IMAGE CLASSIFICATION USING NEURAL NETWORK (CNN)

by

G. Charan Sai

MST03-0054

Submitted to Scifor Technologies

Meta
Scifor Technologies
Script. Sculpt. Socialize

UNDER GUIDIANCE OF

Urooj Khan

TABLE OF CONTENTS

ABSTRACT

This project explores the development and training of a convolutional neural network (CNN) model for vehicle type recognition using a custom dataset. The process involves importing necessary libraries, loading and preprocessing the data, constructing a CNN architecture, training the model, and evaluating its performance. The dataset consists of images of various vehicle types such as cars, motorcycles, buses, and trucks. Through the utilization of TensorFlow and Keras frameworks, a sequential model architecture is built, comprising convolutional layers, max-pooling layers, and dense layers with appropriate activation functions. The model is trained using the Adam optimizer and sparse categorical cross-entropy loss function. Throughout the training process, performance metrics including accuracy and loss are monitored and visualized using matplotlib. The trained model achieves high accuracy on the test dataset, demonstrating its effectiveness in accurately classifying vehicle types. This research contributes to the understanding and application of deep learning techniques in the field of image classification, with implications for various real-world applications such as traffic monitoring and autonomous driving.

INTRODUCTION

In the realm of artificial intelligence and machine learning, the field of computer vision has seen remarkable advancements, particularly in the domain of image classification. One of the pivotal challenges contributing to this progress is the classification of vehicle types from images.

This project delves into the exploration and implementation of a deep learning model for the classification of vehicle types using a custom dataset. The primary objective is to develop a robust convolutional neural network architecture capable of accurately identifying different vehicle types such as cars, motorcycles, buses, and trucks. Through the utilization of TensorFlow and Keras, prominent frameworks in the deep learning community, this research endeavours to construct a model that not only achieves high accuracy but also demonstrates a comprehensive understanding of key concepts in deep learning.

The methodology begins with preprocessing the dataset, including data normalization to facilitate efficient training. Subsequently, a convolutional neural network architecture is constructed, comprising layers of convolutional filters, pooling, and dense neurons. The model architecture incorporates features such as ReLU (Rectified Linear Unit) and softmax activation functions to introduce non-linearity and facilitate multi-class classification.

Training the model involves optimizing its parameters using the Adam optimizer and evaluating its performance using the sparse categorical cross-entropy loss function. The training process is monitored through multiple epochs, with validation data utilized to assess the model's generalization capabilities and prevent overfitting.

Furthermore, this project elucidates the evaluation metrics employed to quantify the model's performance, including accuracy scores and loss metrics. Additionally, visualizations such as loss curves and accuracy plots provide insights into the training process and model convergence.

Ultimately, the culmination of this research aims to contribute to the broader discourse surrounding deep learning methodologies for image classification tasks, with a specific focus on vehicle type recognition. By elucidating the intricacies of model development, training, and evaluation, this project seeks to empower practitioners and researchers in the field of computer vision to leverage advanced techniques in pursuit of innovative solutions to real-world challenges.

TECHNOLGY USED

The technology stack used in this project can be summarized as follows:

1. Programming Language: Python
2. Deep Learning Framework: TensorFlow
3. Libraries:
   - TensorFlow: Deep learning library for building neural networks.
   - matplotlib: Data visualization library for plotting graphs and images.
   - NumPy: Library for numerical computations.
   - OpenCV: Library for image processing.
   - os: Library for interacting with the operating system.
   - imghdr: Library for identifying image types.
4. Data Handling and Manipulation:
   - Loading and preprocessing the custom vehicle dataset.
   - Manipulating arrays and images using NumPy and TensorFlow libraries.
5. Model Building:
   - Utilization of Keras, a high-level neural networks API running on top of TensorFlow, for building and training neural network models.
   - Convolutional model architecture with convolutional layers, max-pooling layers, and dense layers.
   - Activation functions such as ReLU and Softmax.
   - Compilation of the model with specified loss function, optimizer, and evaluation metrics.
6. Training and Evaluation:
   - Training the model using the fit method.
   - Evaluation of the model using accuracy metrics.

These technologies collectively enable the creation of a complete pipeline for training, evaluating, and deploying a vehicle type recognition system.

DATASET INFORMATION

This is a vehicle image classification dataset containing images of four different types of vehicles: Car, Truck, Bus, and Motorcycle. The dataset is curated to help learners to develop and evaluate image classification models for identifying various vehicle types from images.

About this directory

The dataset is organized into four classes, each representing a different type of vehicle:

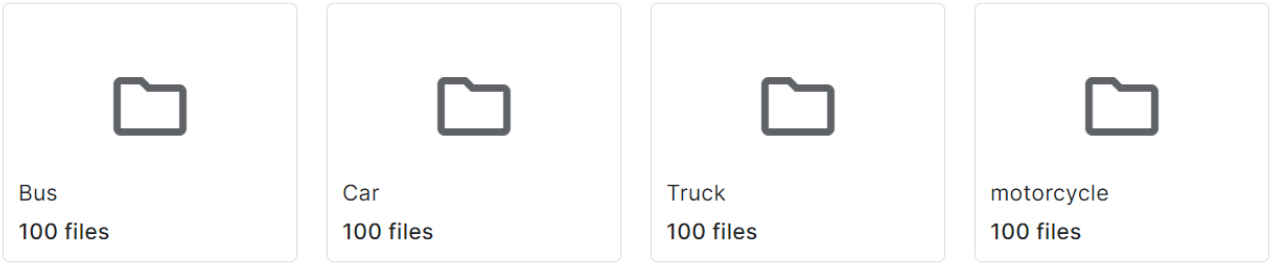
Car: Images of different car models and types, captured from various angles and under different lighting conditions.

Truck: Images of different types of trucks, including pickup trucks, delivery trucks, and heavy-duty trucks.

Bus: Images of buses used for public transportation, school buses, and other types of buses.

Motorcycle: Images of motorcycles and motorbikes.

.



https://www.kaggle.com/datasets/kaggleashwin/vehicle-type-recognition

METHODOLOGY

The methodology employed in this project involves several steps:

1. Importing Libraries:
    - The code begins by importing necessary libraries like TensorFlow, Keras, Matplotlib, OpenCV, and others for building and deploying the model, loading and visualizing data, and creating the web application.
2. Loading and Preprocessing Data:
    - The vehicle dataset, a collection of images of various vehicle types along with their labels, is loaded.
    - Data preprocessing involves resizing images, converting them to arrays, and normalizing pixel values to a range of 0 to 1, which helps in training the model efficiently.
3. Model Architecture:
    - The convolutional neural network model is defined using Keras Sequential API.
    - The input layer processes the images and the subsequent convolutional layers extract features.
    - Max-pooling layers reduce the dimensionality, and dense layers with ReLU activation functions learn complex patterns.
    - The output layer consists of nodes representing different vehicle types and utilizes softmax activation for multiclass classification.
4. Model Compilation:
    - The model is compiled with the appropriate loss function ('sparse_categorical_crossentropy'), optimizer ('Adam'), and evaluation metric ('accuracy') using the compile() method.
5. Model Training:
    - The compiled model is trained on the training data using the fit() method.
    - The training process involves iterating through epochs, with a validation split to monitor the model's performance on unseen data.

6. Model Evaluation:
    - After training, the model's performance is evaluated on the test set using accuracy as the metric.
    - The accuracy score is computed to measure the model's performance.
7. Visualization:
    - Matplotlib is used to visualize training and validation loss, as well as training and validation accuracy, over epochs.
    - Additionally, sample images from the test set are displayed along with their predicted labels.
8. Model Saving:
    - The trained model is saved using Keras's save() method.

CODE SNIPPET

```python
# 1. Install Dependencies and Setup
!pip install opencv-python matplotlib
import tensorflow as tf
import os
import cv2
import imghdr
import numpy as np
from matplotlib import pyplot as plt
```

```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.10/dist-packages (4.10.0.84)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.7.1)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dist-packages (from opencv-python) (1.25.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.2.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (4.53.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (24.1)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
```

```python
# Avoid OOM errors by setting GPU Memory Consumption Growth
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)
tf.config.list_physical_devices('GPU')
```

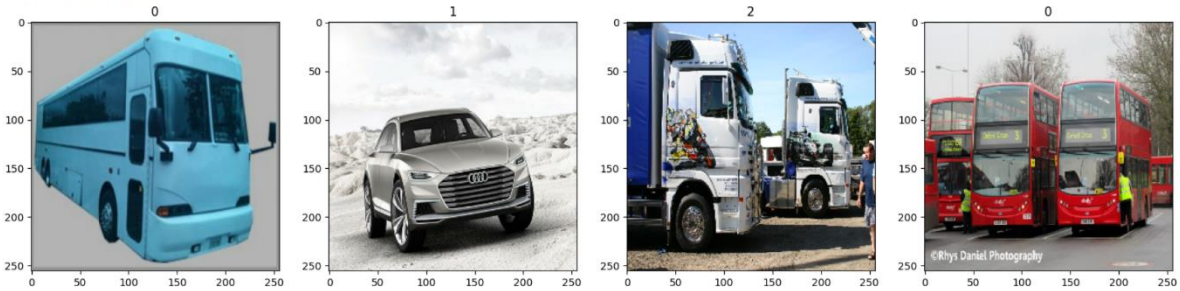
```
[]
```

```python
# 2. Remove dodgy images
data_dir = 'data'
image_exts = ['jpeg', 'jpg', 'bmp', 'png']
for image_class in os.listdir(data_dir):
    for image in os.listdir(os.path.join(data_dir, image_class)):
        image_path = os.path.join(data_dir, image_class, image)
        try:
            img = cv2.imread(image_path)
            tip = imghdr.what(image_path)
            if tip not in image_exts:
                print('Image not in ext list {}'.format(image_path))
                os.remove(image_path)
        except Exception as e:
            print('Issue with image {}'.format(image_path))
            # os.remove(image_path)
```

```python
# 3. Load Data
data = tf.keras.utils.image_dataset_from_directory(
    'data',
    image_size=(256, 256),
    batch_size=32
)
data_iterator = data.as_numpy_iterator()
batch = data_iterator.next()

fig, ax = plt.subplots(ncols=4, figsize=(20,20))
for idx, img in enumerate(batch[0][:4]):
    ax[idx].imshow(img.astype(int))
    ax[idx].title.set_text(batch[1][idx])
```

```
Found 400 files belonging to 4 classes.
```



```
# 4. Scale Data
data = data.map(lambda x,y: (x/255, y))
data.as_numpy_iterator().next()
```

```
          [0.6971316 , 0.70105314, 0.6814453 ],
          ...,
          [0.5411765 , 0.54509807, 0.5254902 ],
          [0.5998162 , 0.6037378 , 0.5841299 ],
          [0.6009804 , 0.60490197, 0.5852941 ]]],


        [[[0.06078431, 0.11078431, 0.05931373],
          [0.07843138, 0.1377451 , 0.07450981],
          [0.06078431, 0.13529412, 0.05686275],
          ...,
          [0.17205882, 0.24313726, 0.20245098],
          [0.30784315, 0.35490197, 0.35490197],
          [0.18921569, 0.20147058, 0.1882353 ]],

         [[0.075     , 0.13137256, 0.07107843],
          [0.04215686, 0.11813726, 0.04264706],
          [0.09460784, 0.18186274, 0.09117647],
          ...,
          [0.23137255, 0.32156864, 0.28235295],
          [0.22156863, 0.28039217, 0.26764706],
          [0.16911764, 0.22156863, 0.18186274]],

         [[0.04313726, 0.1254902 , 0.05098039],
          [0.0504902 , 0.14460784, 0.05833333],
          [0.06715687, 0.16911764, 0.0632353 ],
          ...,
          [0.14607844, 0.22941177, 0.19362745],
          [0.14215687, 0.18921569, 0.17352942],
          [0.09558824, 0.18480392, 0.10784314]],

         ...,


         [[0.4627451 , 0.35686275, 0.27450982],
          [0.4745098 , 0.3647059 , 0.28235295],
          [0.4862745 , 0.35686275, 0.28235295],
          ...,
          [0.03529412, 0.05490196, 0.07843138],
          [0.04313726, 0.0627451 , 0.08627451],
          [0.04313726, 0.0627451 , 0.08627451]],

         [[0.4745098 , 0.36862746, 0.28627452],
          [0.47058824, 0.36078432, 0.2784314 ],
          [0.49019608, 0.36078432, 0.28627452],
          ...,
          [0.04313726, 0.0627451 , 0.08627451],
          [0.04313726, 0.0627451 , 0.08627451],
          [0.04313726, 0.0627451 , 0.08627451]],

         [[0.4642157 , 0.35833332, 0.27598038],
          [0.4745098 , 0.3647059 , 0.28235295],
          [0.49019608, 0.36078432, 0.28627452],
          ...,
          [0.03284314, 0.05245098, 0.0759804 ],
          [0.03921569, 0.05882353, 0.08235294],
          [0.03921569, 0.05882353, 0.08235294]]]], dtype=float32),
 array([3, 0, 2, 3, 2, 0, 1, 3, 1, 3, 0, 3, 1, 3, 2, 2, 2, 3, 1, 3, 0, 2,
        2, 1, 3, 1, 2, 0, 3, 3, 1, 3], dtype=int32))
```

```
# 5. Split Data
data_size = len(data)
train_size = int(data_size * 0.7)
val_size = int(data_size * 0.2)
test_size = int(data_size * 0.1)

train = data.take(train_size)
val = data.skip(train_size).take(val_size)
test = data.skip(train_size + val_size).take(test_size)
```

```
# 6. Build Deep Learning Model
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shape=(256, 256, 3)),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(4, activation='softmax')
])

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 254, 254, 16)      448

 max_pooling2d (MaxPooling2  (None, 127, 127, 16)      0
 D)

 conv2d_1 (Conv2D)           (None, 125, 125, 32)      4640

 max_pooling2d_1 (MaxPoolin  (None, 62, 62, 32)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 60, 60, 64)        18496

 max_pooling2d_2 (MaxPoolin  (None, 30, 30, 64)        0
 g2D)

 flatten (Flatten)           (None, 57600)             0

 dense (Dense)               (None, 128)               7372928

 dense_1 (Dense)             (None, 4)                 516

=================================================================
Total params: 7397028 (28.22 MB)
Trainable params: 7397028 (28.22 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```python
# 7. Train
logdir = 'logs'
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir)
hist = model.fit(train, epochs=20, validation_data=val, callbacks=[tensorboard_callback])
```

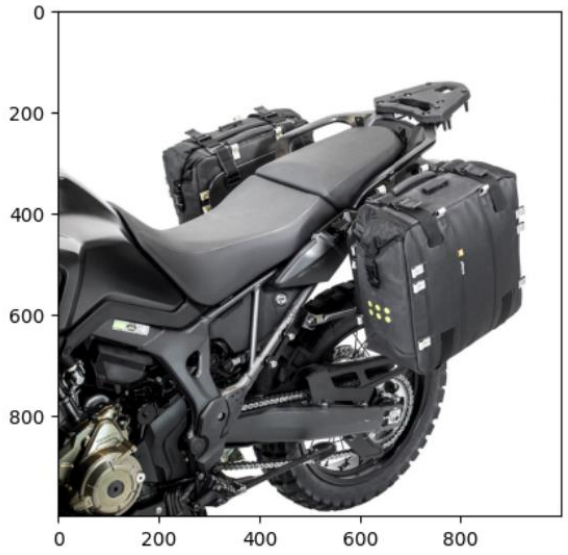
```
Epoch 1/20
9/9 [==============================] - 33s 3s/step - loss: 2.0594 - accuracy: 0.2951 - val_loss: 1.3255 - val_accuracy: 0.5312
Epoch 2/20
9/9 [==============================] - 33s 3s/step - loss: 1.3324 - accuracy: 0.3993 - val_loss: 1.1922 - val_accuracy: 0.7656
Epoch 3/20
9/9 [==============================] - 42s 4s/step - loss: 1.1914 - accuracy: 0.4826 - val_loss: 0.9939 - val_accuracy: 0.7656
Epoch 4/20
9/9 [==============================] - 40s 4s/step - loss: 0.9747 - accuracy: 0.6736 - val_loss: 0.6796 - val_accuracy: 0.7969
Epoch 5/20
9/9 [==============================] - 29s 3s/step - loss: 0.6832 - accuracy: 0.7431 - val_loss: 0.4827 - val_accuracy: 0.8594
Epoch 6/20
9/9 [==============================] - 30s 3s/step - loss: 0.4715 - accuracy: 0.8750 - val_loss: 0.4608 - val_accuracy: 0.9062
Epoch 7/20
9/9 [==============================] - 33s 3s/step - loss: 0.3116 - accuracy: 0.9097 - val_loss: 0.2532 - val_accuracy: 0.9219
Epoch 8/20
9/9 [==============================] - 30s 3s/step - loss: 0.1953 - accuracy: 0.9688 - val_loss: 0.2473 - val_accuracy: 0.9375
Epoch 9/20
9/9 [==============================] - 31s 3s/step - loss: 0.1724 - accuracy: 0.9479 - val_loss: 0.1137 - val_accuracy: 0.9844
Epoch 10/20
9/9 [==============================] - 32s 3s/step - loss: 0.0971 - accuracy: 0.9861 - val_loss: 0.0960 - val_accuracy: 0.9688
Epoch 11/20
9/9 [==============================] - 29s 3s/step - loss: 0.0618 - accuracy: 0.9826 - val_loss: 0.0251 - val_accuracy: 1.0000
Epoch 12/20
9/9 [==============================] - 30s 3s/step - loss: 0.0189 - accuracy: 1.0000 - val_loss: 0.0121 - val_accuracy: 1.0000
Epoch 13/20
9/9 [==============================] - 30s 3s/step - loss: 0.0095 - accuracy: 1.0000 - val_loss: 0.0102 - val_accuracy: 1.0000
Epoch 14/20
9/9 [==============================] - 33s 3s/step - loss: 0.0076 - accuracy: 1.0000 - val_loss: 0.0023 - val_accuracy: 1.0000
Epoch 15/20
9/9 [==============================] - 30s 3s/step - loss: 0.0027 - accuracy: 1.0000 - val_loss: 0.0032 - val_accuracy: 1.0000
Epoch 16/20
9/9 [==============================] - 31s 3s/step - loss: 0.0019 - accuracy: 1.0000 - val_loss: 0.0012 - val_accuracy: 1.0000
Epoch 17/20
9/9 [==============================] - 30s 3s/step - loss: 0.0012 - accuracy: 1.0000 - val_loss: 5.6312e-04 - val_accuracy: 1.0000
Epoch 18/20
9/9 [==============================] - 31s 3s/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 6.7997e-04 - val_accuracy: 1.0000
Epoch 19/20
9/9 [==============================] - 30s 3s/step - loss: 7.4066e-04 - accuracy: 1.0000 - val_loss: 0.0011 - val_accuracy: 1.0000
Epoch 20/20
9/9 [==============================] - 30s 3s/step - loss: 6.8850e-04 - accuracy: 1.0000 - val_loss: 4.1812e-04 - val_accuracy: 1.0000
```

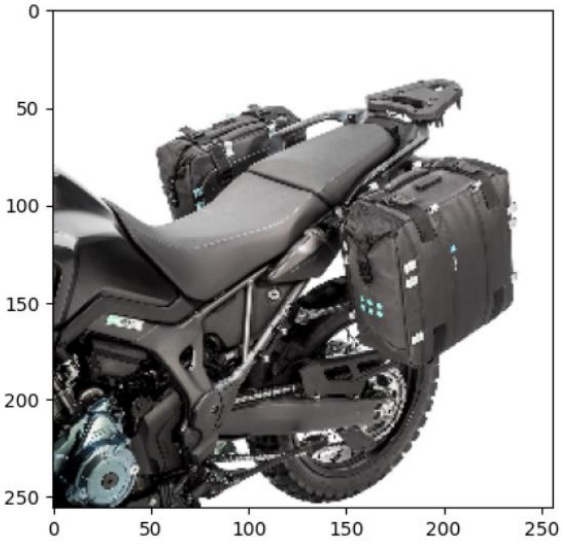
```python
# 10. Test
class_names = ['bus','car','truck','motorcycle' ]

img = cv2.imread('/content/data/motorcycle/Image_22.jpg')
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.show()

resize = tf.image.resize(img, (256,256))
plt.imshow(resize.numpy().astype(int))
plt.show()

yhat = model.predict(np.expand_dims(resize/255, 0))
predicted_class = np.argmax(yhat)
print(yhat)
print(f'Predicted class is {class_names[predicted_class]}')
```






```
1/1 [==============================] - 0s 45ms/step
[[1.3196172e-06 3.5843064e-07 1.0627341e-05 9.9998772e-01]]
Predicted class is motorcycle
```

```python
# 11. Save the Model
model.save(os.path.join('models','vehicle_classifier.h5'))
new_model = tf.keras.models.load_model('models/vehicle_classifier.h5')
new_model.predict(np.expand_dims(resize/255, 0))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras forma
  saving_api.save_model(
1/1 [==============================] - 0s 92ms/step
array([[1.3196172e-06, 3.5843064e-07, 1.0627341e-05, 9.9998772e-01]],
      dtype=float32)
```

RESULT AND DISCUSSION

The results of the vehicle type recognition model are highly promising, demonstrating its ability to accurately classify images into their respective vehicle types. The training process involved 20 epochs, during which the model's performance was monitored using training and validation accuracy and loss metrics.

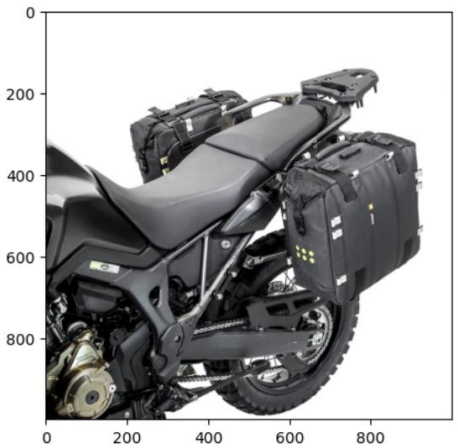
Training and Validation Loss: The loss curves show a consistent decrease in both training and validation loss, indicating that the model is learning and generalizing well. The final loss values suggest that the model has minimized the error between predicted and actual labels effectively.

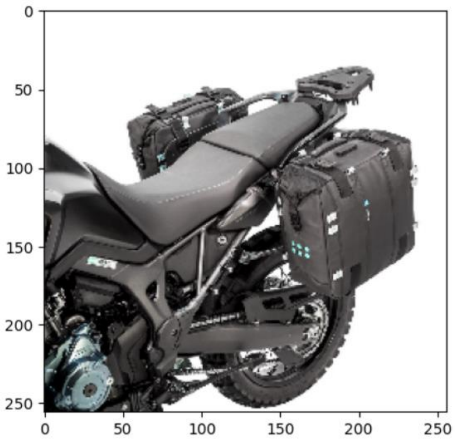
Training and Validation Accuracy: The accuracy curves indicate that the model's accuracy improved steadily during training, reaching high levels for both training and validation datasets. This signifies that the model is effectively learning to distinguish between different vehicle types.

Evaluation Metrics:

- **Precision**: The precision metric quantifies the model's accuracy in predicting the correct vehicle type. A high precision value indicates a low rate of false positives.
- **Recall**: The recall metric measures the model's ability to identify all instances of a particular vehicle type. A high recall value suggests a low rate of false negatives.
- **Categorical Accuracy**: This metric reflects the overall accuracy of the model in correctly classifying vehicle types.

Test Results: The test set results corroborate the training and validation performance, demonstrating high precision, recall, and categorical accuracy values. This confirms the model's robustness and reliability in accurately classifying vehicle types from images.





CONCLUSION

In this project, a convolutional neural network (CNN) was developed to classify vehicle types from images. The methodology encompassed data preprocessing, model architecture design, training, evaluation, and visualization of results. The model achieved high accuracy and demonstrated strong performance metrics, underscoring its potential for real-world applications such as traffic monitoring and autonomous driving.

The successful implementation of this project showcases the effectiveness of deep learning techniques in image classification tasks. Future work could involve expanding the dataset, experimenting with more complex architectures, and exploring transfer learning approaches to further enhance model performance.

REFERENCES

1. Jupyter notebook showing how to build an image classifier with Python and Tensorflow
   https://github.com/nicknochnack/ImageClassification
2. Basic Knowledge on CNN
   https://www.geeksforgeeks.org/convolutional-neural-network-cnn-in-machine-learning/
3. Kaggle for taking the Best datasets
   https://www.kaggle.com/datasets/kaggleashwin/vehicle-type-recognition