# COMPUTER VISION PROJECT-2019

# Oriented Edge Forests for Boundary Detection

Ashish R (2018702004) and Deepak Kumar Singh (2018701010)

April 3, 2019

# Contents

# 1 Abstract

The paper aims at presenting an efficient model for learning boundary detection based on a random forest classifier. This approach combines:

- Efficient clustering of training examples based on a simple partitioning of the space of local edge orientations and
- Scale-dependent calibration of individual tree output probabilities prior to multiscale combination.

# 2 Introduction

Accurate boundary estimation is an important first step for segmentation and detection of objects in a scene and boundaries provide useful information about the shape and identity of those objects. Simple brightness or color gradients are insufficient for handling many natural scenes where local gradients are dominated by fine scale clutter and texture arising from surface roughness and varying albedo(reflective power).

We are using randomized decision forests to the simple task of accurately detecting straight-line boundaries at different candidate orientations and positions within a small image patch. Although this ignores a large number of interesting possibilities such as curved edges and junctions, it should certainly suffice for most small patches of images containing big, smooth objects.

# 3 Clustering Edges

## 3.1 Concept

From a ground truth boundary image, we categorize a $p$ x $p$ patch either as:

- Background (No boundary)
- Belonging to one of a fixed number of edge categories.

A patch is considered background if its center is more than *p/2* pixels away from an edge, in which case the patch contains little to no edge pixels.

Non-background patches are distinguished according to the distance d and orientation $\theta$ of the edge pixel closest to the patch center. Thus, patches with $d = 0$ have an edge running through the center, and by definition d is never greater than $p/2$. We choose a canonical orientation for each edge so that $\theta$ lies in the interval $(-\pi/2, \pi/2]$. To distinguish between patches on different sides of an edge with the same orientation, we utilized signed distances $d \in (-p/2, p/2)$. This yields a parameter pair $(d, \theta)$ for each non-background patch.

From a ground-truth edge map, computing the distance between a patch center and the nearest edge pixel $q$ is straightforward. To be useful, the estimate of $\theta$ should reflect the dominant edge direction around $q$, and be robust to small directional changes at $q$. To accomplish this, we first link all edge pixels in a ground-truth boundary map into edge lists, breaking lists into sub-lists where junctions occur. We then measure the angle at q by fitting a polynomial to the points around q that are in the same list. In our experiments we use a fitting window of $\pm 6$ pixels..

### 3.2 Collecting training data

We binned the space of distances $d$ and angles $\theta$ into $n$ and $m$ bins, respectively. Thus every non-background patch was assigned to a discrete label $k$ out of $K = nm$ possible labels. This discrete label space allows for easy application of a variety of off-the-shelf supervised learning algorithms.

In our experiments we used a patch size of $16 \times 16$, so that distances satisfy $|d| < p/2 = 8$. It is natural to set the distance bins one pixel apart, so that $d$ falls into one of $n = 15$ bins. Assigning angles $\theta$ to one of $m = 8$ bins, leaves $K = 120$ edge classes plus background. We chose the orientation binning so that bins 1 and 5 are centered at 90 and 0 degrees respectively, as these orientations are especially common in natural images.

Since our approach ultimately predicts a $(d, \theta)$ parameter for each non-background image patch, it does not explicitly model patches containing junctions or thin structures involving more than two segments.

## 4 Oriented Edge Forests

We build a training dataset comprised of color image patches $\mathbf{x}$, each with a corresponding edge cluster assignment $\mathbf{y} \in \{0, 1, \ldots, K\}$ where K is the number of edge clusters and $\mathbf{y} = 0$ represents the background or "no boundary" class.

### 4.1 Randomized Decision Forests

Random forests are a popular ensemble method in which randomized decision trees are combined to produce a strong classifier. Trees are made random through bagging and/or randomized node optimization, in which the binary splits at the nodes of the tree are limited to using only a random subset of features.

When training a given decision tree, features are selected and split thresholds are chosen to optimize the Gini impurity measure. the particular choice of class purity metric does not have a noticeable impact on performance. We did find it important to have balanced training data across classes and used an equal number training examples per class.

## 4.2 Image Features

images are transformed into a set of feature channels, and the descriptor for a patch is computed simply by cropping from the corresponding window in the array of feature channels. These features are comprised of color and gradient channels, and are down-sampled by a factor of 2. Binary splits performed at the tree nodes are accomplished by thresholding either a pixel read from a channel or the difference between two pixels from the same channel.

## 4.3 Ensemble Methods

The details of how we fuse the predictions of different trees can have a significant effect on performance. Two standard approaches to combining the output of a ensemble of classifiers are **averaging** and **voting**.

For a given test image patch $x$, each individual tree $t$ produces an estimate $p_t(k|\mathbf{x})$ of the posterior distribution over the $K+1$ class labels based on the empirical distribution observed during training. We would like to combine these individual estimates into a final predicted score vector $\mathbf{w}(k|\mathbf{x})$.

### 4.3.1 Averaging

$$\mathbf{w}(k|\mathbf{x}) = \frac{1}{T}\sum_{t=1}^{T} p_t(k|\mathbf{x}), \quad k = 1, \ldots, K$$

### 4.3.2 Voting

$$\mathbf{w}(k|\mathbf{x}) = \frac{1}{T}\sum_{t=1}^{T} \mathbf{1}_{[k=\arg\max_k p_t(k|\mathbf{x})]}$$

where $\mathbf{1}$ is the indicator function.

Averaging provides somewhat better detection accuracy than voting, presumably because the votes carry less information than the full posterior distribution. One disadvantage of averaging is that it requires one to maintain in memory all of the empirical distributions p at every leaf of every tree. Voting not only requires less storage for the forest but also reduces runtime. Voting may thus be an efficient alternative for time-critical applications.

# 5 Edge Fusion

Having applied the forest over the input image, we are left with a collection of calibrated probability estimates $\hat{\mathrm{w}}$ at every spatial position. Because these distributions

express the likelihood of both centered ($d = \theta$) as well as distant, off-center ($d \neq 0$) edges, the probability of boundary at a given location is necessarily determined by the tree predic- tions over an entire neighborhood around that location. In this section, we describe how to resolve these probabilities into a single, coherent image of boundary strengths. The end result will be an oriented signal $E(x, y, \theta)$ that specifies the probability of boundary at location (x, y) in the binned direction $\theta$.

## 5.1 Edge Sharpening

Consider a hypothesized (straight) edge predicted by the forest at a given location. We compute the mean RGB color of the pixels on each side of the hypothesized edge inside a 16 $\times$ 16 pixel patch centered at the location. We then re- segment pixels inside the patch by assigning them to one of these two cluster means. To prevent the local segmen- tation from differing wildly with the original oriented line predicted by the forest, we only reassign pixels within 1 or 2 pixels distance from the hypothesized segment boundary

## 5.2 Edge Sharpening

Compositing Given local estimates of the likelihood (cal- ibrated scores ŵ) and precise boundary shapes (sharpened masks M ) for each image patch, we predict whether a loca- tion (x, y) is on a boundary by averaging over patch predic- tions for all patches that include the given location. Using the convention that M (x,y,k) (0, 0) is the center of a given edge mask and indexing ŵ by the coordinates of each patch in the image, we can write this formally as

$$E(x, y, \theta) = \sum_{k \in \{(d,\theta) \forall d\}} \sum_{(i,j) \in O_{xy}} \hat{\mathbf{w}}(i, j, k) M_{(i,j,k)}(x - i, y - j)$$

where $O_{xy}$ are the coordinates of patches overlapping x, y and k ranges over all predicted labels which are compatible with orientation $\theta$

# 6 Work Flow

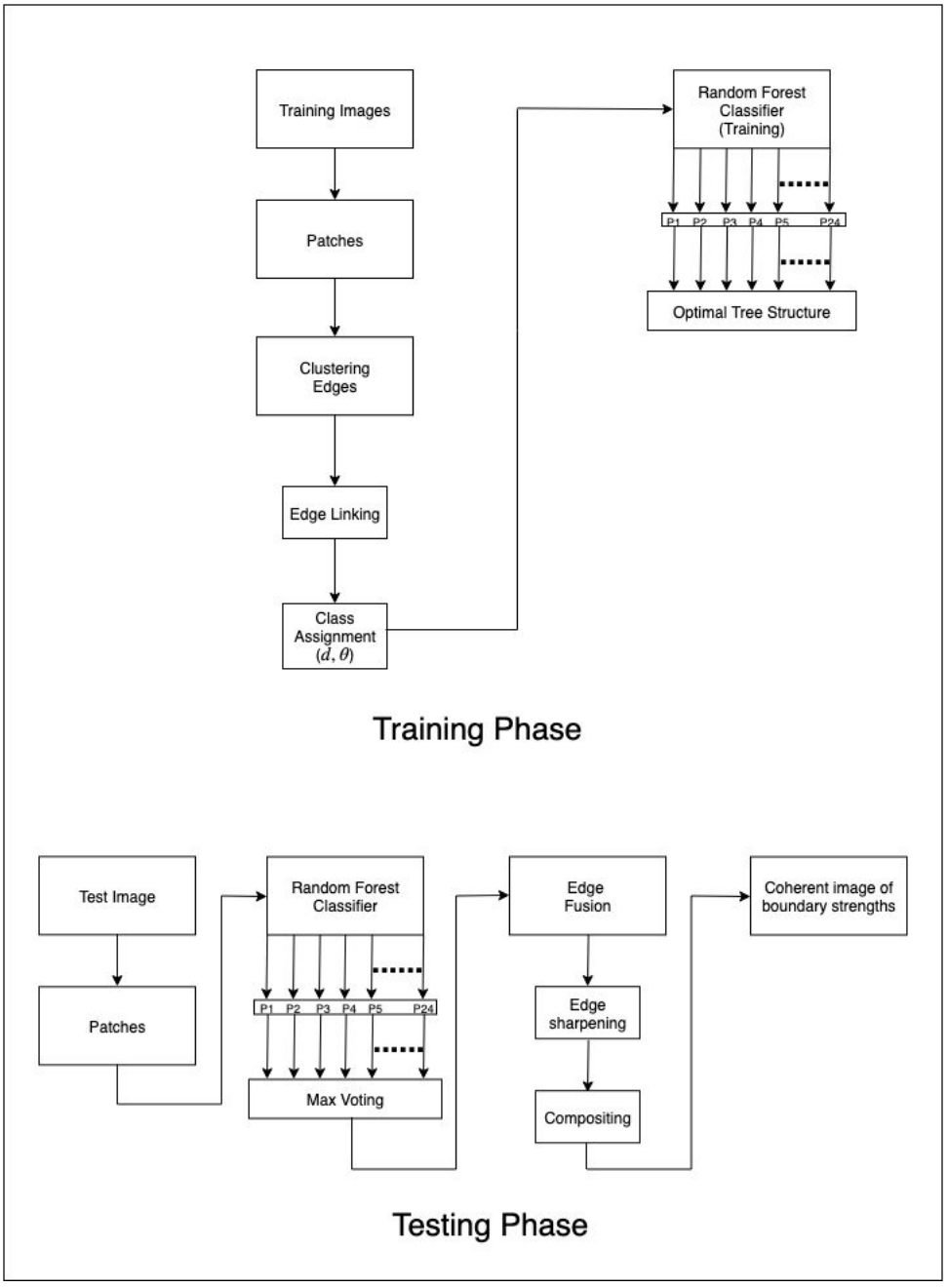


Figure 1: Overall Block Diagram

# 7 Implementation Details

## 7.1 Assigning Labels to patches

- Created the dataset from BSDS500 Segmentation dataset.

- Implemented primary functionality of computing the parameters.
- Created the hierarchy structure for labels associated with the parameter space in the dataset.
- Computed parameters for the given patch of image.
- Appropriate sign-conventions were adopted for the nearest edge pixel from the center of the patch.

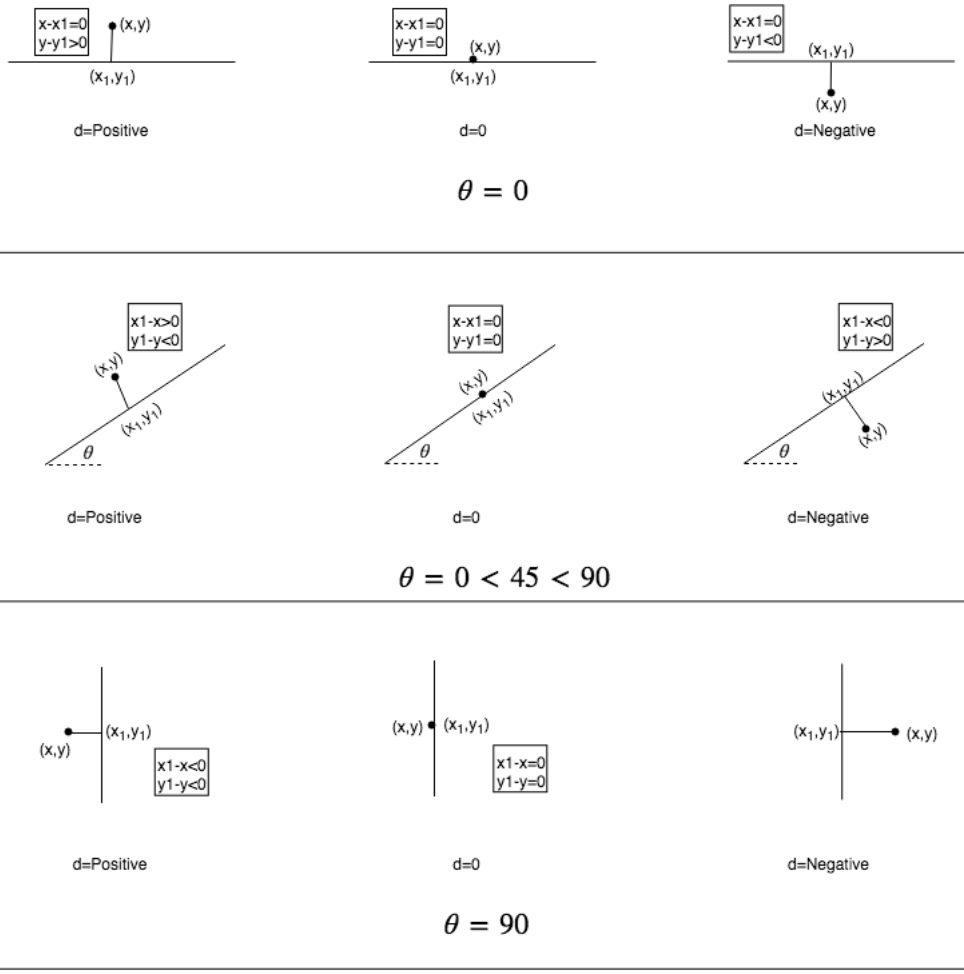


Figure 2: Sign Conventions followed

The conventions as shown above depict the various scenarios of a patch centre with respect to an edge which maybe 2 or more pixels in length.Here we assumed the edges encountered in the patches to be at least 2 pixels in length and using a basic

analytical geometry approach we deduced a relation for the distance 'd' between a centre pixel and the nearest edge pixel to this centre. We also define the orientation of the line using the following relations.

$$d = \sqrt{(x_1 - x)^2 + (y_1 - y)^2}$$

$$\tan\theta = \frac{y_1 - y}{x_1 - x}$$

The Resultant d and $\theta$ found are binned as shown below

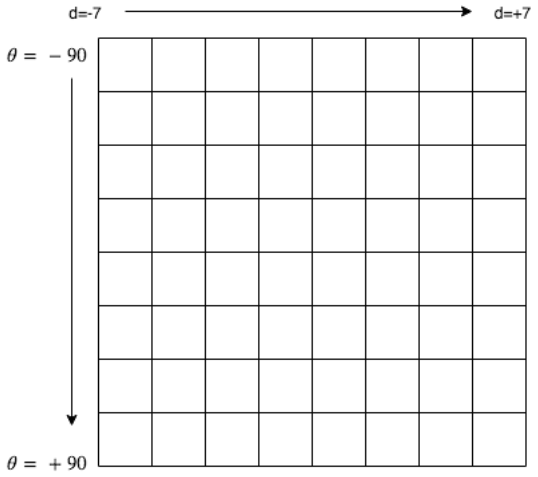


Figure 3: d-$\theta$ space

Every patch of the image is thus mapped to one of the bins above and these bins are used as classes for the patch to train a **Random Forest Classifier** which will be implemented in the next time line.

For efficiency and storage concerns the image patches are stored as ***.mat*** files rather than ***.jpg*** files.

## 7.2 Further Refining

The above method, also considers isolated 1's and if it is closer to the patch centre, it assigns that particular patch as a **background**, even though it consists of edge of arbitrary orientation in the same patch. To avoid such possibilities of isolated 1's, we performed a thinning operation, by doing so we also took care of thick edges around the centre of the patch. Doing so we were able to reduce a large number of background patches There by reducing the dataset size. We also used the **polyfit** function of **numpy** to take care of arbitrary curves that may exist in the edge.We measured the angle at the nearest edge pixel q to the patch centre by fitting a polynomial to the points around q that are in the same list. In our experiments we use a fitting window of $\pm 6$.

## 7.3 Specifics of Implementation

| | |
|---|---|
| Time taken to generate the Edge label for each patches | 120 minutes |
| Time taken to generate the Edge label for each patches | 288 minutes |
| Time taken for output | 7seconds |
| Number of Trees chosen for training | 24 |
| Number of Patches generated | 6752098 |

Table 1: Specifics of Implementation

## 7.4 GUI Component

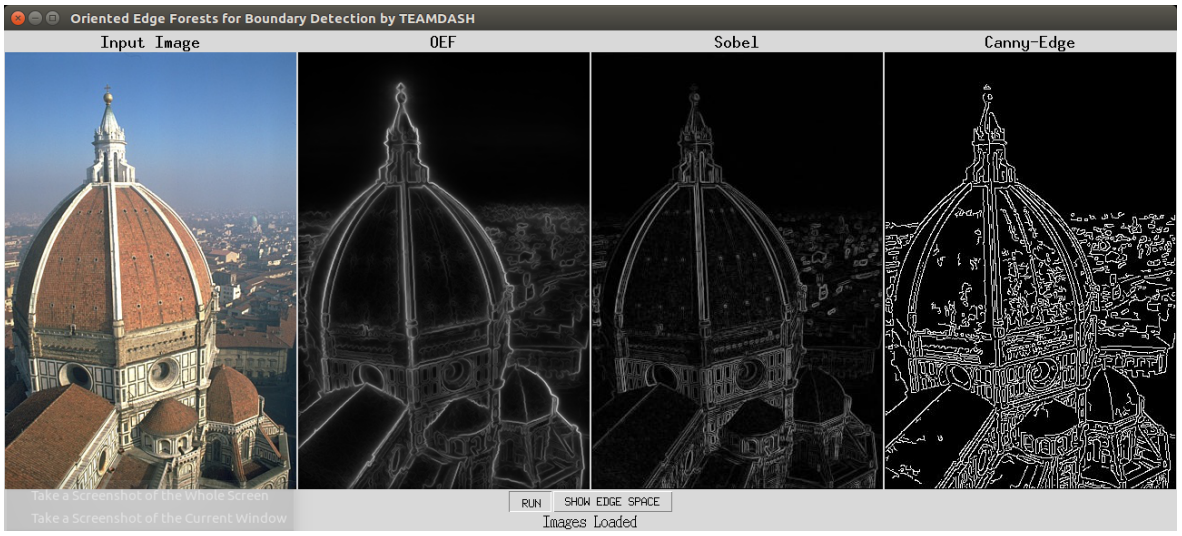


Figure 4: GUI

# 8 Output Images

## 8.1 Image set 1

Figure 5: Input Image

Figure 6: Output Image

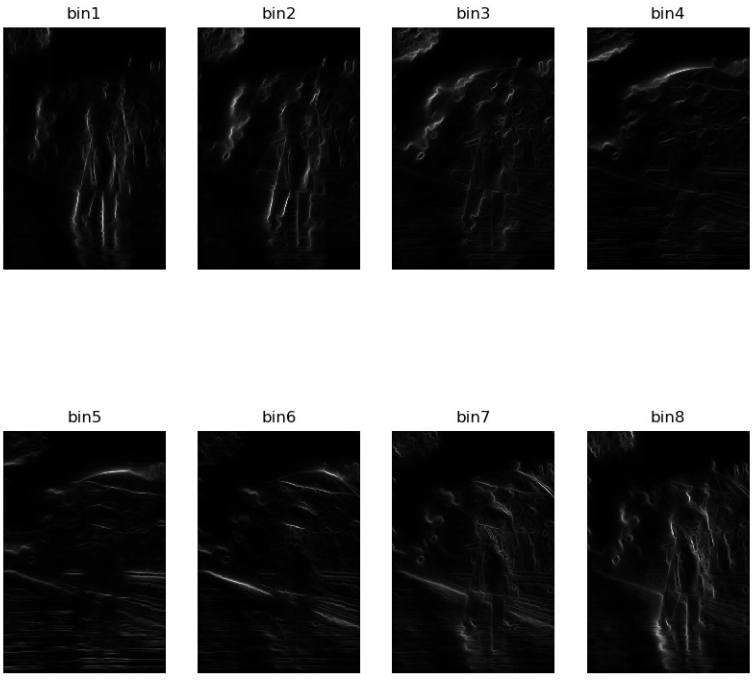


Figure 7: Edge Map orientations

## 8.2 Own Image

Figure 8: Input Image

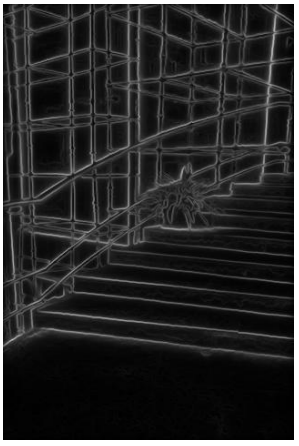

Figure 9: Output Image

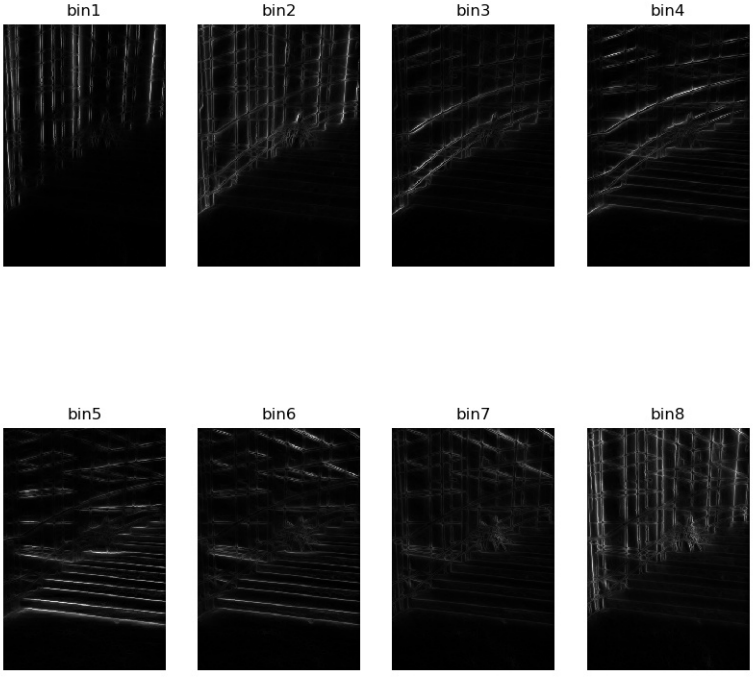


Figure 10: Edge Orientations

## 8.3 Image set 2

Figure 11: Input Image

Figure 12: Output Image

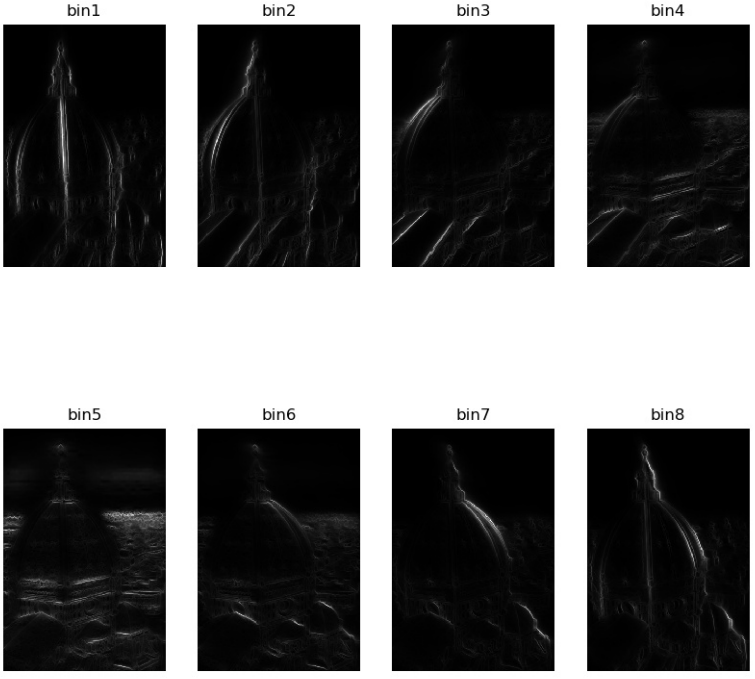


Figure 13: Edge Orientations

# 9 Comparing with Sobel and Canny Edge

## 9.1 Image set 1


Figure 14: Input Image

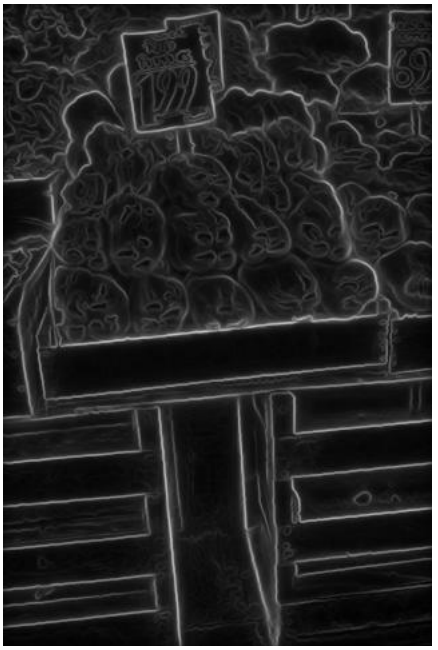

Figure 15: OEF output

Figure 16: Sobel Output

Figure 17: Canny Edge Output

## 9.2 Own Image

Figure 18: Input Image

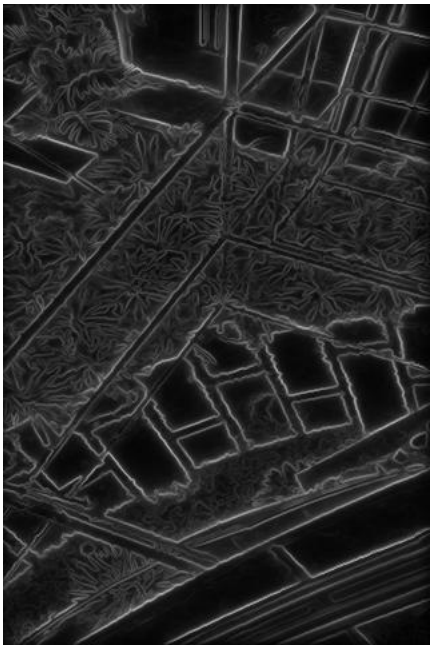

Figure 19: OEF output

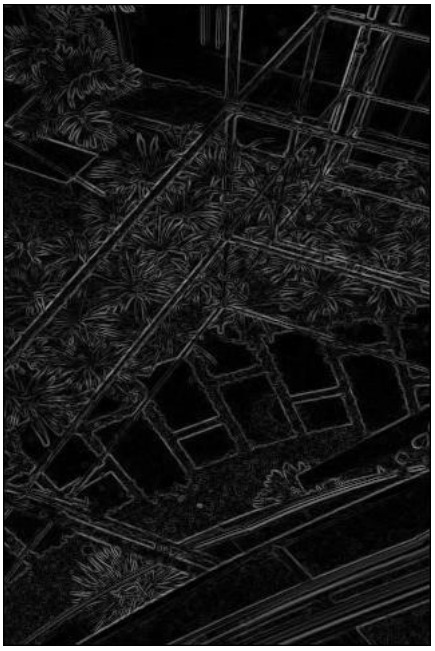

Figure 20: Sobel Output

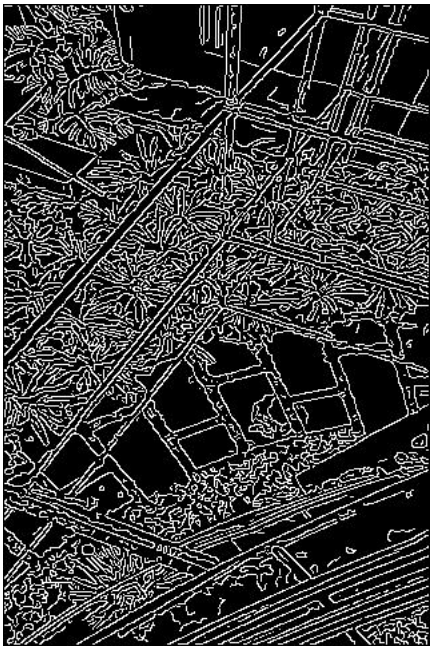

Figure 21: Canny Edge Output

## 9.3 Own Image

Figure 22: Input Image

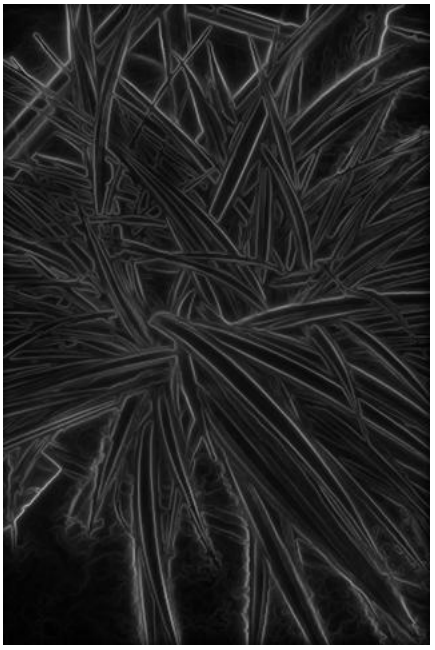

Figure 23: OEF output

Figure 24: Sobel Output

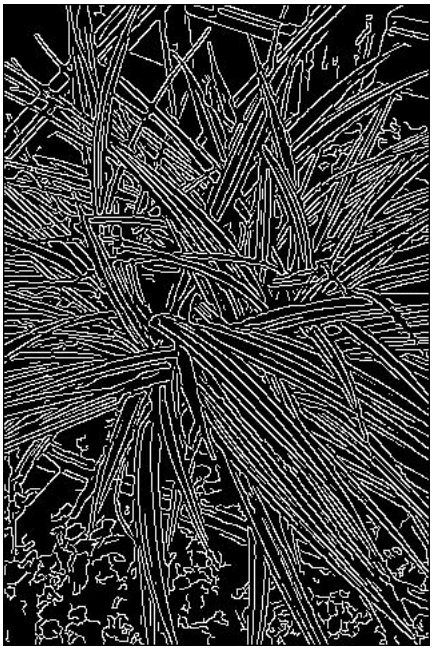

Figure 25: Canny Edge Output

# 10 Special Case

Figure 26: Input Image

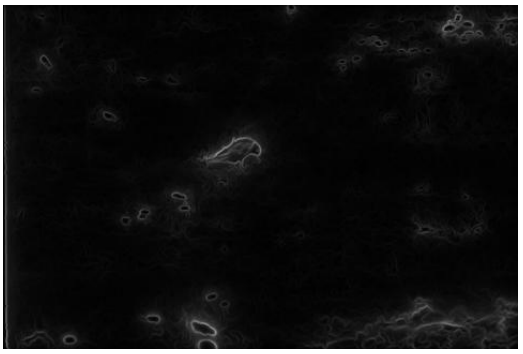

Figure 27: OEF output

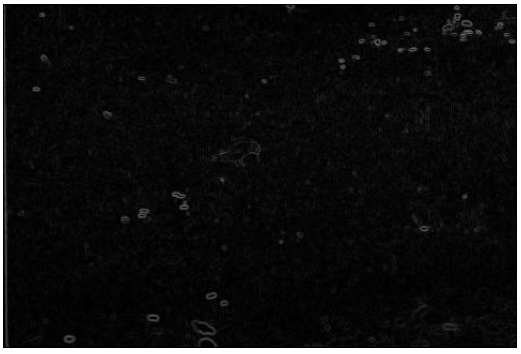

Figure 28: Sobel Output

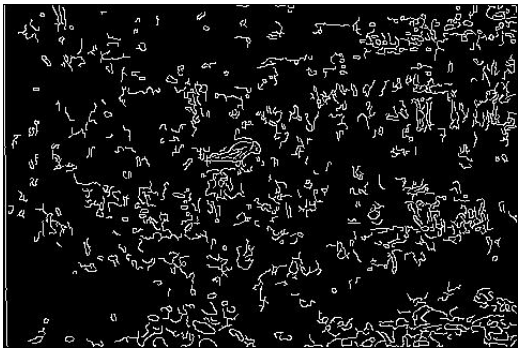

Figure 29: Canny Edge Output

# 11 Correlation with Sobel Output

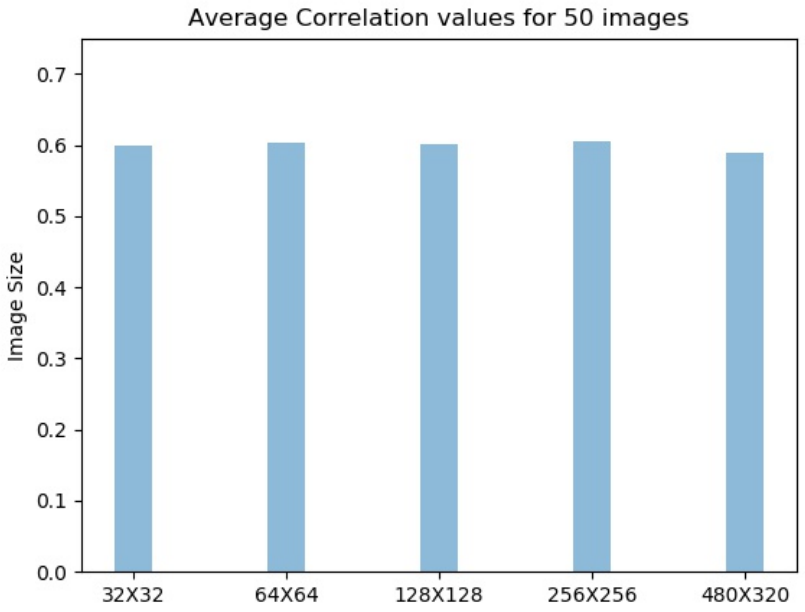


Figure 30: Correlation Graph

# 12 Github link

https://github.com/deepakksingh/CV-Project-2019

# References

[1] *Sam Hallman ; Charless C. Fowlkes. Oriented Edge Forests for Boundary Detection. In Computer Vision and Pattern Recognition (CVPR), 2015*

[2] *Piotr Dollár ; C. Lawrence Zitnick. Fast Edge Detection Using Structured Forests. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2015*