ARTIFICIAL INTELLIGENCE & EDGE COMPUTING

Project Report On

DEVELOP AN CNN MODEL WHICH CAN DETECT AND RECOGNIZE THE OBJECT IN THE IMAGE (OBJECT RECOGNITION IN IMAGES)

Submitted By

GROUP-1

Sl.No	Name Of the Students	Registration No.
1.	Ashutosh Das	2141004162
2.	Anandini Tripathy	2141020043
3.	Bishnupriya Nayak	2141014140
4.	Sumit kumar Purohit	2141016376

B. Tech. (ECE) 7th Semester (Section–B)

DEPT. OF ELECTRONICS & COMMUNICATION ENGINEERING
Institute of Technical Education and Research

SIKSHA 'O' ANUSANDHAN
DEEMED TO BE UNIVERSITY

Bhubaneswar, Odisha, India.

(September 2024)

CONTENTS

DECLARATION

We certify that

a. The work contained in this report is original and has been done by us.

b. The work has not been submitted to any other Institute for any degree or diploma.

c. We have followed the guidelines provided by the Department in preparing the report.

d. Whenever we have used materials (data, theoretical analysis, figures, and text) from other sources, we have given due credit to them by citing them in the text of the report and giving their details in the reference.

Sl.No	Name Of the Students	Registration No.
1.	Ashutosh Das	2141004162
2.	Anandini Tripathy	2141020043
3.	Bishnupriya Nayak	2141014140
4.	Sumit kumar Purohit	2141016376

DATE: 30/09/2024
SIKHSHA 'O' ANUSANDHAN, ITER

1. INTRODUCTION:-

Artificial Intelligence and edge computing are transforming the way data is processed and utilized, especially in environments where real-time decision-making is crucial. Here are some key points highlighting their significance:

Fig. 1 **Artificial Intelligence and edge computing**

- **Real-Time Analytics:**
  In sectors like finance and healthcare, edge computing enables real-time data processing and analysis, allowing organizations to make immediate decisions and enhance service delivery.

- **Enhanced Privacy and Security:**
  Keeping sensitive data on local devices rather than transmitting it to the cloud reduces the risk of data breaches. AI can also enhance security protocols at the edge, identifying anomalies and threats in real time.

- **Reduced Latency:**
  Edge computing processes data closer to the source, minimizing latency. This is crucial for AI applications that require real-time decision-making, such as autonomous vehicles and industrial automation.

- **Scalability and Flexibility:**
  AI models can be deployed and updated at the edge, allowing for scalable solutions that can adapt to changing conditions without relying solely on centralized systems.

- **Support for IoT Applications:**
  The combination of AI and edge computing is vital for the Internet of Things (IoT), enabling smart devices to make intelligent decisions locally, improving efficiency in sectors like healthcare, smart cities, and manufacturing.

- **Innovation and Automation:**
  Edge AI enables innovative applications and automation in various sectors, from smart manufacturing to personalized retail experiences. It allows devices to make intelligent decisions locally, enhancing operational efficiency and user experience.
  AI and edge computing are revolutionizing global technology by enabling real-time data processingThese advancements are crucial for applications like autonomous vehicles, smart cities, and industrial automation, driving innovation and improving operational efficiency across various sectors.

2. AIM OF THE PROJECT:-

Develop a CNN Model to recognize the object in image using CIFAR-10 data-set and check the accuracy, sensitivity, F1 Score .

3. PROBLEM STATEMENT & SOLUTIONS:-

Object recognition in images involves identifying and locating objects within images or video frames, a crucial task in computer vision with applications in self-driving cars, security systems, and image search engines. Using the CIFAR-10 dataset, which contains 60,000 32x32 color images across 10 classes1, you can develop a Convolutional Neural Network (CNN) model. This model will leverage deep learning techniques to accurately detect and recognize objects within the images, enhancing the automation and accuracy of object recognition tasks.

Here are four key points to develop a solution for object recognition using the CIFAR-10 dataset:

1. **Data Preprocessing:** Load and preprocess the CIFAR-10 dataset. Normalize the pixel values to a range of 0 to 1 and perform data augmentation techniques like rotation, flipping, and cropping to enhance the model's robustness.

2. **Model Architecture:** Design a Convolutional Neural Network (CNN) with multiple convolutional layers, pooling layers, and fully connected layers. Use activation functions like ReLU and apply dropout to prevent overfitting.

3. **Training the Model**: Compile the model using an appropriate optimizer (e.g., Adam) and loss function (e.g., categorical cross-entropy). Train the model on the training set, validating it on the validation set, and adjust hyperparameters as needed to improve performance.

4. **Evaluation and Testing:** Evaluate the model's performance on the test set using metrics such as accuracy, precision, recall, and F1-score. Fine-tune the model based on the evaluation results and test it on new, unseen data to ensure generalization.

The proposed solution for object recognition using the CIFAR-10 dataset offers several advantages. By leveraging a Convolutional Neural Network (CNN), the model can efficiently learn and recognize complex patterns in images, leading to high accuracy in object detection. Data augmentation and dropout techniques enhance the model's robustness and prevent overfitting, ensuring it performs well on new, unseen data. Additionally, the structured approach to training, validation, and evaluation allows for continuous improvement and fine-tuning, making the system adaptable to various real-world applications such as self-driving cars and security systems.

4. SOFTWARE REQUIRED:-

Jupyter Notebooks provide an interactive and versatile environment for coding, data analysis, and visualization. They support multiple programming languages and allow for the integration of rich media, making it easy to create comprehensive reports. Notebooks are also great for collaboration and sharing, as they can be accessed from any device with a web browser. This combination of features enhances documentation, reproducibility, and overall productivity in various projects.

Fig.2

Fig.3

To develop an object recognition system using the CIFAR-10 dataset, you'll need the following software:

- **Python**: A versatile programming language, essential for implementing machine learning models.
- **TensorFlow and Keras:** Popular deep learning libraries for building and training Convolutional Neural Networks (CNNs).
- **NumPy:** A fundamental package for numerical computations in Python
- **Matplotlib:** A plotting library for visualizing data and model performance.
- **Jupyter Notebook:** An interactive environment for writing and running code, useful for experimentation and documentation.
- **OpenCV:** A library for image processing tasks, which can be helpful for data augmentation and preprocessing.
- **Additional Convolutional and Pooling Layers:** Stacking more Conv2D and MaxPooling2D layers to build a deeper network that can learn more complex features.
- **Flatten Layer:** Converts the 2D feature maps into a 1D vector to feed into the fully connected layers.
- **Activation Function:** ReLU for hidden layers and Softmax for the output layer to get probability distributions over classes.

These tools will provide a comprehensive environment for developing, training, and evaluating your object recognition model.

5. MODELLING & EVALUATION:-

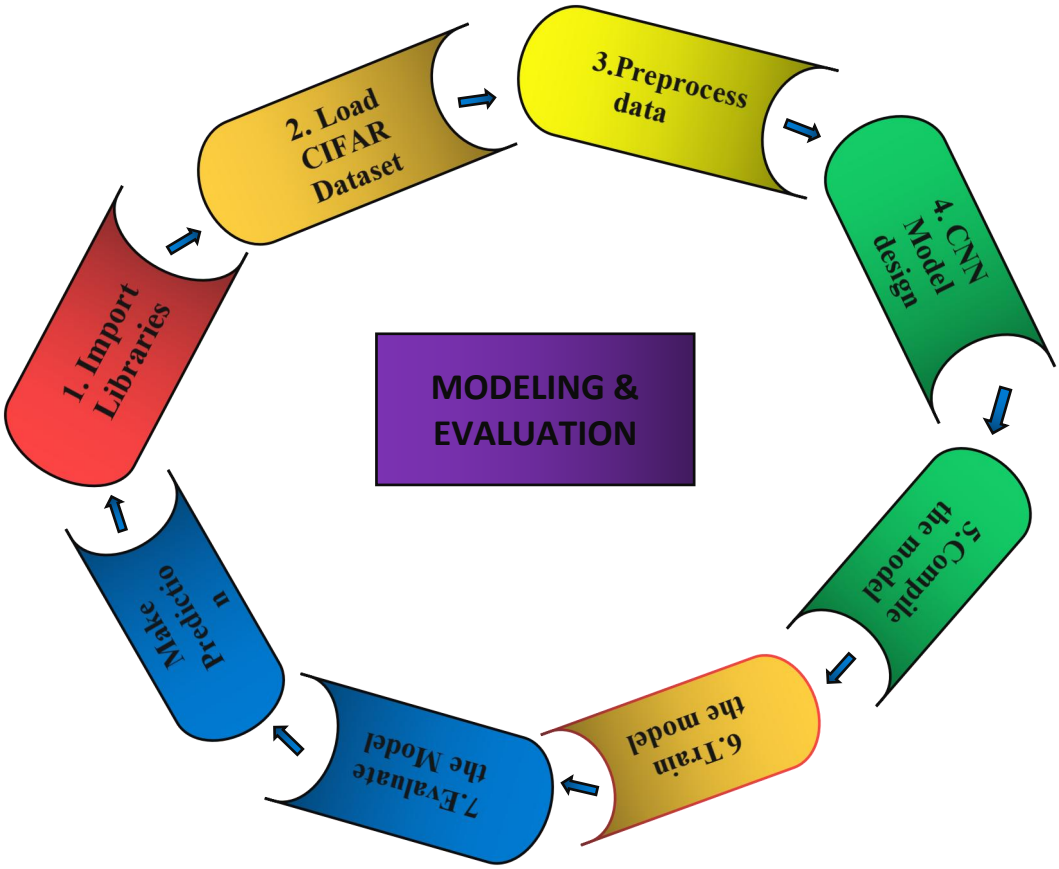


Fig.4. Modeling Process Of designing CNN model for image recognition

5.1. Significant Using CNN:-

Convolutional Neural Networks (CNNs) are highly significant in image recognition due to their ability to automatically learn and extract hierarchical features from images. This capability allows CNNs to identify patterns and objects within images with high accuracy, making them ideal for tasks such as image classification, object detection, and facial recognition.

The convolutional layers in CNNs use shared weights, which reduces the number of parameters and computational complexity, enhancing efficiency and scalability2. Additionally, CNNs are translation invariant, meaning they can recognize objects regardless of their position within the image2. This makes CNNs a powerful tool in various applications, from autonomous vehicles to medical imaging and security systems.

5. PROJECT ARCHITECTURE:-

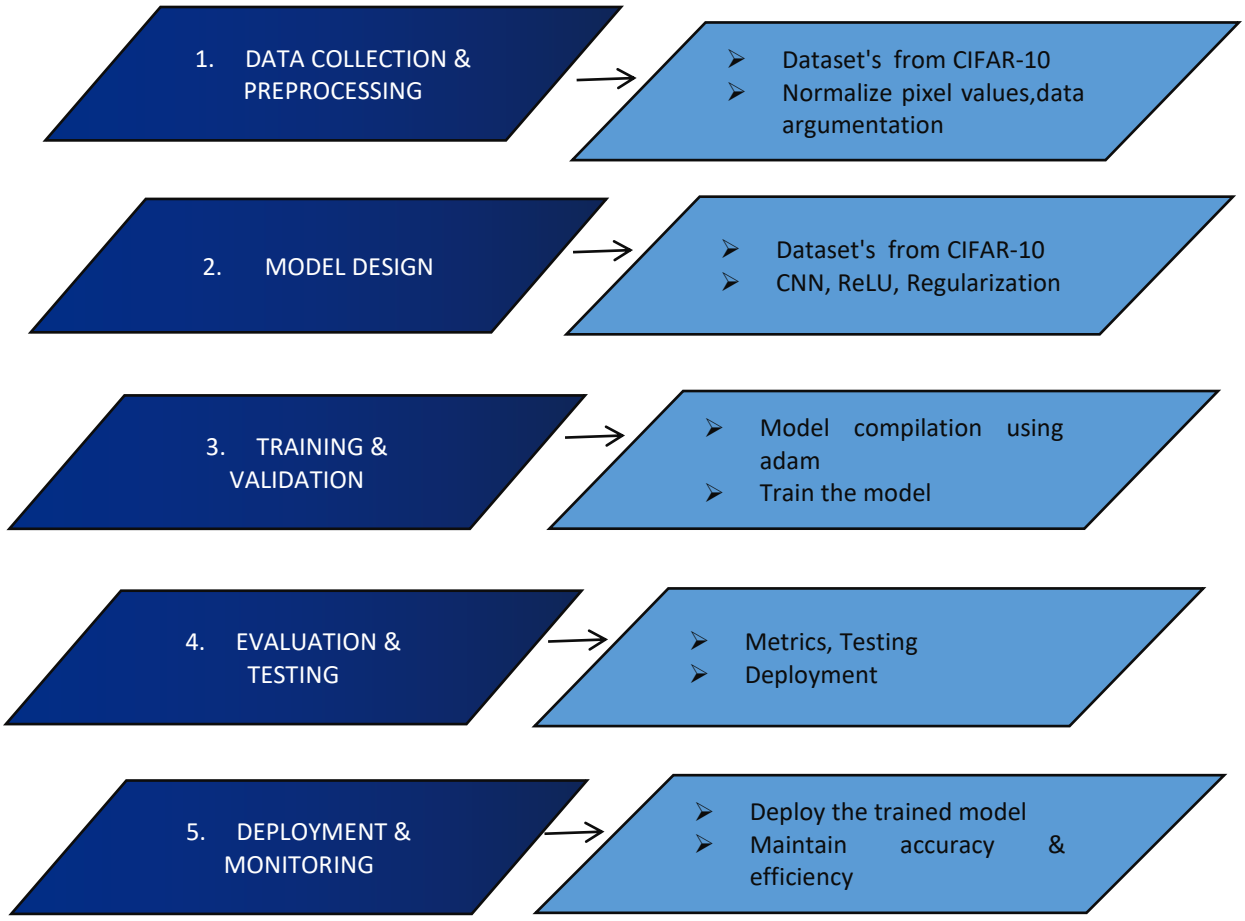


Figure 5. Project Architecture of Image recognition

The project architecture for developing an object recognition system using the CIFAR-10 dataset involves several key steps. First, preprocess the data by normalizing pixel values and applying data augmentation, then split it into training, validation, and test sets. Design a Convolutional Neural Network (CNN) with layers like convolutional, pooling, and fully connected layers, using ReLU activation and dropout for regularization. Train the model with an optimizer like Adam and a loss function such as categorical cross-entropy, validating and adjusting hyperparameters as needed. Evaluate the model using metrics like accuracy and F1-score, fine-tune based on test results, and prepare for deployment. Finally, deploy the model and continuously monitor its performance to ensure it remains effective.

Tips for improvement:

1. Data augmentation
2. Transfer learning
3. Hyperparameter tuning
4. Batch normalization
5. Regularization techniques

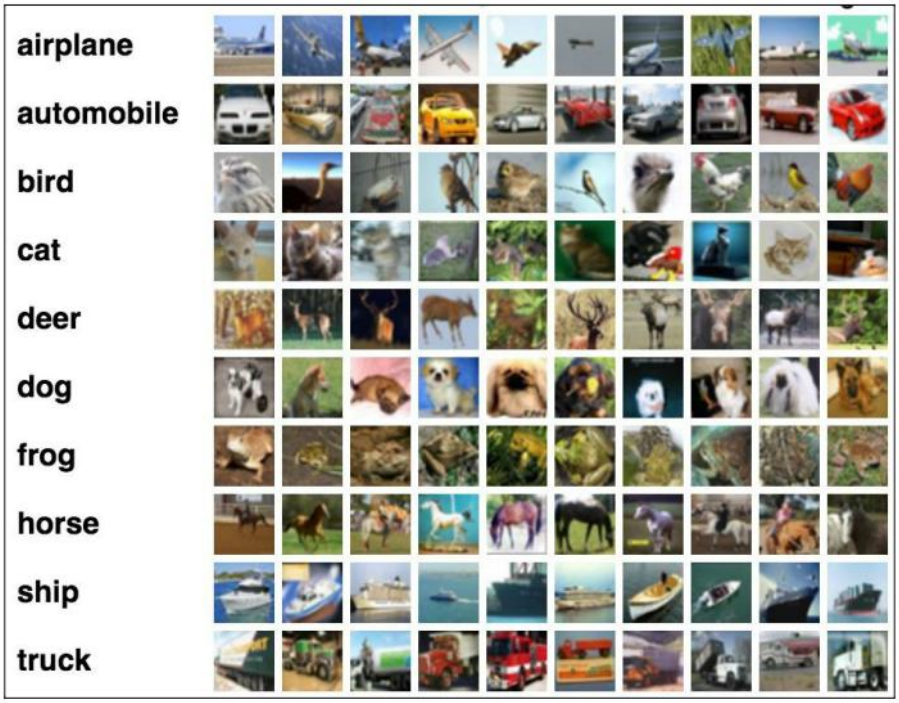
6. Model Justification:-

Convolutional Neural Networks (CNNs) are highly effective for image recognition tasks due to their ability to automatically learn and extract hierarchical features from images. This makes them particularly suitable for complex visual tasks such as object detection, facial recognition, and medical image analysis. CNNs leverage convolutional layers to capture spatial hierarchies in images, pooling layers to reduce dimensionality, and fully connected layers for classification. This architecture allows CNNs to achieve high accuracy and robustness in image recognition tasks.

ModelCode:-

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
from tensorflow.keras import layers
from __future__ import print_function
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D

from keras.datasets import cifar10
from keras.utils import to_categorical
```




```
# Load the dataset
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
print('x_train shape:', X_train.shape)
print('y_train shape:', y_train.shape)
```

```
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# Normalize the data
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0
```

Output:-

```
x_train shape: (50000, 32, 32, 3)
y_train shape: (50000, 1)
50000 train samples
10000 test samples
```

```
// CNN Model
 from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, Input
# Define the CNN model
    model = Sequential()

# Input layer
    model.add(Input(shape=(32, 32, 3)))
# Convolutional layers
  model.add(Conv2D(32, (3, 3), activation='relu'))
  model.add(MaxPooling2D((2, 2)))
  model.add(Conv2D(64, (3, 3), activation='relu'))
  model.add(MaxPooling2D((2, 2)))
  model.add(Conv2D(128, (3, 3), activation='relu'))
  model.add(MaxPooling2D((2, 2)))
# Fully connected layers
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(10, activation='softmax'))
# Compile the model
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# Train the model
history = model.fit(X_train, y_train, epochs=10, batch_size=64, validation_data=(X_test,
y_test))
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np

# Evaluate the model on the test set
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)

# Generate the classification report
report = classification_report(y_true, y_pred_classes, target_names=[
    'airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'])
print(report)
```

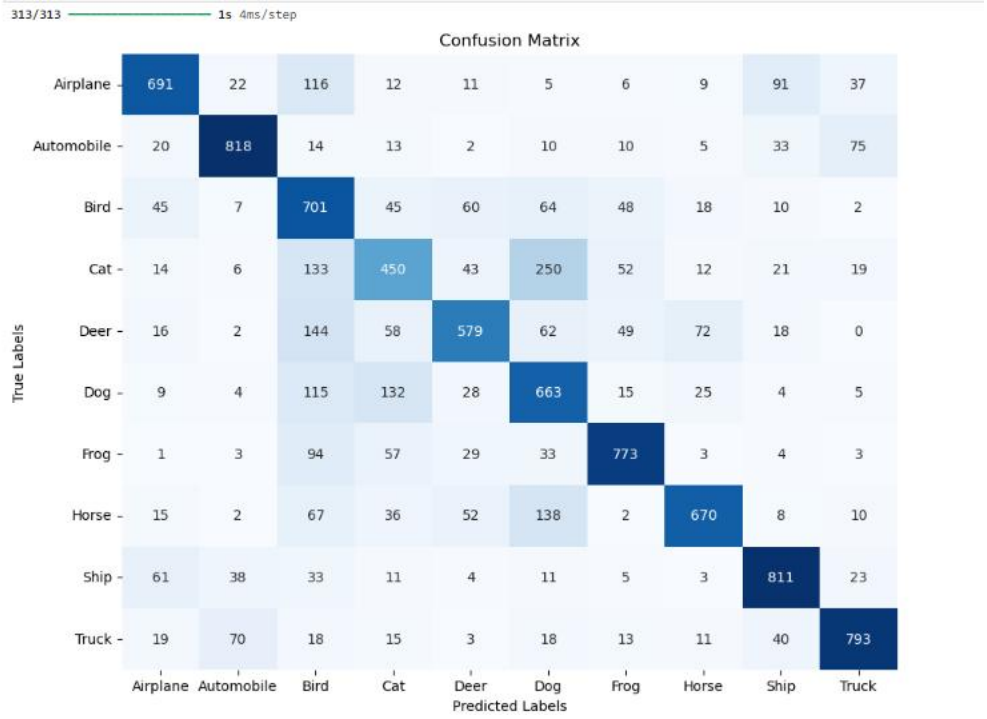
```
# Generate the confusion matrix
cm = confusion_matrix(y_true, y_pred_classes)
print(cm)
```

OUTPUT:-

313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ **2s** 5ms/step

	precision	recall	f1-score	support
airplane	0.78	0.69	0.73	1000
automobile	0.84	0.82	0.83	1000
bird	0.49	0.70	0.58	1000
cat	0.54	0.45	0.49	1000
deer	0.71	0.58	0.64	1000
dog	0.53	0.66	0.59	1000
frog	0.79	0.77	0.78	1000
horse	0.81	0.67	0.73	1000
ship	0.78	0.81	0.80	1000
truck	0.82	0.79	0.81	1000
accuracy			0.69	10000
macro avg	0.71	0.69	0.70	10000
weighted avg	0.71	0.69	0.70	10000

```
[[691  22 116  12  11   5   6   9  91  37]
 [ 20 818  14  13   2  10  10   5  33  75]
 [ 45   7 701  45  60  64  48  18  10   2]
 [ 14   6 133 450  43 250  52  12  21  19]
 [ 16   2 144  58 579  62  49  72  18   0]
 [  9   4 115 132  28 663  15  25   4   5]
 [  1   3  94  57  29  33 773   3   4   3]
 [ 15   2  67  36  52 138   2 670   8  10]
 [ 61  38  33  11   4  11   5   3 811  23]
 [ 19  70  18  15   3  18  13  11  40 793]]
```



```
 # Extract metrics from classification report
 from sklearn.metrics import precision_recall_fscore_support

precision, recall, f1, _ = precision_recall_fscore_support(y_true, y_pred_classes, average='weighted')
accuracy = np.sum(y_pred_classes == y_true) / len(y_true)
# For specificity, we calculate TN for each class
tn = cm.diagonal()
fp = cm.sum(axis=0) - tn
fn = cm.sum(axis=1) - tn
specificity = tn / (tn + fp)

print(f'Accuracy: {accuracy:.4f}')
print(f'Precision: {precision:.4f}')
print(f'Recall: {recall:.4f}')
print(f'F1 Score: {f1:.4f}')
print(f'Specificity: {specificity}')
```

OUTPUT:-

Accuracy: 0.7549
Precision: 0.7095
Recall: 0.6949
F1 Score: 0.6974
Specificity: [0.77553311 0.84156379 0.48850174 0.54282268 0.71393342 0.52870813 0.79445015 0.80917874 0.77980769 0.82006205]

7. Metrices:-

The performance metrics of your image recognition model indicate a solid overall performance with an accuracy of 75.49%, meaning the model correctly classifies approximately three-quarters of the instances. The precision of 70.95% suggests that when the model predicts a positive class, it is correct about 71% of the time. The recall (or sensitivity) of 69.49% indicates that the model successfully identifies about 69% of the actual positive instances. The F1 score, which balances precision and recall, is 69.74%, reflecting a reasonable trade-off between these two metrics. The specificity values, which measure the model's ability to correctly identify negative instances, vary across different classes, with some classes achieving high specificity (e.g., 84.16%) and others lower (e.g., 48.85%). This variability suggests that while the model performs well overall, there may be room for improvement in distinguishing certain classes more accurately.

Accuracy: 0.7549
Precision: 0.7095
Recall: 0.6949 F1
Score: 0.6974
Specificity: [0.77553311 0.84156379 0.48850174 0.54282268 0.71393342 0.52870813 0.79445015 0.80917874 0.77980769 0.82006205]

8. Scope for Enhancement:-

To enhance the recognition capabilities of your image recognition model, you can implement several advanced strategies. Begin with data augmentation to artificially expand your training dataset by applying transformations such as rotations, flips, and color adjustments, which help the model generalize better. Experiment with more sophisticated CNN architectures like ResNet, DenseNet, or EfficientNet, which are designed to capture intricate patterns and improve accuracy. Utilize transfer learning by fine-tuning pre-trained models on your specific dataset, leveraging the knowledge these models have gained from large-scale datasets. Optimize hyperparameters through techniques like grid search or random search to find the best configuration for your model. Incorporate regularization methods such as dropout, L2 regularization, and batch normalization to prevent overfitting and enhance generalization. Consider ensemble methods, where you combine predictions from multiple models to achieve better performance than individual models. Additionally, gathering more labeled data and focusing on improving the model's performance on classes with lower specificity can significantly boost overall accuracy. These enhancements can collectively lead to a more robust and accurate image recognition system.

9. Project Github Link:-

https://github.com/bishnupriya140/ITER_21-25-PS-6-Image-Recognization-using-CNN-model.git