Evaluating Pyramid Feature Attention Networks for Saliency detection

Rajdeep Singh Lather
George Mason University
rlather@gmu.edu

Cody Kidwell
George Mason University
ckidwel@gmu.edu

Ben Lister
George Mason University
blister@gmu.edu

Abstract - Saliency detection is a traditionally important problem in computer vision. Determining the most important part of an image is an issue that crops up in multiple fields of computer vision. In this paper, we compare a deep learning-based saliency detection technique to various traditional ones from OpenCV. The deep learning-based technique uses a Pyramid Feature Attention Network to focus on effective high-level context features and low-level spatial structural features. We use four different metrics to evaluate these methods for static images and one additional metric for video. Finally, based on the performance of various techniques we make suggestions for which method to use for specific applications.

1. Introduction

Saliency detection is about finding the most "salient" or important parts of an image. It's another area in vision where humans excel, as finding out where to pay attention is one of the most important skills to have in the real world. Thus, saliency detection is often done during the preprocessing of more complex applications.

A lot of different kinds of saliency detection methods exist, however, different methods excel at different tasks. Moreover, in recent years convolutional neural networks have also made progress for image detection, paving the way for more accurate results.

In this paper, we quantify the performance of various saliency methods, including traditional algorithms from OpenCV such as Static Spectral Saliency, Static Fine-grained Saliency, and Motion Saliency. Finally, we compare these algorithms to a more black-box approach, by training a Pyramid Feature Attention Network, to detect saliency in multiple images and video.

2. Related Work

Borji, Cheng, Hou, et al.[1] review different salient object detection models. They identify two types of analysis: block-based and region-based, and two indicators for saliency: intrinsic and extrinsic.

In block-based methods, an image is segmented into a grid, and comparisons between features of the rectangular uniformly-sized segments are used to determine the saliency of the segment. This approach suffers from many shortcomings, such as being vulnerable to high-contrast edges, failing to capture smooth boundaries due to the size of the grid, and poor speed due to the number of segments.

In region-based methods, images are segmented using mean-shift or other methods into regions of

varying sizes, and comparisons between features of a segment to the features of all other segments is used to establish saliency. The features that can be extracted from regions are generally more informative, and often make use of priors such as objectness or backgroundness.

Intrinsic methods use information gained from the image to generate a saliency map. Extrinsic methods use global information above the scope of the individual image, such as comparisons to similar images with known ground truths, or comparisons to nearby frames in a video.

3. Dataset

We are using the DUTS dataset, which contains 10,553 input image-ground truth training pairs, and 5019 input image-ground truth test pairs[2]. Accurate pixel-level ground truths for these images were manually annotated by 50 subjects. Each test image is a black and white image with the salient parts being white. All the training images are collected from the ImageNet DET training/validation sets. The test images are collected from the ImageNet DET test set and the SUN data set.

Because images in DUTS are of inconsistent size, when training the Pyramid Neural Network we are resizing the images to be 256 pixels long in their shortest edge. Then we take a center crop and make the longest length 256 as well. After that, we normalize the image. Then to generalize better and avoid overfitting on the training data we use random horizontal flips and randomized crops of the images as a final transformation.

4. Pyramid Feature Attention Network for Saliency detection

For the deep learning approach for saliency detection, we looked into a method proposed by Ting Zhao, et. al. in their paper called Pyramid Feature Attention Network for Saliency detection[3]. They use this method to "focus on effective high-level context features and low-level spatial structural features". They do this using a Context-aware Pyramid Feature Extraction (CPFE) module for high-level feature maps. Next, they use channel-wise attention (CA) after CPFE feature maps and spatial attention (SA) after low-level feature maps, combining their outputs together.

4.1 Architecture

The pyramid feature attention network extracts features obtained from VGG-Net in two paths corresponding to low-level and high-level features, before combining them to eventually produce the output.

Low-level spacial features are obtained from the outputs of vgg1-2 and vgg2-2 in a 256x256x192-dimensional feature map. They are convolved to a 256x256x64-dimensional feature map, and are passed through an attention module that captures salient object/background boundaries. The output of vgg3-3, vgg4-3, and vgg5-3 are passed through a context-aware pyramid feature extraction module similar to SIFT[10] to generate the high-level features as a 64x64x384-dimensional feature map, then a channel-wise attention mechanism extracts the features most important to saliency (as determined by weights obtained through training). These features are fed through a 1x1 convolutional layer to reduce the number of features to 64, then upsampled to produce a 256x256x64 map.

The outputs of the low-level path and the high-level path are combined and then convolved by a 3x3x1 convolutional layer to produce a 256x256x1 gray-level image as the output saliency prediction mask.

4.2 Loss Function

The Pyramid method uses a total loss L as

$$L = \alpha L_S + (1 - \alpha) L_B$$

L_S in the above equation is cross entropy loss defined by

$$L_S = - \sum_{i=0}^{size(Y)} (\alpha_s Y_i log(P_i) + (1 - \alpha_s)(1 - Y_i) log(1 - P_i))$$

Where Y refers to the ground truth and P refers to the saliency map of network output, α_s is a balance parameter of positive and negative samples that was calculated from ground truth of the training set as $\alpha_s = 0.528$.

For L_B, first, Laplace Operator is used to get the boundaries of ground truth and saliency map of network output

$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Next, an absolute operation and then a tanh activation is used

$$\Delta \widetilde{f} = abs(tanh(conv(f, K_{laplace})))$$

Finally, cross-entropy loss is used to supervise the generation of salient object boundaries

$$L_B = - \sum_{i=0}^{size(Y)} (\Delta Y_i log(\Delta P_i) + (1 - \Delta Y_i) log(1 - \Delta P_i))$$

This loss function emphasizes correctly predicting the edges of salient objects, which is why the pyramid feature attention method produces sharp image outputs that look very similar to the binary ground truth images.

4.3 Training the model

We used an existing implementation of the Pyramid attention network that was done using PyTorch by Sai Raj Kishore[6]. Initially, we trained the model on DUTS-TR for 16 epochs with batch size 6. Using this we were able to achieve a test-set mean absolute error (MAE) of 0.102 and 0.1562 average edge saliency loss. Ultimately, we trained the model for a total of 42 epochs, achieving a test-set MAE of 0.0264 and 0.0469 average edge saliency loss.

The final model was trained on a Windows 10 machine with a NVIDIA GeForce RTX 2060 GPU, 2.6 GHz Intel Core i7-9750H CPU, and 16 GB RAM.

5. Metrics and Performance Evaluation

We tried a few different metrics to analyze different aspects of various saliency detection methods. Here, we used a python implementation of these metrics by Tarun Sharma[5]. Each of the saliency detection methods we used returns a grayscale image with higher pixel values for the most salient parts of that image; such that, white would be the most salient, black the least, and various gray levels in between. We divided our experiments into two broad groups, one for testing performance on individual images and another for testing performance in video.

For the individual image experiments we chose four metrics; namely, Area Under ROC Curve (AUC), Pearson's Correlation Coefficient, Normalized Scanpath Saliency, and Similarity (SIM). In order to evaluate the pyramid feature attention network we compared it with two OpenCV implementations of static saliency detection - CV2 Static Spectral and Fine-grained.

For testing on video, we compared how fast these methods were in order to investigate which

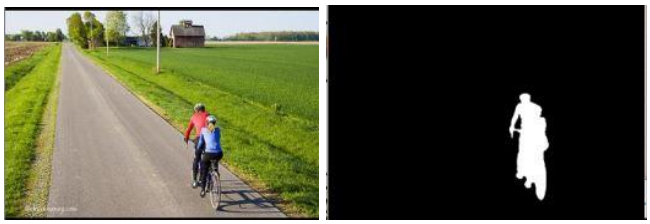
ones could be used in realtime to do saliency detection on a video stream. This we measured using Frames Per Second (FPS) of these methods. The methods we evaluated for video were CV2 Static Spectral and Fine-grained, CV2 motion, and the pyramid feature attention network.

5.1 Area under ROC Curve (AUC)

The area under the ROC curve is the integral of true positive rate with respect to false positive rate[11]. It can be considered a measure of how well a model is able to discriminate between classes. In saliency detection, the gray-level images that the models output are compared to the binary ground truth images. We say a pixel in the output map is salient (positive) if it is at least some threshold; in actuality, it is positive if it has value 255 in the ground truth. The number of pixels predicted positive that are actually positive, over the number of actual positive pixels, is the true positive rate for the output at some threshold level. Similarly, the number of pixels predicted positive that are actually negative, over the number of actual negative pixels, is the false positive rate. This false positive rate - true positive rate pair is a point in the ROC curve, which is formed by computing a point for every threshold value present in the output image. The area under this curve ranges from 0 to 1 and is an indicator of how well the model was able to classify pixels in the image. This quantity was then averaged over 5019 test images in DUTS-TE to produce the following results:

Model	**Avg. Area under ROC**
Pyramid Attention	0.963904
CV2 static spectral	0.827182
CV2 static fine-grained	0.542373

TABLE 1: Average AUC over 5019 test images



Input (left), ground truth (right)

Cv2 static spectral (left), Pyramid attention (middle), cv2 static fine-grained(right)

It is curious that fine-grained saliency performed so poorly when it is the sharpest of the three outputs. This is likely because it operates somewhat similar to edge detection - contiguous regions within an image, such as the biker's coats, are not considered interesting, but the edges of the coat against the road/grass are. However, as we are interested in detecting entire objects in an image, not parts of entire objects, it is reasonable to say that fine-grained performance is worse.

$$\mathrm{tpr_u} = \frac{1}{\mathrm{N_{y_a=255}}} \sum_{\mathrm{a:\ y_a=255}} 1 \text{ if } \widehat{y_a} \geq \mathrm{u} \text{ else } 0$$

$$\mathrm{fpr_u} = \frac{1}{\mathrm{N_{y_b=0}}} \sum_{\mathrm{b:\ y_b=0}} 1 \text{ if } \widehat{y_b} \geq \mathrm{u} \text{ else } 0$$

$$\mathrm{AUC} = \sum_{\mathrm{u} \in \hat{y}} \frac{(tpr_{\mathrm{u+1}} + tpr_{\mathrm{u}}) * (fpr_{\mathrm{u+1}} - fpr_{\mathrm{u}})}{2}$$

AUC using trapezoidal integral approximation

5.2 Pearson's Correlation Coefficient (CC)

Pearson's Correlation Coefficient is a distribution-based metric, it is used to evaluate linear correlation between the prediction (y) and ground truth (ŷ) distributions[7].

$$\frac{\mathrm{cov}(y, \hat{y})}{\sigma_y \sigma_{\hat{y}}}$$

This returns high values in regions where both the ground truth and the saliency map have values of similar magnitudes. Treats false positives and false negatives symmetrically[4].

Model	**Pearson's Correlation Coefficient**
Pyramid Attention	0.667293
CV2 static spectral	0.354822
CV2 static fine-grained	0.097225

TABLE 2: Average CC over 5019 test images

5.3 Normalized Scanpath Saliency (NSS)

NSS is a location-based metric that measure's correspondence between saliency maps and ground truth[8]. It's computed as the average normalized saliency at fixated locations[8].

$$\frac{1}{N_{y_a=255}} \sum_a \widetilde{\hat{y}_a} * y_a$$

Where the prediction map $\hat{y}$ is normalized. NSS is a discrete approximation of CC. Also, since the mean saliency value is subtracted during computation, NSS is invariant to linear transformations.

Model	**Normalized Scanpath Saliency**
Pyramid Attention	2.750861
CV2 static spectral	1.543274
CV2 static fine-grained	0.548191

TABLE 3: Average NSS over 5019 test images

5.4 Similarity (SIM)

Similarity is a distribution-based metric that measures the intersection between histograms of two distributions[9]. SIM is more sensitive to false negatives than false positives[4]. To compute SIM we take the sum of the minimum values at each pixel, after normalizing the input maps[4]. Given a saliency map P and a continuous fixation map Q^D:

$$SIM(P, Q^D) = \sum_i \min(P_i, Q_i^D)$$

$$\text{where } \sum_i P_i = \sum_i Q_i^D = 1$$

Model	**Similarity**
Pyramid Attention	0.438639
CV2 static spectral	0.209816
CV2 static fine-grained	0.109595

TABLE 4: Average SIM over 5019 test images

5.5 Frames Per Second (FPS)

The above-mentioned metrics allowed us to test the accuracy of different saliency methods. To test the performance of our models on video, we used the FPS metric. To calculate this we tested various methods on a video file that was 902 frames long and recorded the total time to run. Dividing by the number of frames gives us the FPS. These tests were

conducted on a Windows 10 machine with a NVIDIA GeForce RTX 2060 GPU, 2.6 GHz Intel Core i7-9750H CPU, and 16 GB RAM.

Model	**Time (in seconds)**	**FPS**
Pyramid Attention	23.58	38.25
CV2 static spectral	5.33	169.23
CV2 static fine-grained	44.05	20.47
CV2 Motion	22.96	39.28

TABLE 5: Average FPS over 902 video frames

6. Conclusion

We evaluated using Pyramid Feature Attention Network for Saliency Detection, comparing it with more traditional implementation from OpenCV. To this end, we used five different metrics to analyze different aspects of saliency detection.

In our tests, the Pyramid attention network achieved the best results overall and was the undisputed winner for all methods involving static images. Nevertheless, this deep learning-based approach was able to provide an FPS over 30 making it suitable for "realtime" video applications.

OpenCV's Static Spectral method was almost as good as the deep learning approach while also providing over four times the FPS at 169.23. This makes it a suitable approach where the performance over video is key, such as applications with high FPS requirements or large video files.

OpenCV's Static Fine-grained method seemed to perform the worst, but that appears to be because of what we were actually testing. The fine-grained method focuses on the edges and minute details of an image. While our test data set had manually annotated images of only a handful of objects (ignoring the minutia). It may perform better in cases where minute details are important.

All three members of our group collaborated on the project and we often coded together as well via screen share. That being said, we did split some tasks up amongst ourselves. For instance, Ben did the initial explorations into finding an implementation of the pyramid feature attention network, training the model, and modifying the code to incorporate OpenCV saliency methods and accommodate our evaluation metrics (see evaluate.py and EvalDataLoader class). He also implemented the AUC metric. Ben was the primary contributor to the 'Related Works', 'Area under ROC Curve', and 'Architecture' sections of the paper. Cody on the other hand also trained the neural network for pyramid feature attention network and helped configure the saliency functions with Ben. He also found and set up the metrics for the various methods, including Pearson's Correlation Coefficient, Normalized Scanpath Saliency, and SIM. In addition, he set up FPS for the videos, converted the pyramid network to handle a video and wrote the 'Introduction' for the paper. Finally, Rajdeep did the initial research into various metrics to compare saliency detection methods and took the initiative on writing the report and the slide deck. He wrote the sections 'Abstract', 'Pyramid Feature Attention Network for Saliency detection', 'Loss Function', Metrics and Performance Evaluation (CC, NSS, SIM, and FPS), and the 'Conclusion'.

6. 1 Rajdeep's Comments

We first started the project thinking that we would be using saliency detection for doing dynamic video compression (compressing only the non-salient parts). This seemed like an intriguing project initially but, after talking with our Professor Dr. Duric we decided to change the project and instead looked into novel approaches of doing

salience detection and comparing them with more traditional ones. I'm quite happy with the results we got as they give a clear indication towards which saliency metric to use. If we had more time, I would have wanted to include more techniques for saliency detection and more metrics for video. I would have liked to test these methods on other datasets as well.

6. 2 Cody's Comments

When we started this project, I was interested to see the various methods that have been implemented to find important features in an image. I expected that a neural network would be the best solution for saliency. As expected this was correct, although I was surprised by how well the other algorithms on OpenCV worked. When comparing the methods I was uncertain what metrics we would use. Although when finding the various methods in the paper, this helped to explain what methods to use and how the methods worked. If we could do anything different, I would like to explore more methods focused on motion.

6. 3 Ben's Comments

As mentioned above, the initial goal for this project was to incorporate saliency methods in video compression, something I am personally interested in and have worked on in the past. When we decided to instead compare different saliency methods, I initially felt unsure on how to proceed. These feelings lessened once I started writing the code, and I am satisfied with what we ended with. If I could do anything differently, I would have liked to understand the mechanics of fine-grained and spectral saliency more to perhaps motivate a more insightful comparison. For the most part, however, this project has inspired me to explore applications of saliency detection, not only in video compression but also as a preprocessing stage in an image classification pipeline.

7. References

[1] Ali Borji, Ming-Ming Cheng, Qibin Hou, Huaizu Jiang, Jia Li, "Salient object detection: A survey", *Computational Visual Media*, 2019.

[2] Wang, Lijun & Lu, Huchuan & Wang, Yifan & Feng, Mengyang & Wang, Dong & Yin, Baocai & Ruan, Xiang. (2017). Learning to Detect Salient Objects with Image-Level Supervision. 3796-3805. 10.1109/CVPR.2017.404.

[3] Zhao, T., & Wu, X. (2019). Pyramid Feature Attention Network for Saliency Detection. 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 3080-3089.

[4] Bylinskii, Zoya & Judd, Tilke & Oliva, Aude & Torralba, Antonio & Durand, Frédo. (2016). What Do Different Evaluation Metrics Tell Us About Saliency Models?. IEEE Transactions on Pattern Analysis and Machine Intelligence. PP. 10.1109/TPAMI.2018.2815601

[5] Tarun Sharma, saliency_metrics, https://github.com/tarunsharma1/saliency_metrics

[6] Sai Raj Kishore, PyTorch-Pyramid-Feature-Attention-Network-for-Saliency-Detection, https://github.com/sairajk/PyTorch-Pyramid-Feature-Attention-Network-for-Saliency-Detection

[7] O. Le Meur, P. Le Callet, and D. Barba. Predicting visual fixations on video based on low-level visual features. Vision Research, 47(19):2483–2498, 2007

[8] R. J. Peters, A. Iyer, L. Itti, and C. Koch. Components of bottom-up gaze allocation in natural images. Vision Research,45(18):2397 – 2416, 2005

[9] A. Santella, M. Agrawala, D. DeCarlo, D. Salesin, and M. Co-hen. Gaze-based interaction for

semi-automatic photo crop-ping. InSIGCHI, pages 771–780. ACM, 2006.

[10] D. G. Lowe. Distinctive image features from scale invariant keypoints. International journal of computer vision, 60(2):91–110, 2004.

[11] T. Fawcett. An introduction to ROC analysis. Pattern recognition letters, 27(8):861–874, 2006