Using Adaptive Activation Functions in Pre-Trained Artificial Neural Network Models

Yevgeniy Bodyanskiy[a], Serhii Kostiuk[a]

[a] *Kharkiv National University of Radio Electronics, Nauky ave 14, Kharkiv, 61166, Ukraine*

Abstract

Researchers continue developing new artificial neural network models and network training methods for various data processing tasks. The current research areas include the development of novel network architectures and activation functions. As the complexity of the models increases, researchers tend to use pre-trained models as the basis for new solutions.

This paper introduces the adaptive activation function replacement method for pre-trained artificial neural network models. The method enables the usage of adaptive functions instead of their non-adaptive counterparts without spending time training the model from scratch. The effectiveness of activation function replacement is evaluated on the image recognition task using a CNN in PyTorch and the CIFAR-10 dataset.

Keywords

Adaptive activation function, pre-trained artificial neural network model, transfer learning, deep neural network

1. Introduction

Artificial neural network models have become essential to modern data processing systems [1]. Convolutional neural networks (CNNs) serve as a core of image processing systems, enabling image processing at scale [2] and machine-assisted driving [3]. Recurrent neural networks (RNNs) can process sequential data, including video feeds [4] and historical data from industrial machinery [5]. The attention-based models, including the Transformer-based models, show state-of-the-art results in natural language translation [6], code generation [7], and language modeling [8-10].

Activation functions provide non-linearity to the network models. While the models in academic studies often use continuous activation functions (Tanh, Sigmoid, etc.), the commercial models tend to employ piece-wise linear activation functions (ReLU, PReLU, Hard Sigmoid, Hard Tanh, etc.) due to their lower computational complexity. At the same time, researchers continue developing new activation functions for different neural network architectures and data processing tasks [11].

One area of research is related to adaptive activation functions for deep neural networks. While networks with adaptive activation functions tend to perform better than the base variants [12], such networks are usually trained from scratch, limiting their applicability in production.

To achieve better performance on the selected datasets, researchers tend to increase the number of trainable parameters in the model, add various layers to the network, and experiment with the overall structure of the neural network. An increase in the number of layers and parameters of the model leads to computational complexities in training and inference. As a result, there is a demand for optimizing the training process and using pre-trained models for new data processing tasks [13].

We introduce the adaptive activation function replacement method for pre-trained artificial neural network models. We demonstrate the method using a CNN model implemented in PyTorch [14]. We compare the performance of the derived network, with and without activation function fine-tuning, to the performance of the base network on the CIFAR-10 dataset [15].

ICST-2023: Information Control Systems & Technologies, September 21-23, 2023, Odesa, Ukraine.
EMAIL: yevgeniy.bodyanskiy@nure.ua (Y. Bodyanskiy); serhii.kostiuk@nure.ua (S. Kostiuk)
ORCID: 0000-0001-5418-2143 (Y. Bodyanskiy); 0000-0003-4196-2524 (S. Kostiuk)

2. Related Works

Several research areas are related to the topic studied in this paper: non-adaptive activation functions development, adaptive activation functions synthesis, network architecture synthesis and manipulation, neural network pre-training, transfer learning, and fine-tuning.

Works [11-12,16] provide an overview of modern activation functions. Work [16] focuses on adaptive activation functions and describes the impact of adaptive activation functions on the effectiveness of the resulting model. The work studies parameterized standard and ensemble-based functions. The work highlights that the extensive comparison of adaptive activation functions requires their evaluation on the same dataset in the same base neural network architecture. However, the researchers cite results from other papers instead of conducting an independent assessment.

Work [11] classifies activation functions by their characteristics, activation function properties, and target applications. As a piece-wise linear activation, ReLU benefits from simpler gradients and lower computational complexity. At the same time, ReLU shows moderately good results in the image processing and classification domains, leading to its common usage in commercial models. In contrast, the Sigmoid and Tanh activations are suboptimal for convolutional networks. The work studies Swish, ESwish, PReLU, and APL among the adaptive functions. As stated, the impact of additional parameters in adaptive activation functions on computational complexity is negligible compared to the overall number of parameters in a deep neural network. The authors compare the performance of models with different activation functions. The general model architecture remains the same across experiments, with activation functions being the only difference. The authors train the derived models from scratch and evaluate the performance on the same data set. The work highlights that the convergence of adaptive functions dramatically depends on their parameter initialization. In general, the models with adaptive functions demonstrate better convergence than their non-adaptive alternatives.

Works [17-21] introduce new adaptive activation functions for deep neural networks, include them in different neural network models, and study their effectiveness on various datasets. The authors in [17] combine elementwise attention with the ReLU activation function to create a learnable activation function. The researchers in [20] replace all fixed activation functions in a BERT-based Transformer architecture with an adaptive Rational Activation Function (RAF). When trained from scratch, the RAF networks perform better than GELU on language modeling tasks. Work [19] introduces a continuous adaptive alternative to the ReLU and SiLU activation functions and demonstrates its ability to change the form and amplitude during training. Work [20] presents a piece-wise adaptive fuzzy activation function. Depending on its parameters and the number of membership functions, this fuzzy activation function can serve as a piece-wise approximation of continuous activations, such as Tanh and Sigmoid, on a pre-defined interval. Work [21] introduces a universal adaptive activation function and studies its ability to replace other non-adaptive activation functions.

We note that works [17-21] only evaluate the effectiveness of models trained from scratch. The authors do not mention activation function replacement in pre-trained networks and do not reuse pre-trained parameters from non-adaptive implementations for further training.

Works [22-24] propose and describe different artificial neural network models for text processing, including the CNN, RNN, and Transformer variants. In [22], the authors propose variants of Transformer and ImageNet with Swish as an adaptive sigmoid-based activation function. As stated, the models with Swish “show strong performance” compared to non-adaptive activations on the language translation and image classification tasks. [23] introduces a deep CNN-based network for text sentiment analysis. The authors in [23] highlight the importance of unsupervised pre-training and learned embeddings for initializing deep text processing models.

Work [24] presents DeepMoji, an RNN-based model for sentiment analysis. While [24] describes the architecture and the principle of operation of DeepMoji, it is only available in a pre-trained form without the original data set. Hence, independent researchers cannot modify the model, train it from scratch, and replicate the results from the paper using the same data set.

Works [25-28] study architecture manipulation for pre-trained neural network models. [25] overviews model compression methods for efficient deployment to resource-constrained devices and specialized hardware accelerators. The authors of [26] review the knowledge distillation methods. The reviewed methods use the teacher-student approach when the original (teacher) model remains

unchanged while the teacher trains the new (student) model from scratch. In [27], the authors combine knowledge distillation with weight pruning for model compression. The authors apply fine-tuning to recover the model performance after pruning. Work [28] proposes the removal of weights close to zero and the associated connections between the nodes after each training epoch. To summarize, while works [25-28] study modifications in the neural network architecture, they do not study the activation function replacement in such architectures.

Works [13,29-31] study the application of fine-tuning, pre-training, and transfer learning techniques in data processing tasks. [13] overviews techniques for efficiently training large artificial neural network models, including pre-training and knowledge transfer. Some methods from this overview propose using pre-trained shallow models as the first layers of larger language modeling models. Other approaches include self-supervised and unsupervised pre-training of the whole model on a large dataset with fine-tuning on a target dataset. Overall, the authors highlight the importance and perspectives of pre-training for deep models like Transformers. Work [29] describes a multi-target evolutionary architecture search approach for CNN with weights sharing and sub-sampling of the target models. The trained network structure consists of pre-defined blocks, with a machine-readable configuration file as the structure definition. The authors apply fine-tuning on the target dataset to recover the accuracy of sub-sampled models. According to the paper and the reference implementation, the activation function remains fixed during the architecture search and training. Work [30] studies the contribution of individual layers in a recurrent Neural Machine Translation (NMT) model. The authors train a complete model on the bi-lingual out-of-domain OpenSubtitles2018 dataset, partially freeze the layers, and continue training on the target COPPA-V2 dataset. The experiment shows that the impact of individual layers during training is relatively small, as the model can still adapt to the new dataset with partially frozen layers. Work [31] overviews and evaluates the performance of transfer learning methods.

To our knowledge, the previous works do not study the methods of activation function replacement in pre-trained artificial neural network models and the effectiveness of such replacement.

3. Method Design

Let N be an n-layer artificial neural network model with sequentially connected layers. A combination of its layers L serves as a network description: $N = \{L_1, L_2, \dots, L_n\}$. Each individual layer L_i implements a linear transformation together with a piece-wise non-linear transformation, implemented by activation function f_i: $L_i = f_i(X_i W_i^T + B_i)$, where X_i – the input tensor, W_i – trainable weights for linear transformation, B_i – the bias tensor, and f_i – the activation function.

Let N_{orig} be the original pre-trained n-layer artificial neural network model, while N_{drv} – the derived network with its activation functions replaced. Hence N_{orig} and N_{drv} can be described as $N_{orig} = \{L_{orig,1}, L_{orig,2}, \dots, L_{orig,n}\}$ and $N_{drv} = \{L_{drv,1}, L_{drv,2}, \dots, L_{drv,n}\}$ correspondingly.

Depending on the implementation, networks N_{orig} and N_{drv} may share the same layers and activation functions for a subset of layers:

$$L_{orig,i} = L_{drv,i} = f_i(X_i W_i^T + B_i) \; \forall \; L_i \in L_{same}, L_{same} \subset L_{orig} \cap L_{drv}.$$

As an alternative, the derived model may have all activation functions replaced while sharing the same values of trainable weights:

$$L_{orig,i} = f_{orig,i}(X_i W_i^T + B_i), L_{drv,i} = f_{drv,i}(X_i W_i^T + B_i)$$
$$\forall \; L_{i,orig} \in L_{orig}, \; L_{i,drv} \in L_{drv}, L_{orig} \cap L_{drv} = \emptyset.$$

Assuming the neural network architecture is known, the activation functions $f_{orig,i}$ can be replaced with the corresponding adaptive alternatives $f_{drv,i}$. As the output of each new layer L_i depends on the output of the previous layer L_{i-1}, changes in the hidden layers may change the distribution of their output values, introduce errors to their output values, invalidate the trainable parameter values in each subsequent layer, and require re-training of the following layers.

The replacement activation function f_{drv} shall strictly follow the shape and amplitude of the original activation function f_{orig} to preserve the pre-trained parameter values W_i and B_i in the derived network. In other words, the difference between the outputs of the original and the replacement activation functions shall be zero for all input values z:

$$E_{f_i} = \left(f_{drv,i}(z) - f_{orig,i}(z)\right)^2 = 0 \; \forall \; \mathrm{z} \in \mathbb{R}\,.$$

In practice, we show that the original activation function can be replaced by its approximation, assuming that the difference between their outputs on the whole range of input values is relatively low:

$$E_{f_i} = \left(f_{drv,i}(z) - f_{orig,i}(z)\right)^2 < \varepsilon \; \forall \; \mathrm{z} \in \mathbb{R}\,.$$

3.1. Selection of Replacement Activation Functions

Achieving the zero difference between the outputs of the original and the replacement activation function is possible if the original activation function is a corner case of the replacement activation function. In this case, expressing the original activation function f_{orig} from the replacement adaptive activation function f_{drv} requires selecting the corresponding activation function parameters $P_{drv} = \{p_1; p_2; \dots; p_m\}$ so that:

$$f_{drv}(P_{drv}; z) = f_{orig}(z).$$

For example, Adaptive Hybrid Activation Function (AHAF) can exactly replace SiLU when $P_{drv} = = \{p_1; p_2\} = \{1; 1\}$:

$$\begin{gathered} f_{orig}(z) = \mathrm{SiLU}(z) = z\sigma(z), \\ f_{drv}(P_{drv}; z) = \mathrm{AHAF}(p_1; p_2; z) = p_1 z\sigma(p_2 z), \\ f_{drv}(P_{drv}; z) = \mathrm{AHAF}(1; 1; z) = 1 \cdot z\sigma(1 \cdot \; z) = z\sigma(z) = \mathrm{SiLU}(z). \end{gathered}$$

When parameter p_2 is close to $+\infty$, the sigmoidal part of AHAF gets close to the step function:

$$\lim_{p_2 \to +\infty} \sigma(p_2 z) = \sigma(+\infty \cdot z) = \begin{cases} 1, & z > 0 \\ 0, & z \le 0 \end{cases}$$

so that AHAF can approximate ReLU when $P_{drv} = \{p_1; p_2\} = \{1; +\infty\}$:

$$\begin{gathered} f_{orig}(z) = \mathrm{ReLU}(z) = \begin{cases} z, & z > 0 \\ 0, & z \le 0 \end{cases}, \\ f_{drv}(P_{drv};\, z) = \mathrm{AHAF}(1; +\infty; z) = 1 \cdot z\sigma(+\infty \cdot z) = \begin{cases} z, & z > 0 \\ 0, & z \le 0 \end{cases} = \mathrm{ReLU}(z) \end{gathered}$$

Modern computers have limited computation accuracy, so, in practice p_2 can be selected as a sufficiently large finite real number. Here and below, we use AHAF with $P_{drv} = \{p_1, p_2\} = \{1, 10^9\}$ as a sufficiently close approximation of ReLU:

$$f_{drv}(P_{drv}, z) = AHAF(1; 1 \cdot 10^9; z) = 1 \cdot z\sigma(10^9 \cdot z) \approx \begin{cases} z, & z > 0 \\ 0, & z \le 0 \end{cases} \approx ReLU(z).$$

The same approach applies to replacing other adaptive activation functions, assuming that the replacement activation function is a generalization of the original one. For example, AHAF can replace the Swish activation function, when $P_{orig} = \{p_{orig,1}\}$, $p_{drv,1} = 1$, $p_{drv,2} = p_{orig,1}$, $P_{drv} = \{p_{drv,1}; p_{drv,2}\} = \{1; p_{orig,1}\}$:

$$f_{orig}(P_{orig}; z) = \text{Swish}(p_{orig,1}; z) = z\sigma(p_{orig,1} \cdot z),$$
$$f_{drv}(P_{drv}; z) = \text{AHAF}(p_{drv,1}; p_{drv,2}; z) = p_{drv,1} \cdot Z\sigma(p_{drv,2} \cdot z),$$
$$f_{drv}(P_{drv}; z) = \text{AHAF}(1; p_{orig,1}; z) = 1 \cdot z\sigma(p_{orig,1} \cdot\ z) = \text{Swish}(p_{orig,1}; z).$$

In contrast, some adaptive activation functions are piece-wise linear activation functions. Such adaptive activation functions can be an exact replacement only for a subset of other piece-wise linear activation functions. For example, the F-neuron activation function can replace the HardTanh activation function, assuming they have the same definition interval $[z_{min}, z_{max}]$:

$$f_{orig}(z) = \text{HardTanh}(z) = \begin{cases} z_{min}, & z < z_{min} \\ z_{max}, & z > z_{max} \\ z, & otherwise \end{cases},$$
$$f_{drv}(P, z) = \text{FN}_{\text{act}}(P, z) = \sum_{j=1}^{m}(FN_{act,j}(z) \cdot p_j) = \begin{cases} z_{min}, & z < z_{min} \\ z_{max}, & z > z_{max} \\ z, & otherwise \end{cases},$$

where $\text{FN}_{\text{act}}(P, z)$ is the F-neuron activation function, $\text{FN}_{\text{act},j}(z)$ is the j-th membership function of $\text{FN}_{\text{act}}(P, z)$, $P = \{p_1, p_2, \dots, p_m\}$ – parameters of the F-neuron activation function, m – the total number of membership functions, $p_j = \text{HardTanh}(z_{min} + (z_{max} - z_{min})/(m-1) * (j-1))$.

When the original function is not piece-wise linear, the F-neuron activation function can only provide a piece-wise linear approximation on an interval with limited accuracy. For example, an F-neuron with 18 membership functions (1 left ramp, 16 triangle-shaped functions, 1 right ramp) can approximate Tanh on the $[-4.0;\ +4.0]$ interval with the maximum delta squared of $4 \cdot 10^{-4}$.

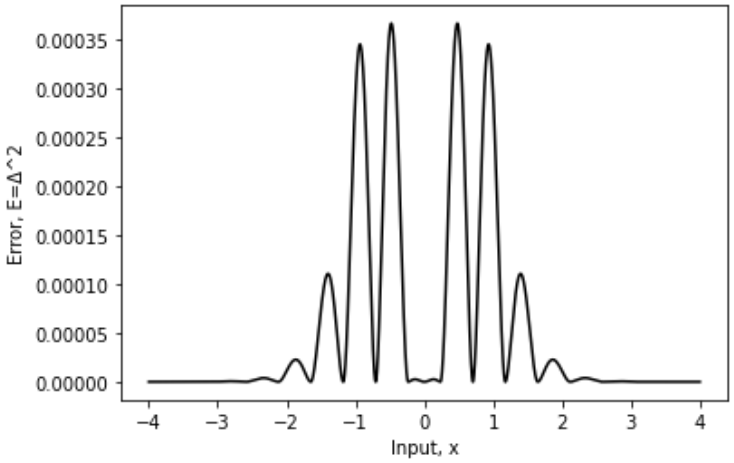

Figure 1: Delta squared error for an F-neuron activation initialized as Tanh with 18 membership functions compared to the base Tanh activation function.

In this case, the trainable parameters $P_{FN_{act}}$ are initialized as:

$$P_{FN_{act}} = \{p_1, p_2, \dots, p_{18}\},\ \text{m} = 18,$$
$$p_j = \tanh\left(-4.0 + \frac{4.0+4.0}{17} * (j-1)\right),$$
$$P_{FN_{act}} \approx \{-0.9993; -0.9992; \dots; -0.031; +0.031; \dots; \dots; 0.9992; 0.9993\}.$$

3.2. Practical Aspects of Activation Function Replacements

Researchers and companies use various data formats to describe, save and distribute their pre-trained neural network models. To improve interoperability and simplify the practical application of the model, the authors use various deep learning frameworks, such as PyTorch [14] and others. Generally, such

frameworks store their models in two separate pieces: the model description in code and the values of its trainable weights in the binary files.

Hence, the implementation of the activation function replacement method is a four-step procedure:

1. Restore the original neural network structure in memory using the stored model description.
2. Replace the original activation functions with their adaptive alternatives to get the derived network structure.
3. Load values of the original trainable parameters from the saved state.
4. Initialize the parameters of the adaptive functions so that the adaptive functions correspond to the original ones, as described in Section 3.1.

Figure 2 illustrates the network initialization and activation function replacement procedure.

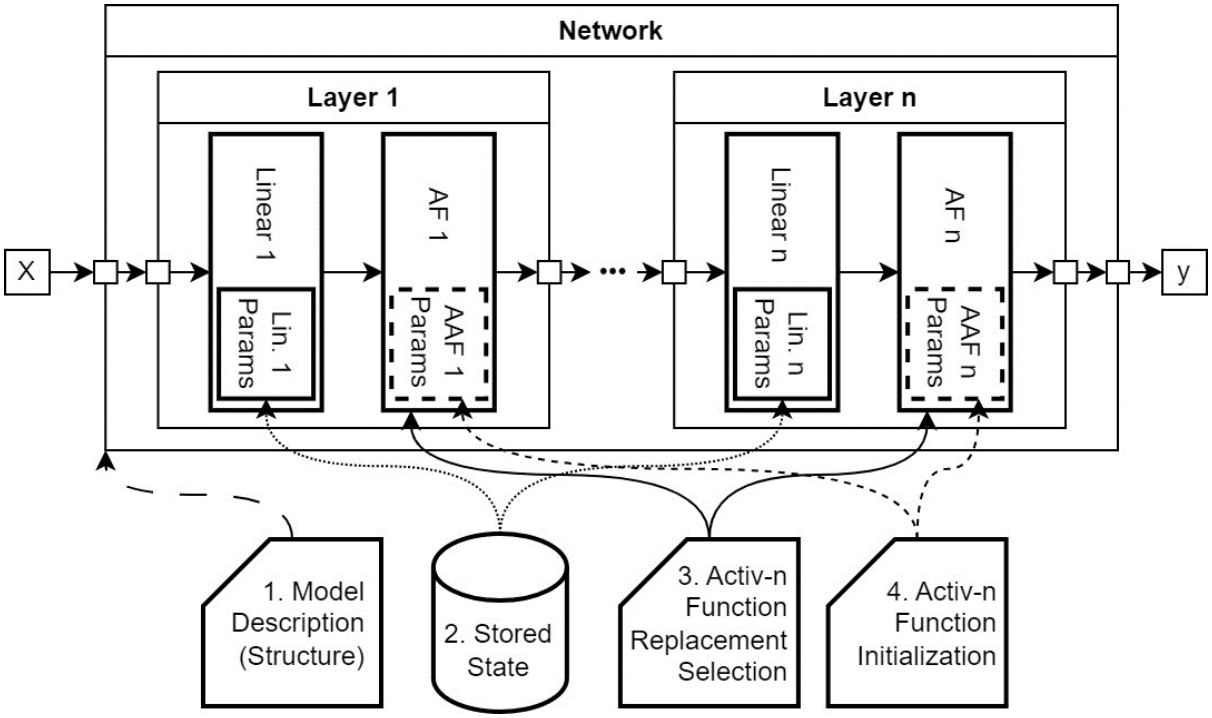

Figure 2: Network model initialization and activation function replacement from the description of the original model description, the stored state, and the replacement rules described in Section 3.1.

In practice, modern deep learning frameworks allow combining steps 1, 3, and 4. To combine the steps, the user shall define a new network in memory, use the same parameter names for common weights, pre-initialize adaptive activation functions with the corresponding parameters, and load the weights from the stored state. Figure 3 shows an example two-step implementation in PyTorch.

```
1    net = NetworkWithAdaptiveActivations()
     saved_state = torch.load(saved_state_file_path)
2    net.load_state_dict(saved_state, strict=False)
```

Figure 3: Loading pre-trained weights into a derived network with adaptive activation functions.

In this figure, “net” indicates a new network in memory, “saved_state_file_path” denotes the pre-trained model’s state path, and “strict=False” instructs PyTorch to load parameters of the pre-trained network, ignoring the adaptive activation function parameters missing from the original model.

3.3. Fine-Tuning Activation Function Parameters in a Pre-Trained Model

An imperfect approximation of the original activation function may cause degradation in the artificial neural network model’s performance (classification accuracy, loss function values, etc.). The replacement activation functions may provide an imperfect approximation of the original ones. The

adaptive activation function parameters can be fine-tuned separately from all other trainable parameters to improve the performance and compensate for possible approximation errors. The non-activation parameters shall be excluded from the training during the fine-tuning process. The following code sample illustrates the approach (figure 4).

```
for p in net.params():
    p.requires_grad = False

for p in net.activation_params:
    p.requires_grad = True

train_network(net, dataset)
```

Figure 4: Fine-tuning the adaptive activation function parameters.

Where "net" is an instance of the artificial neural network model, "dataset" is the training data set, "net.params()" returns all trainable parameters in the network, and "net.activation_params" returns the list of parameters that belong to adaptive activation functions.

In the following sections, we empirically study the effectiveness of networks with fine-tuned adaptive activation functions compared to the original networks and networks with adaptive activation functions that were trained from scratch.

4. Experiment

We evaluate the performance of various neural network variants on the CIFAR-10 [15] data set. Each element in the data set is a 3-channel image, 32x32 pixels in size. Each image in the data set corresponds to one of 10 possible object classes. We use the 5:1 split between the training and the testing data: 50000 images in the training subset and 10000 in the testing subset. During training, we apply image augmentation using a random horizontal flip and a random affine transformation (shifting) by at most 0.1 times vertically and horizontally.

The base neural network architecture is the VGG-like KerasNet [32] network. The architecture includes:

- 4 two-dimensional convolutional layers with the corresponding activation functions.
- 2 layers of 2D max pooling and 2D dropout between the pairs of convolutional layers.
- 1 hidden fully connected layer with the corresponding activation function.
- 1 layer of dropout between the fully connected layers.
- 1 output fully connected layer with SoftMax on its output for classification.

The architecture changes based on the model variant (the reference network, the network with AHAF, the network with the F-neuron activation). Still, the overall list of layers and the list of non-activation components remains the same across all experiments. We run the experiments on a laptop with NVIDIA GeForce GTX 1650 Max-Q and PyTorch [14] version 1.13.1. The implementation is available on GitHub: https://github.com/s-kostyuk/af_replacement.

4.1. Reference Model Training from Scratch

We train four variants of the reference KerasNet model:

1. With the ReLU activation in all layers.
2. With SiLU in all layers.
3. With Tanh in all layers.
4. With Sigmoid in all layers.

We train each variant from scratch for one hundred epochs on the CIFAR-10 dataset. We record the test set accuracy and the training set loss during to visualize and analyze the training process. We save the resulting weights for further reference and activation function replacement. This experiment provides the reference data for comparison with derived models.

4.2. Replacing Activation Functions in Pre-Trained Reference Model

We create the derivative models from the reference models by loading the pre-trained weights and replacing the non-adaptive activation functions with the adaptive ones. This step generates four different model variants:

1. With the AHAF activation in all layers, AHAF initialized as ReLU.
2. With the AHAF activation in all layers, AHAF initialized as SiLU.
3. With Tanh activation in the convolutional layers and the F-neuron activation in the second-to-last fully connected layer, the F-neuron activation approximates Tanh with 18 membership functions.
4. With Sigmoid activation in the convolutional layers and the F-neuron activation in the second-to-last fully connected layer, the F-neuron approximates Sigmoid with 18 membership functions.

We evaluate the performance of derived networks directly after the replacement. This experiment allows validating the replacement approach and studies the effect of imperfect replacements when used with piece-wise linear adaptive activation functions.

4.3. Activation Function Fine-Tuning

We fine-tune the activation function parameters in the derived models to compensate for approximation errors and improve the classification performance. We freeze all trainable parameters, except the activation function parameters, during fine-tuning. We fine-tune for 50 additional epochs on the same CIFAR-10 dataset, so there is no need to adapt the architecture for a new dataset.

We record the test set accuracy and the training set loss during the fine-tuning process. We save the activation function parameters for subsequent visualization. This experiment allows to evaluate the impact of activation function fine-tuning on model's performance and the form of activation functions.

4.4. Training Models with Adaptive Activation Functions from Scratch

We train four additional variants of the KerasNet model from scratch:

1. With the AHAF activation in all layers, AHAF initialized as ReLU.
2. With the AHAF activation in all layers, AHAF initialized as SiLU.
3. With Tanh activation in the convolutional layers and the F-neuron activation in the second-to-last fully connected layer, the F-neuron activation approximates Tanh with 18 membership functions.
4. With Sigmoid activation in the convolutional layers and the F-neuron activation in the second-to-last fully connected layer, the F-neuron approximates Sigmoid with 18 membership functions.

We record the training time, the test set accuracy, and the training set loss. This experiment allows to compare the training time and the resulting effectiveness between models with replaced activation functions and the models originally trained with adaptive activations.

5. Results and Discussions

We recorded and evaluated the results of the experiments. We structure this section as the following:

1. Validation of the parameter initialization and the form for adaptive activation functions.
2. Evaluation of the activation function form after pre-training with all other parameters frozen.
3. Comparison of the performance recordings for different variants of the KerasNet model.

5.1. Validation of Activation Function Replacement

We visualize the activation function form for adaptive activation functions directly after the activation function replacement. As the visualization shows, the form of AHAF activations (grey; AHAF-as-RELU and AHAF-as-SiLU) directly follows the form of the corresponding original functions (black; ReLU and SiLU correspondingly). Hence, we confirm that the AHAF activation can be a direct replacement of ReLU and SiLU activations (figures 5,6).

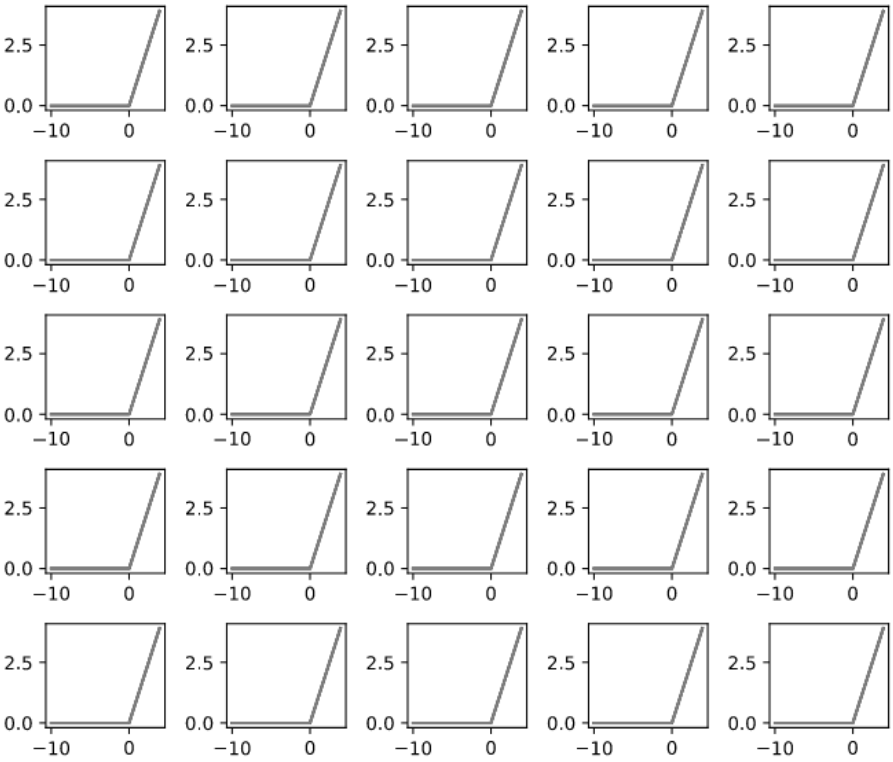

Figure 5: The form of ReLU-like AHAF after replacement.

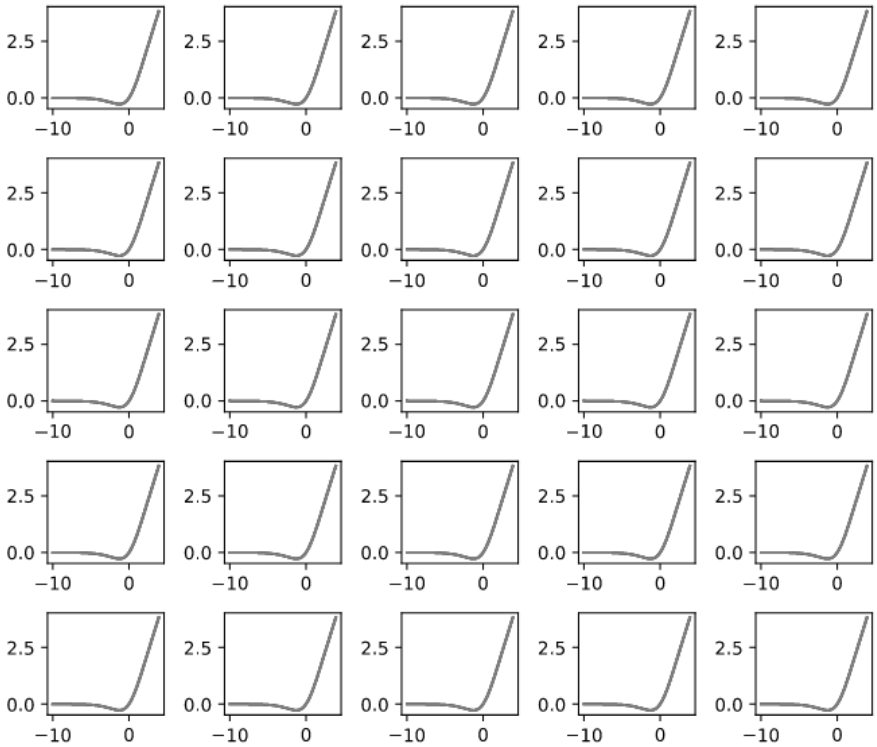

Figure 6: The form of SiLU-like AHAF after replacement.

The visualizations for F-neuron activations show that while the replacement is not perfect, the differences between the original functions (black) and the replacement function (grey) are hard to spot. Hence, the parameter initialization is correct for the F-neuron activations (figures 7,8).

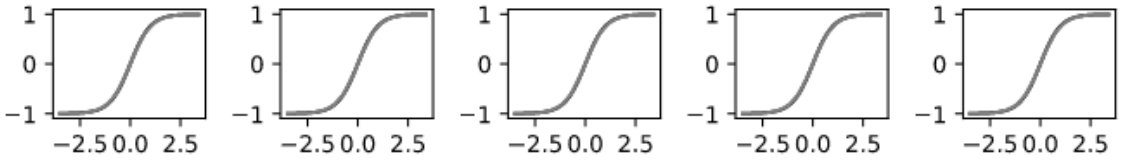

Figure 7: Activation form of the F-neuron activation as Tanh after replacement

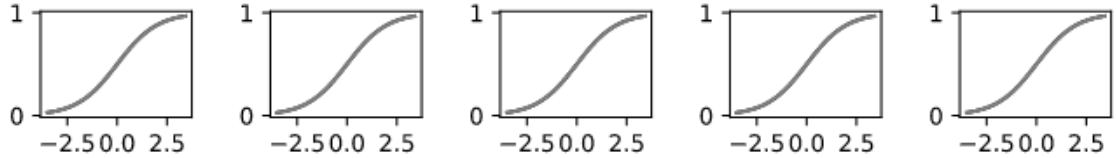

Figure 8: Activation form of the F-neuron activation as Sigmoid after replacement

5.2. Adaptive Activations' Form After Fine-Tuning

We visualize the activation function form for the AHAF and F-neuron activation variants after fine-tuning. The visualization shows that the form of fine-tuned activation functions (grey) noticeably diverges from the original form (black). We note serious deformation of the sigmoid-like F-neuron activations which indicates poor applicability of Sigmoid for convolutional networks. Hence, we confirm that the activation function fine-tuning works and the activation function form changes depending on the requirements of the model (figures 9-12).

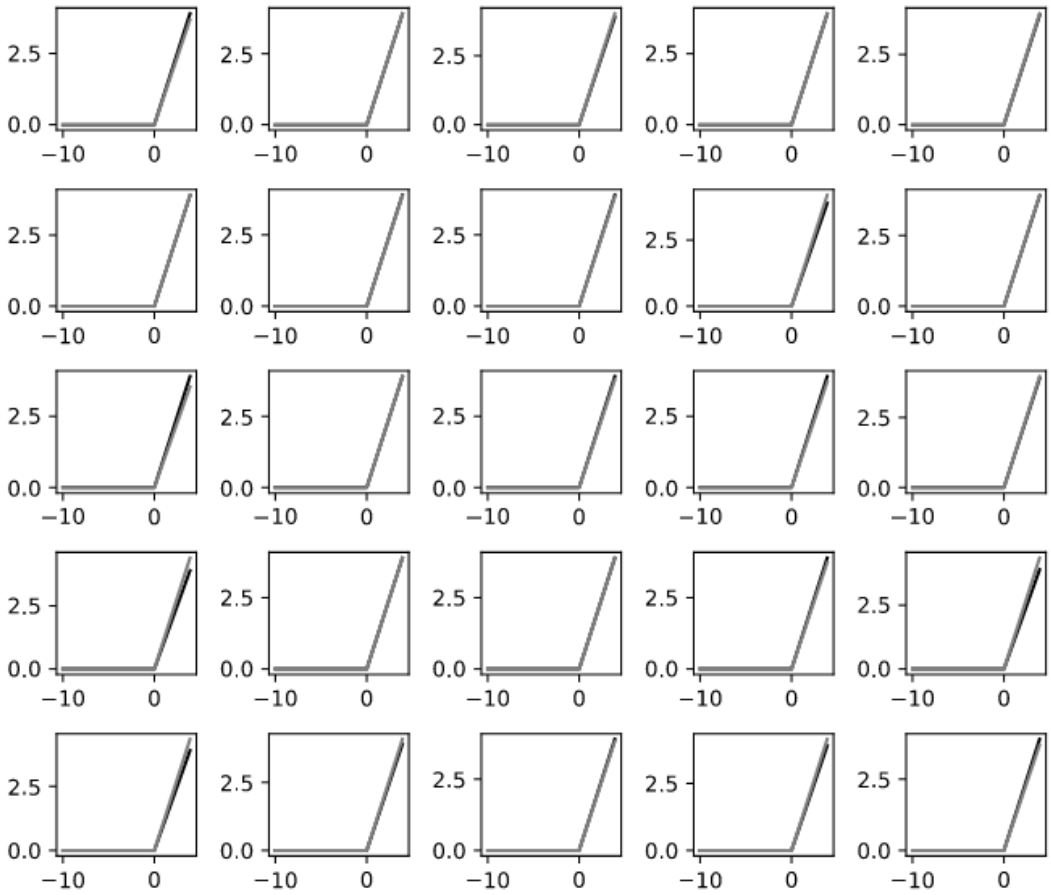

Figure 9: Activation form of AHAF as ReLU after fine-tuning

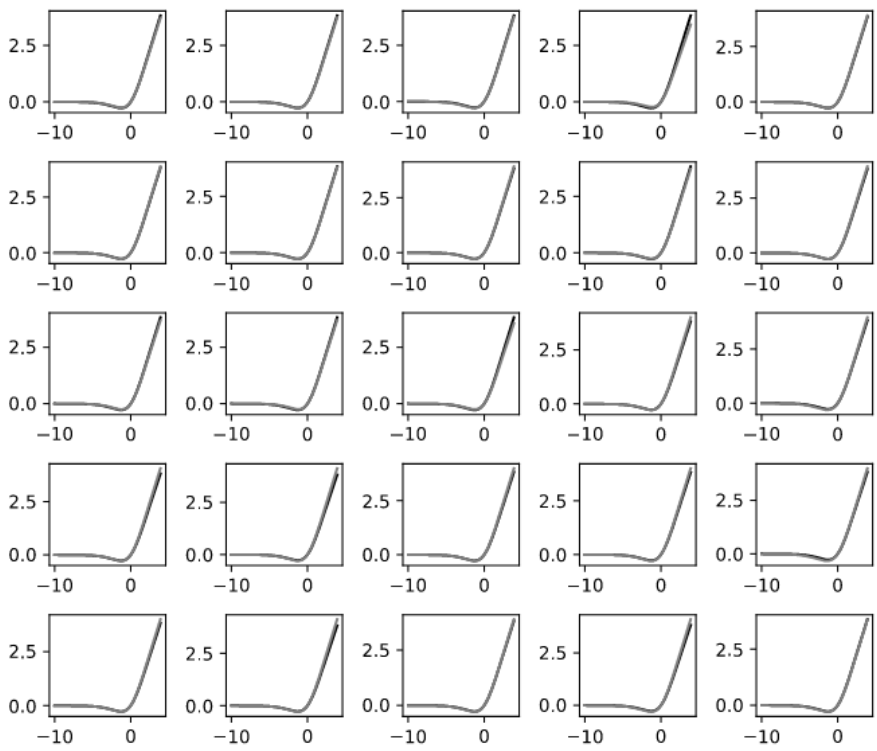

Figure 10: Activation form of AHAF as SiLU after fine-tuning

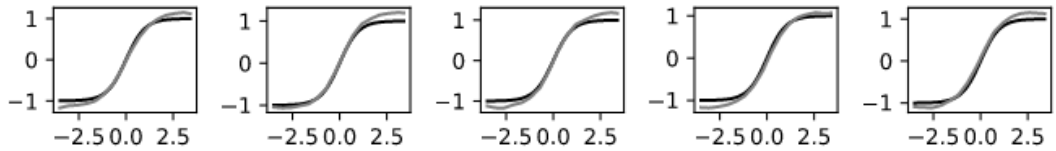

Figure 11: Activation form of the F-neuron activation as Tanh after fine-tuning

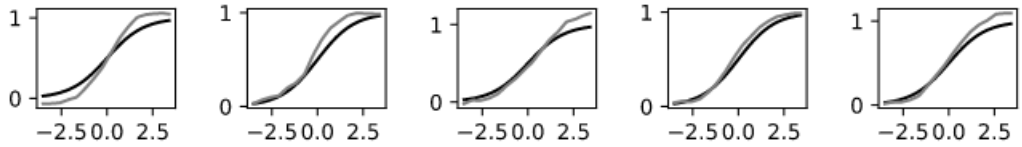

Figure 12: Activation form of the F-neuron activation as Sigmoid after fine-tuning

5.3. Performance Of the Model Variants

We compare the test set accuracy between all variants of the KerasNet model. We group by the results by the reference activation functions used in such networks:

1. Linear units – SiLU-like and ReLU-like.
2. Bounded functions – Tanh-like and Sigmoid-like.

The fine-tuned model with the ReLU-like AHAF activations shows the best test set accuracy on the CIFAR-10 dataset. AHAF successfully replaces the original ReLU activation functions and adapts its form during fine-tuning but tends to overfit on the final epochs. While the overall training time is longer, activation function replacement together with activation function fine-tuning shows itself as a viable solution when the pre-trained model with non-adaptive functions is already available. As expected, the test set accuracy after the AHAF-based activation function replacement is the same as the accuracy of the reference models after one hundred epochs. Table 1 demonstrates the training results for models with linear unit activations.

Table 1
Performance of model variants with SiLU-like and ReLU-like activations.

Variant	CNN AF	FFN AF	Accuracy, %	Training Epoch	Training Time, minutes
Reference	ReLU	ReLU	82.41	96	17.41
AHAF, from scratch	ReLU-like	ReLU-like	82.46	96	24.78
AHAF, AF replaced	ReLU-like	ReLU-like	82.17	100	-
AHAF, fine-tuned AF	ReLU-like	ReLU-like	**82.64**	148	9.13
Reference	SiLU	SiLU	81.57	98	18.00
AHAF, from scratch	SiLU-like	SiLU-like	81.84	98	25.42
AHAF, AF replaced	SiLU-like	SiLU-like	81.54	100	-
AHAF, fine-tuned AF	SiLU-like	SiLU-like	81.95	143	8.64

The model with the Tanh-like F-neuron activation trained from scratch shows the best test set accuracy on the CIFAR-10 dataset. One potential explanation to such results is poor applicability of Tanh and Sigmoid activations for convolutional neural networks. Training the model from scratch allows modification of the activation function form and hence better propagation of the signal between the layers. As expected, the F-neuron-based activation function replacement gives different results to the reference model, but the actual difference is negligible (about 0.01%). Even for networks with Tanh-like and Sigmoid-like activations, activation function replacement together with activation function fine-tuning is a viable option when the pre-trained model is already available. Table 2 demonstrates the training results for models with bounded activations.

Table 2
Performance of model variants with Tanh-like and Sigmoid-like activations.

Variant	CNN AF	FFN AF	Accuracy, %	Training Epoch	Training Time, minutes
Reference	Tanh	Tanh	79.97	100	18.35
Fuzzy, from scratch	Tanh	Tanh-like	**81.04**	100	21.22
Fuzzy, AF replaced	Tanh	Tanh-like	79.97	100	-
Fuzzy, fine-tuned AF	Tanh	Tanh-like	80.34	150	8.11
Reference	Sigmoid	Sigmoid	60.17	100	18.63
Fuzzy, from scratch	Sigmoid	Sigmoid-like	60.55	100	21.36
Fuzzy, AF replaced	Sigmoid	Sigmoid-like	60.43	100	-
Fuzzy, fine-tuned AF	Sigmoid	Sigmoid-like	61.16	147	9.10

Figures 13 and 14 demonstrate the training process for models with linear units and bounded functions correspondingly.

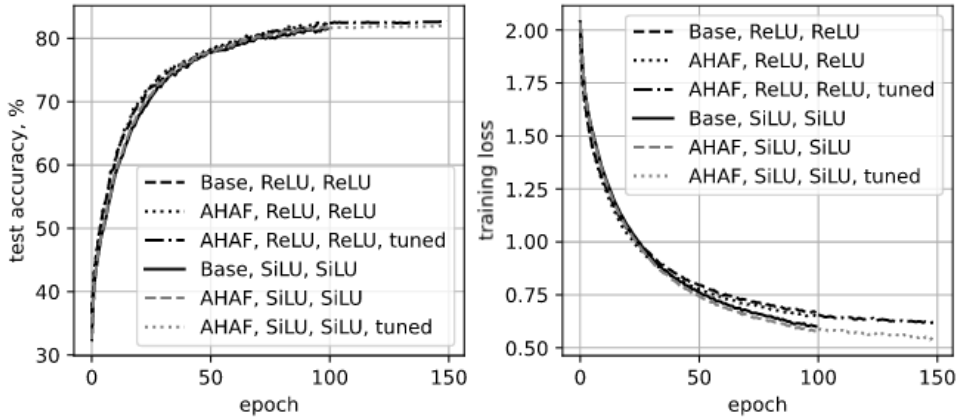

Figure 13: The training process for network variants with SiLU and ReLU activations

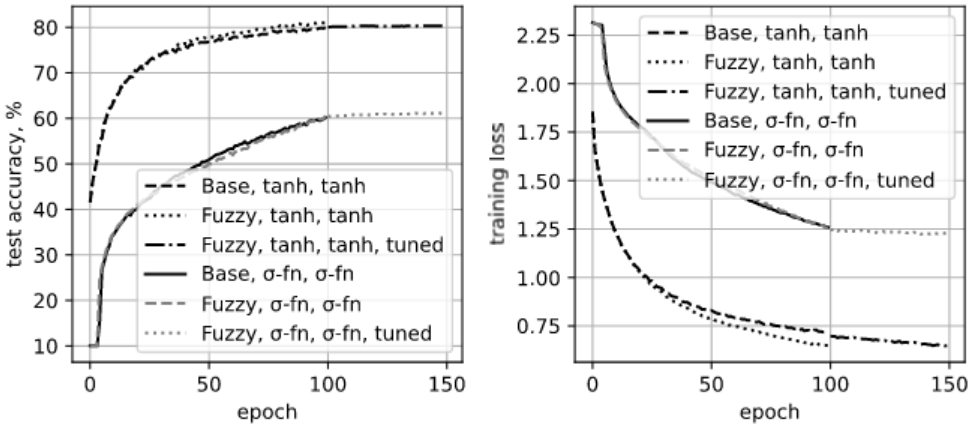

Figure 14: The training process for network variants with Tanh and Sigmoid activations

6. Conclusions

This paper introduces the method of activation function replacement in pre-trained artificial neural network models. The method allows experimenting with various adaptive activation functions without the need in model re-training. The reference implementation of the activation function replacement method in PyTorch is provided in the paper. We evaluate several variants of the KerasNet model on the CIFAR-10 dataset: the reference pre-trained model with non-adaptive activation function, the derived model with replaced activation functions and the derived model trained from scratch. The empirical results confirm the possibility of replacing the Sigmoid and Tanh activations with the corresponding piece-wise approximations without significant changes in the test set accuracy. The instances of AHAF and the F-neuron activation are used as the example adaptive activation functions. The experiment shows the effectiveness of activation function replacement in combination with activation function fine-tuning for improving the test set accuracy of a pre-trained model.

7. References

[1] A. Paleyes, R.-G. Urma, N. D. Lawrence, Challenges in deploying machine learning: A survey of case studies, ACM Computing Surveys 55 (2022) 1–29. doi:10.1145/3533378.

[2] E. Yurtsever, J. Lambert, A. Carballo, K. Takeda, A survey of autonomous driving: Common practices and emerging technologies, IEEE Access 8 (2020) 58443–58469. doi:10.1109/access.2020.2983149.

[3] S. S. A. Zaidi, M. S. Ansari, A. Aslam, N. Kanwal, M. Asghar, B. Lee, A survey of modern deep learning based object detection models, Digital Signal Processing 126 (2022) 103514. doi:10.1016/j.dsp.2022.103514.

[4] S. Grigorescu, B. Trasnea, T. Cocias, G. Macesanu, A survey of deep learning techniques for autonomous driving, Journal of Field Robotics 37 (2020) 362–386. doi:10.1002/rob.21918.

[5] Y. Wang, M. Perry, D. Whitlock, J. W. Sutherland, Detecting anomalies in time series data from a manufacturing system using recurrent neural networks, Journal of Manufacturing Systems 62 (2022) 823–834. doi:10.1016/j.jmsy.2020.12.007.

[6] S. Ranathunga, E.-S. A. Lee, M. P. Skenduli, R. Shekhar, M. Alam, R. Kaur, Neural machine translation for low-resource languages: A survey, ACM Computing Surveys 55 (2023) 1–37. doi:10.1145/3567592.

[7] F. F. Xu, U. Alon, G. Neubig, V. J. Hellendoorn, A systematic evaluation of large language models of code, in: Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming, ACM, 2022. doi:10.1145/3520312.3534862.

[8] GPT-4 Technical Report, Technical Report, OpenAI, 2023. doi:10.48550/ARXIV.2303.08774.

[9] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, A. Rush, Transformers: State-of-the-art natural language

processing, in: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, Association for Computational Linguistics, 2020. doi:10.18653/v1/2020.emnlp-demos.6.
[10] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, in: Proceedings of the 2019 Conference of the North, Association for Computational Linguistics, 2019. doi:10.18653/v1/n19-1423.
[11] S. R. Dubey, S. K. Singh, B. B. Chaudhuri, Activation functions in deep learning: A comprehensive survey and benchmark, Neurocomputing 503 (2022) 92–108. doi:10.1016/j.neucom.2022.06.111.
[12] A. D. Jagtap, K. Kawaguchi, G. E. Karniadakis, Adaptive activation functions accelerate convergence in deep and physics-informed neural networks, Journal of Computational Physics 404 (2020) 109136. doi:10.1016/j.jcp.2019.109136.
[13] X. Han, Z. Zhang, N. Ding, Y. Gu, X. Liu, Y. Huo, J. Qiu, Y. Yao, A. Zhang, L. Zhang, W. Han, M. Huang, Q. Jin, Y. Lan, Y. Liu, Z. Liu, Z. Lu, X. Qiu, R. Song, J. Tang, J.-R. Wen, J. Yuan, W. X. Zhao, J. Zhu, Pre-trained models: Past, present and future, AI Open 2 (2021) 225–250. doi: 10.1016/j.aiopen.2021.08.002.
[14] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, Pytorch: An imperative style, high-performance deep learning library, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems, volume 32, Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper/9015.pdf.
[15] A. Krizhevsky, G. Hinton, Learning multiple layers of features from tiny images, Technical Report, University of Toronto, Toronto, Ontario, 2009. URL: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.
[16] A. Apicella, F. Donnarumma, F. Isgrò, R. Prevete, A survey on modern trainable activation functions, Neural Networks 138 (2021) 14–32. doi:10.1016/j.neunet.2021.01.026.
[17] D. Chen, K. Xu, Arelu: Attention-based rectified linear unit, CoRR abs/2006.13858 (2020). URL: https://arxiv.org/abs/2006.13858. arXiv:2006.13858, pre-print.
[18] H. Fang, J.-U. Lee, N. S. Moosavi, I. Gurevych, Transformers with learnable activation functions, 2022. doi:10.48550/ARXIV.2208.14111. arXiv:2208.14111, pre-print.
[19] Y. Bodyanskiy, S. Kostiuk, Adaptive hybrid activation function for deep neural networks, System research and information technologies (2022) 87–96. doi:10.20535/srit.2308-8893.2022.1.07.
[20] Y. Bodyanskiy, S. Kostiuk, Deep neural network based on f-neurons and its learning (2022). doi:10.21203/rs.3.rs-2032768/v1, pre-print.
[21] B. Yuen, M. T. Hoang, X. Dong, T. Lu, Universal activation function for machine learning, Scientific Reports 11 (2021). doi:10.1038/s41598-021-96723-8.
[22] P. Ramachandran, B. Zoph, Q. V. Le, Searching for activation functions, in: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings, OpenReview.net, 2018. URL: https://openreview. net/forum?id=Hkuq2EkPf.
[23] A. Severyn, A. Moschitti, Twitter sentiment analysis with deep convolutional neural networks, in: Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, 2015. doi:10.1145/2766462.2767830.
[24] B. Felbo, A. Mislove, A. Søgaard, I. Rahwan, S. Lehmann, Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm, in: Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, 2017. doi:10.18653/v1/d17-1169.
[25] B. L. Deng, G. Li, S. Han, L. Shi, Y. Xie, Model compression and hardware acceleration for neural networks: A comprehensive survey, Proceedings of the IEEE 108 (2020) 485–532. doi:10.1109/jproc.2020.2976475.
[26] J. Gou, B. Yu, S. J. Maybank, D. Tao, Knowledge distillation: A survey, International Journal of Computer Vision 129 (2021) 1789–1819. doi:10.1007/s11263-021-01453-z.
[27] N. Aghli, E. Ribeiro, Combining weight pruning and knowledge distillation for CNN compression, in: 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), IEEE, 2021. doi:10.1109/cvprw53098.2021.00356.

[28] D. C. Mocanu, E. Mocanu, P. Stone, P. H. Nguyen, M. Gibescu, A. Liotta, Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science, Nature Communications 9 (2018). doi:10. 1038/s41467-018-04316-3.
[29] Z. Lu, G. Sreekumar, E. Goodman, W. Banzhaf, K. Deb, V. N. Boddeti, Neural architecture transfer, IEEE Transactions on Pattern Analysis and Machine Intelligence 43 (2021) 2971–2989. doi:10.1109/tpami.2021.3052758.
[30] B. Thompson, H. Khayrallah, A. Anastasopoulos, A. D. McCarthy, K. Duh, R. Marvin, P. McNamee, J. Gwinnup, T. Anderson, P. Koehn, Freezing subnetworks to analyze domain adaptation in neural machine translation, in: Proceedings of the Third Conference on Machine Translation: Research Papers, Association for Computational Linguistics, 2018. doi:10.18653/v1/w18-6313.
[31] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, Q. He, A comprehensive survey on transfer learning, Proceedings of the IEEE 109 (2021) 43–76. doi:10.1109/jproc.2020.3004555.
[32] F. Chollet, Train a simple deep cnn on the cifar10 small images dataset — keras 1.2.2., 2017. URL: https://github.com/keras-team/keras/blob/1.2.2/examples/cifar10_cnn.py, online.