Group 5: Dhruv Patel, Fiona Lobo

Real-Time Instance Segmentation Using Mask RCNN

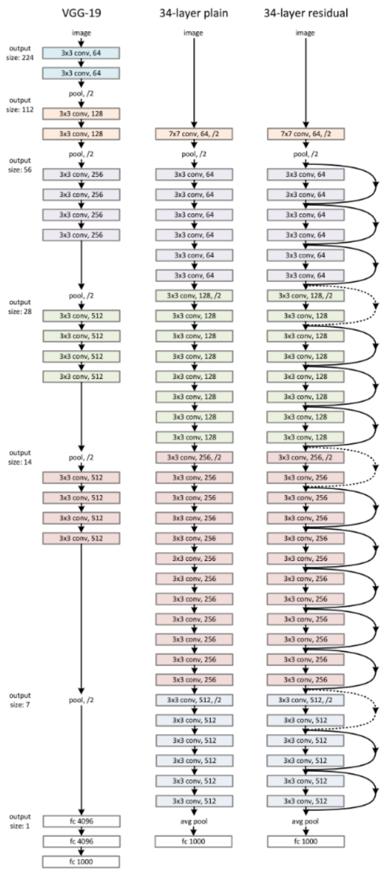
Introduction:

The most intriguing advancements brought by deep learning and neural networks are in the field of computer vision. We associate any problem that has an image or a camera input to encompass problems within computer vision. Self-driving cars, Mars Exploration Rovers, Facial Recognition Systems, Object Detection and Augmented Reality are just a few break throughs in the field. In this project we will look at a new type of neural network architecture known as Mask RCNN.

Mask RCNN:

Mask RCNN works towards the problem of instance segmentation. The process of precise detection and delineation of each distinct object of interest in an image. First let's discuss the backbone architecture on which the Mask RCNN is based:

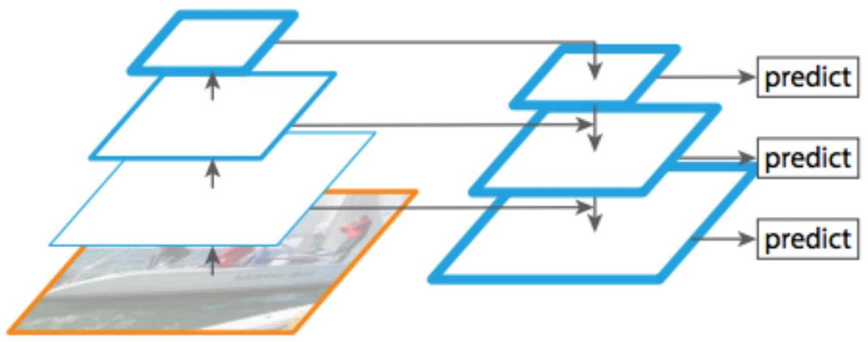
1. RESNET 101:



The backbone of the model consists of a RESNET101 (even RESNET50 can be used). It serves as a feature extractor. The starting layers detect low level features like corners and edges of the image. As the layers increase it begins to detect objects like a table, person or cell phone called higher level features. The image is converted to feature map of shape 32*32*2048. This becomes the input for the next phase.

2. Feature Pyramid Network (FPN):
   FPN is a modified network architecture for object detectors (i.e. bounding box detectors). The architecture aggregates features from many scales (i.e. before each pooling layer) to detect both small and large object. The network is shaped similar to an hourglass.

Feature Pyramid Network



Source: Feature Pyramid Networks paper

They have two branches. The first one is similar to any normal network: Convolutions and pooling. The exact choice of convolutions (e.g. how many) and pooling is determined by the used base network (e.g. ~50 convolutions with ~5x pooling in ResNet-50). The second branch starts at the first one's output. It uses nearest neighbor up sampling to re-increase the resolution back to the original one. It does not contain convolutions. All layers have 256 channels. There are connections between the layers of the first and second branch. These connections are simply 1x1 convolutions followed by an addition (similar to residual connections). Only layers with similar height and width are connected.

Integration of FPN with Faster-RCNN:

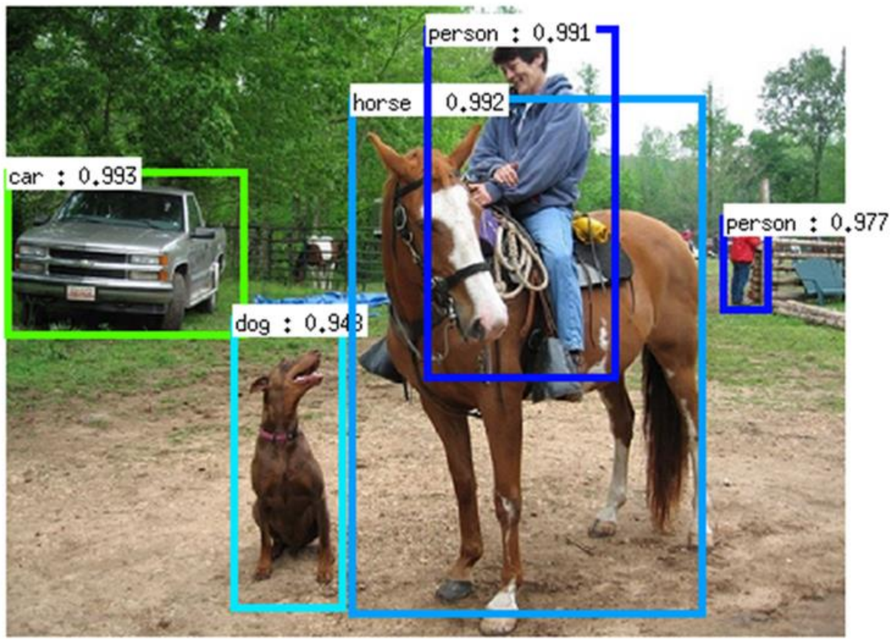
- The output of the above is given to the RPN on their second branch.
- While usually an RPN is applied to a single feature map of one scale, in their case it is applied to many feature maps of varying scales.

- The RPN uses the same parameters for all scales.
- They use anchor boxes, but only of different aspect ratios, not of different scales (as scales are already covered by their feature map heights/widths).
- Ground truth bounding boxes are associated with the best matching anchor box (i.e. one box among all scales).
- Everything else is the same as in Faster R-CNN.

So, to break it down into parts for better understanding, instance segmentation is a combination of two problems.

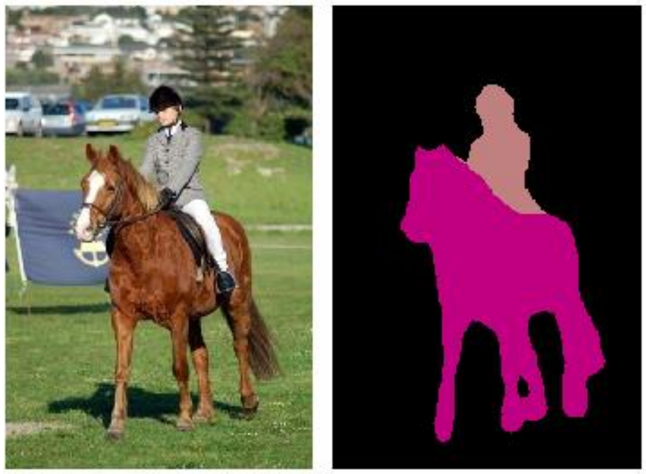
1. Object detection:

   It is classifying a variable number of objects in an image. The number of objects to classify can differ from image to image as different images have different number of objects in them. The image below depicts a crisp example of object detection. It specifies the bounding boxes, which are the different rectangular boxes around the object, the object class (person, dog, horse etc) and the accuracy percentage of each of the class.



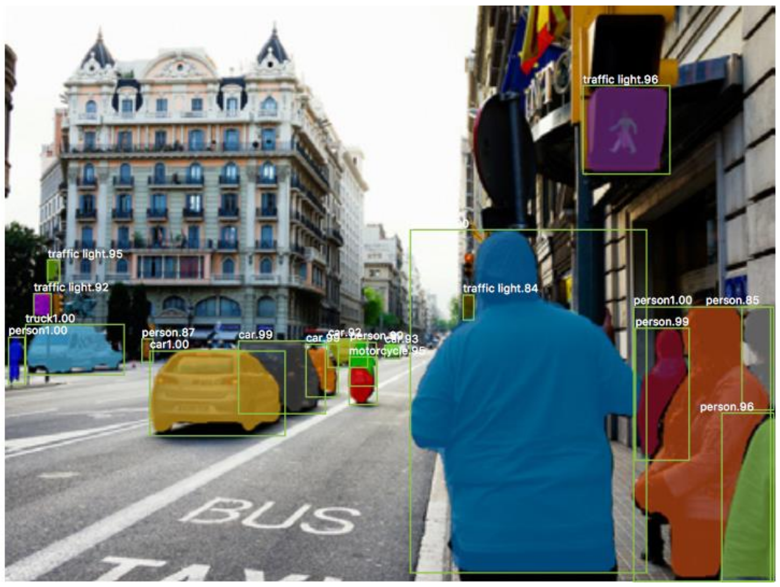
2. Semantic Segmentation:

   It is the understanding of an image at the pixel level. We aim at assigning an object class to each pixel in an image. For example let's consider this image:



In the above image, apart from recognizing the horse and the person riding it we have to delineate the boundaries of each object.

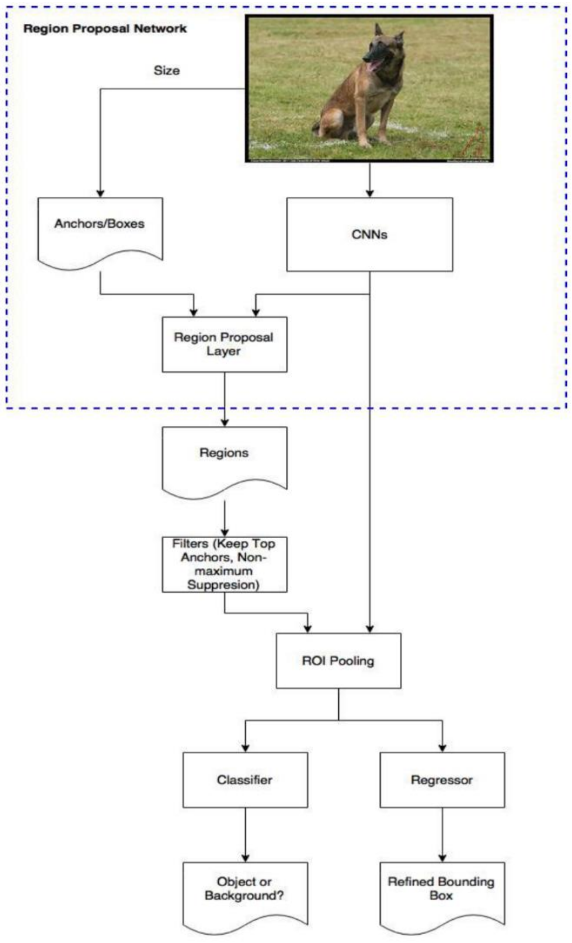
Using object detection and semantic segmentation together, we get instance segmentation:



In the above image, the bounding box is created from object detection and the shaded mask is created from semantic segmentation.
Now that we have had an intuition of Mask RCNN let's have a look at the architecture behind it.

Mask RCNN Architecture:
Since Mask RCNN is a combination of two phases, there are two parts of the architecture:

1. Faster R-CNN:



The Architecture of Faster R-CNN

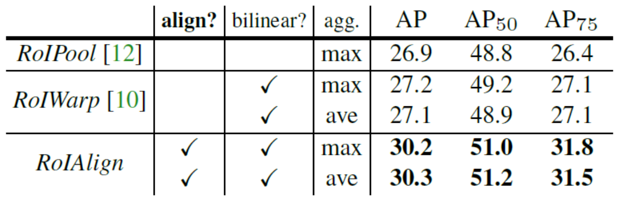
Faster R-CNN is used for the object detection phase of Mask RCNN. But first, what is RCNN?
It is an approach to bounding box object detection thus creating objects of interests or technically Region of Interests (ROIs).
Faster RCNN is an advancement to RCNN which uses Regional Proposal Network (RPN). It detects objects in two stages:

a. Determine the bounding box and hence determine the Region of Interests (ROIs)

b. Now for each ROI we determine the class label of the object which is done by ROI Pooling.

ROI Pooling is not much efficient because there is a possibility of data loss as the stride used is quantized.

Hence a different approach is taken here, it is known as ROI Align. Here the stride is not quantized as it uses bilinear interpolation to calculate the average of the maximum value of every cell thus leading to prevention of data loss. Also by addressing the loss and misalignment problems of RoIPooling , the new RoIAlign leads to improved results. RoIAlign is thus better than RoIPool as it allows us to preserve spatial pixel to pixel alignment for every Region of Interest without any information loss as there is no quantization. Below is the Average Precision (AP) comparison of RoIPool and RoIAlign.



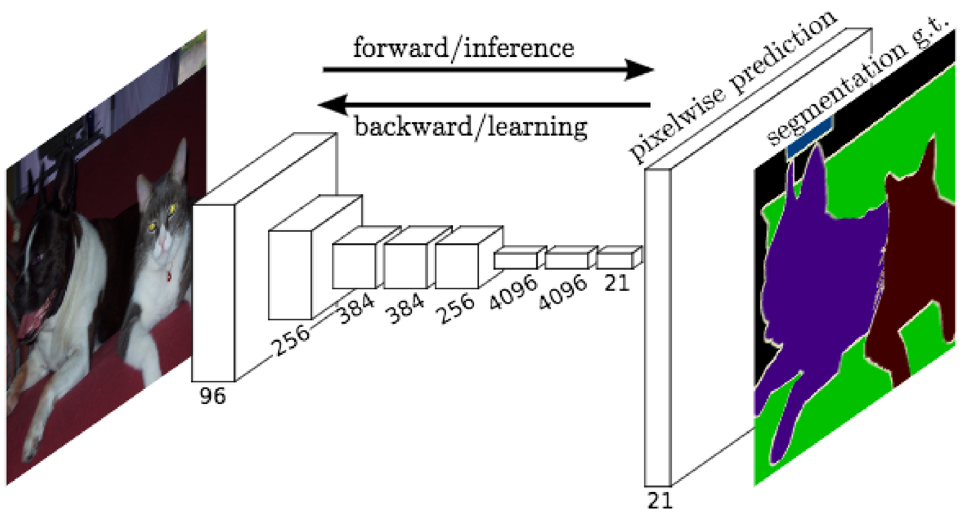
	align?	bilinear?	agg.	AP	AP_{50}	AP_{75}
RoIPool [12]			max	26.9	48.8	26.4
RoIWarp [10]		✓	max	27.2	49.2	27.1
		✓	ave	27.1	48.9	27.1
RoIAlign	✓	✓	max	**30.2**	**51.0**	**31.8**
	✓	✓	ave	**30.3**	**51.2**	**31.5**

(c) **RoIAlign** (ResNet-50-C4): Mask results with various RoI layers. Our RoIAlign layer improves AP by ~3 points and AP_{75} by ~5 points. Using proper alignment is the only factor that contributes to the large gap between RoI layers.

Conceptually Mask RCNN is similar to Faster RCNN. It additionally outputs an object mask using pixel to pixel alignment. This mask is a binary mask outputted for each Region of Interest (RoI). Much overhead is not observed while computing these masks as it is done in parallel with the bounding box creation and classification. Let's understand this:

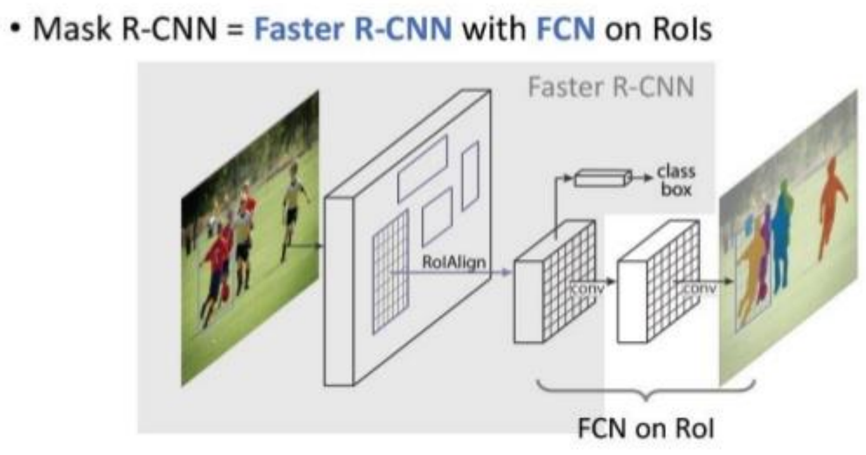
Consider a Region of Interest of m x m pixels. Let's say it consists of k possible objects that can be in that region. For example, in an image if we were trying to categorize humans, dogs and cats, then k would be equal to 3. For each type k, a binary mask of m x m is constructed. It is found that while computing a mask a loss of km^2 is occurred. This is different from the typical approach of constructing a single mask from k classes as the would compete in the mask. This lack of competition is key to good performance in instance segmentation.

2. Semantic Segmentation:



In this phase Fully Convolution Networks are used to predict the masks around from each
Region of Interest. But why use convolution layers for that? It's because convolutional layers retain spatial orientation that is important as we don't want information loss. Here the traditional fully connected layers won't here because the spatial orientation of pixels with respect to each other is lost as they are squished together to form a feature vector.
FCN does dense prediction which is responsible for pixelwise class prediction. First it takes the RoIs from the Regional Proposal Network and then by using series of deep convolution and ReLU layers it down sampled. At the end there is a class prediction layer which tries to predict the class of the object. To obtain the relevant masks the which are equal to the size of the image, the generated masks are up sampled and padding is applied to the same, generating the masks that we desire around the identified diminished masks.

Now we have covered both the phases of the architecture now let's see how both of these architectures are put together:



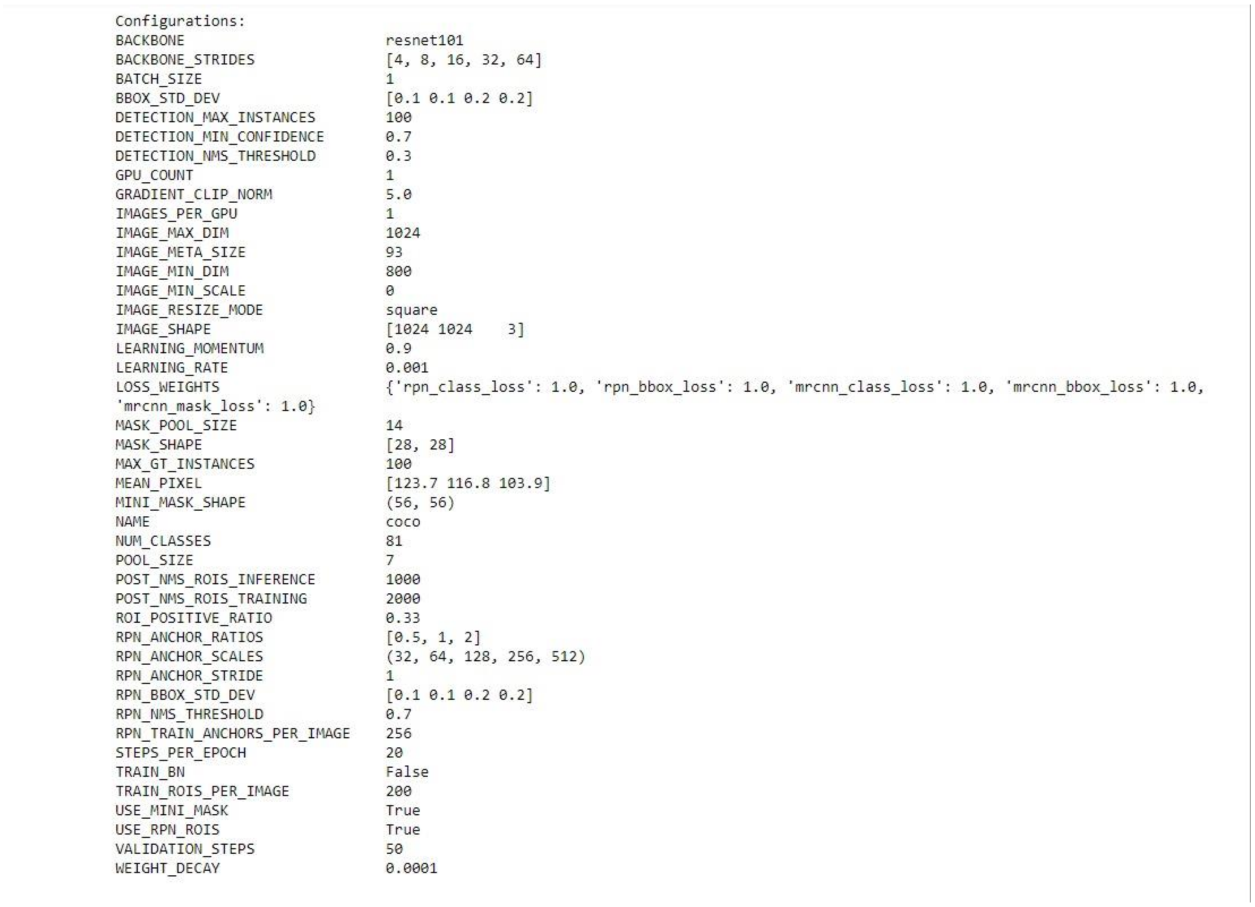
Dataset:

For generating the model, we used the COCO dataset provided my Microsoft. Although, there were latest 2017 dataset's available we chose to work on the images of 2014.

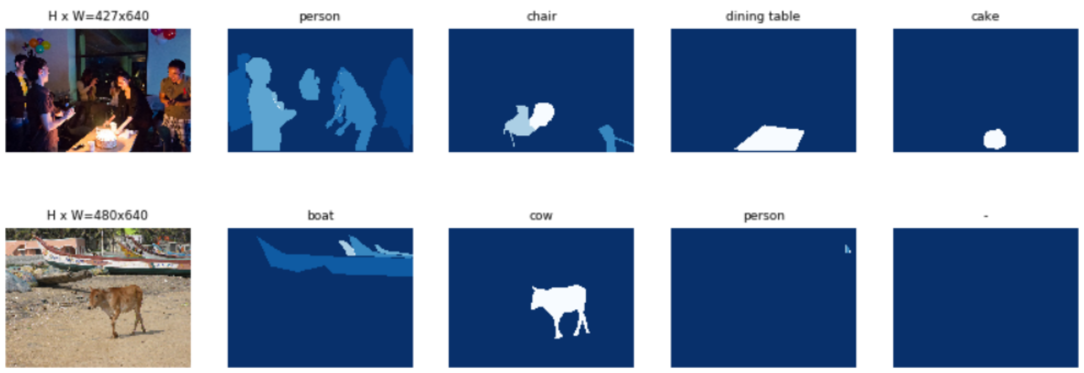
According to the website the best dataset to be used based on the categories we wanted which is detection, captioning and keypoints was the 2014 Train, Validation and Test along with annotations for train/val.

To assist the loading, parsing and visualizing of the images we used the COCOAPI available for python. This can also be found on the website.

There are 81 classes for this dataset. The annotation file saves the data in a json format. It basically consist of how the dataset is saved.



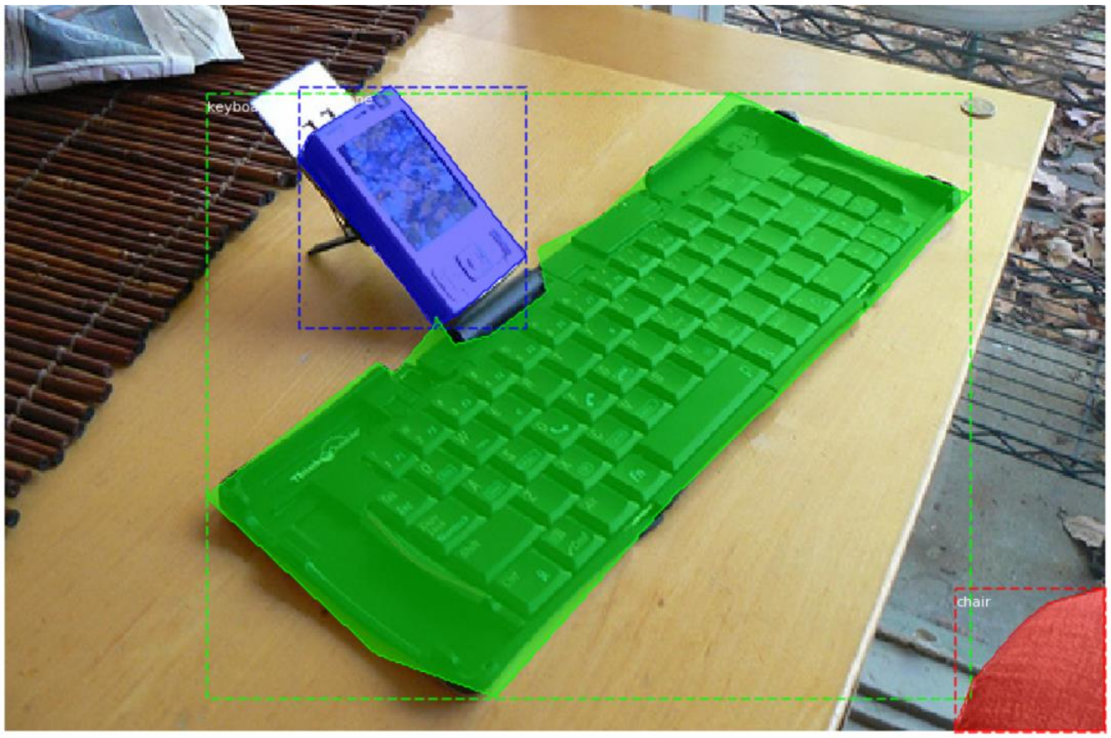
```
Configurations:
BACKBONE                       resnet101
BACKBONE_STRIDES               [4, 8, 16, 32, 64]
BATCH_SIZE                     1
BBOX_STD_DEV                   [0.1 0.1 0.2 0.2]
DETECTION_MAX_INSTANCES        100
DETECTION_MIN_CONFIDENCE       0.7
DETECTION_NMS_THRESHOLD        0.3
GPU_COUNT                      1
GRADIENT_CLIP_NORM             5.0
IMAGES_PER_GPU                 1
IMAGE_MAX_DIM                  1024
IMAGE_META_SIZE                93
IMAGE_MIN_DIM                  800
IMAGE_MIN_SCALE                0
IMAGE_RESIZE_MODE              square
IMAGE_SHAPE                    [1024 1024    3]
LEARNING_MOMENTUM              0.9
LEARNING_RATE                  0.001
LOSS_WEIGHTS                   {'rpn_class_loss': 1.0, 'rpn_bbox_loss': 1.0, 'mrcnn_class_loss': 1.0, 'mrcnn_bbox_loss': 1.0,
'mrcnn_mask_loss': 1.0}
MASK_POOL_SIZE                 14
MASK_SHAPE                     [28, 28]
MAX_GT_INSTANCES               100
MEAN_PIXEL                     [123.7 116.8 103.9]
MINI_MASK_SHAPE                (56, 56)
NAME                           coco
NUM_CLASSES                    81
POOL_SIZE                      7
POST_NMS_ROIS_INFERENCE        1000
POST_NMS_ROIS_TRAINING         2000
ROI_POSITIVE_RATIO             0.33
RPN_ANCHOR_RATIOS              [0.5, 1, 2]
RPN_ANCHOR_SCALES              (32, 64, 128, 256, 512)
RPN_ANCHOR_STRIDE              1
RPN_BBOX_STD_DEV               [0.1 0.1 0.2 0.2]
RPN_NMS_THRESHOLD              0.7
RPN_TRAIN_ANCHORS_PER_IMAGE    256
STEPS_PER_EPOCH                20
TRAIN_BN                       False
TRAIN_ROIS_PER_IMAGE           200
USE_MINI_MASK                  True
USE_RPN_ROIS                   True
VALIDATION_STEPS               50
WEIGHT_DECAY                   0.0001
```



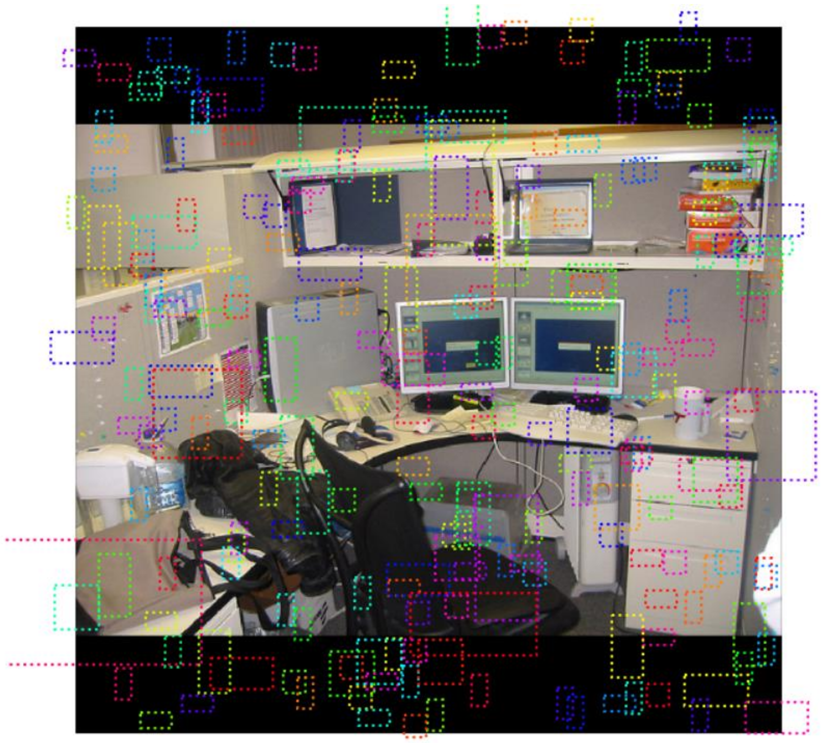
Loading the dataset:

We use the coco.py class to load the dataset. It extracts the annotations and calls the functions to add classes and add image. The load mask generates bitmap masks for every object in the image by drawing a rectangle. The load image function loads the images and the bounding box function provides the bounding features for the image.

Group 5: Dhruv Patel, Fiona Lobo



Anchors:



Group 5: Dhruv Patel, Fiona Lobo

Defining the model:

Mask RCNN is a two stage framework that initially scans the image and generates areas that are likely to contain an object. The second part is classifying the detected objects and applying boxes and masks to it.

The backbone of the model consists of a RESNET101 (even RESNET50 can be used). It serves as a feature extractor. The starting layers detect low level features like corners and edges of the image. As the layers increase it begins to detect objects like a table, person or cell phone called higher level features. The image is converted to feature map of shape 32*32*2048. This becomes the input for the next phase.

Our Approach:

1. Read the Mask RCNN Research paper from the Facebook AI Research
2. Read technical blogs and watched videos on Mask RCNN for better understanding of the whole concept, architecture, model, scope, future applications and much more
3. Found a renown GitHub repository which implemented the Mask RCNN model successfully
4. Executed the demo file of the repository which constituted of the miniature version of the model and gave the required output which was enough to understand how the workflow of the model and the desired output
5. Downloaded the COCO dataset of the year 2014
6. Ran the inspect_data.ipynb file which is used to preprocess the data and produce bounding boxes around the identified objects
7. Changed the hyperparameters in model.py file

Training:

The Mask-RCNN is a large model. Since our base model is RESNET101 and FPN it would take us a longer time. We tried running the model on a smaller dataset i.e. is a subset of coco dataset. However the dataset still being large the training time to longer than a day on AWS p2x large.

For this model we used the model.py provided from the reference to model our data. It consists of batch normalization.

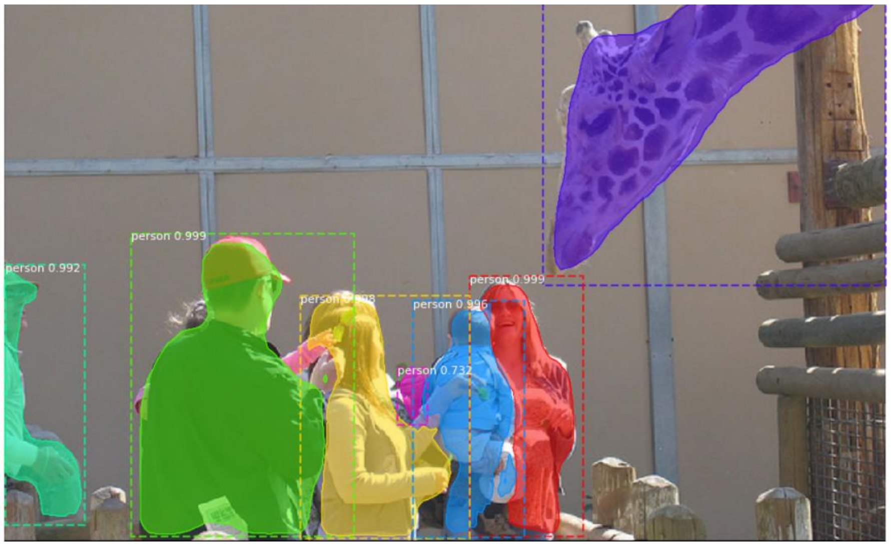
Initially, to train the model we began my training the data on the train data of 2014. For this purpose we began running 100 epochs

We used pre trained weights based on coco_model_weights.h5

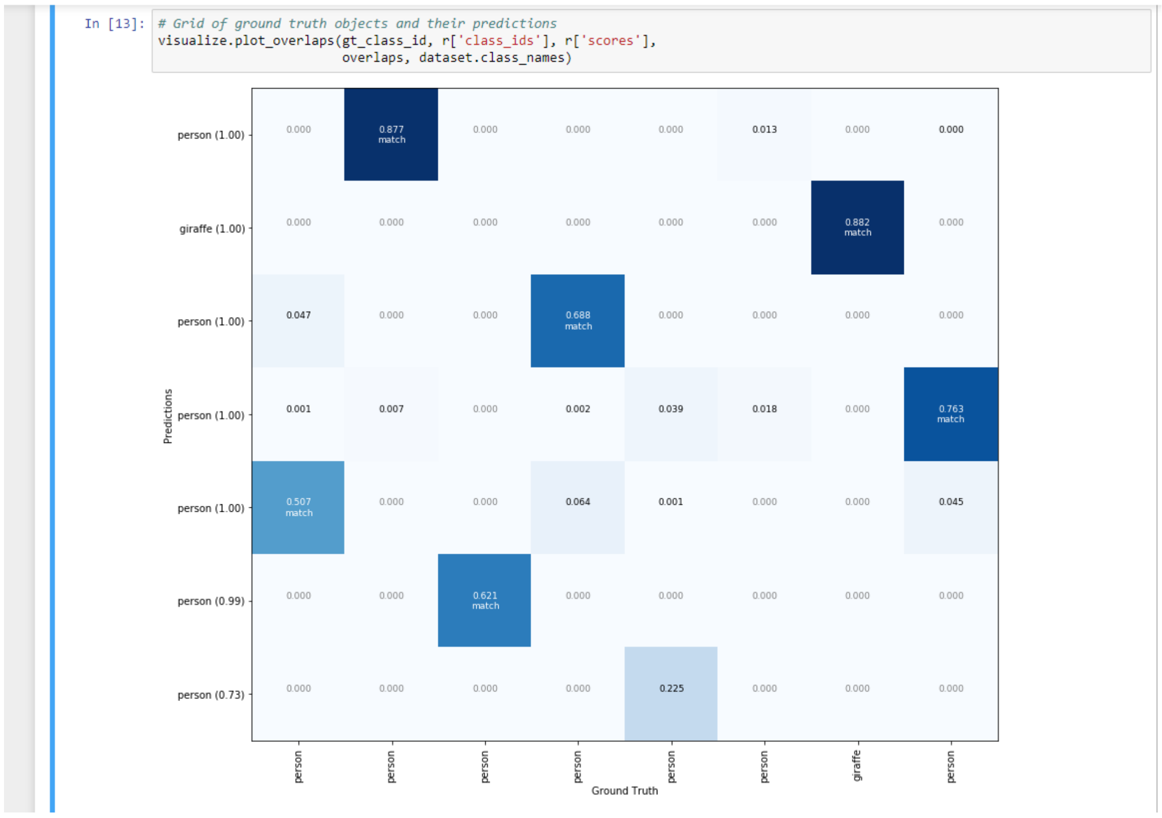
Testing

Group 5: Dhruv Patel, Fiona Lobo

We tested the model and got the following outputs.



Plotting the accuracy of the objects detected:



```
In [13]: # Grid of ground truth objects and their predictions
         visualize.plot_overlaps(gt_class_id, r['class_ids'], r['scores'],
                                 overlaps, dataset.class_names)
```

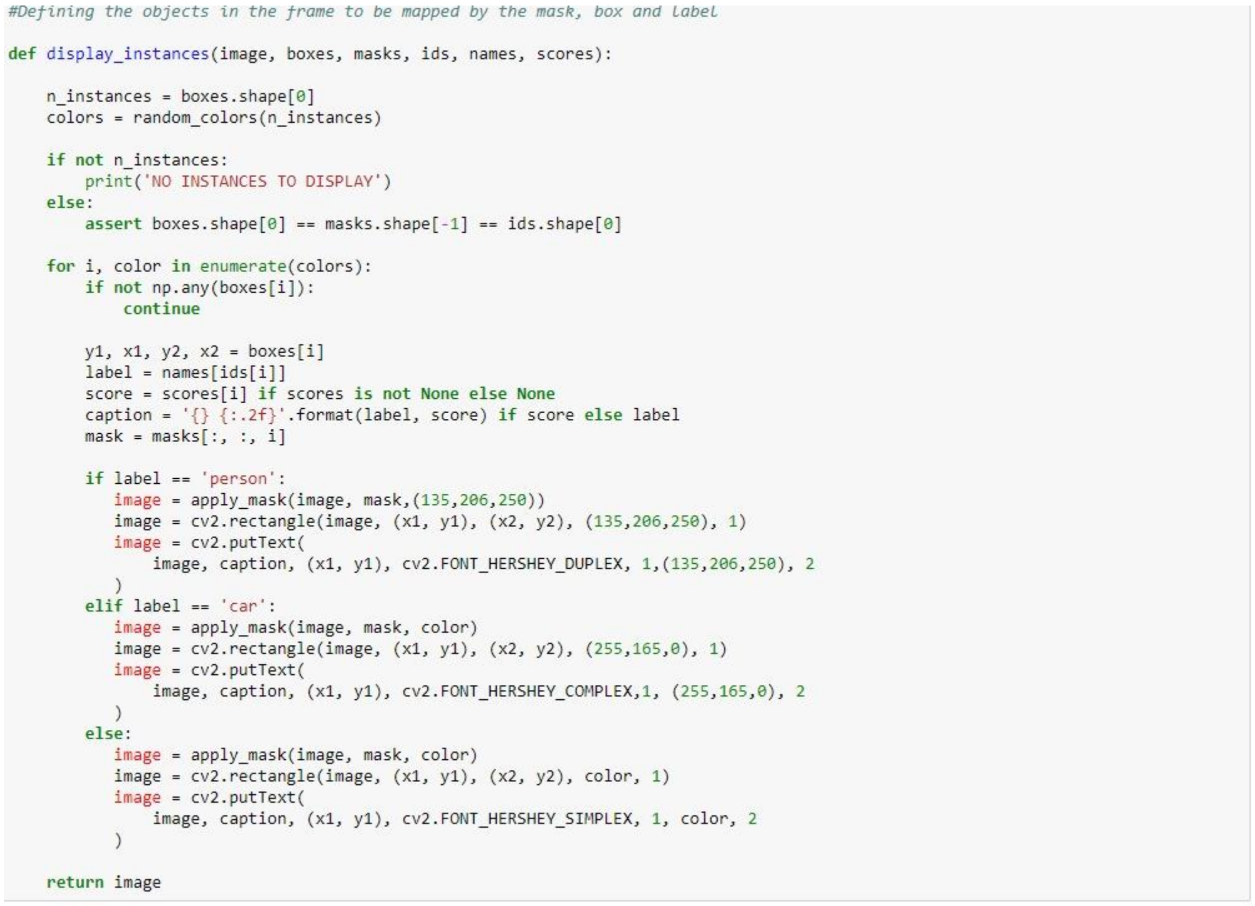
Predictions
Ground Truth
person (1.00): 0.000, 0.877 match, 0.000, 0.000, 0.000, 0.013, 0.000, 0.000
giraffe (1.00): 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.882 match, 0.000
person (1.00): 0.047, 0.000, 0.000, 0.688 match, 0.000, 0.000, 0.000, 0.000
person (1.00): 0.001, 0.007, 0.000, 0.002, 0.039, 0.018, 0.000, 0.763 match
person (1.00): 0.507 match, 0.000, 0.000, 0.064, 0.001, 0.000, 0.000, 0.045
person (0.99): 0.000, 0.000, 0.621 match, 0.000, 0.000, 0.000, 0.000, 0.000
person (0.73): 0.000, 0.000, 0.000, 0.000, 0.225, 0.000, 0.000, 0.000
person, person, person, person, person, person, giraffe, person

Realtime detection using the same model for detection and instance segmentation.

For this we used opencv in python. We pass the model parameters to detect the objects in the frame.

Group 5: Dhruv Patel, Fiona Lobo

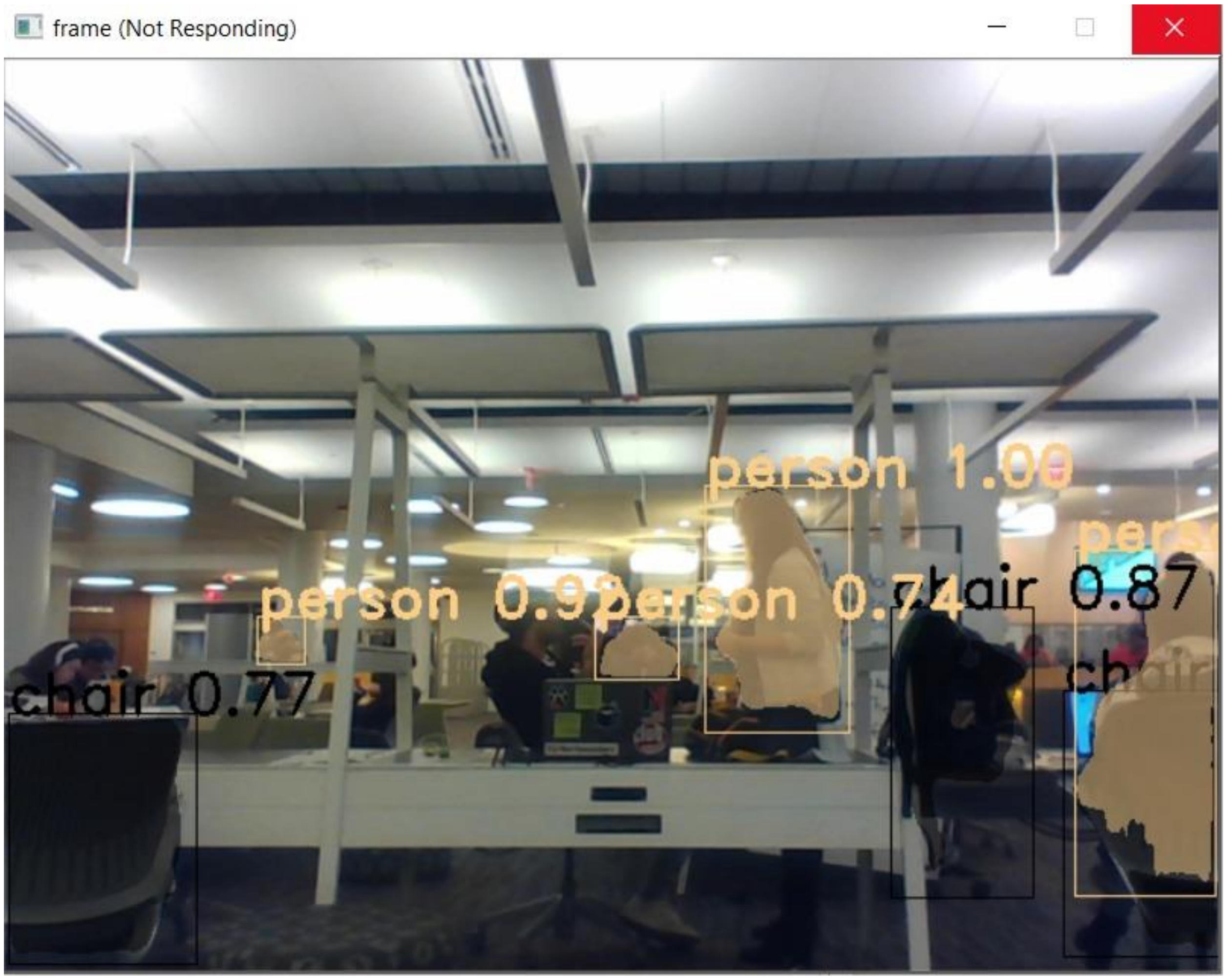
We define a separate display_instance function and random_colors function.



```python
#Defining the objects in the frame to be mapped by the mask, box and label

def display_instances(image, boxes, masks, ids, names, scores):

    n_instances = boxes.shape[0]
    colors = random_colors(n_instances)

    if not n_instances:
        print('NO INSTANCES TO DISPLAY')
    else:
        assert boxes.shape[0] == masks.shape[-1] == ids.shape[0]

    for i, color in enumerate(colors):
        if not np.any(boxes[i]):
            continue

        y1, x1, y2, x2 = boxes[i]
        label = names[ids[i]]
        score = scores[i] if scores is not None else None
        caption = '{} {:.2f}'.format(label, score) if score else label
        mask = masks[:, :, i]

        if label == 'person':
           image = apply_mask(image, mask,(135,206,250))
           image = cv2.rectangle(image, (x1, y1), (x2, y2), (135,206,250), 1)
           image = cv2.putText(
               image, caption, (x1, y1), cv2.FONT_HERSHEY_DUPLEX, 1,(135,206,250), 2
           )
        elif label == 'car':
           image = apply_mask(image, mask, color)
           image = cv2.rectangle(image, (x1, y1), (x2, y2), (255,165,0), 1)
           image = cv2.putText(
               image, caption, (x1, y1), cv2.FONT_HERSHEY_COMPLEX,1, (255,165,0), 2
           )
        else:
           image = apply_mask(image, mask, color)
           image = cv2.rectangle(image, (x1, y1), (x2, y2), color, 1)
           image = cv2.putText(
               image, caption, (x1, y1), cv2.FONT_HERSHEY_SIMPLEX, 1, color, 2
           )

    return image
```

The final output that we got by our model is displayed below:

Group 5: Dhruv Patel, Fiona Lobo



Challenges:

The initial challenge was downloading the dataset. The dataset being huge and consisting of around 30 gb processing it was not an easy task.

We first began doing it on our personal machines as unzipping the datasets would be easier.

Loading the same dataset on aws seemed more time and resource consuming. We kept getting a memory allocation error.

For this we ran the pretrained model on our personal machines and also trained a new model with a subset of the data which took more than a day but without a good output. The original trained model ran on an 8 GPU configured machine for 1-2 days.

To run the model for realtime detection we used the pretrained weights and defined the class to read it frame by frame.

On a CPU machine the frame rate of the real time detection was much slower than that of a single GPUmachine.

We tried to ran the same on AWS by configuring the webcam but were unsuccessful as the webcam would time out and the configuration would fail.

Future Work:

1. Security:
   Instance segmentation can be implemented in video surveillance cameras. The model can be trained to detect explosives and other harmful objects such as knives and guns.
2. Healthcare:
   The model can be trained on images of infected cell/tissue samples and be implemented to detect infected parts of the human body.

Conclusion:

We successfully implemented the real time application of Mask RCNNN and have noted the results which include aspects on the accuracy of prediction, alignment of the mask around an object which is detected.

References:

1. Research Paper link: https://arxiv.org/pdf/1703.06870.pdf
2. Mask RCNN Implementation:https://github.com/matterport/Mask_RCNN
3. COCO API: https://github.com/cocodataset/cocoapi
4. COCO Dataset: http://cocodataset.org/#download
5. Faster RCNN : https://arxiv.org/pdf/1506.01497.pdf
6. FPN: https://arxiv.org/pdf/1612.03144.pdf