UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Computer Science Curriculum

Marti Ingmar Liibert

In Search of the Best Activation Function

Master's Thesis (30 ECTS)

Supervisor(s): Tambet Matiisen, MSc

Tartu 2022

In Search of the Best Activation Function

Abstract: The choice of an activation function in neural networks can have great consequences on the performance of the network. Designing and discovering new activation functions that increase the performance or solve problems of existing activation functions is an active research field. In this thesis, a kind of trainable activation function is proposed - a weighted linear combination of activation functions where the weights are normalized using Softmax, inspired by the DARTS network architecture search method. The activation function is applied at the layer, kernel, and neuron levels. Optimizing the activation function weights is done on training loss and validation loss, as was done in DARTS. The activation function here was tested on two simple datasets, sine wave, and spiral datasets, on image classification tasks and on a robotics task. In the case of image classification, on CIFAR10 using the trainable activation function for initial training the accuracy increased 5% over the baseline, on ImageNet the accuracy increased 1% over the baseline. For the robotics task, CartPole, the mean reward increased by 10 points out of a maximum of 200 when using the already learned activation functions in the case of Deep Q-learning. In the case of Proximal Policy Optimization, the mean reward increased by 2 points approximately over the baseline. For future work, more difficult tasks could be explored for robotics tasks and longer initial search could be explored for image classification tasks.

Keywords: Activation function, trainable activation function, artificial neural network, image classification, reinforcement learning, robotics, CIFAR10, ImageNet, CartPole

CERCS: P170 Computer science, numerical analysis, systems, control; P176 Artificial intelligence

Parima aktivatsioonifunktsiooni otsingul

Lühikokkuvõte: Aktivatsioonifunktsiooni valik tehisnärvivõrkudes tihti mõjutab mudeli sooritusvõimet. Uute aktivatsioonifunktsioonide disainimine ja avastamine, et parandada närvivõrkude sooritust, on aktiivne uurimisvaldkond. Siin töös esitletakse ja testitakse kindlat treenitavat aktivatsioonifunktsiooni - kaalutud lineaarset kombinatsiooni erinevatest aktivatsioonifunktsioonidest, kus kaalud on normaliseeritud kasutades Softmax funktsiooni, mis on inspireeritud DARTS närvivõrkude arhitektuuri otsingu meetodist. Treenitavat aktivatsioonifunktsiooni testitakse närvivõrgu kihi, kerneli ja neuroni tasemel. Aktivatsioonifunktsiooni kaale optimiseeritakse kas treeningkao või validatsioonikao põhjal DARTS-i eeskujul. Treenitavat aktivatsioonifunktsiooni kasutati siin töös kahe lihtsa funktsiooni, siinuse ja spiraali, õppimiseks, piltide klassifitseerimiseks ning ühe robootika ülesande lahendamiseks. Piltide klassifitseerimisel saavutas mudel treenitavate aktivatsioonifunktsioonide kasutamisel CIFAR10 andmestikul 5% suurema täpsuse baasmudeli täpsusest ja ImageNet andmestikul 1% suurema täpsuse baasmudeli täpsusest. Robootika ülesande, CartPole, korral suurenes keskmine preemia umbes 10 punkti võrra 200 punktisest maksimumist kui kasutati juba õpitud aktivatsioonifunktsioone uuesti treenimiseks *Deep Q-learning* algoritmi puhul. *Proximal Policy Optimization* algoritmi puhul suurenes keskmine preemia umbes 2 punkti võrra. Tulevikus saaks uurida selle meetodi rakendamist keerukamatele ülesannetele robootika valdkonnas ning pikemat otsingut piltide klassifitseerimise ülesande puhul.

Võtmesõnad: Aktivatsioonifunktsioon, treenitav aktivatsioonifunktsioon, tehisnärvivõrk, piltide klassifitseerimine, stiimulõpe, robootika, CIFAR10, ImageNet, CartPole

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine; P176 Tehisintellekt

Contents

1 Introduction

Activation functions are a central part of neural networks. Choosing the right activation function can have great consequences on the performance of neural networks. Designing and discovering new activation functions that improve the results of learning is an open field of research. For example, in 2017, the Swish activation function was discovered using a specific search method [RZL17]. Swish increased the performance of image classification tasks and allowed to train deeper networks.

Finding or choosing better activation functions is needed because there are often problems with most used activation functions and the default used activation function might not be optimal. For example, tanh and sigmoid cause vanishing gradients, and ReLUs have the "dying ReLU" problem. ReLU is the most popular activation function and is often used as the default for all tasks, however, for example, in the case of robotics tasks often the most optimal function is tanh. It may be the case that when solving a new problem with neural networks, for example, protein folding, a suboptimal activation function might be used without investigating further.

In this thesis, a kind of trainable activation function is explored - a weighted linear combination of different basic activation functions, where the weights are normalized using Softmax. The motivation comes from the idea of using differentiable architecture search (DARTS) [LSY18] to choose between different network branches. In the case of certain simple functions that we try to learn, it is easy to see that choosing the right activation function helps us learn the function faster and using a smaller network [Mat18]. It may be the case that for more complex functions we can also choose optimal activation functions. The goal was to find out if using this kind of activation function increases the performance of neural networks and if it is possible to choose better activation functions for the network and the problem at hand based on the learned weights. The designed trainable activation function was applied for each layer, for each kernel where applicable, or for each neuron. It was also explored if training the weights of the activation function on the training set or the validation set improved the results more, as was done in DARTS and is done more generally in the case of hyperparameter optimization.

There have been previous works linearly combining activation functions to get trainable activation function units [SBF+18], but in addition to previous approaches that test the trainable activation function on one level, here the trainable activation functions are tested and analyzed on the layer, kernel, and neuron level. In previous works, similar mechanisms have been tested mainly on image classification tasks but here it is also examined how they behave on simpler functions and also on robotics tasks. Additionally, DARTS-like optimization of the activation weights on validation loss and deriving and testing the final architecture is also explored.

The most important results of this thesis are that for image classification using the trainable activation function the accuracy on CIFAR10 increased by 5% on the test set and the accuracy on ImageNet increased roughly by 1% compared to using ReLU. For

the CartPole robotics task, the mean reward increased by 10 points by using the already learned activation function in the case of Deep Q-learning and 2 points in the case of Proximal Policy Optimization.

The thesis is split into three main sections: Background, Methods, and Experiments. The Background section (Section 2) describes the background and previous work. The Methods section (Section 3) describes different methods used and the design of the trainable activation function. The Experiments section (Section 4) begins with simple one-dimensional and two-dimensional functions to validate that this method is viable. From there on, the focus is mostly on image classification and starts from easier tasks and networks and progresses toward more complicated examples. First, the experiments on the CIFAR10 dataset using a small network are analyzed. Secondly, the same experiments using a larger network are analyzed. Concluding the image classification topic, some experiments were run on the ImageNet dataset. From there on, simple robotics task is examined to see if the method works in the case of reinforcement learning. The analysis in all cases is done by looking at how the activation functions were learned and reasoning why.

The code used for training and analysis is available at `http://github.com/ingmarliibert/trainable-activations`

2 Background

There have been many attempts at finding out which activation functions are optimal to use or attempts at designing and searching for new activation functions. Designing new activation functions is an active research field as activation functions can solve many problems. For example, the popularisation of the Rectified Linear Unit (ReLU) helped against the vanishing gradient problem, helped train deeper networks [GBB11] and was partly responsible for the breakthrough of deep neural networks around 2012 [KSH12]. More recently, the Swish activation function was discovered using a specific search method [RZL17]. Swish increased the performance of image classification tasks and allowed to train deeper networks. Below is an introduction to various approaches used when it comes to activation functions. The search technique DARTS is also described in the context of searching for activation functions.

2.1 Activation functions

Activation functions are a central part of neural networks. Each neuron takes a vector of real values as an input and calculates the weighted sum of those values. The output of the neuron is the activation function applied on the sum. Activation functions must be continuous, bounded, and nonconstant for neural networks to be universal function approximators [Hor91]. Activation functions can be classified as fixed-shape activation functions or trainable activation functions [ADIP21].

2.1.1 Fixed-shape activation functions

Fixed-shape activation functions are all the classical activations functions with a fixed shape, for example sigmoid, tanh and ReLU. Rectifier-based functions such as ReLU, and LReLU also form a subgroup inside the fixed-shape group [ADIP21]. Examples of sigmoid and ReLU activation functions can be seen in Figure 1.

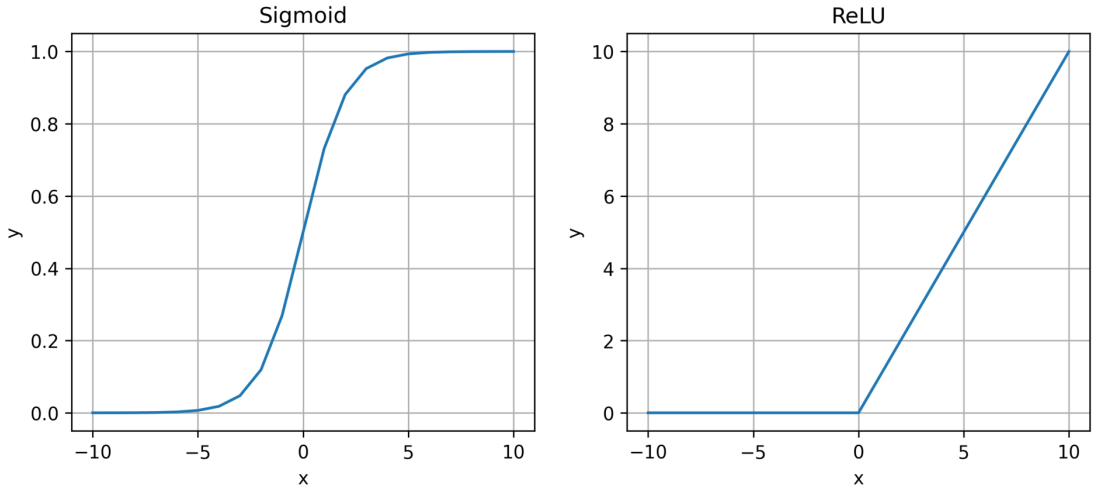

Figure 1. Examples of fixed shape activation functions

2.1.2 Trainable activation functions

The set of trainable activation functions contains all activation functions for which the shape is learned during the training phase [ADIP21]. These group into two different families: parameterized standard functions and functions based on ensemble methods. Parameterized standard functions are derived from standard fixed activation functions with a set of trainable parameters. Functions based on ensemble methods are defined by mixing several distinct functions, for example combining them linearly. These approaches can be often expressed in terms of feed-forward subnetworks which receive one single input value [ADIP21].

Parameterized standard functions Parameterized standard functions are functions where parameters are added to classical activation functions that allow their shape to be tuned. For example, the first attempt at a trainable activation function was the adjustable generalized sigmoid [Hu92]. This is a simple function where two parameters α, β are added to the classical sigmoid function that adjust the function shape:

$$AGSig(a) = \frac{\alpha}{1 + \exp(-\beta a)}$$

In a similar fashion, the Parametric ReLU (PReLU) [HZRS15] has one added parameter α:

$$\text{PReLU}(a) = \begin{cases} a & \text{if } a > 0 \\ \alpha \cdot a & \text{otherwise} \end{cases}$$

The AGSig and PReLU activation functions with different parameter values can be seen respectively in Figures 2 and 3.

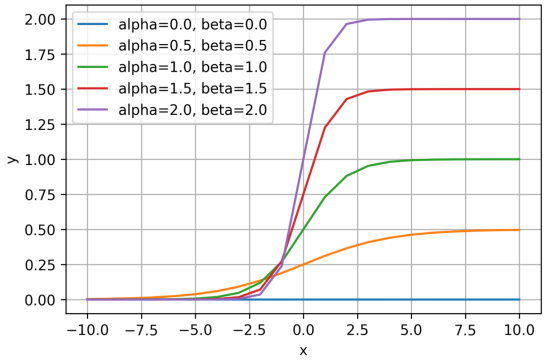

Figure 2. AGSig activation function

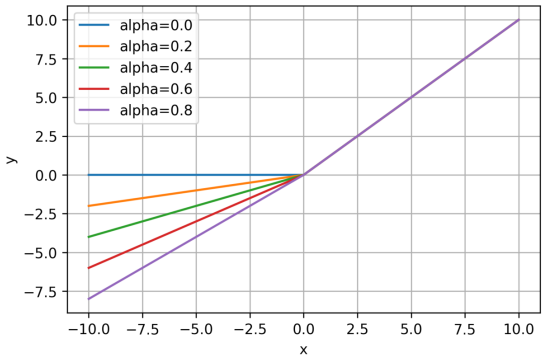

Figure 3. PReLU activation function

Functions based on ensemble methods Ensemble methods refer to merging together different functions. For this, a set of basis functions and a combination model are needed [ADIP21]. A good example of this is the approach by which the Swish function was discovered [RZL17]. They designed a search method that combined different unary and binary functions together. The combination model is the way in which the activation functions must be combined.

A big family of ensemble methods is the linear combination of one-to-one functions [ADIP21]. In this thesis, also a trainable activation function from that family is under consideration, where multiple activation functions are in a weighted linear combination and the weights are normalized using Softmax. There are some similar approaches to this that are explored below.

Adaptive Blending Unit Adaptive Blending Unit (ABU) [SBF+18] is a trainable activation function that combines different functions linearly and sums them:

$$\sum_{i=1}^{k} \alpha_i \cdot f_i(a)$$

where α_i are the parameters to learn and f_i are the activation functions. They applied the trainable function unit at the layer level. They normalized the weights in four ways -

1. dividing the weights by their sum
2. dividing the weights by the sum of their absolute values
3. clipping negative values before normalization and dividing the weights by their sum
4. applying Softmax to the weights

The 4th choice is similar to the approach explored in this thesis. The functions used were tanh, ELU, ReLU, identity, and Swish. They found that using ABU increased the performance in several cases over regular activation functions. For example, in the case of Simple Modular Convolutional Networks (SMCN) trained with Adam on CIFAR10 the increase was approximately 3% accuracy gain over ReLU. The most performant normalization technique was dividing the weights by the sum of their absolute values, but each of the normalization techniques performed similarly enough for the authors not to prefer one over another.

Adaptive Activation Functions Adaptive Activation Functions (AAF) are similar weighted linear combination activation functions where sigmoid normalization was used for each weight [DP16]. They used the trainable activation function on a per-neuron basis. The adaptive activation function was defined as

$$v = \sum_{i=1}^{N} f_i(u) l(g_i)$$

where f_i is the activation function, $l(x) = \frac{1}{1+e^{-x}}$, and g_i is the matrix of parameters to be optimized for. The set of functions used was sigmoid, tanh, and ReLU. They placed the trainable activation functions first at the first layers of the network and then at the last layers. They found that when placing them at the end, the accuracy was better than when placing the functions at the beginning. For the CIFAR10 dataset, the adaptive activation function produced a gain of 2% when compared to the baseline of ReLU and for the Caltech256 dataset there was no gain over using ReLU.

There are also cases of trainable activation functions where the set of weights was learned per feature map in a convolutional network [EWL16]. In this thesis, all of those options are examined - learning a set of weights per neuron, per kernel, and per layer.

2.2 DARTS

Differentiable architecture search (DARTS) is an efficient neural architecture search method that leverages a continuous search space where the architecture can be optimized using gradient descent [LSY18]. The categorical choice of a particular operation is relaxed to a Softmax over all possible operations:

$$\overline{o}^{(i,j)} = \sum_{o \in \mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} o(x)$$

where the operation weights for a pair of nodes (i, j) are parameterized by a vector $\alpha^{(i,j)}$ and O is the set of operations. The search is then reduced to learning a set of variables

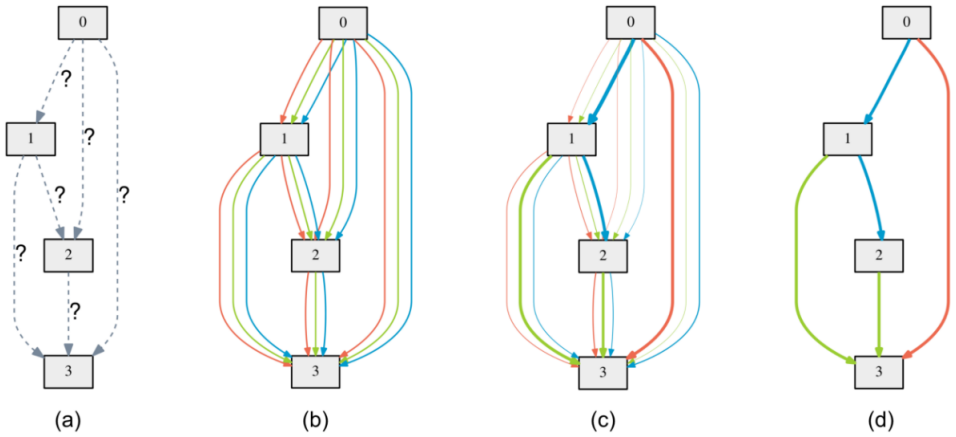

Figure 4. Overview of the DARTS process [LSY18]

$\alpha = \alpha^{(i,j)}$. The process can be seen in Figure 4 - at (a) the operations on the edges are unknown, at (b) a mixture of operations is placed on each edge, at (c) the architecture weights are learned while training, and at (c) the final architecture is induced.

They used both validation loss and training loss to minimize the architecture weights, but using the validation loss proved to be a better choice by reaching a better final architecture. Optimizing architecture weights on validation loss while optimizing the other weights of the network on training loss is a bi-level optimization problem:

$$\begin{aligned} \min_{\alpha} \quad & \mathcal{L}_{val}(w^*, \alpha) \\ \text{s.t} \quad & w^* = \text{argmin}_w \mathcal{L}_{train}(w, \alpha) \end{aligned}$$

where the goal is to find architecture weights α that minimize the validation loss $\mathcal{L}_{val}(w^*, \alpha)$, where the rest of the network weights w^* are found by minimizing the training loss: $w^* = \text{argmin}_w \mathcal{L}_{train}(w, \alpha)$. They proposed a simple approximation scheme to achieve this while avoiding expensive inner optimization and using only a single training step:

$$\nabla_\alpha \mathcal{L}_{val}(w^*, \alpha) \approx \nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha)$$

where w denotes the current weights of the network, α denotes the architecture weights and ξ is the learning rate for a step of the inner optimization.

Deriving the final architecture is done by replacing each mixed operation with k most likely operations, where k was 1 or 2 usually, based on circumstances. The whole process as an iterative algorithm can be seen in Algorithm 1. After deriving the final architecture, they trained the model again on the combination of training and validation dataset and evaluated the model on the test dataset.

Algorithm 1: DARTS

Create a mixed operation $\overline{o}^{(i,j)}$ parametrized by $\alpha^{(i,j)}$ for each edge (i, j)
while *not converged* **do**
    Update architecture α by descending $\nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha)$
    Update weights w by descending $\nabla_w \mathcal{L}_{train}(w, \alpha)$
Derive the final architecture based on the learned α

3 Methods

This section describes the methods used in this thesis. Definition of the trainable activation functions on the layer, kernel, and neuron level, training algorithms, and evaluation are described.

3.1 Trainable activation function

The trainable activation function designed is a weighted linear combination of different activation functions, where the weights are normalized using Softmax. Different sets of activation functions were explored, from which the most commonly occurring were:

1. A set containing basic activation functions:
    - $ReLU$
    - $Tanh$
    - $Sigmoid$
    - $Linear$
2. A set containing all from the basic set and also some other more popular activation functions:
    - $Swish$
    - $Softsign$
    - $Softplus$
    - $SeLU$
    - $Hard\ sigmoid$
    - $GeLU$
    - ELU
3. A set containing all from the previous set with the addition of some more uncommon choices for activation functions:

- $Softmax$
- Sin
- Cos

The trainable activation function is defined as:

$$\sum_{f \in \mathcal{F}} \frac{exp(\alpha_f)}{\sum_{f' \in \mathcal{F}} exp(\alpha_{f'})} f(X)$$

where $\mathcal{F}$ is a set of activation functions, α_f is the weight for activation function f. For the layer level trainable activation function, X is the input to the layer. For the kernel level trainable activation function, X is the input to the kernel. For the neuron level trainable activation function, X is the input to the neuron.

In the case of the layer level activation function, when there are N activation functions in the activation functions set, then there are N activation function weights for each layer. In the case of the kernel level activation function, there are N activation function weights per each kernel. In the case of neuron level activation function, there are N activation function weights per each neuron.

For all of the activation functions to be equally preferred at the start, all of the weights for the activations were initialized at zero.

3.2 Training

In the case of trainable activation functions, usually optimizing the activation function weights is done on the training loss together with the other weights. However, if we approach the issue as hyperparameter optimization, then we may also use the DARTS method of optimizing the weights on the validation loss. In the case of DARTS, optimizing on the validation loss led to better performance, while optimizing on the training loss led to overfitting of the training data, leading to poor generalization [LSY18]. Therefore two training schemes were used - one optimizing on validation loss using a simplified version of the DARTS approximation introduced in the Background section (Section 2) for a simpler implementation, as can be seen in Algorithm 2, and another optimizing on training loss, as can be seen in Algorithm 3. In the algorithms, α denotes the weights of the activation functions, w denotes the weights of the rest of the network.

Algorithm 2: Optimizing on validation loss

while *not converged* **do**
    Update weights w by descending $\nabla_w \mathcal{L}_{train}(w, \alpha)$
    Update activation weights α by descending $\nabla_\alpha \mathcal{L}_{val}(w, \alpha)$

Algorithm 3: Optimizing on training loss

1 **while** *not converged* **do**
2 Update weights w, α by descending $\nabla_{w,\alpha}\mathcal{L}_{train}(w, \alpha)$

Deriving the final architecture Deriving the final architecture after training and analyzing the results was done in the case of the layer level activation function by replacing the mixed operations with k most preferred operations by weight for each layer, where different values of k were tested, seeing if the benefits remained. The k values used were $k = 1$ where the most preferred activation function remained as the only activation function used for the layer, and $k = |F|$ where F is the set of activation functions used, where all of the activation functions remained but the activation weights were fixed and set as not trainable. For example, when there were 4 activation functions in the set - ReLU, sigmoid, tanh and linear - then for $k = 1$ for each layer the function with the highest weight was used for that layer in the derived architecture - for example, sigmoid for the first layer, ReLU for other layers. For $k = 4$ all of the functions remain in a linear combination, but the weights are fixed. After deriving the final architecture, the model was trained again on the whole data and evaluated on the test set.

Evaluation The evaluation was done by comparing the train and test accuracy of a model using trainable activation functions to a baseline model's train and test accuracy. The baseline model uses the default activation function for that type of model and task, for example, ReLU for image classification or tanh in the case of some robotics tasks. Two aspects were evaluated. The first one is the performance of the model after training the model using the trainable activation function. The second one is the performance of the model after deriving the final architecture and training the model again. Activation functions learned on the layer level are also presented as graphs of the learned weights and shapes of the learned functions. For kernel and neuron level activation functions that is not done because of the great number of activation functions learned.

4 Experiments

This section contains the experiments that have been run using the trainable activation unit. For all new dataset and model combinations tested, a baseline with the default activation function was trained to compare against to. For image classification, the default activation function was ReLU, for some robotics tasks and reinforcement learning algorithms the default activation function was tanh. By default the activation function weights were optimized on training loss unless specified otherwise.

4.1 Simple functions

To introduce and illustrate the problem, two small problems are solved using the layer level and neuron level trainable activation function units, motivated by a blog post by Matiisen [Mat18].

4.1.1 Sinusoid

The first function to be learned is the sine wave. The data points of the function to be learned are generated using the sine function with random noise generated from the normal distribution with a scale of 0.15. The data points can be seen in Figure 5. The model used was a simple model with one hidden layer, one input, and one output.

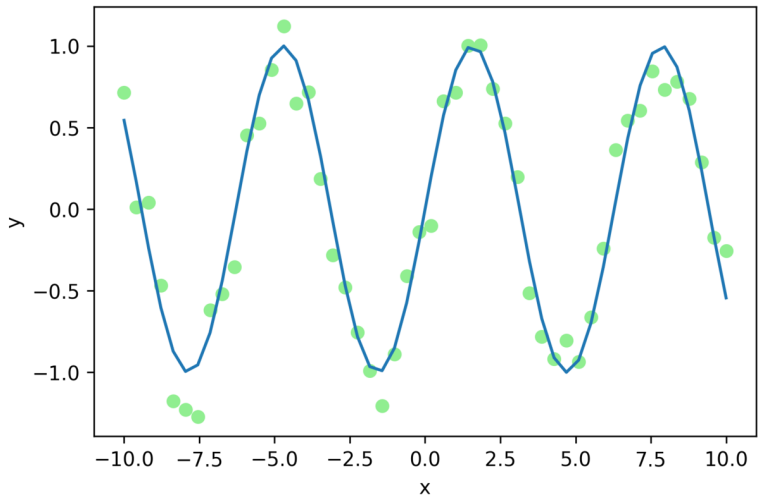

Figure 5. Sine wave data points with noise

The first experiment used a layer level trainable activation function with functions ReLU, tanh, sigmoid, and linear with 10 neurons in the hidden layer. The function was learned successfully by the network. The separate contributions of each node and the

final curve can be seen in Figure 6. The Softmax normalized weights of the learned activation functions can be seen in Figure 6. It can be seen from the learned weights that the learned combination was mostly sigmoid with a little bit of tanh. The learnable activation function successfully chose the most optimal choice from the 4 functions - using ReLU, it would have not been possible to learn the function using only 10 hidden nodes.

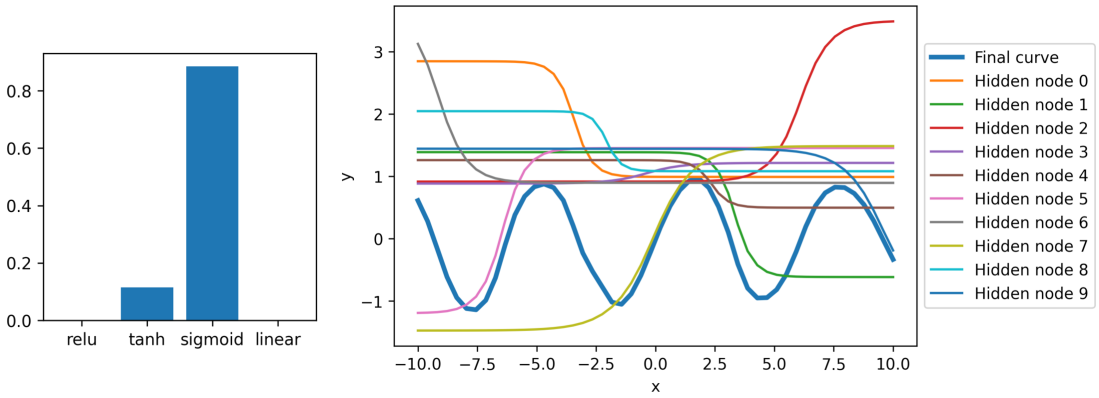

Figure 6. Learned activation weights and individual contributions of hidden nodes

However, when using the cosine function as one of the candidates instead of the linear function, the network can learn the sine wave using only 3 nodes or even 1, as can be seen from Figure 7. It can be seen that the trainable activation function chose cosine as the preferred activation function.

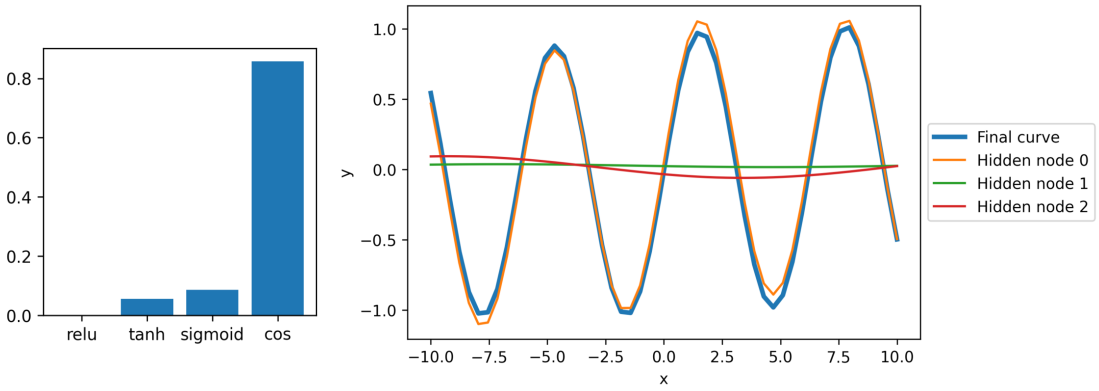

Figure 7. Learned activation weights and individual contributions of hidden nodes

When using the cosine function as the activation function the learned function generalizes beyond the training set, as can be seen in Figure 9 as opposed to when using the sigmoid function, which can be seen in Figure 8.

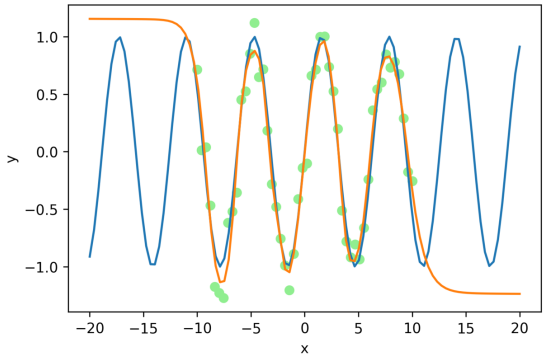

Figure 8. Learned function using sigmoid

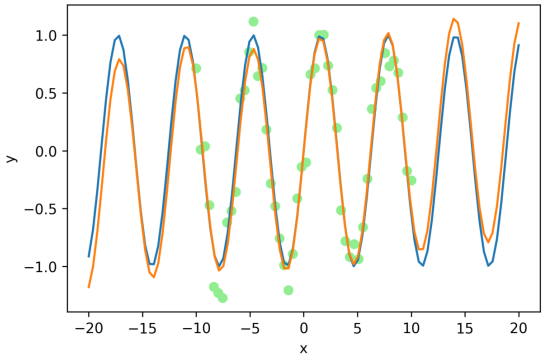

Figure 9. Learned function using cosine

The same effect follows when using the neuron level trainable activation function. As can be seen in Figure 10 and Figure 11, one node chose the cosine function while the other two had very little effect on the curve. As can be seen from Figure 10, hidden nodes 1 and 2 had very small weights compared to hidden node 1 which gave shape to the final curve.

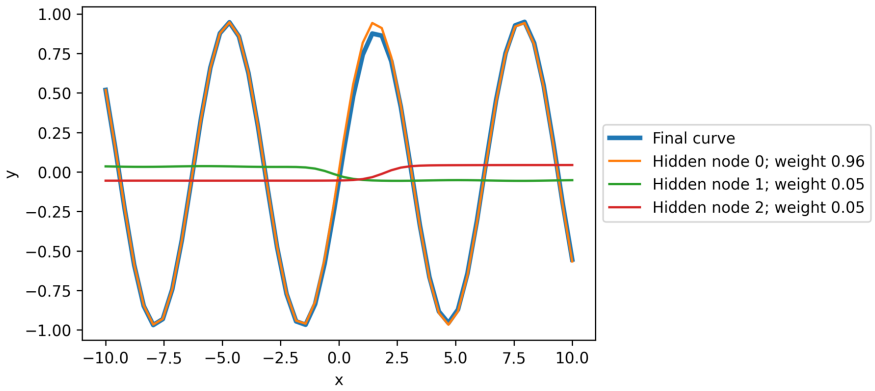

Figure 10. Individual contributions of nodes using neuron level trainable activation function

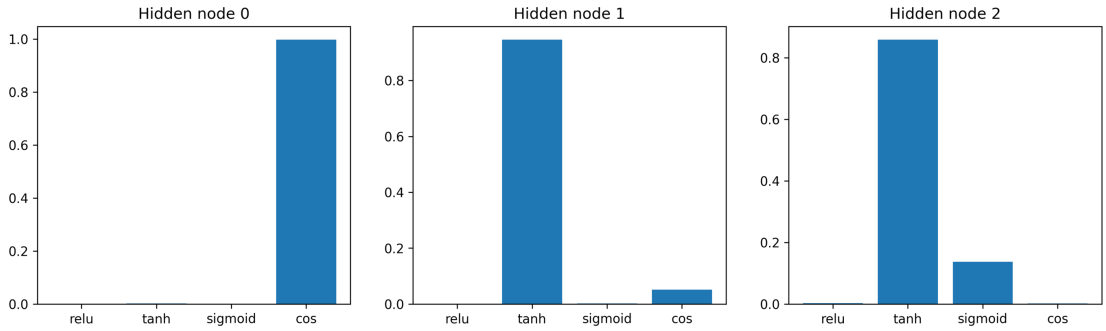

Figure 11. Learned weights of the activation functions for each neuron

4.1.2 Spiral

The second function to be learned is the spiral function with two classes [Pyt17], as can be seen in Figure 12. The model used in this case was a model with two hidden layers with 10 nodes each and two inputs - coordinates for the point. The goal was to classify each point as 0 or 1. Behind both hidden layers was a layer level trainable activation function with the functions ReLU, tanh, sigmoid and sine. The model successfully learned the function, as can be seen in Figure 13 for predictions on train data and Figure 14 for predictions on a wider dataset. Both layers chose sine to be the most preferred function, as can be seen in Figure 15. Sine was chosen to be the activation function because of the periodic and circle-like nature of the dataset. It effectively transforms the coordinates into spherical coordinates which helps to learn the function faster.

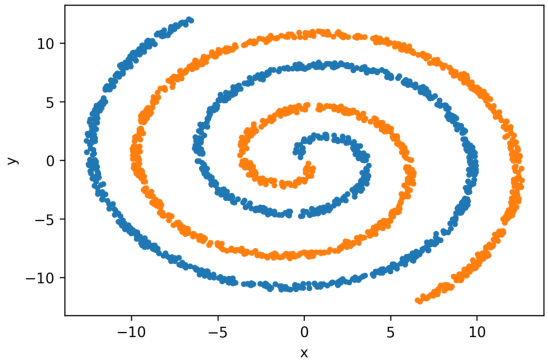

Figure 12. Spiral function

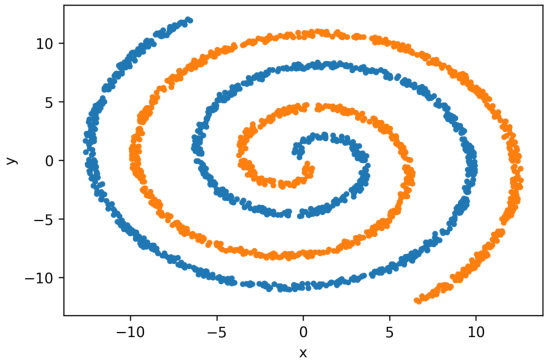

Figure 13. Model prediction on train data

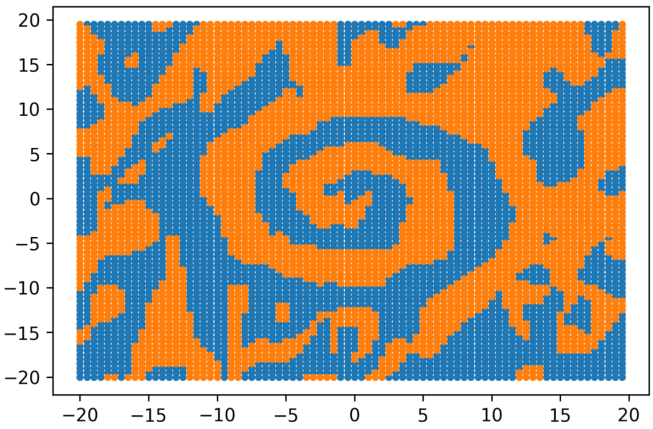

Figure 14. Model prediction on test data

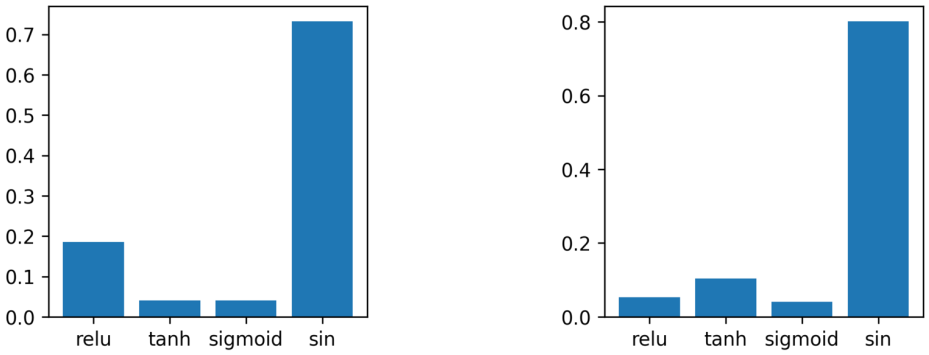

Figure 15. Learned weights of hidden layers

4.1.3 Conclusion

It can be seen from the experiments that the layer level and neuron level activation functions successfully chose the optimal choices from the activation function set. If the technique works in simple examples like that, it could also work in more complicated cases helping choose better activation functions that not only allow the network to learn the training data better but also help to generalize beyond the training set.

4.2 Image classification

Image classification experiments were the main experiments. The experiments start with smaller datasets and smaller models and gradually move to bigger datasets with bigger models. The datasets used were CIFAR10 [KH+09] and ImageNet [RDS+15]. The goal was to find at first results and patterns on simpler networks and datasets and then compare and validate if the same results emerge on larger networks and datasets.

For the experiments with the small model and basic activation function set on the CIFAR10 dataset, each experiment was repeated 3 times to see if the training routines using the trainable activation functions are stable. The means and standard deviations of the accuracies were calculated and the results are presented as $mean \pm std$ in the tables. The layer level learned activation function weights and shapes are used and analyzed as examples from a single run. Experiments on bigger models and datasets are presented as single run accuracies.

4.2.1 CIFAR10 with small model and basic activation functions

Table 1. Non-regularized model

Layer type	Layer output
InputLayer	(N, 32, 32, 3)
Conv2D	(N, 30, 30, 32)
Activation	(N, 30, 30, 32)
MaxPooling2D	(N, 15, 15, 32)
Conv2D	(N, 13, 13, 64)
Activation	(N, 13, 13, 64)
MaxPooling2D	(N, 6, 6, 64)
Conv2D	(N, 4, 4, 64)
Activation	(N, 4, 4, 64)
Flatten	(N, 1024)
Dense	(N, 64)
Activation	(N, 64)
Dense	(N, 10)

Table 2. Regularized model

Layer type	Layer output
InputLayer	(N, 32, 32, 3)
Conv2D	(N, 30, 30, 32)
Activation	(N, 30, 30, 32)
MaxPooling2D	(N, 15, 15, 32)
Conv2D	(N, 13, 13, 64)
Activation	(N, 13, 13, 64)
MaxPooling2D	(N, 6, 6, 64)
Dropout	(N, 6, 6, 64)
Conv2D	(N, 4, 4, 64)
Activation	(N, 4, 4, 64)
Flatten	(N, 1024)
Dense	(N, 64)
Activation	(N, 64)
Dropout	(N, 64)
Dense	(N, 10)

The dataset and model used in this experiment were the CIFAR10 dataset [KH+09] with a small basic convolutional neural network [Ten22]. Two models were used, one without dropouts, which can be seen in Table 1 and is called here the non-regularized model, and one with dropouts, called here the regularized model, which can be seen in

Table 2. Both models have L2 kernel regularization terms on convolutional and dense layers with a factor of 0.001. The optimizer used was Adam with a learning rate of 0.01, no augmenting or preprocessing was done on the images. The training set contained 60,000 images and the test set 10,000 images. The training set and test set were the same for all experiments. The models were trained for 20 epochs, which was the number of epochs where the baseline model did not continue any further learning.

First, the baseline was found by training the models using the ReLU activation function. After that, the models were trained using layer level, kernel level, and neuron level trainable activation functions as the activations. The activation set was comprised of ReLU, tanh, sigmoid and linear functions. Both the non-regularized and regularized model activation functions' weights were optimized on training loss. The results of training can be seen in Table 3.

Table 3. Small model accuracies on CIFAR10 using basic activation function set

Type	**Non-regularized**		**Regularized**	
	Train acc.	**Test acc.**	**Train acc.**	**Test acc.**
ReLU	0.8181 ± 0.0049	0.7096 ± 0.0037	0.6991 ± 0.0029	0.7152 ± 0.0015
Layer	0.8675 ± 0.0010	0.7332 ± 0.0031	0.7724 ± 0.0007	0.7636 ± 0.0054
Kernel	0.8493 ± 0.0018	0.7321 ± 0.0032	0.7516 ± 0.0012	0.7515 ± 0.0070
Neuron	0.8496 ± 0.0028	0.7217 ± 0.0013	0.7469 ± 0.0008	0.7380 ± 0.0044

Baseline The baseline was trained using the ReLU activation function. In the case of the non-regularized model, there can be noted slight overfitting of the model on the training data. For the regularized model, the overfitting is gone and test accuracy slightly rose over the non-regularized model's test accuracy.

Layer level activation functions For the layer level trainable activation function, it can be seen that in the case of the non-regularized model, the training accuracy rose roughly 5% over the baseline and the test accuracy rose 2% over the baseline. In the case of the regularized model, the training accuracy rose 7% over the baseline and the test accuracy rose roughly 5% over the baseline. The use of the trainable activation function raised the accuracies significantly.

The non-regularized model's Softmax normalized weights of the activation functions and the activation functions' shapes can be seen in Figure 16. The shapes are plotted by giving the activation function an input vector ranging from -20 to 20. The input of the function is on the horizontal axis and the output on the vertical axis. It can be seen that the most different from ReLU are the first and second layers, while the last layer is mostly only ReLU. The first layer might be different because of the need to scale or transform the data into a more suitable shape. In the case of using the trainable activation

functions for the regularized model, the first epoch of the training achieved much higher accuracy than the first epoch of the baseline (0.20 vs 0.45), and often in the case of using only ReLU, the regularized model did not begin to train and accuracy remained at 0.1, which did not happen when using trainable activation functions. It may be that this happens because of dead ReLUs and the usage of the trainable activation function helps against this.

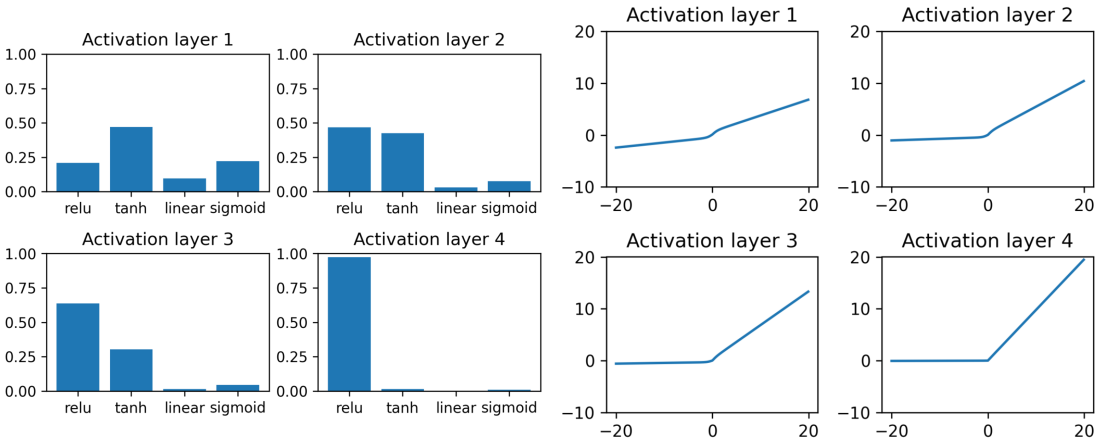

Figure 16. Non-regularized model's learned activation functions' weights and shapes

The regularized model's weights of the activation functions and the activation functions' shapes can be seen in Figure 17. It can be seen that the first 3 activation layers are similar to the non-regularized model, but the last one seems to be mostly linear in shape, suggesting that the last layer does not contribute much to learning. It may be that the last layer is not needed because the expressiveness of the model increased using the trainable activation function.

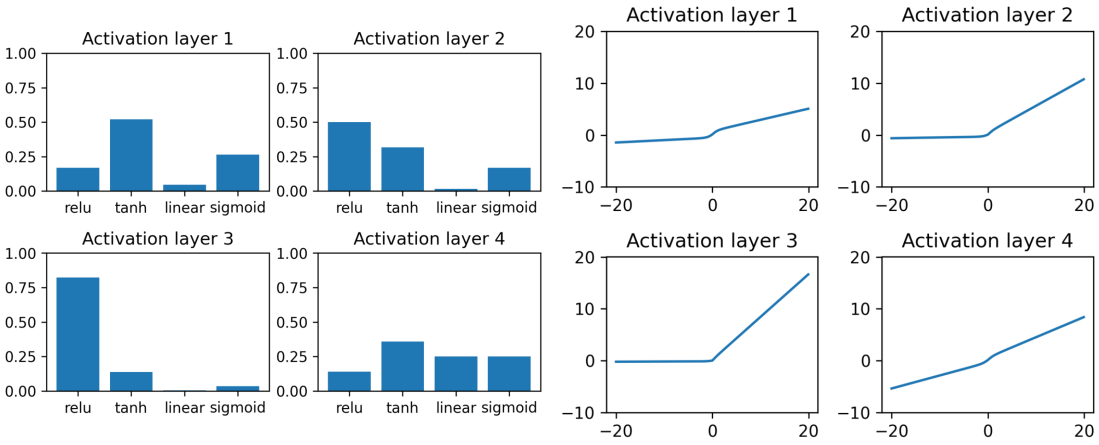

Figure 17. Regularized model's learned activation functions' weights and shapes

Next, deriving the final architecture was done using $k = 4$ and $k = 1$, which means that in the first case all activation functions remained with fixed weights and in the last case one with the greatest weight remained. After deriving the final architecture, the model was trained again. As can be seen in Table 4, the accuracies remained about the same as they were for the layer-level activation function. In the case of $k = 1$, the accuracies decreased significantly, probably because the combination of different functions was more suitable and one activation function did not dominate in most of the cases. Often the dominating activation function was something else than ReLU, which brought the accuracy down. Because deriving the final architecture and training the model again did not give any benefits, it is not repeated for other image classification experiments, except for the experiments where the activation weights were optimized on validation loss.

Table 4. Final derived architecture accuracies

Model type	**Activation type**	**Train accuracy**	**Test accuracy**
Non-regularized	ReLU	0.8181 ± 0.0049	0.7096 ± 0.0037
	Derived (k = 4)	0.7335 ± 0.0052	0.7078 ± 0.0042
	Derived (k = 1)	0.4718 ± 0.0413	0.4646 ± 0.0438
Regularized	ReLU	0.6991 ± 0.0029	0.7152 ± 0.0015
	Derived (k = 4)	0.7640 ± 0.0022	0.7549 ± 0.0042
	Derived (k = 1)	0.4346 ± 0.0118	0.4625 ± 0.0136

Kernel level activation functions Training the models using the kernel level activation functions generally achieved worse accuracy than when using the layer level activation function, but the accuracies still increased over the baseline - the non-regularized model's

train accuracy increased roughly by 3% and test accuracy increased by 2%, the regularized model's train accuracy increased by 5% and test accuracy by 3%. The accuracies can be seen in Table 3.

Neuron level activation functions Training the models using the neuron level activation functions generally achieved worse accuracy than when using the layer level and kernel level activation functions. Using neuron level trainable activation function, the non-regularized model's train accuracy increased by 3% over baseline and test accuracy increased by 1% over baseline. For the regularized model, the train accuracy increased by 4% over baseline and test accuracy increased by 2% over baseline. The accuracies can be seen in Table 3.

Conclusion Using the trainable activation function at any level gave a significant increase over the baseline accuracy of the model and increased the model's expressiveness. The best results were achieved using the layer level trainable activation function with roughly a 5% increase in test accuracy over the baseline in the case of the regularized model. The first layer activation functions were for both models a mix of all of the four activation functions and for the regularized model the last layer resembled the linear function. In the case of kernel and neuron level activation functions, the test accuracy increased a few percent over the baseline, but the results remained worse than for the layer level activation function. It may be that since using the kernel or neuron level trainable activation functions increases the parameter count of the model more significantly than the layer level trainable activation function, it would take longer to train the model.

4.2.2 CIFAR10 with small model and basic activation functions set, optimizing on validation loss

For optimizing on the validation loss, a validation set was extracted from the training set on which the activation weights were optimized. Two different splits were tested, one where the validation set was 20% of the training set and one where the validation set was 50% of the training set (as was done in DARTS). After the training part, the final architecture was derived with $k = 4$ and $k = 1$ and trained again on the combination of the training and validation set.

Validation set 20% of training set The first training results can be seen in Table 5. As can be seen they did not increase when compared to training the whole network on all of the data using layer level activation functions and optimizing all of the weights on training loss. The regularized model train accuracy grew, but cannot be directly compared to the train accuracy of the baseline, because the training dataset was smaller in this

case. The test accuracy of the regularized model using the trainable activation function achieved similar performance to the test accuracy of optimizing on training loss.

Table 5. Accuracies using ReLU and layer level activation function

Type	Model	Train accuracy	Test accuracy
ReLU	Non-regularized	0.8181 ± 0.0049	0.7096 ± 0.0037
	Regularized	0.6991 ± 0.0029	0.7152 ± 0.0015
Layer	Non-regularized	0.8476 ± 0.0032	0.7180 ± 0.0111
	Regularized	0.8204 ± 0.0005	0.7527 ± 0.0029

The layer level activation functions learned can be seen in Figures 18 and 19 for respectively the non-regularized and regularized model. Interestingly, sigmoid was often more preferred than other options. The learned weights were quite similar for both models.

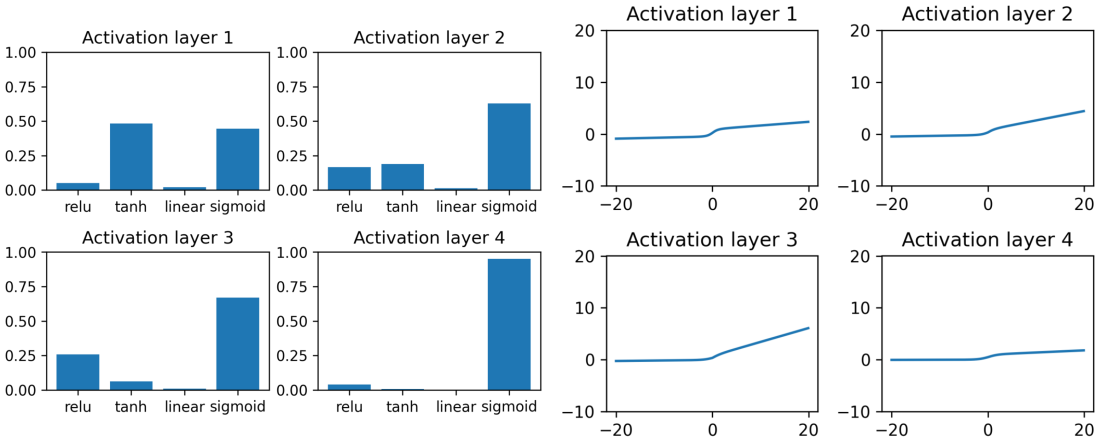

Figure 18. Non-regularized model's learned activation functions' weights and shapes

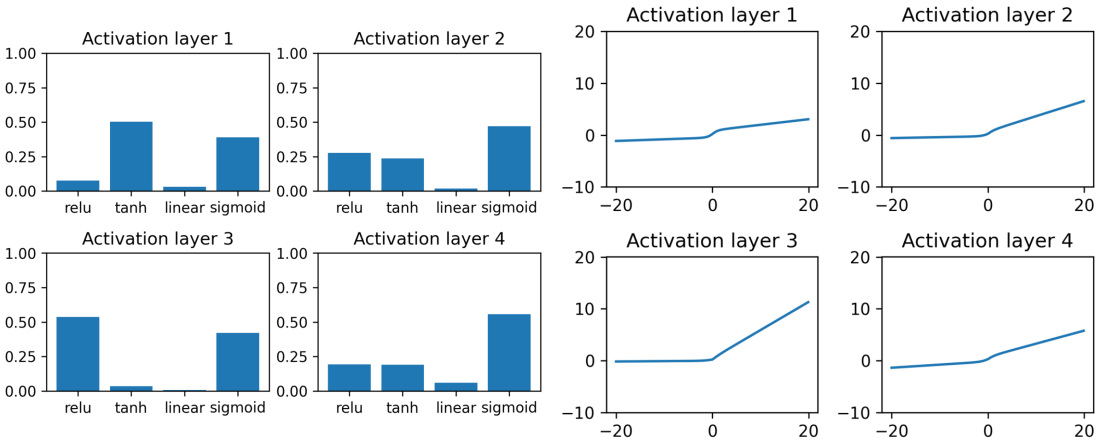

Figure 19. Regularized model's learned activation functions' weights and shapes

After that, for experiments using the layer level activation functions, the final architecture was derived using $k = 4$ and $k = 1$ and the whole network was trained again using fixed activation functions weights. The results can be seen below in Table 6. The accuracies did not increase over the accuracies achieved for initial training using layer level activation functions optimizing on training loss for $k = 4$. For $k = 1$ the accuracies were significantly lower as was the case when optimizing on training loss.

Table 6. Final derived architecture accuracies

Model type	Activation type	Train accuracy	Test accuracy
Non-regularized	ReLU	0.8181 ± 0.0049	0.7096 ± 0.0037
	Derived (k = 4)	0.8491 ± 0.0193	0.7382 ± 0.0079
	Derived (k = 1)	0.5916 ± 0.0517	0.5952 ± 0.0512
Regularized	ReLU	0.6991 ± 0.0029	0.7152 ± 0.0015
	Derived (k = 4)	0.7630 ± 0.0089	0.7558 ± 0.0053
	Derived (k = 1)	0.4354 ± 0.0265	0.4791 ± 0.0363

Validation set 50% of training set When using 50% of the dataset as the validation set on which to optimize the activation functions weights, the results were similar and no significant increase in accuracy was present. The results for training can be seen in Table 7.

Table 7. Accuracies using ReLU and layer level activation function

Type	Model	Train accuracy	Test accuracy
ReLU	Non-regularized	0.8181 ± 0.0049	0.7096 ± 0.0037
	Regularized	0.6991 ± 0.0029	0.7152 ± 0.0015
Layer	Non-regularized	0.8493 ± 0.0035	0.6908 ± 0.0053
	Regularized	0.8161 ± 0.0041	0.7137 ± 0.0104

The learned weights and shapes of the activation functions can be seen in Figures 20 and 21 respectively for the non-regularized model and regularized model. The learned weights are quite similar to the ones learned when using 20% of the training set for activation weights optimization.

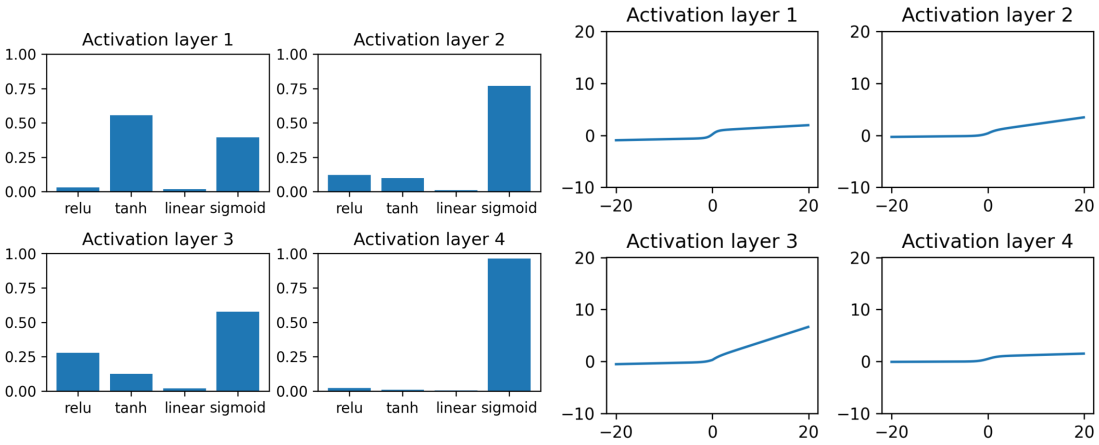

Figure 20. Non-regularized model's learned activation functions' weights and shapes

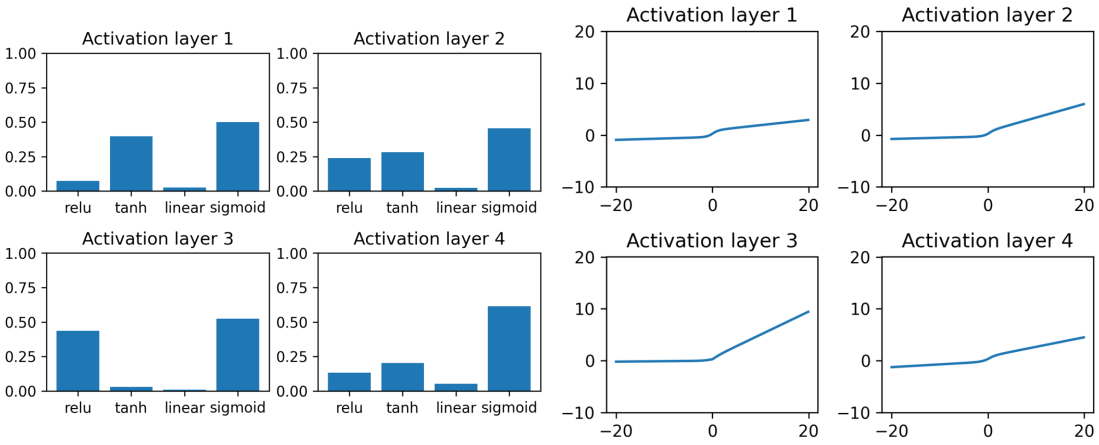

Figure 21. Regularized model's learned activation functions' weights and shapes

The accuracies after deriving the final architecture and training the model again on all training data can be seen in Table 8.

Table 8. Final derived architecture accuracies

Model type	**k**	**Train accuracy**	**Test accuracy**
Non-regularized	ReLU	0.8181 ± 0.0049	0.7096 ± 0.0037
	Derived (k = 4)	0.8482 ± 0.0094	0.7376 ± 0.0019
	Derived (k = 1)	0.6361 ± 0.0261	0.6204 ± 0.0203
Regularized	ReLU	0.6991 ± 0.0029	0.7152 ± 0.0015
	Derived (k = 4)	0.7548 ± 0.0113	0.7547 ± 0.0068
	Derived (k = 1)	0.4617 ± 0.0210	0.5087 ± 0.0210

Conclusion Optimizing the activation function weights on validation loss seem to give no extra benefits, however similar test accuracy was achieved as when optimizing on training loss, meaning that the decrease in training data for the rest of the network did not have very adverse effects. Different weights for activation functions were learned when optimizing on the validation loss, preferring more the sigmoid activation function.

4.2.3 CIFAR10 with small model and more activation functions

For this experiment, more activation functions were used to see if expanding the set brings any differences in performance or learned activation functions. Both non-regularized and regularized versions of the small model were trained again on the CIFAR10 dataset using layer, kernel, and neuron level trainable activation functions. The activation set

comprised of ReLU, tanh, linear, sigmoid, Swish, Softsign, Softplus, SeLU, hard sigmoid, GeLU, and ELU functions. The results of training can be seen in Table 9.

Table 9. Small model accuracies on CIFAR10 using more activation functions

Type	**Non-regularized**		**Regularized**	
	Train acc.	**Test acc.**	**Train acc.**	**Test acc.**
ReLU	0.8181 ± 0.0049	0.7096 ± 0.0037	0.6991 ± 0.0029	0.7152 ± 0.0015
Layer	0.881	0.727	0.777	0.760
Kernel	0.877	0.731	0.771	0.768
Neuron	0.886	0.738	0.777	0.756

Layer level activation functions In the case of the layer level activation function, the non-regularized model's train accuracy rose roughly 7% over baseline, which was slightly more than it was for the basic activation functions set, and test accuracy rose 1% over baseline. For the regularized model, train accuracy rose 8% over baseline which was more than for the basic activation functions set and test accuracy rose roughly 5% over baseline, which was similar to the increase when using the basic activation functions set. Seems that increasing the choices for activation functions enables the model to learn the training data better.

For the non-regularized model, we can see in Figure 22 that no activation function choice was dominant by weight and a combination of them was the result for each layer. It can be seen that the first layer is different from the rest three, which look more or less the same. The same pattern was seen for the basic activation function set in the previous experiments. It could be that when more activation functions are present that longer training is needed for a function to become dominant.

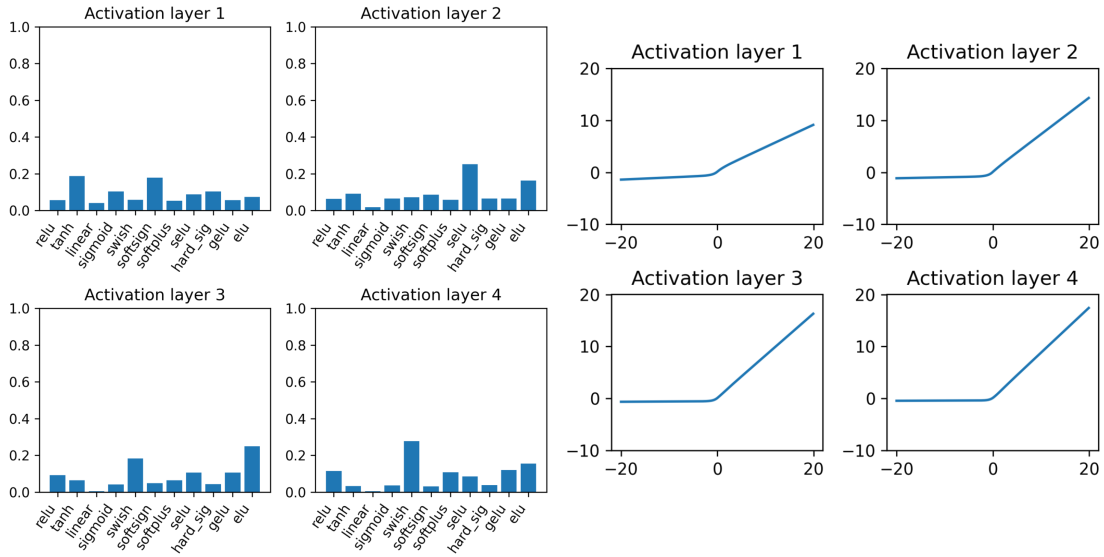

Figure 22. Non-regularized model's learned activation functions' weights and shapes

For the regularized model we can also see in Figure 23 that the more differing activation functions are the first and last layer, as we saw when using the basic activation functions set in previous experiments. The last layer here also is more linear than other layers. In activation layer 3, Swish was the most preferred function by far, where in other layers there were no dominant functions by weight.

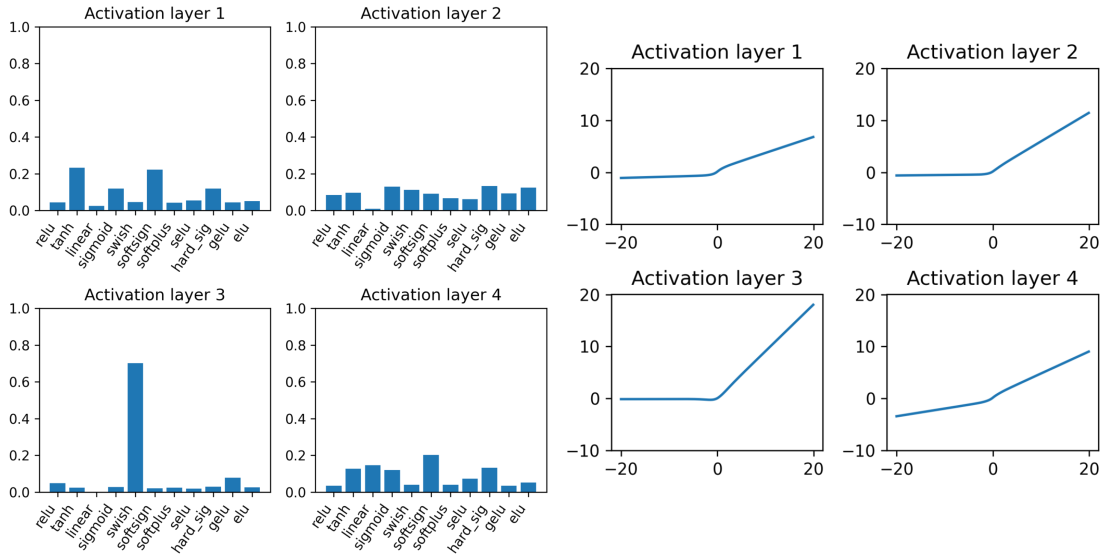

Figure 23. Regularized model's learned activation functions' weights and shapes

Kernel level activation functions Using the kernel level activation function, it can be seen that the non-regularized model's train accuracy rose roughly 7%, which was better than kernel level activation with the basic activation functions set achieved, and test accuracy rose roughly 2% over baseline. For the regularized model, the train accuracy rose 7% and test accuracy rose roughly 5% over baseline, which was slightly better than the layer and kernel level results for the basic activations set.

Neuron level activation functions For the neuron level activation function, it can be seen that the non-regularized model's train accuracy rose roughly by 7%, which was 4% better than in the case of the basic activation functions set, and test accuracy rose roughly 3% over baseline. For the regularized model, the train accuracy rose 7% and test accuracy rose roughly 4% over baseline, which were both better than in the case of the basic activation functions set.

Conclusion It seems that using more activation functions increased both the training and test accuracy in some cases and enabled the model to learn the train data better. When using more activation functions, no activation function was as preferred as when using the basic activation functions set. However, for the layer level activation functions, the learned activation shapes remained similar to the ones learned when using the basic activation functions set. Similarly, for the non-regularized model, the differing layer was the first one and for the regularized model the more differing layers were the first and the

last ones, where the last one was more linear than others.

4.2.4 CIFAR10 with small model and all activation functions

For this experiment, new members Softmax, sin, and cos were added to the previous activation functions set to see if they have any effect. In this case, both non-regularized and regularized versions of the small model were trained on the CIFAR10 dataset using layer, kernel, and neuron level trainable activation functions. The activation set comprised of ReLU, tanh, linear, sigmoid, Swish, Softsign, Softplus, SeLU, hard sigmoid, GeLU, ELU, Softmax, sin, and cos functions. The training results can be seen in Table 10.

Table 10. Small model accuracies on CIFAR10 using all activation functions set

Type	**Non-regularized**		**Regularized**	
	Train acc.	**Test acc.**	**Train acc.**	**Test acc.**
ReLU	0.8181 ± 0.0049	0.7096 ± 0.0037	0.6991 ± 0.0029	0.7152 ± 0.0015
Layer	0.884	0.735	0.779	0.751
Kernel	0.880	0.741	0.774	0.761
Neuron	0.893	0.733	0.781	0.764

Layer level activation functions In the case of the layer level activation function, it can be seen that the non-regularized model's train accuracy increased by 7% and test accuracy rose roughly by 2% over the baseline, which are similar results to the set of more activation functions. For the regularized model, the train accuracy rose by 8% and test accuracy rose roughly by 4% over the baseline, which are similar to the results achieved using the previous set of functions.

Considering the shapes of functions learned, as can be seen in Figures 24 and 25, the same patterns remained, that the most differing layer is the first one, and in the case of the regularized model also the last one. For the last layer of the non-regularized model, Swish was the dominant function by weight, which differed from the experiment ran on the previous set of functions used.

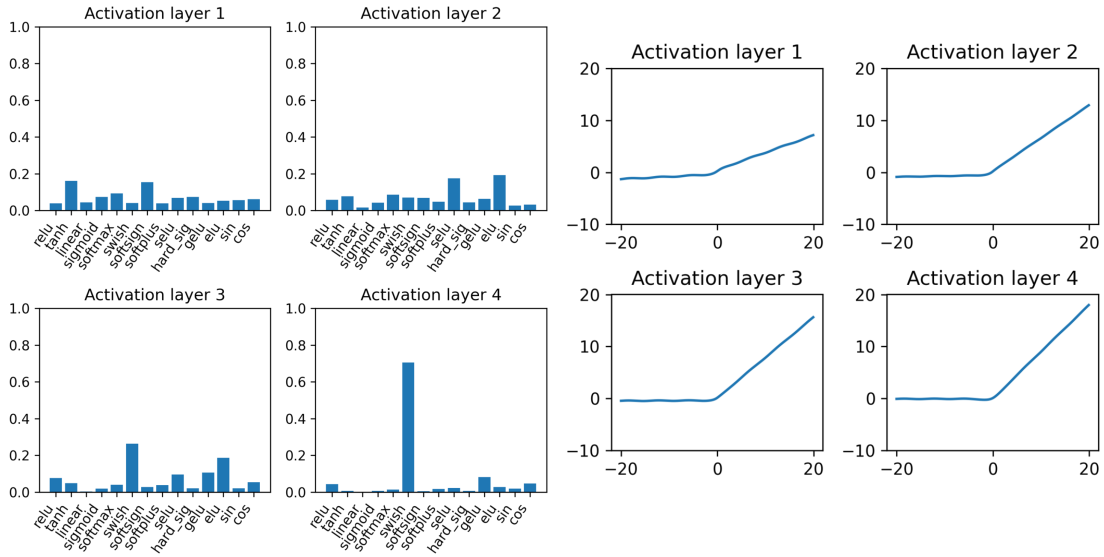

Figure 24. Non-regularized model's learned activation functions' weights and shapes

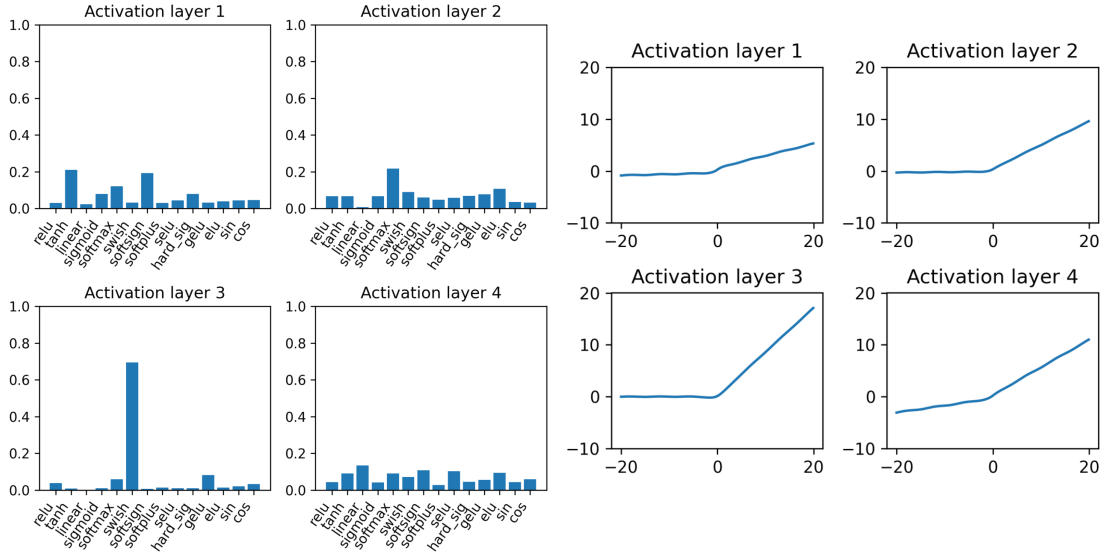

Figure 25. Regularized model's learned activation functions' weights and shapes

Kernel level activation functions Using the kernel level activation function, it can be seen that the non-regularized model's train accuracy rose roughly 7%, and test accuracy rose roughly 3% over baseline, which are similar to the accuracy achieved with previous

activation functions set. For the regularized model, the train accuracy rose 8% and test accuracy rose roughly 5% over baseline, which are similar to the accuracies achieved with the previous activation functions set.

Neuron level activation functions In the case of the neuron level activation function, it can be that the non-regularized model's train accuracy rose roughly 8% and test accuracy rose roughly 2% over baseline, which was similar to the previous set of activation functions used. For the regularized model, the train accuracy rose 8% and test accuracy rose roughly 5% over baseline, which were slightly better compared to the previous set of activation functions used.

Conclusion Adding Softmax, sin, and cos to the activation functions set does mostly not further significantly increase or decrease the train and test accuracy of the models in most cases. It does increase accuracy slightly in the case of neuron level activation functions. In some layers when using more or all activation functions, Swish is preferred strongly, indicating there might be a benefit to using Swish.

In Table 11 are presented all single run accuracies of the experiments ran on small models. From it can be seen that most of the test accuracies achieved on the regularized model are similar ranging from 0.75 to 0.76, which means all of the methods increased the test accuracy of the model significantly by roughly 4% to 5%. Using more activation functions to choose from slightly increased the accuracy further and helped the model fit the training data better than just using the basic four activation functions, however adding Softmax, sin, and cos to the set did not help further. It can be seen that the best test accuracy was achieved by using the kernel level function and the best train accuracy by using the neuron level function, using more and all activation functions sets respectively. Considering that the layer activation function is the most efficient one, adding only a small constant number of parameters to the model, it would be reasonable to use that the most, since kernel level and neuron level accuracy gains over the layer level activation function were mostly negligible.

Table 11. Single run accuracies of experiments ran on small models

Model type	Act. set	Train accuracies			Test accuracies		
		Layer level	Kernel level	Neuron level	Layer level	Kernel level	Neuron level
Regularized	ReLU	0.696			0.713		
	Basic	0.772	0.750	0.747	0.759	0.742	0.732
	More	0.777	0.771	0.777	0.760	**0.768**	0.756
	All	0.779	0.774	0.781	0.751	0.761	0.764
Non-regularized	ReLU	0.813			0.712		
	Basic	0.867	0.848	0.853	0.734	0.736	0.722
	More	0.881	0.877	0.886	0.727	0.731	0.738
	All	0.884	0.880	**0.893**	0.735	0.741	0.733

4.2.5 CIFAR10 with a small ResNet

The next experiment was training a small ResNet model on CIFAR10 using layer, kernel, and neuron level activation functions optimizing on train loss. The motivation for this is to find out if the benefits of using the trainable activation function remain if the baseline accuracy is higher to begin with. To get a better baseline accuracy, a larger model was needed. The chosen model was a small ResNet model [HAB19] designed for use on CIFAR10. The parameters of the model can be seen in Table 12. The optimizer used for training was the SGD optimizer with a learning rate of 0.01 and momentum of 0.9. The learning rate was divided by 10 after validation loss reached a plateau for 5 epochs. The minimum allowed learning rate was 0.0001. The batch size was 64. The models were trained for 30 epochs.

Table 12. ResNet model parameters

N residual units	N filters
0	16
3	16
3	32
3	64

The results can be seen in Table 13. As can be seen, only models using layer level functions with basic or all activation functions set increased in accuracy over the baseline. The increase in accuracy was approximately 1% for the layer level activation function using the basic activation functions set and 0.4% if using all activation functions. The benefit of using the trainable activation functions is smaller in this case.

Table 13. ResNet accuracies on CIFAR10

Act. fn. level	Act. fn. set	Train accuracy	Test accuracy
ReLU		0.987	0.851
Layer	Basic	**0.991**	**0.859**
	More	0.987	0.851
	All	0.988	0.855
Kernel	Basic	0.932	0.839
	More	0.976	0.847
	All	0.946	0.844
Neuron	Basic	0.914	0.830
	More	0.966	0.850
	All	0.952	0.851

The learned activation functions' weights and shapes for the layer level activation functions can be seen in Appendix I.I. For the basic activation functions set it can be seen in Figure 32 that the 4 or 5 first activation layers preferred a mix of all the functions and the layers 6 to the second to last layer all preferred ReLU as the activation function. The last layer however strongly preferred linear, which was the pattern also for the small regularized model. The shapes of the functions can be seen in Figure 33.

When using more activation functions, for layers 1 to 12 there were no dominantly preferred activation functions, as can be seen in Figures 34 and 35 in Appendix I.II. For the last layers from 13 to 19 the preferred layer was either GeLU or SeLU in the last layer. For the all activation functions set it can be seen in Figures 36 and 37 in Appendix I.III that the pattern is the same - for the first 12 layers there are no dominant activation functions by weight and for the last 7 layers it is either GeLU or SeLU in case of the last layer.

To conclude, the benefits of using the trainable activation functions with a larger model that achieves a higher baseline are more negligible. The patterns for the learned activation functions remain the same roughly, but it seems that in the first layers when using larger activation functions set the first layers do not learn to prefer any activation functions, which might be mitigated with longer training.

4.2.6 ImageNet with ResNet-18

The last image classification experiment was to train a bigger model on a bigger dataset to see if there are any benefits to using the trainable activation functions. The dataset used was ImageNet [RDS+15] and the model used was ResNet-18 [HZRS16] as implemented in the package keras-resnet [Kot17]. Training was done using the SGD optimizer with batch size 256. Images were resized to 224 by 224, random rotation with a factor of 0.2, and random flipping augmentations were applied during training to increase the

validation accuracy. The learning rate was 0.1 and it was reduced on validation loss plateau with a patience of 5 epochs. The minimum learning rate was 0.0001. The model parameters can be seen in Table 14. The models were trained for 80 epochs.

Table 14. ResNet-18 model parameters

N residual units	**N filters**
2	64
2	128
2	256
2	512

Three models were trained: one baseline using ReLU as the activation function, one using layer level trainable activations comprised of ReLU, tanh, linear, sigmoid, Swish, referred to as set A, and one using layer level trainable activations comprised of ReLU, tanh, linear, sigmoid, Swish, Softmax, referred to as set B. A smaller set of functions is used because the network did not begin to train with a larger set of functions. The results of training can be seen in Table 15. Using layer level activation function with set A resulted in a small increase of almost 1% in validation accuracy, using set B resulted in a slight decrease in validation accuracy.

Table 15. ResNet-18 accuracies on ImageNet

Act. fn. set	**Train accuracy**	**Val. accuracy**
ReLU	0.771	0.599
A	0.771	0.608
B	0.776	0.593

The learned layer level activation functions' weights and shapes can be seen in Appendix I.IV. As can be seen in Figures 38 and 39 that for the first activation functions set and for the first layers sigmoid is the dominant function by weight and from there on for the rest of the layers the dominant functions are ReLU or Swish. In contrast to other experiments, here there are clear preferred functions from the beginning when in other experiments, usually a mix of functions was preferred in the first layers. This might be because the training was longer or because of the specific architecture, meaning that when using the mechanism for searching for one best activation function, a longer training might benefit. However, the pattern still remained that the first and last layers differ more from ReLU. In many cases also Swish was the preferred function, validating that there are benefits to using Swish.

For the second activation functions set, as can be seen in Figures 40 and 41 the results were mostly the same, except in some cases Softmax was also preferred and in the last

layer, the linear function was the dominant function as was often also the case for other experiments.

4.3 Robotics

The last experiments are about learning robotics tasks using reinforcement learning. The goal was to see if the trainable activation functions would also give a benefit in the case of robotics tasks. The task that had to be learned was CartPole-v0 from the OpenAI Gym environments [BCP+16]. The algorithms used were Deep Q-Learning (DQN) [MKS+13] and Proximal Policy Optimization (PPO) [SWD+17].

The model used for both algorithms had two hidden layers with 64 nodes each. Behind both of the layers was the layer level trainable activation function. Two sets of activation functions were tested - the basic set and the set with more activation functions. Three derived architectures were also trained in addition to initial training with the basic and more activation functions sets. One was derived from the results of training the layer level activation function using the basic activation function set with $k = 4$, meaning the weights were fixed and the model was trained again. The other ones were derived from the results of training the layer level activation function using the more activation functions set with $k = 1$ meaning only the most preferred activation functions remained for each of the layers and $k = 11$ meaning the activation function weights were fixed. A baseline was trained using the tanh activation function for DQN and ReLU activation function for PPO. Each experiment was repeated 5 times and rewards were averaged because of the higher degree of randomness in reinforcement learning coming from choosing the actions more randomly at the beginning. The learned activation functions presented are from one specific experiment as an example. It is separately mentioned if much variation for the learned activation function weights occurred in other runs. The performance evaluation was done by comparing the mean rewards for each configuration. The mean rewards were calculated by taking a mean of the rewards from all of the runs.

4.3.1 Deep Q-learning

The first algorithm tested was DQN as implemented in the keras-rl2 package [McN19]. The Boltzmann policy was used as the exploration policy, the target model update rate was 0.01 and the network optimizer was Adam with a learning rate of 0.001. The model was trained for 50000 steps.

In Figure 26 can be seen the mean rewards by episode for each type of activation function used and the mean rewards with standard deviations can be seen in Table 16. It can be seen that the highest mean rewards were achieved by using two of the derived architectures, followed by tanh. The highest mean reward belongs to the models with activation functions derived from training on the more activation functions set with $k = 1$, which resulted in an increase of 10 points over the baseline of tanh. Training the model

for the first time using layer level activation function with basic and more activation functions resulted in a lower mean reward. Since the standard deviations were quite high, permutation tests were conducted, with the null hypothesis being that the baseline mean rewards and the specific model mean rewards are drawn from the same distribution, and the alternative hypothesis being that the baseline mean reward is less than the specific model's mean reward. With a significance level of 0.05, the null hypothesis failed to be rejected for all types of trainable activation functions used, meaning the increases in mean rewards were not statistically significant. The p-values of the tests can be seen in Appendix II.I.

Table 16. Mean rewards of DQN training

Act. fn. set	**Mean reward**
Tanh	135.7 ± 6.5
Basic	121.9 ± 10.8
More	124.5 ± 9.9
Derived from basic (k = 4)	135.7 ± 8.8
Derived from more (k = 1)	145.3 ± 15.5
Derived from more (k = 11)	131.8 ± 9.6

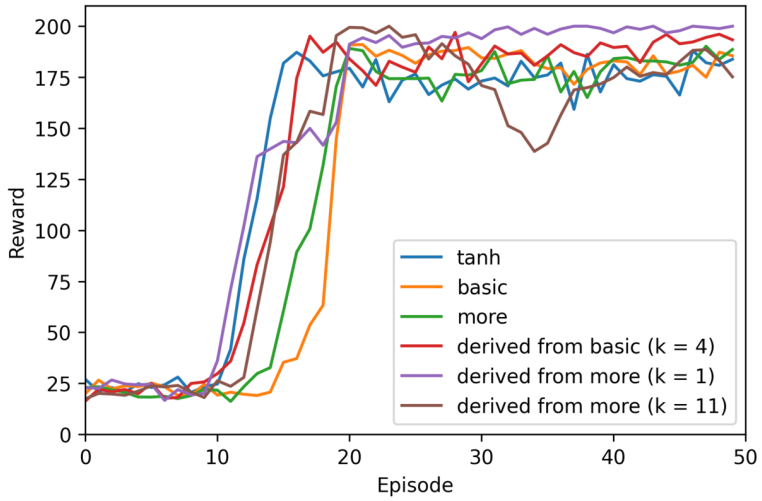

Figure 26. DQN rewards by episodes

The learned activation functions' weights and shapes can be seen below in Figure 27. It can be seen that the most preferred function for both layers was tanh, however, the first layer also had linear and sigmoid in the mix. The search worked as the default or known

best from these functions was found. If we were to derive a final architecture with $k = 1$ then tanh would remain for both layers.

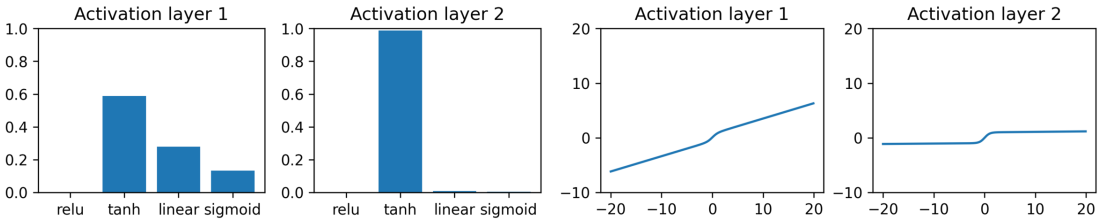

Figure 27. Learned activation functions' weights and shapes

The learned activation functions' weights and shapes for more activation functions set can be seen in Figure 28. In this case there was no clear dominant function by weight, however for the first layer SeLU was most preferred and for the second layer ELU was most preferred. Deriving the final architecture with $k = 1$ from using more activation functions resulted in using SeLU and ELU for the activation functions and gave the highest mean reward as can be seen from Table 16.

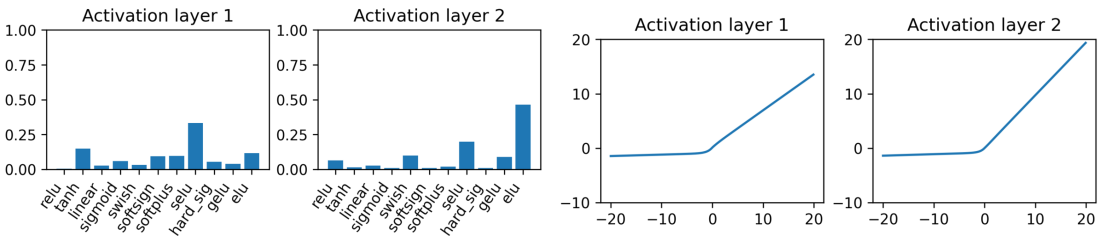

Figure 28. Learned activation functions' weights and shapes

4.3.2 Proximal Policy Optimization

The second algorithm tested was PPO as implemented in the Keras example [Chr21]. The policy learning rate was 0.003 and the model learning rate was 0.01. The model was also trained for 50000 steps. In Figure 29 can be seen the rewards by episode for each type of activation function used and the mean rewards with standard deviations can be seen in Table 17. The highest mean reward was achieved by using the already learned activation functions derived from training using the more activation function set with $k = 1$ where SeLU was chosen as the activation function for the first layer and ReLU for the second layer. Using it resulted in a 2 point increase over the baseline of ReLU. Training the model initially using layer level activation function with basic and more activation functions resulted in slightly lower mean rewards. Since the standard deviations were quite high, permutation tests were conducted, with the null hypothesis

being that the baseline mean rewards and the specific model mean rewards are drawn from the same distribution, and the alternative hypothesis being that the baseline mean reward is less than the specific model's mean reward. With a significance level of 0.05, the null hypothesis failed to be rejected for all types of trainable activation functions used, meaning the increases in mean rewards were not statistically significant. The p-values of the tests can be seen in Appendix II.II.

Table 17. Mean rewards of PPO training

Act. fn. set	**Mean reward**
ReLU	173.4 ± 3.2
Basic	158.4 ± 9.2
More	161.6 ± 11.7
Derived from basic (k = 4)	172.9 ± 8.7
Derived from more (k = 1)	175.7 ± 4.1
Derived from more (k = 11)	174.5 ± 3.4

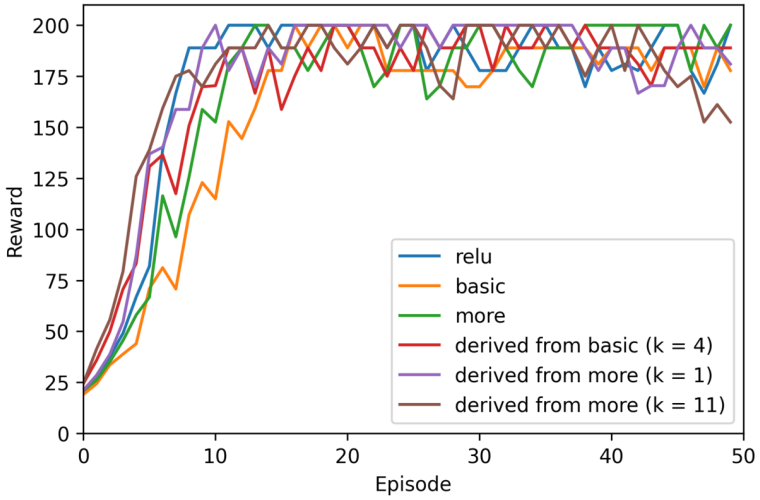

Figure 29. PPO rewards by episodes

The learned activation functions' weights and shapes can be seen below in Figure 30. It can be seen that the most preferred function for both layers was ReLU. The default or known best from these functions was found. If we were to derive a final architecture with $k = 1$ then ReLU would remain for both layers.

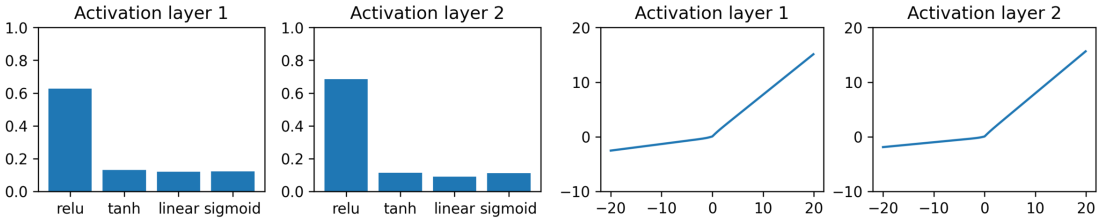

Figure 30. Learned activation functions' weights and shapes

The learned activation functions' weights and shapes for more activation functions set can be seen in Figure 31. In this case, ReLU was only preferred in the second layer and the first layer was a mix of all functions. In other runs of the same experiment, the layers were sometimes switched - ReLU was dominant in the first layer and the second layer was a mix of all the functions.

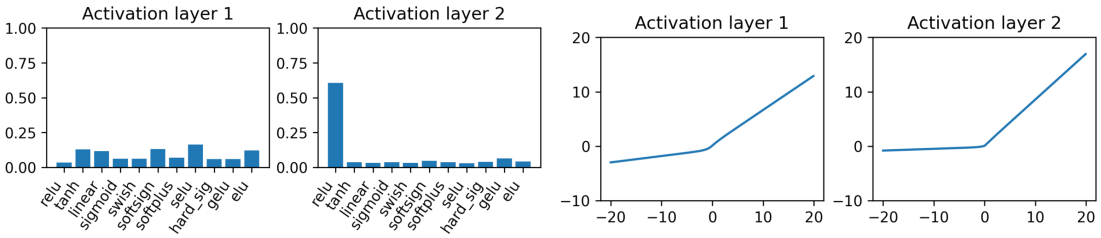

Figure 31. Learned activation functions' weights and shapes

4.3.3 Conclusion

It seems that for both algorithms using the trainable activation function for initial training did not increase the mean rewards of learning the task. However, using the trainable activation function found the optimal activation function and deriving the final architecture or fixing the weights of the learned activation function and training again seemed to bring the mean rewards over the baseline. In the case of DQN, using the derived architecture resulted in 10 point increase over the baseline out of 200 maximum, and in the case of PPO, it resulted in 2 point increase over the baseline. However, the increases in mean rewards were not statistically significant. In the case of more difficult tasks, training for the first time using the trainable activation functions may offer more benefits than seen for this task.

5 Conclusion

Using the proposed trainable activation function improved the performance for both image classification and robotics tasks. In the case of CIFAR10 with the small model, the test accuracy improved by 5% and with the ResNet model the accuracy improved 1% over the baseline using the layer level trainable activation functions for initial training. For ImageNet the accuracy improved approximately 1% over the baseline. Using kernel and neuron level trainable activation functions in the case of image classification usually resulted in worse performance than using layer level activation functions, which might be because of the significant increase of parameters over the layer level and baseline function that would take longer to train. Using a larger activation functions set for the trainable activation functions allowed the model to increase the train accuracy and fit the training data better. In the case of the robotics task, for initial training, the use of trainable activation functions did not increase the mean reward. When using already trained activation functions, the mean reward for the CartPole task increased by 10 points of 200 maximum over the baseline with Deep Q-learning and increased by 2 points over the baseline for Proximal Policy Optimization, however, the increases in mean rewards were not statistically significant.

For image classification tasks the layer level activation functions for the first layer preferred to be a mix of different functions and for regularized models, the last layer was often linear or almost linear. For the middle layers most often ReLU or something similar was preferred. However, the training did not function as a successful search for the one most preferred function for each layer when training the small model on CIFAR10 - choosing the most preferred function and training the model again reduced the performance significantly, because for some of the layers the function chosen was tanh or sigmoid, which decreased the performance. Fixing the weights after training using the trainable activation function did not bring better results as well than just training using the trainable activation function. Optimizing the activation function weights on validation loss in the case of the small model on CIFAR10 did not bring any additional performance gain for the initial training and training the model again using trained activation functions did not increase the performance as was the case for regular optimizing on training loss. In the case of using a larger set of activation functions in the trainable activation function for larger models, training longer could help the model learn to prefer one out of the other functions stronger.

For robotics tasks using the layer level trainable activation functions for initial training did not increase the mean rewards but the training found the default activation function used for that type of algorithm. When the set of activation functions was larger, it found more optimal activation functions for the tasks that increased the mean rewards over the baseline.

In conclusion, it was found that for image classification tasks the trainable activation functions increase the accuracy already in initial training. For robotics tasks, using them

for initial training did not increase the mean reward, but the mean reward increased when using the already learned activation functions or choosing the most preferred activation function for training the task again. It follows, that when using a neural network to learn a new task, it can be certainly useful to use this type of trainable activation function in the network. If it does not find one activation function to use for the task, it is probable that it could still increase the performance of the model.

For future work, more difficult robotics tasks could be explored to see if in initial training the trainable activation functions help more than in the case of the simple task explored here. For image classification, training or running the search longer could be tested for the model to learn to prefer activation functions more strongly in certain cases where more activation functions are used. Experimenting with the Softmax temperature could also be used to help the model learn to prefer the activation functions better and faster.

References

[ADIP21] Andrea Apicella, Francesco Donnarumma, Francesco Isgrò, and Roberto Prevete. A survey on modern trainable activation functions. *Neural Networks*, 138:14–32, Jun 2021.

[BCP+16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.

[Chr21] Ilias Chrysovergis. Implementation of a Proximal Policy Optimization agent for the CartPole-v0 environment. `https://keras.io/examples/rl/ppo_cartpole/`, 2021. Last accessed 14.05.2022.

[DP16] Michael Dushkoff and Raymond W. Ptucha. Adaptive Activation Functions for Deep Networks. In *Computational Imaging*, 2016.

[EWL16] Carson Eisenach, Zhaoran Wang, and Han Liu. Nonparametrically learning activation functions in deep neural nets. 2016.

[GBB11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

[HAB19] Matthias Hein, Maksym Andriushchenko, and Julian Bitterwolf. Why ReLU networks yield high-confidence predictions far away from the training data and how to mitigate the problem. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 41–50, 2019.

[Hor91] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

[Hu92] Z Hu. The study of neural network adaptive control systems. *Control and Decision*, 7:361–366, 1992.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[KH+09] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[Kot17] Raghavendra Kotikalapudi. Keras-ResNet. https://github.com/raghakot/keras-resnet, 2017. Last accessed 14.05.2022.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[LSY18] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[Mat18] Tambet Matiisen. What neural networks actually do? https://neuro.cs.ut.ee/what-neural-networks-actually-do/, 2018. Last accessed 14.05.2022.

[McN19] Taylor McNally. Keras-RL2 library. https://github.com/taylormcnally/keras-rl2, 2019. Last accessed 14.05.2022.

[MKS+13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. *arXiv preprint arXiv:1312.5602*, 2013.

[Pyt17] The Glowing Python. Solving the Two Spirals problem with Keras. https://glowingpython.blogspot.com/2017/04/solving-two-spirals-problem-with-keras.html, 2017. Last accessed 14.05.2022.

[RDS+15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[RZL17] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[SBF+18] Leon René Sütfeld, Flemming Brieger, Holger Finger, Sonja Füllhase, and Gordon Pipa. Adaptive Blending Units: Trainable Activation Functions for Deep Neural Networks. *arXiv preprint arXiv:1806.10064*, 2018.

[SWD+17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[Ten22] Tensorflow. Tensorflow tutorials. Convolutional Neural Network (CNN). https://www.tensorflow.org/tutorials/images/cnn, 2022. Last accessed 14.05.2022.

Appendix

I. ResNet learned activation functions graphs

I.I. Basic activation functions of model trained on CIFAR10

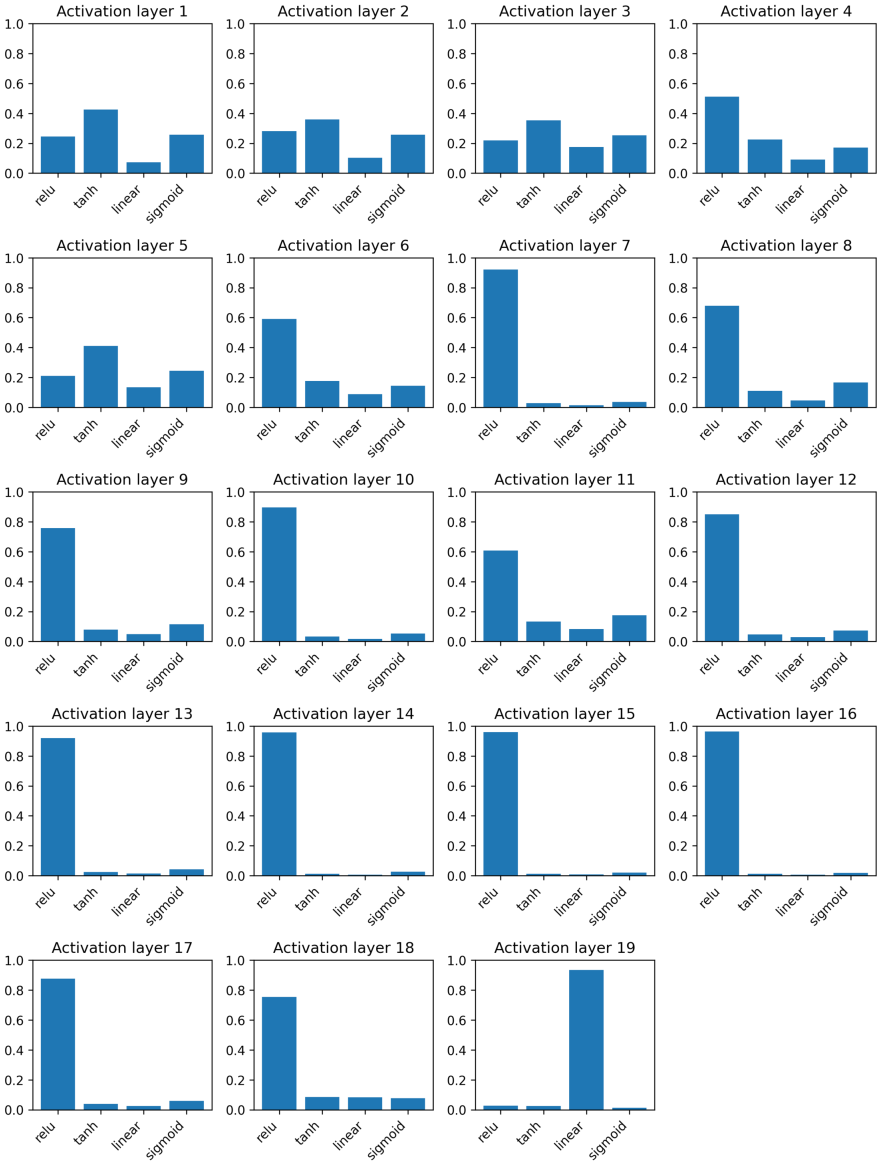

Figure 32. Learned activation functions' weights

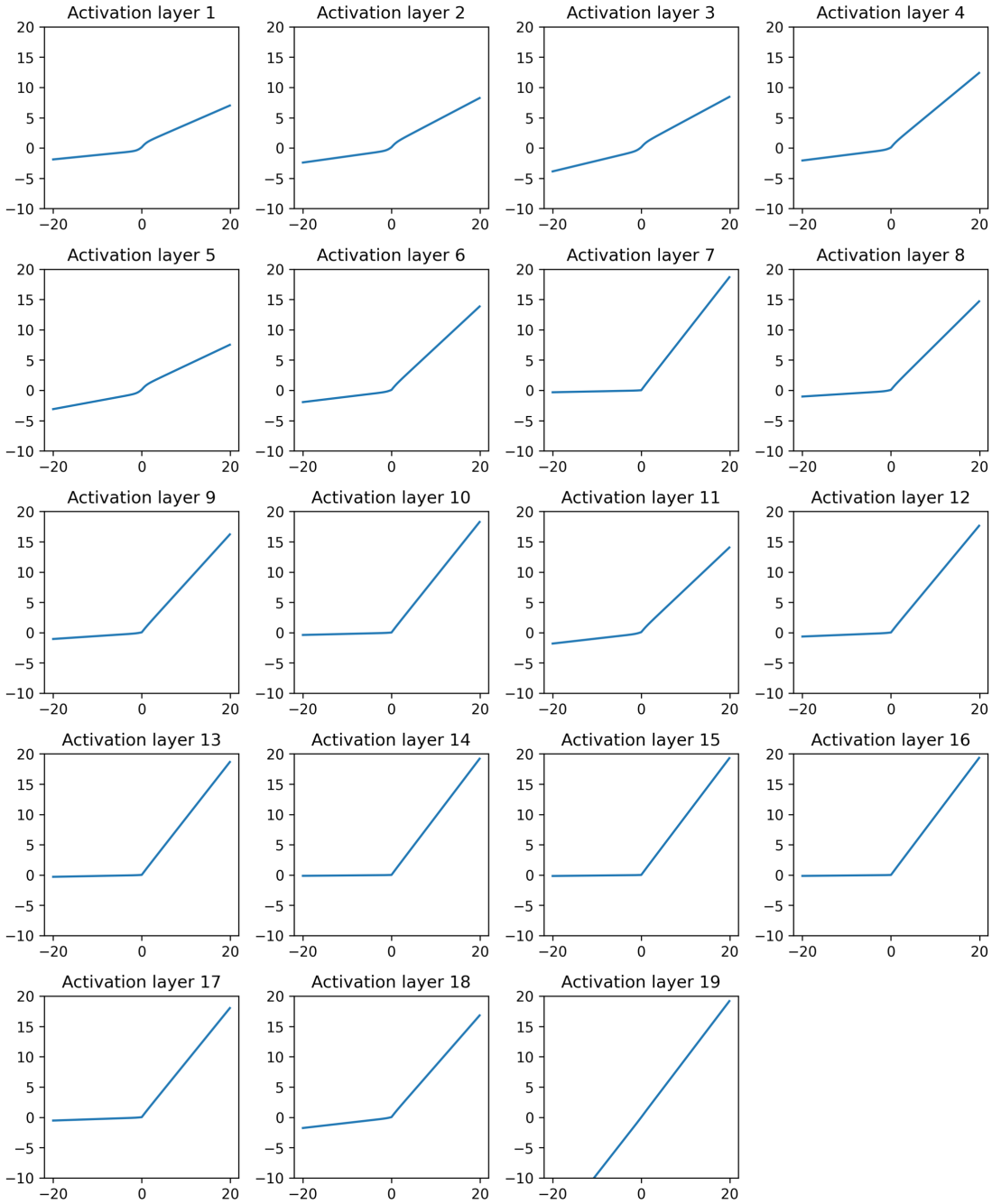

Figure 33. Learned activation functions' shapes

I.II. More activation functions of model trained on CIFAR10

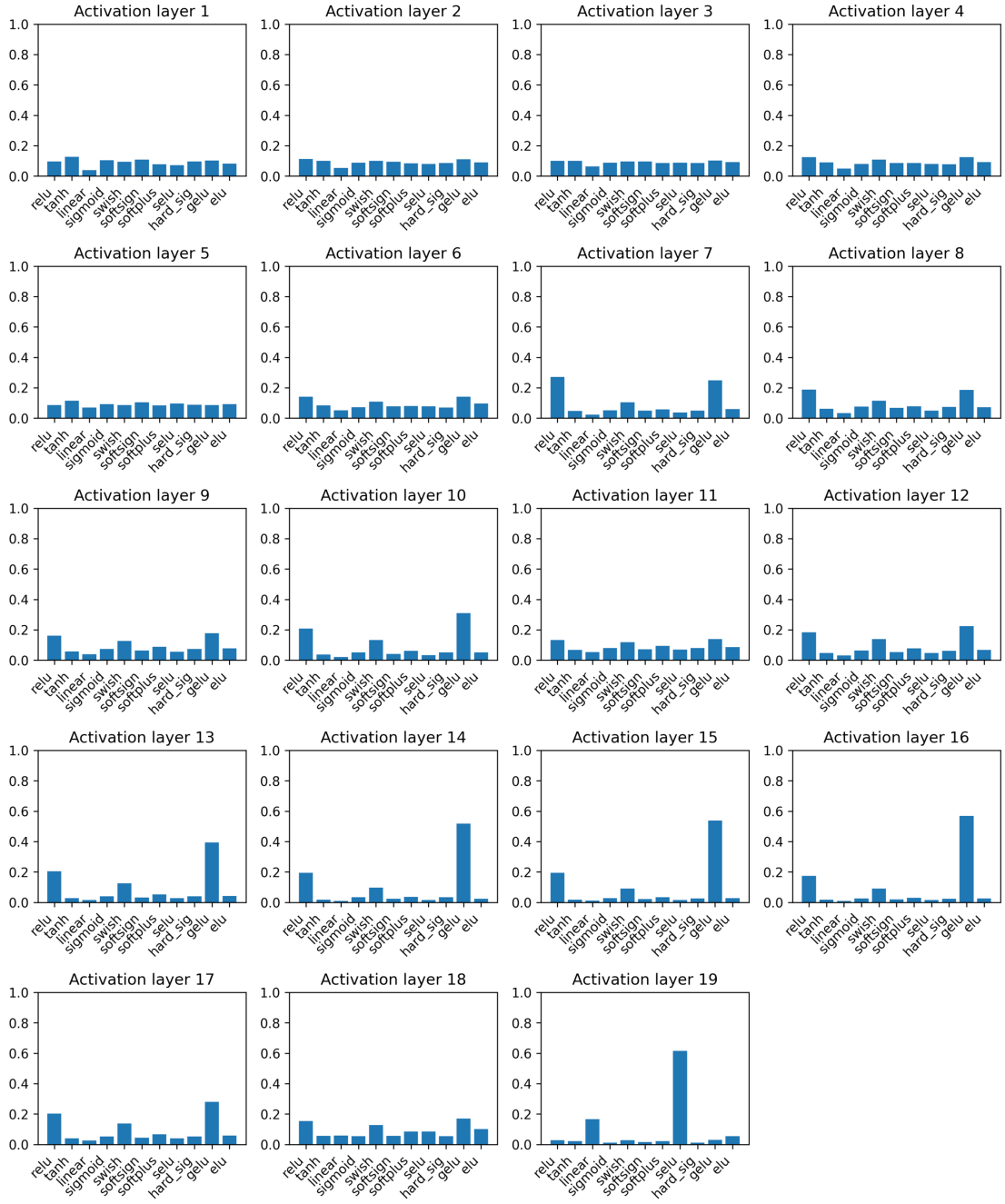

Figure 34. Learned activation functions' weights

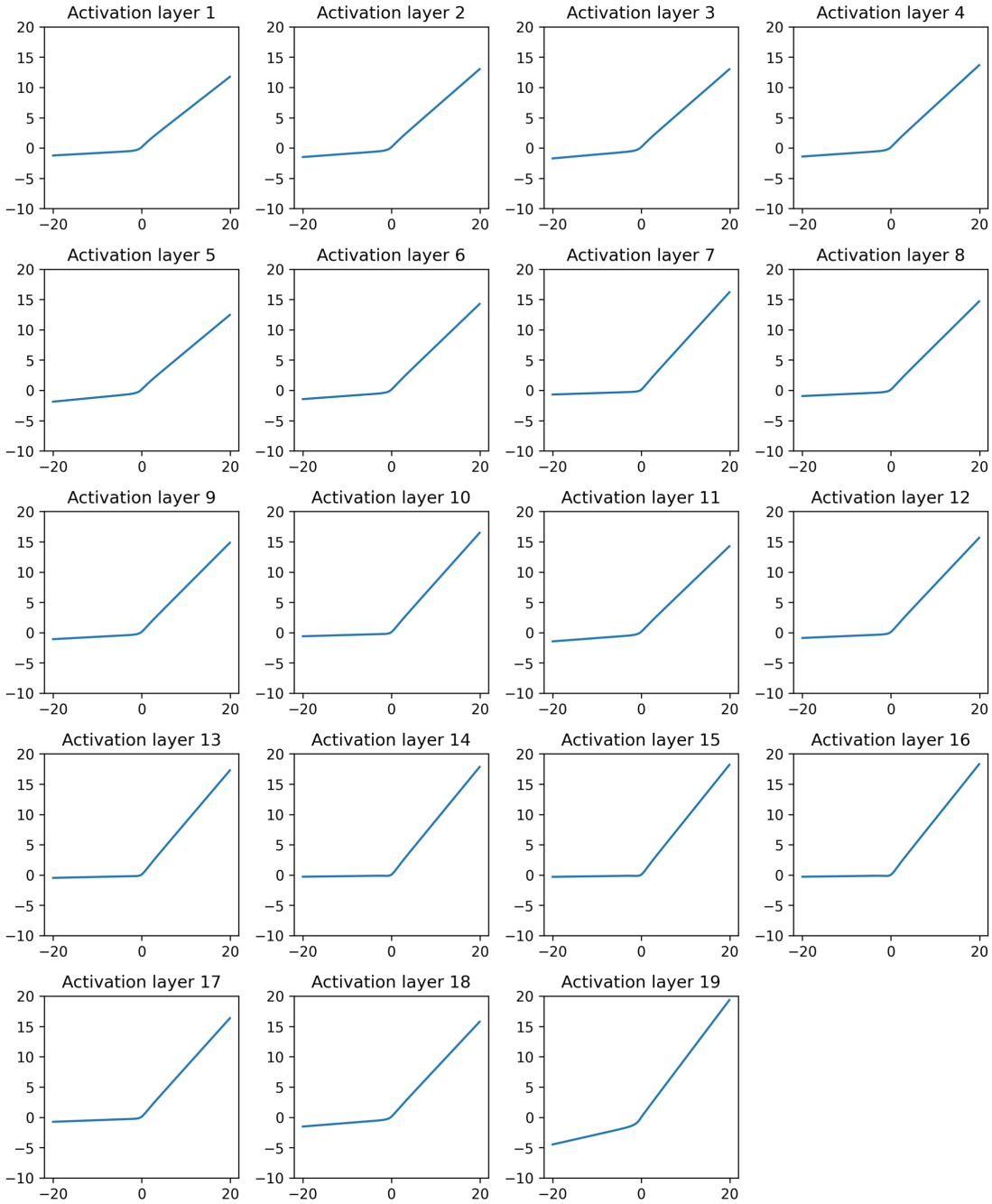

Figure 35. Learned activation functions' shapes

I.III. All activation functions of model trained on CIFAR10

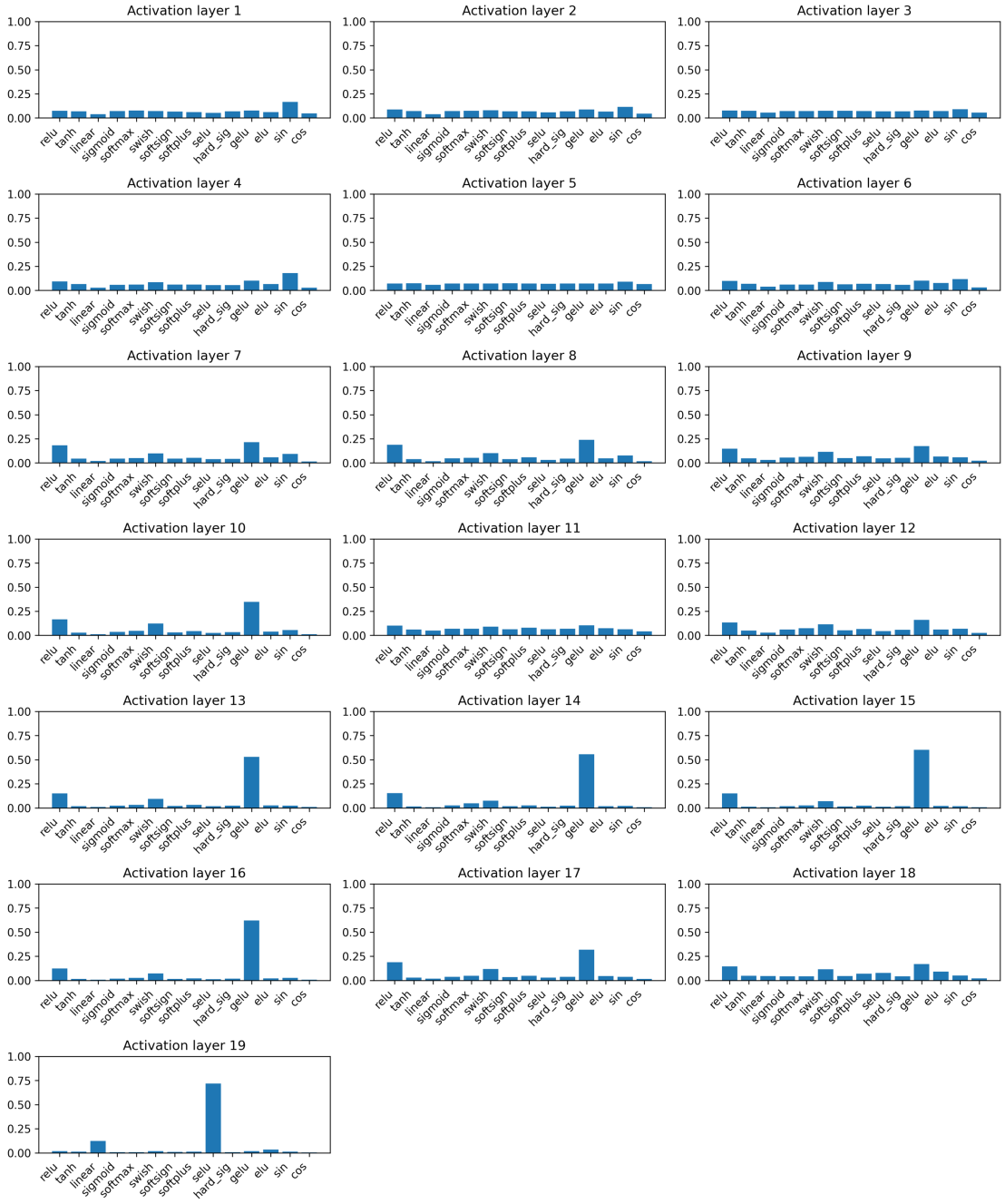

Figure 36. Learned activation functions' weights

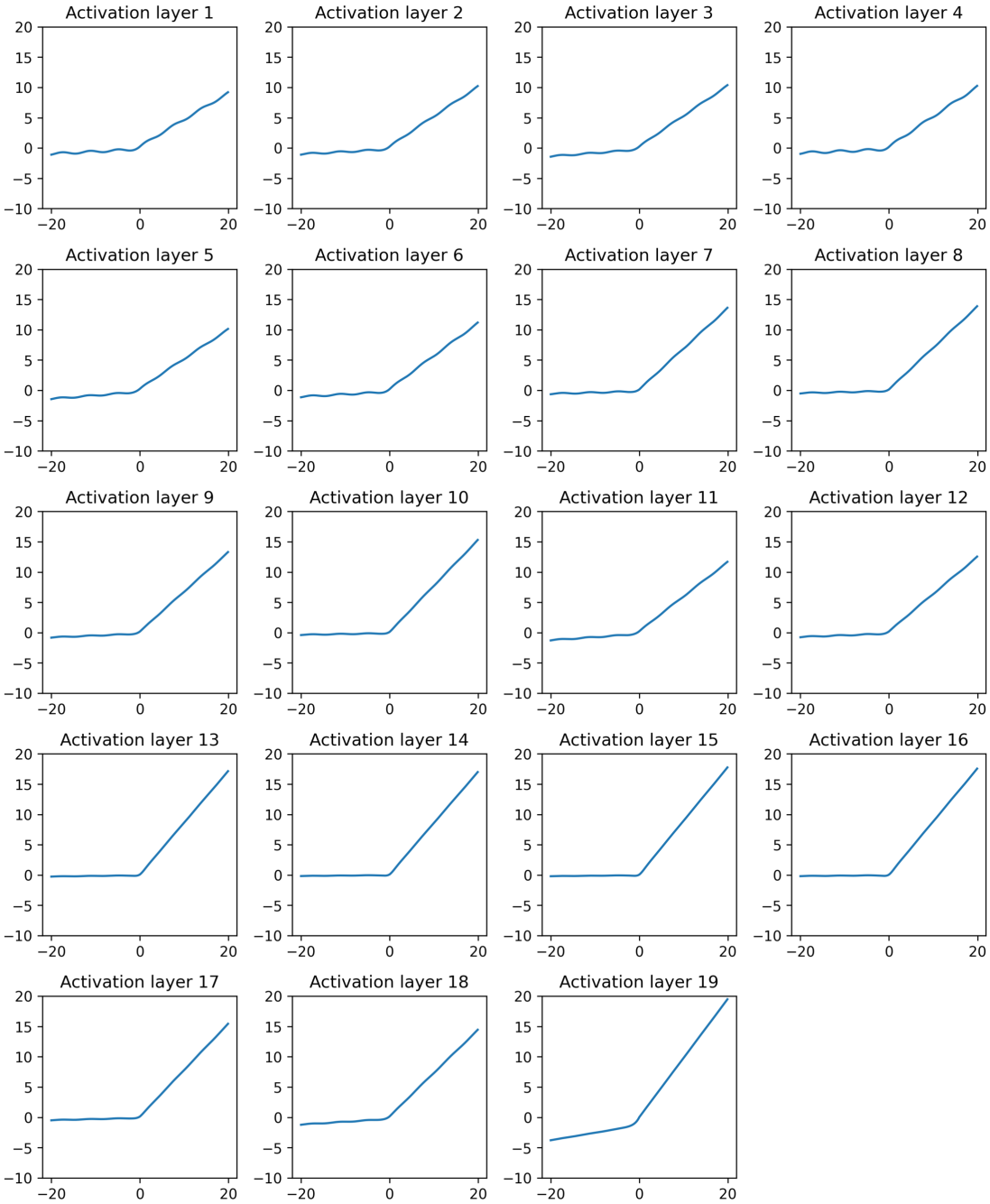

Figure 37. Learned activation functions' shapes

I.IV. Activation functions of models trained on ImageNet

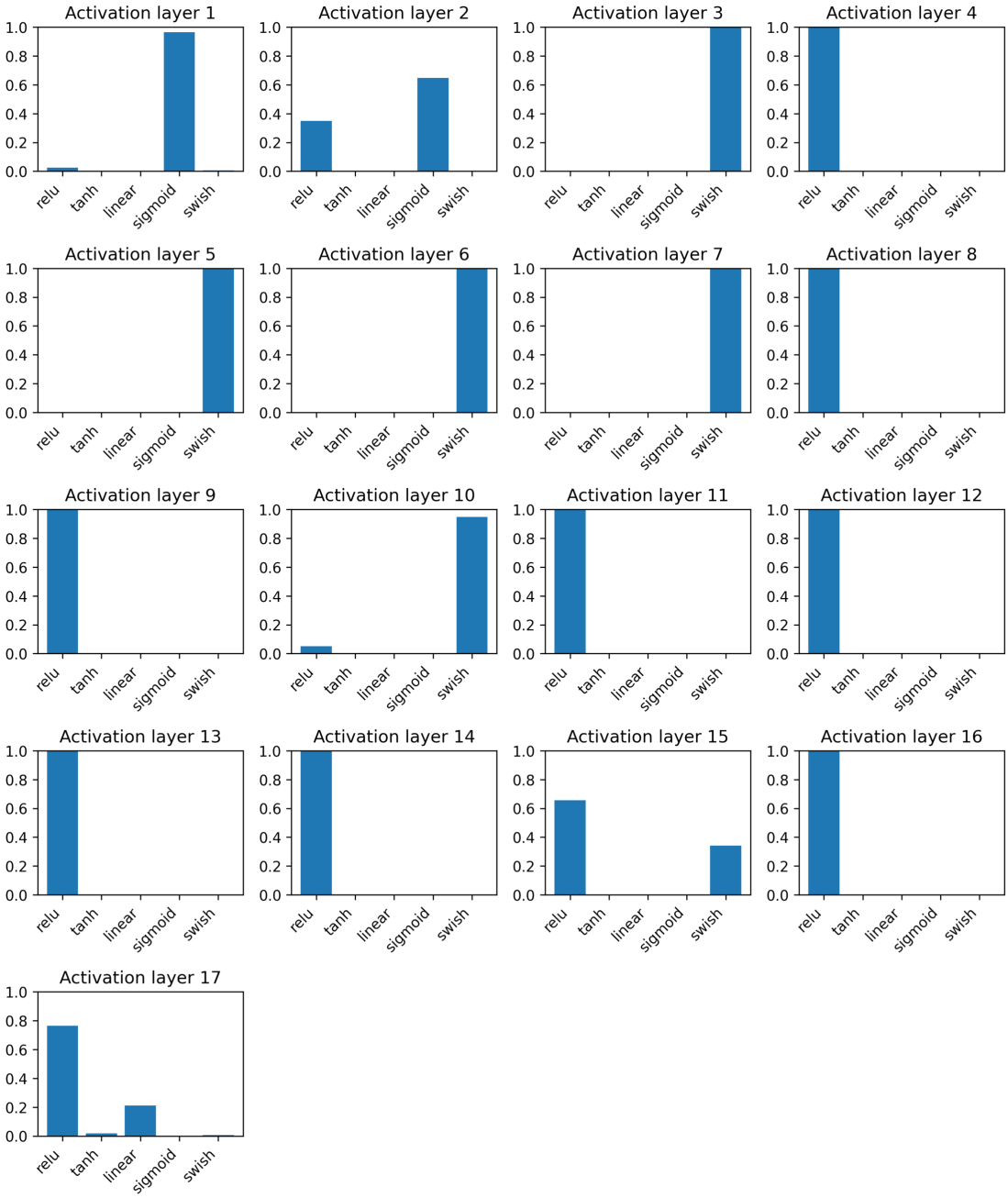

Figure 38. Learned activation functions' weights for set A

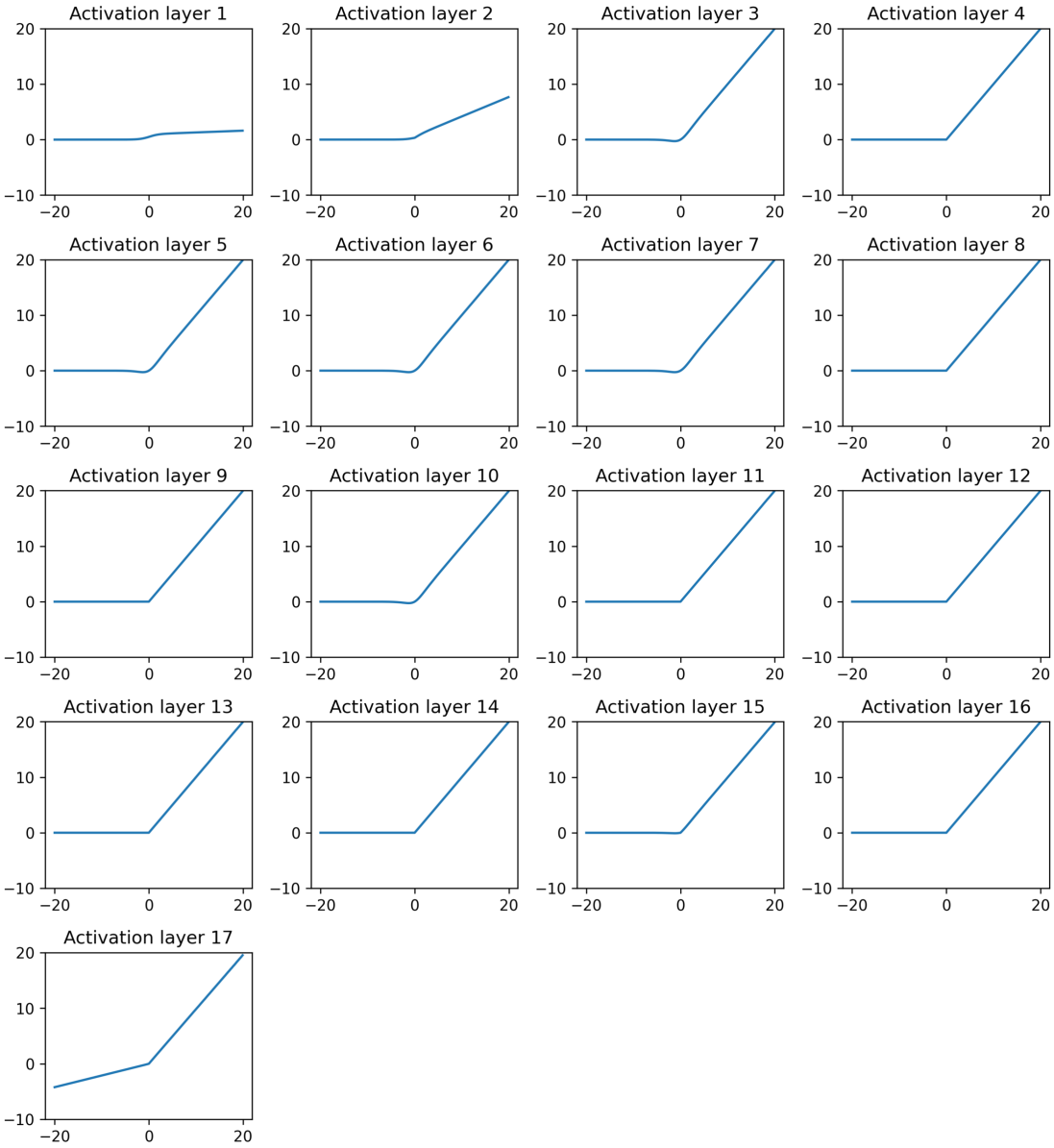

Figure 39. Learned activation functions' shapes for set A

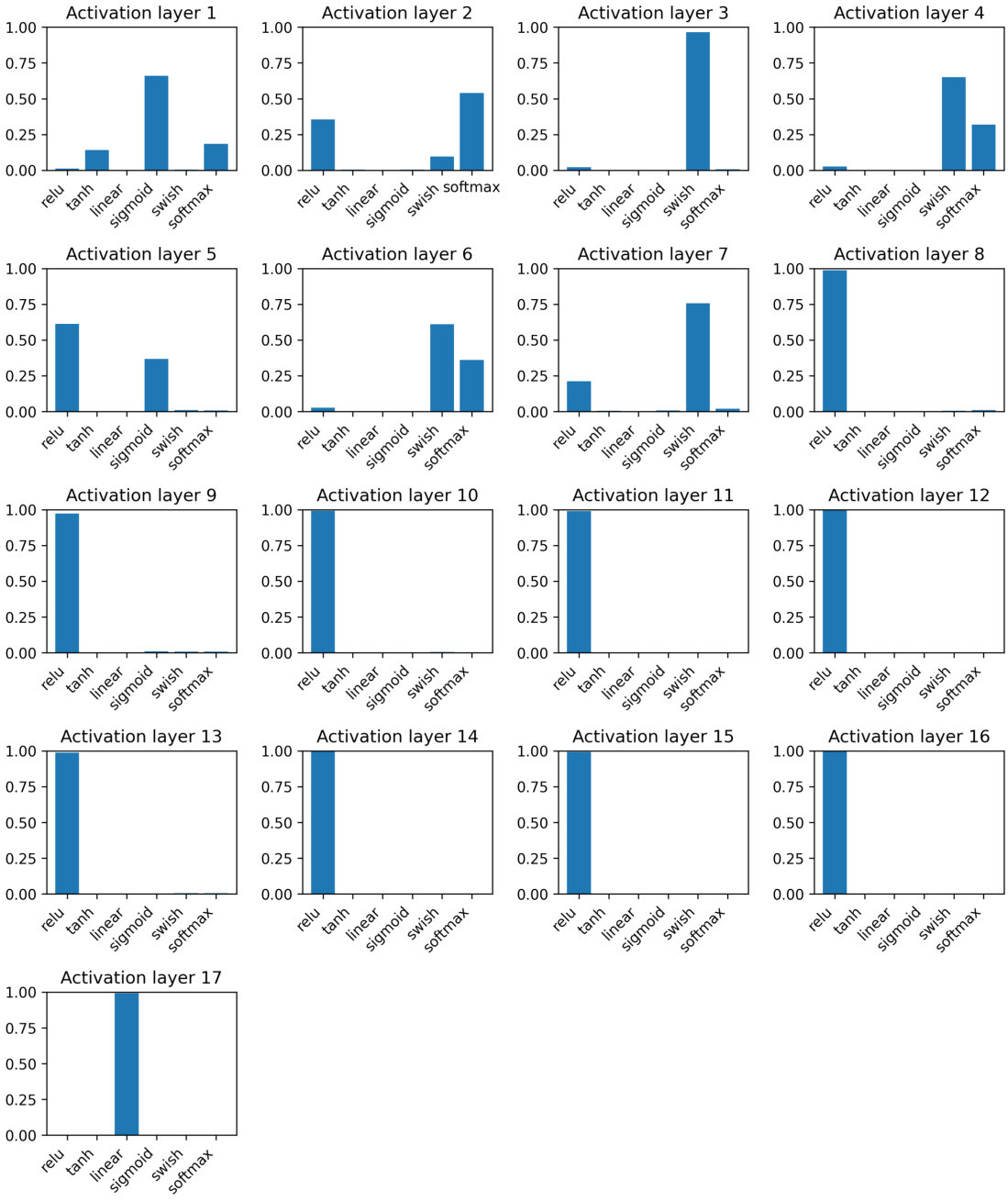

Figure 40. Learned activation functions' weights for set B

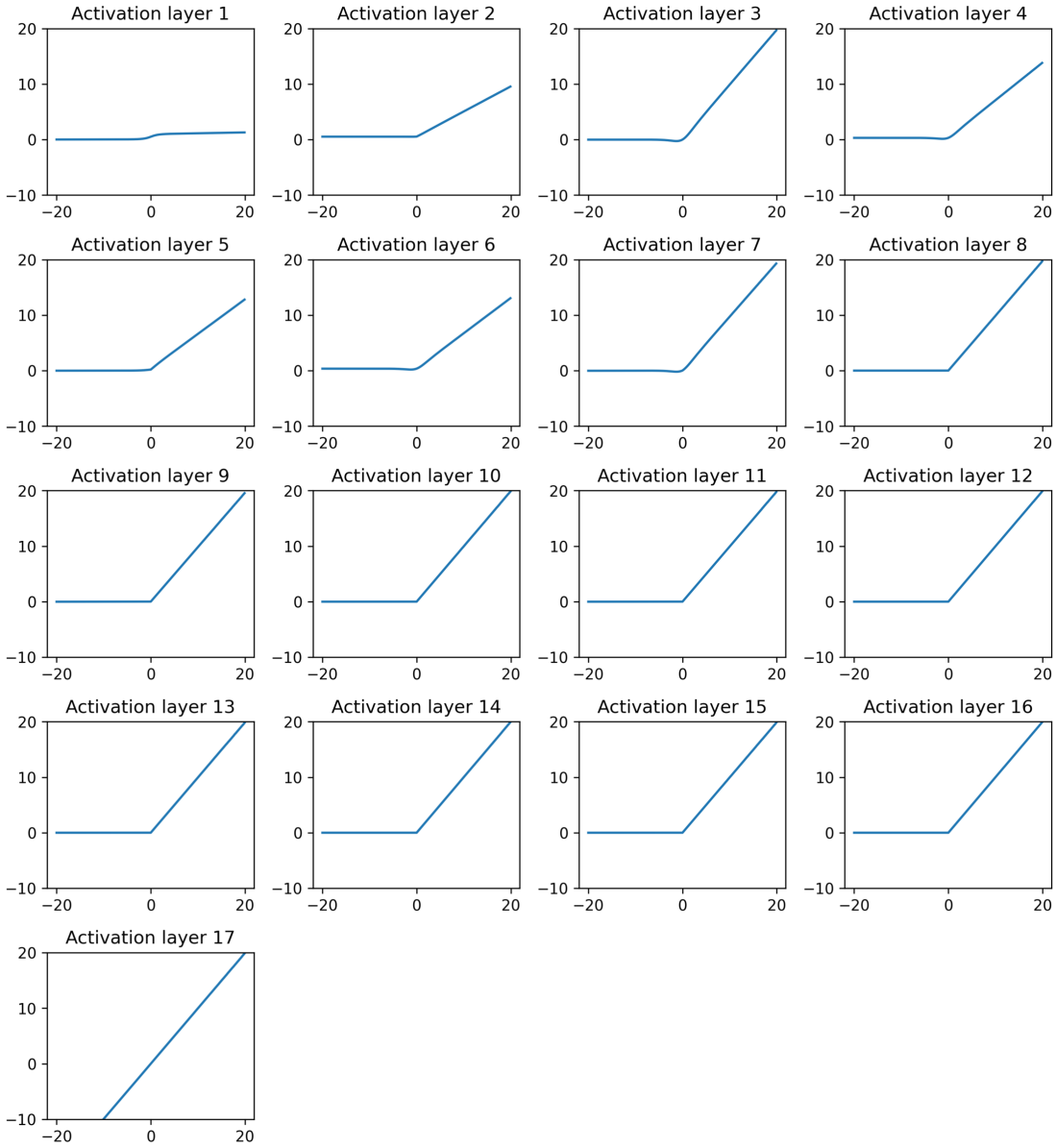

Figure 41. Learned activation functions' shapes for set B

II. Permutation tests' p-values

II.I. Deep Q-learning

Table 18. p-values of permutation tests on DQN training results

Model type	**p-value**
Basic	0.976
More	0.960
Derived from basic (k = 4)	0.504
Derived from more (k = 1)	0.131
Derived from more (k = 11)	0.730

II.II. Proximal Policy Optimization

Table 19. p-values of permutation tests on PPO training results

Model type	**p-value**
Basic	0.992
Normal	0.972
Derived from basic (k = 4)	0.524
Derived from more (k = 1)	0.214
Derived from more (k = 11)	0.317

III. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Marti Ingmar Liibert**,
(author's name)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   In Search of the Best Activation Function,
   (title of thesis)

   supervised by Tambet Matiisen.
   (supervisor's name)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Marti Ingmar Liibert
17/05/2022