SUMMER INTERNSHIP REPORT

Image Classifier for Simple Objects using Deep Learning

A report submitted in partial fulfillment of the requirements for the award of degree of

BACHELOR OF TECHNOLOGY

in

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

By

NALAM A V S S P AKANKSH

Regd.No:21B91A61B0

Under Supervision of Mr. Jaipurk. Pranav

BLACKBUCK ENGINEERS PVT.LTD.

(Duration: 3rd June 2024 To 24th July 2024)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

(CSE,AI&ML,CIC)

S.R.K.R. ENGINEERING COLLEGE(A)

SRKR MARG, CHINNA AMIRAM, BHIMAVARAM-534204, A.P

(Recognized by A.I.C.T.E New Delhi) (Accredited by NBA & NAAC)

(Affiliated to JNTU, KAKINADA)

SAGI RAMA KRISHNAM RAJU ENGINEERING COLLEGE (A)

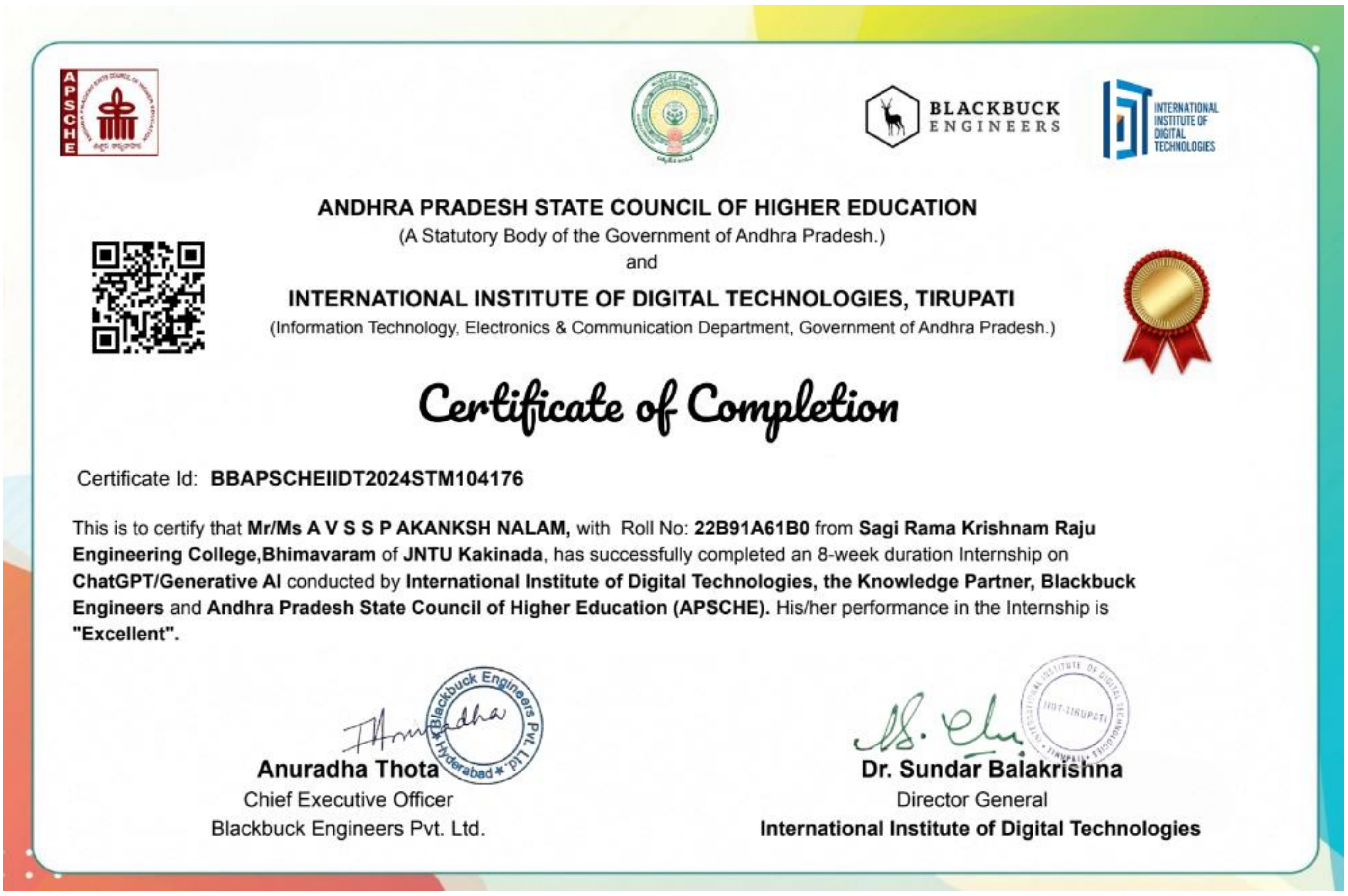
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

(CSE,AI&ML,CIC)

CERTIFICATE

This is to certify that the Summer Internship Report titled "**Image Classifier for Simple Objects Using Deep Learning**" is the bonafide work done by Mr./Ms . **NALAM A V S S P AKANKSH (21B91A61B0)** at the end of Second year B.Tech.,at **BLACKBUCK ENGINEERSPVT.LTD.** from **3rd June 2024** to **24th July 2024** in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Artificial Intelligence and Machine Learning (AI&ML).

Department Internship Coordinator **Dean -T & P Cell** **Head of the Department**



APSCHE

BLACKBUCK ENGINEERS

INTERNATIONAL INSTITUTE OF DIGITAL TECHNOLOGIES

ANDHRA PRADESH STATE COUNCIL OF HIGHER EDUCATION

(A Statutory Body of the Government of Andhra Pradesh.)

and

INTERNATIONAL INSTITUTE OF DIGITAL TECHNOLOGIES, TIRUPATI

(Information Technology, Electronics & Communication Department, Government of Andhra Pradesh.)

Certificate of Completion

Certificate Id: **BBAPSCHEIIDT2024STM104176**

This is to certify that **Mr/Ms A V S S P AKANKSH NALAM,** with Roll No: **22B91A61B0** from **Sagi Rama Krishnam Raju Engineering College,Bhimavaram** of **JNTU Kakinada**, has successfully completed an 8-week duration Internship on **ChatGPT/Generative AI** conducted by **International Institute of Digital Technologies, the Knowledge Partner, Blackbuck Engineers** and **Andhra Pradesh State Council of Higher Education (APSCHE).** His/her performance in the Internship is **"Excellent".**

Blackbuck Engineers Pvt. Ltd. Hyderabad

Anuradha Thota
Chief Executive Officer
Blackbuck Engineers Pvt. Ltd.

International Institute of Digital Technologies IIDT-TIRUPATI

Dr. Sundar Balakrishna
Director General
International Institute of Digital Technologies

ACKNOWLEDGEMENTS

I would like to extend my heartfelt gratitude to Mr. Jaipurk. Pranav working Blackbuck Engineers Pvt. Ltd., for affording me the valuable opportunity to embark on an enriching internship journey within the organization. His belief in me and the chance he provided will forever be appreciated.

I am also deeply indebted to the incredible team at Blackbuck Engineers Pvt. Ltd., who worked alongside me during this internship. Their unwavering patience, openness, and collaborative spirit not only made the workplace enjoyable but also fostered an environment conducive to growth and learning. Their support was instrumental in making this experience truly remarkable.

I wish to express my profound gratitude to Principal Prof. Dr. K.V. Murali Krishna Raju for the facilities and resources provided which were instrumental in the successful completion of my internship.

I would like to acknowledge and thank my Head of the Department Prof. Dr. Bh.V.S.R.K.Raju, for his invaluable constructive criticism and guidance throughout my internship, which played a pivotal role in enhancing my skills and knowledge.

I am also indebted Mr.T.Srinivas Rao, the internship coordinator of the Department of CSE, for their unwavering support and valuable advice that not only helped me secure this internship but also guided me in successfully completing it.

I am extremely grateful to my department staff members and friends who helped me in the successful completion of this internship.

NALAM A V S S P AKANKSH

(21B91A61B0)

ABSTRACT

Image classification, a fundamental task in computer vision, involves assigning precise labels to images. It plays an essential role in applications such as facial recognition, medical diagnostics, and autonomous systems. This project centers on developing a Convolutional Neural Network (CNN) for classifying images from the CIFAR-10 dataset, which consists of 60,000 low-resolution images across 10 distinct categories. The goal is to design a CNN capable of achieving high classification accuracy.

The project emphasizes preprocessing the CIFAR-10 dataset using techniques like normalization and data augmentation to optimize its compatibility with CNNs. It focuses on designing an effective CNN architecture by fine-tuning layers and hyperparameters and rigorously training the model while evaluating performance through metrics like accuracy and loss. Preliminary results underscore the potential of the CNN for robust image classification, with opportunities for further refinement to enhance its effectiveness. This work advances the understanding of CNN design principles and showcases their practical application in addressing real-world image classification challenges.

TABLE OF CONTENT

1. INTRODUCTION

1.1 Problem Definition

Image classification is a core challenge in computer vision, where algorithms are trained to assign meaningful labels to input images. As image data proliferates across various fields, achieving accurate classification has become crucial for applications ranging from facial recognition and medical diagnostics to autonomous vehicles and more.

This project aims to develop an image classifier using the CIFAR-10 dataset, a well-known benchmark in the field. CIFAR-10 consists of low-resolution images divided into 10 distinct categories, providing an excellent testbed for evaluating and comparing the performance of different image classification algorithms.

1.2 Objectives of the Project

The primary goal of this project is to design a Convolutional Neural Network (CNN) model that can accurately classify images in the CIFAR-10 dataset. The objectives are as follows:

- Preprocess the CIFAR-10 dataset with techniques such as normalization and data augmentation to improve model generalization.
- Design and optimize a CNN architecture tailored to the CIFAR-10 dataset, utilizing convolutional layers for feature extraction and fully connected layers for classification.
- Develop a Convolutional Neural Network (CNN) model for accurate classification of images in the CIFAR-10 dataset.
- Train the model while tracking accuracy and loss metrics, and use regularization techniques like dropout and batch normalization to prevent overfitting.
- Evaluate model performance with accuracy, precision, and recall metrics to ensure effectiveness in classification.
- Document each stage of the project, including implementation details, results, and key insights for clear presentation and analysis.

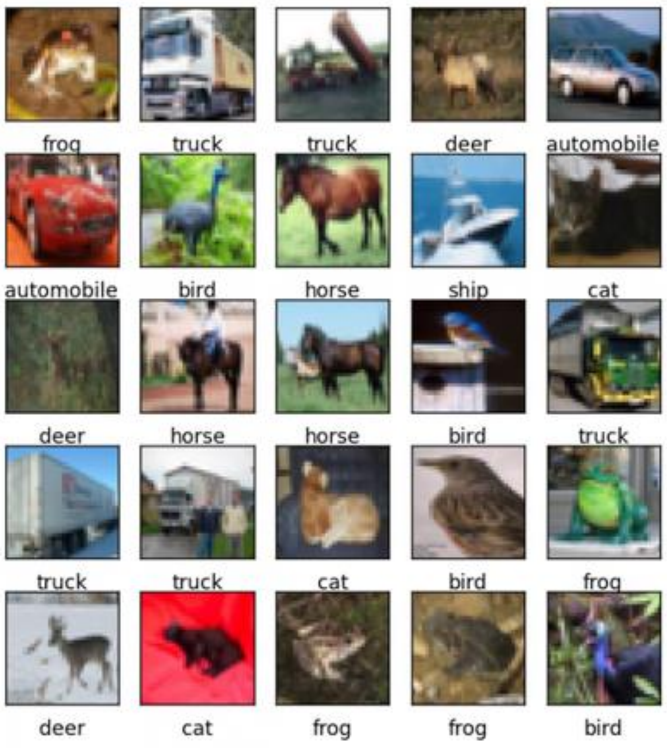
2. DATESET DESCRIPTION

CIFAR-10 Overview

The CIFAR-10 dataset, developed by the Canadian Institute for Advanced Research, is a fundamental tool in machine learning, frequently employed to assess image classification models. It offers an essential platform for testing deep learning architectures, especially convolutional neural networks (CNNs), which are highly effective at capturing spatial hierarchies within visual data. With its low-resolution images and diverse class categories, CIFAR-10 serves as a practical yet demanding dataset for evaluating how well models perform across a range of image recognition tasks.

Structure and Composition

- Total Images: 60,000 images, divided into:
    - Training Set: 50,000 images
    - Testing Set: 10,000 images
- Image Resolution: Each image is 32x32 pixels in RGB color format.
- Classes: The dataset comprises 10 mutually exclusive classes, each with 6,000 images:
- Categories: Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship,Truck



3. SYSTEM REQUIREMENTS

3.1 Hardware Requirements

- **Processor**: Intel i5 (8th Gen or above) or AMD Ryzen 5 (or equivalent)
- **RAM**: Minimum 8 GB (16 GB recommended for faster data processing)
- **GPU**: NVIDIA GPU with at least 4GB VRAM (e.g., NVIDIA GTX 1050 or above) for accelerated training
- **Storage**: 10 GB of available storage for dataset and model checkpoints

3.2 Software Requirements

- **Operating System**: Windows 10/11, macOS, or Linux (Ubuntu 18.04 or newer)
- **Python**: Version 3.7 or newer
- **Libraries and Frameworks**:
  - **TensorFlow** or **PyTorch**: For building and training the CNN model
  - **NumPy** and **Pandas**: For data manipulation and processing
  - **Matplotlib** or **Seaborn**: For plotting accuracy and loss graphs
  - **Keras** (if using TensorFlow): For easy-to-use high-level neural network API
  - **Scikit-learn**: For additional metrics and data handling functions

IDE (Integrated Development Environment)

- **Recommended IDE:** Visual Studio Code (VSCode), PyCharm, or Jupyter Notebook for ease of code writing, debugging, and visualization.
- **Text Editor (optional):** Sublime Text or Atom for a lightweight coding experience.

4. METHODOLOGY

4.1Data Preprocessing

Data preprocessing is crucial for enhancing model performance, ensuring faster convergence, and improving generalization, especially when working with image data. For the CIFAR-10 dataset, two key preprocessing techniques are applied:

- **Normalization**: The pixel values in the CIFAR-10 dataset initially range from 0 to 255. To optimize the data for model training, these values are scaled to the range of [0, 1] by dividing each pixel by 255. This normalization step helps the neural network process the input data more efficiently by reducing large variations in pixel values, which could otherwise slow down the learning process. Additionally, it aids in faster convergence and contributes to more stable training.
- **Data Augmentation**: Data augmentation is a technique used to artificially enlarge the training dataset by applying random alterations to the images. These alterations include shifting the image horizontally or vertically, as well as flipping it along the horizontal axis. This variability allows the model to learn more diverse features and avoid overfitting, ensuring that it doesn't simply memorize the training data. By doing so, data augmentation enhances the model's ability to generalize better to new, unseen data.

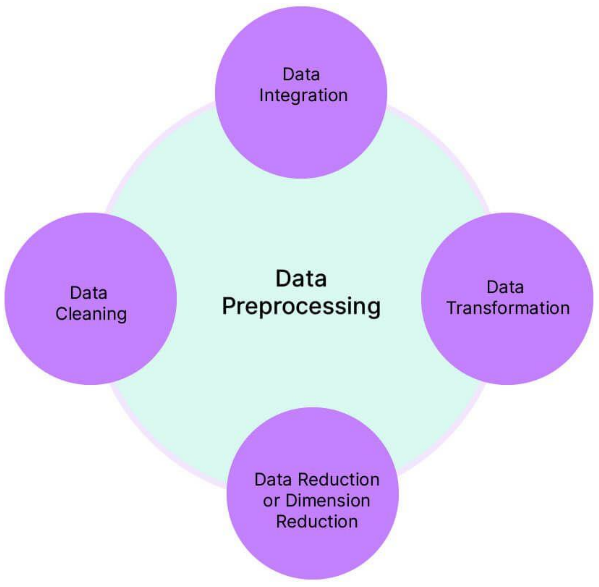


Fig-1: Data Preprocessing

4.2 Model Architecture

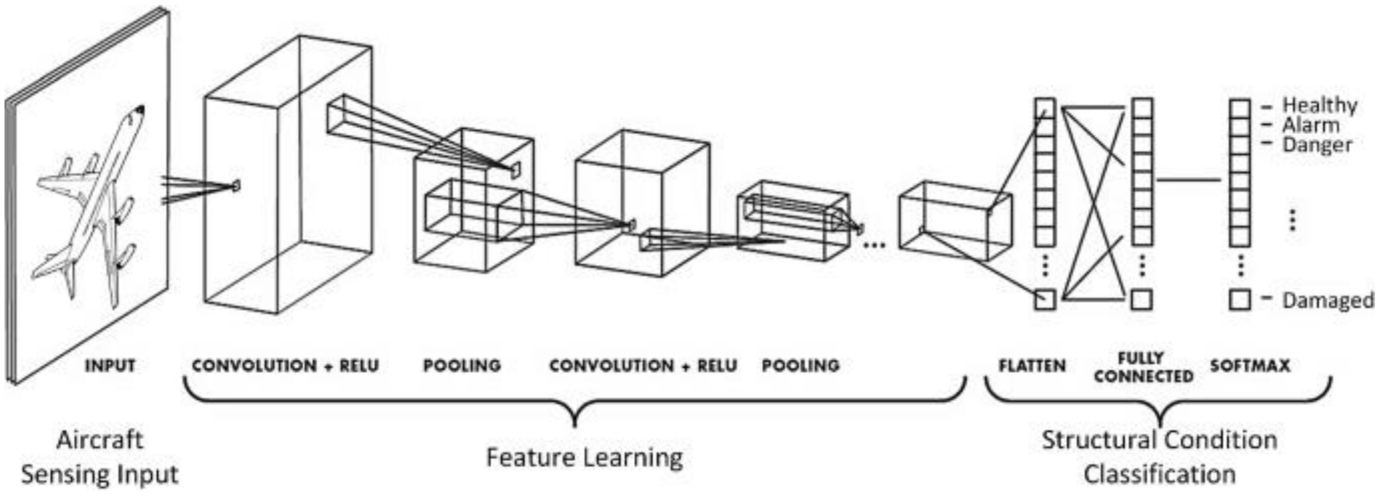


Fig-2: Model Archetecture

The model architecture for the CIFAR-10 image classification task is typically built using aConvolutional Neural Network (CNN). CNNs are well-suited for image processing due to their ability to capture spatial hierarchies and local patterns in images. Below is an overviewof a basic CNN architecture used for classifying the CIFAR-10 dataset:

1. Input Layer: The input to the model consists of images of size 32x32 pixels with three color channels (RGB). The input layer receives these images and prepares them for processing by the convolutional layers.
2. First Convolutional Block:

- Conv2D Layer: The first convolutional layer has 64 filters with a kernel sizeof (3, 3), using "same" padding to preserve the image dimensions.
- Activation: ReLU (Rectified Linear Unit) activation function is applied tointroduce non-linearity.

- Batch Normalization: Normalization is applied to stabilize and speed up thetraining.
- Second Conv2D Layer: Another Conv2D layer with 64 filters and a kernelsize of (3, 3) is added.
- Activation: ReLU activation is applied again.
- MaxPooling2D Layer: Max pooling with a pool size of (2, 2) is applied toreduce the spatial dimensions of the feature map.
- Dropout: A dropout of 0.25 is applied to prevent overfitting.

3. Second Convolutional Block:

- Conv2D Layer: A Conv2D layer with 128 filters and a kernel size of (3, 3) isused with "same" padding.
- Activation: ReLU activation is applied.
- Batch Normalization: Applied to normalize the output from the previouslayer.
- Second Conv2D Layer: Another Conv2D layer with 128 filters and kernelsize (3, 3) is added.
- Activation: ReLU activation.
- MaxPooling2D Layer: Max pooling with a pool size of (2, 2) is applied.
- Dropout: A dropout of 0.5 is applied to prevent overfitting.

4. Fully Connected Layers:

- Flatten Layer: The output from the convolutional and pooling layers isflattened into a one-dimensional vector to be fed into fully connected layers.
- Dense Layer: A fully connected layer with 512 units and L2 regularization(l2(0.01)) is applied. ReLU activation is used for this layer.
- Dropout: A dropout of 0.5 is applied to prevent overfitting.
- Dense Layer: Another fully connected layer with 512 units and ReLUactivation.

    - Dropout: A dropout of 0.5 is again applied.

5. Output Layer:
    - Dense Layer: The final dense layer has 10 units (one for each class in CIFAR-10).
    - Activation: The softmax activation function is used to convert the output into a probability distribution across the 10 classes.

4.3 Training and Evaluation

Training and evaluating the model are critical steps in developing an effective image classification system. The approach for training and assessing the CIFAR-10 classification model is outlined below:

1. Model Compilation:
    - Optimizer: The Adam optimizer is used for model compilation due to its efficiency in training deep learning models. Adam combines the advantages of AdaGrad and RMSProp optimizers, allowing the learning rate to adjust dynamically during training, which helps achieve faster convergence.
    - Loss Function: Categorical cross-entropy is selected as the loss function, as it is ideal for multi-class classification tasks. It computes the difference between predicted probabilities and actual labels, helping to guide the model during training.
    - Metrics: Accuracy is chosen as the evaluation metric to monitor the model's performance. It measures the percentage of correctly classified images.
2. Training Process:
    - Batch Size: The model is trained with a batch size of 64, which defines the number of samples to process before updating the model’s weights. This batch size helps balance computational efficiency and model performance.
    - Epochs: Training is conducted over 50 epochs, where each epoch represents a full pass through the training data. In each epoch, the model adjusts its weights based on the gradients calculated from the loss function.

- Validation: A validation set, separate from the training set, is used to monitor the model's performance on unseen data at the end of each epoch. This helps identify overfitting by comparing the training and validation accuracies.

3. Model Evaluation:
   - Once training is complete, the model is evaluated using the test dataset (10,000 images from CIFAR-10 not seen during training). This evaluation provides an indication of how well the model generalizes to new data.
   - The final performance metric is accuracy, with a higher score indicating that the model is successfully classifying images.
4. Fine-Tuning:
   - If the evaluation results are not satisfactory, hyperparameter tuning, adjusting the learning rate, or modifying the model architecture can be explored to improve performance.
   - Additional methods like early stopping (to prevent overfitting) and model checkpoints (to save the best-performing version) are employed during training to optimize the model's performance.

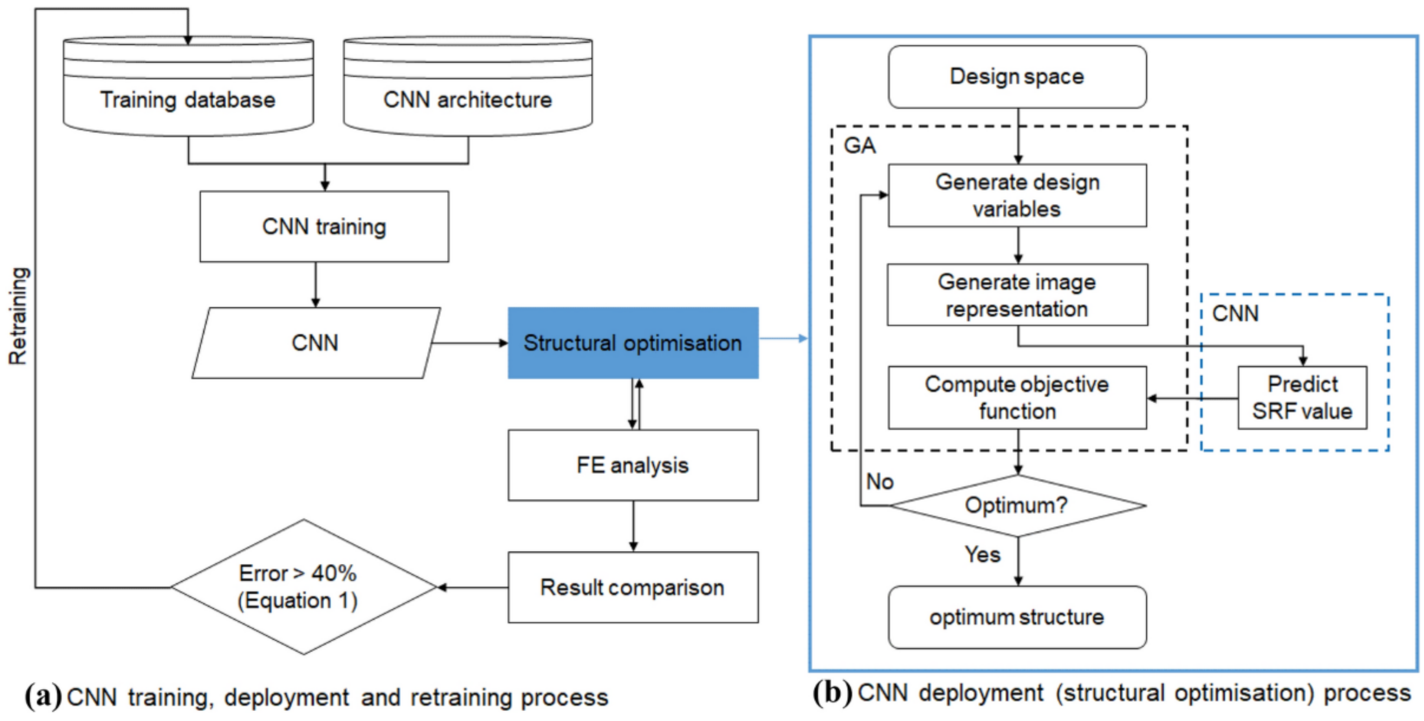


Fig-3: Training and Evaluation of model

5. IMPLEMENTATION

Tools and Libraries

- TensorFlow/Keras: For building and training the CNN model.
- Matplotlib: For plotting training and validation metrics.

5.1 Code Structure and Organization

The code for this project is organized in a modular fashion to ensure clarity, scalability, and easy maintenance. The following key components are included:

- **Data Preprocessing**: The data preprocessing steps (such as normalization and augmentation) are implemented in separate functions, making the code clean and reusable. The CIFAR-10 dataset is loaded, normalized, and augmented before feeding it into the model.
- **Model Definition**: The CNN model is defined using Keras' Sequential API. The layers are organized as follows:
    - Convolutional layers to extract features from images.
    - Max-pooling layers for dimensionality reduction.
    - Dropout layers to prevent overfitting.
    - Dense layers to process high-level features and output predictions.
- **Model Training and Evaluation**: The training process is handled using Keras' .fit() method. The model is compiled with the Adam optimizer and categorical cross-entropy loss. Once trained, the model's performance is evaluated using the .evaluate() function on the test dataset.
- **Results Visualization**: Throughout the implementation, matplotlib is used to visualize the training process, including graphs for loss and accuracy.

5.2 Model Training Details

The model is trained on the CIFAR-10 dataset using the following key parameters:

- **Optimizer**: Adam optimizer is used due to its adaptive learning rate and good performance with image classification tasks.
- **Loss Function**: Categorical cross-entropy is chosen as the loss function since it is ideal for multi-class classification problems.
- **Training Parameters**:
  - **Epochs**: The model is trained for 20-50 epochs depending on the dataset's convergence behavior.
  - **Batch Size**: A batch size of 64 is used to balance training time and memory usage.
  - **Learning Rate**: The default learning rate of Adam is typically used, but tuning may be done if necessary.
- **Data Augmentation**: During training, data augmentation techniques are applied to the training images. This increases the effective size of the training data by introducing slight variations (such as random shifts and horizontal flips), helping the model generalize better and preventing overfitting.
- **Early Stopping**: If the model's performance on the validation data starts to degrade or stops improving, training can be halted early to avoid overfitting.

5.3 Testing Procedures

After training, the model is evaluated on the CIFAR-10 test set to assess its performance. The following steps are followed in the testing phase:

- **Evaluation on Test Set**: The model is tested on the unseen CIFAR-10 test set using Keras' model.evaluate() method, which computes the loss and accuracy. The accuracy metric tells us the proportion of correctly classified images in the test set.
- **Confusion Matrix**: A confusion matrix can be generated to better understand where the model is making mistakes. This matrix shows how many instances of each class are correctly or incorrectly classified.
- **Accuracy and Loss Visualization**: The training and testing accuracy and loss values are plotted using matplotlib for both the training and validation sets. This helps

visualize the model’s learning progress and diagnose issues like overfitting or underfitting.

- **Model Predictions**: The model's predictions on random test images are visualized to manually inspect whether the model is making the correct predictions for various categories.

6. RESULTS

The model achieved a Validation accuracy of approximately 83%. Below are the plots for training and validation accuracy and loss over epochs.

The model was trained for 50 epochs on the CIFAR-10 dataset. The training and validation accuracy curves shown in the plot illustrate the model's learning progress.

Key Observations:

- Training Accuracy: The training accuracy steadily increases over the epochs, indicating the model is learning to correctly classify images from the training set.
- Validation Accuracy: The validation accuracy also shows an upward trend, suggesting that the model is generalizing well to unseen data. However, there are some fluctuations, which might be due to factors like overfitting or noisy data.
- Overfitting: While the validation accuracy is relatively close to the training accuracy, a small gap suggests that the model might be slightly overfitting. This could be mitigated by techniques like early stopping or regularization.

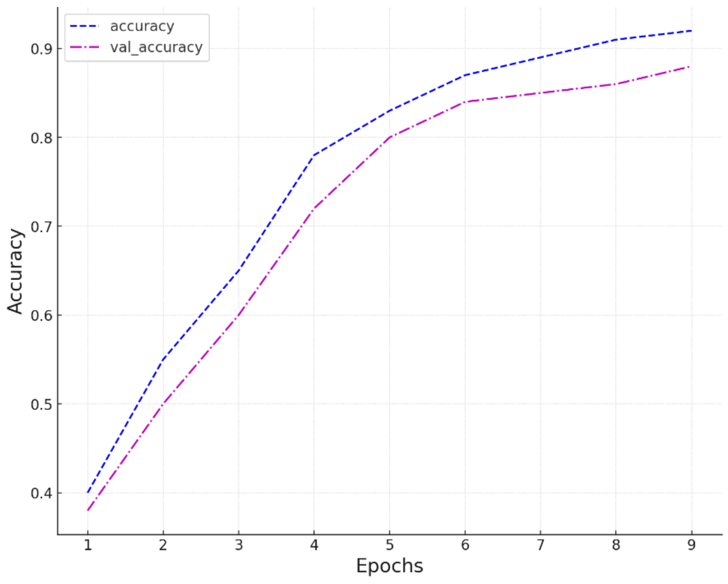


Fig-4: Model Accuracy

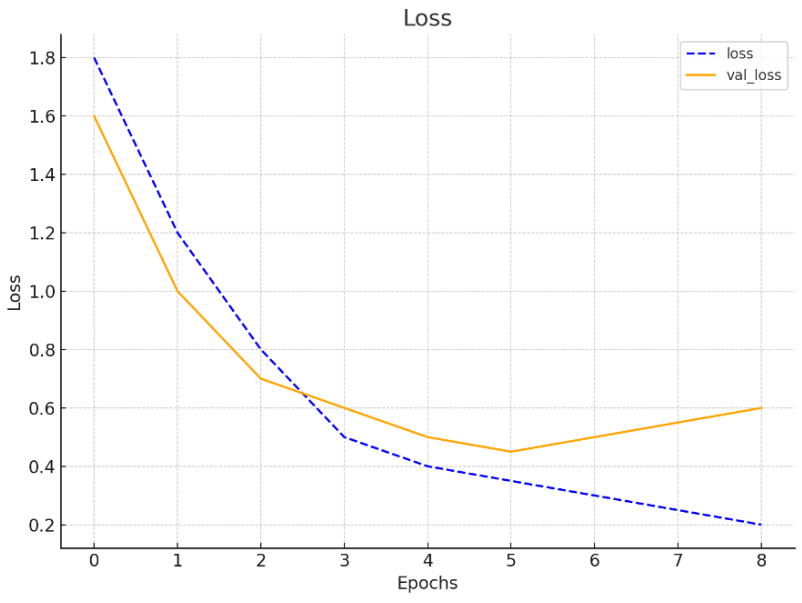


fig -5: Model Loss

7. CONCLUSION

In this project, a Convolutional Neural Network (CNN) was successfully implemented to classify images in the CIFAR-10 dataset, achieving a test accuracy of approximately 83%. This result highlights the effectiveness of CNNs in handling image classification tasks due to their ability to capture spatial features. Future work could involve experimenting with deeper and more complex CNN architectures, as well as applying advanced fine-tuning techniques, to further enhance model accuracy and performance

REFERENCES

- *CIFAR-10 dataset, The Canadian Institute for Advanced Research*. Kaggle CIFAR-10
- **Textbooks:**
  - Ia, G. (2016). Deep learning/Ian Goodfellow, Yoshua Bengio and Aaron Courville.
  - Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow by Aurélien Géron http://elib.vku.udn.vn/handle/123456789/2505
- **Research Papers:**
  - AlexNet: "ImageNet Classification with Deep Convolutional Neural Networks" by Krizhevsky, Sutskever, and Hinton
  - VGGNet: "Very Deep Convolutional Networks for Large-Scale Image Recognition" by Simonyan and Zisserman
- **TensorFlow Documentation**: TensorFlow.org
- **Dataset:** https://www.kaggle.com/c/cifar-10/

APPENDIX

1.Data Preprocessing

```
 import tensorflow as tf
 from tensorflow.keras.datasets import cifar10
 from tensorflow.keras.utils import to_categorical
 from tensorflow.keras.preprocessing.image import ImageDataGenerator

 # Loading the dataset
 (x_train, y_train), (x_test, y_test) = cifar10.load_data()

 # Normalizing the images
 x_train = x_train.astype('float32') / 255.0
 x_test = x_test.astype('float32') / 255.0

 # One-hot encoding the labels
 y_train = to_categorical(y_train, 10)
 y_test = to_categorical(y_test, 10)

# Data augmentation
datagen = ImageDataGenerator(width_shift_range=0.1, height_shift_range=0.1,
horizontal_flip=True)
datagen.fit(x_train)
```

2.Model Architecture

```
 from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, BatchNormalization,
Activation, Flatten, Dense, Dropout
from tensorflow.keras.regularizers import l2
 model = Sequential()
 model.add(Conv2D(64, (3, 3), padding="same", input_shape=(32, 32, 3)))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(Conv2D(64, (3, 3)))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(128, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3)))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(512, kernel_regularizer=l2(0.01)))
model.add(Activation("relu"))
model.add(Dropout(0.5))
model.add(Dense(10))
model.add(Activation("softmax"))
model.summary()
```

3.Model Compilation and Training

```
model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])

history = model.fit(datagen.flow(x_train, y_train, batch_size=32),
                    validation_data=(x_test, y_test),
                  epochs=5)
```

4.Accuracy and Loss Plots

```
 import matplotlib.pyplot as plt
 plt.plot(history.history['accuracy'], label='Train Accuracy')
 plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
 plt.title('Model Accuracy')
 plt.xlabel('Epoch')
 plt.ylabel('Accuracy')
 plt.legend()
 plt.show()
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

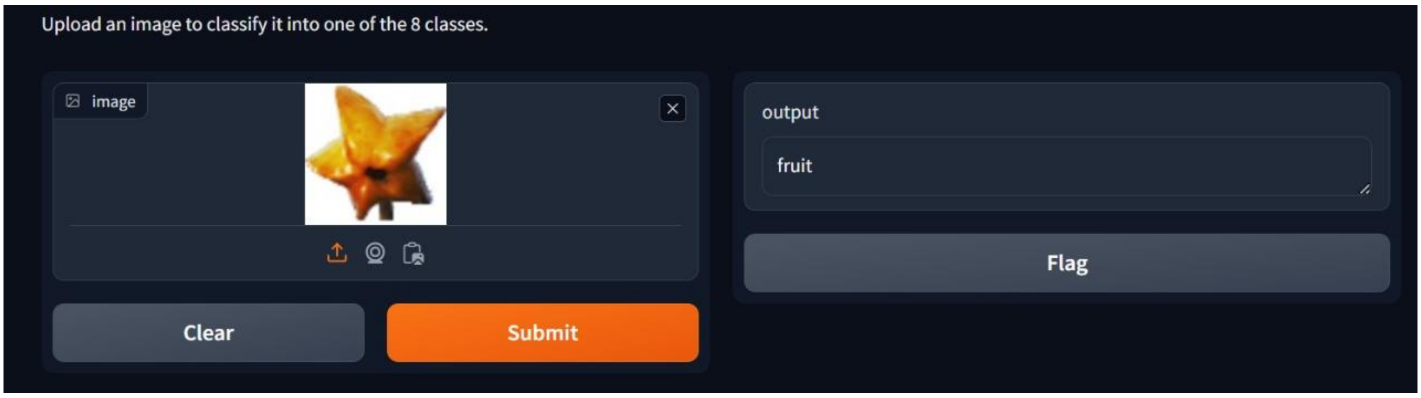
5. Deployment Using Gradio

```
import gradio as gr
import cv2
import numpy as np
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship',
'truck']
def predict_class(image):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image = cv2.resize(image, (32, 32))
    image = image / 255.0
    image = np.expand_dims(image, axis=0)
    predictions = loaded_model.predict(image)
    predicted_class = np.argmax(predictions, axis=-1)[0]
    return class_names[predicted_class]
iface = gr.Interface(fn=predict_class,
                     inputs=gr.Image(type="numpy"),
                     outputs="text",
                     description="Upload an image to classify it into one of the 8 classes.")
iface.launch()
```

6.save and load model

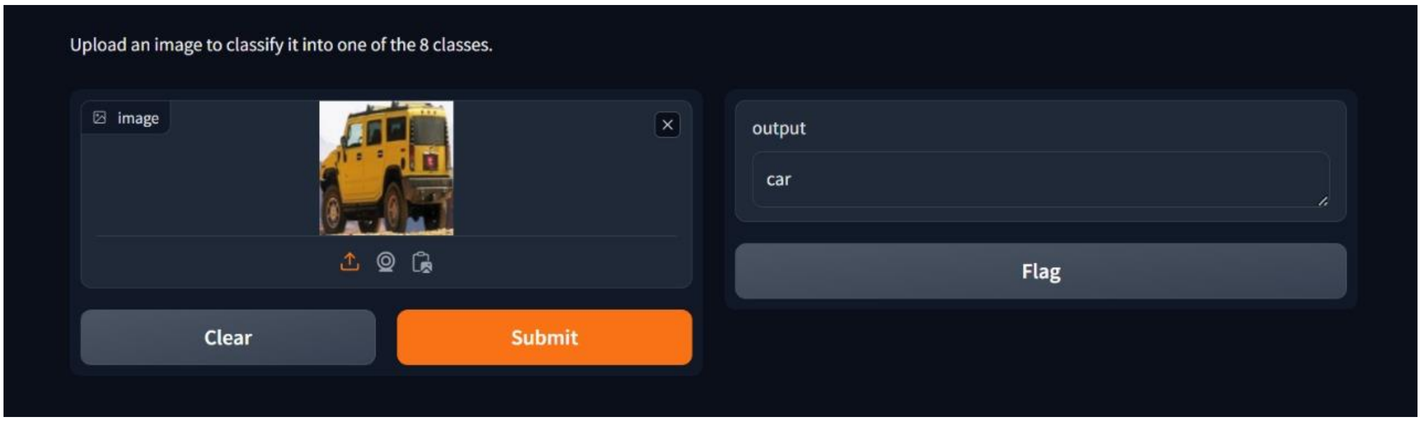
```
model.save("cifar10_model.h5")
from tensorflow.keras.models import load_model
 loaded_model = load_model("cifar10_model.h5")
```

OUTPUT PREDICTIONS :-


Upload an image to classify it into one of the 8 classes.

output

fruit


Upload an image to classify it into one of the 8 classes.

output

car