SSRG International Journal of Electrical and Electronics Engineering
ISSN: 2348-8379/ https://doi.org/10.14445/23488379/IJEEE-V11I11P139

Volume 11 Issue 11, 424-437, November 2024

Original Article

Energy-Aware Task Offloading in Massive IoT Edge Network Using Optimized Convolutional Neural Network

Shoukath Cherukat[1], J. Benita[2]

[1,2]*Department of Electronics and Communication Engineering, Noorul Islam Centre for Higher Education, Tamilnadu, India.*

[1]*Corresponding Author : shoukathniche@gmail.com*

Received: 22 September 2024 Revised: 23 October 2024 Accepted: 21 November 2024 Published: 30 November 2024

Abstract *- The importance of the Internet of Things (IoT) devices is rising in this digital age. However, the high energy consumption of these devices limits their long-term functioning and efficiency. Task offloading technique in IoT is often used to address energy optimization. Conventional task offloading strategies lead to shorter lifespans and inefficient use of devices. Energy-aware task offloading is critical in IoT edge networks within the Mobile Edge Computing (MEC) environments to enhance resource utilization and reduce latency. This paper proposes an optimized Convolutional Neural Network (CNN) for efficient task offloading. The method efficiently predicts optimal offloading decisions using a comprehensive dataset that includes network characteristics, user behavior and resource utilization features. The optimized CNN architecture achieved notable performance with 91.075% accuracy, 91.82% precision, 91.07% recall, and 90.74% F1 score. These results showed that the model ensures efficient resource allocation and extends the operational life of IoT devices. The key findings highlight the potential of Deep Learning (DL) models in contributing to the energy-aware task offloading field in real-time adaptive decision making within the MEC environments.*

Keywords *- Internet of Things, Energy optimization, Mobile Edge Computing, Deep Learning, Convolutional Neural Network.*

1. Introduction

The revolution of IoT transforms the interaction of devices with each other and the digital world. This interconnected network comprises numerous smart devices, sensors and actuators that constantly generate, process and transmit data [1]. As the scale and complexity of IoT grows, the traditional cloud computing solutions that transfer data to centralized servers for processing face several challenges, including high latency, bandwidth constraints and energy inefficiency. Edge computing extends the data source to computational resources, reducing latency and minimizing network congestion. The MEC expands the concept by providing computing resources at the edge of the mobile network infrastructure, like base stations or access points.

An important strategy of MEC is task offloading, which involves transferring computationally demanding tasks from IoT devices to edge servers or cloud resources. The process minimizes energy consumption and maximizes resource utilization to enhance the system's performance. Thus, IoT devices enable faster data processing and real-time analytics by utilizing MEC to offload computationally demanding jobs to adjacent edge servers [2]. This decentralized method enhances data security and privacy without compromising latency by keeping sensitive data closer to its source. By carefully selecting which task should executed locally and which should be offloaded, task offloading achieved a balance between resource usage, energy efficiency and user satisfaction. However, there are significant challenges to effective task offloading due to the dynamic nature of networks, unpredictable task demands and varying device capabilities.

In the context of IoT, energy efficiency is crucial, especially for battery-powered devices, which have high energy consumption and lead to shorter battery life [3]. For the widespread adoption of IoT, prolonged operations without battery replacement and recharging are essential. So, optimization of energy consumption is essential during task offloading. The application of energy-aware task offloading strategies makes IoT deployments more sustainable and cost-effective by extending the operational lifetime of IoT devices. The task offloading methods use Machine Learning (ML), optimization algorithms and network analytics to make intelligent decisions based on task execution and placement [4].

In order to choose the best offloading technique, network congestion, device capabilities, energy consumption, and user

preferences are considered. The availability of real-time data about the network and device condition is critical for effectively designing a task-offloading strategy and ensuring efficient resource utilization. While effective in certain aspects, conventional methods face tasks adjusting to dynamic network conditions, optimizing energy usage, and ensuring low latency in MEC environments. These methods often rely on static decision-making frameworks that fail to respond to real-time changes in network characteristics and user demands. This study addresses these gaps by proposing an optimized deep learning model for energy-aware task offloading in IoT edge networks. The model utilizes advanced techniques like dropout regularization and batch normalization to improve the robustness and adaptability of task offloading decisions. This novel approach links the gap between existing static models and the need for real-time, energy-efficient offloading solutions in dynamic IoT ecosystems. The novelty of this work lies in the optimized design and application of the DL model specifically for task offloading in MEC environments. The main contributions of the suggested study are as follows:

- To develop a novel DL model for optimizing task offloading decisions in IoT edge networks.
- To develop a real-time adaptive offloading mechanism that dynamically adjusts with network and device changes.
- To predict the offloading types based on the input values.

The remaining section of this paper is structured as follows: Section 2 provides related work on task offloading in IoT edge networks, emphasizing the advantages and limitations of various methods and the identified research gaps. Section 3 describes the proposed methodology, including the optimization process. Section 4 gives experimental results and discussions demonstrating the efficiency of the proposed model. Finally, Section 5 concludes the major findings of the study.

2. Literature Review

Sada et al. [5] proposed a Lightweight Inference Task Offloading and Sever Selection (LITOSS) framework with two stage approach employing a lightweight genetic algorithm, considering both the local and server side for task scheduling and Reinforcement Learning (RL) for edge server selection. LITOSS surpassed other meta-heuristic methods in accuracy and speed but had limitations in server load and energy cost estimation.

Pradeep et al. [6] employed the Energy Prediction and Task Optimization (EPTO) algorithm, including LSTM-based energy prediction, dynamic distribution of resources and advanced offloading policies. Results showcased efficient energy enhancement with reduced task completion time, extended device lifespan, and challenges involved with potential energy prediction inaccuracies and computational overhead.

Baker et al. [7] focused on Software-Defined Networks (SDN) and Deep Reinforcement Transfer Learning (DRTL) in edge computing environments for dynamic task offloading by employing Long Short-Term Memory Network (LSTM)-based trust score prediction. Results demonstrated improved latency and energy efficiency with limitations in security risks and adjustments across various IoT contexts. Abdullaev et al. [8] developed a Deep Belief Network (DBN) and Seagull Optimization (SGO) for parameter tuning in task offloading. An objective function with latency and resource constraints minimizes energy consumption. Simulations showed that the model outperformed conventional models with a maximum reward of 89.67% by holding limitations regarding the optimization of makespan.

Almuseelem [9] proposed an energy-aware task offloading with load balancing for Edge-Cloud Computing (ECC) as an integer problem based on an advanced encryption method. According to location, user and data rate, the mobile devices are redistributed with an edge because of the load-balancing algorithm. The results showed remarkable energy saving with an increase of 20.3%. However, limitations included handling mobile device mobility and automating security decisions. Sellami et al. [10] addressed a Deep Reinforcement Learning (DRL) method based on blockchain. The Proof-of-Authority Blockchain consensus was coupled with the Asynchronous Actor-Critic Agent (A3C) policy for network validation. The model outperformed consensus algorithms by 50% better energy efficiency. Challenges included the interoperability between distributed ledgers and non-independent data distribution in distributed agents.

Sellami et al. [11] investigated energy-aware and low-latency task scheduling using a Deep Q-learning process as energy-constrained. Results outperformed A3C algorithms, achieving 87% better energy savings and 50% faster task assignments with data ownership and privacy challenges. Mahapatra et al. [12] proposed Proximal Policy Optimization (PPO) for optimizing energy consumption in edge servers by analyzing the current environment to select the optimum location incorporating both server and link transmission energy consumption. The simulation used Dijkstra's algorithm and achieved an average energy saving of 22.69%, yet was limited by the variability and complexity of the edge environment.

Zhao et al. [13] investigated Orthogonal Frequency Division Multiple Access (OFDMA), considering latency requirements to minimize the energy consumption based on offloading ratio selection, subcarrier and computing resource allocation and transmission power optimization. The results demonstrated that the model can save 20-40% energy by comparing conventional algorithms. Naouri et al. [2] proposed a three-layer framework with device, cloudlet, and cloud layers for task offloading to differentiate tasks based on computing and communication costs. A greedy task graph

partition offloading algorithm was employed to reduce the costs. The main limitation included issues of highly variable network conditions and potential inefficiencies.

Bi et al. [14] focused on Particle swarm optimization based on Genetic Learning (PGL) and formulation and derivation of analytical solutions. Outcomes showed improved energy efficiency within the deadlines by neglecting mobile device energy consumption. Wang et al. [15] used energy beam forming for wireless power transfer employing convex optimization techniques for energy-efficient task offloading in MEC systems, showing minimization compared to both local computing and full offloading with challenges over dynamic networks and nonlinear energy models. Tu et al. [16] developed an Online Predictive Offloading (OPO) algorithm by joining DRL and LSTM to reduce the cost by optimizing task latency, energy consumption, and discard rate for edge computing in IoT.

Developing real-world implementation of an intelligent task offloading solution with real-time data accuracy is crucial. The dynamic network conditions affect the prediction of server loads and energy consumption. Metaheuristic algorithms show promising results, yet the real-world efficiency is not verified. Security, privacy, and handling mobility in dynamic networks remain significant issues. Moreover, reliance on simulated data fails to capture real-world complexities. For instance, while LITOSS demonstrates efficiency through lightweight inference, its limitations in handling server-side constraints must be discussed. Similarly, EPTO achieves dynamic energy optimization but faces computational overhead challenges. Methods like DBN and SGO provide high accuracy but lack adaptability in variable network environments.

3. Materials and Methods

The growing demand for reliable and efficient IoT by managing energy consumption, minimizing latency and maximizing resource utilization is crucial. This study explores energy aware task offloading using an optimized Convolutional Neural Network, ensuring effective operation in resource constrained environments by predicting the types of offloading based on the input values. The collected dataset is preprocessed to transform the data to input into an optimized CNN for testing and prediction. The detailed process flow is illustrated in Figure 1.

3.1. Dataset

In the context of task offloading assessment, certain features were chosen from the dataset with prevalence impact and potential for the decision-making process. The dataset consists of details about network characteristics like Location Area Code (LAC), a unique number that identifies a location area within a cellular network, and Cell Identity (CI), which identifies a cell in the network.

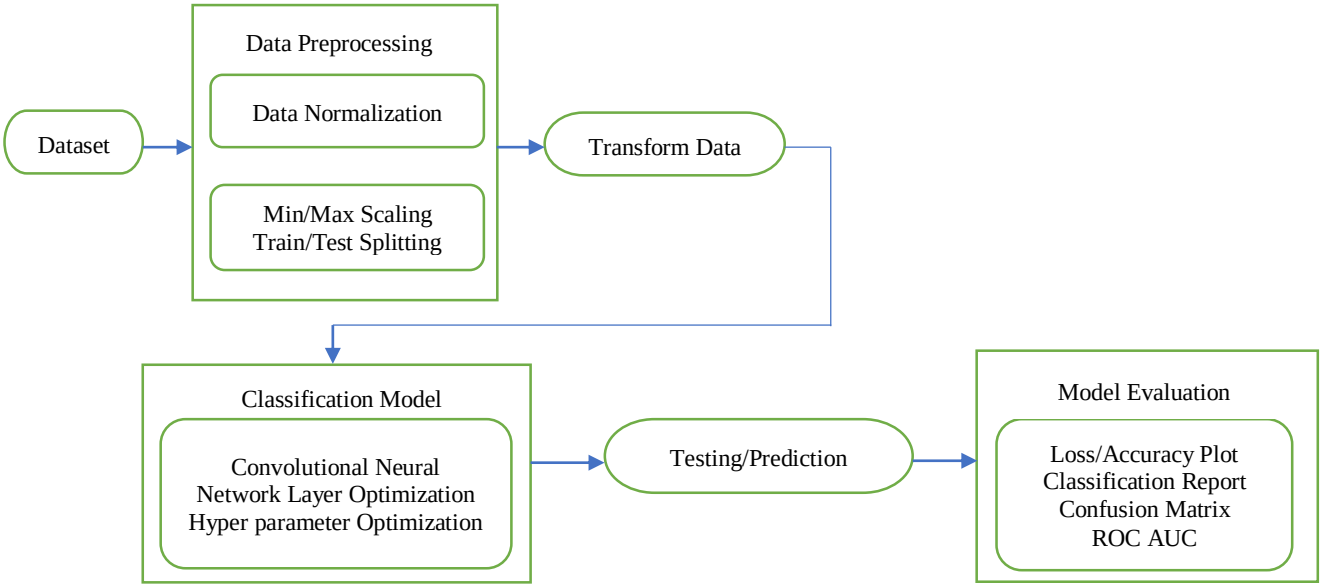


Fig. 1 Block diagram of the workflow

Radio Access Type (RAT) and network traffic information (such as downlink and uplink traffic) with respect to MEC. User behavior features such as end time and duration and user agent highlighting user activities and usage patterns enable the offloading strategy to adapt to dynamic user demands and preferences. Resource utilization features, like the availability of computing resources in the edge cloud, influence the balance between the loads and optimize resource allocation. The characteristics related to latency, such as network latency and Round-Trip Time (RTT), are crucial metrics affecting user

experience for task offloading evaluation. Additionally, the duration of the data transfer, throughput, and data transmission rate are important. These features collectively enable reliable analysis of task offloading decisions, ensuring network conditions, user activities, resource availability, and latency. The sample data is illustrated in Figure 2. Thus, the dataset used in this study is highly relevant to task offloading scenarios in MEC environments, as it incorporates features critical for analyzing real-world IoT systems. Key attributes provide comprehensive insights into the factors influencing offloading decisions. The dataset's large size of 204,823 instances ensures robust training and testing of the model, allowing it to generalize well across diverse scenarios. Furthermore, including dynamic user activity data, such as session durations and traffic patterns, alongside network characteristics like RTT and downlink/uplink traffic, makes the dataset suitable for capturing the variability inherent in IoT ecosystems. This diversity helps address challenges like class imbalance and identifies the most influential features for decision-making. By enabling the development of adaptable offloading strategies, the dataset effectively supports the study's goal of optimizing energy efficiency and resource allocation in MEC environments. The data type and the statistical analysis are given in Figures 3, and 4, respectively.

```
[ ] df.head(10)
```

	Cell_Identity(CI)	Downlink_Traffic	LAC	Duration	RAT	Uplink_Traffic	Latency	Throughput	Offloading_Decision
0	11803	464	3288	2	2	585	2.000000	5.245000e+02	Mobile Device
1	20723	0	43023	15	1	300	15.000000	2.000000e+01	Mobile Device
2	11803	514	3288	1	2	559	1.000000	1.073000e+03	Mobile Device
3	7483	900	3287	82	2	559	87.542193	1.666625e+01	Edge Cloud
4	20723	1276	43023	2131	1	2280	2131.000000	1.668700e+00	Mobile Device
5	45193	0	43046	0	1	60	0.000000	1.099823e+06	Mobile Device
6	3553	79	2184	0	2	63	5.006833	2.836124e+01	Edge Cloud
7	1971	630	43042	7	1	986	7.000000	2.308571e+02	Mobile Device
8	7483	2831	3287	79	2	5836	79.000000	1.097089e+02	Mobile Device
9	3553	384	2184	26	2	1550	26.000000	7.438462e+01	Mobile Device

Fig. 2 Sample dataset

```
[ ] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 204823 entries, 0 to 204822
Data columns (total 9 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   Cell_Identity(CI)    204823 non-null  int64
 1   Downlink_Traffic     204823 non-null  int64
 2   LAC                  204823 non-null  int64
 3   Duration             204823 non-null  int64
 4   RAT                  204823 non-null  int64
 5   Uplink_Traffic       204823 non-null  int64
 6   Latency              204823 non-null  float64
 7   Throughput           204823 non-null  float64
 8   Offloading_Decision  204823 non-null  object
dtypes: float64(2), int64(6), object(1)
memory usage: 14.1+ MB
```

Fig. 3 Data type

The data visualization provides a comprehensive understanding of the characteristics of the task offloading dataset. With count plots, histograms, and correlation analysis, significant insights were observed regarding the distribution of offloading decisions, the range and variation of feature values, and the relationships between features and the target variable. Figure 5 plots the number of tasks offloading cases categorized by the offloading decision: Mobile Device and Edge Cloud. The plot displays a higher count of tasks executed by mobile devices than edge count, highlighting the class imbalance to understand the distribution of task offloading decisions, which is crucial for developing and evaluating offloading strategies in MEC environments.

Histograms provided insights into the distributions of individual features such as latency, throughput and traffic patterns, revealing their ranges and identifying potential imbalances. This will ensure the data quality and effective preprocessing to improve the accuracy of task offloading. The distribution of eight features is given in Figure 6. The correlation analysis given in Figure 7 explains the relationships between features and offloading types. The plot indicated that throughput has the highest positive correlation with the offloading type, indicating it is the most influencing feature regarding the offloading decision between mobile devices and edge cloud. These visualizations help optimize task offloading in MEC environments by identifying the most important factors affecting decision-making.

```
[ ] df.describe()
```

	Cell_Identity(CI)	Downlink_Traffic	LAC	Duration	RAT	Uplink_Traffic	Latency	Throughput
count	204823.000000	2.048230e+05	204823.000000	204823.000000	204823.000000	2.048230e+05	204823.000000	2.048230e+05
mean	26501.715896	1.084362e+04	37885.538182	73.439125	1.164996	1.646325e+03	75.406172	3.715297e+05
std	17093.052232	4.156146e+05	10963.402341	1027.743207	0.371178	6.914085e+04	1027.407201	5.194687e+05
min	11.000000	0.000000e+00	2180.000000	0.000000	1.000000	0.000000e+00	0.000000	4.028587e-03
25%	11451.000000	0.000000e+00	42246.000000	0.000000	1.000000	1.280000e+02	0.000000	7.275847e+01
50%	24821.000000	2.880000e+02	42270.000000	1.000000	1.000000	4.190000e+02	5.150011	6.060003e+02
75%	40178.000000	1.179000e+03	43014.000000	9.000000	1.000000	1.028000e+03	12.594214	1.099823e+06
max	65483.000000	1.555968e+08	57699.000000	86105.000000	2.000000	3.011028e+07	86105.000000	1.099823e+06

Fig. 4 Statistical analysis of the dataset

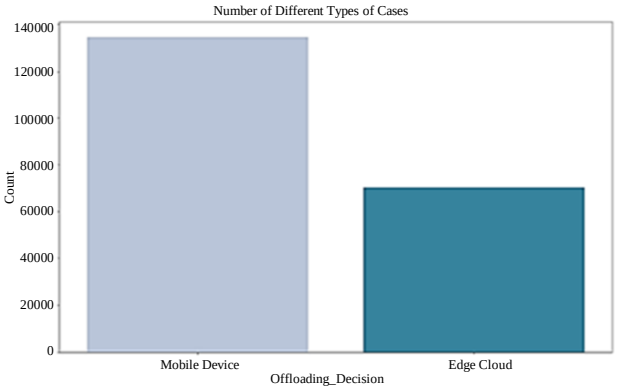


Fig. 5 Count of offloading decision

The correlation analysis given in Figure 7 explains the relationships between features and offloading types. The plot indicated that throughput has the highest positive correlation with the offloading type, indicating it is the most influencing feature regarding the offloading decision between mobile devices and edge cloud.

These visualizations help optimizing the task offloading in MEC environments by identifying the most important factor affecting the decision making. Additionally, the heatmap facilitated the identification of multicollinearity among features by detecting highly correlated feature pairs.

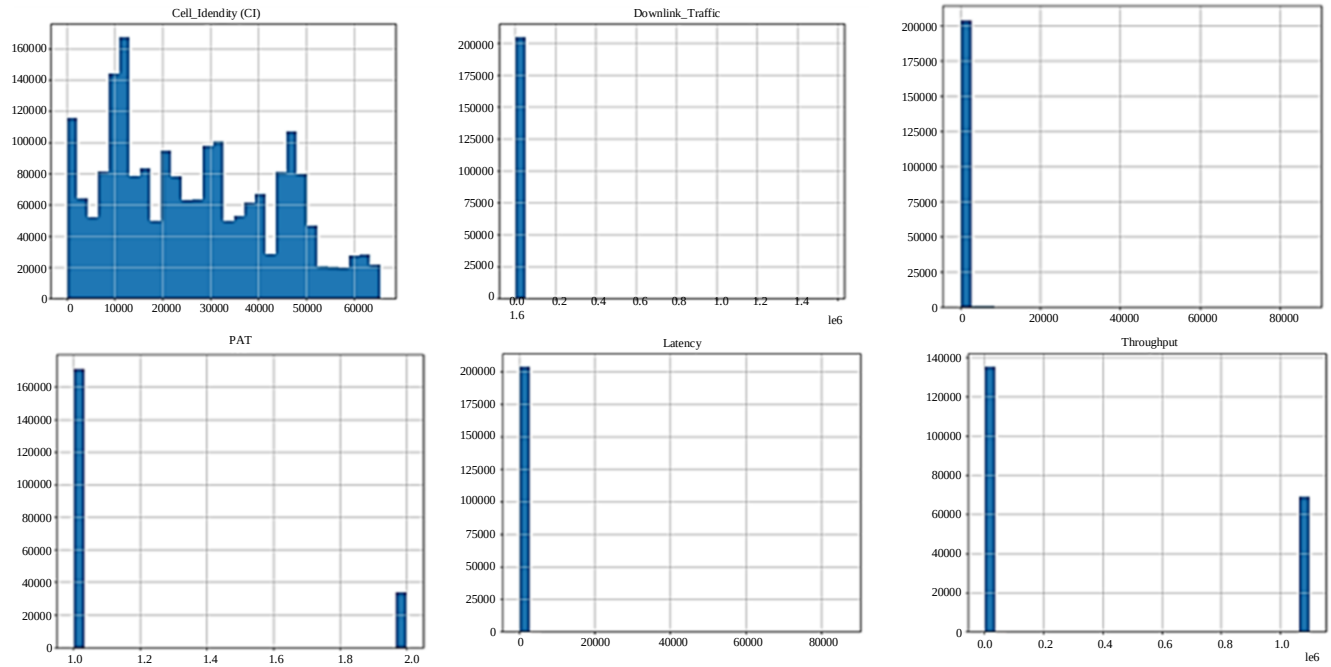


Fig. 6 Feature distribution of the dataset

By highlighting strong and weak correlations, as in Figure 8, the heatmap helps identify the most influential factors in offloading decisions, guiding the optimization of resource allocation and improving the performance of the optimized CNN model.

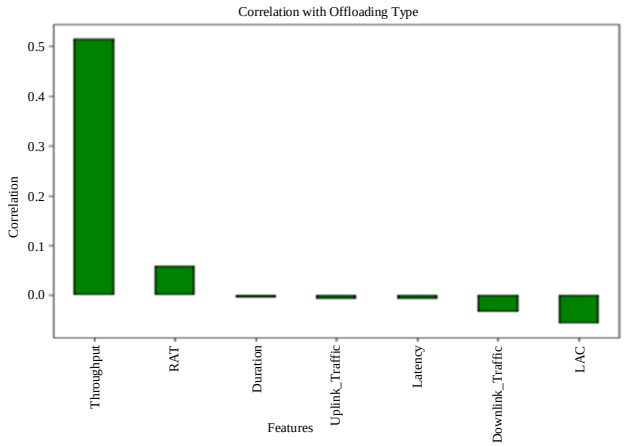


Fig. 7 Correlation of features with offloading type

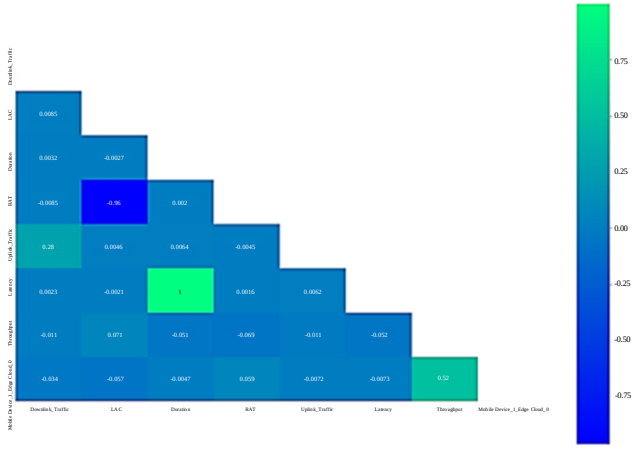


Fig. 8 Heat map of the data

3.2. Data Preprocessing

Data is preprocessed using data normalization and min-max scaling. Data normalization is organizing data in a database to remove redundancy and improve data integrity. The process involves structuring tables and recognizing relationships between them, which implies simplified data management across all records and fields to ensure the streamlining and consistency of the stored data.

The main aim of data normalization is to create a uniform data format across the system, making it more reliable for users to interact with queries and analyze the data effectively [17]. The process reduces duplication with great data quality and more effective retrieval processes. Min-Max scaling scales and transforms each feature individually to a range typically between 0 and 1. The process is an alternative to zero mean, unit variance scaling. Mathematically, (1) represents min-max scaling,

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}} \quad (1)$$

Where x is the original value of a feature x_{min} is the minimum, and x_{max} is the feature's maximum value in the dataset.

By maintaining the relative relationship between the other values, the min max scaler scaled linearly by preserving the effect of outliers, where the maximum value is represented by the largest data point that occurs, and the minimum value is represented by the smallest one.

3.3. Proposed Optimized Convolutional Neural Network

In the context of a massive IoT network, the paper proposed an optimized CNN for accurate and efficient task offloading in an MEC environment. This optimized CNN makes intelligent decisions regarding offloading computational tasks from a mobile to an edge cloud. CNN or ConvNet are specialized deep neural networks widely used to analyze visual imagery [18].

The basic architecture of the CNN is provided in Figure 9. The input layer of CNN is designed to adapt to the feature set of datasets, where each feature represents information about the network characteristics, user behavior, and resource utilization.

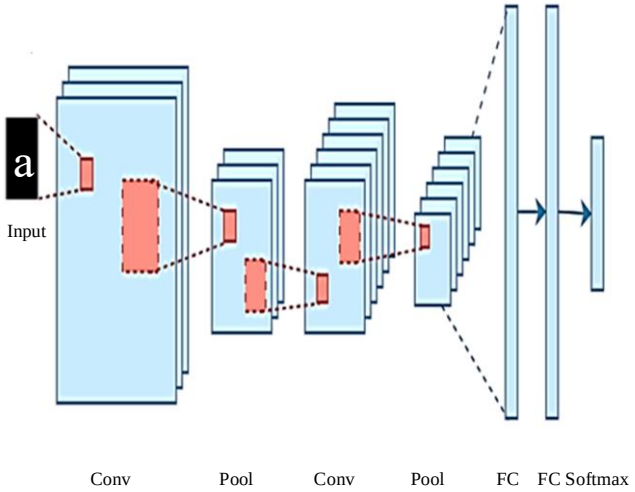


Fig. 9 Architecture of CNN

Unlike conventional methods that rely on matrix multiplication, CNN employs convolution, which involves a mathematical operation on two functions to create a third function that illustrates how one's shape is altered by the other. This is achieved by applying a filter or kernel (3×3 matrix) to the input image to extract convoluted features, which are then passed on to the next layer. Figure 10 represents the convolution operation.

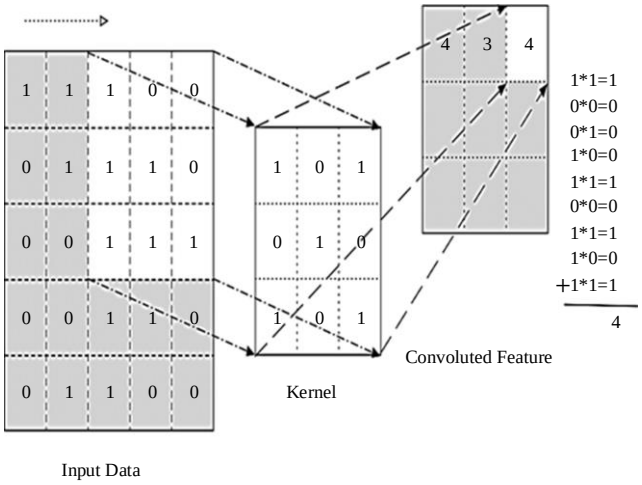


Fig. 10 Convolution operation

Max pooling, another major operation in CNN, chooses the maximum elements from the area of the feature map enclosed by the filter, effectively retaining the most important features of the earlier feature map and minimizing the spatial dimension of the feature maps [19].

The visualization of the max pooling operation is given in Figure 11. The max pooling operation is applied after the convolutional layer in the proposed optimized CNN, which helps maintain a balance between model efficacy and performance, ensuring that the network captures the most relevant features to prevent overfitting while being computationally feasible.

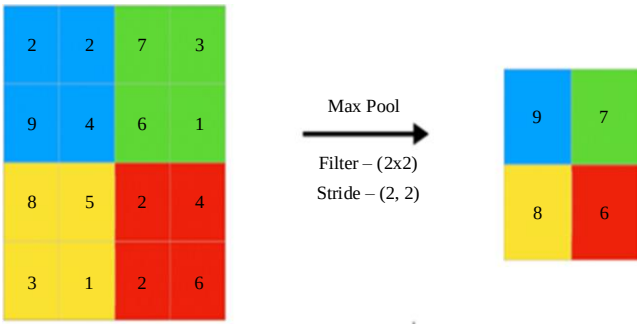


Fig. 11 Visualization of max pooling operation

The activation function, Rectified Linear Unit (ReLU), is mostly used in CNNs because it introduces non-linearity to the model by returning to zero for any negative input. For $f(x) = max\ (0, x)$, the function returns that value for any positive value x. The ReLU function is shown in Figure 12.

3.3.1. Optimized Dropout

In CNN, the dropout regularization technique prevents overfitting, which happens when a model learns to remember the training data instead of generalizing from it. It randomly sets a proportion of neurons to zero during training, which enables the network to learn more important features. The application of dropout regularization is defined by percentage: the standard neural network and network after dropout regularization are given in Figure 13.

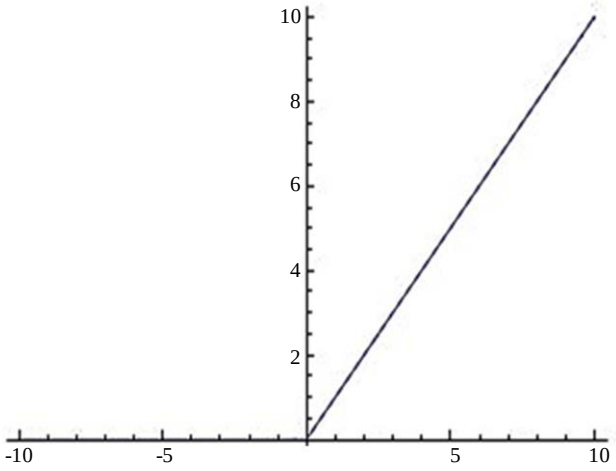
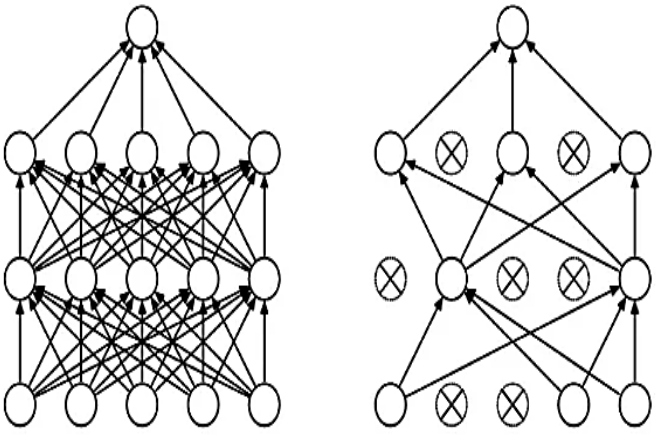


Fig. 12 ReLU function



(a) **(b)**

Fig. 13(a) Standard neural network, and (b) After applying dropout.

3.3.2. Optimized Batch Normalization

Batch normalization is crucial in CNN, which addresses internal covariate shifts by normalizing mini-batch activations. The normalization process involves the computation of the mean as given in (2).

$$\text{Mean, } \mu_{B=\frac{1}{x}\sum_{i=1}^{x} m_i} \tag{2}$$

The variance is calculated by (3),

$$\text{Variance } \sigma_B^2 = \frac{1}{x}\sum_{i=1}^{x}(m_i - \mu_B)^2 \tag{3}$$

Now, the activations are normalized as given by (4),

$$\widehat{x_i} = \frac{m_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad (4)$$

During the training process, batch normalization standardizes the activation of each layer by subtracting the mean and dividing by the standard deviation of the activations within the mini-batch, ensuring the input layer stays within a similar scale throughout training.

After normalization, the batch normalization introduces two learnable parameters per feature map as scale (γ) and shift (β). These parameters allow the model to adaptively scale and shift the normalized activations, giving the network more flexibility in learning the optimal representation for the data. The output after batch normalization is given by (5)

$$y_i = \gamma \widehat{x_i} + \beta \quad (5)$$

3.3.3. Optimized Activity Regularizer

Regularization techniques like activity regularization enhance the generalization of neural networks and prevent overfitting problems by adding a penalty term to the output of a layer based on its activations, discouraging large activation values, and promoting more generalized feature learning. The most common types of regularization using activity regularizer include L1 regularization, L2 regularization, and a combination of both, defined as elastic net regularization. L1 regularization adds a penalty equivalent to the absolute value of the magnitude of coefficients as defined by (6).

$$L1 = \lambda \sum_i |a_i| \quad (6)$$

Where λ represents regularization parameters for governing the strength of the penalty and $a_i$represents the activations. L2 regularizations add a penalty equal to the square of magnitude of the coefficients, defined by (7).

$$L2 = \lambda \sum_i a_i^2 \quad (7)$$

The L1 and L2 regularization combines for an elastic net, balancing the two as defined by (8).

$$Elastic\ net = \alpha L1 + (1 - \alpha) L2 \quad (8)$$

Where the parameter that controls the mix between L1 and L2 regularization is α. The optimizer will minimize the loss function where the regularization terms are added. The total loss with regularization can be given by (9),

$$Loss_{total} = Loss_{original} + \lambda R(a) \quad (9)$$

Where $R(a)$is the regularization term. The choice of the type and strength of regularization depends on factors such as the intricacy of the task, the architecture of the CNN, and the volume of available training data. L1 regularization is effective for feature selection, promoting sparsity by driving less important feature weights to zero. L2 regularization, on the other hand, tends to spread out the weight values more evenly. Elastic Net provides a balanced approach, leveraging the strengths of both L1 and L2 regularization.

3.3.4. Optimized Bias Regularizer

Regularizing the bias terms, in addition to the weights, helps achieve this goal by penalizing large bias values. Bias terms represent the intercept in a linear equation and add additional parameters to each neuron in a layer.

The bias regularization penalizes the bias terms to prevent them from becoming too large. L1, L2 and elastic net regularization are used for this purpose. A penalty equal to the absolute value of the magnitude of the bias terms is added by L1 regularization. The L1 regularization term for the biases $b\ b$ can be defined by (10).

$$L1_{bias} = \lambda \sum_j |b_j| \quad (10)$$

Where λ represents the regularization parameter that controls the strength of the penalty, and $b\ j$ represents the bias terms. L1 regularization encourages sparsity, potentially driving some of the bias terms to zero, which can simplify the model and enhance its interpretability. A penalty equal to the square of the magnitude of the bias terms is added by L2 regularization. The L2 regularization term for the biases b is given by (11).

$$L2_{bias} = \lambda \sum_j b_j^2 \quad (11)$$

L2 regularization discourages large bias values without necessarily driving them to zero, promoting smaller, more evenly distributed bias terms. The Elastic Net regularization term for the biases b given by (12)

$$Elastic\ net = \alpha\lambda \sum_j |b_j| + (1 - \alpha)\lambda \sum_j b_j^2 \quad (12)$$

Regularizing the bias terms and weights makes the neural network less likely to overfit the training data, leading to better performance on unseen data.

Now, in a fully connected layer, the fundamental building of a neural network, each neuron is connected to every neuron in the preceding layer to form a fully connected network, as shown in Figure 14. Each neuron in this layer performs a linear transformation on the input vector using a weights matrix, ensuring that every input influences every output.

Finally, the sigmoid function, used in the output layer, squashes input values to a range between 0 and 1, making it suitable for classification tasks. At z=0, when the curve crosses 0.5, establish the activation function's rules.

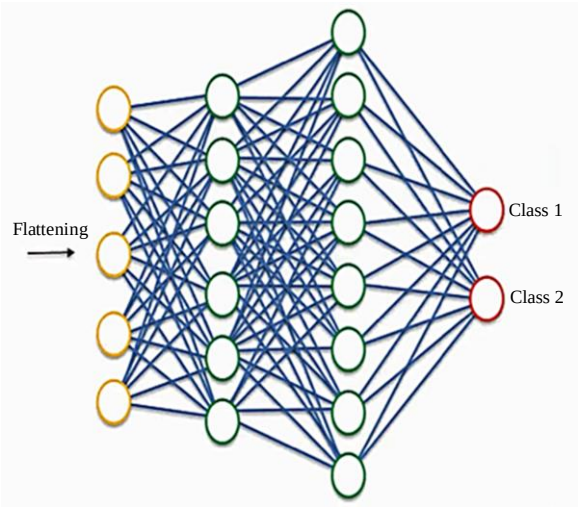


Fig. 14 Fully connected layer

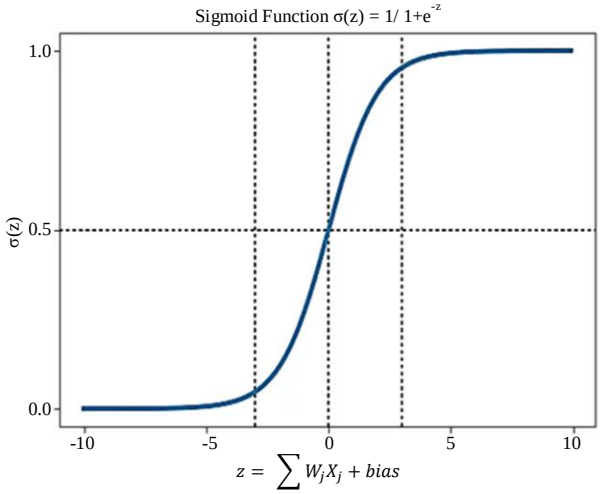


Fig. 15 Sigmoid activation curve

The function outputs 1 if the value is equal to or greater than 0.5 and 0 otherwise, providing a probabilistic interpretation of the outputs. This suite of operations and functions collectively enhances the optimized CNN's ability to learn and generalize from complex data. The sigmoid activation curve is shown in Figure 15. Thus, the optimized CNN architecture, comprising convolutional layers, max pooling, dropout regularization, batch normalization, and activity regularization, was designed to handle complex data patterns efficiently.

Convolutional layers extract essential features, max pooling reduces spatial dimensions, dropout mitigates overfitting, and batch normalization standardizes activations to improve stability during training. A fully connected layer further processes the extracted features, and the sigmoid function in the output layer provides probabilistic task classification.

Hyperparameter optimization was critical in fine-tuning the proposed CNN model to achieve optimal performance. Techniques such as grid search were employed to explore various combinations of parameters systematically. Batch sizes ranging from 128 to 1024 were evaluated to balance computational efficiency and convergence stability. The Adam optimizer was selected for its superior capability in minimizing loss during training, outperforming alternatives such as SGD and RMSProp. The learning rate was fine-tuned to ensure steady and efficient progress toward minimizing the objective function without overshooting or stagnation. Dropout rates were optimized to prevent overfitting while maintaining the model's capacity to learn complex patterns. Validation techniques confirmed that the selected hyperparameters yielded consistent and robust results across the dataset. This detailed optimization process ensured a well-calibrated model capable of making accurate and efficient offloading decisions in dynamic MEC environments. The optimized CNN architecture is illustrated in Figure 16, and the model summary is in Table 1.

Table 1. Model summary of the optimized CNN

Layer Type	Output Shape	Parameters
Conv1D	(None, 7, 128)	512
Conv1D	(None, 7, 128)	49280
MaxPooling1D	(None, 3, 128)	0
Dropout	(None, 3, 128)	0
Batch Normalization	(None, 3, 128)	512
Flatten	(None, 384)	0
Dense	(None, 512)	197120
Dropout	(None, 512)	0
Dense	(None, 1)	513
Total Parameters:		247937
Trainable parameters:		247681
Non-trainable parameters:		256

3.3.5. Hardware and Software Setup

A comprehensive setup is used for this study to ensure a well-equipped environment to handle the demand of neural network training and deployment consisting of NVIDIA GeForce GTX 1080Ti GPU, an Intel Core i7 processor, 32GB of RAM, and the Python-based Keras library integrated with the TensorFlow framework. Keras's intuitive interface and Google Colab's vast computational capabilities made developing models easier and more assured. The dataset was split into testing and training sets for the efficient training of the model. Hyperparameters are the adjustable parameters in a deep learning model that govern the training process and influence the model's performance.

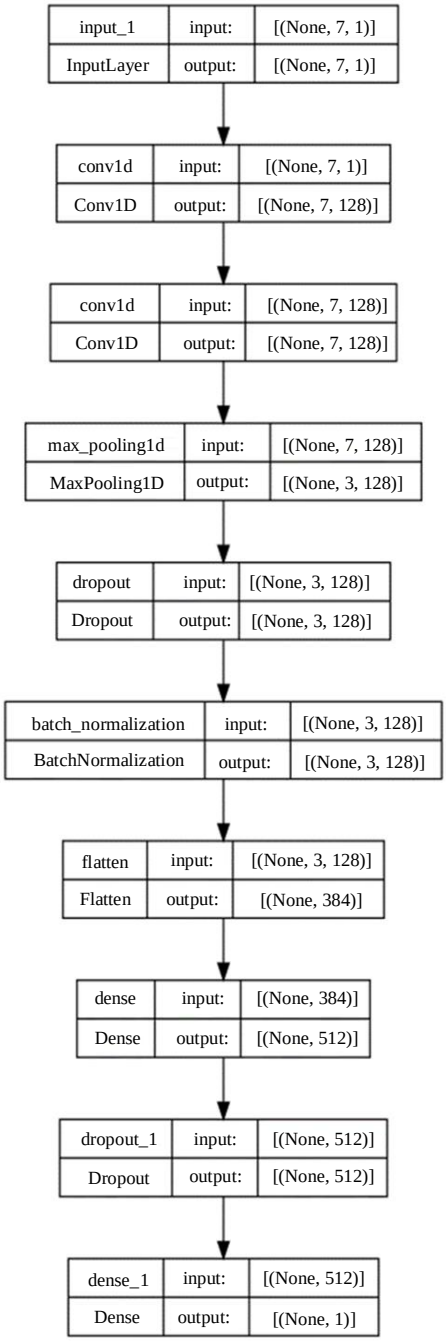


Fig. 16 Optimized CNN architecture

Tuning these parameters is crucial for optimizing the model's accuracy and efficiency. Table 2 mentions the hyperparameters of the proposed model.

4. Results and Discussions

4.1. Performance Evaluation

Performance evaluation of the model was conducted to ensure a comprehensive understanding of its effectiveness using a variety of metrics. The primary metrics in Table 3 highlight different aspects of the model's performance.

Table 2. Hyperparameter specifications

Hyperparameters	Values
Batch size	128,256,512,1024
Iterations	300
Optimizer	Adam, RMSProp, SGD, ADAGrad
Callback	80,85,91

Table 3. Evaluation metrics

Performance Metrics	Equations
Accuracy	(TP+TN)/(TP+TN+FP+FN)
Precision	TP/(TP+FP)
Recall	TP/(TP+FN)
F1 Score	2*(Precision*Recall) / (Precision+Recall)
Where, TP-True Positives, FP-False Positives, TN-True Negatives, and FN-False Negatives	

With the help of the evaluation metrics, the offloading type is predicted to be a mobile device or edge cloud. The classification report of the proposed model is illustrated in Table 4.

Table 4. Classification report

Evaluation Parameters	Results (%)
Accuracy	91.075
Precision	91.82
Recall	91.07
F1-Score	90.74

The classification report summarizes that the classification model predicts the type of offloading based on the dataset. With an accuracy of 91.075 %, the efficiency of the proposed model is demonstrated. A precision of 91.82% indicated the model's efficiency in minimizing false positives. Recall of 91.07 measures the model's ability to identify the most positive instances. The F1-score, which harmonizes precision and recall, was 90.74%, indicating a strong balance between the two and highlighting the model's overall robustness in predicting task offloading decisions accurately and reliably. The obtained miss classification score is 8.9244.

The study's accuracy and loss plots were crucial for assessing the model's performance throughout training. The accuracy plot indicated how well the model was learning, with training accuracy steadily improving and validation accuracy closely following, reflecting good generalization. The loss plot showed a consistent decrease in training and validation loss, suggesting effective learning and error minimization. No significant divergence between training and validation metrics was observed, indicating the absence of overfitting. As in Figure 17, these plots confirmed that the proposed model was

well-tuned, efficiently balancing learning from the training data and generalizing to unseen data.

The confusion matrix for predicting offloading types, with the classes being Edge Cloud and Mobile Device. It summarizes the results of predictions made by the model by comparing them to the actual labels. The matrix allows us to visualize the model's performance, as shown in Figure 18. Figure 19 represents the Receiver Operating Characteristic (ROC) curve, a graphical performance evaluation tool for binary classification models. It plots the TP rate, i.e. recall or sensitivity, versus the FP rate at different threshold settings.

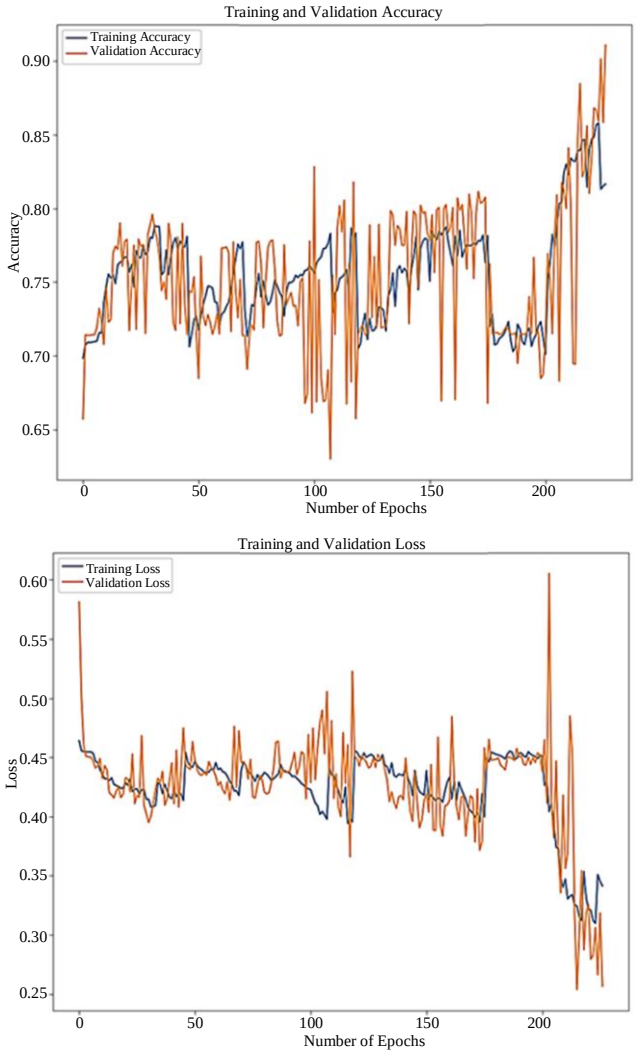


Fig. 17 Accuracy plot and loss plot of the proposed model

TPR measures how many actual positive identifications in the model are correct, and FPR measures vice versa: how many actual negatives are misclassified as positives. The ROC curve describes the trade-off between sensitivity and specificity graphically. The Area Under the Curve (AUC) represents the model performance across all threshold levels, with a value closer to 1 representing good performance. A perfect classifier achieves a point in the ROC space's top-left corner (0,1), reflecting high TPR and low FPR.

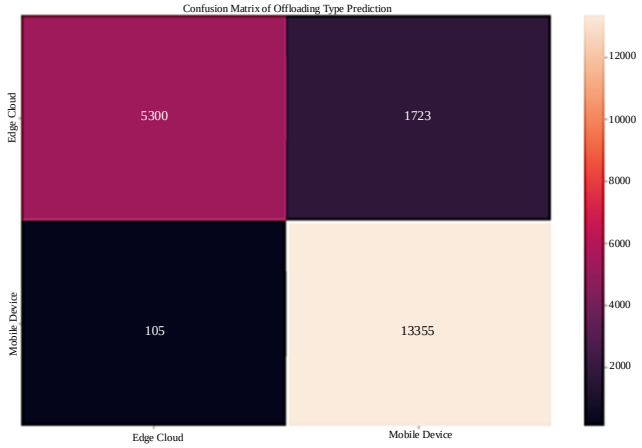


Fig. 18 Confusion matrix of the proposed method

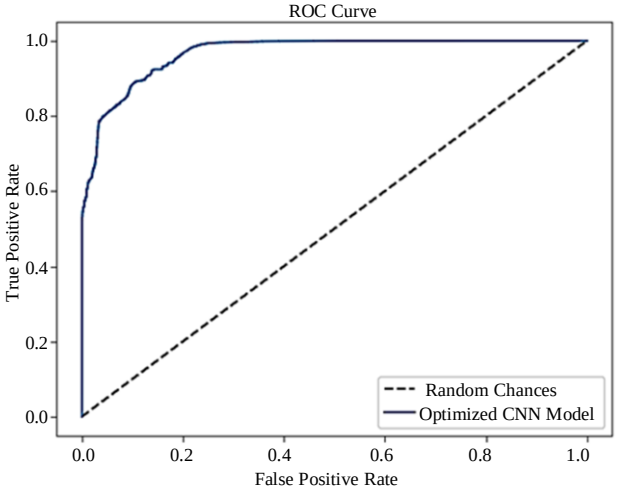


Fig. 19 ROC curve of the model

The proposed CNN model significantly reduces latency by utilizing real-time adaptability and efficient feature extraction. Comparative benchmarks with existing methods demonstrate the model's superior performance, with faster task execution times attributed to its ability to adjust to network conditions and user demands dynamically. For instance, the CNN's capability to prioritize throughput and resource utilization during task offloading results in lower RTT and reduced overall processing time compared to traditional methods like LITOSS and EPTO. The model enhances the user experience by ensuring minimal latency, enabling seamless and timely data processing.

A key strength of the proposed CNN model is its adaptability and generalizability across diverse IoT scenarios. The model's training on a diverse dataset, encompassing features such as latency metrics, user behavior, and network conditions, ensures robust performance in varying

environments. For instance, it can handle low-latency requirements in time-sensitive applications and optimise resource allocation for devices operating under constrained power conditions. The model's capability to generalize to unseen data makes it suitable for deployment in real-world IoT ecosystems, where network characteristics and user demands are highly dynamic. This flexibility highlights the practical relevance of the model in scenarios like smart cities, industrial IoT, and connected healthcare systems, where task offloading efficiency and adaptability are critical.

Dynamic network conditions play a significant role in influencing task offloading decisions. The proposed CNN model effectively addresses these variations by incorporating real-time data into its decision-making processes. For example, the model prioritizes tasks with higher data transfer rates in high-throughput scenarios, optimizing resource utilization. Conversely, in low-latency environments, it allocates tasks to minimize delays, ensuring timely execution. This adaptability enables the model to maintain performance and energy efficiency under varying network conditions, making it highly suitable for dynamic MEC environments.

4.2. Performance Comparison

The performance of traditional learning methods and models based on optimization algorithms is compared with the suggested model. Table 5 shows the accuracy of comparing the proposed model with existing methods.

Table 5. Performance comparison

Methodology	Accuracy (%)
LSTM [7]	89
NB	82
SVM	78
DBN [8]	89.67
LITOSS [5]	81.86
EPTO [6]	89.0
SGO [8]	89.67
Proposed Method	**91.075**

Figure 20 represents the accuracy comparison graph, which shows that the proposed model outperformed other existing methods. The accuracy comparison shows the greater performance of the suggested optimized CNN model, achieving 91.075%, surpassing all the compared methodologies. Traditional ML approaches, such as Naive Bayes (NB) and Support Vector Machine (SVM), achieved 82% and 78% accuracy, respectively, reflecting their limitations in capturing the complex dependencies inherent in task offloading decisions. Advanced deep learning techniques like LSTM (89%) and, DBN and SGO, 89.67% demonstrated improved accuracy due to their capacity to model sequential and feature-rich data.

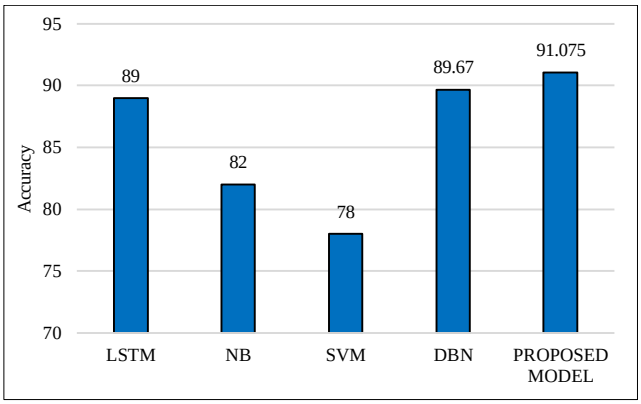


Fig. 20 Accuracy comparison of the proposed method with existing methods

Similarly, heuristic methods such as LITOSS (81.86%) and EPTO (89.0%) performed well in specific scenarios but lacked the adaptability and precision of the proposed CNN. The proposed method's superior performance can be attributed to its optimized architecture, which integrates advanced regularization techniques, batch normalization, and max pooling, enabling it to extract and process features while avoiding overfitting effectively. These results establish the optimized CNN as the most effective approach for energy-aware task offloading in MEC environments, offering robust accuracy and adaptability to dynamic network conditions. Figure 21 demonstrates the performance of the suggested model with existing optimization algorithms.

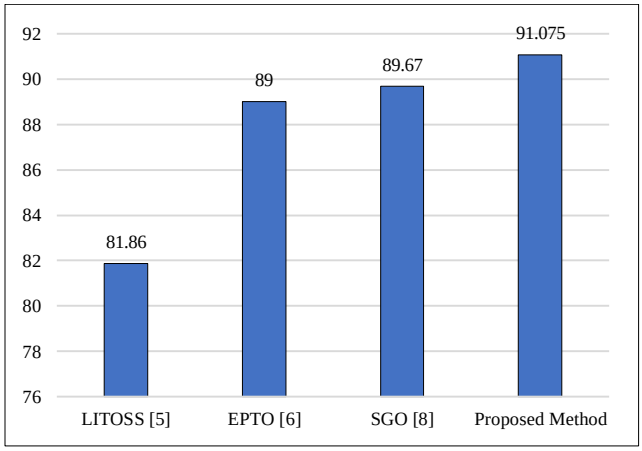


Fig. 21 Accuracy comparison of proposed optimized CNN with other optimization algorithm-based models.

The proposed model achieves the highest accuracy at 91.075%, surpassing SGO with 89.67%, EPTO with 89%, and LITOSS with 81.86%.

5. Conclusion

MEC enhances the network performance and improves user experience by bringing computational capabilities closer to the network edge. Task offloading is crucial for the optimization of performance in the MEC environment. Thus,

an optimized CNN with layer optimization and hyperparameter optimization is developed and analyzed. By utilizing a dataset with different network characteristics, user behavior and resource utilization, the model accurately predicts the optimal offloading decisions, striking a balance between executing tasks locally and offloading them to edge servers.

The proposed model demonstrated robust performance, enhanced with dropout regularization, batch normalization, and bias regularization. The model achieved 91.075% accuracy, 91.82% precision, 91.07% recall, and 90.74% F1 score, indicating a strong balance between precision and recall. These results highlight the effectiveness of the optimized CNN in making intelligent offloading decisions, thereby extending the operational lifetime of IoT devices and ensuring efficient resource allocation.

While the proposed study demonstrates the effectiveness of the optimized CNN model, certain limitations should be noted. Though comprehensive, reliance on a simulated dataset may not fully account for the complexities of diverse IoT ecosystems, such as variations in user behavior, device heterogeneity, and real-time network fluctuations. Additionally, the model's performance under extreme variability in network conditions, such as severe congestion or sudden bandwidth changes, has not been thoroughly investigated. Future work could focus on enhancing the model's ability to handle highly dynamic network parameters and improving its fault tolerance in adverse conditions. Incorporating advanced techniques, such as adaptive regularization methods or hybrid optimization algorithms, could further refine the model's accuracy and robustness. Extending the approach to include optimization under multiple concurrent objectives, such as balancing energy consumption and latency simultaneously, would also be a valuable direction for future research.

Acknowledgments

I want to express my sincere gratitude to all those who contributed to completing this research paper. I extend my heartfelt thanks to my supervisor, family, colleagues and fellow researchers for their encouragement and understanding during the demanding phases of this work.

References

[1] Sahil Garg et al., "Security in IoT-Driven Mobile Edge Computing: New Paradigms, Challenges, and Opportunities," *IEEE Network*, vol. 35, no. 5, pp. 298-305, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[2] Abdenacer Naouri et al., "A Novel Framework for Mobile-Edge Computing by Optimizing Task Offloading," *IEEE Internet of Things Journal*, vol. 8, no. 16, pp. 13065-13076, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[3] Xuming An et al., "Joint Task Offloading and Resource Allocation for IoT Edge Computing with Sequential Task Dependency," *IEEE Internet of Things Journal*, vol. 9, no. 17, pp. 16546-16561, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[4] Zhongjin Li et al., "Energy-Aware Task Offloading with Deadline Constraint in Mobile Edge Computing," *EURASIP Journal on Wireless Communications and Networking*, vol. 2021, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[5] Abdelkarim Ben Sada et al., "Energy-Aware Selective Inference Task Offloading for Real-Time Edge Computing Applications," *IEEE Access*, vol. 12, pp. 72924-72937, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[6] G. Pradeep et al., "Energy Prediction and Task Optimization for Efficient IoT Task Offloading and Management," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 12, no. 1s, pp. 411-427, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[7] Thar Baker et al., "EDITORS: Energy-Aware DynamIc Task Offloading Using Deep Reinforcement Transfer Learning in SDN-Enabled Edge Nodes," *Internet of Things*, vol. 25, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[8] Ilyos Abdullaev et al., "Task Offloading and Resource Allocation in IoT Based Mobile Edge Computing Using Deep Learning," *Computers, Materials & Continua*, vol. 76, no. 2, pp. 1463-1477, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[9] Waleed Almuseelem, "Energy-Efficient and Security-Aware Task Offloading for Multi-Tier Edge-Cloud Computing Systems," *IEEE Access*, vol. 11, pp. 66428-66439, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[10] Bassem Sellami, Akram Hakiri, and Sadok Ben Yahia, "Deep Reinforcement Learning for Energy-Aware Task Offloading in Join SDN-Blockchain 5G Massive IoT Edge Network," *Future Generation Computer Systems*, vol. 137, pp. 363-379, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[11] Bassem Sellami et al., "Energy-Aware Task Scheduling and Offloading Using Deep Reinforcement Learning in SDN-Enabled IoT Network," *Computer Networks*, vol. 210, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[12] Abhijeet Mahapatra et al., "An Energy-Aware Task Offloading and Load Balancing for Latency-Sensitive IoT Applications in the Fog-Cloud Continuum," *IEEE Access*, vol. 12, pp. 14334-14349, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[13] Mingxiong Zhao et al., "Energy-Aware Task Offloading and Resource Allocation for Time-Sensitive Services in Mobile Edge Computing Systems," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 10, pp. 10925-10940, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[14] Jing Bi et al., "Energy-Aware Task Offloading with Genetic Particle Swarm Optimization in Hybrid Edge Computing," *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Melbourne, Australia, pp. 3194-3199, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[15] Feng Wang, Jie Xu, and Shuguang Cui, "Optimal Energy Allocation and Task Offloading Policy for Wireless Powered Mobile Edge Computing Systems," *IEEE Transactions on Wireless Communications*, vol. 19, no. 4, pp. 2443-2459, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[16] Youpeng Tu et al., "Task Offloading Based on LSTM Prediction and Deep Reinforcement Learning for Efficient Edge Computing in IoT," *Future Internet*, vol. 14, no. 2, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[17] Gözde Karataş Baydoğmuş, "The Effects of Normalization and Standardization an Internet of Things Attack Detection," *Avrupa Bilim ve Teknoloji Dergisi*, no. 29, pp. 187-192, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[18] Aleksandr Ometov, Anzhelika Mezina, and Jari Nurmi, "On Applicability of Imagery-Based CNN to Computational Offloading Location Selection," *IEEE Access*, vol. 11, pp. 2433-2444, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[19] Huang Jin Jie, and Putra Wanda, "RunPool: A Dynamic Pooling Layer for Convolution Neural Network," *International Journal of Computational Intelligence Systems*, vol. 13, no. 1, pp. 66-76, 2020. [CrossRef] [Google Scholar] [Publisher Link]