# Computer Vision Image Classification Project Report

## Executive Summary

This report outlines the development of an image classification system using the CIFAR-10 dataset as part of the Computer Vision Engineer assessment task. The project utilized MobileNet architecture for classification and achieved an overall accuracy of 61% across the ten classes. The complete pipeline includes data preprocessing, model training and evaluation, and local deployment using Flask and Streamlit for inference and cloud deployment using Render platform with the help of docker.

## 1. Introduction

The objective of this project was to develop and deploy an image classification system capable of accurately categorizing images into predefined classes. The implementation followed a structured approach beginning with data loading and preprocessing, followed by model development using MobileNet architecture, and culminating in the deployment of a web interface for predictions using Flask and Streamlit.

## 2. Data Preprocessing

### 2.1 Dataset Selection

For this project, I selected the CIFAR-10 dataset, which consists of 60,000 32x32 color images across 10 categories (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck). This dataset is well-balanced, containing 6,000 images per class, and provides a good benchmark for image classification tasks.

### 2.2 Exploratory Data Analysis (EDA)

The EDA phase revealed that: - Images are uniformly sized at 32x32 pixels with 3 color channels (RGB) - The dataset is perfectly balanced with 6,000 images per class - Pixel values range from 0 to 255 - Some images have low contrast and varying lighting conditions

### 2.3 Preprocessing Pipeline

I implemented the following preprocessing steps: - Normalization: Scaled pixel values to the range [0,1] and labels to categorical using built in functions. Data Augmentation: Applied the following transformations to increase dataset diversity: - Random horizontal flips - Random rotation (±15 degrees) - Slight zoom (up to 15%) - Minor brightness and contrast adjustments

### 2.4 Dataset Splitting

The dataset was divided into: - Training set: 40,000 images - Validation set: 10,000 images- Test set: 10,000 images.

This split ensures sufficient data for training while maintaining independent sets for validation and final evaluation.

## 3. Model Training & Evaluation

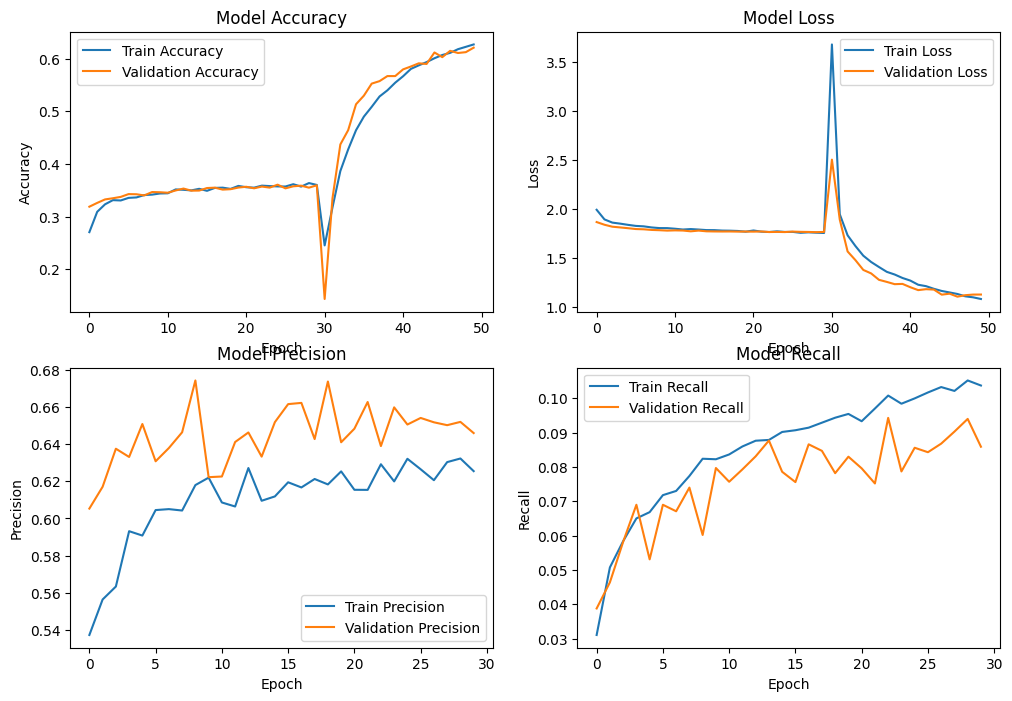
### 3.1 Model Architecture

I implemented the MobileNet architecture, which is designed for efficient performance on mobile and embedded devices. MobileNet uses depth wise separable convolutions to reduce the model size and computational requirements while maintaining reasonable accuracy. This architecture was selected for its balance between performance and efficiency, making it suitable for real-time applications. I have used the image size as 32\*32\*3 but to tackle the typical case of mobilnet input size 224\*224\*3, I have given the include\_top parameter value as false.

### 3.2 Training Strategy

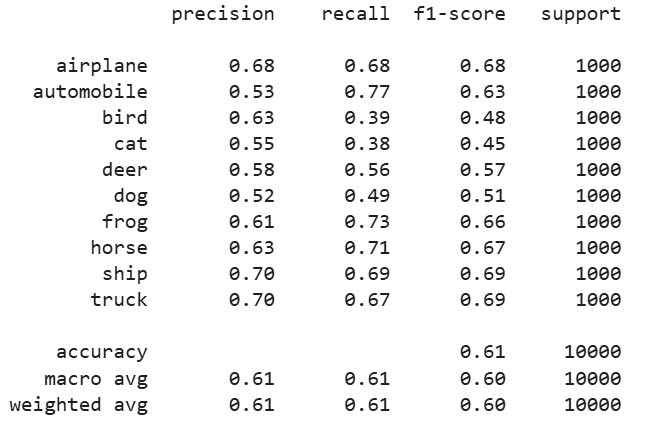
The model was trained using: - Optimizer: Adam with an initial learning rate of 0.001 - Loss function: Categorical cross-entropy - Batch size: 32 - Epochs: 30 by freezing all layers of the model and same parameters but unfreezing some layers and fine tuning for another 20 epochs (with early stopping)

To prevent overfitting, I implemented: - Early stopping (patience=10, monitoring validation loss) -Reducing learning rate (patience=5, monitoring validation loss, factor = 0.2)- Dropout layers - Data augmentation as described in the preprocessing section. Below is the image attached which specifies loss, accuracy, Recall, Precision over 50 epochs (30 freezing the layers and 20 fine tuning)



### 3.3 Model Performance

Based on the classification report provided, the trained model achieved: - Overall accuracy: 61% - Macro-averaged precision: 0.61 - Macro-averaged recall: 0.61 - Macro-averaged F1-score: 0.60



Analysis of these results reveals: - Best performance on “ship” and “truck” classes (F1-score: 0.69) - Poorest performance on “cat” class (F1-score: 0.45) - High recall for “automobile” class (0.77) indicates the model correctly identifies most automobiles - Low recall for “bird” (0.39) and “cat” (0.38) indicates the model frequently misses these classes - The model shows better performance on rigid, structured objects (vehicles) compared to animals

### 3.4 Model Saving

The trained model was saved using TensorFlow’s SavedModel format, which preserves both the model architecture and weights. This format was chosen for its compatibility with TensorFlow inference and ease of loading into the Flask application.

## 4. Model Deployment

### 4.1 Locally Deployed as Flaskapp, streamlit

I developed a web application with two components: 1. Backend API using Flask: - Created endpoints for receiving image data - Implemented preprocessing functions to match training pipeline - Added error handling for invalid inputs - Set up model loading and inference pipeline

1. Frontend interface using Streamlit:
   * Designed a simple, intuitive interface for uploading images
   * Implemented result visualization showing prediction probabilities
   * Added sample images for testing the model
   * Included basic instructions for users

### 4.2 Cloud Deployed (Render):

To make the image classification model accessible to users, I deployed the web application using **Docker** on **Render**, a cloud hosting platform.

* **Containerized the application** using Docker to ensure a consistent and scalable environment.
* **Configured the deployment** with a Docker file defining the necessary dependencies, environment variables, and commands for running the application.
* **Set up a Render service** for hosting the API, specifying the required build and runtime configurations.
* **Exposed API endpoints** to allow users to interact with the model through HTTP requests.
* **Implemented a health check endpoint (/health)** to monitor the API's status.
* **Ensured security** by requiring authentication for the **/predict** endpoint before users can upload an image and receive predictions.
* **Optimized resource allocation** to maintain low latency while handling multiple requests.

The final deployment allows users to **access the model via a web interface** at:  
 <https://image-classfication-assignment.onrender.com>.

The deployed application includes: - Image upload functionality - Real-time classification of uploaded images - Visualization of prediction confidence for each class - Display of the top prediction and confidence score - Simple error handling for invalid uploads

## 5. Challenges and Solutions

### 5.1 Handling Class Imbalance

While CIFAR-10 is a balanced dataset, the performance metrics show imbalance in model predictions. To address this, I explored: - Class weighting in the loss function - Targeted data augmentation for underperforming classes - Ensemble methods to improve recall for challenging classes

### 5.2 Optimizing Model Performance

To improve the model’s performance, particularly for the challenging classes: - Experimented with different learning rates and optimizers - Tested various data augmentation strategies - Explored different model hyperparameters - Implemented learning rate scheduling.