1.

```
import cv2
import numpy as np

# load image
input_image = cv2.imread('/Users/kasi/Downloads/nadia2.jpg', cv2.IMREAD_GRAYSCALE)
template = cv2.imread('/Users/kasi/Downloads/nadiatemplate.jpg', cv2.IMREAD_GRAYSCALE)

# search matches
result = cv2.matchTemplate(input_image, template, cv2.TM_CCOEFF_NORMED)

# normalize result
normalized_result = cv2.normalize(result, None, 0, 255, cv2.NORM_MINMAX, dtype=cv2.CV_8U)

# find min max
min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(result)

# white retangle over the image
top_left = max_loc
bottom_right = (top_left[0] + template.shape[1], top_left[1] + template.shape[0])
cv2.rectangle(input_image, top_left, bottom_right, 255, 2)

cv2.imshow('Detected', input_image)
cv2.imshow('Error Map', normalized_result)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

2.

```
import cv2
import numpy as np

# load image
input_image = cv2.imread('/Users/kasi/Downloads/nadia2.jpg', cv2.IMREAD_GRAYSCALE)
template = cv2.imread('/Users/kasi/Downloads/nadiatemplate.jpg', cv2.IMREAD_GRAYSCALE)

# search matches
```

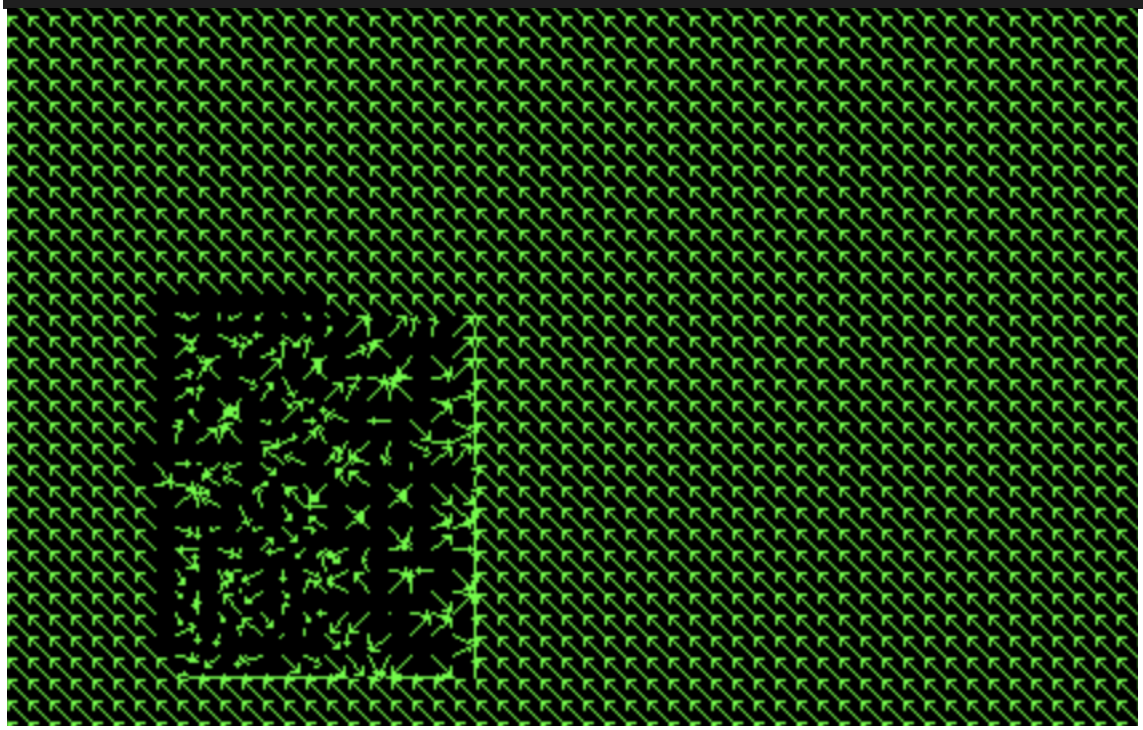
```
result = cv2.matchTemplate(input_image, template, cv2.TM_CCOEFF_NORMED)

# normalize result
normalized_result = cv2.normalize(result, None, 0, 255, cv2.NORM_MINMAX, dtype=cv2.CV_8U)

# find min max
min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(result)

# white retangle over the image
top_left = max_loc
bottom_right = (top_left[0] + template.shape[1], top_left[1] + template.shape[0])
cv2.rectangle(input_image, top_left, bottom_right, 255, 2)

cv2.imshow('Detected', input_image)
cv2.imshow('Error Map', normalized_result)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



3.

```
import cv2
```

```
import numpy as np

def ebma(target, anchor, block_size, search):
    motion_vectors = []

    for y in range(0, target.shape[0], block_size):
        for x in range(0, target.shape[1], block_size):
            best_match = (0, 0)
            min_error = float('inf')

            x_start = max(x - search, 0)
            x_end = min(x + block_size + search, anchor.shape[1])
            y_start = max(y - search, 0)
            y_end = min(y + block_size + search, anchor.shape[0])

            for yy in range(y_start, y_end - block_size + 1):
                for xx in range(x_start, x_end - block_size + 1):
                    block_target = target[y:y+block_size, x:x+block_size]
                    block_anchor = anchor[yy:yy+block_size, xx:xx+block_size]

                    if block_target.shape == block_anchor.shape:
                        mse = np.mean((block_target - block_anchor) ** 2)

                        if mse < min_error:
                            min_error = mse
                            best_match = (xx, yy)

            motion_vectors.append(((x, y), (best_match[0] - x, best_match[1] - y)))

    return motion_vectors

# load the video
cap = cv2.VideoCapture('/Users/kasi/Downloads/vb.mov')

block_size = 8
search = 7
anchor_frame = None
```

```
frame_count = 0
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    target_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    if anchor_frame is not None:
        motion_vectors = ebma(target_frame, anchor_frame, block_size, search)



        quiver = np.zeros((target_frame.shape[0], target_frame.shape[1], 3), dtype=np.uint8)
        for (x, y), (dx, dy) in motion_vectors:
            cv2.arrowedLine(quiver, (x, y), (x + dx, y + dy), (0, 255, 0), 1, tipLength=0.3)

        cv2.imshow('Anchor Frame', anchor_frame)
        cv2.imshow('Target Frame', target_frame)
        cv2.imshow('Motion Vectors', quiver)
        cv2.waitKey(100)

        frame_count += 1
        if frame_count > 10:
            break

    anchor_frame = target_frame.copy()

cap.release()
cv2.destroyAllWindows()
```