CASE STUDY

CiFAR-10

Deep Neural Network

Author: Sameer Hate
Date: 18-05-2025

Table of Contents

Introduction

General

In the era of rapidly advancing technology, image classification has become one of the most prominent and practical applications of deep learning. From autonomous vehicles and facial recognition systems to medical imaging and security, the ability of machines to correctly interpret visual information is critical. To explore the fundamentals of this field, the CIFAR-10 dataset serves as a widely recognized benchmark for evaluating image classification models.

This project focuses on designing, training, and evaluating a deep neural network to classify the CIFAR-10 images accurately. The objective is not only to achieve high classification accuracy but also to understand how different layers, activation functions, and optimization techniques contribute to the model's performance. The outcomes of this project provide valuable insights into the practical application of deep learning for image recognition tasks.

Author's Note

This case study presented in this work has been independently completed by me. This project serves as a medium to deepen my understanding of Data Science concepts and enhance my practical skills in applying machine learning and data analysis techniques to real-world problems.It may be prone to errors but I try my best to eliminate them as much as I can. In case you identify any such error or have a more optimal technique to solve this problem, your feedback and suggestions are always welcome as this will eventually help me strengthen my concepts.

Problem Statement & Expected Results

Objective

The objective of this project is to develop and evaluate a Deep Neural Network model capable of accurately classifying images from the CIFAR-10 dataset into one of ten predefined categories.

Goals

- To understand and implement a Deep Neural Network (DNN) for image classification using the CIFAR-10 dataset.
- To preprocess and normalize image data for optimal training efficiency.
- To design an effective CNN architecture tailored to CIFAR-10's complexity.
- To evaluate the model's performance using metrics such as accuracy, loss, and confusion matrix.
- To gain hands-on experience in applying deep learning to real-world image classification problems.

Expected Outcomes

The expected outcome of this project is the successful development and training of a Deep Neural Network model that can accurately classify images from the CIFAR-10 dataset into their respective categories. The model should demonstrate high accuracy on both training and test data, with minimal overfitting.

Dataset

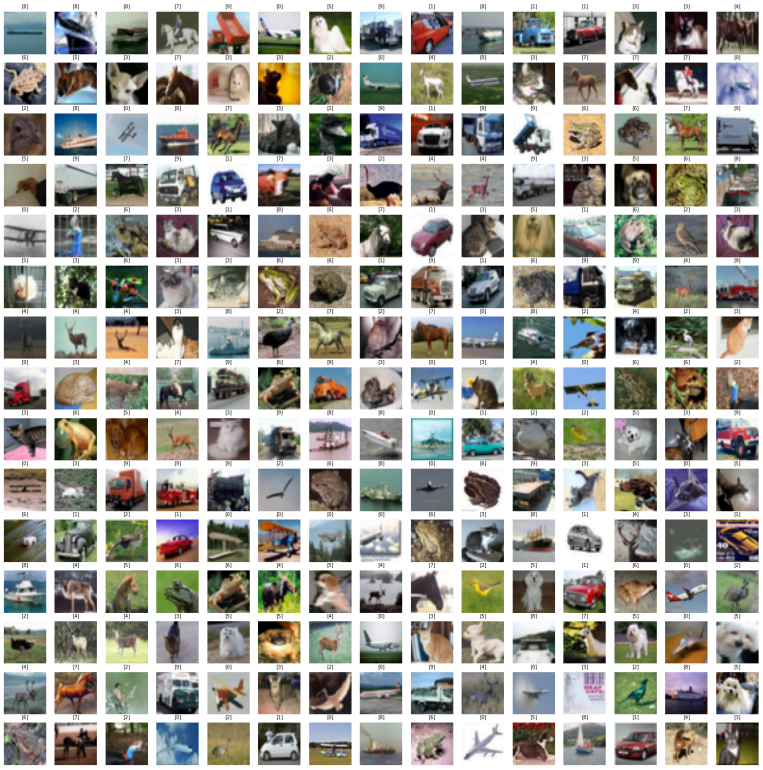
Description

The CIFAR-10 (Canadian Institute for Advanced Research) dataset is a widely used benchmark in the field of computer vision and deep learning. It consists of 60,000 color images that are 32×32 pixels in size, divided into 10 distinct classes with 6,000 images per class.The dataset is evenly split into 50,000 training images and 10,000 test images, enabling both model training and unbiased evaluation.

The ten classes represent everyday object categories:Airplane, Cars, Birds, Cats, Deers, Dogs, Frogs, Horses, Ships, and Trucks.

Each image is labeled with only one class, and the dataset is balanced, meaning each class has an equal number of images.All images are in RGB format, and due to their small size and variability in shape, background, and texture, the CIFAR-10 dataset presents a moderate level of challenge for image classification tasks. Its diversity and compactness make it ideal for developing and testing Convolutional Neural Networks (CNNs) and other deep learning architectures.

Dataset: You can load the dataset directly by using the cifar10.load_data() function from the keras.datasets module. This function automatically downloads and prepares the dataset for use in training and testing a deep learning model.(explained in the approach)



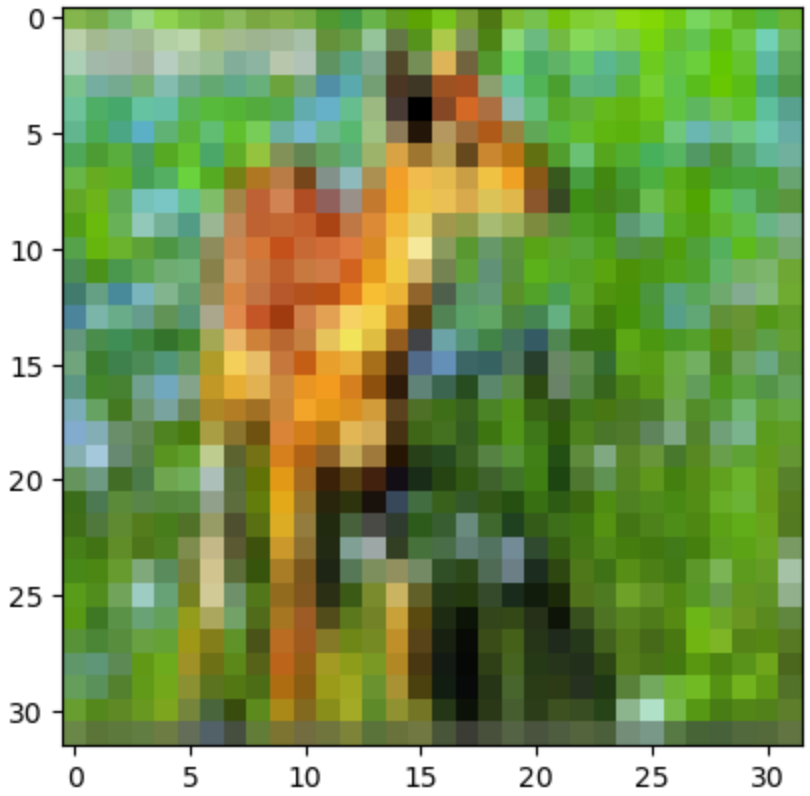
Approach

- #IMPORTING LIBRARIES : In this step, all the essential Python libraries and modules required for data handling, visualization, model building, and evaluation are imported. These imports set up the necessary environment for conducting the deep learning-based image classification task using the CIFAR-10 dataset.
    - pandas and numpy: Fundamental libraries for data manipulation and numerical operations.
    - matplotlib.pyplot and seaborn: Are imported to create informative plots and heatmaps
    - The keras library is imported for building and training deep learning models, and specifically, the cifar10 dataset is loaded from keras.datasets.
    - The confusion_matrix function from sklearn.metrics is imported to evaluate the classification model's performance by comparing predicted and actual labels.

- #IMPORTING & LOADING THE DATASET : This step involves importing and loading the CIFAR-10 dataset using the cifar10.load_data() function from the Keras library. The dataset is automatically downloaded if not already available locally and is then split into training and testing subsets. The variables X_train and y_train contain the 50,000 training images and their corresponding labels, while X_test and y_test contain the 10,000 test images and labels used for evaluating the model's performance on unseen data. Each image is a 32x32x32 pixel RGB image stored as a 3-D Numpy array.

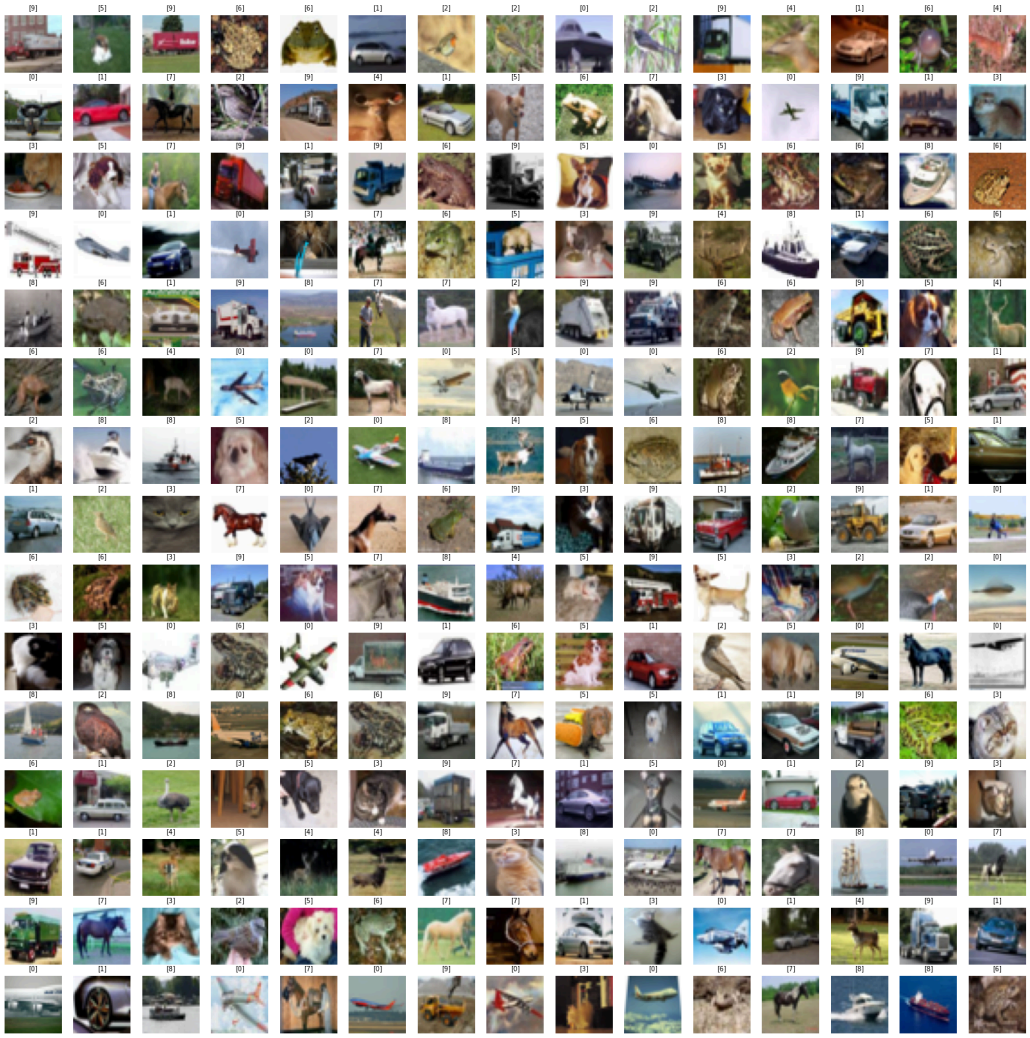
- #PRINTING THE SHAPE : This step is used to inspect the dimensions of the CIFAR-10 dataset after it has been loaded. By printing the shapes of the training and testing sets, the code confirms the structure and size of the input data.

- #VISUALIZING THE DATASET TO CONFIRM IMAGE AND LABELS : This code cell is used to visually inspect a sample image from the CIFAR-10 training dataset and verify that the corresponding label matches the image content. By selecting an index (in this case, i = 1005), the code displays the image using matplotlib.pyplot.imshow() and prints its associated label from y_train.



Label [4] - Deer

- #PRINTING A MATRIX OF IMAGES AND LABELS AT RANDOM : This code cell is designed to provide a visual overview of the training dataset by displaying a grid of randomly selected images along with their corresponding labels. A 15×15 grid (225 images total) is created using matplotlib, offering a comprehensive snapshot of the dataset's diversity.
    - W_grid and L_grid define the grid's width and length, respectively.
    - plt.subplots() creates a figure with 225 subplots, each sized appropriately for image display.
    - Inside the loop, a random index is generated for each subplot using np.random.randint(), and the image at that index is displayed using imshow().
    - The label is shown as the subplot title with a small font size (fontsize=7), and axes are turned off for a cleaner view.



- #PREPARING THE DATA BY CONVERTING THE DATA TYPE : In the first part, the image data is originally stored as 8-bit unsigned integers (uint8), with pixel values ranging from 0 to 255. Converting the data type to float32 allows for more precise computations during training, especially when normalization and gradient calculations are involved. In the latter part, the class labels, which are initially integers from 0 to 9, are converted into binary class matrices using one-hot encoding. This format is required for multiclass classification problems using neural networks with a softmax output layer. Each label is transformed into a 10-element binary vector, where the index corresponding to the class is set to 1 and all others to 0.

- #NORMALIZING THE DATA : This stage performs the normalization of the data. Each pixel in the CIFAR-10 images originally has a value between 0 and 255 (since they are 8-bit RGB images). By dividing all pixel values by 255, the values are scaled to a range between 0 and 1.

- #RESHAPING THE DATA : Here the input shape of the training images is extracted and stored in a variable for later use in defining the neural network architecture. The resulting Input_shape variable is typically used as the input_shape parameter when defining the input layer of a Convolutional Neural Network (CNN) using frameworks like Keras.
    - X_train.shape returns a 4-tuple: (num_samples, height, width, channels).
    - X_train.shape[1:] slices off the first element (number of samples) and stores only the shape of a single input image, which is (32, 32, 3) for CIFAR-10.

- #BUILDING THE CNN : This Convolutional Neural Network (CNN) model is designed for image classification on the CIFAR-10 dataset and is built using a sequential architecture consisting of six convolutional layers grouped into three blocks. Each block contains two convolutional layers with increasing filter sizes (32, 64, and 128), each followed by batch normalization to stabilize and accelerate training. Max pooling layers are used after each block to downsample the feature maps, and dropout layers with increasing rates are applied to prevent overfitting. After the convolutional layers, the model includes a flattening layer followed by a fully connected dense layer with 512 neurons and ReLU activation, along with a dropout layer for further regularization. The final output layer uses softmax activation to classify input images into one of ten CIFAR-10 categories.

- #COMPILING THE CNN : This code compiles the Convolutional Neural Network (CNN) model using Keras' compile() method, which prepares the model for training by specifying the loss function, optimizer, and evaluation metric.This compilation step is essential before training, as it defines how the model will learn and how performance will be measured.
    - Loss Function – categorical_crossentropy: Used for multi-class classification problems where the target labels are one-hot encoded (as in CIFAR-10). This function measures the dissimilarity between the predicted probability distribution and the true distribution.
    - Optimizer – RMSprop:RMSprop (Root Mean Square Propagation) is an adaptive learning rate optimization algorithm that is effective for deep neural networks. It adjusts the learning rate dynamically during training and helps with faster convergence. The learning rate is set to 0.001, which is a commonly used default.
    - Metrics – accuracy: The model will track classification accuracy during training and evaluation, which indicates the proportion of correctly predicted labels.

- #TRAINING THE CNN : This code compiles the Convolutional Neural Network (CNN) model by specifying the loss function, optimizer, and evaluation metric to be used during training. The loss function 'categorical_crossentropy' is appropriate for multi-class classification problems like CIFAR-10, where each input image belongs to one of ten distinct classes. The optimizer chosen is Adam, which is an adaptive learning rate optimization algorithm that combines the benefits of RMSProp and momentum. A custom learning rate of 0.0005 is specified to allow more stable and gradual updates to the model weights, potentially leading to better convergence. The metric 'accuracy' is used to monitor the proportion of correctly predicted labels during training and evaluation, providing an intuitive measure of model performance. This compilation step finalizes the model configuration before it is trained.

- #LISTING THE KEYS OF THE OUTPUT HISTORY : This code cell is used to examine the metrics recorded during the training of the Convolutional Neural Network (CNN) model. When the model is trained using the fit() function in Keras, it returns a History object that stores various metrics for each training epoch. By accessing output.history.keys(), the code lists all the metric names that were tracked and recorded during training, such as 'loss', 'accuracy', 'val_loss', and 'val_accuracy'. These keys represent the training and validation performance of the model over time. Identifying the available metrics is useful for evaluating the model's learning behavior and for plotting visualizations of loss and accuracy to assess model performance and detect potential overfitting.

- #EVALUATING THE CNN : The given code evaluates the performance of a trained Convolutional Neural Network (CNN) on a test dataset using the evaluate() function. This function takes the test features (X_test) and their corresponding true labels (y_test) as input and returns a list of evaluation metrics, typically including the loss and accuracy. The second element of this list (evaluation[1]) represents the test accuracy, which indicates the proportion of test samples correctly classified by the model. By printing this value, the code provides a straightforward way to assess how well the CNN has generalized to unseen data. A high test accuracy suggests that the model has learned meaningful patterns from the training data and is performing well on the test set.

- #PLOTTING LOSS VALUES : This code cell is used to visualize the model's training and validation loss over each epoch, providing insight into the learning process of the CNN. Using the output.history dictionary returned from the model's training, the code plots 'loss' to represent the training loss and 'val_loss' to represent the validation loss. These loss values indicate how well the model is performing in terms of minimizing error on both the training and unseen validation data. The plt.plot() functions create line graphs for each loss type, and plt.legend() distinguishes them with labels. The plot is titled

"Model Loss" and includes labeled axes for epochs and loss values. This visualization is crucial for diagnosing problems like overfitting (where validation loss increases while training loss decreases) or underfitting (when both losses remain high), and helps determine whether the model is learning effectively.

- #MAKING PREDICTIONS : The code initiates the prediction process by using the trained CNN model to generate output probabilities for the test dataset X_test. This is done using the cnn.predict(X_test) function, which returns y_prob, a 2D array where each row corresponds to a test sample and contains the predicted probability for each class. Since most CNN models for classification output softmax probabilities, the next step is to determine the most likely class for each test sample. This is achieved using np.argmax(y_prob, axis=1), which selects the index of the highest probability in each row, effectively converting the probabilities into discrete predicted class labels. The result is stored in predicted_classes, a 1D array of integer class labels representing the CNN's final predictions for the test set.

- #RE-CONVERTING TEST LABELS TO ORIGINAL FORM : In many classification problems, especially when using neural networks, the target labels are often converted into one-hot encoded format before training. This means each label is represented as a binary vector where only the index corresponding to the actual class is set to 1, and all others are 0. For example, if there are 3 classes, the label 2 would be encoded as [0, 0, 1]. After training and prediction, it's often necessary to convert these one-hot encoded labels back to their original form for comparison with predicted labels. The line y_test_labels = np.argmax(y_test, axis=1) does exactly this by finding the index of the 1 in each row of the y_test array, thereby reconstructing the original class labels as integers. This is useful for evaluating model performance using metrics like accuracy, confusion matrices, or classification reports.

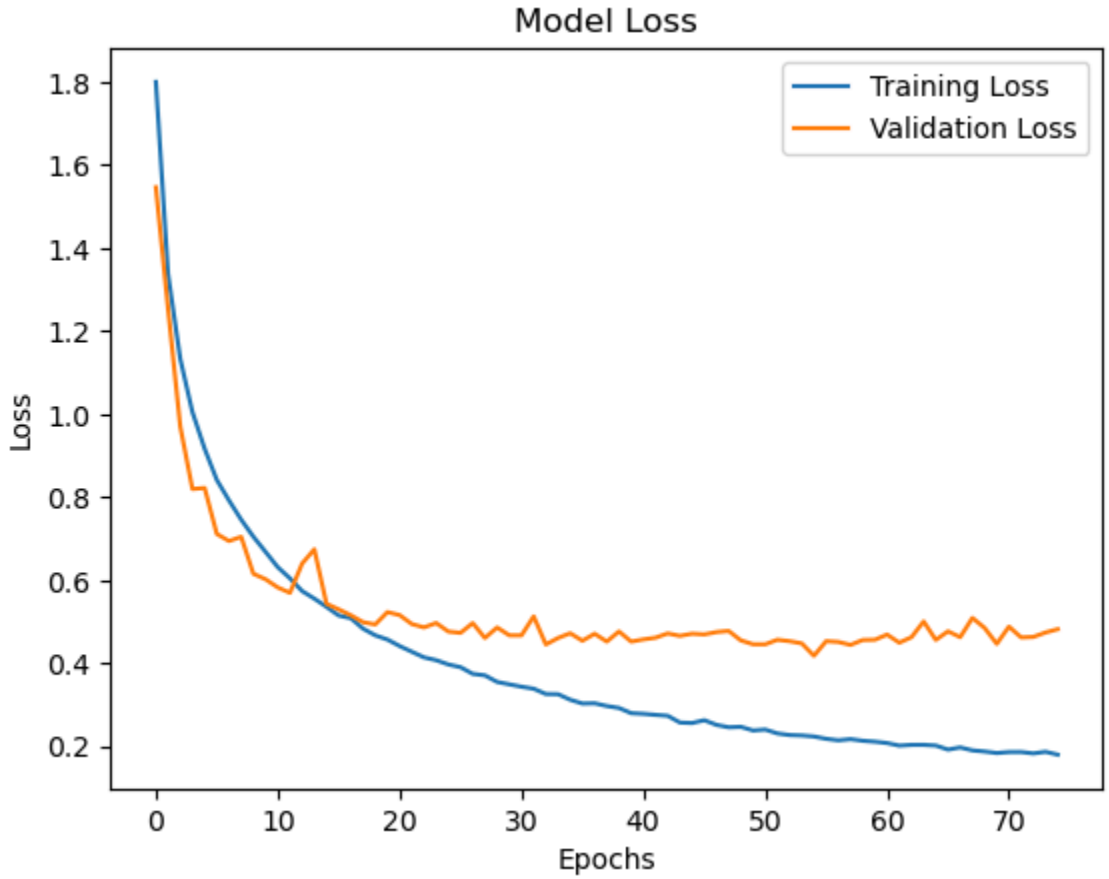
- #PLOTTING PREDICTIONS TO TRUE VALUES : The given code creates a visual grid of test images to compare the predicted class labels with the true labels. It defines the grid size using L = 7 and W = 7, resulting in a 7x7 grid of 49 images. The plt.subplots(L, W, figsize=(20, 15)) command sets up the plotting space, and axes.ravel() flattens the 2D array of subplot axes into a 1D array to allow easy iteration. For each of the first 49 test images, the code uses imshow() to display the image and sets the title of each subplot to show both the model's prediction and the true label, allowing for quick visual comparison. Finally, axes[i].axis('off') removes the axis ticks for a cleaner look, and plt.subplots_adjust(wspace=1) ensures there's adequate space between the images. This type of plot is extremely useful in identifying patterns in model errors and understanding where the model might be confusing certain classes.

- #CHECKING AND PLOTTING THE CONFUSION MATRIX : This code cell is used to evaluate the performance of the classification model by computing and visualizing the confusion matrix. The confusion matrix is generated using the confusion_matrix() function from sklearn.metrics, which takes the true class labels (y_test_labels) and the predicted class labels (predicted_classes) as input. The resulting matrix is a square grid where each row represents the actual class and each column represents the predicted class, allowing for detailed analysis of correct and incorrect predictions across all categories.
  To visualize the confusion matrix, a heatmap is created using Seaborn's heatmap() function. The heatmap displays the count of predictions for each class pair, with annotations enabled (annot=True) to show the exact numbers inside each cell. The figure size is set to (10, 10) to ensure readability, especially when dealing with a large number of classes (such as CIFAR-10 or traffic sign classification). This visualization is essential for identifying patterns of misclassification, such as which classes are most often confused with one another, and provides valuable insight into the model's strengths and weaknesses.

Results

Overall

The trained Convolutional Neural Network (CNN) model achieved a test accuracy of **85.32%** on the CIFAR-10 dataset, demonstrating strong performance in classifying images across the ten distinct categories. This result reflects the effectiveness of the model architecture, which incorporated multiple convolutional layers, batch normalization, dropout regularization to enhance generalization and prevent overfitting



Training and Validation Loss Curves

This graph illustrates the training and validation loss over 75 epochs during the training of the Convolutional Neural Network (CNN) on the CIFAR-10 dataset. The training loss (blue line) shows a steady and consistent decline, indicating that the model continued to learn and improve its performance on the training data with each epoch. In contrast, the validation loss (orange line) decreases initially but begins to stabilize and fluctuate slightly after around 20 epochs, which is a

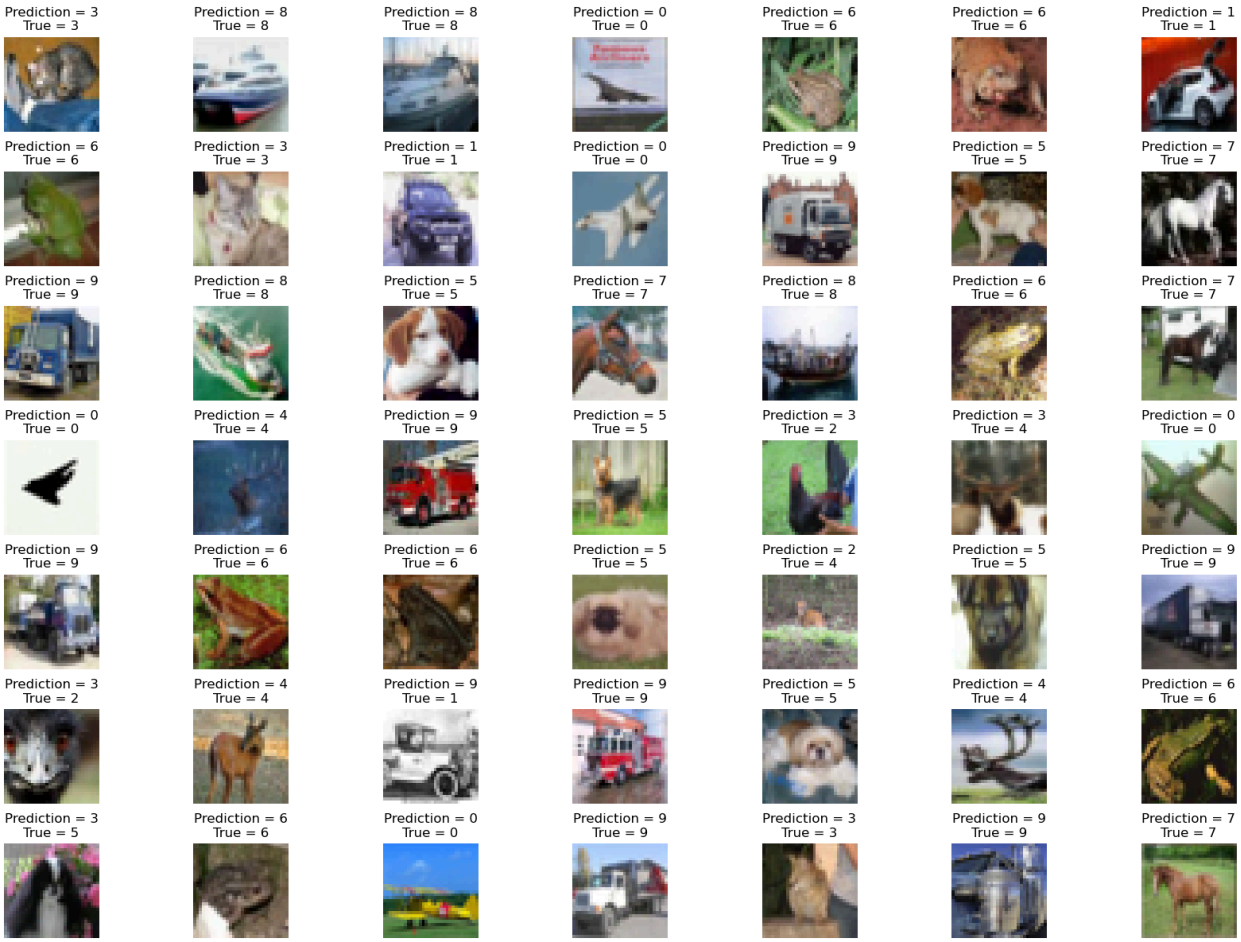
common pattern in deep learning and suggests that the model has reached a point where further improvements on unseen data are incremental.

The widening gap between the training and validation loss beyond this point may indicate the onset of mild overfitting, where the model continues to perform better on the training data but with diminishing returns on the validation set. However, the gap remains controlled, and no drastic overfitting is observed, suggesting a well-regularized and generalizable model.

Due to computational limitations, including memory and processing constraints of the system, the model was trained for only 75 epochs, despite the potential for further performance improvements with extended training. This limitation impacted both training time and the ability to experiment with deeper architectures or extensive hyperparameter tuning. Nevertheless, the model achieved a strong performance within these constraints, demonstrating the effectiveness of the chosen architecture and optimization strategy. This result highlights the balance between resource efficiency and model accuracy in practical machine learning workflows.

Specific Prediction

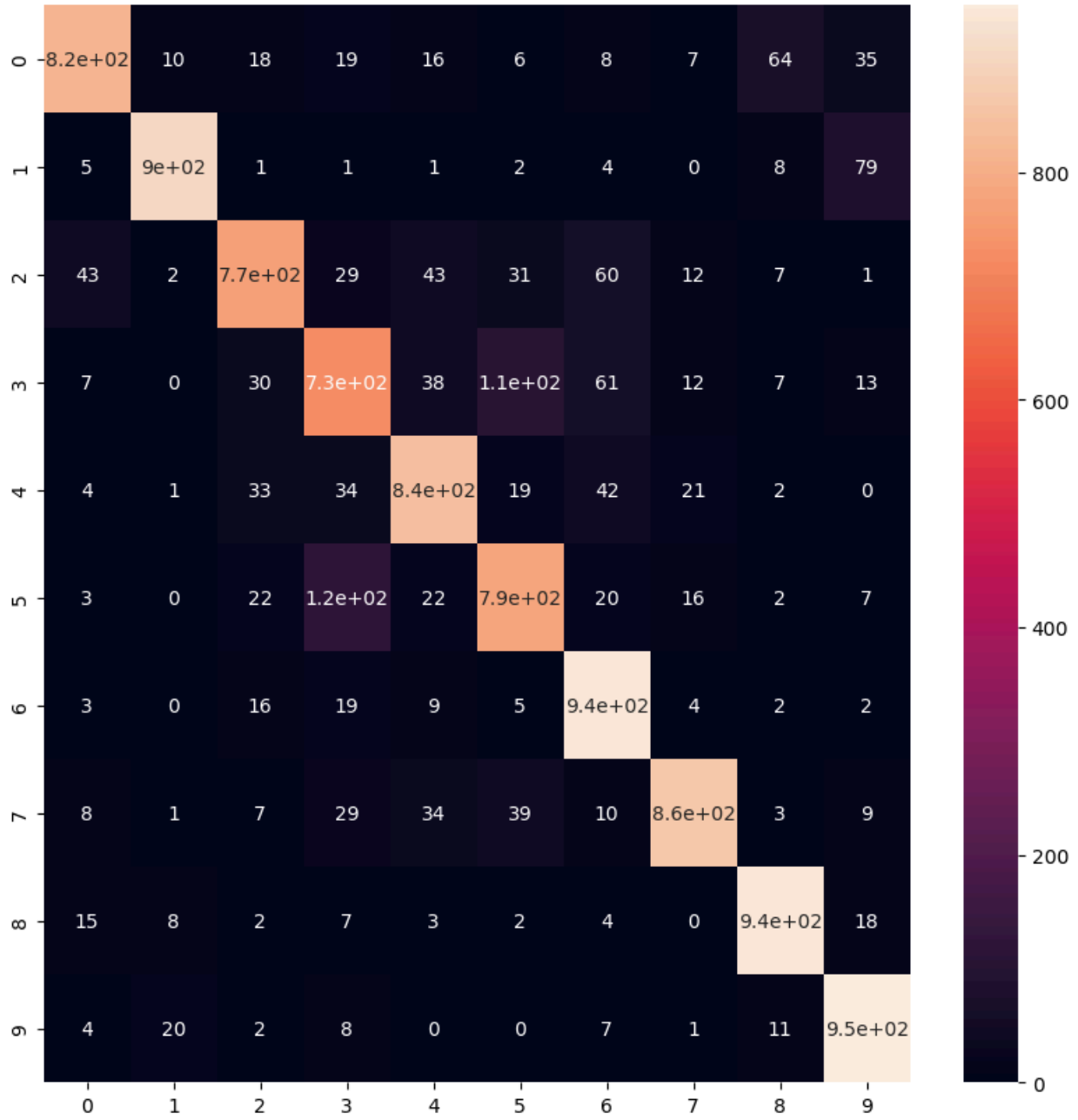
A Prediction vs True value graph was plotted as seen below which highlights the 85.32% accuracy of the Convolutional Neural Network.



Prediction vs True Values

Confusion Matrix

This is a confusion matrix heatmap for the CIFAR-10 classification model — and it's an excellent tool to understand which classes the model predicts well and where it struggles.



Confusion Matrix

References

GitHub Repository - Click Here

****