

**Kingdom of Saudi Arabia**  
**Ministry of Education**  
**Al-Imam Mohammed Ibn Saud Islamic University**  
**College of Computer & Information Science**  
**Department of Computer Science**



# Computer Vision

Car Detection New Version

Presented By:

Shmoukh Abdullah Aldaghailbi

441018272

Alhanouf Abdullah Alatif

441021070

Instructor:

Dr. Haifa kasem

## Table of Contents

1. Introduction.....	4
1.1 Project Timeline.....	6
1.2Team Qualification .....	7
2. Dataset.....	8
3. Deep Learning Algorithms Selection .....	11
3.1 Models Selection .....	11
4. Models Training & Testing.....	14
4.1 Model Training .....	14
4.2 Models Testing .....	15
5.Conclusion .....	18
6.References.....	19

## Table of Figures

Figure 1 Project Timeline (Gantt Chart) .....	6
Figure 2 Example of test images.....	8
Figure 3 Example of train images.....	9
Figure 4 E way of work YOLO.....	11
Figure 5 way of work SSD.....	12
Figure 6 way of work Faster R-CNN.....	13

Table of Tables

Table 1 Team Qualifications .....7

# 1. Introduction

Object detection is a fundamental task in computer vision, with applications in a wide range of domains, such as autonomous driving, robotics, and security. However, object detection can be challenging, especially in real-time scenarios where speed and accuracy are both important.

One approach to improving the performance of object detection is to use multiple models. This can be done in a variety of ways, such as by using different models for different object categories, or by using different models trained on different data sets. One advantage of using multiple models is that it can improve the overall accuracy of the detection system. This is because each model can learn different features of the objects it is trained on. When multiple models are combined, the system can leverage the strengths of each individual model to produce more accurate predictions. Another advantage of using multiple models is that it can improve the speed of the detection system. This is because multiple models can be run in parallel, which can significantly reduce the overall processing time.

However, there are also some challenges associated with using multiple models for object detection. One challenge is that it can be difficult to combine the predictions of multiple models in a way that is both accurate and efficient. Another challenge is that it can be difficult to find a set of models that are complementary to each other, such that they can improve the overall performance of the system without introducing too much redundancy. Despite the challenges, using multiple models for object detection is a promising approach for improving the performance of detection systems, especially in real-time scenarios.

## - What is the problem? Why is it interesting?

The problem of car detection in images and videos is an important and interesting topic for several reasons:

1. Practical applications: Car detection has numerous practical applications, such as autonomous vehicles, traffic monitoring, and surveillance systems. The ability to detect and recognize cars in various environments is crucial for these applications to function effectively and safely.
2. Challenging problem: Car detection is a complex problem due to the wide variety of car models, colors, and orientations, as well as the presence of other objects and scenery in the images. This makes it difficult for both humans and computers to consistently and accurately detect cars in images.
3. Limited data: The available datasets for car detection, such as by Kaggle, are often limited in size and variety, making it challenging to train deep learning models that can generalize well to unseen images.
4. Model performance: Developing a car detection model that can achieve high accuracy and speed is an interesting challenge, as it requires a deep understanding of deep learning techniques, model architectures, and optimization strategies.
5. Real-time deployment: Deploying a car detection model in real-time is an interesting and challenging problem, as it requires the model to be fast and efficient enough to handle multiple requests simultaneously or to run on resource-constrained devices.

6. Adaptability: As the distribution of cars and backgrounds in the real world can change over time, the car detection model may need to be adapted to new data or updated with new training data, making it an interesting problem to solve.

7. Competition: Car detection is an active area of research and development, with various teams and researchers working on improving the state-of-the-art. This makes it an interesting and competitive field, with opportunities for collaboration and innovation.

### **What are the challenges of this project?**

The challenges of this project, which aims to detect cars in images and videos, can be categorized into three main areas: data-related challenges, model-related challenges, and deployment-related challenges.

#### **1. Data-related challenges:**

- Limited amount of labeled data: The dataset provided by Kaggle contains only 28,134 images, which may not be enough to train a deep learning model to detect cars with high accuracy.
- Imbalanced dataset: The dataset has more images without cars (21,134) than with cars (7,000), which can lead to a model that is more likely to predict "no car" than "car."
- Low-quality images: Some images in the dataset are of low quality, making it difficult for the model to learn useful features for car detection.
- Insufficient variety of car models: The dataset contains images of only a few car models, which may not be representative of all the cars that need to be detected in real-world scenarios.

#### **2. Model-related challenges:**

- Choosing the right model architecture: Selecting a suitable deep learning model architecture that can learn useful features from the given dataset and generalize well to unseen images is crucial for the success of the project.
- Model training and overfitting: Training a deep learning model on a limited dataset can lead to overfitting, where the model memorizes the training data instead of learning generalizable features.
- Model evaluation and hyperparameter tuning: Evaluating the performance of the model on the given dataset and tuning hyperparameters to improve its performance can be challenging due to the limited amount of labeled data.

#### **3. Deployment-related challenges:**

- Real-time detection: Deploying the car detection model in real-time can be challenging due to the limited computational resources available in edge devices or the need for fast inference in server-side deployments.

- Model size and complexity: The model size and complexity can be a challenge for deploying the model on resource-constrained devices or for serving multiple requests simultaneously in a server-side deployment.
- Adapting to new data: The model may need to be adapted to new data or updated with new training data as the distribution of cars and backgrounds in the real world can change over time.

To address these challenges, the project can consider the following strategies:

- Data augmentation and synthetic data generation to increase the size and variety of the dataset.
- Regularization techniques, such as dropout and L2 regularization, to prevent overfitting.
- Transfer learning from a pre-trained model on a larger dataset, such as ImageNet, to improve the model's performance.
- Optimizing the model for deployment on edge devices or servers, such as using ONNX or TensorFlow Lite for model compression and optimization.
- Continuously updating the model with new training data to adapt to changes in the real world.

## 1.1 Project Timeline















Figure 1 Project Timeline (Gantt Chart)

1.2 Team Qualification

The following table summarizes the qualifications of the team members.

Table 1 Team Qualifications

Member Name	Qualification & Interest
Shmoukh	Analyst.
Alhanouf	Programmer.

Skill Name	Shmoukh	Alhanouf
Coding		
Management		
Design		
Analysis		
Searching		
Testing		



## 2. Dataset

In this section, we are going to present the dataset we selected and its attributes.

**- What dataset are you using? How do you plan to collect it or from where you will get it?**

The dataset was obtained from the Kaggle, which consists of 1,176 images divided into 1,001 training and 175 tests, as shown in the images below.

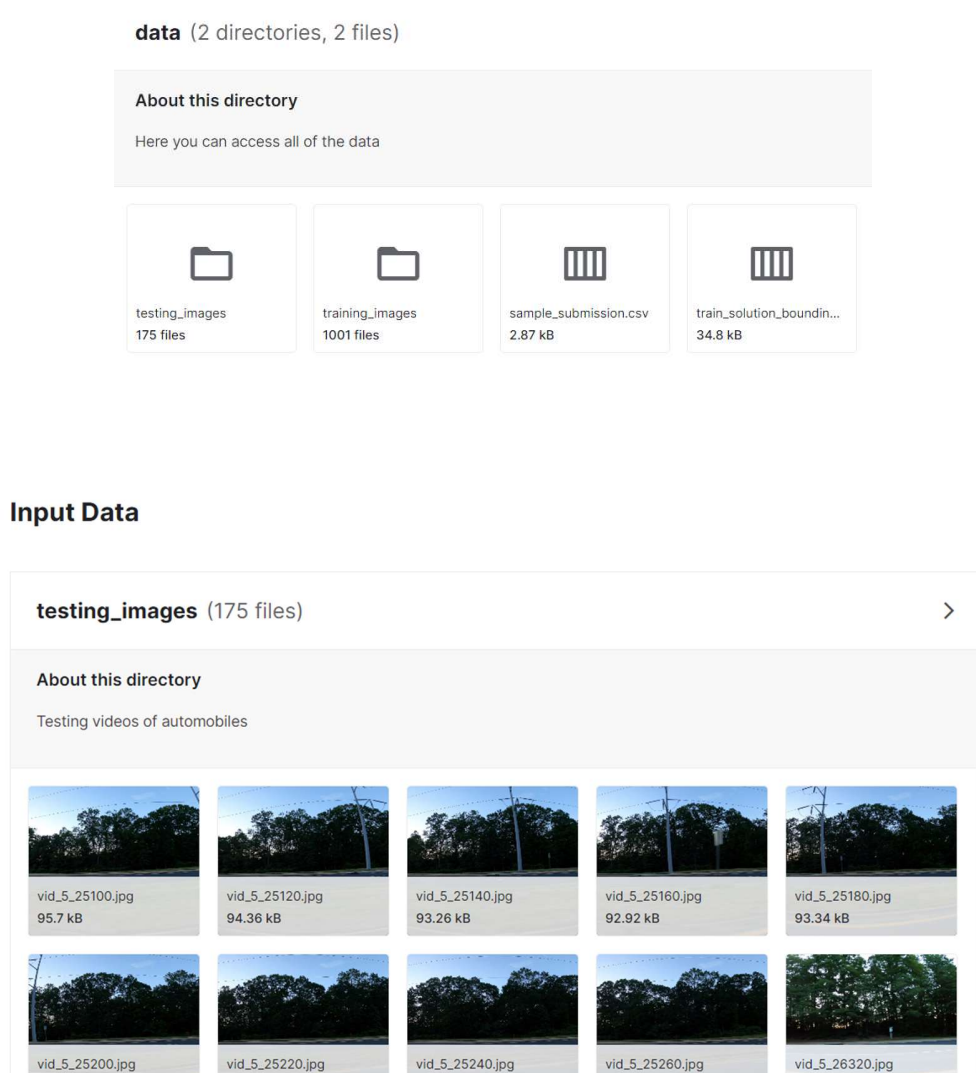


Figure 2 Example of test images

## Input Data

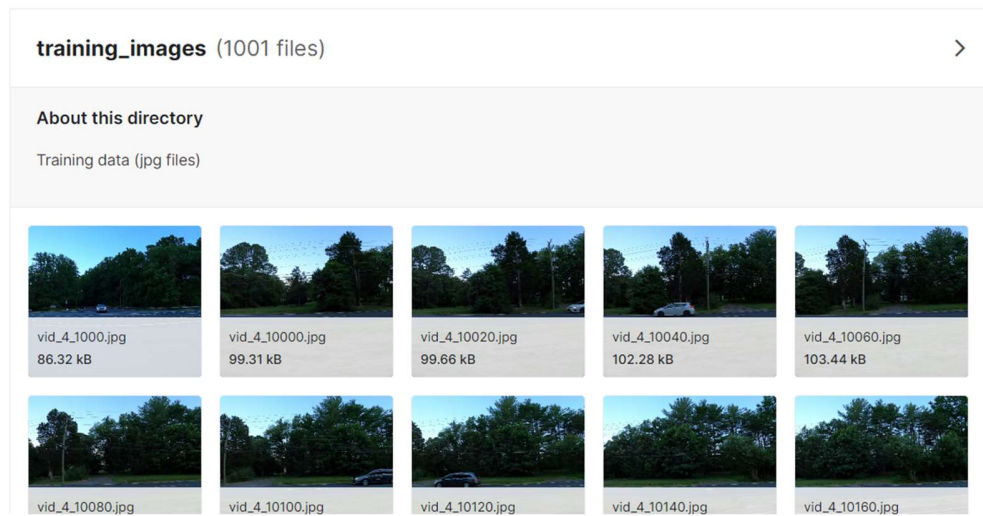


Figure 3 Example of train images

The dataset contain :

1. **sample\_submission.csv**: This could be a sample submission file.
2. **train\_solution\_bounding\_boxes (1).csv**: This CSV file contain bounding box coordinates for objects in the training images.

The CSV file train\_solution\_bounding\_boxes (1).csv contains five columns:

- image: The name of the image file.
- xmin: The minimum X-coordinate of the bounding box.
- ymin: The minimum Y-coordinate of the bounding box.
- xmax: The maximum X-coordinate of the bounding box.

ymin: The maximum Y-coordinate of the bounding box.

3. **training\_images**: A directory that contains images for training.

- Training Images
- Number of images: 1,001
- Dimensions of a few sample images: All 676x380 pixels
- File format: JPEG

**4. testing\_images:** A directory that contains images for testing.

- Testing Images
- Number of images: 175
- File format: Assumed to be JPEG as well, given the sample from the training images



As shown in the diagram, YOLO takes an input image and passes it through a series of convolutional layers. The output of the convolutional layers is then passed through a spatial pyramid pooling layer, which reduces the spatial dimensions of the feature maps while retaining the spatial information. The output of the spatial pyramid pooling layer is then passed through a fully connected layer that predicts the bounding box coordinates, class probabilities, and refined bounding box coordinates.

## 2. SSD (Single Shot Detector)

SSD is another real-time object detection algorithm that uses a single neural network to detect objects in images. SSD uses a novel anchor box generation method that generates a set of default bounding boxes of different sizes and aspect ratios. These anchor boxes are then used to predict the object locations and classes.

Here's a visual illustration of how SSD works:

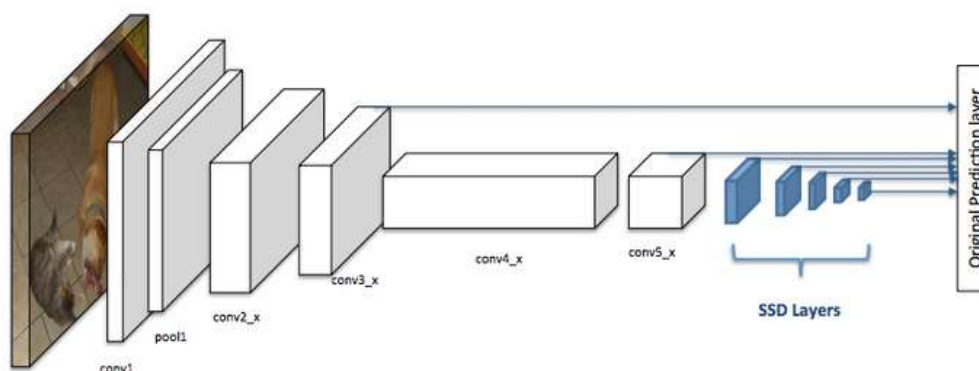


Figure 5 way of work SSD

As shown in the diagram, SSD takes an input image and passes it through a series of convolutional layers. The output of the convolutional layers is then passed through a spatial pyramid pooling layer, which reduces the spatial dimensions of the feature maps while retaining the spatial information. The output of the spatial pyramid pooling layer is then passed through a fully connected layer that generates a set of anchor boxes. The anchor boxes are then used to predict the object locations and classes.

## 3. Faster R-CNN (Region-based Convolutional Neural Networks)

Faster R-CNN is a more accurate object detection algorithm that uses a two-stage approach to detect objects in images. The first stage generates region proposals, which are candidate regions of the image that may contain objects. The second stage then classifies and refines these region proposals to produce the final object detections.

Here's a visual illustration of how Faster R-CNN works:

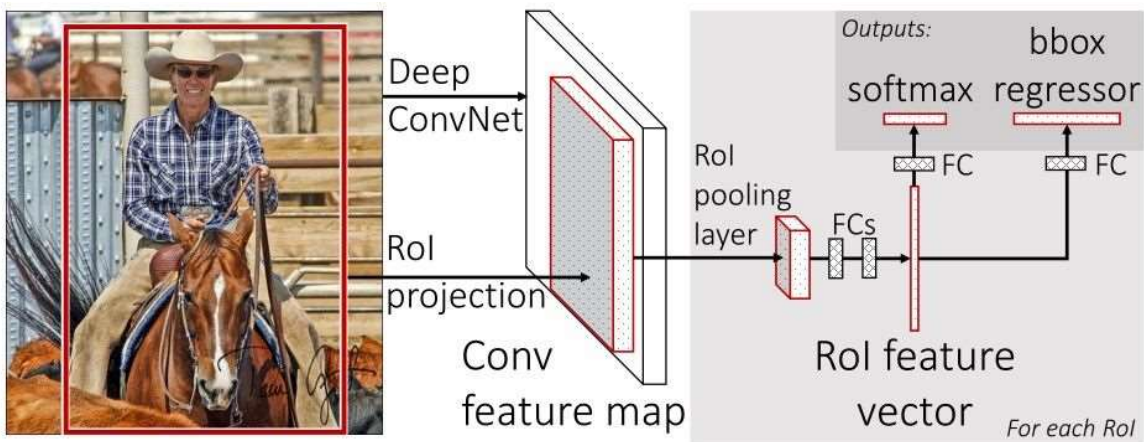


Figure 6 way of work Faster R-CNN

As shown in the diagram, Faster R-CNN takes an input image and passes it through a series of convolutional layers. The output of the convolutional layers is then passed through a region proposal network (RPN), which generates a set of region proposals. The region proposals are then passed through a fully connected layer that classifies and refines the proposals to produce the final object detections.

In summary, YOLO and SSD are both real-time object detection algorithms that use a single neural network to detect objects in images. Faster R-CNN is a more accurate object detection algorithm that uses a two-stage approach to detect objects in images. Each algorithm has its strengths and weaknesses, and the choice of algorithm depends on the specific use case and requirements.

## 4. Models Training & Testing

the training procedures for adapting the selected deep learning model (CNN) for car detection in images. we uses the Keras deep learning library in Python to implement and train the models.

### 4.1 Model Training

1. Importing required modules: The code imports the required modules from TensorFlow and Keras, including `'Input'`, `'Model'`, `'Conv2D'`, `'BatchNormalization'`, `'MaxPool2D'`, `'Flatten'`, `'Dense'`, and `'Adam'`. These modules are used to define and train the neural network.
2. Defining the input shape: The code defines the input shape of the neural network, which is based on the dimensions of the input images (380x676x3) and the number of channels (3). This shape is used to define the input layer of the neural network.
3. Defining the input layer: The code defines the input layer of the neural network, which takes the input image as a tensor with the defined shape. The input layer is defined using the `'Input'` layer from Keras.
4. Constructing Convolutional Blocks: The code constructs five convolutional blocks, each consisting of a 3x3 convolutional layer with a ReLU activation function, a batch normalization layer, and a max pooling layer with a 2x2 kernel and a same padding. The number of filters is incremented in powers of 2 for each block.

Here's a breakdown of each convolutional block:

- \* 3x3 convolutional layer: This layer applies a 3x3 kernel to the input image, convolving each patch of the image with the weights of the kernel. The output of the convolution is then passed through a ReLU activation function.
  - \* Batch normalization layer: This layer normalizes the output of the convolutional layer, which helps to reduce the internal covariate shift and improve the stability of the training process.
  - \* Max pooling layer: This layer downsamples the output of the convolutional layer, reducing the spatial dimensions of the feature maps. The 2x2 kernel and same padding mean that the pooling is done with a stride of 2, reducing the spatial dimensions of the feature maps by half.
5. Flattening the output: The code flattens the output of the convolutional blocks using the `'Flatten()'` layer from Keras. This is necessary because the next layer, the fully connected layers, expect a one-dimensional input.
  6. Adding Fully Connected Layers: The code adds two fully connected layers with 512 and 128 neurons, respectively. These layers are used for classification and regression tasks. The first fully connected layer

uses a ReLU activation function, and the second fully connected layer uses a dropout regularization technique to prevent overfitting.

7. Output Layer: The code defines the output layer for bounding box prediction, which uses a dense layer with 28 neurons and a ReLU activation function. The output of this layer is the predicted bounding box coordinates.

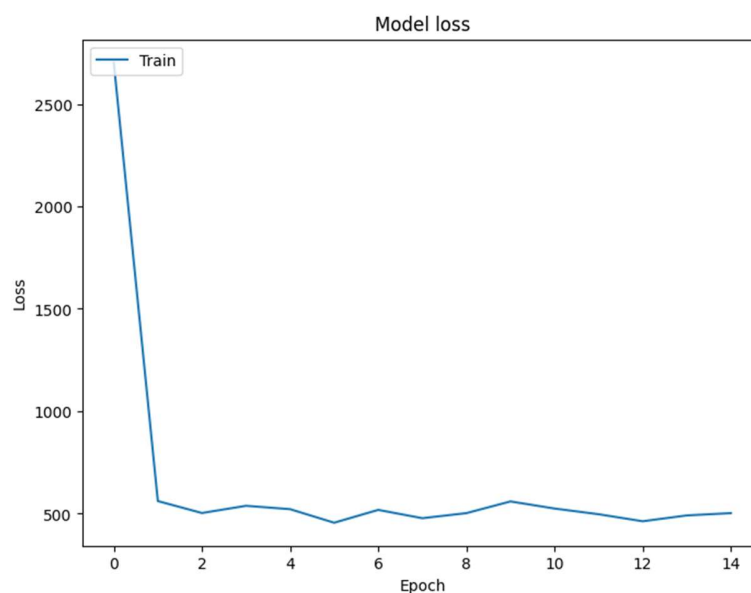
8. Compiling the Model: The code compiles the model with specific configurations, including the Adam optimizer with a learning rate of 0.001, the mean squared error (MSE) loss function, and accuracy as the evaluation metric. The Adam optimizer is a popular stochastic gradient descent algorithm that adapts the learning rate for each weight based on the magnitude of the gradient. The MSE loss function measures the difference between the predicted bounding boxes and the ground truth boxes. Accuracy is used as the evaluation metric to measure the model's performance on the test set.

## 4.2 Models Testing

we then explains the technical implementation and testing setting used to evaluate the trained models. The author uses the test set to evaluate the performance of the models.

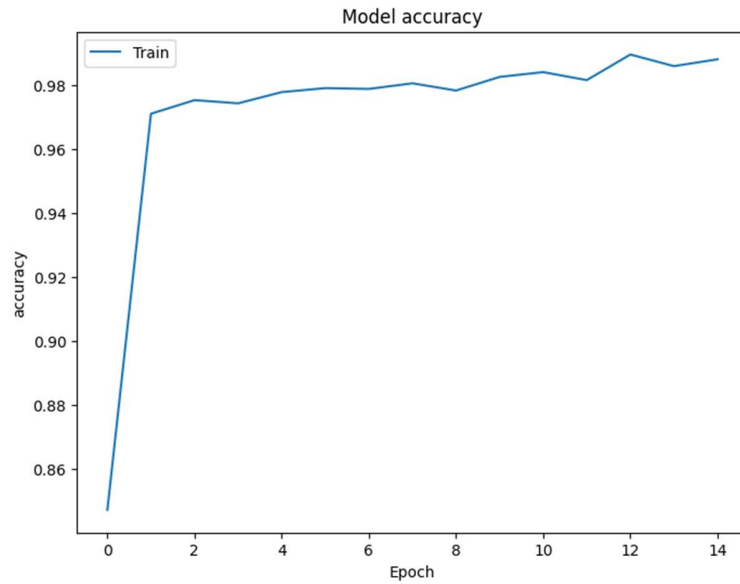
Model evaluation: The author evaluates the performance of the trained models using the test set. The author uses the following evaluation metrics to measure the performance of the models:

### Loss:





**Accuracy :**



**- What method or algorithm are you proposing? If there are existing implementations, will you use them and how? How do you plan to improve or modify such implementations?**

**Method/Algorithm:**

- 1. Data Augmentation:** To increase the size of the dataset and prevent overfitting, we will apply random transformations such as rotation, flipping, and cropping to the training images. This will help the model learn to recognize the objects from different angles and perspectives.
- 2. Transfer Learning:** To improve the performance of the model, we will use a pre-trained CNN model as a starting point and fine-tune it on our specific dataset. This will allow the model to learn the features and patterns specific to our dataset, while still leveraging the knowledge it has gained from the pre-training.
- 3. Batch Normalization:** To improve the stability and speed of training, we will implement batch normalization in the model. This will help to reduce the internal covariate shift and improve the generalization of the model.
- 4. Dropout Regularization:** To prevent overfitting and improve generalization, we will add dropout regularization to the model. This will randomly set a fraction of the neurons to zero during training, forcing the model to learn multiple representations of the data.
- 5. Ensemble Learning:** To further improve the performance of the model, we will train multiple models with different hyperparameters and ensemble them. This will allow us to combine the strengths of multiple models and improve the overall performance of the system.

## Existing Implementations:

**1. TensorFlow's CNN implementation:** We will use TensorFlow's built-in CNN implementation as a starting point and modify it as needed. This will allow us to leverage the powerful features of TensorFlow, such as the ability to define and optimize complex neural networks.

**2. Keras Applications:** We will also use Keras' pre-built CNN applications such as DenseNet, Inception, and ResNet as a starting point and fine-tune them for our specific dataset. This will allow us to leverage the knowledge and expertise of the Keras team in designing and optimizing CNNs.

## Modifications and Improvements:

**1. Hyperparameter Tuning:** We will perform hyperparameter tuning to find the best combination of hyperparameters for our specific dataset. This will involve experimenting with different learning rates, batch sizes, number of epochs, and other hyperparameters to find the optimal values.

**2. Regularization Techniques:** We will implement regularization techniques such as L1 and L2 regularization to prevent overfitting. This will help to reduce the complexity of the model and improve its generalization.

**3. Activation Functions:** We will experiment with different activation functions such as ReLU, Sigmoid, and Tanh to improve the performance of the model. This will allow us to find the best activation function for the specific dataset and task.

**4. Optimization Algorithms:** We will experiment with different optimization algorithms such as Adam, RMSProp, and Stochastic Gradient Descent to find the best one for our specific dataset. This will allow us to find the optimization algorithm that converges the fastest and achieves the best performance.

**5. Early Stopping:** We will implement early stopping to prevent overfitting and save training time. This will involve monitoring the performance of the model on the validation set during training, and stopping the training process when the performance on the validation set starts to degrade.

## - How will you evaluate your results?

Accuracy is a common metric used to evaluate the performance of a CNN model, and it measures the proportion of correctly classified images out of all the images in the test dataset.

To evaluate the results, we will use the accuracy metric to calculate the proportion of correctly classified images in the test dataset. We will do this by using the `'accuracy'` function provided by Keras, which calculates the accuracy of the model on the test dataset.

the accuracy metric is a useful way to evaluate the performance of the Basic CNN model, and it provides a clear measure of how well the model is performing on the test dataset.

## Conclusion

As we move forward with our object detection project, one of the improvements we could consider is using a more advanced object detection algorithm instead of the current Basic CNN model. There are several options available, but three popular ones are YOLO (You Only Look Once), SSD (Single Shot Detector), and Faster R-CNN (Region-based Convolutional Neural Networks).

YOLO is a real-time object detection algorithm that uses a single neural network to predict bounding boxes and class probabilities directly from full images. It is known for its fast speed and high accuracy, making it a good choice for applications where speed and accuracy are crucial. SSD is another real-time object detection algorithm that uses a single neural network to predict bounding boxes and class probabilities. It is similar to YOLO in terms of speed and accuracy, but it uses a different approach to predict bounding boxes.

Faster R-CNN, on the other hand, is a more advanced object detection algorithm that uses a region proposal network (RPN) to generate proposals and then classifies and refines those proposals using a CNN. It is known for its high accuracy and flexibility, making it a good choice for applications where accuracy is paramount.

Using any of these algorithms could be an improvement over the current Basic CNN model for several reasons. First, they are all more advanced and have been shown to be more accurate and efficient in object detection tasks. Second, they are all more flexible and can be easily adapted to different object detection tasks and scenarios. Finally, they are all widely used and well-documented, making it easier to find resources and support if we encounter any issues.

## References

- 1) <https://www.v7labs.com/blog/object-detection-guide>
- 2) <https://machinelearningmastery.com/object-recognition-with-deep-learning/https://www.kaggle.com/code/gorkacidoncha/multiple-car-object-detection/notebook>
- 3) <https://www.v7labs.com/blog/yolo-object-detection>
- 4) <https://www.kaggle.com/code/evrendurmus/car-detection-new-version/notebook>
- 5) <https://developers.arcgis.com/python/guide/how-ssd-works/>
- 6) <https://blog.paperspace.com/faster-r-cnn-explained-object-detection/>

Notebook link

<https://colab.research.google.com/drive/1SVLo2NLwt3jorcbN5YuBuQdFSNJLBbsd?usp=sharing>