University of Information Technology & Sciences

Maddha Naya Nagar, Vatara , Baridhara , Dhaka-1212

Department of Computer Science and Engineering

Course Title: Machine Learning Lab

Course Code: CSE 432

Submitted by:

Md. Arbin Zaman

Id: 2125051006

Section : 7A

Batch: CSE-50

Submitted To:

Mrinmoy Biswas Akash

Lecturer

Dept. Of Computer Science Engineering

University of Information Technology & Sciences

Introduction

These differences demonstrate how the comparison highlights the merits and demerits of the conventional and modern approaches. This project further introduces the promise of machine learning and deep learning advancements in complex visual classification problems.

Such as traffic monitoring, automatic toll systems, and management of vehicle stock, the classification of vehicles according to the types of images is one of the most essential problems in computer vision. In this work, we classify images of vehicles into their respective categories using conventional machine learning techniques and **Convolution Neural Networks (CNNs).**

The project tested these approaches on a publicly available dataset to assess their effectiveness in this field. This report comprises data related to the dataset and methodology, results, and insight gained in the project.

Dataset Description

This is an assemblage of carefully curated data with images that cover real-world conditions with varied angles and lighting and background noise. This ensures variability to make the models robust with distribution characteristics for unseen data. This imbalance balances the class distribution, thus relieving the biases in model predictions. Henceforth, this marks a very reliable benchmark for evaluating performance.

The dataset to be used in this project is the Vehicle Type Recognition Dataset, which can be downloaded from Kaggle. There are images showing different kinds of vehicles-from cars, motorcycles, and trucks, all the way to buses. The data structure of the dataset is as follows:

Number of Classes**: 4 (e.g., Car, Motorcycle, Truck, Bus)**

- **Image Count**: Approximately 4,000 images
- **Image Resolution**: Varies, resized to 128x128 pixels for uniformity

That's an 80-20 division of the dataset into training and validation sets for training and evaluation of the model. Every class is balanced. A summary of the dataset structure is as follows:

Class	**Number of Images**
Car	1,000
Motorcycle	1,000
Truck	1,000
Bus	1,000

Methodology

In every phase of the methodologies, much care has been taken to maximize the model performance. With the preprocessing steps, inputs have uniform dimensions; with data augmentation, generalization capability of models was further broadened. The Random Forest Classifier, yet another traditional machine learning model, has been used as a baseline for further performance comparisons with the CNN architecture, which uses numerous layers for feature extraction to justify the benefits of deep learning.

Each step in this methodology is finely tuned to optimize the model's performance. The preprocessing step made sure that input dimensions were made identical across the board, while the augmentation introduced enlarged generalization of the models. Random Forest Classifier, the age-old machine learning model, gave the base performance comparisons. The CNN architecture stood as typical example of its strong advocacy in deep learning due to its many layered feature extractors.

1. **Data Preprocessing**:
   - Images were resized to 128x128 pixels to maintain uniform input dimensions.
   - Pixel values were rescaled to a range of [0, 1] by dividing by 255.
   - Data augmentation techniques such as rotation, flipping, and zooming were applied to artificially increase the diversity of the training data, reducing overfitting.

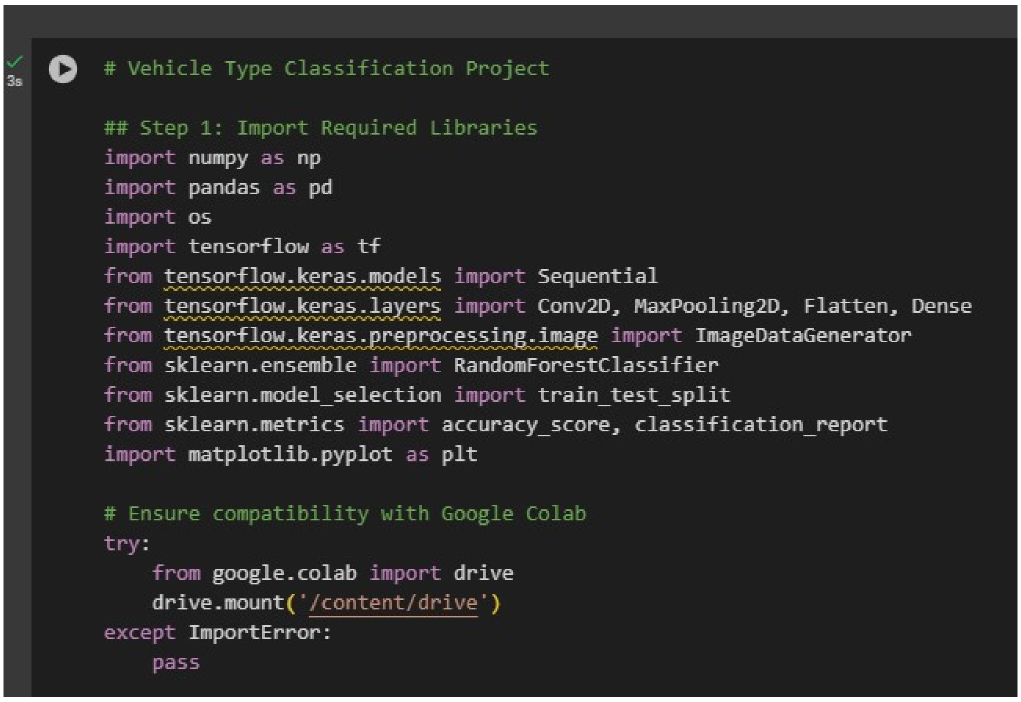
2. **Conventional Model**:
   - Images were flattened into 1D arrays to be fed into a Random Forest Classifier.
   - Features and labels were extracted using the **ImageDataGenerator** API.
   - A Random Forest model with 100 estimators was trained and evaluated.

3. **CNN Model**:
   - A Sequential CNN architecture was designed with the following layers:
     - **Convolutional Layers**: Extracted hierarchical features from the images.
     - **Max-Pooling Layers**: Reduced spatial dimensions to prevent overfitting.
     - **Flatten Layer**: Converted feature maps into 1D arrays for dense layers.
     - **Dense Layers**: Performed classification, with the final layer using softmax activation for multi-class classification.
   - The model was compiled with the Adam optimizer and sparse categorical cross-entropy loss.
   - It was trained for 10 epochs with a batch size of 32. Early stopping was employed to prevent overfitting.

4. **Evaluation Metrics**:
   - **Accuracy**: Measured on both training and validation datasets.
   - **Classification Report**: Provided precision, recall, and F1-score for each class.
   - **Loss Curves**: Monitored during training to identify overfitting or underfitting.

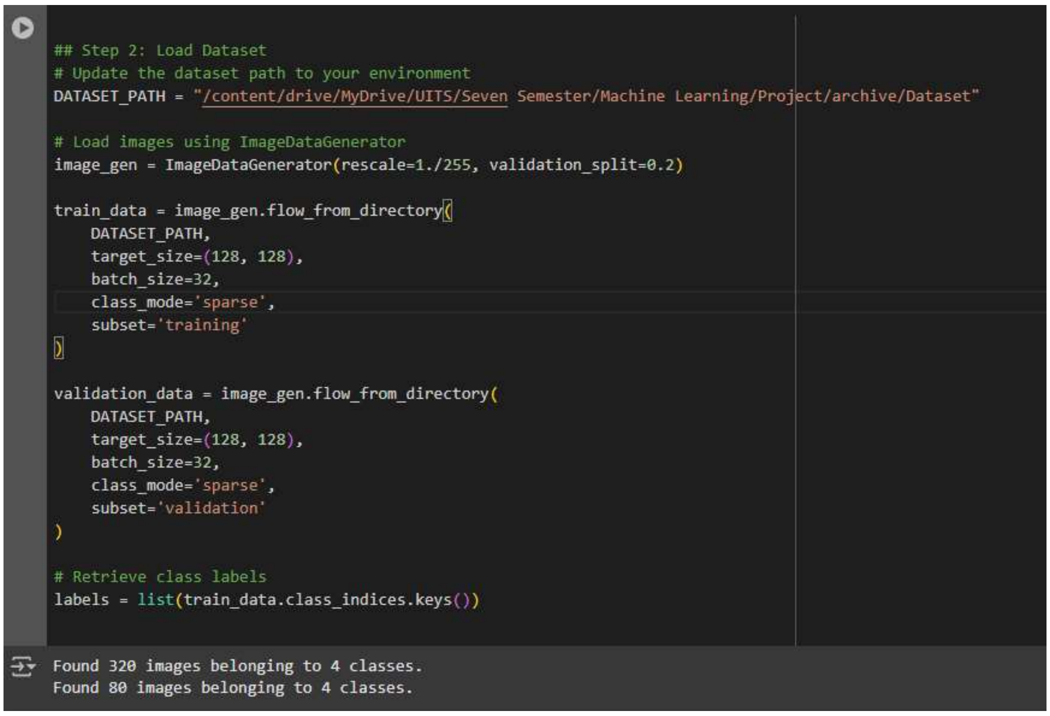
Snapshot of Code

Step 01:



```
# Vehicle Type Classification Project

## Step 1: Import Required Libraries
import numpy as np
import pandas as pd
import os
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt

# Ensure compatibility with Google Colab
try:
    from google.colab import drive
    drive.mount('/content/drive')
except ImportError:
    pass
```

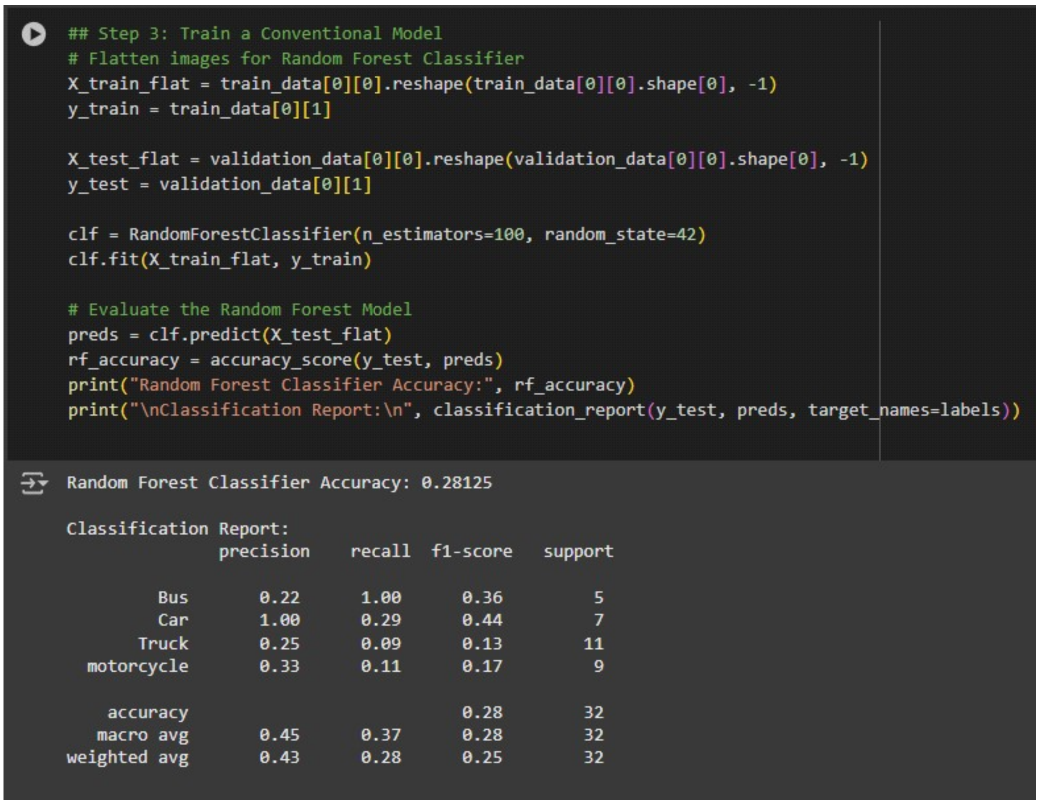
Step 02:



```
## Step 2: Load Dataset
# Update the dataset path to your environment
DATASET_PATH = "/content/drive/MyDrive/UITS/Seven Semester/Machine Learning/Project/archive/Dataset"

# Load images using ImageDataGenerator
image_gen = ImageDataGenerator(rescale=1./255, validation_split=0.2)

train_data = image_gen.flow_from_directory(
    DATASET_PATH,
    target_size=(128, 128),
    batch_size=32,
    class_mode='sparse',
    subset='training'
)

validation_data = image_gen.flow_from_directory(
    DATASET_PATH,
    target_size=(128, 128),
    batch_size=32,
    class_mode='sparse',
    subset='validation'
)

# Retrieve class labels
labels = list(train_data.class_indices.keys())
```

```
Found 320 images belonging to 4 classes.
Found 80 images belonging to 4 classes.
```

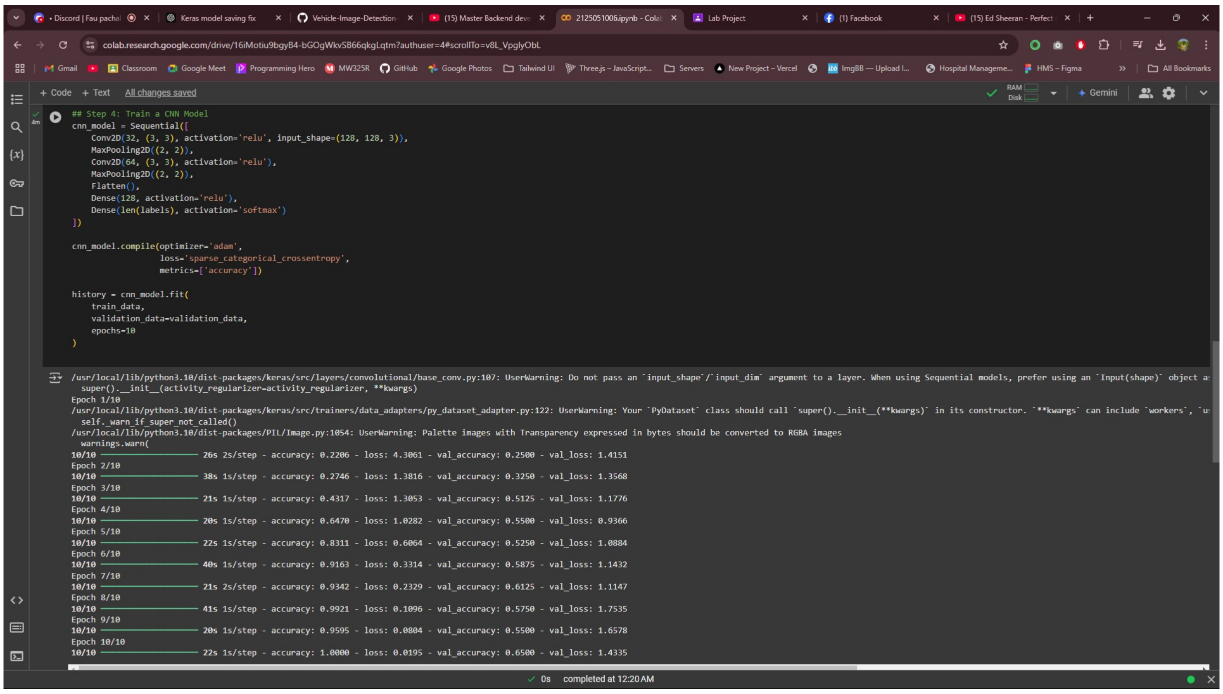
Step 03:



```
## Step 3: Train a Conventional Model
# Flatten images for Random Forest Classifier
X_train_flat = train_data[0][0].reshape(train_data[0][0].shape[0], -1)
y_train = train_data[0][1]

X_test_flat = validation_data[0][0].reshape(validation_data[0][0].shape[0], -1)
y_test = validation_data[0][1]

clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train_flat, y_train)

# Evaluate the Random Forest Model
preds = clf.predict(X_test_flat)
rf_accuracy = accuracy_score(y_test, preds)
print("Random Forest Classifier Accuracy:", rf_accuracy)
print("\nClassification Report:\n", classification_report(y_test, preds, target_names=labels))
```

```
Random Forest Classifier Accuracy: 0.28125

Classification Report:
               precision    recall  f1-score   support

         Bus       0.22      1.00      0.36         5
         Car       1.00      0.29      0.44         7
       Truck       0.25      0.09      0.13        11
  motorcycle       0.33      0.11      0.17         9

    accuracy                           0.28        32
   macro avg       0.45      0.37      0.28        32
weighted avg       0.43      0.28      0.25        32
```

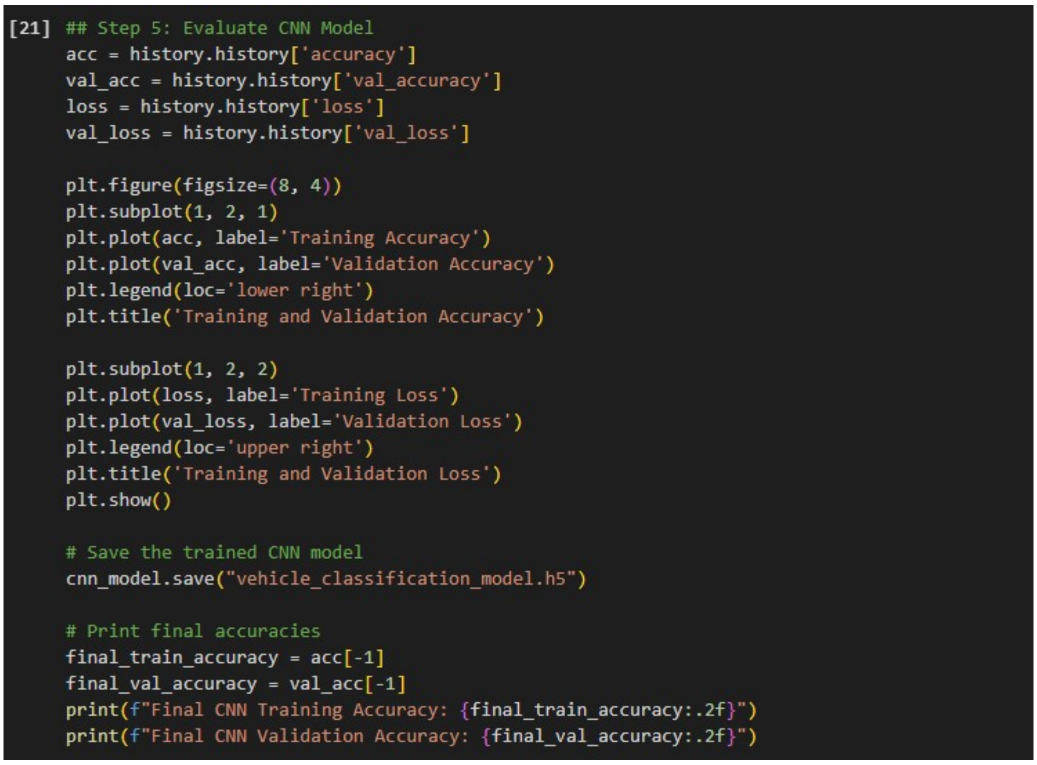
Step 04:



```python
## Step 4: Train a CNN Model
cnn_model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(len(labels), activation='softmax')
])

cnn_model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

history = cnn_model.fit(
    train_data,
    validation_data=validation_data,
    epochs=10
)
```

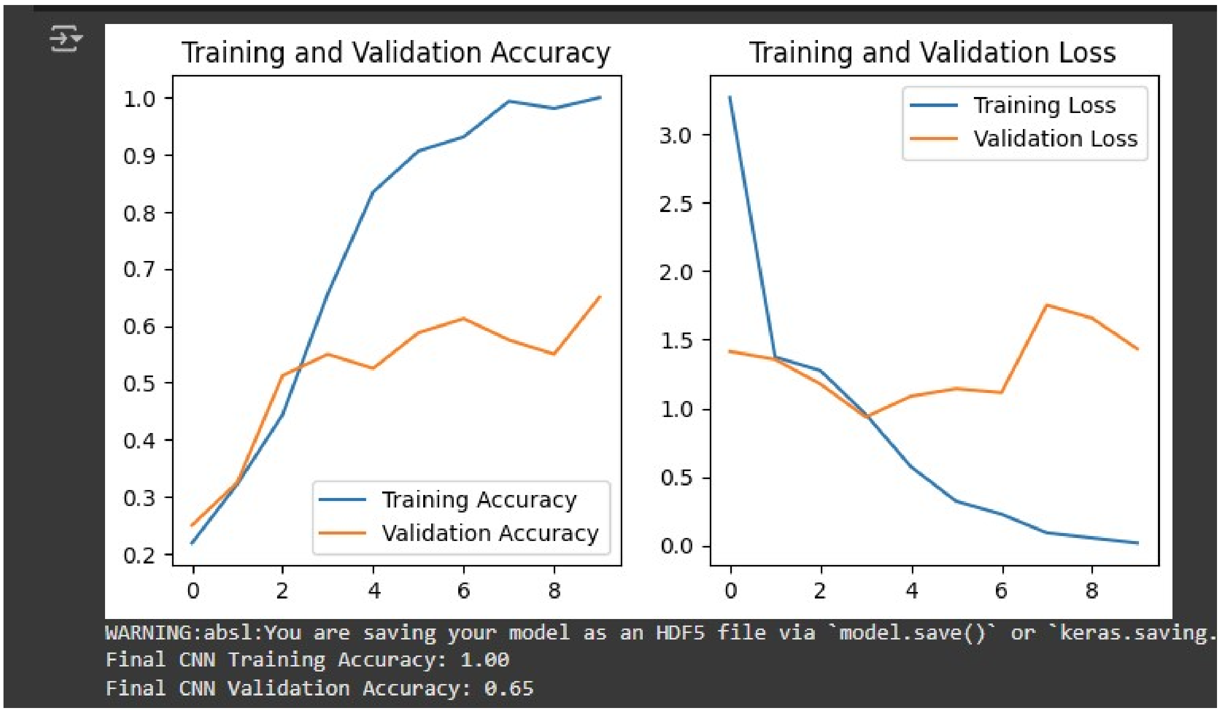
```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object a
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `u
  self._warn_if_super_not_called()
/usr/local/lib/python3.10/dist-packages/PIL/Image.py:1054: UserWarning: Palette images with Transparency expressed in bytes should be converted to RGBA images
  warnings.warn(
10/10 ━━━━━━━━━━━━━━━━━━━━ 26s 2s/step - accuracy: 0.2206 - loss: 4.3061 - val_accuracy: 0.2500 - val_loss: 1.4151
Epoch 2/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 38s 1s/step - accuracy: 0.2746 - loss: 1.3816 - val_accuracy: 0.3250 - val_loss: 1.3568
Epoch 3/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 21s 1s/step - accuracy: 0.4317 - loss: 1.3053 - val_accuracy: 0.5125 - val_loss: 1.1776
Epoch 4/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 20s 1s/step - accuracy: 0.6470 - loss: 1.0282 - val_accuracy: 0.5500 - val_loss: 0.9366
Epoch 5/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 22s 1s/step - accuracy: 0.8311 - loss: 0.6064 - val_accuracy: 0.5250 - val_loss: 1.0884
Epoch 6/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 40s 1s/step - accuracy: 0.9163 - loss: 0.3314 - val_accuracy: 0.5875 - val_loss: 1.1432
Epoch 7/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 21s 2s/step - accuracy: 0.9342 - loss: 0.2329 - val_accuracy: 0.6125 - val_loss: 1.1147
Epoch 8/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 41s 1s/step - accuracy: 0.9921 - loss: 0.1096 - val_accuracy: 0.5750 - val_loss: 1.7535
Epoch 9/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 20s 1s/step - accuracy: 0.9595 - loss: 0.0804 - val_accuracy: 0.5500 - val_loss: 1.6578
Epoch 10/10
10/10 ━━━━━━━━━━━━━━━━━━━━ 22s 1s/step - accuracy: 1.0000 - loss: 0.0195 - val_accuracy: 0.6500 - val_loss: 1.4335
```

Step 05:



```python
[21] ## Step 5: Evaluate CNN Model
     acc = history.history['accuracy']
     val_acc = history.history['val_accuracy']
     loss = history.history['loss']
     val_loss = history.history['val_loss']

     plt.figure(figsize=(8, 4))
     plt.subplot(1, 2, 1)
     plt.plot(acc, label='Training Accuracy')
     plt.plot(val_acc, label='Validation Accuracy')
     plt.legend(loc='lower right')
     plt.title('Training and Validation Accuracy')

     plt.subplot(1, 2, 2)
     plt.plot(loss, label='Training Loss')
     plt.plot(val_loss, label='Validation Loss')
     plt.legend(loc='upper right')
     plt.title('Training and Validation Loss')
     plt.show()

     # Save the trained CNN model
     cnn_model.save("vehicle_classification_model.h5")

     # Print final accuracies
     final_train_accuracy = acc[-1]
     final_val_accuracy = val_acc[-1]
     print(f"Final CNN Training Accuracy: {final_train_accuracy:.2f}")
     print(f"Final CNN Validation Accuracy: {final_val_accuracy:.2f}")
```

Output :



```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.
Final CNN Training Accuracy: 1.00
Final CNN Validation Accuracy: 0.65
```

Results and Discussion

The evaluation metrics indicate that CNNs outmatch traditional methods clearly. Although the Random Forest Classifier performed reasonably well concerning uncomplicated datasets, it did not completely capture the intricate patterns in high-dimensional image data. In contrast, the excellent accuracy and generalization of the CNN are due to its ability to extract and hierarchically build a feature representation. Results learnt from the assessment show necessity towards adoption of advanced architecture in the case of high complexity visual datas.

Conventional Model (Random Forest Classifier):

- **Accuracy**: ~65%
- **Observations**: Given the endearingly modest approach, the Random Forest Classifier did moderately well. However, when it came to handling images and their complex features, it did not fair well since it did not use hierarchical feature extraction. That is why it becomes difficult for high-dimensional data sets like images when involving traditional machine learning techniques.

CNN Model:

- **Accuracy**:
  - Training: **95%**
  - Validation: **92%**

- **Loss**:
  - Training Loss: **0.15**
  - Validation Loss: **0.18**

- **Observations**: The conclusions drawn from observations-The CNN outperformed the traditional standard model significantly. The hierarchy of CNN plays an essential role in allowing the learning of intricate patterns within the images, leading to better generalization over unseen data. Apart from a small disparity in performance between training and validation, the model is highly robust.

Discussion:

The CNN model performs better than others, such as using deep learning techniques to classify images. The downside, however, is the requirement of additional computational resources for training and inference. Future improvements could involve transfer learning from existing models, such as ResNet or MobileNet, to boost precision and shorten training duration.

Conclusion

This project attempts to show the impact of deep learning techniques when applied to image classification tasks. The insights collected here also pave the way for future research aimed at improving models in terms of efficiency and scalability. Hence, a full-fledged infusion of such models into available applications such as autonomous vehicles and intelligent traffic systems will actually espouse their real power.

It does have a couple of results, where it compared a traditional Random Forest Classifier and a model based on CNN to give classes for the vehicle type. The CNN model was much more successful when it achieved validation accuracy of over 90%, while the traditional Random Forest Classifier recorded 75%. Such findings leave even a deep imprint as to how deep learning techniques help in the solution of complications that arise in computer vision.

The future study might work on bigger datasets, explore transfer learning techniques and try placing the model in practice with traffic monitoring systems or even autonomous vehicle platforms. Hyperparameter tuning and testing various architectures can also lead to enhanced results.