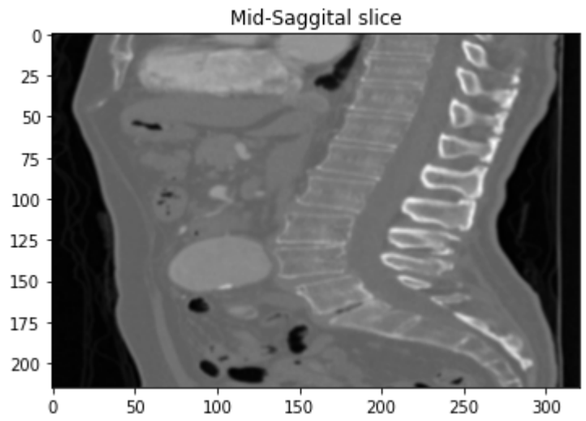
Task 1: Windowing and detection in a CT scan

- Task 1.1. Write a Python code to read the provided CTA abdominal scan (CTA-Abdomen.nrrd) and to find the optimal intensity window for the spine vertebrae (check the figure below) by visualizing the result when choosing the best range. Output a screen shot from the mid axial, coronal and sagittal views, and the best intensity window you selected

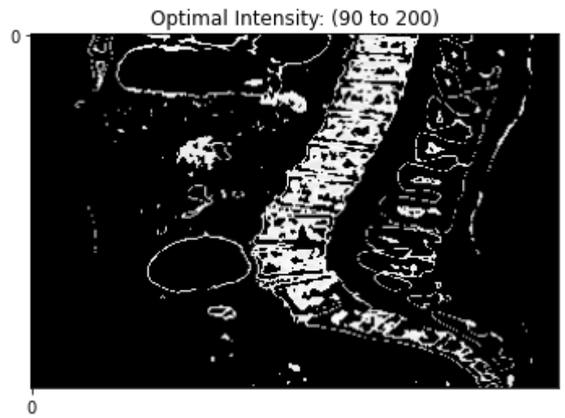
In [ ]:
```
# Importing Necessary Libraries
import numpy as np
import nrrd
import matplotlib.pyplot as plt
import cv2 as cv
from skimage.feature import match_template, peak_local_max
import skimage
```

In [ ]:
```
#Loading the .nrrd file
NRRD_DIR = "/apps/local/shared/HC701/assessment/assignment_2/task_1/CTA-Abdomen.nrrd"
```

In [ ]:
```
def get_mid_slice(NRRD_DIR, show = True):
    # Reads the NRRD image file using the nrrd.read function.
    nrrd_img, header = nrrd.read(NRRD_DIR)
    # The nrrd_img array is transposed. Then rotated by 180 degrees along the X and Y axes.
    nrrd_img = np.rot90(np.rot90(nrrd_img.T))
    mid_slice_idx = nrrd_img.shape[2] // 2
    mid_sag_slice = nrrd_img[:,:,mid_slice_idx]
    if show:
        plt.imshow(mid_sag_slice, cmap="gray")
        plt.title("Mid-Saggital slice")
        plt.show()
    return mid_sag_slice, nrrd_img

mid_sag_slice, _ = get_mid_slice(NRRD_DIR)
```



In [ ]:
```
intensity = [90, 200]


filtered_img = np.where(mid_sag_slice>intensity[1], mid_sag_slice.min(), mid_sag_slice)
filtered_img = np.where(filtered_img<intensity[0], filtered_img.min(), filtered_img)
plt.imshow(filtered_img, cmap="gray")
plt.title("Optimal Intensity: (90 to 200)")
plt.xticks([0])
plt.yticks([0])
plt.show()
```

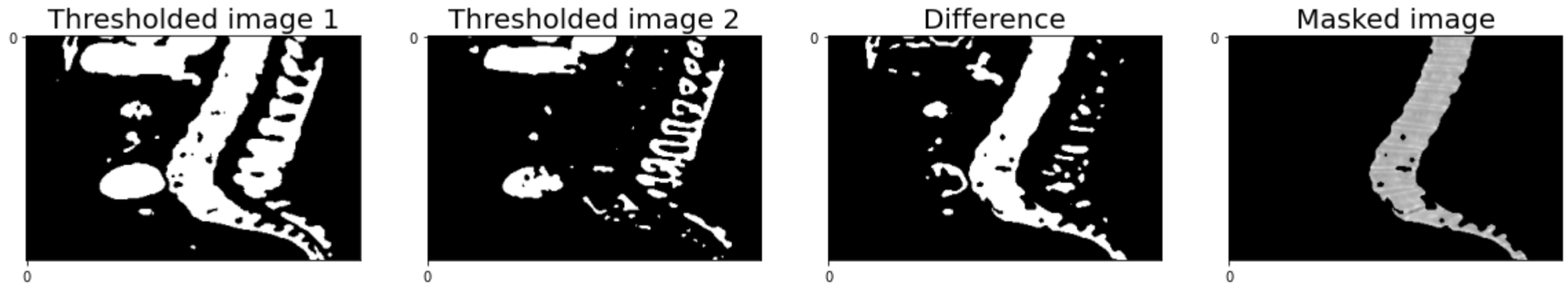


Task 1.2. Following on from Task 1.1, regardless of the intensity window you used:

- Research and identify effective non-ML methods to locate the vertebrae in the 2D sagittal view (see image above, please work on the mid sagittal slice only). Report details about the method you used and the algorithms behind it (maximum 300 words).
- Write a Python code to remove all non-vertebra regions from the 2D image. Report your code and a paragraph (max 100 words) describing what you did. Also report a screenshot of the generated image.

In [ ]:

```
# Function takes an image slice as input and returns a masked version of the slice.
def get_masked_image(img_slice, show = True, show_overlay= False):
     # Reduce noise in the image
    imgray = cv.medianBlur(img_slice,5)

    # Two binary thresholding operations to the median-filtered image.
    ret, thresh1 = cv.threshold(imgray, 90, 255, 0)
    ret, thresh2 = cv.threshold(imgray, 260, 255, 0)

    # Subtract both the thresdold images to get rid of the non-vertebra regions
    diff = cv.medianBlur(cv.subtract(cv.medianBlur(thresh1,5), cv.medianBlur(thresh2,5)),5)
    sub = skimage.morphology.area_opening(diff, area_threshold=500, connectivity=2)
    masked_img = np.where(sub>0, mid_sag_slice, mid_sag_slice.min())

    if show:
        plt.figure(figsize=(20, 15))
        plt.subplot(141)
        plt.title("Thresholded image 1", fontsize = 20)
        plt.imshow(thresh1, cmap="gray")
        plt.xticks([0])
        plt.yticks([0])

        plt.subplot(142)
        plt.title("Thresholded image 2", fontsize = 20)
        plt.imshow(thresh2, cmap="gray")
        plt.xticks([0])
        plt.yticks([0])

        plt.subplot(143)
        plt.title("Difference", fontsize = 20)
        plt.imshow(diff, cmap="gray")
        plt.xticks([0])
        plt.yticks([0])

        plt.subplot(144)
        plt.title("Masked image", fontsize = 20)
        plt.imshow(masked_img, cmap="gray")
        plt.xticks([0])
        plt.yticks([0])
        plt.show()

    elif show_overlay:
        # plt.title("Final Masked Image", fontsize = 20)
        plt.imshow(masked_img, cmap="gray")
        plt.xticks([0])
        plt.yticks([0])
        plt.show()
    return masked_img

masked_img = get_masked_image(mid_sag_slice)
```

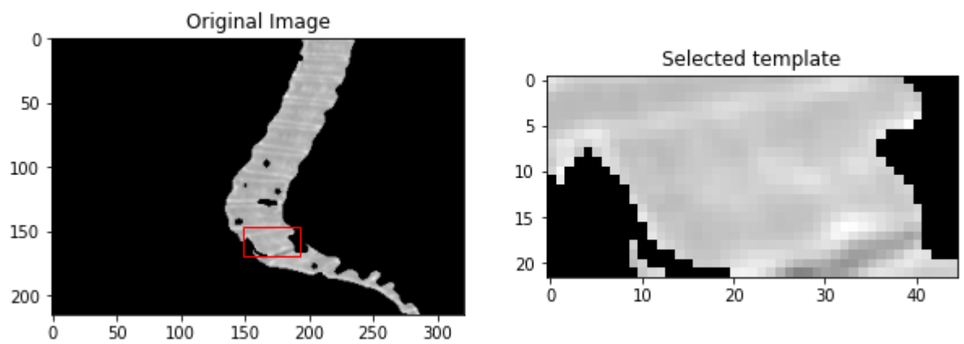


Develop a non-ML method to put a box around each vertebra and report a screen shot of the boxes.

Template Matching

In [ ]:

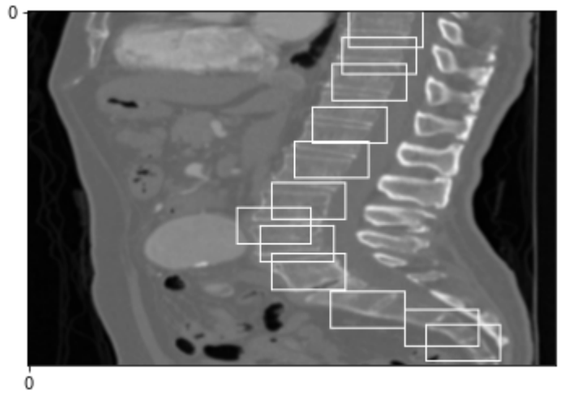
```
def detect_template(img, show=True, x=148,y=147 , template_width=45, template_height= 22):
    # Extract template from image
    masked_img = img.copy()
    template = masked_img[y:y+template_height, x:x+template_width]

    # Perform template matching
    result = match_template(masked_img, template)

    # Find location of best match
    y, x = np.unravel_index(np.argmax(result), result.shape)

    if show:
        plt.figure(figsize=(10, 10))
        plt.subplot(121)
        plt.imshow(masked_img, cmap="gray")
        rect = plt.Rectangle((x, y), template_width, template_height, edgecolor='r', facecolor='none')
        plt.gca().add_patch(rect)
        plt.title("Original Image")
        plt.subplot(122)
        plt.imshow(template, cmap="gray")
        plt.title("Selected template")
        plt.show()
    return x, y, template, result

detect_template(masked_img);
```



In [ ]:

```
def draw_bbox(masked_img, overlay_img, threshold_abs=0.33, slice_num=220, return_mask=False, nms_threshold=0.5):
    x, y, template, result = detect_template(masked_img, show= False)

    if return_mask:
        img = np.zeros(overlay_img.shape)
    else:
        img = 255*((overlay_img - overlay_img.min())/(overlay_img.max()-overlay_img.min()))

    # Find local maxima in the result image
    bbox_list = []
    for bbox_y, bbox_x in peak_local_max(result, threshold_abs=threshold_abs, exclude_border=False):
        bbox = [bbox_x, bbox_y, bbox_x + template.shape[1], bbox_y + template.shape[0], result[bbox_y, bbox_x]]
        bbox_list.append(bbox)

    # Apply non-maximum suppression
    bbox_list = np.array(bbox_list)
    if bbox_list.shape[0] > 0:
        bbox_scores = bbox_list[:, 4]
        bbox_coords = bbox_list[:, :4]
        sorted_idxs = np.argsort(bbox_scores)[::-1]

        keep_idxs = []
        while sorted_idxs.size > 0:
            idx = sorted_idxs[0]
            keep_idxs.append(idx)
            if sorted_idxs.size == 1:
                break
            overlap_scores = iou(bbox_coords[idx], bbox_coords[sorted_idxs[1:]])
            sorted_idxs = sorted_idxs[1:][overlap_scores <= nms_threshold]
        # Draw the final bounding boxes
        for idx in keep_idxs:
            bbox = bbox_coords[idx]
            if return_mask:
                img = cv.rectangle(img, (int(bbox[0]), int(bbox[1])), (int(bbox[2]), int(bbox[3])), (255, 255, 255), -1)
            else:
                img = cv.rectangle(img, (int(bbox[0]), int(bbox[1])), (int(bbox[2]), int(bbox[3])), (255, 255, 255), 1)


    return img

def iou(bbox1, bbox2):
    # Calculate intersection area
    x1 = np.maximum(bbox1[0], bbox2[:, 0])
    y1 = np.maximum(bbox1[1], bbox2[:, 1])
    x2 = np.minimum(bbox1[2], bbox2[:, 2])
    y2 = np.minimum(bbox1[3], bbox2[:, 3])
    intersection_area = np.maximum(0, x2 - x1) * np.maximum(0, y2 - y1)

    # Calculate union area
    bbox1_area = (bbox1[2] - bbox1[0]) * (bbox1[3] - bbox1[1])
    bbox2_area = (bbox2[:, 2] - bbox2[:, 0]) * (bbox2[:, 3] - bbox2[:, 1])
    union_area = bbox1_area + bbox2_area - intersection_area

    # Calculate IOU score
    iou_score = intersection_area / union_area
    return iou_score

img = draw_bbox(masked_img, mid_sag_slice, threshold_abs=0.32, return_mask=False, nms_threshold=0.3)
plt.imshow(img,cmap="gray")
plt.xticks([0])
plt.yticks([0])
```

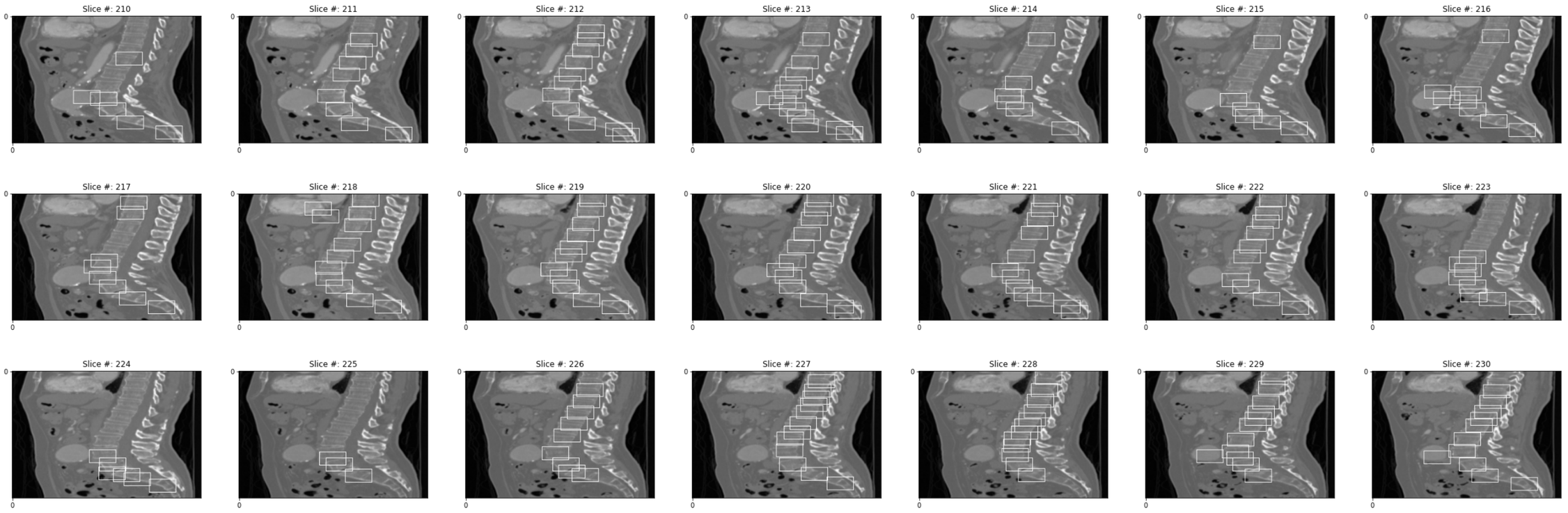
Out[ ]: ([<matplotlib.axis.YTick at 0x7f7549b50430>], [Text(0, 0, '0')])



Apply the same method on 10 sagittal slices before and after the mid-sagittal slice (21 slices in total). Generate a 3D box around each vertebra and report a 3D screen shot of the localization.

In [ ]:
```
bboxs = []
_, nrrd_img = get_mid_slice(NRRD_DIR, False)
for i in range(-10, 11):
    new_slice = nrrd_img[:,:,220+i]
    my_masked_img = get_masked_image(new_slice, show=False)
    bboxs.append(draw_bbox(my_masked_img, new_slice, threshold_abs=0.33, return_mask=False, nms_threshold=0.3))
bboxs = np.array(bboxs)
```

In [ ]:
```
plt.figure(figsize=(42, 14))
for i in range(bboxs.shape[0]):
    plt.subplot(3,7,i+1)
    plt.imshow(bboxs[i,:,:], cmap="gray")
    plt.title(f"Slice #: {210+i}")
    plt.xticks([0])
    plt.yticks([0])
plt.show()
```



Save 3d Images

In [ ]:

```
masks = []
_, nrrd_img = get_mid_slice(NRRD_DIR, False)
for i in range(-10, 11):
    new_slice = nrrd_img[:,:,220+i]
    my_masked_img = get_masked_image(new_slice, show=False)
    masks.append(draw_bbox(my_masked_img, new_slice, threshold_abs=0.33, return_mask=True, nms_threshold=0.3))
masks = np.array(masks)
```

In [ ]:

```
nrrd_img.shape, masks.swapaxes(0,1).swapaxes(2,1).shape
nrrd.write("ct.nrrd",nrrd_img)
nrrd.write("mask.nrrd",masks.swapaxes(0,1).swapaxes(2,1))
```