NAME - PRAKHAR BHARDWAJ

ANDREW ID - prakharb

16720 (B) Object Tracking in Videos - Assignment 6

Instructor: Kris heng-Yu, Jinkun TAs: Arka, Rohan, Rawal, S

Instructions

This section should include the visualizations and answers to specifically highlighted questions from Q1 to Q3. This section will need to be uploaded to gradescope as a pdf and manually graded (this is a separate submission from the coding notebooks)

1. Students are encouraged to work in groups but each student must submit their own work. Include the names of your collaborators in your write up. Code should Not be shared or copied. Please properly give credits to others by LISTING EVERY COLLABORATOR in the writeup including any code segments that you discussed, Please DO NOT use external code unless permitted. Plagiarism is prohibited and may lead to failure of this course.
2. **Start early!** This homework will take a long time to complete.
3. **Questions:** If you have any question, please look at Piazza first and the FAQ page for this homework.
4. All the theory question and manually graded questions should be included in a single writeup (this notebook exported as pdf or a standalone pdf file) and submitted to gradescope: pdf assignment.
5. **Attempt to verify your implementation as you proceed:** If you don't verify that your implementation is correct on toy examples, you will risk having a huge issue when you put everything together. We provide some simple checks in the notebook cells, but make sure you verify them on more complicated samples before moving forward.

6. **Do not import external functions/packages other than the ones already imported in the files:** The current imported functions and packages are enough for you to complete this assignment. If you need to import other functions, please remember to comment them out after submission. Our autograder will crash if you import a new function that the gradescope server does not expect.
7. Assignments that do not follow this submission rule will be **penalized up to 10% of the total score**.

Preliminaries

In this section, we will go through some of the basics of the Lucas-Kanade tracker and the Matthews-Baker tracker. The following table contains a summary of the variables used in the rest of the assignment.

Symbol	Vector/Matrix Size	Description
u	1×1	Image horizontal coordinate
v	1×1	Image vertical coordinate
$\mathbf{x}$	2×1 or 1×1	pixel coordinates: (u, v) or unrolled
$\mathbf{I}$	$m \times 1$	Image unrolled into a vector (m pixels)
$\mathbf{T}$	$m \times 1$	Template unrolled into a vector (m pixels)
$\mathbf{W(p)}$	3×3	Affine warp matrix
$\mathbf{p}$	6×1	parameters of affine warp
$\frac{\partial \mathbf{I}}{\partial u}$	$m \times 1$	partial derivative of image wrt u
$\frac{\partial \mathbf{I}}{\partial v}$	$m \times 1$	partial derivative of image wrt v
$\frac{\partial \mathbf{T}}{\partial u}$	$m \times 1$	partial derivative of template wrt u
$\frac{\partial \mathbf{T}}{\partial v}$	$m \times 1$	partial derivative of template wrt v
$\nabla \mathbf{I}$	$m \times 2$	image gradient $\nabla \mathbf{I}(\mathbf{x}) = \left[\frac{\partial \mathbf{I}(\mathbf{x})}{\partial u} \frac{\partial \mathbf{I}(\mathbf{x})}{\partial v}\right]$
$\nabla \mathbf{T}$	$m \times 2$	image gradient $\nabla \mathbf{T}(\mathbf{x}) = \left[\frac{\partial \mathbf{T}(\mathbf{x})}{\partial u} \frac{\partial \mathbf{T}(\mathbf{x})}{\partial v}\right]$
$\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$	2×6	Derivative of affine warp wrt its parameters
$\mathbf{J}$	$m \times 6$	$\nabla \mathbf{I} \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ or $\nabla \mathbf{T} \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$
$\mathbf{H}$	6×6	$\mathbf{J}^T \mathbf{J}$

Template

A template describes the object of interest (eg. a car, football) which we wish to track in a video. Traditionally, the tracking algorithm is initialized with a template, which is represented by a bounding box around the object to be tracked in the first frame of the video. For each of the

subsequent frames in the video, the tracker will update its estimate of the object in the image. The tracker achieves this by updating its affine warp.

Warps

What is a warp? An image transformation or warp $\mathbf{W}$ is a function that acts on pixel coordinates $\mathbf{x} = [u \quad v]^T$ and maps pixel values from one place to another in an image $\mathbf{x}' = \begin{bmatrix} u' & v' \end{bmatrix}^T$. Simply put, $\mathbf{W}$ maps a pixel with coordinates $\mathbf{x} = [u \quad v]^T$ to $\mathbf{x}' = \begin{bmatrix} u' & v' \end{bmatrix}^T$. Translation, rotation, and scaling are all examples of warps. We denote the parameters of the warp function $\mathbf{W}$ by $\mathbf{p}$:

$$\mathbf{x}' = \mathbf{W}(\mathbf{x}; \mathbf{p})$$

Affine Warp

An affine warp is a particular kind of warp that can include any combination of translation, scaling, and rotations. An affine warp can be represented by 6 parameters $\mathbf{p} = [p_1 \ p_2 \ p_3 \ p_4 \ p_5 \ p_6]^T$. One of the most convenient things about an affine warp is that it is linear; its action on a point with coordinates $\mathbf{x} = [u \quad v]^T$ can be described as a matrix operation by a 3×3 matrix $\mathbf{W}(\mathbf{p})$:,

$$\begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = \mathbf{W}(\mathbf{p}) \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

$$\mathbf{W}(\mathbf{p}) = \begin{bmatrix} 1 + p_1 & p_3 & p_5 \\ p_2 & 1 + p_4 & p_6 \\ 0 & 0 & 1 \end{bmatrix}$$

Note: For convenience, when we want to refer to the warp as a function, we will use $\mathbf{W}(\mathbf{x}; \mathbf{p})$ and when we want to refer to the matrix for an affine warp, we will use $\mathbf{W}(\mathbf{p})$. We will use affine warp and affine transformation interchangeably.

Theory Questions (30 pts)

Before implementing the trackers, let's study some simple problems that will be useful during the implementation first. The answers to the below questions should be relatively short, consisting of a few lines of math and text.

Q1.1

Assuming the affine warp model defined above, derive the expression for the $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ in terms of the warp parameters $\mathbf{p} = [p_1 \ p_2 \ p_3 \ p_4 \ p_5 \ p_6]'$.

Answer -

We know that for the point $\mathbf{x} = \begin{bmatrix} x & y \end{bmatrix}^T$,

the corresponding warped point is given by -

$$\mathbf{W(x, p)} = \begin{bmatrix} (1+p_1)x & p_3 y & p_5 \\ p_2 x & (1+p_4)y & p_6 \end{bmatrix}$$

In terms of the warp parameters $\mathbf{p} = [p_1 \ p_2 \ p_3 \ p_4 \ p_5 \ p_6]'$, the derivative $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ is given by

$$\frac{\partial \mathbf{W}}{\partial \mathbf{p}} = \begin{bmatrix} \frac{\partial \mathbf{W}_x}{\partial \mathbf{p}_1} & \frac{\partial \mathbf{W}_x}{\partial \mathbf{p}_2} & . & . & . & \frac{\partial \mathbf{W}_x}{\partial \mathbf{p}_6} \\ \frac{\partial \mathbf{W}_y}{\partial \mathbf{p}_1} & \frac{\partial \mathbf{W}_y}{\partial \mathbf{p}_2} & . & . & . & \frac{\partial \mathbf{W}_y}{\partial \mathbf{p}_6} \end{bmatrix}$$

$$\frac{\partial \mathbf{W}}{\partial \mathbf{p}} = \begin{bmatrix} x & 0 & y & 0 & 1 & 0 \\ 0 & x & 0 & y & 0 & 1 \end{bmatrix}$$

Q1.2

Find the computational complexity (Big O notation) for each runtime iteration (computing $\mathbf{J}$ and $\mathbf{H}^{-1}$) of the Lucas Kanade method. Express your answers in terms of n, m and p where n is the number of pixels in thetemplate $\mathbf{T}$, m is the number of pixels in an input image $\mathbf{I}$ and p is the number of parameters used to describe the warp W.

You may refer to the supplementary PDF for more detailed descriptions of the algorithm.

Answer -

m and n will have same number of pixels as they are equal because $\mathbf{I}$ is generated by warping $\mathbf{T}$

Steps of Lucas-Kanade method -

-- We warp $\mathbf{I}$ with $\mathbf{W(x, p)}$ and find $\mathbf{I(W(x, p))}$

- $\mathbb{O}(np)$ because warping involves multiplying parameters with every pixel coordinates

-- We compute the error image $\mathbf{E = T(x) - I(W(x, p))}$

- $\mathbb{O}(n)$ because it's finding elementwise difference

-- We warp the gradient $\nabla \mathbf{I}$ with $\mathbf{W(x, p)}$

- $\mathbb{O}(n)$ assuming that we already have the warped image else $\mathbb{O}(np)$

-- We evaluate the Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$

- $\mathbb{O}(np)$ as this is computed per pixel and using every parameter

-- we compute the steepest descent images $\nabla \mathbf{I} \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$

- $\mathbb{O}(np)$ as again this involves manipulation of a n x p matrix

-- We compute the Hessian matrix - $\mathbf{H} = \left[\nabla \mathbf{I} \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]^T \nabla \mathbf{I} \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$

- This Hessian computation takes $\mathbb{O}(np^2)$ due to the above matrix multiplication - multiplying p x n matrix by n x p matrix can be done fastest in $\mathbb{O}(pnp)$

-- We compute $\left[\nabla \mathbf{I} \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]^T \mathbf{E}$

- $\mathbb{O}(np)$ due to the above matrix multiplication
- multiplying p x n matrix by n x 1 matrix can be done fastest in $\mathbb{O}(pn)$

-- We compute $\nabla \mathbf{p} = H^{-1} \left[\nabla \mathbf{I} \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]^T \mathbf{E}$

- Inverting the Hessian takes $\mathbb{O}(p^3)$ while multiplying it with the rest would need an additional $\mathbb{O}(p^2)$ - multiplying p x p matrix by p x 1 matrix can be done fastest in $\mathbb{O}(p^2)$

-- We update the parameters $\mathbf{p}$

- $\mathbb{O}(p)$ because it's a simple update by addition per parameter

Hence, we can say that the overall time complexity per iteration of Lucas-Kanade tracking is $\mathbb{O}(p^3 + np^2)$

Q1.3

Find the computational complexity (Big O notation) for the initialization step (Precomputing $\mathbf{J}$ and $\mathbf{H}^{-1}$) and for each runtime iteration of the Matthews-Baker method. Express your answers in terms of n, m and p where n is the number of pixels in the template $\mathbf{T}$, m is the number of pixels in an input image $\mathbf{I}$ and p is the number of parameters used to describe the warp W. You may refer to the supplementary PDF for more detailed descriptions of the algorithm.

How does this compare to the run time of the regular Lucas-Kanade method?

Answer -

As $\mathbf{I}$ is generated by warping $\mathbf{T}$, m and n should be equal and thus will have same number of pixels

Precomputation of Matthews-Baker tracking:

-- We evaluate the gradient $\nabla\mathbf{T}$ of the template $\mathbf{T(x)}$

- $\mathbb{O}(n)$ because it would involve iterating over each pixel

-- We evaluate the jacobian $\frac{\partial\mathbf{W}}{\partial\mathbf{p}}$ at $\mathbf{(x, 0)}$

- $\mathbb{O}(np)$ as this is computed per pixel and using each parameter

-- We compute the steepest descent images $\nabla\mathbf{T}\frac{\partial\mathbf{W}}{\partial\mathbf{p}}$

- $\mathbb{O}(np)$ as again this involves manipulation of a n x p matrix

-- We compute the hessian matrix - $\mathbf{H} = \left[\nabla\mathbf{T}\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\right]^T \nabla\mathbf{T}\frac{\partial\mathbf{W}}{\partial\mathbf{p}}$

- Hessian computation takes $\mathbb{O}(np^2)$ due to the above matrix multiplication -
- Multiplying p x n matrix by n x p matrix can be done fastest in $\mathbb{O}(pnp)$

One iteration of Matthews-Baker tracking:

-- We warp $\mathbf{I}$ with $\mathbf{E} = \mathbf{W(x, p)}$ to find $\mathbf{I(W(x, p))}$

- $\mathbb{O}(n)$ assuming that we already have the warped image else $\mathbb{O}(np)$

-- We compute error image $\mathbf{E} = \mathbf{I(W(x, p)) - T(x)}$

- $\mathbb{O}(n)$ as it's just finding elementwise difference

-- We compute $\left[\nabla\mathbf{T}\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\right]^T\mathbf{E}$

- $\mathbb{O}(np)$ due to the above matrix multiplication -
- multiplying p x n matrix by n x 1 matrix can be done fastest in $\mathbb{O}(pn)$

-- We compute $\nabla\mathbf{p} = H^{-1}\left[\nabla\mathbf{T}\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\right]^T\mathbf{E}$

- Inverting the hessian takes $\mathbb{O}(p^3)$
- Multiplying it with the rest would need an additional $\mathbb{O}(p^2)$
- Multiplying p x p matrix by p x 1 matrix can be done fastest in $\mathbb{O}(p^2)$

-- We update the warp $\mathbf{W(x; p) = W(x; p) \cdot W(x;\nabla p)^{-1}}$

- $\mathbb{O}(p^2)$

Precomputation has total complexity of $\mathbb{O}(np^2)$

whereas $\mathbb{O}(p^3 + np)$is for one iteration of Matthews-Baker tracking

We see that Matthews-Baker tracking is faster with a smaller time complexity ($\mathbb{O}(p^3 + np)$) as compared to Lucas-Kanade tracking ($\mathbb{O}(p^3 + np^2)$).

While the precomputation in Matthews-Baker tracking has a time complexity of $\mathbb{O}(np^2)$, this part only runs once.

So, for k iterations,

Lucas-Kanade tracking would have a time complexity of $\mathbb{O}(kp^3 + knp^2)$ and Matthews-Baker tracking would have $\mathbb{O}(kp^3 + knp) + \mathbb{O}(np^2)$.

It can clearly be seen that as the number of iterations increases, Matthews-Baker runs way faster than Lucas-Kanade.

Jacobian and Hessian is calculated in each iteration in the algorithm of Lucas - Kanade while in Matthews - Baker the inverse of Hessian and steapest decend is calculated only once

Coding Questions Write-up

Q1.1

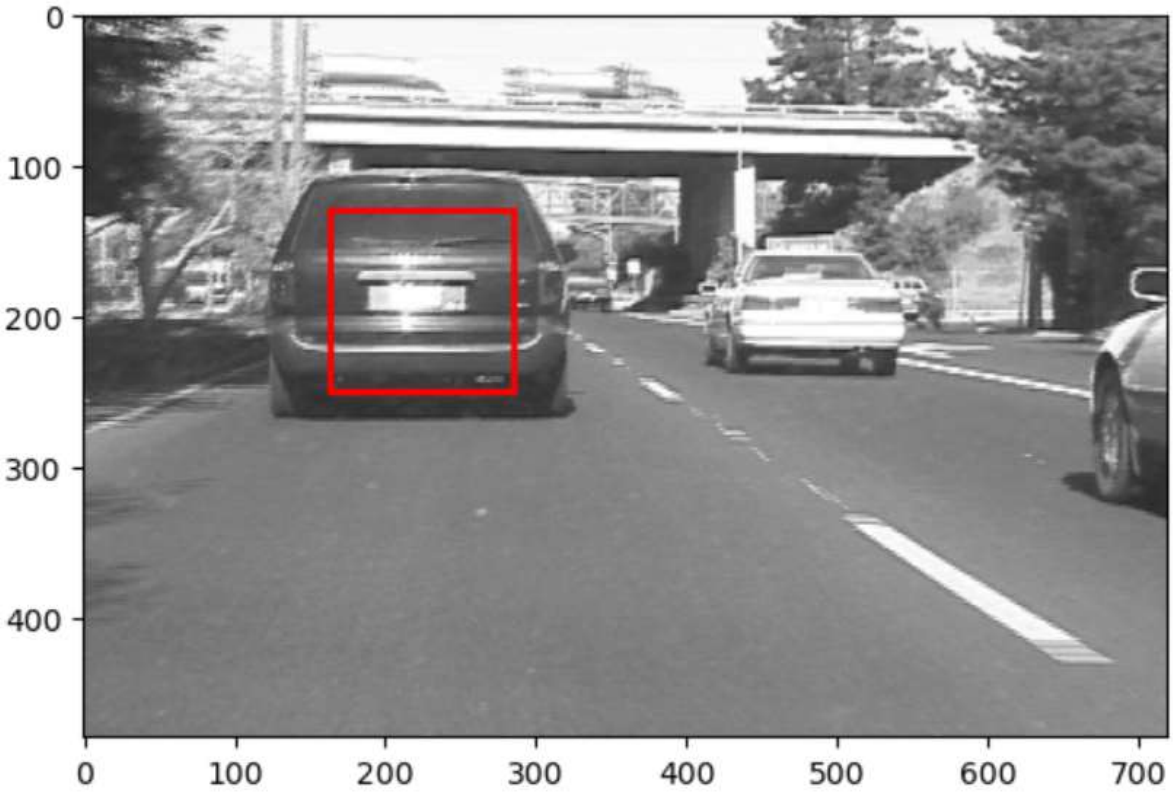
In [ ]:

```
def LucasKanade(It, It1, rect, thresh=.025, maxIters=100):

    #  threshold = thresh

    x1, y1, x2, y2 = rect

    s1 = np.arange(It.shape[0])
    s2 = np.arange(It.shape[1])
    s3 = np.arange(It1.shape[0])
    s4 = np.arange(It1.shape[1])
    interit = RectBivariateSpline(s1, s2, It)
    interit1 = RectBivariateSpline(s3, s4, It1)

    random_x = np.arange(x1, x2 + 0.5)
    random_y = np.arange(y1, y2 + 0.5)
    x, y = np.meshgrid(random_x, random_y)
    x = x.flatten()
    y = y.flatten()
    T_ev = interit.ev(y, x)

    p = np.zeros(2)
    for i in range(maxIters):

        x1 = x + p[0]
        y1 = y + p[1]

        I_ev = interit1.ev(y1, x1)

        Ix = interit1.ev(y1, x1, dx=0, dy=1).reshape(-1, 1)
        Iy = interit1.ev(y1, x1, dx=1, dy=0).reshape(-1, 1)
        dI = np.hstack((Ix, Iy))
        dWdp = np.eye(2)
        A = dI @ dWdp

        b = T_ev - I_ev

        dp = np.linalg.lstsq(A, b, rcond=None)[0]
        p = p + dp

        if np.sqrt(np.sum(dp ** 2)) <= thresh:
            break

    return p
```

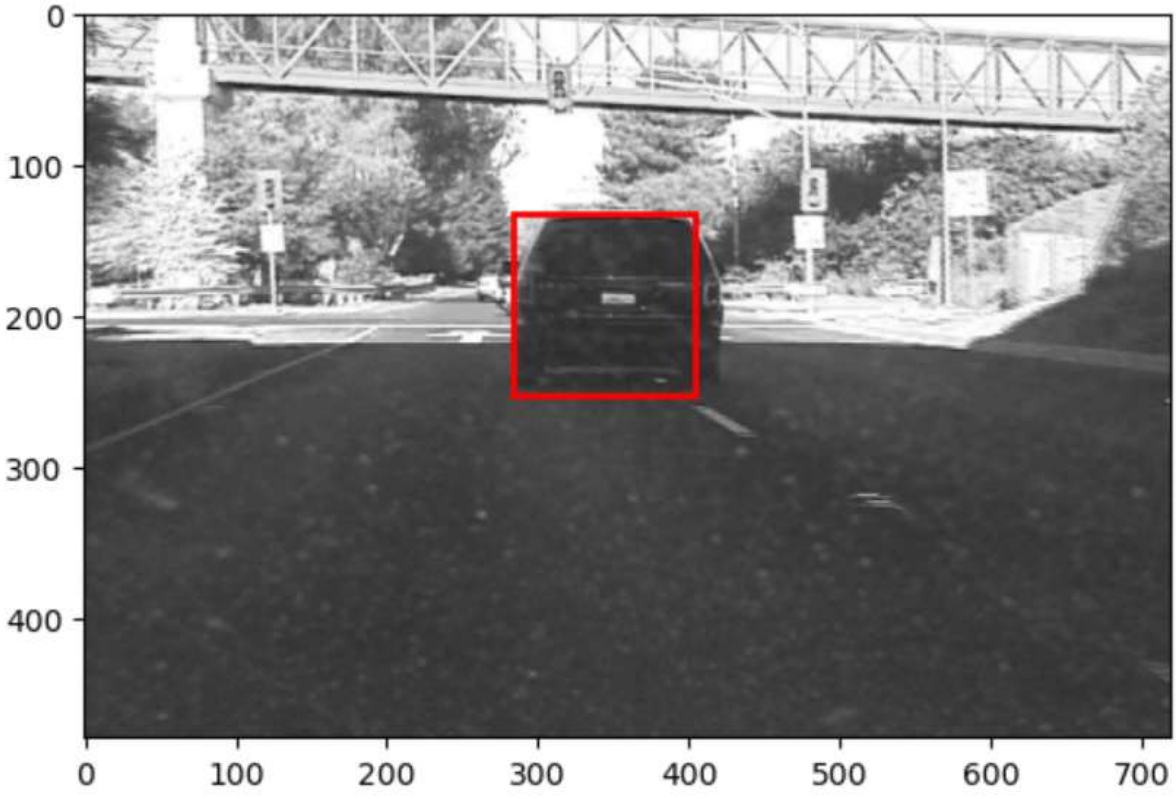
Results from each dataset -

1. Car1


0
100
200
300
400
0
100
200
300
400
500
600
700


0
100
200
300
400
0
100
200
300
400
500
600
700


0
100
200
300
400
0
100
200
300
400
500
600
700


0
100
200
300
400
0
100
200
300
400
500
600
700

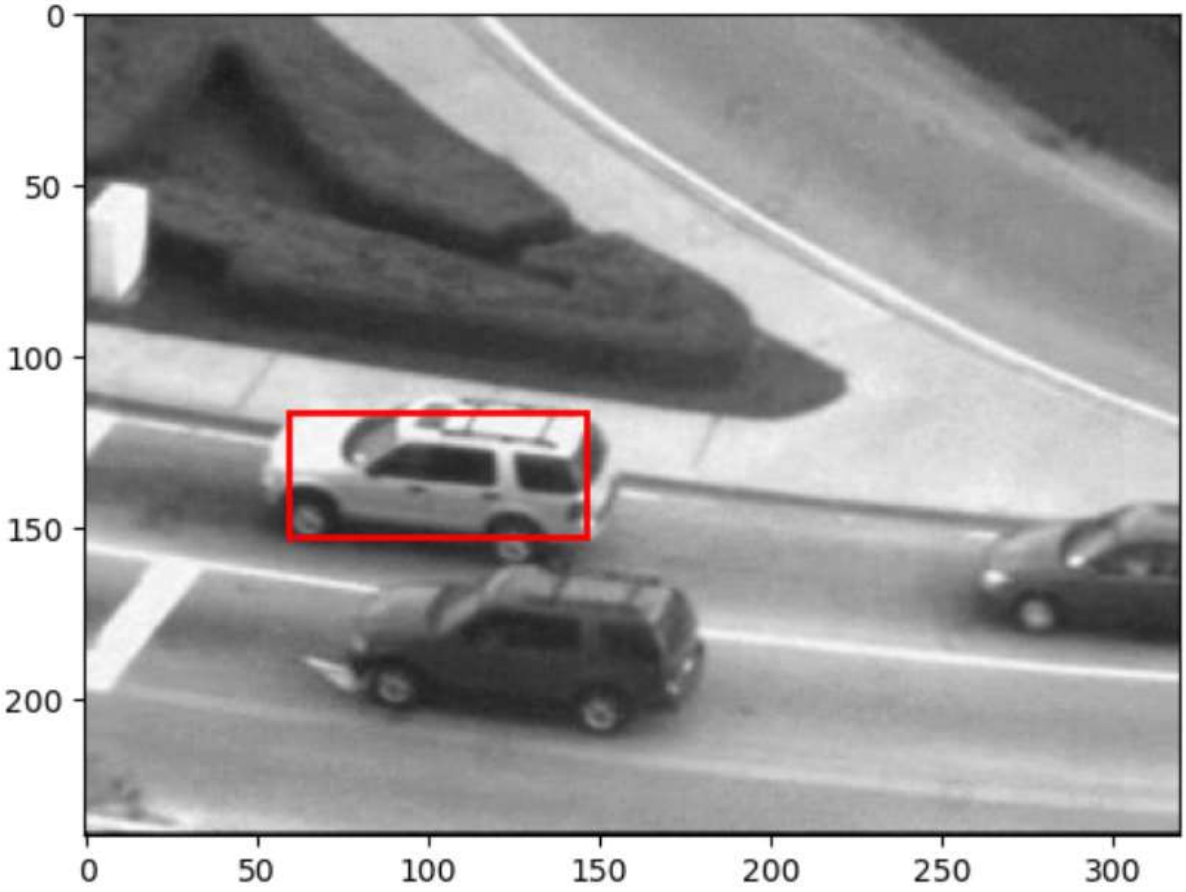

0
100
200
300
400
0
100
200
300
400
500
600
700

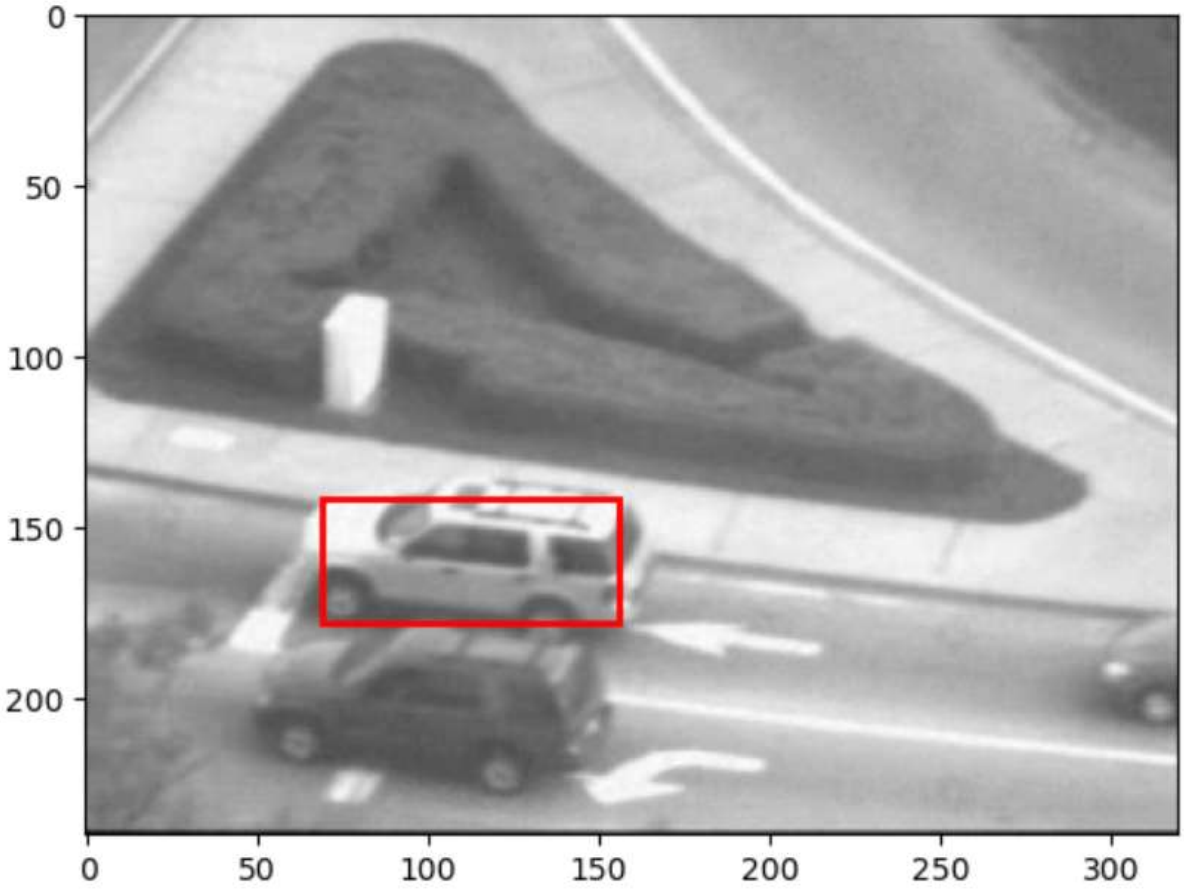

0
100
200
300
400
0
100
200
300
400
500
600
700

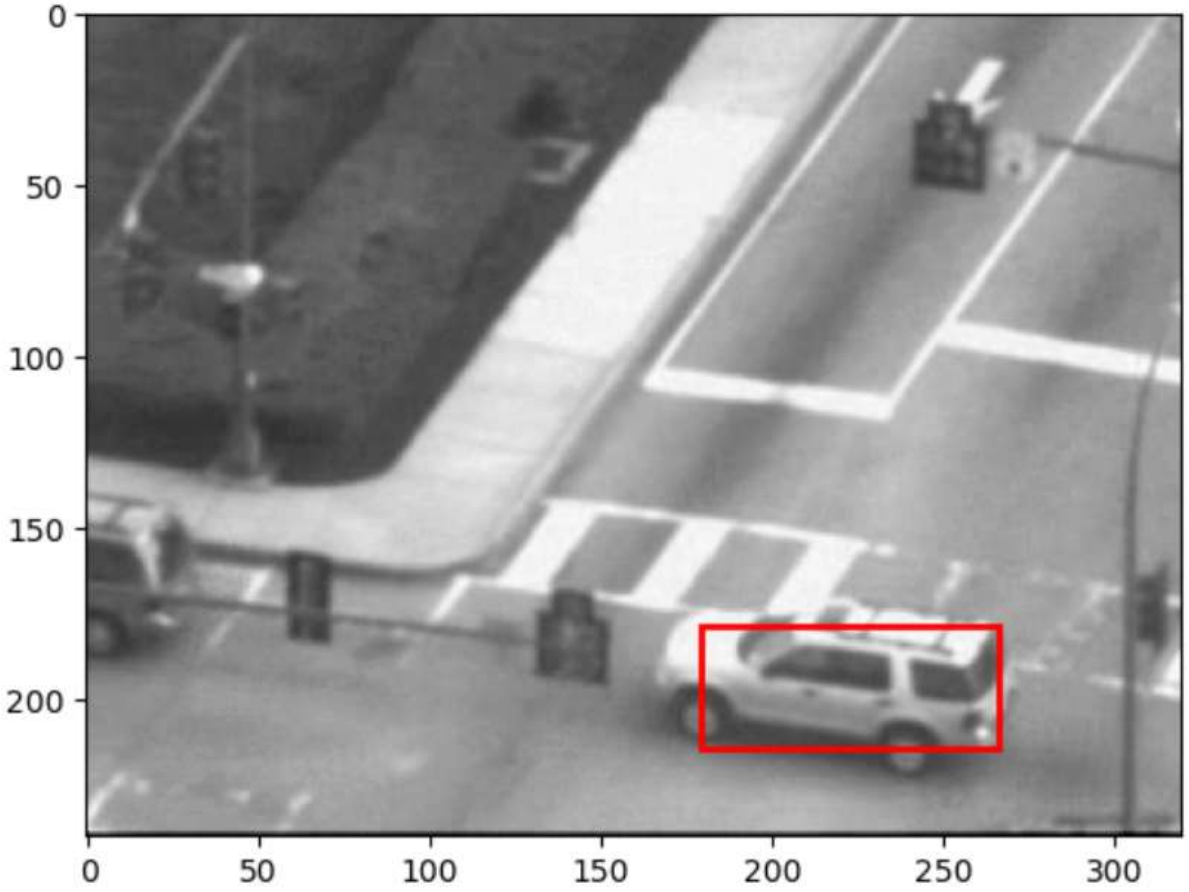

0
100
200
300
400
0
100
200
300
400
500
600
700

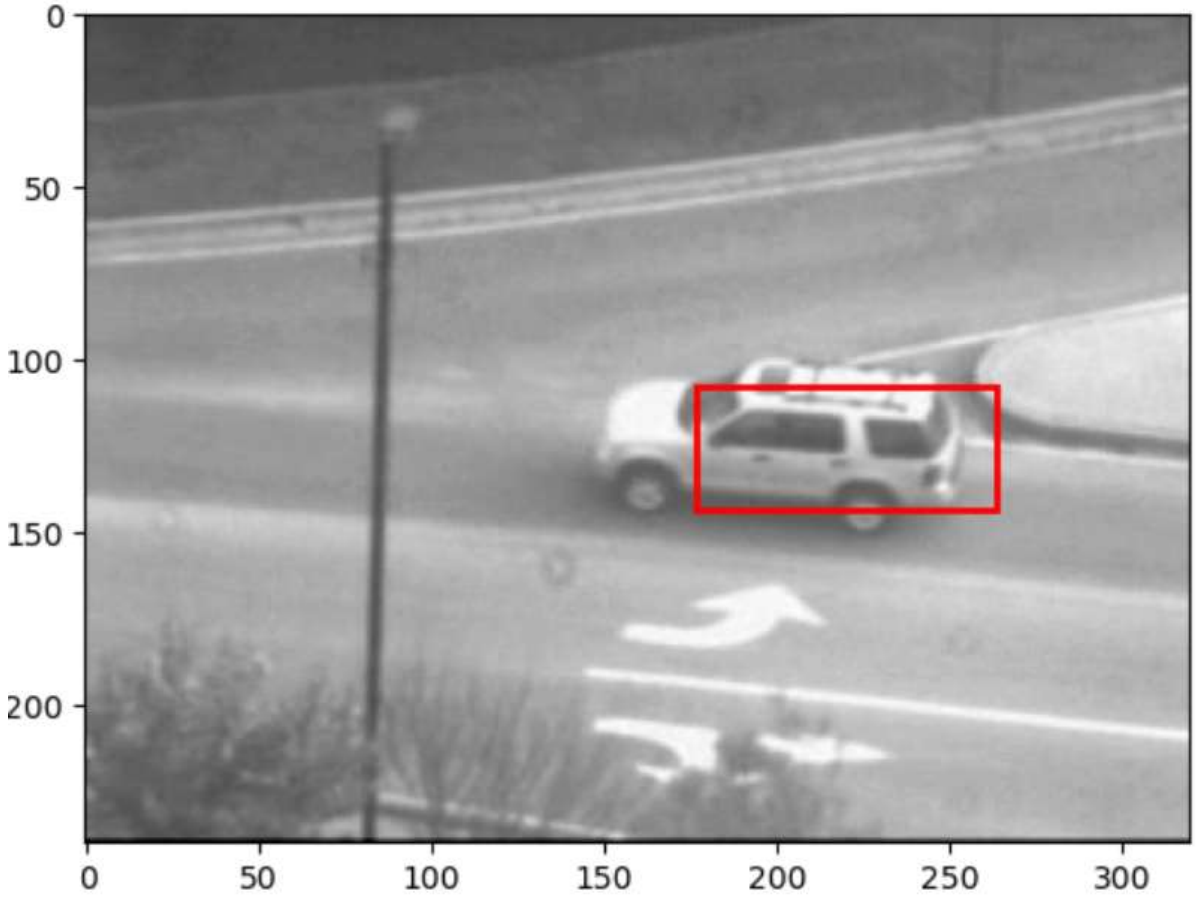

0
100
200
300
400
0
100
200
300
400
500
600
700

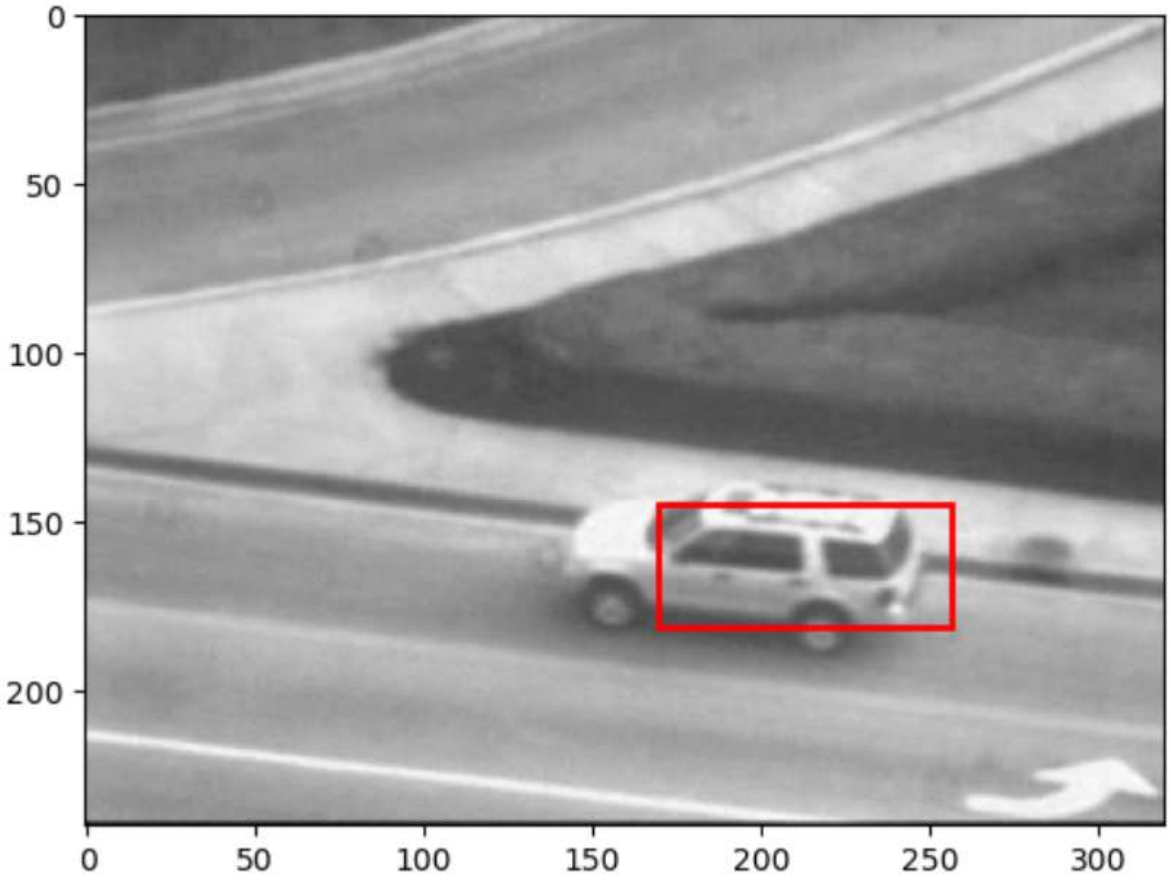
2. Car2

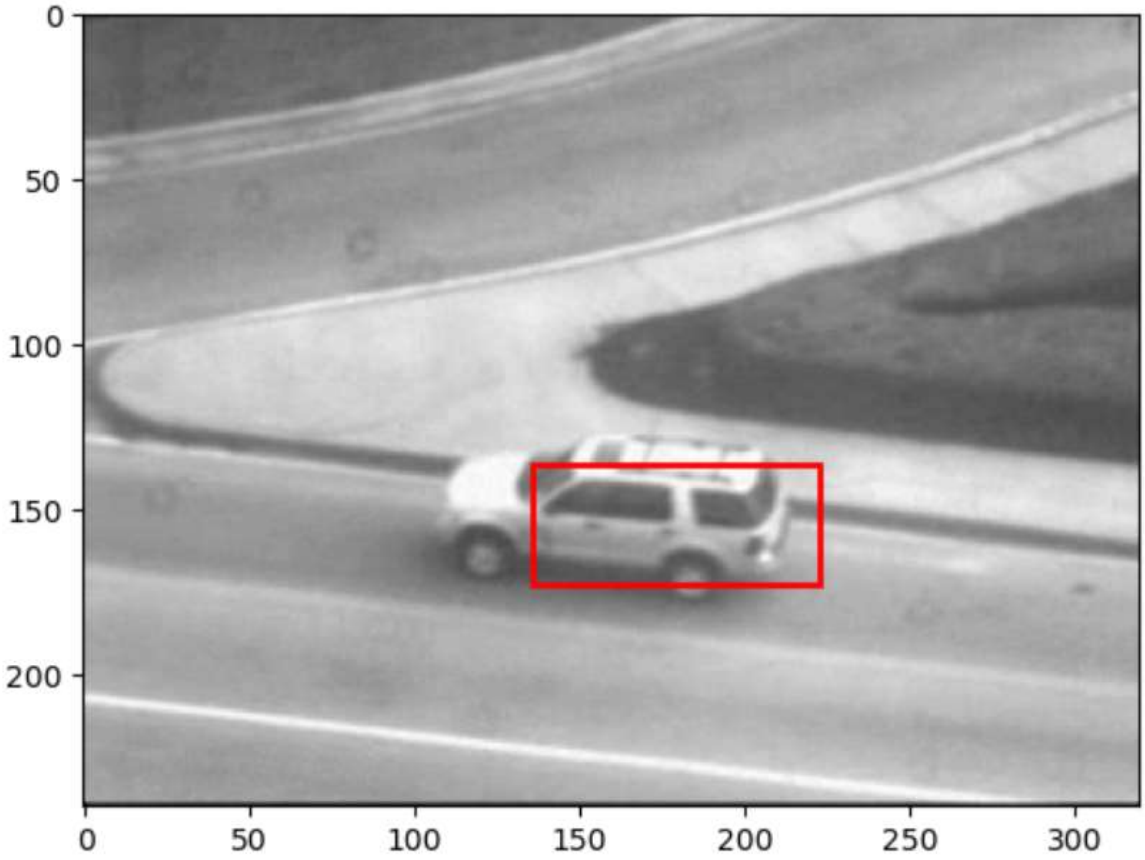

0
50
100
150
200
0
50
100
150
200
250
300


0
50
100
150
200
0
50
100
150
200
250
300

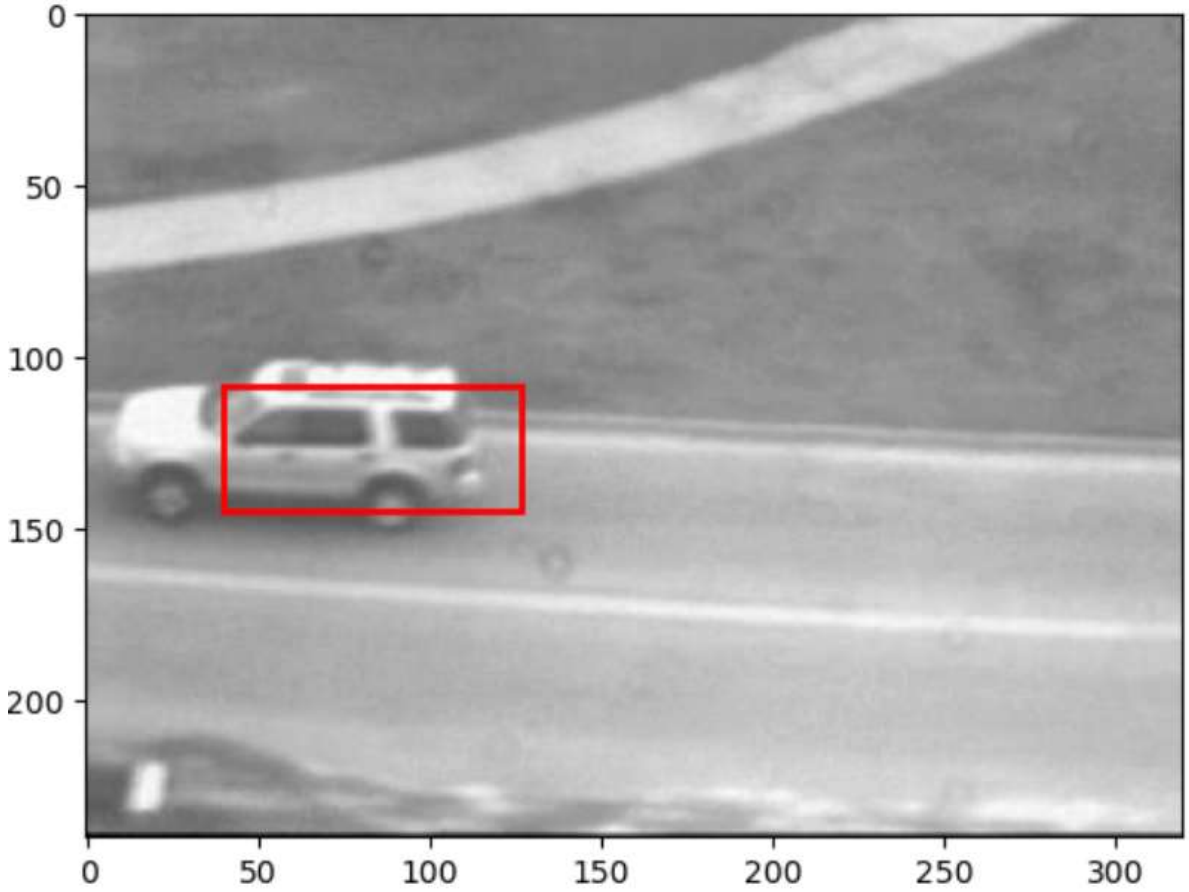

0
50
100
150
200
0
50
100
150
200
250
300

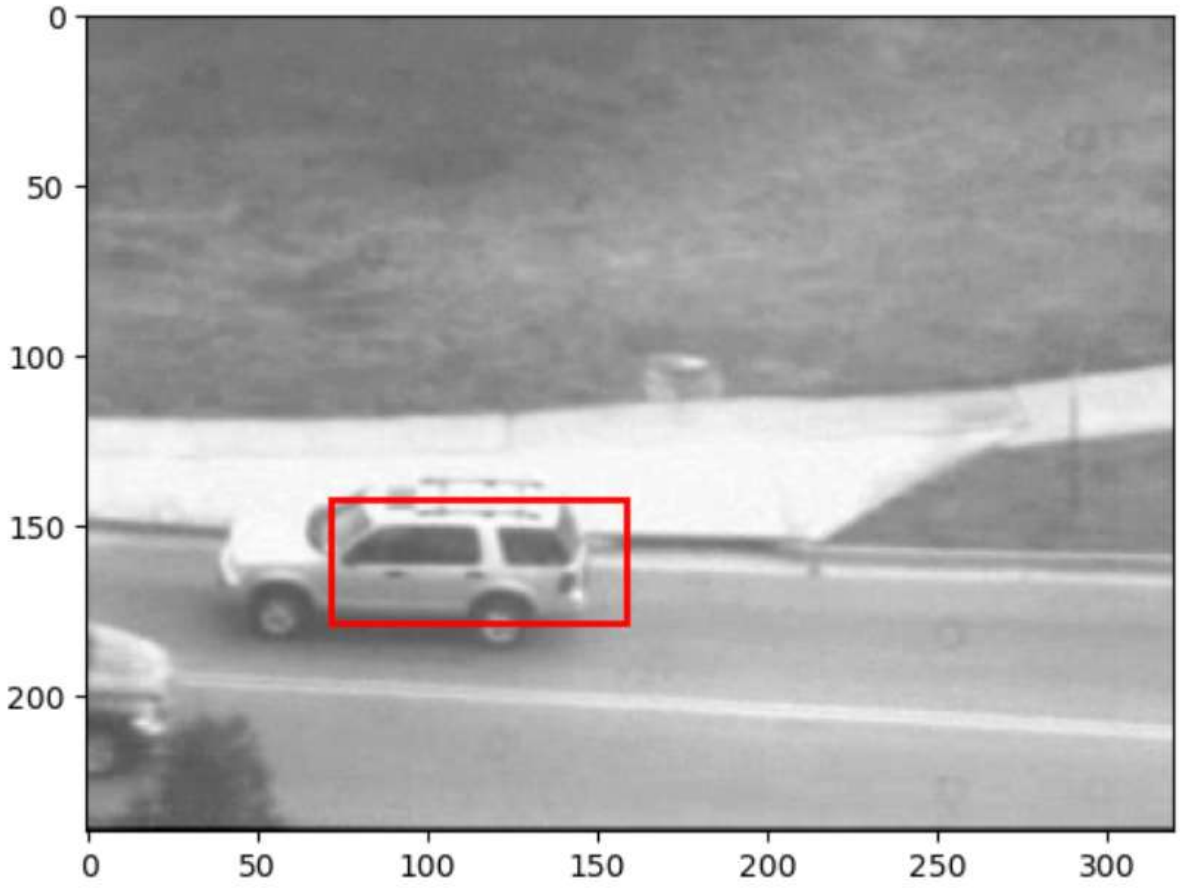

0
50
100
150
200
0
50
100
150
200
250
300


0
50
100
150
200
0
50
100
150
200
250
300

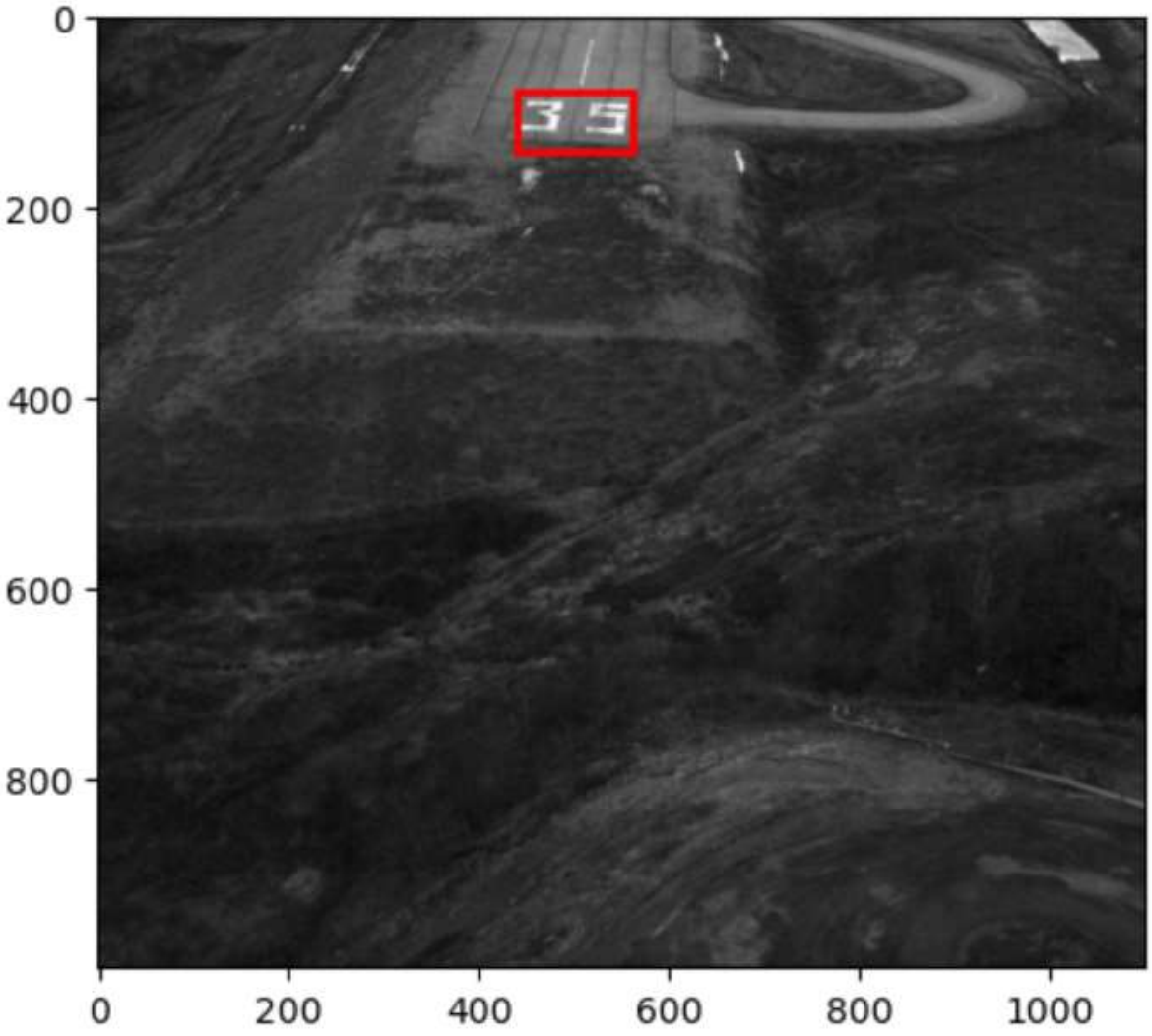

0
50
100
150
200
0
50
100
150
200
250
300

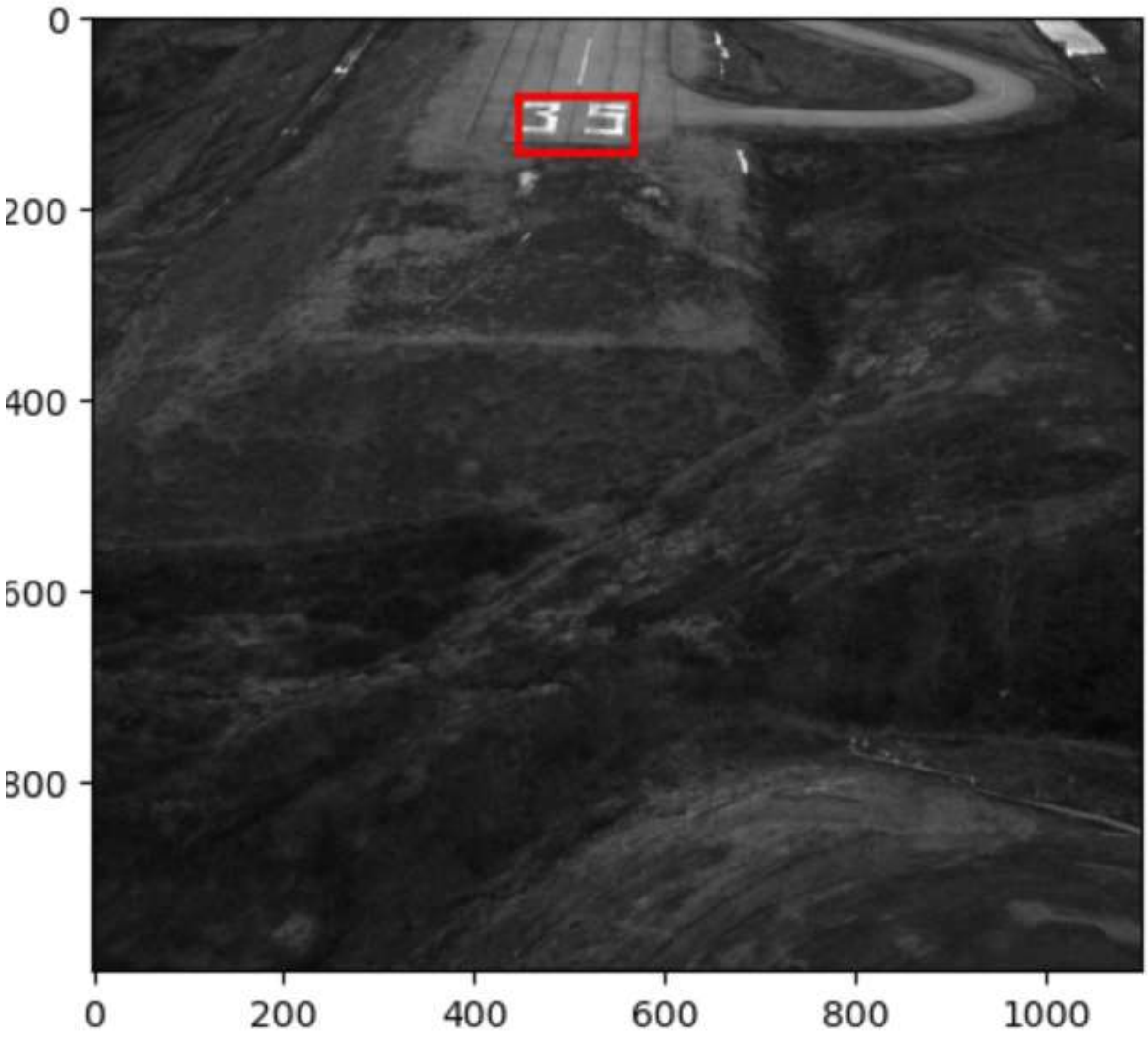

0
50
100
150
200
0
50
100
150
200
250
300

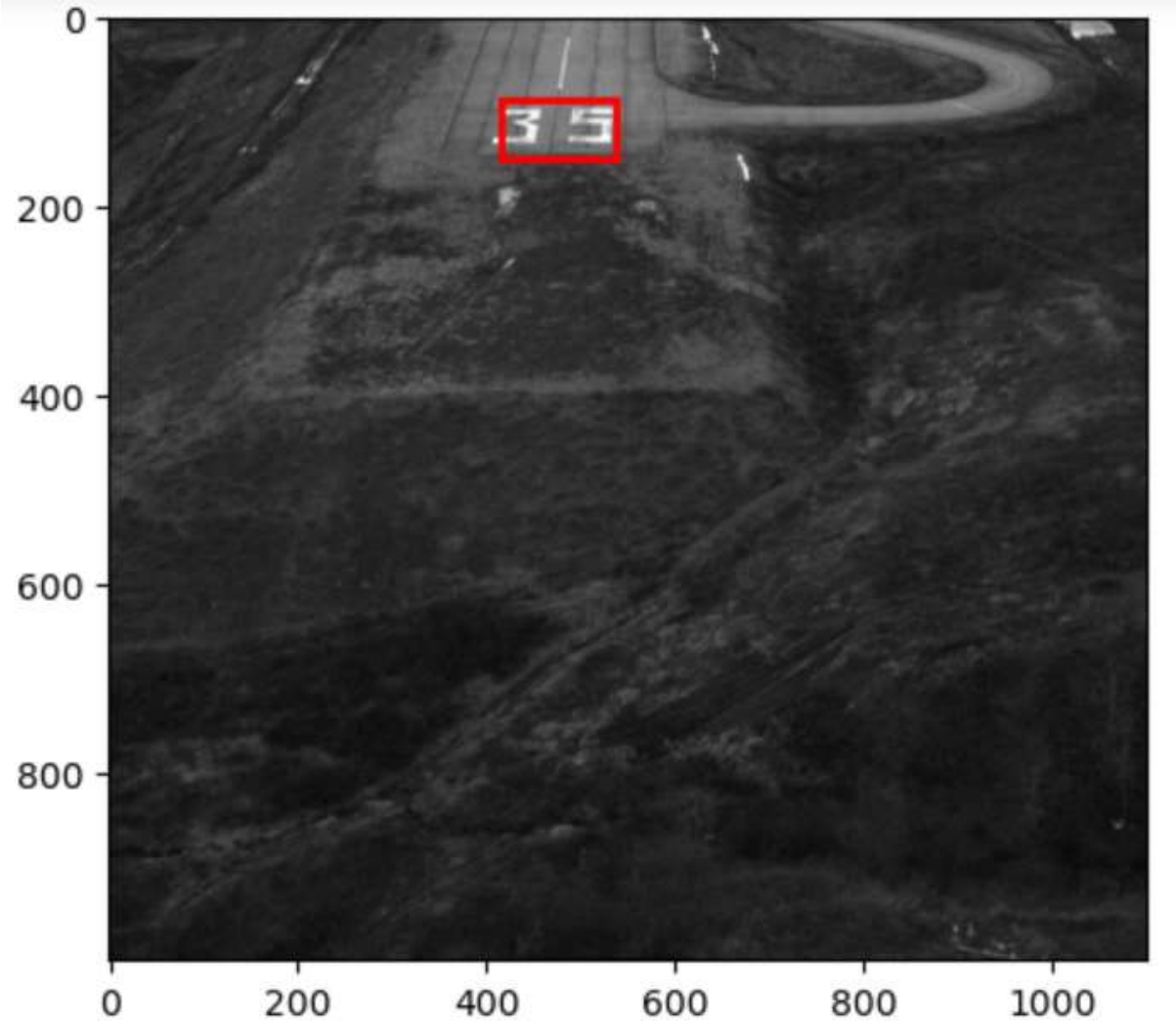

0
50
100
150
200
0
50
100
150
200
250
300

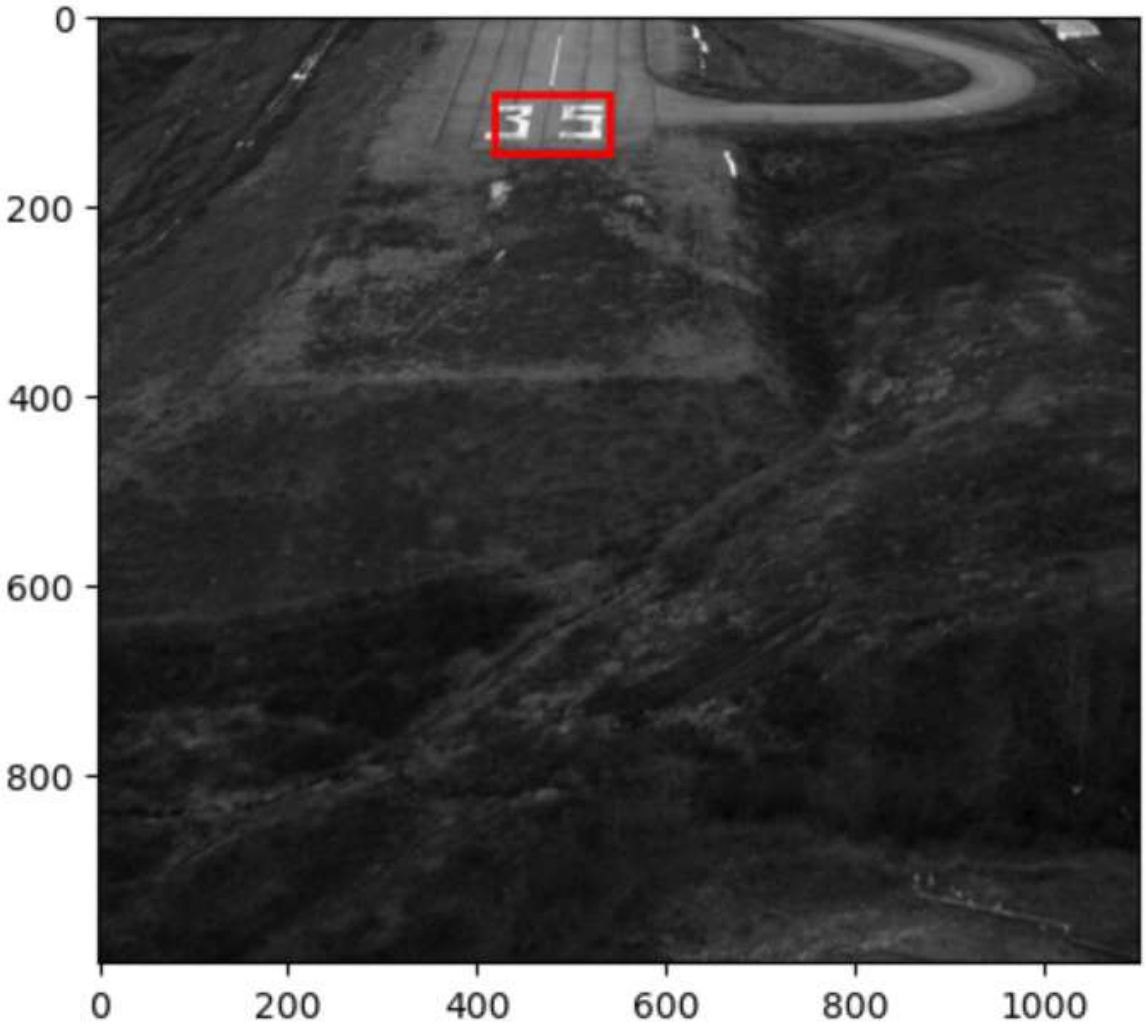


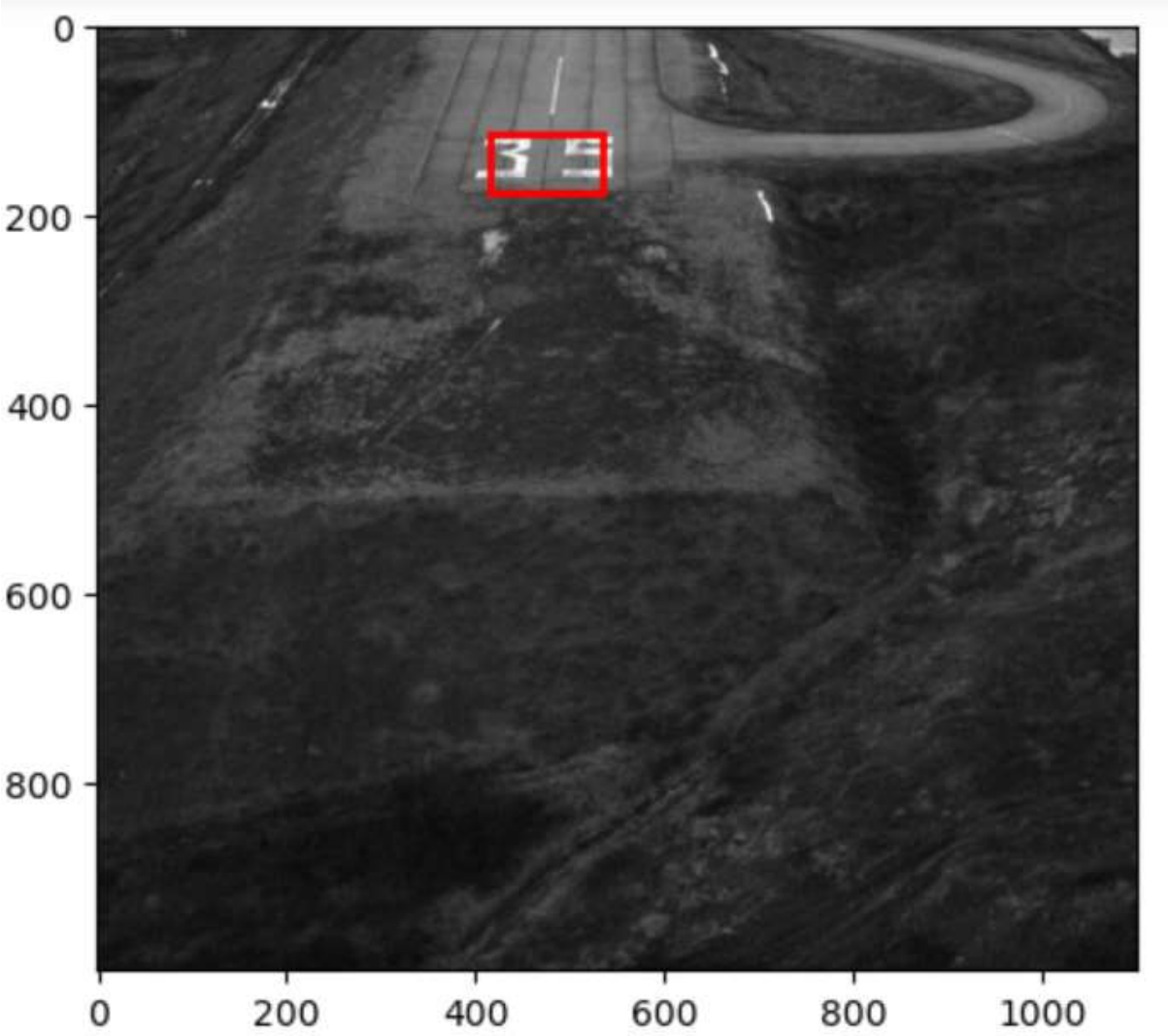
3. Landing

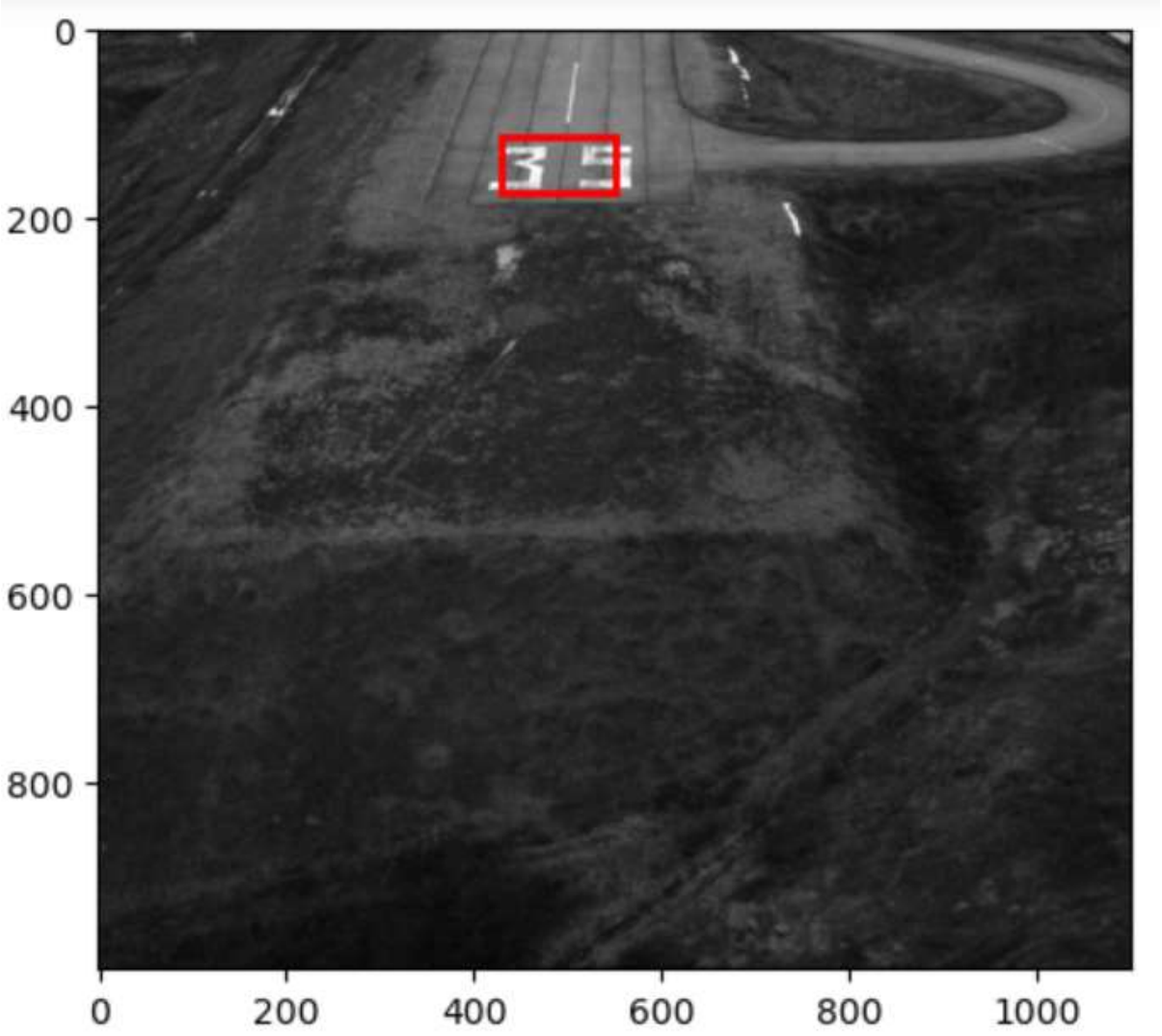

0
200
400
600
800
0
200
400
600
800
1000

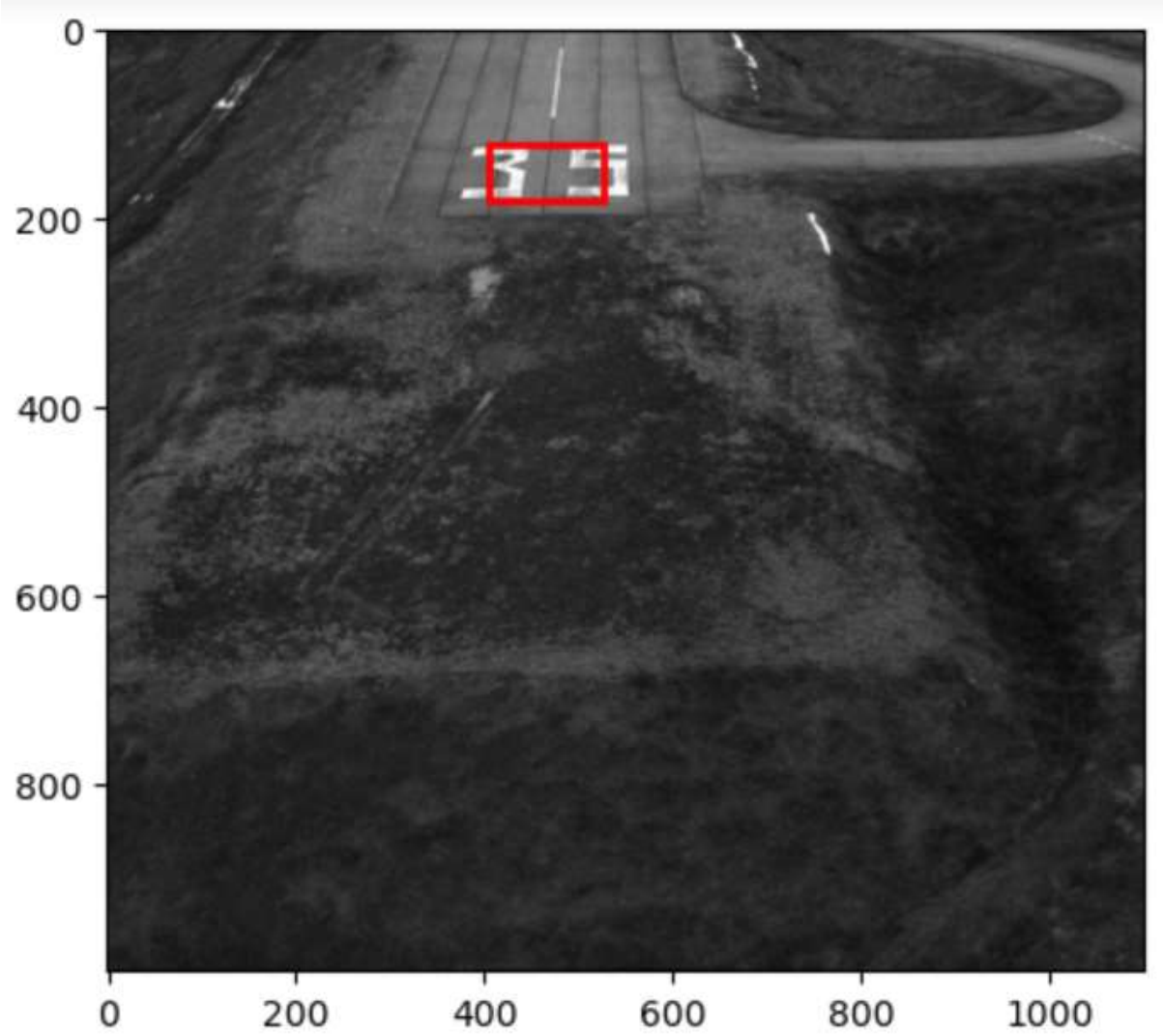

0
200
400
600
800
0
200
400
600
800
1000

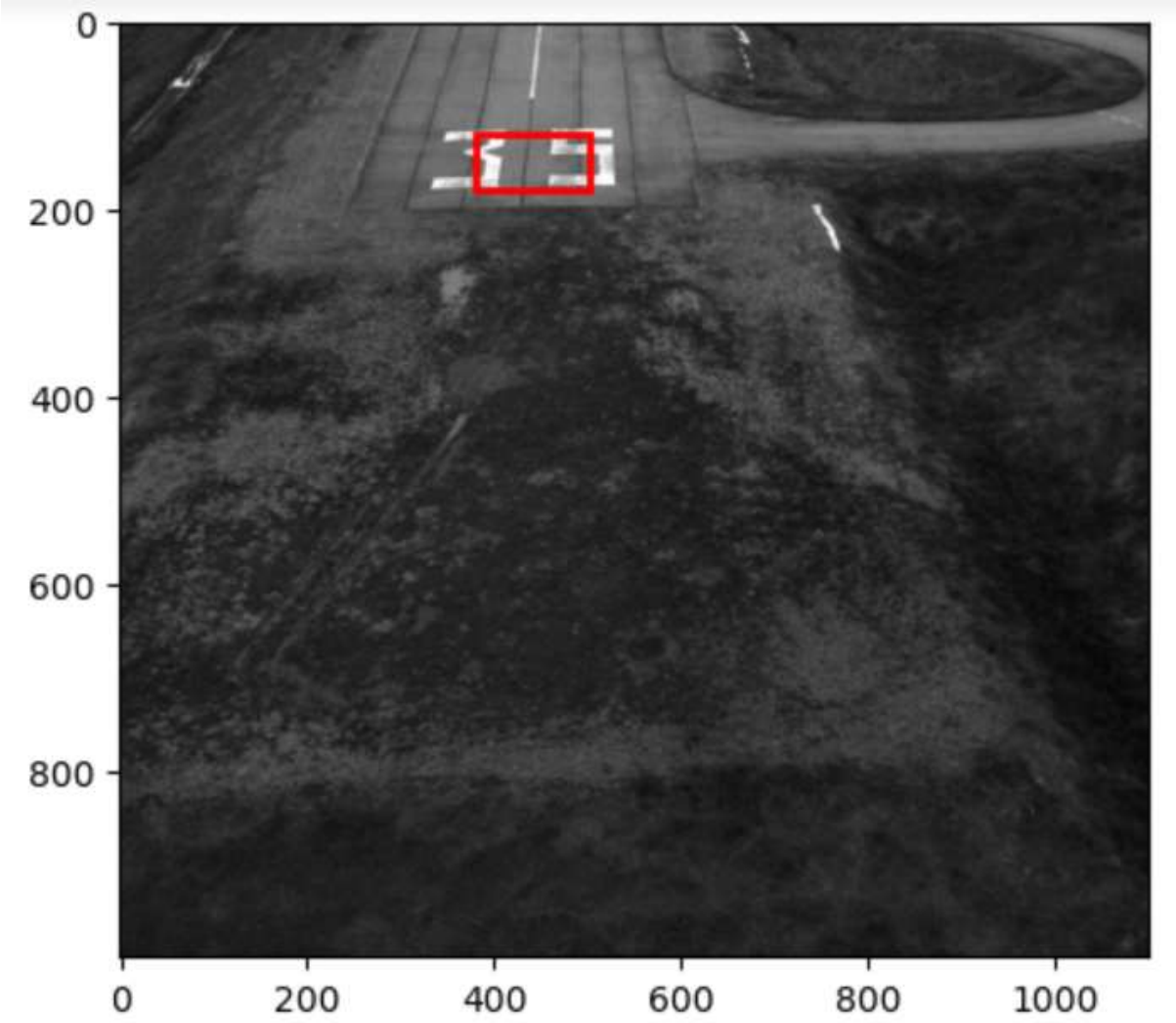

0
200
400
600
800
0
200
400
600
800
1000

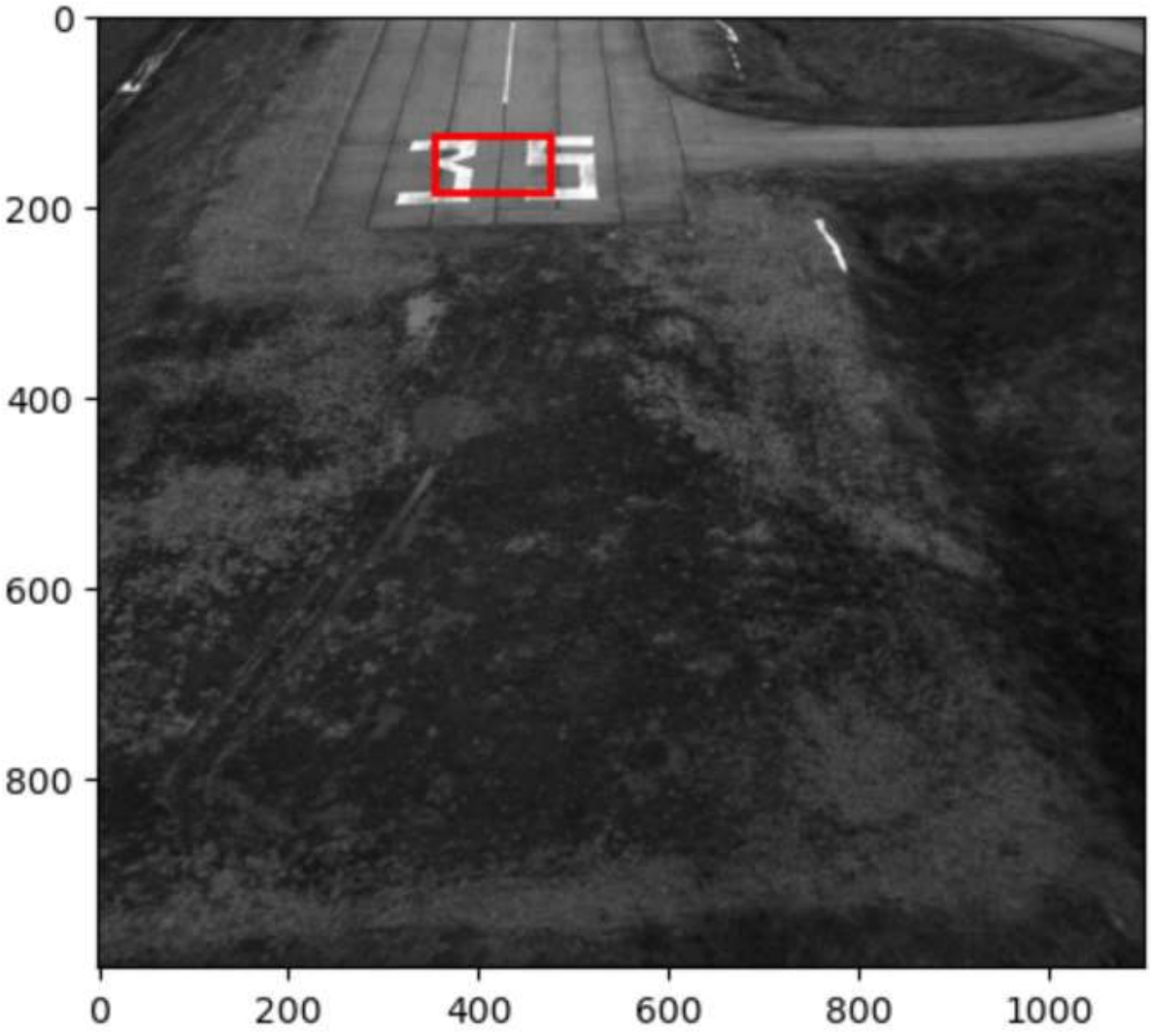

0
200
400
600
800
0
200
400
600
800
1000


0
200
400
600
800
0
200
400
600
800
1000

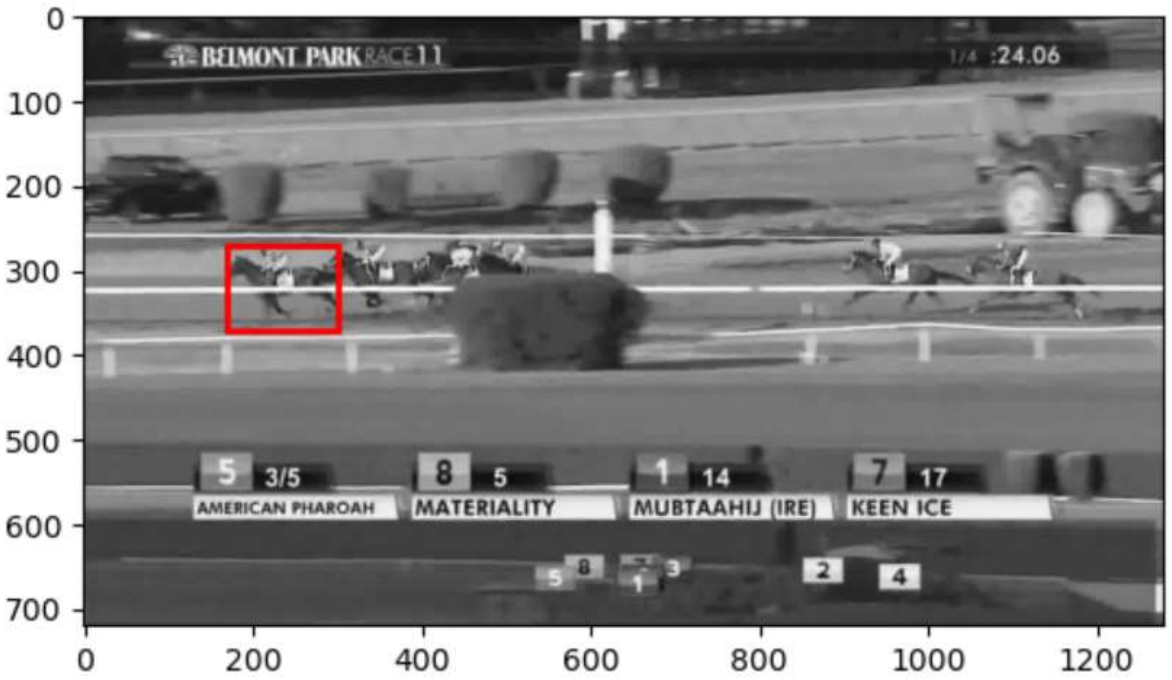

0
200
400
600
800
0
200
400
600
800
1000

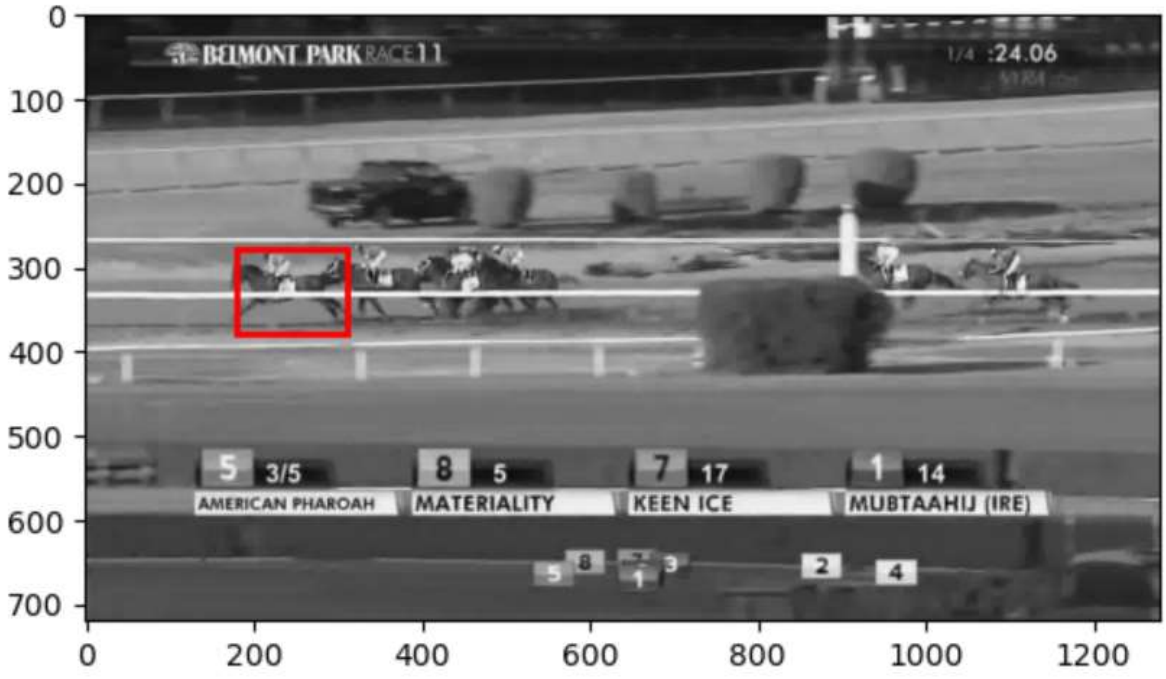

0
200
400
600
800
0
200
400
600
800
1000


0
200
400
600
800
0
200
400
600
800
1000

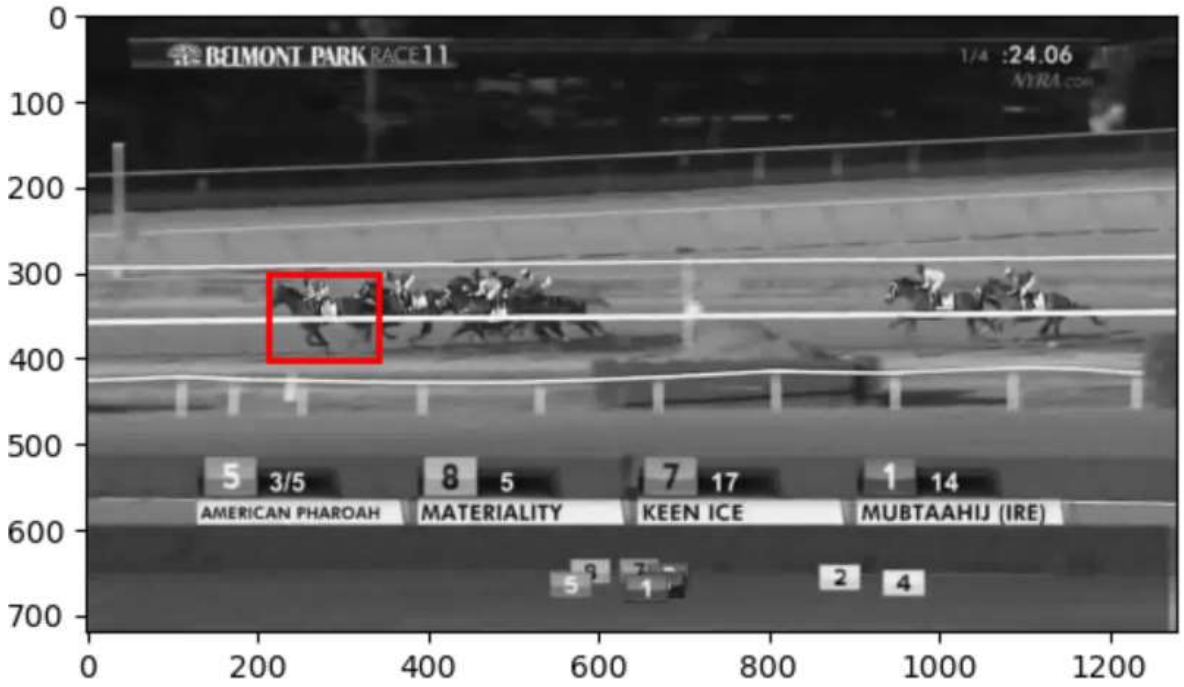


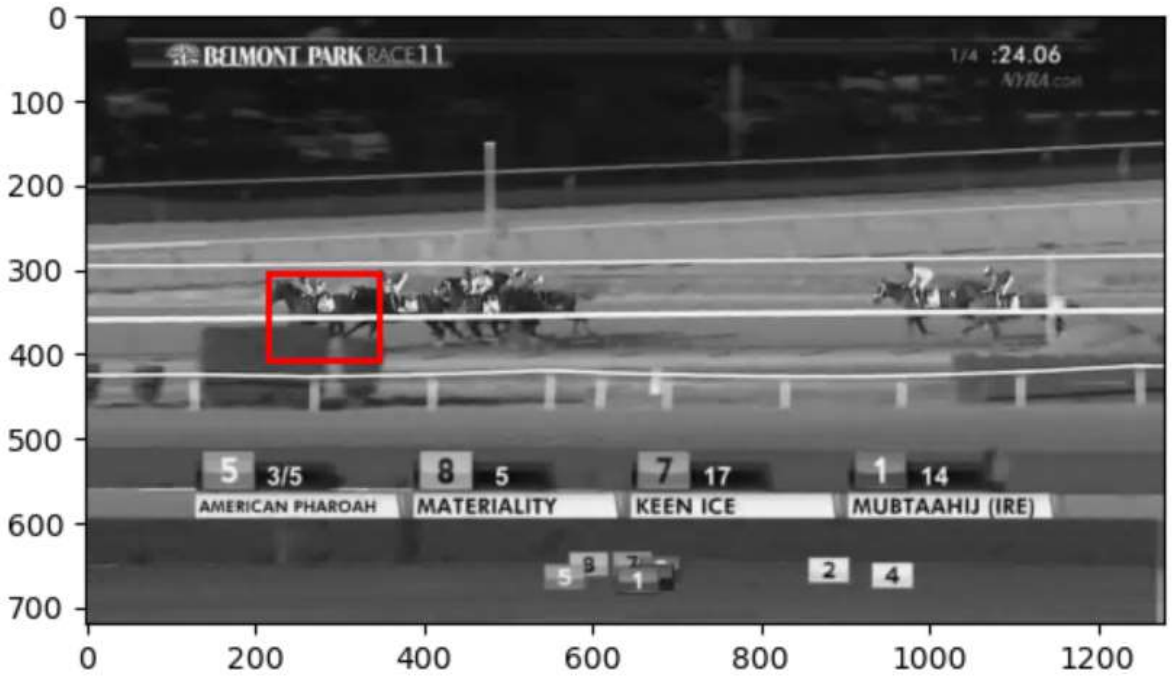
4. Race

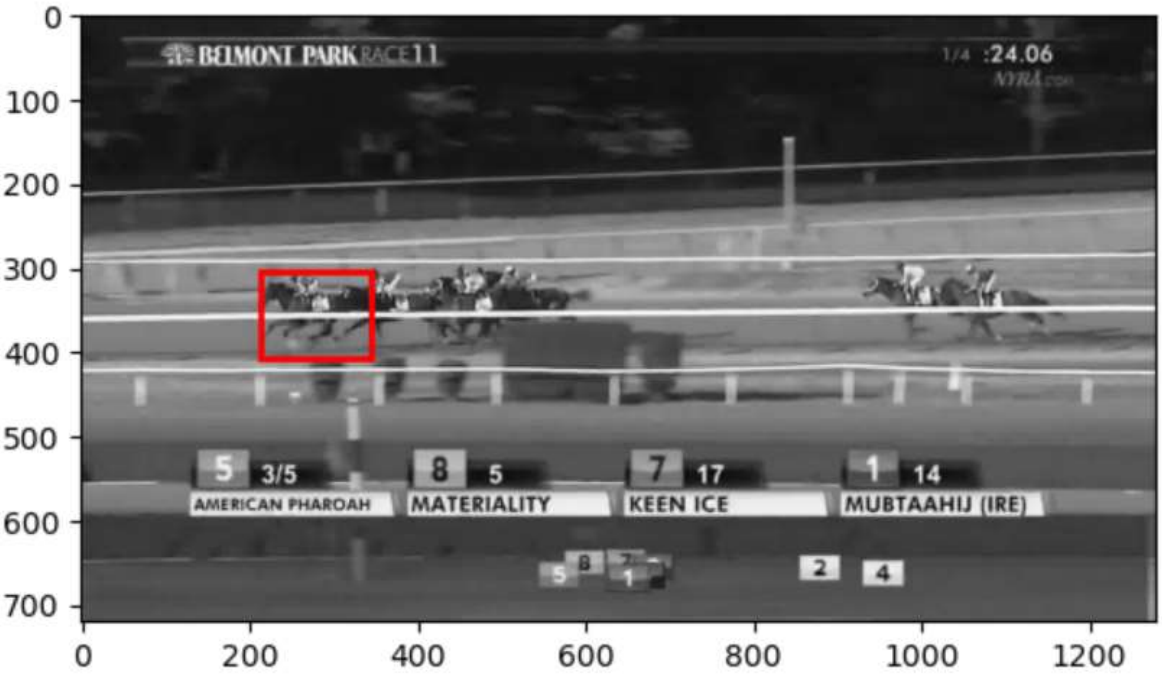

BELMONT PARK RACE 11
1/4 :24.06
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
1 14
MUBTAAHIJ (IRE)
7 17
KEEN ICE
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

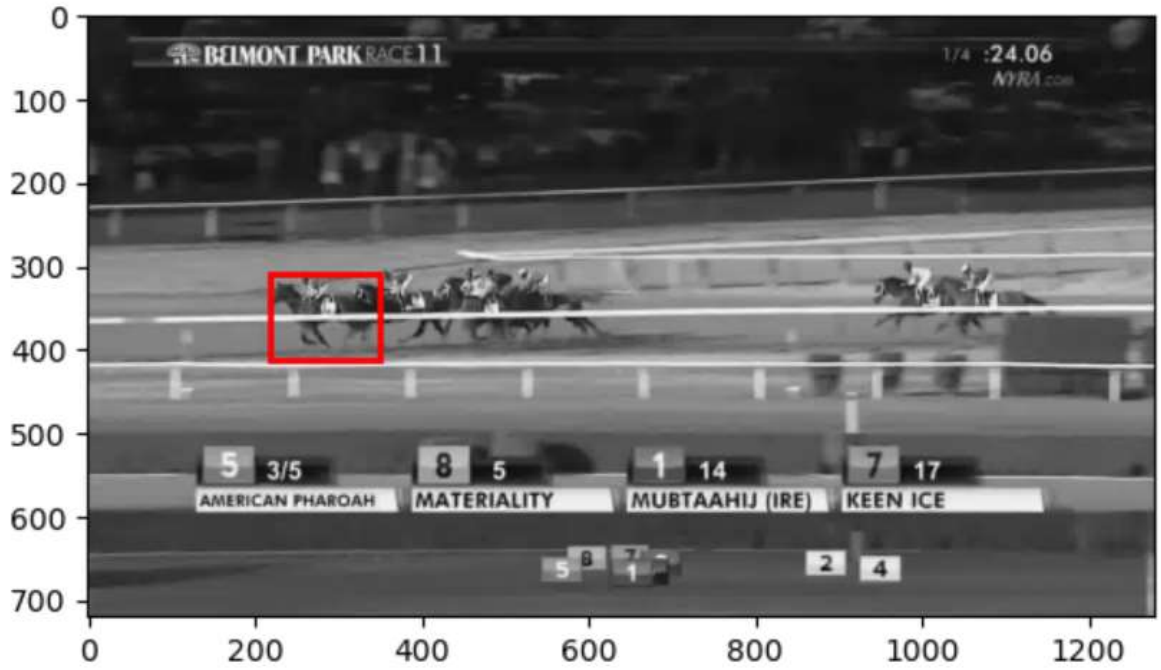

BELMONT PARK RACE 11
1/4 :24.06
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
7 17
KEEN ICE
1 14
MUBTAAHIJ (IRE)
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

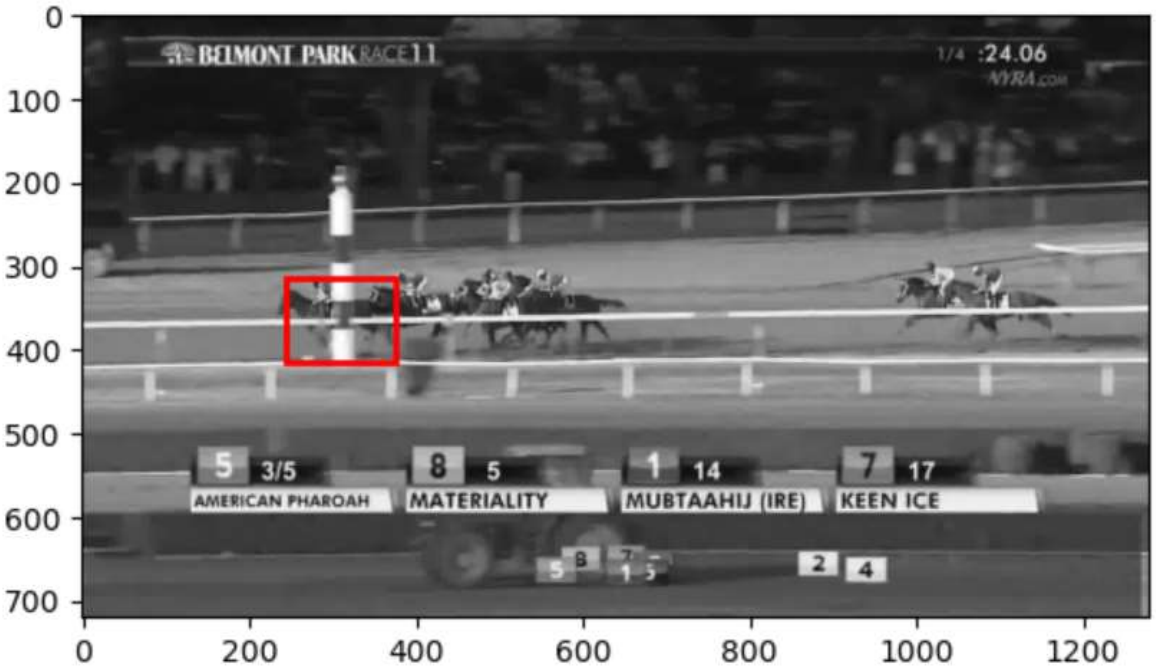

BELMONT PARK RACE 11
1/4 :24.06
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
7 17
KEEN ICE
1 14
MUBTAAHIJ (IRE)
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

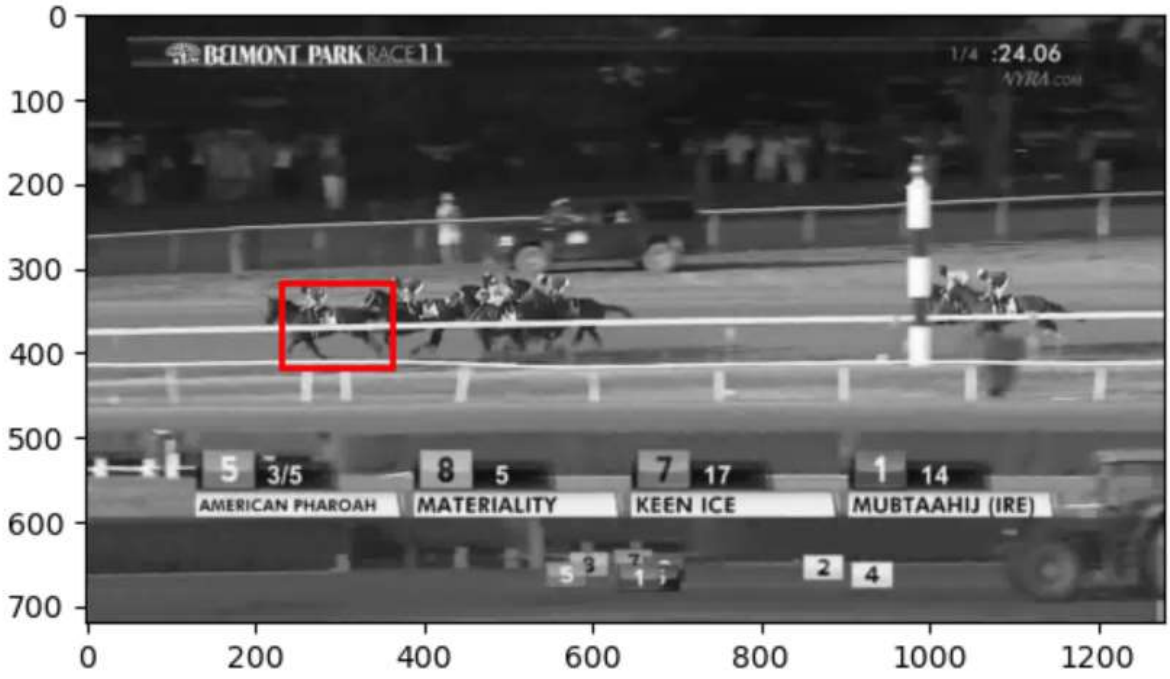

BELMONT PARK RACE 11
1/4 :24.06
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
7 17
KEEN ICE
1 14
MUBTAAHIJ (IRE)
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


BELMONT PARK RACE 11
1/4 :24.06
NYRA.com
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
7 17
KEEN ICE
1 14
MUBTAAHIJ (IRE)
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

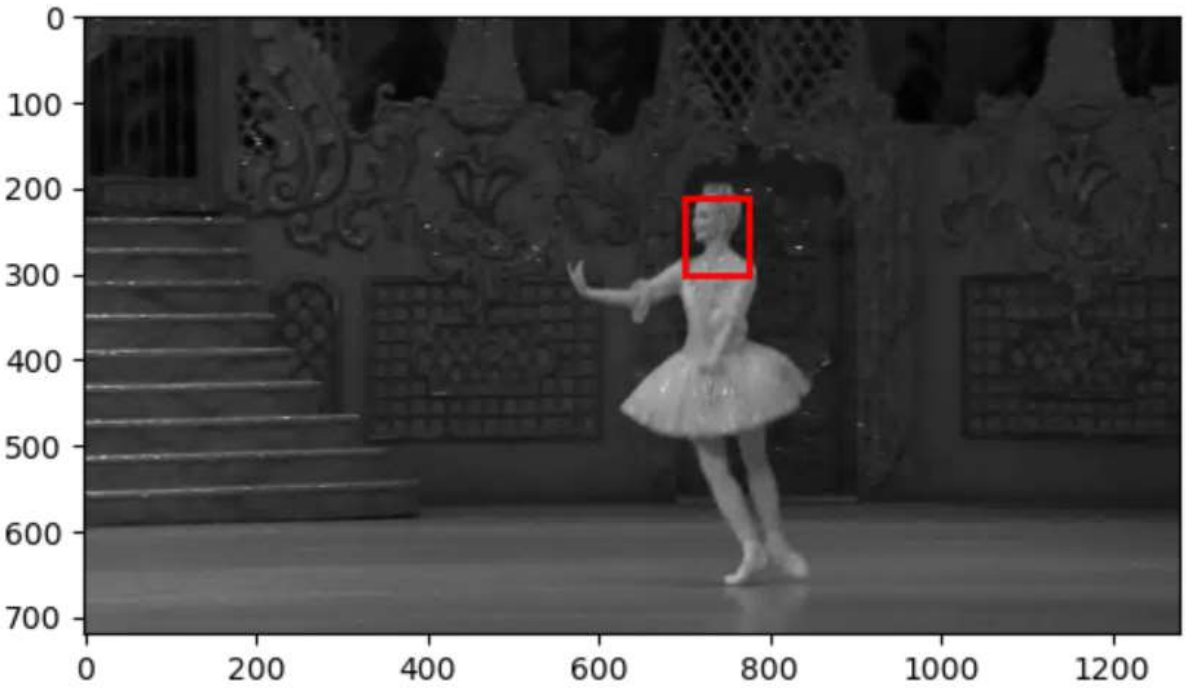

BELMONT PARK RACE 11
1/4 :24.06
NYRA.com
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
7 17
KEEN ICE
1 14
MUBTAAHIJ (IRE)
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

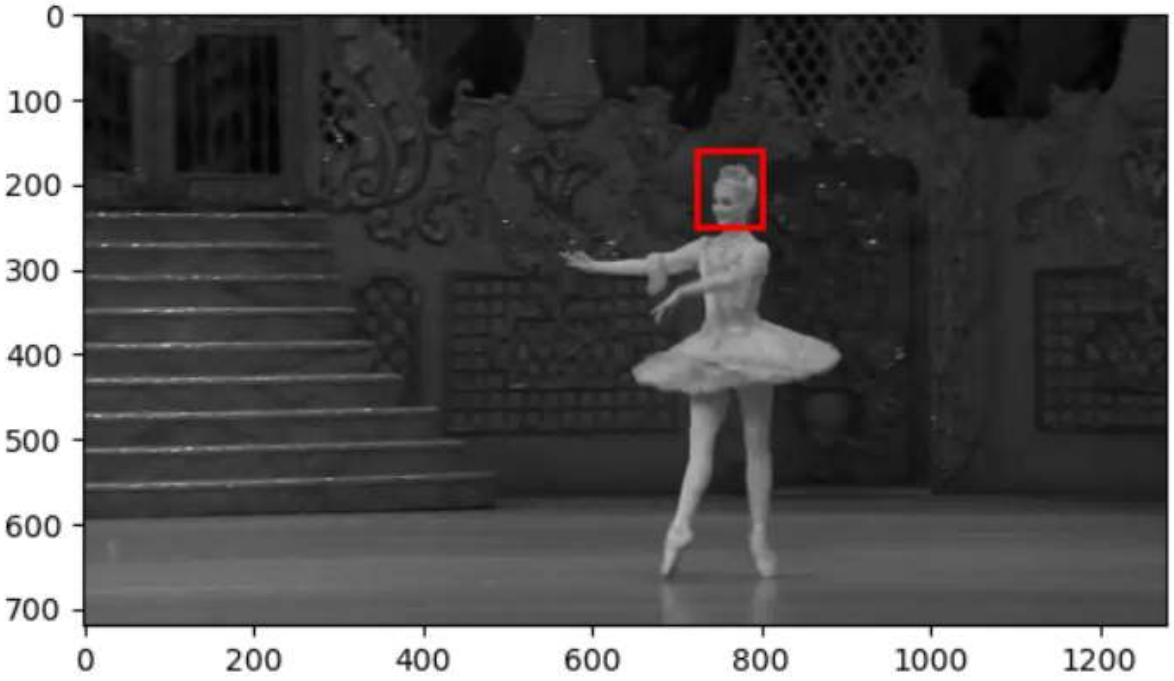

BELMONT PARK RACE 11
1/4 :24.06
NYRA.com
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
1 14
MUBTAAHIJ (IRE)
7 17
KEEN ICE
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

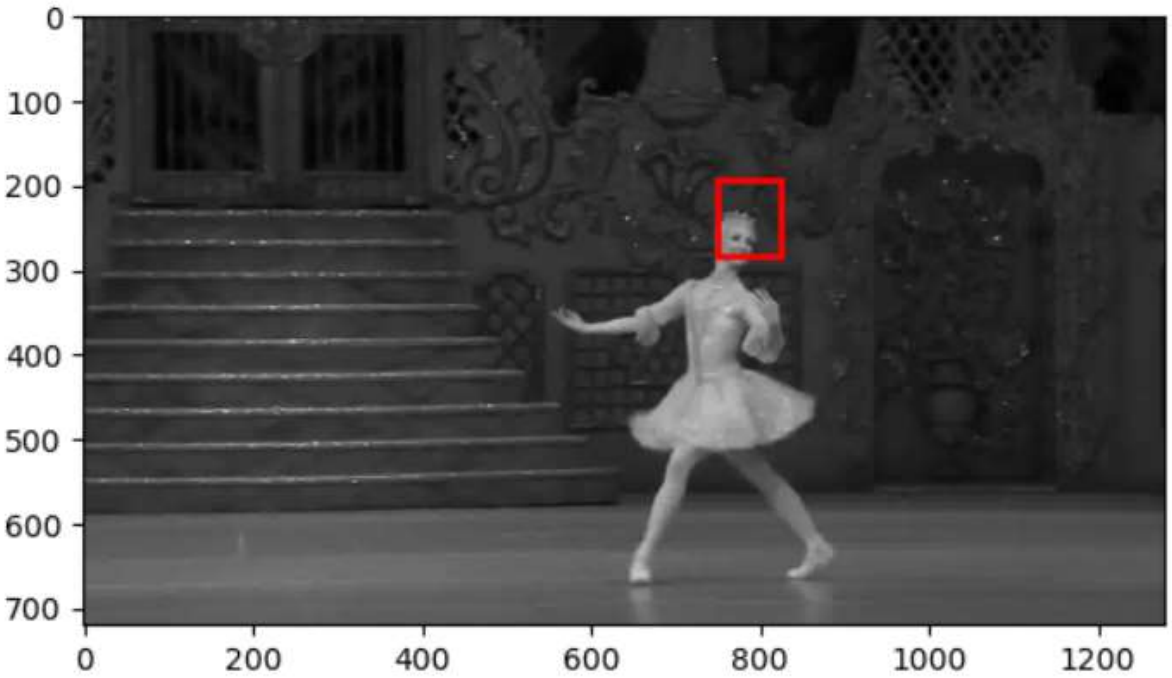

BELMONT PARK RACE 11
1/4 :24.06
NYRA.com
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
1 14
MUBTAAHIJ (IRE)
7 17
KEEN ICE
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

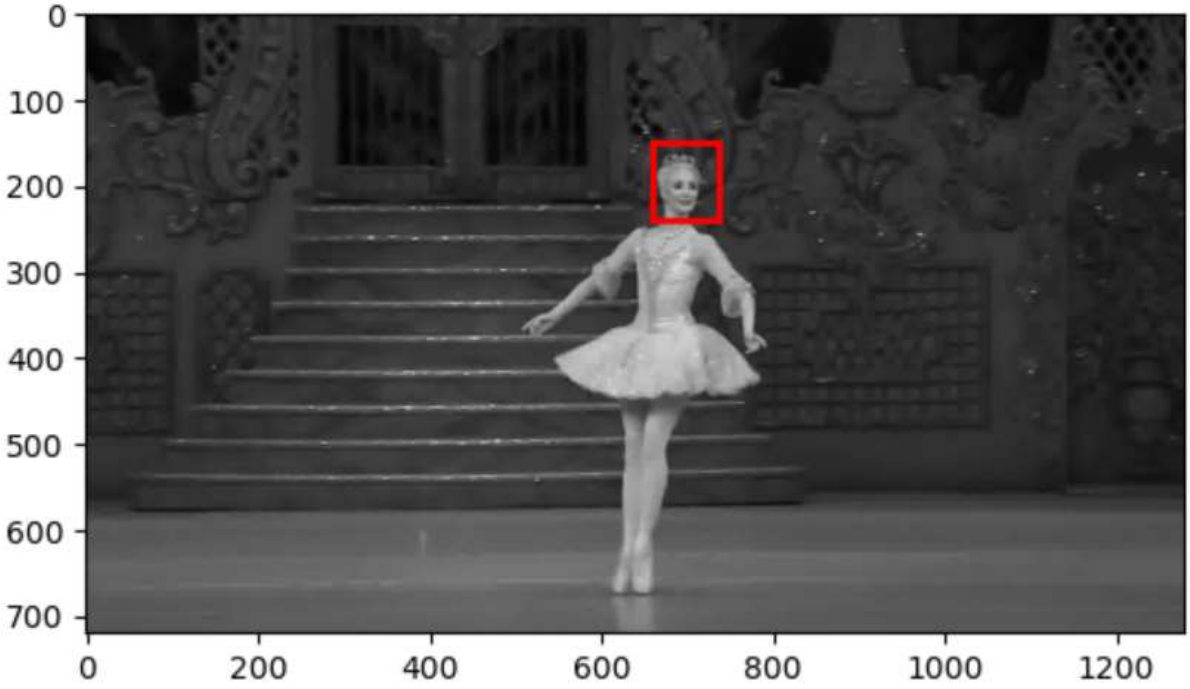

BELMONT PARK RACE 11
1/4 :24.06
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
7 17
KEEN ICE
1 14
MUBTAAHIJ (IRE)
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

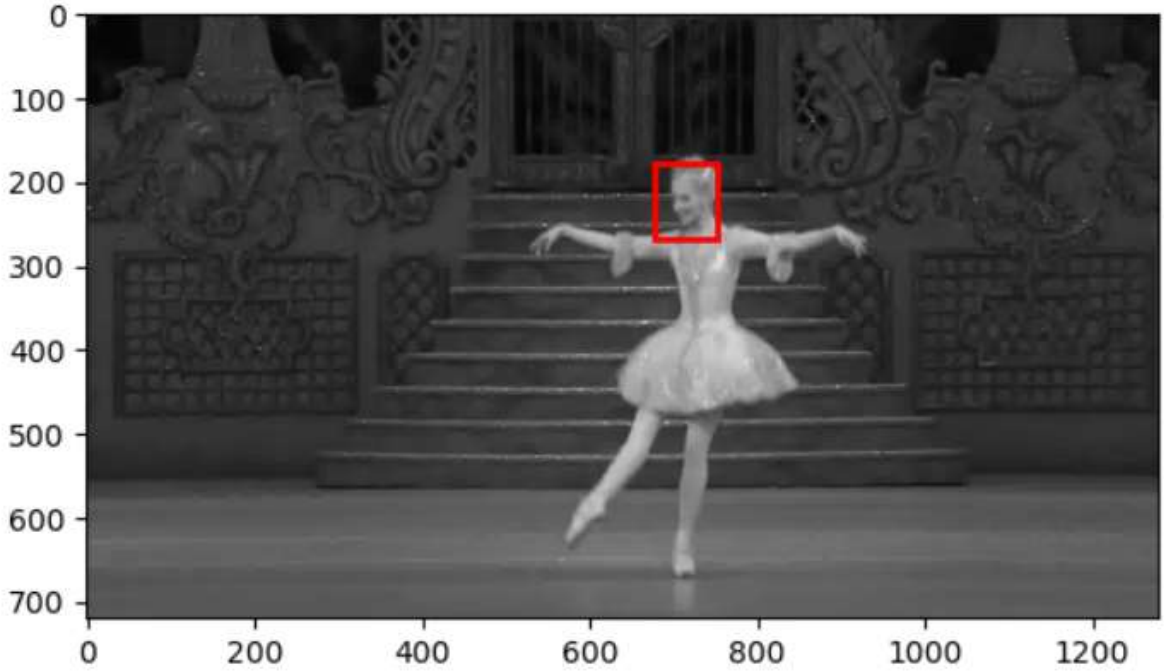
5. Ballet

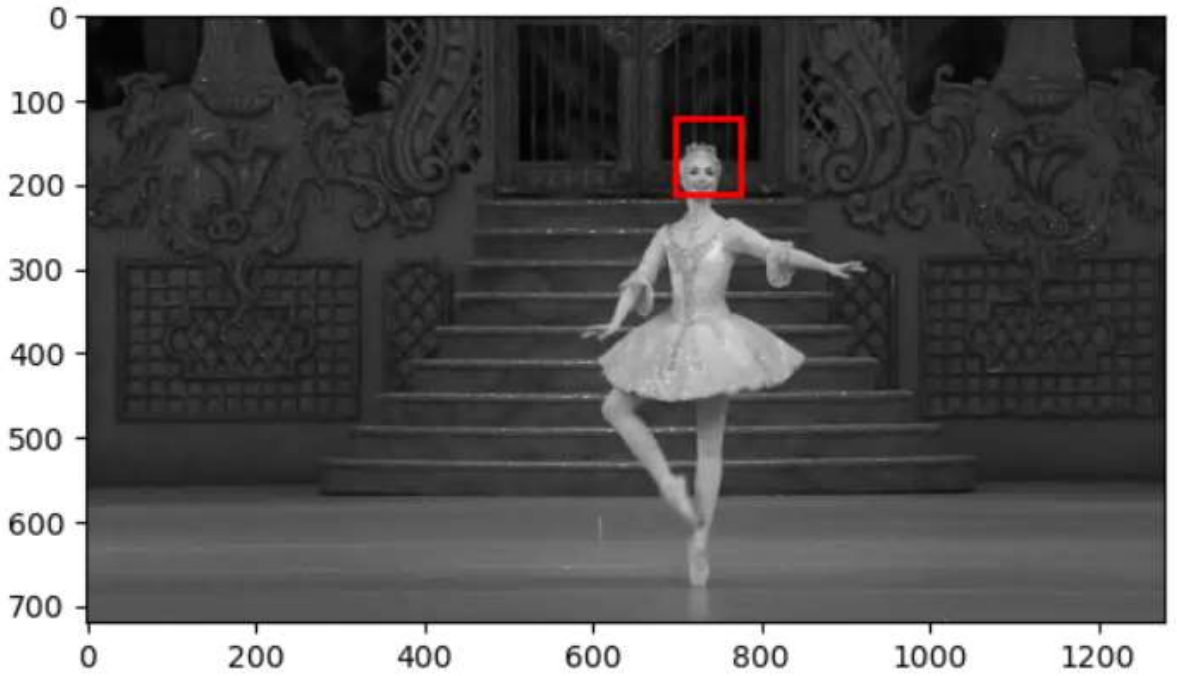

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

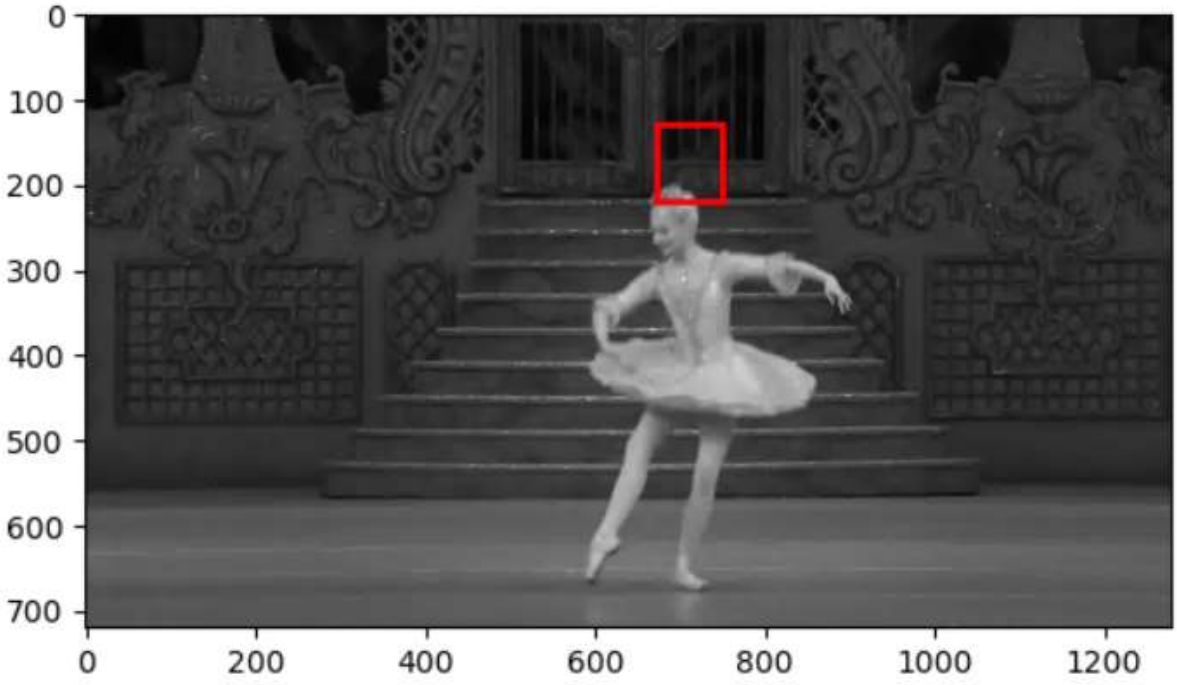

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

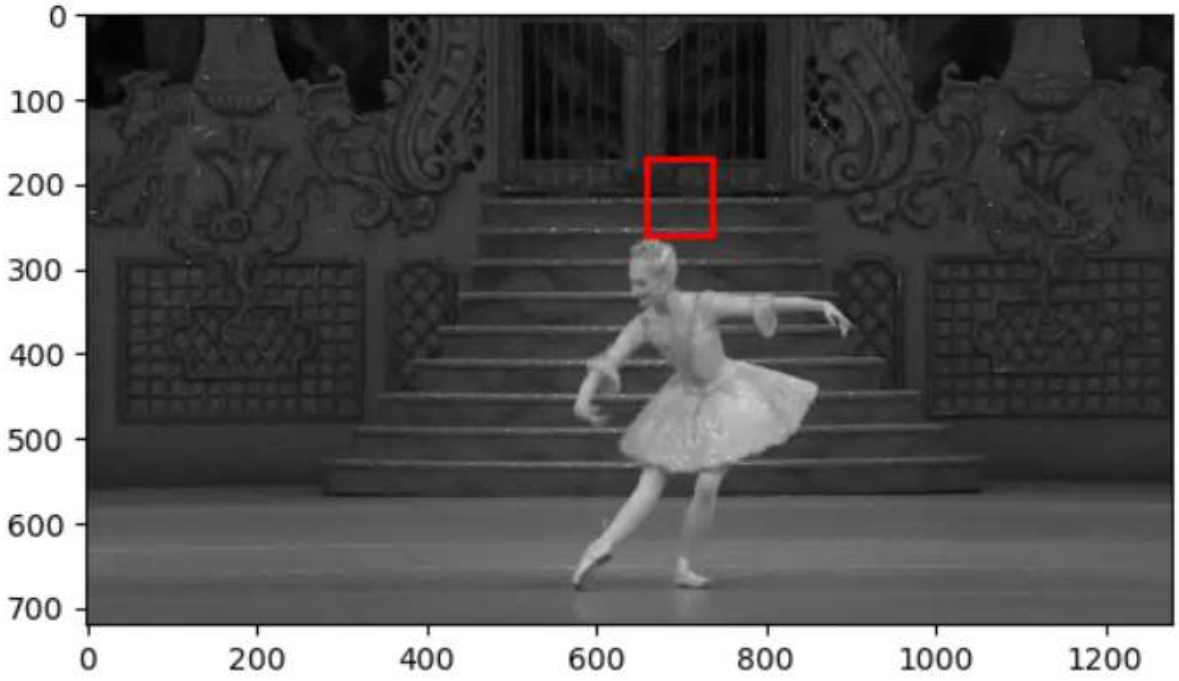

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

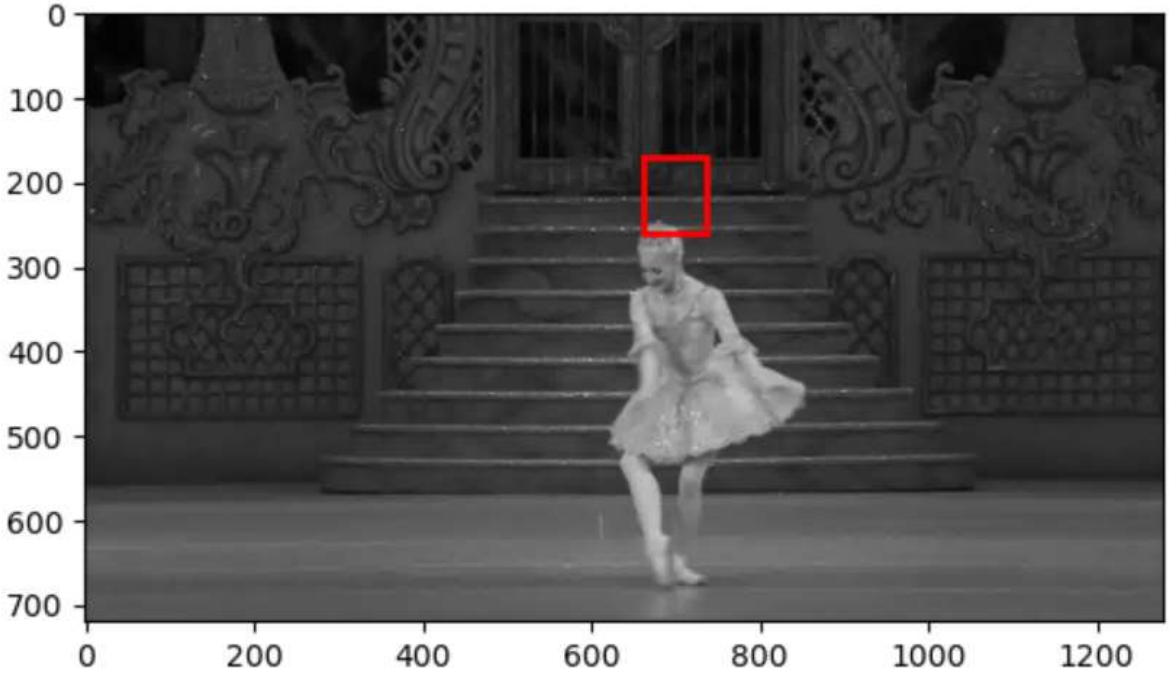

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

Q1.2

In [ ]:

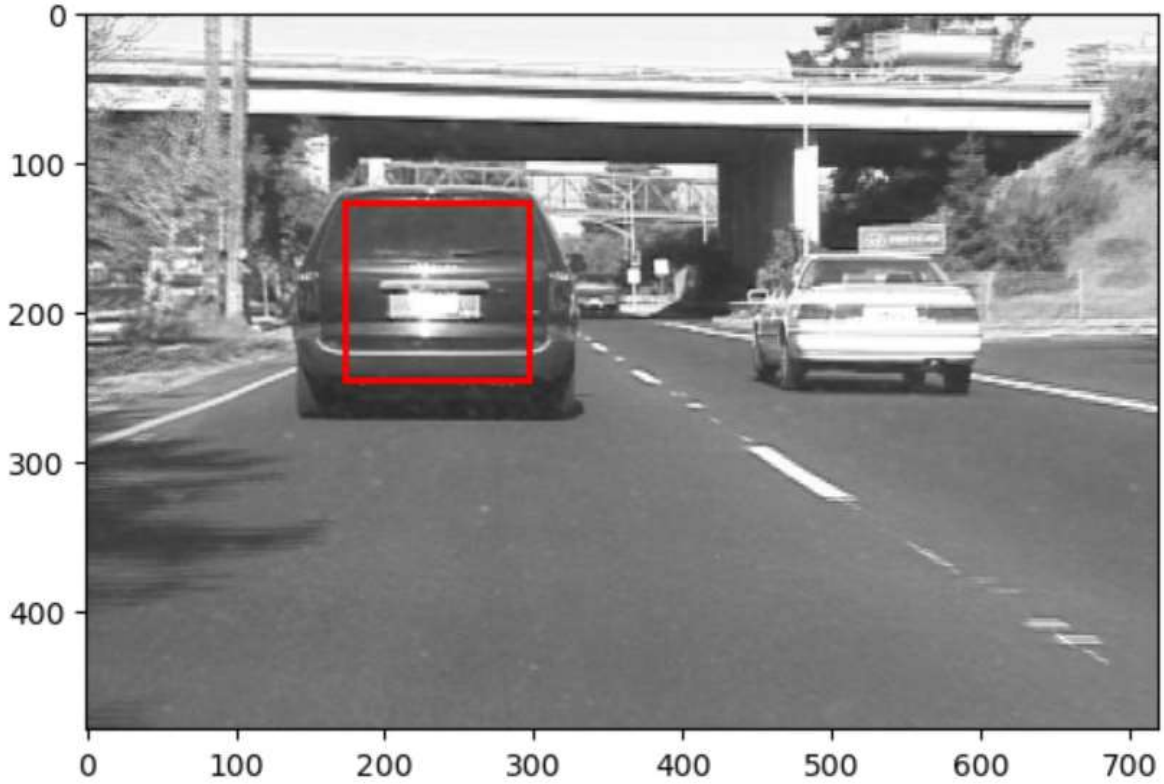
```
def LucasKanadeAffine(It, It1, rect, thresh=.025, maxIters=100):

    # threshold = thresh
    M = np.hstack((np.eye(2), np.zeros(2).reshape(-1, 1)))
    x1, y1, x2, y2 = rect

    s1 = np.arange(It.shape[0])
    s2 = np.arange(It.shape[1])
    s3 = np.arange(It1.shape[0])
    s4 = np.arange(It1.shape[1])

    interit = RectBivariateSpline(s1, s2, It)
    interit1 = RectBivariateSpline(s3, s4, It1)

    random_x = np.arange(x1, x2 + 0.5)
    random_y = np.arange(y1, y2 + 0.5)

    x, y = np.meshgrid(random_x, random_y)
    x = x.flatten()
    y = y.flatten()

    T_ev = interit.ev(y, x)
    coords = np.hstack((x.reshape(-1, 1), y.reshape(-1, 1)))

    for i in range(maxIters):

        coords_ = M @ (np.hstack((coords, np.ones(coords.shape[0]).reshape(-1, 1

        x1 = coords_[0].flatten()
        y1 = coords_[1].flatten()

        I_ev = interit1.ev(y1, x1)

        Ix = interit1.ev(y1, x1, dx=0, dy=1)
        Iy = interit1.ev(y1, x1, dx=1, dy=0)

        A = np.zeros((x1.shape[0], 6))

        A[:, 0] = x1 * Ix
        A[:, 1] = x1 * Iy
        A[:, 2] = y1 * Ix
        A[:, 3] = y1 * Iy
        A[:, 4] = Ix
        A[:, 5] = Iy

        b = T_ev - I_ev

        dp = np.linalg.lstsq(A, b, rcond=None)[0]
```

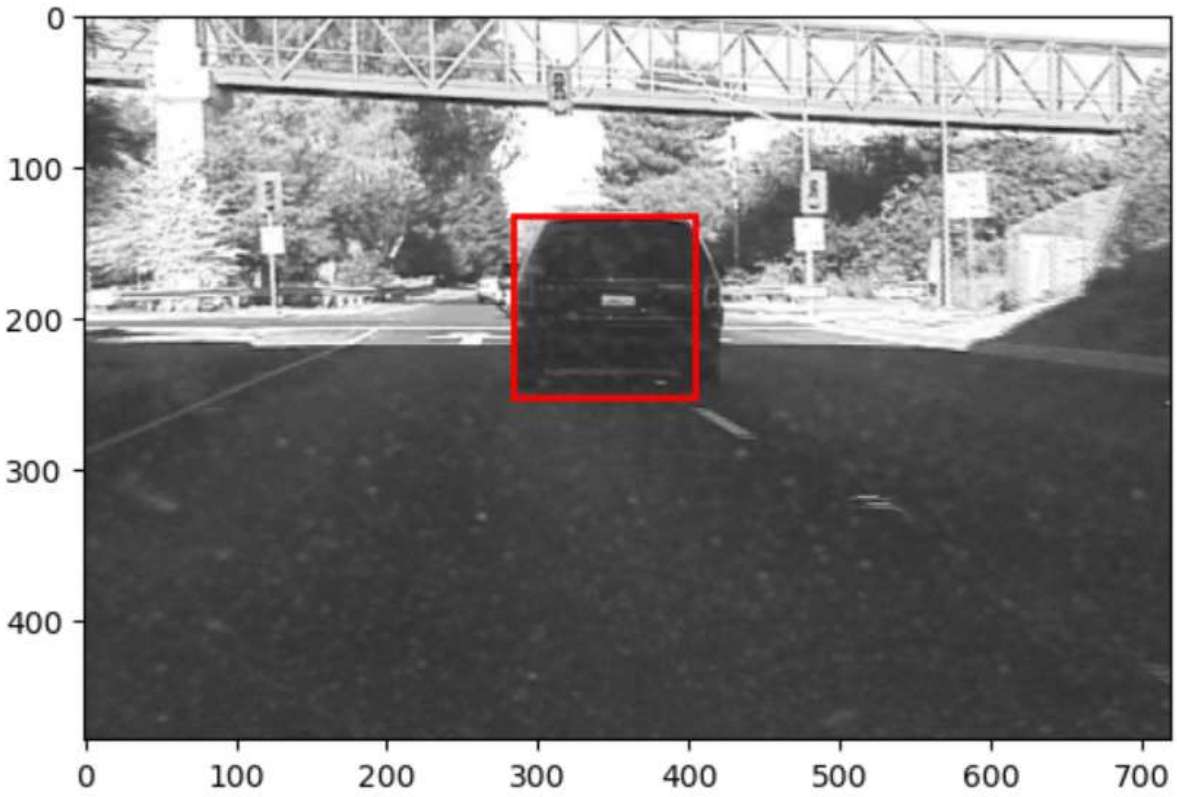
```
        M = M + dp.reshape(np.flip(M.shape)).T

        if np.sqrt(np.sum(dp ** 2)) <= thresh:
            break

    return M
```

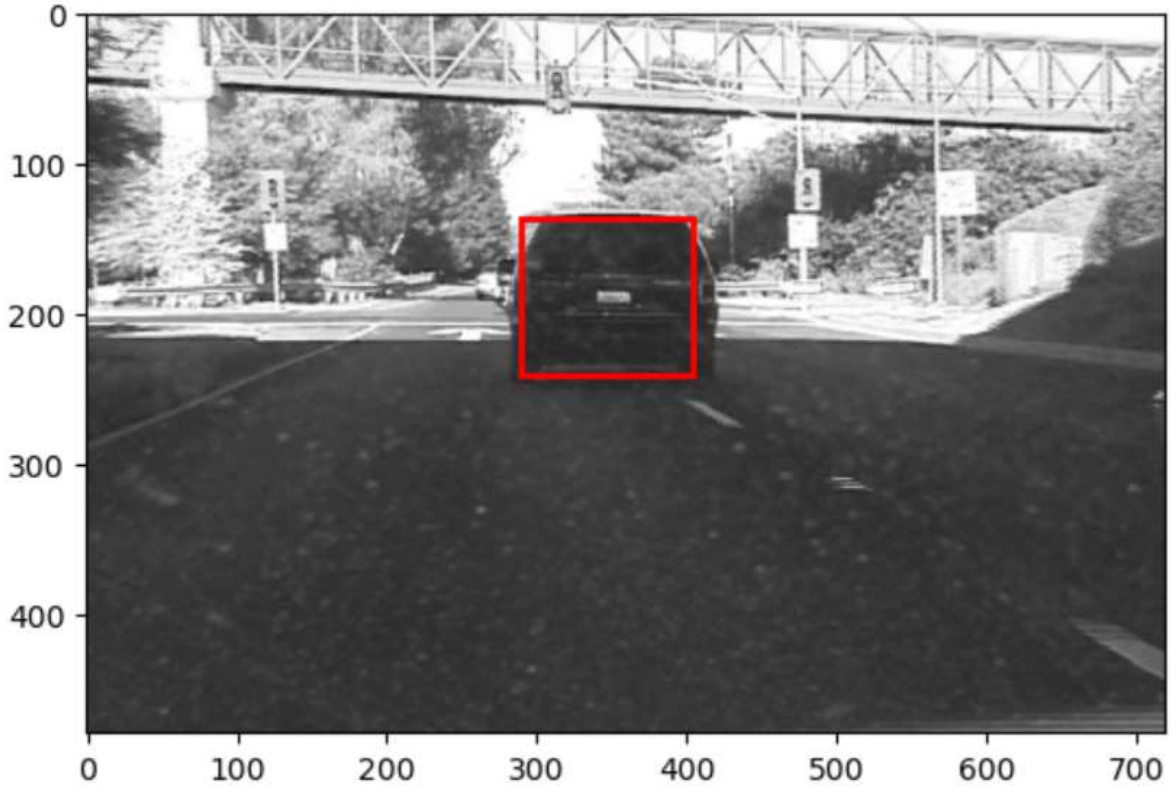
Results from each dataset -

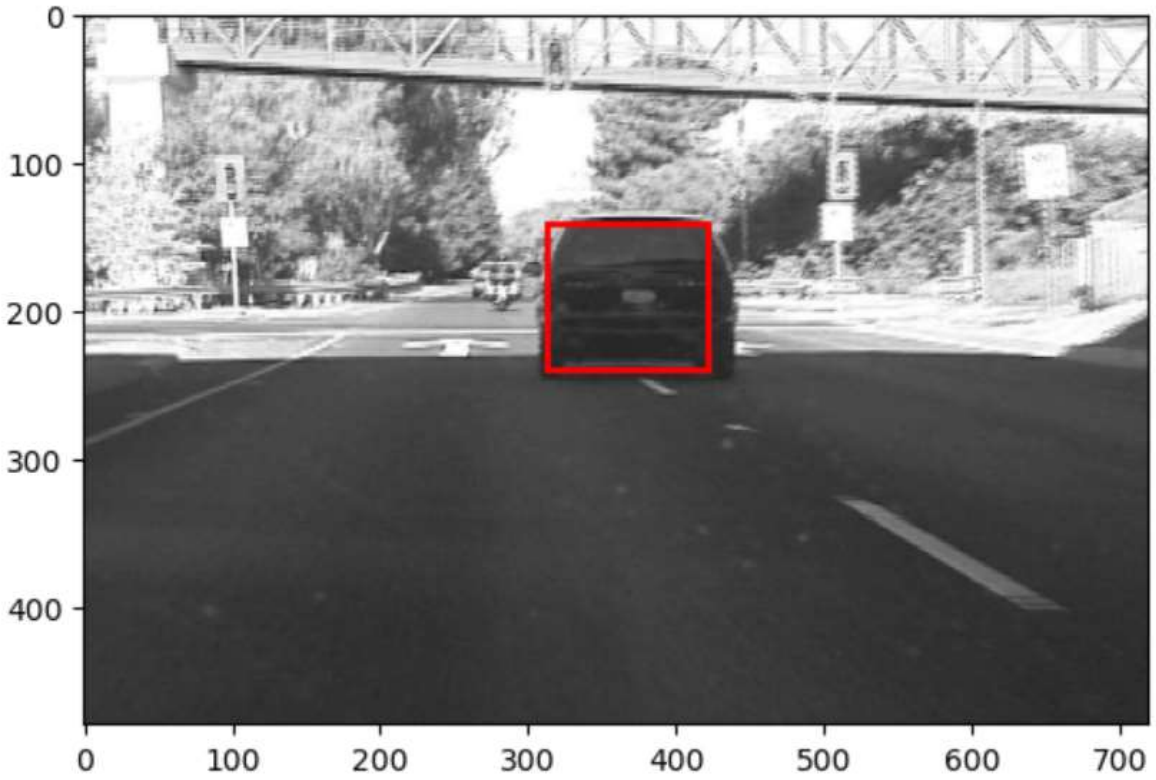
1. Car1


0
100
200
300
400
0
100
200
300
400
500
600
700

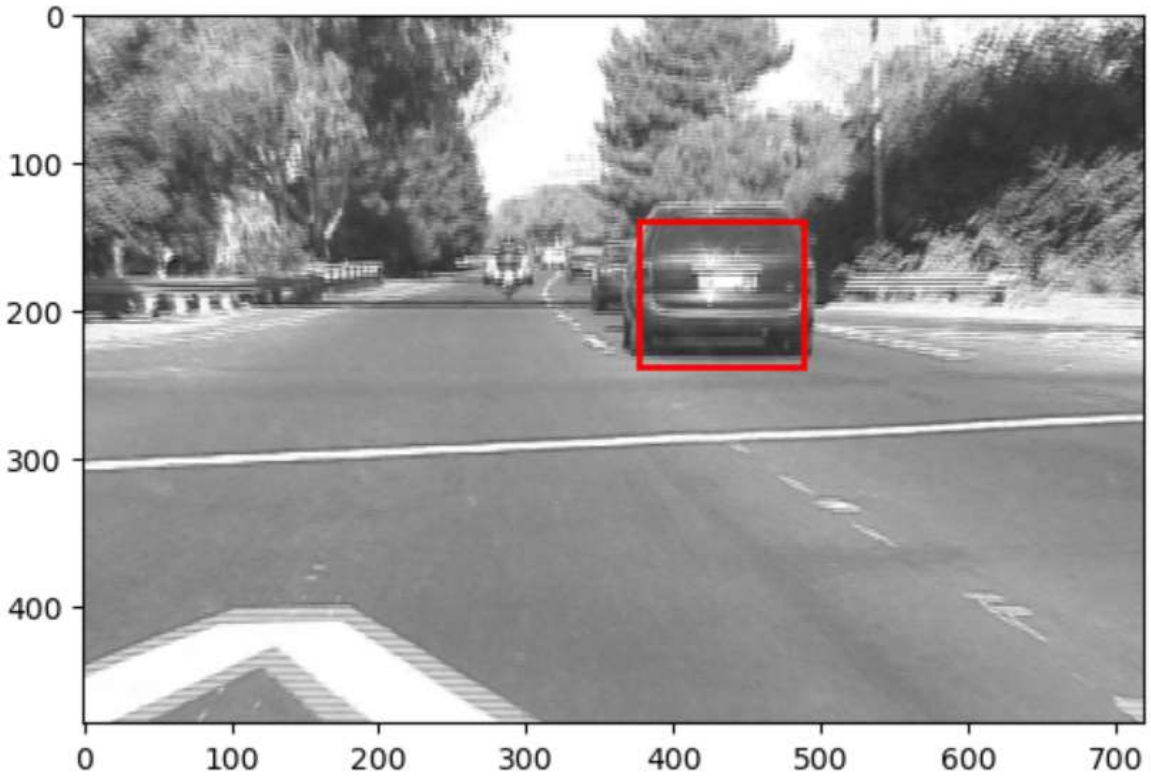

0
100
200
300
400
0
100
200
300
400
500
600
700


0
100
200
300
400
0
100
200
300
400
500
600
700


0
100
200
300
400
0
100
200
300
400
500
600
700

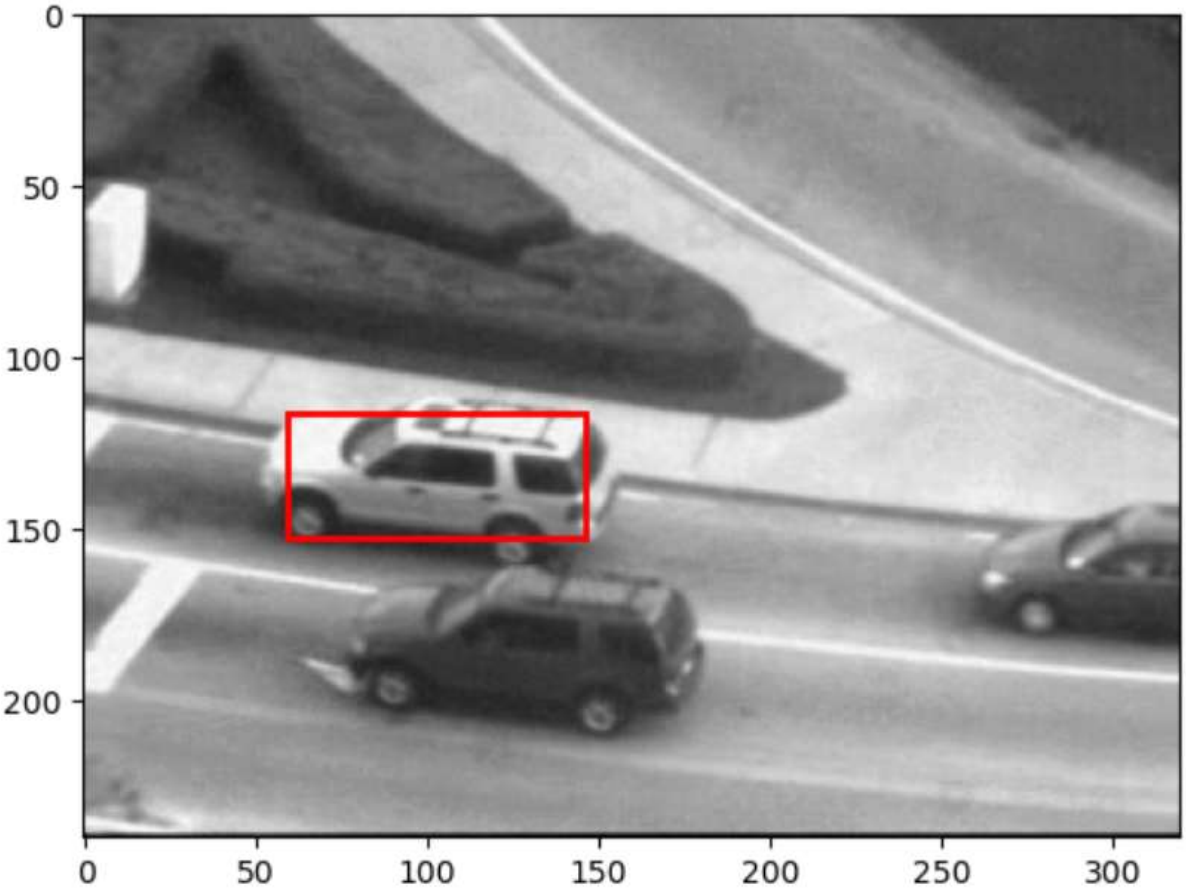

0
100
200
300
400
0
100
200
300
400
500
600
700


0
100
200
300
400
0
100
200
300
400
500
600
700

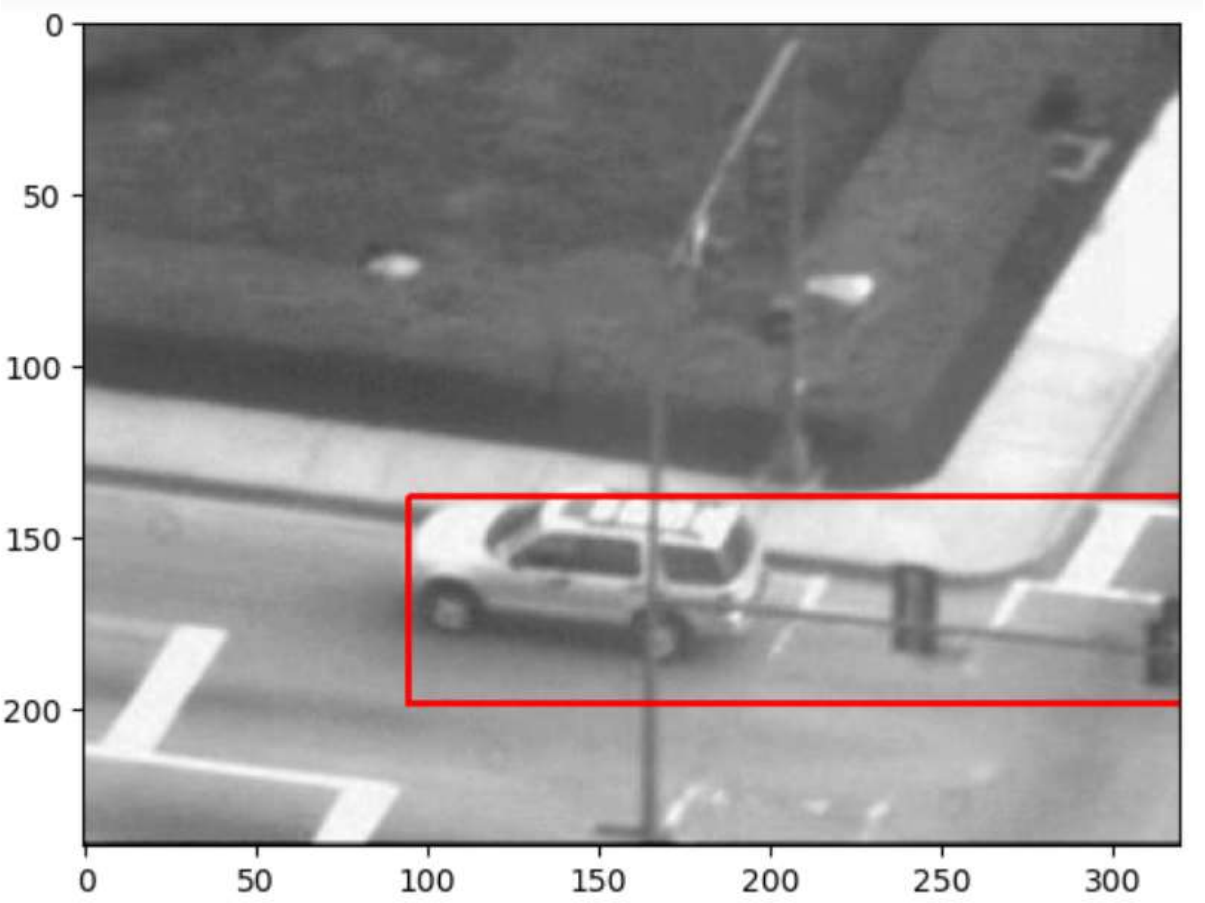

0
100
200
300
400
0
100
200
300
400
500
600
700

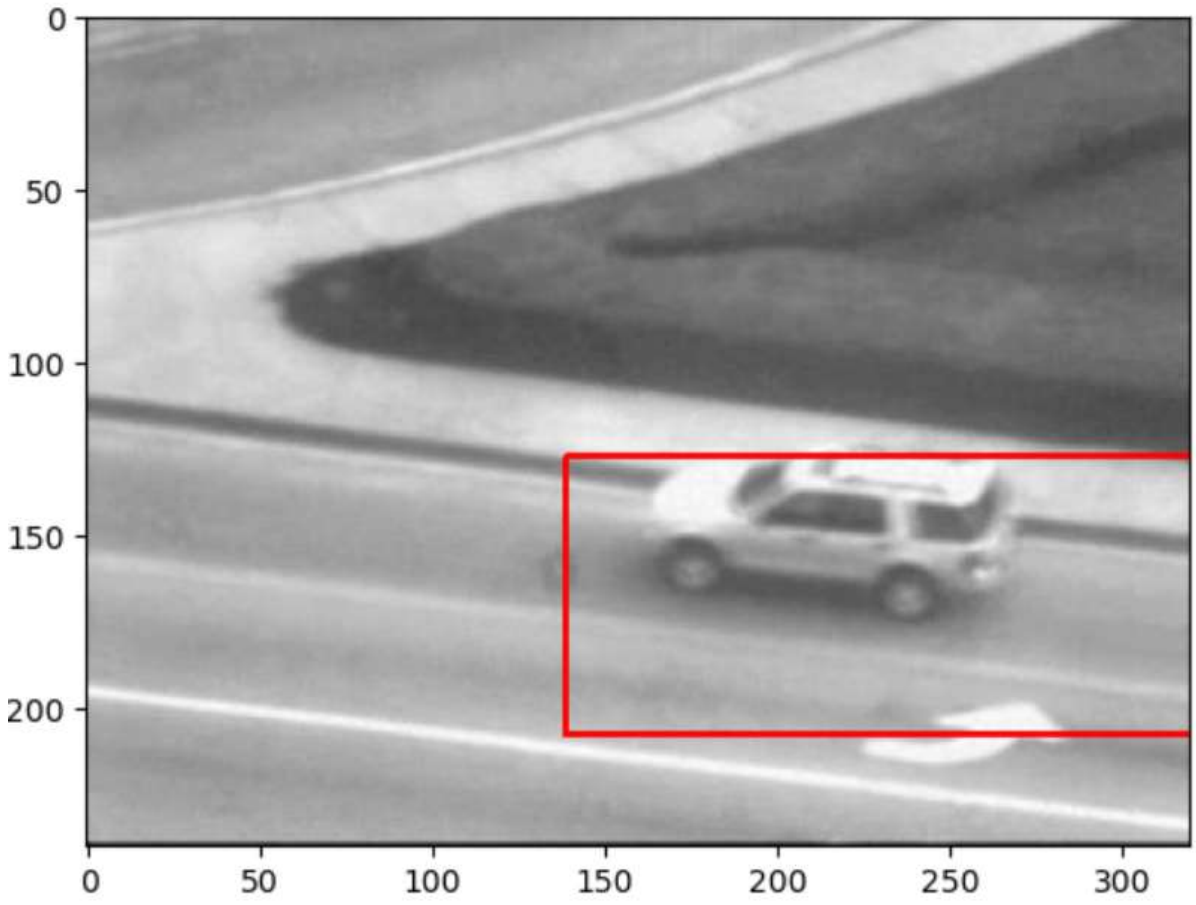

0
100
200
300
400
0
100
200
300
400
500
600
700

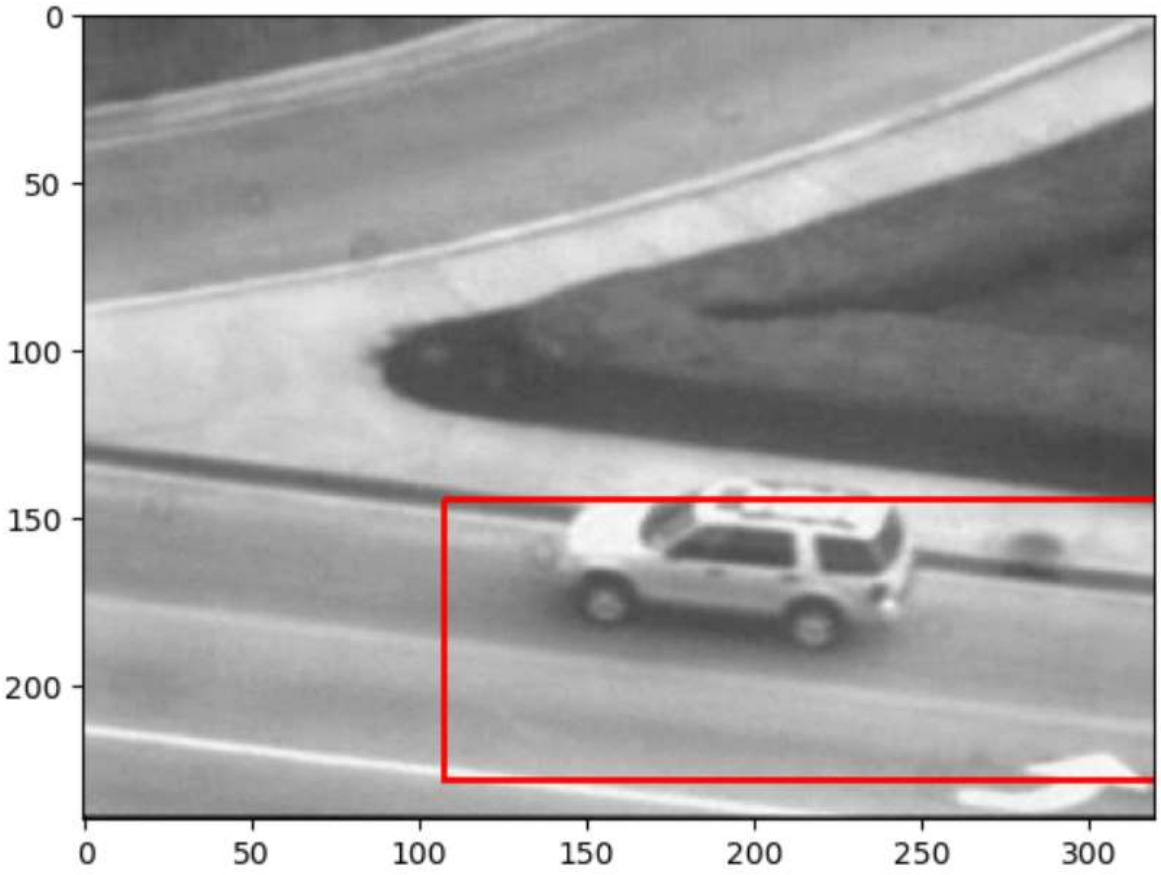
2. Car2

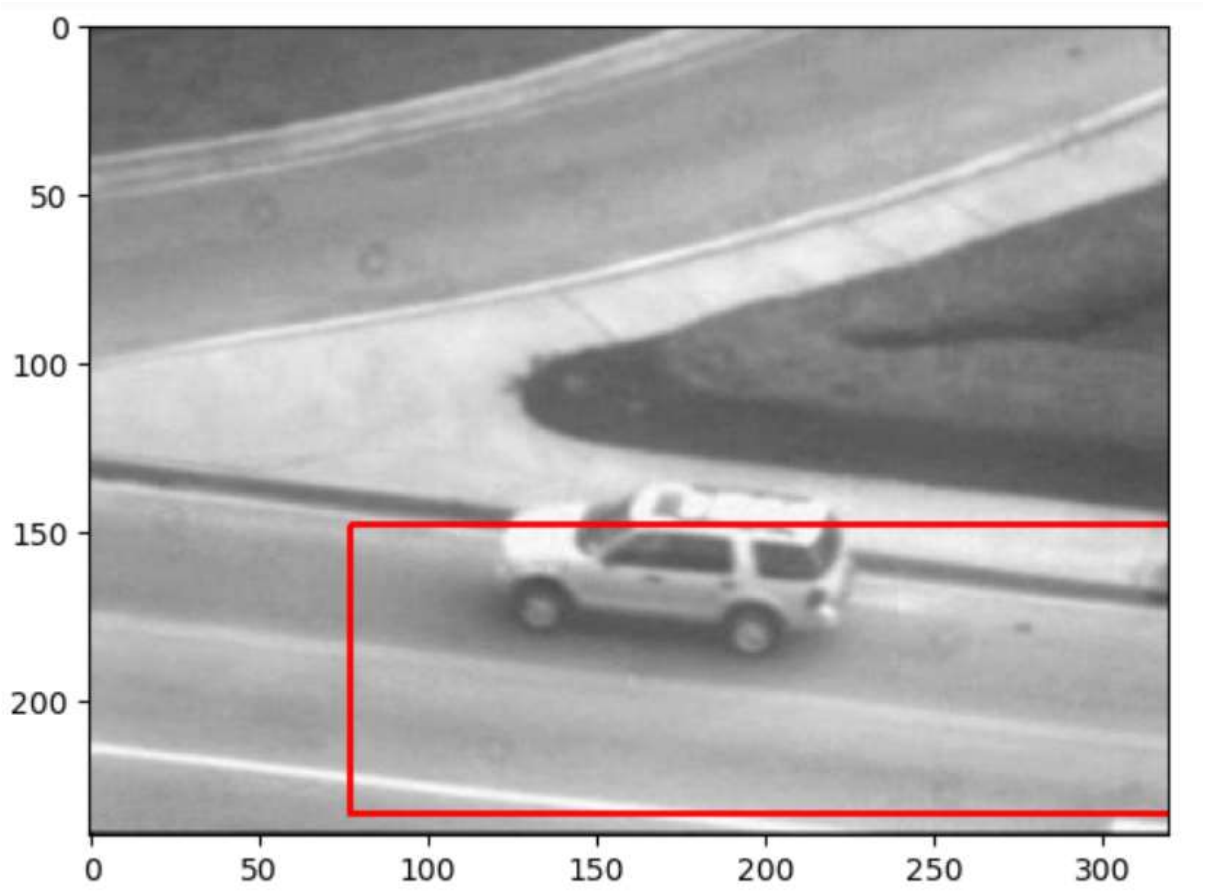



0
50
100
150
200
0
50
100
150
200
250
300


0
50
100
150
200
0
50
100
150
200
250
300


0
50
100
150
200
0
50
100
150
200
250
300

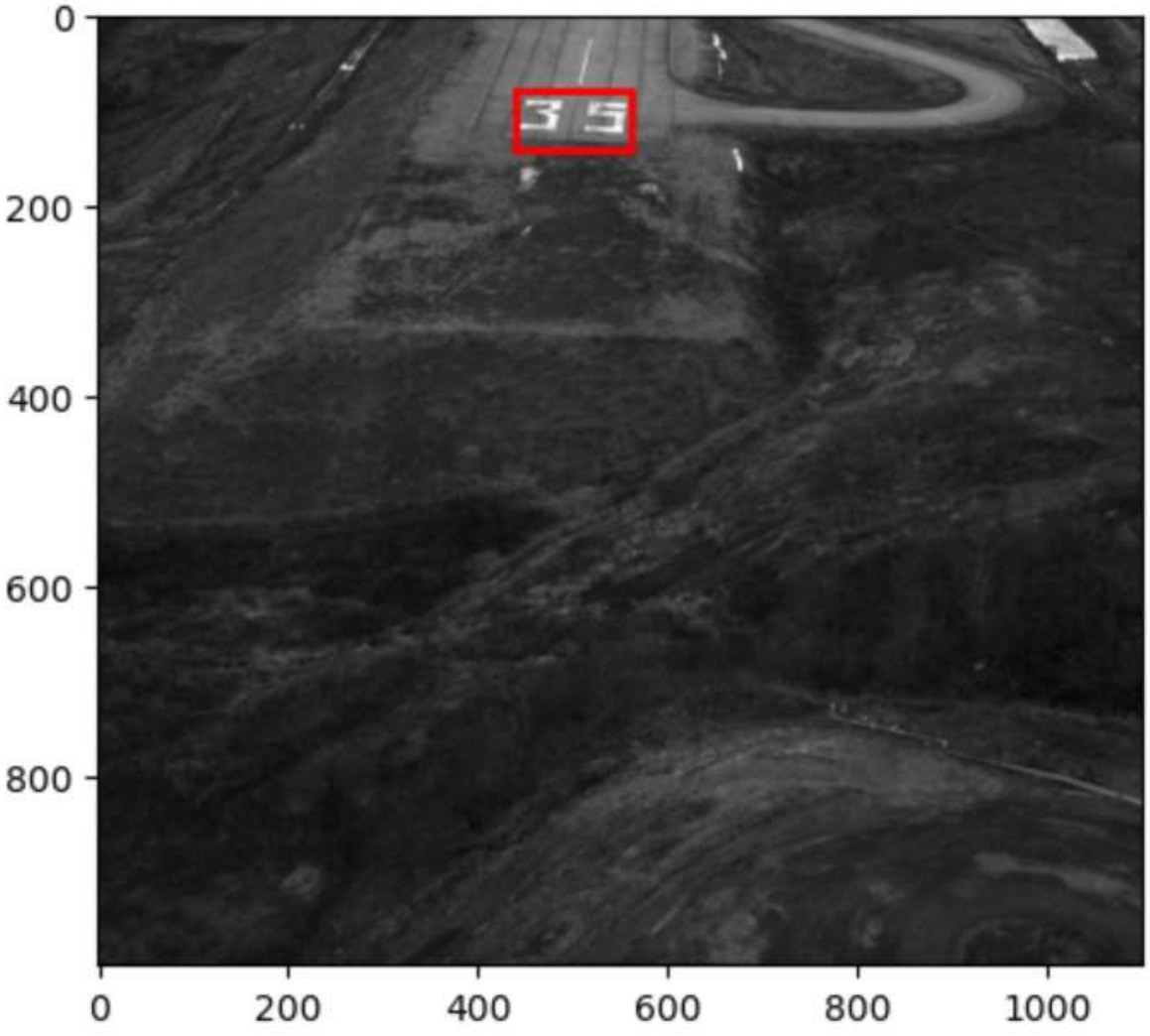

0
50
100
150
200
0
50
100
150
200
250
300

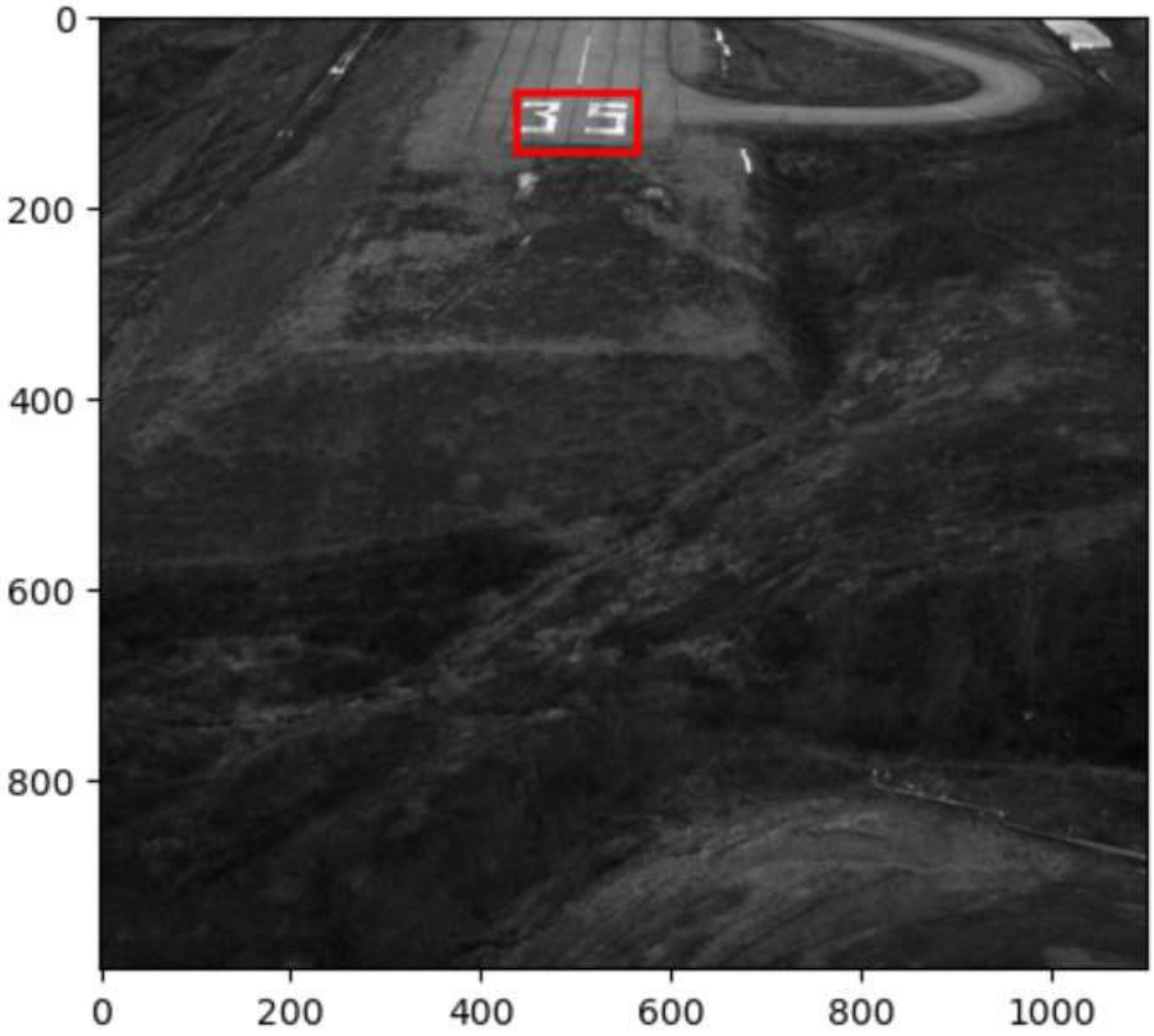

0
50
100
150
200
0
50
100
150
200
250
300

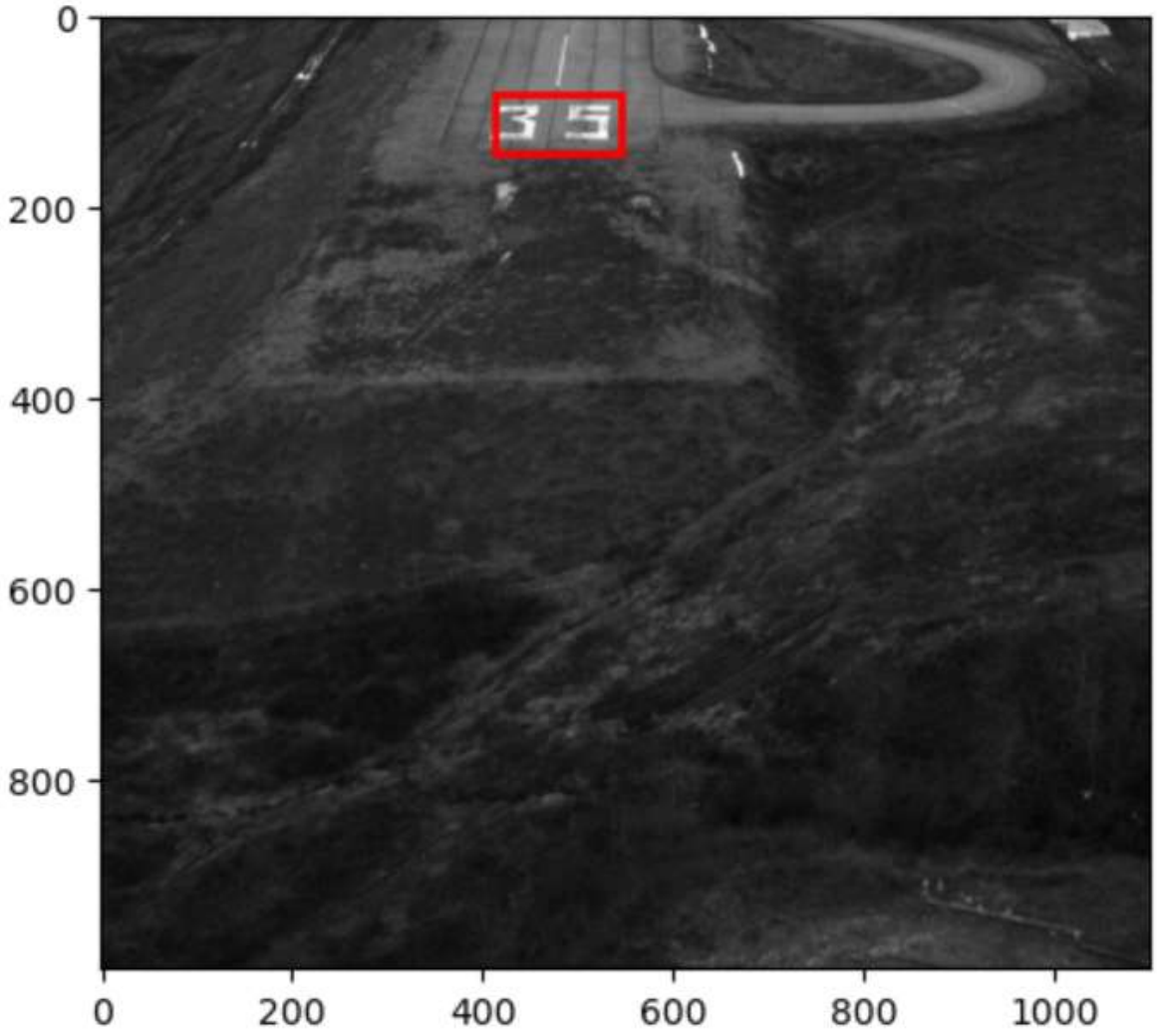

0
50
100
150
200
0
50
100
150
200
250
300

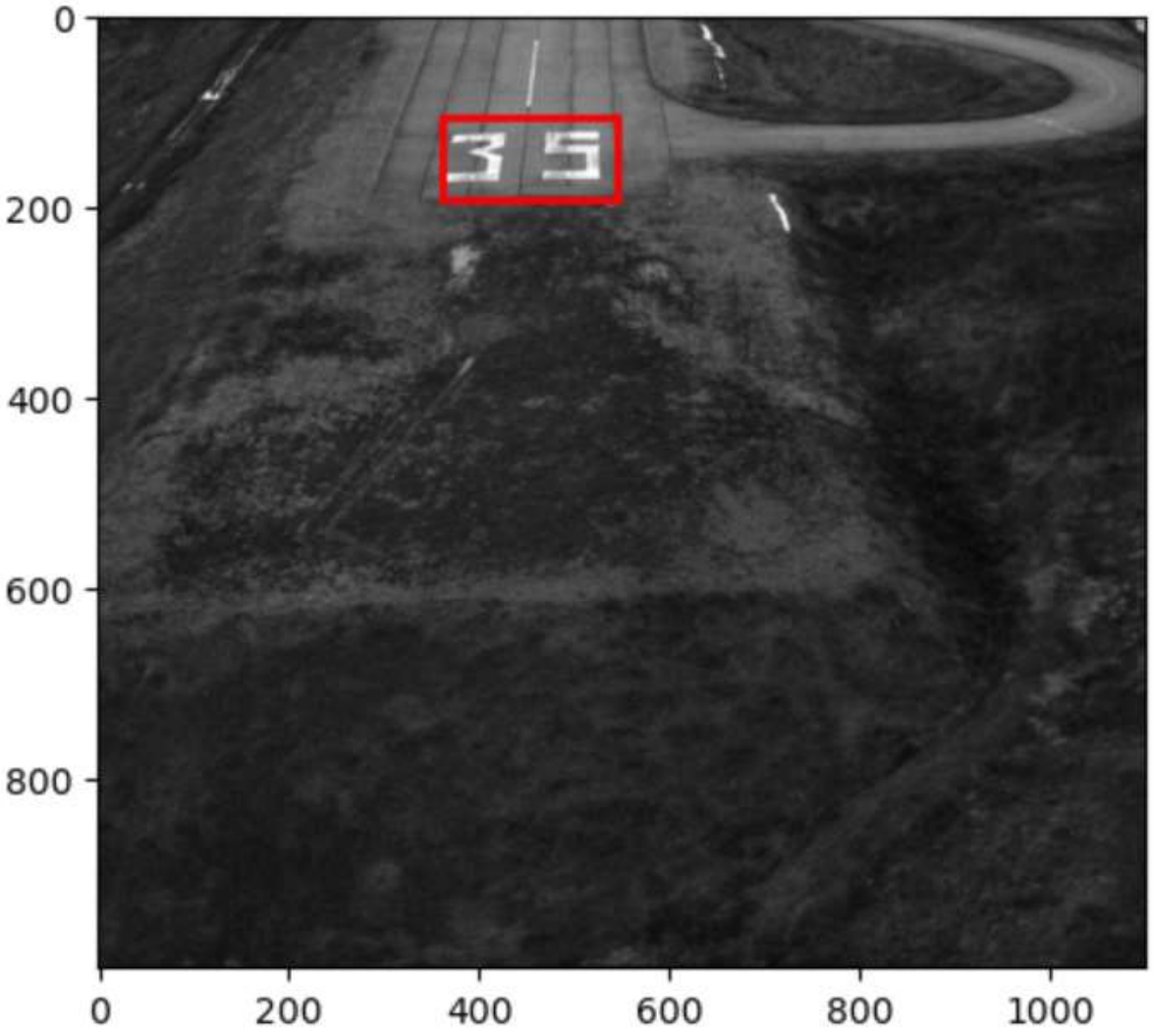

0
50
100
150
200
0
50
100
150
200
250
300

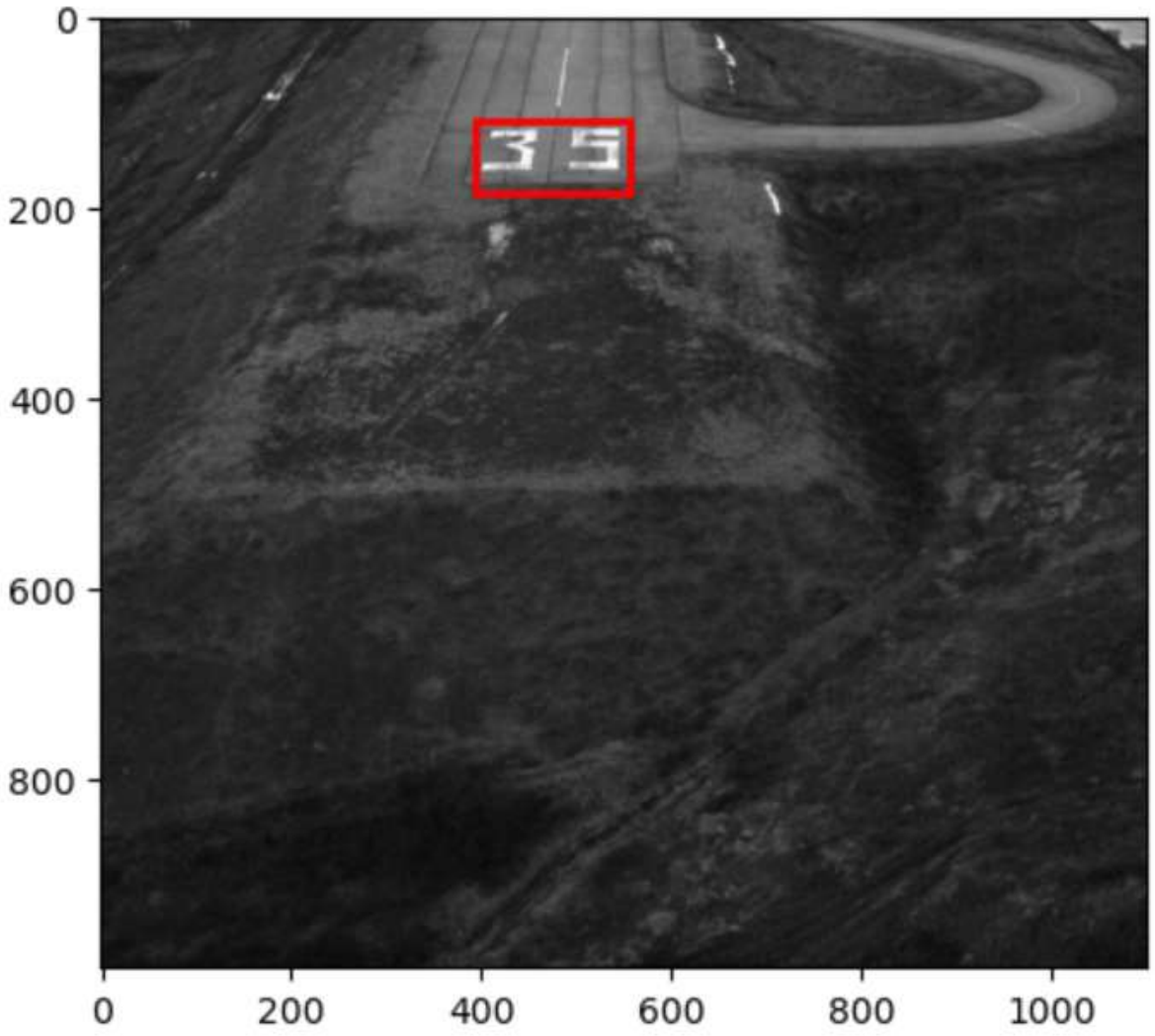
3. Landing

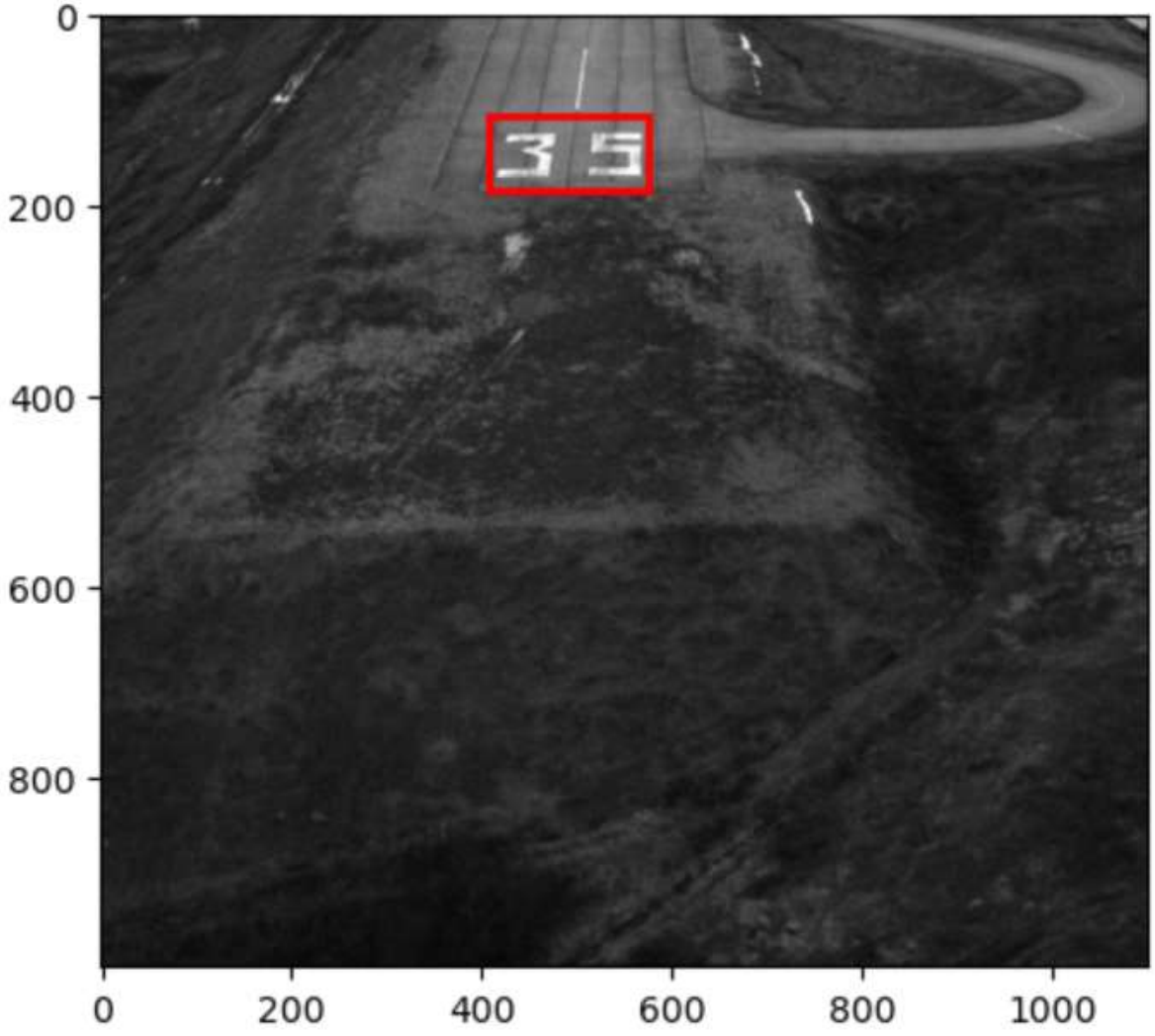

0
200
400
600
800
0
200
400
600
800
1000

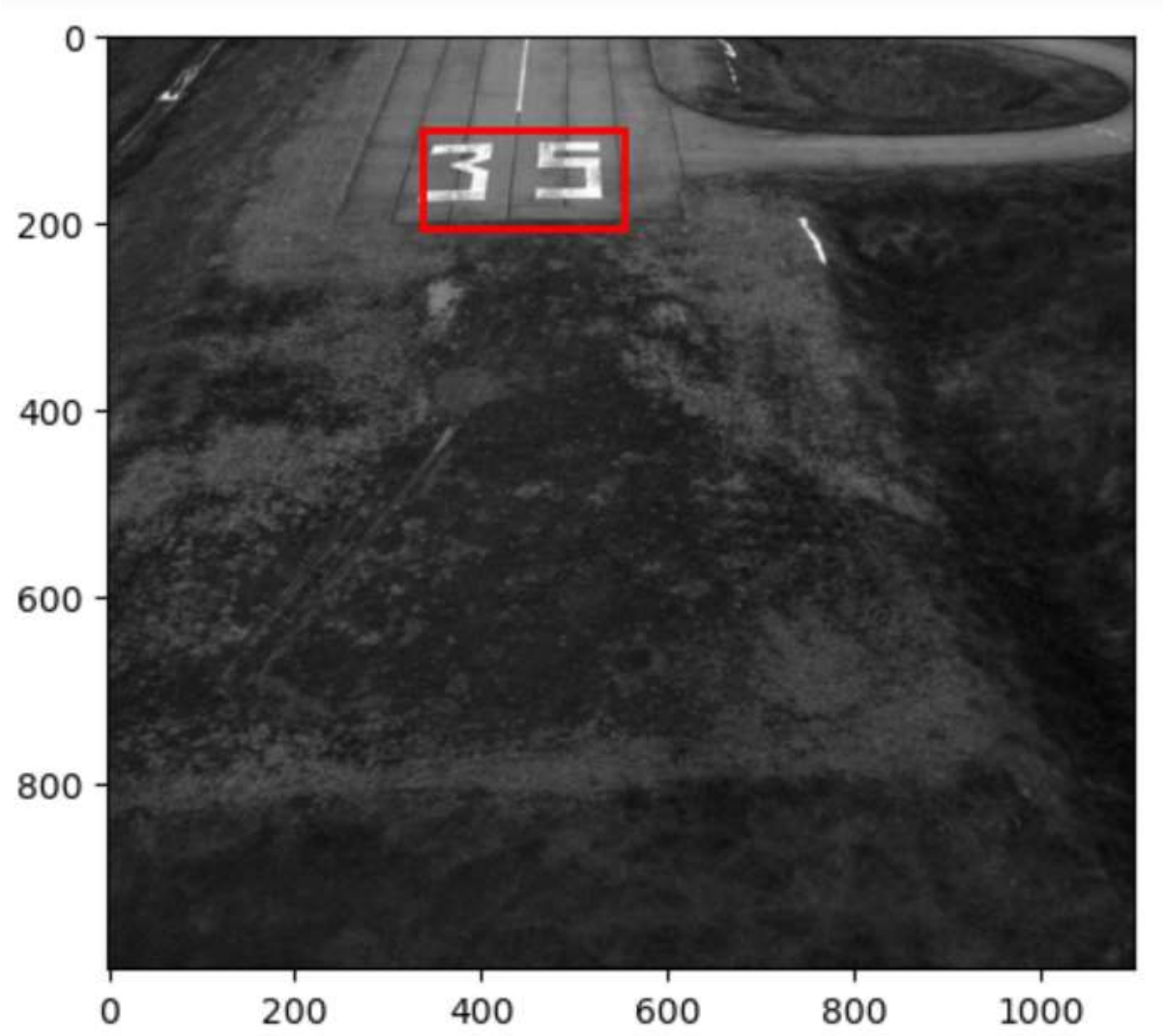

0
200
400
600
800
0
200
400
600
800
1000

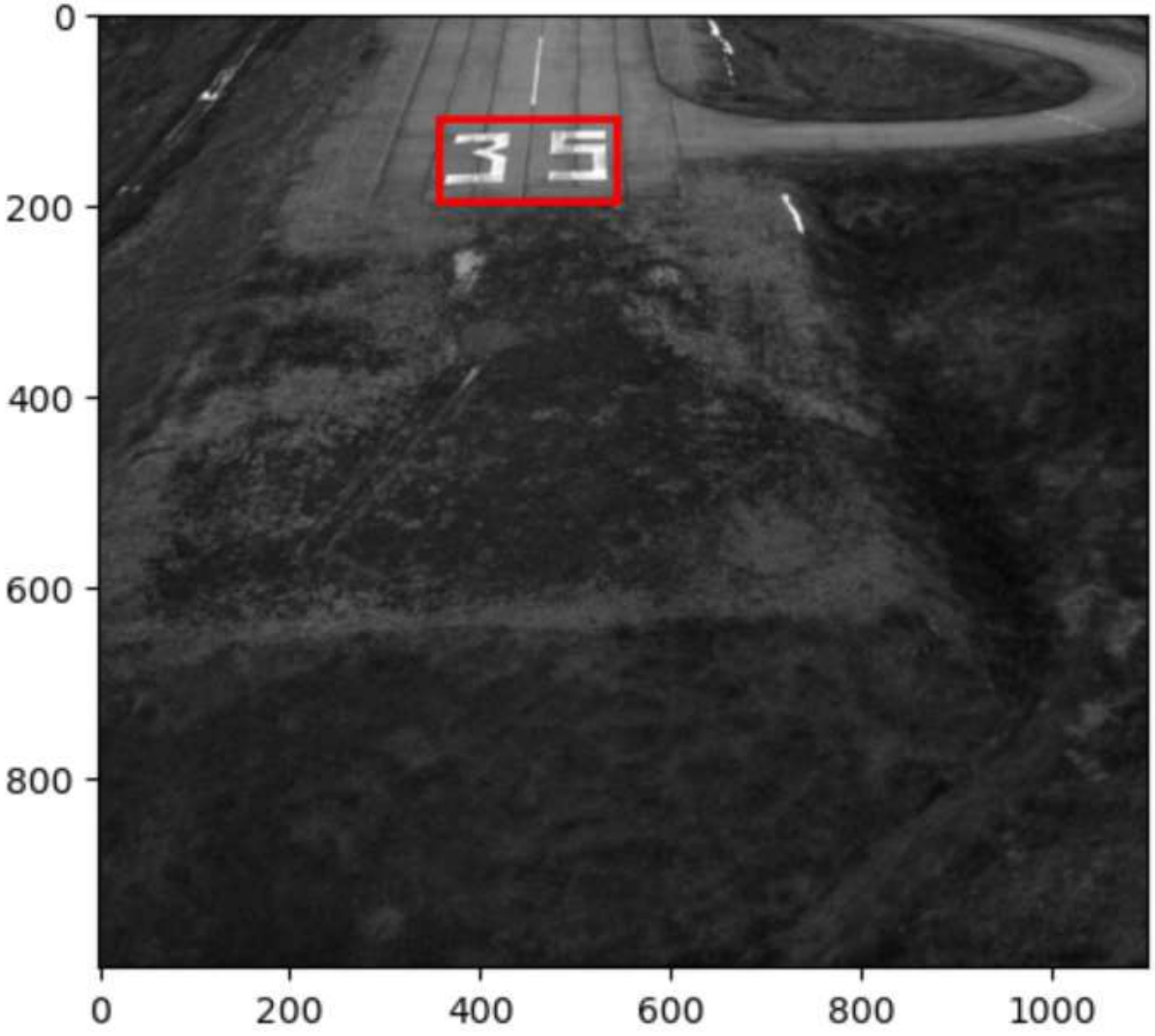

0
200
400
600
800
0
200
400
600
800
1000

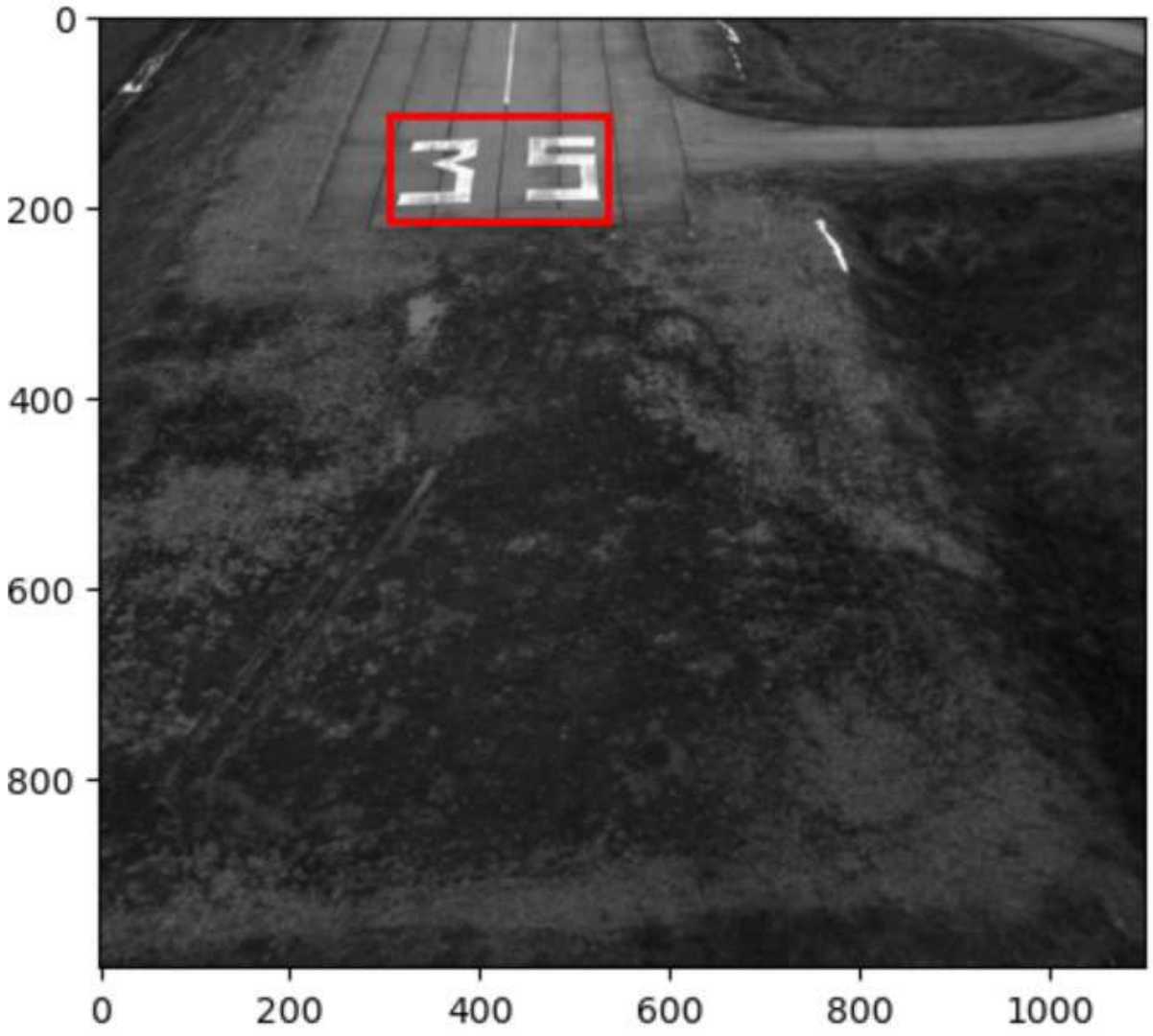

0
200
400
600
800
0
200
400
600
800
1000


0
200
400
600
800
0
200
400
600
800
1000

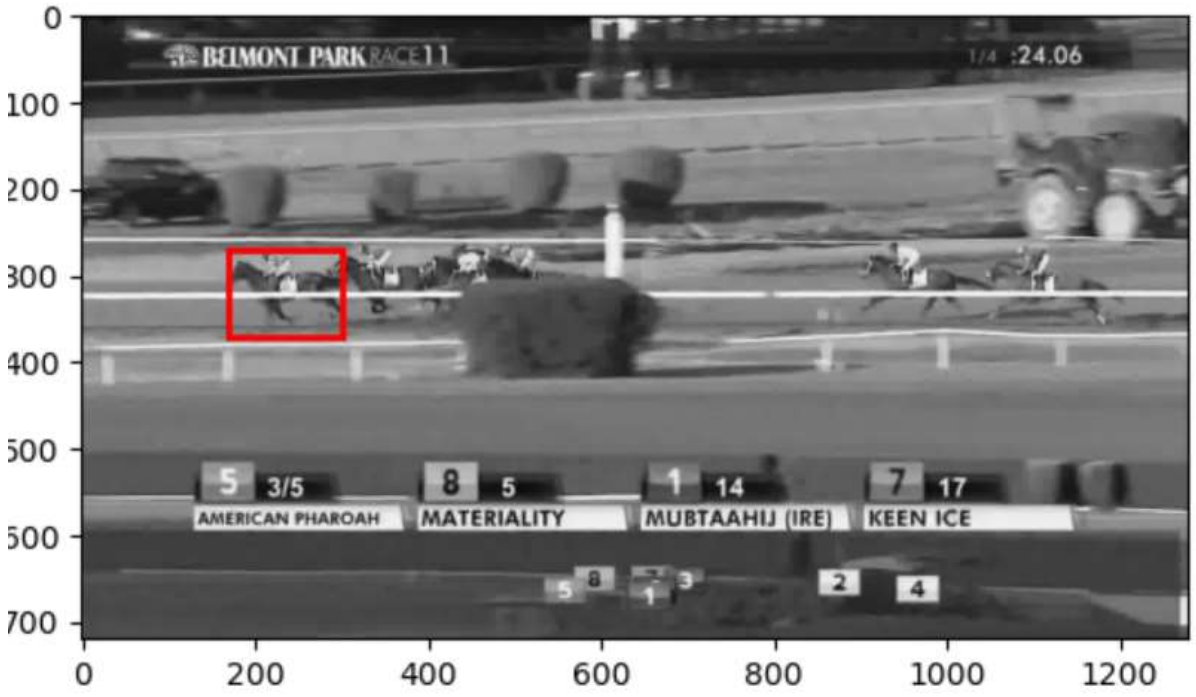

0
200
400
600
800
0
200
400
600
800
1000

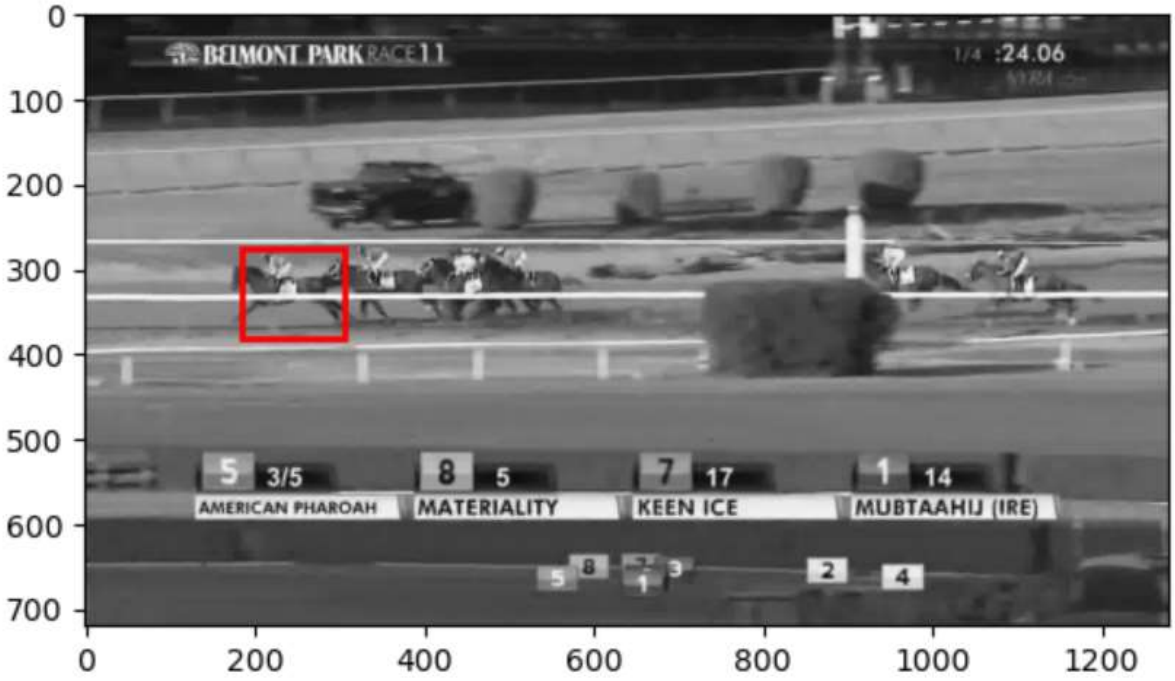

0
200
400
600
800
0
200
400
600
800
1000

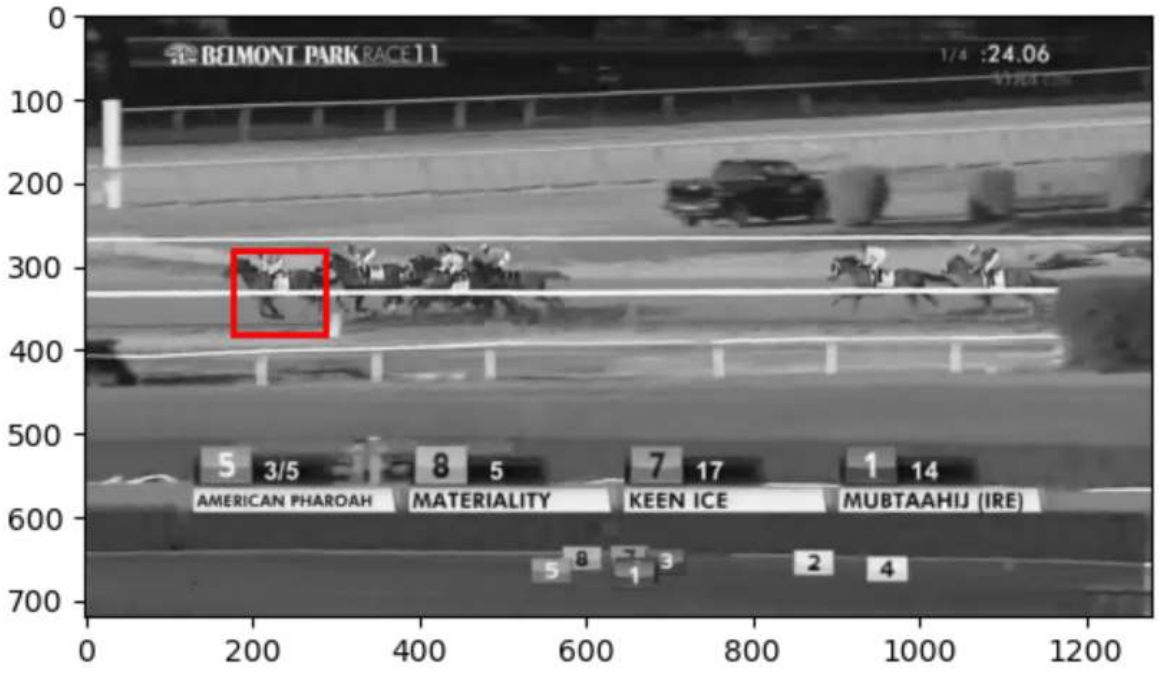

0
200
400
600
800
0
200
400
600
800
1000

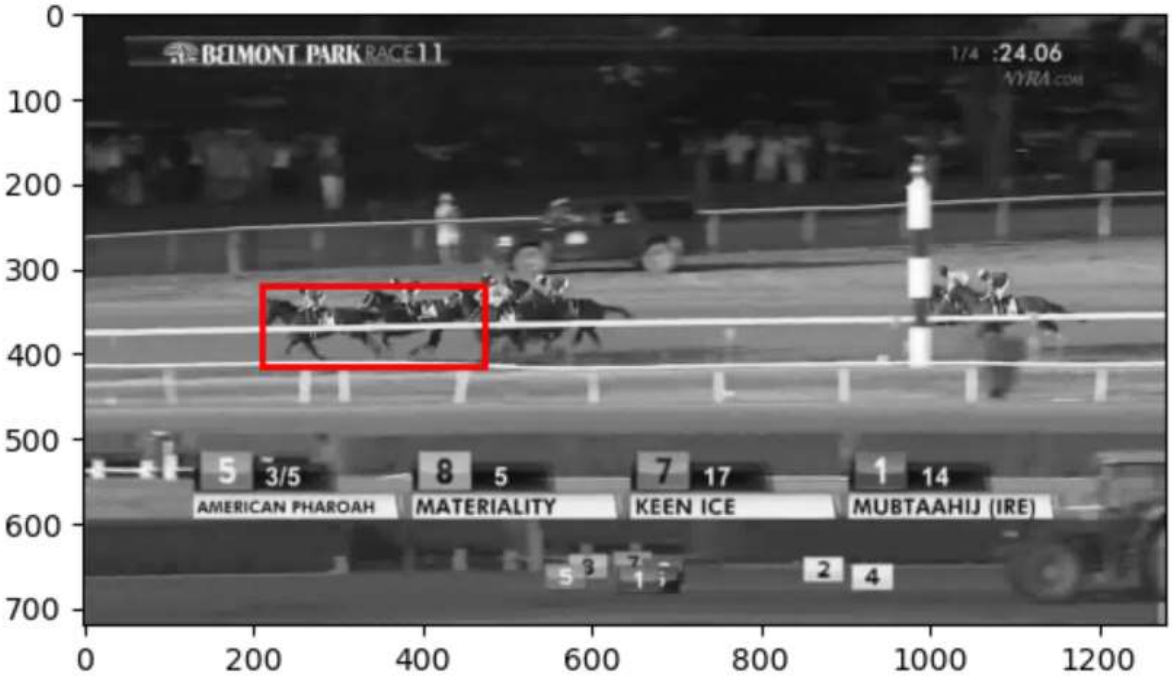


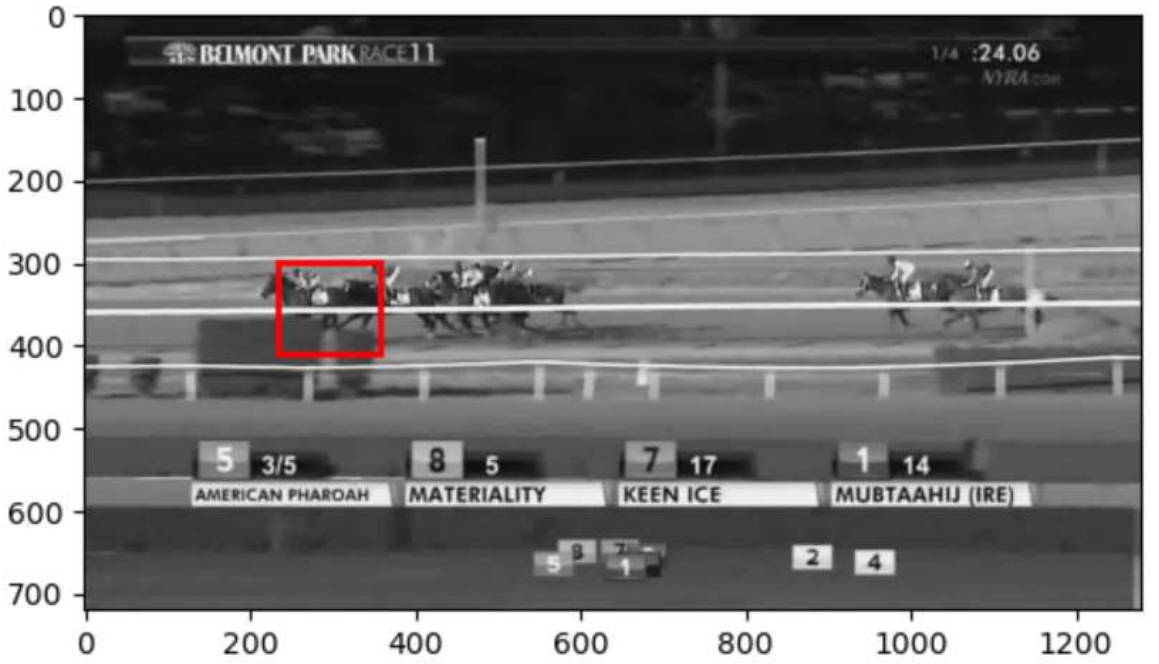
4. Race

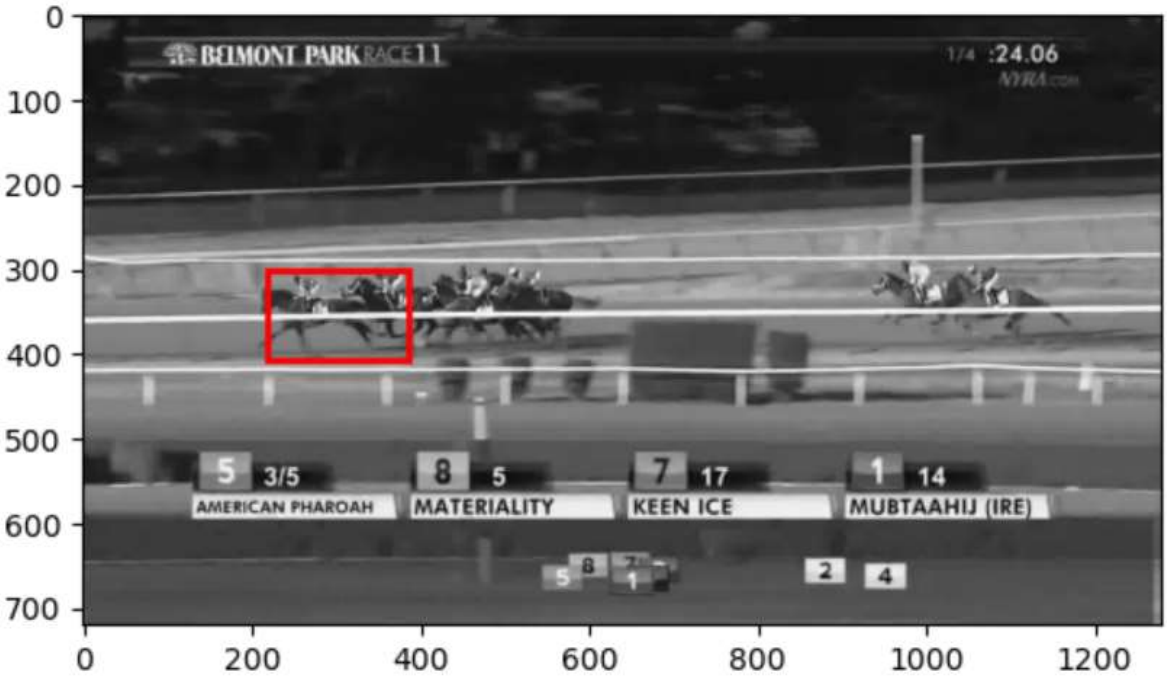

BELMONT PARK RACE 11
1/4 :24.06
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
1 14
MUBTAAHIJ (IRE)
7 17
KEEN ICE
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

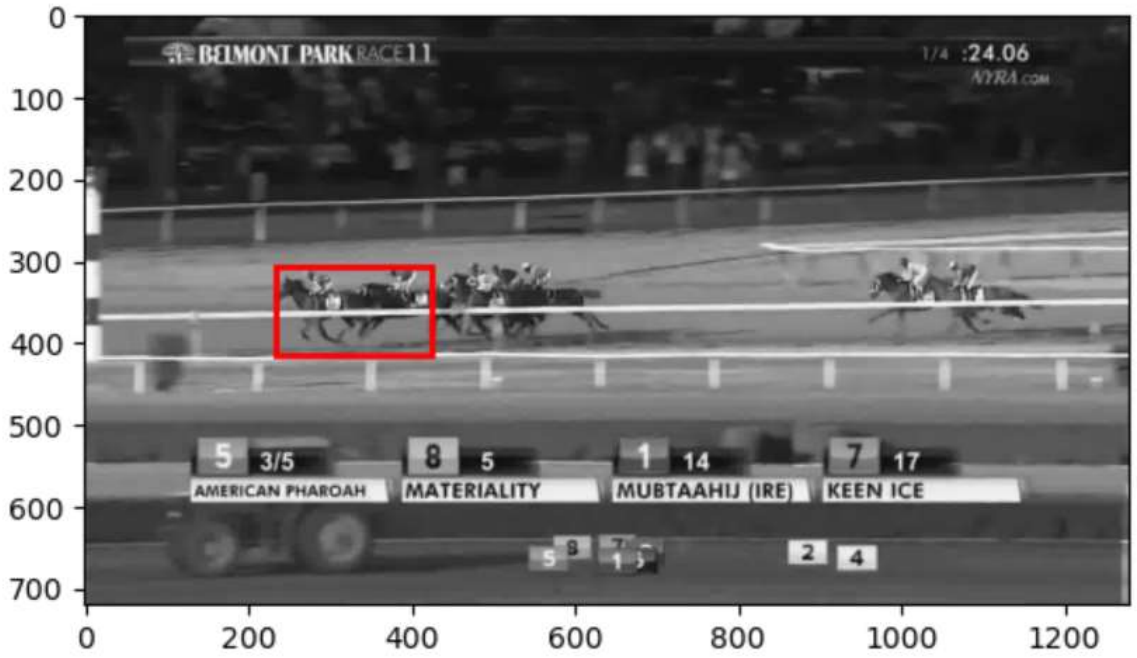

BELMONT PARK RACE 11
1/4 :24.06
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
7 17
KEEN ICE
1 14
MUBTAAHIJ (IRE)
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

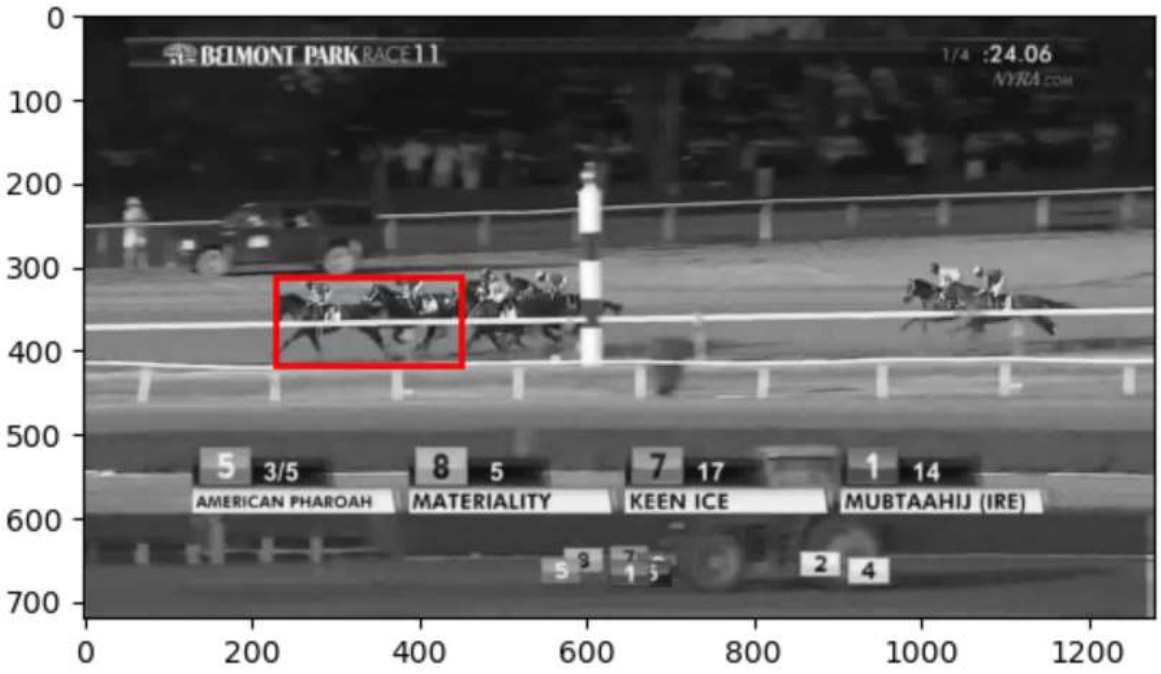

BELMONT PARK RACE 11
1/4 :24.06
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
7 17
KEEN ICE
1 14
MUBTAAHIJ (IRE)
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

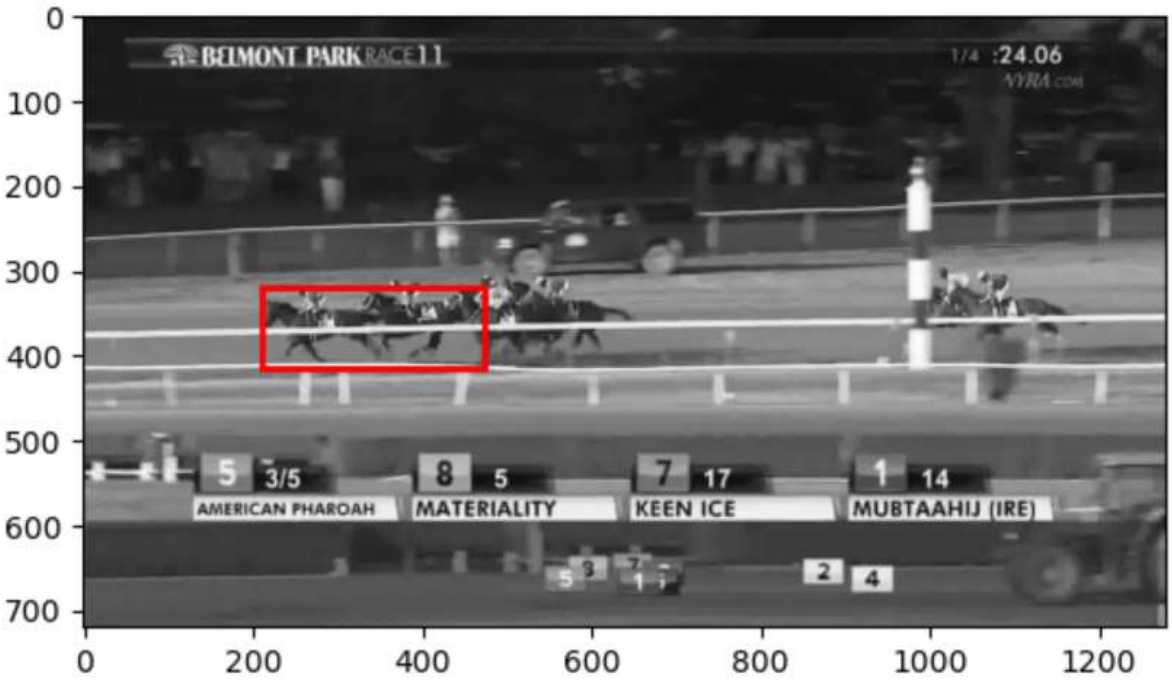

BELMONT PARK RACE 11
1/4 :24.06
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
7 17
KEEN ICE
1 14
MUBTAAHIJ (IRE)
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


BELMONT PARK RACE 11
1/4 :24.06
NYRA.com
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
7 17
KEEN ICE
1 14
MUBTAAHIJ (IRE)
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

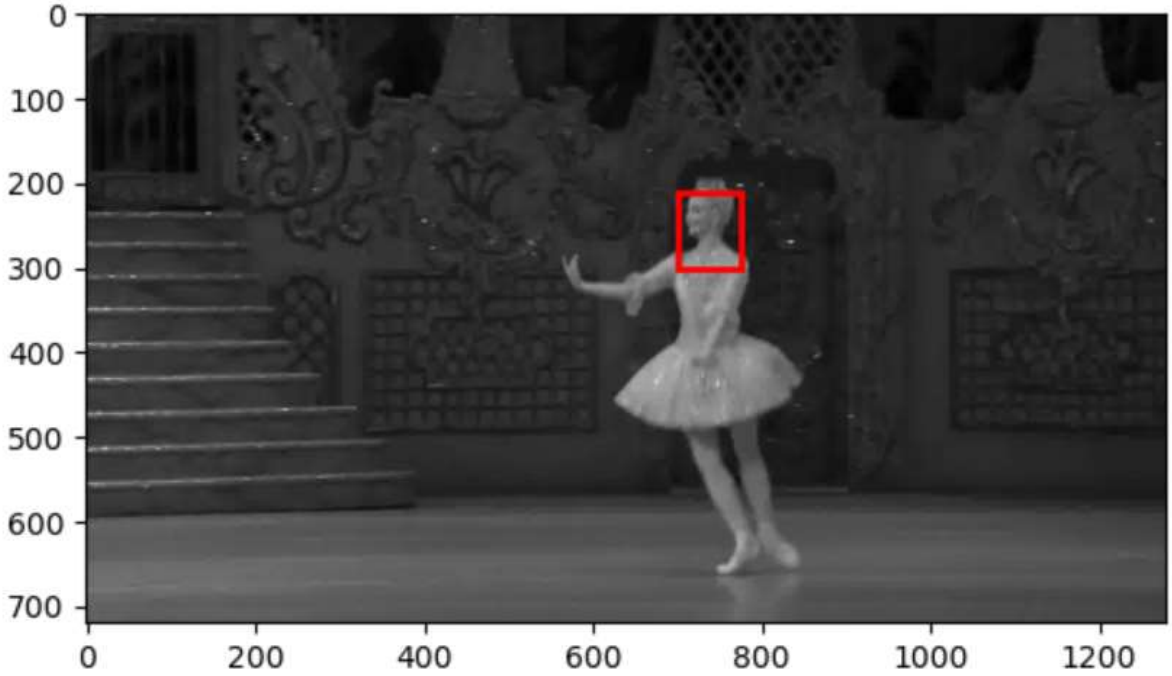

BELMONT PARK RACE 11
1/4 :24.06
NYRA.com
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
7 17
KEEN ICE
1 14
MUBTAAHIJ (IRE)
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

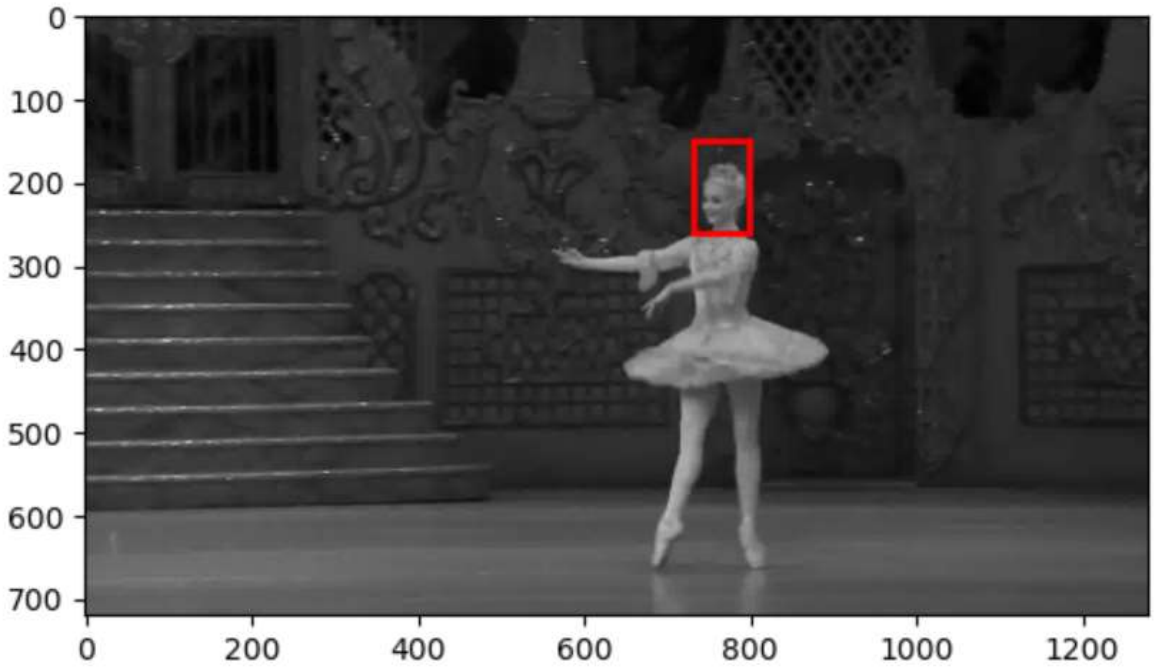

BELMONT PARK RACE 11
1/4 :24.06
NYRA.COM
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
1 14
MUBTAAHIJ (IRE)
7 17
KEEN ICE
2
4
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

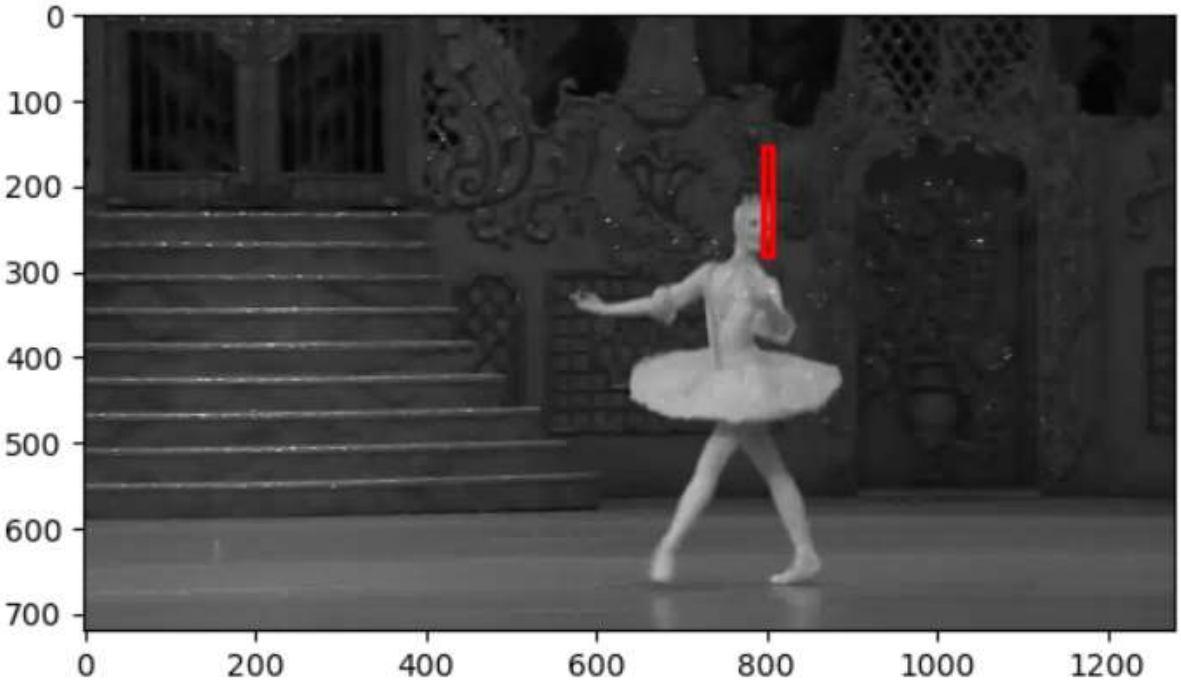

BELMONT PARK RACE 11
1/4 :24.06
NYRA.COM
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
7 17
KEEN ICE
1 14
MUBTAAHIJ (IRE)
2
4
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

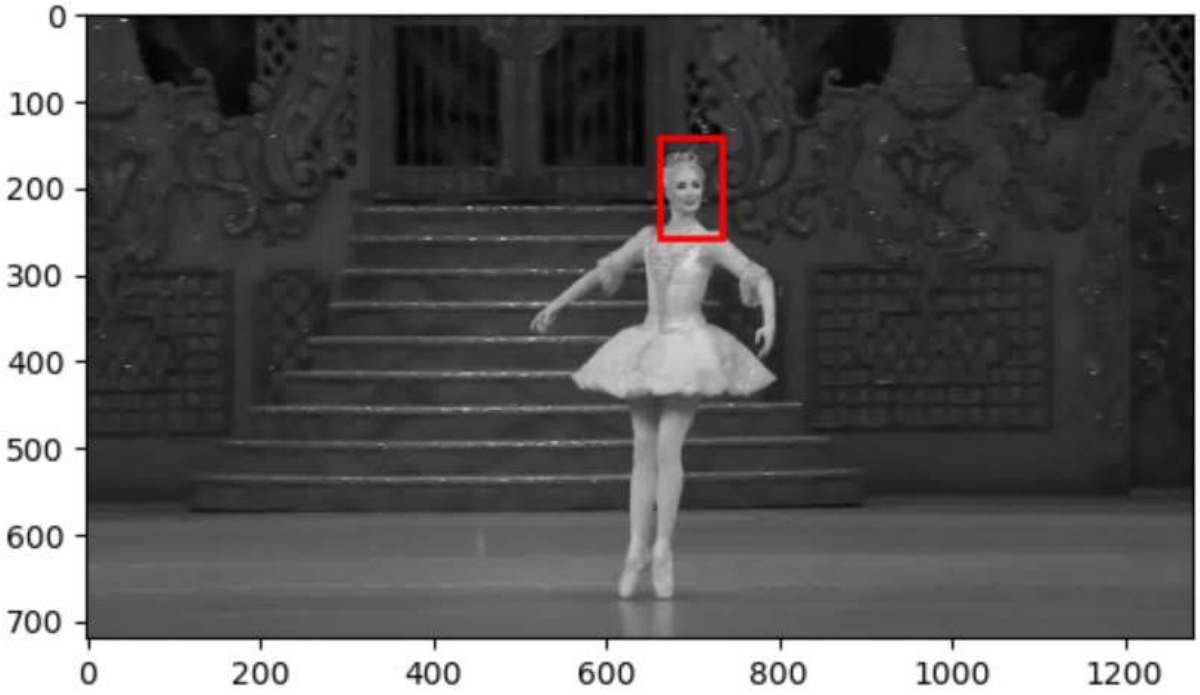

BELMONT PARK RACE 11
1/4 :24.06
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
7 17
KEEN ICE
1 14
MUBTAAHIJ (IRE)
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

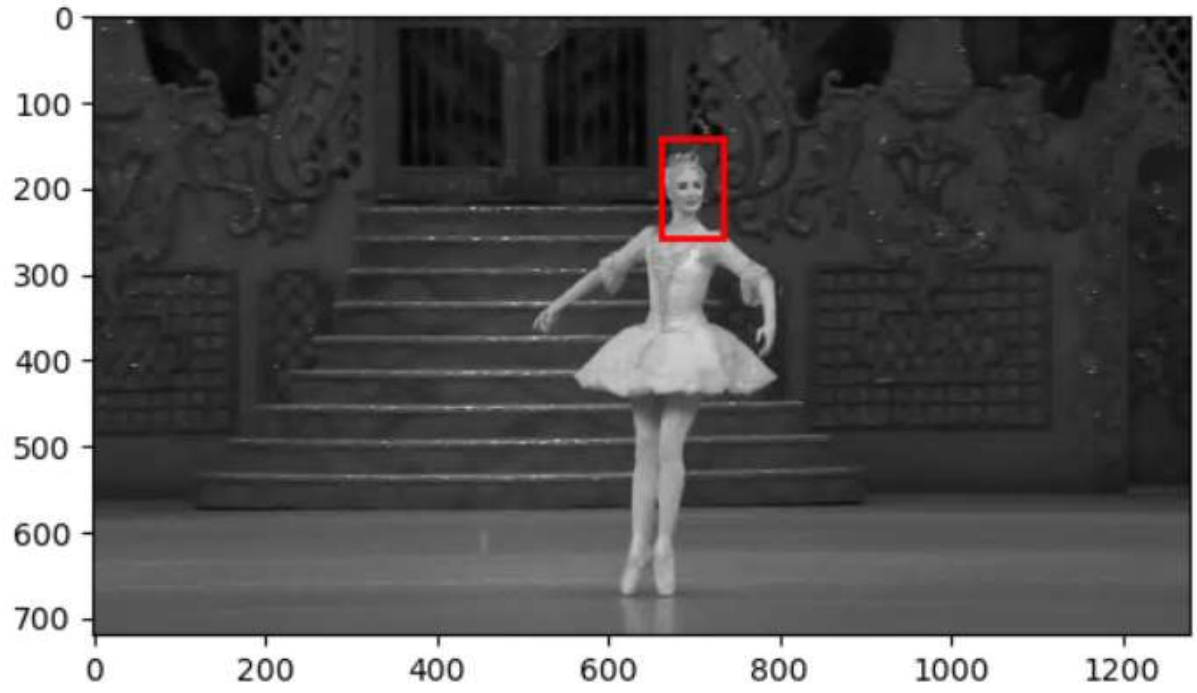
5. Ballet

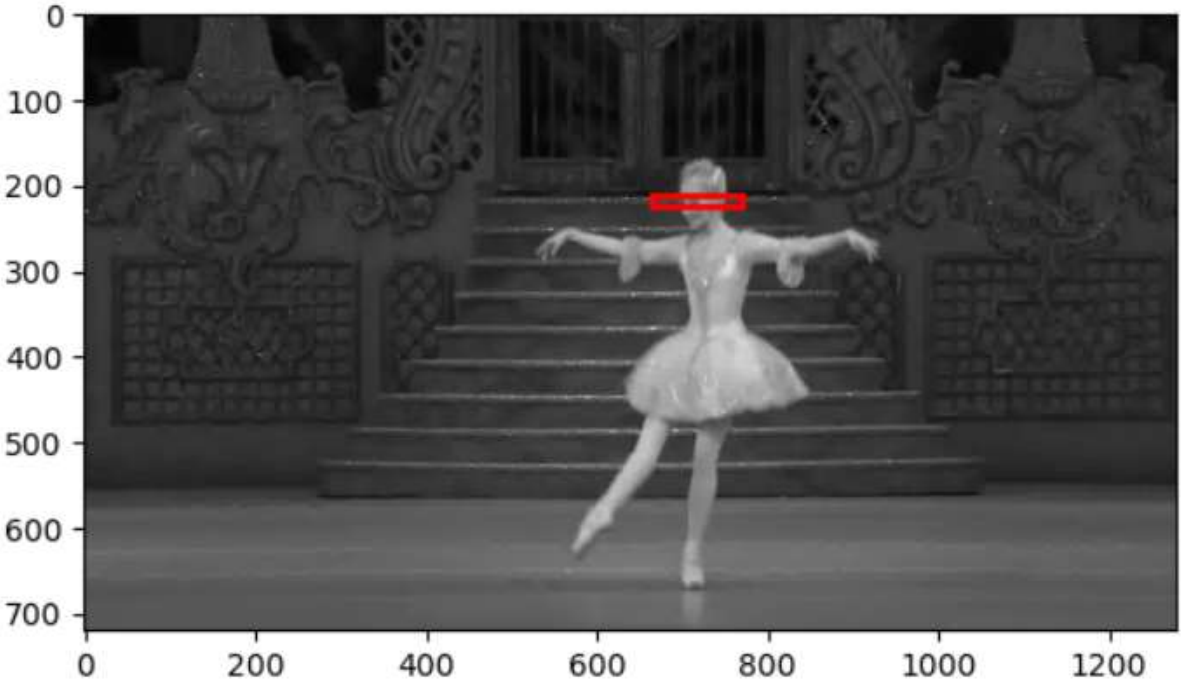

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

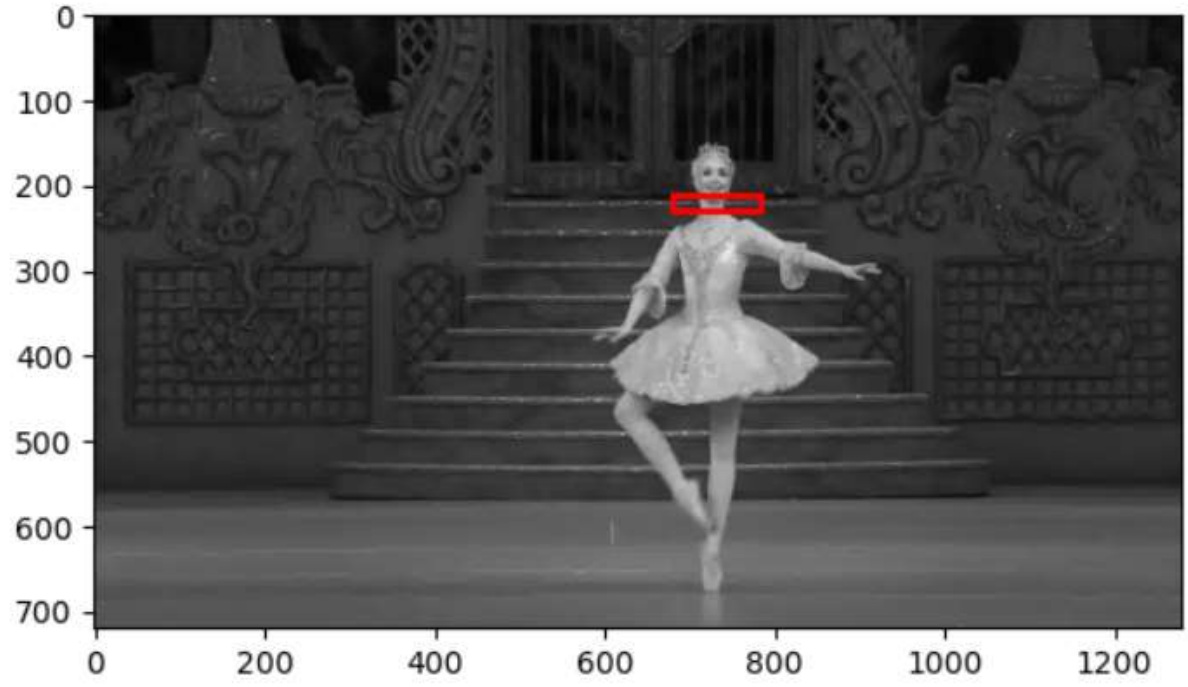

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

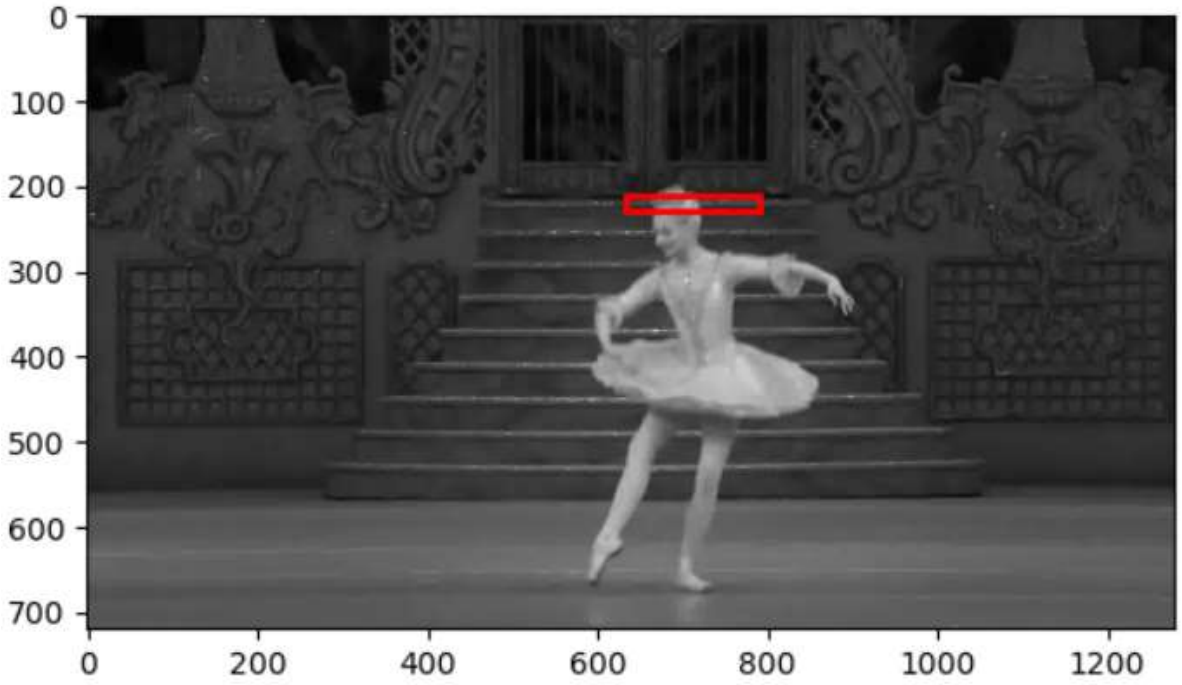

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

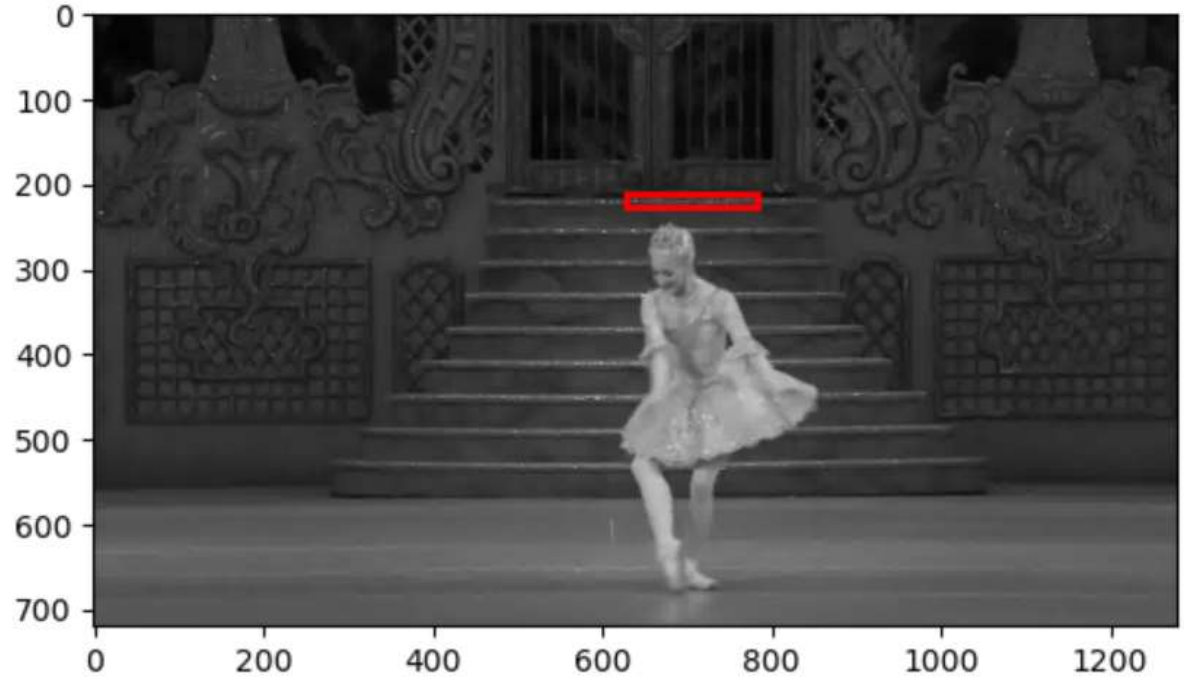

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

Q2.1

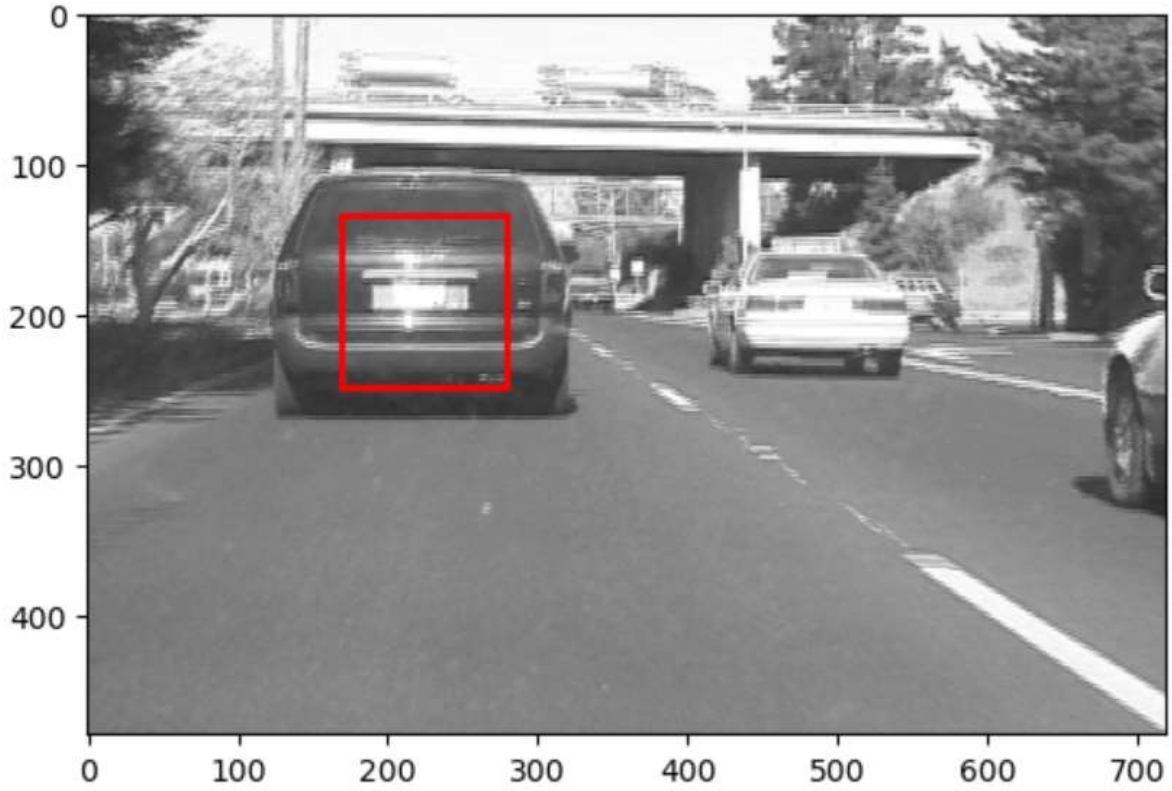
In [ ]:

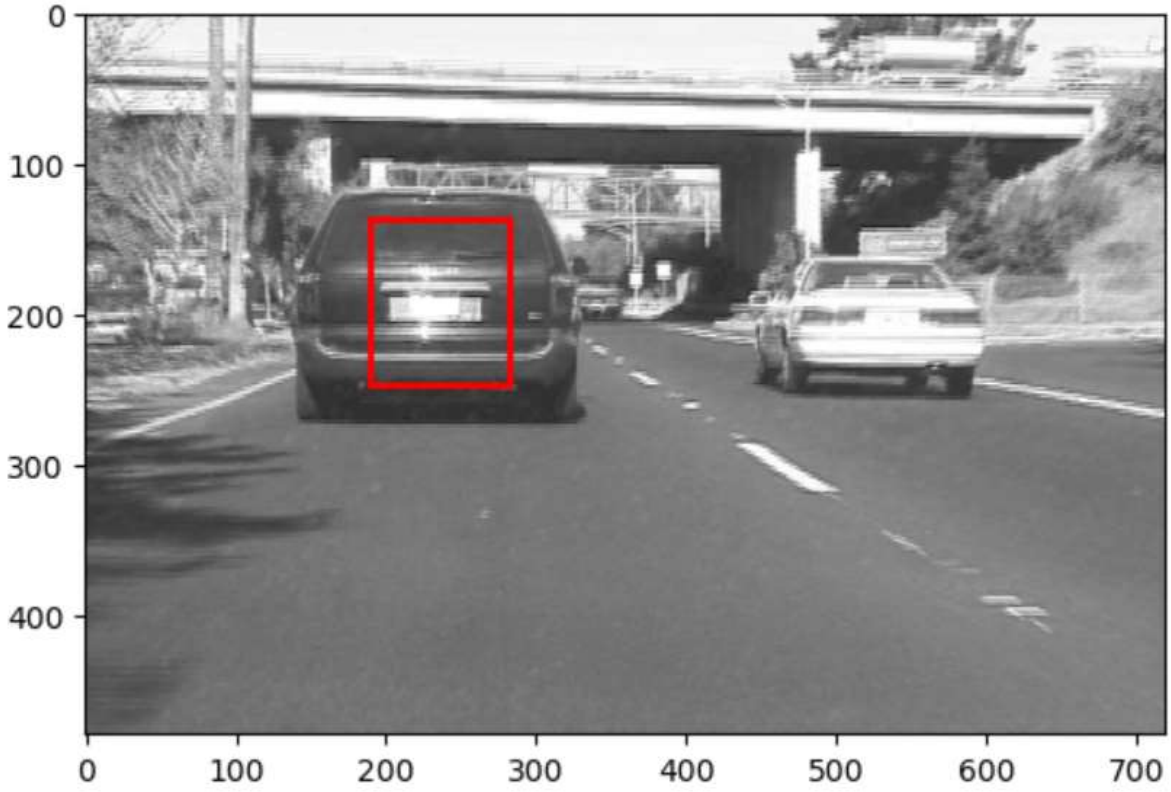
```
def InverseCompositionAffine(It, It1, rect, thresh=.025, maxIters=100):

    M = np.eye(3)
    P = np.zeros((3,3))
#     threshold = thresh

    x1, y1, x2, y2 = rect

    if x2 < x1 or y2 < y1:
        return M[: 2]

    s1 = np.arange(It.shape[0])
    s2 = np.arange(It.shape[1])
    s3 = np.arange(It1.shape[0])
    s4 = np.arange(It1.shape[1])

    interit = RectBivariateSpline(s1, s2, It)
    interit1 = RectBivariateSpline(s3, s4, It1)

    random_x = np.arange(x1, x2 + 0.1)
    random_y = np.arange(y1, y2 + 0.1)
    x, y = np.meshgrid(random_x, random_y)
    x = x.flatten()
    y = y.flatten()
    T_ev = interit.ev(y, x)

    coords_ = np.hstack((x.reshape(-1, 1), y.reshape(-1, 1)))

    T_x = interit.ev(y, x, dx=0, dy=1)
    T_y = interit.ev(y, x, dx=1, dy=0)

    A = np.zeros((x.shape[0], 2, 6))

    A[:,0,0] = A[:,1,3] =  x
    A[:,0,1] = A[:,1,4] =  y
    A[:,0,2] = A[:,1,5] =  1

    grad = np.hstack((T_x.reshape(-1, 1), T_y.reshape(-1, 1))).reshape(-1, 1, 2)

    A = np.matmul(grad, A).reshape(-1, 6)

    for i in range(maxIters):

        coords = M @ (np.hstack((coords_, np.ones(coords_.shape[0]).reshape(-1,

        x = coords[0].flatten()
        y = coords[1].flatten()
```

```
        I_ev = interit1.ev(y, x)

        b = I_ev - T_ev

        dp = np.linalg.lstsq(A, b, rcond=None)[0]

        dp = dp.reshape(2, 3)

        dM = np.eye(3)
        dM = dM + np.vstack((dp, np.array([0, 0, 0])))
        M = M @ np.linalg.pinv(dM)

        if np.linalg.norm(dp) <= thresh:
            break

    return M[: -1]
```

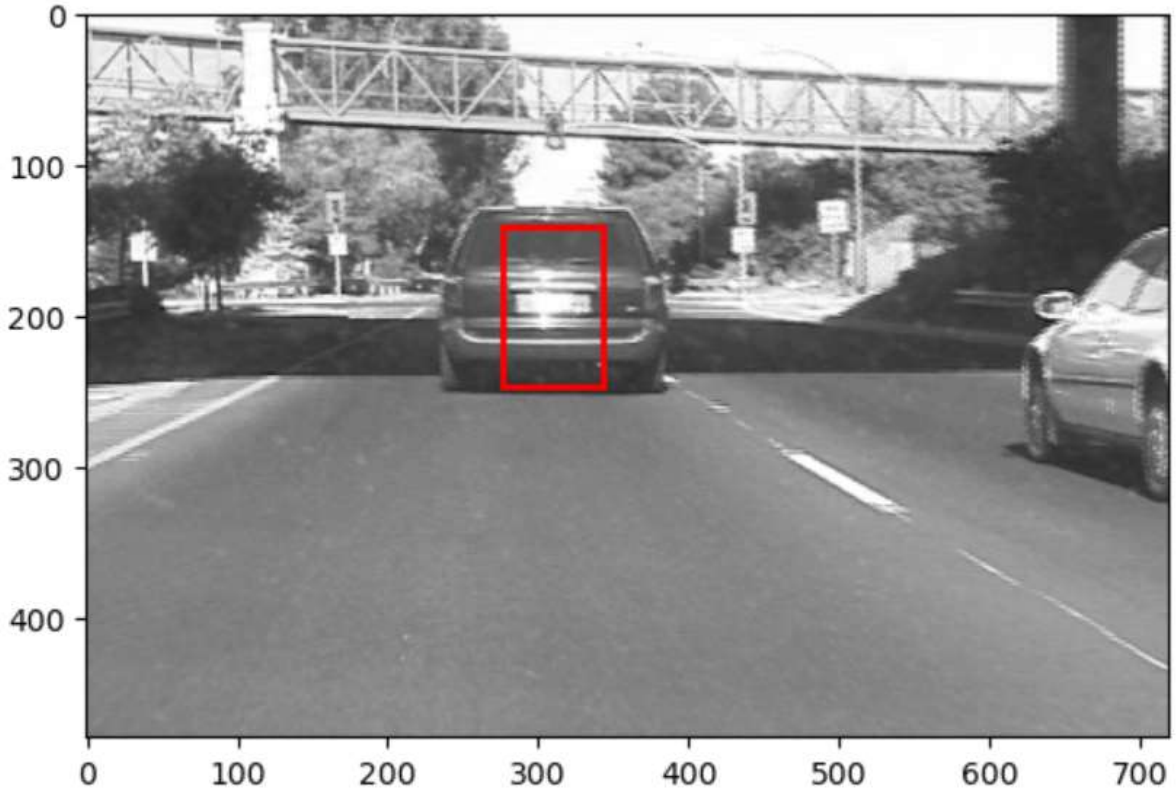
Results from each dataset -

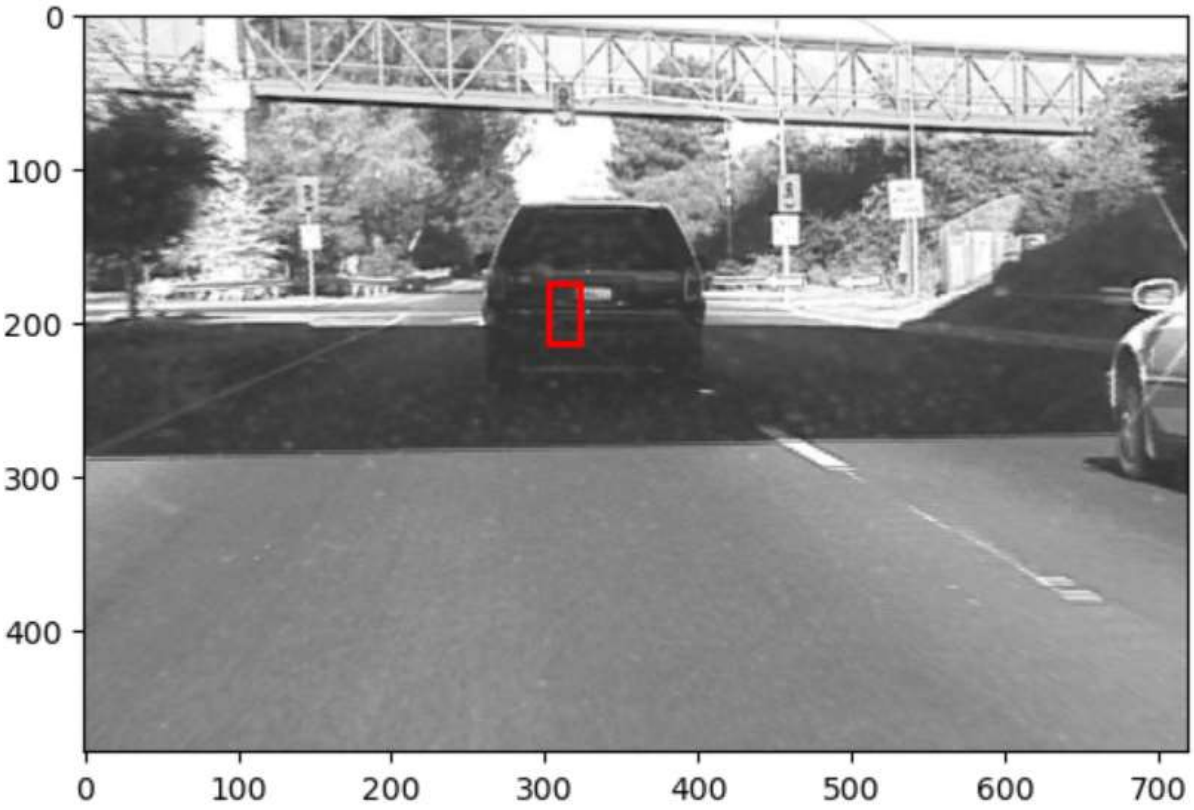
1. Car1


0
100
200
300
400
0
100
200
300
400
500
600
700


0
100
200
300
400
0
100
200
300
400
500
600
700


0
100
200
300
400
0
100
200
300
400
500
600
700


0
100
200
300
400
0
100
200
300
400
500
600
700

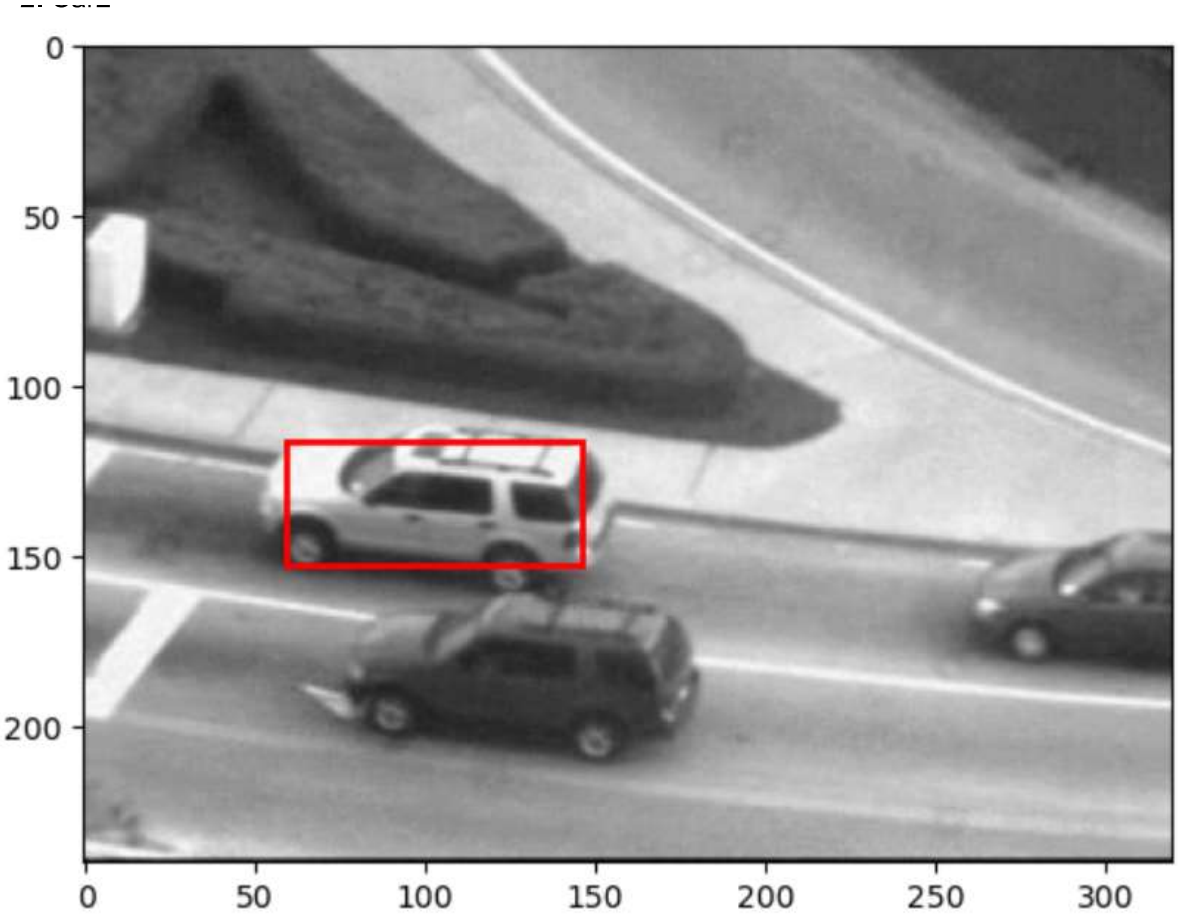

0
100
200
300
400
0
100
200
300
400
500
600
700

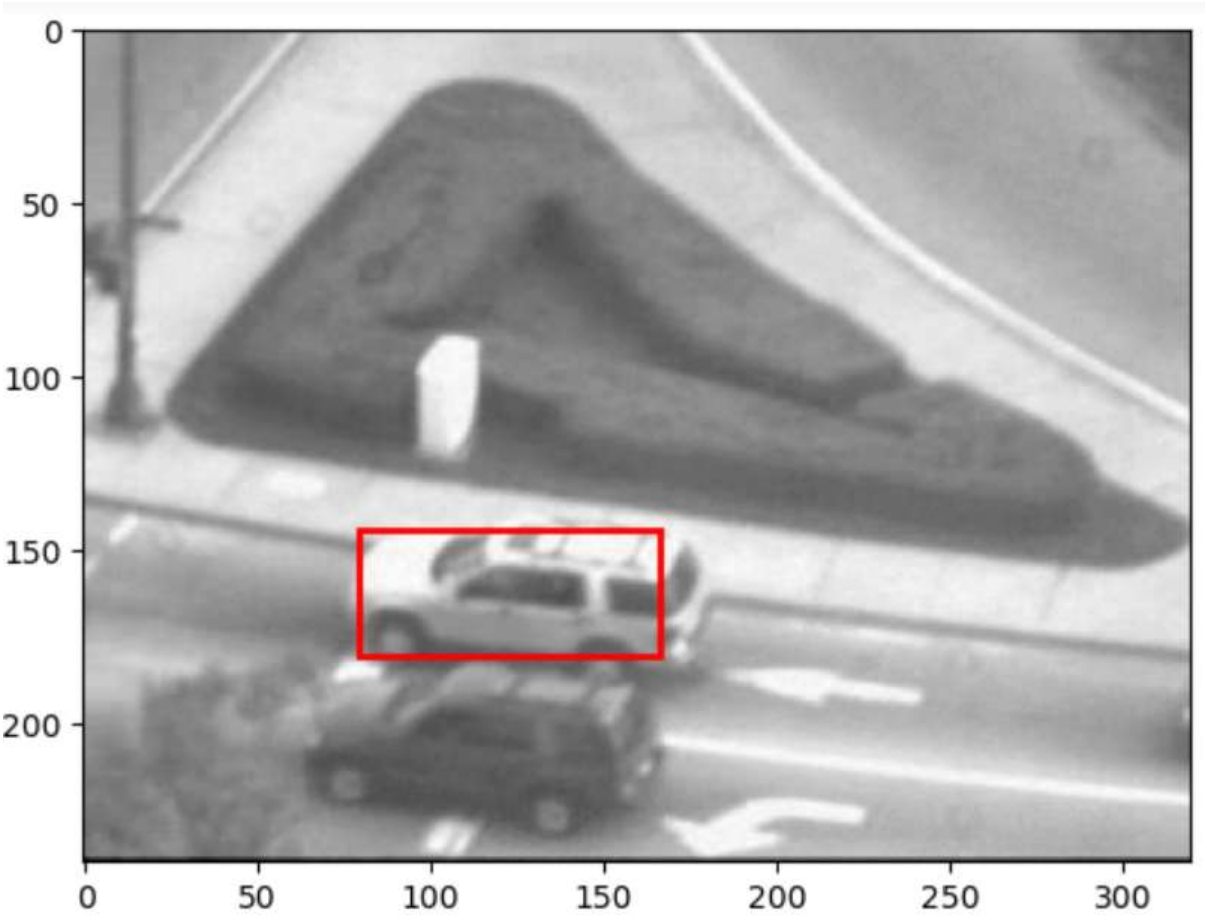

0
100
200
300
400
0
100
200
300
400
500
600
700

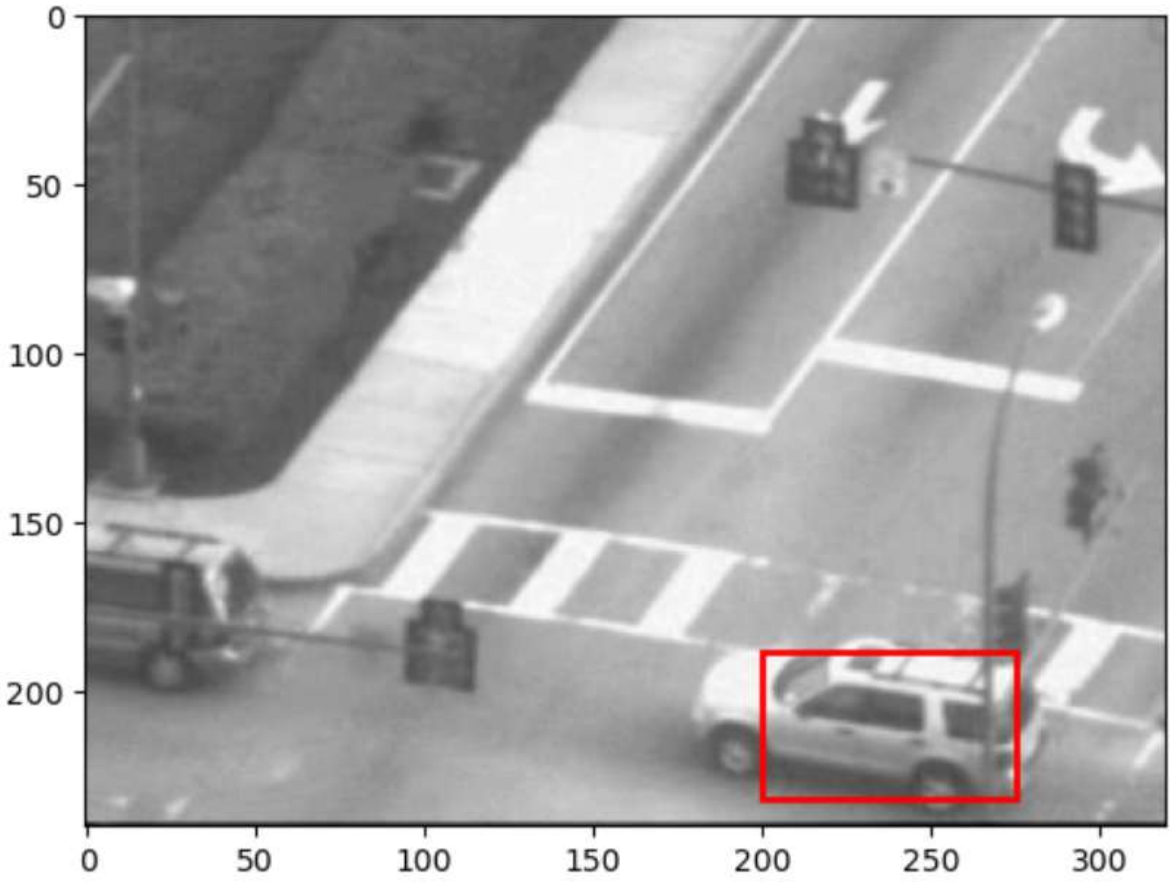

0
100
200
300
400
0
100
200
300
400
500
600
700


0
100
200
300
400
0
100
200
300
400
500
600
700

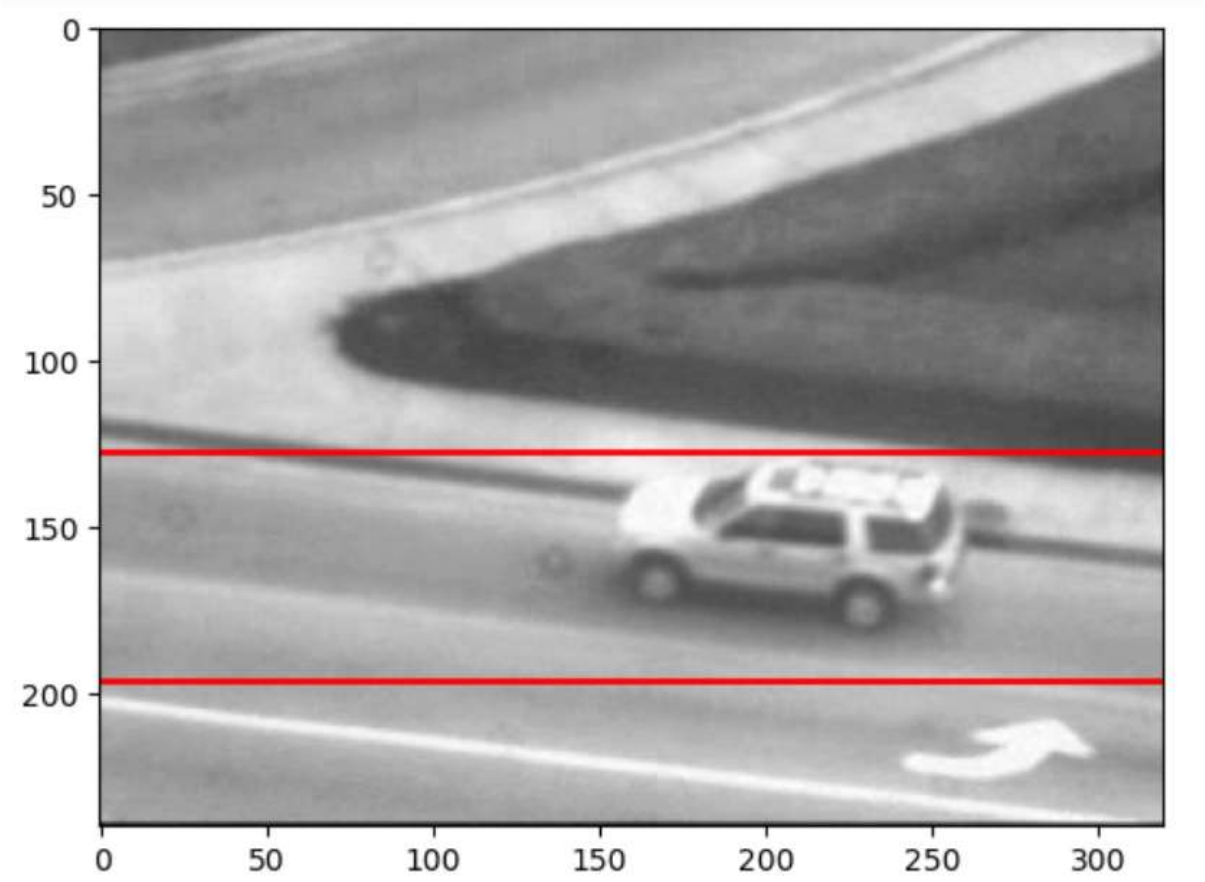
2. Car2

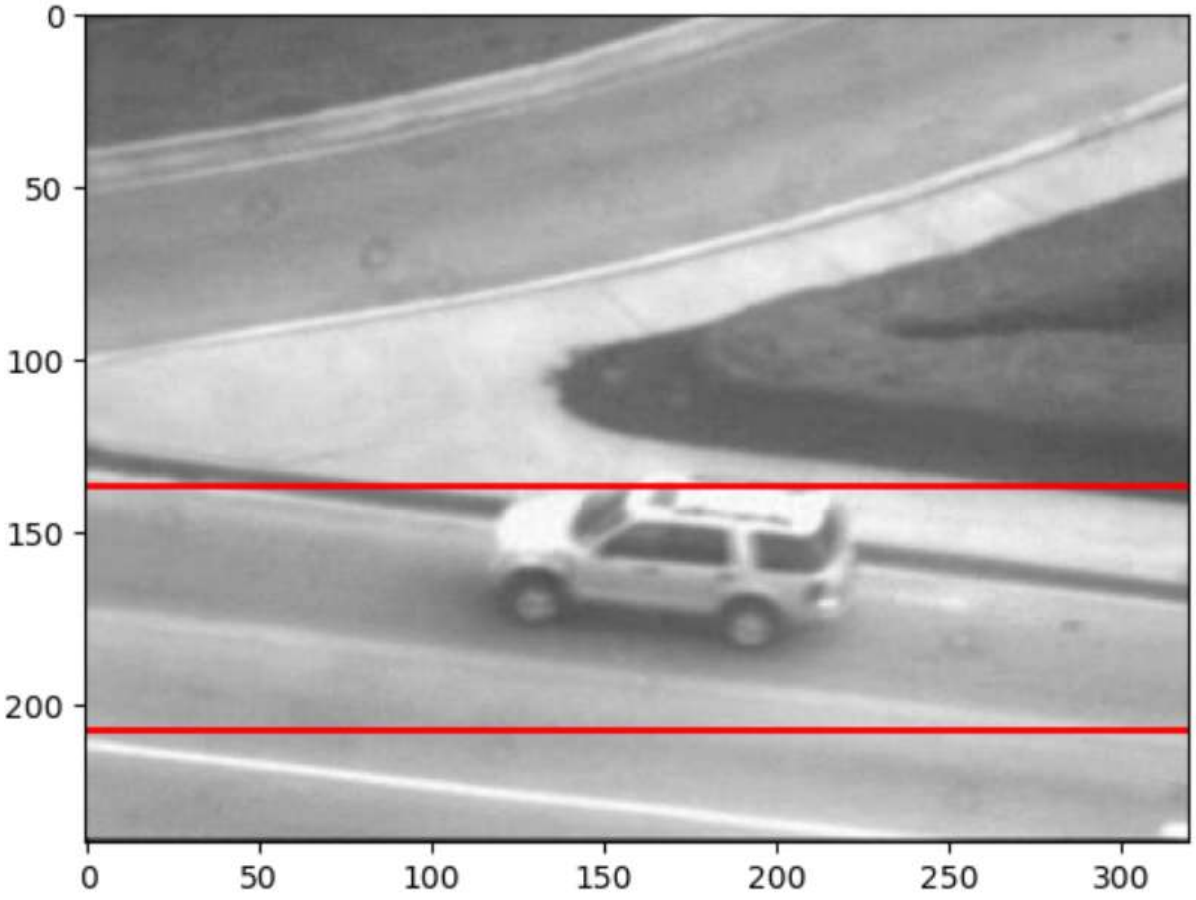

0
50
100
150
200
0
50
100
150
200
250
300


0
50
100
150
200
0
50
100
150
200
250
300


0
50
100
150
200
0
50
100
150
200
250
300


0
50
100
150
200
0
50
100
150
200
250
300

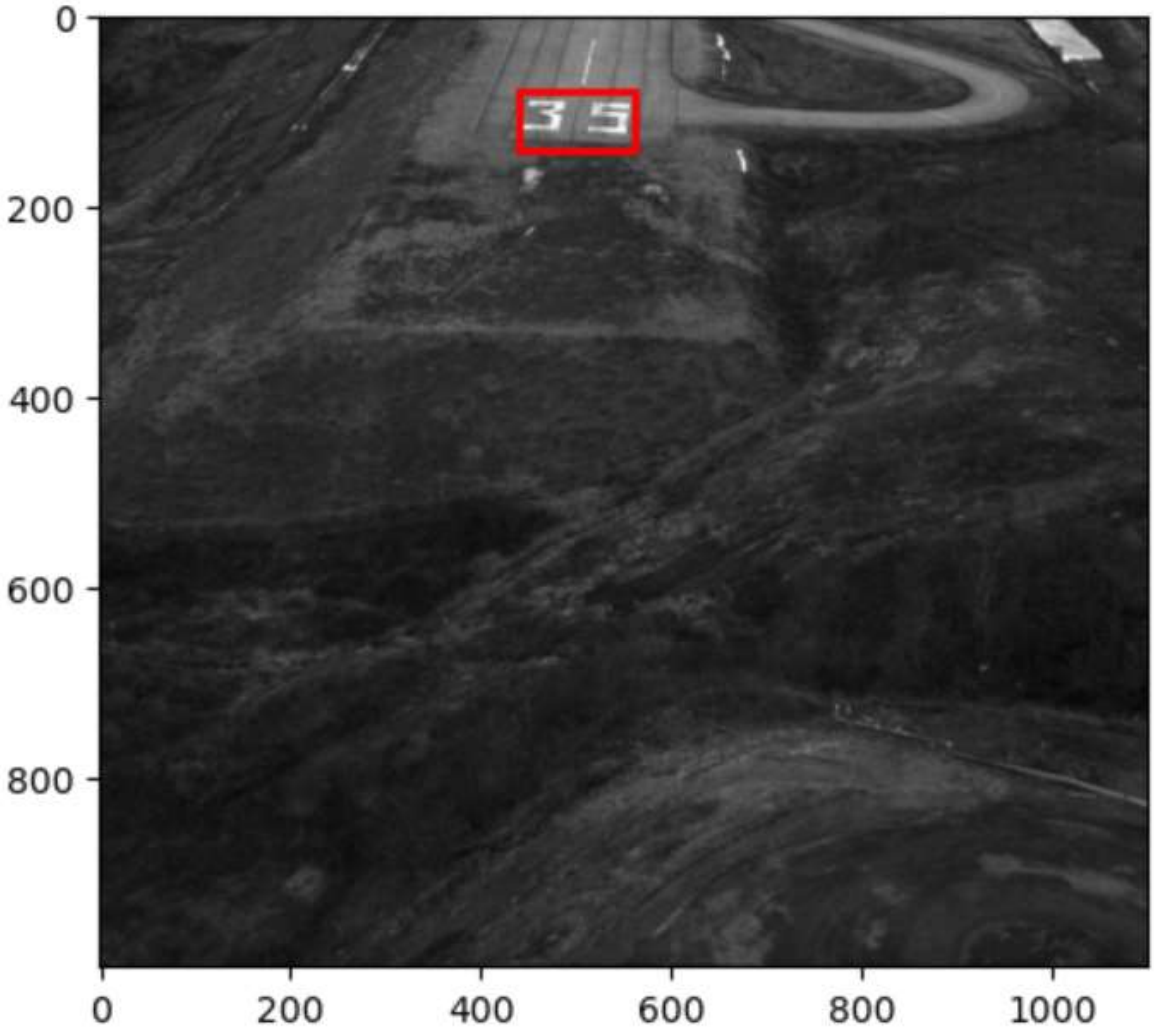

0
50
100
150
200
0
50
100
150
200
250
300

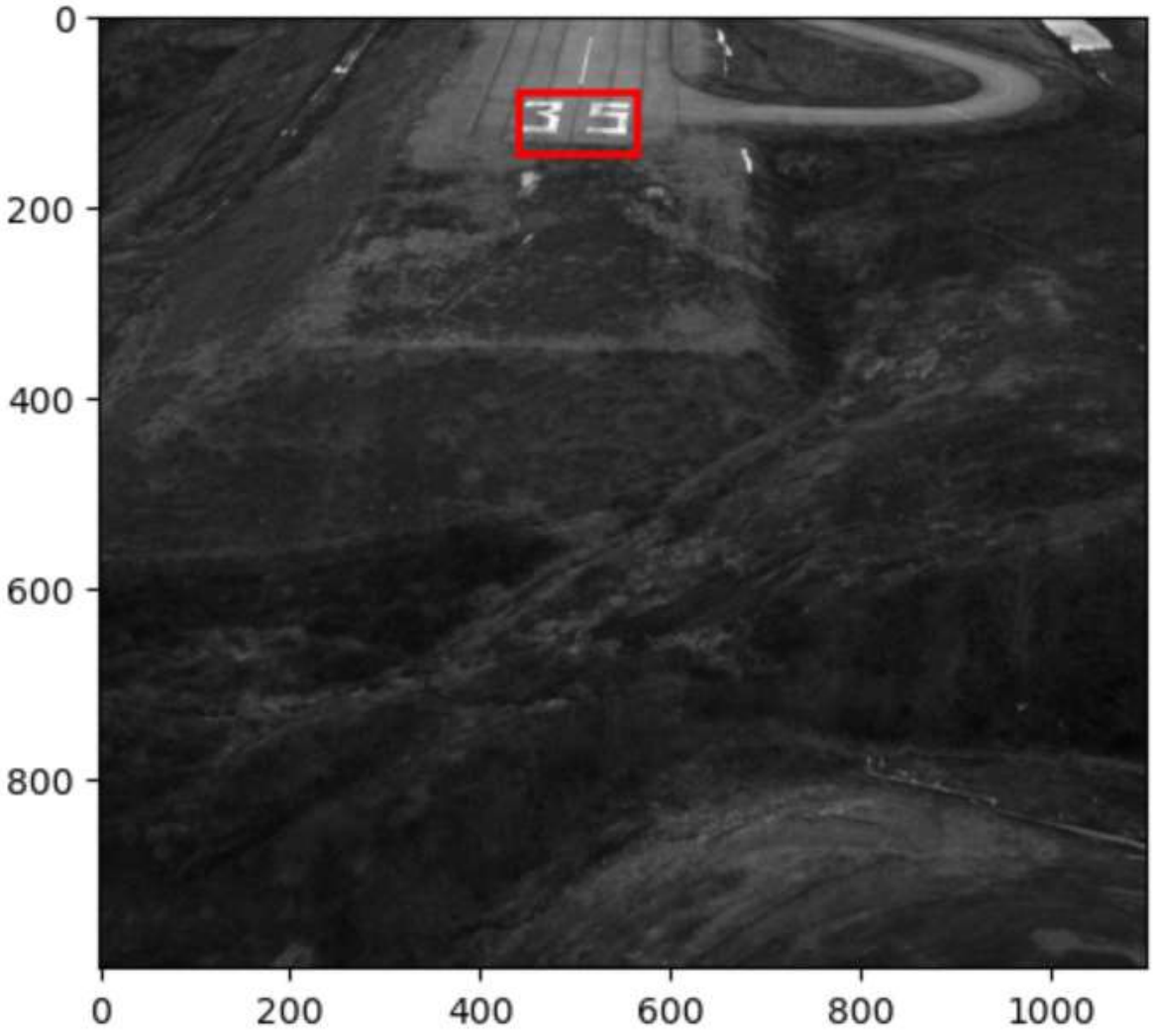

0
50
100
150
200
0
50
100
150
200
250
300

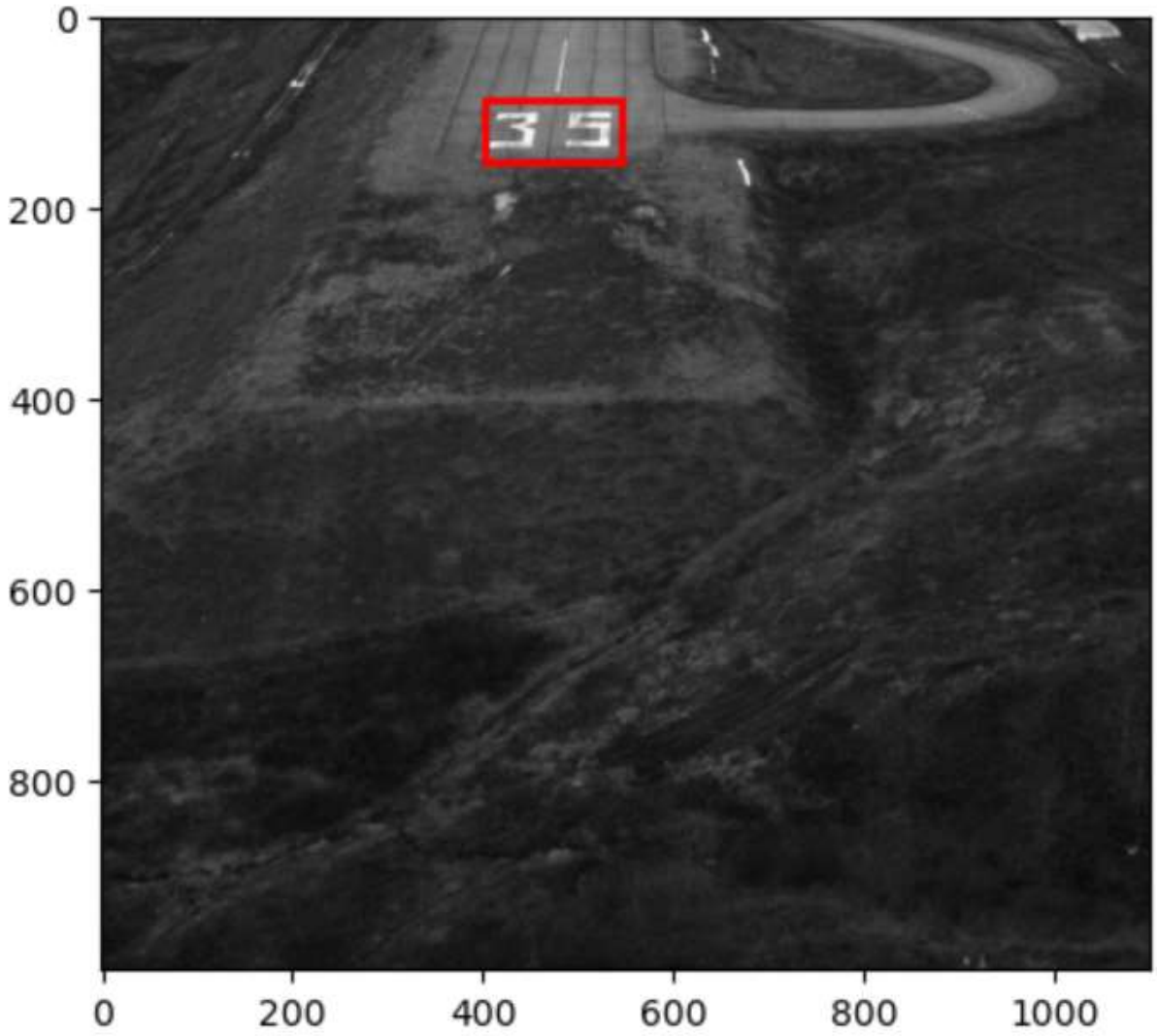

0
50
100
150
200
0
50
100
150
200
250
300

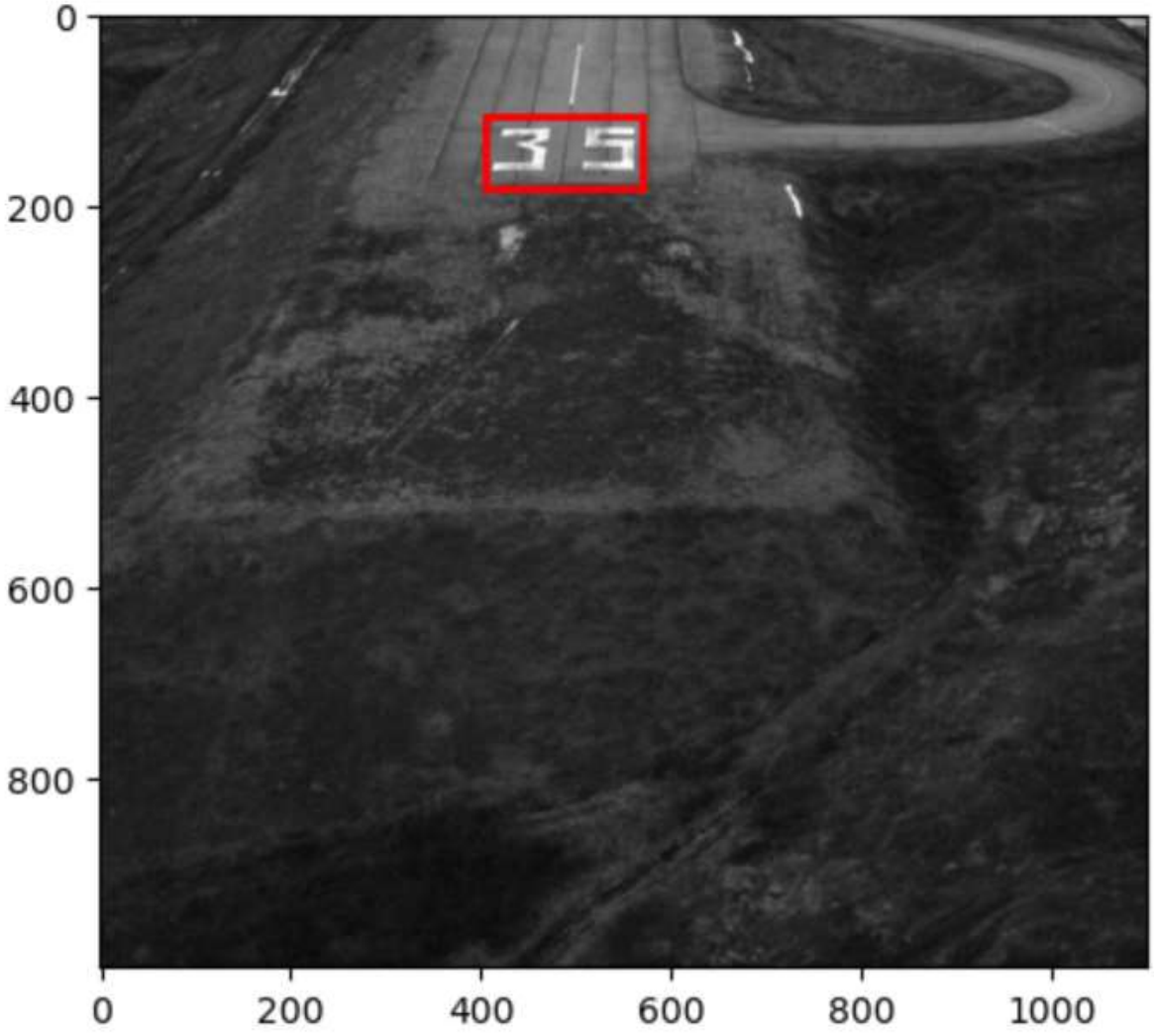

0
50
100
150
200
0
50
100
150
200
250
300

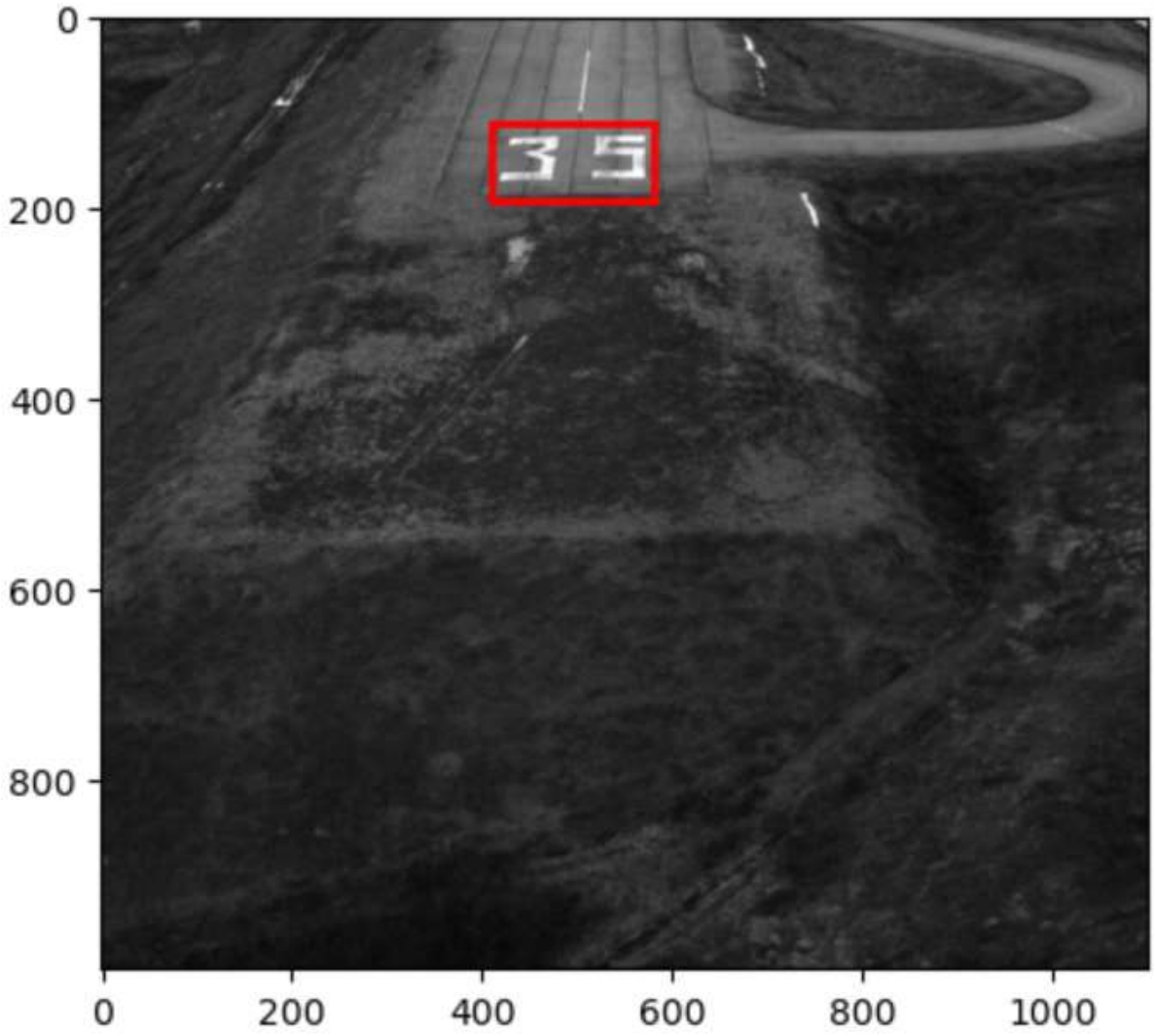
3. Landing

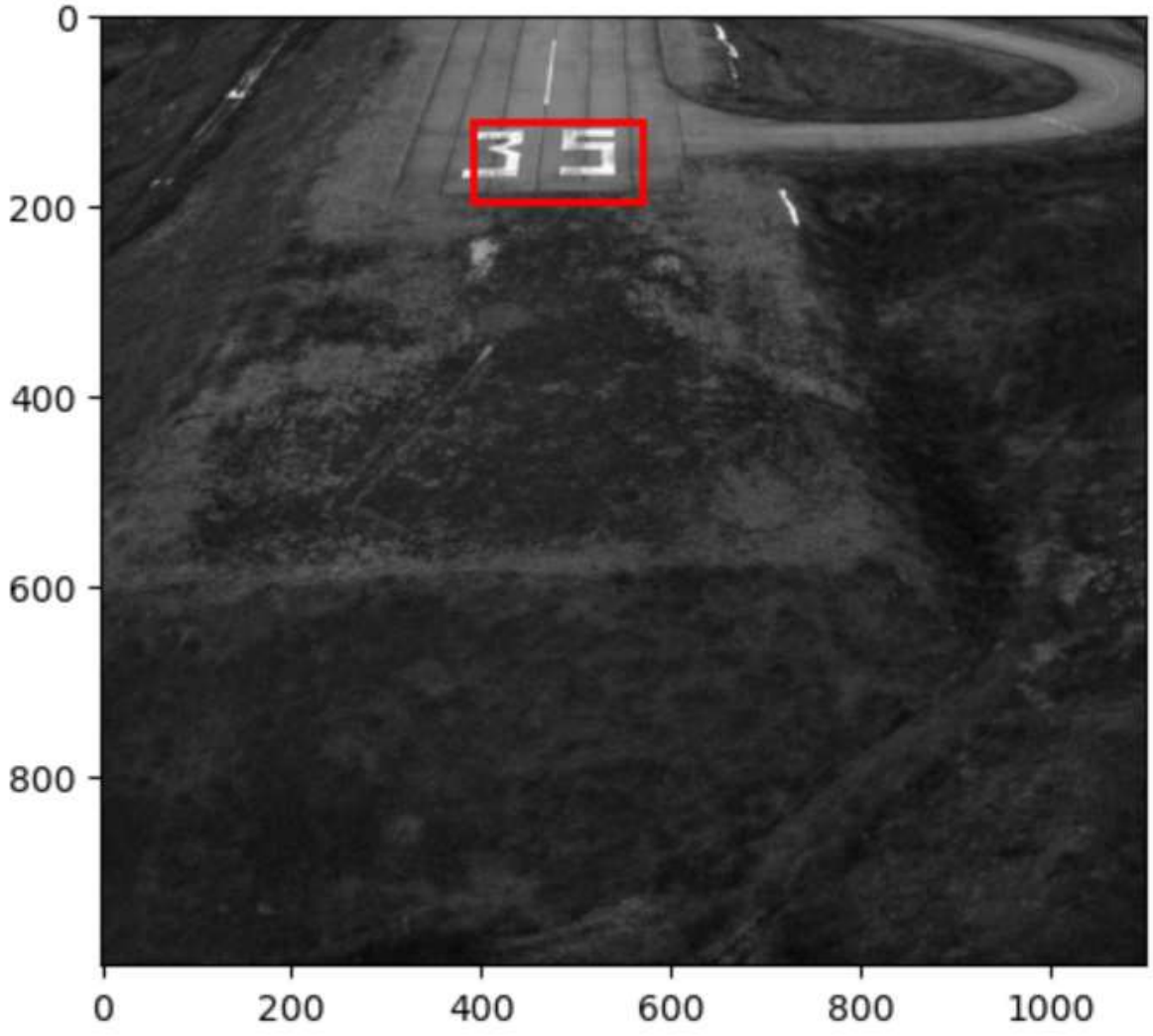

0
200
400
600
800
0
200
400
600
800
1000

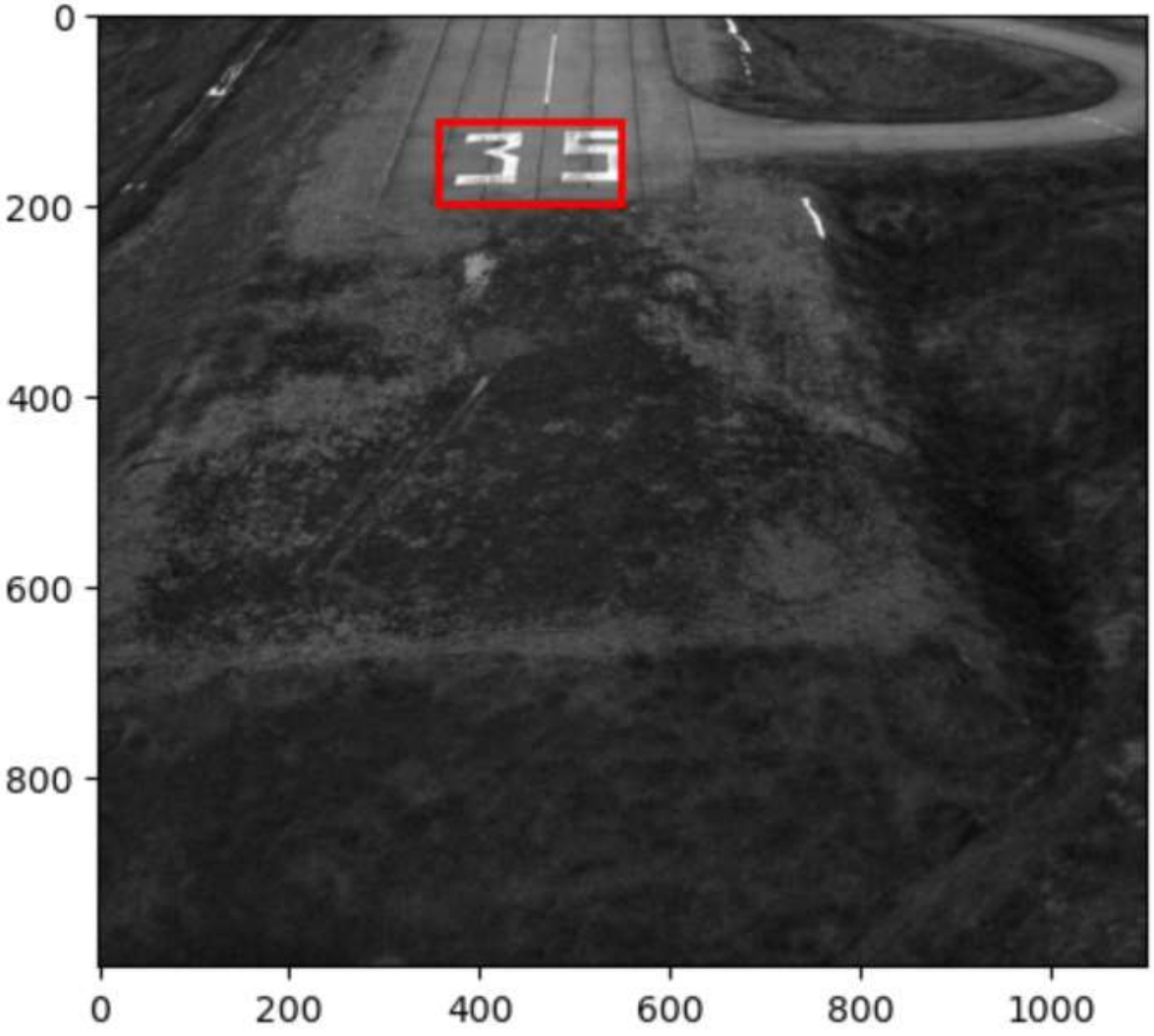

0
200
400
600
800
0
200
400
600
800
1000

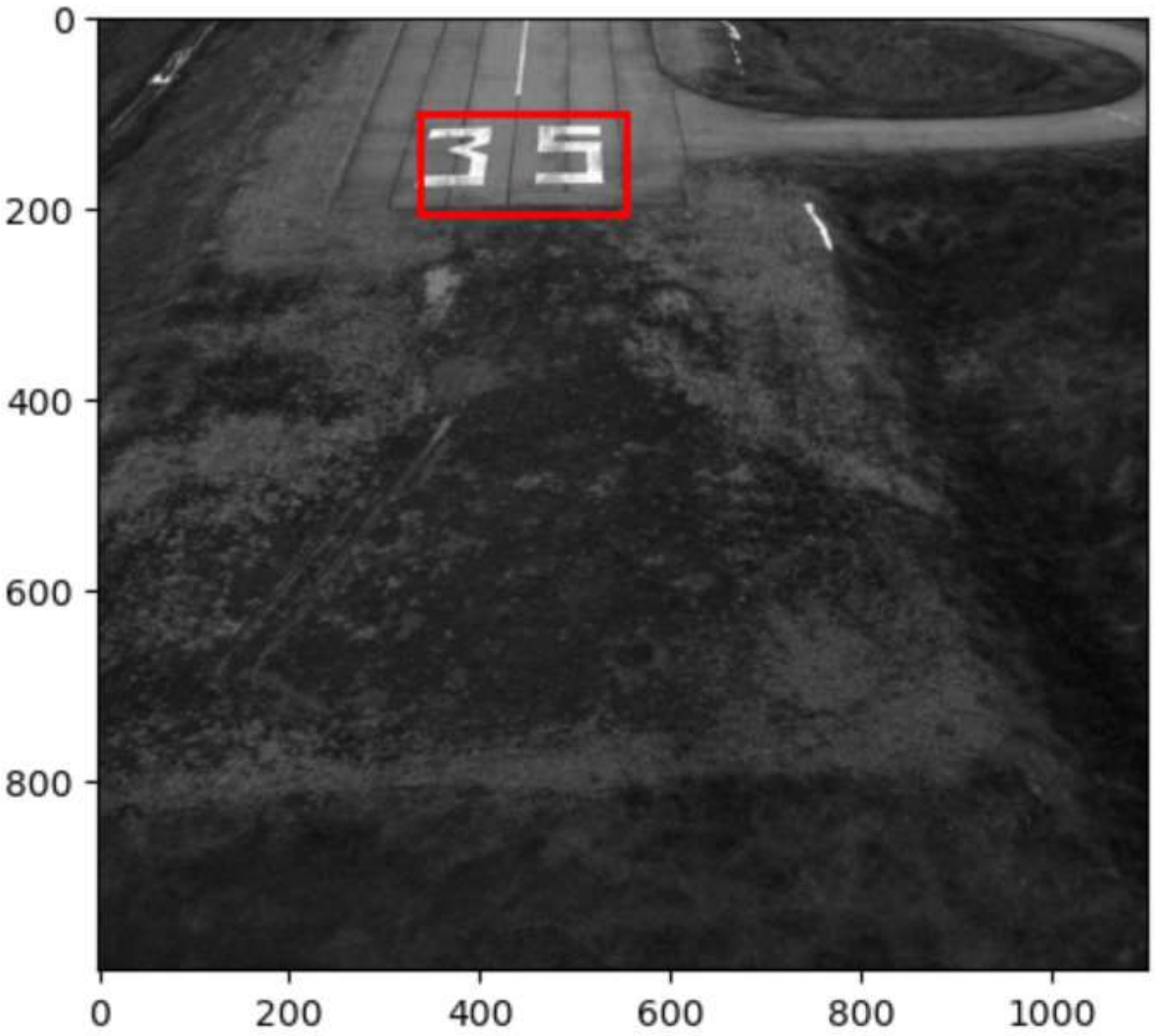

0
200
400
600
800
0
200
400
600
800
1000

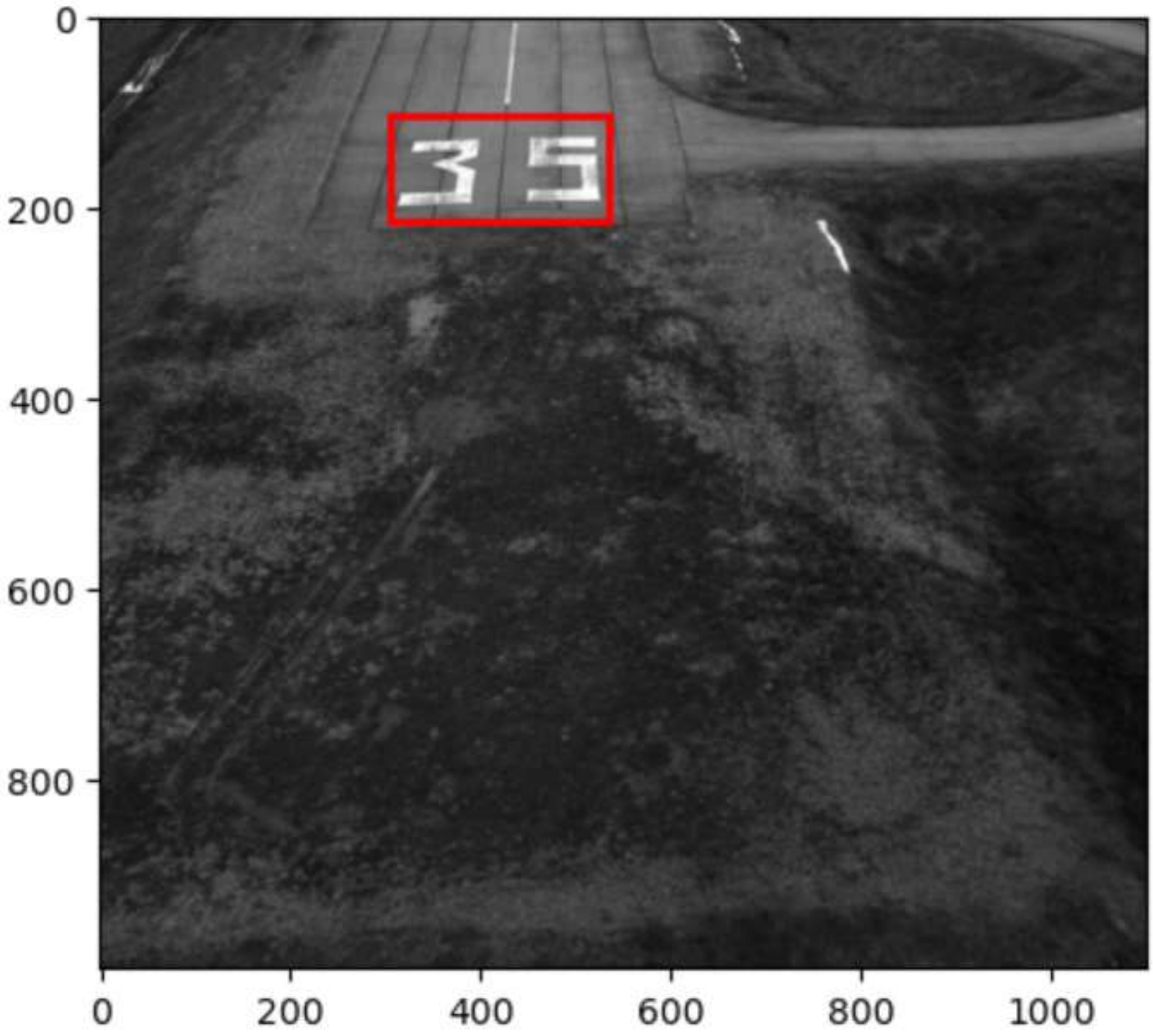

0
200
400
600
800
0
200
400
600
800
1000


0
200
400
600
800
0
200
400
600
800
1000

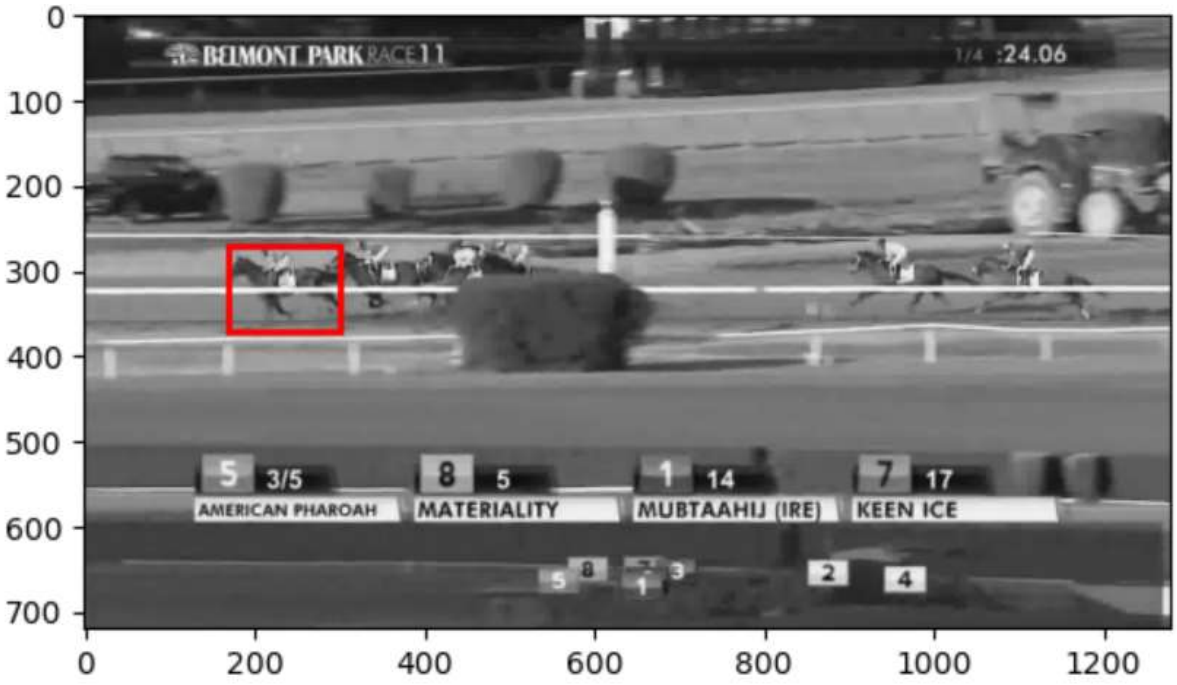

0
200
400
600
800
0
200
400
600
800
1000

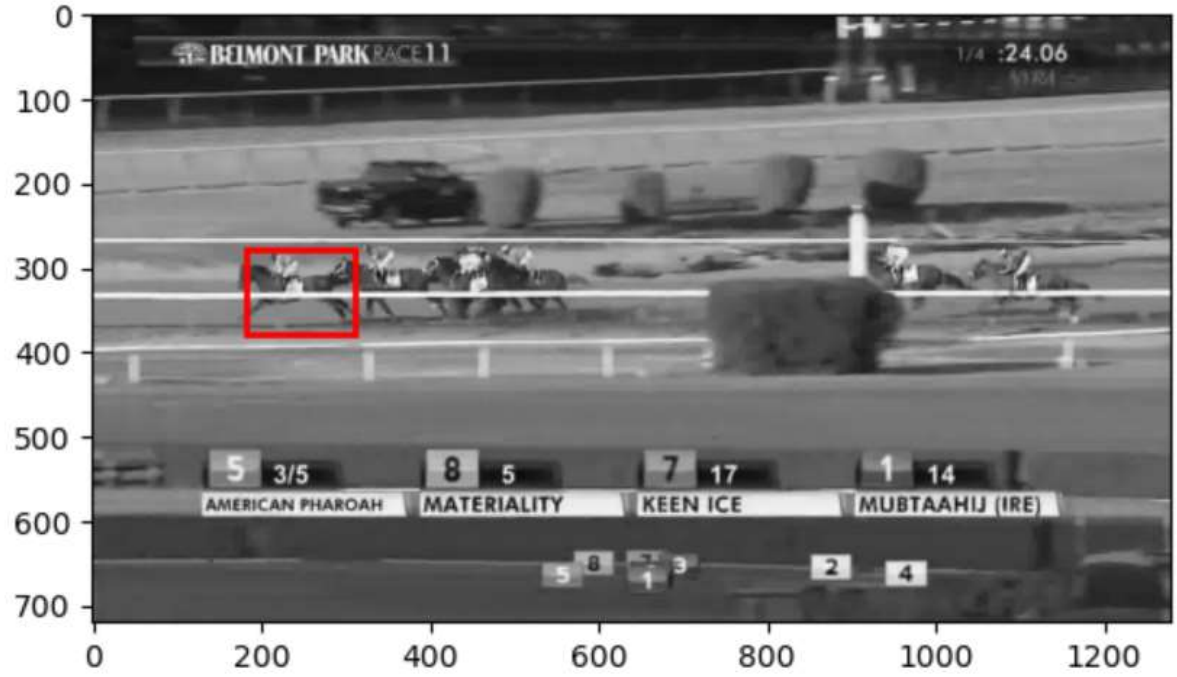

0
200
400
600
800
0
200
400
600
800
1000

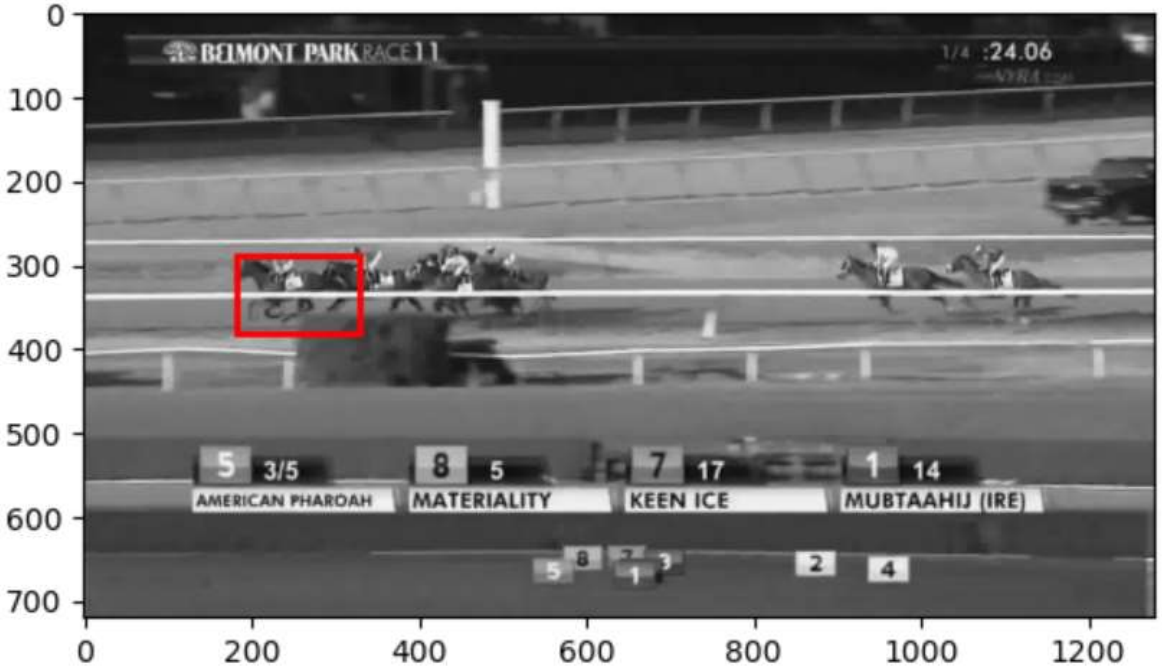

0
200
400
600
800
0
200
400
600
800
1000

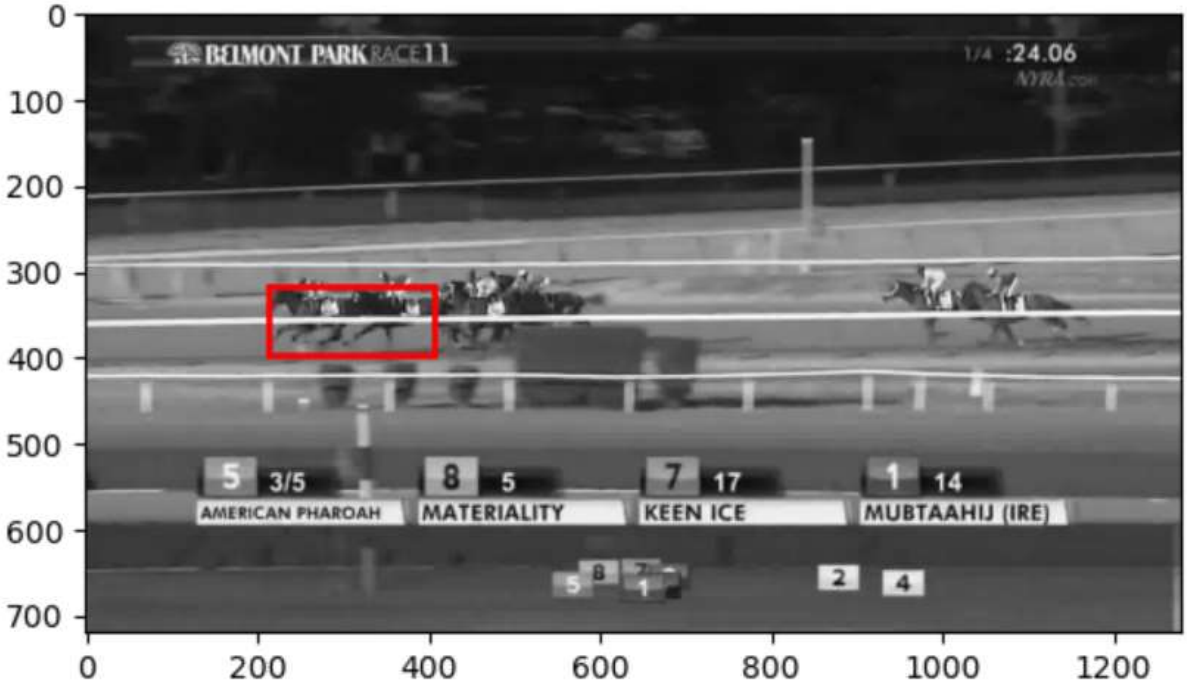


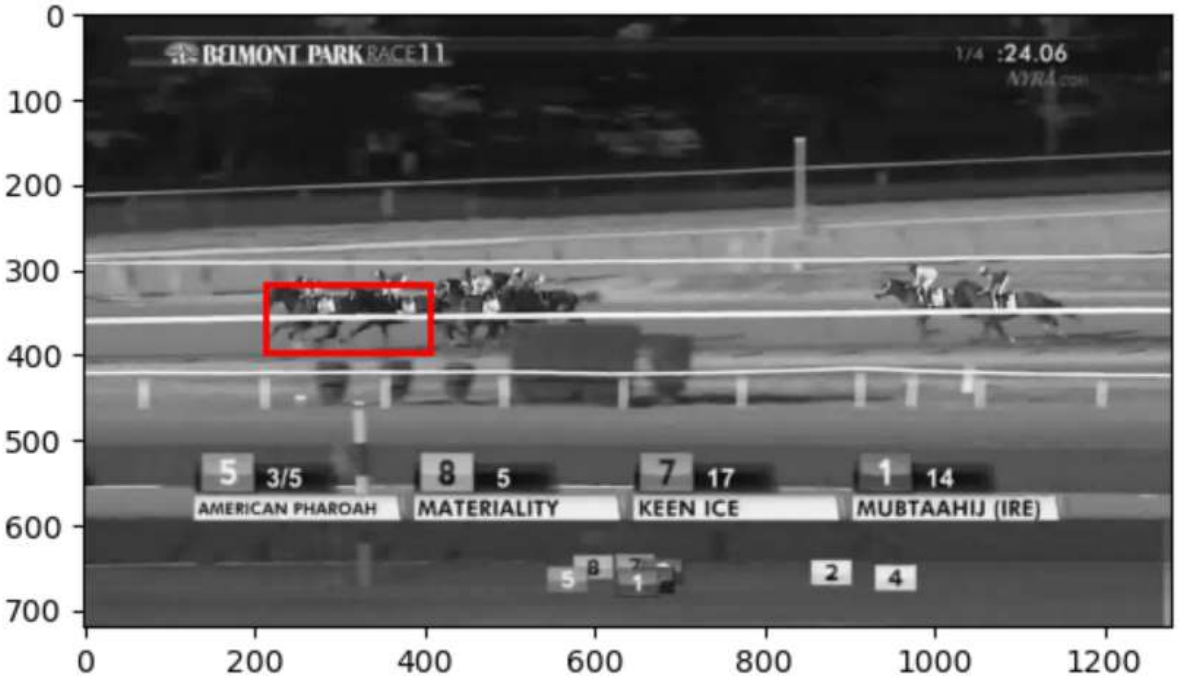
4. Race

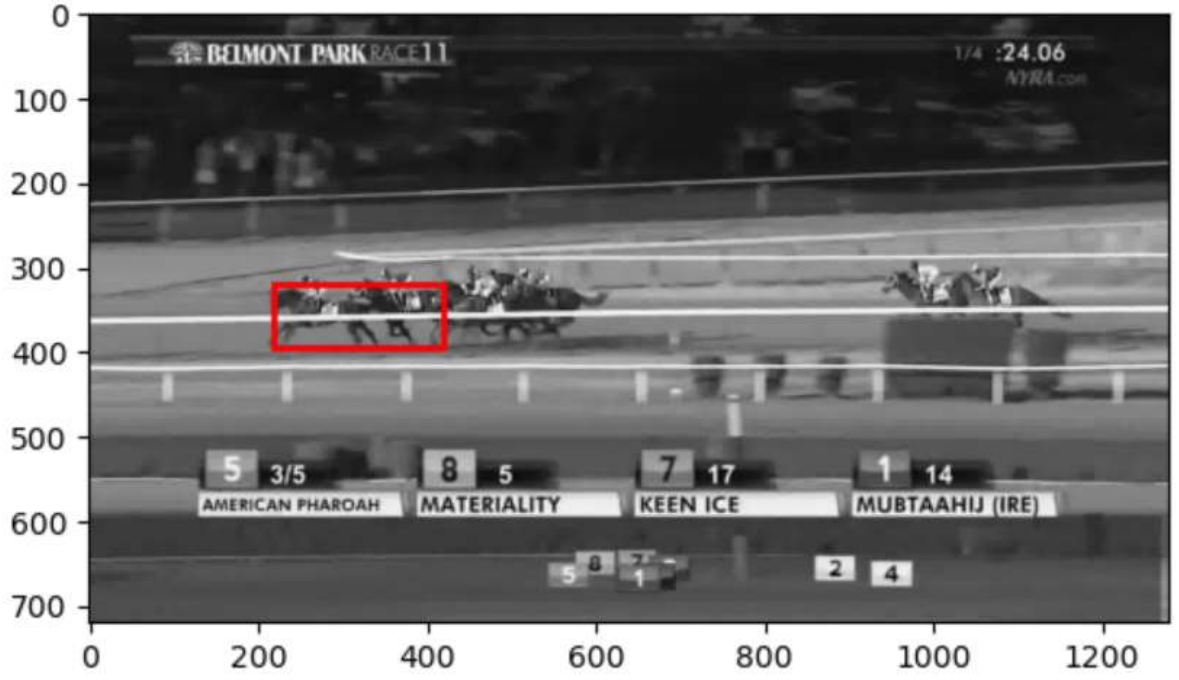

BELMONT PARK RACE 11
1/4 :24.06
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
1 14
MUBTAAHIJ (IRE)
7 17
KEEN ICE
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

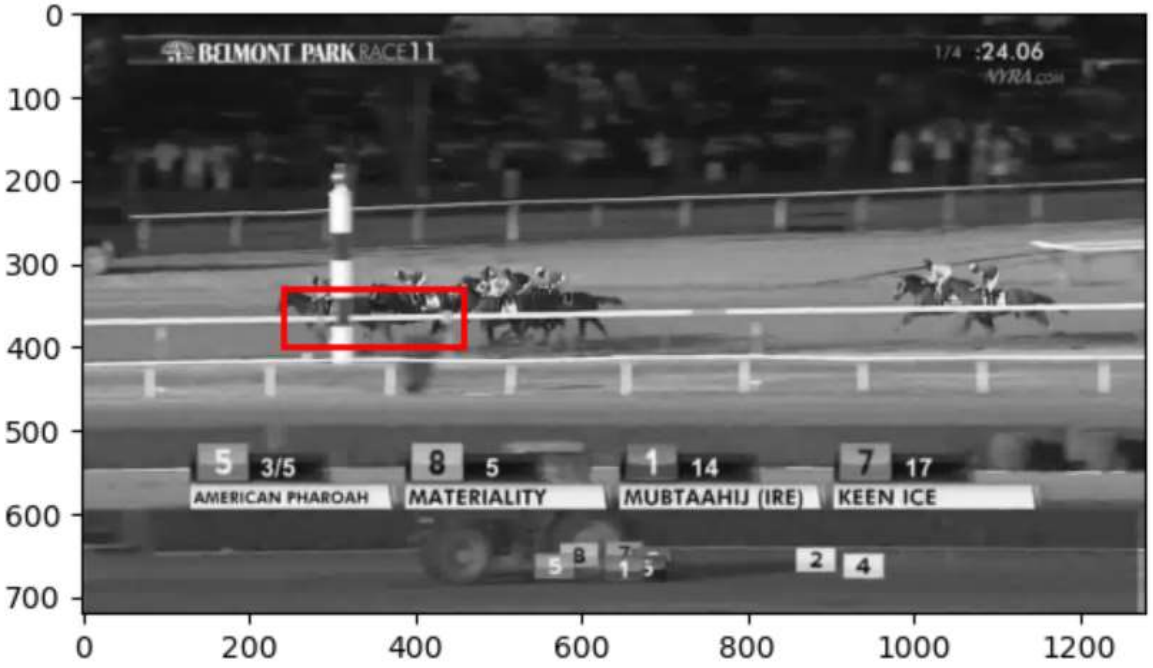

BELMONT PARK RACE 11
1/4 :24.06
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
7 17
KEEN ICE
1 14
MUBTAAHIJ (IRE)
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

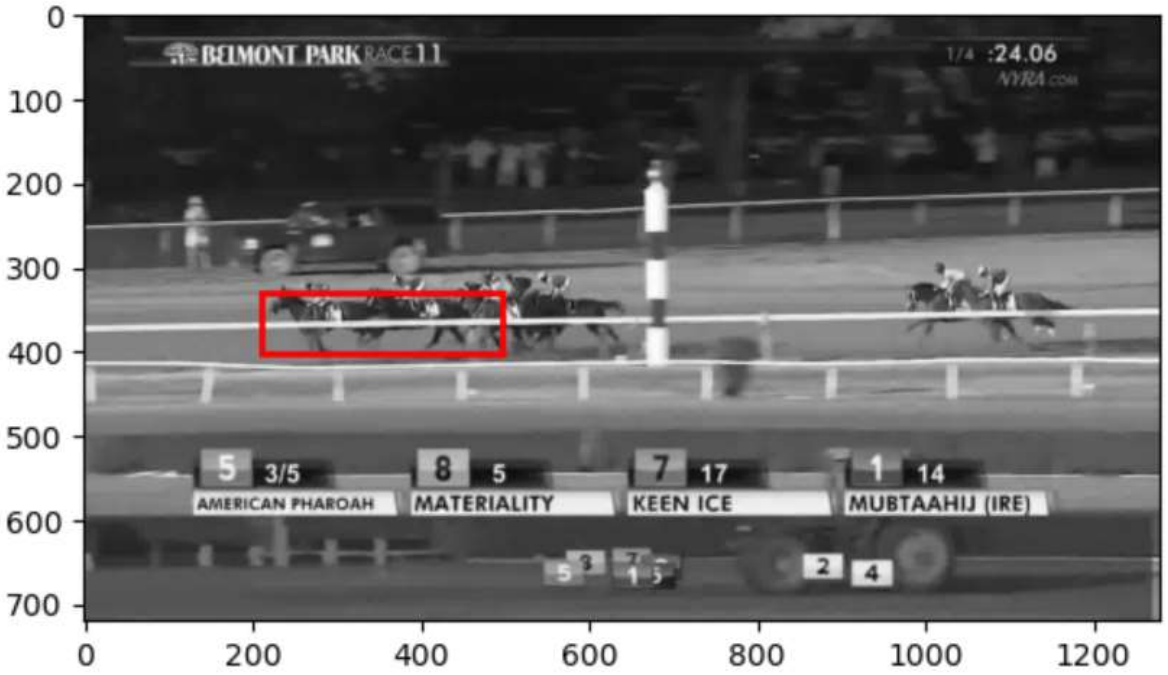

BELMONT PARK RACE 11
1/4 :24.06
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
7 17
KEEN ICE
1 14
MUBTAAHIJ (IRE)
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

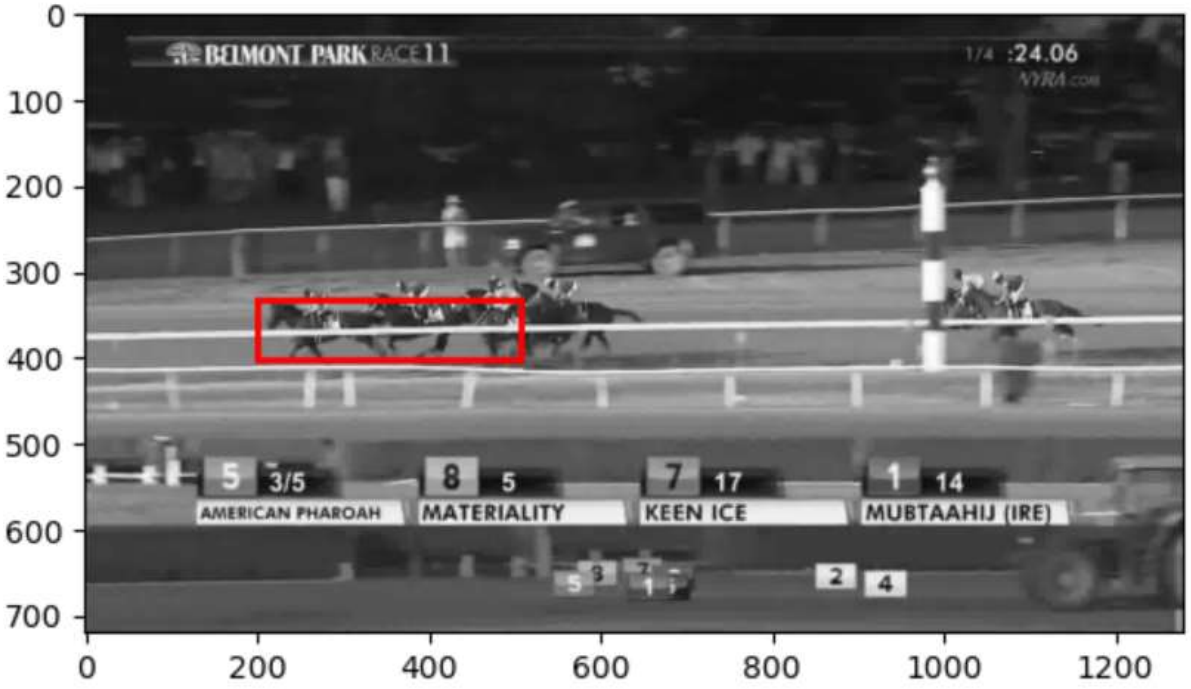

BELMONT PARK RACE 11
1/4 :24.06
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
7 17
KEEN ICE
1 14
MUBTAAHIJ (IRE)
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


BELMONT PARK RACE 11
1/4 :24.06
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
7 17
KEEN ICE
1 14
MUBTAAHIJ (IRE)
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

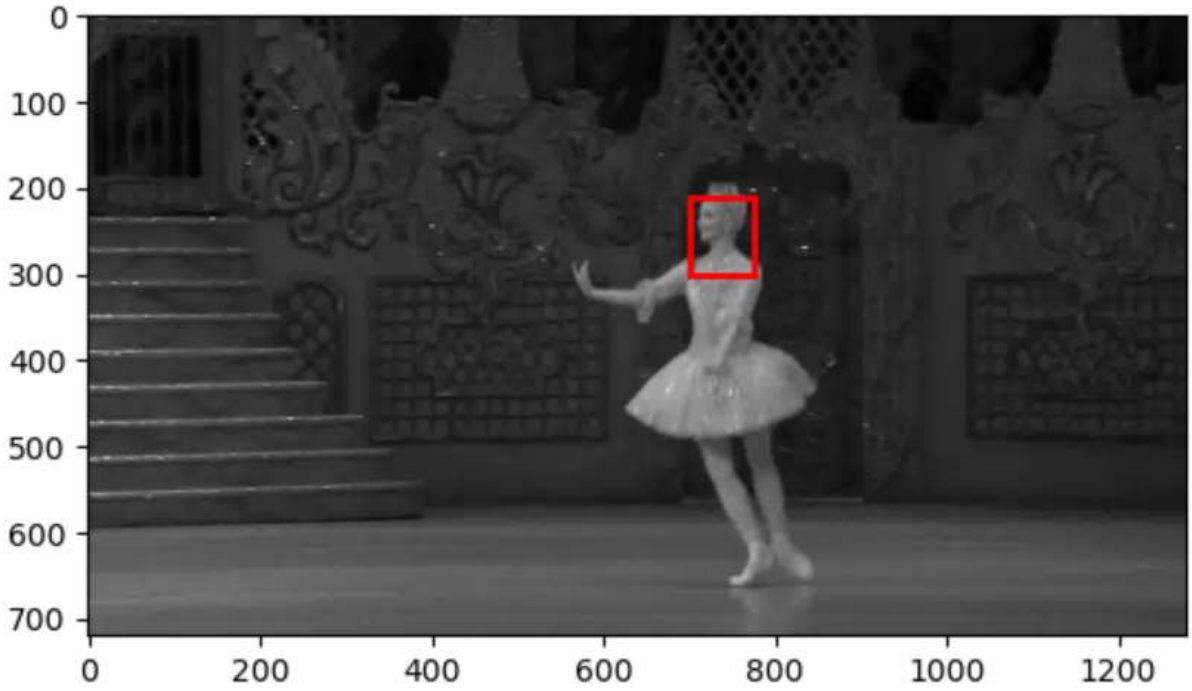

BELMONT PARK RACE 11
1/4 :24.06
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
7 17
KEEN ICE
1 14
MUBTAAHIJ (IRE)
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

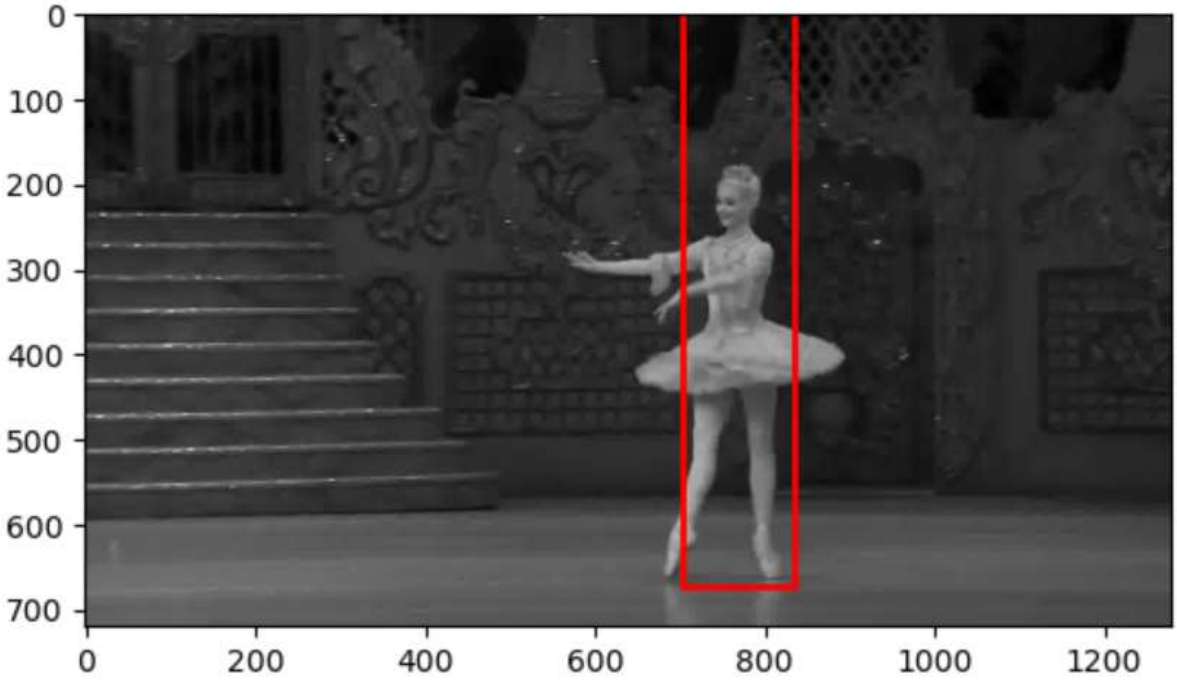

BELMONT PARK RACE 11
1/4 :24.06
NYRA.COM
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
1 14
MUBTAAHIJ (IRE)
7 17
KEEN ICE
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

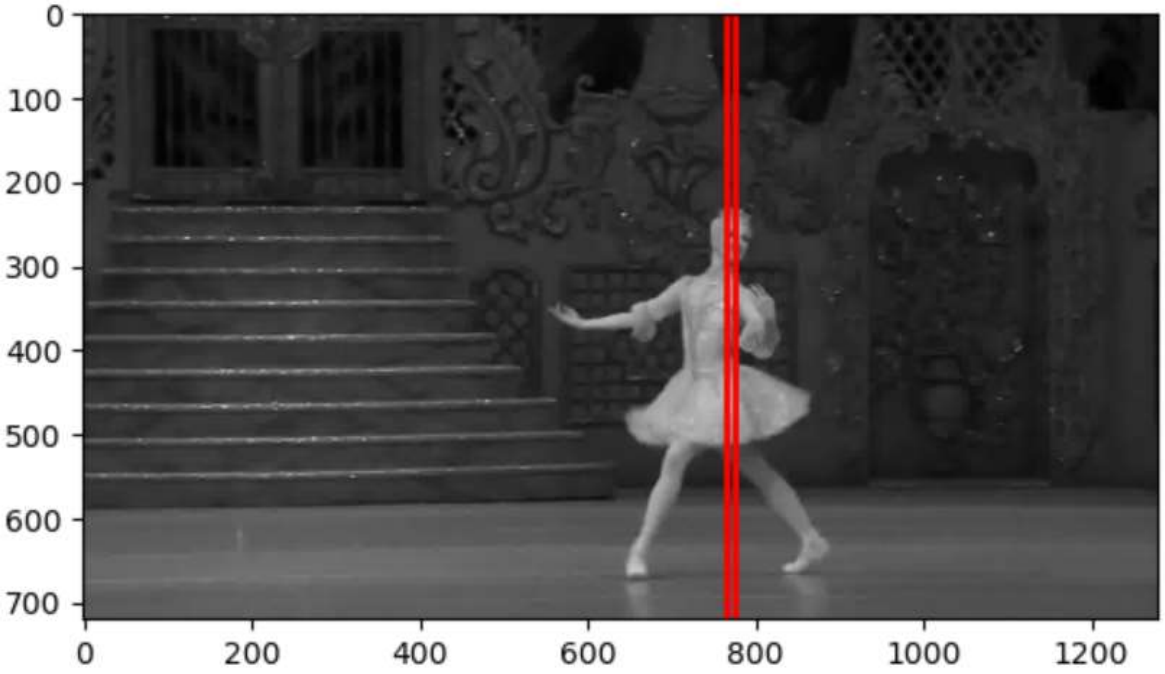

BELMONT PARK RACE 11
1/4 :24.06
NYRA.COM
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
7 17
KEEN ICE
1 14
MUBTAAHIJ (IRE)
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

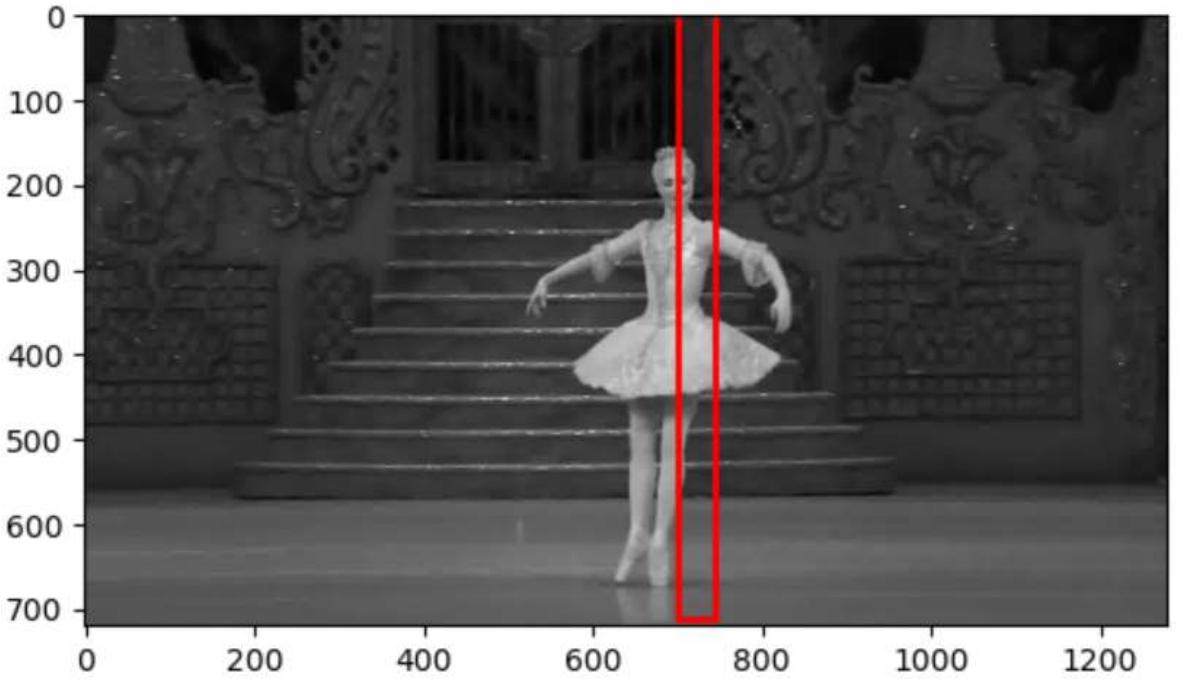

BELMONT PARK RACE 11
1/4 :24.06
5 3/5
AMERICAN PHAROAH
8 5
MATERIALITY
7 17
KEEN ICE
1 14
MUBTAAHIJ (IRE)
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

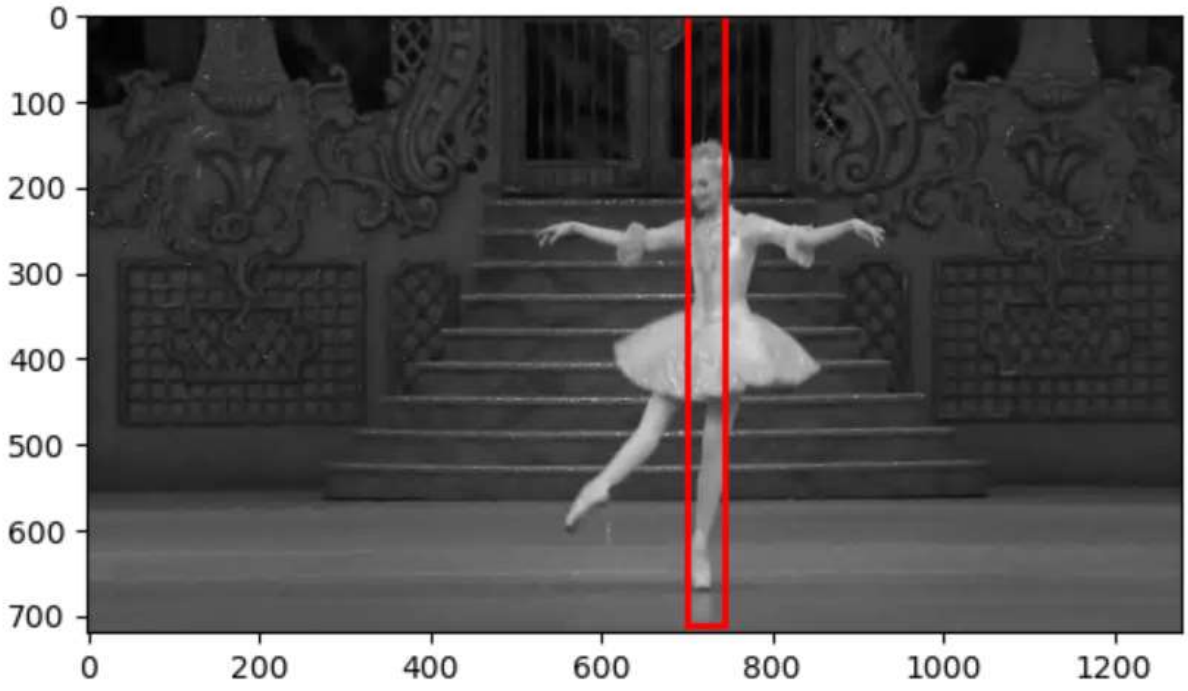
5. Ballet

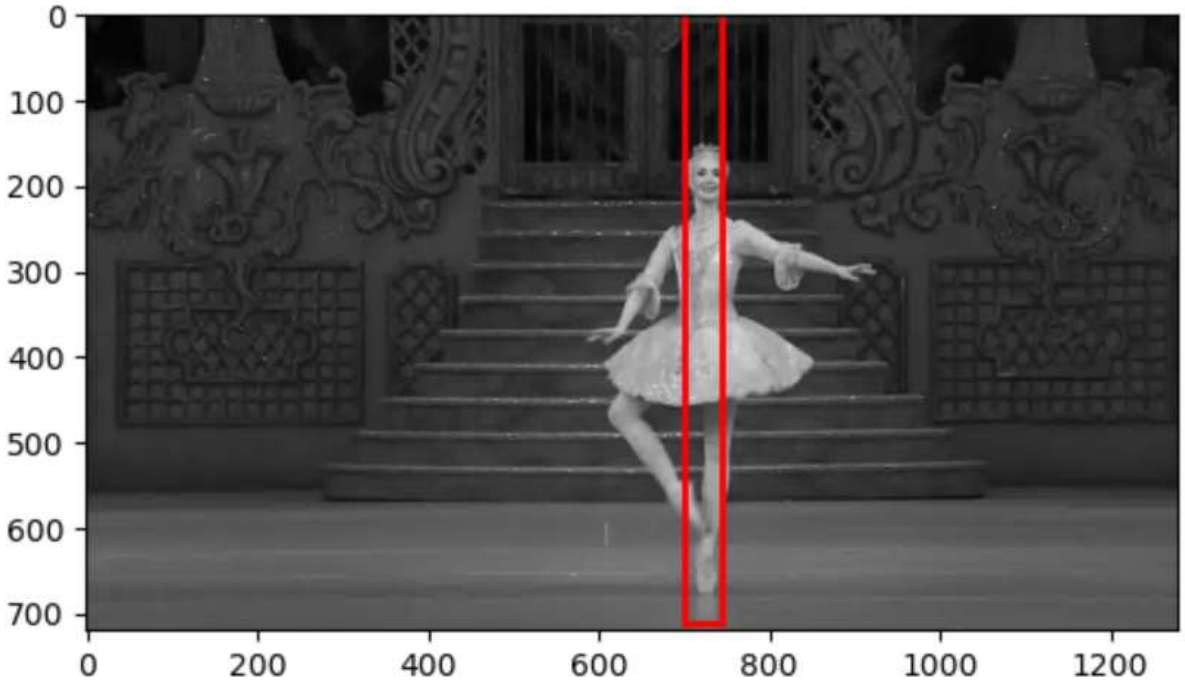

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

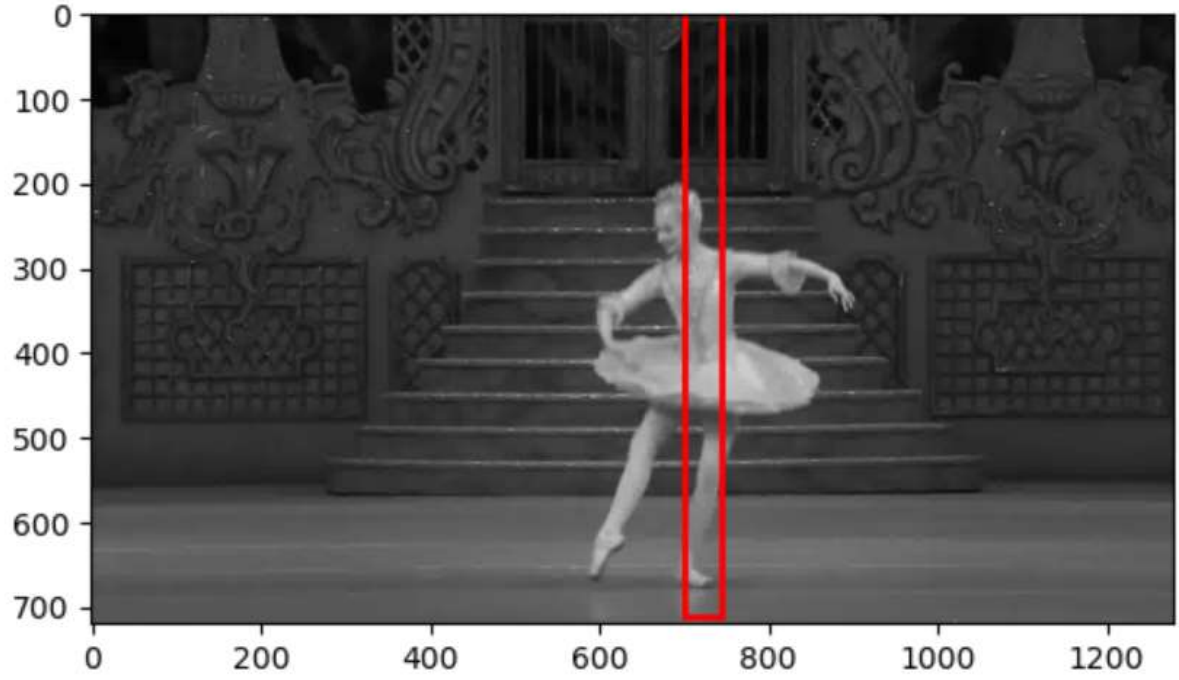

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

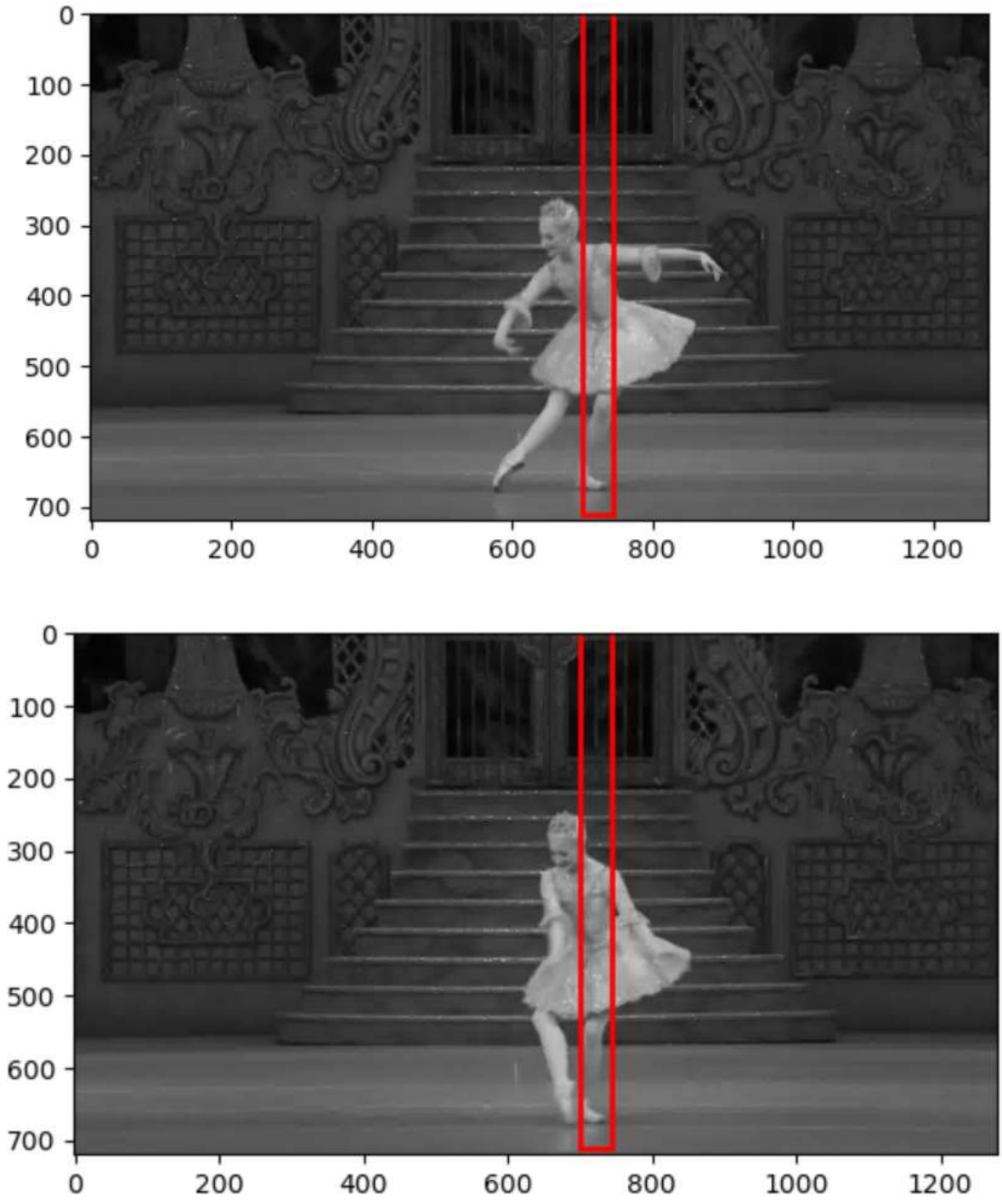

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200



Performance comparision of each dataset

Car1 - The Matthews-Baker tracking technique is surpassed by the Lucas-Kanade Affine matrix algorithm. Once the automobile reaches the shadows, the Matthews-Baker algorithm screws up the tracking.

Car2 - On this dataset, the Lucas-Kanade Affine and Matthews-Baker tracking both appear to perform poorly. That may be because when the traffic sign enters the scene, it causes an increase in the scaling component of the affine filter learned because it was previously mistaken for being a part of the car. The Lucas-Kanade translation only implementation did not encounter this problem because this was not practicable.

Landing - On this dataset, the Lucas-Kanade Affine and Matthews-Baker tracking both perform admirably. The translation alone implementation cannot obtain the proper bounding box because no scaling factor is present.

Race - The performances of Matthews-Baker tracking and Lucas-Kanade Affine are comparable. The tracked horse is within the bounding box, along with a portion of another horse.

Ballet - On this dataset, Matthews-Baker performs poorly because its bounding box is once again entirely outstretched. When it comes to tracking, Lucas-Kanade affine also performs poorly.

Because there are no scale factors in the lucas-kanade translation only implementation, it cannot track objects accurately because their size would change as they moved in front of or away from the camera.

However, both the Matthews-Baker tracking and the affine implementation overcome this by using many parameters to create the affine filter matrix.

Compared to Lucas-Kanade, Matthews-Baker is substantially faster since the Hessian only needs to be computed once. In Lucas-Kanade, the Hessian must be computed for each iteration.

All of these algorithms, nevertheless, only function when a select few requirements are met - A point's brightness does not change over time, and there is very little x and y displacement between two frames. Therefore, if these requirements are not met—car1 having abrupt brightness changes when it emerges from the shadows and ballet having a few instances of abrupt movement—the algorithms fail.

Q2.2

Performance comparision of each dataset

Car1 - The Matthews-Baker tracking technique is surpassed by the Lucas-Kanade Affine matrix algorithm. Once the automobile reaches the shadows, the Matthews-Baker algorithm screws up the tracking.

Car2 - On this dataset, the Lucas-Kanade Affine and Matthews-Baker tracking both appear to perform poorly. That may be because when the traffic sign enters the scene, it causes an increase in the scaling component of the affine filter learned because it was previously mistaken

for being a part of the car. The Lucas-Kanade translation only implementation did not encounter this problem because this was not practicable.

Landing - On this dataset, the Lucas-Kanade Affine and Matthews-Baker tracking both perform admirably. The translation alone implementation cannot obtain the proper bounding box because no scaling factor is present.

Race - The performances of Matthews-Baker tracking and Lucas-Kanade Affine are comparable. The tracked horse is within the bounding box, along with a portion of another horse.

Ballet - On this dataset, Matthews-Baker performs poorly because its bounding box is once again entirely outstretched. When it comes to tracking, Lucas-Kanade affine also performs poorly.

Because there are no scale factors in the lucas-kanade translation only implementation, it cannot track objects accurately because their size would change as they moved in front of or away from the camera.

However, both the Matthews-Baker tracking and the affine implementation overcome this by using many parameters to create the affine filter matrix.

Compared to Lucas-Kanade, Matthews-Baker is substantially faster since the Hessian only needs to be computed once. In Lucas-Kanade, the Hessian must be computed for each iteration.

All of these algorithms, nevertheless, only function when a select few requirements are met - A point's brightness does not change over time, and there is very little x and y displacement between two frames. Therefore, if these requirements are not met—car1 having abrupt brightness changes when it emerges from the shadows and ballet having a few instances of abrupt movement—the algorithms fail.

Q3

YOUR ANSWER HERE

Q4 (Extra Credit) Short notes on important optical flow papers (15 points)

In this section we will go over three important optical flow papers and summarize them. For each paper, please follow these guidelines:

- Please read the papers in detail, focussing on the method described in the paper.
- For each paper, write 5 itemized points (6 max, 4 min) describing the method the authors use to solve the problem.

- Each point should have *no more* than *2 medium length sentences* and *a math equation*. You will **lose points for verbose descriptions**.
- By reading your summary, a person who is motivated about the optic flow problem and has the background on what optic flow is, should understand how the authors posed and solved the problem.
- You may add one point (not exceeding max 6 points) to mention something interesting about the results of the paper. E.g. how well does it generalize, how much real world data it needs etc.

Paper 1: GOTURN (5 Pts)

Learning to Track at 100 FPS with Deep Regression Networks. Held, Savarese and Thrun. ECCV'16 (https://davheld.github.io/GOTURN/GOTURN.html)

Additional material: A PyTorch implementation (https://github.com/nrupatunga/goturn-pytorch)

Answer:

- In this paper, first video frames are input to the neural network, and the network outputs the location of the tracked item in each frame one by one.
- If a video contains more than one objects, an image of the target item is provided as input.
- Herein, the tracker needs to be aware of the target object's past location in order to locate it in the current frame. Based on the object's previous location, a search area is selected in our current frame. The search region is used to crop the current frame, which is then input into our network.
- The target object's predicted mean location is given by:
$$\mathbf{c_0} = (\mathbf{c_0 x}, \mathbf{c_0 y})$$
is where the crop of the current frame, t, is centered. The constant position motion model is therefore represented by setting $\mathbf{c}_0$ to equal $\mathbf{c}$. The current frames' dimensions are $k_2 w$ and $k_2 h$, where w and h are the predicted bounding box dimensions in the previous frame and k_2 is the search radius for the target item.
- Note that in case of single-target tracking, the tracker is initialized using a ground-truth bounding box from the first frame. In predicting where the object will be in frame t, clipped pictures from frames t_1 and t are input into the network at each succeeding frame.
- For the remaining frames of the video, frames are recropped and supplied into the network. The network will then follow the movement of the target item throughout the full film.

Paper 2: RAFT (5 Pts)

RAFT: Recurrent All-Pairs Field Transforms for Optical Flow. Teed and Deng ECCV'20 (https://github.com/princeton-vl/RAFT)

Additional material: Implementation (https://github.com/princeton-vl/RAFT), Talk (unofficial) (https://www.youtube.com/watch?v=r3ZtW30exoo)

Answer:

- The algorithm in the paper has three distinct differentiable parts which are later combined in the model architecture that is end-to-end trainable: feature extraction, computing visual similarity, and the iterative updates.
- The input photos are processed by a neural network to extract features. The input images are mapped to dense feature maps at a lower resolution by the feature encoder network, which is applied to both I1 and I2.
- Visual similarity is found using the process of building a complete correlation volume between all pairs.
- The dot product of each pair of feature vectors yields the correlation volume.
- From a starting point of $\mathbf{f}_0 = 0$, the update operator estimates a series of flow estimates $\mathbf{f_1}, \ldots, \mathbf{f_N}$ and then generates an update direction Δf with each iteration, which is applied to the current estimate: $\mathbf{f}_{k+1} = \Delta f + f_{k+1}$.
- The network has been supervised on the l1 Loss between the expected and ground truth flow throughout the entire sequence of predictions $\mathbf{f_1}, \ldots, \mathbf{f_N}$ with increasing weights.

Paper 3: GM Flow (5 Pts)

GMFlow: Learning Optical Flow via Global Matching. Xu et al. CVPR'22 (https://arxiv.org/pdf/2111.13680.pdf)

Additional material: Implementation (https://github.com/haofeixu/gmflow)

Answer:

- A weight-sharing convolutional network is used to extract downsampled dense features from two successive video frames, I1 and I2, first. $\mathbf{F}1, F2 \in R^{H \times W \times D}$.
- $\mathbf{H}, W, D$ stand for height, width, and feature dimension, respectively.
- The relation is recognized using a differentiable matching layer.
- The softmax procedure is used to normalize the final two dimensions of C.
- The correspondance is calculated by taking a weighted average of the 2D coordinates of the pixel grid with the matching distribution M.
- Optical flow V is found using the difference between the matching pixel coordinates.
- L1 loss is used to supervise all flow estimates for all ground truths.

Q5.1 Loading the bounding boxes, video and visualization

In [ ]:

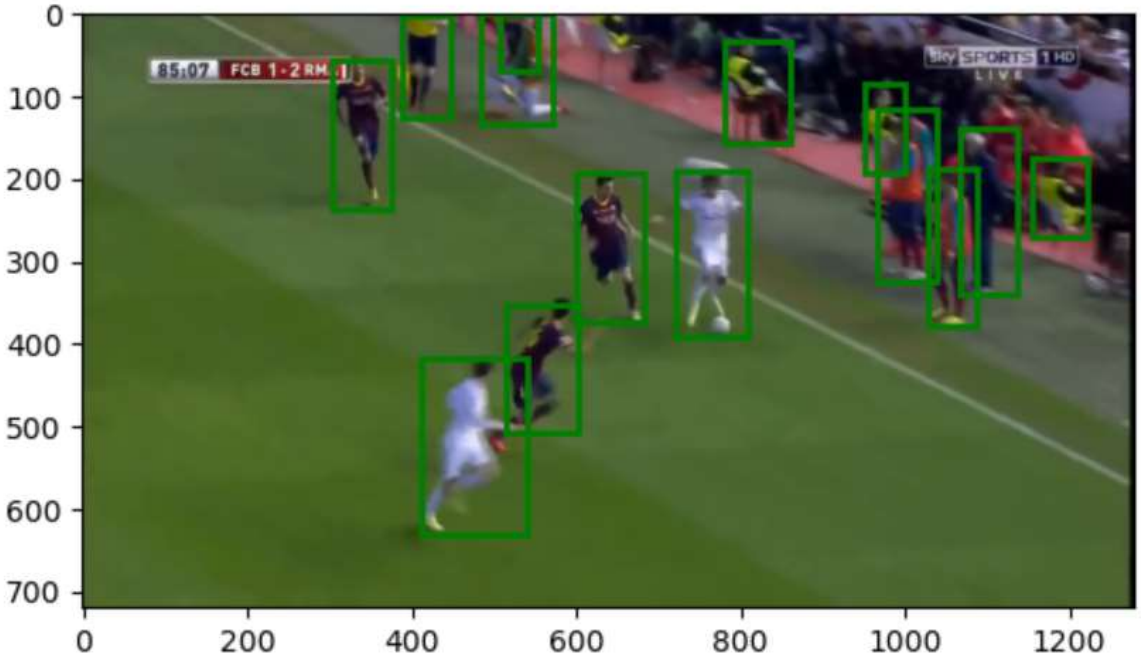
```
def load_images_and_boxes(img_path, box_path):

    data = 'soccer_images'
    imgs = np.load(img_path % data,allow_pickle = True)

    data_box = 'soccer_boxes'
    f = open(box_path % data_box)
    boxes = json.load(f)

    return imgs, boxes
c=['red','blue','grey','green','orange','black','white','brown','cyan','violet',
def render_single_frame(image, bboxes, colors = False):

    if colors == 1:
        fig = plt.figure(1)
        ax = fig.add_subplot(111)
        for i in bboxes:
            ax.add_patch(patches.Rectangle((i[0],i[1]),i[2]-i[0]+1,i[3]-i[1]+1,
                                           linewidth = 2, edgecolor = 'green', f
        plt.imshow(image)
        plt.show()
        ax.clear()
    else:
        fig = plt.figure(1)
        ax = fig.add_subplot(111)
        o = 0
        for i in bboxes:
            ax.add_patch(patches.Rectangle((i[0],i[1]),i[2]-i[0]+1,i[3]-i[1]+1,
                                           linewidth=2, edgecolor=colors[o], fil

            o = o + 1
        plt.imshow(image)
        plt.show()
        ax.clear()
```

In [ ]:

```
a, b = load_images_and_boxes('./%s.npy','./%s.json')
render_single_frame(a[124],b[124]['boxes'], colors = True)
render_single_frame(a[0],b[0]['boxes'], colors = True)
```

Frame - 124

Frame - 1



Q5.2 Implementing a similarity metric

In [ ]:

```
def overlappingArea(l_1, r_1, l_2, r_2):
    xx = 0
    yy = 1

    area_ = abs(l_1[xx] - r_1[xx]) * abs(l_1[yy] - r_1[yy])

    area__ = abs(l_2[xx] - r_2[xx]) * abs(l_2[yy] - r_2[yy])

    x_dist = (min(r_1[xx], r_2[xx]) - max(l_1[xx], l_2[xx]))

    y_dist = (min(r_1[yy], r_2[yy]) - max(l_1[yy], l_2[yy]))

    area_ = 0

    if x_dist > 0 and y_dist > 0:
        area_ = x_dist * y_dist

    ar = (area_ + area__ - area_)
    return ar, area_

def compute_iou(boxes1, boxes2):

    len_box1 = len(boxes1)
    len_box2 = len(boxes2)

    m = np.zeros((len_box1,len_box2))
    for i in range(len_box1):
        for j in range(len_box2):

            a,b = overlappingArea(np.array((boxes1[i][0],boxes1[i][1])),
                                  np.array((boxes1[i][2],boxes1[i][3])),np.array
                                  np.array((boxes2[j][2],boxes2[j][3])))
            m[i][j] = b/a

    return m
```

In [ ]:

```
array = compute_iou(b[124]['boxes'],b[125]['boxes'])
print(array)
render_single_frame(a[124],b[124]['boxes'], colors=1)
render_single_frame(a[125],b[125]['boxes'], colors=1)
```

```
[[0.         0.95658136]
 [0.98216011 0.        ]
 [0.         0.        ]]
```

Q5.3 Matching with the Hungarian Algorithm

In [ ]:

```
def compute_assignment(iou, threshold=0.8):

    thresh = threshold
    shape0 = iou.shape[0]
    shape1 = iou.shape[1]
    b1 = np.zeros(shape0)
    b2 = np.zeros(shape1)

    for i in range(shape0):
        for j in range(shape1):
            if iou[i][j] > thresh:
                b1[i] = 1
                b2[j] = 1

    return np.array(b1), np.array(b2)

iou = compute_iou(b[124]["boxes"], b[125]["boxes"])
print("iou matrix", iou)
b1__iou,b2__iou = compute_assignment(iou, threshold = 0.8)
print("b1_iou" , b1__iou)
print("b2_iou" , b2__iou)
```

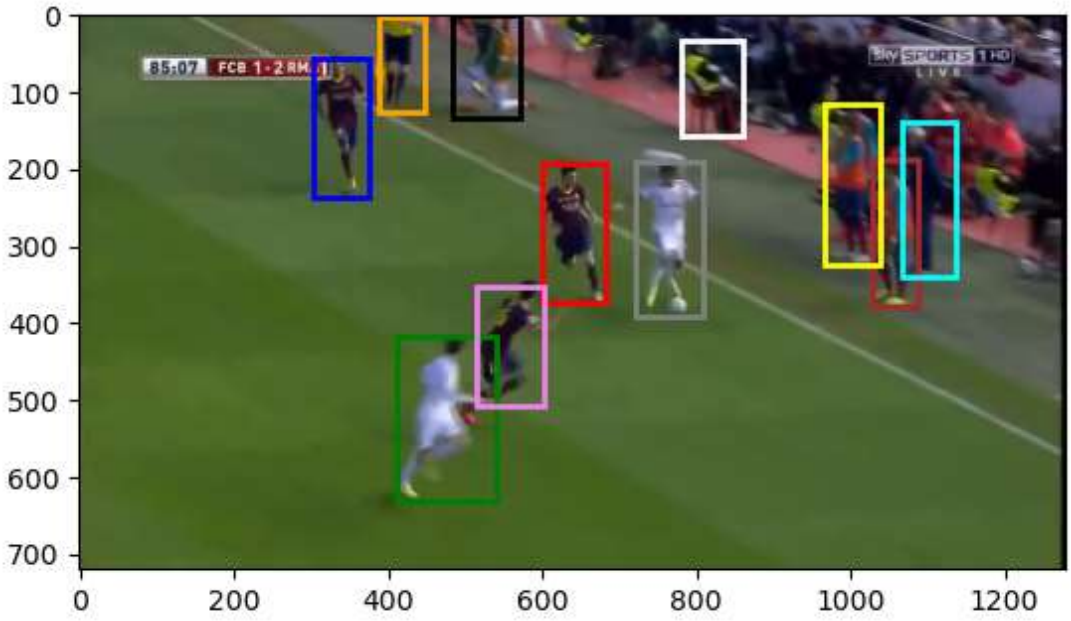
```
iou matrix [[0.         0.95658136]
 [0.98216011 0.        ]
 [0.         0.        ]]
b1_iou [1. 1. 0.]
b2_iou [1. 1.]
```

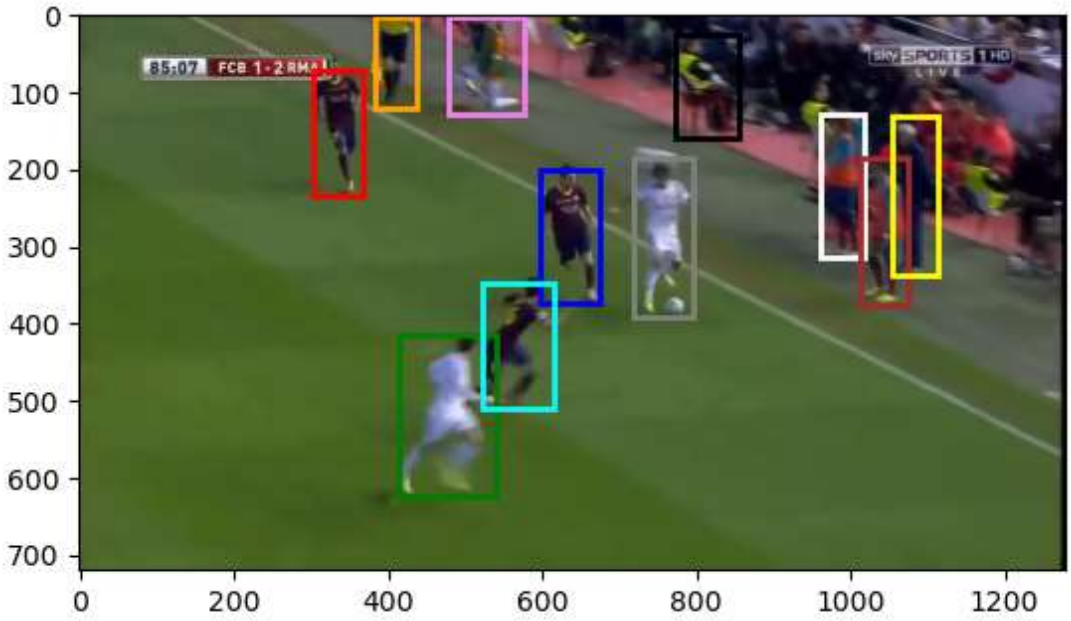
Q5.4 Putting it all together

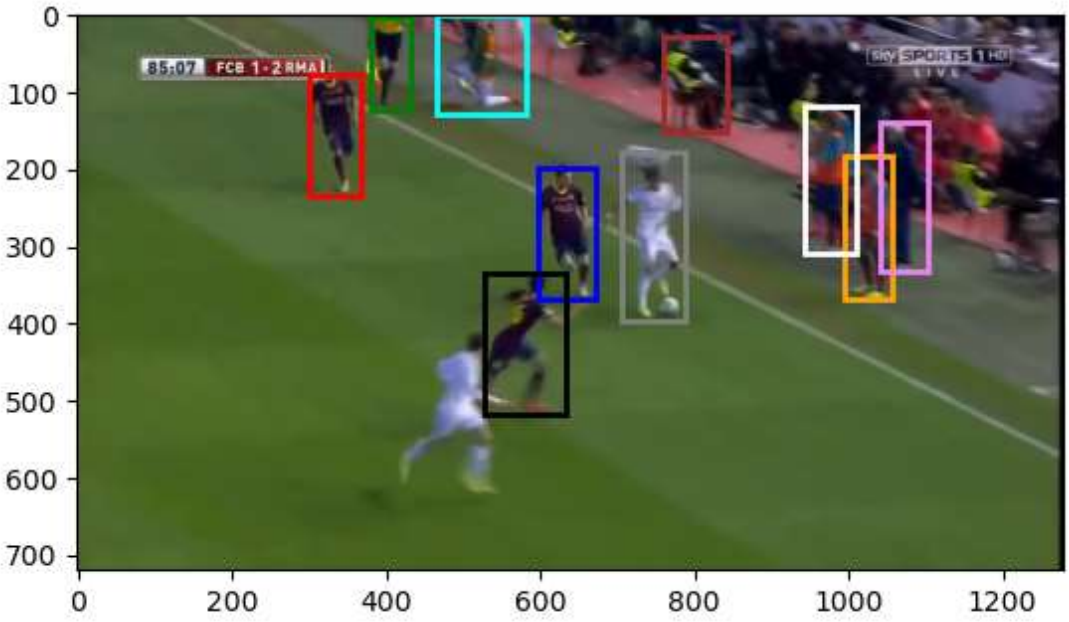
In [ ]:

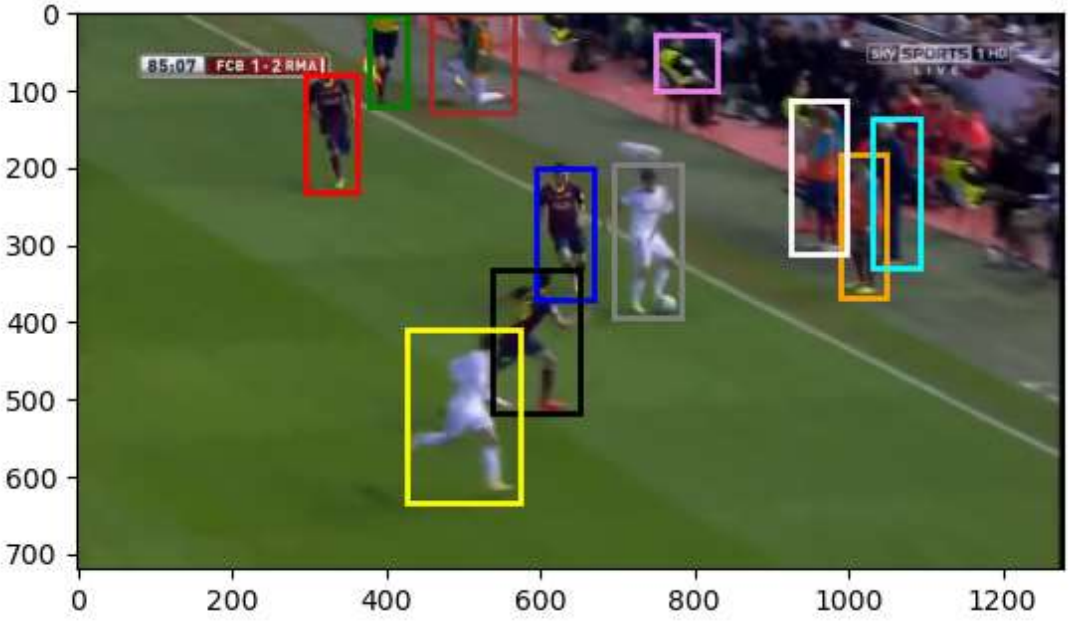
```
def run_tracker(images, boxes,rang, P, K, iou_thresh):

    all_frame0 = []
    all_frame1 = []
    all_frame2 = []
    a = images
    b = boxes

    for i in range(249):
        all_frame0.append(compute_iou(b[i]['boxes'], b[i+1]['boxes']))

    for i in range(249):
        b1, b2 = compute_assignment(all_frame0[i], iou_thresh)
        all_frame1.append(b1)
        all_frame2.append(b2)

    c = []
    for i in range(250-P):
        a_frame = all_frame1[i]
        u = a_frame.shape[0]
        array_u = np.zeros(u)

        for j in range(P):
            if all_frame1[i+j].shape[0] >= u:
                for k in range(u):
                    if a_frame[k] == 1 and all_frame1[i+j][k] == 1:
                        array_u[k] = array_u[k] + 1
            else:
                for o in range(all_frame1[i + j].shape[0]):
                    if a_frame[o] == 1 and all_frame1[i+j][o] == 1:
                        array_u[o] = array_u[o]+1
        c.append(array_u)
    import copy

    for i in range(len(c)):
        for j in range(c[i].shape[0]):
            if c[i][j] < P-K-1:
                c[i][j] = 0
    b_a = []
    b_d = []
    for i in range(len(c)):
        b_d = []
        for j in range(c[i].shape[0]):
            if c[i][j]>0:
                b_d.append(b[i]['boxes'][j])
        b_a.append(b_d)
    for i in range(len(c)):
        print("No of bounding boxes - ",len(b_a[i]))
```

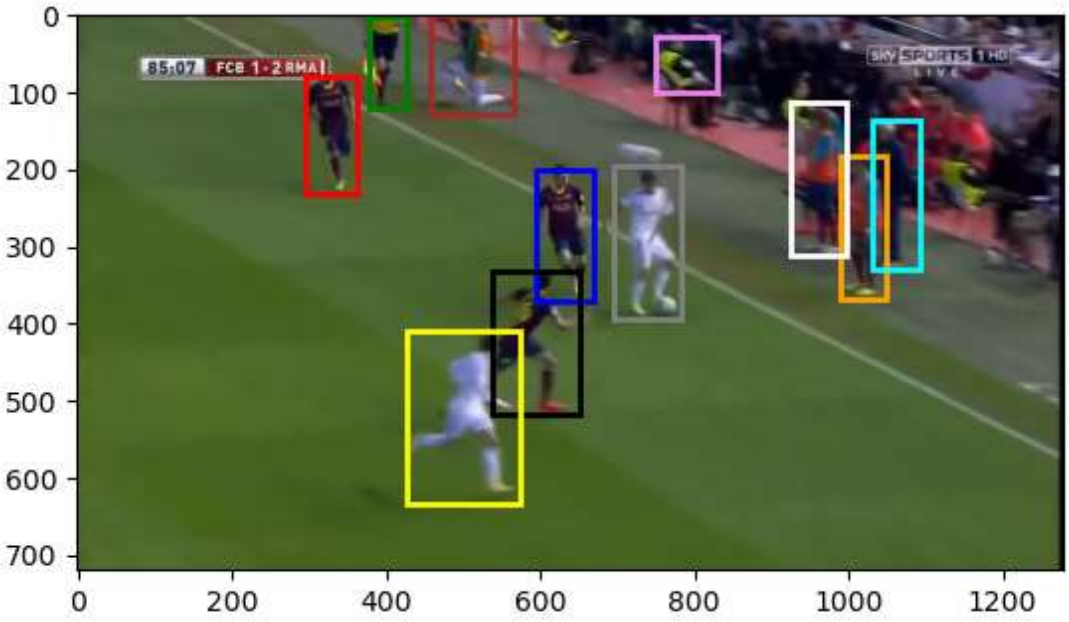
```
        render_single_frame(a[i],b_a[i],rang)

p = run_tracker(a, b, c, 10, 3, 0.4)
```

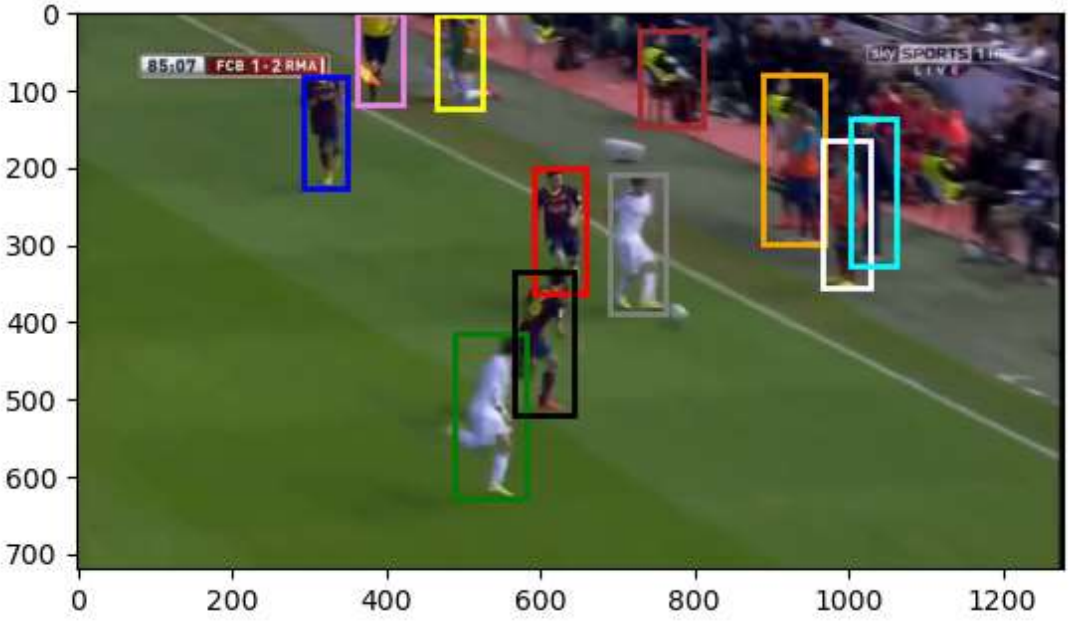


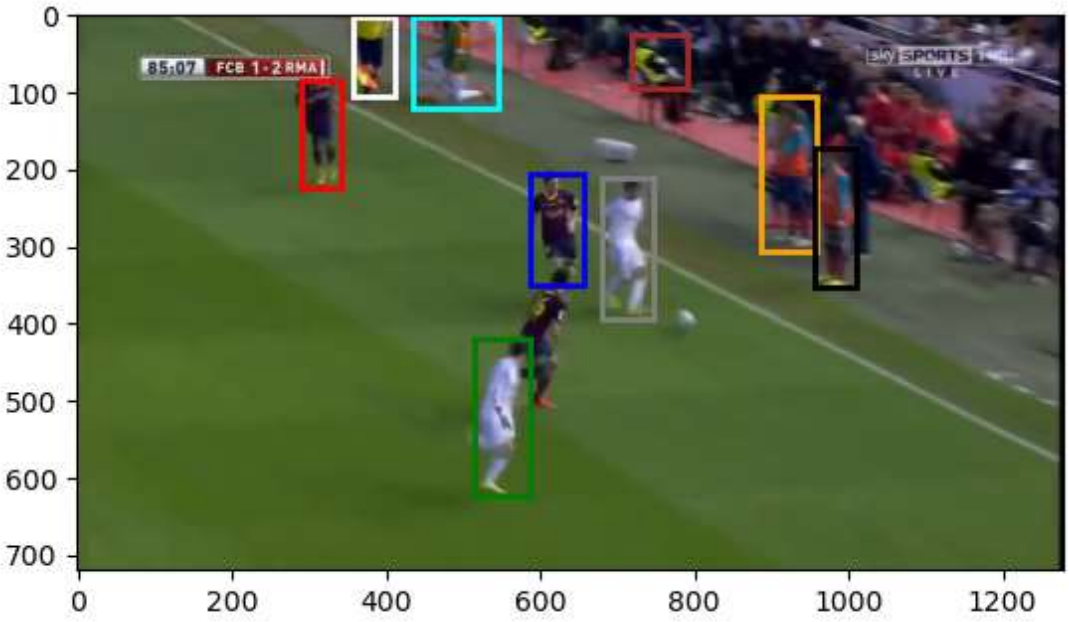


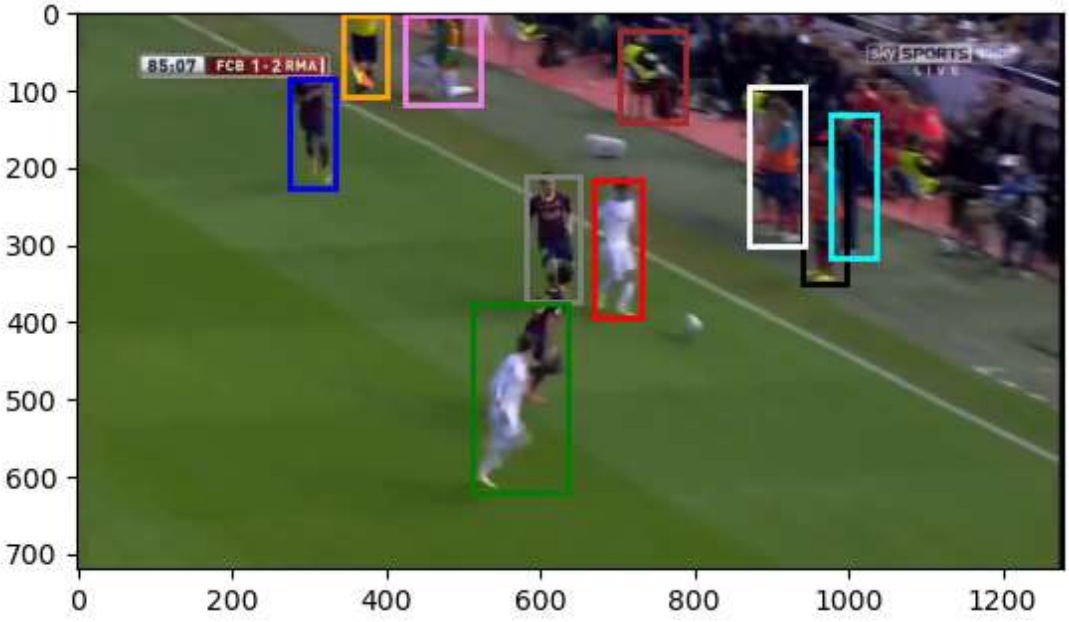

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200
85:07
FCB 1-2 RMA
SKY SPORTS 1 HD

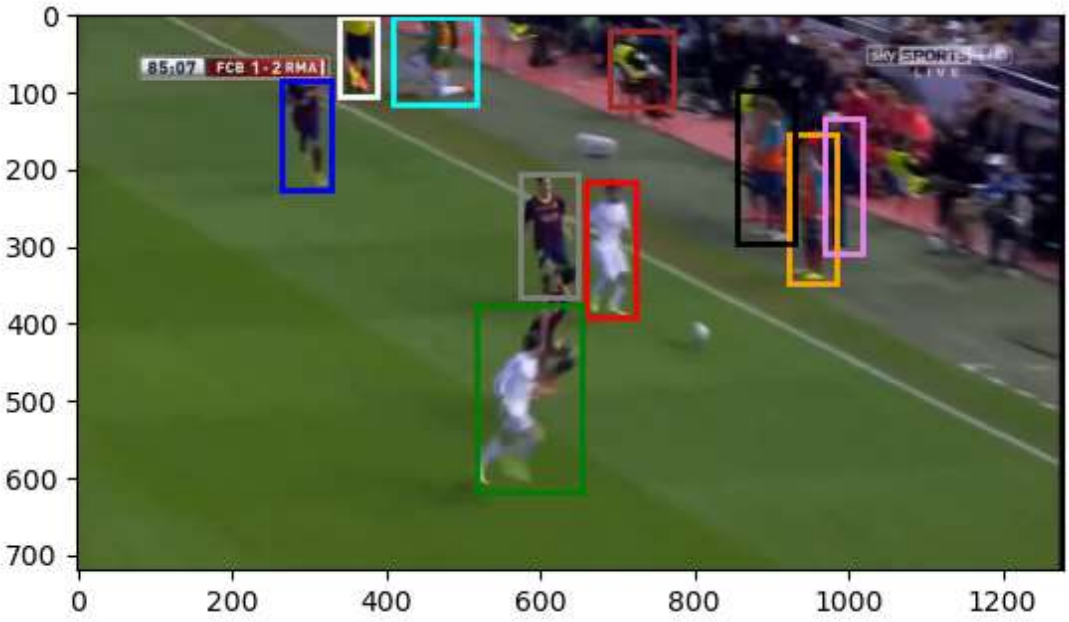

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200
85:07
FCB 1-2 RMA
SKY SPORTS 1 HD

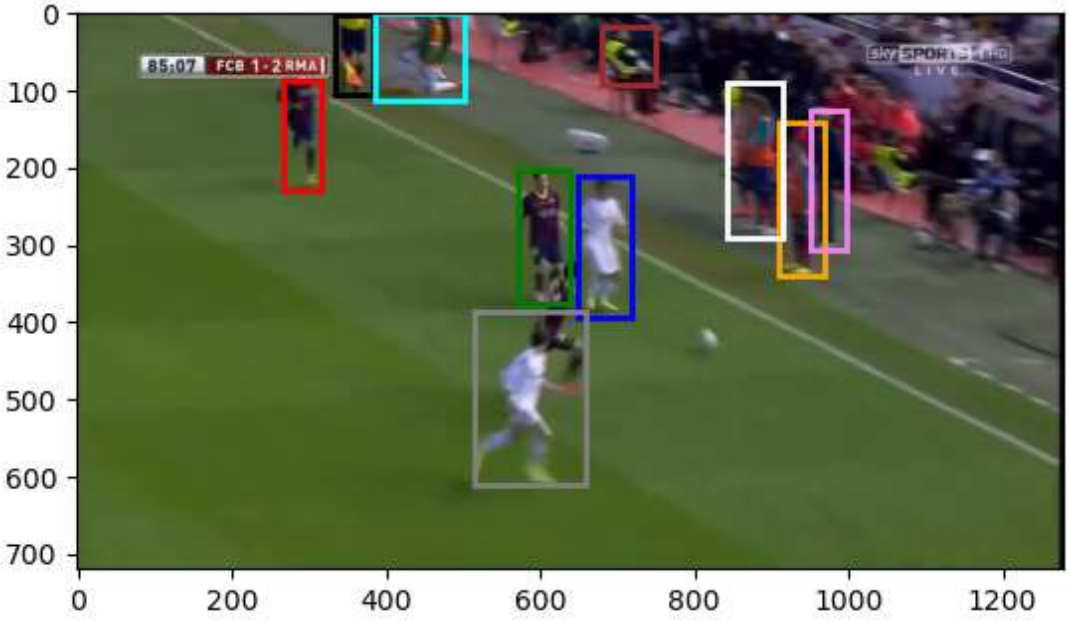

0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200
85:07 FCB 1-2 RMA
SKY SPORTS 1 HD


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200
85:07 FCB 1-2 RMA
SKY SPORTS 1


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200
85:07
FCB 1-2 RMA


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200
85:07
FCB 1-2 RMA





Q - Question from 12/7 guest lecture

FIVE of the worst annotation examples

Image 1

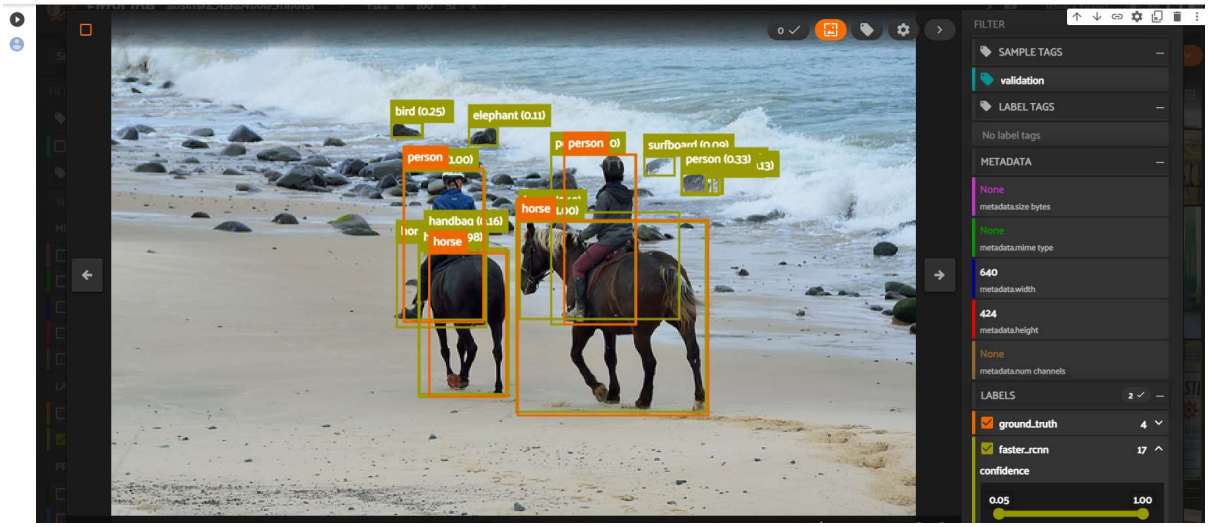


Image 2

Image 3

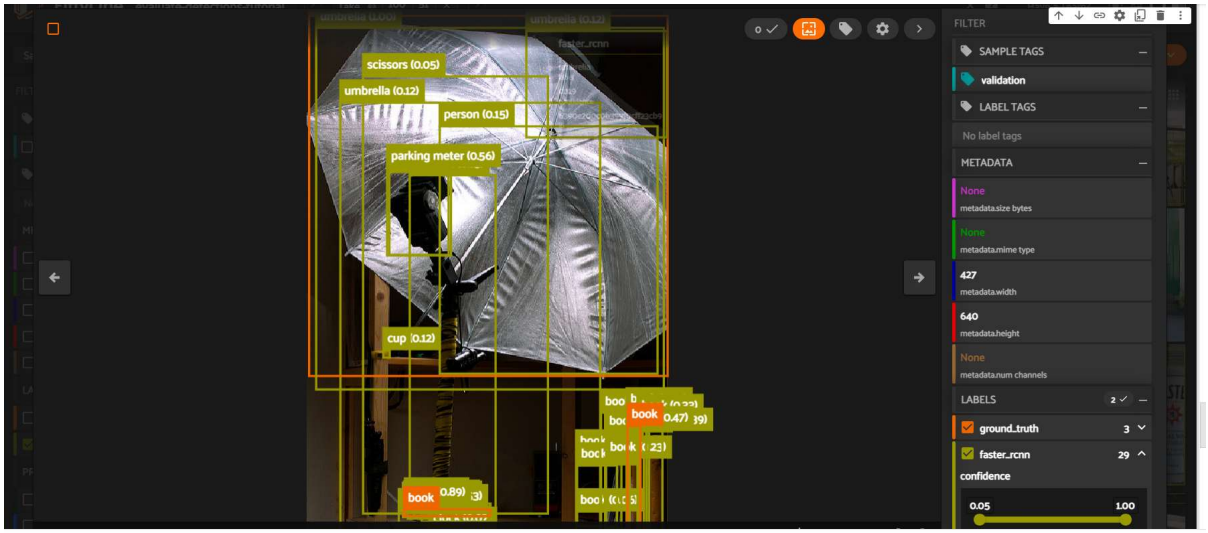


Image 4

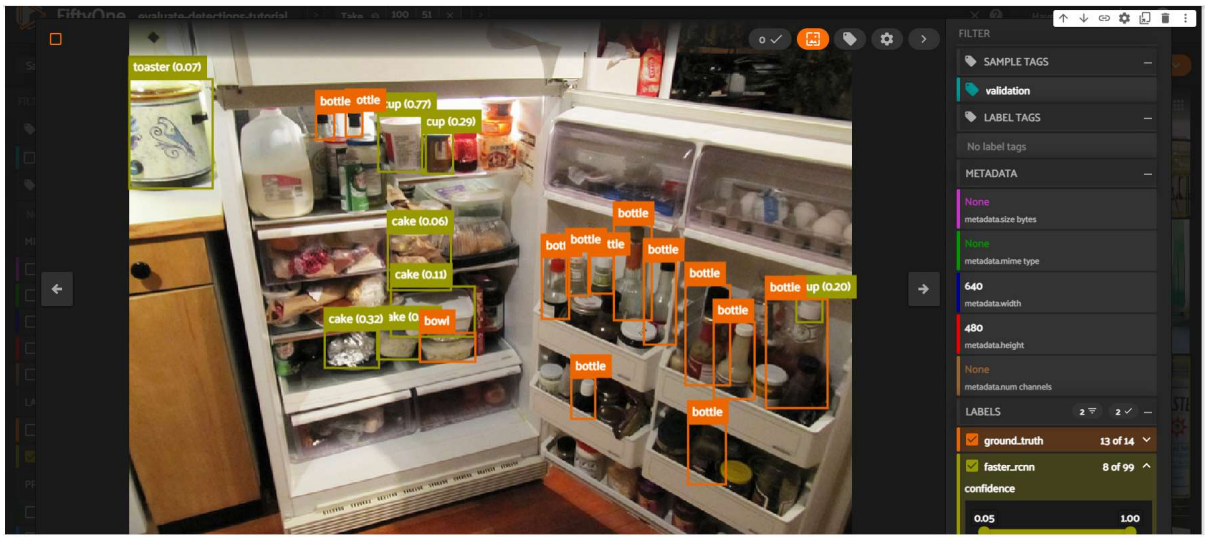
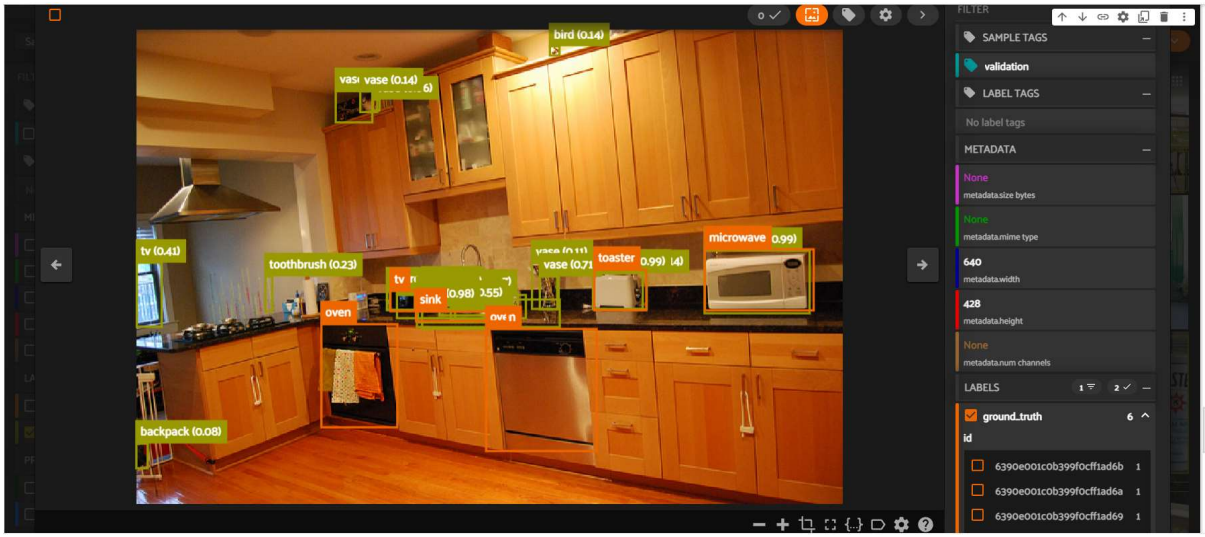


Image 5


bird (0.14)
vase (0.14)
tv (0.41)
toothbrush (0.23)
sink
oven
toaster 0.99)
microwave 0.99)
backpack (0.08)
FILTER
SAMPLE TAGS
validation
LABEL TAGS
No label tags
METADATA
None
metadata.size bytes
None
metadata.mime type
640
metadata.width
428
metadata.height
None
metadata.num channels
LABELS
ground_truth
id
6390e001c0b399f0cff1ad6b
6390e001c0b399f0cff1ad6a
6390e001c0b399f0cff1ad69

In [ ]: