**Assignment 3 – Image Classification on CIFAR-10 using CNN**

Name: Shubham Gautam Dahane

Roll No: 382015

PRN: 22310312

**Problem Statement**

**Implement a Convolutional Neural Network (CNN) in Python using Keras and TensorFlow to classify images from the CIFAR-10 dataset into 10 categories.**

**Objectives**

* To understand CNN architecture for image classification.
* To preprocess the CIFAR-10 dataset (normalization + one-hot encoding).
* To implement and train a CNN model with multiple convolution and pooling layers.
* To evaluate model performance using test accuracy.
* To visualize accuracy and loss curves over epochs.
* To test the model on random images from the test set.

**Requirements**

* Operating System: Windows/Linux/MacOS
* Python Version: 3.x
* Tools: Jupyter Notebook / Anaconda / Google Colab
* Hardware: CPU (GPU recommended for faster CNN training)
* Libraries Used:
  + TensorFlow, Keras
  + NumPy
  + Matplotlib
  + Random

**Theory**

The CIFAR-10 dataset consists of 60,000 images (32×32 pixels, RGB) divided into 10 categories: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

A Convolutional Neural Network (CNN) is highly effective for image classification tasks.

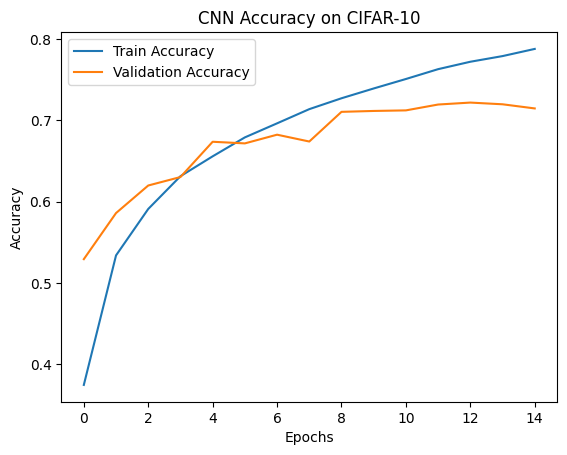
* Convolutional Layers (Conv2D): Extract spatial features from images.
* Pooling Layers (MaxPooling2D): Reduce dimensionality while retaining important features.
* Fully Connected Layers (Dense): Learn higher-level representations.
* Dropout: Prevents overfitting by randomly deactivating neurons.
* Softmax: Produces probability distribution across 10 classes.

**Methodology**

1. **Dataset Acquisition**
   * Load CIFAR-10 dataset using Keras.
   * Dataset contains 50,000 training and 10,000 test images.
2. **Data Preparation**
   * Normalize pixel values (0–255 → 0–1).
   * Convert labels to one-hot encoded format.
3. **Model Architecture**
   * Conv2D (32 filters, ReLU) + MaxPooling2D
   * Conv2D (64 filters, ReLU) + MaxPooling2D
   * Conv2D (128 filters, ReLU) + MaxPooling2D
   * Flatten → Dense(128, ReLU) → Dropout(0.5)
   * Output: Dense(10, Softmax) for 10-class classification.
4. **Model Compilation**
   * Optimizer: Adam
   * Loss Function: Categorical Crossentropy
   * Metric: Accuracy
5. **Model Training**
   * Train for 15 epochs with batch size = 64.
   * Validation on test set.
6. **Model Evaluation**
   * Report accuracy on test data.
7. **Model Testing**
   * Select random test images.
   * Predict class using trained model.
   * Display both true label and predicted label.

**Graphs and Visualizations**

1. **Accuracy vs. Epochs**
   * Training vs. validation accuracy plotted to show model learning progress.



1. **Loss vs. Epochs**
   * Training vs. validation loss plotted to analyze convergence.

A graph of loss of a train loss

AI-generated content may be incorrect.

1. **Sample Predictions**
   * Random test images displayed with true vs. predicted class labels.

A blurry image of a deer

AI-generated content may be incorrect.

**Advantages**

* CNNs automatically extract image features, reducing manual feature engineering.
* Works effectively for large-scale image classification tasks.
* Achieves high accuracy with relatively shallow networks.
* Can be extended to larger datasets like CIFAR-100 or ImageNet.

**Limitations**

* Training CNNs on image datasets is computationally expensive.
* Requires large datasets for best generalization.
* Sensitive to hyperparameter tuning (epochs, learning rate, filters).
* Risk of overfitting without proper regularization (Dropout, Data Augmentation).

**Applications**

* Computer Vision (object recognition, detection).
* Autonomous Vehicles (road sign / object classification).
* Healthcare (disease detection in medical images).
* Security Systems (image-based verification).
* Robotics (environment understanding).

**Working / Algorithm**

Step 1: Import required libraries (TensorFlow, NumPy, Matplotlib).  
Step 2: Load CIFAR-10 dataset.  
Step 3: Normalize image pixel values and one-hot encode labels.  
Step 4: Build CNN architecture with multiple Conv2D, Pooling, Dense, Dropout layers.  
Step 5: Compile model with Adam optimizer and categorical crossentropy loss.  
Step 6: Train model for 15 epochs with validation.  
Step 7: Evaluate model accuracy on test set.  
Step 8: Plot accuracy and loss graphs.  
Step 9: Predict random test images and compare with true labels.

**Conclusion**

The CNN model successfully classified images from the CIFAR-10 dataset into 10 categories. The model achieved high accuracy on the test set, demonstrating the effectiveness of CNNs in image recognition tasks.

Visualizations of accuracy, loss, and random predictions confirmed the reliability of the trained model. With further tuning (e.g., data augmentation, deeper architectures), the model can achieve even higher performance, making it suitable for real-world computer vision applications.