Department of Computer Science and Engineering

(Artificial Intelligence & Machine Learning)

Faculty of Engineering & Technology

Jain Global Campus, Kanakapura Taluk - 562112
Ramanagara District, Karnataka, India

2025
(8th Semester)

A report on

"CIFAR-100 CNN Model"

Submitted in partial fulfillment for the project work

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING
(CYBER SECURITY)

Submitted by
Atuliya (21BTRCL131)

Under the guidance of

S Sivaram
Trainer
PhD. In Cybersecurity

1. Introduction

The objective of this project is to develop a deep learning model for image classification using the CIFAR-100 dataset. This dataset consists of 100 distinct categories, with each class containing 600 images. Due to its complexity, CIFAR-100 presents a challenging classification problem that requires an advanced neural network to effectively learn patterns and classify images accurately. By utilizing a Convolutional Neural Network (CNN), we aim to improve the model's ability to recognize and categorize images efficiently.

2. Dataset Description

The CIFAR-100 dataset contains a total of 60,000 images, each with a resolution of 32x32 pixels and three color channels (RGB). These images are divided into 50,000 training samples and 10,000 test samples. The dataset covers a wide variety of categories, including animals, vehicles, household objects, and plants, making it a highly diverse and complex dataset for classification.

2.1 Data Preprocessing

To enhance model performance and ensure efficient training, several preprocessing steps were applied to the dataset:

Normalization: All pixel values were scaled to a range between 0 and 1 to ensure uniformity and accelerate convergence during training.

One-Hot Encoding: The categorical labels of images were converted into one-hot encoded vectors, enabling multi-class classification.

Data Splitting: The training dataset was further split into training and validation subsets, with 80% of the data used for training and 20% allocated for validation, ensuring the model's generalization ability is properly evaluated.

3. Model Architecture

The Convolutional Neural Network (CNN) architecture implemented in this project consists of multiple layers specifically designed to extract meaningful features from the input images. CNNs are well-suited for image classification tasks as they effectively capture spatial hierarchies in image data.

3.1 Layer Descriptions

Input Layer: Accepts images of shape (32, 32, 3), corresponding to the dataset's image dimensions.

Convolutional Layers: Four convolutional layers were used, each employing a 3x3 filter size. These layers extract various features such as edges, textures, and patterns from the images.

Max Pooling Layers: Applied after each convolutional layer to reduce the spatial dimensions, allowing the model to focus on the most prominent features while reducing computational cost.

Dropout Layers: Implemented to mitigate overfitting by randomly deactivating a fraction of neurons during training.

Flatten Layer: Converts the extracted feature maps into a 1D feature vector, enabling classification in the fully connected layers.

Dense Layers: A fully connected layer with 64 neurons and ReLU activation was added to further process the extracted features.

Output Layer: The final classification layer contains 100 neurons with softmax activation, assigning probabilities to each of the 100 categories.

4. Model Compilation and Training

4.1 Compilation Settings

Before training, the model was compiled using the following settings:

Optimizer: Adam, which is widely used due to its adaptive learning rate and efficient performance.

Loss Function: Categorical Cross-Entropy, which is suitable for multi-class classification problems.

Evaluation Metric: Accuracy, used to measure the model's performance during training and testing.

4.2 Training Process

The model was trained for **50 epochs** with a batch size of **64**, allowing the network to learn complex patterns in the dataset.

A validation split of **20%** was used to assess performance on unseen data.

The training process involved backpropagation and weight adjustments using the Adam optimizer to improve accuracy and minimize loss.

5. Model Performance Evaluation

Upon completion of training, the model was evaluated on the test dataset. The accuracy achieved on the test set was:

```
Accuracy on test data: [test_acc]
```

This accuracy indicates the effectiveness of the model in classifying images from the CIFAR-100 dataset. Further enhancements can be made to improve performance.

6. Image Prediction and Visualization

To validate the model's performance, an image from the test dataset (index 30) was selected for prediction. The model predicted the class using softmax probabilities, and the highest probability class was chosen as the predicted label. The predicted label was then compared with the actual label.

6.1 Prediction Example

A sample test image was processed through the trained model, and the predicted output was compared against the actual label.

Actual Label: Chair

Predicted Label: Chair

To visualize the classification results, Matplotlib was used to display the image along with its actual and predicted labels.

7. Conclusion

The CNN model successfully classified images from the CIFAR-100 dataset with a reasonable level of accuracy. The results indicate that deep learning techniques can effectively be utilized for multi-class image classification problems. However, improvements can be made to achieve higher accuracy and robustness.

OUTPUT :

```
import matplotlib.pyplot as plt
import numpy as np

image_index = 30
sample_image = test_images[image_index].reshape(1, 32, 32, 3)

predictions = model.predict(sample_image)
predicted_label_index = np.argmax(predictions)
predicted_label = class_names_cifar100[predicted_label_index]

plt.figure(figsize=(3, 3))
plt.imshow(test_images[image_index])
plt.xlabel("Actual: " + class_names_cifar100[int(np.argmax(test_labels[image_index]))])
plt.title("Predicted: " + predicted_label)
plt.show()
```

```
1/1 [==============================] - 0s 15ms/step
```