Saliency Detection using a Recurrent U-NET architecture

Binbin Ren
Georgia Institute of Technology
bren35@gatech.edu

Pranjal Gupta
Georgia Institute of Technology
pgupta332@gatech.edu

***Abstract*—Deep networks have been proven to capture high level semantic features and achieve superior performance in saliency detection. The U-NET is one such architecture, which is popular due to its relatively small number of parameters. In this paper, we take this model further by exploring a recurrent architecture of the original U-NET. This helps us incorporate saliency priors, for more accurate prediction. In addition, our model can be easily trained and does not require any complex training procedures, like pre-training and fine tuning, that some of the other models resort to, in order to obtain good results.**

***Index Terms*—Saliency detection, Saliency priors, pre-training, fine-tuning, U-NET**

I. Introduction

Saliency detection can be generally divided into two subcategories: salient object segmentation [1] and eye-fixation detection [2]. This paper mainly focuses on salient object segmentation, which aims to identify the most conspicuous and eye-attracting regions in an image. It has been used as a pre-processing step to facilitate wide range of vision applications. Although much progress has been made, it is still a very challenging task to develop effective algorithms capable of handling real world adverse scenarios.

Most existing methods address saliency detection with hand-crafted models and heuristic saliency priors. For instance, contrast prior formulates saliency detection as center-surrounding contrast analysis and captures salient regions either characterized by global rarity or locally standing out from their neighbors. In addition, boundary prior regards boundary regions as background and detects foreground objects by propagating background information to the rest of the image areas. Although these saliency priors have been proved to be effective in some cases (Fig. 1 first row), they are not robust enough to discover salient objects in complex scenes (Fig. 1 second row). Furthermore, saliency prior based methods mainly rely on low-level hand-crafted features which are unable to capture the semantics of objects. As demonstrated in the third row of Fig. 1, high level semantic information, in some cases, plays a central role in distinguishing foreground objects from background with similar appearance.

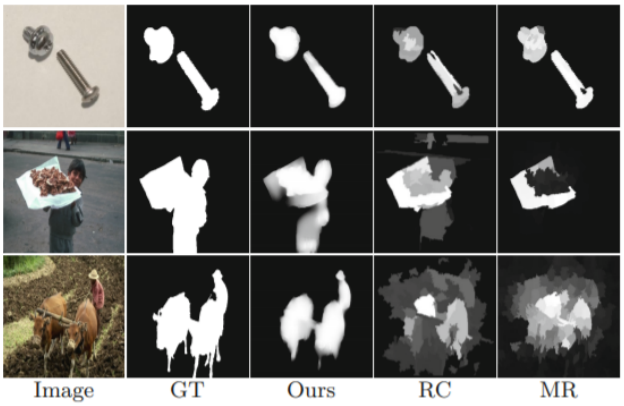

Fig. 1: Saliency detection results by different methods. From left to right: original image, GT mask, Authors proposed RFCN, RC [3] and MR [4]

Recently, deep convolutional neural networks (CNNs) have delivered record breaking performance in many vision tasks, like image classification [5], object detection [6], object tracking [7], semantic segmentation [8], etc. This suggests that deep CNNs are very effective at handling complex scenes by accurately identifying semantically salient features (Fig. 1 third row). Though better performance has been achieved, there are still a few major issues of the early CNN based saliency detection methods. First, saliency priors, which are shown to be effective, are completely discarded by most CNN based methods. Second, CNNs predict the saliency label of a pixel only considering a limited size of local image patch. This inevitably leads to incorrect predictions.

However, with feed-forward architectures, CNNs can not refine their predictions. Third, the training methods used in the past are sometimes very complex. Most people often resort to pre-training, fine-tuning, training with centrally cropped images, etc. However, it would be much better, if a model could achieve good results with a simple training method.

To address the above issues, we investigate a recurrent U-NET architecture for saliency detection. In each time step, we feed-forward both the input RGB image and a saliency prior map through the recurrent U-NET to obtain the predicted saliency map which in turn serves as the saliency prior map in the next time step. The prior map in the first time step is initialized by incorporating saliency priors from heuristic methods, such as contrast priors. Our recurrent U-NET architecture has three advantages over existing CNN based methods: a) Saliency priors are exploited to make training deep models easier, as well as yield a more accurate prediction; b) In contrast to feed-forward networks, the predicted saliency map of our network in each time step, is provided as a feedback signal, such that the recurrent U-NET is able to refine the saliency prediction by correcting its previous mistakes until it produces the final prediction in the last time step; c) While achieving good results, our approach does not employ any of the more complex training methods, such as pre-training, fine-tuning, etc. This would make it simpler and faster for people to train and deploy our model for their applications.

In summary, the contributions of our work is three folds. First, Saliency priors, which are widely ignored by other CNNs, are utilized by us to make training our models easier and yield more accurate predictions. Second, most CNNs employ a feed-forward architecture, that is unable to correct its inevitable mistakes. However, our network uses the predicted saliency map in each time step as a feedback signal, to correct its previous mistakes until the last time step. Third, our method does not resort to complex training methods, such as pre-training, fine-tuning, etc. This makes it easier and faster to train and deploy our model, while achieving results that are comparable to state of the art methods.

II. Methodology

This section discusses our technique in detail. Section 2.A talks about how we obtained our Saliency priors. Section 2.B discusses the conventional Fully Convolutional Network architecture(Fig. 2). In section 2.C, we describe our recurrent U-NET architecture. Finally in section 2.D, we mention our Training procedure.

A. Saliency Priors

In order to facilitate faster training and more accurate inference, we want to implement the heuristic saliency priors. We encode prior knowledge into a saliency prior map which serves as the input to the network. [9] To build a prior saliency map, we used the sailency model introduced from [10]. This is a featured-based salicency model. It will first extract the independent color and intensitiy channels from the RGB input images.

$$
\begin{aligned}
lI &= (r+g+b)/3 \\
R &= r-(g+b)/2 \\
G &= g-(r+b)/2 \\
B &= b-(r+g)/2 \\
Y &= \frac{r+g}{2} - \frac{|r-g|}{2}
\end{aligned}
$$

After that it will utilize the method of Gaussian Pyramids and Gabor Pyramids to get the different scales related figures for color channels and intensity channel. Then we can extract the center-surround difference features based on these Pyramids images for intensity, orientation and color.

$$
\begin{aligned}
I(c,s) &= |I(c) \ominus I(s)| \\
\mathcal{O}(c,s,\theta) &= |O(c,\theta) \ominus O(s,\theta)| \\
\mathcal{RG}(c,s) &= |(R(c)-G(c)) \ominus (G(s)-R(s))| \\
\mathcal{BY}(c,s) &= |(B(c)-Y(c)) \ominus (Y(s)-B(s))|
\end{aligned}
$$

In general we will get 6 feature maps of intensity contrast, 12 feature maps of color and 25 feature maps of orientation maps for each input image. These in total 42 feature maps will then be combined into the saliency map by the method of winner-take-all. But before we combined the maps, we will first normalize the maps by introducing a normalize parameter $\mathcal{N}$. Then the combined saliency map will be the mean of the feature maps for intensitity($\mathcal{N}(\bar{I})$) , color($\mathcal{N}(\bar{C})$) and orientation ($\mathcal{N}(\bar{O})$)

$$
\begin{aligned}
\mathcal{N}(F(c,s)) &= (\max(F(c,s)) - \bar{m})^2 (F(c,s) - \min(F(c,s))) \\
S &= \frac{1}{3}(\mathcal{N}(\bar{I}) + \mathcal{N}(\bar{C}) + \mathcal{N}(\bar{O}))
\end{aligned}
$$

B. Conventional Fully Convolutional Network architecture

Convolutional layers as building blocks of CNNs are defined on a translation invariance basis and have shared weights across different spatial locations. Both the input and the output of convolutional layers are 3D tensors called feature maps, where the output feature map is

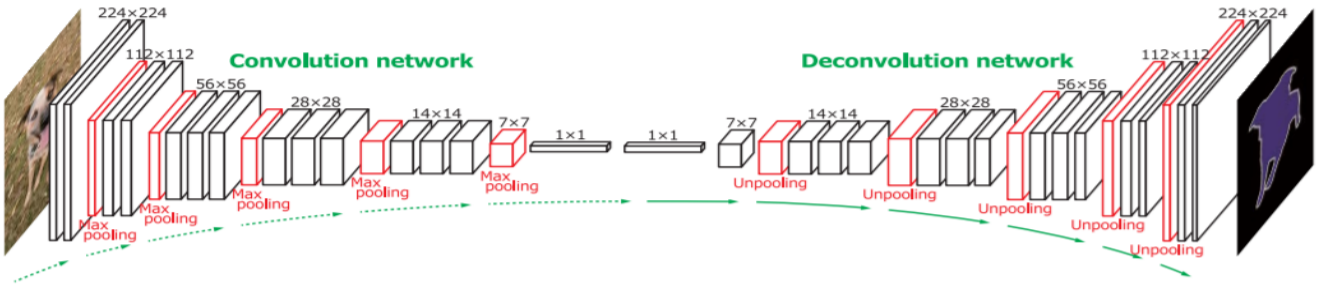

Fig. 2: A conventional Fully Convolutional Network architecture.

obtained by convolving convolutional kernels on the input feature map as

$$f_s(X; W, b) = W *_s X + b$$

Where X is the input feature map; W and b denote kernel and bias respectively; $*_s$ represents convolution operation with stride s. As a result, the resolution of the output feature map $f_s(X; W, b)$ is downsampled by a factor of s. Typically, convolutional layers are interleaved with max pooling layers and non-linear units (e.g., ReLUs) to further improve translation invariance and representation capability. The output feature map of the last convolutional layer can then be fed into a stack of fully connected layers which discard the spatial coordinates of the input and generates a global label for the input image. (See Fig. 3 (a))

For efficient dense inference, we can convert CNNs to fully convolutional networks (FCNs) (Fig. 3(b)) by casting fully connected layers into convolutional layers with kernels that cover their entire input regions. [8] This allows the network to take input images of arbitrary sizes and generate spatial output by one forward pass. However, due to the stride of convolutional and pooling layers, the final output feature maps are still coarse and downsampled from the input image by a factor of the total stride of the network. To map the coarse feature map into a pixel-wise prediction of the input image, FCN upsamples the coarse map via a stack of deconvolution layers (Fig. 3(c))

$$\hat{Y} = U_s(F_s(I, \theta); \psi)$$

where I is the input image; $F_S(I; \theta)$ denotes the output feature map generated by the convolutional layers of FCN with total stride of S and parameterized by θ. $U_s(.; \psi)$ denotes the deconvolution layers of FCN networks parameterized by ψ that upsamples the input by a factor of S to ensure the same spatial size of the output prediction $\hat{Y}$ and the input image I. Different from simple bilinear interpolation, the parameters ψ of deconvolution layers are jointly learned. To explore the fine-scaled local appearance of the input image, the skip architecture can also be employed to combine output feature maps of both lower convolutional layers and the final convolutional layer for more accurate inference.

In the context of saliency detection, we are interested in measuring the saliency degree of each pixel in an image. To this end, the FCN takes the RGB image I of size h x w x 3 as input and generates the output feature map $\hat{Y} = U_S(F_S(I; \theta); \psi)$ of size h x w x 2. We denote the two output channels of $\hat{Y}$ as background map $\hat{B}$, and salient foreground map $\hat{H}$, indicating the scores of all the pixels being background and foreground, respectively. By applying softmax function, these two scores are transformed into foreground probability as

$$p(l_{i,j} = fg \mid I, \theta, \psi) = \frac{\exp(\hat{H}_{i,j})}{\exp(\hat{H}_{i,j}) + \exp(\hat{B}_{i,j})}$$

where $l_{i,j}$ indicates the foreground/background label of the pixel indexed by (i, j). The background probability $p(l_{i,j} = bg \mid I, \theta, \psi)$ can be computed in a similar way. Given the training set $\{Z = (I, C)\}_1^N$ containing both training images I, and its pixelwise saliency annotation C, the FCN network can be trained end-to-end for saliency detection by minimizing the following 'binary-crossentropy' loss

$$\arg\min_{\theta,\psi} - \sum_Z \sum_{i,j} I.(C_{(i,j} = fg) \ln p(l_{i,j} = fg \mid I, \theta, \psi) + I.(C_{(i,j} = bg) \ln p(l_{i,j} = bg \mid I, \theta, \psi),$$

where $I.()$ is the indicator function. The network parameters θ and ψ can then be iteratively updated using stochastic gradient descent (SGD) algorithm.

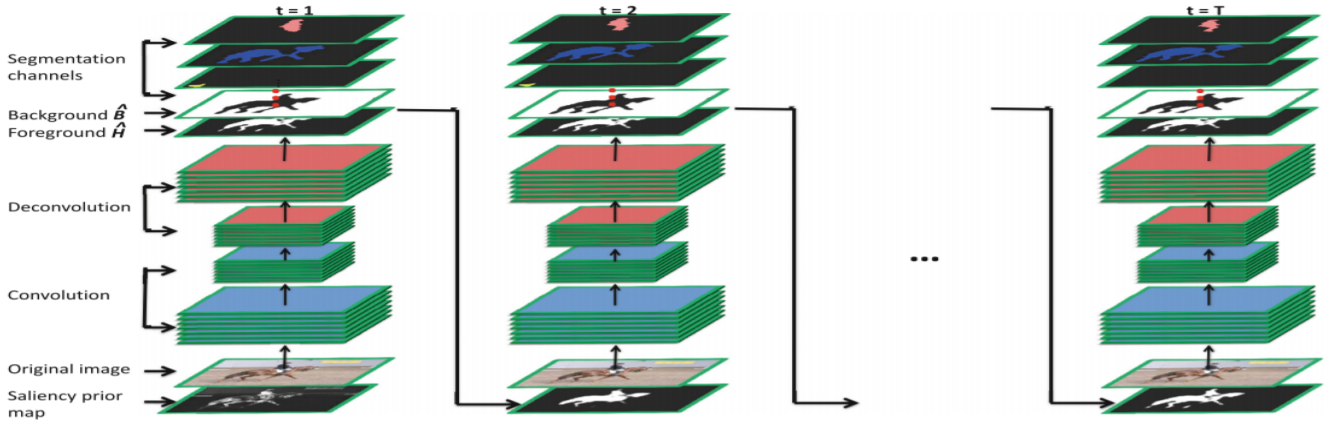

Fig. 3: Principle of the recurrent architecture

C. Recurrent U-NET architecture

The above FCN network is trained to approximate the direct nonlinear mapping from raw pixels to saliency values and ignores the saliency priors which are widely used in existing methods. Although, heuristic saliency priors have their limitations, they are easy to compute and shown to be very effective under a variety of cases. Thus, we believe that leveraging saliency prior information can facilitate faster training and more accurate inference. This has been verified by our experiments. We also note that the output prediction by FCN may be very noisy and lack of label consistency. However, the feed forward architecture of FCN fails to consider feedback information, which makes it impossible to correct prediction errors. Based on these observations, we make two improvements over the FCN network and design the RFCN by: (i) incorporating saliency prior into both training and inference; and (ii) recurrently refining the output prediction. (see Fig. 3)

As in Section 2.B, we divide our network into two parts, i.e., convolution part $F(.;\theta)$ and deconvolution part $U(.;\psi)$. Our architecture is almost completely based of the original U-NET (see Fig. 4). However, one major difference is that our recurrent architecture also incorporates the saliency prior map P into the convolution part by modifying the first convolution layer as.

$$f(I) = W_I * I + W_P * P + b,$$

where I and P denote the input RGB image and saliency prior,respectively; W_I and W_P represent their corresponding convolution kernels; b is bias parameter. In the first time step, our recurrent U-NET architecture takes the input RGB image concatenated with the Saliency Prior along the channel axis, as input. This gives the predictions of our network at the first time step as

$$\hat{Y}^1 = U(F(I, P\ ;\ \theta);\ \psi)$$

In the following time step, the prediction that is generated at the preceding time step is incorporated with the same input RGB image, i.e. it is the new Saliency prior. We can write this as

$$\hat{Y}^t = U(F(I, \hat{Y}^{t-1}\ ;\ \theta);\ \psi)$$

For the above recurrent architecture, forward propogation of the input image concatenated with the prediction of the previous time step, is conducted in every time step. The initial condition is $\hat{Y}^0 = P$, where P is the saliency prior.

To obtain the Saliency priors, we consider two methods: (i) Use the saliency priors from heuristics, such as contrast priors; (ii) Obtain the saliency priors from a U-NET that we trained, in order to get a more accurate Saliency prior. Both these methods work pretty well and converge after very few time steps.

D. Training Procedure

As we mentioned before, our training procedure is very simple. First, we obtain the PASCAL VOC 2010 semantic segmentation dataset. [11] This dataset contains about 2000 images and their corresponding semantic segmentation masks. Saliency detection and semantic segmentation are highly correlated but essentially different in that saliency detection aims at separating generic salient objects from the background, whereas semantic segmentation focuses on distinguishing objects of different categories. Since we are concerned with solving the

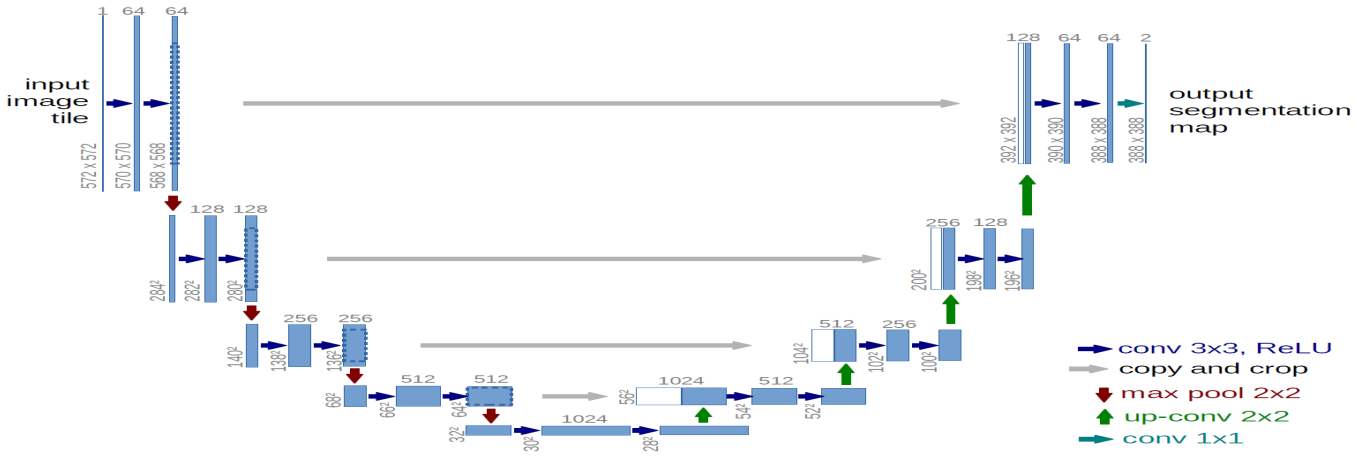

Fig. 4: The original U-NET architecture

former problem, we convert the semantic segmentation masks into binary saliency detection masks. These masks are also called ground truths (GT). We also normalize the RGB images (Divide by 255). Next, we resize these image to 224 x 224 x 3 (Since the images are of variable lengths). The mask along with the resized images consists of our dataset. We then perform a 10% validation split on this dataset. After this split, we have our training set, denoted by Z.

We then find the saliency priors using either heuristics, or a trained U-NET. This prior is then incorporated into the RGB image. Thus, the input is now 224 x 224 x 4. We pass this through our recurrent U-NET. At every time step, we compare the prediction of the network to the Ground truth. The loss function is then the binary-crossentropy between the prediction and GT at every time step. We call this the recurrent loss function, which is shown below

$$L(\boldsymbol{\theta}, \boldsymbol{\psi}) =$$
$$-\sum_{t=1}^{T}\sum_{Z}\sum_{i,j} I.(l_{i,j} = \boldsymbol{fg}) \ln p\left(l_{i,j} = \boldsymbol{fg} | \boldsymbol{I}, \hat{\boldsymbol{Y}}^{t-1}, \boldsymbol{\theta}, \boldsymbol{\psi}\right)$$
$$+ I.(l_{i,j} = \boldsymbol{bg}) \ln p\left(l_{i,j} = \boldsymbol{bg} | \boldsymbol{I}, \hat{\boldsymbol{Y}}^{t-1}, \boldsymbol{\theta}, \boldsymbol{\psi}\right)$$

Minimizing the above loss function with respect to the parameters θ and ψ gives us our trained model. As for choice of our hyperparameters, we just stuck with the De-facto standards used while training any similar deep neural network. We mention these in detail in our analysis section.

III. Results

In this section, we show the results obtained by our two recurrent U-NET models. As mentioned before, we have two ways of obtaining our saliency priors: (i) From Heuristics (ii) From a trained U-NET that takes as input RGB images only (no priors). These correspond to two different recurrent U-NET models; Heuristic priors recurrent U-NET and U-NET priors recurrent U-NET. To show the results, we randomly pick two images from the training set and then display the predicted saliency map at every time step. We do the same for two images from the validation set. We also report the pixel-wise training and cross validation accuracy over the training and validation sets. In addition, we show the learning curves to show the process of training. This will also help us observe the faster convergence.

The results obtained from the training and validation sets of our Heuristic priors recurrent U-NET model are shown in Fig. 5 and Fig. 6 respectively. The results obtained from our U-NET priors recurrent U-NET are shown in Fig. 8 and Fig. 9 respectively.The training

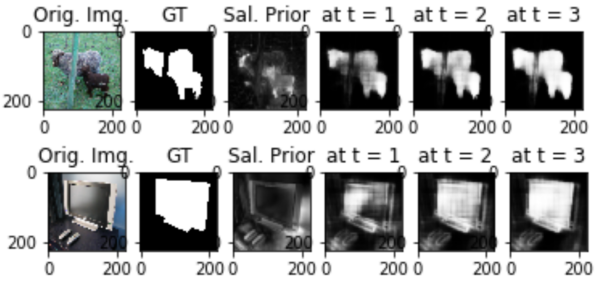

Fig. 5: Results of our Heuristic priors recurrent U-NET for two randomly chosen images from the training set

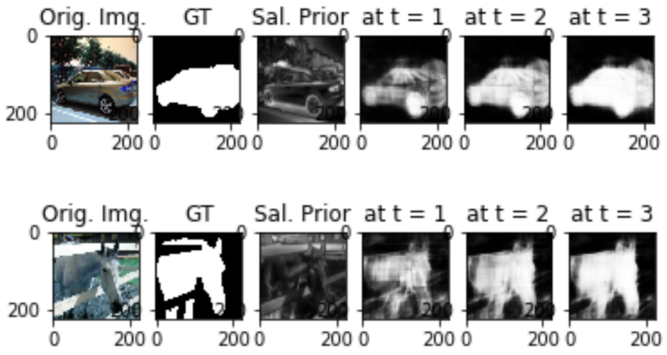

Fig. 6: Results of our Heuristic priors recurrent U-NET for two randomly chosen images from the validation set

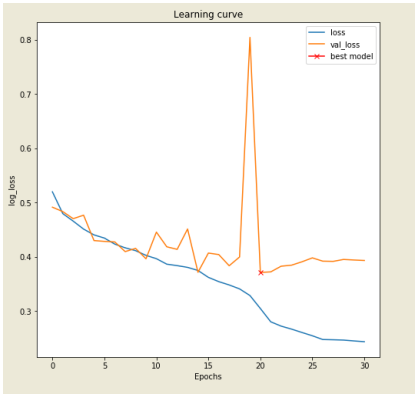

Fig. 7: The learning curve while training Heuristic priors recurrent U-NET

and validation accuracies of our two recurrent U-NET models are mentioned in Table. I. From the above results, we can see that the predictions from both our recurrent U-NET models are very close to the ground truth masks. Our training accuracy is higher than the validation accuracy, but not by too much. This shows that

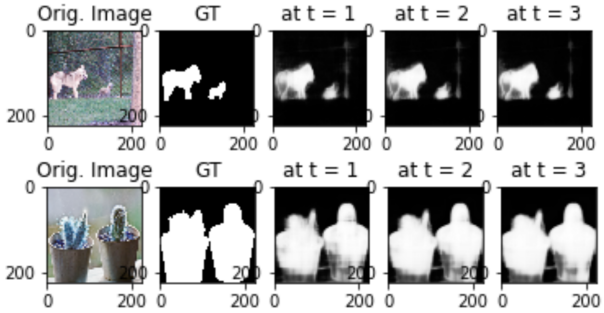

Fig. 8: Results of our U-NET priors recurrent U-NET for two randomly chosen images from the training set

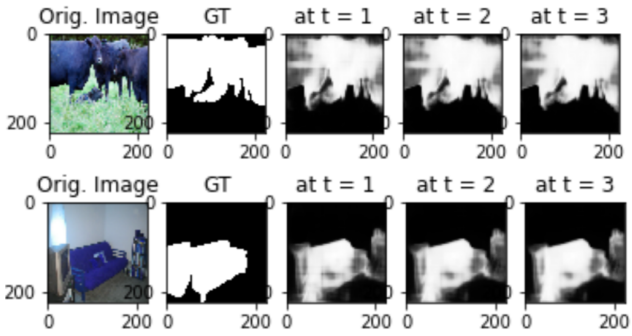

Fig. 9: Results of our U-NET priors recurrent U-NET for two randomly chosen images from the validation set

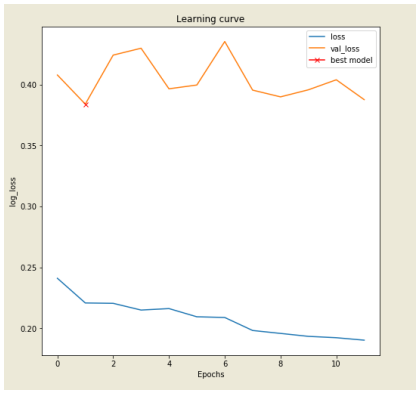

Fig. 10: The learning curve while training U-NET priors recurrent U-NET

our model does generalize well to unseen examples. In all the cases, our accuracy is above 83 %, which is quite good, considering our very simple training approach. In particular, the results from our Heuristics Priors recurrent U-NET model are dynamic, and improving over every time step. In contrast, the results from our U-NET priors recurrent U-NET are not quite as dynamic. But, our second model is definitely more accurate than our first one. Also notice from the learning curves (see Fig. 7 and Fig. 10), training our U-NET priors recurrent U-NET model is much faster than training our Heuristics priors recurrent U-NET model. These are some reasons why, if we have the computational resources, we prefer our second model.

TABLE I: Accuracy obtained by our two Recurrent U-NET models

Model	Feature-based Prior Map	trained U-NET
Training Accuracy	88.13%	91.69%
Validation Accuracy	83.56%	83.75%

IV. Thorough and Detailed Analysis

In this section, we try to explain our thought process while trying to build a saliency detector. We go in depth about what we tried before, as well as our reflections. In addition, we try to explain our choice of architecture and hyperparameters. We also try to demystify the results and observations.

A. Initial Attempt and Reflections

Our primitive approach revolved around the Authors (Huchuan Lu et. al.) model. The Authors had used a VGG-16 like autoencoder with the recurrent architecture that we previously mentioned. We implemented almost an identical architecture, but without the recurrent feedback. This is shown in Fig. 11.

Since the model was huge (approx. 250 million parameters), we decided to use Transfer Learning. We froze the convolutional layers (did not train them), and trained only the deconvolutional layers. We got an accuracy of about 70 %. The results of the model are shown in Fig. 12.

This sort of white circle in the middle and black everywhere else is what we got for most of our images (see Fig. 12, last column) . Normally, 70 % accuracy is not bad for a classification task. But we have to remember that this is pixel-wise accuracy. Another thing to keep in mind is that the salient regions are normally much smaller than the background regions. This means that our data is skewed towards the background class. This is why our network can get a pretty high accuracy by predicting something trivial.

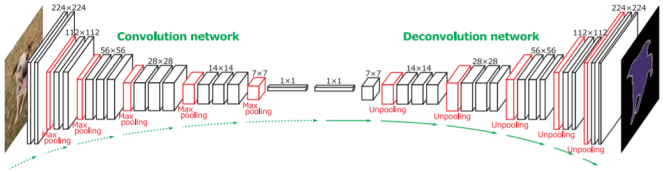

Fig. 11: Architecture of our very first model

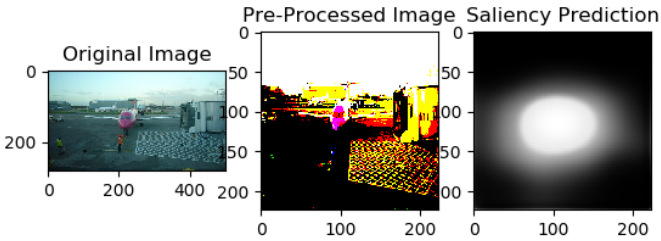

Fig. 12: Results from our first model

Another observation to make is the pre-processed image (see Fig. 12, second column). Since the authors of the VGG network used mean subtraction as the pre-processing step, we were forced to do so, due to Transfer Learning. But notice how the pre-processed image looks very distorted, and the saliency characteristic is completely lost. The original VGG network was used for image classification, which is why this sort of pre-processing might be okay. But for saliency detection, we believe that the input must retain the saliency characteristics, even after pre-processing.

This is why, we decided to make some changes. First, we decided to change the pre-processing, to just normalizing the input image (divide by 255). This way we can ensure that saliency characteristics are retained. This also keeps our input within a small range, and ensures that the weights of our network are comparable. This is considered good practice, because it ensures no weight is dominant while deciding the output. Second, the authors of the paper did not mention things like Batch Normalization and Dropout, which are considered very important while training deep models. We decided to include these in our network. We also chose our other hyperparameters slightly differently. We mention these choices later. Third, we decided to train our model from scratch. Since the original VGG-16 was pre-processed in a different way, Transfer Learning does not make sense. The results that we got by making these changes are shown in Fig. 13 and Fig. 14.

As we can clearly see from the results, they are much better than our first model. So our model does actually learn something. Even the accuracies are much higher this time. Our model is able to locate the object well. But there are definitely some discernible problems. As

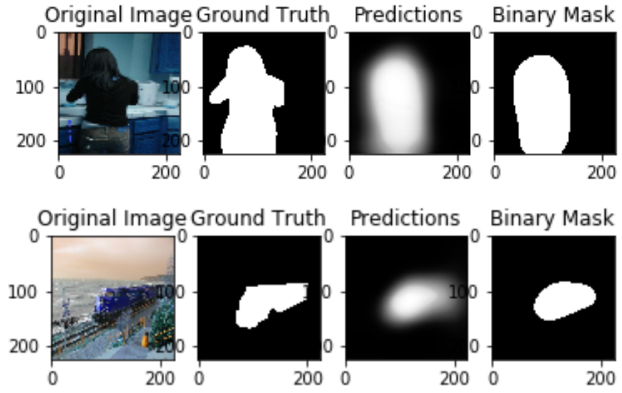

Fig. 13: Results of our VGG-16 autoencoder for two randomly chosen images from the training set

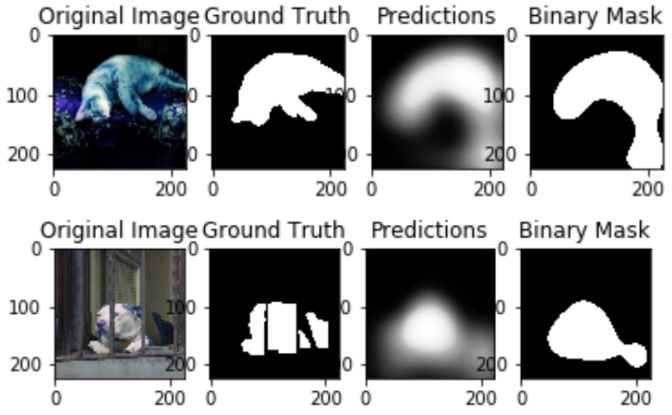

Fig. 14: Results of our VGG-16 autoencoder for two randomly chosen images from the validation set

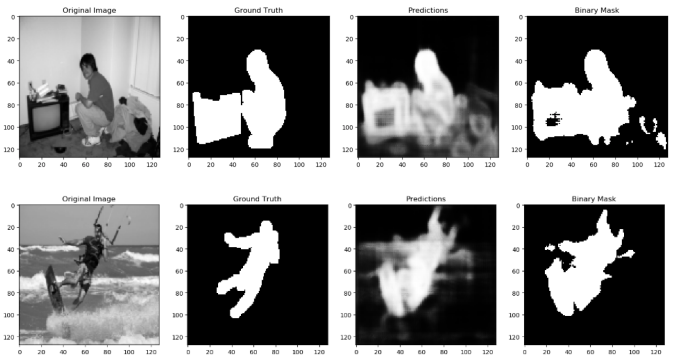

Fig. 15: Results of our mini U-NET for two randomly chosen images from the training set

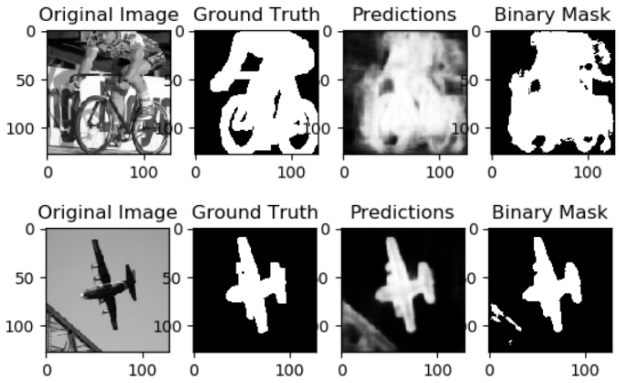

Fig. 16: Results of our mini U-NET for two randomly chosen images from the validation set

we can see, the boundaries are not being preserved. But, the Authors model was able to preserve boundaries pretty accurately (Fig. 1, third column) . We believe that this is because of their Pre-training and fine-tuning approach. In the pre-training stage, the authors trained their model on a semantic segmentation dataset, with 20 classes. This helps their model capture general features of an object. The other advantage is that since they are using segmentation with 20 classes, their models are more strongly supervised. In addition, they had more training examples (10,103 examples vs our 2,000 examples).

This was when we decided to try a different architecture. There were three main reasons for this. First, we did not have access to the semantic segmentation dataset with 20 classes. We could only use our saliency detection dataset with binary masks. Second, we did not have enough data. Our initial model was very large and was thus very prone to overfitting, because of our limited data. We needed to try a smaller model with fewer parameters, to generalize better. Third, we believe that a simple training approach is much more convenient than a complex one. This made us want to achieve results similar to the ones achieved by the author, but with a simple training method.

B. Choice of architecture

In our search for a different architecture, we came across many models that were similar to the VGG network. But we chose the U-NET(Fig. 4) because of two reasons. (i) The U-NET was a model built for the purpose of segmentation, and had achieved good results with the simple training method that we mentioned. (ii) The model was much smaller than the VGG autoencoder (about 19 million parameters). This made it perfect for training on the limited data that we had.

Before we implemented a full-fledged U-NET, we first implemented a mini U-NET, which had about 1.1 million parameters. This is because we did not want to waste our time training a big network, only to find that our change did not help much. Our mini U-NET has basically the same architecture as the U-NET, but with the input as 128x128 grayscale images. Also, the number of channels were reduced to 1/4th of the original. The results are shown in Fig. 15 and Fig. 16.

Even though the accuracy is not very good, we can see that the mini U-NET is able to locate objects pretty accurately. More importantly, it does seem to be doing much better at preserving boundaries of the object. We believe that the lower accuracy is due to underfitting (not enough parameters). Encouraged by these results, we implemented the full U-NET with some minor changes. We will explain these changes next. The results from our

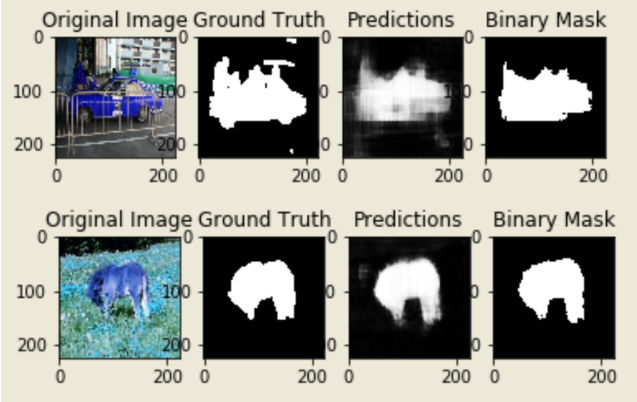

Fig. 17: Results of our UNET for two randomly chosen images from the training set

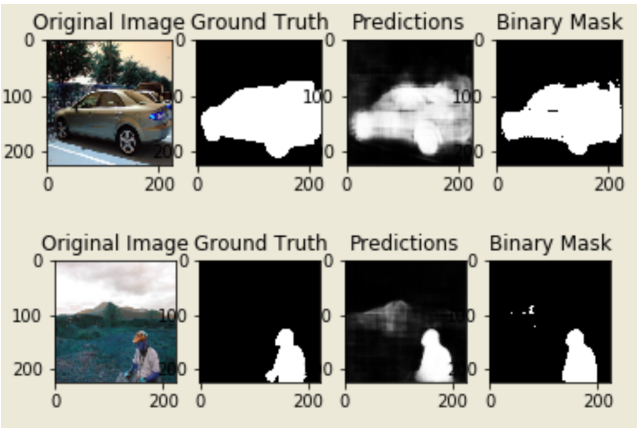

Fig. 18: Results of our UNET for two randomly chosen images from the validation set

trained U-NET model are shown in Fig. 17 and Fig. 18. The training and validation accuracies of our three initial models are mentioned in Table. II.

As we can see, this time our accuracies are higher than the mini U-NET. Also, our results are much closer to the ground truth. This is because the U-NET has more parameters, and thus fits the data better.

We believe there are two reasons for the superior results of U-NET, as compared to the VGG autoencoder. First, the original VGG network was meant for classification. This meant that localization of the object was never taken into consideration. However, our saliency detection task does need to take this into account. The VGG network is very good at classification, because after every max-pooling layer, it learns more complex features of the object. But the price it pays is the loss of spatial information. So after every max-pool layer, it loses more spatial information, and cannot recover this information

TABLE II: Accuracy obtained by three different Model for analysis

Model	VGG-16 auto.	mini U-NET	U-NET
Training Accuracy	89.83%	81.16%	89.7%
Validation Accuracy	81.97%	80.58%	83.44%

while upsampling. In contrast, the U-NET employs skip connections. These skip connections allow it to pass spatial information from the contracting path, to the expanding path. Since there is more spatial information in the contracting path (it is one level higher), this allows the U-NET to localize well during its expanding path, while learning complex features in its contracting path. Second, the VGG autoencoder uses fixed bilinear interpolation in its deconvolutional part. This prevents it from learning the upsampling parameters. In contrast, the U-NET uses transposed convolutional layers while upsampling, which allows it to learn the upsampling parameters, for more accurate upsampling. As mentioned before, we did make some changes to the original U-NET architecture. First, the original U-NET does not use padding in its convolutional layers. This means that the activation maps grow smaller and smaller. Due to this, we would need to crop the activation maps in the skip connections. To prevent this unnecessary hassle, we implemented padding after each convolutional layer. Padding also ensures that the pixels near the boundary are given equal consideration as well. Second, we got rid of the bottleneck layer(the layer with 1x1 activation maps). As we can see from the results of our mini U-NET, our model is able to locate the object well. The bottleneck layer is simply used for better recognition of the object, but there would be a huge toll on its ability to localize. In addition, the bottleneck layer adds the most parameters to the network. Due to these reasons, we did not include it in our architecture.

C. Choice of Hyper-parameters

As mentioned previously, we just used the standard hyper-parameters for training our deep models. These have been proven to work very well by other practitioners. Specifically, we used the 'ReLU' activation after every convolutional layer, except the final one, which was 'sigmoid'. The sigmoid function returns values between 0 and 1. Since our last layer was responsible for the pixel-wise predictions of saliency, sigmoid was necessary.

We also used Batch Normalization after every convolutional layer. This prevents the vanishing gradients

problem, which is quite common in deep networks. To prevent overfitting, after every block of convolutional layers (a block is two convolutional layers in succession), we used Dropout with keep_prob = 0.1. The dropout mechanism ensures that the dependency of the output, on any one of our filters is not too high. (It has a regularization effect).

In addition, we used the Adam optimizer. The Adam optimizer is a variant of the stochastic gradient descent (SGD) algorithm, but leads to smoother training because it takes exponentially weighted averages of the gradients obtained in Gradient Descent. We also implemented callbacks, such as Reduce learning on plateau. This means that if the validation loss did not improve after a 5 continuous epochs, we reduce our learning rate, but we do not go below a threshold.This means that we only reduce the learning rate when it is really needed, and give the optimizer enough time to find a better path towards the minima.

We also implemented early stopping, which stops the training if validation loss does not improve after 10 continuous epochs. We then select the best model as the one with lowest validation loss. Lowest validation loss corresponds to the bests generalization by the network, which is more important than the training loss. Our model had roughly 19 million parameters to train. Since the model is quite difficult to train on a CPU, we trained it on a google colab GPU.

REFERENCES

[1] H. Jiang, J. Wang, Z. Yuan, Y. Wu, N. Zheng, and S. Li, "Salient object detection: A discriminative regional feature integration approach," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, 6 2013, pp. 2083–2090. [Online]. Available: http://ieeexplore.ieee.org/document/6619115/

[2] S. Ramanathan, H. Katti, N. Sebe, M. Kankanhalli, and T. S. Chua, "An eye fixation database for saliency detection in images," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2010.

[3] M.-M. Cheng, N. J. Mitra, X. Huang, P. H. S. Torr, and S.-M. Hu, "Global Contrast based Salient Region Detection," *IEEE TPAMI*, vol. 37, no. 3, pp. 569–582, 2015.

[4] C. Yang, L. Zhang, H. Lu, X. Ruan, and M. H. Yang, "Saliency detection via graph-based manifold ranking," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2013.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Communications of the ACM*, 2017.

[6] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2014.

[7] L. Wang, W. Ouyang, X. Wang, and H. Lu, "Visual tracking with fully convolutional networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015.

[8] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2015.

[9] L. Wang, L. Wang, H. Lu, P. Zhang, and X. Ruan, "Salient Object Detection with Recurrent Fully Convolutional Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 7, pp. 1734–1746, 7 2019.

[10] L. Itti, E. Niebur, and C. Koch, "A model of saliency-based visual attention for rapid scene analysis," 1998.

[11] M. Everingham, L. Van˜Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The {PASCAL} {V}isual {O}bject {C}lasses {C}hallenge 2010 {(VOC2010)} {R}esults," http://www.pascal-network.org/challenges/VOC/voc2010/workshop/index.html.