Vi Dimensions Project Report

This report enlists the work I performed during my internship at Vi Dimensions. I was slated to work on their flagship video anomaly detection model for surveillance cameras (ARVAS). Initially, I was My first few weeks involved literature review, and getting accustomed with state-of-the-art image classification models, video action classification, and object detection models. For the latter, I specifically explore background subtraction models for object detection models to improve accuracy.
Through this report, I will document the work I did at Vi Dimensions, interspersed with certain topics I gained insights in through my time there.

Model Benchmarking

I was initially asked to benchmark their statistical anomaly detection model by first visualizing the positive triggers sent by the model on an actual sample video. I took the mode of a sliding window of values, and set the window to 1 if the mode/threshold was greater than certain value.
I also used a gaussian distribution to convolve with the window, giving convolution as one uniform set of values, rather than one with dissimilar weights. Through the visualization, I realized that the data was really sparse, and a standard accuracy test would not have been a correct indicator of the performance. This led me to read and understand more about the performance scores that exist in model benchmarking:

Confusion Matrix Metrics

- True positive (*TP*): Prediction is +ve and X is diabetic, (we want that)
- True negative (*TN*): Prediction is -ve and X is healthy, (we want that too)
- False positive (*FP*): Prediction is +ve and X is healthy, (false alarm, bad)
- False negative (*FN*): Prediction is -ve and X is diabetic, (the worst)

In our case, false positives are just a false alarm. In a 2nd more detailed scan it'll be corrected. But a false negative label, this means that they think they're healthy when they're not, which is — in our problem — the worst case of the 4.
Whether *FP* & *FN* are equally bad or if one of them is worse than the other depends on your problem. This piece of information has a great impact on our choice of the performance metric, so it should be given a lot of thought before deciding

Selecting a Performance Metric

Accuracy

It's the ratio of the correctly labeled subjects to the whole pool of subjects. Accuracy is the most intuitive one.

Accuracy answers the following question: How many students did we correctly label out of all the students?

Accuracy = (TP+TN)/(TP+FP+FN+TN)

numerator: all correctly labeled subject (All trues); denominator: all subjects

Precision

Precision is the ratio of the *correctly* +ve labeled by our program to all +ve labeled.
Precision answers the following: How many of those who we labeled as diabetic are actually diabetic?

Precision = TP/(TP+FP)

numerator: +ve labeled diabetic people; denominator: all +ve labeled by our program (whether they're diabetic or not in reality).

Recall (aka Sensitivity)

Recall is the ratio of the correctly +ve labeled by our program to all who are diabetic in reality.
Recall answers the following question: Of all the people who are diabetic, how many of those we correctly predict?

Recall = TP/(TP+FN)

numerator: +ve labeled diabetic people; denominator: all people who are diabetic (whether detected by our program or not)

F1-score (aka F-Score / F-Measure)

F1 Score considers both precision and recall.
It is the harmonic mean(average) of the precision and recall.
F1 Score is best if there is some sort of balance between precision (p) & recall (r) in the system. Oppositely F1 Score isn't so high if one measure is improved at the expense of the other.
For example, if P is 1 & R is 0, F1 score is 0.

F1 Score = 2*(Recall * Precision) / (Recall + Precision)

Specificity

Specificity is the *correctly* -ve labeled by the program to all who are healthy in reality.
Specificity answers the following question: Of all the people who are healthy, how many of those did we correctly predict?

Specificity = TN/(TN+FP)

numerator: -ve labeled healthy people; denominator: all people who are healthy in reality (whether +ve or -ve labeled)

Summary

Yes, *accuracy is a great measure* but only when we have symmetric datasets (false negatives & false positives counts are close), also, false negatives & false positives have similar costs. If the

cost of false positives and false negatives are different then F1 is your savior. F1 is best if we have an uneven class distribution.
Precision is how sure you are of your true positives whilst recall is how sure you are that you are not missing any positives. We choose Recall if the idea of false positives is far better than false negatives, in other words, if the occurrence of false negatives is unaccepted/intolerable, that we'd rather get some extra false positives (false alarms) over saving some false negatives, like in our diabetes example. We'd rather get some healthy people labeled diabetic over leaving a diabetic person labeled healthy.
We choose precision if you want to be more confident of your true positives. For example, Spam emails. We'd rather have some spam emails in your inbox rather than some regular emails in our spam box. So, the email company wants to be extra sure that email *Y* is spam before they put it in the spam box and you never get to see it.
We choose Specificity if you want to cover all true negatives, meaning you don't want any false alarms, you don't want any false positives. for example, you're running a drug test in which all people who test positive will immediately go to jail, you don't want anyone drug-free going to jail. False positives here are intolerable.

Conclusion

— Accuracy value of 90% means that 1 of every 10 labels is incorrect, and 9 is correct.
— Precision value of 80% means that on average, 2 of every 10 diabetic labeled student by our program is healthy, and 8 is diabetic.
— Recall value is 70% means that 3 of every 10 diabetic people in reality are missed by our program and 7 labeled as diabetic.
— Specificity value is 60% means that 4 of every 10 healthy people in reality are miss-labeled as diabetic and 6 are correctly labeled as healthy.

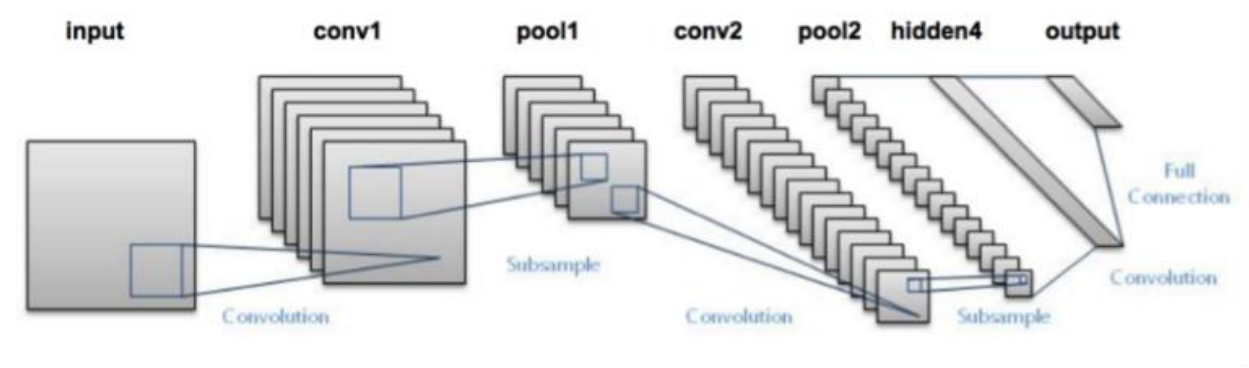
Video Action Classification

My mentor, Isamb, wanted me to understand the heuristics involved in Video Action Classifiers. And since ARVAS was at the end of the day, a video anomaly detection model, I thought it made sense to foray into understanding what Video Action Classifiers entail. Many Deep Learning articles and tutorials primarily focus on three data domains: images, speech, and text. These data domains are popular for their applications in image classification, speech recognition, and text sentiment classification. Another very interesting data modality is video. From a dimensionality and size perspective, videos are one of the most interesting data types alongside datasets such as social networks or genetic codes. Video uploading platforms such as YouTube are collecting enormous datasets, empowering Deep Learning research.
A video is really just a stack of images. I reviewed a paper on video classification research, leb by Andrej Karpathy, currently the Director of AI at Tesla.

https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/42455.pdf

IMAGE CLASSIFICATION

A video is really just a stack of images. With this in mind, I wanted to gain some hands on experience with image classifiers. I followed the following tutorial from Pyimagesearch and implemented my first deep image classification model based on LeNet: *Image classification with Keras and deep learning - PyImageSearch*

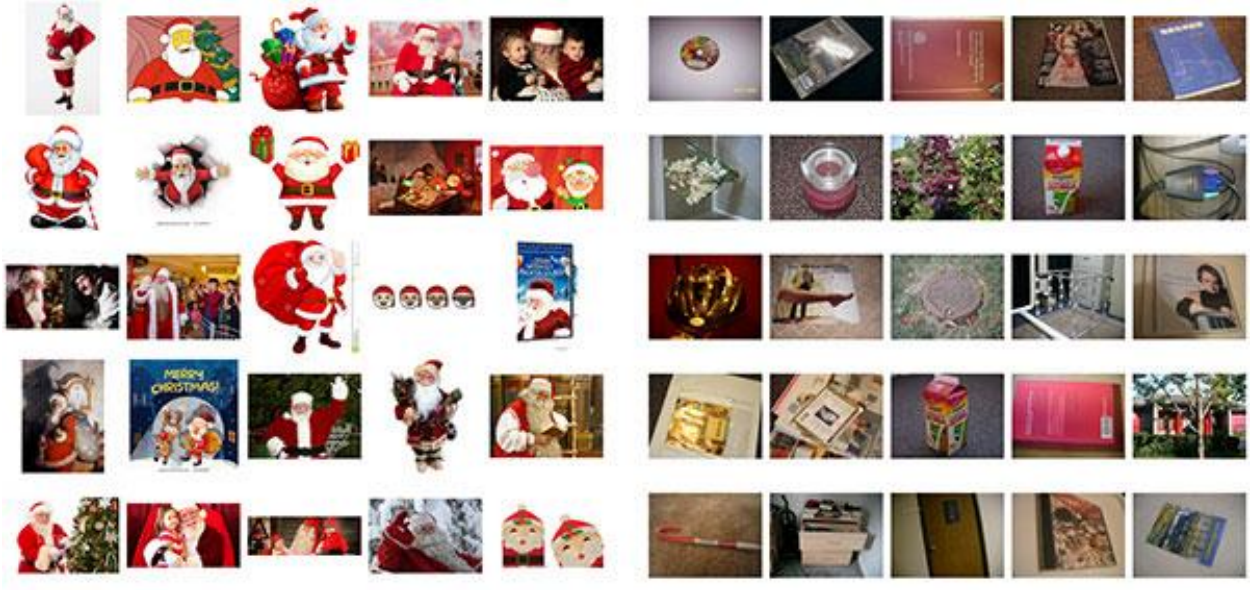


This was a binary classifier, that detected two classes:

* Images *containing* Santa ("Santa").
* Images that *do not contain* Santa ("Not Santa").

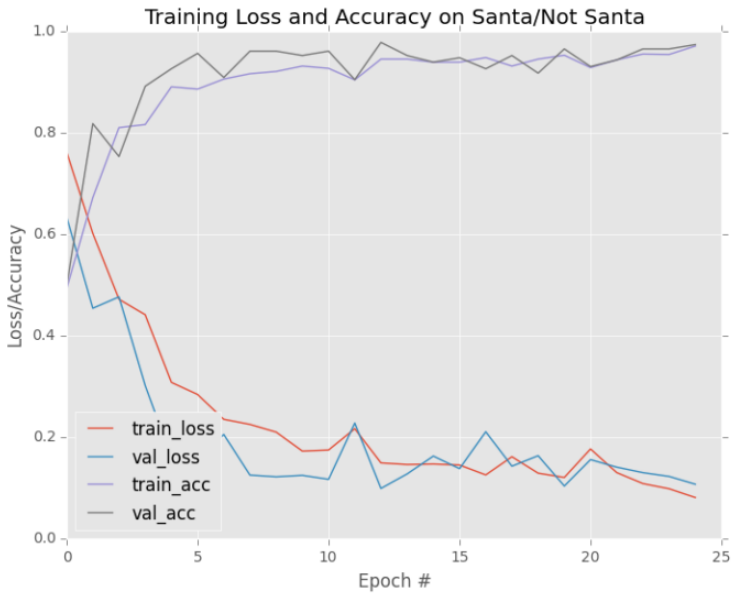
The LetNet architecture is an excellent "first image classifier" for Convolutional Neural Networks. Originally designed for classifying handwritten digits, we can easily extend it to other types of images as well.

I started by scraping training images for the model, and by querying and scraping Google, I was able to collect 461 images containing Santa (left).



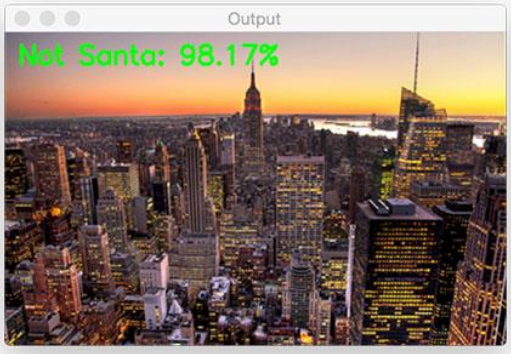
I then randomly sampled 461 images that do not contain Santa (right) from the *UKBench dataset* , a collection of ~10,000 images used for building and evaluating Content-based Image Retrieval (CBIR) systems (i.e., image search engines).

By just training the model for 25 epochs, I was able to get an accuracy of 98.2%



Evaluation

Using testing images from the same dataset, the classification model was able to distinguish the two classes with great results!



FASTER RCNN - BAGS AND PERSONS

A big problem that Vi Dimensions wanted to address was to detect unattended bags, suitcases etc. through their anomaly detection algorithm. This meant building an object detector and a classifier. Today, there are many deep object detector models - R-CNN, Fast-R-CNN and Faster R-CNN (discussed), along with other methods like Single Shot Detection (SSD). I chose to implement Faster R-CNN as it poses to have better performance over the other R-CNN models, and its implementation covered a lot of the basics of Deep Learning. The .ipynb file for the same can be found on the GitHub repository.

BACKGROUND SEGMENTATION

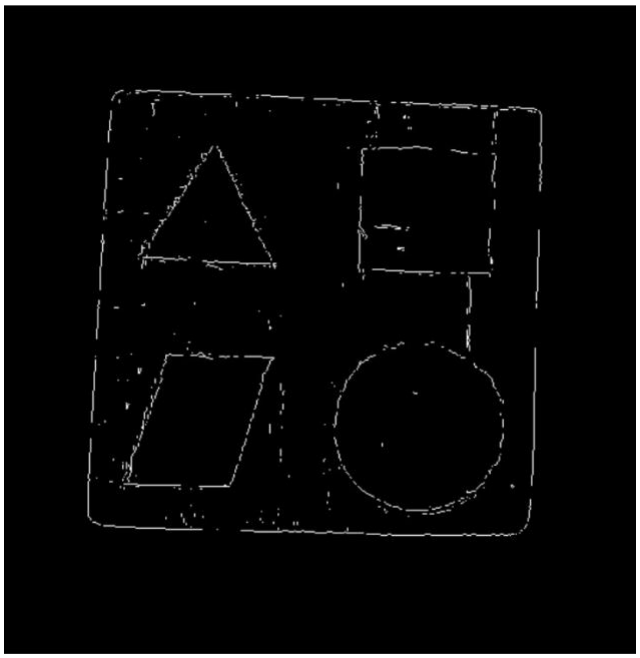
I started the second half with a more determined focus – figuring out the optimum background subtraction model for the object detection model to improve its accuracy. The following were the robustness conditions that I was supposed to follow were as follows:

Robustness to:
1. Lighting conditions
2. Noise
3. dynamic background (trees, traffic lights, waterfall etc)
4. multiple objects in the same frame (train station w/ a lot of people)

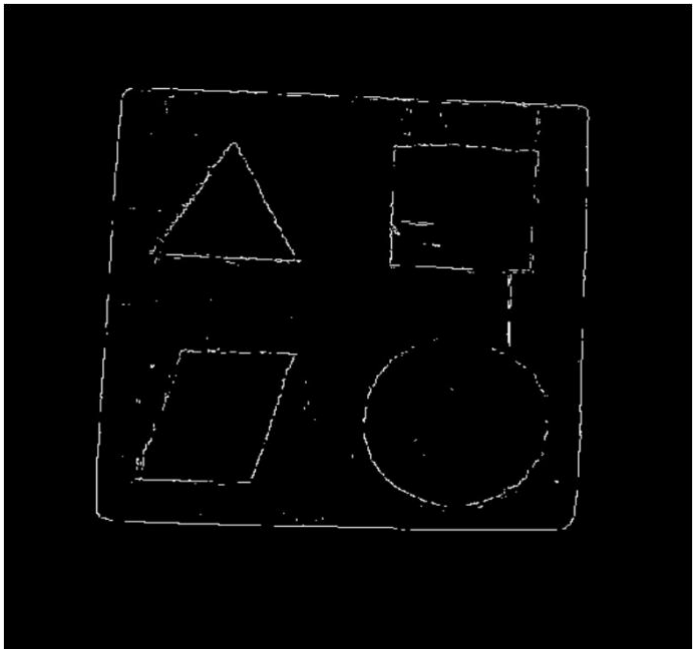
I started out with literature review and implementing some edge detectors on a sample image, paired with K-means (based on color as well as intensity). The results are as follows:
Original

Sobel Filter

Prewitt



Roberts



Canny

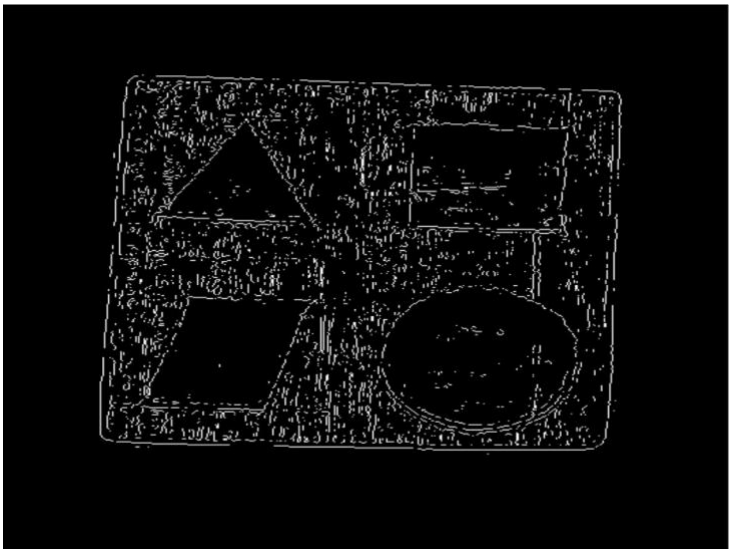
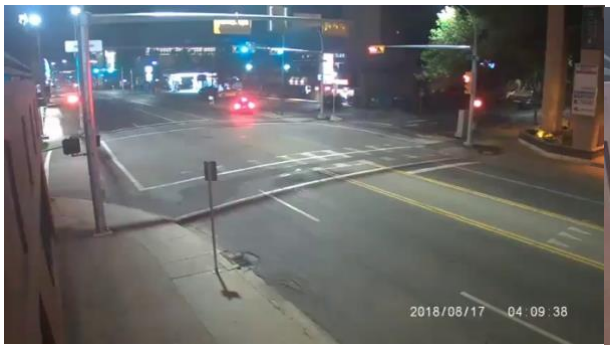


IMAGE SEGMENTATION

Night ***Day***



Parametric

- One of the models I tried on the parametric approach are the Gaussian Models (by monitoring the pixel's intensity value). These models are called so due to parameters like learning rate, changing which changed the performance of the model.
- Mixture of Gaussians (MOG):
- FGD
- CNT - faster than MOG
- KNN - (Very efficient if number of foreground pixels is low)
- GSOC - (*appeared to work best*)

A general trend I noticed with parametric models was this "ghosting" effect – where the pixels trailed off along with a moving object.

Non-parametric

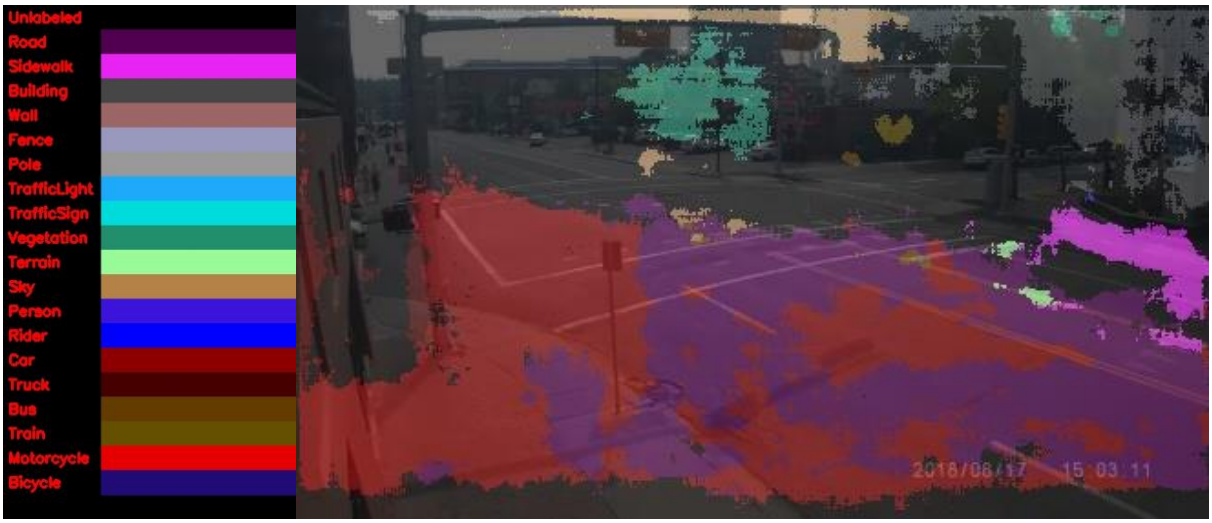
Since non-parametric means that each frame is treated independently, this meant this model was significantly faster and more sensitive to temporal changes in between the frames. The model I tested in this category was again a standard OpenCV function – GMG. One of the drawbacks for this model however was significantly greater false positive rate.

Decomposition Model

Many seminal papers on background subtraction involve decomposing the image frames into low-rank sparse matrix and then removing the component corresponding to the background. Using the 'lrslibrary', I tested the performance of the R-PCA segmentation model.

Deep models

While these are highly impractical to be deployed on a large scale (since we'd require a lot of parallel compute power), I still wanted to test some deep models for semantic segmentation. I explored a very popular model for self-driving cars, ENet, trained on the Cityscapes dataset. Since this model was trained on videos from a car dashboard (cityscapes dataset), and because of lack of datasets for surveillance cameras, this model didn't perform as well.



Hough Transform

Roberto suggested that a better way to segment images (subtracting background) could be though Hough Transform. Since most of the static objects in a video stream are typically manmade objects - like buildings, roads, traffic signs etc. This means that they would be highly geometric in nature (circles and straight lines), and thus a model like Hough Transform would be robust to other objects like noise and shadows, besides giving better performance with limited compute resources.

The Hough transform is a feature extraction technique used in image analysis, computer vision, and digital image processing. The purpose of the technique is to find imperfect instances of objects within a certain class of shapes by a voting procedure. This voting procedure is carried out in a parameter space, from which object candidates are obtained as local maxima in a so-

called accumulator space that is explicitly constructed by the algorithm for computing the Hough transform.
In Hough transform, the points are linked by determining first if they lie on the curve of specified shape. Unlike the local analysis method, where given n points of an image are taken into consideration. Suppose we want to find the subset of these points that lie on the straight lines. One possible solution is to find all the lines determine by every pair of points and then find all subsets of points that are close to particular lines. The problem with this procedure is that it involves finding n (n-1)/2 ~ n^2 lines and then performing (n) (n (n-1)) ~ n^3 comparisons of every point to all lines. This approach is computationally prohibitive in all but in most the trivial applications. The Hough transform is a method that, in theory, can be used to find features of any shape in an image. In practice it is only generally used for finding straight lines or circles. The computational complexity of the method grows rapidly with more complex shapes. Assume we have some data points in an image which are perhaps the result of an edge detection process, or boundary points of a binary blob. We wish to recognize the points that form a straight line. Consider a point (xi , yi) in the image.

The general equation of a line is

$$y = ax + b.$$

There are infinitely many lines that pass through this point, but they all satisfy the condition

$$y_i = ax_i + b$$

For varying a and b. We can rewrite this equation as

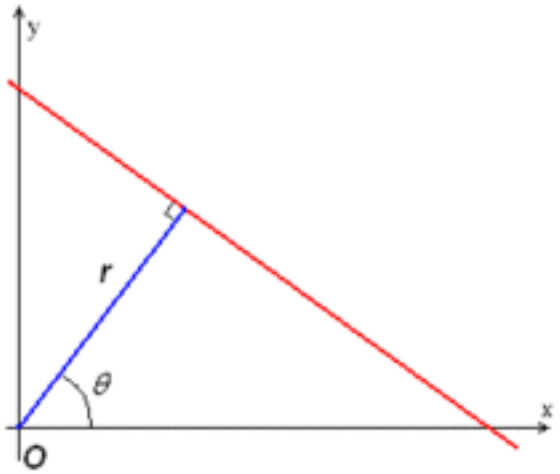
$$b = -x_i a + y_i$$

And plot the variation of a and b. If we divide parameter space into a number of discrete accumulators cells, we can collect `votes' in a b space from each data point in x y space. Peaks in a b space will mark the equations of lines of co-linear points in x y space.

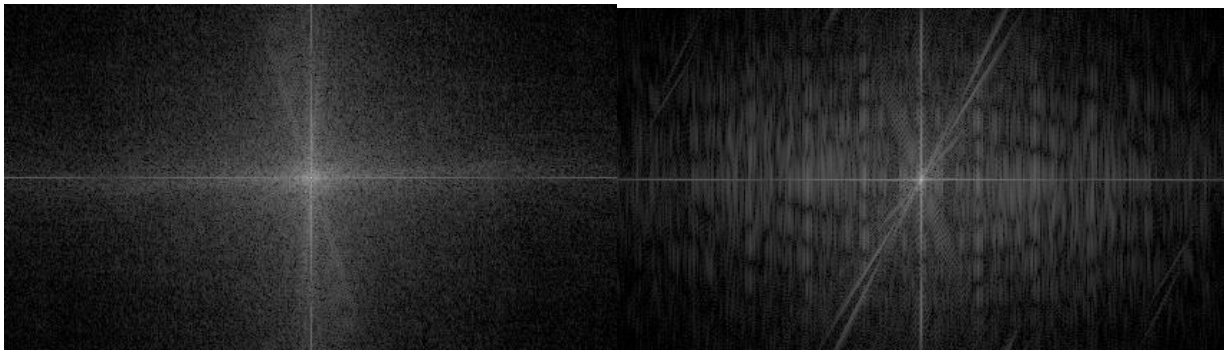
The simplest case of Hough transform is detecting straight lines. In general, the straight line $y = mx + b$ can be represented as a point (b, m) in the parameter space. However, vertical lines pose a problem. They would give rise to unbounded values of the slope parameter m. Thus, for computational reasons, Duda and Hart proposed the use of the Hesse normal form:

$$r = x \cos\theta + y \sin\theta$$

where r is the distance from the origin to the closest point on the straight line, and θ (*theta*) is the angle between the x axis and the line connecting the origin with that closest point.
It is therefore possible to associate with each line of the image a pair (r, θ). The (r, θ) plane is sometimes referred to as *Hough space* for the set of straight lines in two dimensions. This representation makes the Hough transform conceptually very close to the two-dimensional Radon transform. (They can be seen as different ways of looking at the same transform.)



(https://en.wikipedia.org/wiki/Hough_transform)



Distance from Center vs Angle for Image Spectral Domain

Given above is an example showing the results of a Hough transform on the traffic intersection picture. The results of this transform were stored in a matrix. Cell value represents the number of curves through any point. Higher cell values are rendered brighter. The distinctly bright spots are the Hough parameters of the lines in the image. From these spots' positions, angle and distance from image center of the lines in the input image can be determined.