AMRITA SCHOOL OF ENGINEERING,
BANGALORE

# FOREGROUND AND BACKGROUND SEPERATION USING TRUNCATED SVD AND ROBUST PCA

A PROJECT REPORT

***Submitted by***


BL.EN.U4AIE19052 Punitha Vancha

BL.EN.U4AIE19054 Ruthvik K

BL.EN.U4AIE19057 Samudyatha Devalla


***for the course***
***19MAT212-MATHEMATICS FOR INTELLIGENT SYSTEMS -4***

Guided and Evaluated by

Dr. Murali K.
Dr. Subramani R.

# ABSTRACT

We will be studying an application of Robust Principal component Analysis (PCA) and Singular Value Decomposition (SVD) in image processing. Our project is about separation of foreground and the background using SVD and Robust Principal component Analysis. Foreground and background separation plays an important role in computer vision systems, which includes motion capture, action recognition, video compressing, surveillance tracing and teleconferencing. The input is a video of mp4 or avi format. The video contains a stationary background and a moving foreground. The best example of such video is a surveillance video. This video will be processed into different frames. Frames are instances of the video at a certain time. These frame are converted into matrices using which we differentiate between the stationary and moving components in the frame comparing to other frames and checking the differences to separate the foreground and background using Principal component Analysis and Singular Value Decomposition. In simple words, if we think of a frame from the video to be a matrix, we are splitting that matrix into 2 matrices whose sum gives the initial matrix. When we use SVD, we split the matrix into a low rank background matrix and a high rank foreground matrix. When we use robust PCA, we split our data matrix into a Low rank background matrix and a Sparse foreground matrix.

# INTRODUCTION

We need to understand what is SVD, PCA and other basic concepts used in this project. So, Let's see about them in brief below.

## SINGULAR VALUE DECOMPOSITION (SVD):

Singular Value Decomposition is one of the most important matrix factorization methods which is a foundation for several other data methods including PCA which are used world-wide. It decomposes a matrix into three numerically stable matrices which contain the column space, singular values and row space respectively. Singular Value Decomposition is used to obtain low – rank approximations to matrices and can be used to decompose high-dimensional data matrices into lower dimensional matrix containing statistically most important factors.

## PRINCIPAL COMPONENT ANALYSIS:

Principal Component Analysis (PCA) is one of the powerful tool for analysing data and a dimensionality reduction technique. It is a statistical representation of Singular Value Decomposition. It gives us a hierarchical coordinate system based on data to represent the statistical variations in dataset. PCA helps us to reduces dimensions of the data by find patterns in the data and compressing it so that important information is not lost. In simple words, it seeks the best rank-r estimate L of M minimizing $\|M - L\|$ using truncated SVD. It has several applications in image compression and image processing.

**ROBUST PCA:**

Principle Component Analysis is fragile and sensitive to outliers, data corruption and noise. It can work with white noise but when there are outliers and highly corrupted data, it cannot be used effectively. Therefore we use Robust PCA which works efficiently with noisy and corrupted data. Robust PCA decomposed a data matrix into a low rank matrix and a sparse matrix which contains the outliers. The principal components in the low rank matrix are robust to the outliers in the sparse matrix. It has been used in video surveillance, face recognition, natural language procession and several other applications.

# CONCEPTUAL

Now let's look at the maths behind these concepts and how they can be used for image processing.

**SINGULAR VALUE DECOMPOSITION (SVD):**

SVD of a data Matrix X of size $m \times n$ where m>>n, gives three matrices U, Σ and V.

$$\text{SVD(X)} = \text{U } \Sigma \ V^T$$

$$\begin{bmatrix} | & | & \dots & | \\ | & | & \dots & | \\ | & | & \dots & | \\ x_1 & x_2 & \dots & x_n \\ | & | & \dots & | \\ | & | & \dots & | \end{bmatrix} = \begin{bmatrix} | & | & \dots & | \\ | & | & \dots & | \\ | & | & \dots & | \\ u_1 & u_2 & \dots & u_m \\ | & | & \dots & | \\ | & | & \dots & | \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & \sigma_2 & : & : \\ | & 0 & : & : \\ | & : & | & \sigma_n \\ | & | & | & | \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \text{———} & v_1 & \text{—} \\ \text{———} & v_2 & \text{—} \\ \text{———} & : & \text{—} \\ & : & \\ & & \\ \text{———} & v_n & \text{—} \end{bmatrix}$$

$m \times n$ $\quad$ $m \times m$ $\quad$ $n \times m$ $\quad$ $n \times n$

Here U is an $m \times m$ matrix which contains information about the column space of matrix X. Σ is a diagonal matrix of size $n \times m$ and tells us how important carious columns of U or V are. It contains the singular values in the order of priority in the diagonal elements. V is a $n \times n$ matrix and contains information about the row space of the matrix X. The columns in the matrices U and V are also in the decreasing order of importance and this importance is encoded in the singular values in Σ. When we find an approximation to X using these matrices U, Σ and V we get a matrix with lower rank which is minimum of m,n. when we multiply these matrices together we get a sum of individual matrices of rank 1 based on their priorities.

$$\text{U } \Sigma \ V^T = \sigma_1 u_1 v_1 + \sigma_2 u_2 v_2 + \sigma_3 u_3 v_3 \dots\dots + \sigma_n u_n v_n = \hat{U}\hat{\Sigma}V^T$$

Here $\widehat{U}$ has only the first n columns and $\hat{\Sigma}$ is an $n \times n$ matrix. So this is called economical Singular Value Decomposition where we get a lower rank approximation of X by multiplying U $\Sigma$ $V^T$. If we want to truncate this further and obtain fewer components we use truncated Singular Value Decomposition. In truncated Singular Value Decomposition based on the required rank r (which is lower than n) we approximate the matrix X by summing only the first r rank-1 matrices and ignoring the rest. This gives the best approximation to X with rank r.

Truncated SDV: $X \sim \sigma_1 u_1 v_1 + \sigma_2 u_2 v_2 + \sigma_3 u_3 v_3 \ldots\ldots + \sigma_r u_r v_r = \widetilde{U}\widetilde{\Sigma}\widetilde{V^T}$

**RANDOMIZED SINGULAR VALUE DECOMPOSITION (SVD):**

Even though Singular Value Decomposition is a fantastic tool for reducing data into a low dimensional representation for extracting key features and patterns, it can be quite expensive. So, Randomized Singular Value Decomposition gives a faster and cheaper way of finding approximation by truncated Singular Value Decomposition for a large data matrix by using randomization to speed up the computations. In this method, we randomly sample the column space of data matrix X. We first take a projection matrix P such that P$\epsilon$ $\mathbb{R}^{m \times r}$ where r is the target rank. We then compute

$$Z = XP$$

This step will shrink the column space from n columns to r columns. So Z is of size $n \times r$. Because of the randomness dominant column space of X is captured in Z with high probability. Then we can compute the QR decomposition of matrix Z which will give us an orthonormal basis for X.

$$Z = QR$$

Where Q is a low rank orthogonal matrix of size $m \times m$ and R is an upper triangular matrix of size $m \times r$. Next step is to compute

$$Y = Q^T X$$

Now we compute Singular value decomposition of Y = $U_Y \Sigma V^T$

Since Q is orthonormal and has the dominant column space of X, $\Sigma$ and V are same for X and Y. to get $U_X$ we multiply Q and $U_Y$.

$$U_X = Q U_Y$$

This is how we compute singular value decomposition at lower cost. When we apply randomized SVD on our data matrix containing images from the video and multiply the three matrices obtained from it, we are actually computing the low rank approximation of it which gives the background of the images. It gives the background because the background is almost constant and hence the number of independent columns in the matrix containing the background are as less as 1 or 2. So by applying randomized SVD for obtaining 2

components or 1 component can give the low rank background matrix. The foreground can be computed by subtracting this low rank approximation from the original data matrix. Since there is movement in the foreground, it is a high rank matrix.

**ROBUST PCA:**

Robust principal component analysis decomposes X into a low rank matrix L and a Sparse matrix S such that

$$X = L + S$$

In our project, L is the data matrix representing the background of the image and S is the matrix containing information about the foreground. So, our problem is to minimize [ rank(L) ] and minimize $L_0$ norm of S subject to X= L+S.

This problem can have infinitely many solutions so we can add a penalty term to promote only a single solution. But minimizing rank of L and minimizing $L_0$ norm of S both of them are non-convex thus making this problem a highly non-convex optimization problem which cannot be solved by computational mean efficiently. Therefore, we introduce convex relaxation by using proxies for the two properties.

rank(L) → $\|L\|_*$ (nuclear norm which is the sum of singular values as proxy for rank)

$\|S\|_0$ → $\|S\|_1$ ( $L_1$ norm which is sum of absolute values of entries as a proxy for $L_0$ $norm$)

We will use an algorithm called principal component pursuit and now our problem is to optimize :

$$\min\|L\|_* + \lambda\|S\|_1 \quad \text{subject to L + S = X}$$

this can be solved using Augmented Lagrange Multipliers method using a more special case of ALM called as Alternating directions method which helps in a faster convergence with high accuracy. ALM is used in this problem because it requires no tuning of parameters and at the same time maintains the rank of the iterates same as the initial rank and hence is highly efficient. So the problem now is:

$$l(L, S, Y) = \|L\|_* + \lambda\|S\|_1 + \langle Y, M - L - S\rangle + \frac{\mu}{2}\|M - L - S\|_F^2$$

In this problem in general we can solve it by repeatedly updating

($L_k$ , $S_k$ ) = arg $min_{L,S}$ l( L, S, $Y_k$ ) and then updating $Y_{k+1} = Y_k + \mu(M - L_k - S_k)$ till the solution converges. But for our low rank and sparse decomposition we can say that $min_L$ $l$(L, S, Y ) and $min_S$ l(L, S, Y ) have very simple and efficient solutions so we can avoid having to solve a series of convex problems. We can estimate L by fixing S and we can estimate S by fixing L. We will use a shrinkage operator which will help us take any singular value that are less than $\tau$ (tau) and round them to zero so as to ignore the smaller singular values and truncate the matrix. So for the sparse matrix approximation we apply the shrinkage operator, remove all the smaller singular values for the low rank approximations we have singular value thresholding were we take the SVD, do the shrinkage operation on the

singular values and multiply back together to reconstruct the matrix. So the approximation for the sparse matrix is to apply the shrinkage operator while the alternating aspect is to estimate the low rank matrix L and subtract the sparse matrix from the full matrix M while keeping track of the error. And then again we alternate back to estimating the sparse matrix that is the full rank matrix M subtracted by the low rank matrix. Thus we will estimate alternatingly both the matrices L and S.

# IMPLEMENTATION

```
import moviepy.editor as mpe

from glob import glob


import sys, os

import numpy as np

import scipy

from scipy import misc

#%matplotlib inline

import matplotlib.pyplot as plt


video = mpe.VideoFileClip("download.mp4")


video.subclip(0,7).ipython_display(width=300)


video.duration


(b,l)=video.size


def create_data_matrix_from_video(clip, k, scale):

    return np.vstack([scipy.misc.imresize(grayscale(clip.get_frame(i/float(k))).astype(int), scale).flatten() for i in range(k * int(clip.duration))]).T
```

```
def grayscale(x):
    return np.dot(x[...,:3], [0.299, 0.587, 0.114])

scale = 6.25   # to change resolution of image

dims = (int(l * (scale/100)), int(b * (scale/100)))

!pip3 install --user scipy==1.2.0


M = create_data_matrix_from_video(video, 100, scale)


print(dims, M.shape)

plt.figure(figsize=(12, 12))
plt.imshow(M, cmap='gray')

from sklearn import decomposition
u, s, v = decomposition.randomized_svd(M, 2)
u.shape, s.shape, v.shape


low_rank = u @ np.diag(s) @ v
low_rank.shape


plt.figure(figsize=(12, 12))
plt.imshow(low_rank, cmap='gray')
```

```
#original
plt.imshow(np.reshape(M[:,340], dims), cmap='gray');



plt.imshow(np.reshape(low_rank[:,340], dims), cmap='gray');

plt.imshow(np.reshape(M[:,340] - low_rank[:,340], dims), cmap='gray');

u, s, v = decomposition.randomized_svd(M, 1)
u.shape, s.shape, v.shape


low_rank = u @ np.diag(s) @ v
low_rank.shape



plt.figure(figsize=(12, 12))
plt.imshow(low_rank, cmap='gray')

plt.imshow(np.reshape(M[:,550] - low_rank[:,550], dims), cmap='gray');

plt.imshow(np.reshape(M[:,340] - low_rank[:,340], dims), cmap='gray');

!pip3 install fbpca

from scipy import sparse
from sklearn.utils.extmath import randomized_svd
```

```
import fbpca

T=1e-9

def converged(Z, F_norm):
    err = np.linalg.norm(Z, 'fro') / F_norm
    print('Error: ', err)
    return err < T

 #Shrinkage operator for M = sign(M) max(|M| − τ, 0)
def shrink(M, tau):
    S = np.abs(M) - tau
    return np.sign(M) * np.where(S>0, S, 0)

#truncated SVD
def T_svd(M, rank): return fbpca.pca(M, k=min(rank, np.min(M.shape)), raw=True)



def sigma1(M): return T_svd(M, 1)[1][0]
#return the largest singular value

#singular value threshold operator
def svd_reconstruct(M, rank, min_sv):
    u, s, v = T_svd(M, rank)
    s -= min_sv
    n = (s > 0).sum()
    # number of singular values > 1/mu is returned as n
    return u[:,:n] @ np.diag(s[:n]) @ v[:n], n
```

```
def pcp(X, maxiter=10, k=10):
    m, n = X.shape
    trans = m<n
    if trans: X = X.T; m, n = X.shape

    # BEST CHOICE FOR λ = 1/sqrt (max(m, n) ) since m >> n:
    lamda = 1/np.sqrt(m)
    sig1 = sigma1(X)
    Y = np.copy(X) / max(sig1, np.linalg.norm( X, np.inf) / lamda)
    mu = k*1.25/sig1;
    mu_bar = mu * 1e7; rho = k * 1.5

    F_norm = np.linalg.norm(X, 'fro')
    L = np.zeros_like(X); sv = 1

    examples = []

    for i in range(maxiter):
        print("rank :", sv)
        X2 = X + Y/mu

        # update estimate of Sparse Matrix by shrinking original - low-rank matrix
        S = shrink(X2 - L, lamda/mu)

        # update estimate of Low-rank Matrix by doing truncated SVD of rank sv &
reconstructing.

        L, svp = svd_reconstruct(X2 - S, sv, 1/mu)

        # If svp < sv, we already have enough singular values. If not, add 5% (in this case 240)
to sv
```

```
        sv = svp + (1 if svp < sv else round(0.05*n))


        Z = X - L - S
        Y += mu*Z; mu *= rho

        examples.extend([S[340,:], L[340,:]])

        if m > mu_bar: m = mu_bar
        if converged(Z, F_norm): break

    if trans: L=L.T; S=S.T
    return L, S, examples

#5 % of the original rank
m, n = M.shape
round(m * .05)

L, S, examples =  pcp(M, maxiter=5, k=10)

def plots(ims, dims, figsize=(15,20), rows=1, interp=False, titles=None):
    if type(ims[0]) is np.ndarray:
        ims = np.array(ims)
    f = plt.figure(figsize=figsize)
    for i in range(len(ims)):
        sp = f.add_subplot(rows, len(ims)//rows, i+1)
        sp.axis('Off')
        plt.imshow(np.reshape(ims[i], dims), cmap="gray")
```

```
plots(examples, dims, rows=5)


f = plt_images(M, S, L, [340], dims)


def plt_images(M, A, E, index_array, dims, filename=None):
    f = plt.figure(figsize=(15, 10))
    r = len(index_array)
    pics = r * 3
    for k, i in enumerate(index_array):
        for j, mat in enumerate([M, A, E]):
            sp = f.add_subplot(r, 3, 3*k + j + 1)
            sp.axis('Off')
            pixels = mat[:,i]
            if isinstance(pixels, scipy.sparse.csr_matrix):
                pixels = pixels.todense()
            plt.imshow(np.reshape(pixels, dims), cmap='gray')
    return f

#to see a couple of sample outputs - the original, foregrounda n background
f = plt_images(M, S, L, [0, 100, 1000], dims)
```

# OUTPUTS

## SAMPLE VIDEO 1

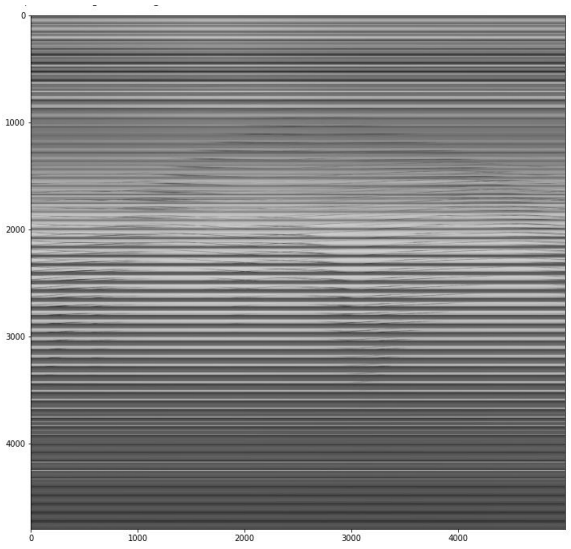


Fig: original data matrix

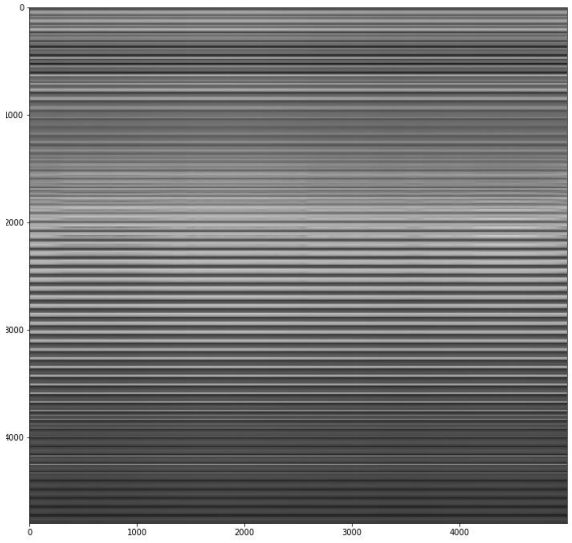


fig: low rank approximation by SVD of M which contains the foreground.

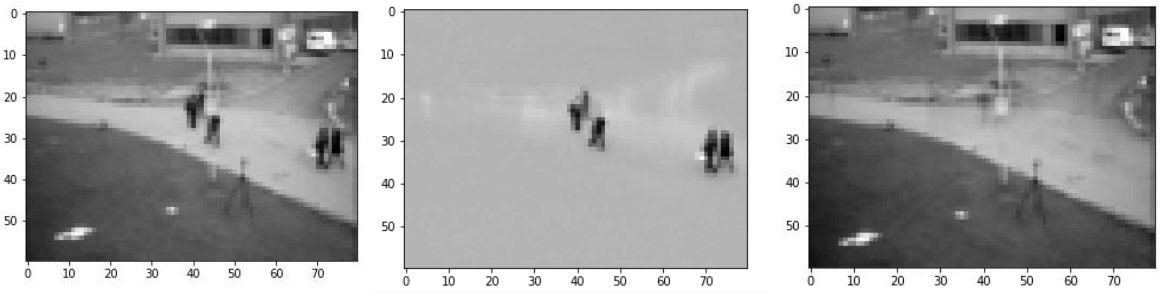


Fig: original sample frame ; foreground ; background after SVD

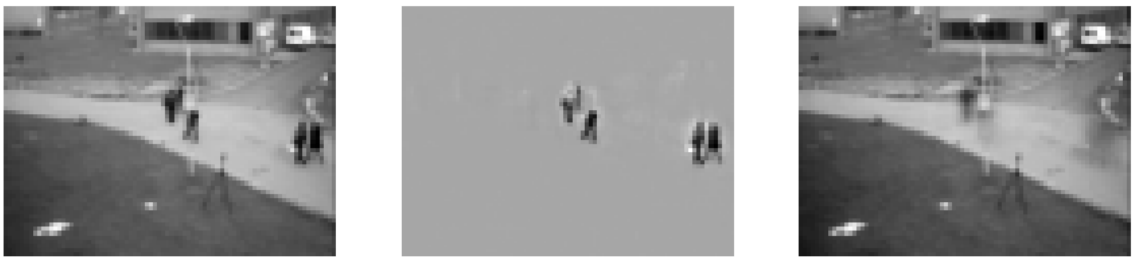

Fig: separation by robust PCA.

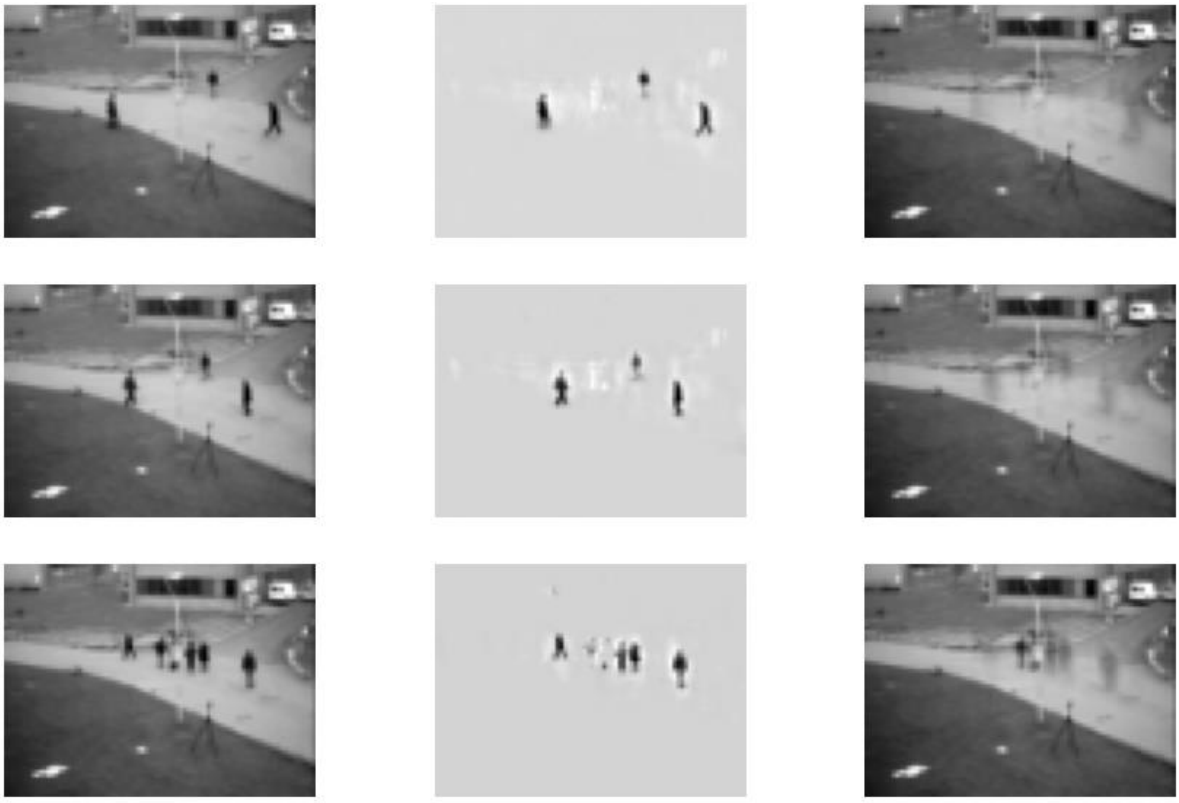

Fig : few sample frames processed by robust PCA

## SAMPLE VIDEO 2

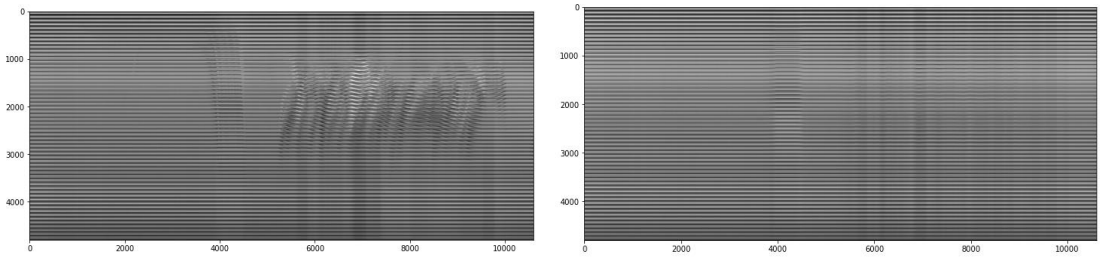


Fig: original and low rank matrix for video sample2.

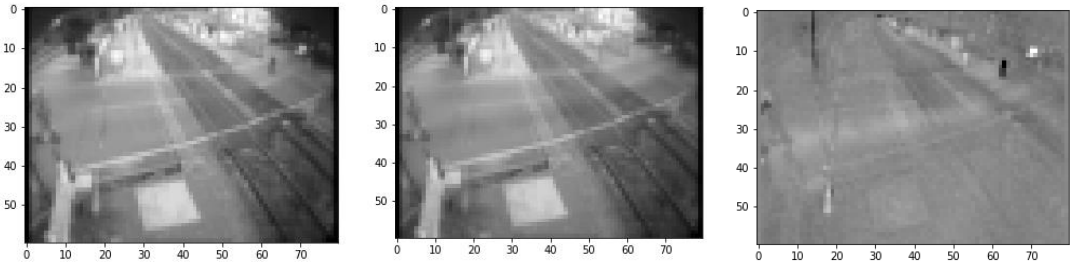


Fig: original sample frame ; foreground ; background after SVD

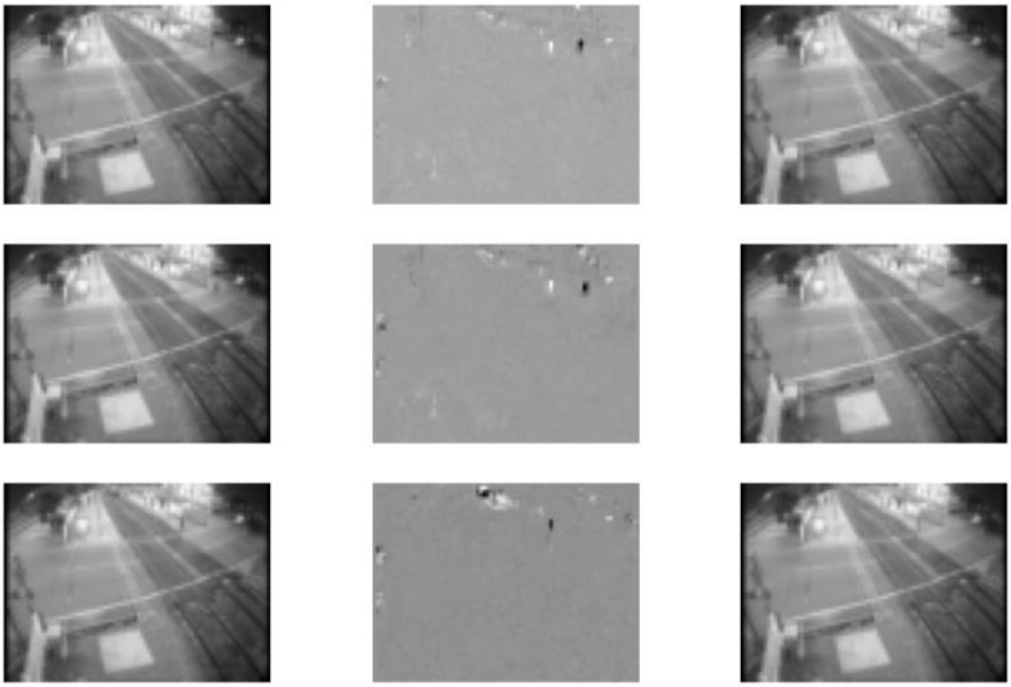

Fig : few sample frames processed by robust PCA

## CONCLUSION

We can use Randomized truncated Singular Value Decomposition for actively computing the background information only. We can get the foreground by subtracting the low rank background matrix from the original matrix. Whereas Robust Principal Component Analysis can be used to find both the background and foreground directly from the decomposition. Principle component pursuit is used for Robust PCA because it maintains the rank of the initial low rank estimate throughout the iterations while other algorithms such as Accelerated Proximal Gradient method cannot do. It also helps us to get the convergence in fewer iterations and is less costly. So this is how we separated foreground from the background in an image using Randomized SVD and Robust PCA.

## REFERENCES


https://www.sciencedirect.com/science/article/pii/S2405959515000041

https://arxiv.org/pdf/0912.3599.pdf

https://arxiv.org/pdf/1608.02148.pdf

https://arxiv.org/pdf/1810.06860.pdf