HC701 Assignment-II

Anees Ur Rehman Hashmi

March 9, 2023

1 Task 1: Windowing and detection in a CT scan

1.1 Task 1.1

The CTA-Abdomen.nrrd file was loaded using the *nrrd* library in Python. Then *Slicer* was used to find the optimal pixel intensity window for visualizing the spine vertebrae. The optimal window is approximately 90 to 200, however, mere thresholding can not separate the vertebrae from other organs visible in the abdomen (see figure 1).

Figure 1: Image after intensity thresholding to separate Vertebrae

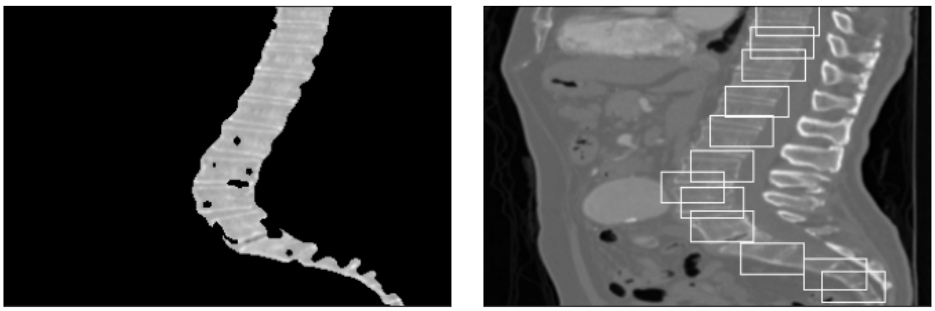
1.2 Task 1.2

This task is for locating the vertebrae in the mid-saggital slice using any non-ML method. Figure 2(a) shows the overlayed mask for mid-saggital slice. Following steps were taken to obtain an optimal mask for the vertebrae:

- Firstly, median blur with filter size 5X5 was used to blur the original mid-saggital slice. This helped in reducing the sharp changes in the pixel intensities, which proved to be useful in the following steps.
- Secondly, two different masks for the vertebrae were created with different thresholding. These masks will be subtracted (pixel-wise) in order to remove the common objects other than the vertebrae. Figure 3 shows the two different created masks and their difference.
- Next step was to remove the smaller objects around the vertebrae. Here, *area opening* technique was used to remove the objects having area less than a specific threshold. *skimage* library was used to remove objects smaller than 500 $pixel^2$. This results in a clean mask for the vertebrae which is overlayed on the original slice (shown in figure 2(a)).

The next task was to put bounding boxes around the vertebrae using any non-ML method. One of the suitable method is to do template based object detection/matching. The detail of the task is as following:

- Firstly, a suitable template of size 22X45 is chosen and used as reference to find other similar objects.



(a) Separated vertebrae mask overlayed on the mid-saggital slice

(b) Mid-saggital slice with bbox

Figure 2: Segmentation and object detection

- The selected template is matched with each part of the image as a sliding window (similar to convolution) to calculate a similarity score. Subsequently, Bounding boxes are displayed around all the regions having similarity scores above the selected threshold.
- Another important step in order to get a cleaner object detection, without many overlapping boxes is to put apply non-maximum-suppression (NMS). It removes the overlapping bounding-boxes by keeping only the one with highest confidence/score. This step removes the spurious objects from the mid-saggital slice (figure 2(b)).
- A total of 12 vertebrae were detected in the mid-saggital slice after applying NMN with threshold = 0.3.

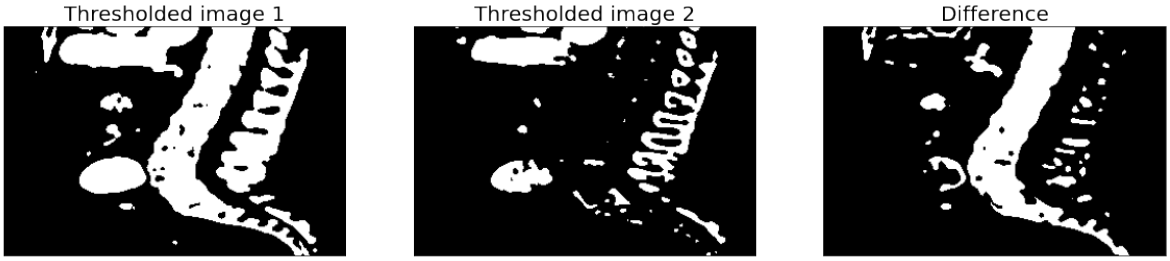


Figure 3: Masks created with different thresholds and their difference

2 Task 2: Classification of Tuberculosis in X-ray

2.0.1 3D boxes

In order to display 3D boxes around the CT image, masks were concatenated and stored as a *nrrd* file. After this both CT and generated masks were visualized in the *Slicer*, screenshots of the overlap are shown in figure 4. Furthermore, the attached Jupyter-notebook contains the 2D bounding boxes generated for each slice.

2.1 Task 2.1

Create a CSV file for this. Report the number and the range of filenames for each class in training and testing sets

In this task, two classes (tb and health) of the given dataset are used for classification. Firstly, a list of all the images in the respective class directory are sorted and divided into testing and training set. There were 800 images for class *tb* and 3800 images for class *health*. First 20% of the images were

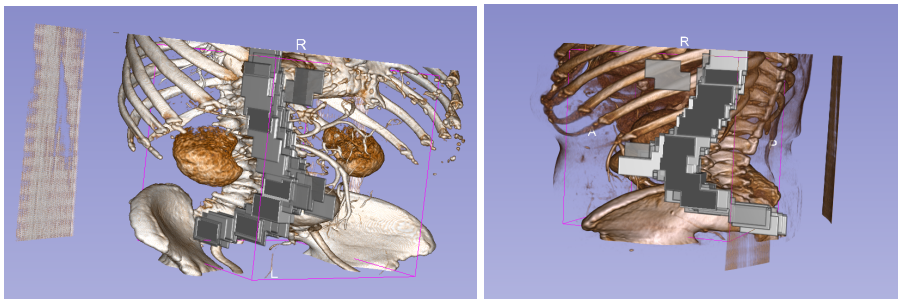


Figure 4: 3D rendering of the generated masks over the CT in Slicer

class_name	dataset	image_count	first_img	last_img
tb	test	160	tb0003.png	tb0248.png
	train	640	tb0250.png	tb1199.png
health	test	760	h0001.png	h0993.png
	train	3040	h0995.png	h5000.png

Table 1: Details of the created CSV files

used for testing and the remaining 80% images were used for training. The total counts and range of selected images per class is given in the Table 1.

2.2 Task 2.2

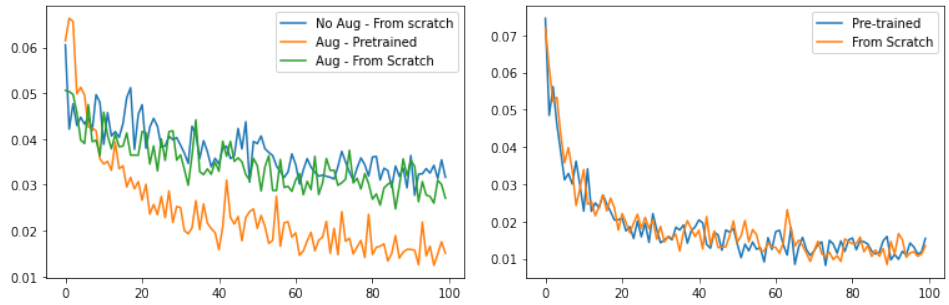
In this task, 4 deep learning models were used to classify the given two classes. The choice of the models and the augmentation strategy was based on the specific benefits of a network architecture and the data modality of the dataset, respectively. Table 2 shows an overview of the models and the hyper-parameters used used in each of the experiments. Please note that the code for this task is available on this github repository and is not submitted because of more than one files for the task.

The experiments are designed to see the affect of using data augmentation and transfer learning using two classification models. All experiments were conducted using Stochastic Gradient Descent (SGD) optimizer for 100 epochs with learning rate 0.001 and momentum 0.9. Table 2 contains the weighted-F1 score for all five experiments and figure 6 shows the confusion matrices for all five experiments.

- The first experiment is carried out without using any pre-trained weights or augmentations for the

S #	Backbone	Pre-Trained	Data Augmentation	Epochs	LR	F1-Score
1	Resnet18 [Gre93]	NA	NA	100	0.001	0.9217
2	Resnet18 [Gre93]	NA	Random Horizontal Flip Random Rotation Random Resized Crop	100	0.001	0.926
3	Resnet18 [Gre93]	ImageNet	Random Horizontal Flip Random Rotation Random Resized Crop	100	0.001	0.961
4	MobileNet-V3 [HSC+19]	NA	Random Horizontal Flip Random Rotation Random Resized Crop	100	0.001	0.977
5	MobileNet-V3 [HSC+19]	ImageNet	Random Horizontal Flip Random Rotation Random Resized Crop	100	0.001	0.976

Table 2: Details of the Experiments and hyper-parameters



(a) Training loss of Resnet-18 models (experiment 1-3)

(b) Training loss of MobileNet-V3 models (experiment 4-5)

Figure 5: Training loss for all 5 experiments.

	Model	FLOPs	Params
0	resnet18_aug_scratch	9.51334G	11.1775M
1	resnet18_aug	9.51334G	11.1775M
2	resnet18_no_aug_scratch	9.51334G	11.1775M
3	mobilenet_aug_scratch	1.18266G	4.20459M
4	mobilenet_aug	1.18266G	4.20459M

Table 3: Model Params and FLOPs

ubiquitous Resnet-18 [Gre93] model, which contains skip connections that allow deeper networks and hence often good classification performance. This is the a vanilla experiment without any data augmentations or transfer-learning. This can be considered as a baseline experiment which is 93% accuracy and 0.92 weighted F1-score. One important thing to note is that the classes are highly imbalanced, and hence classification models are prone to over-fit the majority class, which is visible in the confusion matrix 2.2.

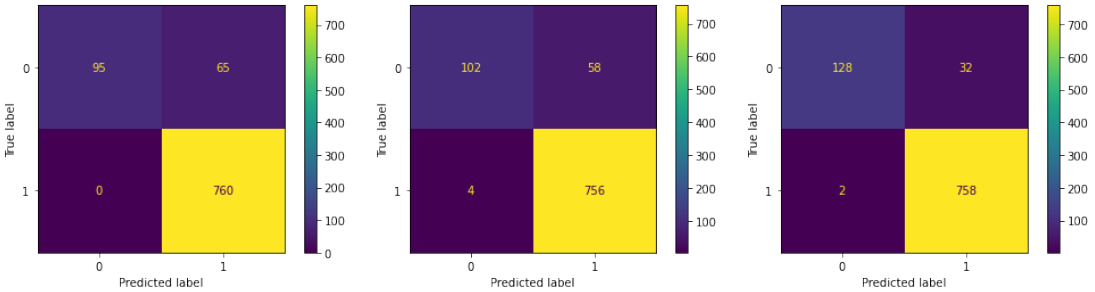
- In the second experiment Resnet-18 is trained from scratch again but this time with data augmentation. The feasible data augmentations for X-ray images were used namely; random-horizontal-flip, random-rotation up-to 10 degrees and random-resized-cropping. This experiment is conducted to see the effect of adding augmentations. The model showed around 1% rise in F1-score on the test-set.

- Experiment 3 is performed using an ImageNet pre-trained Resnet-18 backbone and data augmentation. Fine-tuning the last layer of the model increased the F1-score on test set by 4%. This suggests that having a good initialization, even if it's from a different domain, helps the model to learn better/richer feature representation.

- In the fourth experiment MobileNet-V3 [HSC+19] is trained from scratch alongside with data augmentations. The rationale for choosing a MobileNet model is the good performance to computation ratio they offer with the depth-wise-separable convolution operation. MobileNet-V3 is expected to learn complex representations of the data without very high computational difference than Resnet18 model. This experiment resulted in around 1.6% increase in the performance than the resnet-18 in experiment 3. Table 3 shows that despite having more than 5X less FLOPS and less than half number parameters, MobileNet-V3 is able to achieve better performance than resnet-18.

- In the fifth and the last experiment an ImageNet pre-trained MobileNet-V3 [HSC+19] is fine-tuned alongside with data augmentations. This experiment resulted in 0.97 F1-score and 97% accuracy, which makes it the second best model. This suggests that, even though using training a model from scratch can help achieve very good performance (in some settings) but using transfer learning can still give competitive results, which can be improved by fine-tuning the whole model

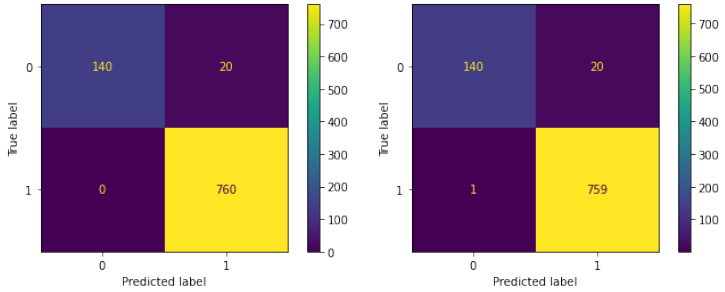
instead of just the last layer. However, this hypothesis can not be verified due to constraint of a total of 5 experiments.



(a) Resnet-18 trained from scratch without data augmentation

(b) Resnet-18 trained from scratch without augmentation

(c) Resnet-18 fine-tuned with data augmentation



(d) MobileNet-V3 trained from scratch with data augmentation

(e) MobileNet-V3 fine-tunned with data augmentation

Figure 6: Confusion matrices for all 5 experiments.

References

[Gre93] George D. Greenwade. The Comprehensive Tex Archive Network (CTAN). *TUGBoat*, 14(3):342–351, 1993.

[HSC+19] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1314–1324, 2019.

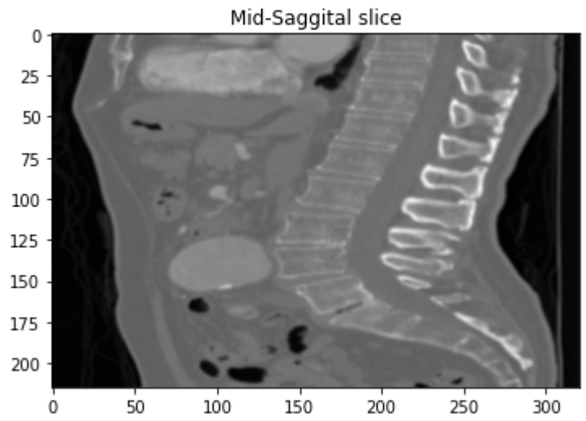
Task 1: Windowing and detection in a CT scan

- 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

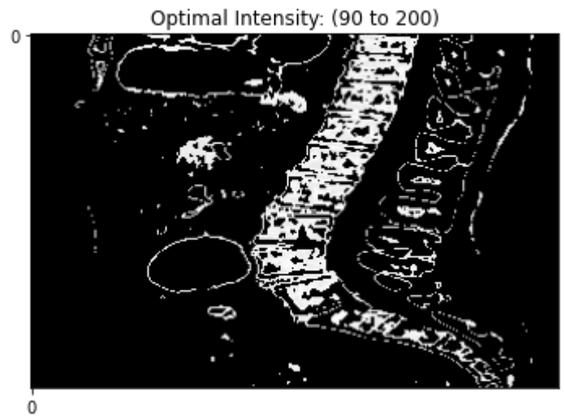
In [ ]:
```
# Importing Necessary Libraries
import numpy as np
import nrrd
import matplotlib.pyplot as plt
import cv2 as cv
from skimage.feature import match_template, peak_local_max
import skimage
```

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()
```

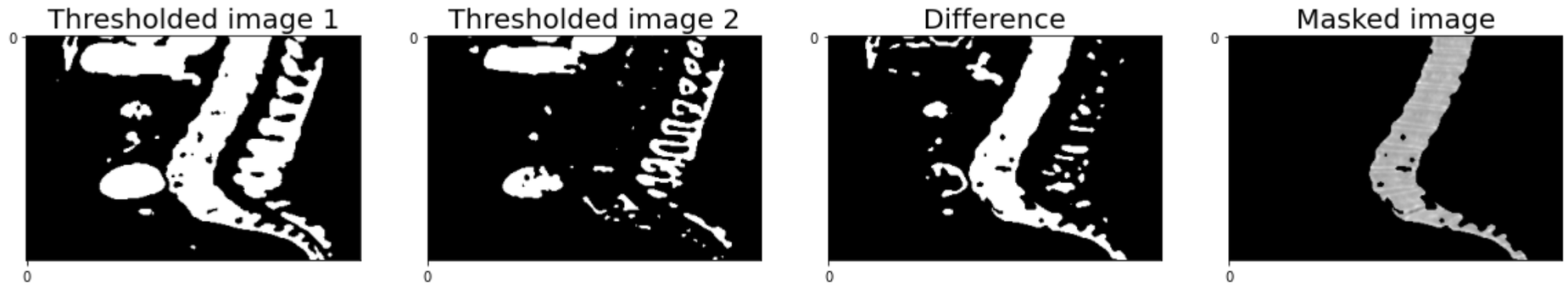


Task 1.2. Following on from Task 1.1, regardless of the intensity window you used:

- 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)
```

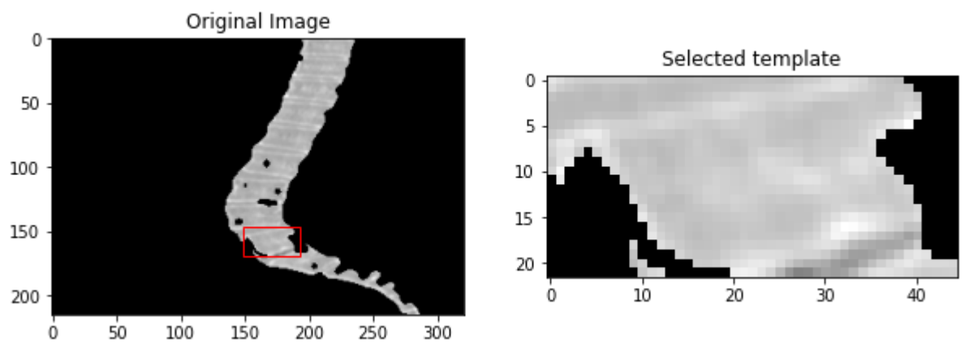


Develop a non-ML method to put a box around each vertebra and report a screen shot of the boxes.

Template Matching

In [ ]:

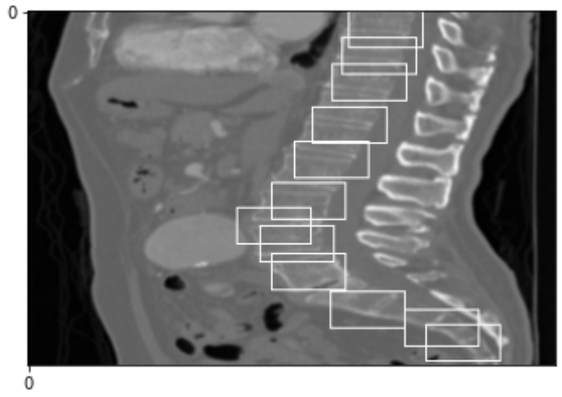
```
def detect_template(img, show=True, x=148,y=147 , template_width=45, template_height= 22):
    # Extract template from image
    masked_img = img.copy()
    template = masked_img[y:y+template_height, x:x+template_width]

    # 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])
```

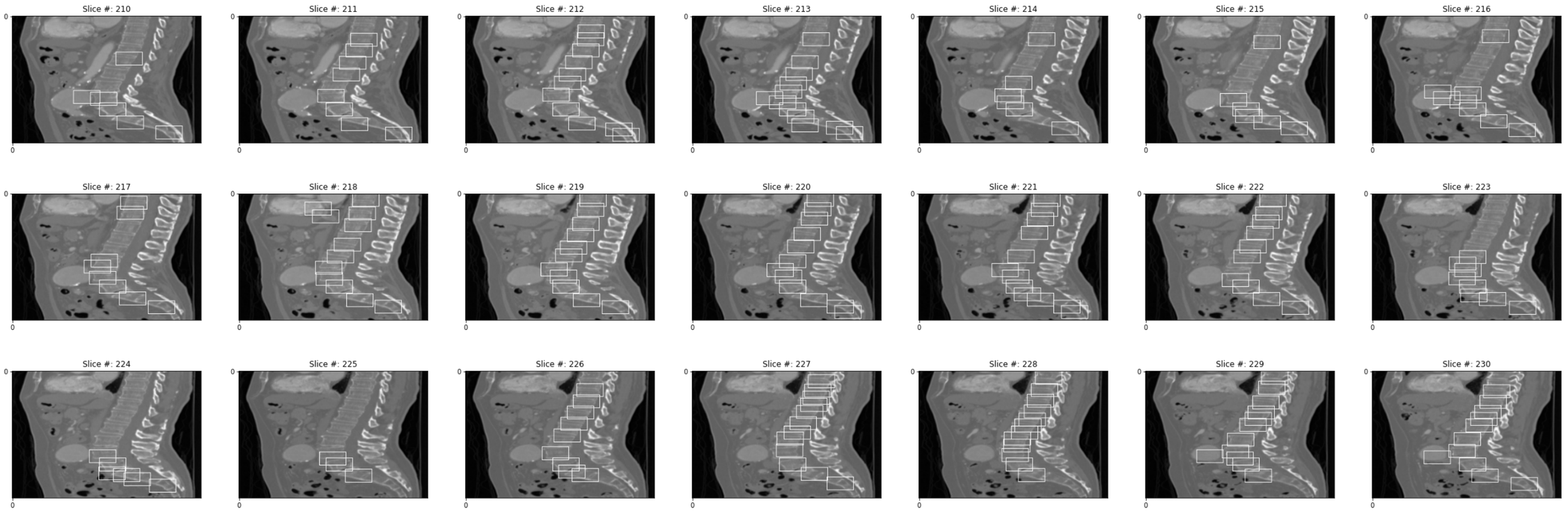
Out[ ]: ([<matplotlib.axis.YTick at 0x7f7549b50430>], [Text(0, 0, '0')])



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))
```