Homework 2 for 2020 spring EE569
Edge Detection & Half-toning
Chengyao Wang
69615991816
chengyao@usc.edu

Problem 1, Edge Detection:

1. Motivation & Approaches:

Edge detection of digital images is a major topic in computer vision. It tries to determine if every single pixel is an edge or not, thus could be modeled as a binary classification problem. In edge detection, edges often consist of boundaries between different objects and surface texture of a single object. But in most cases, we refer to the former one only. And in such cases, the latter will be the source of disruption and bottleneck the performance of edge detection algorithms, especially differentiation-based algorithms. Also, worth pointing out, true labels are annotated by humans to serve as a performance criterion of different algorithms and also training data if the algorithm is machine learning based. But it, as will be discussed later, has some problems.

Sobel Edge Detectors:

There are mainly two approaches to perform edge detection within the scope of traditional computer vision: differentiation based, and machine learning based. Differentiation based algorithms are based on the assumption that sudden change in pixel values, no matter in grayscale or colored images, indicates the presence of edges. Sobel edge detector is such an algorithm and is considered as first order (gradient) methods. Detailed description of Sobel edge detectors will not be repeated here. The operation of getting gradients at a particular pixel location *(i, j)* can also be generalized into linear convolution operations. The Sobel edge detector kernels implemented are:

x-gradient (column gradient)	y-gradient (row gradient)
1/4, 0, -1/4	-1/4, -1/2, -1/4
1/2, 0, -1/2	0, 0, 0
1/4, 0, -1/4	1/4, 1/2, 1/4

There are many ways to handle corner cases in edge detecting algorithms. In this case where Sobel kernel is 3 by 3 in size, we can either do odd / even reflection extension, zero-padding, or just omit the gradient calculation on the most outer set of pixels. Here, we can't find any persuasive preference over any of the three, ***thus the third one is adopted in this assignment***.

For visualizing ∇_x, ∇_y, we need to normalize the gradient values to 0 ~ 255 (inclusive) such that the higher the gradient is at a certain location, brighter the pixel. Worth noticing, ***we took absolute operation of negative values*** because they indicate (left -> right) & (down -> up) gradients. After such operations, the edges will appear to be white with background black. ***Without absolute operations, the background will be greyish with edges being white or black***. ***Both methods are valid for visualization purposes***. To calculate magnitude mapping, we need trace back to unnormalized ∇_x, ∇_y and for each pixel:

$$\mathrm{m}ag(i,j) = \sqrt{\nabla_x^2(i,j) + \nabla_y^2(i,j)}$$

Then again likewise, we need to normalize the magnitude mapping matrix to visualize the results and do thresholding. Notice here that one of the direct approaches may be:

$$bias = min\big(mag(i,j)\big), ratio = \frac{255}{max - min} \Rightarrow mag`(i,j) = (mag(i,j) - bias) * ratio$$

But in the real process of tuning thresholding values for best performance, best thresholding range is found to lie in 30 ~ 50 (for "Gallery") and 50 ~ 100 (for "Dogs"), which is rather small compared to the maximum pixel intensity range. ***This is resulted the un-uniform distribution of the magnitudes where most of the magnitude values are <50 / <100, and fixed step size will lower the precision of the threshold tuning process.*** Thus, as inspired by non-maximum suppression, "min" & "max" above are substituted by empirical fixed values:

$$min = 10, max = 50\ for\ Gallery$$
$$min = 20, max = 80\ for\ Dogs$$

The "normalized" magnitude is cropped to 0 & 255 if they are smaller than "min" or larger than "max". Since all the normalization steps are done on floating points, it will lead to a higher precision threshold tuning process and possibly better result. This argument is also well supported by the actual results. Rigorous implementation of modified normalization mentioned above should involve *min/max* tuning stage, which may be associated to PDF plotting & analysis. Here the fixed constants are determined empirically by observation.

Canny Edge Detectors:

Canny edge detectors are also differentiation-based edge detectors with some additional procedures and tricks for boosting performance. A summary of Canny Edge Detectors steps is:

- 5 by 5 Gaussian filter denoising to increase robustness to noises.
- Gradient magnitude mapping generation, all kinds of first-order kernels may apply.
- ***Non-maximum suppression - Suppress all $mag(i, j)$ which are not local maximum.***
    - ***Aiming to produce "thin edges" or in other words to de-blur the magnitude map or probability map.***
    - ***Local maximum indicates the sharpest edge / most intense edge response locally, thus serves as a representative of the neighborhood.***
- ***Double thresholding & Hysteresis tracking:***
    - ***All pixels having magnitude values below min will marked as "non-edge".***
    - ***All pixels having magnitude values above max will marked as "strong edge".***
    - ***All pixels having magnitude values between min & max will be marked as "weak edge" and will be marked as edge in hysteresis tracking stage if its 8-pixel neighborhood has a "strong edge". Blob analysis is often used for finding such connections efficiently.***

Low threshold mainly plays the role of filtering out noises and texture information. And high threshold determines the set of "seeds" to grow edges because edges often comes connected and in contours. But their effects are coupled, thus needs to be tuned jointly.

CART & Random Forest:

Decision trees, often called classification and regression trees (CART) in machine learning, have long served as an abstraction to decision making pipeline. After involving cost functions & stopping criterion, it was made possible to build automatic, programmable learning algorithms for decision trees to extract features from the dataset given. The construction of decision trees has many tricks, mainly to battle its greedy nature, overfit risks and time complexity. Standardized, idealized construction may be described as follows:

1. Perform brute force traverse over all possible thresholds and all features, calculate target function decrease if such threshold is applied. Notice that target functions in decision tree are additive and related to label purities, e.g. Gini index or cross entropy.
2. Apply the thresholding with the highest information gain or overall label purities.
3. Iteratively do 1, 2 until stopping criterion is met. Common stopping criterion includes when all the regions have data points lesser than some amount, label impurities reached minimum, preset maximum tree depth or number of tree nodes.

The most characteristic of decision trees other than good interpretability is its elasticity. We have a better control on model complexity by specifying different stopping criterions, which makes it suitable for ensemble methods. Many decision trees perform classification independently, and the final result is aggregated by performing majority vote. Weight can be applied but not essential. Or if the model is expected to return a probability distribution of label predictions, ***such result can also be achieved by the occurrence frequency among all the outputs of individual decision trees*** since we can only get one label prediction from each tree. Random forest consists of hundreds of trees, and bagging is used to reduce the covariance between trees effectively alleviating ensemble model variance. There are other tricks commonly implemented also to decrease covariance, including feature sub-sampling & dataset sub-sampling. Since trees are parallel, they form a "forest" & the tricks mentioned above all reflect "randomness".

Structured Edge Detectors:

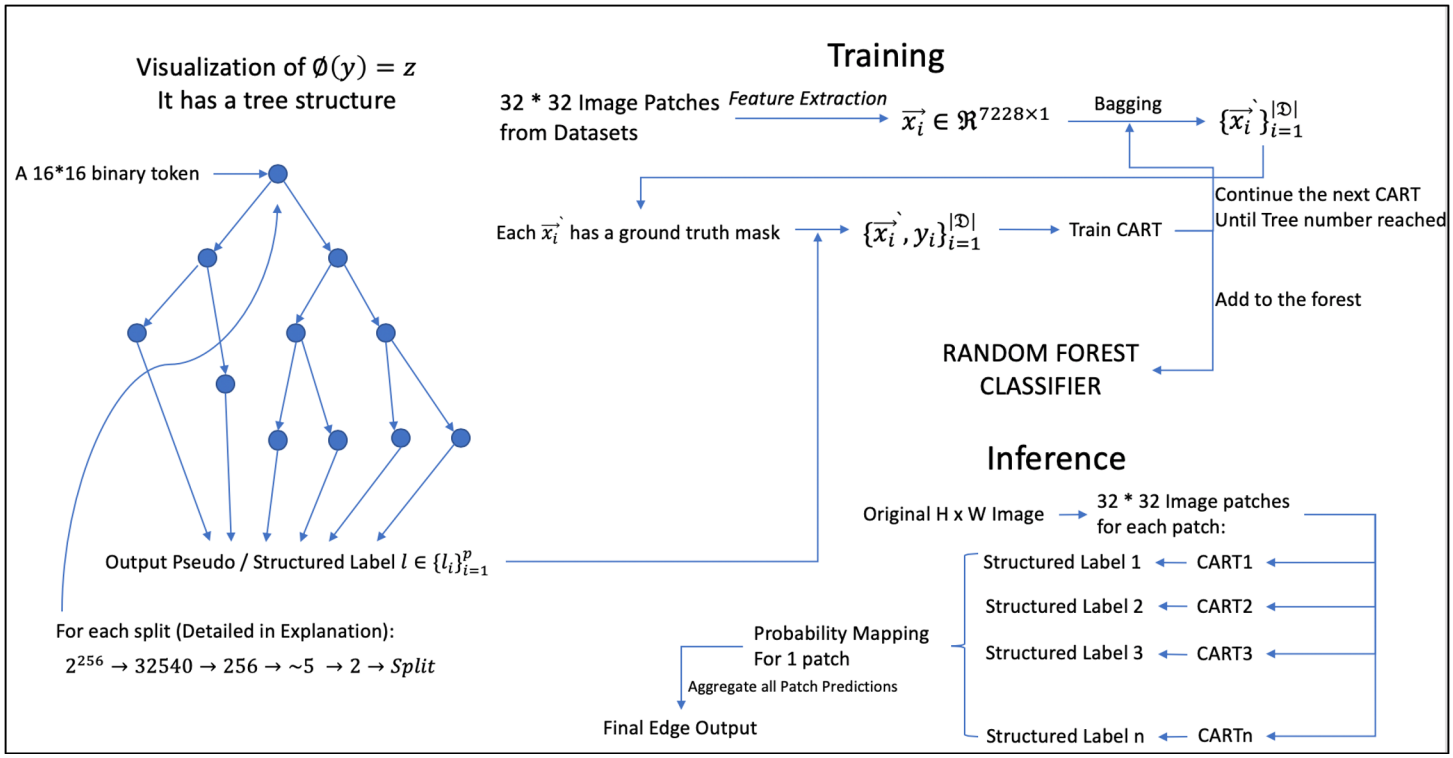
Structured edge detectors are machine learning based which implements a more data-driven approach to the classification problem. According to the original paper[1], there exists variants & enhanced version of SE. But the basic version of SE detectors can be described as follow. Most part of its method is similar to *Structured Token* and detailed part of training a Random forest classifier is omitted as they are described in the previous section:

1. Latent label mapping function $\phi(y) = z$. Just like in *Structured Token,* ***<u>we need to extract features and cluster all possible tokens to assigned structured labels to feasify supervised learning</u>***. Because there're too many, we need to perform dimension reduction to the extent where it's supported by downstream classifiers or else, we'll be facing many problems. Unlike *Structured Token* where the pool of tokens is sampled from ground truths, *Structured Forest* are facing with all possible tokens in a 16 * 16 local window. Such feature extraction & dimensionality reduction is conducted sequentially:
    a. $2^{256} -> C^2_{16*16} = 32640$. There are 32640 pair of pixels in one segmentation mask, and each takes 1 if they belong to the same segmentation mask.
    b. $C^2_{16*16} ->$ m. Sample m dimensions of *Z,* usually takes m = 256.
    c. $m -> \sim 5$. Perform Principle Component Analysis (PCA) and take the first 5 principle components.
    d. $\sim 5 -> 2$. Do K-means clustering with K = 2.
2. ***<u>The mapping function is by nature a hierarchical divisive clustering algorithm and can be visualized as a binary decision tree</u>*** since *dim(z) = 2*. At each split of the tree, we perform such dimensionality reduction & clustering to assign two "pseudo labels" to the tokens in the pool before splitting. Then the "pseudo labels" serve to conduct standard information gain guided splitting.
3. Construction of the mapping function ϕ, or the tree will be stopped after "label" purity criterion / expectation is reached. This tree will eventually produce *p* leaf nodes, which means all the tokens are grouped in *p* clusters. We've successfully "assigned labels" to all the original tokens and the total class = *p*.

Then the labels are used to the train the random forest classifier. We predict 16 * 16 segmentation mask we just clustered & "labeled" in the previous step using features extracted from 32 * 32 image patch. This part is quite similar to *Sketched Token.* Features extracted include:

1. 3 color channels in CIE-LUV color space.
2. Normalized gradients magnitude at 2 scales: original and half resolution.
3. Split each magnitude channel into 4 channels resulting in 8 orientation channels.
4. Blur the channels with a radius of 2 triangle filter and downsample by a factor of 2.
5. Total features 32 * 32 * (3 + 2 + 8 + 13/4) = 7228 per 32 * 32 image patch.

Then bagging tricks are applied to each decision trees and form a forest. Likewise *Sketched Token*, each tree generates one prediction and the total number of trees aggregate to form probability outputs. The inference stage is omitted.



[1] "Fast edge detection using structured forests" by Piotr Dolla´r and C. Lawrence Zitnick

Performance Evaluation:

Definition of performance is always a critical problem in pattern recognition. Different choices will impact model validation stage and possibly yield different favorite parameters. It's always problem dependent and biased inductively towards human's preferences. Confusion matrix is a common practice for visualizing classification results of an algorithm:

		Predicted results	
		Positive	Negative
True Labels	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

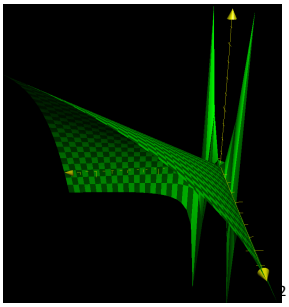
There are mainly three basic evaluation metrics associated base on the four values, which are:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}, precision = \frac{TP}{TP + FP}, recall = \frac{TP}{TP + FN}$$

But, $accuracy$ is much less persuasive under the existence of class imbalance. And $precision$ & $recall$ individually are dependent on the bias in numerous models, especially those with thresholding and linear classification models, like logistic regression. Thus, F1-score will be a good starting point to address both problems and is defined by:

$$F1 = 2 \times \frac{precision \times recall}{precision + recall} = \frac{2TP}{2TP + FP + FN}$$

$precision$ & $recall$ respectively measures two types of errors (Type-I & Type-II error) in binary classification problems. The former measures how often do your model generate a false alarm and the latter shows how often is a true alarm missed. The two are often inversely correlated, meaning increasing one will decrease the other. $F1$ is naturally put forward as combining the two by taking the mean. It's explicitly shown by the following graph:



There are some clever thoughts behind the harmonic mean taken:

1. ***When one if significantly higher than the other one, F1 score will likely be every low. Because the range of precision and recall are restricted to [0, 1], the higher one can't infinitely boost itself to compensate for the loss when the smaller one is ~0. We want to penalize the result if any of the two gets ~0.***
2. ***F1 will be better if we keep $precision \approx recall$, best if equal. It is obvious from the graph, and easily to be proven mathematically. When the sum of the two is a constant:***

$$\frac{d}{dx}F1 = \frac{d}{dx}\left(2\frac{x(\alpha - x)}{\alpha}\right) = 0 \Rightarrow x = \alpha - x = \frac{1}{2}\alpha$$

However, we need to modify the definition of *precision* & *recall* to fit edge detection cases though it's a classification problem in nature. ***TP/FP/FN IS NOT CALCULATED BY PIXEL TO PIXEL CORRESPONDANCE. This part is originally introduced by the evaluation part of the "structured forest" MATLAB source code[3] and migrated into C++.*** A fine-tuned result based on ***pixel-to-pixel correspondence F1*** will be presented for reference later in the report but is extremely low. The reason lies in the edge detection task itself, and more importantly, non-maximum suppression. ***In one sentence, one-to-one correspondence is too strict to be meaningful for edge detection tasks.*** There can be potential edge of width 2 ~ 4 presented to algorithms and they, if includes non-maximum suppression, will only choose one based on the highest magnitude. ***In other words, algorithms have explicit, quantitative criterion for choosing 1 pixel along the normal direction to be an edge, but problems arose when humans usually don't and can't.*** Human labeler cannot distinguish between a pixel with magnitude 100, and another with 105. They choose randomly from the algorithm's perspective. On the other hand, if the algorithm doesn't have *NMS* part, *precision* will decrease sharply around this local region. As thanks to the definition of F1 score again, its value will be heavily restricted to a small range and will lose some reasonability as a performance metric.

[2] This graph is generated by MacOS built-in application - Grapher. The domains are not restricted to [0, 1] x [0, 1], because the function behavior is consistent across all $\Re^+ \times \Re^+$

[3] https://github.com/pdollar/edges, file: edgesEvalImg.m

Thus, the F1 score calculation method adopted by "**Structure forest**" is a much more loose and tolerant calculation method. Some changes are forced to be made and the reasons will be explained. The main idea is listed as follows and it will be denoted as ***SE-F1*** for the rest of the report:

1. *Precision* & *recall* are calculated from 1 & 5 (for 5 ground truth) matching matrices respectively, which are all calculated by the following process.
2. To derive the matching matrix for *precision*:
    a. For each pixel *(i, j)* in prediction which is positive, we are finding corresponding pixel in GT_i and the search scope is a small neighborhood centered around *(i, j).* Of course, if a corresponding pixel exists:
    $$Matching\ Matrix\ Edge\ [i,j]\ =\ True$$
    b. The detailed procedure is conducted by **an pre-compiled MATLAB function *correspondPixels()*** thus is impossible to peek inside. But the search range is determined explicitly by Euclidean distances by its input arguments, with the maximum being a small ratio of the diagonal length. **Structured Forest** takes 0.0075, that is approximately 9 * 9 square region for the image we're working on. The matching process is substituted by:
    $$Matching\ Matrix\ Edge\ [i,j]\ =\ True\ if\ there\ is\ one\ or\ more\ positive\ in\ the\ search\ range$$
    Such criterion may be too loose; thus, we shrunk the search range to 5 * 5. Also, we can't see any relationship between edge predictions and image sizes, because the former is often regarded as a local procedure.
    c. 5 different matching results with GT_i are aggregated by OR operations – we only need 1 match among the 5 GT_i to be True.
    d. $precision\ =\ {number\ of\ true\ in\ matching\ matrix}/{total\ positive\ predictions}$
3. To calculate *recall*:
    a. We reverse matching procedure by starting from each pixel *(i, j)* in GT_i which is positive to predictions.
    b. 5 different matching results by GT_i are aggregated by ADD operations – we are adding up the number of **TRUEs** in 5 matching matrices respectively.
    c. $recall\ =\ {number\ of\ true\ in\ 5\ matching\ matrices}/{total\ postive\ pixels\ in\ all\ 5\ GTs}$
4. The rest is the same for calculating F1 score.

We can see 5 major difference with respect to ***pixel-to-pixel correspondence F1***:

1. It allows to search for correspondence within a range.
2. Match between any GT will count as true positive.
3. 5 ground truths are taken as one when calculating *recall*, thus its definition of "mean" is different.
4. *Precision* & *recall* are calculated separately, with true positives in two formulas are no longer the same.
5. It doesn't pay any attention to True negatives.

But for analyzing the difficulty of different images and making guesses on difficulties, the method of calculating F1 score should not affect the overall judgements. ***We can say "Gallery" will generally have a higher score than "Dogs". One convincing reason can be textures of the grass in "Dogs" will confuse algorithms into generating numerous false positives which will likely to be omitted by the eyes of the human labelers.*** They will not benefit from the ***SE-F1***. There could be debates over directly comparing F1 score between different images, examples like the image size (the number of predictions to make) or sampling conditions like focus problem in "Dogs" are not related to image semantics but will effect the performance of algorithms. But it will always be a good metric for tuning the parameters of the algorithm for achieving maximum performance on one single image.

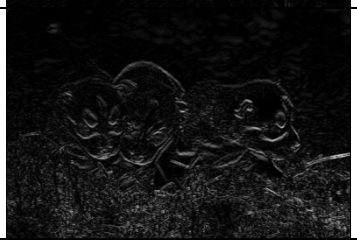
Specifications and methods & tricks during implementation:

1. Canny edge detector & Structured edge detector are all called from OpenCV library. The latter is a machine learning based algorithm, thus requires training data to train its random forests model. ***Here we directly download a pre-trained model from[4], and load it to perform prediction.*** But it still outputs probability maps, thus we can also tune the thresholding constant for better performance on top of its results.
2. Standard implementation of Canny edge detectors involves denoising preprocessing stage with 5 * 5 gaussian kernels. But here since the images are obviously free of noise, ***we are omitting this step because it will decrease the performance of detectors*** because low / high-pass filtering are opposite operations.
3. Non-maximum suppression is always adopted by Canny edge detectors but is also implemented in SE detectors though original author of SE doesn't explicitly mention the use of *NMS*. With the help of ***SE-F1***, *NMS* is will increase the score by increasing *precision*.
4. ***Parameters in the concerned algorithms are tuned using exclusive grid search over most of the parameter space***. They are mostly thresholds in the last classification step (1 for Sobel, 2 for Canny and 1 for SE).
5. ***Little are known about the downloaded pre-trained SE models, though it was archived by the OpenCV official library***. Since "Gallery" & "Dogs" all come from BSDS, they are believed to be part of the training dataset. In other words, they were supposed to be seen by the model during training and their "distributions" are not an anomaly to the distribution pre-trained model learned.
6. Ground truth labels are copy & pasted from MATLAB by hand into text files. Then headers are added, and they turned into *.pgm files for the program to read.

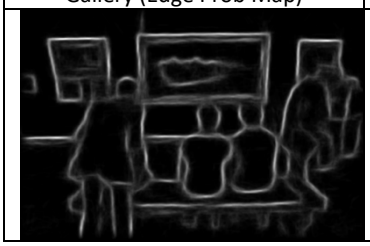
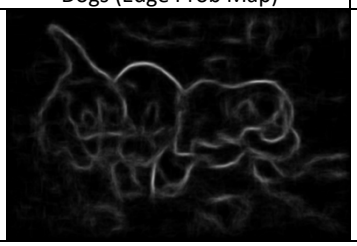
[4] https://github.com/opencv/opencv_extra/blob/master/testdata/cv/ximgproc/model.yml.gz

2. Results & Analysis:

Direct results are put in the appendix for a better view & completeness of charts.

Grey Scale Image	X(Col)-gradient	Y(Row)-gradient	Magnitude Mapping
			
			

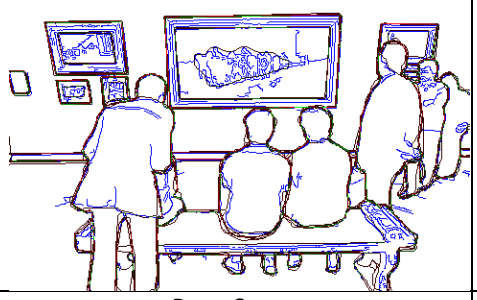
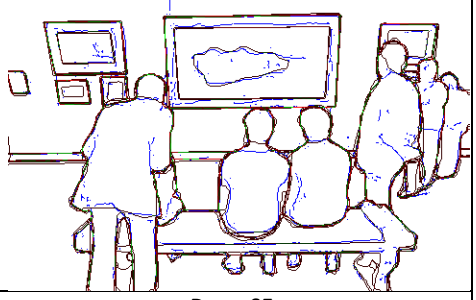
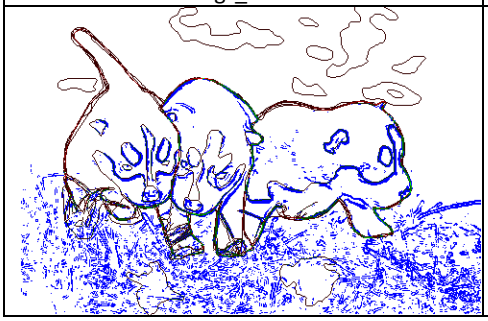
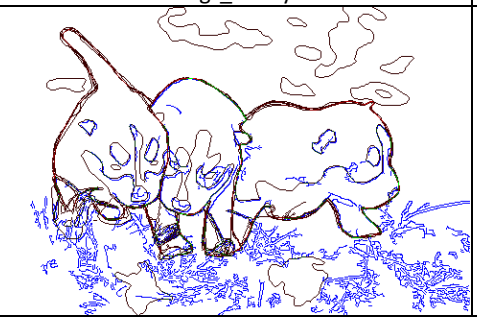
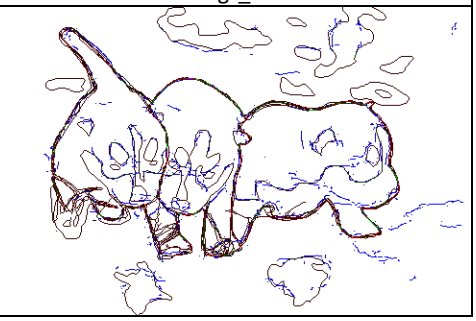
Structured Edge Detector: Direct output of OpenCV package

Gallery (Edge Prob Map)	Gallery (Non-Maximum Sup)	Dogs (Edge Prob Map)	Dogs (Non-Maximum Sup)
			

A summary of the best F1 achieved by the algorithms

pix-pix-F1	Sobel	Canny	SE	**SE-F1**	Sobel	Canny	SE
Gallery	0.1791	0.1498	0.2165		0.7796	0.7635	0.8440
Dogs	0.1078	0.1035	0.1455		0.4989	0.4574	0.7229

Error Visualization on Weighted Ground Truth (TP, FP, FN)
(Based on pix-to-pix correspondence)

Gallery_Sobel	Gallery_Canny	Gallery_SE
		
Dogs_Sobel	**Dogs_Canny**	**Dogs_SE**
		

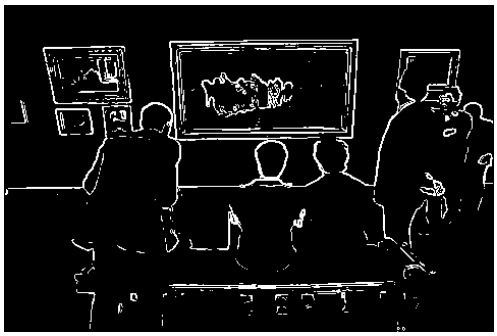
The unique characteristics are quite obvious among the results generated by three edge detectors that make them distinguishable. ***Sobel edge detector doesn't have non-maximum suppressions thus tend to generate much wider edges and more spatially isolated predictions. Canny***

edge detector in comparison produces much narrower and heavily connected edges, but still has limited capabilities in filtering out texture patterns. SE edge detector benefits from previous experiences, which makes it robust to texture interference. Notice that SE is better equipped with non-maximum suppressions because as probability mapping shown above, it outputs are thick edges that will possibly lower precision. Many positives generated are redundant.

From F1 score's perspective, we can see that SE detector is the best with Sobel & Canny being similar. But visually speaking, Canny are better since its output are clearer and neater. And, as afore predicted, all three algorithms tend to achieve better scores on "Gallery" than "Dogs". A detailed analysis on the three algorithms are as follows:

Sobel Edge Detector:

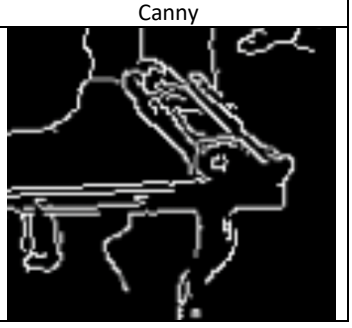
1. As one of the simplest differentiation base algorithm, it does exploit difficulties in filtering out the floor patterns and grass which is considered as textures. Afterall, it is only based on gradient magnitude information and nothing more. ***Thus, simple linear thresholding performs badly when distribution of gradient magnitude of true edges between dominant objects heavily overlaps that of texture patterns***. This is similar to the case of Naïve Bayes Classifiers. As an observation from thresholding tuning process, *the "residual dots" on the floor* of the "best result" share the same gradient magnitude as the "dots" belonging *to the right leg of the long chair and men's shoes*. Result image with higher threshold is a good illustration and proof because they get filtered away simultaneously:



2. Things get more complicated in "Dogs". ***The two distributions are completely interweaved***, and we will lose the whole picture if we try to get rid of the grass. It seems to be even more a disaster for the tail part of the leftmost dog where Sobel (and even Canny) completely missed all of the edges, but there some other factors if considered carefully:
   a) ***Focus of the camera.*** Notice the fact that camera lied its focus on the faces of the dogs, making the tail part blur. When computing gradients, such blurring can also "average" away the edges. And that's the reason why Canny edge detector chooses its denoising filter carefully and we are not easily introducing denoising steps.
   b) ***It's predicting upon gray scale images***. The color of the tail should be black, but the grass is green. If done on G channel individually, Sobel is believed to get better results because the variation of pixel values mainly took part on the G channel only. Though G took the largest ~60 percentage when generating grey scale images, ***it can be seen from the true color->gray results that tails became hardly observable even by human eyes***. As one of the proofs to this argument, notice the forehead of the dogs. Sobel always generates edge here for the rightmost dog no matter the threshold, simply because the color pair on the two sides are black and white. The back of the rightmost dog is the same case.
3. Also, from the summary below, we could see Sobel tends to generate more positive predictions than the other two algorithms. But surprisingly for the two images here, ***SE-F1*** are not substantially lower than *SE* and even slightly higher than *Canny*. This is believed to be the result of how true positives are determined by ***SE-F1*** because 1 isolated, positive in ground truth may help 10+ positives become true positives. ***But since humans generally prefer Canny's results, we could implement such favor by adding additional terms to penalize on more positive predictions or just make smaller search regions in SE-F1, which can be some function of the number of positive predictions & total pixels (image size).***

Canny Edge Detector:

1. Canny edge detector is also based on differentiation but distinguishes itself from Sobel detectors on two additional processes: Non-maximum suppression & Hysteresis tracking. NLM tries to lower false positive rate while the latter seeks to utilize spacial information. We can observe the consequence of both by comparing its results to those of Sobel's:

Non-local Maximum		Double thresholding & Hysteresis Tracking	
Sobel	Canny	Sobel	Canny
			

The predictions by Sobel detector are obviously thicker than those of Canny's. ***This is the result of non-maximum suppression and the direct reason why Canny is generating far less positives than Sobel as the char below mentioned***. And by "growing" edges from "strong pixels", edge pixels generated by Canny detector are more correlated to its neighborhood ***thus Canny's predictions often come in contours and much correlated.*** Also as stated in the algorithm introduction part, the two thresholds work jointly to make ***Canny detector more robust to textures and makes fewer positive predictions that are "not continuous" & "isolated"***. The floor and the grass part are good proofs.

2. Notice the fact that Canny detectors have a slightly lower F1 score than Sobel detectors, and that's quite unexpected. Here is a summary of the factors that contribute to F1 calculation. They are respectively the best image under two F1's respectively thus is not the same image. ***But interestingly, it is also noticed that parameters of the best result for algorithms under different F1 are actually not far, which means that they are both serving as a good metric. SE-F1*** can be proven to be more tolerant on *precision* than *recall.*

pixel-to-pixel correspondence F1								**Structured edge F1**			
Gallery		TP	FP	FN	TN	Precision	Recall				
Sobel	GT1	2346	15516	2107	134432	0.1313	0.5268	Matched Positive in PRED	13105	Precision	0.6905
	GT2	1476	16386	2498	134041	0.0826	0.3714	Total Positive in PRED	18978		
	GT3	1856	16006	2375	134164	0.1039	0.4387	Matched Positive in GT	20074	Recall	0.8950
	GT4	1787	16075	2139	134400	0.1000	0.4552	Total Positive in GT	22429		
	GT5	2093	15769	3752	132787	0.1172	0.3581				
Canny	GT1	1099	7554	3354	142394	0.1270	0.2468	Matched Positive in PRED	7534	Precision	0.6657
	GT2	742	7911	3232	142516	0.0858	0.1867	Total Positive in PRED	11317		
	GT3	915	7738	3316	142432	0.1057	0.2163	Matched Positive in GT	20074	Recall	0.8950
	GT4	1004	7649	2922	142826	0.1160	0.2557	Total Positive in GT	22429		
	GT5	1051	7602	4794	140954	0.1215	0.1798				

Surprisingly, the two detectors have the same recall which means the *"false negatives"* lie in the blind spot of differentiation & magnitude methods, at least to *Sobel kernels*. Thickened edges in *Sobel* detectors still cannot establish correspondence to whose pixels. These regions are fractional and distributed across the whole image. For *precision*, we can first notice that *Canny* produced 2 times lesser positives than *Sobel.* And also, in our ***SE-F1*** calculation, 1 positive in GT can possibly generate 25 "*true positives"* thus *Sobel* is not penalized for making more guesses. A simply mathematical example will be a great explanation:

$$\frac{a+h}{b+h} > \frac{a}{b} \; where \; b > a > 0, h > 0$$

However honestly, such pixel matching procedure need to be further analyzed to fully explain the results in the chart, e.g. the inversibility of matchings, whether if it's too tolerant or how much extra bias it has towards precision. And also, such method is likely to lose feasibility outside edge detection scenarios because such search windows rely on spatial structure of the data.

3. Canny detector does perform better in filtering out those high edge response pixels from textures:

Below is an simple reasoning explanation:

a) Their surroundings pixels are defined as "weak edges" or "non-edges" by increasing the lower threshold and filtered away.
b) Theirselves can't become "strong edges" by increasing the higher threshold.
c) They become isolated, thus eventually also get filtered away.

And it's also better in extracting real edges from regions like "human shoes" & "bench legs", where the edge response is close to those relative high response from textures elsewhere and they are coupled if we perform global thresholding. Because they are connected to some "strong edges", these kind of "weak edges" get eventually classified as edges.

4. Low and high threshold need to be tuned jointly:

A very natural conclusion from the chart of different T1/T2. Low threshold can't be too low to filter out low response textures nor too high to give low response edges a chance to be connected. High threshold can't be too low to preserve the chance of filtering out high response unconnected textures nor too high because we may not recover some true edges with too less "seeds".

Structured edge detectors:

Machine learning based algorithms are desired to beat differentiation-based algorithms, simply because it has the access to training data. ***And from slightly different perspective, SE actually utilizes a much larger window than Sobel, Canny (16 * 16 vs 3 * 3)***. More importantly, as afore mentioned with regard to ground truths, human labelers are not that universally correct often sketching edges as opposed to label for general machine learning instances. ***They exploit some personal style and preference in their sketches***, which is obvious if 5 ground truths are put into comparison. ***SE learn such style statistically***. As some of the proofs, *SE* are capable of noticing the tail of the leftmost dog, omitting the details of the center painting and sketching out the back of all the dogs.

But it is still unknown why *SE* produces so many "isolated pixel dots". But we can perform some easy tricks to remove the dots by post-processing. And also, as the setback of all machine learning algorithms, especially random forest, data-hunger is even more of a problem to edge detections simple because labeling edges is much more expensive than labeling traditional machine learning instances and automatic data mining will not solve the problem.

Problem 2, Half Toning:

1. Motivation & Approaches:

Digital half-toning problem seeks to find the best way to binary quantize a gray / colored image while retaining the original information to the most. It's widely used in the printing industry because printers are only capable of apply or not apply ink to a certain location. And also, such quantization makes the size of the images much smaller, which is also good for storage and memory-limited situations. Since we are binarizing the colors, the target color space after half-toning consists of 2 / 8 colors and is 1 / 3 dimensional:

$$For\ gray\ scale\ images: 0\ Black, 255\ white$$

Colors	Black	White	Red	Green	Blue	Magenta	Cyan	Yellow
Coordinate	0, 0, 0	255, 255, 255	255, 0, 0	0, 255, 0	0, 0, 255	255, 0, 255	0, 255, 255	255, 255, 0

There are mainly two approaches to perform half-toning: dithering and error diffusion. Dithering by nature is thresholding method – every pixel is quantized to 255 if above T and 0 if below T. Error diffusion takes the notion of "quantization error" and tries to compensate the error that current pixel generated by distributing to the pixels later on.

Unlike other topics like image denoising and edge detection, there seems to be no universally accepted or applicable quantitative method for determining the performance of different algorithms. It bases more on visual impression, thus is more subjective.

Most of the algorithms covered here are explicitly explained in the statement of the problem, thus is again not repeated here. But here are some brief description of specifications and implementation methods & tricks:

1. **For "random thresholding" dithering method, uniform number generators are used. It is provided by STD library of C++.**
2. "Dithering matrix" is implemented in the form of "shifting mask" with no overlapping. For corner cases where the mask reaches out of the image boundary, ***they are just skipped***.
3. Index matrix / threshold matrix is defined mathematically by recursive functions, thus is also generated recursively in code.
4. ***"Error diffusion" can be generalized as additive de-convolution operations***. Unlike any other methods before where another piece of memory is necessary and is allocated elsewhere to store the result image because operations to pixels are parallel and independent, ***error diffusion methods all work on pixels sequentially thus "diffusion of error" is done directly on the original image in place for each pixel***.
5. For each kernel in error diffusion methods, its mirror image is also defined in the meantime. Because serpentine traversal requires the latter be applied to odd row index cases.
6. ***Functions for determining the affiliation of individual pixels in MBVQ-based methods are presented in the original paper by Hewlett Packard, and L2-norm is used for fetching the closest binary color in the 3D color space for simplicity***. But under time efficiency requirements, we could manually formulate decision trees for fetching closest points because L2-norm metric & linear Euclidean space naturally leads to linear decision boundaries.
7. Target color space only takes 8 values thus 1 byte (unsigned char) is just sufficient to encode 6 potential tetrahedral with the locations of 1s indicating its vertices. It's defined as constants in the header and once fetched, iterate through its bits to find the closest point.

```
// Define constants for MBVQ Color diffusion
// Use 8 bits to encode Quadruples information
// White(7) / Yellow(6) / Cyan(5) / Magenta(4) / Green(3) / Red(2) / Blue(1) / Black(0)
# define CMYW 0xF0; // 11110000
# define MYGC 0x7A; // 01111000
# define RGMY 0x5C; // 01011100;
# define KRGB 0x0F; // 00001111;
# define RGBM 0x1E; // 00011110;
# define CMGB 0x3A; // 00111010;
```

Color Halftoning – Separable Error Diffusion & MBVQ:

For colored images, it's always feasible to do operations channel-wise by pretending they're grey scale images. So is the case for color halftoning. ***But one the other hand, they don't fully utilize cross-channel information to help generate better results.*** In halftoning, we are essentially facing binary quantization problem in the target color space – 3D if colored, 1D if gray scaled. ***Conducting halftoning operations separately when fetching the nearest quantized color is equivalent to substituting the colored pixel itself with its projections on the three orthogonal axes, which may not be the nearest point (best results) in 3D space***. Thus, MBVQ-based halftoning is put forward by Hewllet Packard:

$$Core\ steps\ of\ \ MBVQ-based\ method: Find\ pixel\ affiliations \rightarrow\ find\ closest\ point$$

The core idea of MBVQ-based method lies in brightness. It's long been widely accepted that human eye are low pass filters and Hewllet Packard took this as entry point, reaching to a reasonable argument that we are more sensitive to brightness than any other thing else because they just got filtered away. It solves the most crucial problem in Half-toning – lacking explicit quantitative objective functions. If inspected carefully, the brightness scale HP put forward are arranged in positive relationship with their wavelength for base colors (RGB), and binary-colors (CMY) are just the opposite to their compliments. ***This idea is embodied in the first step stated above – finding tetrahedral***. And after determining a candidate pool of 4 colors, the rest is left to any mathematically well-defined metric functions. For implementation, MBVQ-based methods are generally the same as separate approaches except their definition of "closest points". Thus, they use utilize the same error diffusion schema – the same code.

The color space said to be used by printers are CMYK but is not related to this problem. Printer name colors according to the wave they are absorbing, thus ***cyan*** is still on earth ***red*** to human eye. If one argues that such transformation is mandatory, there are still reasons to believe that linear transformations and its inverse in Euclidean spaces will collapse into identity mappings, thus will give the same results because we have to transform it back before plotting.

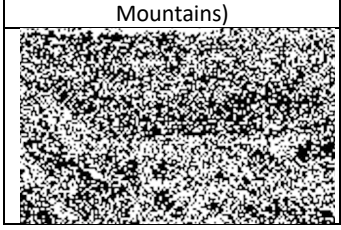
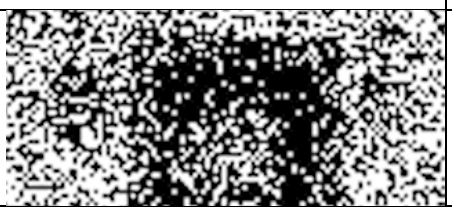
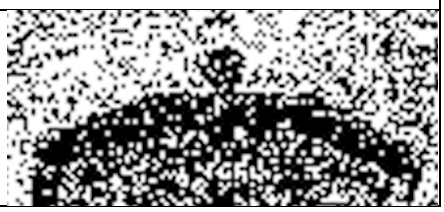
2. Results & Analysis:

Results are put in the appendix for a larger size presentation and better pixel-level details.

Comment & analysis are as follows:

Dithering:

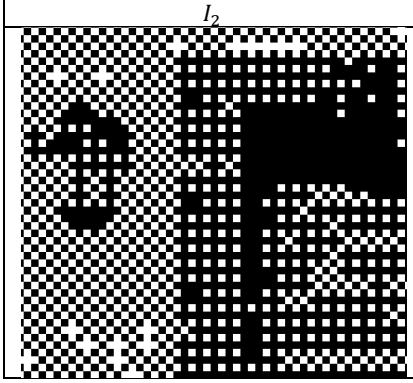
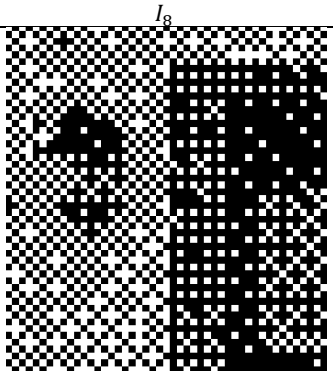
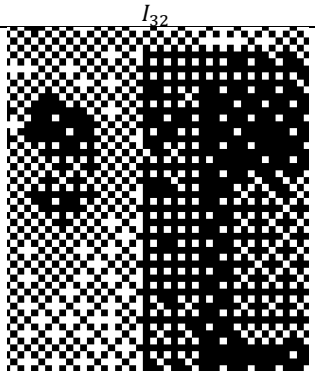
1. Fixed thresholding is more of a style transformation rather than a half-toning method because there is no visual similarity. ***Varying the threshold probably will produce better results***.
2. Adding noise to the threshold is equivalent to subtracting the same amount from pixel values, which is just merely generates noise. But it does exploits image patterns and information. ***One crucial reason is the use of uniform noise instead of other noises like gaussian noise.*** Take a small local neighborhood with similar pixel values for justification, the number of pixels that will turn white due to noise (for a dark neighborhood) is proportional to their relative pixel values with respect to 50% / 128. Under the automatic low pass filtering, or brightness theory by Hewllet Packard since it's a gray scale image, the ensemble behavior around the region will be able to recover the original image to some extent. Such information recovery rate first decreases then increases again with the increase of region variance. I.e. it will favor extremely sharp edges or uniform regions the most. A simple illustration:

Uniform Regions (Background Mountains)	Intermediate (Door & door frame & glass)	Sharp Edges (Tip of the light house)
		

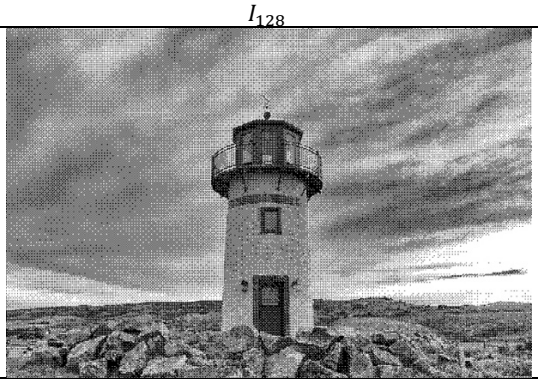
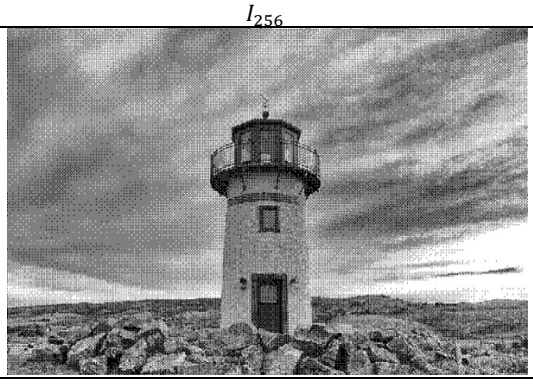
3. Dithering matrix can also be taken as applying noises to the thresholds, though the noise is determined. By the carefully constructed dithering matrix, such recursive threshold pattern could have a more promising, stable way of "dithering" errors round its neighborhood than using completely random noise. Here is a 4-size index matrix:

$$I_4 = \begin{bmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{bmatrix}$$

Notice small numbers, e.g. 4, 1, is surrounded by the largest numbers meaning that if a dark pixel is lucky enough to be turned on at "1", its neighbors should be compensating for the rest of the pixel values. And also, if argued carefully, vice versa. ***And this "cross" pattern in index matrix is present no matter the size and it results in the most noticeable characteristic of images processed by dithering matrix – chess board crosses:***

I_2	I_8	I_{32}
		

As expected, larger dithering matrix produces better results, visually, simply because larger matrices possess higher precisions in distributing brightness. But it has a limit.

I_{128}	I_{256}
	

One reason might be related to the pixel value distribution of the original image. Some images don't need such a high dithering precision, because most of the values cluster around some certain values. Also notice the largest index for $I_{2n} = (2n)^2 - 1$, thus the range of I_{16} already exceeded the range of pixel values, which is $256 = 16^2$.

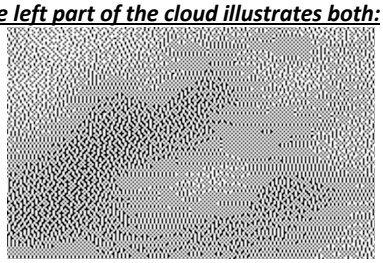
There can be some improvements made to dithering matrix. As one conclusion of the previous section, larger matrix does not guarantee better results. Because larger matrices are defined recursively from smaller matrices, it's thus actually only differs itself in thresholding precisions. ***But the larger size additionally provides spaces for generating more "random"/ evenly distributed patterns, rather than the one it inherited from I_2.*** A toy proposal for improving dithering matrix method can be using the following dithering matrix:

$$T = \begin{bmatrix} 240 & 144 & 208 & 48 \\ 176 & 64 & 0 & 112 \\ 96 & 32 & 80 & 160 \\ 16 & 192 & 224 & 128 \end{bmatrix}$$

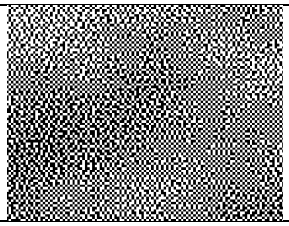
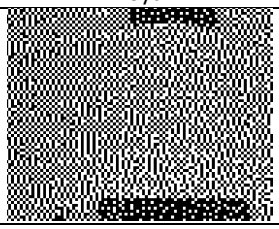
4. Visually speaking, the performance of three different error diffusion methods can be sorted as follow:

$$Floyd - SteinBerg's \leq JJN \approx Stucki$$

The first inequality mainly lies in the "un-random noises" / high occurrence of similar patterns and noticeably sharper separateness between adjacent regions. The left part of the cloud illustrates both:



The precision of Floyd-Steinberg's matrix limits its capabilities to diffuse errors in a randomized manner. Its size limits error within far too small region thus creating artifacts mentioned above.
The rest two has their own advantages and disadvantages. *Stucki* is larger sized *Floyd* because they have similar error diffusion weight distribution, in comparison with a more uniform distribution that of *JJN's*. Thus, *Stucki* also produces similar characteristics with *Floyd*, but does lowers the occurrence of similar patterns.

Less but still observable boundaries (Left cloud)		Lesser pattern reoccurrences (Door - window)	
Floyd	Stucki	Floyd	Stucki
Presented above			

Due to much more uniformly distributed error diffusion weights, *JJN* presents a much lesser noticeable boundary and also much less pattern reoccurrence. But *JJN's* results gives a much more darker feeling, and if observed carefully, dark pixels do tend to gather more easily than *Stucki's*. A convincing reason still need further exploration, but it may still be related to more "uniform" distributions – pixels are more influenced by others in *JJN* and errors accumulate faster than *Stucki's*.

5. ***For approaches to optimize error diffusion methods (on gray scale images)***, validating different weight distributions & matrix sizes are always a good starting point. ***A better weight design principle is expected to be the same as the suggestion in Point 3 – a larger and more dispersed / randomized / evenly distributed matrix***. Secondly, we could build a new kernel on top of existing ones by (Take *JJN* for example):

$$\frac{1}{48}\begin{bmatrix} & & & & & \\ & & & & & \\ & & x & 7+\varepsilon & 5+\varepsilon \\ 3+\varepsilon & 5+\varepsilon & 7+\varepsilon & 5+\varepsilon & 3+\varepsilon \\ 1+\varepsilon & 3+\varepsilon & 5+\varepsilon & 3+\varepsilon & 1+\varepsilon \end{bmatrix} \quad where\ \varepsilon \sim Normal(0, 0.1)$$

Kernel at every position will be slightly different than the other one and it seems to be capable of alleviating repeat patterns. Thirdly, we could also use a full weight kernel rather than current half-valued ones. Because, that increases the error dispersion ranges, splits the errors across more pixels and will now be bi-directional because error propagates into the past. In order to implement such kernels, we could perform forward + backward serpentine traversal multiple times until something converges, e.g. quantization results or total quantization errors.

6. ***The biggest problem of separate error diffusion is still neglecting cross-channel relationship as afore mentioned.*** This will not result in such devastating consequences here is error diffusion and more importantly, this image chosen. **Because the image is mostly consisted of black + green for background and red + yellow + pink (red + white) for flowers**. They already are or is close to our 8 target colors, thus lesser error will produce therefore fewer artifacts. "Brightness" assumption made by MBVQ is actually implemented in separate methods because for target color space is 1D and the two core steps of MBVQ stated above collapsed into one which is equivalent to thresholding.
7. ***JJN's*** results here in turn are a little brighter than the other two methods, mainly in the background part. ***But there seems to be no eye observable difference between separated methods & MBVQ-based methods.*** However, error diffusion does make the images brighter than the original picture. One can argue that error diffusion methods enhances edges, but still reasons are unknown. Some say it's related to zipping format of the image and also screen quality & driver programs. Here another guess is

that “the perception of brightness of human eyes is not linear”. 10 white + 10 black pixel will be visually brighter than 20 grey pixels, especially when black & white pixels are interpolated. May aimed to address the intrinsic brightness enhancement, WIKI[5] suggests a 4-channel color model that orthogonalize brightness from RGB values and conduct the subsequent process. This is left for further exploration.

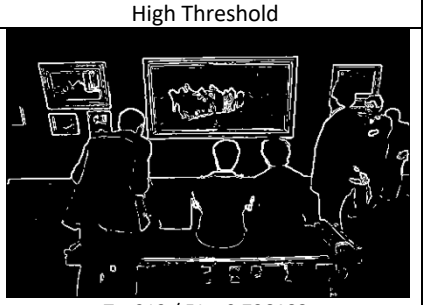
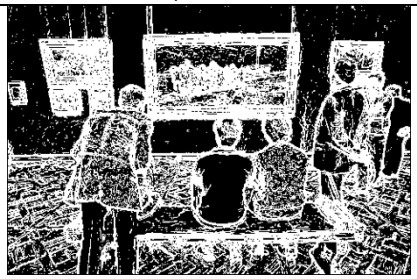
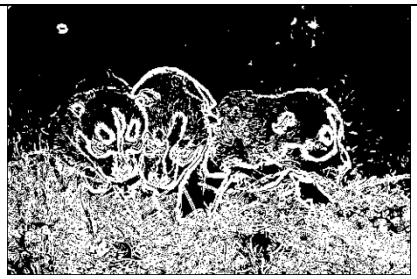
Appendix:

Main functions index:

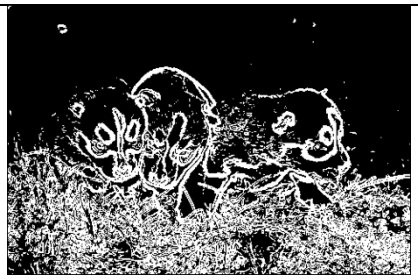
1.	ee569_hw2_sol::f1_score_cal()	returns *pix-to-pix f1* or *SE-F1*
2.	ee569_hw2_sol::kernel_init()	returns all weight matrices / kernel used in assignment 2
3.	ee569_hw2_sol::Sobel_edge_detector()	main function for Sobel edge detector
4.	ee569_hw2_sol::Canny_edge_detector()	main function for Canny edge detector
5.	ee569_hw2_sol::Structured_edge_detector()	main function for SE edge detector
6.	ee569_hw2_sol::Dithering_fixed_T() / random_T()	main function for pixel dithering
7.	ee569_hw2_sol::Dithering_Mx()	main function for dithering matrix
8.	ee569_hw2_sol::Error_diffuse_DO()	perform error diffusion
9.	ee569_hw2_sol::Error_diffusion_Floyd / JJN / Stucki	main functions for gray image diffusion
10.	ee569_hw2_sol::Error_diffusion_NaiveColored	main function for separate color diffusion
11.	ee569_hw2_sol::fetch_quadruples / fetch closet	utility functions for all error diffusion
12.	ee569_hw2_sol::Error_diffusion_MBVQ	main function for MBVQ-based error diffusion

Results:

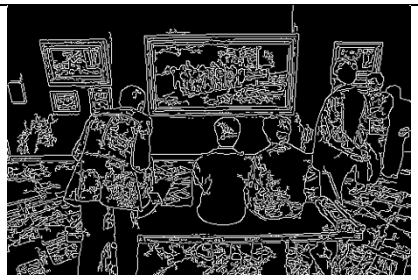
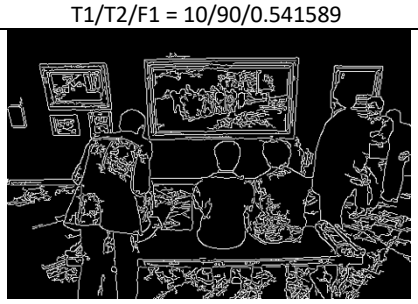
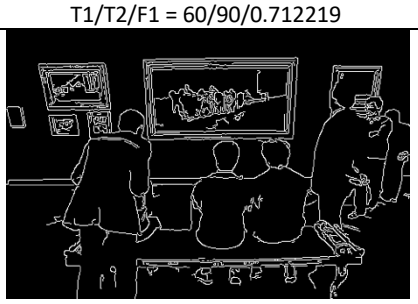
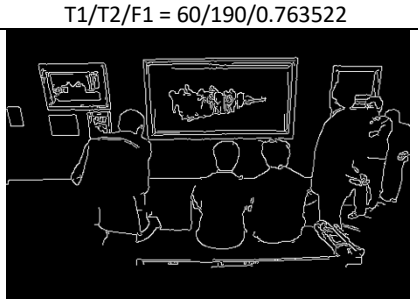
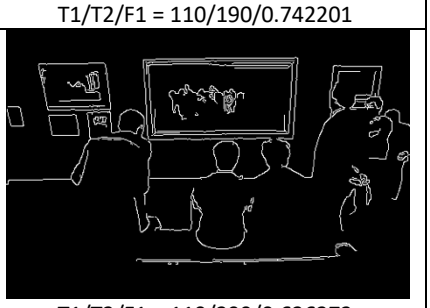
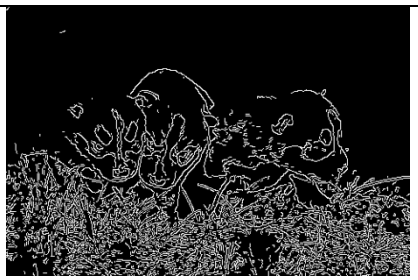
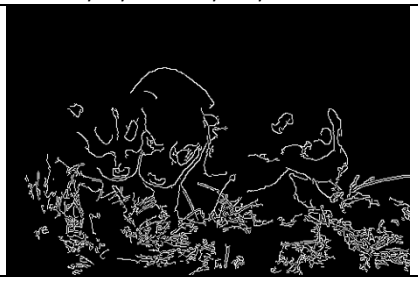
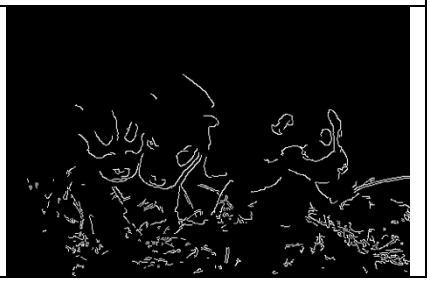
Sobel Edge Detector (Thresholding on After-Normalized Magnitude Mapping)
Parameter search range T : 0 ~ 256, 1

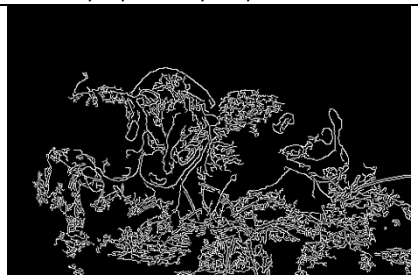
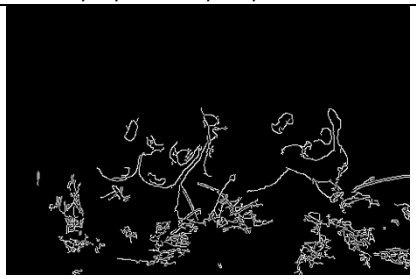
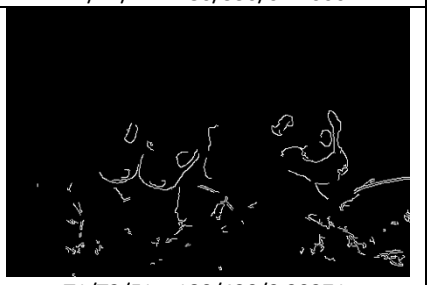
Gallery	Low Threshold	Best Threshold	High Threshold
Modified Normalization	 T = 13 / F1 = 0.71432	 T = 113/ F1 = 0.779626	 T = 213 / F1 = 0.736466
Standard Normalization	 T = 8/ F1 = 0.620964	 T = 33/ F1 = 0.779586	 T = 150/ F1 = 0.139441
Dogs			
Standard Normalization	 T = 23 / F1 = 0.461995	 T = 53 / F1 = 0.498867	 T = 83 / F1 = 0.45645

[5] https://en.wikipedia.org/wiki/Error_diffusion

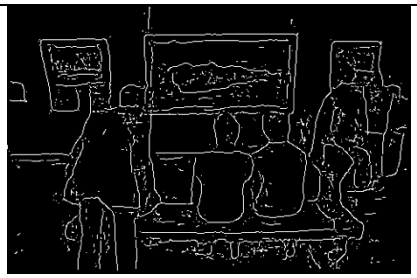
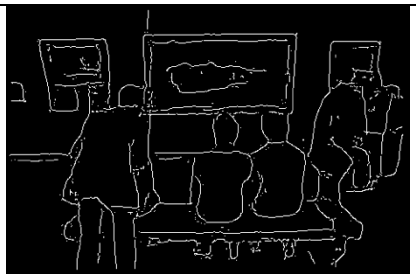
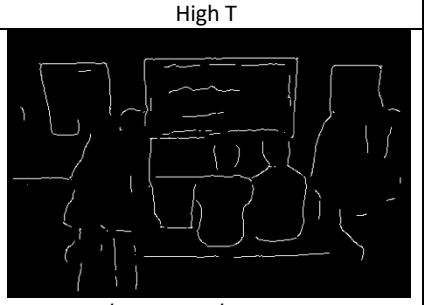
Modified Normalization	 T = 6 / F1 = 0.472722	 T = 76 / F1 = 0.499511	 T = 146 / F1 = 0.474595

Canny Edge Detector (Double Thresholding T1 = low threshold, T2 = high threshold)
Parameter search range T1 : 0 ~ 300, 10; T2 : T1+10 ~ 500, 10

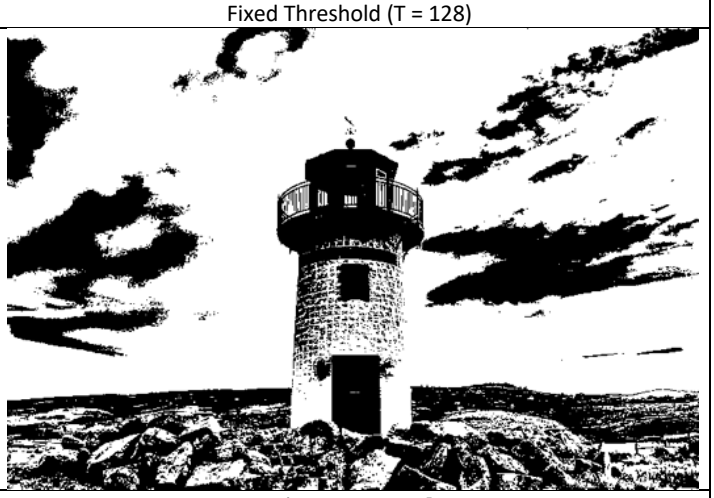
Gallery	Low T1	Best T1	High T1
Low T2	 T1/T2/F1 = 10/90/0.541589	 T1/T2/F1 = 60/90/0.712219	No Image for T1/T2 = 110/90
Best T2	 T1/T2/F1 = 10/190/0.61154	 T1/T2/F1 = 60/190/0.763522	 T1/T2/F1 = 110/190/0.742201
High T2	 T1/T2/F1 = 10/390/0.603978	 T1/T2/F1 = 60/390/0.701353	 T1/T2/F1 = 110/390/0.636272
Dogs			
Low T2	 T1/T2/F1 = 20/130/0.389411	 T1/T2/F1 = 100/130/0.43418	No Image for T1/T2 = 180/130
Best T2			

	T1/T2/F1 = 20/330/0.386864	T1/T2/F1 = 100/330/0.457369	T1/T2/F1 = 180/330/0.42066
High T2			
	T1/T2/F1 = 20/430/0.40048	T1/T2/F1 = 100/430/0.346362	T1/T2/F1 = 180/430/0.30271

Structured Edge Detector: Thresholding on (*_nms)
Parameter search range T : 0 ~ 1, 0.005

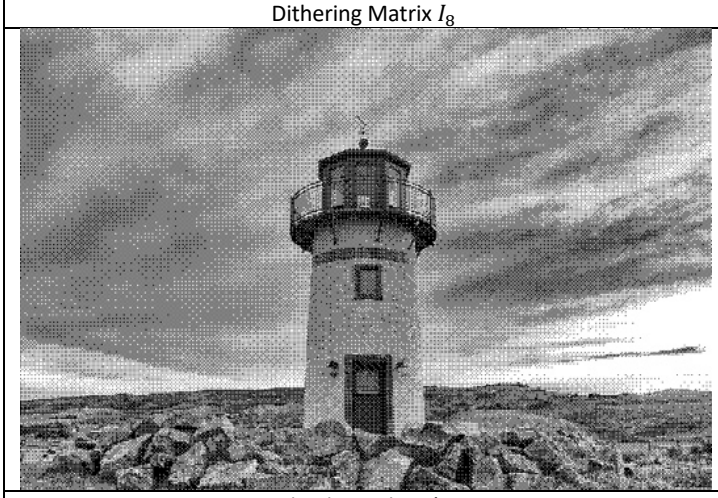
	Low T	Best T	High T
Gallery			
	T/F1 = 0.035 / 0.771287	T/F1 = 0.105/0.843988	T/F1= 0.305 / 0.662105
Dogs			
	T/F1 = 0.035/0.592911	T/F1 = 0.115/0.722879	T/F1 = 0.255/0.493074

Results for Digital Half-Toning

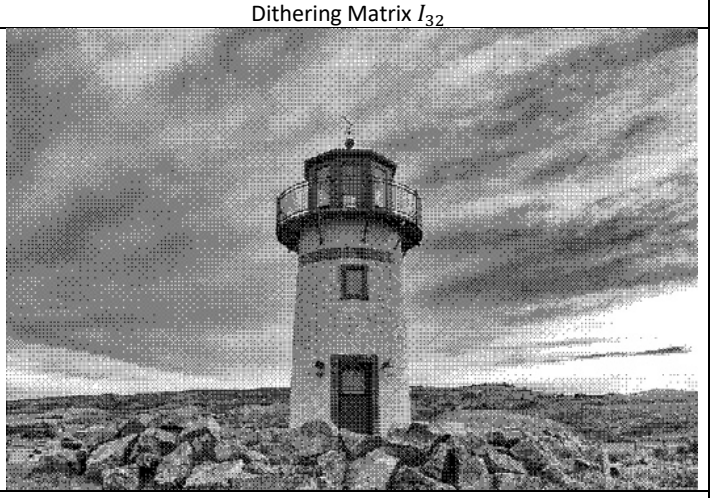
Original	Fixed Threshold (T = 128)
	
Random Threshold (Uniform)	Dithering Matrix I_2

Dithering Matrix I_8

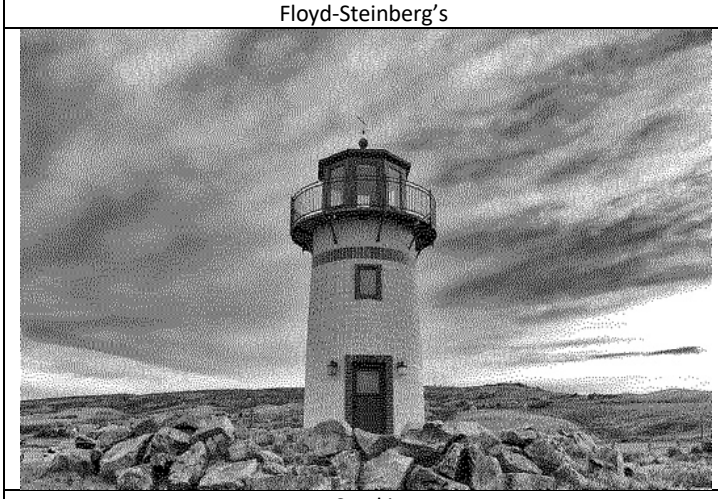
Dithering Matrix I_{32}



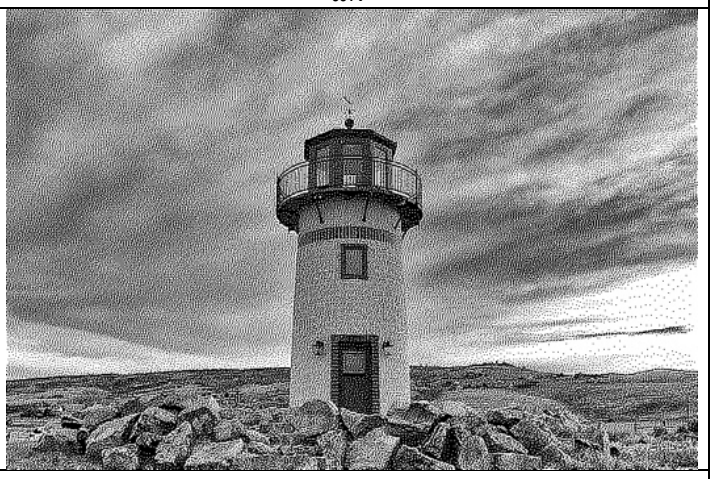
Floyd-Steinberg's

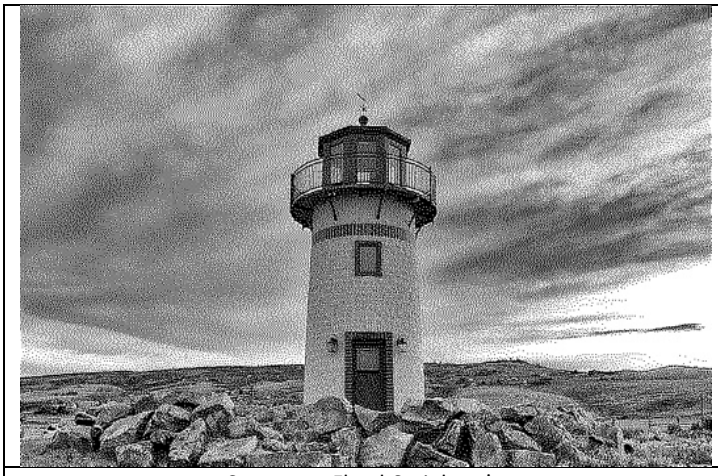


JJN



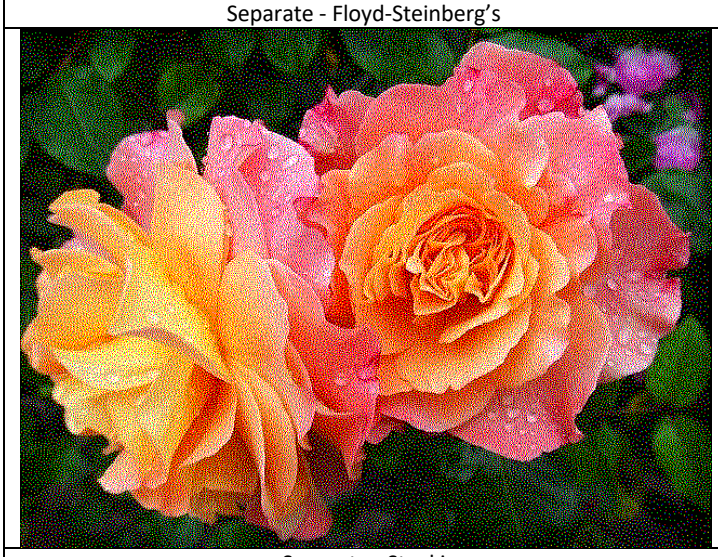
Stucki



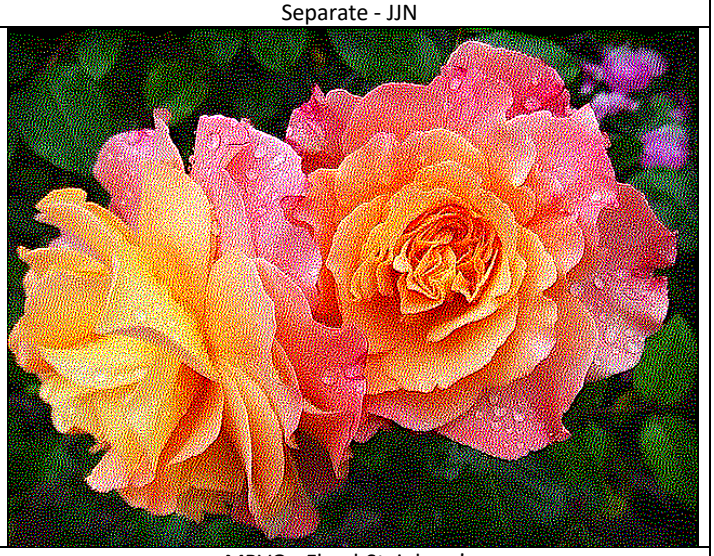


Separate - Floyd-Steinberg's

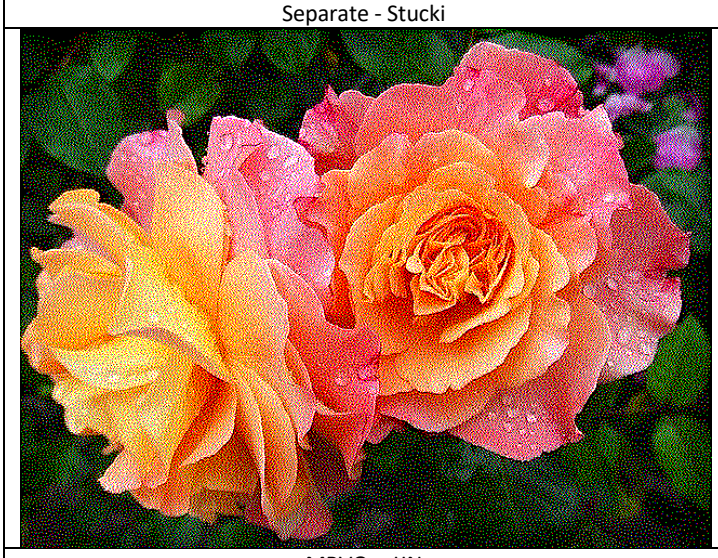
Separate - JJN



Separate - Stucki



MBVQ - Floyd-Steinberg's



MBVQ – JJN

MBVQ - Stucki