Volume: 8 Issue: XII Month of publication: December 2020
DOI: https://doi.org/10.22214/ijraset.2020.32594

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 8 Issue XII Dec 2020- Available at www.ijraset.com

# Extraction of Real-World Objects in the Environment through Computer Vision

Prof. Kavita Jain[1], Varun Singh[2], Rohan Talele[3], Kartik Kashyap[4]
[1, 2, 3, 4]Department of Computer Engineering, Xavier Institute of Engineering, Mumbai University, Mumbai, India

***Abstract: For the past decade, there has been a surge of interest in self-driving vehicles. This is because of leaps in the field of deep learning especially Artificial Neural Networks which are trained to perform tasks that regularly require human intervention. Genetic algorithms (GA) apply models to identify patterns and features in the environment making them desirous in cases of uncertainty. In this project we have trained an evolving Neural network by using Genetic Algorithms in Unity Game Engine using the ML-agents provided by them. By repeated simulation of the agent in the environment, using this we have created an agent which learns unique features from the environment based on its input and generates a forecast permitting the vehicle to drive without requiring external commands from the player.***

***Keywords: Computer Vision (CV), Boundary-Aware Salient Object Detection (BASNet), Boundary Detection, React Native.***

## I. INTRODUCTION

A Computer Vision and Machine Learning prototype application that allows cutting elements from your surroundings and pasting them in an image editing software. For this we propose a technique to transfer an image taken from the real-world, removing its background detail and just keeping the image. Then transferring that image of the object to a computer screen by positioning the mobile camera in the direction of the computer screen to paste it.

### *A. Aim*

A Computer Vision and Machine Learning prototype application that allows cutting elements from your surroundings and pasting them in an image editing software.

### *B. Objectives*

*1)* The app is being created using React Native and the expo framework [8]. We plan on keeping the GUI relatively simple since the main focus is on the transfer feature of the app. We hope to create a technique to transfer an image taken from the real-world, removing its background detail and just keeping the image. We achieve this using BASNet (Boundary-Aware Salient Object Detection) [2] [3].

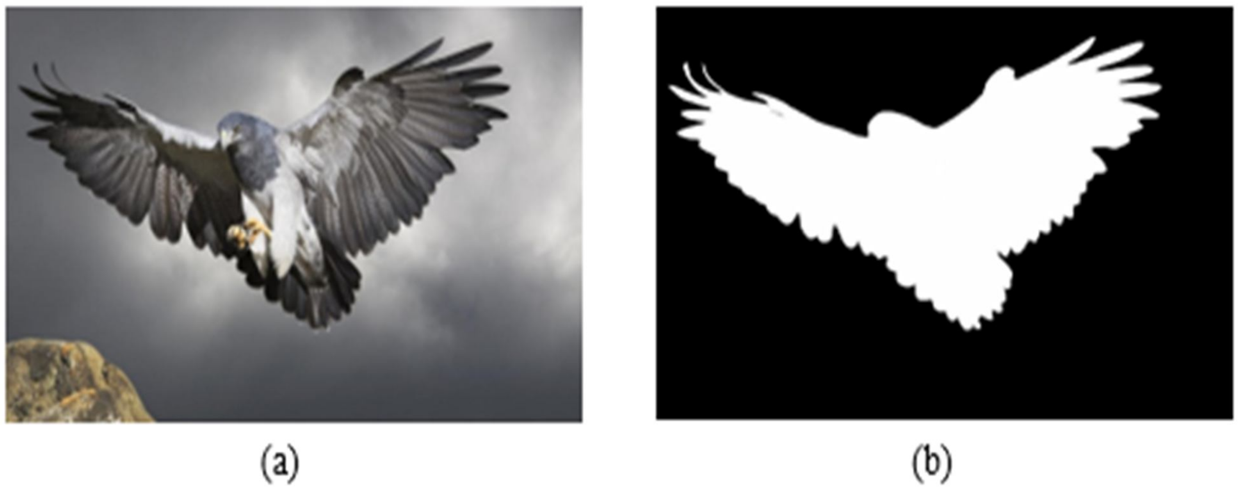


Figure 1. (a) and (b) show input and output respectively

2) Then for the second part transferring that image of the object to a computer screen by holding the mobile camera and positioning it in the direction of the computer screen to paste it.

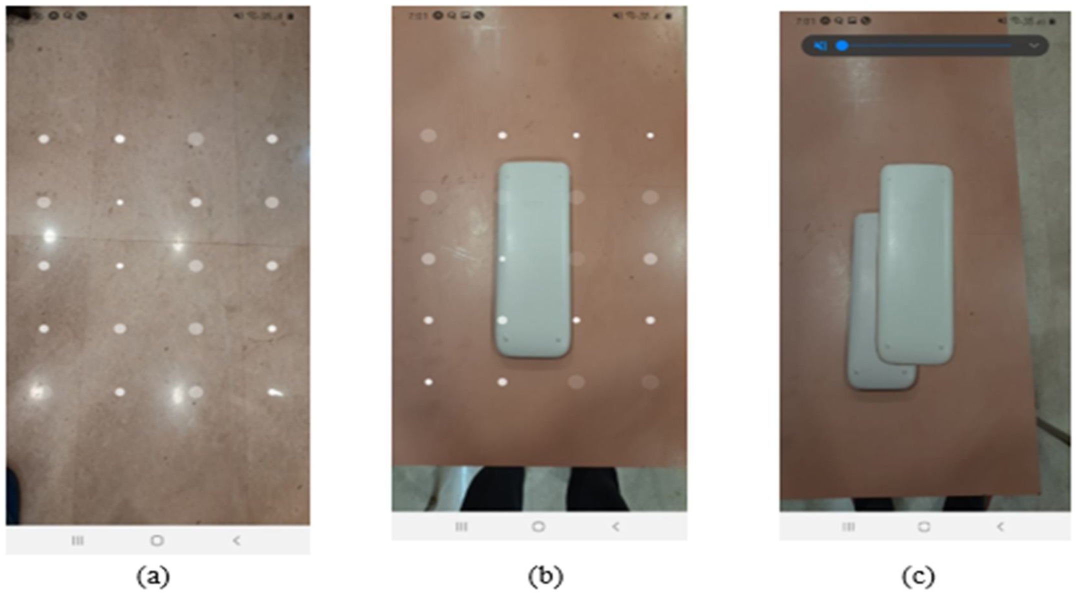


Figure 2. (a) shows the GUI with the animated circles indicating extraction is happening. (b) show the example of a calculator being extracted. (c) shows the extracted calculator

This project is designed to shorten the process of transferring images from our phones to our computers through Emailing, Bluetooth, WhatsApp, etc by bypassing all of this. Instead, we would capture the image from our phones, extract only the subject of the image and then transfer that extracted image directly to our computer screen. For now, we are only considering Adobe Photoshop, an image and graphics editing software as the screen on which the extracted image would be displayed. From there the image can then be further edited in photoshop, copied or saved on the computer.

This type of application would be helpful for editing images by photography professionals or just for transferring images quickly since it would cut down the work required to transfer the image manually through email or some other service and then in photoshop remove the background and extract the subject of the image manually.

There are some photoshop plugins that use AI to extract the subject of the image. Some these are Mask Pro, EZ Green Screen 5, Fluid Mask and many more. However, these all work within photoshop on the input image with background still in it. Our system plans to directly display the extracted image by transferring it from our phone.

Google Lens is an image recognition technology developed by Google, designed to bring up relevant information related to objects it identifies using visual analysis based on a neural network. Google lens was used as motivational inspiration for this project since one of its features is to scan and translate text in the environment. Our project plans are also to scan objects from the environment, however instead of scanning text we plan on scanning objects as images and then transferring the main subject of the image into photoshop. So, our application can be thought of as a combination of both the above-mentioned features. After extensive search we haven't found a system existing that exactly does what we plan on doing, however there might be one being developed.

## II. RELATED WORK

Saliency detection aims at modeling human visual attention mechanism to detect distinct regions or objects, on which people likely focus their eyes in visual scenes. Contextual information plays an essential role in this visual task. As one of the earliest pioneering computational saliency models, calculate the feature difference between each pixel and its surrounding regions as the contrast to infer saliency. Numerous methods have been subsequently developed that utilize local or global contexts as the reference to evaluate the contrast of each image location (i.e., local or global contrast). These models aggregate visual information at all the locations of the referred context region into a contextual feature to infer contrast. First, we searched on boundary detection and saliency detection of images. We had our general idea of extracting the image using boundary edge detection of the image, however we had to identify the algorithm we would use since there are many of them.

## *A. PiCANet: Learning Pixel-wise Contextual Attention for Saliency Detection*

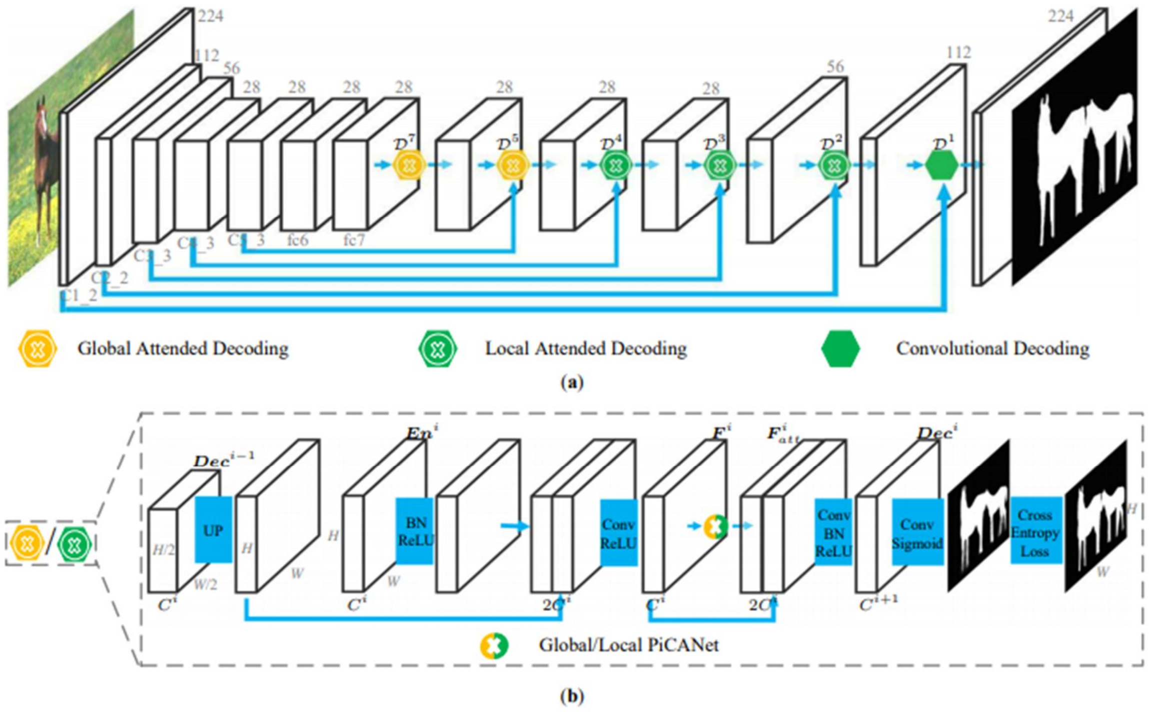


Figure 3. (a) Architecture of PiCANet. (b) Illustration of an attended decoding module

The first algorithm we found was PiCANetR (Learning pixel-wise contextual attention for saliency detection) [4]. It works by giving importance to contexts for saliency detection task. However, given a context region, not all contextual information is helpful for the final task. They propose a novel pixel-wise contextual attention network, i.e., the PiCANet, to learn to selectively attend to informative context locations for each pixel. Specifically, for each pixel, it can generate an attention map in which each attention weight corresponds to the contextual relevance at each context location. An attended contextual feature can then be constructed by selectively aggregating the contextual information.

Performance Evaluation metrics are F-measure score which comprises both precision and recall, the second metric being wieghted F-measure score in case of preference to either precision or recall and the final metric being Mean Absolute Error (MAE) which computes the absolute per-pixel difference between predicted saliency maps and the actual maps [4].

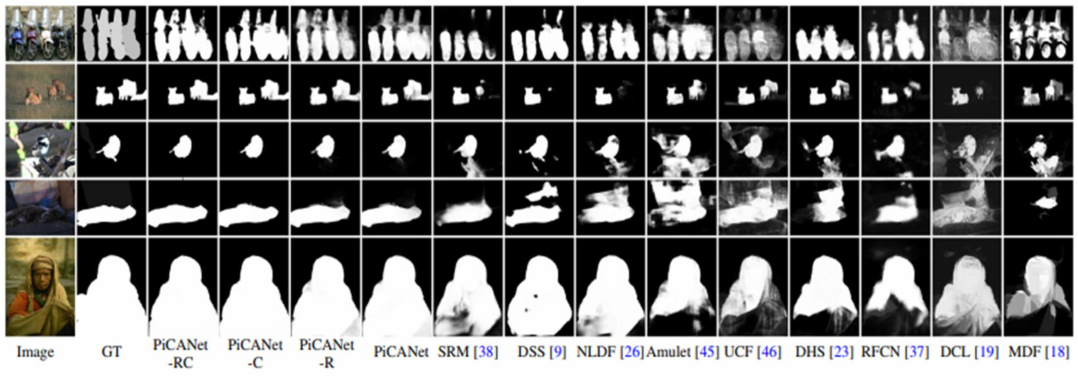


Figure 4. Qualitative comparison of PiCANet. (GT: ground truth)

PiCANet has high accuracy and overall performance on standard datasets, however we found out about BASNet (Boundary-Aware Salient Object Detection) which works much better overall and has a much better performance than PiCANet.

### *B. BASNet: Boundary-Aware Salient Object Detection*

Most of the deep learning methods focus on the region prediction when it comes to saliency prediction but now, they have created a new loss function where the BOUNDARY of the object is also considered [3]. This technique is known as BASNet (Boundary-Aware Salient Object Detection). There are other ways and libraries to achieve this as well namely the YOLO (You only look once) library, however in the above-mentioned source as well as in the paper published on BASNet it is clearly seen that BASNet is superior or close enough to the other methods. Hence, we have decided to go with BASNet [3] [4].

Two parts of the saliency detection network, one is the predict network, which can obtain the coarse saliency region, and the other is the fine network following the prediction network, which is used to further refine the coarse saliency region obtained in the previous step, Get a more accurate saliency map. The network structure of these two networks is roughly the same, both are classic Encode-Decode networks, but the predict network has a deeper structure, while the fine network is shallower.

Intersection-over-Union (IoU), the model can pay attention to the pixel-level, patch-level and map-level of the image. Level of saliency information. In order to obtain more precise significance results.

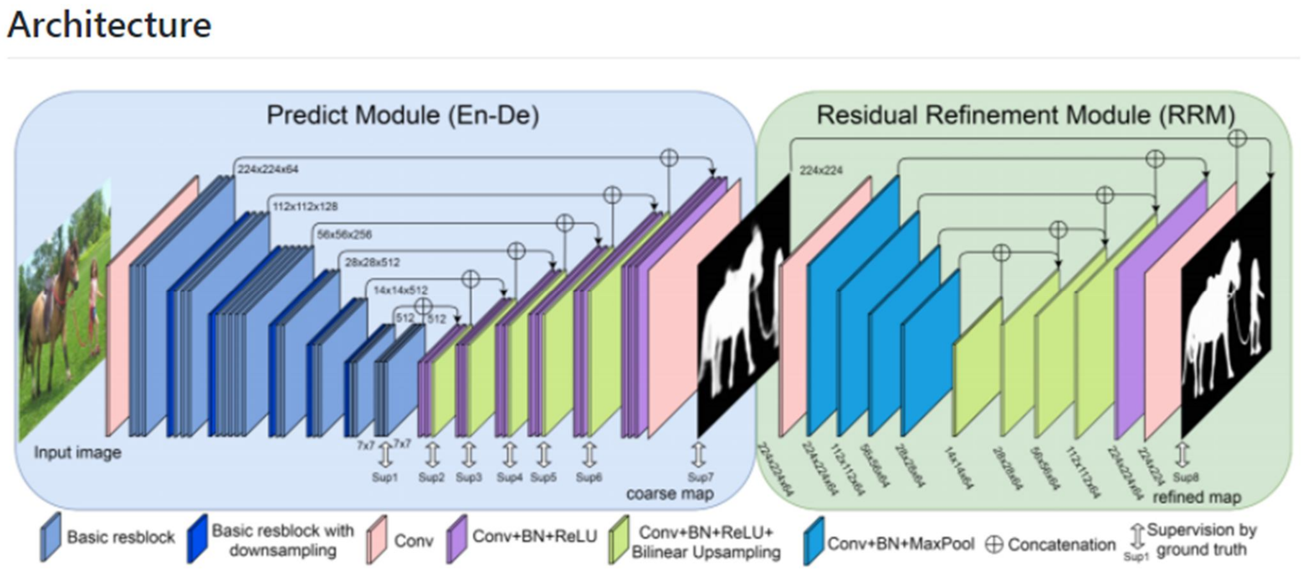


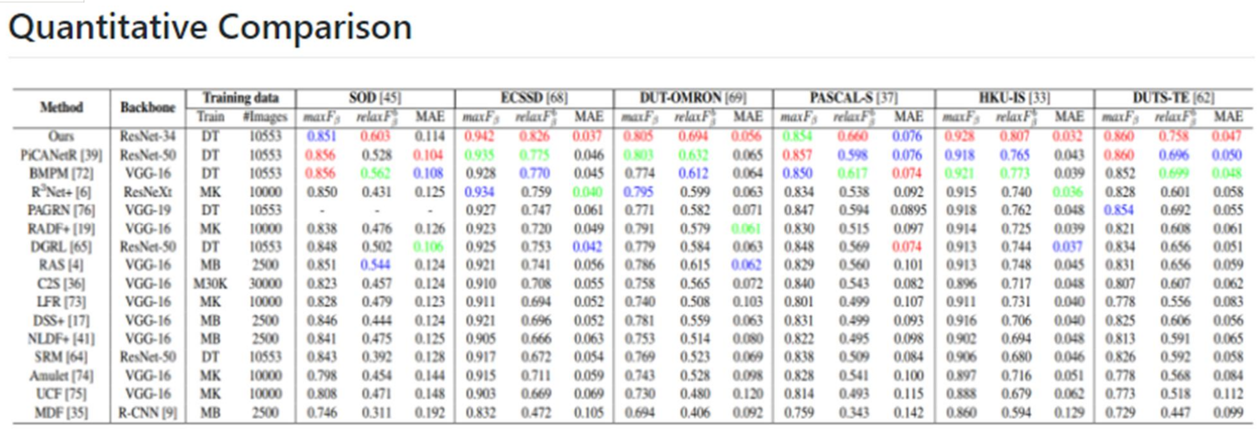
Figure 5. Architecture of BASNet.[3] [4]

The above network structure is the network structure of the entire BASNet algorithm, and it is also the pipeline of the entire algorithm.

### *C. The entire BASNet network Structure is Divided Into two Parts*

One part is the Predict Module. This part of the network inputs an image, and then passes through the encode and decode layers to output the preliminary predicted saliency map. This part of the network is, after all, the classic Encode-Decode network. The previous Encode extracts features of the image, and uses the Pooling method to obtain high-level semantic features with a gradually decreasing resolution. The latter Decode part is responsible for gradually restoring and amplifying high-level semantic information. In this way, a large-resolution feature map is gradually obtained, and a saliency map of the same size as the original image is finally output. Between Encode and Decode, there will be a shortcut to add the feature map of the same resolution, so that the final output feature map can take into account both low-level and high-level features. It is worth mentioning that in the decoding process, there are a total of 6 feature maps with different resolutions, plus the feature map of the last layer of the encode stage. A total of 7 feature maps are used for loss calculation. The multi-loss method is a bit similar to relay loss. On the one hand, it can help the network converge better, and on the other hand, it can make the network pay attention to saliency maps of different scales.

The other part is the Residual Refinement Module. The network structure of this part is actually the same as the previous Predict Module network structure. Use conv, BN, and RELU to construct encode and decode, but compared with the previous Predict Module, the network structure of this part is Simpler, lower network depth. In addition, this part of the loss only uses the output of the last layer as loss, and the output of the middle layer does not.



## Quantitative Comparison

| Method | Backbone | Training data | | SOD [45] | | | ECSSD [68] | | | DUT-OMRON [69] | | | PASCAL-S [37] | | | HKU-IS [33] | | | DUTS-TE [62] | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Train | #Images | $maxF_\beta$ | $relaxF_\beta^b$ | MAE | $maxF_\beta$ | $relaxF_\beta^b$ | MAE | $maxF_\beta$ | $relaxF_\beta^b$ | MAE | $maxF_\beta$ | $relaxF_\beta^b$ | MAE | $maxF_\beta$ | $relaxF_\beta^b$ | MAE | $maxF_\beta$ | $relaxF_\beta^b$ | MAE |
| Ours | ResNet-34 | DT | 10553 | 0.851 | 0.603 | 0.114 | 0.942 | 0.826 | 0.037 | 0.805 | 0.694 | 0.056 | 0.854 | 0.660 | 0.076 | 0.928 | 0.807 | 0.032 | 0.860 | 0.758 | 0.047 |
| PiCANetR [39] | ResNet-50 | DT | 10553 | 0.856 | 0.528 | 0.104 | 0.935 | 0.775 | 0.046 | 0.803 | 0.632 | 0.065 | 0.857 | 0.598 | 0.076 | 0.918 | 0.765 | 0.043 | 0.860 | 0.696 | 0.050 |
| BMPM [72] | VGG-16 | DT | 10553 | 0.856 | 0.562 | 0.108 | 0.928 | 0.770 | 0.045 | 0.774 | 0.612 | 0.064 | 0.850 | 0.617 | 0.074 | 0.921 | 0.773 | 0.039 | 0.852 | 0.699 | 0.048 |
| $R^3$Net+ [6] | ResNeXt | MK | 10000 | 0.850 | 0.431 | 0.125 | 0.934 | 0.759 | 0.040 | 0.795 | 0.599 | 0.063 | 0.834 | 0.538 | 0.092 | 0.915 | 0.740 | 0.036 | 0.828 | 0.601 | 0.058 |
| PAGRN [76] | VGG-19 | DT | 10553 | - | - | - | 0.927 | 0.747 | 0.061 | 0.771 | 0.582 | 0.071 | 0.847 | 0.594 | 0.0895 | 0.918 | 0.762 | 0.048 | 0.854 | 0.692 | 0.055 |
| RADF+ [19] | VGG-16 | MK | 10000 | 0.838 | 0.476 | 0.126 | 0.923 | 0.720 | 0.049 | 0.791 | 0.579 | 0.061 | 0.830 | 0.515 | 0.097 | 0.914 | 0.725 | 0.039 | 0.821 | 0.608 | 0.061 |
| DGRL [65] | ResNet-50 | DT | 10553 | 0.848 | 0.502 | 0.106 | 0.925 | 0.753 | 0.042 | 0.779 | 0.584 | 0.063 | 0.848 | 0.569 | 0.074 | 0.913 | 0.744 | 0.037 | 0.834 | 0.656 | 0.051 |
| RAS [4] | VGG-16 | MB | 2500 | 0.851 | 0.544 | 0.124 | 0.921 | 0.741 | 0.056 | 0.786 | 0.615 | 0.062 | 0.829 | 0.560 | 0.101 | 0.913 | 0.748 | 0.045 | 0.831 | 0.656 | 0.059 |
| C2S [36] | VGG-16 | M30K | 30000 | 0.823 | 0.457 | 0.124 | 0.910 | 0.708 | 0.055 | 0.758 | 0.565 | 0.072 | 0.840 | 0.543 | 0.082 | 0.896 | 0.717 | 0.048 | 0.807 | 0.607 | 0.062 |
| LFR [73] | VGG-16 | MK | 10000 | 0.828 | 0.479 | 0.123 | 0.911 | 0.694 | 0.052 | 0.740 | 0.508 | 0.103 | 0.801 | 0.499 | 0.107 | 0.911 | 0.731 | 0.040 | 0.778 | 0.556 | 0.083 |
| DSS+ [17] | VGG-16 | MB | 2500 | 0.846 | 0.444 | 0.124 | 0.921 | 0.696 | 0.052 | 0.781 | 0.559 | 0.063 | 0.831 | 0.499 | 0.093 | 0.916 | 0.706 | 0.040 | 0.825 | 0.606 | 0.056 |
| NLDF+ [41] | VGG-16 | MB | 2500 | 0.841 | 0.475 | 0.125 | 0.905 | 0.666 | 0.063 | 0.753 | 0.514 | 0.080 | 0.822 | 0.495 | 0.098 | 0.902 | 0.694 | 0.048 | 0.813 | 0.591 | 0.065 |
| SRM [64] | ResNet-50 | DT | 10553 | 0.843 | 0.392 | 0.128 | 0.917 | 0.672 | 0.054 | 0.769 | 0.523 | 0.069 | 0.838 | 0.509 | 0.084 | 0.906 | 0.680 | 0.046 | 0.826 | 0.592 | 0.058 |
| Amulet [74] | VGG-16 | MK | 10000 | 0.798 | 0.454 | 0.144 | 0.915 | 0.711 | 0.059 | 0.743 | 0.528 | 0.098 | 0.828 | 0.541 | 0.100 | 0.897 | 0.716 | 0.051 | 0.778 | 0.568 | 0.084 |
| UCF [75] | VGG-16 | MK | 10000 | 0.808 | 0.471 | 0.148 | 0.903 | 0.669 | 0.069 | 0.730 | 0.480 | 0.120 | 0.814 | 0.493 | 0.115 | 0.888 | 0.679 | 0.062 | 0.773 | 0.518 | 0.112 |
| MDF [35] | R-CNN [9] | MB | 2500 | 0.746 | 0.311 | 0.192 | 0.832 | 0.472 | 0.105 | 0.694 | 0.406 | 0.092 | 0.759 | 0.343 | 0.142 | 0.860 | 0.594 | 0.129 | 0.729 | 0.447 | 0.099 |

Figure 6. Quantitative comparison of BASNet.[3] [4]

The following figure is a comparison of experimental results. It can be found that the performance is still relatively strong. On the GPU, 256*256 images can also reach 25fps. Although the improvement of the network structure only superimposes the Encode-Decode structure, and the improvement and innovation of loss is only a combination of three kinds of loss, the final result is still relatively work, and it also shows that the author's improvement is simple and effective.
BASNet mostly outperforms all the other algorithms and techniques for the most part, with all the different training data sets given to it. A much better showcase of optimum results would be through the comparison of different outputs produced by different algorithms.
To obtain high quality regional segmentation and clear boundaries, we propose to define ℓ (k) as a hybrid loss:

$$\ell\ (k) = \ell\ (k)\ bce + \ell\ (k)\ ssim + \ell\ (k)\ iou$$

where ℓ (k) bce, ℓ (k) ssim and ℓ (k) iou denote BCE loss, SSIM loss and IoU loss respectively. BCE loss is the most widely used loss in binary classification and segmentation. It denotes the predicted probability of being salient object.
SSIM is originally proposed for image quality assessment. It captures the structural information in an image. Hence, we integrated it into our training loss to learn the structural information of the salient object ground truth. IoU is originally proposed for measuring the similarity of two sets and then used as a standard evaluation measure for object detection and segmentation. Recently, it has been used as the training loss

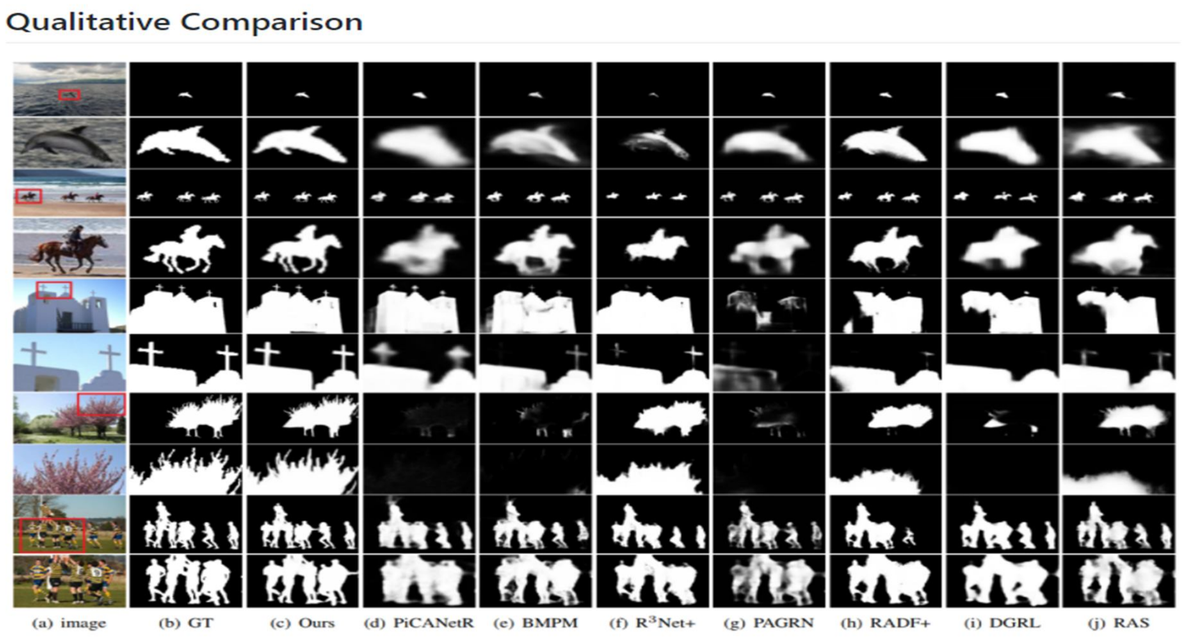


Figure 7. Qualitative comparison of BASNet. (GT: ground truth) [3] [4]

## III. IMPLEMENTATION METHODOLOGY

To start from the app design, we had decided early on to make sure our app is cross platform compatible, therefore we had to choose between different stacks namely Flutter and React Native. We ultimately decided to go with React Native since it provided better support for Machine Learning algorithms and required javascript for coding which we were familiar with.

This Prototype will have 3 independent modules

### A. *The Mobile App*

The Base Application can be developed using React Native (For Multiplatform support) or A Java Based Android Application

### B. *The Local Server*

This will be interface between mobile app and Photoshop. (We will be using Adobe Photoshop Software for Pasting Purpose).
For finding the Position Pointed On screen by the Cameras we will use **Open-CV SIFT** library.
This will Find the (x,y) co-ordinate of the Centroid of an image Pointing at another Image.

### C. *Object Detection/Background Removal Service*

Now we will be Using BASNet. It performs Object Detection and also separated the object image from the Background image.
Now for using BASNet we have to deploy model on local server as an Http service.

All the Images will be Pasted in Adobe Photoshop.

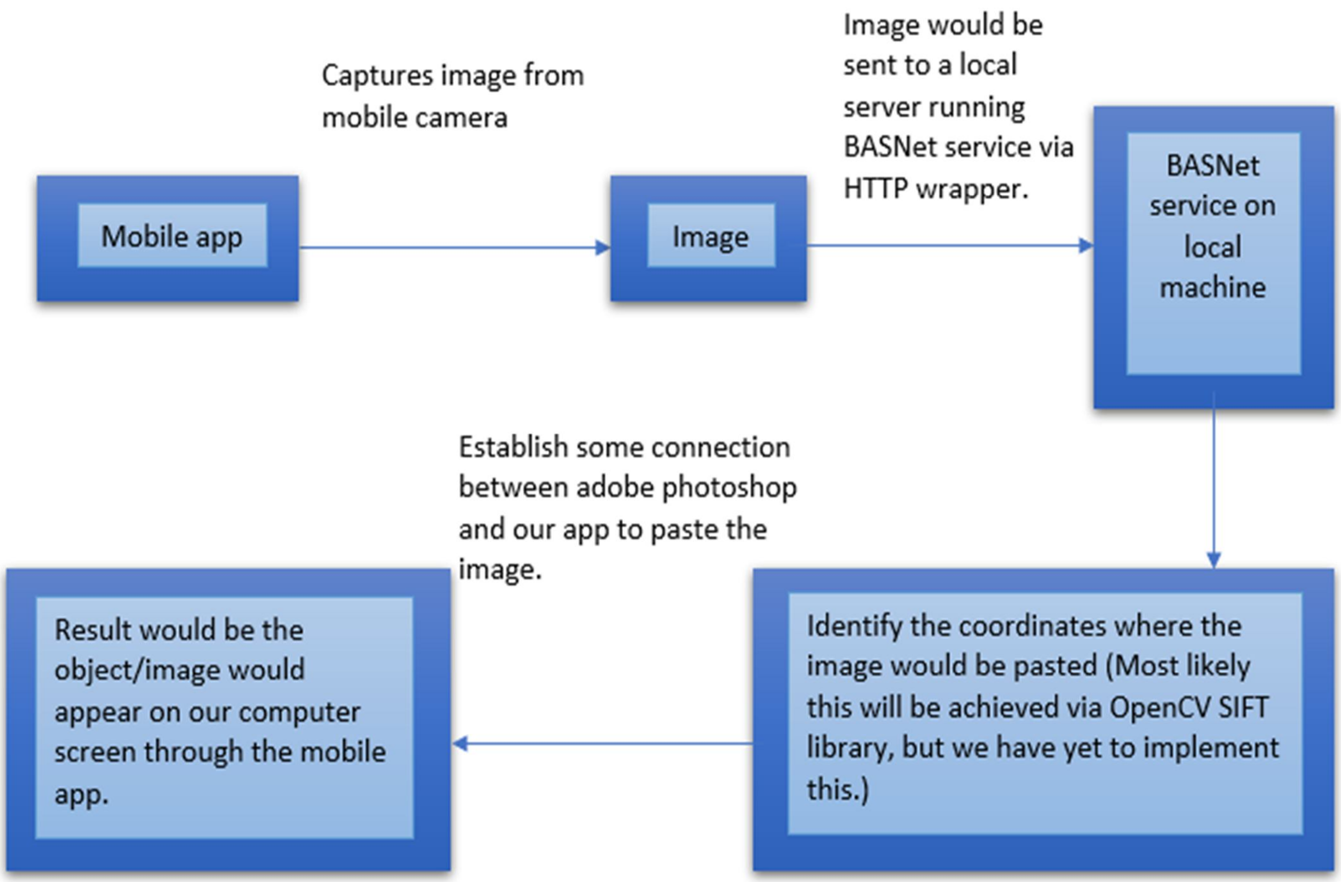

Captures image from mobile camera
Image would be sent to a local server running BASNet service via HTTP wrapper.
Mobile app
Image
BASNet service on local machine
Establish some connection between adobe photoshop and our app to paste the image.
Result would be the object/image would appear on our computer screen through the mobile app.
Identify the coordinates where the image would be pasted (Most likely this will be achieved via OpenCV SIFT library, but we have yet to implement this.)

Figure 8. Block diagram of system

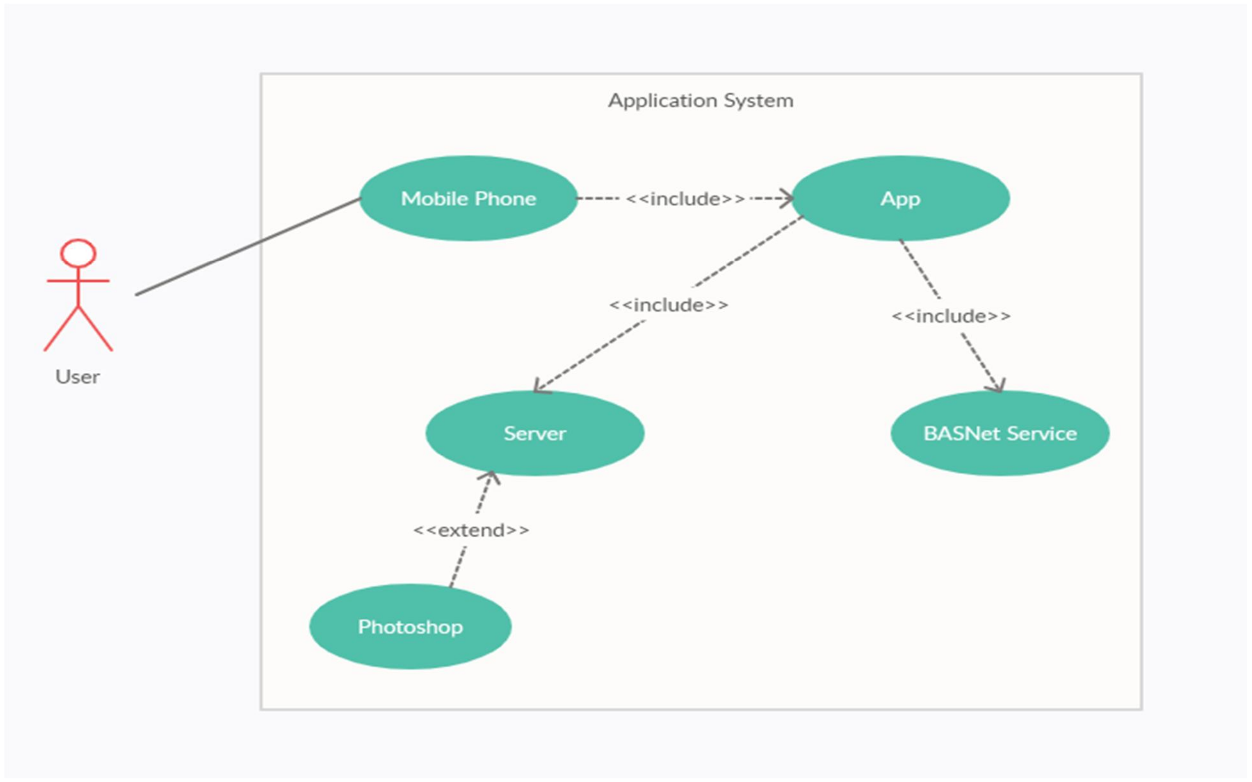


Figure 9. Use case diagram

The GUI of the project was designed through React Native Framework owned by Facebook. We have kept the GUI almost nonexistent for the time being since we are more focused on the extraction and transfer part of our project. To make apps through React Native there are two approaches, one is to use expo framework which is an open-source platform for making universal native apps for Android, iOS, and the web with JavaScript and React.

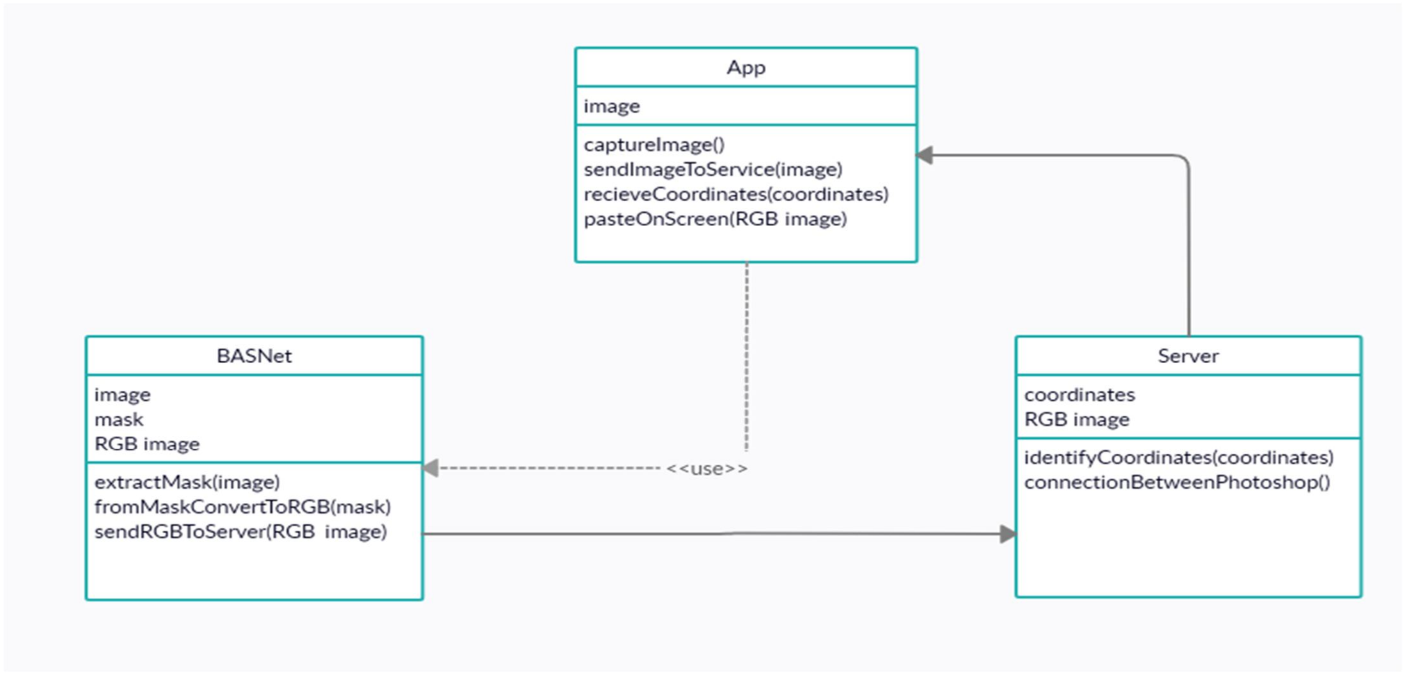

App
image
captureImage()
sendImageToService(image)
recieveCoordinates(coordinates)
pasteOnScreen(RGB image)
BASNet
image
mask
RGB image
extractMask(image)
fromMaskConvertToRGB(mask)
sendRGBToServer(RGB image)
<<use>>
Server
coordinates
RGB image
identifyCoordinates(coordinates)
connectionBetweenPhotoshop()

Figure 10. Class Diagram

*ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429*
*Volume 8 Issue XII Dec 2020- Available at www.ijraset.com*

The other approach is to start from scratch using only the basic primitive components React native provides. We chose to go with the first option i.e. expo, since this was our first time building cross platform app and it would be easier to focus on the Computer Vision part of our project. Identify the object you want to extract, point the camera towards it and then press and hold the mobile screen to capture and extract the object. The extraction process might take around 10 – 20 seconds depending on how much the camera screen is being shaken by the user. Once the extraction is done the object would appear on the mobile phone screen as long as the user has been pressing the mobile screen.

While extracting the image we have kept an indicative UI that informs the user that the extraction is taking place. To execute our application, we would require the user to start our app through the expo app installed on their mobile phones through google play store for android users. For iOS users the inbuilt QR scanner would do since we need to scan the QR code that appears to run our project. Once the code is scanned the project will first build the JavaScript bundle then the app would start. The app on starting will look like this which is just like the default camera of your device.

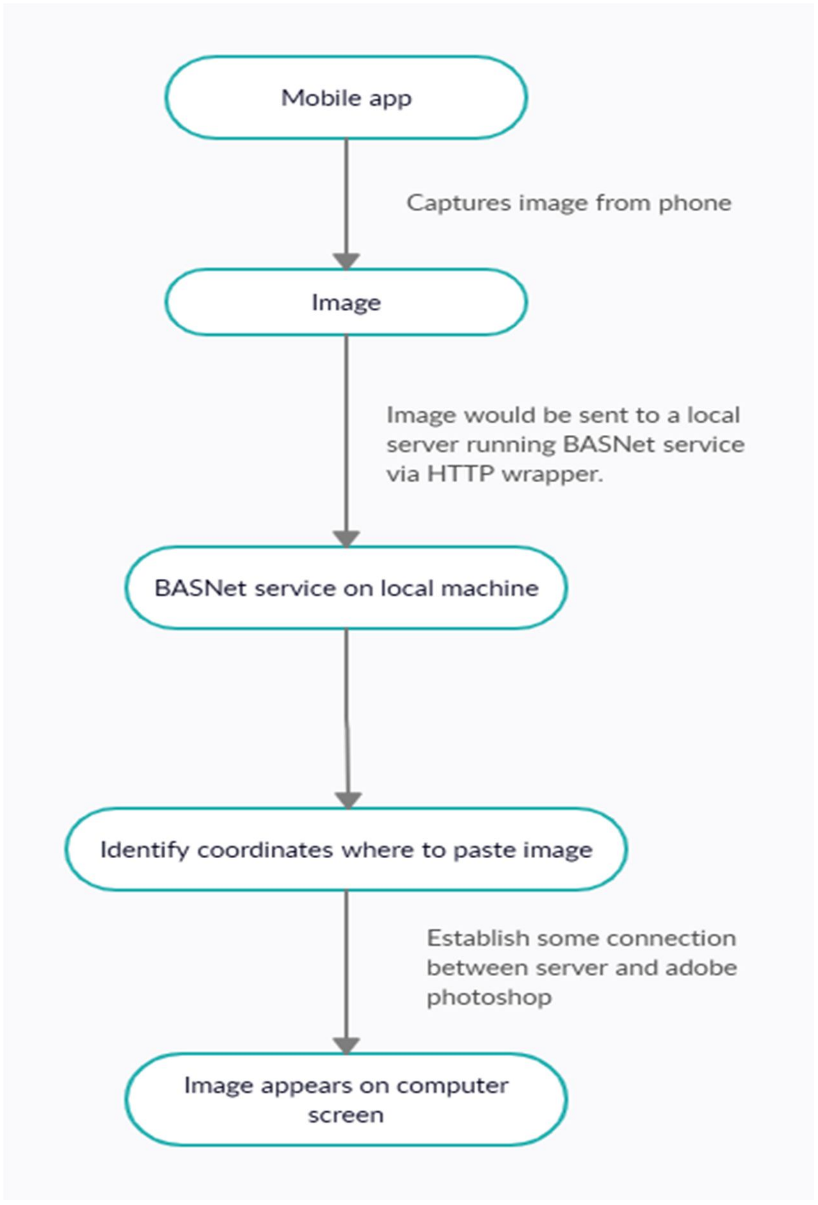

Mobile app
Captures image from phone
Image
Image would be sent to a local server running BASNet service via HTTP wrapper.
BASNet service on local machine
Identify coordinates where to paste image
Establish some connection between server and adobe photoshop
Image appears on computer screen

Figure 11. Activity Diagram

While the object is appearing, the phone can be moved around with the object and screenshots can be taken with the object from the phone in a different environment, making it appear as if the object was present over there all along.

We need to run and execute the HTTP service wrapper for BASNet on our machine. Ideally the BASNet service should be running on a dedicated server but for this project we are enabling our own PC to be the local server as well as the client.

*1)* First for the BASNet service we clone this GitHub repository [1] in which the BASNet source code is provided along with the pre trained model for salient object detection. We just need to ensure that BASNet works on our local machine, so by editing the 'main.py' file in BASNet-http folder and in the end change the host parameter which is set as (0.0.0.0) to your own local PC IP address which in our case is (192.168.0.105). Save the file. Run the file main.py by running the command (python main.py). The commands have to be executed on anaconda prompt [5] [7].

*2)* Now we can test the service with a sample image. For this open another separate anaconda prompt instance. Navigate to desktop and run the given command to execute (curl -F "data=@picture.jpg" http://localhost:8080 -o result.png)

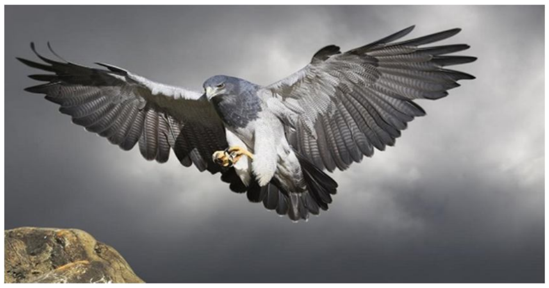


Figure 12: Sample BASNet input image of eagle [6]

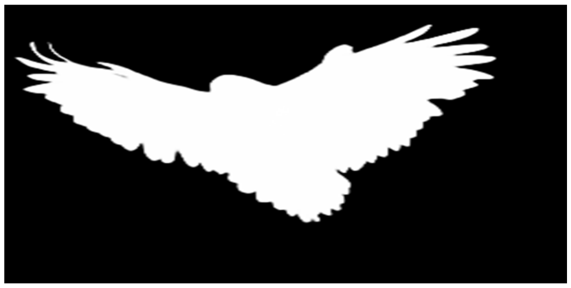


Figure 13: Sample BASNet Output image of eagle

*3)* The mask of the object is generated for the salient object detection through the BASNet HTTP wrapper on our machine. The resultant image is compressed to 256x256 resolution for further input purposes.

*4)* To run the app from the client side we need to download and install expo. After doing that in another anaconda prompt run the 'npm start' command which will run the application on the local machine. After which a browser link would open with a QR code would appear. This code will also appear on the prompt window. This code has to be scanned either from the browser window or the prompt by the expo app.

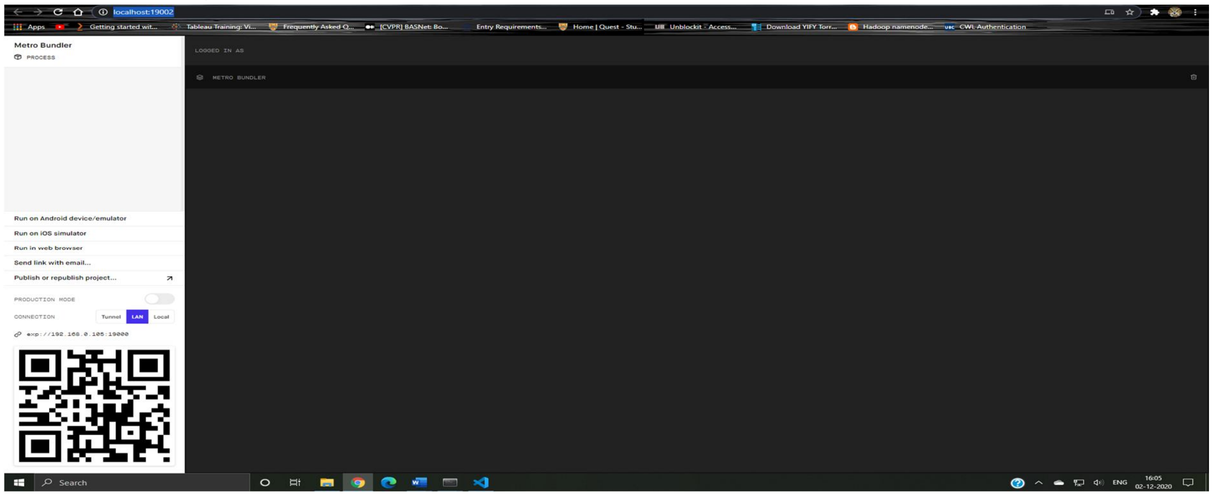


Figure 14: Browser window for execution of app through expo

*5)* Once the code is scanned the project will first build the javascript bundle then the app would start. The app on starting will look like this which is just like the default camera of your device.

*6)* Identify the object you want to extract, point the camera towards it and then press and hold the mobile screen to capture and extract the object.

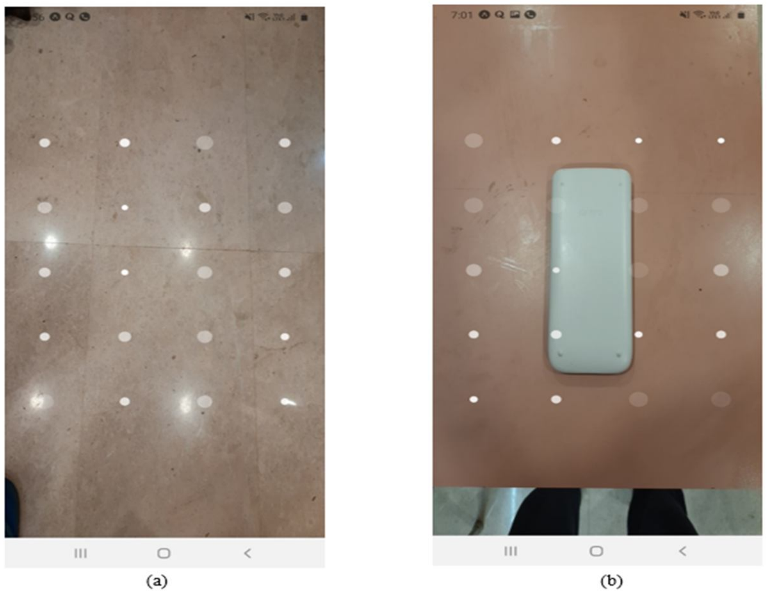


Figure 15: (a) shows the extraction indicator that is displayed while the user had pressed and held the mobile screen. Right now, there is no subject in the image. (b) It shows the extraction of a calculator in process, notice how the calculator is present in the view of the animated circles.

*7)* The extraction process might take around 10 – 20 seconds depending on how much the camera screen is being shaken by the user. Once the extraction is done the object would appear on the mobile phone screen as long as the user has been pressing the mobile screen.

*8)* While the object is appearing, the phone can be moved around with the object and screenshots can be taken with the object from the phone in a different environment, making it appear as if the object was present over there all along.

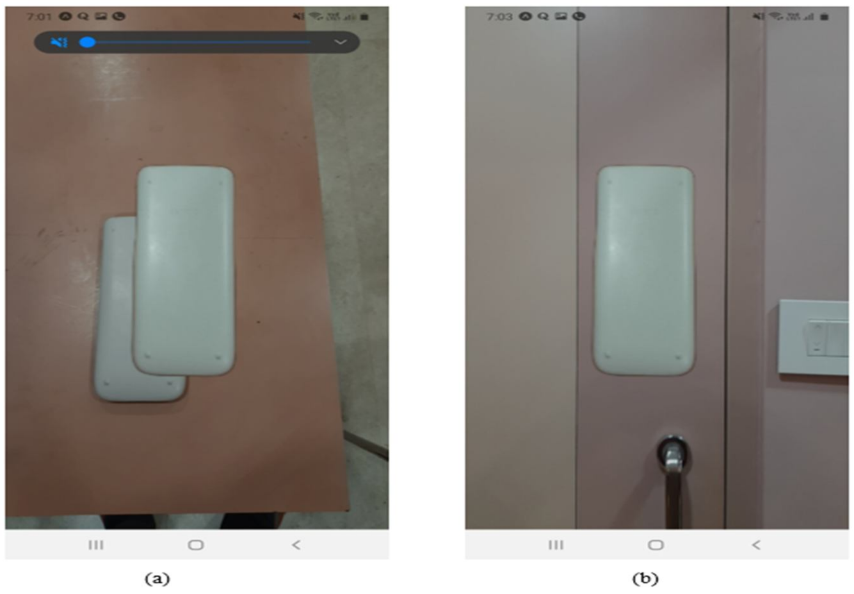


Figure 16: (a) After the image has been extracted then it can be moved around the environment. (b) With proper positioning the extracted image can then be placed upon the environment and a screenshot from the mobile phone can be taken.

To ensure the entire project runs smoothly we would need to match certain hardware and software requirements.

### *D. Hardware Requirements*

*1)* A computer with at least 8gb RAM to ensure smooth executing of different processes.
*2)* A mobile phone either of android or iOS with the expo app installed (not required for iOS).
*3)* Camera functionality is required in the phone.
*4)* A Nvidea graphic card (GPU) can be used to drastically speed up the entire execution time of the app, however it is not mandatory.

### *E. Software Requirements*

For the different modules and services different python libraries and modules were required to be installed as pre requisites for the app to even execute.

### *F. Overall Requirements*

*1)* Python 3 programming language has to be installed.
*2)* Anaconda navigator and anaconda prompt has to be installed.

### *G. BASNet HTTP Service Requirements*

The following python libraries are required for the execution of BASNet service.

*1)* (Flask==1.1.1)
*2)* (flask-cors==3.0.8)
*3)* gunicorn==19.9.0
*4)* numpy==1.15.2
*5)* scikit-image==0.14.0
*6)* Pillow==6.2.2
*7)* torchvision==0.2.1
*8)* pytorch

### *H. Server Requirements*

The following python libraries are required for the server.

*1)* Flask==1.1.1
*2)* flask-cors==3.0.8
*3)* pyscreenshot==1.0
*4)* Pillow==7.1.2
*5)* requests==2.23.0

### *I. App Requirements*

*1)* Node.js is required to be downloaded from the expo.io website.
*2)* The expo app is to be installed on the mobile phone in case of android. For iOS it is not required.

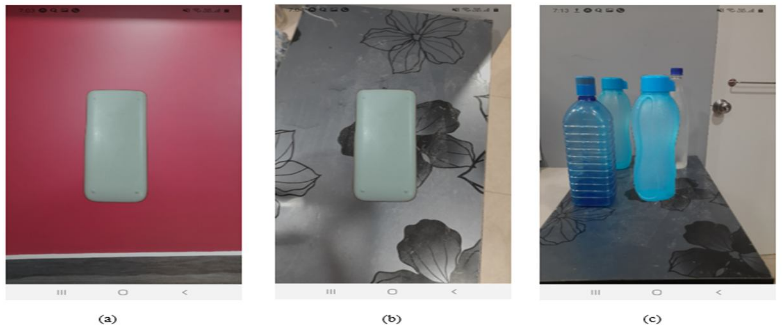


Figure 17: (a), (b) and (c) all show different outputs of the extracted image.

## IV. FUTURE SCOPE

The project is half way done, with extraction of the object from the image is done. The part of transmitting the extracted image to the computer is remaining which we plan to do next semester. Right now, the extraction process takes about 10-20 seconds depending on whether the user has held the camera still or not. The performance of the BASNet service is also responsible for the performance. However generally the results have been good.The project is half way done, with extraction of the object from the image is done. The part of transmitting the extracted image to the computer is remaining which we plan to do next.
Next, we plan to transfer the extracted image to a computer screen by holding the mobile camera and positioning it in the direction of the computer screen to paste it. We have few ideas as to how we would do this and achieve but right now nothing is solid about it. If time permits, we are also thinking of adding OCR (Optical Character Recognition) functionality.

## V. CONCLUSIONS

This project is designed to shorten the process of transferring images from our phones to our computers through Emailing, Bluetooth, Whatsapp, etc by bypassing all of this. Instead, we would capture the image from our phones, extract only the subject of the image and then transfer that extracted image directly to our computer screen. For now, we are only considering Adobe Photoshop, an image and graphics editing software as the screen on which the extracted image would be displayed. From there the image can then be further edited in photoshop, copied or saved on the computer. This type of application would be helpful for editing images by photography professionals or just for transferring images quickly since it would cut down the work required

## REFERENCES

[1] https://github.com/NathanUA/BASNet
[2] https://openaccess.thecvf.com/content_CVPR_2019/papers/Qin_BASNet_Boundary-Aware_Salient_Object_Detection_CVPR_2019_paper.pdf
[3] https://medium.com/@SeoJaeDuk/cvpr-basnet-boundary-aware-salient-object-detection-a025c7f7d94f
[4] https://openaccess.thecvf.com/content_cvpr_2018/CameraReady/1251.pdf
[5] http://tutorialspots.com/centos-7-how-to-install-basnet-http-server-6004.html
[6] https://www.visityork.org/explore/national-centre-for-birds-of-prey-p795431
[7] https://drive.google.com/file/d/1s52ek_4YTDRt_EOkx1FS53u-vJa0c4nu/view
[8] https://reactnative.dev/