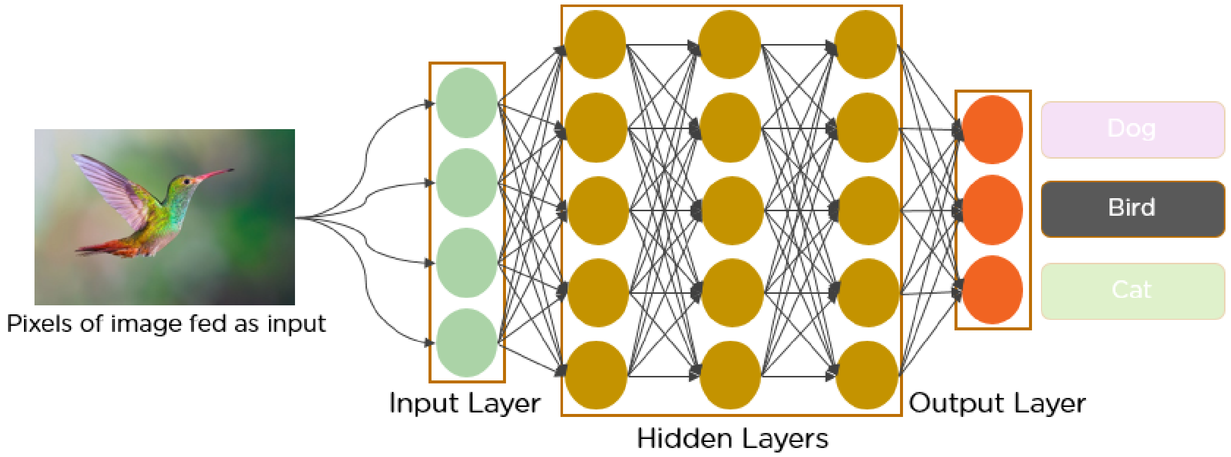
Deep Learning algorithms to demonstrate Object Detection

Algorithms used for Object Detection

1. CNN (Convolutional Neural Network)

Convolutional neural networks (CNNs) are a form of artificial neural network and deep learning technique that are typically employed for image recognition and processing applications.

- The main part of a CNN is its **convolutional layers**, where filters are applied to the input image to extract features like edges, textures, and forms.
- The output of the convolutional layers is then sent through **pooling layers**, which are employed to down-sample the feature maps and retain the most crucial data while lowering the spatial dimensions.
- One or more **fully connected layers** are then applied to the output of the pooling layers in order to predict or categorize the image.



Multiple layers of a CNN are possible, and each layer trains the CNN to recognize the various elements of an input image. Each image is given a **filter or kernel** to create an output that gets better and more detailed with each layer. The filters may begin as basic characteristics in the lower layers.

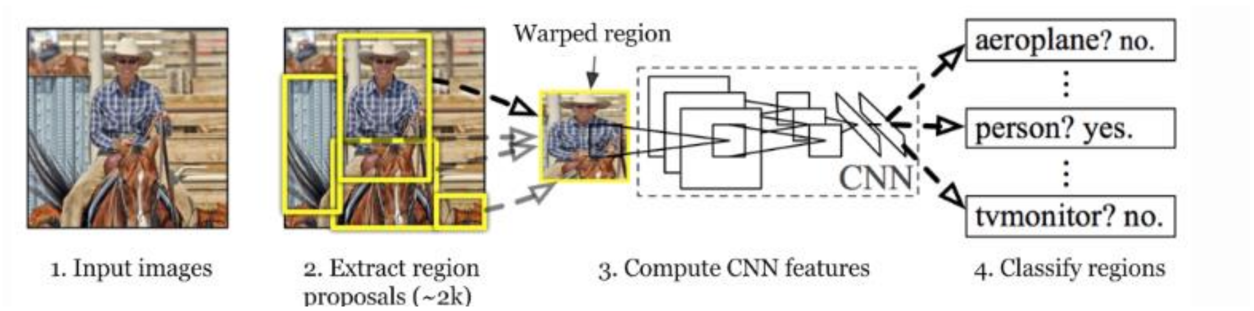
At each successive layer, the filters increase in complexity to check and identify features that uniquely represent the input object. Thus, the output of each convolved image -- the partially recognized image after each layer --

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.

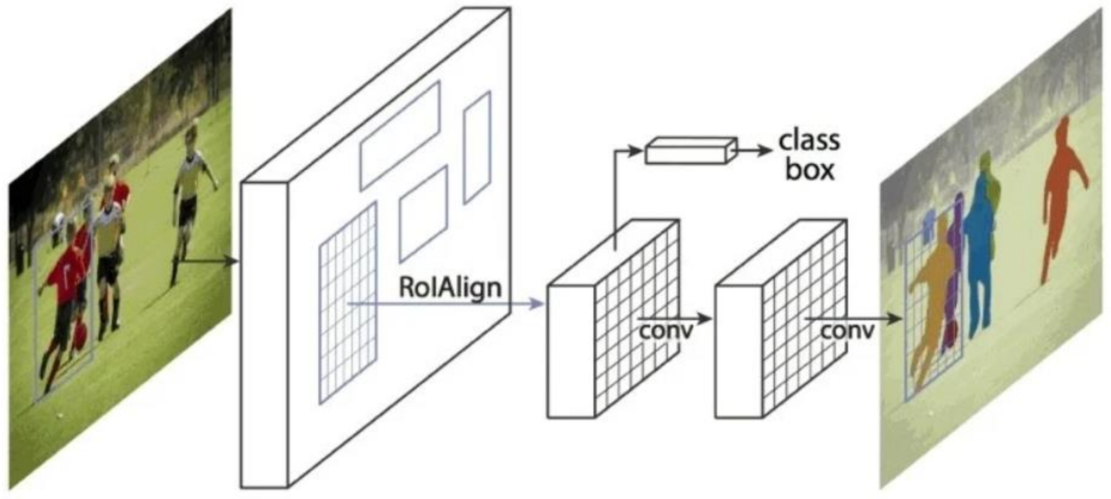
3. Mask R-CNN

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



Dataset used

Source: https://public.roboflow.com/object-detection/vehicles-openimages

This dataset contains 627 images of various vehicle classes for object detection. Out of these images 100 were chosen for the training dataset and 30 were chosen for the validation dataset. These images are derived from the open source computer vision datasets.

Annotation were performed on each image and they were extracted in COCO JSON format.

Classes in the dataset:

- Car
- Bus
- Truck
- Motorcycle
- Ambulance

Size info of images:
416x416

3.17Kb

96dpi

24 bit

Hardware used: Intel® Core™ i5- 10210 CPU @ 1.60GHz

Intel® UHD Graphics

Comparison of algorithms

Aspect	**CNN**	**R-CNN**	**Mask R-CNN**
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
Input	Fixed-size images	Variable-size images	Variable-size images
Output	Image classification	Object proposals, class labels, and bounding boxes	Object proposals, class labels, bounding boxes, and segmentation masks
Advantages	Efficient training and inference Fast inference time	Accurate object localization Improved detection performance	Accurate object localization and instance segmentation Improved detection and segmentation performance

Disadvantages	Limited to fixed-size inputs No object localization No instance segmentation	Slower inference compared to CNN Training can be time-consuming Not suitable for real-time applications	Slower inference compared to CNN and R-CNN Training can be time-consuming Not suitable for real-time applications
Accuracy/Precision comparison on dataset	Accuracy: 0.88 Loss: 0.4367	Accuracy: 0.8543 Loss: 0.276	mAP(Mean Average Precision) : 0.8

Implementation of algorithms:

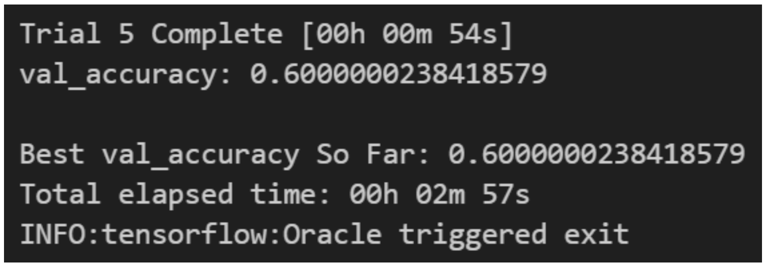
1.CNN

Following steps were performed

1. Import the necessary libraries
2. Define the paths to the image directories:
    - cars, bus, ambulance, motorcycle, and truck
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

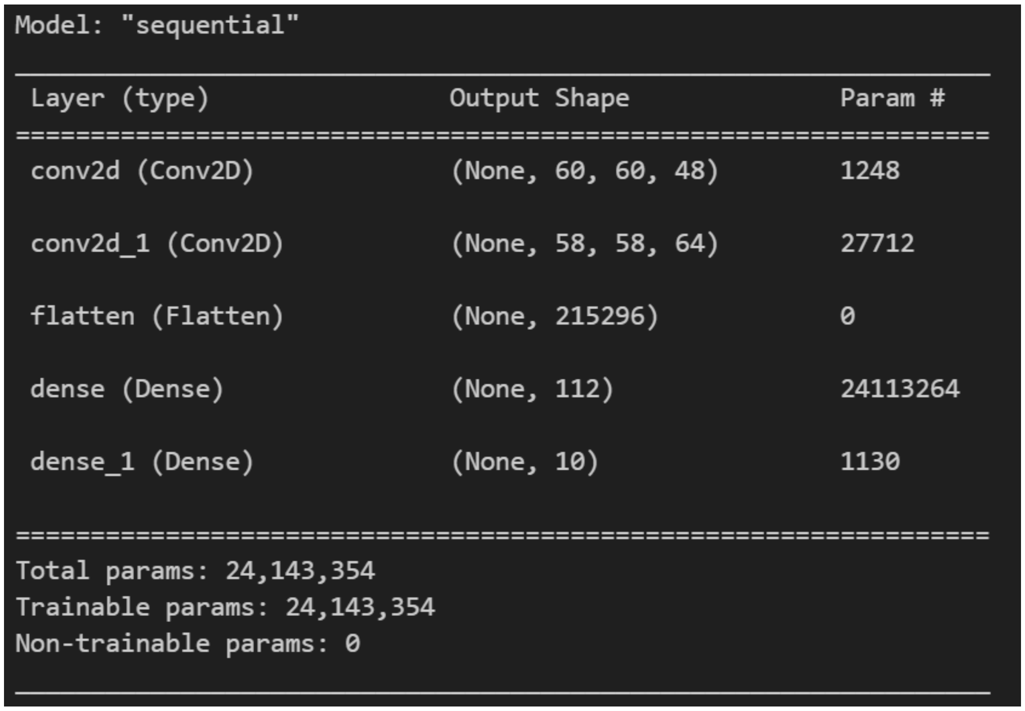
- Create a function build_model(hp) that takes a hyperparameter object hp as an argument
    - Inside the function, define the architecture of the CNN using keras.Sequential()
    - Configure the hyperparameters using hp.Int and hp.Choice for filters, kernel size, dense units, and learning rate
    - 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
    - Get the best model from the tuner using tuner_search.get_best_models()
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

After tuning



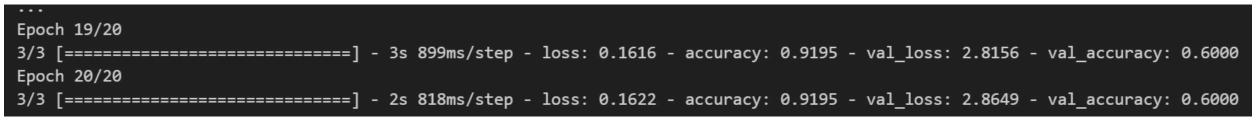
```
Trial 5 Complete [00h 00m 54s]
val_accuracy: 0.6000000238418579

Best val_accuracy So Far: 0.6000000238418579
Total elapsed time: 00h 02m 57s
INFO:tensorflow:Oracle triggered exit
```

Summary of the model



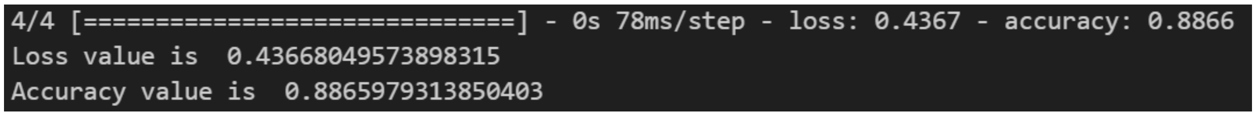
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 60, 60, 48)        1248

 conv2d_1 (Conv2D)           (None, 58, 58, 64)        27712

 flatten (Flatten)           (None, 215296)            0

 dense (Dense)               (None, 112)               24113264

 dense_1 (Dense)             (None, 10)                1130

=================================================================
Total params: 24,143,354
Trainable params: 24,143,354
Non-trainable params: 0
_________________________________________________________________
```

After training for 20 epochs



```
...
Epoch 19/20
3/3 [==============================] - 3s 899ms/step - loss: 0.1616 - accuracy: 0.9195 - val_loss: 2.8156 - val_accuracy: 0.6000
Epoch 20/20
3/3 [==============================] - 2s 818ms/step - loss: 0.1622 - accuracy: 0.9195 - val_loss: 2.8649 - val_accuracy: 0.6000
```

Evaluation



```
4/4 [==============================] - 0s 78ms/step - loss: 0.4367 - accuracy: 0.8866
Loss value is  0.43668049573898315
Accuracy value is  0.8865979313850403
```

Loss: 0.43

Accuracy: 0.88

2.RCNN

Following steps were performed

1. Import the necessary libraries
2. Define the paths to the image directories:
    - cars, bus, ambulance, motorcycle, and truck

3. Load and preprocess the images
4. Concatenate the images and labels
5. Split the dataset into train and test sets
6. Extract features using a pre-trained VGG16 model:
    - Load the VGG16 model using tensorflow.keras.applications.vgg16.VGG16()
    - Create a function extract_features(images, proposals) that takes a list of images and corresponding proposals as input
    - Iterate through each image and its corresponding proposals
    - Extract the region of interest (ROI) from the image based on the proposal coordinates
    - Resize the ROI to match the input size of VGG16 (e.g., 416x416 pixels)
    - Preprocess the ROI according to VGG16 requirements using preprocess_input()
    - Extract features from the ROI using the VGG16 model and flatten the feature vector
    - Append the feature vector to a list (features)
    - Return the list of features
7. Train a classifier (e.g., SVM) on the extracted features
    - Create an instance of the classifier (e.g., SVC())
    - Fit the classifier to the training features and labels using classifier.fit()
8. Make predictions on the test set:
    - Use the trained classifier to predict the labels for the test features using classifier.predict()
9. Calculate accuracy:
    - Calculate the accuracy of the predictions by comparing them to the ground truth labels using accuracy_score() from sklearn.metrics

Accuracy: 0.8543

3.Mask RCNN

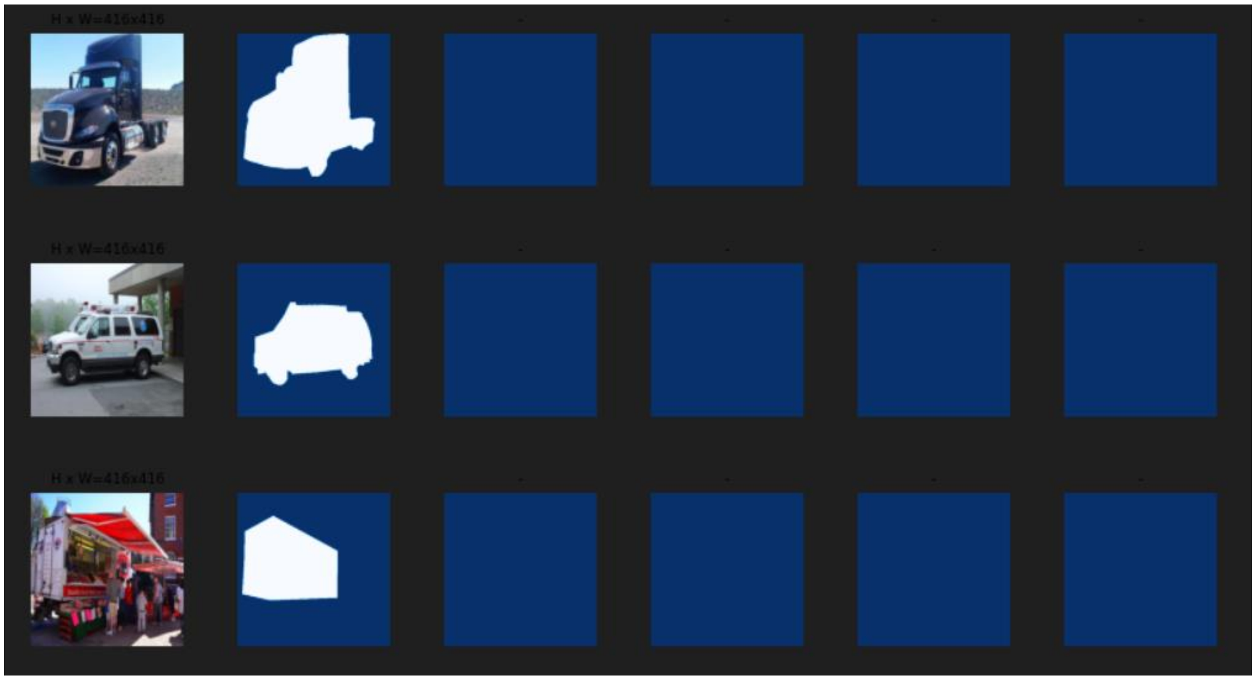
Following steps were performed

1. Import the necessary libraries and set up the environment

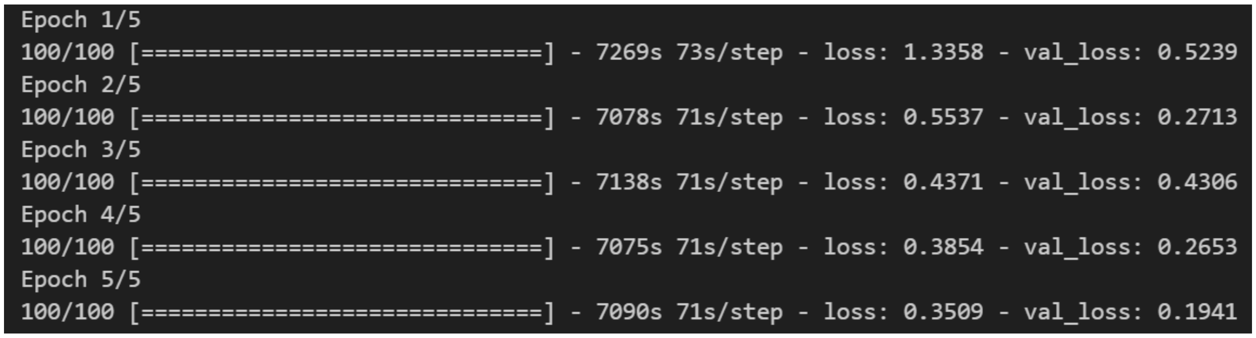
2. Define a custom dataset class (CocoLikeDataset) that extends the utils.Dataset class:
    - This class is responsible for loading the COCO-like dataset from a JSON file and preparing it for training or validation.
    - The load_data method loads the dataset from a JSON file and adds the class names to the dataset using the add_class method.
    - The load_mask method loads instance masks for a given image, converting them into a bitmap format expected by Mask R-CNN.
3. Load the training and validation datasets:
    - Create instances of CocoLikeDataset for the training and validation datasets.
    - Use the load_data method to load the dataset from the corresponding JSON file and images directory.
    - Use the prepare method to prepare the dataset for training or validation.
4. Visualize the dataset:
    - Iterate through a few randomly selected image IDs from the training dataset.
    - Load the image, masks, and class IDs using the load_image and load_mask methods.
    - Use the display_top_masks function from mrcnn.visualize to display the image with masks and class labels.
5. Define the configuration for the Mask R-CNN model:
    - Create a custom configuration class that extends Config.
    - Set the configuration parameters such as the name of the configuration, number of classes, and steps per epoch.
    - Display the configuration using the display method.
6. Set up the model:
    - Set the root directory for logs and trained models.
    - Load the pre-trained weights from the COCO dataset, excluding the output layers.
    - Create an instance of the Mask R-CNN model in training mode, specifying the model directory and configuration.
7. Train the model:
    - Call the train method of the model, passing the training dataset, validation dataset, learning rate, number of epochs, and layers to train.

- The model will start training, optimizing the network weights for the given number of epochs

Masks:

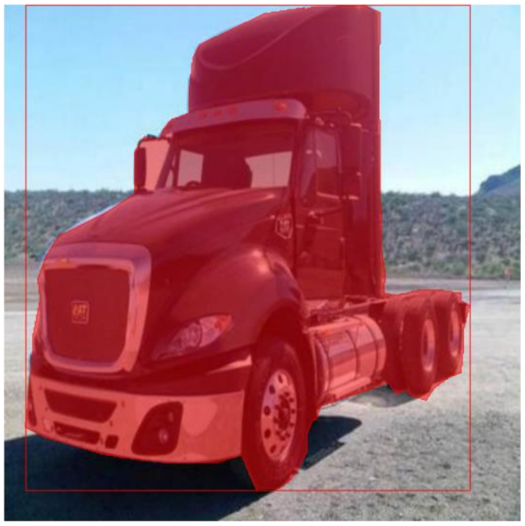


After training for 5 epochs



```
Epoch 1/5
100/100 [==============================] - 7269s 73s/step - loss: 1.3358 - val_loss: 0.5239
Epoch 2/5
100/100 [==============================] - 7078s 71s/step - loss: 0.5537 - val_loss: 0.2713
Epoch 3/5
100/100 [==============================] - 7138s 71s/step - loss: 0.4371 - val_loss: 0.4306
Epoch 4/5
100/100 [==============================] - 7075s 71s/step - loss: 0.3854 - val_loss: 0.2653
Epoch 5/5
100/100 [==============================] - 7090s 71s/step - loss: 0.3509 - val_loss: 0.1941
```

mAP value: 0.8

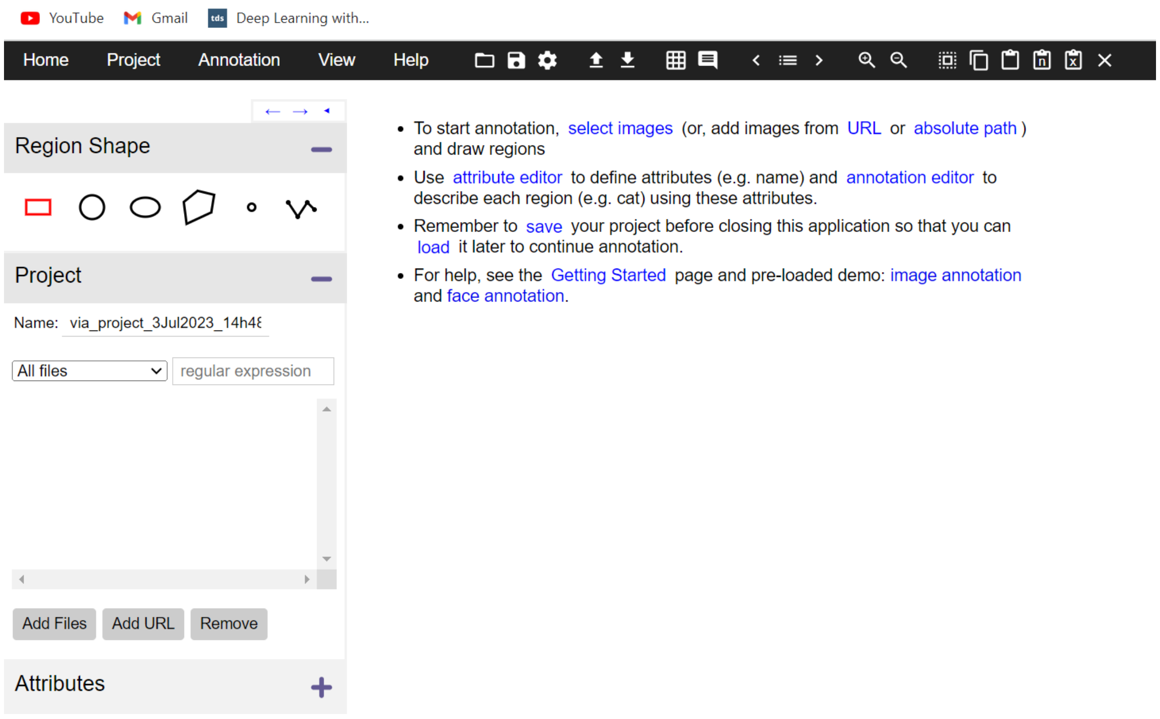


Performing Annotations:

Annotations of images was done using the free, open source VGG Image Annotator. It is small and lightweight to use and we can run it in our web browser. We can draw bounding boxes, or polygons in our images or even videos for our dataset for various algorithms

To start the project:

1.Go to the online portal for the application which is an HTML page
http://www.robots.ox.ac.uk/~vgg/software/via/via.html?ref=blog.roboflow.com



Name: via_project_3Jul2023_14h48

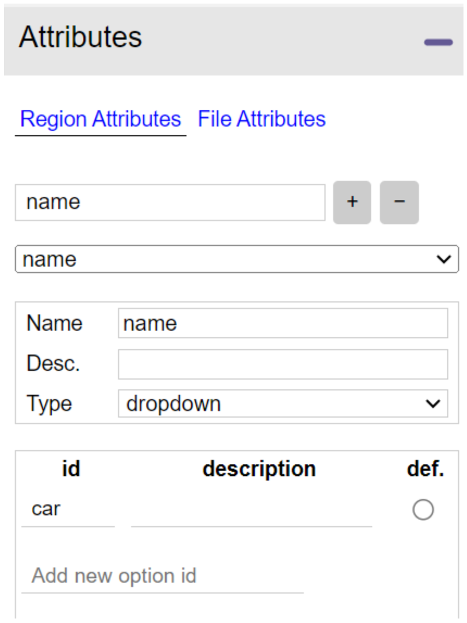
All files | regular expression

- To start annotation, select images (or, add images from URL or absolute path) and draw regions
- Use attribute editor to define attributes (e.g. name) and annotation editor to describe each region (e.g. cat) using these attributes.
- Remember to save your project before closing this application so that you can load it later to continue annotation.
- For help, see the Getting Started page and pre-loaded demo: image annotation and face annotation.

2.Go to attributes section and give the ‘Attribute Name’ and click on ‘+’ button

3.Go to type dropdown and select ‘dropdown’

4.Enter an id under ‘add new id’ option and keep on entering ids by pressing Enter key



name

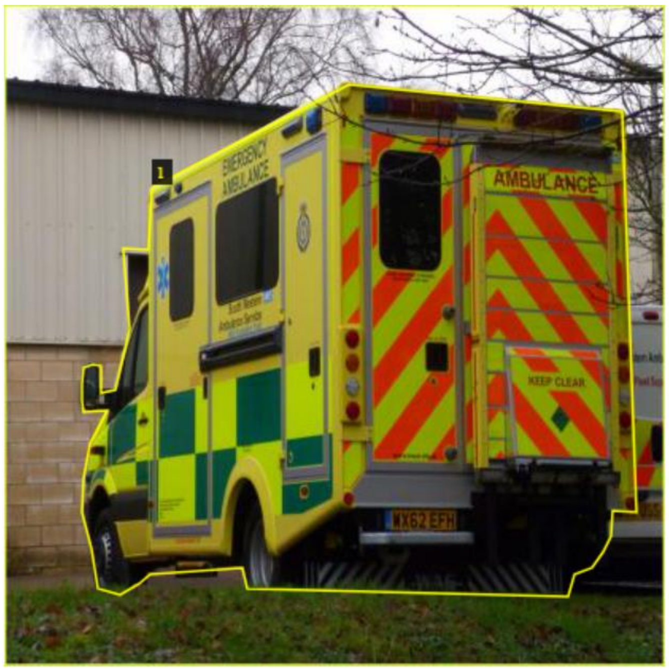
name

Name	name
Desc.	
Type	dropdown

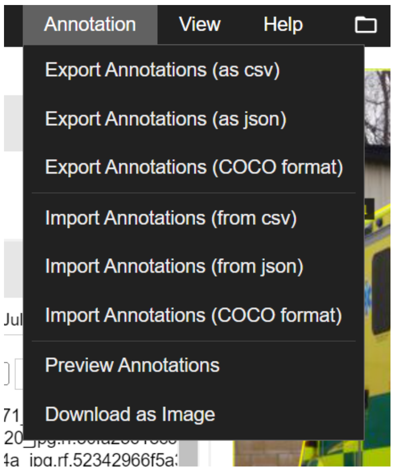
id	description	def.
car		○
Add new option id		

5.Click on Add Files button and add all the files we want to add in our project.

6.Select the region shape (eg polygon) and then start dragging mouse around the regions you need. After that select the class of the region. In this way annotate all the images.



7.After annotating all images go to Annotations section and export the annotations in the format we want to



Annotation
View
Help
Export Annotations (as csv)
Export Annotations (as json)
Export Annotations (COCO format)
Import Annotations (from csv)
Import Annotations (from json)
Import Annotations (COCO format)
Preview Annotations
Download as Image