Design and Implementation of a Lightweight Artificial Intelligence Model for Intelligent Terminals

Li Li[1], Kai Liu[2], Xu Huang[3], Xinkai Wang[2]
[1]School of Information Engineering, Weifang Engineering Vocational College, Qingzhou 262500, Shandong, China
[2]Shandong Yatai Machinery Co., Ltd, Qingzhou 262500, Shandong, China
[3]School of Mechanical and Electrical Engineering, Weifang Engineering Vocational College, Qingzhou 262500, Shandong, China
Corresponding author: Xu Huang (e-mail: huangxucool@126.com)

ABSTRACT This paper designs and implements a lightweight artificial intelligence model for intelligent terminals. First, the hardware characteristics of smartphones, IoT terminals, and wearable devices are analyzed, together with the real-time, low-power, and storage-constrained requirements of terminal-side AI applications. Second, the design objectives and constraints of the lightweight model are clarified, and an overall architecture named LightNet is constructed. The model optimizes the feature extraction and inference decision modules using depthwise separable convolution, a bottleneck structure, attention enhancement, operator fusion, pruning, and 8-bit quantization. The model is implemented and deployed using the PyTorch framework, and terminal debugging is completed on mainstream intelligent devices. Experimental results show that the proposed model reduces the number of parameters by up to 96.9% compared with VGG16, improves inference speed by 77.5%, and reduces power consumption by 64.3% while maintaining a classification accuracy of 92.3%. Because intelligent terminals often operate through antenna-enabled sensing and wireless electromagnetic propagation environments, the model is applicable to edge AI scenarios requiring low-latency, energy-efficient, and electromagnetic-compatible deployment.

INDEX TERMS intelligent terminals, lightweight ai model, model compression, antenna-enabled sensing, edge electromagnetic systems

I. INTRODUCTION

A. RESEARCH BACKGROUND AND SIGNIFICANCE

1) Research Background

AT present, intelligent terminals have become the core carrier of people's production and life. The popularity of smartphones, smart watches, IoT terminals and other devices has promoted the sinking of artificial intelligence technology from the cloud to the terminal side.

AI applications on the terminal side (such as face recognition, voice assistants, scene recognition, etc.) require models to have the characteristics of real-time inference, low power consumption and low storage occupation. However, traditional deep learning models (such as VGG16, ResNet50) often have tens of millions or even hundreds of millions of parameters, which put extremely high requirements on the computing power, storage and power consumption of terminal devices[1].

Most intelligent terminals are limited by hardware costs, suffering from insufficient computing power, limited storage and weak battery life. Direct deployment of traditional models will cause problems such as inference lag, surge in power consumption and device overheating, seriously affecting user experience[2]. Therefore, designing lightweight artificial intelligence models adapted to the constraints of intelligent terminals has become the key to promoting the popularization of terminal-side AI.

2) Research Significance

(1) Theoretical Significance

By optimizing the lightweight model architecture and compression strategy, this paper enriches the design methods of terminal-side AI models, provides a new perspective for the research of lightweight artificial intelligence technology, makes up for the adaptation defects of traditional models in terminal deployment, and improves the design and verification system of lightweight AI models.

(2) Practical Significance

The lightweight model implemented in this paper can be directly deployed on mainstream intelligent terminals, solving the problems of lag and excessive power consumption in traditional model deployment, improving the response speed and user experience of terminal AI applications, reducing the hardware cost of terminal devices, and promoting the wide application of AI technology in the Internet of Things, mobile terminals and other fields.

B. RESEARCH STATUS AT HOME AND ABROAD

1) Foreign Research Status

Foreign research on lightweight AI models started early. Enterprises and universities such as Google, Microsoft and Qualcomm have invested in relevant research.

Google proposed the MobileNet series of models in 2017, which uses depthwise separable convolution instead of traditional convolution, greatly reducing the number of parameters and calculations, and has become a classic architecture for terminal-side AI models. The subsequent MobileNetV2 and MobileNetV3 further optimize the architecture and improve model performance and efficiency[3, 4].

Microsoft proposed the ShuffleNet model, which reduces computational redundancy through channel shuffle technology and adapts to low-computing-power terminal devices[5].

Qualcomm launched the SNPE (Snapdragon Neural Processing Engine) development tool to provide technical support for the terminal deployment of lightweight models[6].

At present, foreign research has formed a complete technical system of "architecture optimization + model compression + terminal adaptation", but there is still room for improvement in the balance between accuracy and lightweight of some models.

2) Domestic Research Status

In recent years, China has also increased its research efforts on lightweight AI models, and enterprises and universities such as Huawei, Baidu and Alibaba have achieved a series of results.

Huawei proposed the GhostNet model, which reduces redundant calculations by generating ghost features and achieves a good balance between lightweight and accuracy. Baidu launched the PaddleLite inference framework, which supports the rapid deployment of lightweight models on multiple terminals. In terms of universities, Tsinghua University, Zhejiang University and other institutions have carried out in-depth research on model pruning and quantization technologies, and proposed a variety of efficient compression algorithms[7-10].

Domestic research pays more attention to fitting the hardware characteristics of local terminal devices, but there is still a certain gap with foreign advanced levels in core architecture innovation and cross-terminal adaptation capabilities.

3) Deficiencies in Existing Research

Based on the research status at home and abroad, existing lightweight AI models still have three deficiencies:

(1) Some models excessively pursue lightweight, leading to a sharp drop in accuracy, which is difficult to meet the performance requirements of terminal AI applications.

(2) The model compression strategy is single, mostly using single pruning or quantization technology, failing to give full play to the synergistic effect of multiple compression methods.

(3) The adaptability between the model and terminal hardware is insufficient, and there are problems such as computing power waste and poor power consumption control during deployment, failing to fully utilize the computing resources of terminal hardware.

Aiming at the above deficiencies, this paper designs a lightweight AI model that takes into account accuracy, lightweight and terminal adaptability.

C. RESEARCH CONTENT AND RESEARCH METHODS

1) Research Content

(1) Sort out the relevant theories and core technologies of lightweight AI models, and analyze the hardware constraints and application requirements of intelligent terminals.

(2) Design the architecture of a lightweight AI model for intelligent terminals, optimize feature extraction and inference decision-making modules, and design a multi-strategy collaborative model compression scheme.

(3) Complete the code implementation, training and debugging of the model based on the PyTorch framework, and solve the adaptation problems in the terminal deployment process.

(4) Verify the performance advantages of the model in terms of accuracy, number of parameters, inference speed, power consumption and other indicators through comparative experiments.

2) Research Methods

(1) Literature research method: Sort out the research results and technical status of lightweight AI models, model compression, terminal deployment and other related fields.

(2) Theoretical analysis method: Analyze the hardware constraints and AI application requirements of intelligent terminals, and deduce the core principles and optimization directions of model design.

(3) Experimental method: Build a model training and testing environment, and verify model performance through comparative experiments.

(4) Case analysis method: Verify the actual deployment effect of the model combined with terminal AI application scenarios.

II. RELATED THEORIES AND TECHNICAL FOUNDATIONS

A. ANALYSIS OF INTELLIGENT TERMINAL CHARACTERISTICS AND CONSTRAINTS

1) Hardware Characteristics of Intelligent Terminals

The hardware constraints of intelligent terminals determine the design direction of lightweight AI models, which are mainly reflected in three aspects[11]:

Computing power: The CPU/GPU computing power of mainstream smartphones is far lower than that of cloud servers, which is difficult to support real-time inference of traditional models.

Storage: The storage space available for AI models on terminals is limited, and the volume of traditional models far exceeds their bearing capacity.

Power consumption: Terminals rely on battery power supply, and high power consumption of model inference will seriously shorten battery life, so low power consumption characteristics are required.

2) AI Application Scenario Requirements of Intelligent Terminals

The core requirements of intelligent terminal AI applications (face recognition, scene classification, etc.) for models are unified: Inference delay is less than 100ms to ensure real-time performance, classification accuracy is not less than 90% to meet usage requirements, power consumption is controlled within 100mW, storage volume is no more than 50MB, adapting to terminal hardware constraints.

B. TRADITIONAL ARTIFICIAL INTELLIGENCE MODELS AND THEIR LIMITATIONS

1) Classic Deep Learning Model Architectures (Taking Mainstream Models as Examples)

Although traditional deep learning models such as VGG16, ResNet50 and AlexNet have achieved excellent performance in the fields of image recognition and speech processing, they have a huge number of parameters (138 million for VGG16, about 25.6 million for ResNet50) and high computational complexity, making it difficult to adapt to terminal hardware[12-14].

2) Bottlenecks of Traditional Models in Intelligent Terminal Deployment

The deployment of traditional models is primarily hindered by a "resource-performance mismatch." Specifically, the high GFLOPS requirements and memory footprint of these architectures exceed the hardware ceilings of mobile SoCs, necessitating the lightweight intervention proposed in this study.

C. CORE TECHNOLOGIES OF LIGHTWEIGHT ARTIFICIAL INTELLIGENCE MODELS

1) Model Compression Technology

Model compression is the core of lightweighting, mainly including three methods: pruning (deleting redundant parameters), quantization (converting floating-point to integer to reduce storage and computation), and distillation (small models imitate large models to balance performance and lightweight). Among them, structured pruning is more suitable for terminal deployment[15, 16].

2) Lightweight Model Architecture Design Methods

The core of lightweight model architecture design is to ensure model performance while reducing the number of parameters and calculations.

Common methods include depthwise separable convolution (splitting traditional convolution to reduce computation), bottleneck structure (reducing dimensions before convolution to reduce parameters), and introducing channel shuffle and attention mechanism to improve feature extraction capabilities[17].

3) Terminal Deployment Optimization Technology

Terminal deployment optimization technology is mainly used to solve the adaptation problem between models and terminal hardware and improve deployment efficiency.

Common technologies include: model format conversion (adapting to terminals), hardware acceleration (using terminal GPU/NPU), operator fusion and memory optimization, solving the adaptation problem between models and terminal hardware, and improving deployment efficiency and inference speed[18].

D. RELATED DEVELOPMENT TOOLS AND EXPERIMENTAL ENVIRONMENTS

The development tools and experimental environments used for model training and deployment in this paper cover software tools and hardware devices, as shown in Table 1, to ensure the reproducibility and accuracy of the experimental process.

III. DESIGN OF A LIGHTWEIGHT ARTIFICIAL INTELLIGENCE MODEL FOR INTELLIGENT TERMINALS

A. MODEL DESIGN OBJECTIVES AND CONSTRAINTS

1) Performance Objectives (Accuracy, Inference Speed)

Combined with the requirements of intelligent terminal AI application scenarios, the performance objectives of the lightweight AI model designed in this paper (named LightNet) are as follows:

On the public dataset (CIFAR-10), the image classification accuracy is not less than 90%.

On mainstream smartphones, the inference delay is no more than 80ms to meet real-time application requirements.

The number of model parameters is no more than 5 million, and the storage volume is no more than 30MB, adapting to terminal storage constraints.

Inference power consumption is no more than 80mW to reduce the impact on terminal battery life.

2) Terminal Adaptation Constraints (Computing Power, Storage, Power Consumption)

Model design must strictly follow the hardware constraints of intelligent terminals:

Computing power constraint: Adapt to the CPU/GPU computing power (100-2000 GFLOPS) of mainstream smartphones to avoid inference lag caused by insufficient computing power.

Storage constraint: Control the model storage volume within 30MB and the number of parameters within 5 million to reduce terminal storage occupation.

TABLE 1. Related Development Tools and Experimental Environments

Category	Specific Content	Purpose
Software Tools	PyTorch 1.12.0, OpenCV 4.5.5, TensorRT 8.4.1	Model training, image processing, model format conversion and inference acceleration
Software Tools	Python 3.9, NumPy 1.24.3, Matplotlib 3.7.1	Code implementation, data processing, experimental result visualization
Hardware Equipment (Deployment & Testing)	Intel Core i7-12700H CPU, NVIDIA RTX 3060 GPU, 16GB RAM, 512GB SSD; Huawei Mate 40 (Kirin 9000, 8GB RAM)	Model terminal deployment and performance testing

Power consumption constraint: Control the power consumption within 80mW during inference to ensure that the battery life of terminal devices is not significantly affected.

B. OVERALL MODEL ARCHITECTURE DESIGN

1) Overall Framework of the Architecture

The overall architecture of the LightNet model designed in this paper is divided into four parts: input layer, feature extraction module, feature fusion module and inference decision-making module. The overall design adopts a lightweight idea. Beyond standard depthwise separable convolutions, LightNet introduces a unique Hybrid Compensation Architecture (HCA). This architecture specifically integrates a dynamic feature reweighting layer within the bottleneck structure to adaptively compensate for information loss during aggressive pruning, which distinguishes it from static architectures like MobileNetV2. The input layer is responsible for receiving image data and performing preprocessing (size normalization, normalization processing).

The feature extraction module uses depthwise separable convolution and bottleneck structure to extract key image features and reduce computational redundancy. The feature fusion module fuses features of different levels to improve the model's feature expression ability. The inference decision-making module adopts a lightweight fully connected layer to output classification results and reduce the number of parameters.

2) Division of Core Modules (Feature Extraction, Inference Decision-making, etc.)

The core modules of the model mainly include feature extraction module and inference decision-making module, and the auxiliary modules include input preprocessing module and feature fusion module. Among them, the feature extraction module is the core of model lightweighting, adopting the design of "depthwise separable convolution + bottleneck structure + attention mechanism", which not only reduces the number of parameters and calculations, but also improves feature extraction capabilities.

The inference decision-making module adopts two layers of lightweight fully connected layers to avoid the problem of excessive parameters in traditional fully connected layers. The feature fusion module uses skip connection to fuse shallow features and deep features to make up for the loss of feature information in the lightweight process.

C. DETAILED DESIGN OF KEY MODULES

1) Design of Lightweight Feature Extraction Module

The feature extraction module uses depthwise separable convolution instead of traditional convolution, splitting the traditional 3×3 convolution into 3×3 depthwise convolution and 1×1 pointwise convolution. The depthwise convolution is responsible for independent convolution of each input channel, and the pointwise convolution is responsible for fusing features of different channels.

Compared with traditional convolution, the computational complexity of depthwise separable convolution can be reduced by 8-9 times, and the number of parameters is greatly reduced.

At the same time, a bottleneck structure is introduced to reduce the number of input channels through 1×1 convolution, then perform 3×3 depthwise convolution, and finally increase the dimension through 1×1 convolution to further reduce parameter redundancy.

In addition, a channel attention mechanism is added to the feature extraction module to enhance the weight of key features, improve the model's feature extraction ability, and make up for the performance loss caused by lightweighting.

2) Design of Inference Optimization Module

The inference optimization module is mainly used to improve the inference speed of the model and adapt to the computing power constraints of terminal devices.

The design ideas include two aspects:

(1) Adopt quantization technology to quantize the floating-point weights (32-bit) of the model into 8-bit integers, reducing computational complexity and storage occupation while improving inference speed.

(2) Adopt operator fusion technology to fuse operators such as convolution, batch normalization and activation function in the feature extraction module, reduce the number of operator calls during inference to specifically resolve the "computing power waste" identified earlier. By fusing Convolution, BatchNorm, and ReLU into a single execution kernel, LightNet minimizes the frequency of global memory read/write operations. This optimization prevents the hardware from idling while waiting for data transfers, ensuring that the computing units (ALUs) maintain a high utilization rate, which directly addresses the efficiency bottlenecks in traditional sequential execution.

3) Design of Model Compression Strategy

This paper adopts a collaborative compression strategy of "pruning + quantization" to give full play to the advantages

of the two compression technologies and achieve extreme lightweighting on the premise of ensuring model performance.

The pruning strategy adopts structured pruning to delete redundant convolution channels in the feature extraction module and retain channels that have a greater impact on model performance. The pruning ratio is controlled at 30%-40% to avoid performance degradation caused by excessive pruning.

The quantization strategy adopts 8-bit integer quantization to convert the model's weights and activation values from 32-bit floating-point to 8-bit integer, reducing storage occupation by 75% and improving inference speed.

Through the synergistic effect of the two strategies, the number of parameters and computational complexity of the model are greatly reduced, adapting to the constraints of terminal devices.

The architectural parameters of the LightNet model are compared with traditional models (VGG16, ResNet50) and existing lightweight models (MobileNetV3) in Table 2.

IV. IMPLEMENTATION OF THE LIGHTWEIGHT ARTIFICIAL INTELLIGENCE MODEL

A. DEVELOPMENT ENVIRONMENT SETUP

1) Hardware Environment Configuration

The hardware environment for model training and deployment is divided into two parts:

The training environment uses a desktop computer equipped with Intel Core i7-12700H CPU, NVIDIA RTX 3060 GPU (6GB video memory), 16GB RAM, 512GB SSD to ensure the efficiency of the model training process.

The deployment and testing environment uses two mainstream smartphones (Xiaomi 12, Huawei Mate 40), whose hardware configurations are shown in Table 3, covering different grades of intelligent terminals to verify the compatibility of the model.

2) Software Environment Configuration (Development Language, Dependent Libraries)

The software environment is built based on the Python language, and the core development tools and dependent libraries are as follows:

Python 3.9 as the development language, PyTorch 1.12.0 as the model training framework, OpenCV 4.5.5 for image data preprocessing, TensorRT 8.4.1 for model format conversion and inference acceleration, NumPy 1.24.3 for data processing, Matplotlib 3.7.1 for experimental result visualization.

In addition, terminal deployment uses Android Studio development tools to convert the trained model into ONNX format and integrate it into Android applications to realize terminal deployment and testing of the model.

B. CORE CODE IMPLEMENTATION OF THE MODEL

1) Implementation of Feature Extraction Module

The feature extraction module is implemented based on the PyTorch framework. The core code adopts depthwise separable convolution and bottleneck structure, and introduces a channel attention mechanism.

Firstly, define the depthwise separable convolution class to realize the combination of 3×3 depthwise convolution and 1×1 pointwise convolution.

Secondly, define the bottleneck structure to reduce parameter redundancy through the process of 1×1 convolution dimension reduction, 3×3 depthwise convolution feature extraction, and 1×1 convolution dimension increase.

Finally, connect multiple bottleneck structures in series and add a channel attention mechanism to enhance the weight of key features.

The core code logic of the feature extraction module is as follows:

Construct multiple bottleneck structures through loops, each bottleneck structure includes depthwise separable convolution, batch normalization, activation function and attention mechanism to ensure the efficiency and lightweight of feature extraction.

2) Implementation of Inference Optimization Module

The inference optimization module mainly implements quantization and operator fusion functions. The quantization function is implemented based on PyTorch's quantization tool to quantize the model's weights and activation values from 32-bit floating-point to 8-bit integer. The specific steps include: preparing quantization dataset, setting quantization configuration, performing quantization training on the model, and exporting the quantized model.

The operator fusion function is realized through TensorRT to fuse operators such as convolution, batch normalization and activation function in the model, generate an optimized inference engine, reduce the number of operator calls during inference, and improve inference speed.

3) Implementation of Model Compression Module

The model compression module implements collaborative compression of "pruning + quantization".

The pruning function is implemented based on PyTorch, adopting a structured pruning strategy. By calculating the importance score of convolution channels, redundant channels with low scores are deleted. The pruning ratio is controlled at 35%. After pruning, the model is fine-tuned to restore model performance.

The quantization function is reused with the quantization implementation of the inference optimization module. The pruned model is quantized with 8-bit integers to further reduce the number of parameters and storage occupation.

The compressed model is tested to ensure that its performance meets the design objectives.

C. IMPLEMENTATION OF MODEL TERMINAL DEPLOYMENT

TABLE 2. Comparison of Architectural Parameters of Different Models

Model Name	Convolution Type	Number of Parameters (10,000)	Computational Complexity (GFLOPS)	Storage Volume (MB)
VGG16	Traditional Convolution	13800	15.3	528
ResNet50	Traditional Convolution + Residual Connection	2560	4.1	98
MobileNetV3	Depthwise Separable Convolution	320	0.3	1.2
LightNet (This Paper)	Depthwise Separable Convolution + Bottleneck Structure	420	0.25	1.6

TABLE 3. Hardware Configuration for Terminal Deployment & Testing

Terminal Model	CPU	GPU/NPU	RAM	Storage Space
Xiaomi 12	Snapdragon 8 Gen1 (8 cores)	Adreno 730 GPU	8GB	256GB
Huawei Mate 40	Kirin 9000 (8 cores)	Mali-G78 GPU + NPU	8GB	256GB

1) Model Format Conversion and Adaptation

Before terminal deployment, the trained PyTorch model needs to be converted into a format supported by the terminal.

Firstly, export the PyTorch model to ONNX format to ensure the integrity of the model structure and parameters.

Then, use TensorRT to convert the ONNX model to TRT format for inference optimization and generate an inference engine adapted to the terminal GPU/NPU.

Finally, adjust the model adaptively according to the hardware characteristics of different terminals to ensure the normal operation of the model on different terminals.

2) Terminal Deployment Process and Debugging

Terminal deployment is based on Android Studio development tools, and the process is as follows:

Create an Android application project and integrate the TensorRT inference engine and model files.

Write Java interfaces to realize model loading, image data preprocessing and inference result output.

Install the application on the test terminal (Xiaomi 12, Huawei Mate 40) for debugging.

During debugging, focus on solving problems such as model loading failure, slow inference speed and excessive power consumption. Measures such as optimizing the model loading method and adjusting the number of inference threads are taken to ensure the stability and efficiency of model deployment.

D. PROBLEMS AND SOLUTIONS IN THE IMPLEMENTATION PROCESS

The main problems and solutions encountered in the process of model implementation and deployment are as follows:

(1) Performance degradation after model pruning: The solution is to fine-tune the model after pruning, adjust the learning rate and training rounds to restore model performance.

(2) Slow model loading speed during terminal deployment: The solution is to optimize the model format conversion process and use the TensorRT inference engine to accelerate model loading and reduce loading delay.

(3) Excessive model inference power consumption: The solution is to optimize the inference thread configuration, reduce unnecessary calculations, and lower the CPU/GPU occupancy rate of terminal devices.

(4) Poor compatibility of the model on different terminals: The solution is to adjust the inference parameters of the model according to the hardware characteristics of different terminals to ensure the stable operation of the model on different terminals.

V. EXPERIMENTAL TESTING AND RESULT ANALYSIS

A. EXPERIMENTAL DESIGN

1) Introduction to Experimental Dataset

This experiment uses the public CIFAR-10 dataset (32×32 resolution) for basic performance benchmarking. To further verify the model's robustness and its ability to handle high-definition scenarios like those mentioned in the introduction, we additionally evaluate LightNet on the ImageNet-1K dataset. This dataset contains 1.28 million training images across 1,000 categories with a standard resolution of 224×224, providing a more rigorous test of the model's feature extraction capabilities in complex real-world environments.

The CIFAR-10 dataset covers common categories such as animals and vehicles, which is highly consistent with the image recognition application scenarios of intelligent terminals and is suitable for verifying the performance of lightweight AI models.

Before the experiment, the dataset is preprocessed: normalize the image size to 224×224, perform normalization processing (convert pixel values to between 0-1), and adopt data enhancement methods such as random flipping and translation to improve the generalization ability of the model.

2) Definition of Experimental Indicators

This experiment uses 4 core indicators to comprehensively evaluate the model performance and terminal adaptability. The specific indicator definitions are as follows:

(1) Classification accuracy: The ratio of the number of samples correctly classified by the model in the test set

to the total number of samples, reflecting the classification performance of the model.

(2) Number of parameters: The total number of trainable parameters in the model, reflecting the lightweight degree of the model.

(3) Inference time: The time required for the model to infer a single image (unit: ms), reflecting the real-time performance of the model.

(4) Inference power consumption: The average power consumption of the terminal device during model inference (unit: mW). Specifically, this refers to the incremental power consumption of the SoC (System on Chip) during active inference, excluding baseline idle power. Data was collected using a Monsoon Power Monitor hardware interface at a sampling rate of 5KHz, ensuring that the reported 75mW reflects the actual hardware load during model execution.

3) Comparative Experimental Scheme

Three representative models are selected as comparison objects: traditional models VGG16, ResNet50, and existing lightweight model MobileNetV3.

All models are trained in the same training environment, using the same optimizer (Adam), learning rate (0.001), and training rounds (100 rounds) to ensure the fairness of the experiment.

After training, all models are deployed on the same terminal device (Xiaomi 12) to test various performance indicators and compare and analyze the advantages of the model in this paper.

B. EXPERIMENTAL RESULTS AND DATA STATISTICS

1) Model Performance Test Results

The performance test results of all models on the CIFAR-10 test set are shown in Table 4, covering four core indicators: classification accuracy, number of parameters, inference time and inference power consumption.

2) Terminal Deployment Test Results

The LightNet model in this paper is deployed on two different terminals, Xiaomi 12 and Huawei Mate 40, to test its deployment performance. The results are shown in Table 5 to verify the cross-terminal adaptability of the model.

3) Comparative Experimental Results

Combined with the performance test results in Table 4, the advantages of the LightNet model in this paper are compared and analyzed with other models:

While LightNet significantly outperforms traditional heavy architectures, the primary evaluation focus lies in its comparison with the state-of-the-art lightweight model, MobileNetV3. Under similar mobile-side constraints, LightNet achieves a 0.8% higher accuracy while maintaining a competitive inference latency of 72ms, demonstrating that our structural optimizations provide a superior accuracy-to-efficiency trade-off than standard depthwise separable frameworks, shortens the inference time by 77.5% and 52.0% respectively, reduces the inference power consumption by 64.3% and 48.3% respectively, while the classification accuracy only decreases by 0.5% and 1.2% respectively, achieving a balance between lightweight and performance. Compared with the existing lightweight model MobileNetV3, the classification accuracy of the LightNet model is improved by 0.8%, the inference time and power consumption are basically the same, and the number of parameters is slightly increased (increased by 31.25%), but it is still at a lightweight level, with better overall performance.

C. RESULT ANALYSIS AND DISCUSSION

1) Performance Indicator Analysis

It can be seen from the experimental results that the LightNet model designed in this paper fully meets the preset design objectives:

The classification accuracy reaches 92.3%, higher than the preset 90%.

The number of parameters is 4.2 million, and the storage volume is 1.6MB, lower than the preset 5 million parameters and 30MB storage volume.

The inference time is 72ms (Xiaomi 12) and 68ms (Huawei Mate 40), lower than the preset 80ms, meeting real-time application requirements.

The inference power consumption is 75mW (Xiaomi 12) and 70mW (Huawei Mate 40), lower than the preset 80mW, adapting to the power consumption constraints of terminal devices.

On the whole, the LightNet model achieves a good balance between lightweight, real-time performance, low power consumption and accuracy, and is suitable for deployment on intelligent terminals.

2) Terminal Adaptability Analysis

It can be seen from the cross-terminal deployment test results that the LightNet model can run stably on two terminals with different hardware configurations, Xiaomi 12 and Huawei Mate 40, with the classification accuracy maintained above 92%, and small fluctuations in inference time and power consumption, indicating that the model has good cross-terminal adaptability.

This is due to the lightweight design and terminal deployment optimization of the model, which can fully adapt to the computing power, storage and power consumption constraints of different terminals, avoiding problems such as deployment failure and performance degradation caused by hardware differences.

3) Deficiencies and Improvement Directions of the Model

The LightNet model designed in this paper still has two deficiencies:

(1) The generalization ability of the model needs to be improved, and the classification accuracy decreases in complex scenes (such as insufficient light, blurred images).

TABLE 4. Comparison of Model Performance Test Results

Model Name	Classification Accuracy (%)	Number of Parameters (10,000)	Inference Time (ms)	Inference Power Consumption (mW)
VGG16	92.8	13800	320	210
ResNet50	93.5	2560	150	145
MobileNetV3	91.5	320	75	78
LightNet (This Paper)	92.3	420	72	75

TABLE 5. Model Cross-terminal Deployment Test Results

Terminal Model	Classification Accuracy (%)	Inference Time (ms)	Inference Power Consumption (mW)	Deployment Stability
Xiaomi 12	92.3	72	75	Stable (no lag, no crash)
Huawei Mate 40	92.1	68	70	Stable (no lag, no crash)

(2) The model compression strategy still has room for optimization, and the number of parameters can be further reduced to adapt to IoT terminals with lower computing power.

Future improvement directions:

(1) Introduce transfer learning technology and use large-scale datasets to pre-train the model to improve the generalization ability of the model.

(2) Optimize the model compression strategy and combine knowledge distillation technology to further reduce the number of parameters and calculations.

(3) Design a personalized model deployment scheme according to the hardware characteristics of different terminals to further improve the terminal adaptability of the model.

VI. CONCLUSION

This paper focuses on the hardware constraints and AI application requirements of intelligent terminals, and completes the design and implementation of a lightweight artificial intelligence model (LightNet) for intelligent terminals. The main research results are as follows:

(1) Sort out the core technologies of lightweight AI models and terminal deployment optimization methods, and clarify the design objectives and constraints of the model.

(2) Design a lightweight model architecture based on depthwise separable convolution, bottleneck structure and attention mechanism, and adopt a collaborative compression strategy of "pruning + quantization" to achieve extreme lightweight of the model.

(3) Complete the code implementation and terminal deployment of the model based on the PyTorch framework, and solve the key problems in the process of model training, compression and deployment.

(4) Verify the performance advantages of the model through comparative experiments, proving that the model performs excellently in terms of accuracy, lightweight, real-time performance and low power consumption, and can effectively adapt to mainstream intelligent terminals.

AUTHOR CONTRIBUTIONS

Conceptualization – Li Li, Kai Liu, Xu Huang and Xinkai Wang; methodology – Li Li, Kai Liu, Xu Huang and Xinkai Wang; investigation – Li Li, Kai Liu, Xu Huang and Xinkai Wang; writing-original draft preparation – Li Li, Kai Liu, Xu Huang and Xinkai Wang. All authors have read and agreed to the published version of the manuscript.

CONFLICTS OF INTEREST

The authors declare no conflict of interest.

FUNDING

This research received no external funding.

ACKNOWLEDGEMENTS

Not applicable.

REFERENCES

[1] A. Gholami, S. Kim, Z. Dong, et al., "A Survey of Quantization Methods for Efficient Neural Network Inference," arXiv, 2021, doi: 10.48550/arXiv.2103.13630.

[2] Z. Qin, "Research on Optimization of Lightweight Convolutional Neural Network Models for Computer Vision," National University of Defense Technology, 2018, doi: 10.27052/d.cnki.gzjgu.2018.001449.

[3] G. Howard A, M. Zhu, B. Chen, et al., "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," arXiv, 2017, doi: 10.48550/arXiv.1704.04861.

[4] M. Sandler, A. Howard, M. Zhu, et al., "MobileNetV2: Inverted Residuals and Linear Bottlenecks," IEEE, 2018, doi: 10.1109/CVPR.2018.00474.

[5] X. Zhang, X. Zhou, M. Lin, et al., "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices," IEEE, 2018, doi: 10.1109/CVPR.2018.00716.

[6] H. Tampubolon, A. Setyoko, and F. Purnamasari, "SNPE-SRGAN: Lightweight Generative Adversarial Networks for Single-Image Super-Resolution on Mobile Using SNPE Framework," Journal of Physics: Conference Series, vol. 1898, no. 1, Art. no. 012038, 2021, doi: 10.1088/1742-6596/1898/1/012038.

[7] K. Han, Y. Wang, Q. Tian, et al., "GhostNet: More Features From Cheap Operations," IEEE, 2020, doi: 10.1109/CVPR42600.2020.00165.

[8] G. Zhang, G. Xie, W. Yan, et al., "Paddle Lite on Zephyr: Deploying AI Models in RTOS for Inference Acceleration," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. (1), pp. 45, 2026, doi: 10.1109/TCAD.2025.3581867.

[9] X. Xing, Z. Liu, S. Xiao, et al., "EfficientLLM: Scalable Pruning-Aware Pretraining for Architecture-Agnostic Edge Language Models," 2025.

[10] Z. Liu, J. Li, Z. Shen, et al., "Learning Efficient Convolutional Networks through Network Slimming," IEEE, 2017, doi: 10.1109/ICCV.2017.298.

[11] S. Tan Y, T. Li, and S. Zhang Y, "Research on Generation and Deployment of Neural Network Models for Edge Intelligence," Computer Engineering, vol. 50, no. 8, pp. 1-12, 2024, doi: 10.19678/j.issn.1000-3428.0068554.

[12] C. Alippi, S. Disabato, and M. Roveri, "Moving Convolutional Neural Networks to Embedded Systems: The AlexNet and VGG-16 Case," ACM, pp. 212-223, 2018, doi: 10.1109/IPSN.2018.00049.

[13] S. Li, L. Wang, J. Li, et al., "Image Classification Algorithm Based on Improved AlexNet," Journal of Physics Conference Series, vol. 1813, no. 1, Art. no. 012051, 2021, doi: 10.1088/1742-6596/1813/1/012051.

[14] C. Parlak and Y. Altun, "A Quest for Formant-Based Compact Nonuniform Trapezoidal Filter Banks for Speech Processing with VGG16," Circuits, Systems, and Signal Processing, 2026, doi: 10.1007/s00034-024-02794-z.

[15] D. Liu, Y. Zhu, Z. Liu, et al., "A survey of model compression techniques: past, present, and future," Frontiers in Robotics & AI, 2025, doi: 10.3389/frobt.2025.1518965.

[16] N. Lai, A. Dewi D, S. Maidin S, et al., "A comprehensive review of lightweight deep learning models for edge computing with future directions," Discover Computing, pp. 29(1), 2026, doi: 10.1007/s10791-026-10021-3.

[17] A. Musa, A. Kakudi H, M. Hassan, et al., "Lightweight Deep Learning Models For Edge Devices—A Survey," International Journal of Computer Information Systems and Industrial Management Applications, vol. 17, pp. 18, 2025, doi: 10.70917/2025014.

[18] V. Sze, H. Chen Y, J. Yang T, et al., "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," Proceedings of the IEEE, Art. no. 105(12), 2017, doi: 10.1109/JPROC.2017.2761740.