Distilling Neural Networks for Real-Time Drone Control

Anna Schua

Master's thesis

Distilling Neural Networks for Real-Time Drone Control

Anna Schua

October 18, 2021

Chair of Data Processing
Technische Universität München

Anna Schua. *Distilling Neural Networks for Real-Time Drone Control.* Master's thesis, Technische Universität München, Munich, Germany, 2021.

Supervised by Prof. Dr.-Ing. Klaus Diepold ; submitted on October 18, 2021 to the Department of Electrical and Computer Engineering of the Technische Universität München.

Chair of Data Processing, Technische Universität München, 80290 München, Germany, `http://www.ldv.ei.tum.de/`.

Abstract

Knowledge Distillation (KD) is a way to compress neural networks. In most cases a fully trained, large neural network is available that is used to train a smaller model. Several factors have an influence on the success of KD. For example, the training data used and the architecture of the models can have an impact.

The problem studied in this thesis is the transfer of a neural network to a drone which has limited memory. Therefore, the utilization of the large network on the drone is not directly possible and KD is employed. This work specifically examines the effects of KD, using a model with many parameters that has been trained for a specific flight maneuver with the help of Reinforcement Learning (RL). The knowledge of this model is transferred to a smaller one, which will then be utilized on the drone and perform the flight maneuver in real time and in the real world.

An experiment, examining different data generation methods, shows that the training data used has a large impact on the distillation result. Other tests use one or more additional models to distill the knowledge. These additional medium models have a size that is between the large and small model. In the experiments that use a medium model, the medium network is trained first using the large network and afterwards it is used to train the small model. The tests reveal that additional training steps can further improve the results. The experiments show that with the right training data and with one or more distillation steps very good small models can be trained that outperform the large network originally trained with RL.

Contents

List of Abbreviations

DNN	Deep Neural Network
EM	Expectation Maximum
FSP	Flow of Solution Procedure
INQ	Incremental Network Quantization
KD	Knowledge Distillation
NBE	Naive Bayes Estimation
RL	Reinforcement Learning
TA	Teacher Assistant

List of Figures

List of Algorithms

1 Introduction

In recent years, Deep Neural Networks (DNNs) have become increasingly popular due to their improved accuracy and their ability to solve difficult problems. They are the tool of choice for complex tasks such as natural language processing ([11], [44]), image classification ([10], [32]) and many other applications ([13], [14], [25], [39]). A DNN consists of several layers with hidden nodes that are connected with each other. Due to this structure, a DNN has lots of parameters and therefore needs a large storage space [45]. Furthermore, calculating the outputs of the neural network can require a powerful processor. These facts suggest that accuracy is not sufficient as a sole requirement for training DNNs.
A research area that is currently on the rise is embedded systems. Today, many devices contain microprocessors that are often powered by a battery or accumulator. Therefore, an important requirement for these devices is low power consumption. This often results in low processor performance to reduce power consumption and another requirement that may be essential in some applications is real-time capability. Furthermore, some embedded systems are constrained due to their little memory.
To combine these two developments, a solution must be found that allows DNNs to run on embedded systems ([1], [7], [17], [45]). This seems to be contradictory, since a DNN has millions or even billions of parameters and thus requires a lot of memory. Furthermore, the calculation of the outputs can be computationally intensive. Both are problematic for an application on an embedded system and result in more stringent requirements for the complexity of the neural networks.

This thesis focuses on a method, named Knowledge Distillation (KD), to combine the prerequisites of embedded systems and DNNs [18]. Specifically, it involves a small drone that is to perform a flight maneuver with the help of a neural network. The quadrotor consists of an embedded system and has a low power microprocessor, little memory and is battery powered. A small neural network with only a few hidden layers is necessary, in order to utilize it on the drone and to calculate the outputs in real time. However, the original model that is trained to perform the maneuver using Reinforcement Learning (RL) is a DNN and cannot be transferred directly to the quadrotor. This thesis investigates how KD can be used to train a small neural network that achieves a similar reward as the trained original model. Particular attention will be paid to the training data that is used for generating and transferring the knowledge from the original network to the smaller network. For the generation of the training data different methods are evaluated. The achieved performance of the trained small models is compared to the original model performance, to determine the best approach. Subsequently, the superior approach is selected and it is checked, whether the performance of the model can be further improved with the help of a method called Teacher Assistant Knowledge Distillation [28].

One of the research questions this thesis attempts to answer is whether the results of KD can be improved by modifying training data. The goal is to find a method that is able to generate training data that trains a small model which achieves a similar reward as the existing larger neural network. Another research question to be answered by this work is whether a Teacher Assistant (TA), a model with a neuron number between the original and the small model, positively influences the training of the smaller neural network. This work investigates a good architecture of the TA model that is used by Mirzadeh et al. to enhance the performance of the small model [28].

2 Related Work

There are several attempts to combine the requirements of an embedded system and a neural network, as described in the introduction. Some of the developed methods aim at compressing the trained neural network [7]. One approach investigated by researchers is parameter pruning and quantization [9]. This method is used to reduce the required memory space of the neural network by reducing the size of the weight matrices. The individual weights are then represented with fewer bits and in case of binarization even with only one bit. Zhou et al. showed that their method called Incremental Network Quantization (INQ) is able to compress AlexNet without accuracy loss when using 5-bit and 4-bit quantization [45]. AlexNet, introduced by Krizhevsky et al., is a DNN trained to classify the ImageNet LSVRC-2010 data set with 1,000 different classes [21]. When using INQ with a 5-bit or 4-bit quantization, the required memory space to store the weight matrices of AlexNet is reduced compared to a representation of weights with 32-bit [45].
Network pruning is another approach that reduces the memory usage and the network size [7]. Usually in neural networks all nodes of one layer are connected with all nodes of the next hidden layer. Network pruning deletes some of these connections. This reduces the number of weights in the network. One technique, called Optimal Brain Damage, searches for connections to delete or to set to zero that cause a minimum increase of the objective function [23]. After deleting this parameters the network is retrained and the algorithm searches again for weights to remove. With the help of the loop it is possible that the accuracy of the neural network remains the same even if up to 60% of the parameters are deleted or set to zero. Furthermore, low-rank factorization is a possible choice that aims at reducing the size of the network and thus using less memory space [9]. Low-rank factorization tries to generate several smaller weight matrices factorizing a large weight matrix. A often used method for factorization is singular value decomposition. However, calculating the decomposition uses lots of computational resources [7].
The following section describes a method named Knowledge Distillation (KD) that can be used in order to transfer knowledge from a large model to a model with smaller network size [18]. This is another form of model compression.

2.1 Knowledge Distillation

During the training of a model there are usually no constraints concerning memory space or the computational complexity. Often the main aim is to achieve a high accuracy. The before mentioned requirements, like little memory usage and real-time capability, are important when we utilize neural networks for example on mobile devices or embedded systems [45].
Bucila et al. proposed a method called model compression [6]. This method is a predecessor of KD. A small and fast model, called student, is trained to approximate the function learned by a larger model, named teacher, trained with high accuracy. The large network and its learned function can provide labels for a huge amount of training data. The smaller model is then trained with the created samples. It is important to note that the network with fewer layers and connections cannot overfit on the original problem as long as the large model is not overfitting and sufficient training data is used [6]. The student learns to approximate the function generated by the teacher well.

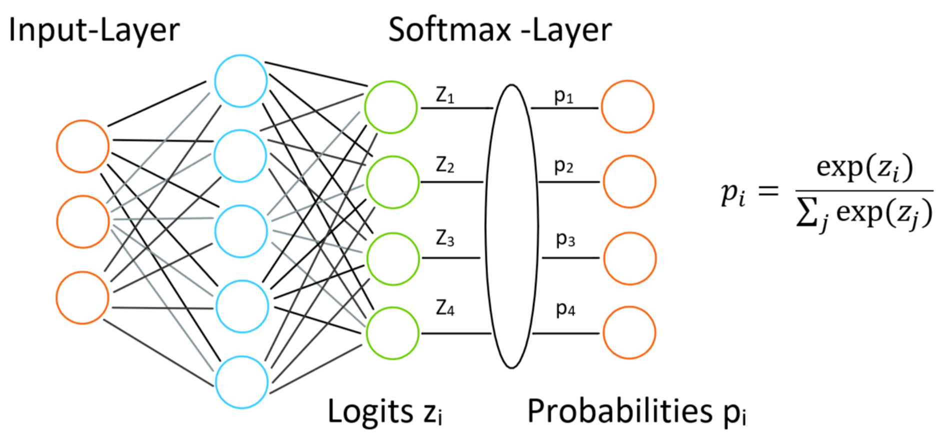

Figure 2.1: Logits are inputs into the softmax-layer and probabilities the corresponding outputs.

Model compression uses the logits produced by the teacher model as targets for training the student model ([6], [18]). Logits are the inputs into the last softmax layer as visualized in figure 2.1. They are used instead of the probabilities that are the outputs of this softmax layer. The main reason not to use the probabilities, is that logits provide larger numbers and therefore richer information that can be used as knowledge to train the smaller model [4]. For example, when the logits are $[90, 40, 60]$ then the corresponding probabilities are $[0.9999, 4.2e-18, 2.1e-9]$. This means that the last two targets may be ignored and the focus is solely on the first target. As a result, probabilities provide less information that the student can learn and mimic than logits. The used objective function is calculated as follows:

$$L(W,\beta) = \frac{1}{2T} \sum_i \|\beta \mathbf{f}(\mathbf{W}\mathbf{x}^{(\mathbf{i})}) - \mathbf{z}^{(\mathbf{i})}\|_2^2 \tag{2.1}$$

where $f(.)$ is the activation function which is applied element by element, W is a weight matrix between input and hidden layer and β is the weight matrix between hidden layer and output layer [4]. The parameter $x^{(i)}$ is the input of the ith training sample and $z^{(i)}$ are the corresponding logits. In order to optimize the performance of the student model, it is best to train with new data that was not used for training the teacher model [4]. Ba et al. stated that it is possible that the teacher model has overfitted to the original problem at those training points [4].

The term Knowledge Distillation (KD) was introduced by Hinton et al. and it describes the transfer of knowledge from a large teacher model to a small student model [18]. Figure 2.2 illustrates this method. Knowledge is described as a function that maps the inputs of a system to their respective outputs. In the meantime several papers deal with KD and its applications, some studying for example, regression problems [38] or KD in the context of RL [2] but the focus is mainly on classification problems ([6], [18], [28]). Given certain inputs the network is able to select one specific class as the output, based on the maximum probability. KD benefits from these assigned likelihoods [18]. The large teacher network calculates not only the probability of the final output but all probabilities of each possible output called "soft targets". These likelihoods contain information about the generalization of the model and can be used to train the small student model. "Hard targets", on the other hand, only provide information about the correct output but no details about which outputs can be easily mistaken. When we use "soft targets" instead of "hard targets", the number of training samples can often be reduced.

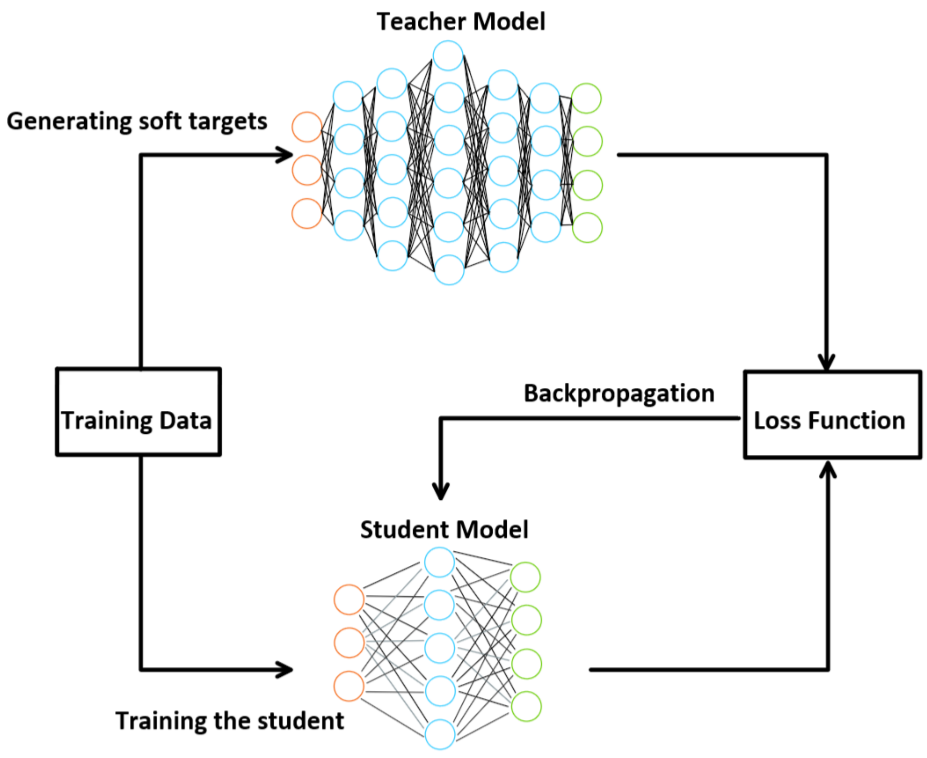

Figure 2.2: Illustration of the process of KD.

The proposed model compression algorithm by Bucila et al. uses logits as "soft targets" [6]. KD, on the other hand, calculates probabilities based on logits [18]. These probabilities are scaled using a temperature T. The goal is that they are not too small, so that they still provide information about the generalization. Hinton et al. showed that using the original training set works well when the loss function is adjusted in a way that the model not only learns to predict the final label, but also learns the generalization provided by the teacher model [18]. The used softmax function calculates the probability p_i given the logit z_i compared with other logits. In contrast to the softmax calculation in figure 2.1, they added a temperature to their softmax function. The "soft targets" are calculated as follows:

$$p_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)} \tag{2.2}$$

where T describes the temperature and can be set to a high value to produce softer targets. The student network is trained with the same temperature T that was used while generating training samples with the teacher model. After training T is set to 1. The training can be even further improved when the correct labels are known and a combination of two objective functions is used ([18], [28]). One function calculates the Kullback-Leibler divergence with the "soft targets" and a high temperature T [28]. The same temperature is utilized that was used to generate the "soft targets". This loss function aims at matching the "soft targets" of the teacher p_t and student p_s:

$$L_{KD} = T^2 KL(p_s, p_t) \tag{2.3}$$

where KL corresponds to the Kullback-Leibler divergence loss. T^2 is needed to rescale the magnitude of the gradients [18]. The second function calculates a cross entropy with the outputs of the student, T set to 1 and the real labels ([18], [28]). This cross-entropy loss function is used in supervised learning to ensure, that the output of the student network and the corresponding label are the same and to penalize mismatches [28].

$$L_{SL} = H(p_s, l) \tag{2.4}$$

In formula 2.4 p_s represents the calculated output of the student network but with the temperature T set to 1, l corresponds to the ground truth label and H represents the cross entropy function. The combination is composed of the weighted average of both functions [18]. In order to achieve good results it is best to use a much smaller weight on the second loss function L_{SL} [28].

$$L_{student} = (1 - \lambda)L_{SL} + \lambda L_{KD} \tag{2.5}$$

Logits that are a lot smaller than the other logits are ignored when a lower temperature is used [18]. This can be beneficial in case the student model is too small to capture the complexity of the teacher network.

2.1.1 Knowledge

There are different kinds of knowledge that can be transferred from the teacher to the student [17]. The knowledge can either be the output of the teacher model or the knowledge can be transferred from the hidden layers and used for training the student model.

Response-based knowledge is the most basic kind of knowledge that is used in a lot of distillation scenarios [17]. Model compression and KD use knowledge provided by the last layer to train a student model ([6], [18]). Response-based knowledge uses only the output of the last layer as knowledge that is transferred from the teacher to the student. This output, calculated in formula 2.2, is referred to as "soft targets" [18]. A typical structure of a response-based knowledge distillation can be seen in figure 2.3.

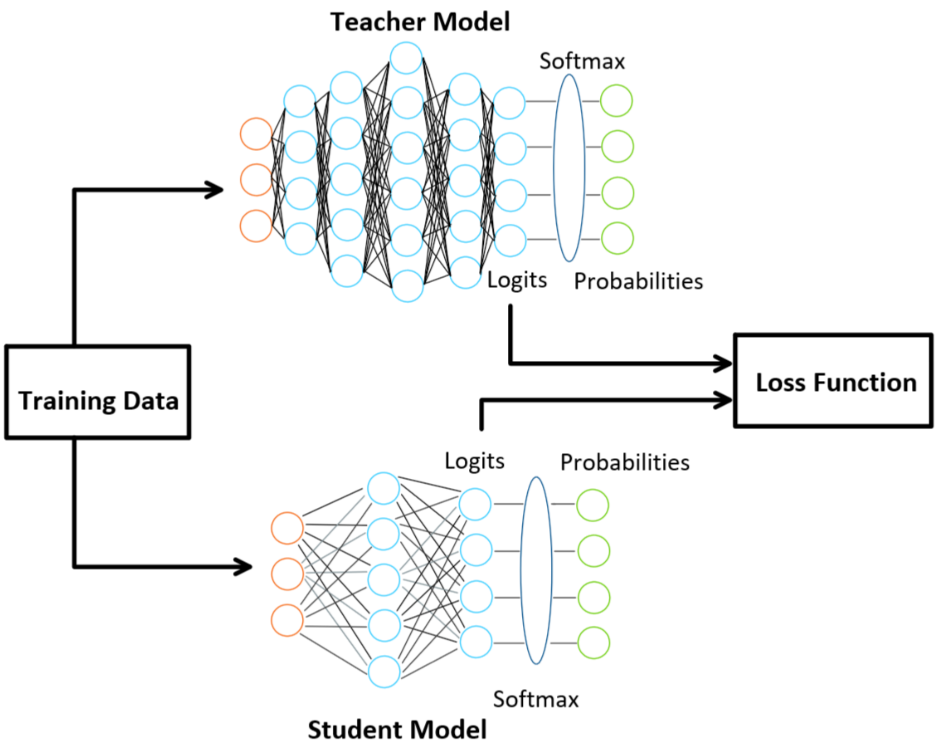

Figure 2.3: Training process using response-based knowledge redrawn after Gou et al. [17]. The knowledge is extracted in the last layer and used for calculating the loss function and training the student.

Feature-based knowledge describes the knowledge learned in the hidden layers [17]. It can be used as additional information to the response-based knowledge that can be transferred to the student. Training the student with feature-based knowledge aims not at matching the outputs but the feature activations. Romero et al. used a feature-based approach to train student models called FitNets that are thinner and deeper than their teachers [33]. They introduced so called hints that are used to help training the student. A hint is the output extracted of one of the teacher's hidden layers, the so called hint layer. In the student network a layer is chosen to be guided by the hints of the teacher. This guided layer is trained to predict the teacher's hint layer. In their approach the teacher network is wider than the student and this is why they introduced a regressor to match the size of the guided layer and the hint layer. The training is structured in different stages. At first all layers up to the guided layer, including the regressor parameter, are trained using a special introduced loss function. Afterwards, the whole FitNet is trained using the normal KD procedure. The feature-based knowledge transfer is visualized in figure 2.4.

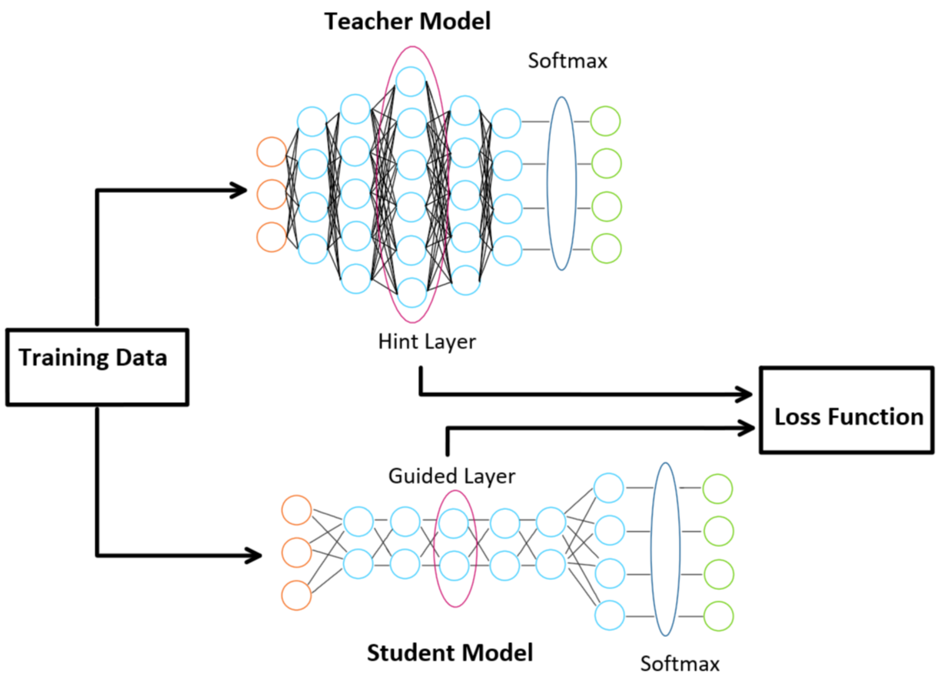

Figure 2.4: Training process using feature-based knowledge redrawn after Gou et al. [17]. The guided layer of the student is trained to match the hint layer of the teacher.

Relation-based knowledge is the third type of knowledge that is mentioned by Gou et al. [17]. The name implies that it uses the relationship between the hidden layers or the data samples for training the student model.
Park et al. introduced a method called Relational Knowledge Distillation where they use mutual relations of data samples as knowledge that is transferred from the teacher to the student [31]. They compute a potential between n data samples and use this knowledge to train the student. One function called distance-wise potential calculates the Euclidean distance between two data samples using the output representation. Distillation then aims at matching the calculated distance-wise potentials between teacher and student. A loss function is presented that promotes the same distance between data samples in their output representation spaces of teacher and student. Another introduced potential function is called angle-wise potential. This potential calculates the angles of three data samples in their output representation space. They further introduce a loss function to promote the same angles in the output representation of teacher and student. In figure 2.5 a relation-based knowledge approach can be seen that makes use of instance relations introduced by Park et al. [31].

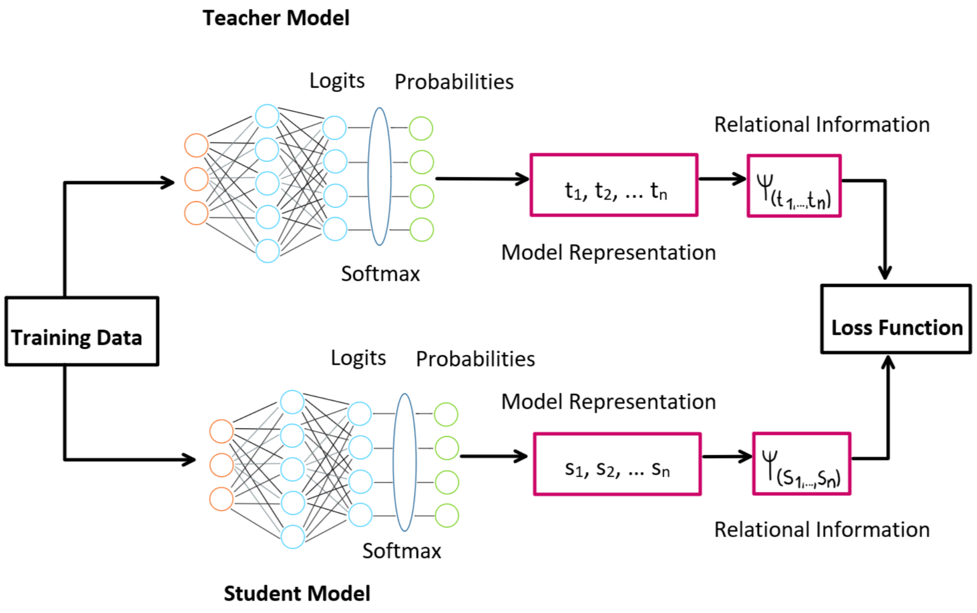

Figure 2.5: Relation-based knowledge redrawn after Gou et al. [17]. Relational Knowledge Distillation calculates relational information using a potential function $\psi(.)$ and uses this information to train the student.

Another approach introduced by Yim et al. uses the relation between different feature maps [43]. The student does not need to know the intermediate teacher results of a specific input sample, it is more important that the student knows the steps necessary to solve a specific task. In this approach, the student tries to mimic the generated features of the teacher, whereas Romero et al. tries to approximate the intermediate results [33]. The features provided by the teacher are the solution process that the student tries to mimic. This solution process for a specific type of input is used as knowledge that is distilled form teacher to student. It is defined as relationship between two intermediate results and is stored in a so called Flow of Solution Procedure (FSP) matrix. The FSP matrix of teacher and student with same spatial size is used to calculate the loss. This loss is then used to minimize the difference between the FSP matrices and afterwards a normal training process takes place where the students initial weights are set to the pretrained weights.

2.1.2 Distillation

There are three different approaches towards knowledge distillation: offline-, online- and self-distillation [17].

Offline distillation is the most basic form and requires a pretrained teacher model that is used to train the student [17]. At first the large teacher model is trained with the original training data. The teacher model then provides additional information like logits [6] or "soft targets" [18]. This knowledge is used to train the smaller student model. Most papers assume that the pretrained teacher model already exists and they search for methods to extract knowledge from that model [18]. Training the teacher model can be computationally expensive, however, training the student afterwards is usually more resource friendly.

Online-distillation is another method where teacher and student model are trained at the same time [17]. It is used when there is no pretrained model available. Teacher and student may switch their functionality while training and the teacher becomes the student and vice versa. This means there is no fixed teacher and student model. One method inspired by online-distillation is called codistillation [3]. This approach uses several networks with the same layout and trains them simultaneously. All models are trained with the same dataset and several networks are distilling from each other. This means that there is no specific teacher and student model but each model is trained using the provided knowledge of the other networks. An advantage of codistillation is that it can accelerate training.

Self-distillation is the third and last form of distillation presented by Gou et al. [17]. This approach uses one network which is teacher and student at the same time. There are several techniques of how and which knowledge is transferred within the training time. Lan et al. proposed a method called Self-Referenced Deep Learning [22]. This method does not need a teacher model. At first the network is trained using a normal supervised approach with a conventional loss function and afterwards the learned knowledge is used to further train the model. During this training stage it is tried to minimize two loss functions, the ordinary loss used for supervised training and a imitation loss that uses the extracted knowledge.

2.1.3 Teacher Student Architecture

The quality of the resulting student model is not only determined by the knowledge that is transferred from the teacher, but also by the network architecture of both models [17]. In most distillation scenarios the teacher and the student network size is fixed and there are no changes during distillation. KD was first used to transfer knowledge from a teacher to the student, a smaller network with little depth and width [18]. The teacher on the other hand is usually a large model with lots of layer and a high capacity. This large discrepancy in the network size is called capacity gap.
Mirzadeh et al. showed that this can be ineffective and cause an inferior performance of the student [28]. They proposed a method called Teacher Assistant Knowledge

Distillation that can reduce the gap and improve the student's accuracy. In their paper they were able to illustrate that a student distilled from a teacher with a large capacity performs worse than a student trained with the help of a smaller teacher network. The basic recipe for this method is to use an additional model called Teacher Assistant (TA), which has a size that is smaller than the teacher but larger than the student. The TA is trained with the help of the teacher model and afterwards the student model is distilled form the TA. A TA is especially useful when the size of the teacher and student are fixed beforehand.
Mirzadeh et al. show that a larger and more complex teacher model leads to an increased accuracy of the teacher model itself [28]. The student model that is distilled from the teacher model, however, can benefit from a slightly larger teacher model but once the teacher gets too complex, the accuracy of the distilled student decreases. There are three factors that influence this result. The first one is the increased teacher accuracy that leads to better predictions which are superior for supervising the student model and therefore increase the distillation performance. The second factor that needs to be considered is that a larger teacher model is more complex and it is possible that the small student network is not able to mimic the learned function of this complex model. And finally the teacher achieves an increased accuracy and gets more confident about the predicted output this leads to "soft targets" that cannot convoy as much information to the student because they become closer to the "hard targets". These last two factors are reasons why the student's performance decreases once the teacher model becomes too complex and predominate the first factor. With the help of a TA, the second and third factor can become less important. This is because the student learns from the TA network, which is less complex than the teacher. Furthermore, the TA produces softer targets and is less certain about its decisions. On the other hand, introducing a TA may reduce the positive influence of the first factor. However, Mirzadeh et al. showed that this does not lead to an inferior performance of Teacher Assistant Knowledge Distillation because improving the second two factors outweighs the negative impact on the first factor [28]. They further investigated what the best size of the TA is and found out that distillation benefits from a TA regardless of its size. However, they discovered that the best performance provides a network size that is close to the teachers and students average performance. To evaluate the best size, networks with different number of neurons are trained from scratch and without distillation. Based on the performances of these networks, the average performance is calculated. The network size with a performance closest to the average performance is the best size for a TA. The accuracy of the student model can be even further improved when several TAs are used.
The student model does not have to have fewer layers than the teacher. There are other architectures possible too. Distillation is advantageous even when the architecture of teacher and student is similar [3]. The important point is that there are some dissimilarities between them, like another training set, mixed order of training samples or a different weight initialization.

2.2 Deep Neural Networks and Shallow Neural Networks

The universal approximation theorem states that a feedforward neural network with a single layer is able to learn any measurable function to a desired degree of accuracy ([19], [20], [24]). However, Ba et al. showed that training a DNN using the original training data produces a better performing model with higher accuracy than training a shallow neural network with the same data [4]. Using model compression, neural networks with a shallow architecture can be trained that achieve a better performance than networks with the same architecture that are trained from scratch with the original training data. They proposed several possible answers to the question why model compression is able to train shallow networks that achieve a higher accuracy than shallow networks trained directly on the original data. One possibility is that in the original training data some samples are mislabeled but the teacher has learned to predict them correctly. The teacher is then able to provide correct labels for the synthetic data that is used for training the student model. Another reason why mimic models may achieve a higher accuracy is that the teacher has learned a function based on complex training data. The data provided by the teacher could be simpler. The third possibility is that it is more difficult to learn from "hard targets" (0 or 1). The student is trained with data labeled by the teacher and can even learn from the uncertainties of the teacher because it is trained using "soft targets".
Whenever shallow models are trained with the original training data they tend to overfit [4]. This prevents them from learning the same function as a DNN when using the same training data. Other regularization techniques could help to overcome the overfitting problem. However, the results proposed by Ba et al. suggest that a shallow neural network is as capable of learning complex functions as a DNN, sometimes even with the same number of parameters [4]. They showed that small neural networks with only a single layer can achieve as good or even better results than a DNN when they are trained to mimic a powerful teacher. In their experiments they used model compression and trained at first a large teacher model. Afterwards, they employed this model to teach a shallow student model. Based on these results, one can say that KD or its predecessor model compression are superior methods and often perform better than training a shallow neural network from scratch.

2.3 Data Generation Methods

Creating synthetic data is challenging [6]. Often large amounts of unlabeled data is hard to access. Therefore, pseudo data needs to be generated. The student model will fail to approximate the function of the larger model when the synthetic data only reflects a small field of the distribution of the true data. However, when the pseudo data is generated from a very large distribution then lots of samples are needed to reflect the region of the true data well enough. It is best when the synthetic data is sampled within

the region the true data comes from. When both data, true and artificial, come from the same distribution, then fewer samples are needed to train the student to approximate the function learned by the teacher well.
Three approaches to generate artificial data are presented by Bucila et al. [6]. These three methods, RANDOM, MUNGE and Naive Bayes Estimation (NBE), enable us to create an arbitrarily large amount of training data.

2.3.1 RANDOM

Random sampling is a commonly used and simple approach to generate artificial data [6]. The RANDOM method mentioned by Bucila et al. uses the existing training set and generates additional data points by choosing for each attribute a value uniformly at random from all values of the same attribute that are available in the training set [6]. Therefore, it uses a non-parametric bootstrap approach. This method can create huge amounts of artificial samples in an easy way. However, the generated data represents a large field that does not exactly reflect the true data distribution. Within the true data there may be some structure present between the attributes. This structure is lost when we sample each value of the attributes independently. Loosing the structure in the training data can lead to more data that is needed for training because most of the artificial data does not represent the region of interest.

2.3.2 MUNGE

Bucila et al. proposed in their paper another approach to generate pseudo data named MUNGE [6]. This method calculates the nearest neighbor of each training sample. The algorithm differs between continues and non-continues attributes.
Whenever the samples are non-continues then one attribute of a training example is swapped with the attribute of its nearest neighbor with a probability p. Continues attributes e_a are changed to another value with probability p, too. The new value they get assigned is drawn randomly from a normal distribution with the mean of the nearest neighbor e'_a and a standard deviation $sd = |e_a - e'_a|/s$, where s is the local variance parameter. This method is able to approximate the true data distribution well and preserves the structure between the attributes.

2.3.3 NBE

Naive Bayes Estimation (NBE) is another method mentioned by Bucila et al. [6]. This method is also able to overcome the weakness proposed in the previous section 2.3.1 and the structure of the data can be maintained [6]. When using NBE the artificial data is sampled from the joint distribution of the attributes of the training samples. With the help of Expectation Maximum (EM) the joint distribution of the data is learned [12]. This method allows to generate artificial samples that are lying in the region of interest. The

NBE method is a mixture model algorithm and was first introduced by Lowd et al. [26]. It can estimate the joint distribution of continues and discrete attributes. In algorithm 1 the training process of NBE is shown step by step.

Algorithm 1 : NBE iterative learning algorithm [26]

```
Input : training set $T$, hold-out set $H$, initial number of components $k_0$, and
        convergence thresholds $\delta_{EM}$ and $\delta_{Add}$;
Output : $M_{best}$
Initialize $M$ with one component;
$k \leftarrow k_0$;
repeat
    Add $k$ new mixture components to $M$, initialized using $k$ random examples
      from $T$;
    Remove the $k$ initialization examples from $T$;
    repeat
        E-step: Fractionally assign examples in $T$ to mixture components, using
          $M$;
        M-step: Adjust parameters of $M$ to maximize the likelihood of this
          fractional assignment;
        if $logP(H|M)$ is highest so far then
            $M_{best} \leftarrow M$;
        end
    until $logP(H|M)$ fails to improve by ratio $\delta_{EM}$ over the last iteration;
    $k \leftarrow 2$ x $k$;
until $logP(H|M)$ fails to improve by ratio $\delta_{Add}$ over the last iteration;
Execute E-step and M-step twice more on $M_{best}$, using examples form both $H$
  and $T$;
```

2.4 Data Exploration Methods

Another approach to generate synthetic data is to use data exploration methods. In Reinforcement Learning (RL) exploration strategies are needed to improve the actor performance [36]. RL uses an agent that takes actions within an environment and the environment returns an observation and the corresponding reward. The ultimate goal of RL is to maximize the total reward the agent achieves and to minimize the time it takes to train the agent. An actor that acts only in the environment it knows and tries to improve its performance within this space, may not discover the parts of the environment that would yield an even better reward. On the other hand, if an agent only acts randomly, the training can take a long time and require a lot of computational resources. Therefore, a good balance between exploration and exploitation is neces-

sary.
The pretrained teacher model that is used in KD is able to provide labels to a random set of inputs. In this thesis the same simulation environment is used for data generation as for the training of the teacher model. Using data exploration methods one can generate new actions or inputs that are then labeled with the help of the teacher network. Training the student network with these generated samples can improve the overall performance since the added samples introduce new states of the agent which is additional knowledge that is transferred to the student. These new states differ from the states that are introduced using only the data provided by the simulation. This can be beneficial for the student's performance.
The ϵ-greedy exploration approach or random exploration is a simple method often used in RL [36]. This strategy has only one parameter $0 \leq \epsilon \leq 1$ that is used to decide which action to take. With a probability of ϵ the actor takes a random action and otherwise, the actor chooses the optimal action based on the prediction of the teacher network.

3 Problem Setting

This master thesis was created as part of a project called Phoenix. The project is a work of several PhD students and students of the Chair of Data Processing. The goal of the project is to train a drone within a simulation using Reinforcement Learning (RL). Different flight maneuvers are trained and afterwards the neural network is transferred to the drone to perform the flight maneuvers in the real world. The drone used to test the neural network in the real world is the Crazyflie 2.1[1] which is produced by Bitcraze AB.

3.1 Crazyflie 2.1

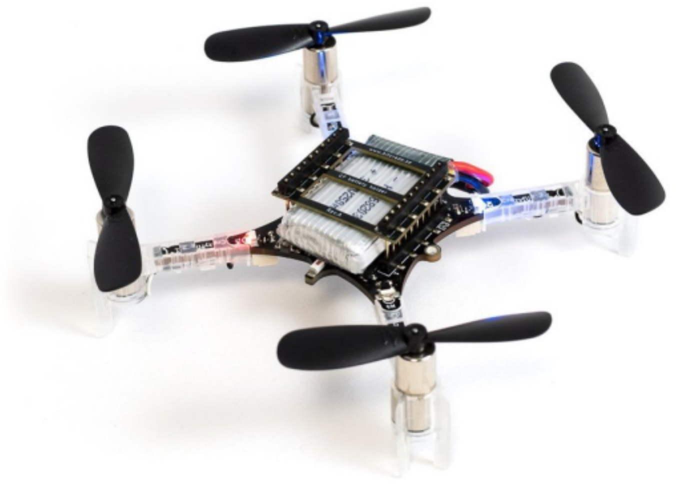

Figure 3.1: Image of the drone Crazyflie 2.1[1].

The project uses the Crazyflie 2.1, because it has an open source platform like its predecessor Crazyflie 2.0 and is therefore particularly suitable for research and development [16]. All technical data can be found in the data sheet of the Crazyflie 2.1[1]. The system is very light with 27 grams and small, about the size of a hand, as can be

[1] https://store.bitcraze.io/products/crazyflie-2-1

seen in figure 3.1. It is equipped with four coreless DC motors and a 250mAh battery, which allows a flight time of up to seven minutes. On the board there are two microcontrollers, one main microcontroller and one for the radio and power management. The controller for main applications is a Cortex-M4 processor that runs with 168MHz and has a SRAM with the size of 192kb. The Crazyflie 2.1 supports Bluetooth Low Energy and can be controlled with a smartphone. Furthermore, it provides a so called Crazyradio which enables communication with a personal computer. This is tested to one kilometer range line-of-sight. The Crazyflie 2.1 is also able to perform real-time logging and variable setting, which makes it a perfect fit for the Phoenix-Project.
In 2009 the three founders of Bitcraze[2] startet to create the Crazyfie. They decided to provide an open source development platform for their products. This enables the users of the Crazyflie to conduct research and to modify the product according to their individual needs. The client API of the Crazyflie 2.1 is written in Python but other implementations can be downloaded from GitHub like Ruby, C#, C/C++, Java and more.

3.2 Simulation Environment

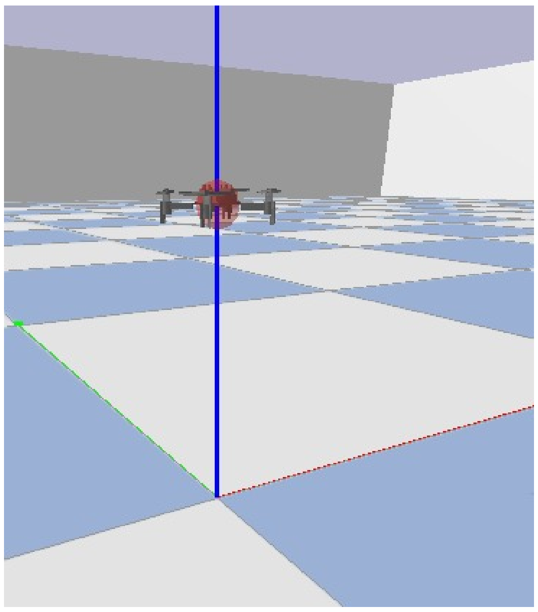

Figure 3.2: Image of the simulation environment gym-pybullet-drones.

OpenAI Gym[3] can be used when working with RL, because it offers a collection of different benchmark problems [5]. It provides several abstractions of environments that

[2]https://www.bitcraze.io/about/bitcraze/
[3]https://gym.openai.com/

share a common interface. OpenAI Gym uses a series of episodes to demonstrate the agent's experience. The agent starts his actions from an initial state which is randomly sampled. The interaction between agent and environment is stopped when a terminal state is reached by the environment. One advantage of OpenAI Gym is that each environment has a version number that is changed each time the environment changes. This ensures that published results are reproducible.
Gym-pybullet-drones is a Gym environment that can handle multiple agents [30]. It is especially designed for quadcoptors. In the Phoenix-Project, RL is used to train a drone in a simulation environment to accomplish different flight maneuvers. The acquired model is then applied to a quadcoptor and performs the flight maneuver in the real world. However, there are some effects in the real world that need to be considered in the simulation as well, in order to train a stable flight maneuver. Gym-pybullet-drones is a suitable environment because it supports realistic collisions, extensible dynamics via Bullet Physics and some aerodynamic effects [30]. The supported effects are drag, ground effect and downwash. With the help of PyBullet it is possible to model these effects separately and use them in combination [30]. Drag is a force that is produced by the rotating propellers and acts in the direction opposite to the movement. The ground effect appears whenever the quadcoptor hovers near the ground. The proximity to the ground results in a change in air circulation, which leads to increased thrust. Finally, the downwash is an effect that appears when two drones fly over each other. This effect can cause a reduced lift for the drone at the bottom. These three aerodynamic effects are included in the simulation environment and improve the experience when the models are applied in real world. However, in the end, we are not able to model all effects that appear in the real world. Another positive aspect of this simulation environment is that their default quadcopter model is the Bitcraze Crazyflie 2.x which corresponds to the used drone in the Phoenix-Project.

3.3 Scope of Work

This section describes the experimental setup and the performed experiments in more detail. The objectives of the work and the approaches to achieve these objectives are also described more precisely. Furthermore, the thesis uses a modified version of KD which is explained below.

3.3.1 Experimental Setup

In the project, a neural network is trained using Reinforcement Learning (RL) within a simulation. The model learns a specific flight maneuver, such as hovering at a certain point. Depending on how well the model performs the given task, a reward is calculated. Various penalties reduce the actor's reward. Whenever the actor turns too fast, the reward will be reduced or if the episode terminates before the maximum time

steps have expired. These penalties are used to calculate the actor's final reward. With the help of the simulation and RL a few models are trained with different sizes, but all models have one thing in common, they have two hidden layers and several hidden nodes. This results in a large network with many parameters, which requires a lot of memory. One of these networks will later be transferred to the drone and will execute the previously trained flight maneuver in the real world. The problem here is that models of this size cannot be utilized on the drone. It has been empirically shown that two hidden layered neural networks with 32 neurons each are small enough in order to be executed with the required control frequency on the drone. Based on this experience the target size for neural networks in this thesis is set to two layers each with a maximum of 32 neurons.
This raises the question of why smaller models are not trained directly using RL. Training with RL is very time-consuming and the project is still in the initial phase. Many of the current problems deal with different settings and architectures that are possible and are looking for ways to optimize them. For example, if a new RL model has to be trained for each architecture, this takes much more time than training once a large teacher model and use this teacher to distill different student architectures using KD.

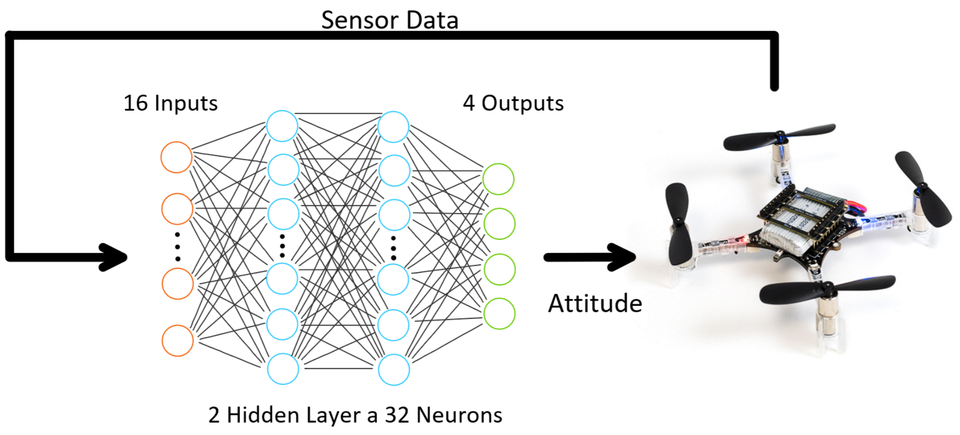

Figure 3.3: Crazyflie 2.1[1] controlled by a neural network using sensor data as inputs and the attitude as outputs to control the drone.

The sensor data of the drone serve as inputs on which the calculations of the neural network are based. Here the position, the velocities, roll, pitch and yaw and their derivatives, as well as the last executed action are transferred and used as input into the neural network. Based on this data, the values of the output from the neural network, the attitude, are calculated and passed to the quadrotor. Figure 3.3 visualizes the neural network and its corresponding inputs and outputs.

3.3.2 Applied Knowledge Distillation

The Knowledge Distillation (KD) used in the project is different from the one presented in section 2.2 by Hinton et al. [18]. In the project KD is used without the transfer of additional knowledge. Furthermore, no temperature is adjusted during the training. The teacher learns in a difficult RL environment and transfers its learned function to the student based on the outputs. In the Phoenix-Project, the problem is a regression problem and thus, only the "hard targets" prepared by the teacher are transferred to the student. In most KD scenarios it is a classification task where "soft targets" are used to train the student which provide additional knowledge ([18], [15]).

3.3.3 Objectives and Approaches

One aim of this work is to investigate which data is best suited for training the student model. Training data can be generated using the simulation and the teacher model. The simulation provides the sensor data and the model uses this data to calculate the outputs, the so-called action. This action is then executed in the simulation and new input data is generated. The generated data sets are stored and used to train the student model. Since it is a stochastic simulation, not all executed trajectories are the same. The parameters of the environment are changed randomly. Thus, several trajectories lead to different data sets. However, since the learned flight maneuver is always the same, the drone's movements are very similar in all trajectories. Therefore, there is a risk that in case of some disturbances or deflections of the drone, it is not able to compensate them. As a result it may be useful to add more data for training the student when deflecting the drone to a different position during the simulation, providing new sets of sensor and output data. In the optimal case, the complete function and all its local shapes that the teacher has learned is conveyed to the student during training.
This thesis therefore explores different methods to generate data in addition to the pure simulation data in order to obtain a diverse data set for the training of the student model. The performance of the teacher is compared with that of the student and the extent to which the results are reproducible is also evaluated. The reproducibility of the results is evaluated by training several student models with the same architecture, the same teacher and the same training data and comparing the results obtained in each case. The closer the results of the individual students are to each other, the higher the reproducibility. When the models are trained, their performance is compared by calculating their achieved reward using the simulation. To determine the reward, the trained model is loaded into the simulation environment and executes a predefined number of trajectories. For each trajectory the performance of the actor is calculated as reward, which is stored. The average reward based on the number of trajectories is then returned and used as a basis for comparison.

Another aim of this thesis is to study the effects of a Teacher Assistant (TA). The best method to generate training data is chosen and used to check whether the difference in size between teacher and student leads to a decrease in performance and whether a better model can be trained with the help of a TA. This approach aims at reducing the capacity gap and can lead to a better result of the student [28]. Different architectures of TAs are investigated while the architectures of student and teacher are determined in advance. The size of the teacher model is pre-determined, since it is available already trained and the size of the student is limited by the constraints of the drone.

4 Methods

The following section presents the data generation methods that are investigated in this thesis. In particular, it is examined whether the method is suitable to perform a successful KD and to generate a well-performing student. Based on the findings, it will be determined whether the student can be further improved with the help of a TA.

4.1 Data Generation

As described in section 2.1, KD trains a student model using input data to which the teacher generates the corresponding output labels. It is important to choose the input data carefully, so that the student is able to learn the exact function that the teacher has already learned. Input data can be generated with the help of the simulation presented in section 3.2. The following subsections examine different ways to generate diverse input data, which differ to some extent from those inputs generated in the simulation.

4.1.1 Pure Simulation Data Generation

One way to train a student model is to use only the data pairs generated in the simulation, called pure simulation data. Using the simulation, an unlimited amount of data can be created and the neural network can be trained with any number of training samples. For the generation of the data points, the simulation is performed which provides the input data and executes the trained teacher network. The teacher then generates the corresponding actions. The input data and the produced actions are used as training data for the student. The teacher network that is executed in the simulation is trained and performs repeatedly the learned flight maneuver. The disadvantage of this method, however, is that as soon as a disturbance acts on the drone, the student model does not learn to compensate for it and thus cannot execute the flight maneuver as desired. Although some aerodynamic effects are taken into account in the simulation, there are several more in reality. These are not included in the data pairs of the simulation. In addition, only one specific flight maneuver is performed, for example hovering at a certain point. The teacher used in the simulation for data generation is already trained and therefore the deflections of the drone around the target point during the simulation are only very small. If the student is later deflected more than the teacher in the training data, it cannot compensate for the deflection and the drone of the student model crashes. A student model that was only trained with pure simulation data performs less successful than neural networks that were trained with data sets that additionally consist to a certain extent of sample data that were generated otherwise. The additional data points are used to transfer more knowledge from the teacher to the student. Not only data pairs that reflect the simulation but additional data pairs that represent, for example, a disturbance. To better compare the different data generation methods, the variant with pure simulation data as reference and baseline is included in the results.

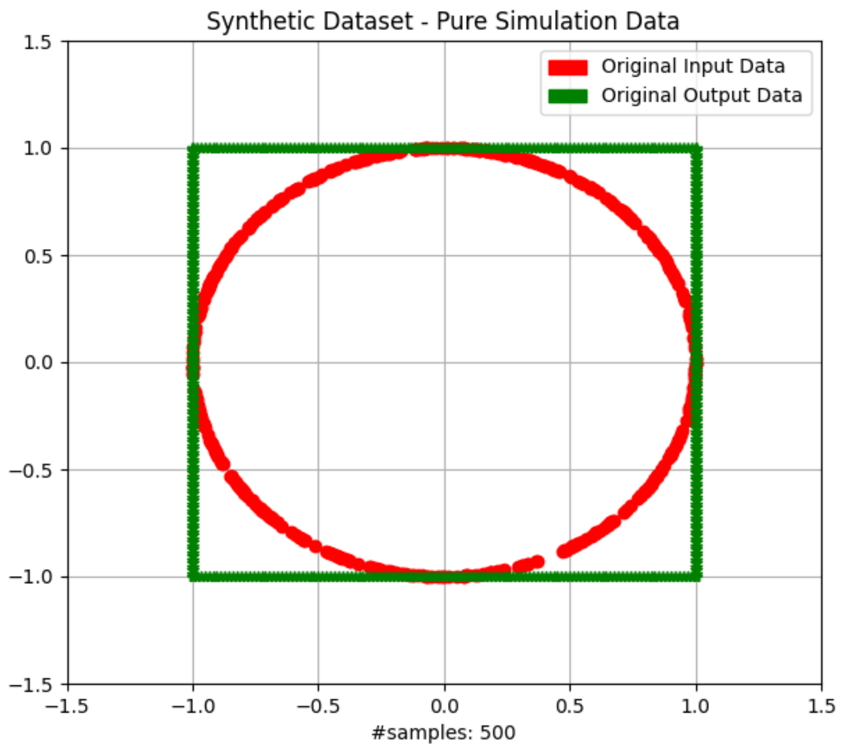

Figure 4.1: Original input and output data that is generated using Pure Simulation Data Generation.

Figure 4.1 shows a toy example, with two-dimensional data points. The pure simulation data is also called original data. The graph shows both the original inputs and outputs. This is an easy to understand example with simplified sample data. The real data is generated using the simulation and the teacher network and consists of 16 different inputs and 4 different outputs. This graph is used to compare the different data generation methods presented in the following sections and to better understand the different approaches.

4.1.2 Epsilon-Greedy Data Generation

This method is based on the ϵ-greedy algorithm presented in section 2.4. The Epsilon-Greedy Data Generation method is described in algorithm 2 and uses the data produced in the simulation environment. A adjustable parameter determines the probability with which an arbitrary action is generated. This action is not added to the final training set, but it is used to deflect the drone and the subsequent data pairs are then included in the training set. Described differently, this method is a combination of the pure simulation data and additional arbitrary actions leading to modified input data. This way the drone is not only trained hovering at the optimal point but also during a certain percentage of deflections and their subsequent movements to compensate for them again. If the parameter of probability is set to a low value, predominantly the pure

Algorithm 2 : Epsilon-Greedy Data Generation

```
Input : number of training samples $n$, probability parameter to generate a
        random action $\epsilon$, teacher model $tm$
Output : training dataset $T$
$X \leftarrow \emptyset$;
$Y \leftarrow \emptyset$;
$T \leftarrow \emptyset$;
$x \leftarrow$ reset the environment to a random initial state and get input data;
for number of samples to generate $n$ do
    if probability $\epsilon$ then
        $a \leftarrow$ draw a random action from the action space;
    else
        $a \leftarrow$ compute action based on the teacher model $tm$ and input data $x$;
        $X \leftarrow x$, add current input to dataset;
        $Y \leftarrow a$, add current action to dataset;
    end
    $x, done \leftarrow$ compute a new timestep with new input data $x$ in the
      environment based on the current action $a$ and indicate whether a
      episode reached its end $done$;
    if end of episode $done$ then
        $x \leftarrow$ reset the environment to a random initial state and get input data;
    end
end
$T \leftarrow [X,Y]$;
```

simulation data is used. Otherwise, a deflection of the drone is produced and the following data is then included in the training set. Because the random actions differ from the simulation data, the trained student model can be expected to perform better than a model trained with pure simulation data. This can be assumed, because additional knowledge is transferred from the teacher to the student model. In case of occurring disturbances and thus deviating input data, the neural network is also trained to some extend in these regions.

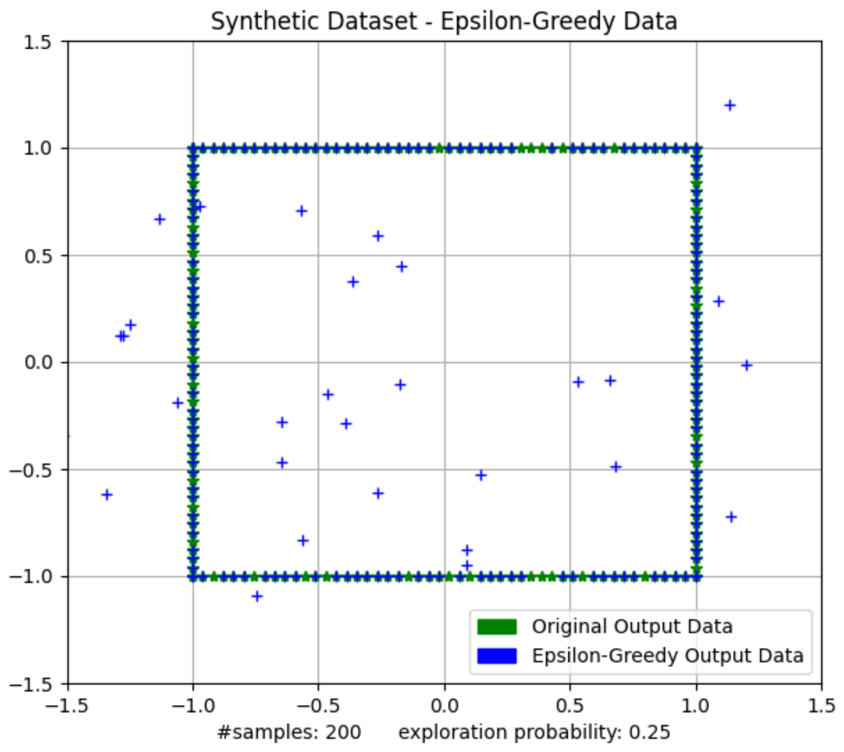

Figure 4.2: Comparison between original output data and data generated using Epsilon-Greedy Data Generation.

Figure 4.2 shows again the toy example, with two-dimensional data points. The Epsilon-Greedy Data Generation method changes the output data. The original output data are the data points generated by the Pure Simulation Data Generation method. With a certain percentage, the so-called exploration probability, not the original output data is used but an arbitrary output is generated. The graph shows both the original output in green and the output generated by the Epsilon-Greedy Data Generation method in blue. In the final training set, the randomly modified data points themselves are not added, but the subsequent input and output data pairs.

4.1.3 Random Data Generation

The Random Data Generation approach uses a certain percentage ϵ of randomly generated input data in addition to the pure simulation data. Algorithm 3 shows the procedure that is needed in order to generate the training data. The additional data is

Algorithm 3 : Random Data Generation [6]

```
Input : number of training samples $n$, probability parameter to take a random
        input $\epsilon$, teacher model $tm$
Output : training dataset $T$
$X \leftarrow \emptyset$;
$Y \leftarrow \emptyset$;
$T \leftarrow \emptyset$;
$x \leftarrow$ reset the environment to a random initial state and get input data;
for number of samples to generate $n$ do
    if probability $\epsilon$ and $X$ contains more than two input samples then
        $r \leftarrow$ calculate the range of each input attribute based on current
          available samples in $X$;
        $x \leftarrow$ draw a input sample with random attributes that are calculated
          within range $r$;
        $a \leftarrow$ compute an action based on the generated input sample $x$;
    else
        $a \leftarrow$ compute an action based on the teacher model $tm$ and the input $x$;
    end
    $X \leftarrow x$, add current input to dataset;
    $Y \leftarrow a$, add current action to dataset;
    $x, done \leftarrow$ compute a new timestep with new input data $x$ in the
      environment based on the current action $a$ and indicate whether a
      episode reached its end $done$;
    if end of episode $done$ then
        $x \leftarrow$ reset the environment to a random initial state and get input data;
    end
end
$T \leftarrow [X,Y]$;
```

produced by randomly sampling the input data based on the range of each input attribute of the current training data which was generated using the simulation. This creates new input combinations, but within the range of the usual input attributes that are generated within the simulation. Based on these inputs, the associated outputs are determined using the teacher model. By this strategy additional data points are generated, which are within a range that occurs in the simulation, thus, are not completely random but possibly did not occur so far in this combination, as they are generated arbitrarily. This covers a larger range of possible input data than using only pure simulation data, but also keeps the range rather small compared to a completely random version where all possible values of an attribute are considered. After all, the larger the range in which the input data lies, the more pairs of data are necessary to adequately represent the true data distribution that is needed by the student model to approximate the function of the teacher model well [6]. In figure 4.3 an example is shown using Random Data Generation. The example shown is based on an existing example presented in the work of Bucila et al. [6]. This is again the toy example with simplified data already presented in the previous sections.

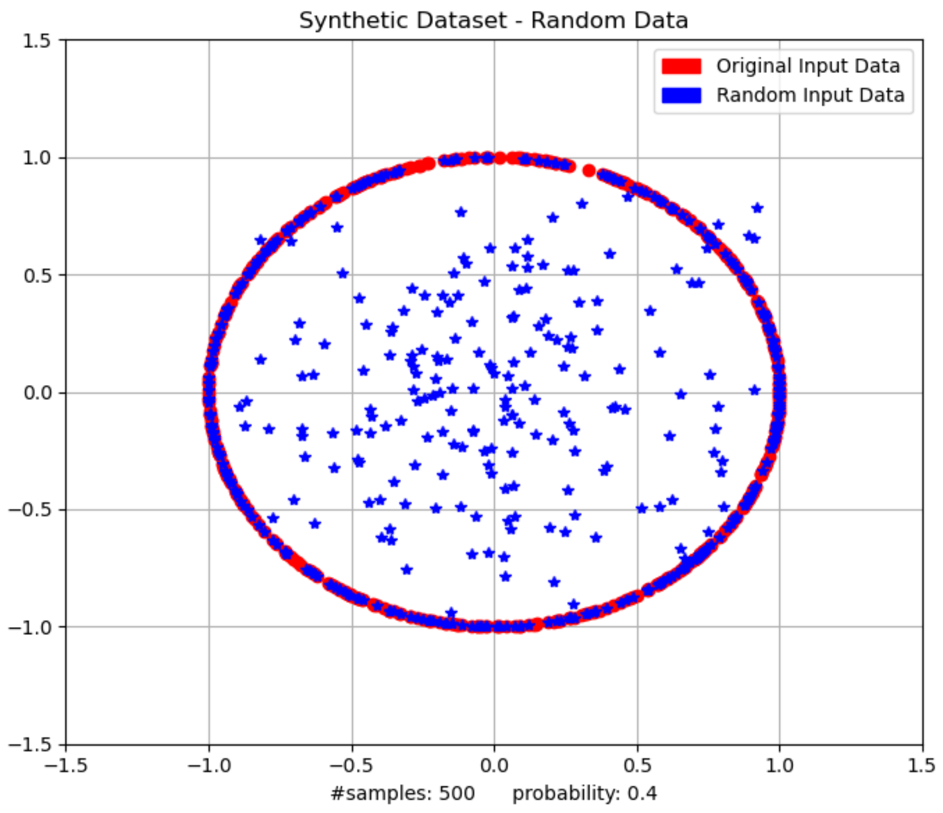

Figure 4.3: Comparison between original input data and data generated using Random Data Generation.

The distribution of the original data forms a circle with the center at the zero point and a radius of one. These original data points are visualized in red and represent data points generated by the simulation. In blue the data generated using the Random Data Generation Method can be seen. It is obvious that the true data distribution is not exactly met and the generated data differs from the original input data. The algorithm used to create the random data pairs focuses on the midpoint between the minimum and maximum value of the input attribute. Therefore, more data points are generated around the center point than in the corners.

4.1.4 Munge Data Generation

The algorithm named MUNGE was first introduced by Bucila et al. [6]. The main goal of this method is to produce synthetic data that is very similar to the real data distribution. This method is described further in subsection 2.3.2. If we apply the method to the problem at hand and generate synthetic training points in addition to the data generated with the Pure Simulation Data Generation method, which serve as the original training set, we generate a larger data set with a similar data distribution as the original training data. Due to the very similar distribution of the artificial data and the original data, we can expect the results obtained to be similar to the results using pure simulation data as training data but with still some little variations. The input data has 16 different attributes and is therefore 16-dimensional. Thereby it is very likely that the actual data distribution lies in some kind of tube within these 16 dimensions and not simply arbitrarily.
In algorithm 4 the procedure employed in this thesis is formulated and an implementation from GitHub[1] is used. The procedure is based on the MUNGE algorithm introduced by Bucila et al. [6]. Using Munge Data Generation, the additional data points are very similar to the original data. This can be seen in figure 4.4. The visualized example is based on the experiment and toy example described by Bucila et al. [6]. The original input data provided by the Pure Simulation Data Generation method is again visualized in red and the created data samples using the Munge Data Generation method are drawn in blue. The used number of samples is 250 and half of the samples are swapped using the munge algorithm whereas the other half remains the same as the original data. The example clearly shows that the Munge Data Generation method generates data pairs that are very similar to the original data and differ only slightly. The original data distribution is preserved.

[1] https://github.com/alperengormez/munge_python

Algorithm 4 : Munge Data Generation [6]

```
Input : number of training samples $n$, swap probability $\epsilon$, size multiplier $k$, local
        variance $v$, teacher model $tm$
Output : training dataset $T$
$X, X' \leftarrow \emptyset$;
$Y, Y' \leftarrow \emptyset$;
$T, T' \leftarrow \emptyset$;
$x \leftarrow$ reset the environment to a random initial state and get input data;
for number of samples to generate $n$ do
    $a \leftarrow$ compute action based on the teacher model $tm$ and the input data $x$;
    $X \leftarrow x$, add current input to dataset;
    $Y \leftarrow a$, add current action to dataset;
    $x, done \leftarrow$ compute a new timestep with new input data $x$ in the
      environment based on the current action $a$ and indicate whether a
      episode reached its end $done$;
    if end of episode $done$ then
        $x \leftarrow$ reset the environment to a random initial state and get input data;
    end
end
$T' \leftarrow [X,Y]$;
for number in size multiplier $k$ do
    for input sample $x$ in $X$ do
        for attribute $x_a$ of sample $x$ do
            if probability $\epsilon$ then
                $nn_a \leftarrow$ calculate the attribute of the nearest neighbor sample
                  using Euclidean distance;
                $nn_a \leftarrow norm(x_a, sd)$, where $sd = |x_a - nn_a|/v$;
                $x_a \leftarrow norm(nn_a, sd)$, where $sd = |x_a - nn_a|/v$;
            end
        end
        $a \leftarrow$ calculate action based on current sample $x$;
        $X' \leftarrow$ add current input sample $x$ to the dataset;
        $Y' \leftarrow$ add current action to dataset;
    end
end
$T \leftarrow [X',Y']$;
```

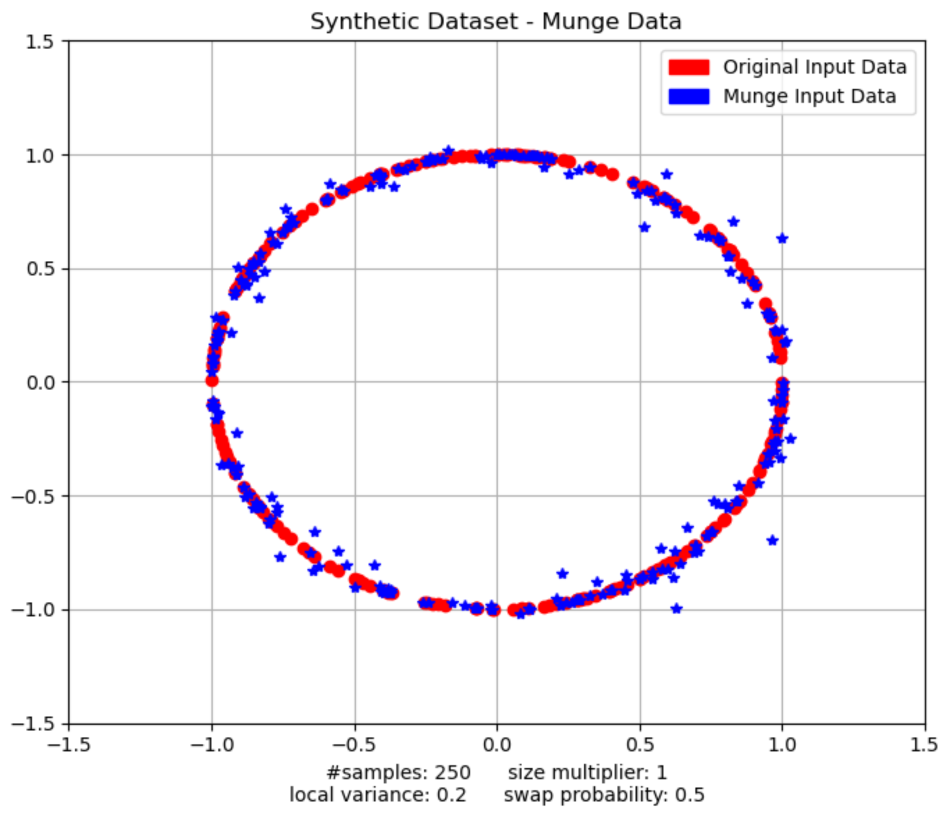

Figure 4.4: Comparison between original input data and data generated using Munge Data Generation.

4.2 Teacher Assistant Knowledge Distillation

A Teacher Assistant (TA) is used to check if the performance of the student models can be further improved. In subsection 2.1.3 the method presented by Mirzadeh et al. is described in more detail [28]. In this chapter, the implementation within this experiment is discussed, as well as the training data used. For a TA to be useful in the training process, one needs defined teacher and student network architectures. This part of the thesis uses a teacher architecture with two layers of 500 neurons each and a student architecture with two layers of 32 neurons each. This teacher is used because it was available and fully trained at the beginning of the thesis. The TA also consists of two layers, but experiments are performed with different numbers of neurons. However, the number must be between that of the teacher and the student. The Epsilon-Greedy Data Generation method is used to generate the training data for the TA. This method was selected because it achieved the best results when comparing the different data generation methods. First, the TA is trained with a data set that was created with the Epsilon-Greedy Data Generation method, where the outputs were calculated using the teacher model. Subsequently the training data for the student uses the input data from the already created training set for the TA. The corresponding outputs are then calculated with the help of the trained TA model. The TA then becomes the teacher and trains the student. In summary, the input data is the same for the TA and student training, but the output data is adjusted by the outputs of the respective teacher.

5 Results and Interpretation

In the following, the experiments performed and their results are described. If KD is used, then the student model is trained to reproduce the teacher's results as closely as possible. Thus, the student model cannot overfit the original problem, if one assumes that the teacher himself has not overfitted and sufficient training data is used as stated by Bucila et al. [6]. This property was also confirmed empirically using pure simulation data. Due to this fact, overfitting will not be considered further in the experiments.
All experiments in this section were performed with a patience of 40 and 10 student models were trained in each case, which are evaluated in the different graphs. In order to compare the models with each other, the performance of the individual models was determined. The calculated reward was used as a measure for the performance. The higher it is, the better the generated student model. The trained model was tested in the simulation in which the training data was generated. However, test and training data differ from each other because the simulation selects random initial states and there are also some noise parameters which lead to unique data sets. Furthermore, the different data generation methods introduce further variations in the training data. With the help of the simulation, the reward of the respective model can be calculated. For the calculation of the reward, several trajectories are flown which consist of 500 data points each. In the following section, 500 trajectories were used in each case to achieve rewards that are comparable. This leads to a total of 250,000 data points that were used for testing the individual models.

5.1 Settings

Good parameters must be found for each data generation method in order to compare all experiments in a fair way. In this thesis, a manual search was used to find well performing parameters for each method. Since it takes a long time to find these settings, first the number of training samples needed to train a model that achieves satisfactory results is investigated.

5.1.1 Amount of Training Data

When training a neural network, it is important to strike a balance between sufficient training data and the training duration. In this work, the training duration describes the time needed to generate training data in addition to train the neural network until a

satisfactory performance is achieved or in our case until the early stopping criteria is met. The time needed to generate the training data is dependent on two influencing factors. On the one hand, the amount of data and on the other hand, the used method for the data generation.
In order to find an appropriate number of training samples for the following experiments, the performance of different data amounts is examined in this section. Pure simulation data was used as data set and four different sizes of sets were investigated. The first size called "small" contains 100 files, each including 2,000 data samples. One trajectory, using the simulation, consists of 500 samples. A slightly larger data set called "middle", uses 100 files with 20,000 samples a file, which corresponds to 40 trajectories per file. The "large" set consists of 50 files and the "extra large" set of 100 files with 200,000 data samples or in other words 400 trajectories per file. The results of the trained students are shown in the following boxplot 5.1.

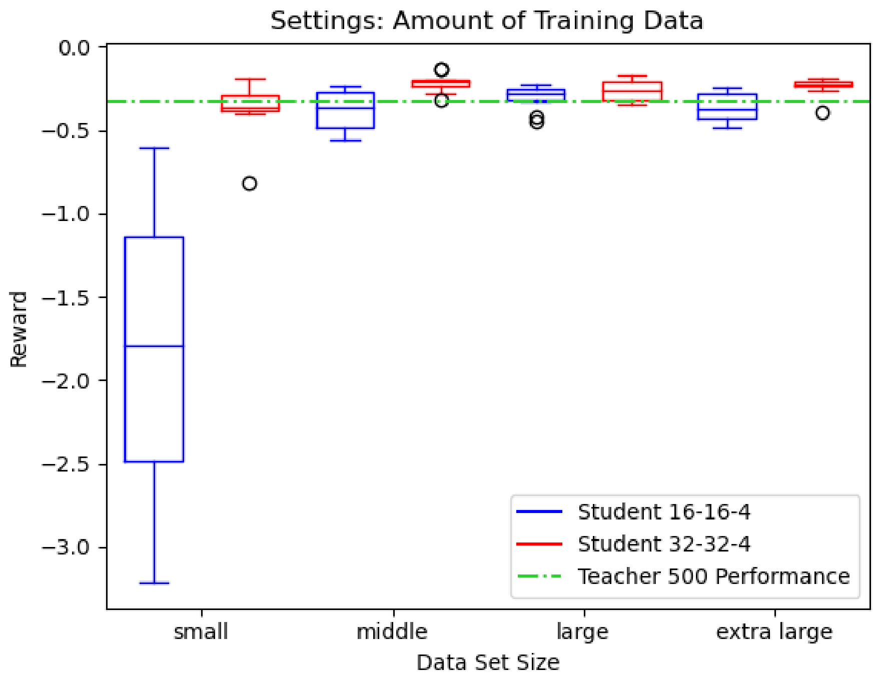

Figure 5.1: Experiment using different number of training samples of pure simulation data to identify best data set size.

The outcome of the experiment shows that with using the Pure Simulation Data Generation Method to generate training data we can reach a model performance of the student that is about as good as the performance of the teacher. Further, one can see that using more data leads to better reproducible results. This means that the 10 models generated using the same data set have a more similar performance. Of course it is not practical to use vast amounts of data because the training time will increase drastically. This experiment was conducted in order to find the appropriate amount of data that is necessary to achieve good results while keeping the training duration low. When investigating the plot in figure 5.1 we can see that when using the data set named "small" the results vary strongly, especially for an architecture of two hidden layers with 16 neurons each. On the other hand, if one uses the data set called "middle", one generates models that achieve a higher reward and are also more reproducible. Compared to all other numbers of training data, it achieved nearly as good results while keeping the training effort low. If doubling the data from the data set called "large" to "extra large", no significant improvement is visible. Weighing up the additional training time required, it is advisable to work with the smaller data set called "large". In summary, it can be concluded from this experiment that the data set named "middle" is sufficient to perform smaller experiments, such as determining the best parameters of a method and the data set size named "large" represents a good trade off between training duration and achieved reward for the final investigation and comparison of the different student models that are trained in the following experiments.

5.1.2 Random Data Generation

The random method has only one parameter and that is the probability with which an input sample is used that was not produced using the teacher model and the simulation but an input sample that was generated randomly. The method is described in more detail in section 4.1.3.
Initial tests with data sets of size "small" showed that the best performance is achieved with a probability value around 15%, so this area was investigated in more detail in the experiment. Figure 5.2, showing the results of the experiment with a data set size "middle", confirms the initial experiments and demonstrates that a probability factor of 15% achieves the best results. This seems to be a good balance between pure simulation data and random deflections of the drone via changed input data.

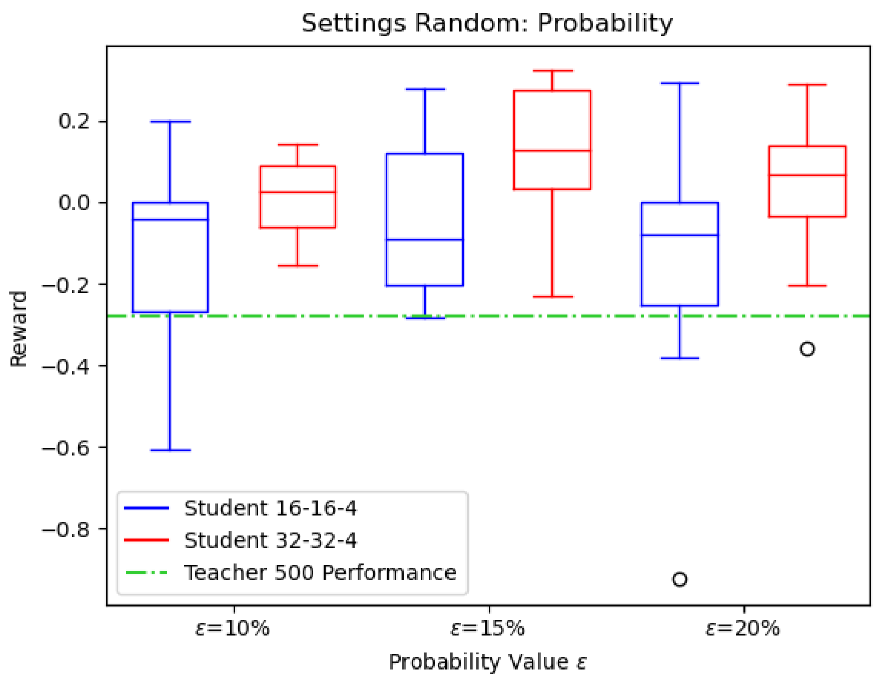

Figure 5.2: Experiment using different probabilities ϵ with which a random input sample is generated.

5.1.3 Epsilon-Greedy Data Generation

The only parameter that can be adjusted in the Epsilon-Greedy Data Generation method is the probability parameter as well. Therefore, another experiment was conducted in order to find a proper value for this parameter. The same settings were used as in the experiments before.
The boxplot in figure 5.3 shows the results of the 10 models trained using a data set of size "middle". The best trained model of a run is represented either by the whiskers or by an outlier marked as a circle outside the whiskers. If one examines with which parameters the best model can be trained, one finds that the larger the probability parameter, the better the reward the best model achieves. However, at the same time the reproducibility of the results decreases, which can be seen in the larger boxes. Therefore, a good balance must be found, which seems optimal at a parameter of 25%. Based on this experiment $\epsilon = 25\%$ is used in all further tests.

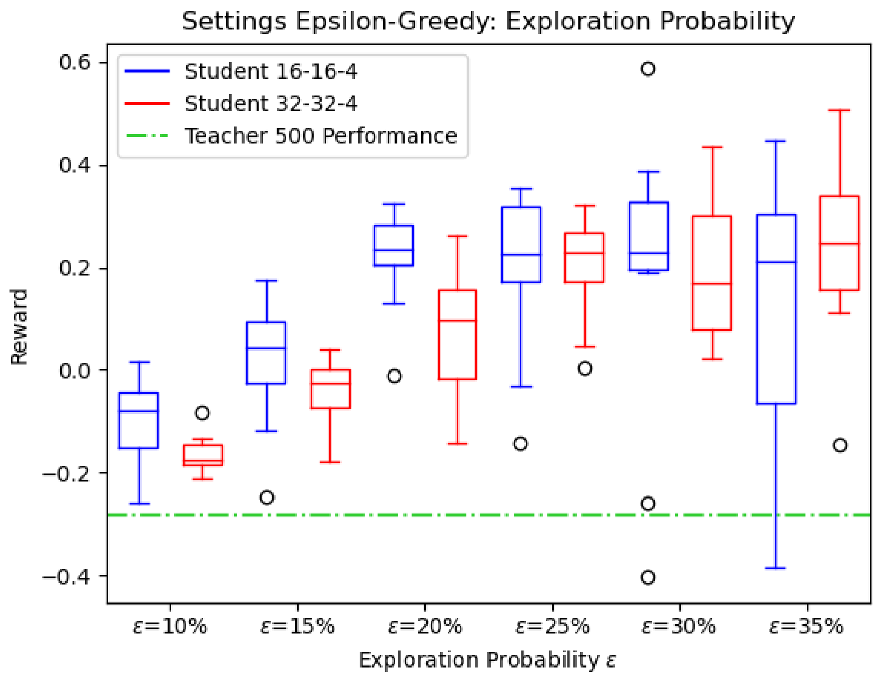

Figure 5.3: Experiment using different probabilities ϵ with which a random action is generated.

5.1.4 Munge Data Generation

The Munge Data Generation method has several parameters for which a suitable value must be found, the size multiplier m, a local variance v and a swap probability p. A disadvantage of this method is that it takes a long time to generate the data and, therefore, the training time is significantly longer than with the other methods presented in this section. Due to this disadvantage and the fact that the Munge Data Generation method uses three hyperparameters instead of one, the search for good settings takes much more time. Therefore, the preselection of the parameters of this method was made with the help of the data set size "small".

At first several variances were tested while keeping the swap probability p and the size multiplier m the same.

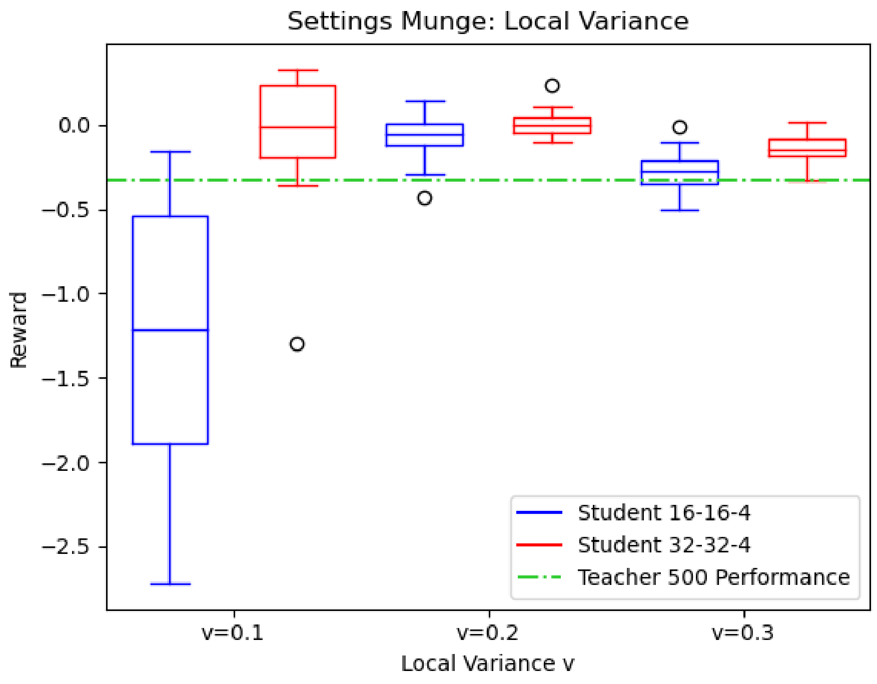

Figure 5.4: Experiment using different local variances v for the Munge algorithm with $m = 1$ and $p = 50\%$.

In figure 5.4 again the results are shown, which were obtained with the help of 10 runs and a patience of 40. Since the main focus is on the local variance, the other two parameters were not changed. In the graph one can see that a very small variance leads to highly scattered results. In this example, a variance of 0.2 seems to be the most suitable.
With the help of this observation, further tests were carried out to find a good size multiplier. The parameter of the local variance was set to 0.2 and a swap probability of 50% was used. The results are shown in figure 5.5.

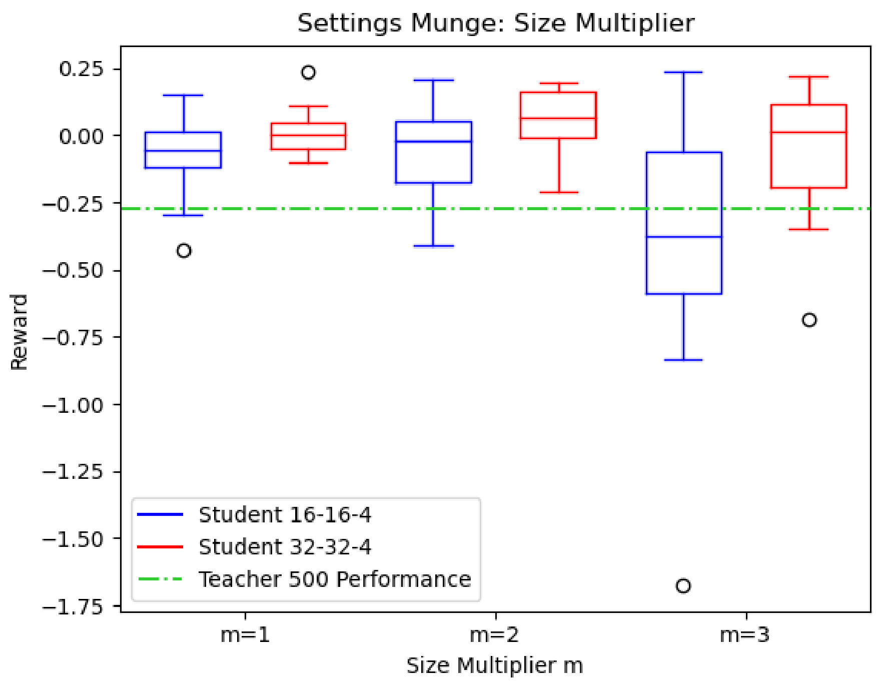

Figure 5.5: Experiment using different size multipliers m for the Munge algorithm with $v = 0.2$ and $p = 50\%$.

In this experiment it is more difficult to determine the best parameter. On the one hand the results are closer to each other and thus more reproducible with a size multiplier of one, but on the other hand, with $m = 2$ a higher median is obtained. If the size multiplier is increased further, the results are even more scattered and the median also deteriorates. For further tests, a size multiplier of two is selected, since it can be expected that the scattering of the results will be reduced even further with a larger data set. This assumption was made based on the results in subsection 5.1.1, which show that with a larger data set, the results of the trained students become more similar.
With the help of the previous experiments good values for the local variance and the size multiplier have been found, these are now used to find a suitable value for the swap probability. If a low swap probability is used, the generated data set is very similar to the pure simulation data and the original data is changed only very little. Due to this fact, the assumption can be made that a very low probability leads to worse rewards, which can also be seen in the graph in figure 5.6. However, if a very high swap probability is used, almost all data points are changed and only a few training samples correspond to the simulation data, which may also lead to worse results. This is due to the fact that the pure simulation data is important for the training process. Only

a few additional deflections of the drone should be included in the data set to provide additional stability while still training the basic movements of the flight maneuver.

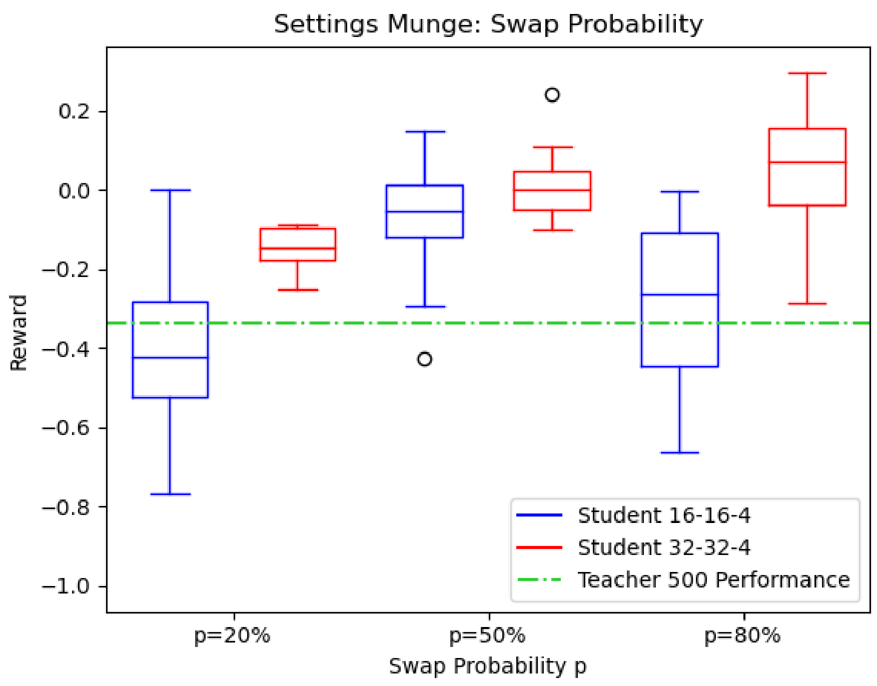

Figure 5.6: Experiment using different swap probabilities p for the Munge algorithm with $v = 0.2$ and $m = 2$.

Looking at the results, it can be seen that a probability of 50% yields solid results. Of course, not every combination of parameters was tested with this manual search and it is possible to find a even better combination. However, for the following experiments these values, $v = 0.2$, $p = 50\%$ and $m = 2$, are used to compare this method to the others.

5.2 Comparison of Data Generation Methods

In this chapter, the different methods for data generation are compared. The best method is the one that creates student models that achieve the highest rewards. For this experiment the data set size "large" was used. In section 5.1.1 it was shown that this size achieves well evaluable results without an extensive training duration using most of the data generation methods.

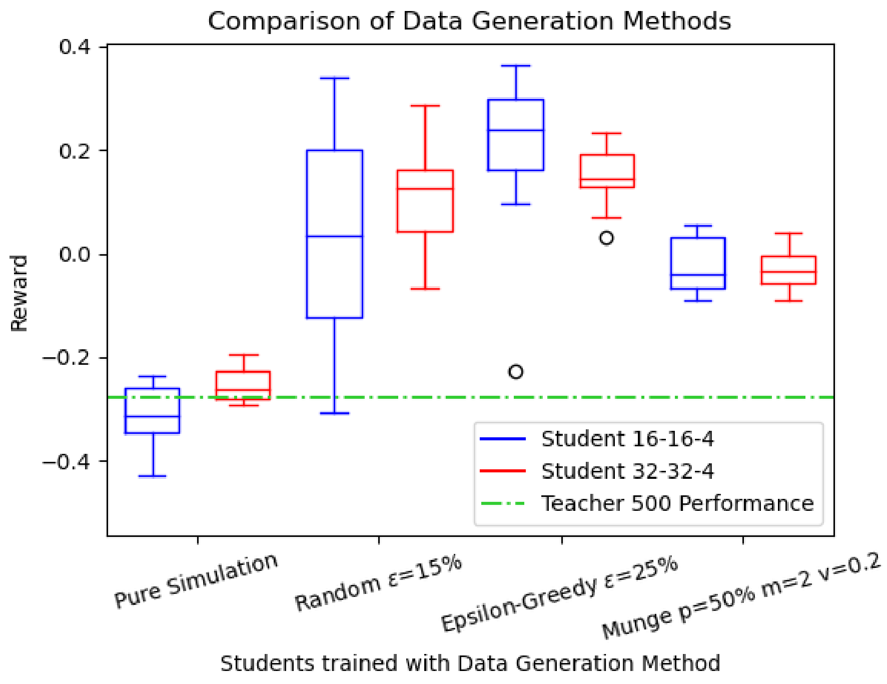

Figure 5.7: Comparison of the achieved rewards of students trained with different data generation methods.

In figure 5.7 the performance of the student models is visualized, which were trained with different data sets. The settings determined in the previous section were used to generate the training data. Again, 10 student models each were trained with the same data set and a patience of 40 was used. To make the comparison meaningful, some student models were trained with pure simulation data. This data set generated student models that achieved approximately the same reward as their teacher. Using the Random Data Generation method with a probability factor of $\epsilon = 15\%$ resulted in even better models that achieved higher rewards as their teacher. The median of the two architectures trained with the random data set was significantly higher than the median of the pure simulation data. However, the results of the students varied a lot. This means that the reproducibility of the results was rather low, which is evident from the large boxes. The third experiment shown in the graph are the student models trained with data from the Epsilon-Greedy Data Generation method. The previously determined probability value of $\epsilon = 25\%$ was used. With the help of this data set even better models were trained and at the same time the results were much more reproducible than when using data of the Random Data Generation method. The last boxes show the performance of a data set created with the Munge Data Generation

method. The obtained rewards of the trained models were significantly better than those of the models trained with the pure simulation data, but at the same time worse than those trained using the Epsilon-Greedy data set. In addition, generating the data points took a lot of time. Looking at the results of the different methods, it is clear that each of the approaches studied represents an improvement over the pure simulation data. The student models trained with the different data generation methods outperform the teacher. This seems very surprising at first and needs an explanation. In most cases where KD has been used in previous research, the teacher still achieves higher rewards than the student ([29], [40], [42], [46]). However, some experiments have been conducted in which the student performs better ([15], [27], [33], [41]). So far, there is a lack of studies which fully describe this phenomenon. However, in this work we try to find an possible explanation.
A model with the same architecture as the students, for example with two layers and 32 neurons each, trained using only RL, achieved a reward of about -0.3. This result was determined by averaging the achieved rewards of 500 trajectories and it shows that KD produces superior models, which are better than the models currently trained directly using RL within the project. This phenomenon was already tried to be explained by Ba et al. and can be read in section 2.2. Since in this experiment no additional knowledge was transferred from the teacher to the student except the outputs of the teacher, this can be neglected as possible reason. However, the problem that the teacher tries to learn with the help of RL is much more complex than the provided input-output pairs that the student has to learn. This may explain the better performance of the student compared to a model with the same size trained with RL. But, this still does not explain why the student outperforms its teacher. One possible reason is that the student has learned to generalize better and therefore achieves higher rewards.
When comparing the different data generation methods, we can see clear differences in performance. If we analyze the student models of the Random Data Generation method and compare them to the Epsilon-Greedy method, we can see that with Epsilon-Greedy data students can be trained that achieve more similar and at the same time better rewards. This can also be explained by the fact that with the Random Data Generation approach the inputs are chosen randomly to a certain extent, whereas with the Epsilon-Greedy approach the outputs are determined randomly. Since there are 16 different input data, they are chosen from a 16-dimensional space, whereas the space for the outputs is only 4-dimensional. This makes it more likely that the random chosen outputs are closer to the actual distribution of the data that can occur in the simulation than the inputs, since the 16-dimensional space allows much more leeway in the randomly chosen samples. Presumably, the input data that can occur in the simulation and also in reality lie within some sort of hose in this 16-dimensional space, but this may not be reflected very well by the additional data points created with the used Random Data Generation method. However, if we look at the models that were generated using the Munge Data Generation method, we see that the dispersion of the rewards of the individual models is very low. Similarly low as for the models

trained with pure simulation data. The performance of the students trained with data generated by the Munge Data Generation method is better than the performance of models trained with pure simulation data, but still worse than the performance when using other methods. This is probably because this approach generates very similar data points to the data generated by the Pure Simulation Data Generation method and the additional data points are very likely to lie within the before mentioned hose. This means the additional data points lie very close to the simulation data points within the hose and reflect the true data distribution but they do not capture the whole hose very well. Therefore, these samples provide some variation, which improves the results, but the variations are not as diverse as in the Epsilon-Greedy approach for example. This may create the difference in the achieved rewards of the student models.

5.3 Comparison of different Teacher Sizes

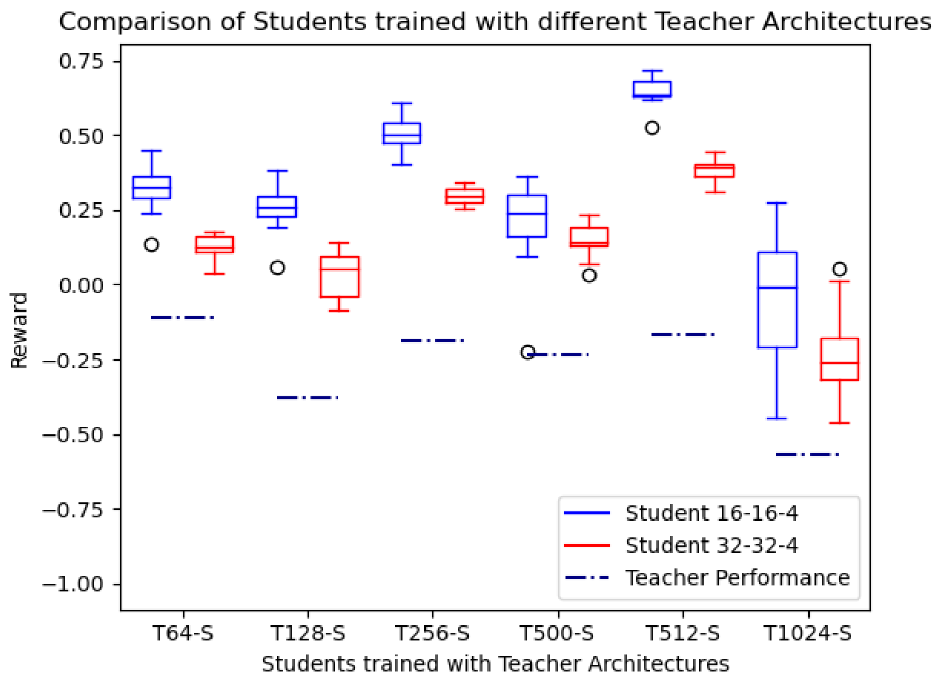

Figure 5.8: The performance of students trained with different teacher sizes is compared. The architecture of the teachers consists of two layers with 64, 128, 256, 500, 512 or 1024 neurons each.

In the course of this thesis, further teacher models were trained in different sizes within the project. This section investigates which size is able to train the best students. The Epsilon-Greedy Data Generation method was used, since it achieved the best results in the previous section.
Figure 5.8 shows the results of the experiment. The student architectures trained consist of two layers with 32 neurons each and 16 neurons each. The size of the teacher used for the training is indicated as label on the x-axis. The achieved reward of the teacher is shown as a dashed line and the performance of the 10 students trained is visualized using a boxplot. This experiment used a patience of 40 and 500 trajectories were evaluated to determine the rewards. If one looks at the achieved rewards of the teachers with different sizes, one can already see differences. The situation is similar for the trained students.
It is striking that the largest teacher with 1024 neurons achieved the worst reward and furthermore, produced the worst performing students. It seems like this teacher could not be trained sufficiently with RL and the simulation. Although some papers state that larger models with more parameters usually produce better results ([21], [35], [37]), this does not seem to be the case in the problem at hand. At the very least, the architecture used is unfavorable to other architectures with fewer neurons when training with RL. Consequently, students trained with this teacher perform significantly worse than students trained with other teacher sizes. Another reason why these students perform worse, could be that the teacher has to many parameters. Cho et al. showed that large models with many parameters often perform worse as teachers than smaller ones, even if they themselves achieve a higher accuracy than a smaller teacher architecture [8]. That is, regardless of the achieved reward of the large model, it can be a poor teacher from which the student cannot learn successfully.
If we look at the other teachers and their respective students, we see that the teacher with 64 neurons in each layer achieved the best reward but did not train the best students. This means that the teacher's performance is not directly related to the performance of the students that are trained with the help of this teacher. The teacher with 512 neurons trained the best students in this experiment, and the student architecture with two layers of 16 neurons each performed particularly well. Both the teacher itself and the students achieved very good rewards.
In summary, this section shows that there are variations in the performance of students trained by different teachers in different sizes. The best teacher with the highest rewards does not necessarily train the best students. Similar observations have been made by Cho et al. [8].

5.4 Teacher Assistant Training

In this section, the experiments performed with the help of a Teacher Assistant (TA) are described in more detail. The size of the teacher is, as stated in section 4.2, two

layers each consisting of 500 neurons and the student models that are trained have a size of two layers each with 32 neurons. Since the other teacher models with different sizes were only created and evaluated in the course of the thesis, the model with 500 neurons is still used in the following section, although it was shown in the previous experiment that significantly better students can be generated using the model with 512 neurons as teacher.
As described in section 2.1.3, at first the teacher trains a TA which is then used to train a student model. The settings used in the following experiments are again a patience of 40 and 500 test trajectories to evaluate the rewards. The Epsilon-Greedy Data Generation method was used to generate the training data, since it performed best in section 5.2.

5.4.1 Single Teacher Assistant

Different architectures of TAs were used to find a size that trains well-performing students. First, 10 different TAs were trained in different sizes. The best TA was then used to train 10 students.

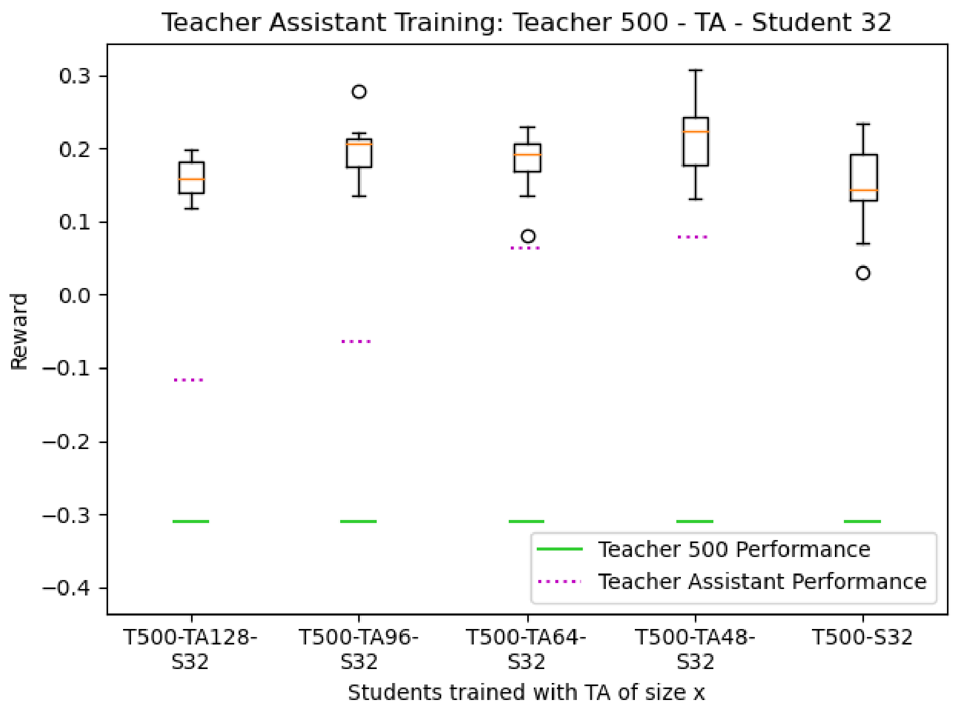

Figure 5.9: Comparison of the student performance trained with different TA architectures using Epsilon-Greedy Data Generation method with $\epsilon = 25\%$.

In figure 5.9, one can see a boxplot of the 10 different students that were trained using a single TA. The achieved reward of the corresponding TA is drawn beneath the student performance as dotted line in the diagram. The size of the TA is visible as a label at the bottom. The achieved reward of the TA, which is shown in the graph, corresponds to the achieved reward of the best TA out of the 10 that were trained. On the far right one can see the performance of the students which were trained directly by the teacher with 500 neurons per layer. In comparison, the other boxplots show the rewards of the students trained with the corresponding TAs. These also have an architecture consisting of two layers but with a varying number of neurons [128, 96, 64, 48]. The solid line visualizes the performance of the teacher, which was used to train the TA. The median of all student models trained with a TA is better than that of the student models trained directly by the teacher. The graph shows that all TAs achieved higher rewards than the teacher shown in green. If we now compare the different architectures of the TAs, we see that two layers each with 48 neurons trained the best students which achieved the highest rewards. These are, as already mentioned, better than the achieved rewards of the students which were trained directly using the teacher. Similar to the experiments of Mirzadeh et al. the number of neurons of the best TA is not the average between teacher and student size [28]. Looking at the results, we can conclude that the use of a TA definitely improves the final rewards of the students. In our experiment, a smaller TA with 48 neurons in each layer achieved higher performing student models than those with more neurons.

5.4.2 Comparison of Teacher Assistant Training and Vanilla-KD

This subsection investigates whether the use of a TA is beneficial even if the teacher is very large, or whether it is better to use a smaller teacher and perform KD without a TA in this experimental setting. In this experiment we first used a large teacher architecture with two layers each consisting of 1024 neurons and trained 10 TAs with 500 neurons in each layer. The best TA was further used to train 10 students with two layers each consisting of 32 neurons. Afterwards, the results were compared with students trained directly using a smaller teacher with 500 neurons in each layer.
Figure 5.10 visualizes the results of the experiment. The large teacher with 1024 neurons per layer trained TAs that achieved similar results than the teacher itself. The best TA was slightly better than the large teacher but worse than the teacher with 500 neurons. The best trained TA was then used to train students. These students finally achieved better results than the teacher and also than the TA. However, the rewards achieved by the 10 students were widely spread and lower than the rewards of the students trained with the smaller teacher. One possible reason for the better results that can be achieved with the help of the smaller teacher, is that the teacher itself already achieves a better reward than both, the large teacher with 1024 neurons and its TA. Moreover, we have already found in subsection 5.3 that the large teacher with

1024 neurons trains the worst students compared to the other teacher architectures that were investigated.

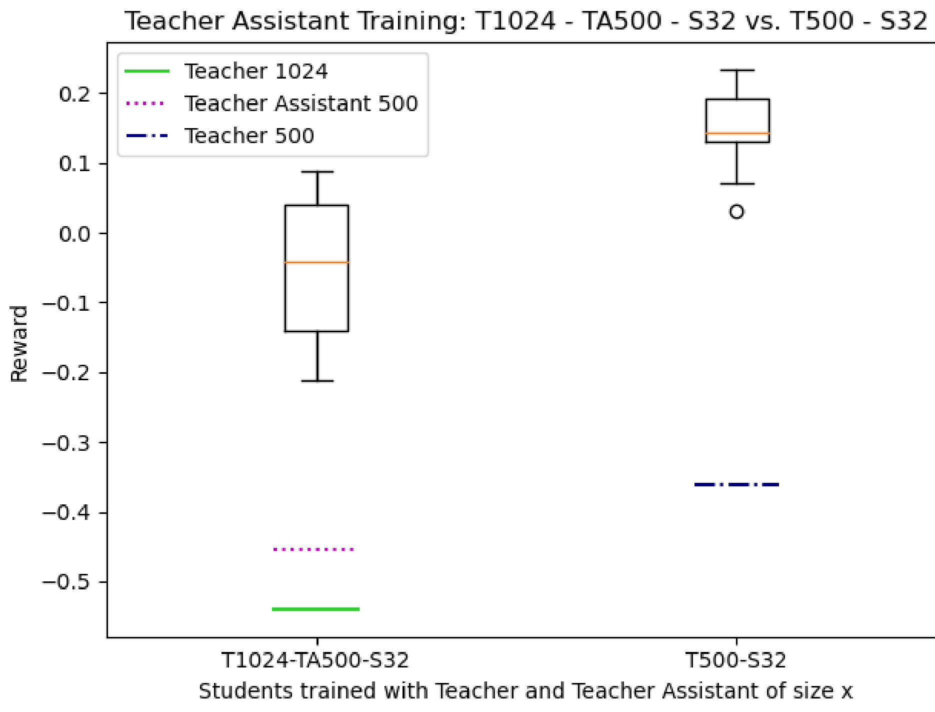

Figure 5.10: Students trained using a teacher with 1024 neurons and a TA with 500 neurons compared to students trained using only a teacher with 500 neurons.

In this experimental setup it is therefore not reasonable to use a larger teacher, unless the teacher himself achieves a reward that is much better than that of the smaller teachers which were trained. In such a case, the results should be re-evaluated. If one compares the students that were trained by the large teacher with 1024 neurons and a TA with those that were trained directly by the large teacher (section 5.8), one can recognize the positive influence of a TA here as well. Just as in the previous section, training with a TA improves the achieved rewards of the students, even though the TA does not represent a major increase in performance compared to the teacher.

5.4.3 Two Teacher Assistants

In this section, we investigate whether multiple TAs further improve the results. Since the TAs with an architecture of two layers each with 96 neurons and 48 neurons achieved good results in the previous section, these architectures were used as TAs in this experiment. The teacher with two layers and 500 neurons trained 10 TAs with 96 neurons. The best of these TAs was used to train 10 smaller TAs which have an architecture of two layers each with 48 neurons. Afterwards, the best TA with this architecture then trained 10 students.

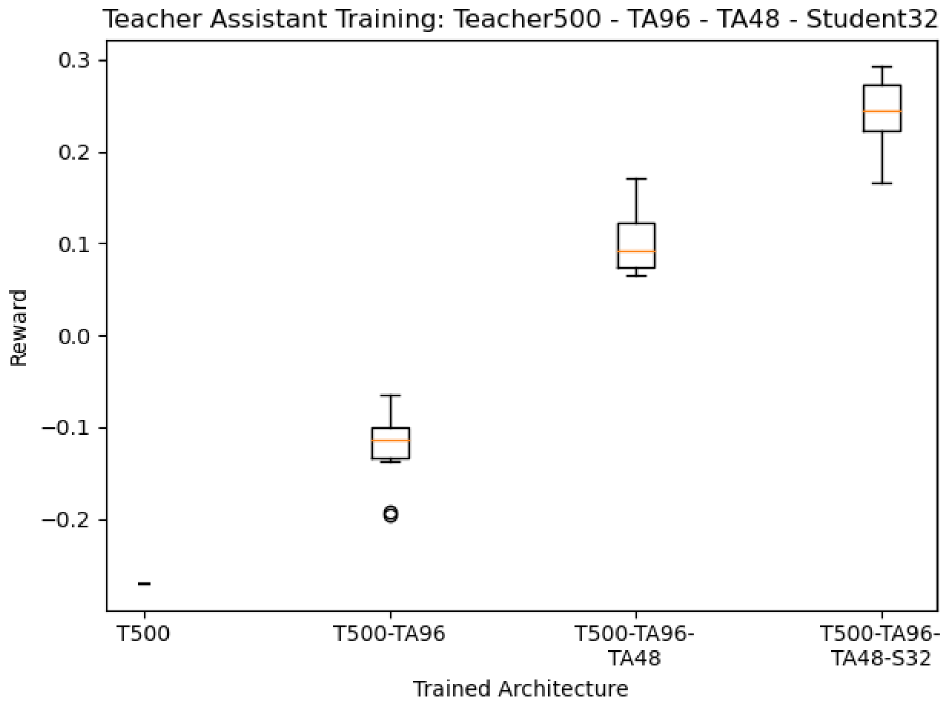

Figure 5.11: Training students with the help of two TAs. Visualized is the performance of the teacher, TAs and students.

Figure 5.11 shows the performances of teacher, TAs and the trained students. It is evident that the performance improved with each step of the experiment.
The graph in figure 5.12 visualizes the different performances that students achieved when they were trained with one or two TAs and compares this to students that were trained without any TA. In the graph on the right the achieved rewards of the students are visualized which were trained without a TA. In the middle the results of the students trained with a single TA are shown. The respective architectures of teacher and TA as well as the student architecture is indicated as label on the x-axis. On the left the performance of the students trained with two TAs is shown. It is clearly visible that the additional TA improved the achieved rewards even more. These students outperformed the conventional method without a TA, as well as the students trained

with only one TA. This means that the rewards achieved by these students that were trained with two TAs have a higher median than the rewards of the other students.

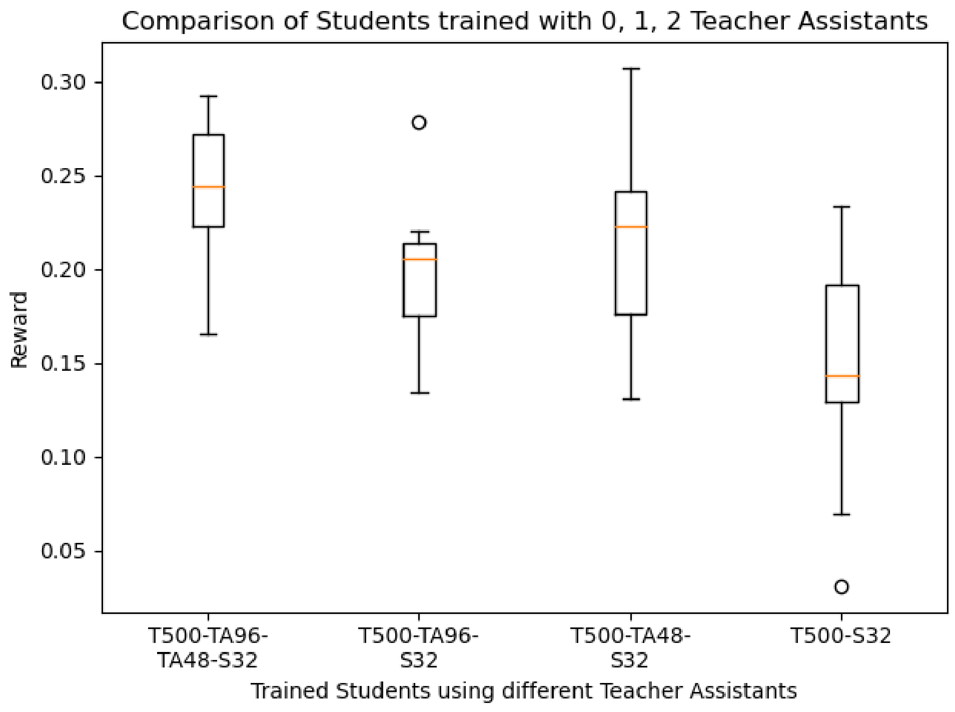

Figure 5.12: Comparison of students trained with one or two TAs and students trained without a TA.

Mirzadeh et al. came to the conclusion that each TA improves the performance further, even if from a certain number of TAs only small improvements are visible [28]. This experimental setup confirms their results and more TAs do improve the achieved reward compared to a single TA or the direct training without one.

5.4.4 Sequential Knowledge Distillation

Previous experiments have shown that each trained TA improves the results of the corresponding students. Therefore, this subsection investigates whether sequential knowledge distillation also achieves performance improvements. The Born Again Neural Networks presented and studied by Furlanello et al. represent a similar approach [15].
For this experiment, a teacher was used which has an architecture with two layers and 32 neurons. This teacher then trained TAs or students with the same architecture,

which in turn trained other models with a similar architecture. This experiment differs from the experiments before because the TAs have the same size as the teacher and the student model. In each training step 10 models are trained from which the best model is used to train 10 models of the next TA or student generation. As in the previous experiments, the data generated by the Epsilon-Greedy Data Generation method was used as training data.

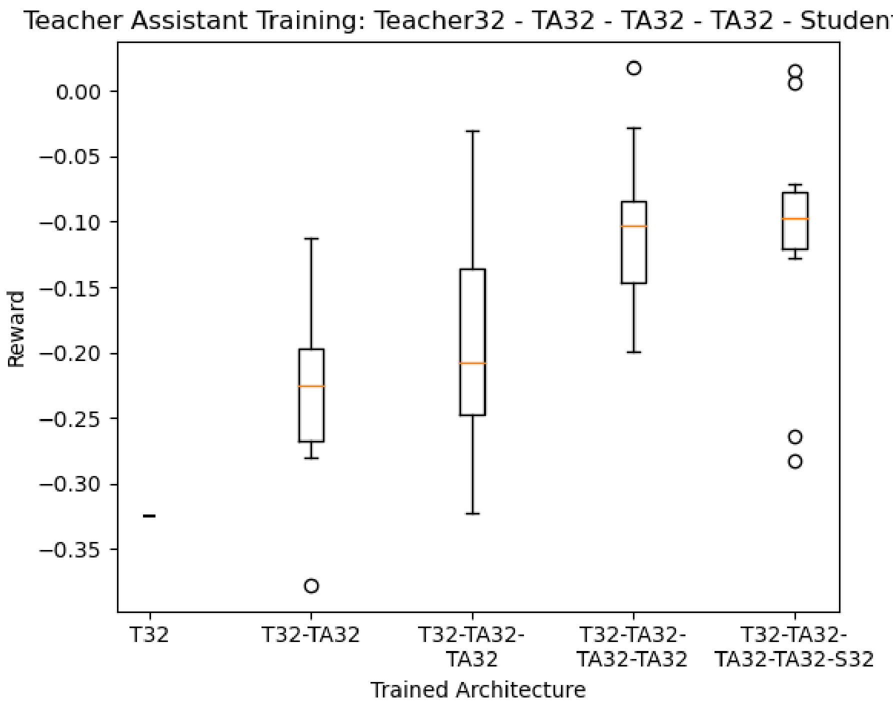

Figure 5.13: Knowledge distillation applied to models with the same architecture. The shown TAs and students were trained with the best trained model on their left side.

The line on the left side of figure 5.13 shows the performance of the teacher, which was used to train the TAs visualized on its right. The best TA was then used to train the next generation of TAs and so on. As in the experiments before, it is evident that the TAs and students perform better than the teacher. Furlanello et al. also reported similar results, where in most cases the students outperform their teacher [15]. A research work based on the Born Again Networks also confirms these results and states that in the first generations, the obtained results are improved but at some point they remain the same [41].
Looking at the results shown in figure 5.13, we can see that compared to the original teacher, the results are slightly improved in each step. However, it is visible that the

improvement becomes smaller from step to step which could be evidence for a saturation that is mentioned by Yang et al. [41].
This experiment does not aim at compressing the models, rather teacher and students have the same architecture. Thus, the improvements achieved cannot be explained by the different network architecture of teacher and student. Since in this experimental setup no additional knowledge of the teacher is transferred to the student and only the outputs are adjusted, this explanation for the improved performance of the students is also not meaningful. Rather, this experiment shows that the training data used is crucial for the good rewards of the students and could be the reason why the students in this thesis outperform their teacher.

6 Conclusion

In this thesis, the first part investigated which method for data generation achieves the best results in KD. Different approaches were examined and it was shown that the Epsilon-Greedy Data Generation method trains the best students. This simple approach uses random actions to deflect the drone in the simulation. Using a certain amount of random actions generates a diverse data set that is able to train students with a good performance. Surprisingly, the trained students are even better than their teacher. This is probably mainly due to the training data set, since the pure simulation data produces students that achieve a comparable performance to their teacher. Furthermore, it is possible that a smaller architecture, similar to that of the students, is beneficial for this problem setting. This could also be a reason why, even when using the pure simulation data, the students perform about as well as the teacher without an decrease in the achieved rewards. On the other hand, RL was also used to train a model with two layers and 32 neurons, the same architecture as the students, and this model achieved a reward of about -0.3 which is roughly equivalent to the teacher with 500 neurons and worse than its trained students. The experiment with sequential knowledge distillation also shows that improvements can be achieved even with the same architecture. This observation suggests that the data set is the key reason why the students perform better than their teacher.
Furthermore, the thesis investigated whether a certain teacher architecture achieves better results in KD than others. Since these networks with different architectures were only available in the course of the thesis, most of the experiments were performed with the help of a teacher with two layers and 500 neurons each. Already, the teachers themselves, achieve different rewards and thus also produce students that perform differently. Surprisingly, the best teacher does not necessarily generate the best students. However, more precise assumptions about the best architecture for teacher and student cannot be made solely with one experiment. It only shows that of the available teachers, the one with 512 neurons is able to train the best students. Specifically, students with 16 neurons in each layer perform even better than those with 32.
In the last section of the paper, it was examined whether a TA can further positively influence the training. The first step was to investigate to what extent a single TA influences the performance of the students. Already a single TA could confirm the positive influence independent of its size. Adding a second TA improved the results even more. However, in another experiment it was shown that it makes more sense to think about the optimal teacher size instead of just adding a TA. For example, if a

very large teacher is used, which, as in our case, also achieves worse rewards, and a TA is added, the trained students achieve significantly worse results than students trained directly with a better fitting teacher. One experiment shows the results that can be obtained with the same architecture of teacher, TA and student but with multiple distillation steps. In the experiment, TAs were trained three times and afterwards the corresponding students. The results show that even with the same architecture, the performance of TAs and students become better and better per distillation step. From this experiment, it can be seen that the use of a TA can be worthwhile even if there is no difference in size between teacher and student.

Currently, several other research questions regarding KD are investigated in the Phoenix-Project, which among other things also examine a proper network architecture. In the future it would be interesting to investigate whether the optimal architecture, trained using RL, performs better than a trained RL model in combination with KD. In addition, it should be investigated whether other architectures of students achieve better results and which combination of teacher and student architecture performs best. Shin et al. showed that students with more layers may achieve better results, as they are often better able to generalize the knowledge they have learned [34]. According to them, this is due to the architecture of a deep network being able to handle complicated features.
In this work, the settings for the methods were determined by a manual search. In this approach, only a few parameters were tested. It is possible that even better parameters can be found using a different hyperparameter search. In subsequent investigations, the parameter search can be extended to achieve even better results. Further work could also investigate whether a data set consisting of different flight maneuvers yields even better results than the Epsilon-Greedy Data Generation method. With a mixed data set of several flight maneuvers such as hovering, flying in circles, flying up and down and so on, a very diverse data set can be created, which could train models that are able to compensate well for different deflections of the drone.

Bibliography

[1] C. Alippi, S. Disabato, and M. Roveri. “Moving Convolutional Neural Networks to Embedded Systems: The AlexNet and VGG-16 Case”. In: *17th ACM/IEEE International Conference on Information Processing in Sensor Networks*. 2018, pp. 212–223.

[2] Andrei A. Rusu, Sergio Gomez Colmenarejo, Çaglar Gülçehre, G. Desjardins, J. Kirkpatrick, Razvan Pascanu, V. Mnih, K. Kavukcuoglu, and Raia Hadsell. “Policy Distillation”. In: *4th International Conference on Learning Representations* (2016).

[3] R. Anil, G. Pereyra, A. Passos, R. Ormándi, G. E. Dahl, and G. E. Hinton. “Large scale distributed neural network training through online distillation”. In: *6th International Conference on Learning Representations*. 2018.

[4] L. J. Ba and R. Caruana. “Do deep nets really need to be deep?” In: *Advances in Neural Information Processing Systems* 3 (2014), pp. 2654–2662.

[5] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. “OpenAI Gym”. In: *arXiv preprint arXiv:1606.01540* (2016).

[6] C. Bucila, R. Caruana, and A. Niculescu-Mizil. “Model compression”. In: *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2006, pp. 535–541.

[7] Y. Cheng, D. Wang, P. Zhou, and T. Zhang. “Model Compression and Acceleration for Deep Neural Networks: The Principles, Progress, and Challenges”. In: *IEEE Signal Processing Magazine* (1) (2018), pp. 126–136.

[8] J. H. Cho and B. Hariharan. “On the Efficacy of Knowledge Distillation”. In: *IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 4793–4801.

[9] T. Choudhary, V. Mishra, A. Goswami, and J. Sarangapani. “A comprehensive survey on model compression and acceleration”. In: *Artificial Intelligence Review* 53(7) (2020), pp. 5113–5155.

[10] D. Ciresan, U. Meier, and J. Schmidhuber. “Multi-column deep neural networks for image classification”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012, pp. 3642–3649.

[11] R. Collobert and J. Weston. “A unified architecture for natural language processing”. In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 160–167.

[12] A. P. Dempster, N. M. Laird, and D. B. Rubin. “Maximum Likelihood from Incomplete Data Via the EM Algorithm”. In: *Journal of the Royal Statistical Society: Series B (Methodological)* (1) (1977), pp. 1–22.

[13] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov. “Scalable Object Detection Using Deep Neural Networks”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 2155–2162.

[14] S. Feng, H. Zhou, and H. Dong. “Using deep neural network with small dataset to predict material defects”. In: *Materials & Design* 162 (2019), pp. 300–310.

[15] T. Furlanello, Z. C. Lipton, M. Tschannen, L. Itti, and A. Anandkumar. “Born Again Neural Networks”. In: *International Conference on Machine Learning* (2018), pp. 1607–1616.

[16] W. Giernacki, M. Skwierczynski, W. Witwicki, P. Wronski, and P. Kozierski. “Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering”. In: *22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*. 2017, pp. 37–42.

[17] J. Gou, B. Yu, S. J. Maybank, and D. Tao. “Knowledge Distillation: A Survey”. In: *International Journal of Computer Vision* (2021), pp. 1–31.

[18] G. Hinton, O. Vinyals, and J. Dean. “Distilling the Knowledge in a Neural Network”. In: *NIPS Deep Learning and Representation Learning Workshop*. 2015.

[19] K. Hornik. “Approximation capabilities of multilayer feedforward networks”. In: *Neural Networks* 4(2) (1991), pp. 251–257.

[20] K. Hornik, M. Stinchcombe, and H. White. “Multilayer feedforward networks are universal approximators”. In: *Neural Networks* 2(5) (1989), pp. 359–366.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton. “ImageNet classification with deep convolutional neural networks”. In: *Communications of the ACM* 60(6) (2017), pp. 84–90.

[22] X. Lan, X. Zhu, and S. Gong. “Self-Referenced Deep Learning”. In: *Computer Vision – ACCV*. 2018, pp. 284–300.

[23] Y. LeCun, J. S. Denker, and S. A. Solla. “Optimal Brain Damage”. In: *Advances in Neural Information Processing Systems 2* 2 (1990), pp. 598–605.

[24] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken. “Multilayer feedforward networks with a nonpolynomial activation function can approximate any function”. In: *Neural Networks* 6(6) (1993), pp. 861–867.

[25] R. Lindsey, A. Daluiski, S. Chopra, A. Lachapelle, M. Mozer, S. Sicular, D. Hanel, M. Gardner, A. Gupta, R. Hotchkiss, and H. Potter. “Deep neural network improves fracture detection by clinicians”. In: *Proceedings of the National Academy of Sciences* 115(45) (2018), pp. 11591–11596.

[26] D. Lowd and P. Domingos. “Naive Bayes Models for Probability Estimation”. In: *Proceedings of the 22nd International Conference on Machine Learning*. 2005, pp. 529–536.

[27] P. Luo, Z. Zhu, Z. Liu, X. Wang, and X. Tang. “Face Model Compression by Distilling Knowledge from Neurons”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 30(1) (2016).

[28] S. I. Mirzadeh, M. Farajtabar, A. Li, N. Levine, A. Matsukawa, and H. Ghasemzadeh. “Improved Knowledge Distillation via Teacher Assistant”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34(04) (2020), pp. 5191–5198.

[29] G. K. Nayak, K. R. Mopuri, V. Shaj, R. V. Babu, and A. Chakraborty. “Zero-Shot Knowledge Distillation in Deep Networks”. In: *International Conference on Machine Learning* (2019), pp. 4743–4751.

[30] J. Panerati, H. Zheng, S. Zhou, J. Xu, A. Prorok, and A. P. Schoellig. “Learning to Fly – a Gym Environment with PyBullet Physics for Reinforcement Learning of Multi-agent Quadcopter Control”. In: *arXiv e-prints arXiv:2103.02142v3* (2021).

[31] W. Park, D. Kim, Y. Lu, and M. Cho. “Relational Knowledge Distillation”. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 3962–3971.

[32] W. Rawat and Z. Wang. “Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review”. In: *Neural Computation* (9) (2017), pp. 2352–2449.

[33] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. “FitNets: Hints for Thin Deep Nets”. In: *3rd International Conference on Learning Representations* (2015).

[34] I.-H. Shin, Y.-H. Moon, and Y.-J. Lee. “Towards Understanding Architectural Effects on Knowledge Distillation”. In: *International Conference on Information and Communication Technology Convergence (ICTC)*. 2020, pp. 1144–1146.

[35] K. Simonyan and A. Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *3rd International Conference on Learning Representations*. 2015.

[36] R. S. Sutton and A. Barto. *Reinforcement learning. An introduction*. Second edition. Adaptive computation and machine learning. The MIT-Press, 2018. 526 pp.

[37] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. “Going deeper with convolutions”. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 1–9.

[38] M. Takamoto, Y. Morishita, and H. Imaoka. “An Efficient Method of Training Small Models for Regression Problems with Knowledge Distillation”. In: *IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*. 2020, pp. 67–72.

[39] S. Tobiyama, Y. Yamaguchi, H. Shimada, T. Ikuse, and T. Yagi. “Malware Detection with Deep Neural Network Using Process Behavior”. In: *IEEE 40th Annual Computer Software and Applications Conference*. 2016, pp. 577–582.

[40] M.-C. Wu, C.-T. Chiu, and K.-H. Wu. “Multi-teacher Knowledge Distillation for Compressed Video Action Recognition on Deep Neural Networks”. In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2019, pp. 2202–2206.

[41] C. Yang, L. Xie, S. Qiao, and A. L. Yuille. “Training Deep Neural Networks in Generations: A More Tolerant Teacher Educates Better Students”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33(01) (2019), pp. 5628–5635.

[42] Z. Yang, L. Shou, M. Gong, W. Lin, and D. Jiang. “Model Compression with Two-stage Multi-teacher Knowledge Distillation for Web Question Answering System”. In: *Proceedings of the 13th International Conference on Web Search and Data Mining*. 2020, pp. 690–698.

[43] J. Yim, D. Joo, J. Bae, and J. Kim. “A Gift from Knowledge Distillation: Fast Optimization, Network Minimization and Transfer Learning”. In: *30th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 7130–7138.

[44] T. Young, D. Hazarika, S. Poria, and E. Cambria. “Recent Trends in Deep Learning Based Natural Language Processing”. In: *IEEE Computational Intelligence Magazine* 13(3) (2018), pp. 55–75.

[45] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen. “Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights”. In: *5th International Conference on Learning Representations* (2017).

[46] Y. Zhou, X. Gu, R. Fu, N. Li, X. Du, and P. Kuang. “Effective Knowledge Distillation for Human Pose Estimation”. In: *16th International Computer Conference on Wavelet Active Media Technology and Information Processing*. 2019, pp. 170–173.