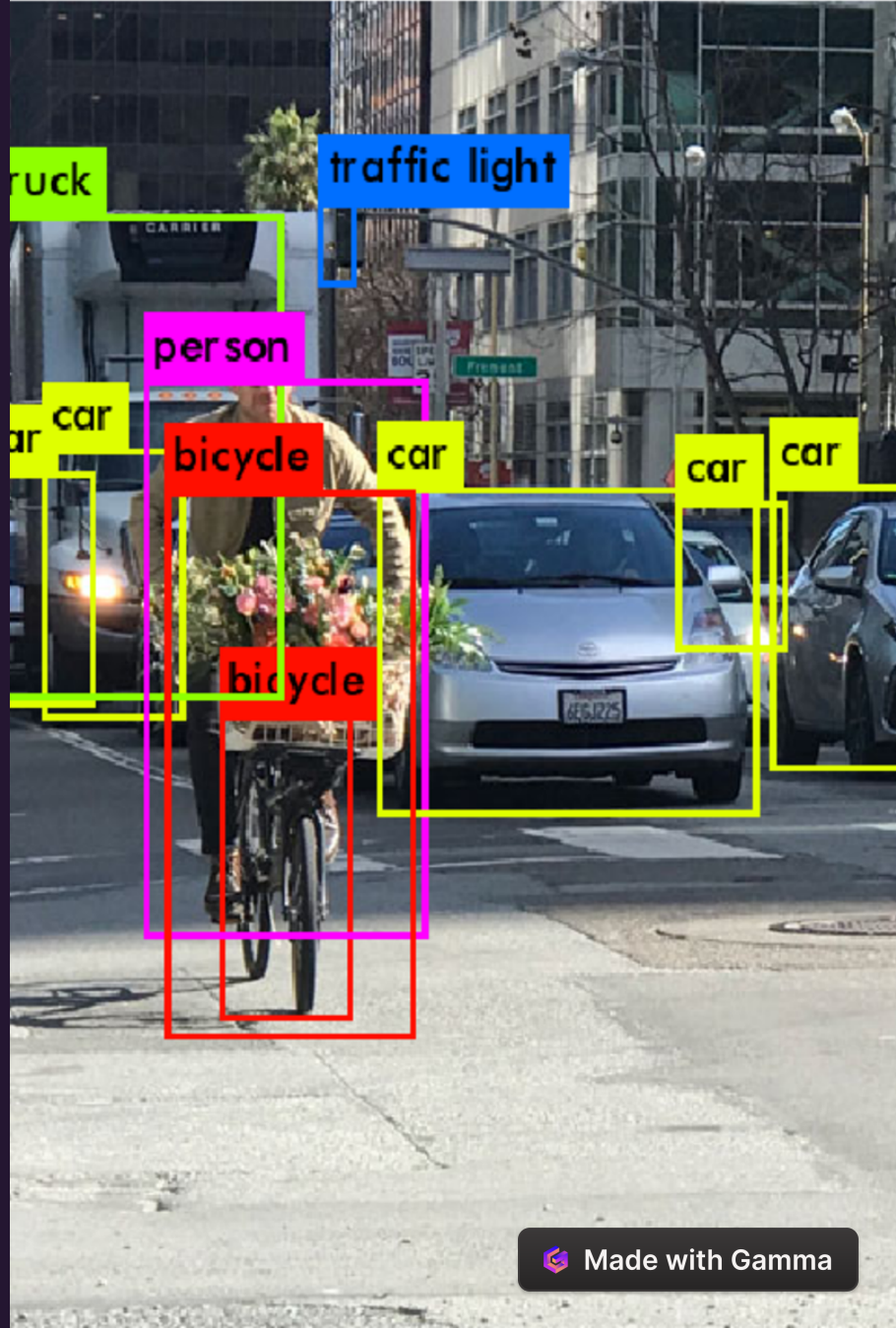
Car Object Detection Using OpenCV

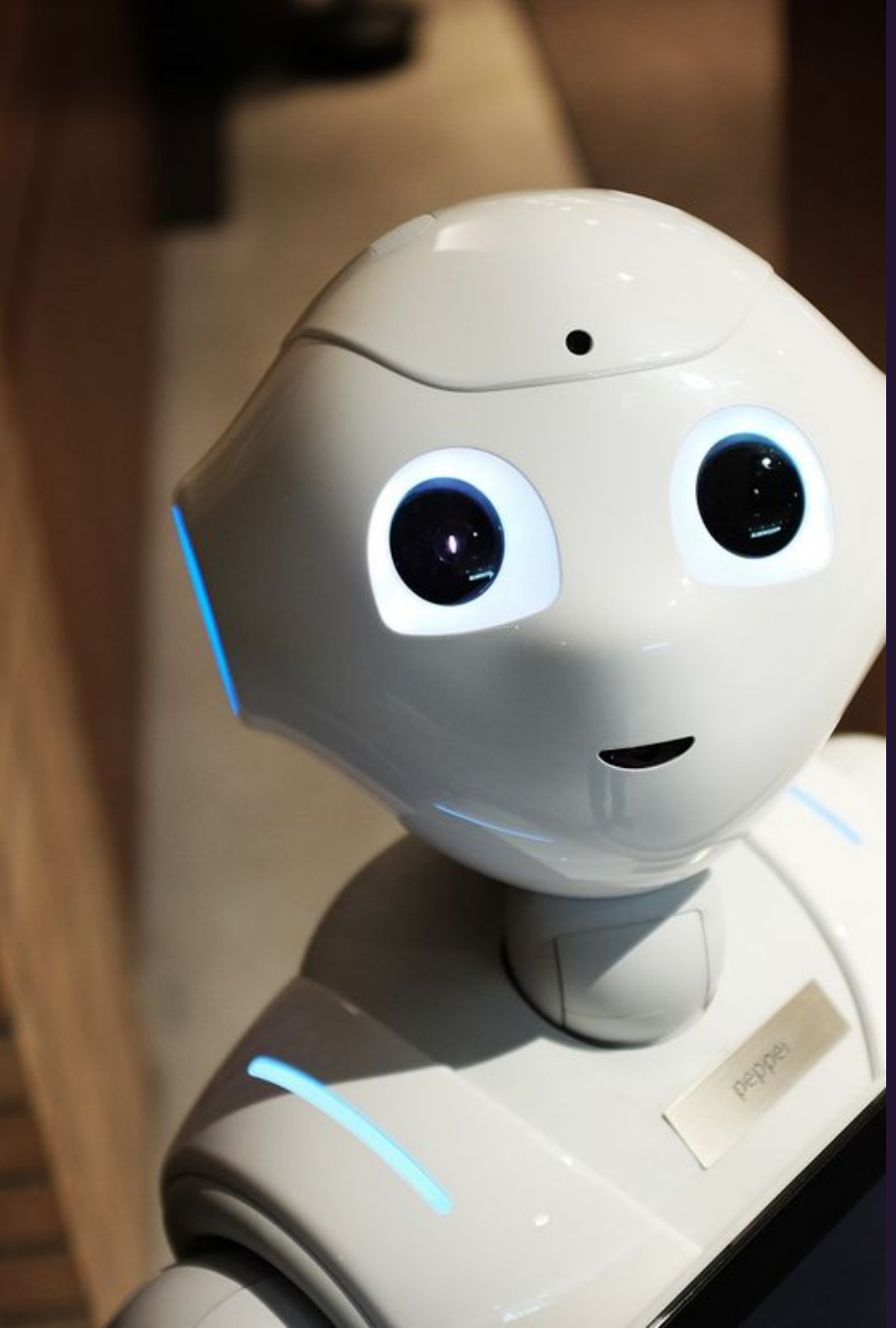
Welcome to my OpenCV project where I explore the fascinating world of car object detection from images. Join me on this journey!

by Ka Ka Shi



Building a Car Detection Model from Scratch

My mini project focuses on detecting a car from images. I created a custom trained model to power this project. With this model, you can upload any image and the system will detect if there is a car in it.



The Importance of Ethical AI

As artificial intelligence continues to advance, it is crucial to consider the ethical implications of its use.

- **Transparency and Explainability: Trust**
- **Fairness and Non-discrimination**
- **Privacy and Security**
- **Accountability and Responsibility**
- **Human Rights and Social Impact**

Object Detection Techniques

Haar Cascade Classifiers

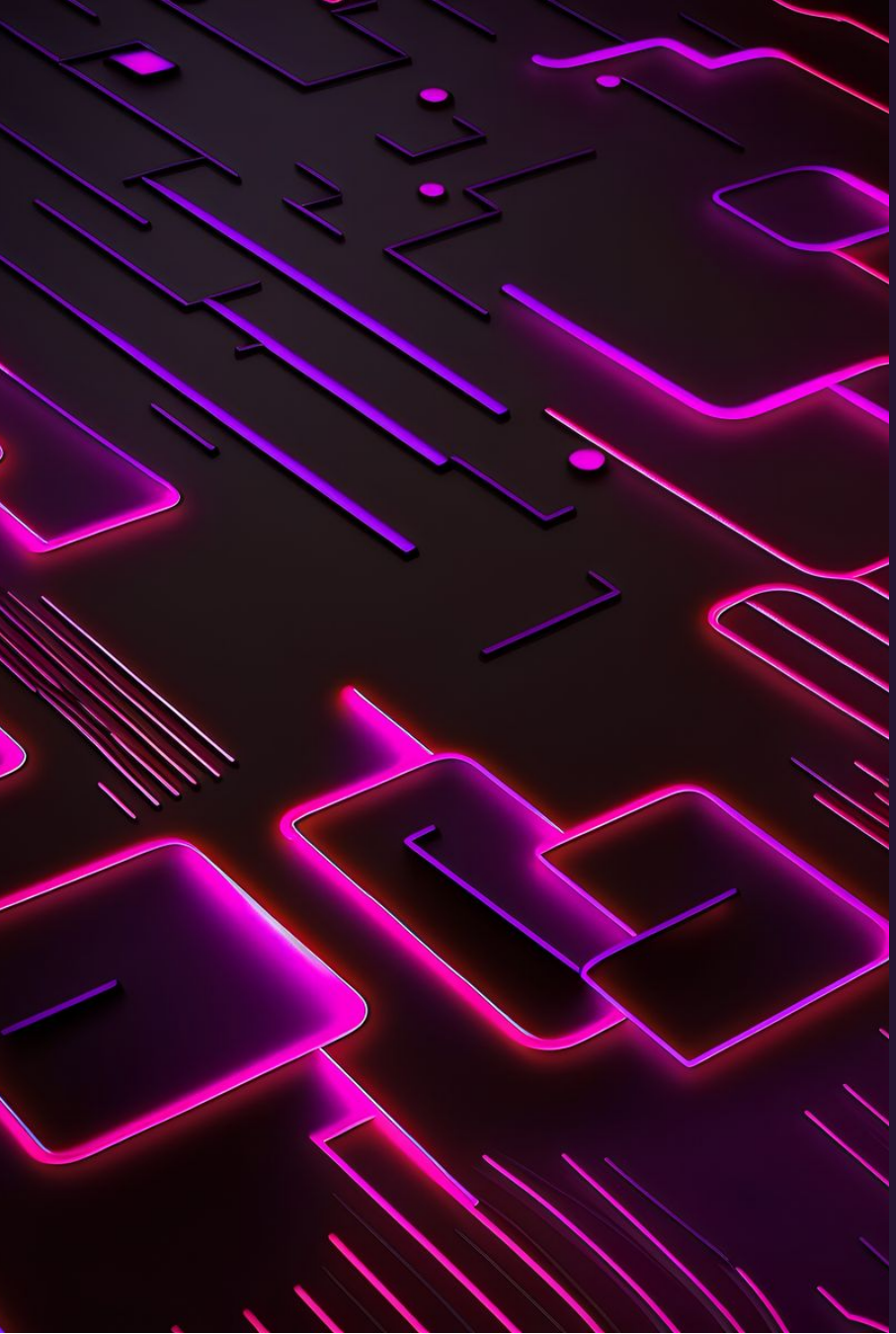
Learn about this popular method that uses machine learning to detect objects by analyzing patterns of intensity.

Deep Learning Approaches

Explore modern techniques like the YOLO (You Only Look Once) algorithm and Faster R-CNN for accurate and efficient object detection.

Template Matching

Discover how this simple yet powerful technique uses pixel-by-pixel comparison to detect objects based on predefined templates.



Why I Chose Deep Learning

- **High Accuracy:** Deep learning models can achieve high accuracy on complex tasks compared to traditional methods.
- **Automatic Feature Learning:** Deep learning automatically extracts features from data, eliminating the need for manual feature engineering.
- **Scalability:** Deep learning models can handle large amounts of data efficiently.

Project Timeline

To effectively manage the progress of my car detection project, I used Notion as a project management tool.

CarObj Detection 8

Task name	Due	Priority	Assignee	Status
Planning and Dataset Confirmation	December 6, 2023	High	Ka Ka Shi	Done
Dev Environment setup with Google Colab, Initial Setup	December 7, 2023	High	Ka Ka Shi	Done
Data Visualization, Model Building and Training	December 8, 2023	High	Ka Ka Shi	Done
Demo win Colab	December 9, 2023	High	Ka Ka Shi	Done
Convert to Streamlit Project	December 10, 2023	High	Ka Ka Shi	Archived
Convert to Application layer	December 12, 2023	High	Ka Ka Shi	Archived
Buffer	December 14, 2023	High	Ka Ka Shi	Archived
Preparation for Final Result	December 15, 2023	High	Ka Ka Shi	Archived

Exploring Dataset

For training and testing car object detection models, the Kaggle Car Object Detection dataset provides a rich collection of annotated images. With this dataset, developers can build and evaluate their own car detection algorithms, contributing to the advancement of computer vision and autonomous driving technologies.

The Dataset that I used in this project:
https://www.kaggle.com/datasets/sshikamaru/car-object-detection/data

What's in the Box?

The dataset contains media of cars in all views!

2 directories

- Training Images (folder with 1001 images)
- Testing Images (folder with 175 images)

2 files

- sample_submission.csv
- training_image_bounding_boxes.csv

Image Preprocessing for Car Detection

1

Rescale / Resize

Current dataset has just some amount of data images. So, data argumentation is needed.

2

Zooming

creating for new images for training process with a purpose of data argumentation.

3

Shear range

To be distorted along an axis, mostly to create or rectify the perception angles.

Training the Object Detection Model

Data Augmentation

As I mention before, I just used scaling, shear range and zoom range for data argumentation.

Model Training

In model training, I used a convolutional neural network (CNN) based approaches to detect cars from images with impressive accuracy.

Model Comparison

Kaggle's Example Model

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
image (InputLayer)           [(None, 380, 676, 3)]     0

conv2d (Conv2D)              (None, 380, 676, 8)       224

batch_normalization (Batch  (None, 380, 676, 8)       32
Normalization)

max_pooling2d (MaxPooling2  (None, 190, 338, 8)       0
D)

conv2d_1 (Conv2D)            (None, 190, 338, 16)      1168

batch_normalization_1 (Bat  (None, 190, 338, 16)      64
chNormalization)

max_pooling2d_1 (MaxPoolin  (None, 95, 169, 16)       0
g2D)

conv2d_2 (Conv2D)            (None, 95, 169, 32)       4640

batch_normalization_2 (Bat  (None, 95, 169, 32)       128
chNormalization)

max_pooling2d_2 (MaxPoolin  (None, 48, 85, 32)        0
g2D)

conv2d_3 (Conv2D)            (None, 48, 85, 64)        18496

batch_normalization_3 (Bat  (None, 48, 85, 64)        256
chNormalization)

max_pooling2d_3 (MaxPoolin  (None, 24, 43, 64)        0
g2D)

conv2d_4 (Conv2D)            (None, 24, 43, 128)       73856

batch_normalization_4 (Bat  (None, 24, 43, 128)       512
chNormalization)

max_pooling2d_4 (MaxPoolin  (None, 12, 22, 128)       0
g2D)

conv2d_5 (Conv2D)            (None, 12, 22, 256)       295168

batch_normalization_5 (Bat  (None, 12, 22, 256)       1024
chNormalization)

max_pooling2d_5 (MaxPoolin  (None, 6, 11, 256)        0
g2D)

conv2d_6 (Conv2D)            (None, 6, 11, 512)        1180160

batch_normalization_6 (Bat  (None, 6, 11, 512)        2048
chNormalization)

max_pooling2d_6 (MaxPoolin  (None, 3, 6, 512)         0
g2D)

conv2d_7 (Conv2D)            (None, 3, 6, 1024)        4719616

batch_normalization_7 (Bat  (None, 3, 6, 1024)        4096
chNormalization)

max_pooling2d_7 (MaxPoolin  (None, 2, 3, 1024)        0
g2D)

conv2d_8 (Conv2D)            (None, 2, 3, 2048)        18876416

batch_normalization_8 (Bat  (None, 2, 3, 2048)        8192
chNormalization)

max_pooling2d_8 (MaxPoolin  (None, 1, 2, 2048)        0
g2D)

conv2d_9 (Conv2D)            (None, 1, 2, 4096)        75501568

batch_normalization_9 (Bat  (None, 1, 2, 4096)        16384
chNormalization)

max_pooling2d_9 (MaxPoolin  (None, 1, 1, 4096)        0
g2D)

flatten (Flatten)            (None, 4096)              0

dense (Dense)                (None, 256)               1048832

dense_1 (Dense)              (None, 32)                8224

coords (Dense)               (None, 4)                 132

=================================================================
Total params: 101761236 (388.19 MB)
Trainable params: 101744868 (388.13 MB)
Non-trainable params: 16368 (63.94 KB)
_________________________________________________________________
```

My Custom Model

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 380, 676, 3)]     0

conv2d (Conv2D)              (None, 380, 676, 8)       224

batch_normalization (Batch  (None, 380, 676, 8)       32
Normalization)

max_pooling2d (MaxPooling2  (None, 190, 338, 8)       0
D)

conv2d_1 (Conv2D)            (None, 190, 338, 16)      1168

batch_normalization_1 (Bat  (None, 190, 338, 16)      64
chNormalization)

max_pooling2d_1 (MaxPoolin  (None, 95, 169, 16)       0
g2D)

conv2d_2 (Conv2D)            (None, 95, 169, 32)       4640

batch_normalization_2 (Bat  (None, 95, 169, 32)       128
chNormalization)

max_pooling2d_2 (MaxPoolin  (None, 48, 85, 32)        0
g2D)

conv2d_3 (Conv2D)            (None, 48, 85, 64)        18496

batch_normalization_3 (Bat  (None, 48, 85, 64)        256
chNormalization)

max_pooling2d_3 (MaxPoolin  (None, 24, 43, 64)        0
g2D)

conv2d_4 (Conv2D)            (None, 24, 43, 128)       73856

batch_normalization_4 (Bat  (None, 24, 43, 128)       512
chNormalization)

max_pooling2d_4 (MaxPoolin  (None, 12, 22, 128)       0
g2D)

conv2d_5 (Conv2D)            (None, 12, 22, 256)       295168

batch_normalization_5 (Bat  (None, 12, 22, 256)       1024
chNormalization)

max_pooling2d_5 (MaxPoolin  (None, 6, 11, 256)        0
g2D)

conv2d_6 (Conv2D)            (None, 6, 11, 512)        1180160

batch_normalization_6 (Bat  (None, 6, 11, 512)        2048
chNormalization)

max_pooling2d_6 (MaxPoolin  (None, 3, 6, 512)         0
g2D)

conv2d_7 (Conv2D)            (None, 3, 6, 1024)        4719616

batch_normalization_7 (Bat  (None, 3, 6, 1024)        4096
chNormalization)

max_pooling2d_7 (MaxPoolin  (None, 2, 3, 1024)        0
g2D)

conv2d_8 (Conv2D)            (None, 2, 3, 2048)        18876416

batch_normalization_8 (Bat  (None, 2, 3, 2048)        8192
chNormalization)

max_pooling2d_8 (MaxPoolin  (None, 1, 2, 2048)        0
g2D)

conv2d_9 (Conv2D)            (None, 1, 2, 4096)        75501568

batch_normalization_9 (Bat  (None, 1, 2, 4096)        16384
chNormalization)

max_pooling2d_9 (MaxPoolin  (None, 1, 1, 4096)        0
g2D)

flatten (Flatten)            (None, 4096)              0

dense (Dense)                (None, 256)               1048832

dense_1 (Dense)              (None, 32)                8224

coords (Dense)               (None, 4)                 132

=================================================================
Total params: 101761236 (388.19 MB)
Trainable params: 101744868 (388.13 MB)
Non-trainable params: 16368 (63.94 KB)
_________________________________________________________________
```

Comparing Trained Models for Detection

Kaggle Model : 0.98 accuracy

From the example of the kaggle model has 0.98 accuracy.

Resource is Here:

https://www.kaggle.com/code/bala418/car-object-detection

My Custom Model: 0.80 accuracy

My custom model has only 0.80 accuracy for now.

Google Colab Notebook is Here:
https://colab.research.google.com/drive/11M1kXB2AW-VAuZa02_6Jy3iwK5alM1zD?usp=sharing

Model Results Comparison

```
1/1 [==============================] - 0s 43ms/step
1/1 [==============================] - 0s 28ms/step
```

```
500/500 [==============================] - 161s 322ms/step - loss: 114.3404 - accuracy: 0.9911
Epoch 9/9
1/1 [==============================] - 0s 79ms/step
1/1 [==============================] - 0s 51ms/step
1/1 [==============================] - 0s 76ms/step
```

```
500/500 [==============================] - 165s 331ms/step - loss: 116.9058 - accuracy: 0.9869
```

Fig 1. Kaggle example Model's result

```
Epoch 1/10
35/35 - 149s - loss: 42004.4180 - accuracy: 0.5939 - 149s/epoch - 4s/step
Epoch 2/10
35/35 - 40s - loss: 20481.1602 - accuracy: 0.6619 - 40s/epoch - 1s/step
Epoch 3/10
35/35 - 40s - loss: 17257.3340 - accuracy: 0.6673 - 40s/epoch - 1s/step
Epoch 4/10
35/35 - 37s - loss: 14680.7959 - accuracy: 0.7352 - 37s/epoch - 1s/step
Epoch 5/10
35/35 - 40s - loss: 14876.5732 - accuracy: 0.7352 - 40s/epoch - 1s/step
Epoch 6/10
35/35 - 38s - loss: 14193.2100 - accuracy: 0.7835 - 38s/epoch - 1s/step
Epoch 7/10
35/35 - 40s - loss: 13889.1689 - accuracy: 0.7531 - 40s/epoch - 1s/step
Epoch 8/10
35/35 - 37s - loss: 13124.7021 - accuracy: 0.7943 - 37s/epoch - 1s/step
Epoch 9/10
35/35 - 39s - loss: 13224.4844 - accuracy: 0.7764 - 39s/epoch - 1s/step
Epoch 10/10
35/35 - 37s - loss: 12984.3701 - accuracy: 0.8086 - 37s/epoch - 1s/step
```

Fig 2. My Custom Model's Result

Models' Limitation!

⚠

- Both models can only detect single car object.
- They cannot detect multiple car objects yet.
- Test image should be an image that consists the car is only ahead to the left (more accuracy).
- car and image ratio should be most likely the same to the training images' ratio.

Results on Streamlit

Car Object Detection

A simple and accurate **CV Project**.

Check out this project on Github.

Disclaimer

⚠ This demo is only dedicated for single car object detection.

Choose an image for detection

Drag and drop file here
Limit 200MB per file • JPG, JPEG
Browse files

vid_5_26620.jpg 93.9KB

Original Image

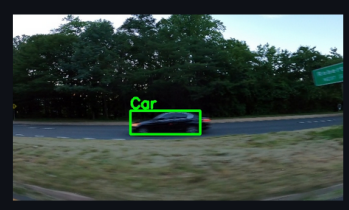


Image with Bounding Boxes

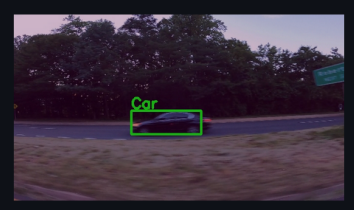


Image with Heatmap, Bounding Boxes and Text

Car Object Detection

A simple and accurate **CV Project**.

Check out this project on Github.

Disclaimer

⚠ This demo is only dedicated for single car object detection.

Choose an image for detection

Drag and drop file here
Limit 200MB per file • JPG, JPEG
Browse files

vid_5_26620.jpg 93.9KB

Original Image

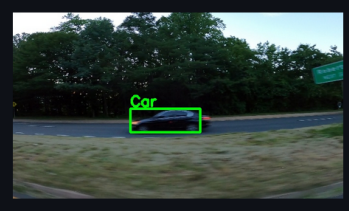


Image with Bounding Boxes

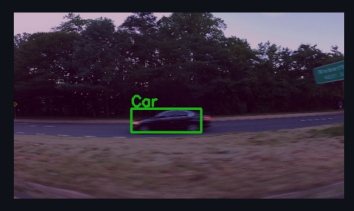


Image with Heatmap, Bounding Boxes and Text

DEMO

Source code is Available on Github.

https://github.com/kakashihatakhae/Demo-Single-Car-Obj-Detection/

Overcoming Project Challenges

The obstacles encountered while developing the car object detection model and the strategies used to overcome them.

- [ ] Ground knowledge acquisition
- [ ] Data Requirements
- [ ] Computational costs (GPU)
- [ ] Some deployment bottlenecks

Future Improvements and Conclusions

1

Real-Time Object Detection

Explore the exciting possibilities of real-time car detection using techniques like SSD (Single Shot MultiBox Detector).

2

Multi-Class Object Detection

Discover how to extend the car detection model to detect other objects like pedestrians, bicycles, and traffic signs.

3

Conclusion

Reflect on the journey and the importance of object detection in various domains, from autonomous vehicles to surveillance systems.

References

- https://www.kaggle.com/datasets/sshikamaru/car-object-detection/data
- https://www.kaggle.com/code/bala418/car-object-detection
- https://en.wikipedia.org/wiki/Machine_learning
- https://github.com/chandravenky/Computer-Vision---Object-Detection-in-Python
- https://bard.google.com/

Thanks In Advance!

First, a big thank you to everyone who supported this project! If you have any questions about the car object detection model, its implementation, or future directions, please don't hesitate to ask. Your feedback is invaluable!