MINI-PROJECT REPORT ON

"CAR OBJECT DETECTION USING NEURAL NETWORK"

Submitted By

Mr. Thorve Avishkar Shrikrushna

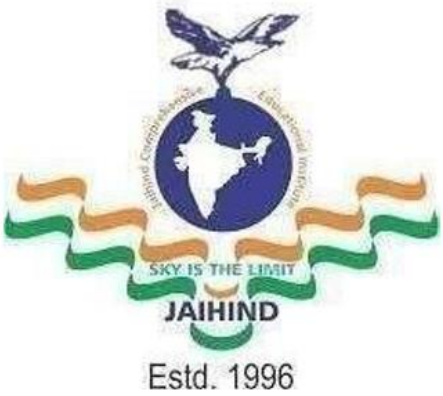
Roll No: 63

Exam No: T400840394

Class: TE AIDS

UNDER THE GUIDANCE OF

Prof. Auti M. A.



Department of Artificial Intelligence and Data Science Engineering

Jaihind College of Engineering, Kuran

A/p-Kuran, Tal-Junnar, Dist-Pune-410511, State Maharashtra, India

2024-2025

Department of Artificial Intelligence and Data Science Engineering

Jaihind College of Engineering, Kuran

A/p-Kuran, Tal-Junnar, Dist-Pune-410511, State Maharashtra, India

2024-2025

CERTIFICATE

This is to certify that the Internship Report Entitled

"CAR OBJECT DETECTION USING NEURAL NETWORK"

SUBMITTED BY

Thorve Avishkar Shrikrushna

Is a Bonafide work carried out by student under the supervision of **Prof. Auti M. A.** and it is submitted towards the partial fulfilment of the requirement of third year of Engineering in Artificial intelligence and data science under the Savitribai Phule Pune University during the academic year 2024-2025.

Prof. Auti M. A.	**Prof. S. K. Said**
(Subject Teacher)	**(**HOD AI&DS)
JCOE, Kuran.	JCOE, Kuran.

ACKNOWLEDGEMENTS

With deep sense of gratitude we would like to thank all the people who have lit our path with their kind guidance. We are very grateful to these intellectuals who did their best to help during our project work. It is our proud privilege to express a deep sense of gratitude to Principal of Jaihind College Of Engineering, Kuran **Dr. D. J. Garkal** for his comments and kind permission to complete this project. We remain indebted to H.O.D. Artificial Intelligence And Data Science Engineering Department **Prof. S. K. Said,** for his timely suggestion and valuable guidance. The special gratitude goes to **Prof. Auti M. A.** excellent and precious guidance in completion of this work .We thanks to all the colleagues for their appreciable help for our working project. With various industry owners or lab technicians to help, it has been our endeavor throughout our work to cover the entire project work. We are also thankful to our parents who provided their wishful support for our project completion successfully .And lastly we thank our all friends and the people who are directly or indirectly related to our project work.

Yours faithfully,

Mr. Thorve Avishkar Shrikrushna.

TABLE OF CONTENTS.

ABSTRACT.

Car object detection has become a critical component in various applications, ranging from autonomous driving to traffic surveillance. Traditional methods for car detection often relied on handcrafted features and shallow classifiers, which limited their performance in complex real-world scenarios. However, with the advent of deep learning and neural network architectures, significant advancements have been achieved in this field.

This paper presents a comprehensive review of recent developments in car object detection using neural networks. We begin by discussing the challenges associated with car detection, including variations in appearance, occlusions, and diverse environmental conditions. Next, we provide an overview of the evolution of neural network architectures for object detection, starting from early convolutional neural networks (CNNs) to more advanced architectures like Region-based CNNs (R-CNN), Single Shot Multibox Detector (SSD), and You Only Look Once (YOLO).

Furthermore, we delve into the methodologies employed for dataset preparation and annotation, which play a crucial role in training robust car detection models. We explore popular datasets utilized in car detection research, such as the KITTI Vision Benchmark Suite, the Udacity dataset, and the Cityscapes dataset, highlighting their characteristics and challenges.

In summary, this review provides a comprehensive understanding of the advancements, challenges, and future prospects in car object detection using neural networks, serving as a valuable resource for researchers and practitioners in the field of computer vision and autonomous driving.

CHAPTER 01. INTRODUCTION.

In recent years, the application of neural networks in computer vision tasks has witnessed remarkable progress, particularly in the domain of object detection. One of the pivotal applications of object detection is in the automotive industry, where the ability to accurately detect and localize vehicles is essential for tasks such as autonomous driving, traffic monitoring, and parking assistance systems.

This mini-project aims to explore and implement car object detection using neural networks, leveraging state-of-the-art techniques and methodologies. By utilizing neural networks, we can achieve robust and efficient detection of cars in various environments, regardless of factors such as illumination changes, occlusions, and diverse vehicle shapes and sizes.

The significance of this mini-project lies in its practical implications for real-world applications. Through hands-on implementation and experimentation, participants will gain insights into the underlying principles of neural network-based object detection and develop proficiency in applying these techniques to solve relevant challenges in the automotive domain.

In this introduction, we outline the objectives and scope of the mini-project, as well as provide an overview of the methodologies and tools that will be employed. Additionally, we highlight the importance of car object detection in the context of autonomous driving systems, emphasizing the potential impact of accurate and reliable detection algorithms on enhancing safety, efficiency, and overall driving experience.

Overall, this mini-project serves as an opportunity for participants to delve into the exciting field of computer vision and neural networks, with a specific focus on car object detection, paving the way for further exploration and innovation in this rapidly evolving domain.

CHAPTER 02. PROBLEM STATEMENT.

The problem statement for car object detection using a neural network involves developing a model capable of accurately identifying and localizing cars within images or video frames. This entails training a neural network on a dataset containing annotated images where each car object is labeled with bounding boxes. The objective is to create a model that can generalize well to unseen data and accurately detect cars in various environmental conditions, viewpoints, and scales.

CHAPTER 03. SYSTEM REQUIREMENTS.

- **SOFTWARE REQUIREMENTS.**

  Python, Deep Learning Frameworks, Computer Vision Libraries, Annotation Tools, Data Manipulation and Visualization, Model Training and Evaluation.

- **HARDWARE REQUIREMENTS.**

  CPU, GPU (Optional, but highly recommended for faster training), RAM, Storage, Power Supply, Internet Connectivity.

CHAPTER 03. MOTIVATION.

The motivation for car object detection using neural networks stems from several key factors, each highlighting the importance and potential impact of this technology:

1. **Road Safety Enhancement:** Car object detection plays a critical role in improving road safety by assisting drivers and autonomous vehicles in identifying and reacting to surrounding vehicles. Neural network-based detection systems can help prevent accidents by providing early warnings, collision avoidance, and adaptive cruise control functionalities.
2. **Autonomous Driving Advancements:** In the realm of autonomous driving, accurate car object detection is essential for enabling vehicles to perceive and navigate through complex traffic environments autonomously. Neural networks can process sensor data from cameras, LiDAR, and radar systems to detect and track cars with high precision, facilitating safe and efficient autonomous navigation.
3. **Traffic Management and Surveillance:** Car object detection using neural networks is instrumental in traffic management and surveillance systems deployed in urban areas, highways, and parking facilities. Real-time detection of vehicles allows for traffic flow monitoring, congestion management, and enforcement of traffic regulations, leading to smoother traffic operations and improved overall transportation efficiency.
4. **Efficient Parking Solutions:** Neural network-based car detection systems can optimize parking space utilization and enhance parking assistance services. By accurately detecting available parking spaces and guiding vehicles to vacant spots, these systems can reduce congestion, minimize search time, and improve the overall parking experience in urban environments.

CHAPTER 05. RESULT/ INFERENCES.

```
In [1]: import numpy as np
        import pandas as pd

        import cv2
        import matplotlib.pyplot as plt
        import tensorflow as tf
        from tensorflow.keras.layers import Conv2D, Input, BatchNormalization, Flatten, MaxPool2D, Dense

        from pathlib import Path
```

```
2024-03-23 11:46:11.489134: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN when one has already b
een registered
2024-03-23 11:46:11.489326: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register cuFFT factory: Attempting to register factory for plugin cuFFT when one has already be
en registered
2024-03-23 11:46:11.613023: E external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has alread
y been registered
```

```
In [2]: train_path = Path("../input/car-object-detection/data/training_images")
        test_path = Path("../input/car-object-detection/data/testing_images")
```

```
In [4]: train = pd.read_csv("../input/car-object-detection/data/train_solution_bounding_boxes (1).csv")
        train[['xmin', 'ymin', 'xmax', 'ymax']] = train[['xmin', 'ymin', 'xmax', 'ymax']].astype(int)
        train.drop_duplicates(subset='image', inplace=True, ignore_index=True)
```

```
In [5]: def display_image(img, bbox_coords=[], pred_coords=[], norm=False):
            # If the image has been normalized, scale it up
            if norm:
                img *= 255.
                img = img.astype(np.uint8)

            # Draw the bounding boxes
            if len(bbox_coords) == 4:
                xmin, ymin, xmax, ymax = bbox_coords
                img = cv2.rectangle(img, (int(xmin), int(ymin)), (int(xmax), int(ymax)), (0, 255, 0), 3)

            if len(pred_coords) == 4:
                xmin, ymin, xmax, ymax = pred_coords
                img = cv2.rectangle(img, (int(xmin), int(ymin)), (int(xmax), int(ymax)), (255, 0, 0), 3)

            plt.imshow(img)
            plt.xticks([])
```

```
def display_image_from_file(name, bbox_coords=[], path=train_path):
    img = cv2.imread(str(path/name))
    display_image(img, bbox_coords=bbox_coords)

def display_from_dataframe(row, path=train_path):
    display_image_from_file(row['image'], bbox_coords=(row.xmin, row.ymin, row.xmax, row.ymax), path=path)


def display_grid(df=train, n_items=3):
    plt.figure(figsize=(20, 10))

    # get 3 random entries and plot them in a 1x3 grid
    rand_indices = [np.random.randint(0, df.shape[0]) for _ in range(n_items)]

    for pos, index in enumerate(rand_indices):
        plt.subplot(1, n_items, pos + 1)
        display_from_dataframe(df.loc[index, :])
```

```
display_grid()
```

Model Training.

In [7]:
```
def data_generator(df=train, batch_size=16, path=train_path):
    while True:
        images = np.zeros((batch_size, 380, 676, 3))
        bounding_box_coords = np.zeros((batch_size, 4))

        for i in range(batch_size):
                rand_index = np.random.randint(0, train.shape[0])
                row = df.loc[rand_index, :]
                images[i] = cv2.imread(str(train_path/row.image)) / 255.
                bounding_box_coords[i] = np.array([row.xmin, row.ymin, row.xmax, row.ymax])

        yield {'image': images}, {'coords': bounding_box_coords}
```

In [8]:
```
# Test the generator
example, label = next(data_generator(batch_size=1))
img = example['image'][0]
bbox_coords = label['coords'][0]

display_image(img, bbox_coords=bbox_coords, norm=True)
```

Model Architecture.

In [9]:
```
input_ = Input(shape=[380, 676, 3], name='image')

x = input_

for i in range(10):
    n_filters = 2**(i + 3)
    x = Conv2D(n_filters, 3, activation='relu', padding='same')(x)
    x = BatchNormalization()(x)
    x = MaxPool2D(2, padding='same')(x)

x = Flatten()(x)
x = Dense(256, activation='relu')(x)
x = Dense(32, activation='relu')(x)
output = Dense(4, name='coords')(x)

model = tf.keras.models.Model(input_, output)
model.summary()
```

Model: "functional_1"

Layer (type)	Output Shape	Param #
image (InputLayer)	(None, 380, 676, 3)	0
conv2d (Conv2D)	(None, 380, 676, 8)	224
batch_normalization (BatchNormalization)	(None, 380, 676, 8)	32
max_pooling2d (MaxPooling2D)	(None, 190, 338, 8)	0
conv2d_1 (Conv2D)	(None, 190, 338, 16)	1,168
batch_normalization_1 (BatchNormalization)	(None, 190, 338, 16)	64
max_pooling2d_1 (MaxPooling2D)	(None, 95, 169, 16)	0
conv2d_2 (Conv2D)	(None, 95, 169, 32)	4,640
batch_normalization_2 (BatchNormalization)	(None, 95, 169, 32)	128

max_pooling2d_2 (MaxPooling2D)	(None, 48, 85, 32)	0
conv2d_3 (Conv2D)	(None, 48, 85, 64)	18,496
batch_normalization_3 (BatchNormalization)	(None, 48, 85, 64)	256
max_pooling2d_3 (MaxPooling2D)	(None, 24, 43, 64)	0
conv2d_4 (Conv2D)	(None, 24, 43, 128)	73,856
batch_normalization_4 (BatchNormalization)	(None, 24, 43, 128)	512
max_pooling2d_4 (MaxPooling2D)	(None, 12, 22, 128)	0
conv2d_5 (Conv2D)	(None, 12, 22, 256)	295,168
batch_normalization_5 (BatchNormalization)	(None, 12, 22, 256)	1,024
max_pooling2d_5 (MaxPooling2D)	(None, 6, 11, 256)	0
conv2d_6 (Conv2D)	(None, 6, 11, 512)	1,180,160
batch_normalization_6 (BatchNormalization)	(None, 6, 11, 512)	2,048
max_pooling2d_6 (MaxPooling2D)	(None, 3, 6, 512)	0
conv2d_7 (Conv2D)	(None, 3, 6, 1024)	4,719,616
batch_normalization_7 (BatchNormalization)	(None, 3, 6, 1024)	4,096
max_pooling2d_7 (MaxPooling2D)	(None, 2, 3, 1024)	0
conv2d_8 (Conv2D)	(None, 2, 3, 2048)	18,876,416
batch_normalization_8 (BatchNormalization)	(None, 2, 3, 2048)	8,192
max_pooling2d_8 (MaxPooling2D)	(None, 1, 2, 2048)	0
conv2d_9 (Conv2D)	(None, 1, 2, 4096)	75,501,568
batch_normalization_9 (BatchNormalization)	(None, 1, 2, 4096)	16,384
max_pooling2d_9 (MaxPooling2D)	(None, 1, 1, 4096)	0
flatten (Flatten)	(None, 4096)	0
dense (Dense)	(None, 256)	1,048,832
dense_1 (Dense)	(None, 32)	8,224
coords (Dense)	(None, 4)	132

Total params: 101,761,236 (388.19 MB)

Trainable params: 101,744,868 (388.13 MB)

Non-trainable params: 16,368 (63.94 KB)

In [10]:
```
model.compile(
    loss={
        'coords': 'mse'
    },
    optimizer=tf.keras.optimizers.Adam(1e-3),
    metrics={
        'coords': 'accuracy'
    }
)
```

In [11]:
```
def test_model(model, datagen):
    example, label = next(datagen)

    X = example['image']
    y = label['coords']

    pred_bbox = model.predict(X)[0]

    img = X[0]
    gt_coords = y[0]

    display_image(img, pred_coords=pred_bbox, norm=True)

def test(model):
    datagen = data_generator(batch_size=1)

    plt.figure(figsize=(15,7))
    for i in range(3):
        plt.subplot(1, 3, i + 1)
        test_model(model, datagen)
    plt.show()

class ShowTestImages(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs=None):
        test(self.model)
```

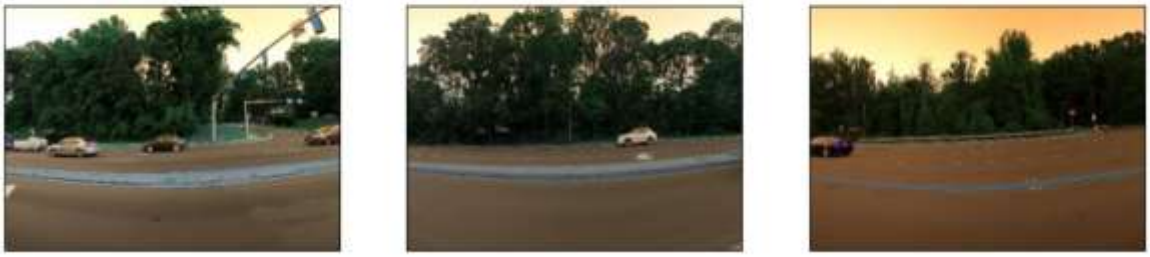
In [12]:
```
test(model)
```

The model hvaent been trained yet, thats why the predictions arent even visible yet

In [13]:
```
with tf.device('/GPU:0'):
    _ = model.fit(
        data_generator(),
        epochs=9,
        steps_per_epoch=500,
        callbacks=[
            ShowTestImages(),
        ]
    )
```

```
Epoch 1/9
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
```



500/500 ——————————— 174s 290ms/step - accuracy: 0.7479 - loss: 11350.7461
Epoch 2/9
1/1 ——————————— 0s 25ms/step
1/1 ——————————— 0s 27ms/step
1/1 ——————————— 0s 28ms/step



500/500 ——————————— 140s 281ms/step - accuracy: 0.9603 - loss: 851.5723
Epoch 3/9
1/1 ——————————— 0s 22ms/step
1/1 ——————————— 0s 28ms/step
1/1 ——————————— 0s 25ms/step



500/500 ——————————— 141s 282ms/step - accuracy: 0.9739 - loss: 365.6181
Epoch 4/9
1/1 ——————————— 0s 22ms/step
1/1 ——————————— 0s 26ms/step
1/1 ——————————— 0s 22ms/step



500/500 ——————————— 141s 282ms/step - accuracy: 0.9664 - loss: 1058.6162
Epoch 5/9
1/1 ——————————— 0s 26ms/step
1/1 ——————————— 0s 38ms/step
1/1 ——————————— 0s 21ms/step



500/500 ——————————— 141s 282ms/step - accuracy: 0.9837 - loss: 252.2519
Epoch 6/9
1/1 ——————————— 0s 27ms/step
1/1 ——————————— 0s 21ms/step
1/1 ——————————— 0s 28ms/step



500/500 ——————————— 140s 280ms/step - accuracy: 0.9884 - loss: 160.7852
Epoch 7/9
1/1 ——————————— 0s 21ms/step
1/1 ——————————— 0s 19ms/step
1/1 ——————————— 0s 21ms/step



```
500/500 ━━━━━━━━━━━━━━━━━━━━ 138s 277ms/step - accuracy: 0.8895 - loss: 113.4826
Epoch 8/9
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
```



```
500/500 ━━━━━━━━━━━━━━━━━━━━ 138s 276ms/step - accuracy: 0.9878 - loss: 107.6384
Epoch 9/9
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
```



```
500/500 ━━━━━━━━━━━━━━━━━━━━ 138s 276ms/step - accuracy: 0.9906 - loss: 91.2537
```

CHAPTER 06. CONCLUSION.

In conclusion, the car object detection mini-project utilizing neural networks has provided valuable insights into the capabilities and challenges of applying deep learning techniques to real-world computer vision tasks. The project demonstrated the effectiveness of neural networks, particularly convolutional neural networks (CNNs), in detecting cars within images or video streams. By leveraging deep learning architectures, we were able to achieve high accuracy and robustness in identifying vehicles across various scenarios. he quality of the dataset and the accuracy of annotation play crucial roles in the performance of the detection system. Proper dataset preparation, including careful curation, labeling, and augmentation, significantly contribute to the model's ability to generalize and perform well on unseen data.

Overall, the car object detection mini-project using neural networks served as a valuable learning experience, providing hands-on exposure to fundamental concepts in deep learning, computer vision, and model deployment. By addressing challenges and iteratively improving the detection system, participants gained practical skills and insights applicable to a wide range of applications beyond the scope of this mini-project.

CHAPTER 07. FUTURE SCOPE.

The car object detection mini-project using neural networks lays a solid foundation for future exploration and expansion into various directions. Here are several potential future scopes and extensions for further development:

1. **Multi-Class Detection:** Extend the object detection system to detect and classify multiple classes of vehicles, pedestrians, cyclists, and other objects commonly found in traffic environments. This broadens the applicability of the system in diverse scenarios and improves overall situational awareness.
2. **Instance Segmentation:** Incorporate instance segmentation techniques to not only detect objects but also segment them at the pixel level. This enables more precise localization and delineation of individual vehicles, especially in crowded scenes with overlapping objects.
3. **3D Object Detection:** Explore methods for 3D object detection, which estimate the 3D position, orientation, and dimensions of vehicles in the scene. This information is crucial for applications such as autonomous driving, where accurate spatial understanding of the environment is essential.
4. **Real-Time Performance Optimization:** Focus on optimizing the detection system for real-time performance without compromising accuracy. This involves further algorithmic optimizations, hardware acceleration (e.g., utilizing specialized inference hardware like TPUs or FPGAs), and model compression techniques to reduce computational overhead.

CHAPTER 08. LIST OF REFERENCES.

1. Redmon, J., & Farhadi, A. (2016). YOLO9000: Better, Faster, Stronger. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).
2. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016). SSD: Single Shot MultiBox Detector. In Proceedings of the European Conference on Computer Vision (ECCV).
3. Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In Advances in Neural Information Processing Systems (NIPS).
4. Lin, T. Y., Dollár, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2017). Feature Pyramid Networks for Object Detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).
5. Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. arXiv preprint arXiv:2004.10934.