparameter configuration as an individual in the population. They apply operations such as crossover and mutation to evolve the population to better performance. This approach excels in navigating complex hyperparameter spaces despite being highly dependent on the calibration of operational parameters.

Gradient-Based: These methods are extremely often and successfully used to optimize neural network parameters with millions of dimensions [Soydaner,2020).]. However, in general, no gradient information is available to respect hyperparameters, so this approach is not commonly used for hyperparameter optimizations[https://wires.onlinelibrary.wiley.com/doi/epdf/10.1002/widm.1484]. Despite this fact, [ Lorraine et al. (2020)] have shown the possibility of training neural networks based on differentiation along the path of the best response model parameters concerning the hyperparameters.

There are three common strategies to parallelize training:

* Data parallelism – Each computing unit holds an identical copy of the neural network model in this strategy with the same parameters. The data is partitioned, and each device processes a different subset of data. Training is accomplished by finding gradients on all devices and updating the global model.
* Model parallelism—This approach is typically applied to models with large parameter sizes [ cite https://arxiv.org/abs/1802.04924], i.e., when the model is too large to fit into the memory of a single GPU of a machine; it involves splitting the model itself across multiple computational resources. Unlike data parallelism, the data is not partitioned among the devices.
* Pipeline parallelism—Originally introduced by [https://arxiv.org/abs/1811.06965], it divides the model structure into several stages, each computed on different resources. The mini-batch of data is divided into micro-batches, which are processed at various stages. Example [] It aims to create an assembly line in a factory where each part of the product is made simultaneously but at different stages of the line.

A diagram of a parallelism

Description automatically generated 

3.3 Tools that help implement Parallel Hyperparameter tuning

In modern times, various tools and platforms are available for detailed and interactive analysis of hyperparameter tuning. They offer a complex infrastructure that allows for the easy identification of optimal hyperparameters through a simple interface, while also implementing optimization algorithms to enhance the efficiency and effectiveness of the tuning process. In this section, we compare some prominent tools.

Weights & Biases (wandb): It is a powerful tool exploited by a large number of researchers and engineers for machine learning experiments automatization, tracking, and visualizing. It supports parallel hyperparameter tuning by a technique named Sweep[https://docs.wandb.ai/guides/sweeps], which makes the process of comparing performances from different hyperparameter configurations intuitive and user-friendly for scientists.

Ray Tune: It is part of Ray framework, designed for distributed computing. This tool supports a wide range of optimization algorithms including Bayesian optimization and population-based methods. His advantage is the capability to scaling from single machine to large clusters, making it a suitable solution for heavy computational tasks. Not so straightforward to understand for a beginner researcher, compared with wandb.

Optuna: It is an open-source hyperparameter optimization framework, which unlike other approaches allows for the definition of more complex optimization logic and can handle a variety of trial schedules and pruning strategies. The advantage is extensions from Optuna available directly in the development environment (VsCode), making it possible to combine the development of machine learning solutions and visualizing the hyperparameter tuning in one application.

3.3.1 Sweeps

For this thesis, Weights & Biases (WandB) has been selected as the platform of choice for tracking and implementing parallel hyperparameter tuning. WandB is known for its simplified approach to automating hyperparameter searches and its ability to provide rich and interactive data visualization. It supports three widely used search methods: Bayesian optimization, grid search, and random search.

In WandB, a “sweep’ refers to a series of experiments designed to explore defined hyperparameter configurations. The sweeps are managed by Sweep Controller, hosted on the platform’s cloud infrastructure. This controller generates a set of instructions taken from the configuration provided by the user, which includes a method for navigating the search space, metric to optimize, and the defined ranges of hyperparameters. These instructions are then picked up by agents, who are tasked with executing the runs. Agents have the capability to pause, resume, stop, and cancel each run as needed.

4. Network

This thesis investigates the impact of data augmentation methods on ultrasound image classification, which requires the selection of an appropriate neural network architecture.

Among the various award-winning architectures – VGGNet[], GoogLeNet[], LeNet-5[], and ResNet[]—the latter, known as Residual Network, was selected. Developed by researchers at Microsoft, ResNet gained prominence after taking first place at the ILSVRC 2015 classification task[link].

This architecture is notable for the introduction of residual connections, an innovation designed to address the problem of degradation that occurs with increasing network depth.

Residual Blocks in ResNet

The fundamental component of ResNet is the residual block, which is designed to learn specific transformations on the input data. To understand the mechanics, consider a residual block with two layers visualized on the image [image].

The first layer applies a transformation F1 to the input x, which may include operations such as convolution, batch normalization, and ReLU as activation function. As a result, the first layer will output intermediate output u=ReLU(F1(x)).

Subsequently, the second layer performs another transformation F2 on the output u from the first layer, resulting in v = F2(u). However, since it is a residual block, instead of treating v as the final output, we learn a residual; that is, the output of block is F(x) = v + x, where x is the input that bypasses the two transformational layers via a shortcut connection. At the end, we apply ReLU, so the final output of the residual block is y = ReLU(F(x)).

Residual Learning

The core principle behind ResNet is that layers within residual block do not need to learn the full transformation from x to y. Instead, they focus on learning the residual R(x) = F(x) – x. This approach alleviates the vanishing gradient problem – a common challenge in training deep neural networks where gradients diminish in magnitude through the layers – by allowing a direct flow of gradients through short connections.

Moreover, even if deeper layers start learning slowly, the feature x continues to spread across the network, supporting the development of deeper network architectures.