CompSci-302 Final Project

Musical Note Detection

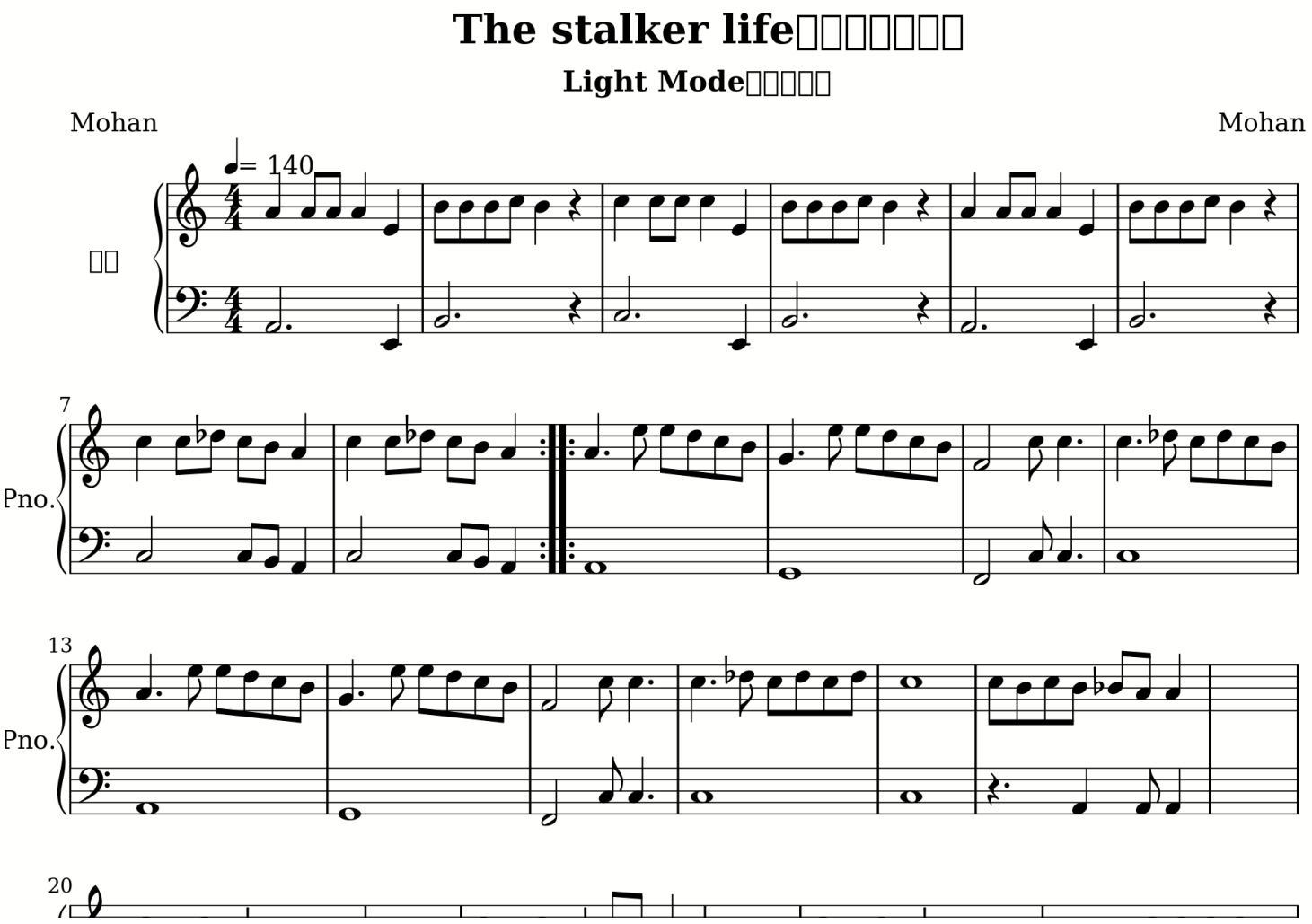
By Yueqian Lin

Introduction

I am a fan of singing

Introduction

But I am not good at reading the music sheet



Introduction

Wait! I am also a fan of Computer Vision

Introduction

Wait! I am also a fan of Computer Vision

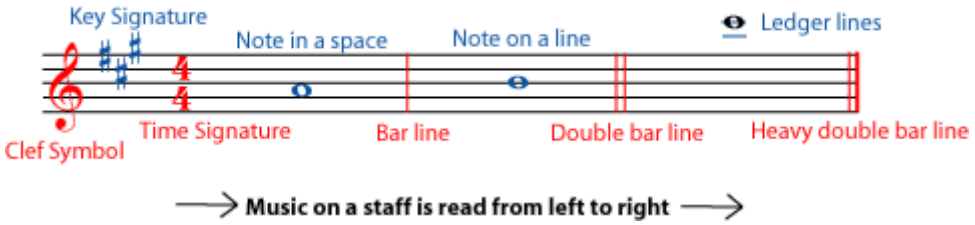
We can apply what we learned in class (the traditional CV techniques) to solve this problem

Background Knowledge

Basic Music Theory

- The Staff: 5 lines and 4 spaces
- The Clef: the treble clef (G clef) and the bass clef (F clef)
- The Notes: the pitch and the duration
- Time Signature: the number of beats per measure and the note value that represents one beat
- Key Signature: the sharps or flats placed at the beginning of a staff

Example


Key Signature
Note in a space
Note on a line
Ledger lines
Clef Symbol
Time Signature
Bar line
Double bar line
Heavy double bar line
Music on a staff is read from left to right

Thinking

- So many parts of the music sheet
- How to detect the staff?
- How to detect the clef?
- How to detect the notes?
- ……?

Goal

- Detect the pitch
- If possible, detect the duration
- If possible, detect the time signature, or other information

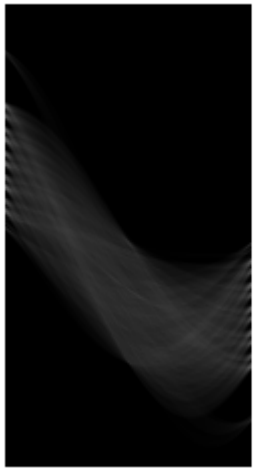
Staff Detection

- The staff is a set of parallel lines
- We learn the Hough Transform in class
    - Hough Transform can transform the lines in the image to the parameter space
- We can use Hough Transform to detect the staff

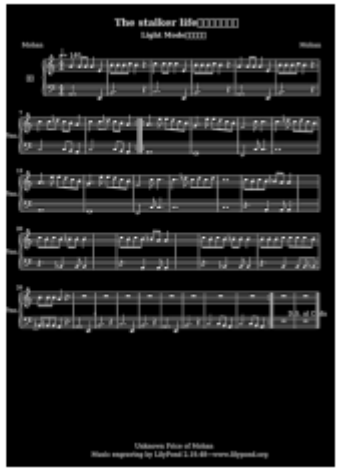
In [ ]:

```
from skimage.transform import hough_line, hough_line_peaks

# create a vector of angles with steps of of 0.5 degree.
tested_angles = np.linspace(-np.pi/2, np.pi/2, 360, endpoint=False)

# Classic straight-line Hough transform
h, theta, r = hough_line(binary_image, theta=tested_angles)
# theta and r are vectors containing the Hough space bin coordinates
# plotting the hough space
plt.figure(figsize=(14, 7))

# create correct axis labels
angle_step = 0.5 * np.diff(theta).mean()
r_step = 0.5 * np.diff(r).mean()
bounds = [np.rad2deg(theta[0] - angle_step),
          np.rad2deg(theta[-1] + angle_step),
          r[-1] + r_step, r[0] - r_step]
```

Hough space



Original image



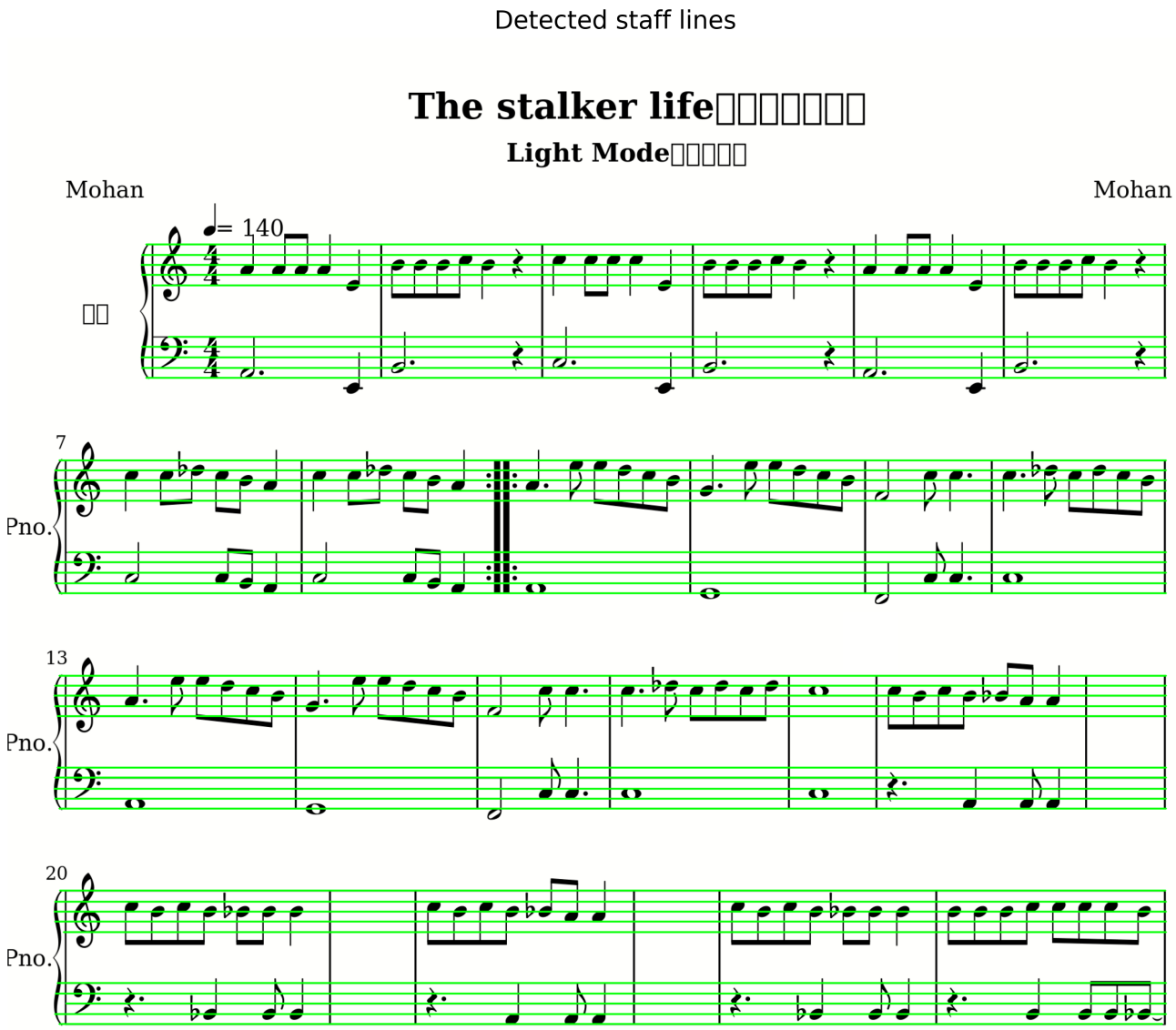
Staff Detection

- From the image to the parameter space we see the peaks
- We should be able to detect the staff by finding the peaks

Algorithm

- Set the minimum line length as 0.75 times the width of the image.
- Apply the Hough Line Transform on a binary image, detecting lines with a certain threshold.
- Filter out non-horizontal lines and keep only the horizontal lines.
- Sort the horizontal lines based on their vertical position.
- Calculate the differences between consecutive line positions.
- Determine the minimum distance between consecutive lines.
- Combine lines that are close to each other by averaging their coordinates if their distance is less than `y_diff_min` multiplied by a threshold.

Preview of Staff Detection



Next Step

- Choice 1: Detect the notes with the staff
- Choice 2: Detect the notes without the staff

Next Step

- Choice 1: Detect the notes with the staff

- Choice 2: Detect the notes without the staff

My Choice: Detect the notes without the staff

- The staff is not always in the image
- The staff is not always horizontal
- The staff is not always straight

Remove the Staff

- Initial thought: We know the position of the staff, we can remove it

- But we combine some of the lines to get the staff, we don't know the epsilon value between the line positions

Remove the Staff

- Initial thought: We know the position of the staff, we can remove it

- But we combine some of the lines to get the staff, we don't know the epsilon value between the line positions

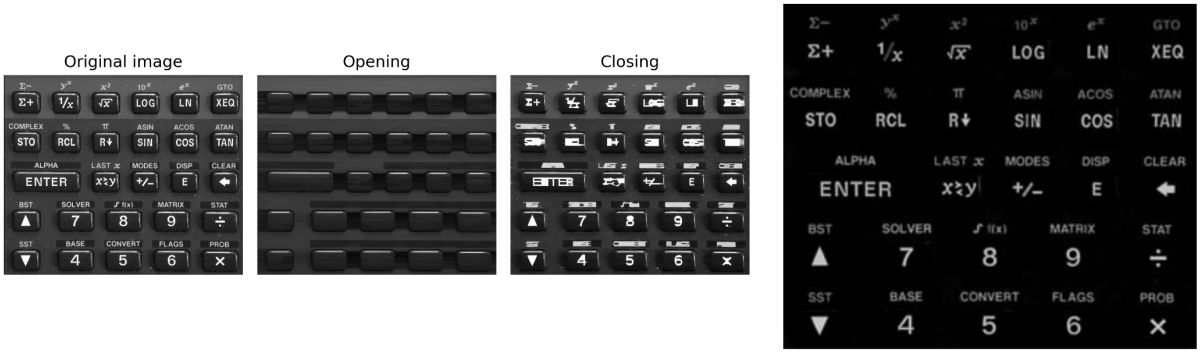
We also learn *Morphology* and I presented the Morphology of Grayscale Images in class

Remove the Staff using Morphology

Recall the example I showed in the research presentation

Example of Opening and Closing

- Opening to suppressing the horizontal reflection on the top of the keys
- Closing to smoothing out the boundaries of bright features in the image



Algorithm

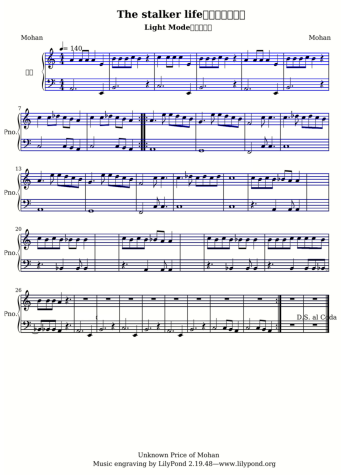
- Define a horizontal kernel (hor_kernel) for morphology operations.
- Use morphological opening on the binary image using the defined kernel to detect and emphasize horizontal lines (`cv2.morphologyEx() function`).
- Find contours in the detected lines using the `cv2.findContours()` function.
- Iterate over each contour and draw it on the original image (img) using `cv2.drawContours()` .
- Define a repair kernel (`rep_kernel`) for image repair operations.
- Perform image repair by first inverting the image colors `(255 - img)` , applying morphological closing using the repair kernel, and then inverting the result back to obtain the final repaired image (`cv2.morphologyEx()` function).

In [ ]:

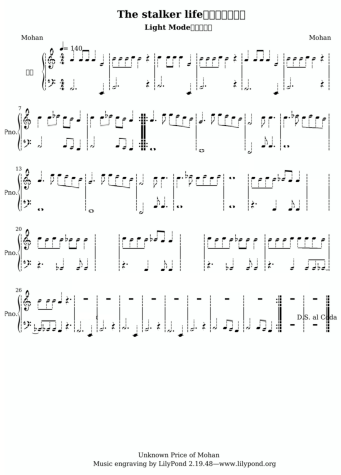
```
# horizontal kernel
hor_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (25,1))
detected_lines = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, hor_kernel, i
# find contours
cnts = cv2.findContours(detected_lines, cv2.RETR_EXTERNAL, cv2.CHAIN_AP
cnts = cnts[0] if len(cnts) == 2 else cnts[1]
for c in cnts:
    cv2.drawContours(img, [c], -1, (255,255,255), 2)
# repair image
rep_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (1,6))
result = 255 - cv2.morphologyEx(255 - img, cv2.MORPH_CLOSE, rep_kernel,
```

Illustration of Morphology

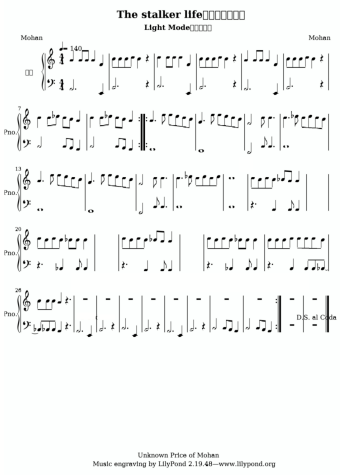
Detected Lines



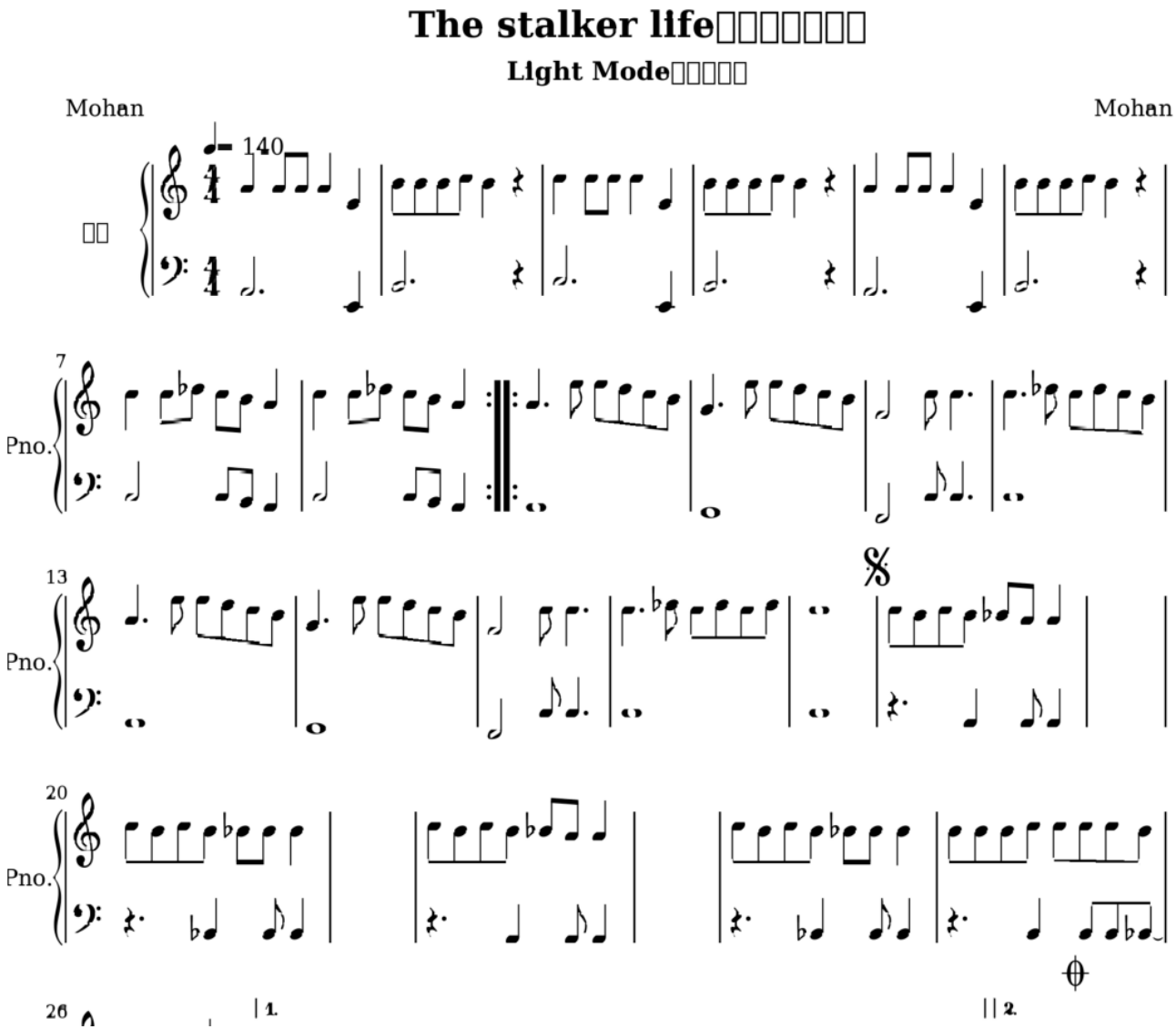
After Removing Horizontal Lines



After Repair



Preview of Staff Removal



Next Step

- I have the notes without the staff, and I know the position of the staff

- I can detect the note positions and infer the pitch according to the position

Next Step

- I have the notes without the staff, and I know the position of the staff

- I can detect the note positions and infer the pitch according to the position

- Before that: I choose to segment the image into parts according to the staff for standardization

Example of segmentation

A Long Journey

- First, I tried to detect the notes straightly using `cv2.templateMatch()` function

- But it has several problems:

  - The notes are not always in the same size as the template
  - Many irrelevant matches
  - Time consuming

A Long Journey

How to solve these problems?

- The notes are not always in the same size as the template

  - Match the notes with the template with scaling

Algorithm

- Iterates over a range of scales and resizes each template accordingly.
- Template matching is applied using the resized template and the input image.
- The locations where the matching score exceeds a given threshold are identified.
- The scale that produces the highest number of matching locations is considered the best scale.
- The best matching locations and the corresponding best scale are returned as the output of the function.

Algorithm

- Iterates over a range of scales and resizes each template accordingly.
- Template matching is applied using the resized template and the input image.
- The locations where the matching score exceeds a given threshold are identified.
- The scale that produces the highest number of matching locations is considered the best scale.
- The best matching locations and the corresponding best scale are returned as the output of the function.

However, it returns many matches, many are overlapping

A Long Journey

How to solve these problems?

- The notes are not always in the same size as the template

  - Match the notes with the template with scaling
  - Overlapping matches

Algorithm

- What `cv2.templateMatch()` function returns is a list of matches in the form of `(x, y, w, h)`, where `(x, y)` is the top-left corner of the matched region, and `(w, h)` is the width and height of the matched region.
- We use a class `Note` to represent the matched region, and we can use the `Note` class to filter out the overlapping matches, and further use the `Note` class to remember the matched region and the corresponding pitch or other information.
- `merge_notes(notes, threshold)`:
    - Enters a while loop until all notes have been processed.
    - Pops the first Note (`r`) from the list.
    - Sorts the remaining notes based on their distance to `r`.
    - Enters another while loop to merge notes if they overlap or are close enough to each other.
    - If a merge occurs, updates `r` with the merged note and sets merged to True.
    - If no merge occurs, exits the loop.
    - Appends the final merged Note to the filtered_notes list.

A Long Journey

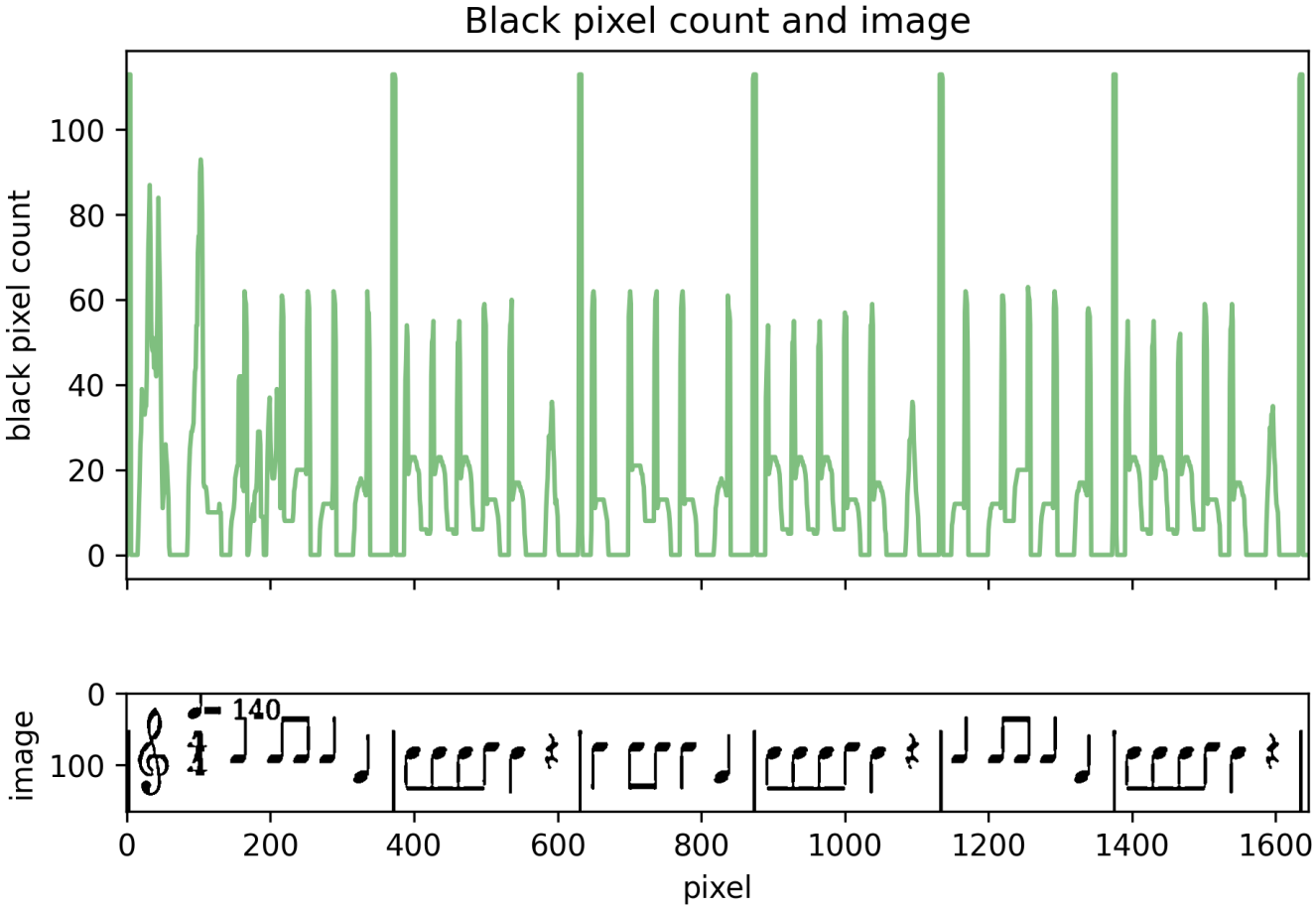
How to solve these problems?

- The notes are not always in the same size as the template
  - Match the notes with the template with scaling
  - Overlapping matches
- Many irrelevant matches
  - Find interesting regions first (recall the region proposal in object detection)

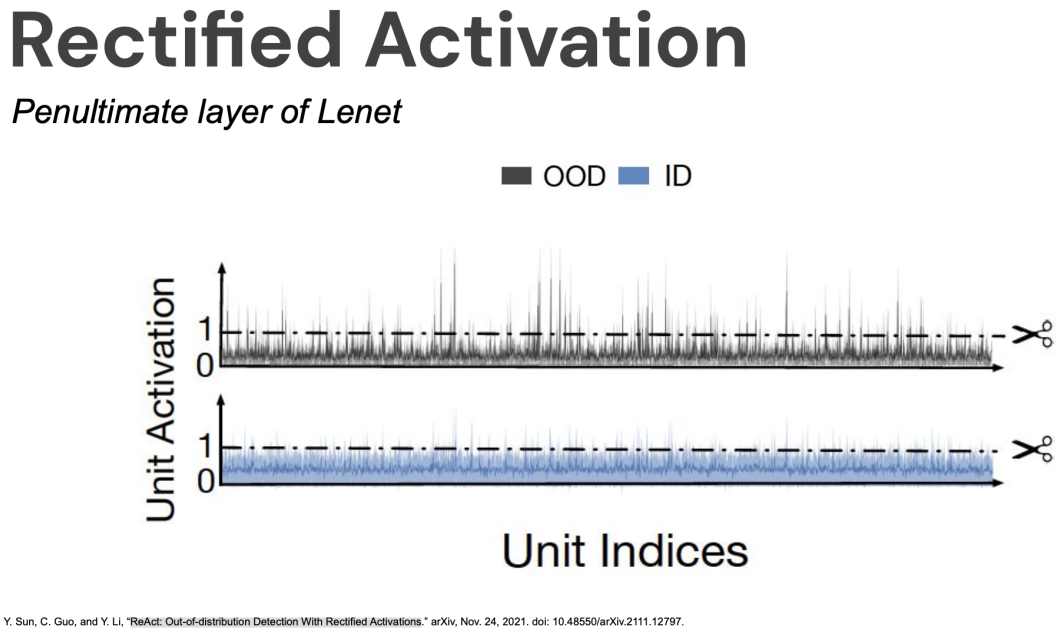
Region Proposal

Many black pixels in the note -> plot the black pixels

Note: As mentioned by Xingyu, if for one position has two notes stacked together, the black pixel count will possibily exceed the threshold. More advanced pattern matching is needed.

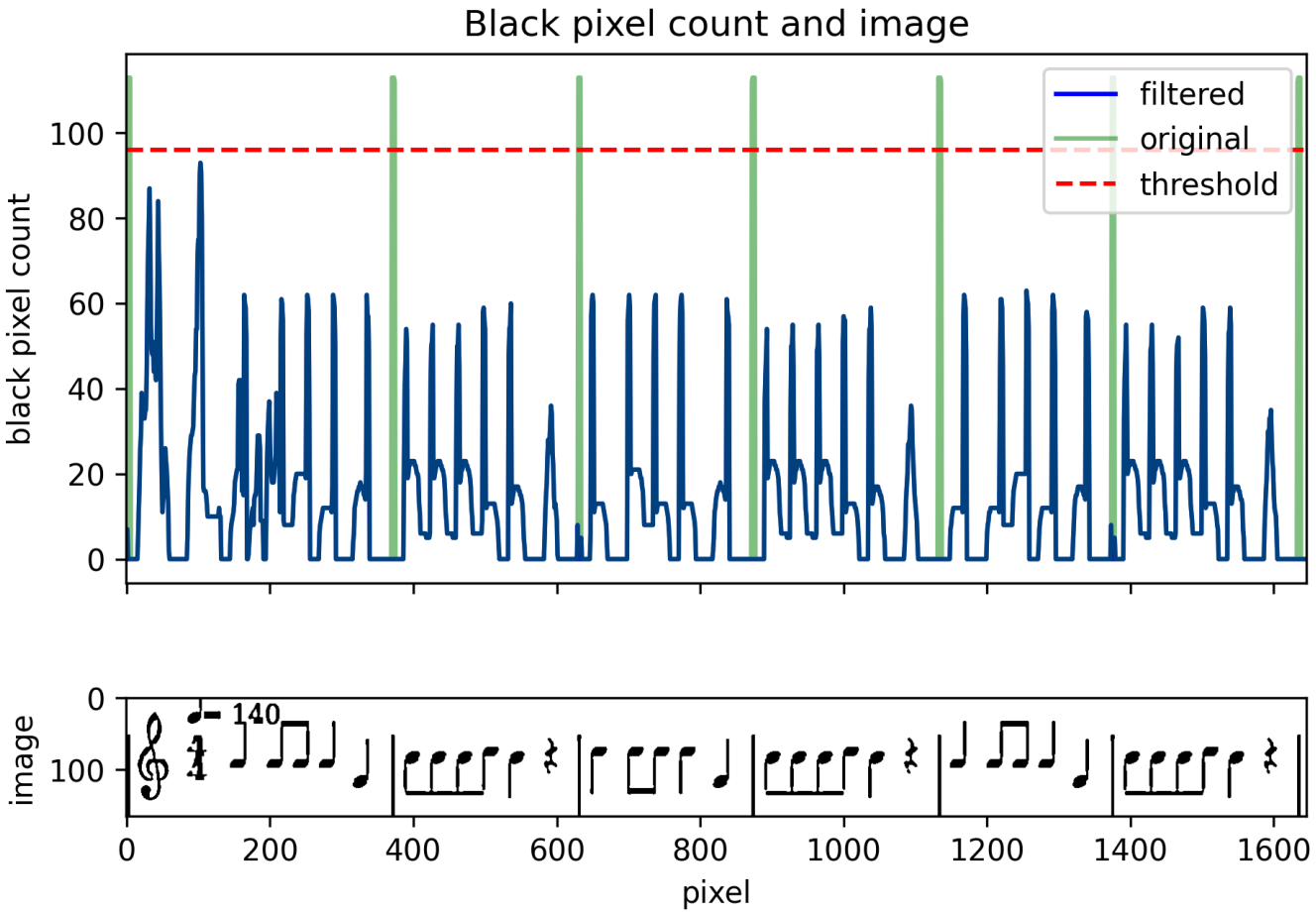

Black pixel count and image
black pixel count
0
20
40
60
80
100
image
0
100
140
0
200
400
600
800
1000
1200
1400
1600
pixel

ReAct: Out-of-distribution Detection With Rectified Activations



Y. Sun, C. Guo, and Y. Li, "ReAct: Out-of-distribution Detection With Rectified Activations." arXiv, Nov. 24, 2021. doi: 10.48550/arXiv.2111.12797.

Clip the Black Pixels



A Long Journey

How to solve these problems?

- The notes are not always in the same size as the template
  - Match the notes with the template with scaling
  - Overlapping matches
- Many irrelevant matches
  - Find interesting regions first (recall the region proposal in object detection)
  - Connect the near region

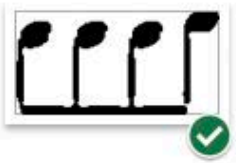
A Long Journey

How to solve these problems?

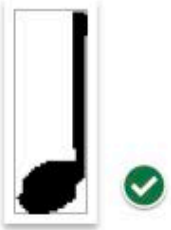
- The notes are not always in the same size as the template
  - Match the notes with the template with scaling
  - Overlapping matches
- Many irrelevant matches
  - Find interesting regions first (recall the region proposal in object detection)
  - Connect the near region

- Now we have interested notes
  - But there are notes with bar

Preview of Note Collection



386.png



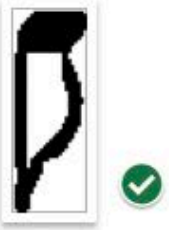
391.png

392.png

399.png

402.png

458.png



466.png

506.png

521.png

532.png

545.png

547.png

A Long Journey

How to solve these problems?

- The notes are not always in the same size as the template

  - Match the notes with the template with scaling
  - Overlapping matches

- Many irrelevant matches

  - Find interesting regions first (recall the region proposal in object detection)
  - Connect the near region

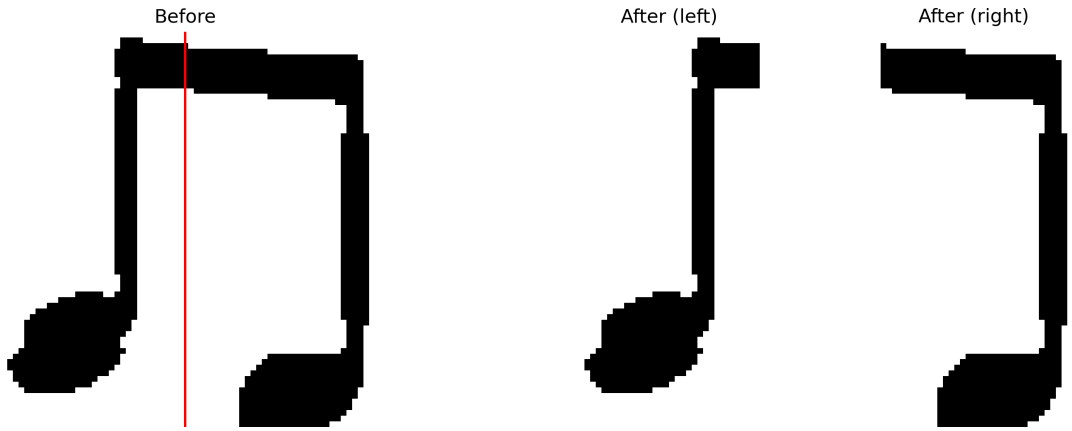
- Now we have interested notes

  - But there are notes with bar
  - We can split them by a unit length

Algorithm

- `get_unit_width(notes)` gets the most common width
- `cut(notes, unit_width):`
    - Gets the width of the current note.
    - Checks the width against a series of conditions to determine the number of cuts to make:
    - If cut is determined, the note is divided into cut parts.
    - For each division, a new Note object is created with adjusted coordinates and dimensions based on the cut.
    - The cut_notes list is updated with the new Note objects.

Preview of Note Splitting



A Long Journey

- Find interesting regions
- Get unit length note
- Detect notehead for pitch

Algorithm

- The notehead within the note image is detected using the `detect_notehead()` function.
- If the notehead is valid, its coordinates are adjusted to be relative to the staff image.
- The function determines the pitch of the note by comparing its center Y-coordinate (`note_center_y`) with the Y-coordinates of the staff lines.
- If the `note_center_y` falls within the staff lines, the pitch is assigned based on the mapping provided by the clef_info dictionary.
- If the `note_center_y` falls above or below the staff lines, the function iterates to find the nearest line above or below the note and adjusts the pitch accordingly.
- The pitch of the note is assigned based on the note index, octave, and note names defined in the `note_names` list.
- If the note is not within the staff or the pitch cannot be determined, `None` is returned.

Algorithm

- The notehead within the note image is detected using the `detect_notehead()` function.
- If the notehead is valid, its coordinates are adjusted to be relative to the staff image.
- The function determines the pitch of the note by comparing its center Y-coordinate (`note_center_y`) with the Y-coordinates of the staff lines.
- If the `note_center_y` falls within the staff lines, the pitch is assigned based on the mapping provided by the clef_info dictionary.
- If the `note_center_y` falls above or below the staff lines, the function iterates to find the nearest line above or below the note and adjusts the pitch accordingly.
- The pitch of the note is assigned based on the note index, octave, and note names defined in the `note_names` list.
- If the note is not within the staff or the pitch cannot be determined, `None` is returned.

A Long Journey

- Find interesting regions
- Get unit length note
- Detect notehead for pitch (and prepare for duration)
- Detect duration accordingly

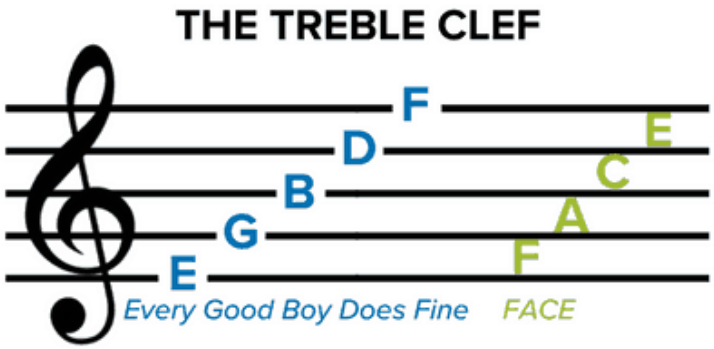
Algorithm

- For each note in the input list `notes`, the function retrieves the duration information stored in the `attr` attribute of the note's `head`.
- Based on the duration text, the function assigns a numerical value to the `dur` variable:
- If the note has been cut (split into multiple parts), the `dur` value is divided by the number of cuts (`cut`) to adjust the duration accordingly.

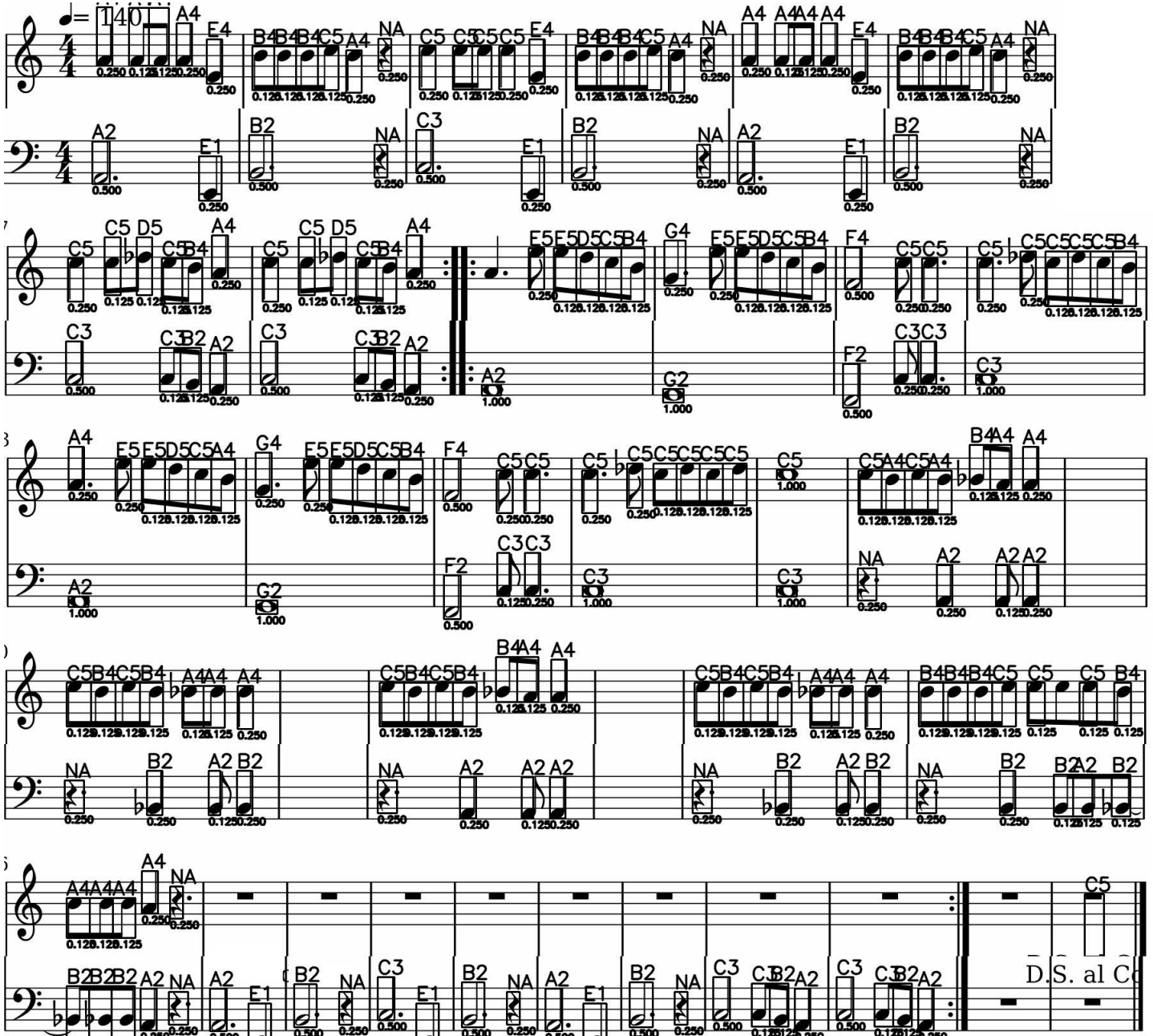
- The `duration` attribute of the note is updated with the formatted `dur` value, rounded to three decimal places.

- Note: Xingyu mentioned afterwards that the duration is calculated wrong because of my bad music theory knowledge. The duration corresponds to the total number of horizontal bar the notehead is linked. For example, a quarter note is on one bar, so its duration is 1/8. A quarter note is on two bars, so its duration is 1/16.

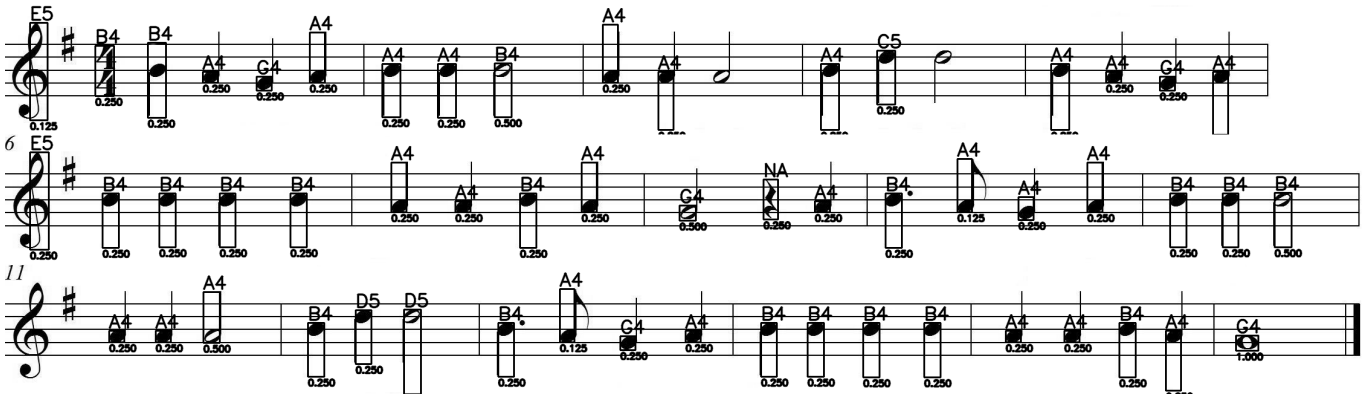
Recall



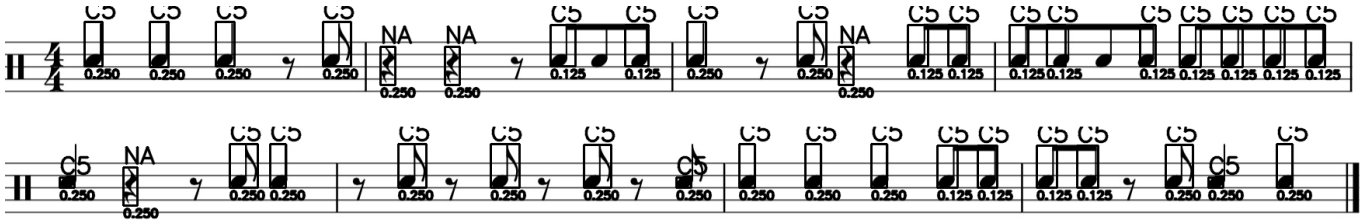
Result TEMP FIXED, still not perfect



More Examples



Mary had a little lamb



Drum

Future Work

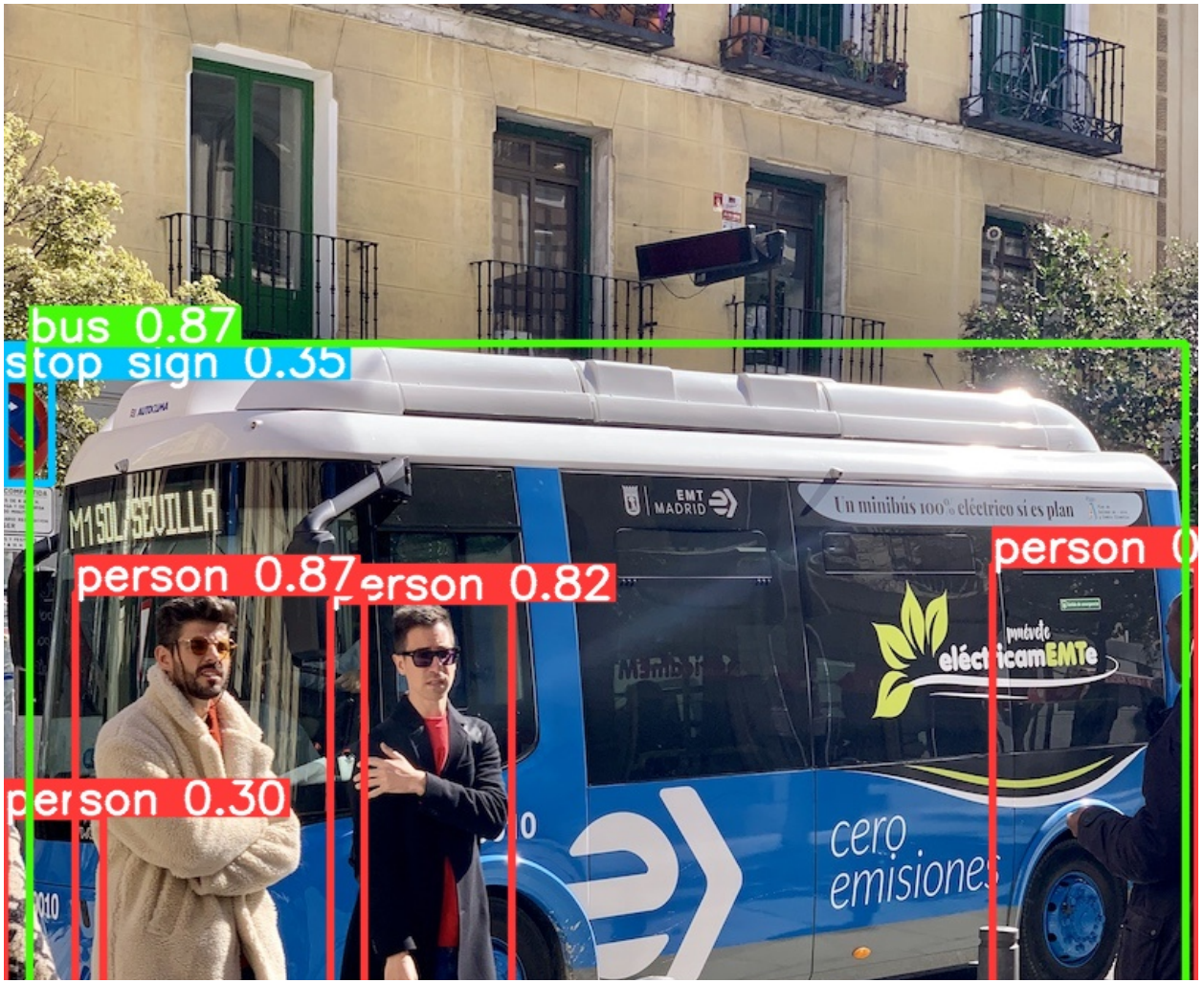
- Increase the robustness
- Increase the ability to detect other information
- Compare results with Deep Learning model
- Used in pipeline of AIGC

Thoughts

- The process of this project is similar to the process of a real project
- ChatGPT is a powerful tool for generating code when you have a clear idea of what you want to do
- Compared with the deep learning model, the traditional method is more flexible and easier to debug
- But the traditional method is not as robust as the deep learning model

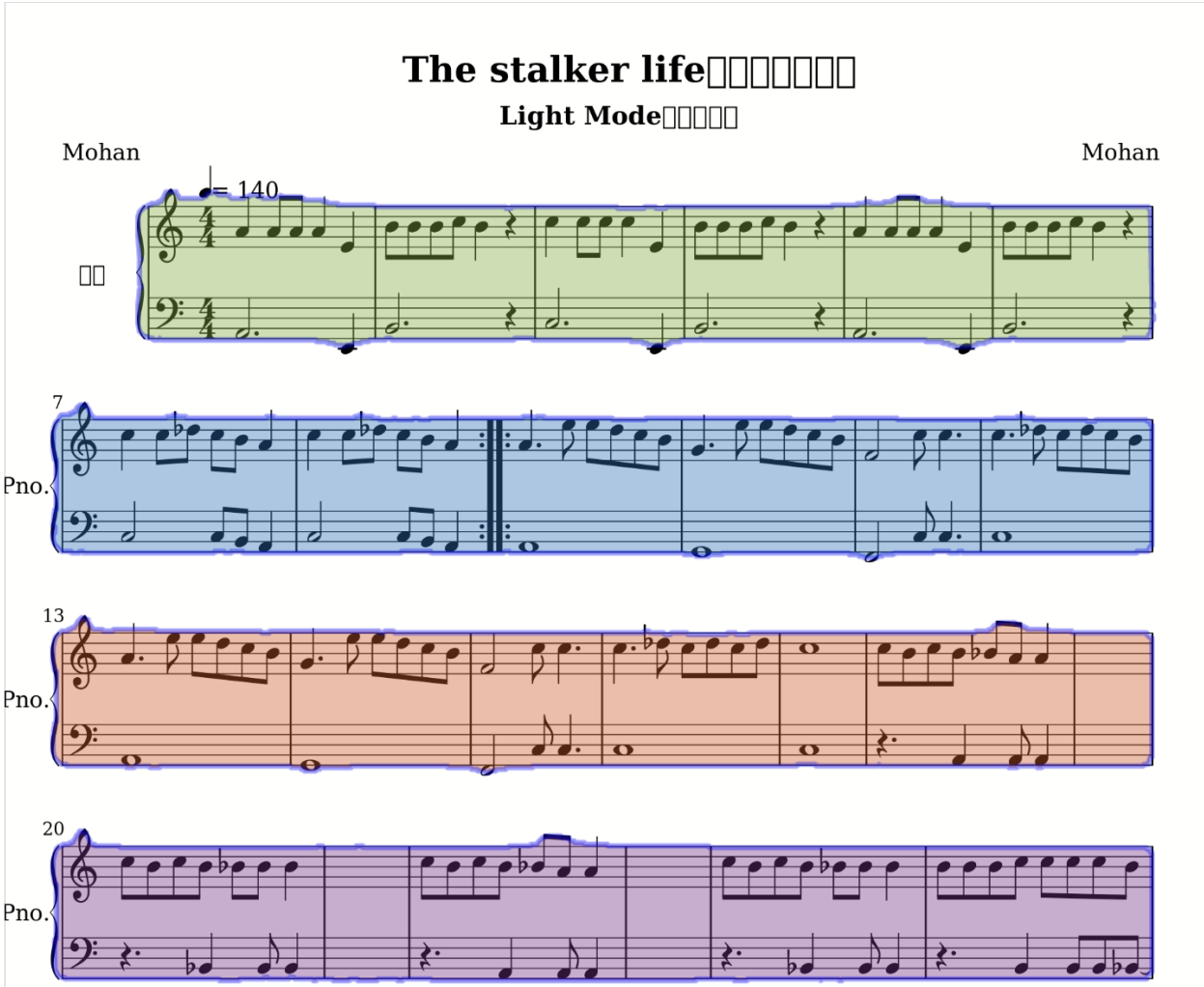
Deep learning model

- Yolo v8: you can `pip install` it and use it directly (may need to train first)
- segment anything + prompt



Deep learning model

- Yolo v8: you can `pip install` it and use it directly (may need to train first)
- segment anything + prompt



References

- ChatGPT is used for general code implementation of template matching, and presentation's algorithm wording.
- A few GitHub repos that inspire me : https://github.com/aashrafh/Mozart, https://github.com/BreezeWhite/oemer, https://github.com/fchandra09/music-sheet-recognition, https://github.com/afikanyati/cadenCV and https://github.com/sc-2020-siit/MusicSheetsRecognition.