UDC 004.852

DOI: https://doi.org/10.30837/bi.2026.1(104).05

V. Bukhanovskyi[1], N. Ryabova[2]

[1]Kharkiv National University of Radioelectronics, Kharkiv, Ukraine,
volodymyr.bukhanovskyi@nure.ua, ORCID iD: 0009-0004-9647-4194

[2]Kharkiv National University of Radioelectronics, Kharkiv, Ukraine,
nataliya.ryabova@nure.ua, ORCID iD: 0000-0002-3608-6163

FUZZY-ADAPTERS: INTEGRATION OF FUZZY MEMBERSHIP FUNCTIONS FOR VISION TRANSFORMERS, EFFECTIVE TRANSFER LEARNING

This paper introduces Fuzzy-Adapter, a PEFT method for efficiently adapting Vision Transformers (ViT) under resource constraints. The approach replaces standard ReLU activations with trainable Gaussian membership functions to improve uncertainty modelling. Experimental results on the CIFAR-10 dataset using deit_tiny demonstrate that Fuzzy-Adapter achieves 92.6% accuracy, significantly outperforming the ReLU-based adapter (90.2%) and the baseline (85.7%). The findings suggest that integrating neuro-fuzzy logic into bottleneck architecture enhances model adaptability and performance in data-limited environments.

VISION TRANSFORMER; PARAMETER-EFFICIENT FINE-TUNING; FUZZY LOGIC; ADAPTER; IMAGE CLASSIFICATION

В.О. Бухановський, Н.В. Рябова. Fuzzy-adapters: Інтеграція нечітких функцій належності для ефективного трансферного навчання Vision Transformers. У цій роботі представлено Fuzzy-Adapter – це метод PEFT, розроблений для ефективної адаптації Vision Transformers (ViT) в умовах обмежених ресурсів. Підхід передбачає заміну стандартних активацій ReLU на навчальні Гауссові функції належності для покращення моделювання невизначеності. Експериментальні результати на наборі даних CIFAR-10 з використанням моделі deit_tiny демонструють, що Fuzzy-Adapter досягає точності 92,6%, що значно перевищує показники адаптера на основі ReLU (90,2%) та базового методу (85,7%). Отримані висновки свідчать про те, що інтеграція нейро-нечіткої логіки в архітектуру bottleneck підвищує адаптивність та продуктивність моделі в умовах дефіциту даних.

VISION TRANSFORMER; PARAMETER-EFFICIENT FINE-TUNING; НЕЧІТКА ЛОГІКА; АДАПТЕР; КЛАСИФІКАЦІЯ ЗОБРАЖЕНЬ

Introduction

Vision Transformer (ViT) architectures have become the standard in many computer vision tasks. It demonstrated exceptional performance by modelling global data dependencies. However, this success comes at a cost: modern ViT models are massive and training them from scratch requires large-scale datasets and significant computational resources [1].

The field of Parameter-Efficient Fine-Tuning (PEFT) emerged as a branch of transfer learning. One of the most successful approaches in this domain is the use of adapters. Adapters are small modules inserted into the layers of a pre-trained model. During adaptation to a new task, the weights of the backbone model remain "frozen", while only the parameters of these compact modules are trained. This approach reduces the number of trainable parameters and computational costs significantly [2].

A key element of popular Bottleneck adapters is the non-linear activation function applied between the down-projection and up-projection layers. Traditionally, standard functions such as ReLU are chosen [3]. However, these functions operate with crisp, binary, or semi-linear transformations. We hypothesise that such logic limits the adapter's ability to model the uncertainty and imprecision inherent in new datasets.

In this work, we explore a hybrid neuro-fuzzy approach by replacing standard activations with fuzzy membership functions. We introduce Fuzzy-Adapters, where the activation is a trainable Gaussian membership function [4]. This allows the adapter not merely to apply fixed non-linearity, but to learn a flexible, adaptive transformation that better suits the specifics of the target task.

Our contribution lies in the practical implementation and validation of this idea. We integrated Fuzzy-Adapters into the "deit_tiny" ViT model and conducted experiments on a reduced CIFAR-10 dataset to simulate a resource-constrained scenario. We demonstrate that our approach provides a statistically significant improvement in accuracy over standard ReLU adapters and the baseline of training only the classifier.

1. Analysis of last achievements and publications

Classification is a fundamental task in computer vision, assigning an image to a specific category from a predefined set. However, modern state-of-the-art models, such as Vision Transformers, require massive datasets to achieve high accuracy from scratch. To mitigate this, transfer learning techniques have emerged. It allows the transfer of knowledge obtained from one set of tasks or domains to another.

To understand the proposed approach, it is necessary to consider three key pillars underpinning our research: the Vision Transformers architecture, the adapter-based training methodology, and the fundamental principles of neuro-fuzzy systems.

Vision Transformer architecture is a successor to the successful Transformer architecture (initially developed for natural language processing) in the field of computer

vision. Unlike Convolutional Neural Networks (CNNs), which process information hierarchically through local receptive fields [5], ViT models an image as a sequence of tokens. The operating principle involves dividing the input image into fixed, non-overlapping patches. Each patch is linearly projected into a vector space (embedding), forming a sequence of "tokens". Positional information is added to these tokens via positional encoding. In classification tasks, a special [CLS] token is typically added to the sequence, and its final representation is used to predict the image class. The core of architecture is the self-attention mechanism [6]. This mechanism enables the model to assess the importance of each token relative to all others in the sequence and effectively capture global relationships between different (even spatially distant) regions of the image. The capability for global contextual modelling ensures high ViT performance, especially when training on large datasets.

Transfer learning is a technique that enables the adaptation of knowledge acquired from one set of tasks or domains to another [7]. This implies that a model trained using vast resources, including data, computational power, time, and cost, which have been made open-source, can be fine-tuned and reused in new settings by the broader engineering community at a significantly lower price than the initial resource requirements. This represents a significant step forward toward the democratisation of large models and, more broadly, artificial intelligence.

Traditionally, three primary approaches define the transfer learning landscape: pre-trained models as classifiers, fine-tuning, and feature extraction [8, p. 254].

The fine-tuning approach involves taking a pre-trained model (e.g., one trained on ImageNet), replacing the final classification layers, and retraining the entire network (or a portion of it) on the new dataset.

The Feature Extraction approach also utilises a pre-trained network but keeps the backbone weights "frozen". Only the new classification layers are trained on the target dataset.

While full fine-tuning often yields better accuracy, the immense size of modern models makes it computationally expensive and data-intensive. This challenge has driven the development of Parameter-Efficient Fine-Tuning (PEFT) methods that adapt pre-trained models with a minimal number of updated parameters. Among these methods, adapters have emerged as a fundamental and efficient solution.

Adapters are among the most fundamental and efficient PEFT methods. These are small, supplementary modules integrated into the layers of a frozen backbone model. During training, only the weights of these adapters and, usually, the final classification layer are updated, while millions (or billions) of parameters of the main model remain unchanged.

In this work, we focus on the typical Bottleneck Adapter architecture (Figure 1). Such a module is embedded inside a Transformer block (after the feed-forward or attention layer). It includes a fully connected layer that compresses the input representation into a lower-dimensional space, a non-linear function application layer, and a fully connected layer that projects the representation back to its initial dimension.

The entire module is connected via a residual connection. The key advantage is that the number of trainable parameters is significantly smaller than the total number of parameters in a layer. Although there are other popular PEFT methods, such as Low-Rank Adaptation (LoRA) [9], IA3 [10], Prefix Tuning [11], or Prompt Tuning [12], Bottleneck adapters provide a reliable and well-researched basis for integrating new mechanisms.

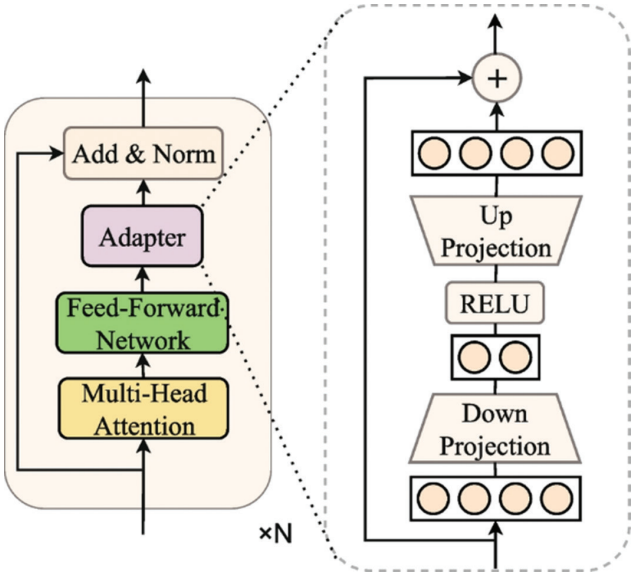

Fig. 1. Bottleneck Adapter architecture

The research gap we address lies in non-linear activation within the adapter. Standard approaches use "crisp" functions that apply complex, semi-linear transformations. We propose turning to neuro-fuzzy systems.

Neuro-fuzzy logic represents a hybrid intelligent system architecture that synergises the learning capabilities of artificial neural networks with the reasoning power of fuzzy logic. While artificial neural networks excel at pattern recognition, data adaptation, and learning from vast datasets, they often function as "black boxes" with crisp internal representations. Conversely, fuzzy logic provides a framework for modelling uncertainty, imprecision, and ambiguity, allowing for reasoning that approximates human cognition. The integration of the two paradigms results in systems capable of learning from data while operating with interpretable, rule-based knowledge structures

The central concept of fuzzy logic is the membership function. Unlike classical Boolean logic, where an element either belongs to a set (value 1) or not (value 0), fuzzy logic allows an element to belong to a set with a certain "degree of membership" – a real number in the range [0, 1].

In the context of hybrid neuro-fuzzy systems, these functions play a role analogous to activation functions in deep learning (such as ReLU or Sigmoid) but offer the additional benefit of interpretability. They transform crisp input values (such as pixel intensities or feature vectors) into fuzzy degrees of membership, enabling the system to process inherently incomplete or noisy information.

There are many forms of such functions: triangular, trapezoidal, Gaussian, etc (Figure 2). Among the various available membership functions, we specifically select the Gaussian Membership Function (GMF) for integration into the adapter architecture. This choice is driven by several factors that make GMF particularly suitable for gradient-based optimisation in neural networks.

First and foremost is the property of smoothness and differentiability. Unlike triangular or trapezoidal functions, which have sharp corners and non-differentiable points, the Gaussian function is smooth and continuously differentiable across its entire domain. This is a mandatory requirement for modern deep learning, as it ensures gradients can flow uninterrupted during backpropagation, enabling stable weight updates.

Secondly, the Gaussian function offers specific trainable parameters: centre and width. They directly correspond to interpretable properties of the data distribution. By treating these as learnable parameters rather than fixed hyperparameters, the neural network gains additional degrees of freedom. This allows the adapter to independently tune the activation shape to match the specific uncertainty profile of the new dataset.

Finally, the Gaussian function naturally models uncertainty. While a standard ReLU activation applies a hard threshold (zeroing out all negative values), a Gaussian activation provides a soft window of attention around a specific centre. This enables the model to assign high activation only to features that fall within a learned range of relevance, effectively filtering out noise and irrelevant data.

While integrating fuzzy membership functions as activation mechanisms offers significant advantages in terms of flexibility and uncertainty modelling, it introduces specific challenges compared to standard activations.

The first thing to mention is computational Complexity. Standard activation functions are computationally inexpensive and require only simple thresholding. In contrast, fuzzy functions, particularly Gaussian MFs, involve exponential calculations. This increases the floating-point operations (FLOPs) needed for both the forward pass and gradient calculation during backpropagation.

The second trade-off is optimisation difficulty. In a standard adapter, the activation function is usually fixed. In the proposed neuro-fuzzy approach, the parameters of the membership function (e.g., centre and width) are trainable. This increases the complexity of the optimisation landscape. If not carefully initialised, parameters may vanish or explode, leading to numerical instability or gradients that hinder convergence. By making the activation function parametric, the model's total number of trainable parameters increases slightly. But in the context of Bottleneck adapters, this minimal increase requires the optimiser to find a delicate balance between the weights of the linear projections and the shape of the activation function.

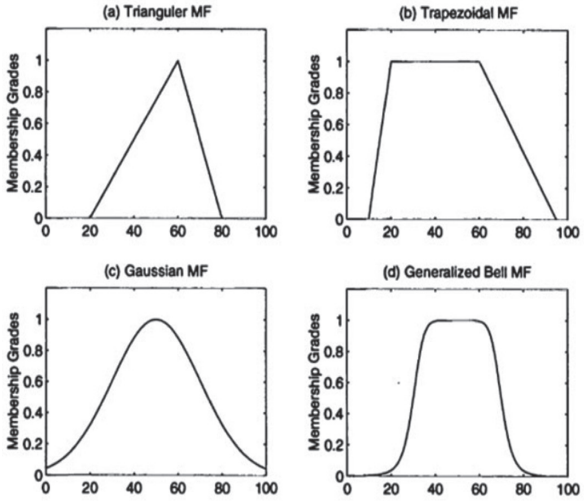

Fig. 2. Membership function types: a) triangular, b) trapezoidal, c) gaussian, d) generalised bell [4]

Despite these limitations, we hypothesise that replacing the standard ReLU activation in the Bottleneck adapter with a trainable Gaussian membership function will enable the adapter module to learn a more flexible, smoother transformation. Instead of a fixed non-linearity, the parameters of the activation function itself (its centre and width) become part of the optimisation process. This provides the model with additional degrees of freedom for a more refined adaptation of internal representations to the specifics of the new dataset.

Our work builds on the standard Bottleneck adapter architecture. The standard module consists of a dimensionality-reducing layer, a non-linear activation function, and a dimensionality-restoring layer with a residual connection.

Our key contribution is to replace the standard “crisp” activation with a trainable fuzzy membership function. In this work, we focused on the Gaussian Membership Function (GMF). We implemented this function as a separate module. Mathematically, it is described by the formula (1).

$$f(x,m,\sigma) = e^{-\frac{1}{2}\left(\frac{x-m}{\sigma+\varepsilon}\right)^2}, \quad (1)$$

where ε – a small constant for numerical stability; m – centre; σ – width.

m and σ are not fixed hyperparameters, but trainable parameters. This enables the neural network to independently optimise the shape of the activation function for each adapter during training.

In practice, in addition to the main Bottleneck layers described above, the input tensor undergoes normalisation

before the down-projection layer, followed by GMF, dropout, the up-projection layer, and finally the residual connection [13].

2. Materials and methods

Selecting an appropriate backbone architecture involves a trade-off between representational power and computational efficiency. Standard Vision Transformers (ViT-Base), comprising approximately 86 million parameters, use global self-attention with quadratic complexity and typically require massive datasets (e.g., JFT-300M) to surpass Convolutional Neural Networks (CNNs) due to their lack of inherent inductive biases.

An alternative modern architecture is the Swin Transformer [14], which introduces hierarchical feature representations and shifted windowing mechanisms to achieve linear complexity with respect to image size. While Swin Transformers excel in dense prediction tasks, even their lightest variant, Swin-T (Tiny), contains approximately 29 million parameters.

In contrast, the Data-efficient Image Transformer (DeiT) utilises a specialised distillation token to learn effectively from teacher ConvNets on smaller datasets like ImageNet-1k [15]. The "deit_tiny" variant selected for our experiments contains only about 5 million parameters – roughly 17% of the size of Swin-T and 6% of ViT-Base. This extreme compactness makes it uniquely suited to the resource-constrained environment of our research, enabling rapid iteration on the reduced CIFAR-10 subset without the significant computational overhead of Swin or standard ViT architectures.

For the base architecture, we selected the "deit_tiny_patch16_224" model. This characteristic makes it particularly suitable for scenarios where access to large-scale data for fine-tuning is limited, aligning with our experimental constraint of using a reduced CIFAR-10 [16] subset (10% of the original data). Additionally, the "tiny" variant is highly compact, reducing computational resource requirements and accelerating the experimentation process while maintaining competitive performance.

The integration process was architected through a custom wrapper class "ViTWithAdapter", designed to encapsulate the pre-trained "deit_tiny" backbone. First, to preserve the learned feature representations, all parameters of the base ViT model were "frozen" by disabling gradient computation for all parameters. Second, a single instance of the Fuzzy-Adapter was sequentially injected into the architecture, positioned to process the output of each Transformer block. Next, the original classification head was discarded and replaced with a new linear layer mapping the model's embedding dimension of 192 to an output dimension of 10, corresponding to the CIFAR-10 classes. Consequently, the optimisation process was strictly isolated to the adapter modules and the new head. This configuration resulted in a highly efficient setup with 156 718 trainable parameters against a total of 5 681 134, representing merely 2.76% of the model"s capacity.

Data preprocessing and augmentation pipelines were implemented utilising the "torchvision. transforms" module from the PyTorch ecosystem – a critical preprocessing step involved addressing the resolution disparity between the input data and the model architecture. Since the CIFAR-10 dataset consists of 32 32-pixel images, and the pre-trained "deit_tiny_patch16_224" model was designed for 224 224 inputs, the pipeline included upsampling transformations to match the required dimensions.

To mitigate the risk of overfitting, we integrated standard data augmentation techniques alongside statistical normalisation. Furthermore, to strictly enforce the experimental constraint of limited data availability, the training process did not utilise the full dataset. Instead, we selected indices at random to create a reduced training subset containing only 10% of the original CIFAR-10 dataset (5,000 images). The test dataset was used in its entirety.

To evaluate the effectiveness of our approach, we compared three model configurations:

1. ViT Head-Only – the baseline model, where only the new classification head is trained;
2. ViT with ReLU Adapter – the control model using an identical Bottleneck Adapter architecture but with standard ReLU activation;
3. ViT with Fuzzy Adapter (Gaussian) – our proposed model with activation based on the Gaussian membership function.

The implementation utilised the PyTorch framework as the primary engine for tensor computations, model construction (torch.nn), and optimisation (torch.optim). The torchvision library was employed to manage the CIFAR-10 dataset and to execute image processing pipelines, specifically utilising torchvision. Transforms for data augmentation and normalisation. To access the pre-trained "deit_tiny_patch16_224" model, we integrated the timm (PyTorch Image Models) library. NumPy was used for high-performance numerical operations and array handling, particularly for managing data indices when creating the reduced dataset subsets. Matplotlib was used to visualise the training dynamics, including loss and accuracy curves, and to display prediction samples. Finally, to ensure the reliability and reproducibility of our results, the random module was used to set a fixed random seed across the environment, and the os module managed file system interactions for saving model checkpoints. There are snippets of GMF implementation in Figure 3 and the Fuzzy Adapter implementation in Figure 4.

```
class GaussianMembershipFunction(nn.Module):
    def __init__(self, mu=0.0, sigma=1.0):
        super(GaussianMembershipFunction, self).__init__()
        self.mu = nn.Parameter(torch.tensor(mu, dtype=torch.float))
        self.sigma = nn.Parameter(torch.tensor(sigma, dtype=torch.float))

    def forward(self, x):
        eps = 1e-6
        return torch.exp(-((x - self.mu) ** 2) / (2 * self.sigma ** 2 + eps))
```

Fig. 3. Snippet of GMF implementation

```python
class BottleneckAdapter(nn.Module):
    def __init__(self, input_dim, bottleneck_dim, dropout=0.1, init_scale=1e-3, mu=0.0, sigma=1.0, activation_type='gaussian'):
        super(BottleneckAdapter, self).__init__()

        self.down_proj = nn.Linear(input_dim, bottleneck_dim)

        if activation_type == 'gaussian':
            self.activation = GaussianMembershipFunction(mu=mu, sigma=sigma)
        elif activation_type == 'relu':
            self.activation = nn.ReLU()

        self.up_proj = nn.Linear(bottleneck_dim, input_dim)
        self.dropout = nn.Dropout(dropout)
        self.layer_norm = nn.LayerNorm(input_dim)

        self.alpha = nn.Parameter(torch.tensor(0.1, dtype=torch.float))
        with torch.no_grad():
            self.down_proj.weight.data.normal_(mean=0.0, std=init_scale)
            self.down_proj.bias.data.zero_()
            self.up_proj.weight.data.normal_(mean=0.0, std=init_scale)
            self.up_proj.bias.data.zero_()

    def forward(self, x):
        residual = x
        x = self.layer_norm(x)
        x = self.down_proj(x)
        x = self.activation(x)
        x = self.dropout(x)
        x = self.up_proj(x)

        return residual + self.alpha * x
```

Fig. 4. Snippet of Fuzzy Adapter implementation

The training process was identical for all models, containing adapters to ensure a fair comparison. Standard Cross-Entropy Loss was used, given that this is a multi-class classification task [17]. The Adam optimiser was selected [18]. The model was trained for five epochs. At each epoch, loss and accuracy were calculated on the training set, followed by full validation on the test set. The key performance metric was test-set accuracy, as the data distribution was balanced [19].

The key result of the study is the accuracy comparison of the three approaches: (1) ViT Head-Only, (2) ViT with ReLU Adapter, and (3) ViT with Fuzzy Adapter. The final accuracy scores on the test dataset are summarised in Table 1. A visualisation of the predictions for our proposed variant is shown in Figure 5.

Table 1

Comparison of model accuracy

Model	Accuracy, %
ViT Head-Only	85,7
ViT with ReLu Adapter	90.2
ViT with Fuzzy Adapter	**92.6**

Fig. 5. Prediction visualisation

The results clearly demonstrate the significant advantages of using adapters for transfer learning compared to training only the classification head. The ViT Head-Only model yielded the lowest accuracy (85.7%). Integrating standard adapters with ReLU activation increased accuracy by 4.5%, to 90.2%. Our proposed model, which utilises a trainable Gaussian membership function, achieved an accuracy of 92.6%. This outperforms the standard ReLU adapter by 2.4%, confirming the effectiveness of integrating fuzzy activation functions for more flexible model adaptation.

3. Conclusion and perspectives of further development

The obtained results confirm the effectiveness of parameter-efficient fine-tuning (PEFT) but, more importantly, highlight the significant impact of the choice of activation function within adapters.

First, the significant gap of 4.5% between ViT Head-Only (85.7%) and ViT with ReLU Adapter (90.2%) confirms that training only the classification head is insufficient, whereas adding adapters allows the model to adapt significantly better to a new data domain.

Second, the key result of this study is that ViT with Fuzzy Adapter (92.6%) outperforms ViT with ReLU Adapter (90.2%). We hypothesise that this is due to differences in activation functions. This provides the model with significantly greater flexibility. The ability to model uncertainty and learn smoother transformations is likely the reason for the superior adaptation to the specifics of the new dataset. Despite the promising results, this work has limitations. The study was conducted on a single reduced dataset (CIFAR-10), using one base model ("deit_tiny") and a single fuzzy function, and evaluating only one metric: accuracy.

The obtained results show several promising directions for further research:

— investigation of other membership functions;

— integration into other PEFT architectures;

— analysis of hyperparameters and other ViT models (e.g. LaViT [20]);

— utilisation of other data sources and more specific domains.

In this paper, we investigated the effectiveness of integrating fuzzy logic into parameter-efficient fine-tuning methods for ViT. We introduced Fuzzy-Adapters – a modification of Bottleneck adapters where a trainable Gaussian membership function replaces the standard ReLU activation function.

Experiments conducted on a reduced CIFAR-10 dataset using the "deit_tiny" model demonstrated the superiority of the proposed approach. Our ViT with Fuzzy Adapter model achieved the highest accuracy of 92.6%. This outperforms both the baseline approach, which trains only the classification head (85.7%), and the standard ViT with ReLU Adapter (90.2%).

The results suggest that providing adapters with greater flexibility through parameterised, smooth activation functions derived from fuzzy logic enables the model to adapt more effectively to the features of a new domain. So, this study confirms that hybrid neuro-fuzzy approaches are a promising direction for the further development of parameter-efficient fine-tuning methods.

References

[1] An image is worth 16x16 words: transformers for image recognition at scale / A. Dosovitskiy et al. 2021. URL: https://doi.org/10.48550/arXiv.2010.11929.

[2] Parameter-Efficient fine-tuning methods for pretrained language models: a critical review and assessment / L. Xu et al. 2023. URL: https://doi.org/10.48550/arXiv.2312.12148.

[3] Fuzzy logic membership function. URL: https://researchhubs.com/post/engineering/fuzzy-system/fuzzy-membership-function.html.

[4] Parameter-Efficient transfer learning for NLP / N. Houlsby et al. 2019. URL: https://doi.org/10.48550/arXiv.1902.00751.

[5] O'Shea K., Nash R. An introduction to convolutional neural networks. 2015. URL: https://doi.org/10.48550/arXiv.1511.08458.

[6] Attention is all you need / A. Vaswani et al. 2023. URL: https://doi.org/10.48550/arXiv.1706.03762.

[7] Bukhanovskyi V., Ryabova N. Transfer learning methods in computer vision. Сучасні проблеми і досягнення в галузі радіотехніки, телекомунікацій та інформаційних технологій. Запоріжжя, Україна, 2024. P. 323–324.

[8] Azunre P. Transfer learning for natural language processing. Manning Publications Co. LLC, 2021.

[9] LoRA: low-rank adaptation of large language models / E. J. Hu et al. 2021. URL: https://doi.org/10.48550/arXiv.2106.09685.

[10] Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning / H. Liu et al. 2022. URL: https://doi.org/10.48550/arXiv.2205.05638.

[11] Li X. L., Liang P. Prefix-Tuning: optimizing continuous prompts for generation. 2021. URL: https://doi.org/10.48550/arXiv.2101.00190.

[12] Lester B., Al-Rfou R., Constant N. The power of scale for parameter-efficient prompt tuning. 2021. URL: https://doi.org/10.48550/arXiv.2104.08691.

[13] Deep residual learning for image recognition / K. He et al. 2015. URL: https://doi.org/10.48550/arXiv.1512.03385.

[14] Swin transformer: hierarchical vision transformer using shifted windows / Z. Liu et al. 2021. URL: https://doi.org/10.48550/arXiv.2103.14030.

[15] Training data-efficient image transformers\& distillation through attention / H. Touvron et al. 2021. URL: https://doi.org/10.48550/arXiv.2012.12877.

[16] Krizhevsky A., Nair V., Hinton G. CIFAR-10 and CIFAR-100 Datasets. URL: https://www.cs.toronto.edu/~kriz/cifar.html.

[17] Mao A., Mohri M., Zhong Y. Cross-Entropy loss functions: theoretical analysis and applications. 2023. URL: https://doi.org/10.48550/arXiv.2304.07288.

[18] Kingma D. P., Ba J. Adam: a method for stochastic optimization. 2017. URL: https://doi.org/10.48550/arXiv.1412.6980.

[19] Google for Developers. Classification: accuracy, recall, precision, and related metrics. URL: https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall.

[20] You only need less attention at each stage in vision transformers / S. Zhang et al. 2024. URL: https://doi.org/10.48550/arXiv.2406.00427.

Received (Надійшла) 09.02.2026

Accepted for publication (Прийнята до друку) 01.03.2026

Publication date (Дата публікації) 27.03.2026