Dohyun Kim (Kimmy)

Capstone Project Report

- Topic: OCR for dynamic text in video

1. Introduction

The CLAMS project aims to facilitate access, analysis and interpretation of old multimedia materials including videos and audio recordings for archivists and researchers. CLAMS consists of various apps or tools to produce metadata and other information from the audiovisual content, which also can be used to create customized workflows and pipelines by computer scientists and developers.

As part of the CLAMS project, my topic lies on advancing Optical Character Recognition, particularly for dynamic text within videos. While existing OCR engines demonstrated proficiency in extracting text from static images, extracting the dynamic text in videos have still more challenges. Here, "dynamic text" refers to text elements with dynamic effects such as ending credits that appear at the end of a video, horizontally scrolling chyrons, and text that appears and disappears at various locations on the screen, with content that changes or continues.

In the CLAMS workflow, first, the swt-app of CLAMS is applied to the video document to detect the timepoints (TimePoints) in the video where scenes containing text occur. The swt-app analyzes still images taken at the timepoints sampled at regular time intervals throughout the video and determines what type of text is present at each timepoint. The classification labels for the type of text include 'bars,' 'slate,' 'chyron,' 'credits,' and so on. Scenes that do not contain any text are classified as 'NEG.' The swt-app also groups consecutive TimePoints classified with the same label to define a TimeFrame, where the 'target' of this TimeFrame is each individual TimePoint. The dswt-app is applied to the TimeFrames classified as 'credits' from the swt-app output MMIF. In this project, we assumed that the swt performed all classifications correctly although this is not actually the case. Therefore, we manually edited the 'target' of the TimeFrame Annotations in the swt output MMIF and started from that MMIF file.

For the TimeFrames identified as credits from swt-app, the effort to improve the performance of the OCR module reading dynamic text in the correct order from the Timepoints (scenes) in the TimeFrames involves four main steps.

First, apply OCR to the first 20 timepoints captured at regular short length of intervals for each relevant timeframe. Second, compare the outputs to determine the optimal length of timepoint interval that minimizes text overlaps while ensuring nothing is missed. Third, resample timepoints across the entire timeframe at the optimal length of timepoint interval. Fourth, apply OCR to the resampled timepoints and concatenate the outputs to produce a single TextDocument, considering the positional arrangement of text blocks or columns within each scene.

Here, docTR (https://github.com/mindee/doctr) is utilized for the OCR module and the final ‘dynamic scene with text reader (dswt-reader)’ is wrapped as a CLAMS app. The detailed description for each step is in the following sections.

Dohyun Kim (Kimmy)

2. Data

Here, we use three ending credits videos (video [1], [4] and [5]) from random movies downloaded from YouTube for testing the performance of the app. I have manually annotated the gold transcription. Since time was a bit tight, the initial version of the app was built and tested focusing only on the first of the three main types: vertically rolling credit processing. However, I have logically implemented functions to classify given scenes with dynamic text into one of the three types and to concatenate the OCR output according to the type. Therefore, in the next updated version, this app will be applicable to all three types.

- Example of the dynamic text in video

There are three main types of dynamic text in video as follows:

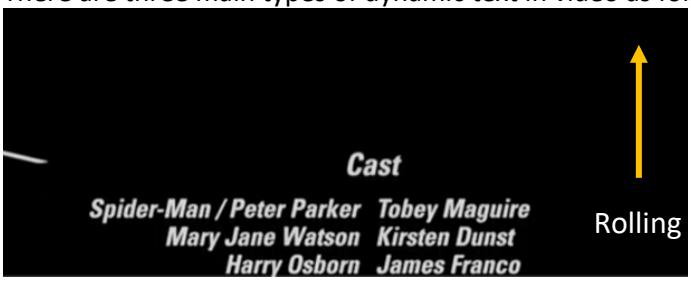

Figure 1 Example of vertically rolling credit (video [1])

Figure 2 Example of fade-in and fade-out text. (video [2])

Figure 3 Example of horizontally scrolling text (chyron) (video [3])

Dohyun Kim (Kimmy)

3. Observation

Types of dynamic text in a video can be classified based on the direction in which the text moves, and the direction of the moving text can be identified by the type of overlapping text in two consecutive timepoint scenes.

The docTR module reads and outputs the text within an image at the word, line, block, and page levels, which correspond to Token, Sentence, Paragraph, and TextDocument in the URIs of annotation types. Compare the docTR output of two consecutive scenes within a timeframe classified as dynamic scenes with text. Depending on whether the overlapping text read by docTR module is at the Sentence level, entire TextDocument level, or character level, the type of dynamic text in that timeframe can be classified as vertically rolling, fade-in and out, or horizontally scrolling.
We assume that the speed at which the text moves is consistent within a single TimeFrame.

1) For vertically rolling credits, overlapping text between two scenes from consecutive timepoints would be at the sentence level as follows:

Figure 4 Example of two consecutive timepoints image of vertically rolling credits (tp_1 and tp_2) (video [1])

From tp_1 above, the docTR module produces *'Cast', 'Spider-Man / Peter Parker Tobey Maguir', 'Mary Jane Watson Kirsten Dunst', 'Harry Osborn James Franco'* as sentence level outputs. From tp_2, the docTR module produces *'Cast', 'Spider-Man / Peter Parker Tobey Maguir', 'Mary Jane Watson Kirsten Dunst', 'Harry Osborn James Franco', 'Doc Ock/Dr. Otto Octavius Alfred Molina', 'May Parker Rosemary Harris'*. As a result, overlapping text between two consecutive timepoints would be *'Cast', 'Spider-Man / Peter Parker Tobey Maguir', 'Mary Jane Watson Kirsten Dunst', 'Harry Osborn James Franco'*, which is at the sentence level.

2) For fade-in and fade-out credits, two scenes from consecutive timepoints would have an overlap at the entire textDocument level if it exists as follows:

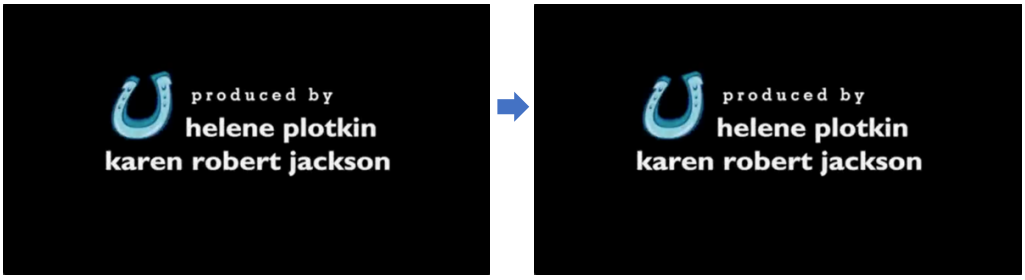

Figure 5 Example of two consecutive timepoints image of fade-in and fade-out credits (tp_1 and tp_2) which have an overlap. (video [2])

Figure 6 Example of two consecutive timepoints image of fade-in and fade-out credits (tp_2 and tp_3) which have no overlap. (video [2])

From tp_1 above, the docTR module produces *'produced by\nhelen plotkin\nkaren robert jackson'* at the page (TextDocument) level. From tp_2, the docTR module produces *'produced by\nhelen plotkin\nkaren robert jackson'*. From tp_3, it is *'executive producer\nsarah mcarthur'*. If there is an overlap as in tp_1 and tp_2, the overlap would be the entire TextDocument. In the case of fade-in and fade-out credits, when comparing the OCR outputs of several consecutive scenes with short time intervals, the entire TextDocument overlaps across the scenes, then transitions to a scene where there is no overlap at all, which is then followed by scenes where all the text overlaps for a while.

3) When comparing two scenes from consecutive timepoints within a timeframe that includes horizontally scrolling texts (or chyron), there is a pair of sentences that overlaps at the character level as follows:

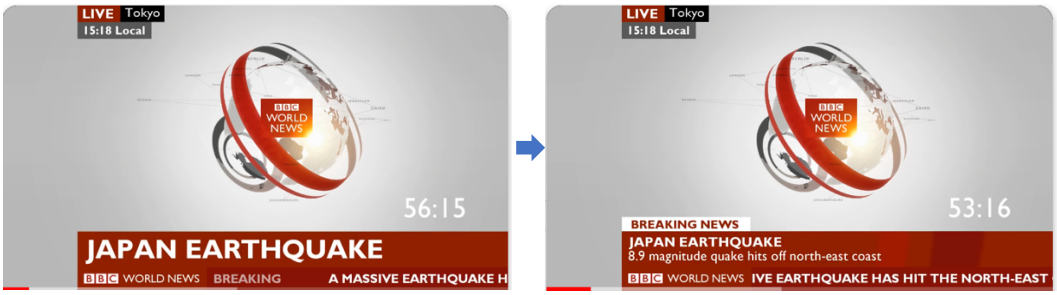

Figure 7 Example of two consecutive timepoints image of horizontally scrolling text (tp_1 and tp_2) (video [3])

From tp_1 above, the docTR module produces *'LIVE Tokyo', '15:18 Local', 'BBC', 'WORLD', 'NEWS', 'JAPAN EARTHQUAKE', 'BBC WORLD NEWS BREAKING A MASSIVE EARTHQUAKE H'* at the sentence level. From tp_2, the last sentence from the sentence level docTR output would be *'BBC WORLD NEWS IVE EARTHQUAKE HAS HIT THE NORTH-EAST'*. The overlap between the last sentences from each scene is at the character level like the part written in orange as follows:

Tp_1: *'BBC WORLD NEWS BREAKING A MASSIVE EARTHQUAKE H'*
TP_2: *'BBC WORLD NEWS IVE EARTHQUAKE HAS HIT THE NORTH-EAST'*

We use edit distance (i.e. normalized Levenstein distance) to determine which sentences overlap, as errors in character recognition by docTR module can cause the same sentence to be read differently at different scenes (TimePoints). For the same reason, in the first step of 'dswt,' when detecting the direction of the moving text and determining the optimal length of the interval between consecutive TimePoints, we compare 20-30 TimePoints with close intervals, rather than just a few.
The normalized Levenstein distance is computed as follows:

$$Normalized\ Levenstein\ distance = \frac{Levenstein\ distance\ (s1, s2)}{\max\ (len(s1),\ len(s2))}$$

where the *Levenstein distance* represents the minimum number of edits at character level needed to transform one string to match another.

4. Logic of the code in dswt-reader app

Starting from the output MMIF file from the swt-detection app in CLAMS, we sample the first 20 seconds (the default, but adjustable) of the timeframe classified as dynamic credit at 1-second intervals. Based on the observations shown above, we can detect the type of text - vertically rolling, fade-in and fade-out, or horizontally scrolling - in the timeframe by comparing OCR result on the 1-second interval timepoints sampled from the first 20 at each level (text document, sentence and character).
The initial version of the app focuses on the vertically rolling credits. Therefore, the following description pertains to processing the timeframe that is identified as vertically rolling credits.

1) **Determining the best length of interval (Example needed)**
   To reduce processing time by minimizing the number of scene images to be processed, we Find the optimal length of interval of the timepoints in the timeframe classified as dynamic credit. The optimal length of interval is determined to minimize overlaps between consecutive timepoints while ensuring nothing is missed.
   Starting from the output MMIF file from the swt-detection app in CLAMS, we sample the first 20 seconds (the default, but adjustable) of the timeframe classified as dynamic credit at 1-second intervals. We then run the docTR reader (module) on a scene (image) of each sampled timepoint. The resulting text is split into sentences and stored in a list, and a dictionary is created that maps each timepoint to a list of sentences. Next, we find the best length of intervals that minimizes overlaps while ensuring nothing is missed by incrementing the length of the interval one by one and comparing the sentences of two consecutive timepoints. I wrote two helper functions, `find_overlap(list1, list2)` and `find_best_interval(sorted_tp, map_tp2sentences)`, which work together to identify the best length of interval for selecting time points such that the text overlap between them is minimized.

2) **Resampling and Reading**
   Resample the timepoints at an optimal interval length throughout the entire timeframe, and extract texts from the scenes at the resampled timepoints. At the best interval length identified in the previous step, resample the timepoints between the first and last timepoint in the "targets" property within the Timeframe annotation. Then, run the docTR reader (module) on the scenes at the resampled timepoints. For each timepoint, extract the text at both the sentence and paragraph levels, along with the corresponding bounding box coordinates from the docTR output. Finally, fill in the following attributes:

   ```
   self.tp2sentences_final = {}  # a dictionary mapping timepoint_id to a list of sentences
   self.TP2bb = defaultdict(dict) # a dictioary mapping timepoint_id to a dictionary mapping paragraph_id to its bounding box coordinates
   self.pa2txt = {} # a dictionary mapping paragraph_id to text read at the paragraph level
   ```

3) **Dealing with scenes with multiple columns**

: In analyzing vertically rolling credits, multiple blocks of text sometimes appear within a single scene, often in the form of music track credits as follow:

Figure 8 Example of a scene with multiple columns of text (video [1])

When a paragraph is split across scenes, the OCR process can misread the order of the text. To address this, I developed a function that uses bounding box coordinates to identify scenes containing multiple columns of text. The bounding box coordinates are obtained by docTR reader. Another function groups these paragraphs within a scene based on their x-coordinates, ensuring that text blocks are correctly recognized as belonging to separate columns. Additionally, I created a function to merge these grouped paragraphs across consecutive scenes, using both x and y coordinates to maintain the correct order.

The process begins by identifying scenes with multi-column text, grouping paragraphs with similar y-coordinates. If multiple paragraphs share similar y-coordinates, it indicates the presence of parallel columns in the scene. The paragraphs are then grouped by their x-coordinates, with a y-coordinate constraint to prevent unrelated text from being grouped together.

After identifying and grouping the paragraphs in each scene, consecutive scenes with multi-column text are merged. The merging function compares x-coordinates to align the text correctly, ensuring that the paragraphs maintain their intended structure. This process is repeated for all relevant scenes, and the final merged text is used to update the time point-to-sentences mapping, preserving the correct order of the text across scenes.

- A simple example of the process:

i. For given three consecutive timepoints which are classified as scenes with multiple columns (`multicolumn_list_TP = [[232000, 240000, 248000]]`, where the optimal length of interval is 8000ms), the corresponding screenshots of scenes are shown in Figure 10, 11 and 12. Each paragraph recognized by docTR module is enclosed in a yellow box within the scene images. shown in the yellow boxes.
Each scene image (Figure 10, 11 and 12) is followed by information on the coordinates of the paragraphs, the recognized sentences within each scene, and the resulting paragraph

groupings within that scene. :

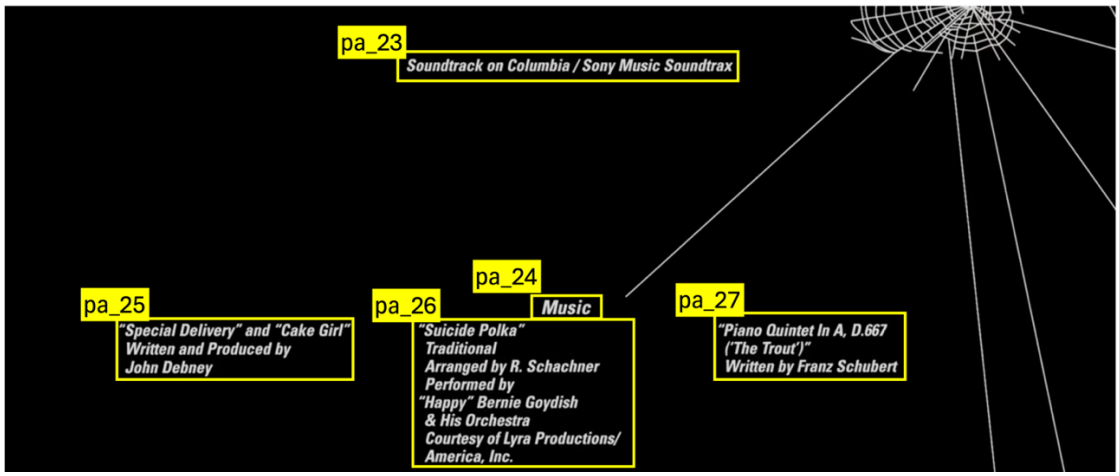

Figure 10 scene image at timepoint 232000 (video [1])

```
### At timepoint 232000

data1 = {'pa_23': ((0.36132812, 0.14355469), (0.65234375, 0.18261719)),
         'pa_24': ((0.47749949, 0.64843744), (0.53226614, 0.6933593)),
         'pa_25': ((0.10449219, 0.6992187), (0.31347653, 0.8193359)),
         'pa_26': ((0.3700989, 0.6962891), (0.5546875, 1.0)),
         'pa_27': ((0.6405041, 0.6969609), (0.8017578, 0.81640625))}

{'pa_23': 'Soundtrack on Columbia / Sony Music Soundtrax',
 'pa_24': 'Music',
 'pa_25': 'Special Delivery" and "Cake Girl"\nWritten and Produced by\nJohn Debney',
 'pa_26': '"Suicide Polka"\nTraditional\nArranged by R. Schachner\nPerformed
by\n"Happy" Bernie Goydish\n& His Orchestra\nCourtesy of Lyra Productions/\nAmerica,
Inc.',
 'pa_27': '"Piano Quintet In A, D.667\n(\'The Trout\')"\nWritten by Franz Schubert'}

grouped1 = [['pa_23'], ['pa_24'], ['pa_25'], ['pa_26'], ['pa_27']]
```

Dohyun Kim (Kimmy)

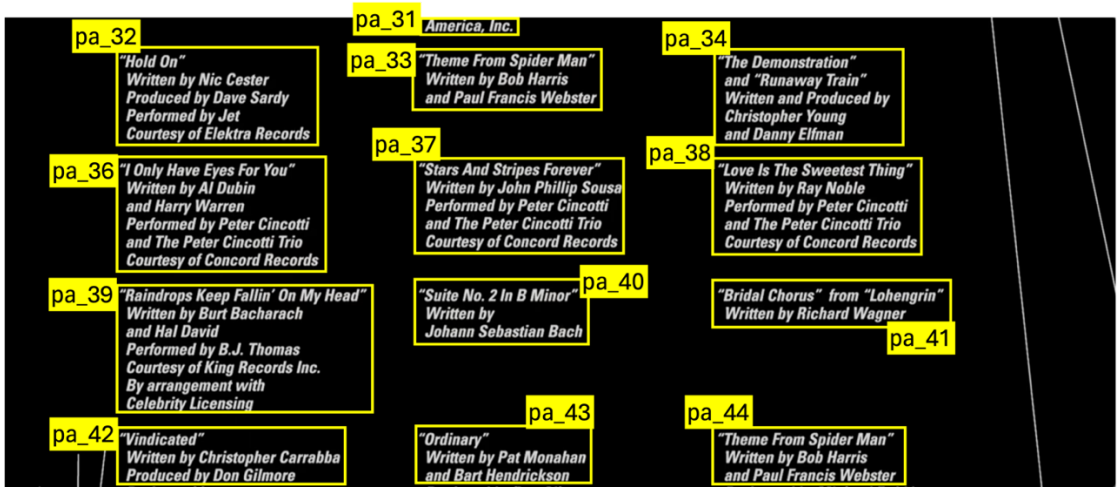

Figure 11 scene image at timepoint 240000 (video [1])

```
### At timepoint 240000,

data2 = {
'pa_31': ((0.37448516, 1.0048828138969839), (0.4608958, 1.0513410456478596)),
 'pa_32': ((0.10546875, 1.0859375), (0.27636716, 1.2744140326976776)),
 'pa_33': ((0.37285033, 1.0732421949505806), (0.53566706, 1.196004956960678)),
 'pa_34': ((0.64257807, 1.0859374850988388), (0.79687494, 1.2763671278953552)),
 'pa_36': ((0.10316178, 1.307279646396637), (0.2853568, 1.5429412722587585)),
 'pa_37': ((0.37293255, 1.305130511522293), (0.5602086, 1.5036299228668213)),
 'pa_38': ((0.64061797, 1.30909264087677), (0.8213357, 1.5056312084197998)),
 'pa_39': ((0.10480399, 1.5740464329719543), (0.32910395, 1.8884633898735046)),
 'pa_40': ((0.37402344, 1.5751953125), (0.5205078, 1.685546875)),
 'pa_41': ((0.64060193, 1.5691418647766113), (0.8459962, 1.6582280397415161)),
 'pa_42': ((0.10802129, 1.8711094346046448), (0.3049383, 2.0014028549194336)),
 'pa_43': ((0.36470693, 1.87109375), (0.52441406, 1.9892578125)),
 'pa_44': ((0.6425781, 1.87890625), (0.8017578, 1.9892578125))
}

{'pa_31': 'America, Inc.',
'pa_32': '"Hold On"\nWritten by Nic Cester\nProduced by Dave Sardy\nPerformed by
Jet\nCourtesy of Elektra Records',
'pa_33': '"Theme From Spider Man"\nWritten by Bob Harris\nand Paul Francis Webster',
'pa_34': '"The Demonstration"\nand "Runaway Train"\nWritten and Produced
by\nChristopher Young\nand Danny Elfman',
'pa_36': '"I Only Have Eyes For You"\nWritten by Al Dubin\nand Harry
Warren\nPerformed by Peter Cincotti\nand The Peter Cincotti Trio\nCourtesy of
Concord Records',
'pa_37': '"Stars And Stripes Forever"\nWritten by John Phillip Sousa\nPerformed by
Peter Cincotti\nand The Peter Cincotti Trio\nCourtesy of Concord Records',
'pa_38': '"Love Is The Sweetest Thing"\nWritten by Ray Noble\nPerformed by Peter
Cincotti\nand The Peter Cincotti Trio\nCourtesy of Concord Records',
'pa_39': '"Raindrops Keep Fallin\' On My Head"\nWritten by Burt Bacharach\nand Hal
David\nPerformed by B.J. Thomas\nCourtesy of King Records Inc.\nBy arrangement
with\nCelebrity Licensing',
'pa_40': '"Suite No. 2 In B Minor"\nWritten by\nJohann Sebastian Bach',
'pa_41': '"Bridal Chorus" from "Lohengrin"\nWritten by Richard Wagner',
'pa_42': '"Vindicated"\nWritten by Christopher Carrabba\nProduced by Don Gilmore',
'pa_43': '"Ordinary"\nWritten by Pat Monahan\nand Bart Hendrickson',
'pa_44': '"Theme From Spider Man"\nWritten by Bob Harris\nand Paul Francis Webster'}

grouped2 = [['pa_31', 'pa_33', 'pa_37', 'pa_40'], ['pa_32', 'pa_36', 'pa_39',
'pa_42'], ['pa_34', 'pa_38', 'pa_41'], ['pa_43'], ['pa_44']]
```

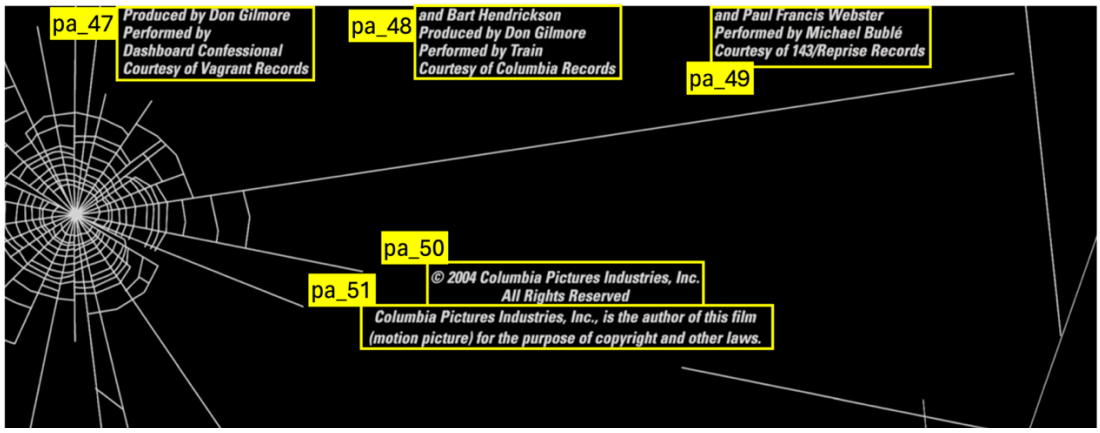

Figure 12 scene image at timepoint 248000 (video [1])

```
### At timepoint 248000,

data3 = {'pa_47': ((0.111328125, 2.0), (0.2802734, 2.1503905951976776)),
 'pa_48': ((0.3779297, 2.0009765625), (0.5576172, 2.1503905951976776)),
 'pa_49': ((0.64746094, 2.0), (0.8388672, 2.11328125)),
 'pa_50': ((0.39007583, 2.554974913597107), (0.6371633, 2.637037515640259)),
 'pa_51': ((0.33342367, 2.639177620410919), (0.69069546, 2.7264272570610046))}

{'pa_47': 'Produced by Don Gilmore\nPerformed by\nDashboard Confessional\nCourtesy
of Vagrant Records',
'pa_48': 'and Bart Hendrickson\nProduced by Don Gilmore\nPerformed by
Train\nCourtesy of Columbia Records',
'pa_49': 'and Paul Francis Webster\nPerformed by Michael Bublé\nCourtesy of
143/Reprise Records',
'pa_50': 'C 2004 Columbia Pictures Industries, Inc.\nAll Rights Reserved',
'pa_51': 'Columbia Pictures Industries, Inc., is the author of this film\n(motion
picture) for the purpose of copyright and other laws.'}

grouped3 = [['pa_47'], ['pa_48'], ['pa_49'], ['pa_50'], ['pa_51']]
```

ii. Then the function `merge_groupings(grouped1, grouped2, data1, data2, x_threshold)` returns `merged_group = [['pa_23'], ['pa_24'], ['pa_25','pa_32', 'pa_35', 'pa_36', 'pa_39', 'pa_42', 'pa_45'], ['pa_26', 'pa_31', 'pa_33', 'pa_37', 'pa_40'], ['pa_27','pa_34', 'pa_38', 'pa_41'], ['pa_43'], ['pa_44']]`.

iii. By applying the `merge_groupings` function to the just returned `merged_group` and `group3`, the final merged group is obtained as follows:
`[['pa_23'], ['pa_24'], ['pa_25','pa_32', 'pa_35', 'pa_36', 'pa_39', 'pa_42', 'pa_45', 'pa_47'], ['pa_26', 'pa_31', 'pa_33', 'pa_37', 'pa_40'], ['pa_27','pa_34', 'pa_38', 'pa_41'], ['pa_43', 'pa_48'], ['pa_44', 'pa_49'], ['pa_50'], ['pa_51']]`

iv. Delete the timepoint key *240000* and *248000* from `self.tp2sentences_final`.

v. Update `self.tp2sentences_final[`*232000*`]=[sentences read in order of ['pa_23', 'pa_24', 'pa_25','pa_32', 'pa_35', 'pa_36', 'pa_39', 'pa_42', 'pa_45', 'pa_47', 'pa_26', 'pa_31', 'pa_33', 'pa_37', 'pa_40', 'pa_27','pa_34', 'pa_38', 'pa_41', 'pa_43', 'pa_48', 'pa_44', 'pa_49', 'pa_50', 'pa_51']`

4) **Text concatenation**
: To handle texts with multiple columns, I devised a method to concatenate sentences while preserving their reading order. The process begins with the texts from the first time point and involves looping through each pair of consecutive time points. At each step, I use the find_overlap function to identify any overlapping sections between the texts of the current and next time point.

Only the non-overlapping part of the text from the next time point is added to a list of concatenated texts. This approach ensures that duplicate content is avoided, and the sequence of text remains intact. Once all relevant texts are processed, the concatenated list is joined into a single string. This combined text is then saved as a text document, aligned with the corresponding timeframe for accurate reference.

A simple example of input and output:
- Given
`sorted_tp = ['tp1', 'tp2', 'tp3']` and
`tp2sentences_final = {'tp1': ['sentence 1'], 'tp2': ['sentence 1', 'sentence 2'], 'tp3': ['sentence 2', 'sentence 3']}`,
`concatenated_text` is initialized with `['sentence1']`.
- For the first loop,
`find_overlap(['sentence 1'], ['sentence 1', 'sentence 2'])` would find and return `['sentence1']` and `'sentence2'` would be appended to `concatenated_text`, resulting in `['sentence 1', 'sentence 2']`.
- For the second loop,
`find_overlap(['sentence 1', 'sentence 2'], ['sentence 2', 'sentence 3'])` would find and return `['sentence2']` and `'sentence3'` would be appended to `concatenated_text`, resulting in `['sentence 1', 'sentence 2', 'sentence3']`.
- Finally, it will return a single string `'sentence 1\nsentence 2\nsentence 3'`.

4. Evaluation

The evaluation was conducted on three vertically rolling end credits videos. First, the swt-app of CLAMS is applied to the three videos to obtain the output MMIF files. As described in the introduction section, Since the dswt-app operates on TimeFrames classified as 'credit,' evaluating the performance of this app requires that the swt-app accurately classifies the ‘credit’ TimePoints. To make that assumption, we manually edited the 'target' of the TimeFrame Annotations in the swt output MMIF so that the start and end of the 'target' correspond to the timepoints where the rolling credits begin and end in the video. Then, the well-configured swt output MMIF is input into the dswt-app. The WER, CER, and BLEU scores are calculated by comparing the text value of the TextDocument generated in the dswt output MMIF with the gold transcript.

	sample_1 (video [1]) start_tp: 0 end_tp: 282000	sample_2 (video [4]) start_tp: 10000 end_tp: 250000	sample_3 (video [5]) start_tp: 52000 end_tp: 295000
WER	0.1178	0.2232	0.2589
CER	0.0604	0.1174	0.1751
BLEU	0.8723	0.7758	0.8102

- The main source of errors

1) docTR's paragraph segmentation errors
: In scenes with multiple columns, if the spacing between adjacent columns is narrow, the line may be recognized as a single sentence, causing the paragraphs to be improperly grouped and the reading order to be incorrect.
For example, for the scene below:

Figure 13 Example of a scene where paragraph segmentation error occurs (video [3])

The paragraph segments we expect, considering the reading order (Job followed by names), are highlighted in yellow boxes in the following image:

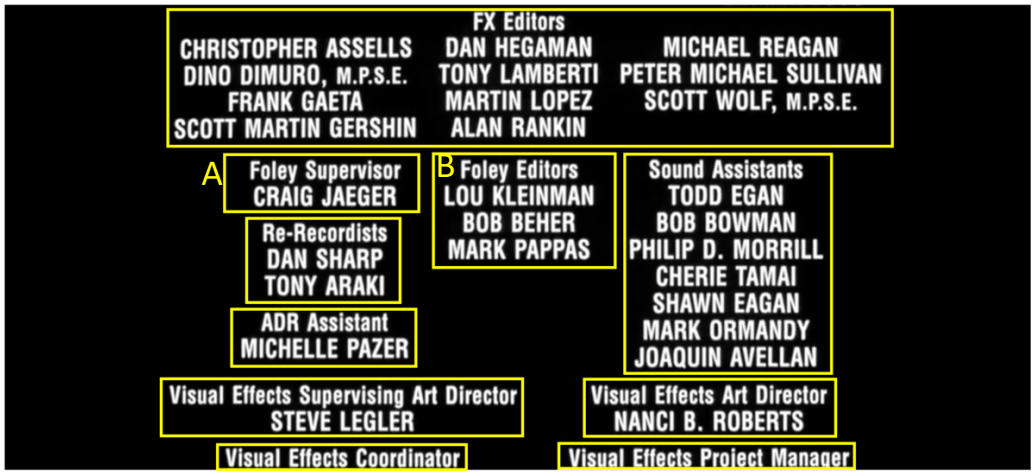

Figure 14 Example of paragraph segmentation expected (video [3])

However, in lines where the spacing between texts belonging to different columns or boxes is narrow, those texts were grouped into single sentences, as shown in the green boxes in the following image:

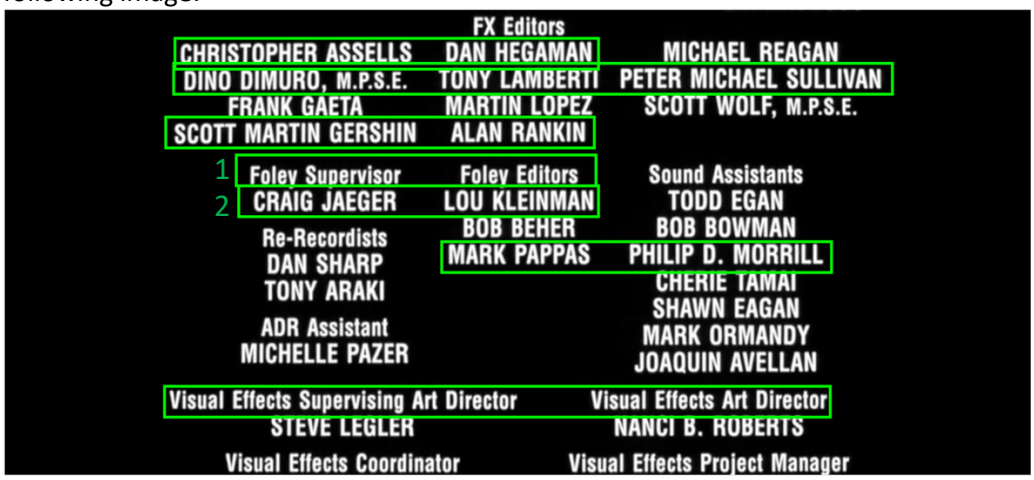

Figure 15 Example of sentence recognition which leads a paragraph segmentation error (video [3])

Job titles or names that should be grouped into separate paragraphs get combined into a single paragraph, which disrupts the reading order. For example, the text in paragraph A and paragraph B of Figure 14 should be read in the order of "Foley Supervisor", "CRAIG JAEGER", and "Foley Editors", "LOU KLEINMAN", followed by "BOB BEHER", and "MARK PAPPAS". However, as shown in lines 1 and 2 of Figure 15, they were grouped differently and read in the order of "Foley Supervisor", "Foley Editors", "CRAIG JAEGER", and "LOU KLEINMAN". As a result, the names of the people corresponding to each job do not follow directly after the job titles, leading to errors in reading order and making it more difficult to extract metadata from the resulted text.

2) Recognizing logos as text
: At the end of the rolling credits, sponsor company logos always appear as in the follow screenshot.

Figure 16 Example of a scene with logos. (video [1])

Since docTR module try to read all the texts in each scene, it also reads and returns the characters that make up logos. For example, the docTR module outputs the following characters from the logo section in Figure 16 above.

FILMED NI\n1\n DANAVISION\nPRINTS BY DELUXE €\n\n #39546\n\nMARREMANG\nV\nKodak\nMO\nTUR FILH\n-\nMENTEL\n\nDIC DOLBY\nITA\n\nsaas Sory Dynamic Sound-\nELECTE IPIDNO n\n\ndts\n\nMOTION PETURE NOLLNOGNSY e AMERN\n

The texts in the logos are usually either too small or in a unique font, leading to incorrect and strange OCR output, which makes the scene with logos problematic.

3) Character recognition errors with specific fonts
: Just as the unique fonts in logos caused errors, there were also characters in the plain text of the credits that were not accurately recognized by the pretrained docTR module due to the font. For example, the font of the rolling credits in video [4] is as follows:

Figure 17 Example of fonts in video [4]

In this credit, the docTR module repeatedly misread “JOHN”, indicated by the yellow box, as “NHOI”.

As another example, “BARRY SONNENFELD” and "ETAN COHEN" in the following scenes were not recognized as text by the docTR module:

Figure 18 Example of fonts in video [6]

Likewise, when specific characters or words are consistently misread throughout a video due to the font, the error rate increases.

5. Future work

1) **Expanding use cases**
   We can a step to classify the type of given dynamic scenes with text - vertically rolling, fade-in and out, or horizontally scrolling text as shown in Section 3. Then, improve the text concatenation step, which is currently focused on vertically rolling credits, to be more customized for each type of dynamic text.

2) **Fine tunning**
   Fine-tuning can help overcome the limitations of the pretrained OCR module, such as paragraph segmentation errors and character recognition errors due to fonts. We can have more data by annotating the gold transcription for more videos containing dynamic credits. In future work, we can use this data to tune hyperparameters of the docTR module's OCR predictor, such as the `paragraph_break` value to make the docTR module more suitable for reading dynamic credits. The x and y coordinate threshold values of the parameters of dswt-app also can be fine tunned with more annotated data.

3) **Logo detection**
   It would be good to be able to distinguish and skip the logo part so that the docTR module does not try to read texts in the logos. In future work, some object recognition packages like YOLO (https://github.com/ultralytics/ultralytics) can be combined to this app and detect logos.

6. Video sources

[1] https://www.youtube.com/watch?v=QiNvOjgIRSQ
[2] https://www.youtube.com/watch?v=4rL_4Xqak1Y
[3] https://www.youtube.com/watch?v=J5uMMOOMSGU
[4] https://www.youtube.com/watch?v=jRefrJH8cfk
[5] https://www.youtube.com/watch?v=5pL4tH2kC0g
[6] https://www.youtube.com/watch?v=Mgxv3fT5eC0

7. Git-hub link to this project

https://github.com/clamsproject/app-dswt-reader