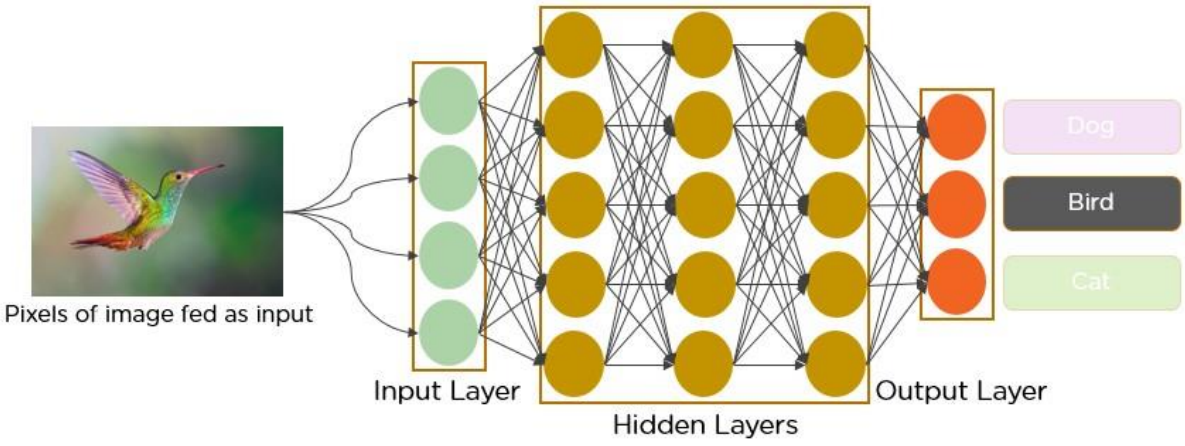
Deep Learning algorithms to demonstrate Object Detection

Algorithms used for Object Detection

1. CNN (Convolutional Neural Network)

A Convolutional Neural Network (CNN) is a type of deep learning algorithm and an Artificial Neural Network that is generally used for image recognition and processing tasks

- The **convolutional layers** are the key component of a CNN, where **filters** are applied to the input image for **feature extraction** such as edges, textures, and shapes.
- The output of the convolutional layers is then passed through **pooling layers**, which are used to down-sample the feature maps, reducing the spatial dimensions while retaining the most important information.
- The output of the pooling layers is then passed through one or more **fully connected layers**, which are used to make a prediction or classify the image.



A CNN can have multiple layers, each of which learns to detect the different features of an input image. A filter or kernel is applied to each image to produce an output that gets progressively better and more detailed after each layer. In the lower layers, the filters can start as simple features.

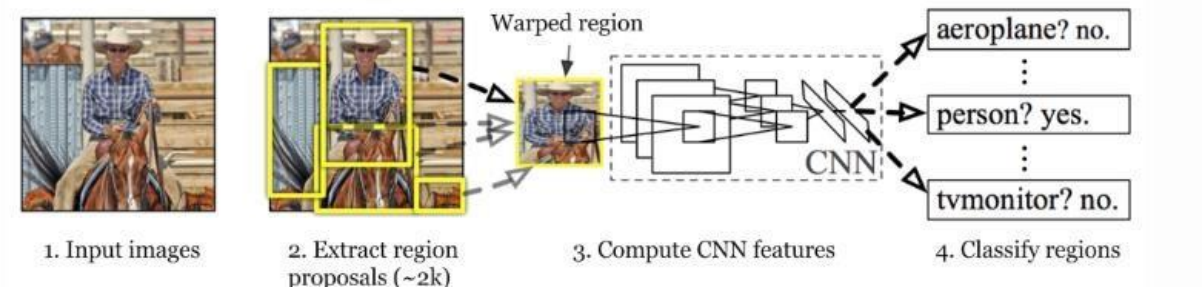
At each successive layer, the filters increase in complexity to check and identify features that uniquely represent the input object. Thus, the output of each convolved image -- the partially recognized image after each layer –

becomes the input for the next layer. In the last layer, which is an FC layer, the CNN recognizes the image or the object it represents.

2. R-CNN (Region based CNN)

Since Convolution Neural Network (CNN) with a fully connected layer is not able to deal with the frequency of occurrence and multi objects. So, one way could be that to select a region and apply the CNN model on that, but the problem of this approach is that the same object can be represented in an image with different sizes and different aspect ratio. While considering these factors we have a lot of region proposals and if we apply deep learning (CNN) on all those regions that would computationally very expensive

- R-CNN architecture uses the selective search algorithm that generates approximately 2000 region proposals.
- These 2000 region proposals are then provided to CNN architecture that computes CNN features.
- These features are then passed in an SVM model to classify the object present in the region proposal. An extra step is to perform a bounding box regressor to localize the objects present in the image more precisely.



RCNN architecture

Region proposals are simply the smaller regions of the image that possibly contains the objects we are searching for in the input image

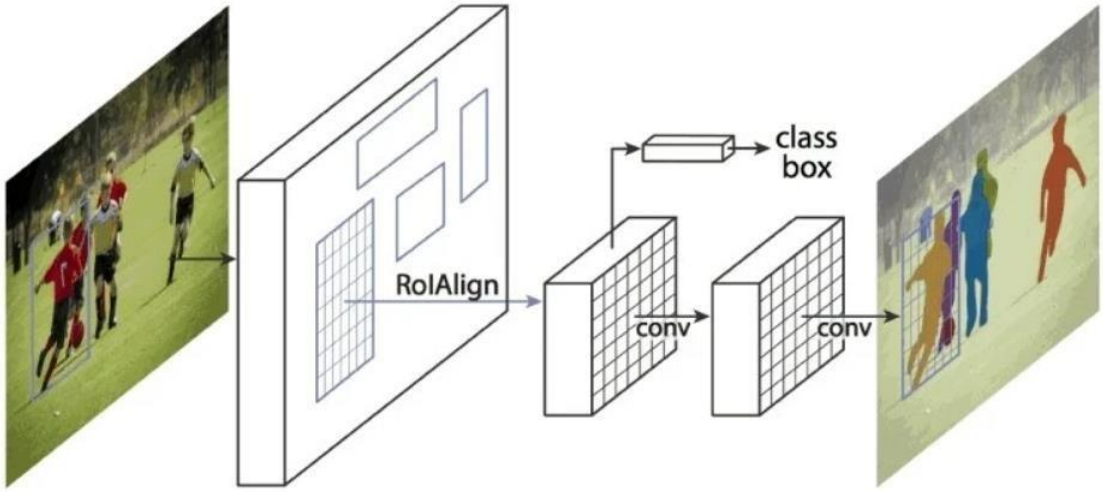
Selective search is a greedy algorithm that combines smaller segmented regions to generate region proposal.

3. Mask R-CNN

Mask RCNN, is a Convolutional Neural Network (CNN) and state-of-the-art in terms of image segmentation and instance segmentation. Mask R-CNN was developed on top of Faster R-CNN, a Region-Based Convolutional Neural Network.

The Mask RCNN architecture addresses the task of identifying and localizing objects in an image as well as generating a pixel level mask for each object. This allows for fine grained segmentation of individual objects within the image It consists of

- A pre trained CNN such as ResNet or VGG is used as a backbone network. It serves as a feature extractor and processes the input image to extract a set of feature map
- The Region Proposal Network generates a set of region proposals which are potential bounding boxes around objects
- Region based Convulational Neural Network (ROI CNN) takes the region proposals generated by the RPN and performs object classification and bounding box regression. ROI CNN also generates a pixel level mask for each object proposal using an additionally fully convulational network branch



4.Fast R-CNN

Architecture details:

To better understand how and why the Fast R-CNN improved efficiency and performance of R-CNN and SPP Networks, let's first look into its architecture.

- The Fast R-CNN consists of a CNN (usually pre-trained on the ImageNet classification task) with its final pooling layer replaced by an "ROI pooling" layer and its final FC layer is replaced by two branches — a (K + 1) category softmax layer branch and a category-specific bounding box regression branch.

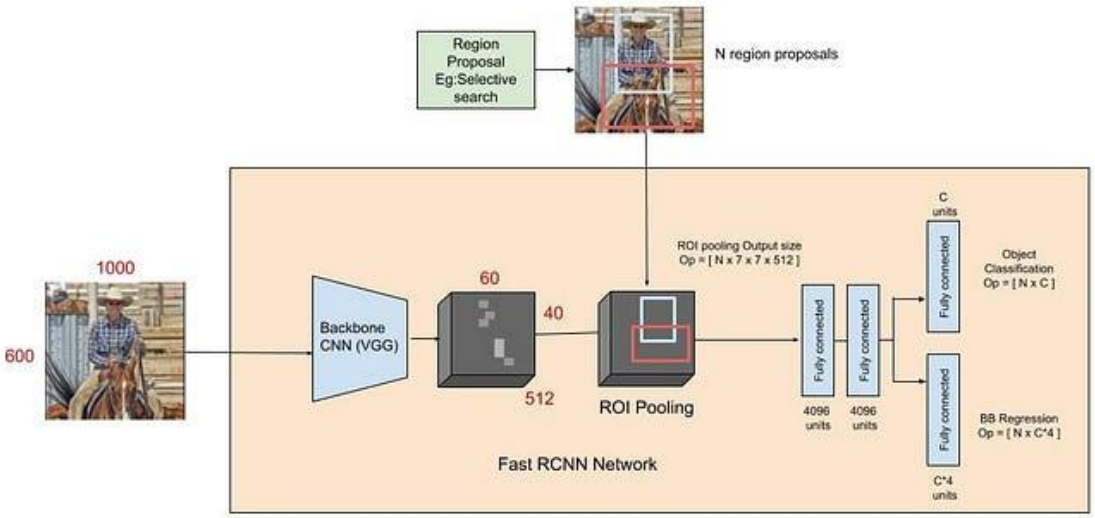


Figure 1: The Fast R-CNN pipeline

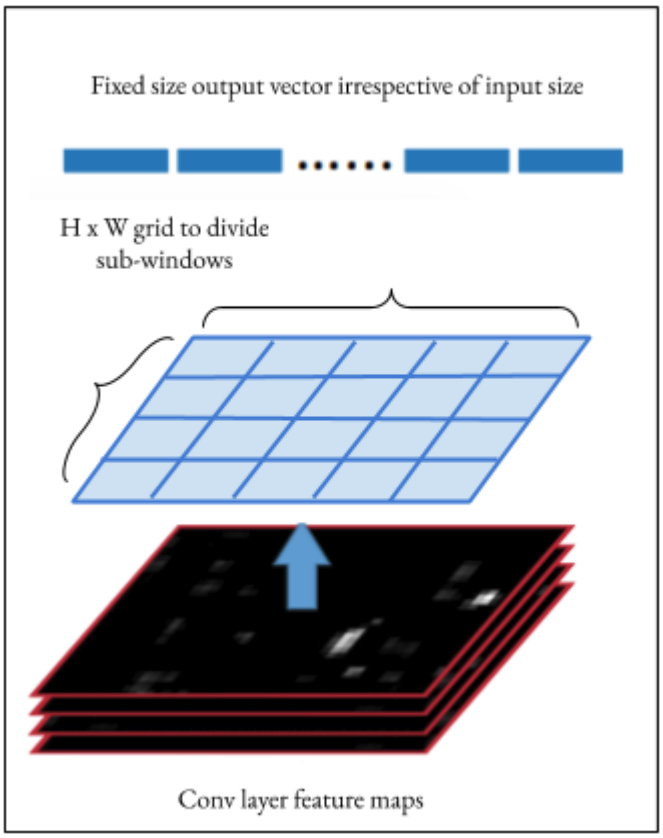


Figure 2: The ROI pooling layer as a special case of SPP layer

- The entire image is fed into the backbone CNN and the features from the last convolution layer are obtained. Depending on the backbone CNN used, the output feature maps are much smaller than the original image size. This depends on the stride of the backbone CNN, which is usually 16 in the case of a VGG backbone.
- Meanwhile, the object proposal windows are obtained from a region proposal algorithm like selective search[4]. As explained in Regions with CNNs, object proposals are rectangular regions on the image that signify the presence of an object.
- The portion of the backbone feature map that belongs to this window is then fed into the ROI Pooling layer.

- The ROI pooling layer is a special case of the spatial pyramid pooling (SPP) layer with just one pyramid level. The layer basically divides the features from the selected proposal windows (that come from the region proposal algorithm) into sub-windows of size h/H by w/W and performs a pooling operation in each of these sub-windows. This gives rise to fixed-size output features of size (H x W) irrespective of the input size. H and W are chosen such that the output is compatible with the network's first fully-connected layer. The chosen values of H and W in the Fast R-CNN paper is 7. Like regular pooling, ROI pooling is carried out in every channel individually.
- The output features from the ROI Pooling layer (N x 7 x 7 x 512 where N is the number of proposals) are then fed into the successive FC layers, and the softmax and BB-regression branches. The softmax classification branch produces probability values of each ROI belonging to K categories and one catch-all background category. The BB regression branch output is used to make the bounding boxes from the region proposal algorithm more precise.

5.Yolo V8

YOLO, an acronym that stands for "You Only Look Once" is a very fast and efficient algorithm, now available for some years. An Open Source version in Python is available and it often used in custom projects. It has been developed by a company called Ultralytics.

The important news at the end of 2022 is that Ultralytics has announced and made available a **new version**, version 8, which promises to be more accurate, faster and easier to use.

Architecture

The architecture of YOLOv8 builds upon the previous versions of YOLO algorithms. YOLOv8 utilizes a convolutional neural network that can be divided into two main parts: the backbone and the head.

A modified version of the CSPDarknet53 architecture forms the backbone of YOLOv8. This architecture consists of 53 convolutional layers and employs cross-stage partial connections to improve information flow between the different layers.

The head of YOLOv8 consists of multiple convolutional layers followed by a series of fully connected layers.

These layers are responsible for predicting bounding boxes, objectness scores, and class probabilities for the objects detected in an image.

One of the key features of YOLOv8 is the use of a self-attention mechanism in the head of the network. This mechanism allows the model to focus on different parts of the image and adjust the importance of different features based on their relevance to the task.

Another important feature of YOLOv8 is its ability to perform multi-scaled object detection. The model utilizes a feature pyramid network to detect objects of different sizes and scales within an image. This feature pyramid network consists of multiple layers that detect objects at different scales, allowing the model to detect large and small objects within an image.

Comparison of algorithms

Aspect	CNN	R-CNN	Mask R-CNN	Fast R-CNN	YOLO V8
Workflow	Convolutional layers followed by fully connected layers	Region proposal, feature extraction, classification, and bounding box regression	Region proposal, feature extraction, classification, bounding box regression, and mask prediction	Region proposal, feature extraction, classification, bounding box regression	YOLO divides an input image into an S × S grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object. Each grid cell predicts B bounding boxes and confidence scores for those boxes.
Input	Fixed-size images	Variable-size images	Fixed-size images	Fixed-size images	Fixed-size images
Output	Image classification	Object proposals, class labels, and bounding boxes	Object proposals, class labels, bounding boxes, and segmentation masks	Object proposals, class labels, and bounding boxes	Object proposals, class labels, and bounding boxes
Accuracy	78.75%	86.88%	97.52%	99.48%	66.67%
Loss	72.62%	38.89%	17.22%	17.07%	54.13%
MAP_0.5			98.92%	92.9%	99.5%

Precession			94.84%	97.5%	89.39%
Recall			93.76%	82.5%	98.33%

Dataset used:

Primary Dataset: https://www.kaggle.com/datasets/devkhant24/cars-and-bikes-prediction

Classes in the dataset:

- Car
- Bike

Implementation of algorithms:

1.CNN
Following steps were performed

1. Import the necessary libraries:
2. Define the paths to the image directories:
    - Cars and Bikes

3. Load and preprocess the images:
    - Iterate through each image file in the directories ○ Read the image using cv2.imread() and convert it to grayscale ○ Resize the image to 64x64 pixels using cv2.resize() ○ Normalize the pixel values to the range [0, 1] by dividing by 255 ○ Append the processed image to the corresponding image list

4. Concatenate the images and labels
5. Split the dataset into train and test sets
6. Define the CNN model:
    - Create a function build_model(hp) that takes a hyperparameter object hp as an argument

- Inside the function, define the architecture of the CNN using keras.Sequential()
- Configure the hyperparameters using hp.Int and hp.Choice for filters, kernel size, dense units, and learning rate
- Compile the model with the Adam optimizer, sparse categorical cross-entropy loss, and accuracy metric
- Return the model

7.Perform hyperparameter tuning:

- Create an instance of RandomSearch tuner with build_model as the model-building function, the objective set to 'val_accuracy', and the maximum number of trials (e.g., 5)
- Call tuner_search.search() with the train images and labels, number of epochs, and validation split to search for the best hyperparameters

8.Train the model:

- Fit the model to the train data using model.fit()
- Specify the number of epochs, validation split, and initial epoch if necessary

9.Evaluate the model

10.Make predictions

Summary of the model

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (16, 256, 256, 3)         0

 sequential_1 (Sequential)   (None, 256, 256, 3)       0

 conv2d (Conv2D)             (None, 254, 254, 32)      896

 max_pooling2d (MaxPooling2D  (None, 127, 127, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 125, 125, 64)      18496

 max_pooling2d_1 (MaxPooling  (None, 62, 62, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 60, 60, 64)        36928

 max_pooling2d_2 (MaxPooling  (None, 30, 30, 64)       0
 2D)

 conv2d_3 (Conv2D)           (None, 28, 28, 64)        36928

 max_pooling2d_3 (MaxPooling  (None, 14, 14, 64)       0
 2D)

 conv2d_4 (Conv2D)           (None, 12, 12, 64)        36928

 max_pooling2d_4 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 conv2d_5 (Conv2D)           (None, 4, 4, 64)          36928

 max_pooling2d_5 (MaxPooling  (None, 2, 2, 64)         0
 2D)

 flatten (Flatten)           (None, 256)               0

 dense (Dense)               (None, 64)                16448

 dense_1 (Dense)             (None, 3)                 195

=================================================================
Total params: 183,747
Trainable params: 183,747
Non-trainable params: 0
_________________________________________________________________
```

Fig- CNN Model Summary

After training for 20 epochs

```
Epoch 16/20
40/40 [==============================] - 64s 2s/step - loss: 0.1572 - accuracy: 0.9438 - val_loss: 1.0001 - val_accuracy: 0.760
6
Epoch 17/20
40/40 [==============================] - 63s 2s/step - loss: 0.1749 - accuracy: 0.9391 - val_loss: 0.8414 - val_accuracy: 0.784
2
Epoch 18/20
40/40 [==============================] - 64s 2s/step - loss: 0.1872 - accuracy: 0.9375 - val_loss: 0.6464 - val_accuracy: 0.760
6
Epoch 19/20
40/40 [==============================] - 64s 2s/step - loss: 0.1445 - accuracy: 0.9453 - val_loss: 0.9223 - val_accuracy: 0.732
4
Epoch 20/20
40/40 [==============================] - 63s 2s/step - loss: 0.1680 - accuracy: 0.9359 - val_loss: 0.8639 - val_accuracy: 0.647
9
```

```
scores = model.evaluate(test_ds)
```

```
5/5 [==============================] - 4s 393ms/step - loss: 0.7262 - accuracy: 0.7875
```

```
scores
```

```
[0.7262041568756104, 0.7875000238418579]
```

Fig- CNN Results

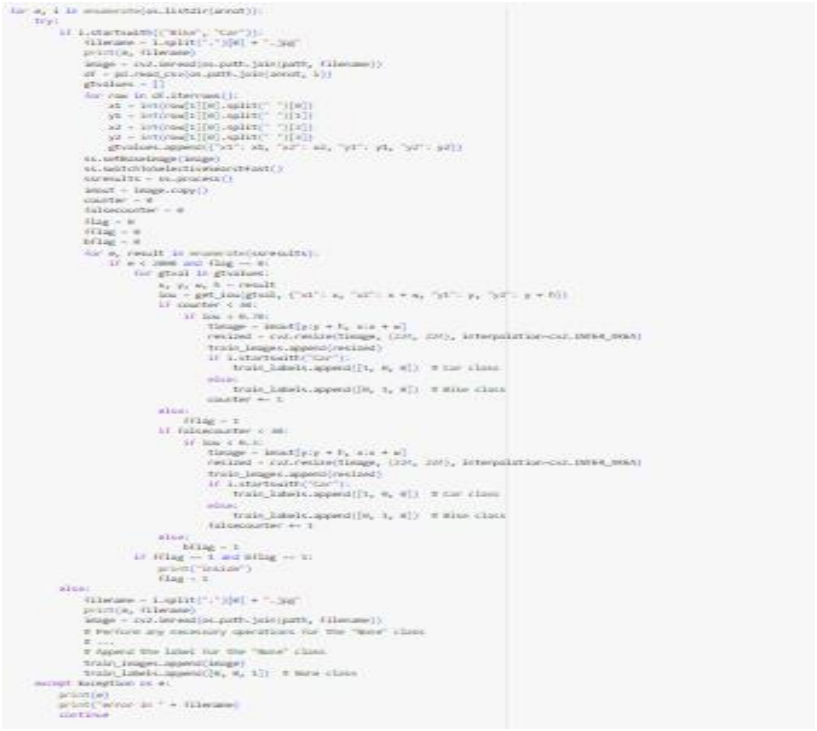
LOSS: 0.73

ACCURACY: 0.79

2.RCNN

Following steps were performed

1. First step is to import all the libraries which will be needed to implement R-CNN. We need cv2 to perform selective search on the images. To use selective search we need to download opencv-contrib-python. To download that just run **pip install opencv-contrib-python** in the terminal and install it from pypi.
2. Now we are initialising the function to calculate IOU (Intersection Over Union) of the ground truth box from the box computed by selective search.

3. 

The above code is pre-processing and creating the data-set to pass to the model. **As in this case we can have 3 classes. These classes are that whether the proposed region can be a foreground (i.e. Car and Bike) or a background**. So we will set the label of foreground (i.e. Airplane) as [1,0,0],[0,1,0] and the label of background as [0,0,1]. The following steps are being performed in the above code block.

1. Loop over the image folder and set each image one by one as the base for selective search using code **ss.setBaseImage(image)**

2. Initialising fast selective search and getting proposed regions using using code **ss.switchToSelectiveSearchFast()** and **ssresults = ss.process()**

3. Iterating over all the first 2000 results passed by selective search and calculating IOU of the proposed region and annotated region using the **get_iou()** function created above.

4. Now as one image can many negative sample (i.e. background) and just some positive sample (i.e. Car and Bike) so we need to make sure that we have good proportion of both positive and negative sample to train our

model. Therefore we have set that we will collect maximum of 30 negative sample (i.e. background) and positive sample (i.e. Car and Bike) from one image.

After running the above code snippet our training data will be ready.
List **train_images=[]** will contain all the images and **train_labels=[]** will contain all the labels marking Car and Bike images as [1,0,0], [0,1,0] and none images (i.e. background images) as [0,0,1].

4. After completing the process of creating the dataset we will convert the array to numpy array so that we can traverse it easily and pass the datatset to the model in an efficient way**.**
5. Now we will do transfer learning on the imagenet weight. We will import VGG16 model and also put the imagenet weight in the model.
6. In this part in the loop we are freezing the first 15 layers of the model. After that we are taking out the second last layer of the model and then adding a **2 unit softmax dense layer** as we have just 2 classes to predict i.e. foreground or background. After that we are compiling the model using **Adam optimizer with learning rate of 0.001**. We are using **categorical_crossentropy** as loss since the output of the model is categorical. Finally the summary of the model will is printed using model_final.summary().

7. After creating the model now we need to split the dataset into train and test set. Before that we need to one-hot encode the label. For that we are using **MyLabelBinarizer()** and encoding the dataset. Then we are splitting the dataset using **train_test_split** from sklearn. We are keeping 10% of the dataset as test set and 90% as training set.

8. Now we will use Keras **ImageDataGenerator** to pass the dataset to the model. We will do some augmentation on the dataset like horizontal flip, vertical flip and rotation to increase the dataset.

9. Now once we have created the model. We need to do prediction on that model. For that we need to follow the steps mentioned below:

1. pass the image from selective search.

2. pass all the result of the selective search to the model as input using **model_final.predict(img)**.

3. If the output of the model says the region to be a foreground image (i.e. airplane image) and if the confidence is above the defined threshold then create bounding box on the original image on the coordinate of the proposed region.

```
5/5 [==============================] - ETA: 0s - loss: 0.2523 - accuracy: 0.8938
Epoch 5: val_loss did not improve from 0.20194
5/5 [==============================] - 3s 664ms/step - loss: 0.2523 - accuracy: 0.8938 - val_loss: 0.2223 - val_accuracy: 0.8906
Epoch 6/10
5/5 [==============================] - ETA: 0s - loss: 0.2733 - accuracy: 0.8687
Epoch 6: val_loss did not improve from 0.20194
5/5 [==============================] - 5s 1s/step - loss: 0.2733 - accuracy: 0.8687 - val_loss: 0.2803 - val_accuracy: 0.8750
Epoch 7/10
5/5 [==============================] - ETA: 0s - loss: 0.4335 - accuracy: 0.8250
Epoch 7: val_loss did not improve from 0.20194
5/5 [==============================] - 4s 701ms/step - loss: 0.4335 - accuracy: 0.8250 - val_loss: 0.3736 - val_accuracy: 0.7812
Epoch 8/10
5/5 [==============================] - ETA: 0s - loss: 0.2401 - accuracy: 0.8938
Epoch 8: val_loss did not improve from 0.20194
5/5 [==============================] - 3s 667ms/step - loss: 0.2401 - accuracy: 0.8938 - val_loss: 0.2686 - val_accuracy: 0.8438
Epoch 9/10
5/5 [==============================] - ETA: 0s - loss: 0.4201 - accuracy: 0.8125
Epoch 9: val_loss did not improve from 0.20194
5/5 [==============================] - 4s 873ms/step - loss: 0.4201 - accuracy: 0.8125 - val_loss: 0.2741 - val_accuracy: 0.8594
Epoch 10/10
5/5 [==============================] - ETA: 0s - loss: 0.3889 - accuracy: 0.8687
Epoch 10: val_loss did not improve from 0.20194
5/5 [==============================] - 6s 1s/step - loss: 0.3889 - accuracy: 0.8687 - val_loss: 0.3245 - val_accuracy: 0.8906
```

Fig- RCNN Results

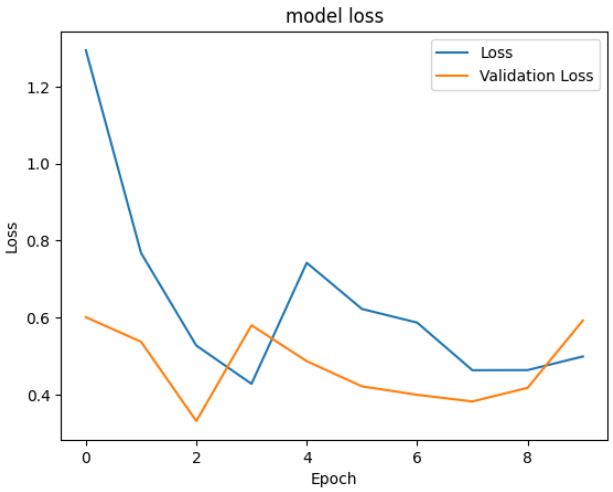


Fig- RCNN Loss Graph

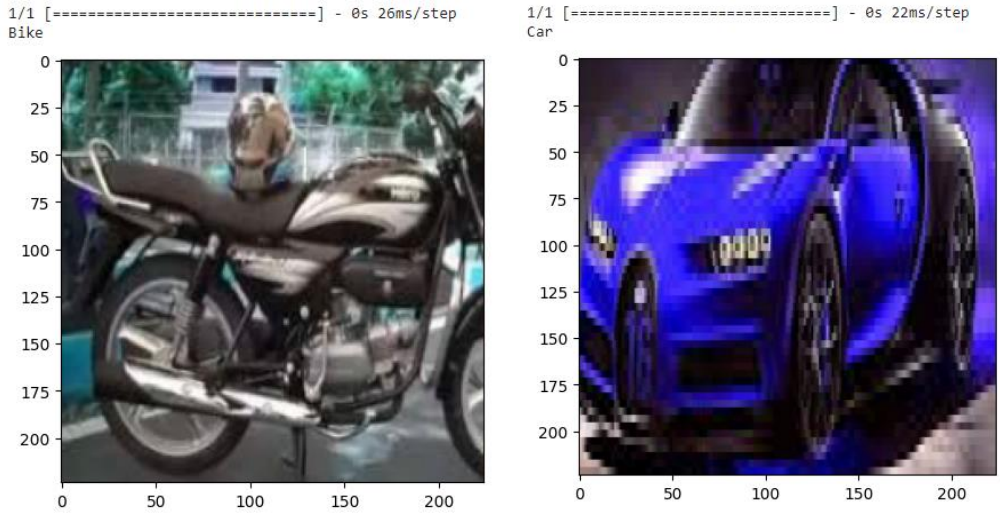


Fig – RCNN Predictions

Accuracy: 0.86

3.Mask RCNN

Following steps were performed

1. Load the dataset
2. Annotate and Label the dataset in Bounding Box Format.
3. Augment and Resize your dataset like Horizontal flip, brightness control, apply of shearing etc. (you can do it in various online software like Roboflow or You can do it in your code)
4. Export the Annotated Dataset in COCO JSON Format.
5. Split Your dataset for test, train, validation
6. Import Detectron2 Model For Mask-RCNN and Fast-RCNN
7. Import Necessary Libraries
8. Import Dataset and Annotations
9. Visualize your Dataset
10. Configure Mask RCNN with Detectron 2 Model and do Hyperparameter Tuning for model training
11. Mention output Directory for trained model
12. Visualize the model parameters
13. Do training of this model for 2000 number of epochs and 20 itretions

14. After Successful completion of training fetch the following Results by tensorboard Library:

Masks:

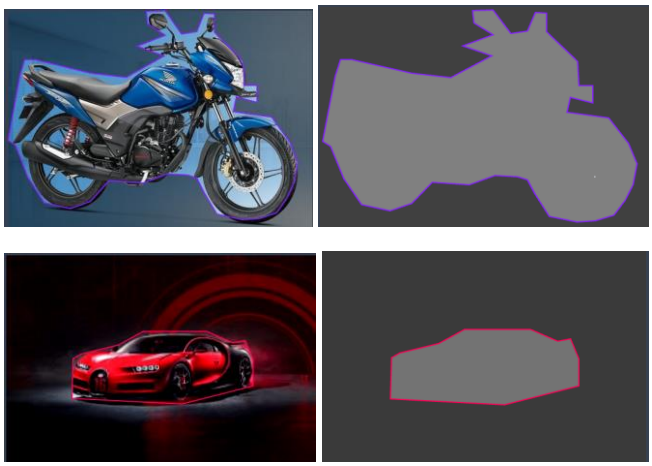
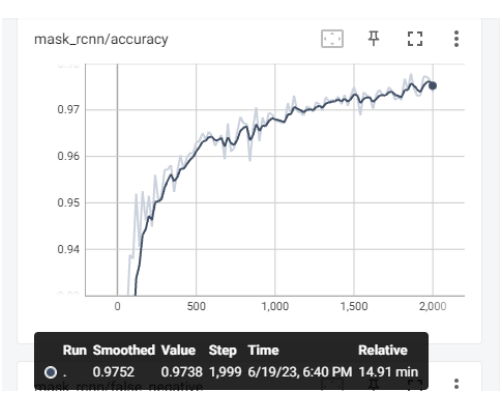
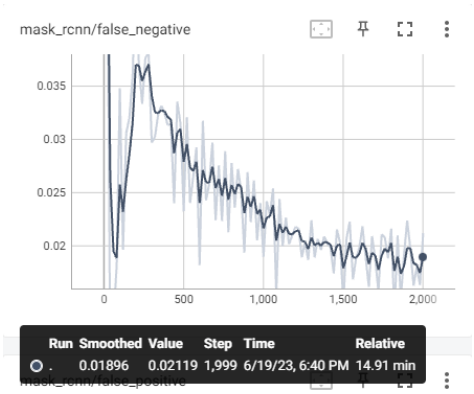
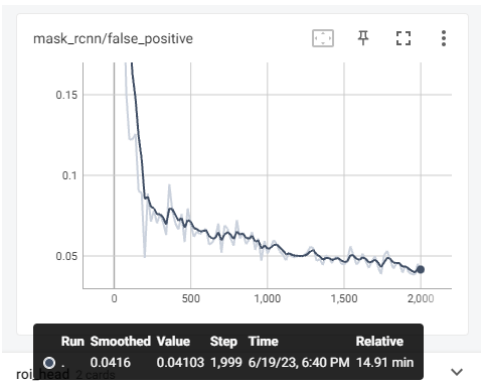


Fig- Masks for Different Objects







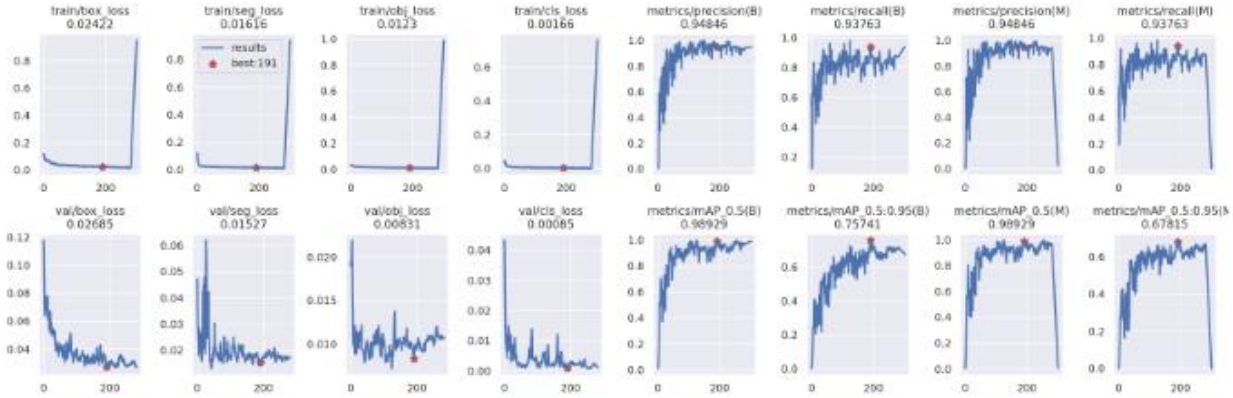


Fig – Mask RCNN Results

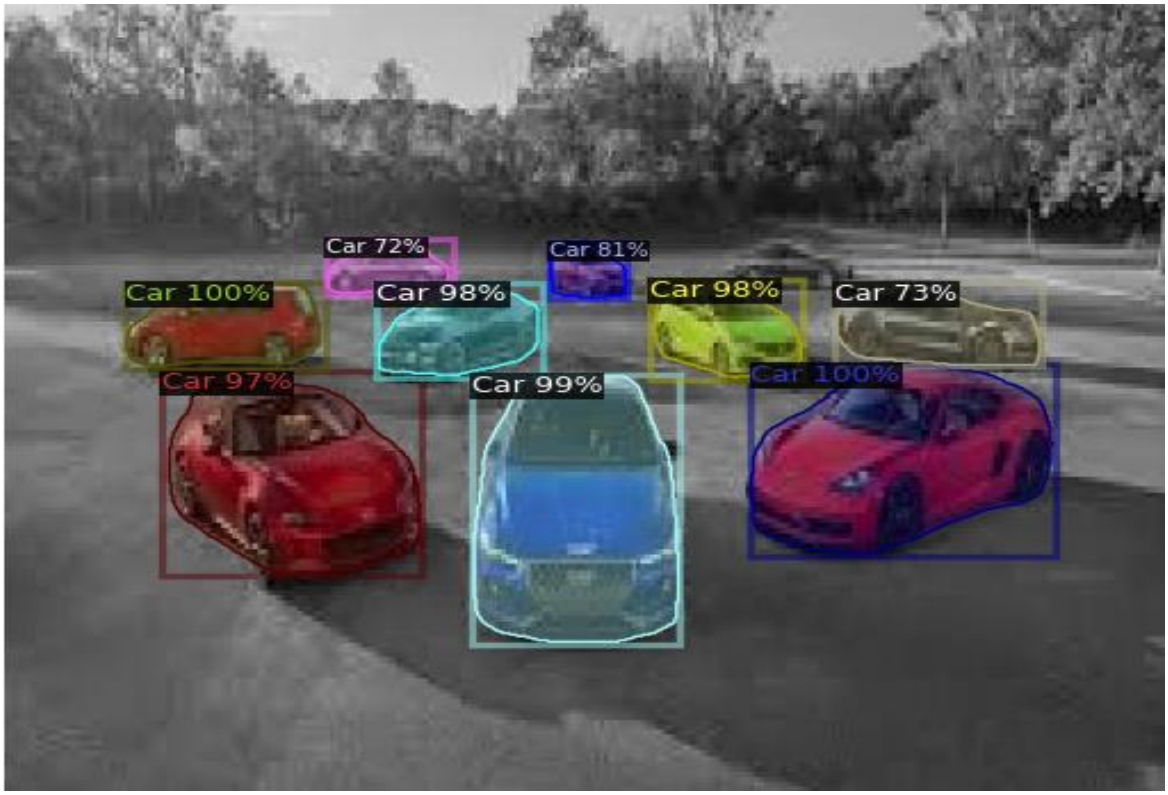


Fig – Prediction of Cars by Mask RCNN

Fig- Prediction of Bike by Mask RCNN

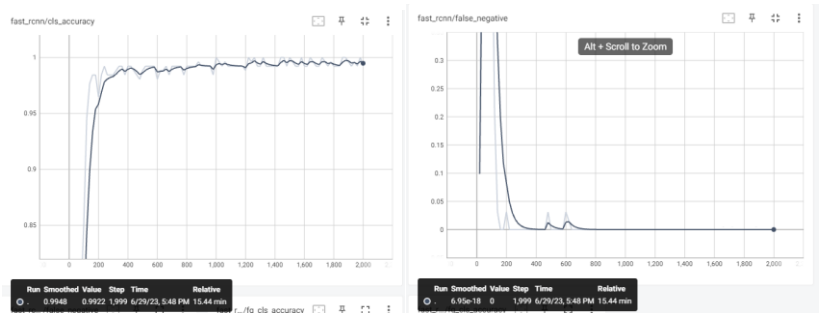
4.Fast R-CNN:

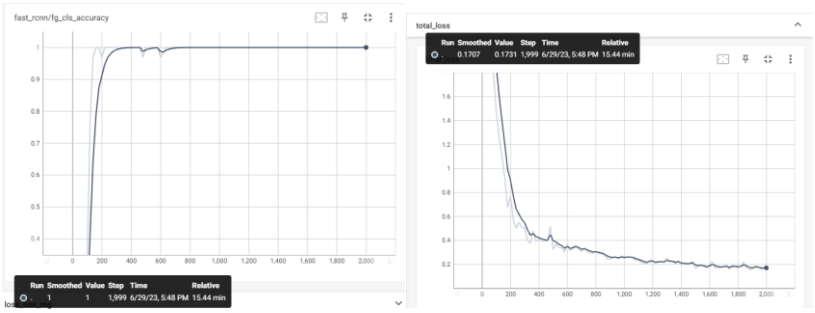
CodeLink:

Following steps were performed

15. Load the dataset
16. Annotate and Label the dataset in Bounding Box Format.
17. Augment and Resize your dataset like Horizontal flip, brightness control, apply of shearing etc. (you can do it in various online software like Roboflow or You can do it in your code)
18. Export The Annotated Dataset in COCO JSON Format.
19. Split Your dataset for test, train, validation
20. Import Detectron2 Model For Mask-RCNN and Fast-RCNN
21. Import Necessary Libraries

22. Import Dataset and Annotations
23. Visualize your Dataset
24. Configure Fast RCNN with Detectron 2 Model and do Hyperparameter Tuning for model training
25. Mention output Directory for trained model
26. Visualize the model parameters
27. Do training of this model for 2000 number of epochs and 20 itretions
28. After Successful completion of training fetch the following Results by tensorboard Library:





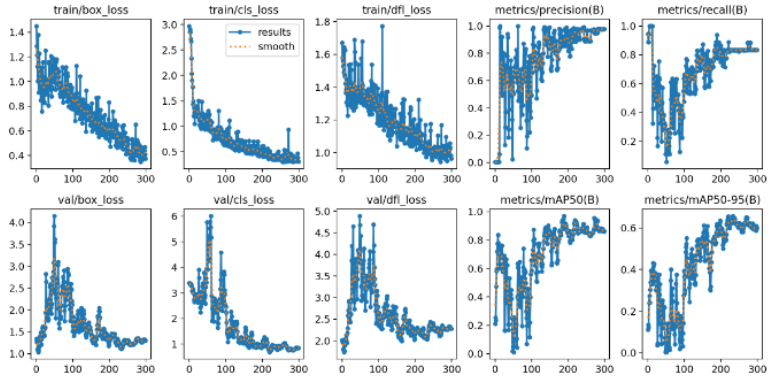


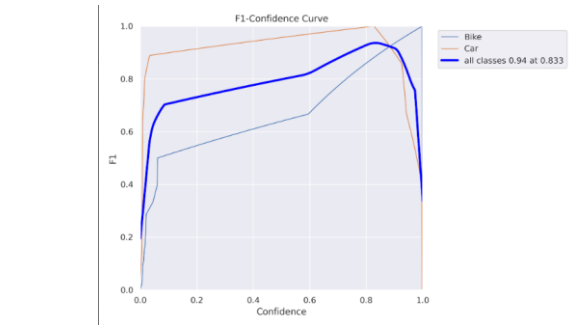
Fig – Fast RCNN Results

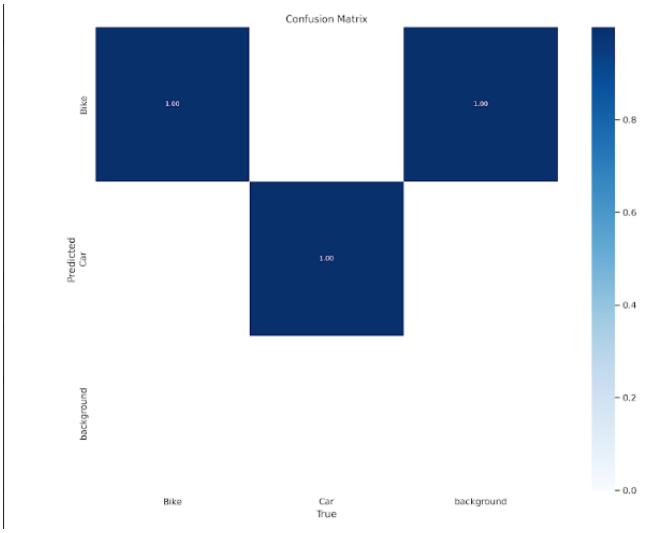
Fig – Fast RCNN Predictions

5.YOLO V8:

29. Load the dataset
30. Annotate and Label the dataset in Yolo V8 Format.
31. Augment and Resize your dataset like Horizontal flip, brightness control, apply of shearing etc. (you can do it in various online software like Roboflow or You can do it in your code)
32. Export The Annotated Dataset in Yolo COCO JSON Format.
33. Split Your dataset for test, train, validation
34. Import Necessary Libraries

35. Import Dataset and Anotations
36. Apply Yolo V8 model to train your dataset
37. After Successful completion of training fetch the following Results:





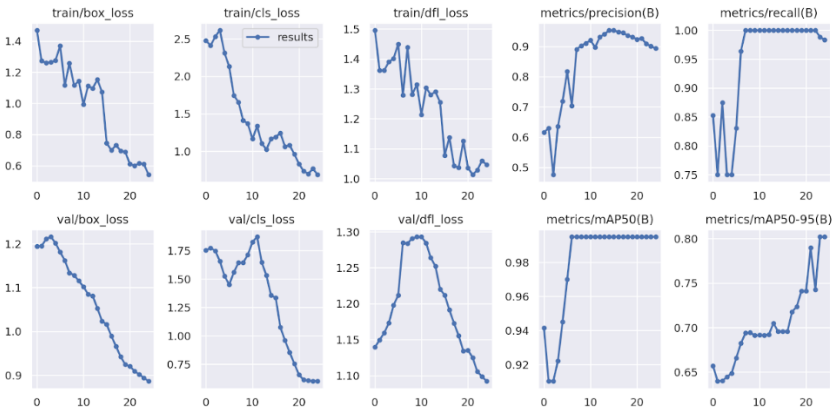


Fig - YOLO V8 Results

Fig-Predictions by Yolo V8