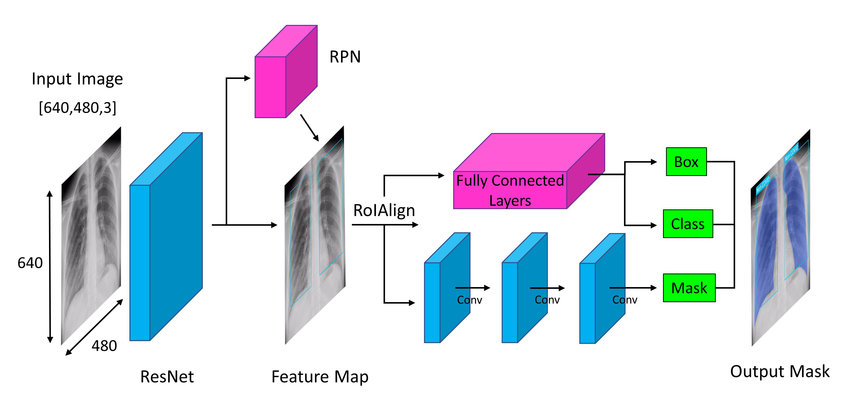
**Instance Segmentation using Mask RCNN**

**Introduction:**

My teammate and I have Collected images of Cars and Sign Boards in the university across one-stop parking and around Bergami Hall and in the Maxcy hall Parking area. Our main aim is to detect these 2 classes using Instance Segmentation. Instance segmentation is a computer vision technique that identifies individual objects within an image and assigns each object uniquely. Label and pixel-level mask or polygon that precisely delineates its boundaries.

In this project, we have implemented instance segmentation using Mask RCNN architecture. Using this architecture, we have generated segmentation masks for each class, using a series of convolution layers. This model first loads the pre-trained masks generated by the RCNN model that was trained on the custom dataset we have provided.

**MASK RCNN ARCHITECTURE:**

Three key parts make up the Mask R-CNN architecture:

* backbone network
* Region Proposal Network (RPN)
* Mask Head.

A convolutional neural network that has already been trained and is used as the backbone network collects features from an input picture. Any common network, including VGG, ResNet, or Inception, can serve as this one. By sliding a tiny network over the feature map produced by the backbone network, the RPN creates region suggestions. In order to produce object detections and object masks, these suggestions are utilized. The Mask Head creates object masks using the feature map produced by the backbone network and the suggested regions from the RPN. The Mask Head is a compact network that receives a binary mask for each proposed region as input and outputs it.

**DATA LOADING AND MASK GENERATION**:

We need to establish a loop that iterates over each image in the directory in our custom dataset. The code opens each file inside the loop and uses the “json-load function” to load the JSON data. JSON data is used to obtain the image's width and height. The JSON data is used to extract the coordinates of the polygon that encircles the item of interest in the image. The polygon's coordinates are transformed into a collection of tuples. The JSON data is used to extract the label of the target item. The “new” function generates a new, empty picture with the same size as the old image.

The image Draw function creates a 'draw' object. The 'draw.polygon()' function is used to draw the polygon on the empty picture. The 'convert ()' function transforms the blank picture into grayscale. The evaluation function converts the grayscale picture into a binary mask by setting all pixels higher than 0 to 255 and all other pixels.

Next, we convert the mask is applied to the original picture to create the masked image. The filename is changed to include the ".JPG" suffix, and the label for the image is also added to a dictionary named "target labels." The key's changed filename is used to add the masked picture to a dictionary called "masked images." Until all of the files in the folder have been processed, the cycle continues.

With the masked images, we can generate a binary image to find the coordinates of the image and then construct the bounding box rectangle. Now, we need to get the image name along with the bounding boxes coordinates into a list and then display those masked images. We need to plot those target labels vs masked images with respective to the boundary values we have taken.

**Sample Output:**

Graphical user interface, application

Description automatically generated

**Data Loading:**Dataset class loads the data to train for the model. We need to make a list of initializes with image name and it’s masked images with labels, bounding boxes along with the transforms where included in the training. Now, get item method retrieves the labels along with the image’s bounding boxes corresponding to the name of the image in a dictionary format after performing the transforms on each image.

Creating an instance for the custom Dataset with parameters of the image names, masks, labels, bounding boxes and transformers. We need to spilt into a three parts and create a list of indices with training, validation and testing. Then create SubsetRandomSampler objects are created for sampling random samples from the dataset. We need to combine the different separate samples into a batch and then transform a list of image with target tuples into a list format for giving the model input.

**Model Training:**

The get\_model\_instance\_segmentation function, which uses the Mask R-CNN architecture to construct an instance segmentation model, is defined in the code. A computer vision job called instance segmentation entails locating and categorizing each instance of an object present in a picture. The num\_classes argument, which the function requires, specifies how many different object classes the model will be taught to recognize. For each object that is identified, the model will provide a collection of bounding boxes and masks.

A pre-trained Mask R-CNN model that has been trained on the COCO (Common Objects in Context) dataset is then loaded by the function. This model may be used as a starting point for training a new instance segmentation model because it has already trained to recognize and categorize a large number of object types.

The function then swaps out the model's pre-trained head for a new one that is tailored to the amount of object classes we're looking to identify. The head of the model uses the characteristics that were taken from the picture to forecast the class labels and bounding bounds for each object that is identified.

A new mask predictor that is tailored to the number of object classes we wish to identify is also substituted for the original one by the function. The pixel-wise segmentation masks for each recognized item are created by the mask predictor.

The function then transfers the model to the CPU or GPU that was defined before and returns the finished model. The number of object classes are specified as n\_classes, and the get\_model\_instance\_segmentation () method is used to initialize an instance segmentation model. The code then establishes a model optimizer. Specifically, it updates the model parameters during training using the stochastic gradient descent (SGD) algorithm. All model parameters that call for gradient computation are listed in the params variable and will be updated during backpropagation.   
  
**Results with Original Hyper Parameters value:**

The learning rate (lr), which controls how much the optimizer modifies the parameters in response to calculated gradients, is set to 0.00001. The optimizer moves in the same direction as earlier stages and speeds up convergence when the momentum parameter (momentum) is set to 0.9. By penalizing big weights, the weight decay (weight\_decay) parameter, a type of L2 regularization that aids in preventing overfitting, is set to 0.0005.

Overall, this code configures the hyperparameters for the optimizer and the instance segmentation model for training.  
  
Here, the result is calculated for 50 epochs, with the above learning rate, weight decay and momentum values.

Graphical user interface, application

Description automatically generated  
  
**Experiments 02**

I have updated the hyper parameters values with the above values and following are the results and graph plots. There is no much difference but I can see that at the 10th epoch, there is a raise and also similar to epoch at 20th.   
Graphical user interface, text, application, email

Description automatically generated  
  
Graphical user interface, text, application, Word, email

Description automatically generated

**Experiment-03:**After decreasing the learning rate, the model has led to overfitting, which means that, those values don’t work accordingly.

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, Word

Description automatically generated  
 **Results:**After iterating through 50 epochs, due to low data and complexity with the dataset we have taken, we are unable to display the image. The objects we have chosen is a car image and the sign boards which are typically in a polygon shape. We can even observe the result in the form of tensors which are displayed down below.   
 **OUTPUT IMAGES:**

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated