Syed Saad Ahsan

March 10, 2020

Capstone Project

Machine Learning Engineer Nanodegree

Definition

Project Overview:-

In this project, we are going to implement an image segmentation algorithm. Image segmentation is a field of computer vision where the task is to partition an image into different segments assigning labels to different pixels often to help in identifying different regions. Most often, this means identifying interesting shapes and/or objects in the image.

In this particular project, we investigate a foreground identification image segmentation algorithm. The goal of this class of image segmentation algorithms is to differentiate between the background portion of the image and the foreground portion. The idea is that a quick and efficient method for generating a saliency map to partition out the foreground can be used to select regions of interest – which can then be extracted for higher level information. This is often very useful for e-commerce applications where background removal helps speed learning.

In this project, we recreate in open-source the algorithm described in detail in the paper by Wang et.al. "GraB: Visual Saliency via Novel Graph Model and Background Priors" [1]. This is an unsupervised learning algorithm that utilizes a graph structure with geodesic distance measured from regional texture and color cues with idea of background prior associated with the boundaries. We use the THUS10K [2] dataset to do the training, validation and testing for this project.

Problem Statement:-

The goal of this project is to create an open-source code that implements the algorithm as described in the paper by Wang et.al. [1] as follows:-

1) Download and explore the THUS10K dataset [2]
2) Segment image into superpixels via Simple Linear Iterative Clustering and build regional field descriptor from color and texture information.
3) Construct an undirected weighted graph where each set of nodes corresponds to superpixel while edges are constructed using graph model using weight matrix calculated using distances from extracted feature descriptors.
4) Select three borders as query seed and acquire saliency estimation assuming these borders are background.
5) Optimize saliency map

The final application is expected to be more effective than existing open-source code which produces similar Saliency maps.

Metric:-

The goal is to create a saliency map that has a binary classifier of either being background or foreground. Luckily for us, the THUS10K dataset [2] that we plan to use has an associated image with the ideal saliency map attached too. This allows us to calculate metric for or saliency maps that takes into account both precision and recall where

$$Precison = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negatives}$$

Here *True Positive* refers to this pixels that our algorithm identifies correctly, i.e. that are also labeled as such on the reference saliency map. *False Positive* refers to those pixels that the algorithm identifies as being salient which disagrees with the ideal saliency maps while *False Negatives* are those pixels identified by the model as not salient which the ideal saliency map does consider to be salient. Clearly, there is a tradeoff between Precision and Recall. Our metric will be to look at the tradeoff curve.

Another metric that we want to look at is time. One big advantage of an unsupervised learning approach is the time that is saved as opposed to using a supervised learning approach such as DRFI [3]. The algorithm as implemented in Wang et.al. [1] lists the time as implemented in MATLab code as 0.8msec. Thus, we ideally would want the time that this code takes in our open-source python implementation to be around 1 sec.

Analysis

Data Exploration:-

The THUS10K data set has roughly 10,000 images with pixel-level saliency labels as well. The images are taken from the MSRA Salient Object Database [2], which is a large dataset of more than 20,000 images with bounding boxes. However, these marked bounding boxes are often too coarse for fine-grained evaluation as observed by Wang and Li [4]. The database is open-source and can be downloaded from the following website https://mmcheng.net/msra10k/.

The dataset itself is split as follows. All the *.jpg images are the original three channel color images while all the *.png images are the binary single-channel images with the pixel-level saliency designation as seen in Fig 1. The images are of different sizes as well.

Data Exploration:-

A cursory screening of the images in the database after they are unzipped reveals that the images encapsulate a wide variety of different photographic styles and themes. Some of them are black and white images, most are colored while a few of them are mixed with some areas of the image in color and other areas in black and white too. In some images, the background is clear with things being less noisy while other images are quite busy where multiple things in the image could be considered to be

the foreground portion. In most images, one can quickly understand that the pixel-level saliency designations are actually quite subjective labels and it is possible to designate larger or smaller areas as salient foreground (see Fig 2).

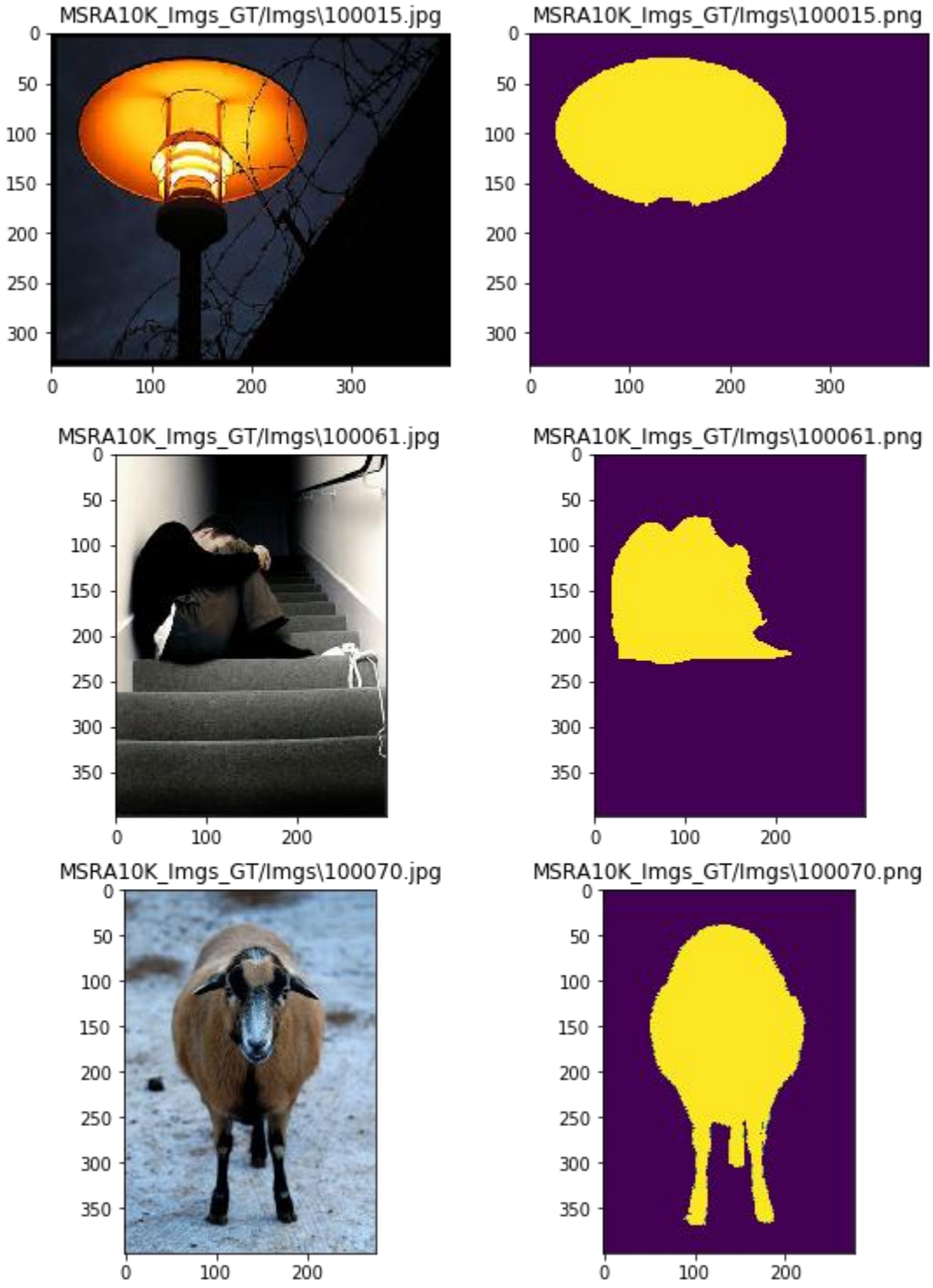

Fig 1. Some sample images from the THUS10K dataset. Original images on left, saliency maps on right.

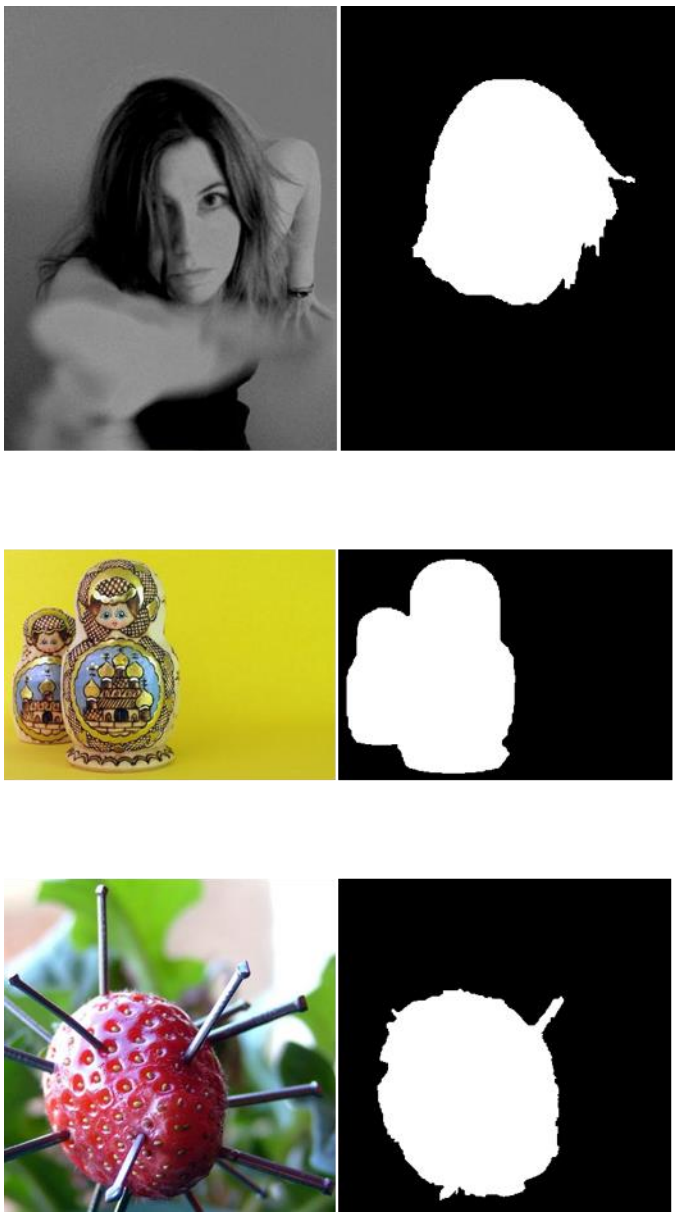

Fig 2. While most are color, some images are grey-scale (top). Some of the images can be very cleanly segmented into foreground and background maps (middle) while other images are more busy and such labeling are more subjective (bottom) and (top).

Algorithms and Techniques:-

The unsupervised saliency map generator that we replicate from the paper in Wang et.al. [1] is based on graph based manifold ranking model with geodesic distance similar to that proposed by Yang et.al. [5] In addition, boundary prior sampling is used similar to Li et.al. [6] with multi-scale fusion as in Yan et.al. [7]. Unlike, supervised methods that require quite a bit of training to be done, this algorithm is state-of-the-art, fast and does not require any training data set. However, this does require that the selection of the hyper-parameters be ideal which is hard to gauge.

In the end, the saliency map generator will spit out an assigned probability for the two classes designated as a number between 0 and 1 where 1 corresponds to it being foreground; this is then used to reduce the number of false positives by setting a threshold. The tradeoff here is, of course, that there are tradeoffs between precision and recall.

The following parameters can be tuned to further optimize the classifier. We suggest this ought to be done by any user that decides to use this for larger databases through a grid search. The hyper-parameters saved in default here are almost certainly not optimal.

- Classification threshold as defined above
- Hyper-parameters defined in the final function call
    - N_segments: The number of super-pixels generated in the lowest pyramid image
    - l1: Similar to λ1 in Wang et.al. This designates the weighting parameter on distance calculations from differences in average color features between superpixels
    - l2: Similar to λ2 in Wang et.al. This designates the weighting parameter on distance calculations from chi-squared distance of color features between histograms of 2 superpixels with K number of bins.
    - l3: Similar to λ3 in Wang et.al. This designates the weighting parameter on distance calculations from chi-squared distance of texture features between histograms of 2 superpixels with K number of bins.
    - s2: This is similar to constant σ^2 in Wang et.al. This designates the decay ration of distance to weights in the graph model.
    - K: This is number of bins specified in the histograms for the chi-squared distance calculated as above.
    - mew: This is similar to $1/(\mu + 1)$ in Wang et.al. μ represents the controlling parameter in ranking assignment algorithms as discussed in Zhou et.al.
    - damp1: This is the damping constant used in Tikhonov regularization. We use this damping constant to regularize the inversion for the first time we perform the ranking assignment for background priors. This is to ensure that we capture as much of the background in the image as possible circumventing the fact that our inversion matrix is often singular and pseudo-inverse often gives unstable results with high rankings for some super-pixels.
    - quant: This is the quantile to which we normalize our inversion the second time we perform the ranking assignment for foreground priors. Here we want to normalize the probability priors and use 'thresh' below and use some quantile measure to set the ranking to.
    - thresh: This is the threshold to which we normalize our inversion the second time we perform the ranking assignment for foreground priors. We use this in conjunction with the 'quant' variable above to circumvent the fact that our inversion matrix is often singular and pseudo-inverse often gives unstable results with high rankings for some super-pixels.

After coding out our algorithm from scratch, we perform a quick grid search to optimize some of the hyper-parameters above using random 20 image baselines. We then perform testing on a random 300 images selected from the database.

Benchmark:-

We wanted to compare the saliency probability may to another open-source deployment which also replicates another state-of-the-art technique. In this case, we will compare this model to another implementation to generate saliency maps as reported in paper by Perazzi et.al. "Saliency filters: Contrast based filtering for salient region detection" [8] and as implemented in open source on github https://github.com/arifqodari/saliencyfilters. The primary comparison is to look at Precision Recall Response curves.

Another benchmark, we want to take a look at is the time it takes to process a single image. As reported in Wang et.al [1], this takes 800 msec on MATLab implementation. We aim for something similar here ~ 1sec which we benchmark on a MacBook Pro computer with 2.9 GHz 6-Core Intel Core i9 chip with Radeon Pro 560X 4 GB and Intel UHD Graphics 630 1536 MB card.

Methodology

Data Preprocessing:-

We did not perform any data preprocessing for our algorithm implementation here. The dataset that we use is already sufficiently clean. Wang et.al. [1] does discuss two different pre-processing steps, namely: guided image filtering and imaging smoothing via relative total variation; however, these pre-processing steps are optional.

Implementation:-

We implement the following algorithm following the steps as outlined in Wang et.al. [1]. These are as follows:-

1) Partition image into N super pixels and build a graph model
2) Calculate Weight and Geodesic Distance from local and global color and texture cues from each super pixel
3) Select three out of 4 borders as query seed to get initial estimate of background prior
4) Acquire initial saliency estimation by performing ranking assignment
5) Optimize the saliency estimation by minimization of cost function and then secondary ranking assignment to get final saliency estimation
6) Repeat steps 1-5 for multi-scale fusion for pyramid type length scales.

1) Super Pixel Partitioning and Graph Model Creation-

We use Simple Linear Iterative Clustering to partition the image into N segments to start out with. Keeping in mind that we will eventually perform these series of steps for pyramid multi-scale fusion, the lowest level of the pyramid, or the one at lowest scale, is chosen to have a suggested 300 segments. As can be seen in Fig 3, both 100, 200 and 300 preserve the boundary of various objects in the image. After this step, we define an adjacency matrix, $A=[a_{ij}]_{N\times N}$ where $a_{ij}=1$ if super pixels s_i and s_j are adjacent, otherwise $a_{ij}=0$. We also define a border set, *B,* of all the super pixels on the border.

Additionally we also define the different groups of edges in the graph model based on the following three rules as defined in Wang et.al. [1]:-

Rule 1: All super pixels adjacent to one another and one-hop away are connected.

$E1 = \{ (s_i,s_j) | s_i,s_j \in S, a_{ij} = 1 \} \cup \{ (s_i,s_k) | s_k \in S, a_{kj} = 1 \}$

Rule 2: All super pixels are also connected to each super pixel in the Border set, *B*. However, the final weight calculations will be weighted such that these weights are divided by total number of super pixels in the border set.

$E2 = \{ (s_i,s_j) | s_i \in S, s_j \in B \}$, $w_{ij} = \text{weight}(r_i,r_j)/|B|$

Rule 3: Finally all super pixels in the Border set are connected to each other via an edge too which is not normalized as in rule 2.

$E3 = \{ (s_i,s_j) | s_i,s_j \in B \}$

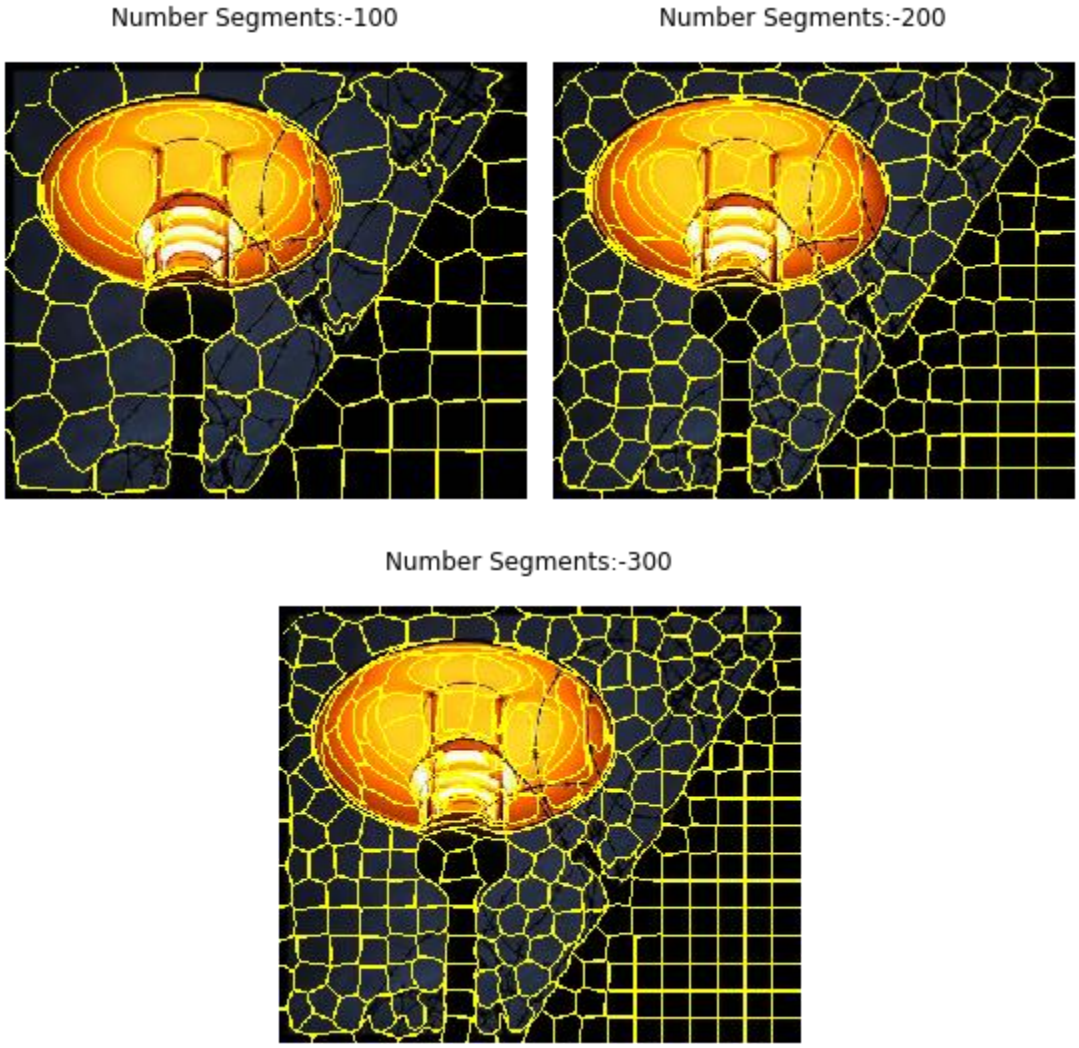

Fig 3. Reference image after SLIC partitioning with suggested N=100 segments on (top left), N = 200 segments on (top right) and N=300 segments on bottom. Note that the implementation attempts to segment to suggested N, however final segmentation varies slightly

2) Weight and Degree Matrix Calculation from Color and Texture Cues-

Next we attempt to extract regional feature descriptors from color and texture cues. As in Wang et.al. We do this by extracting color via CIELAB color space for each super pixel. This gives us a three channel image. Each channel is multiplied by one another to get a single channel.

For texture, we utilize the responses from the Leung-Malik (LM) filter bank. The code to generate the filter bank is obtained from github open source github.com/CVDLBOT/LM_filter_bank_python_code.git [9]. The code is modified slightly to make it work in python 3 and to update the scale parameters slightly (see Fig 4).

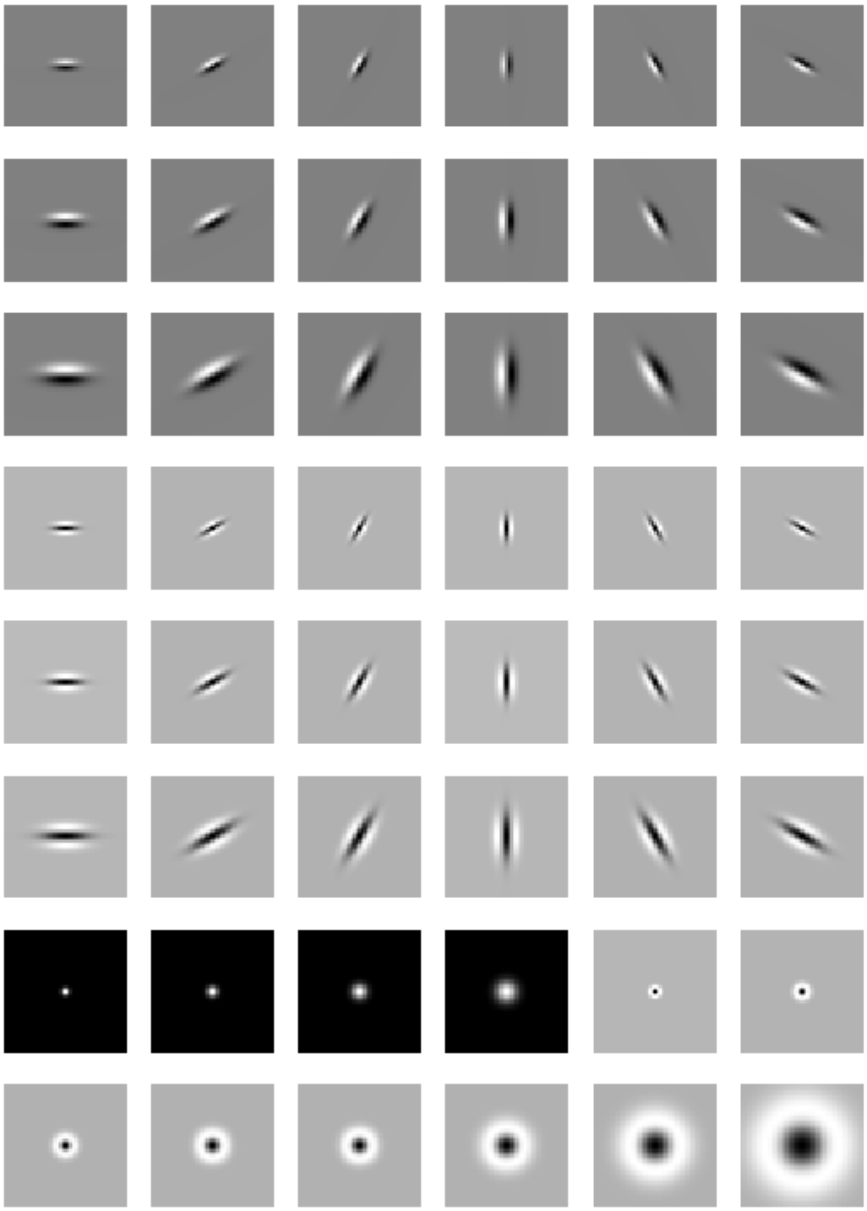

Fig 4. The Leung-Malik filter bank contains 48 filters that encapsulate a wide variety of texture cues as seen above.

We use scipy signal.fftconvolve to get the filter responses from the image – this seems to be the fastest implementation to get filter responses available in open source. After getting filter responses from each filter against a grey-scale version of the original image, we have a stack of 48 image responses. We take the max value of this stack at each pixel to get a single stack image that corresponds to the maximum Leung-Malik filter response.

While Wang et.al. does not specify this, both of these images are then scaled to have values between 0 and 1. Ultimately, this normalizes the input that we provide to the geodesic distance and weight calculators (see Fig 5).

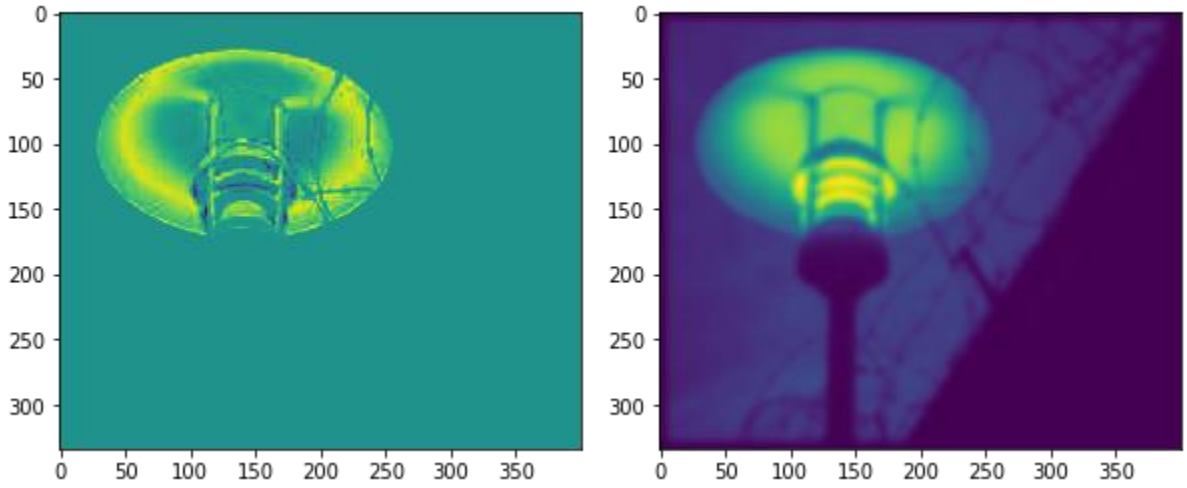

Fig 5. Image on left represents color features. This is obtained by converting original image to CIELAB Color space and multiplying L*a*b. Image on right is the maximum response for each pixel to the 48 filters in the Leung-Malik Filter bank.

Once we have these two images, we can define v^{lab}, h^{lab}, h^{tex} to be the mean L*a*b* color, L*a*b* histogram and max LM response histogram of superpixel, *s*, respectively. We define the distance as:

$$Dist\ (s_i, s_j) = \lambda_1 * || v_i^{lab} - v_j^{lab} || + \lambda_2 * \chi^2 (h^{lab}{}_i , h^{lab}{}_j) + \lambda_3 * \chi^2 (h^{tex}{}_i , h^{tex}{}_j)$$

where λ_1, λ_2, λ_3 are weighting parameters that we define in the hyper-parameters section in Algorithms and Techniques above. Here

$$\chi^2(h_1, h_2) = \sum_{i=1}^{K} \left(2 \frac{(h_1(i) - h_2(i))^2}{h_1(i) + h_2(i)} \right)$$

where K is the number of bins in the histogram. Wang et al don't specify either the value of K that is used or the range of the histograms. One could possibly have two options, setting the range between the minimum and maximum of the values in the union of h1,h2 or to have a pre-set range. Here we set the range of the histograms to between 0 and 1. In this particular case, we also realized that the creations of such histograms was taking a huge amount of time, thus, we experimented with a few different open-source histogram creation options and found that fast_histogram code from

https://github.com/astrofrog/fast-histogram [10] provided the fastest implementation which we suggest using.

For the rest of the algorithm, we use the values suggested by Wang et al [1] where $\lambda_1 = 0.25$, $\lambda_2 = 0.45$, $\lambda_3 = 0.3$. While these values are certainly not ideal given the fact that the images are normalized differently, we leave it to the final user to tune these parameters before use.

We use the distance formula above to calculate the distance between adjacent super pixels. For non-adjacent super pixels, the reference paper in Wang et al is rather unclear as to how to calculate distance. In this implementation we simply use Dijkstra's algorithm to calculate the minimum distance to all other nodes giving us a non-directional graph.

We then proceed to calculate the weight for each edge that is defined in our graph using the formula

$weight\ (s_i\ ,\ s_j\ ) = exp\ [\ Dist\ (s_i,\ s_j)\ /\ \sigma^2\ ]$

keeping in mind to normalize by dividing the weight for edges in Edge Group 2 by the number of super pixels in the Border Set. Here σ^2 is a constant that defines the decay ration of distance to weights.

Finally after we have defined our weight matrix, $W = [\ w_{ij}\ ]_{N \times N}$ from the rules above, we calculate the degree matrix , D, where $D = diag(d_1,d_2,...,d_n)$, where $d_i = \sum_j (w_{ij})$.

3) Select three out of 4 borders to generate query seed to get initial estimate of background prior

Next we make the assumption of the background prior. This prior is that the foreground object seldom touches the boundary. In this case, we generate an initial query seed for the background prior using the borders as seeds.

To do this, we agglomerate all the superpixels in the four borders into one super pixel. Then, we calculate the 4 by 4 weight matrix of these four boundary agglomerated super pixels using the same techniques as discussed above. From the weight matrix, we calculate the degree matrix and the boundary with largest degree number is then removed.

The initial estimate of the query seed, $y_b\ (i)$, is set to be 1 if corresponding super pixels, $s(i)$ is within one of the three boundary, b, specified above; else, it is set to 0.

4) Acquire initial saliency estimation by performing ranking assignment

After having an initial query seed, y_b, for each of the three boundaries, we can obtain a ranking function, f_b, that ranks all the super pixels by relevance to the query seeds provided. Zhou et al [11] provide a great deal of detail as to how these ranking searches are done in graph-based model in their paper. For simplicity's sake, we simply use their optimized result similar to that used in Wang et al where

$$f_b = (D - \frac{W}{\mu\ +\ 1})^{-1} y_b$$

where μ is a controlling parameter; similar to Wang et al, we set $1/\ \mu + 1$ to 0.99. We advise users that use this algorithm to further tune this parameter as required.

One thing to note here is that the matrix that needs to be inverted is singular and therefore, cannot be easily inverted. Additionally, the pseudo-inverse is somewhat instable giving huge values for the ranking for some of the pixels. Wang et al don't discuss this problem but we solve this issue by using Tikhonov regularization. This gives us a smoother distribution of ranking values that highlight different regions of suspected background. We then normalize this ranking vector, f_b, to values between 0 and 1.

To get the initial saliency estimation, for each pixel, *i*, we use formula:

$$S_b(i) = 1 - f_b(i); \quad S_{init}(i) = \prod S_b(i)$$

Where we take 1 - f_b to calculate the saliency estimator from each boundary and then take product of each of these three estimators, to get initial saliency estimator as seen in Fig 6.

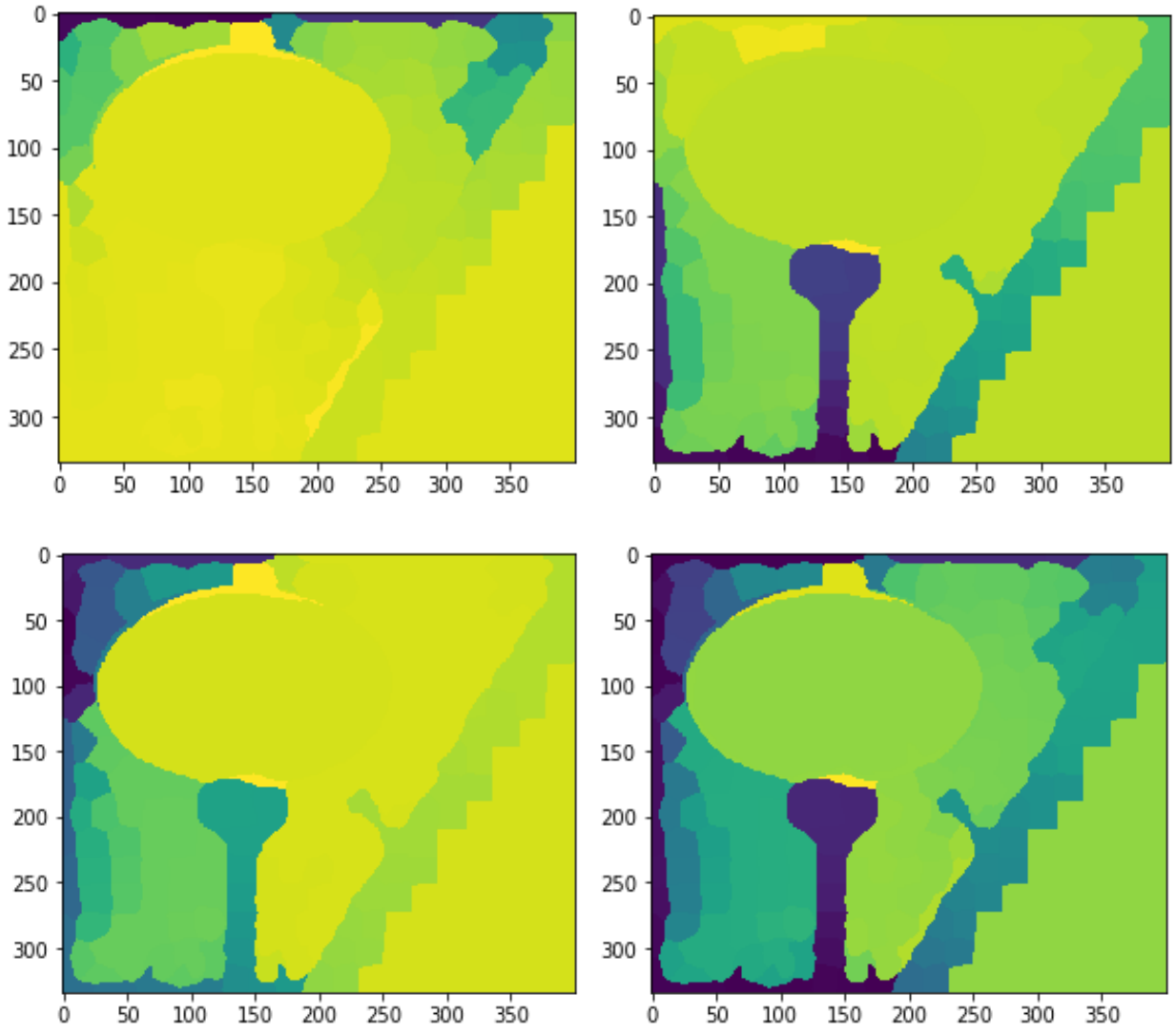

Fig 6. Boundary Saliency estimator from top boundary (top left), bottom boundary (top right), left boundary (bottom left) and the Initial Saliency map from product of these three (bottom right).

5) Optimize saliency estimation

After we have our initial saliency map, we optimize further by minimizing cost function, as described by Wang et al.

$$S_{optim}(i) = \arg min_S \sum_{i,j} w_{ij} \, (S_{init}(i) - S_{init}(j))^2 + \sum_{i=1}^{N} F(i) \, (S_{init}(i) - 1)^2 + \, B(i) \, (S_{init}(i))^2$$

where *F(i), B(i)* are the conditional such that *F(i)* > mean(S_{init}) and *B(i)* < mean(S_{init}).

To speed up the implementation, we also feed the optimizer with the Jacobian function of the above which is relatively easy to calculate. This give us a new Saliency estimator. We get the final saliency estimator, $S_{final}\ (i)$, for this scale of image is calculated as follows:-

$$S_{final} = (D - \frac{W}{\mu \, + \, 1})^{-1} S_{optim}$$

Here we run into similar problems of matrix to be inverted being singular. We circumvent this issue by taking a pseudo-inverse and using the high quantile (i.e. 95th quantile value * threshold) to set a max value to. Finally, the values are all normalized between 0 and 1 as can be seen in Fig 7 below.

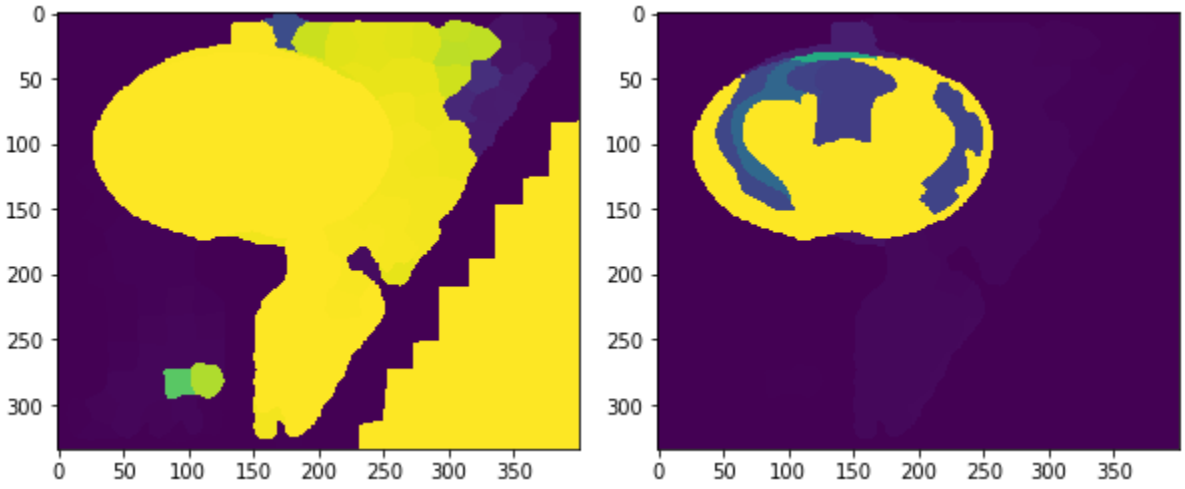

Fig 7. Saliency estimator post optimization, $S_{optim,}$ (on left) and final saliency estimator for image at this scale, S_{final} (on right).

6) Multi-scale fusion

We finally obtain the multiscale fusion by averaging final saliency estimator at multiple length scales. To do this, we implement L-layer Gaussian pyramid for robustness similar to that employed in Wang et al. First, we have discussed the techniques for the lowest level of the pyramid above, where we set N_segments at 300, n^0 = 300, and utilize the original image, I^0. For higher levels in the pyramid, the image, I^l, is obtained by following formula:-

$$I^l = \sum_{s=-2}^{2} \sum_{t=-2}^{2} w(s,t) I^{l-1}(2x + s, 2y + t), \qquad l \geq 1$$

Where w(s,t) is a Gaussian weighting function, while the number super pixel segments in the lth level of the pyramid, n^l, is obtained by following equation:-

$$n^l = \frac{n^{l-1}}{2^{2(l-1)}}$$

Similar to Wang et al, we set the maximum l to be 3, and take the average of the final saliency estimator at each layer to get the ultimate saliency map as shown in Fig 8.

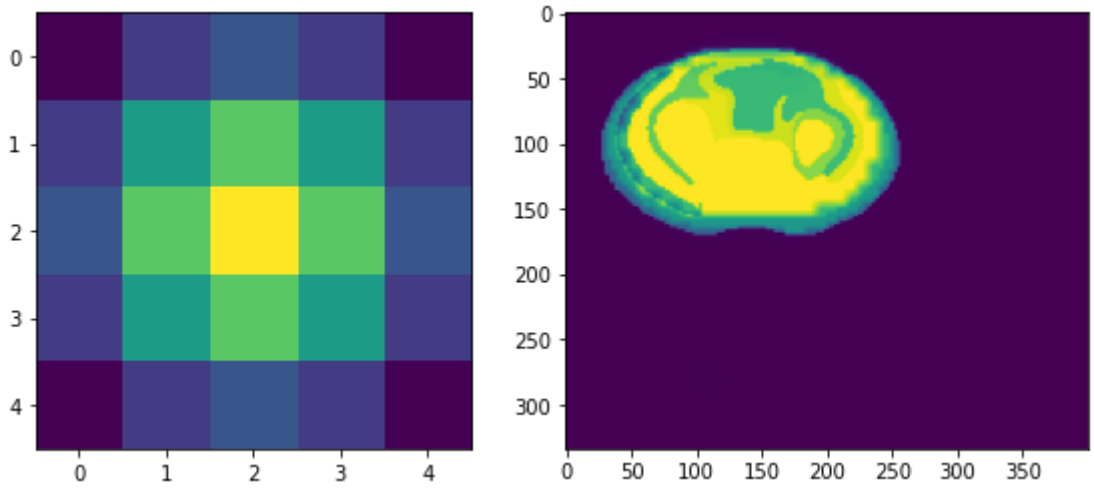

Fig 8. Gaussian weighting function, *w(s,t),* (on left), saliency map post multiscale fusion (on right)

Refinement:-

We further optimize the hyper parameters for our algorithm implementation above by performing a grid search. We randomly select 20 images from the dataset to tune our hyper-parameters and define cost as the average accuracy of the 20 images after placing a classification threshold of 50%. To be clear, our grid search is far from exhaustive and a more exhaustive grid search would yield better final results but we lacked the computational space on our labtop to do this.

Results

Model Evaluation and Validation:-

From our grid search, we are able to obtain accuracies of above 88% for the 20 images we use to perform the hyper parameter tuning on. When evaluated on a testing set of 300 images, the total accuracy with 50% threshold is close to 80% which is pretty good given the subjective nature of the saliency maps in the first place. Of course, depending on user need, the model can be tuned to optimize either recall, or precision and the threshold can be set appropriately as required. Additionally, the total processing time per image is found to be 0.8 sec which is what Wang et al also report [1].

From a subjective and casual overview perspective, the classification algorithm works quite well on a suite of images performing better for less busy images and less so for more busy complicated images as can be seen in Fig 9.

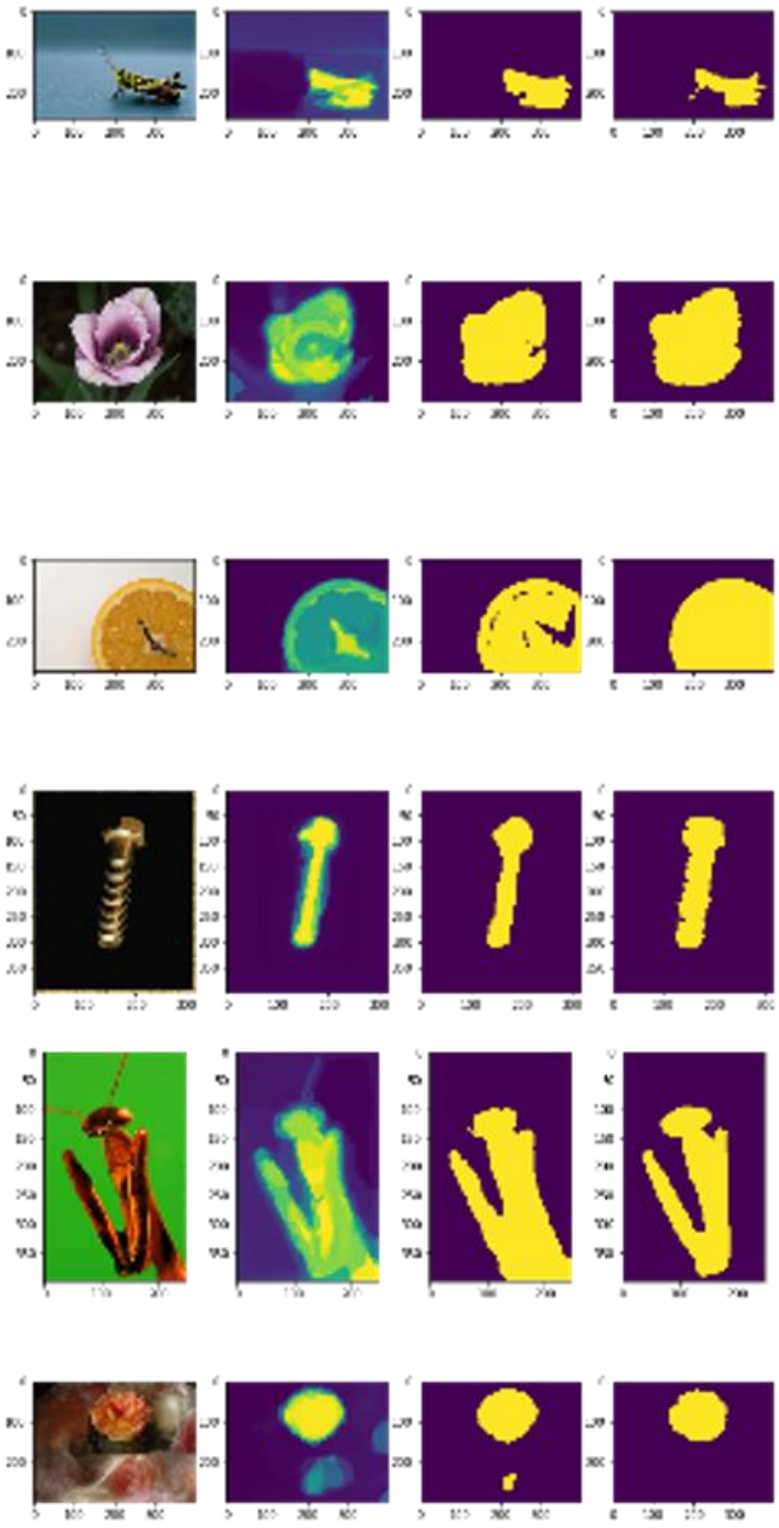

Fig 9. Some qualitative saliency maps above on a suite of images. From left to right, the original image, the multiscale fusion map from our algorithm implementation, the binary saliency map post thresholding, and the corresponding label from the THUS10K dataset

Justification:-

From the open-source github implementation of our benchmark model [12], we can see that the benchmark model performs fairly well at creating saliency maps from contrast based cues as seen in Fig 10. For benchmarking, we calculate the precision and recall values for a wide range of thresholds for both models to get a precision recall curve. As can be seen in Fig 10, our model performs significantly better than the benchmark model and there is further space for improvement by further hyper parameter tuning. We suggest that future work ought to focus on this improvement as well as to test whether this unsupervised model performs at the same level as supervised learning approaches from DRFI [3] for example as reported in Wang et al [1].

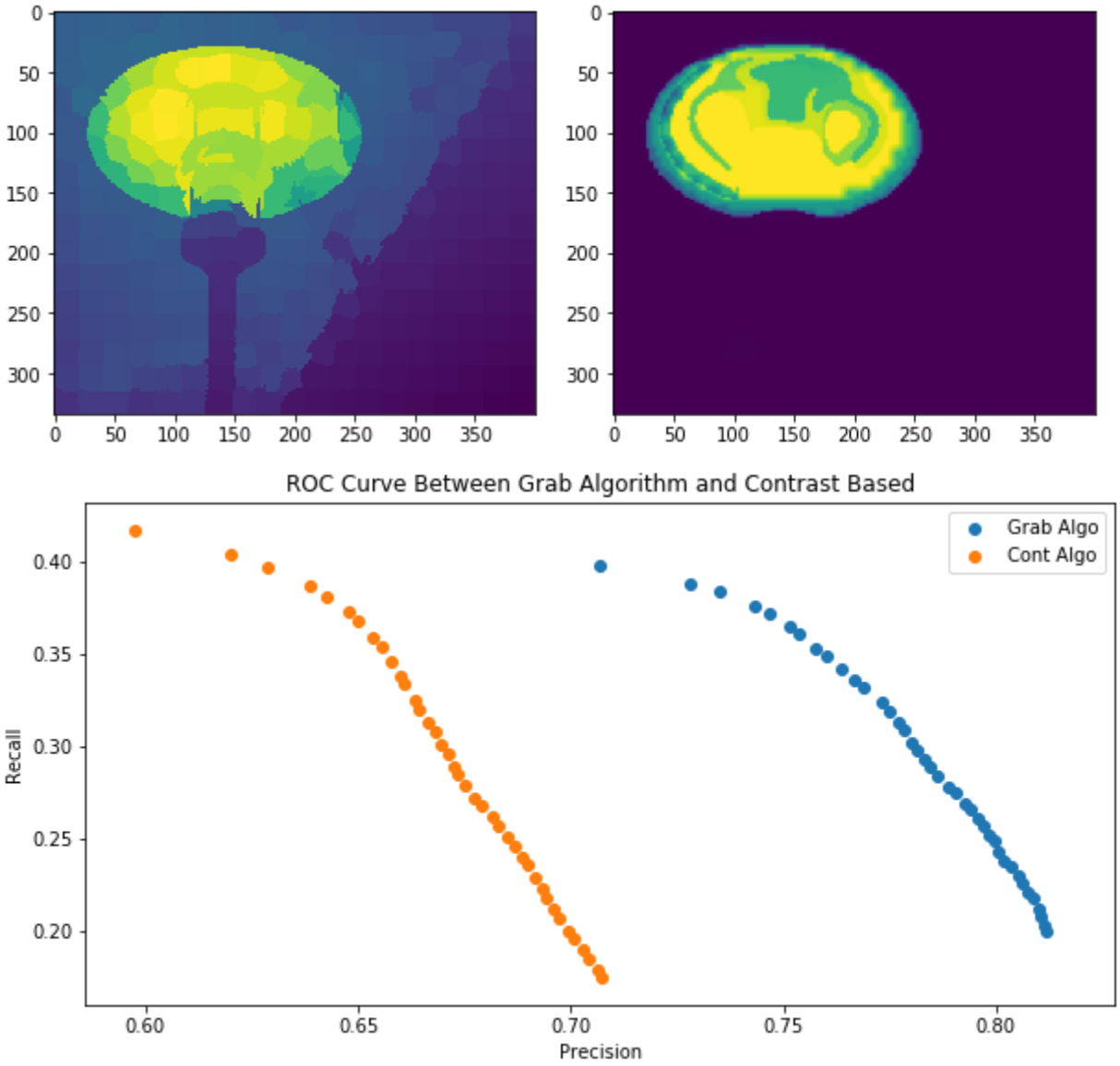

Fig 10. Saliency map from contrast based benchmark model (on top left). Saliency map from our implementation of GraB algorithm (on top right). ROC curve for Precision vs Recall between the two implementations. Grab Algorithm significantly outperforms other implementation.

References:-

[1] Wang, Qiaosong et al. “GraB: Visual Saliency via Novel Graph Model and Background Priors.” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016): 535-543.

[2] Deeply supervised salient object detection with short connections, Q Hou, MM Cheng, X Hu, A Borji, Z Tu, P Torr, **IEEE TPAMI**, 2018. https://mmcheng.net/msra10k/

[3] H. Jiang, J. Wang, Z. Yuan, Y. Wu, N. Zheng, and S. Li. Salient object detection: A discriminative regional feature integration approach. In CVPR, 2013.

[4] Wang, Z., & Li, B. (2008). A two-stage approach to saliency detection in images. In *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings* (pp. 965-968). [4517772] https://doi.org/10.1109/ICASSP.2008.4517772

[5] C. Yang, L. Zhang, H. Lu, X. Ruan, and M.-H. Yang. Saliency detection via graph-based manifold ranking. In CVPR, pages 3166–3173. IEEE, 2013.

[6] C. Li, Y. Yuan, W. Cai, Y. Xia, and D. D. Feng. Robust saliency detection via regularized random walks ranking. In CVPR. IEEE, 2015.

[7] Q. Yan, L. Xu, J. Shi, and J. Jia. Hierarchical saliency detection. In CVPR, June 2013.

[8] F. Perazzi, P. Kr¨ahenb¨uhl, Y. Pritch, and A. Hornung. Saliency filters: Contrast based filtering for salient region detection. In CVPR, pages 733–740. IEEE, 2012.

[9] github.com/CVDLBOT/LM_filter_bank_python_code.git

[10] https://github.com/astrofrog/fast-histogram

[11] D. Zhou, J. Weston, A. Gretton, O. Bousquet, and B. Sch¨olkopf. Ranking on data manifolds. NIPS, 16:169– 176, 2004.

[12] https://github.com/arifqodari/saliencyfilters