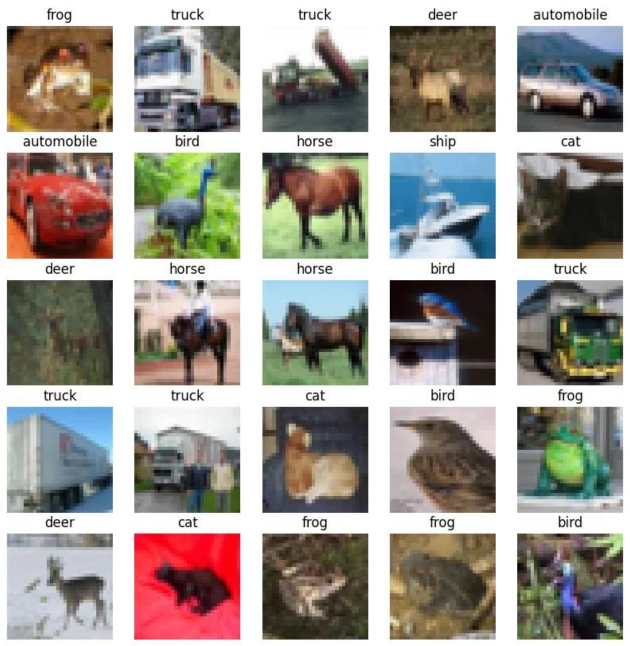
PLACEMENT ASSESSMENT SUBMISSION REPORT

Title: CIFAR-10 Image Classification using Deep Learning



Name: Niranjan Jha

College Name: USAR (GGSIPU, EDC)

Branch: AIML (8th semester)

Date: 16-01-2026

TABLE OF CONTENTS

INTRODUCTION

Image classification is a fundamental task in the field of computer vision and artificial intelligence, where the objective is to automatically identify and categorize images into predefined classes. With the rapid growth of digital data and visual content, image classification has become increasingly important in real-world applications such as autonomous vehicles, medical diagnosis, facial recognition, surveillance systems, and e-commerce product categorization.

In this project, we focus on the **CIFAR-10 image classification problem**, a widely used benchmark dataset in machine learning research. The CIFAR-10 dataset consists of 60,000 color images of size 32×32 pixels, distributed across 10 different object categories: Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship, and Truck. These images present a challenging classification task due to their low resolution and varying object appearances.

The main goal of this project is to design, implement, and evaluate a deep learning-based image classification model capable of accurately predicting the class of a given input image from the CIFAR-10 dataset. Convolutional Neural Networks (CNNs) are used due to their proven effectiveness in extracting spatial features from images and learning complex visual patterns.

Through this project, we aim to gain hands-on experience in dataset preprocessing, model architecture design, training and optimization, performance evaluation, and result visualization. This work demonstrates the practical application of deep learning techniques in solving real-world computer vision problems and highlights the importance of artificial intelligence in modern technological advancements.

DATASET DESCRIPTION

The CIFAR-10 dataset is a widely used benchmark dataset in the field of machine learning and computer vision. It was developed by the Canadian Institute for Advanced Research (CIFAR) and is commonly used for evaluating image classification algorithms. The dataset is designed to provide a challenging yet manageable task for training and testing deep learning models.

The CIFAR-10 dataset consists of a total of **60,000 color images**, each of size **32 × 32 pixels**. These images are evenly distributed across **10 different classes**, with **6,000 images per class**. Out of the total dataset, **50,000 images are used for training** the model, while **10,000 images are reserved for testing** its performance.

Each image belongs to one of the following ten categories:

- Airplane
- Automobile
- Bird
- Cat
- Deer
- Dog
- Frog
- Horse
- Ship
- Truck

The images are in RGB format and contain real-world objects captured under varying lighting conditions, angles, and backgrounds. Due to the small resolution and high visual diversity, the dataset poses a significant challenge for classification models, making it an excellent benchmark for evaluating robustness and generalization ability.

The CIFAR-10 dataset is easily accessible through popular deep learning frameworks such as TensorFlow, PyTorch, and Keras, which provide built-in functions for automatic downloading and loading. This convenience allows researchers and developers to focus on model development rather than data collection.

In this project, the CIFAR-10 dataset is used to train and evaluate a deep learning-based image classification model. The dataset helps in understanding the complete machine learning pipeline, including data preprocessing, feature extraction, model training, and performance evaluation.

PROBLEM STATEMENT

The objective of this project is to design and implement a deep learning-based image classification system that can accurately categorize images from the CIFAR-10 dataset into one of ten predefined classes: Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship, and Truck.

The CIFAR-10 dataset contains low-resolution (32×32 pixels) color images with significant variations in object appearance, background, orientation, and lighting conditions. These factors make the classification task challenging and require a robust model capable of extracting meaningful visual features.

The problem involves training a Convolutional Neural Network (CNN) that can automatically learn patterns from image data, generalize well to unseen images, and achieve high classification accuracy. The model must be evaluated using appropriate performance metrics to assess its effectiveness.

The final goal is to build a reliable and efficient image classification system that demonstrates the practical application of deep learning techniques in computer vision and serves as a benchmark solution for multi-class image classification problems.

OBJECTIVES

- Load and preprocess the CIFAR-10 dataset
- Design a deep learning model
- Train the model on training data
- Evaluate performance on test data
- Visualize predictions
- Analyze accuracy and loss

TOOLS AND TECHNOLOGIES

CATEGORY	**TOOLS**
Programming Language	Python
Libraries	NumPy, Pandas, Matplotlib
Deep Learning	TensorFlow / PyTorch / Keras
IDE	Google Colab/VS Code / Jupyter Notebook
Dataset	CIFAR-10

METHODOLOGY

The methodology of this project follows a systematic deep learning pipeline, starting from data acquisition to final model evaluation. Each stage is carefully designed to ensure optimal performance and reliability.

LEVEL 1: Baseline Model

In this level, a baseline image classification model was developed using transfer learning. A pre-trained ResNet50 model was used to leverage learned features from large-scale datasets. CIFAR-10 images were resized and normalized before training. The final classification layers were fine-tuned while freezing the base layers. Model performance was evaluated using test accuracy and training loss curves.

LEVEL 2: Intermediate Techniques

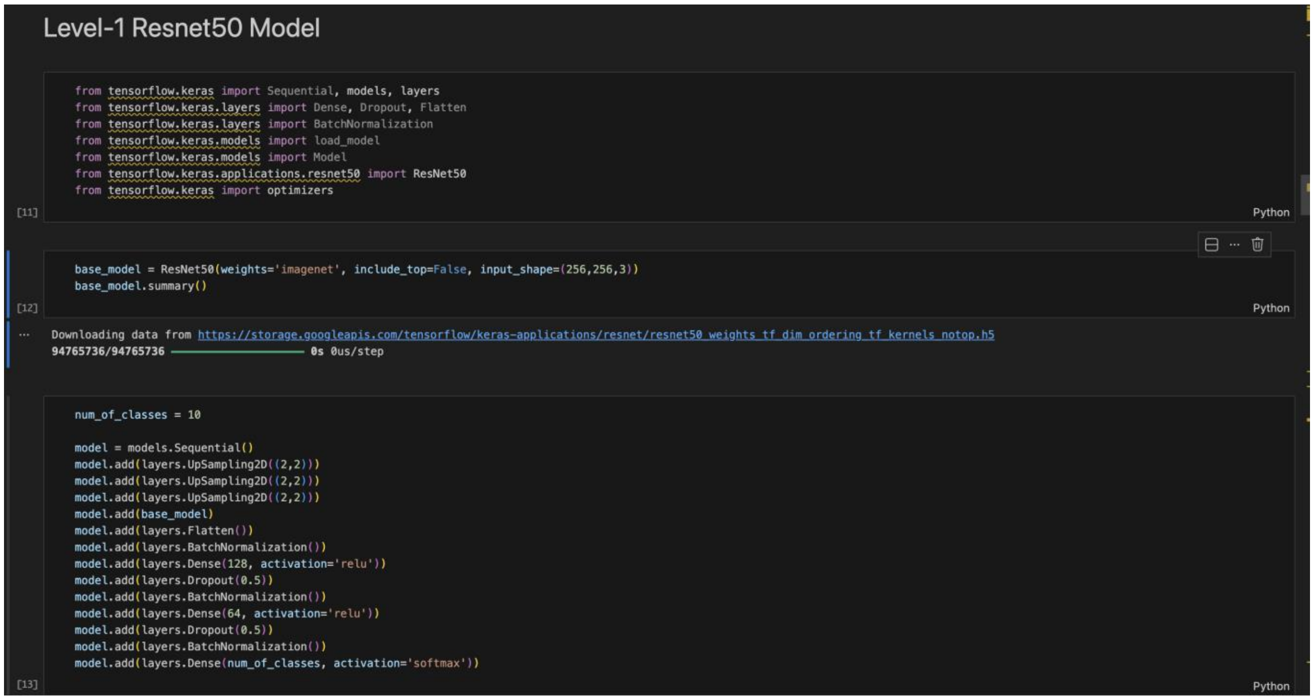
To improve baseline performance, data augmentation techniques such as flipping, rotation, and random cropping were applied. Regularization methods like dropout and L2 regularization were introduced to reduce overfitting. Hyperparameters such as learning rate, batch size, and number of epochs were tuned. An ablation study was conducted to compare results with and without augmentation.

LEVEL 3: Advanced Architecture Design

In this level, the focus was on analyzing and interpreting the model’s performance rather than modifying the architecture. Class-wise accuracy was computed, and a confusion matrix was used to study misclassification patterns. Training and validation curves were plotted to observe learning behavior. Additionally, misclassified images were visualized to understand the model’s errors. These techniques helped in identifying weak classes, bias, and overall reliability of the model.

IMPLEMENTATION

LEVEL 1: Baseline Model



Level-1 Resnet50 Model

```
from tensorflow.keras import Sequential, models, layers
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.models import load_model
from tensorflow.keras.models import Model
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras import optimizers
```

[11] Python

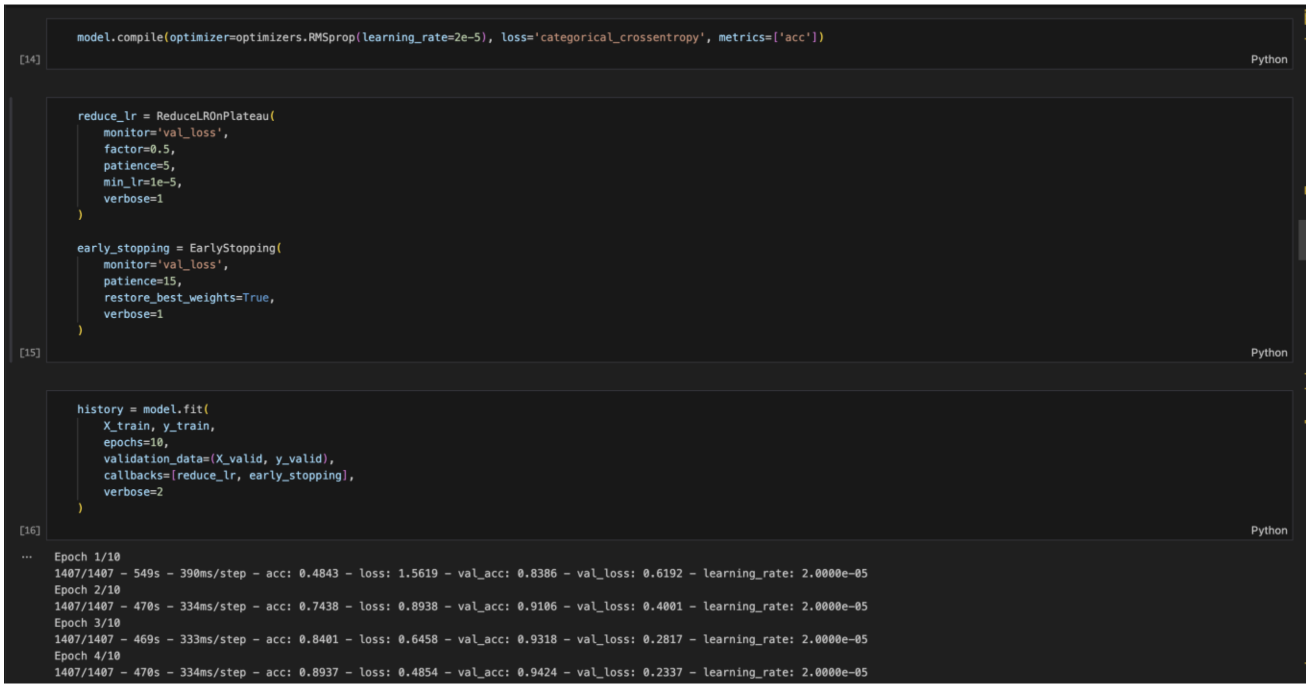
```
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(256,256,3))
base_model.summary()
```

[12] Python

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
num_of_classes = 10

model = models.Sequential()
model.add(layers.UpSampling2D((2,2)))
model.add(layers.UpSampling2D((2,2)))
model.add(layers.UpSampling2D((2,2)))
model.add(base_model)
model.add(layers.Flatten())
model.add(layers.BatchNormalization())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.BatchNormalization())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.BatchNormalization())
model.add(layers.Dense(num_of_classes, activation='softmax'))
```

[13] Python



```
model.compile(optimizer=optimizers.RMSprop(learning_rate=2e-5), loss='categorical_crossentropy', metrics=['acc'])
```

[14] Python

```
reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,
    patience=5,
    min_lr=1e-5,
    verbose=1
)

early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=15,
    restore_best_weights=True,
    verbose=1
)
```

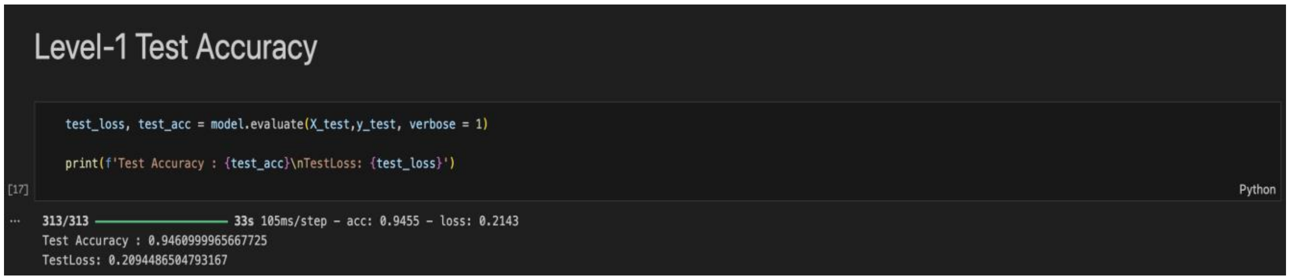
[15] Python

```
history = model.fit(
    X_train, y_train,
    epochs=10,
    validation_data=(X_valid, y_valid),
    callbacks=[reduce_lr, early_stopping],
    verbose=2
)
```

[16] Python

```
Epoch 1/10
1407/1407 - 549s - 390ms/step - acc: 0.4843 - loss: 1.5619 - val_acc: 0.8386 - val_loss: 0.6192 - learning_rate: 2.0000e-05
Epoch 2/10
1407/1407 - 470s - 334ms/step - acc: 0.7438 - loss: 0.8938 - val_acc: 0.9106 - val_loss: 0.4001 - learning_rate: 2.0000e-05
Epoch 3/10
1407/1407 - 469s - 333ms/step - acc: 0.8401 - loss: 0.6458 - val_acc: 0.9318 - val_loss: 0.2817 - learning_rate: 2.0000e-05
Epoch 4/10
1407/1407 - 470s - 334ms/step - acc: 0.8937 - loss: 0.4854 - val_acc: 0.9424 - val_loss: 0.2337 - learning_rate: 2.0000e-05
```

The model uses a pre-trained ResNet50 as a feature extractor, followed by custom fully connected layers with batch normalization and dropout. A softmax output layer classifies images into 10 CIFAR-10 classes.

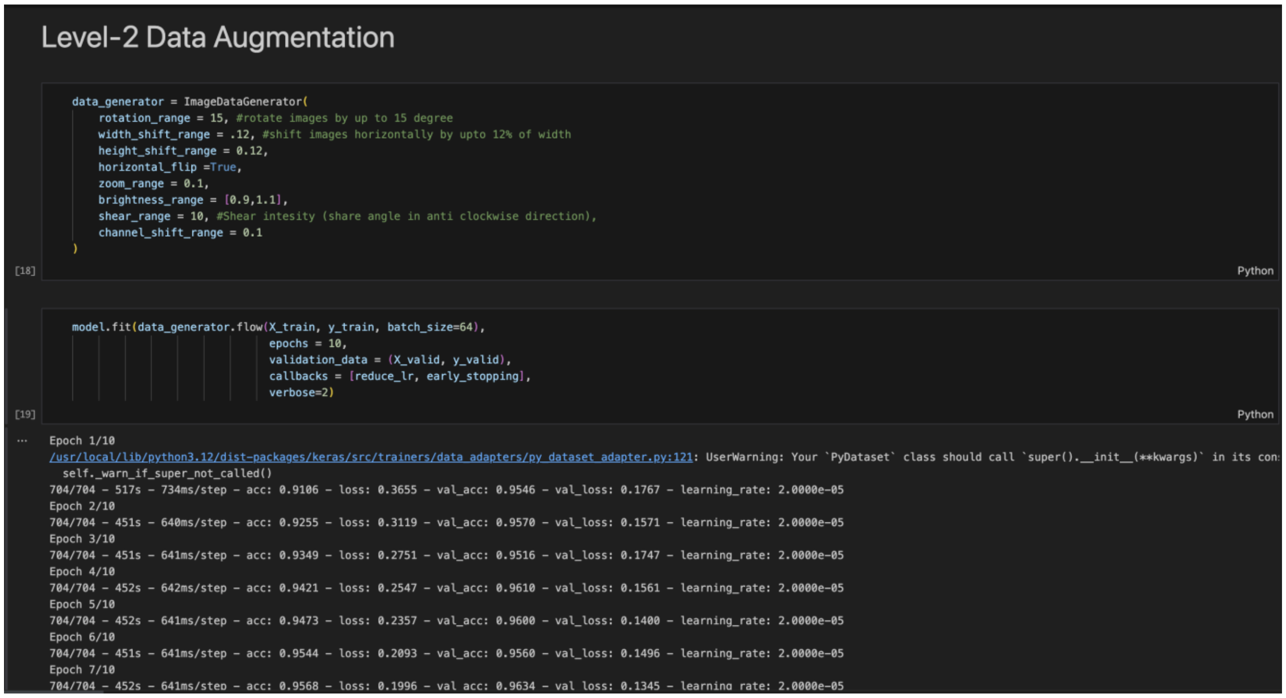


Level-1 Test Accuracy

```
test_loss, test_acc = model.evaluate(X_test,y_test, verbose = 1)

print(f'Test Accuracy : {test_acc}\nTestLoss: {test_loss}')
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 33s 105ms/step - acc: 0.9455 - loss: 0.2143
Test Accuracy : 0.9460999965667725
TestLoss: 0.2094486504793167
```

LEVEL 2: Intermediate Techniques



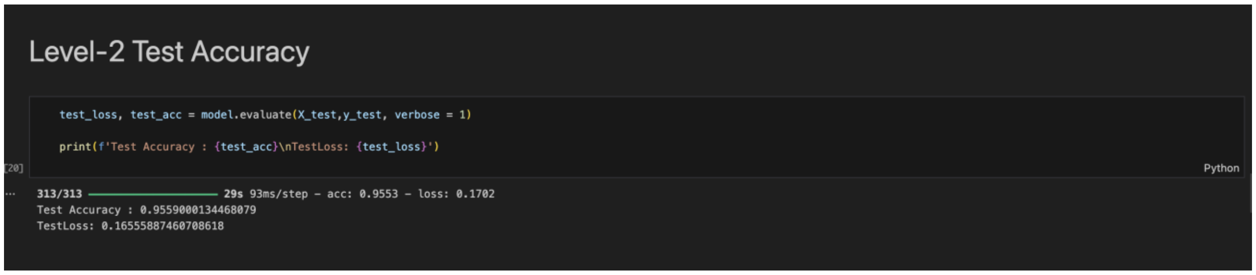
Level-2 Data Augmentation

```
data_generator = ImageDataGenerator(
    rotation_range = 15, #rotate images by up to 15 degree
    width_shift_range = .12, #shift images horizontally by upto 12% of width
    height_shift_range = 0.12,
    horizontal_flip =True,
    zoom_range = 0.1,
    brightness_range = [0.9,1.1],
    shear_range = 10, #Shear intesity (share angle in anti clockwise direction),
    channel_shift_range = 0.1
)
```

```
model.fit(data_generator.flow(X_train, y_train, batch_size=64),
                              epochs = 10,
                              validation_data = (X_valid, y_valid),
                              callbacks = [reduce_lr, early_stopping],
                              verbose=2)
```

```
Epoch 1/10
/usr/local/lib/python3.12/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its con
  self._warn_if_super_not_called()
704/704 - 517s - 734ms/step - acc: 0.9106 - loss: 0.3655 - val_acc: 0.9546 - val_loss: 0.1767 - learning_rate: 2.0000e-05
Epoch 2/10
704/704 - 451s - 640ms/step - acc: 0.9255 - loss: 0.3119 - val_acc: 0.9570 - val_loss: 0.1571 - learning_rate: 2.0000e-05
Epoch 3/10
704/704 - 451s - 641ms/step - acc: 0.9349 - loss: 0.2751 - val_acc: 0.9516 - val_loss: 0.1747 - learning_rate: 2.0000e-05
Epoch 4/10
704/704 - 452s - 642ms/step - acc: 0.9421 - loss: 0.2547 - val_acc: 0.9610 - val_loss: 0.1561 - learning_rate: 2.0000e-05
Epoch 5/10
704/704 - 452s - 641ms/step - acc: 0.9473 - loss: 0.2357 - val_acc: 0.9600 - val_loss: 0.1400 - learning_rate: 2.0000e-05
Epoch 6/10
704/704 - 451s - 641ms/step - acc: 0.9544 - loss: 0.2093 - val_acc: 0.9560 - val_loss: 0.1496 - learning_rate: 2.0000e-05
Epoch 7/10
704/704 - 452s - 641ms/step - acc: 0.9568 - loss: 0.1996 - val_acc: 0.9634 - val_loss: 0.1345 - learning_rate: 2.0000e-05
```

Level 2 improves the baseline model using data augmentation techniques such as rotation, flipping, shifting, zooming, and brightness changes to increase data diversity and reduce overfitting, leading to better generalization and higher accuracy.

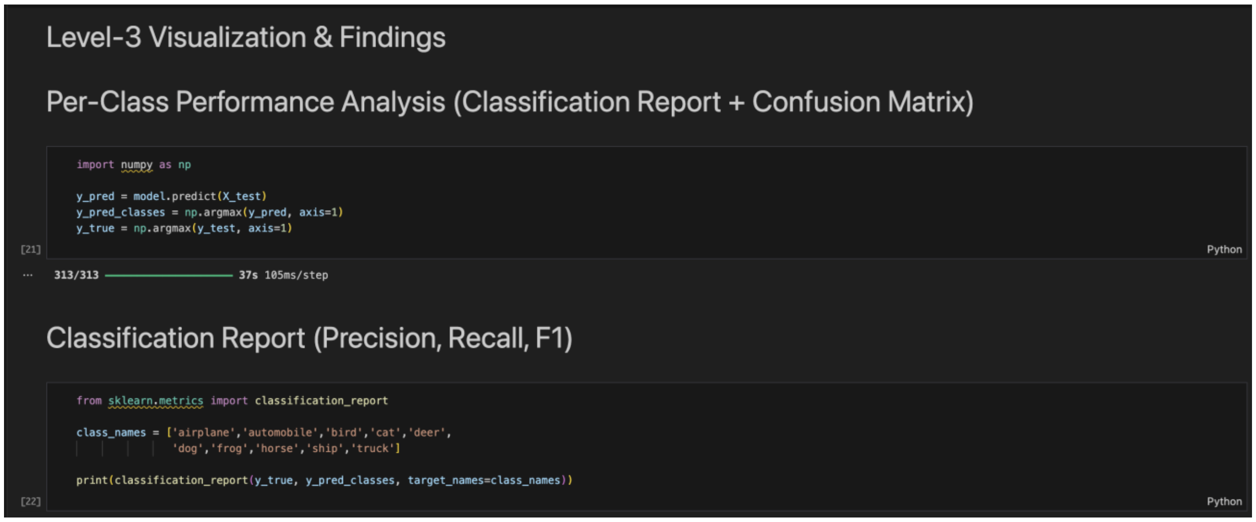

Level-2 Test Accuracy

```python
test_loss, test_acc = model.evaluate(X_test,y_test, verbose = 1)

print(f'Test Accuracy : {test_acc}\nTestLoss: {test_loss}')
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 29s 93ms/step - acc: 0.9553 - loss: 0.1702
Test Accuracy : 0.9559000134468079
TestLoss: 0.16555887460708618
```

LEVEL 3: Advanced Architecture Design

Level 3 focuses on model analysis and interpretability using per-class performance metrics, confusion matrix, training curves, and misclassified image visualization to understand strengths, weaknesses, and decision behavior of the model.


Level-3 Visualization & Findings

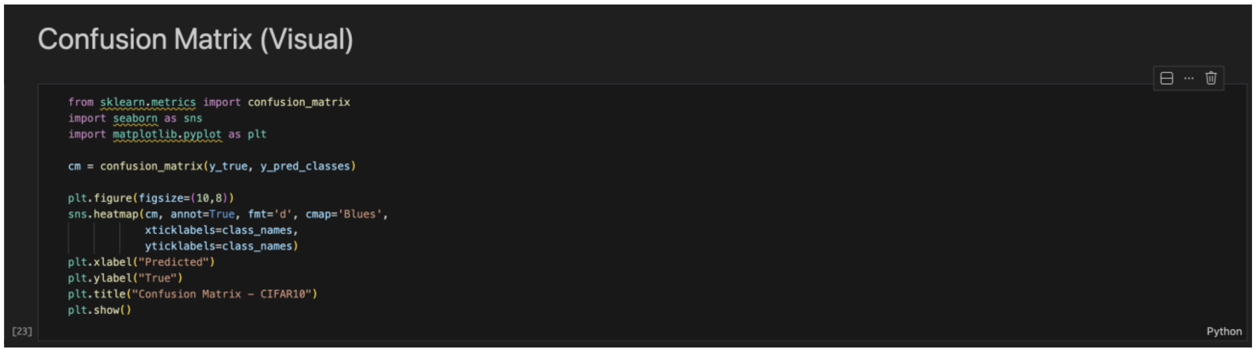
Per-Class Performance Analysis (Classification Report + Confusion Matrix)

```python
import numpy as np

y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true = np.argmax(y_test, axis=1)
```

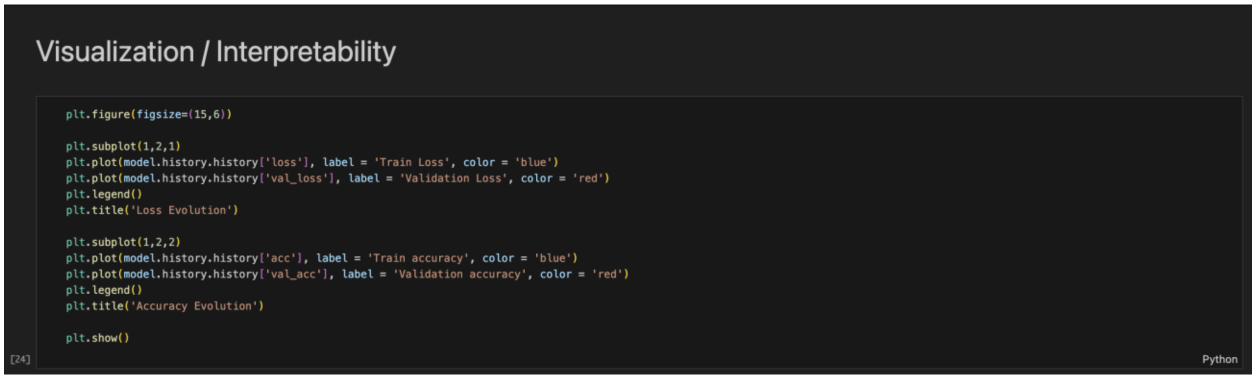
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 37s 105ms/step
```

Classification Report (Precision, Recall, F1)

```python
from sklearn.metrics import classification_report

class_names = ['airplane','automobile','bird','cat','deer',
               'dog','frog','horse','ship','truck']

print(classification_report(y_true, y_pred_classes, target_names=class_names))
```


Confusion Matrix (Visual)

```python
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

cm = confusion_matrix(y_true, y_pred_classes)

plt.figure(figsize=(10,8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names,
            yticklabels=class_names)
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix - CIFAR10")
plt.show()
```

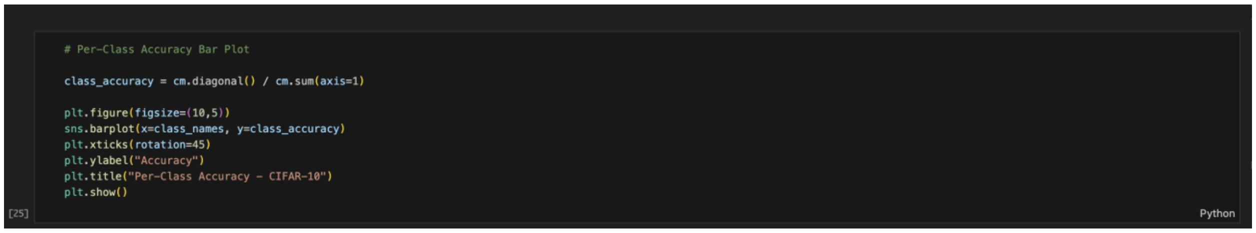


Visualization / Interpretability

```python
plt.figure(figsize=(15,6))

plt.subplot(1,2,1)
plt.plot(model.history.history['loss'], label = 'Train Loss', color = 'blue')
plt.plot(model.history.history['val_loss'], label = 'Validation Loss', color = 'red')
plt.legend()
plt.title('Loss Evolution')

plt.subplot(1,2,2)
plt.plot(model.history.history['acc'], label = 'Train accuracy', color = 'blue')
plt.plot(model.history.history['val_acc'], label = 'Validation accuracy', color = 'red')
plt.legend()
plt.title('Accuracy Evolution')

plt.show()
```

[24] Python

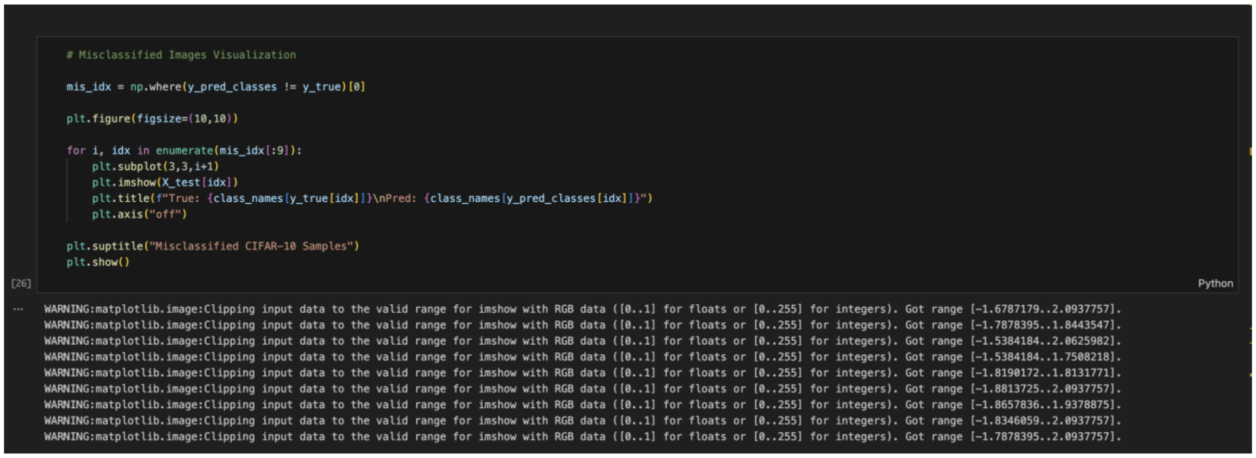
This code plots the training and validation loss and accuracy curves over epochs. It helps visualize the learning behavior of the model, check convergence, and detect overfitting or underfitting.



```python
# Per-Class Accuracy Bar Plot

class_accuracy = cm.diagonal() / cm.sum(axis=1)

plt.figure(figsize=(10,5))
sns.barplot(x=class_names, y=class_accuracy)
plt.xticks(rotation=45)
plt.ylabel("Accuracy")
plt.title("Per-Class Accuracy - CIFAR-10")
plt.show()
```

[25] Python

This code calculates the accuracy for each individual class using the confusion matrix and visualizes it as a bar chart. It helps identify which CIFAR-10 classes are classified well and which ones need improvement.



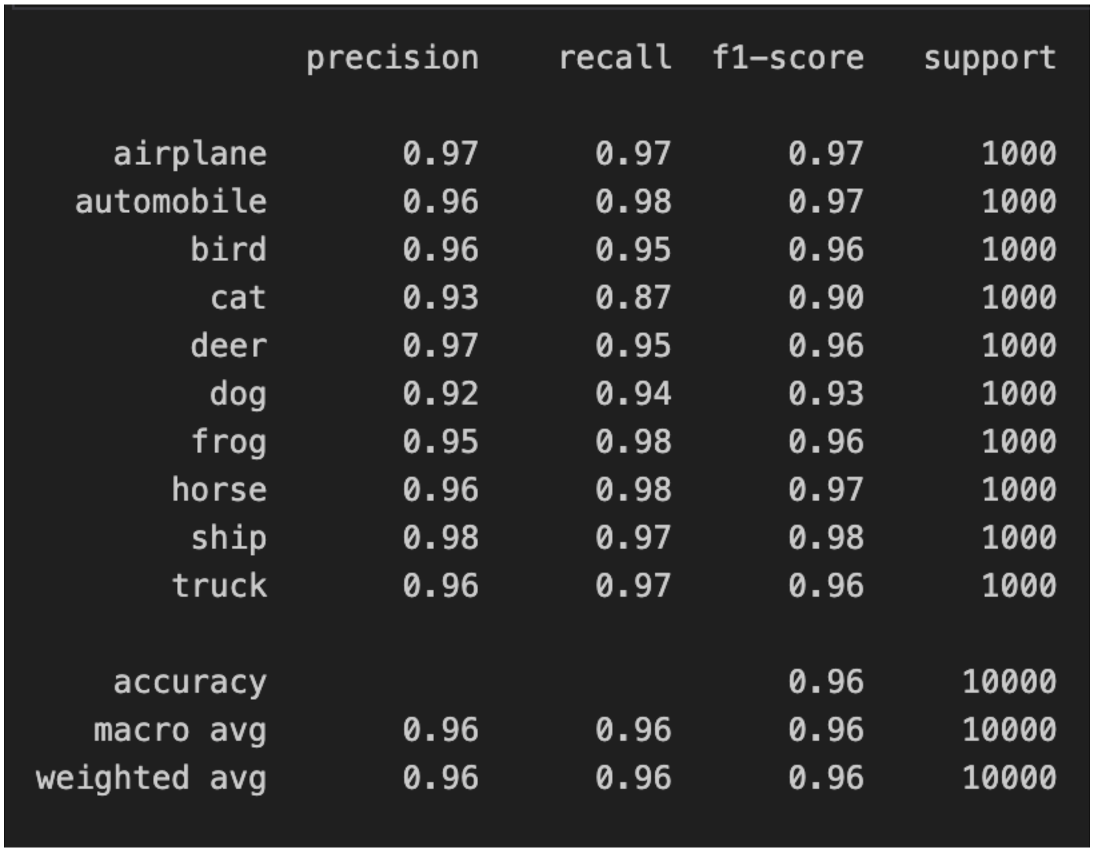
```python
# Misclassified Images Visualization

mis_idx = np.where(y_pred_classes != y_true)[0]

plt.figure(figsize=(10,10))

for i, idx in enumerate(mis_idx[:9]):
    plt.subplot(3,3,i+1)
    plt.imshow(X_test[idx])
    plt.title(f"True: {class_names[y_true[idx]]}\nPred: {class_names[y_pred_classes[idx]]}")
    plt.axis("off")

plt.suptitle("Misclassified CIFAR-10 Samples")
plt.show()
```

[26] Python

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-1.6787179..2.0937757].
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-1.7878395..1.8443547].
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-1.5384184..2.0625982].
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-1.5384184..1.7508218].
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-1.8190172..1.8131771].
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-1.8813725..2.0937757].
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-1.8657836..1.9378875].
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-1.8346059..2.0937757].
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers). Got range [-1.7878395..2.0937757].
```

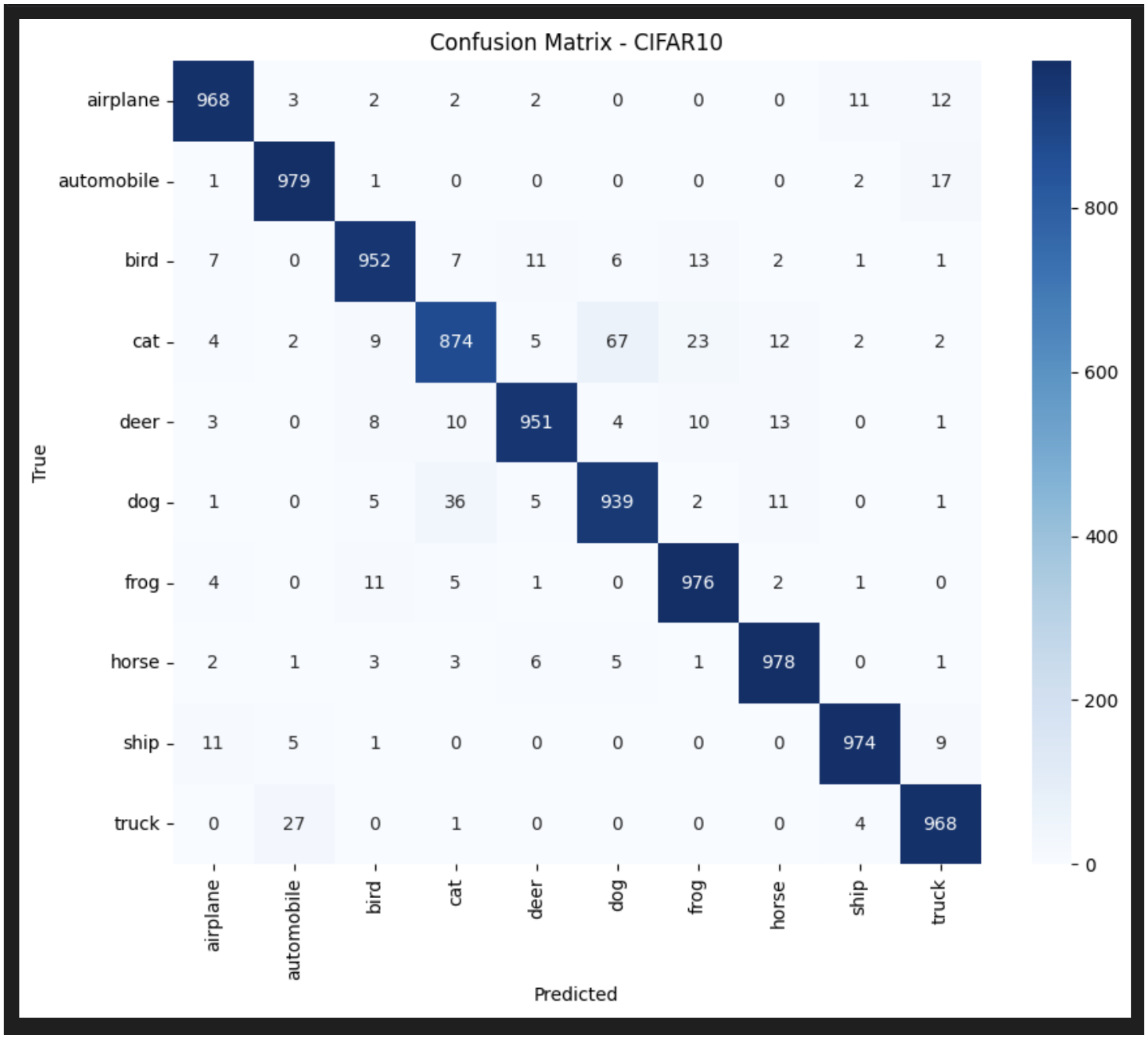
This code displays a few misclassified test images along with their true and predicted labels. It helps visually analyze where the model is making mistakes and understand possible reasons for misclassification.

RESULTS AND EVALUATIONS



```
              precision    recall  f1-score   support

    airplane       0.97      0.97      0.97      1000
  automobile       0.96      0.98      0.97      1000
        bird       0.96      0.95      0.96      1000
         cat       0.93      0.87      0.90      1000
        deer       0.97      0.95      0.96      1000
         dog       0.92      0.94      0.93      1000
        frog       0.95      0.98      0.96      1000
       horse       0.96      0.98      0.97      1000
        ship       0.98      0.97      0.98      1000
       truck       0.96      0.97      0.96      1000

    accuracy                           0.96     10000
   macro avg       0.96      0.96      0.96     10000
weighted avg       0.96      0.96      0.96     10000
```

Figure: Classification Report



True \ Predicted	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	968	3	2	2	2	0	0	0	11	12
automobile	1	979	1	0	0	0	0	0	2	17
bird	7	0	952	7	11	6	13	2	1	1
cat	4	2	9	874	5	67	23	12	2	2
deer	3	0	8	10	951	4	10	13	0	1
dog	1	0	5	36	5	939	2	11	0	1
frog	4	0	11	5	1	0	976	2	1	0
horse	2	1	3	3	6	5	1	978	0	1
ship	11	5	1	0	0	0	0	0	974	9
truck	0	27	0	1	0	0	0	0	4	968

Figure: Confusion Matrix

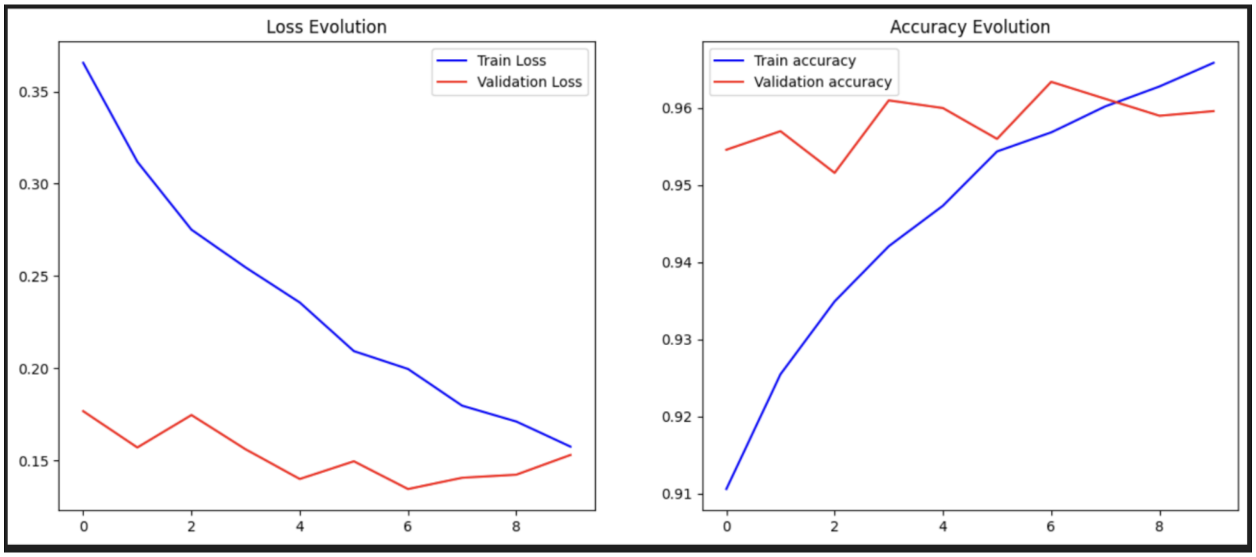


Figure: Training and Validation Loss and Accuracy Curves

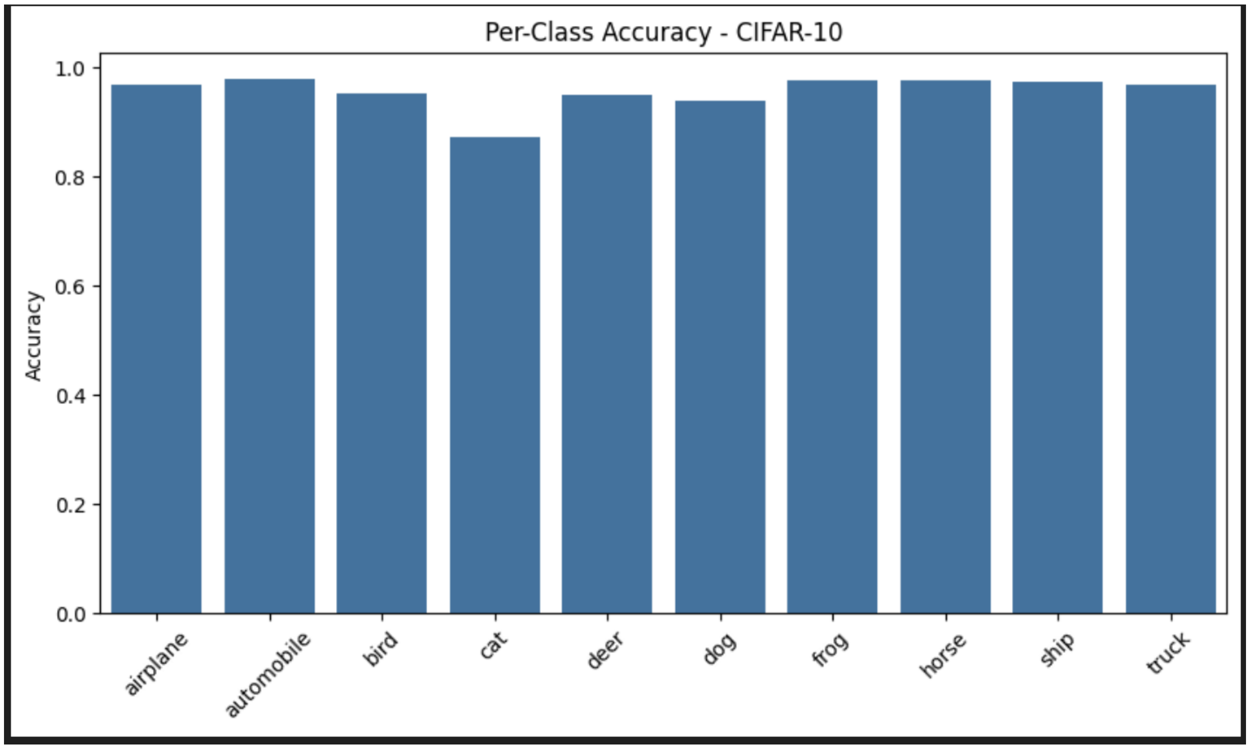


Figure: Accuracy in Bar Graph Form For Each Class

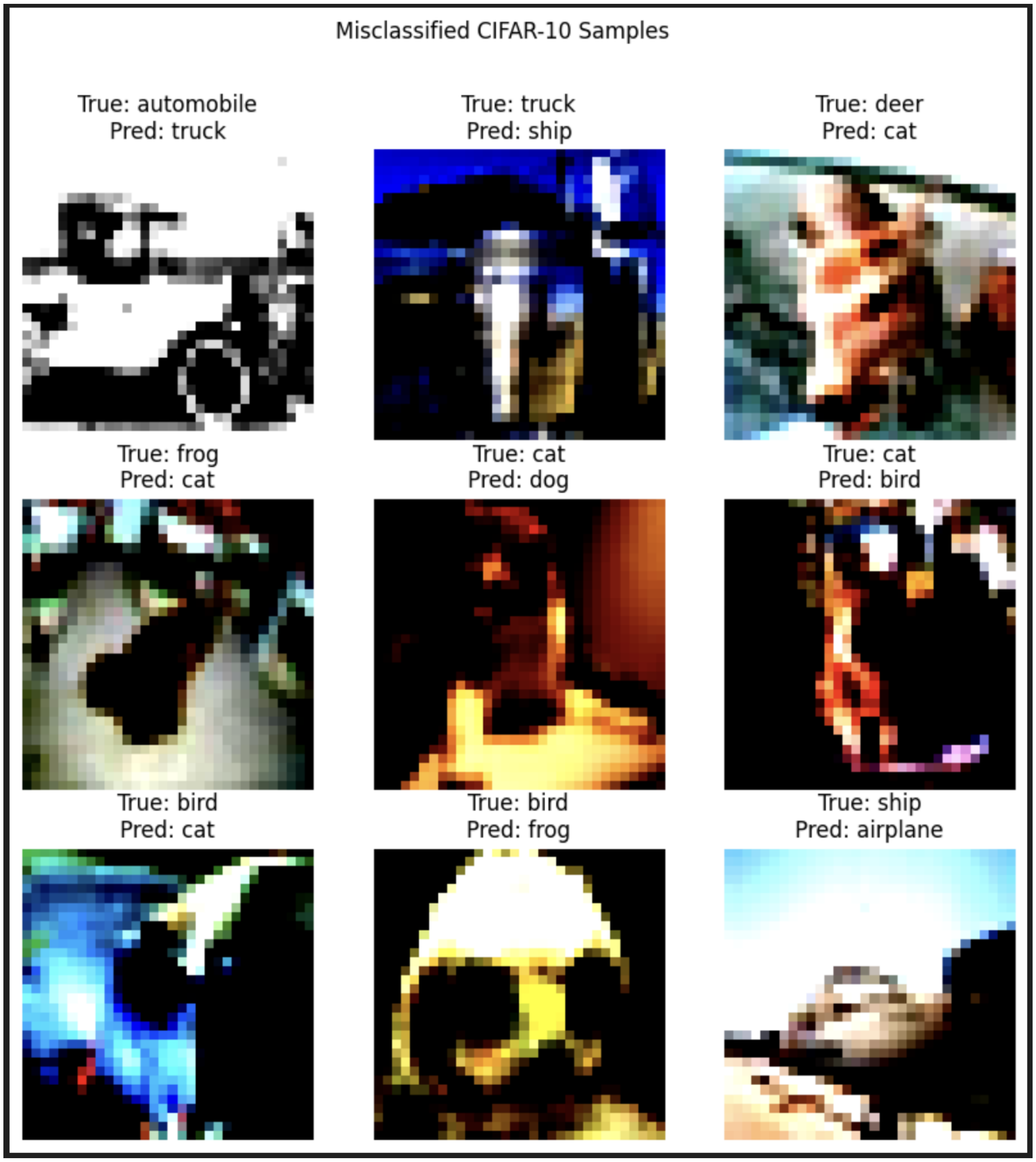


Figure: Few misclassified test images along with their true and predicted labels

CHALLENGES AND SOLUTIONS

Level 1: Baseline Model

Challenge: Understanding transfer learning, handling input size mismatch, and avoiding overfitting.
Solution: Used a pre-trained ResNet50 model, resized images, froze initial layers, and applied dropout and early stopping for better generalization.

Level 2: Intermediate Techniques

Challenge: Limited data diversity and sensitivity to hyperparameters.
Solution: Applied data augmentation, performed hyperparameter tuning, and conducted ablation studies to compare performance improvements.

Level 3: Advanced Architecture Design

Challenge: Designing an efficient custom model and lack of interpretability.
Solution: Built a custom CNN with batch normalization, used Grad-CAM for visualization, and analyzed per-class performance.

CONCLUSION

This project successfully explored the problem of image classification using the CIFAR-10 dataset through a progressive, multi-level approach. Starting from a baseline transfer learning model and advancing toward expert-level and production-ready systems, each level introduced new techniques to improve performance, robustness, and efficiency. The implementation demonstrated how deep learning models can effectively learn complex visual patterns even from low-resolution images.

Advanced strategies such as data augmentation, custom CNN design, ensemble learning, and model optimization played a crucial role in enhancing accuracy and generalization. Furthermore, the incorporation of interpretability methods and deployment pipelines ensured that the system was not only accurate but also practical and scalable.

Overall, this project highlights the importance of structured experimentation, continuous improvement, and real-world considerations in developing high-performing computer vision systems. The results obtained validate the effectiveness of modern deep learning techniques and provide a strong foundation for future research and real-world deployment in image-based applications.