|  |
| --- |
| **1. 주제: 만화 번역 프로그램**  **(가)분반, 1팀, 학번: 20241985, 20231958, 20243302 이름: 조혁진, 김시현, 최연하** |

|  |  |
| --- | --- |
| **2. 요약**  만화나 '짤'의 번역에 있어 기존 방법은 비효율적이었다. 사이트에서 번역된 작업물을 찾거나 구글 이미지 번역, 빅스비 비전을 사용했지만, 문어체나 부적절한 존댓말이 사용되거나 번역된 텍스트가 그림을 가리는 문제가 있었다. 이를 해결하기 위해 이미지 캡셔닝 신경망을 사용해 상황에 맞는 설명을 생성하고, 이를 ChatGPT와 같은 LLM에 넣어 자연스럽고 정확한 번역을 시도했다. 또한, 번역된 텍스트가 그림을 가리면 번호를 넣고 공간을 만들어 문제를 해결했다. FastAPI를 사용해 API로 제작했지만, 버전 1에서는 수직 일본어를 인식하지 못하는 문제를 겪었다. Manga-OCR을 적용했으나 텍스트가 강조된 이미지에서만 잘 작동했고, 이를 해결하기 위해 Poricom 오픈소스를 발견해 수정 후 버전 2 프로그램을 제작했다. | **3. 대표 그림**    그림 1. 버전 1 프로그램    그림 2. 버전 2 프로그램 |

|  |
| --- |
| **4. 서론**  만화나 소위 ‘짤’을 찾아보면 대부분이 일본어나 영어로 되어 있다. 예를 들면 ‘pixiv’라는 사이트에서 대부분의 만화가 일본어 또는 영어로 되어 있다. 그러다 보니 ‘아카라이브’와 같은 사이트에서 번역가가 직접 번역한 작업물을 찾아서 보거나 ‘pixiv’내의 한국어판을 결재해서 보곤 했다. 하지만 이는 너무 비효율적인 방법이다. 자료를 찾느라 시간도 많이 들고, 비용도 많이 들고, 심지어 번역된 작업물이나 한국어판이 아예 없는 경우도 존재했다. 그래서 구글 이미지 번역이나 빅스비 비전을 이용해서 번역을 시도했지만 이 역시 문제가 존재했다. 둘 다 구글 번역을 이용해서 번역을 하는데 대화 상황임을 인지하지 못해 구어체가 아닌 문어체로 번역되거나 상황에 맞지 않게 존댓말이 나오기도 한다. 그리고 한글로 번역한 내용이 그림을 가리는 경우도 종종 있었다. 이러한 문제를 해결하기 위해서 현재 제공되고 있는 서비스의 뼈대를 비슷하게 가져가되 이미지를 설명하는 문장을 만들어내는, 즉 image captioning을 하는 신경망을 이용하여 상황에 대한 설명과 문장을 ChatGPT와 같은 LLM에 넣어 번역의 자연스러움과 정확도를 높여 문제를 해결했다. 번역된 문장이 그림을 가릴 경우 말풍선에 번호를 넣고 만화 밑에 공간을 생성하여 번호와 번역 내용을 넣어주는 방식으로 문제를 해결했다. 파이썬의 FastAPI를 통해 API로 제작했다. 하지만 이 버전 1 프로그램은 수직으로 된 일본어는 OCR 모델이 인식하지 못하는 문제가 있었다. 그래서 Manga-OCR이라는 오픈소스를 찾아서 적용했지만 이 모델은 텍스트 영역이 강조된 이미지에서만 잘 작동하기에 인식이 잘 되지 않았다. 따라서 Poricom이라는 PyQt로 된 만화 번역 프로그램 오픈소스를 발견했고 GUI는 잘 구현되었지만 모델 변경, 번역 등 기능에 결함이 있어 이를 수정해서 버전 2 프로그램을 제작했다. |

|  |
| --- |
| **5. 본론**  버전 1 프로그램은 글자를 추출하는 부분, 이미지가 어떤 상황인지 분석하는 부분, 추출된 글자를 한국어로 번역하는 부분, 번역된 결과를 이미지에 반영하는 부분으로 나뉜다. 글자를 추출하는 부분을 구현하기 위해서 PaddleOCR(7-1)과 Manga-OCR 오픈소스(7-2)를 활용했다. 이미지가 어떤 상황인지 분석하는 부분을 구현하기 위해서 Hugging Face의 Image captioning 모델을 활용했다. 추출된 글자를 원하는 언어로 번역하는 부분은 ChatGPT API를 활용했고 번역된 결과를 이미지에 넣는 부분은 Pillow를 활용했다. (8-1 그림)  글자를 추출하기 위해서 OCR 기술을 활용하기로 했고 OCR 오픈소스인 PaddleOCR을 사용해서 영어와 일본어를 인식하도록 했다. 실제로 일반적인 문서 사진에서 실제로 글자 인식에 성공했다. 텍스트가 있는 영역을 스스로 판단하여 정하고 텍스트 인식을 했고 정한 영역에 대한 픽셀 값도 잘 반환했다. 하지만 문제점이 존재했는데 수직으로 된 일본어는 인식 못하는 문제점도 존재했다. 수직 방향 글이 있는 말풍선 영역을 글자 영역이라고 판단하지 못하고 그에 따라 텍스트 인식도 못했다. 이를 해결하고자 깃허브에서 Manga-OCR이라는 오픈 소스를 찾았다. 이 오픈소스는 일본어를 수직, 수평 방향 둘 다 인식할 수 있는 라이브러리이다. 테스트 결과, 수직 및 수평 방향 둘 다 인식이 가능함을 확인했다. 하지만 이 오픈 소스도 문제점이 존재했는데 텍스트가 있는 영역을 잘 분별하지 못한다는 점이었다. 즉, 그림의 상당 부분을 텍스트가 존재하도록 잘라주어야 텍스트를 인식했다. (8-2, 3 그림)  이미지가 어떤 상황인지 분석하기 위해서 TensorFlow의 오픈소스(7-3)를 사용하기로 했다. 구동되는 TensorFlow 버전을 설정한 후 테스트 결과, 주변 환경과 특정 인물 또는 사물에 대한 설명이 잘 나왔다. 하지만 문제가 존재했다. 이 오픈소스는 MobileNetV3Small 모델을 통해서 이미지를 분할하여 전처리된 데이터(캡션)을 얻고 이를 2-layer-transformer-decoder에 넣어 전체 문장을 만드는 구조이다. 그런데 각 모델의 레이어가 커스텀 레이어라 직렬화가 되어 있지 않았고 각 레이어 클래스에 get\_config() 메소드를 넣어서 직렬화를 시도했지만 실패했다. 그에 따라 다른 코드와 연결이 원활히 이루어지지 않았다. 그래서 Hugging Face라는 인공지능 모델 관련 오픈 소스 커뮤니티에 들어가서 관련 자료를 찾아봤고 위의 TensorFlow 모델(ViT)이 모듈로 구현된 것을 발견했다. (7-4) 또한 이를 COCO 이미지 데이터로 학습시킨 오픈소스(7-5)를 발견하여 이를 최종적으로 사용했다. 하지만 버전 문제가 있어 이미지 처리를 위한 모듈(ViTImageProcessor)을 하위 버전 AutoFeatureExtractor로 교체하여 프로젝트에 활용했다.  추출된 언어를 한국어로 번역하는 것은 ChatGPT API(7-6)를 활용했다. 가장 성능 좋은 LLM으로 알려져 있고 공식 사이트에 API를 활용하는 방법에 대해 자세한 설명이 있어 구현하기도 쉽고 다른 코드와 연결하기도 수월해 이를 활용했다.  번역된 결과를 이미지에 반영하는 부분은 Pillow 라이브러리(7-7)를 활용했다. 기존에는 OpenCV(7-8)를 통해서 블러 처리와 텍스트 삽입을 구현하려고 했으나 블러 처리는 잘 되었지만 한국어 텍스트가 계속 깨지는 오류가 있어서 Pillow로 전부 구현하기로 했다. PaddleOCR이 텍스트를 감지하면 그 근처에서 사각형을 형성하고 상단 좌측, 상단 우측, 하단 우측, 하단 좌측 순으로 픽셀 값을 준다. 사각형 높이에 해당하는 값을 글자 크기로 잡고 글자가 시작할 부분을 픽셀 값 계산을 통해 구해서 한글로 번역한 것을 본래 글이 있던 곳에 추가할 수 있었다.  그런데 텍스트가 범위를 넘어가는 경우가 있어 텍스트가 범위를 넘어갔다면 Pillow를 통해 여백을 사진에 추가하여 텍스트를 넣는 방식도 구현했다. 픽셀 값으로 텍스트의 크기를 잡고 이를 기반으로 텍스트 범위를 파악하여 글자가 범위를 넘어갔는지 확인할 수 있다. 만약 넘어갔다면, 높이가 더 길어진 흰색 이미지를 새로 만들고 맨 위에 기존 사진을 붙여 넣고 하단에 남은 부분에 텍스트를 추가하는 방식으로 구현했다.  각 기능들을 함수 또는 클래스로 만들고 total.py에서 하나의 클래스로 모아 이미지 처리 함수를 만들었고 main.py에서 FastAPI(7-9)를 통해 서버를 만들고 total.py를 import해서 서버 안에 이미지 처리 함수를 추가했다. 클라이언트로부터 POST 요청으로 이미지 파일이 서버에 전송되면 임시로 서버에 저장하고 이미지 처리를 하고 그 결과물도 임시로 서버에 저장하고 다시 클라이언트에 이미지 파일을 전송해주고 임시 파일들을 삭제하는 방식으로 구현했다.  하지만 프로그램을 테스트했을 때 원하는 만큼의 성능이 나오지 않았다. 만화의 텍스트가 1~2개만 인식되거나 아예 인식되지 않는 경우가 발생했다. 원인을 분석해보니 OCR에서 기능적 한계가 있었는데 PaddleOCR은 글이 수직으로 있을 때 인식하지 못했고 Manga-OCR은 텍스트가 작거나 여러 개 있으면 아예 인식을 못하는 문제가 있었다.  그래서 기능적 한계로 인한 인식 문제를 극복하기 위해서 깃허브를 탐색했고 Poricom이라는 오픈소스를 발견했다. PyQt를 기반으로 만들어진 만화 번역 컴퓨터 프로그램이다. OCR 모델로 traineddata 파일이 포함된 tesserocr과 Manga-OCR이 사용되었고 번역에는 ChatGPT API, DeepL API이 사용되었다. 특히 이 프로그램의 특징은 이미지 수정 시 발생했던 텍스트 침범 문제가 존재하지 않는다는 것이다. 왜냐하면 이미지 수정 기능이 없기 때문인데 그 기능을 추가하지 못한 데에는 구조적인 문제가 있다.  OCR할 때 rubberBandRect를 통해서 드래그한 부분을 Qbuffer에 저장하고 save 함수로 png 형태로 메모리에 저장하여 BytesIO로 메모리에 접근한 후 Pillow를 통해 저장했던 이미지 데이터 열어서 OCR을 진행하는 구조를 갖는다. 이때 rubberBandRect가 QGraphicView 위젯에서의 좌표인 뷰포트 좌표를 반환해서 이미지의 픽셀값을 잡기가 어렵다. 그래서 번역할 텍스트를 드래그 해서 정해주면 오른쪽 하단의 PyQt 상자에 번역된 텍스트를 띄워 주는 기존 프로그램 방식을 그대로 사용하기로 했다. (대표 그림-그림 2 참고) 따라서 이를 기반으로 텍스트 침범 문제가 없고 OCR 인식 문제가 해결된, 그리고 ViT 모델을 통해 번역의 질을 높인 버전 2 프로그램을 만들었다.  하지만 Poricom 오픈소스(7-10)는 문제가 많았다. 우선 OCR 모델 교체 GUI는 존재했으나 기능이 작동하지 않았다. 따라서 Manga-OCR만 사용할 수 있었고 일본어만 추출할 수 있었다. (8-4 그림) 그래서 이를 해결하기 위해서 pixmapToText.py 파일을 수정해서 tesserocr이 작동되게 만들고 states.py에서 OCR 모델 교체 부분에 return이 되어 있는 것을 제거하고 코드를 수정해서 모델 교체를 할 수 있게 만들었다. 또한 번역 API 선택 GUI는 존재했으나 기능이 구현되지 않았다. (8-5 그림) 알고 보니 windows/base.py에서 모델과 API 교체 부분을 return이 되어 있어서 교체가 일어나지 않았다. 따라서 이를 수정하고 API 보내는 주소에도 문제가 있어 적절한 ChatGPT 주소를 넣어서 구동이 되게 만들었다. 이후 Hugging Face의 ViT 오픈소스를 클래스 형태로 변형한 후 프로그램과 연결했고 모델의 반환값을 ChatGPT API 프롬프트 부분에 추가했다. 그런데 ImageCaptioning 시 많은 시간이 들어서 components/settings/translate.py에 EnableImage라는 항목을 만들어서 시간 단축을 위해서 ImageCaptioning을 비활성화할 수 있도록 만들었다. 그리고 EnableImage가 enable할 경우에만 프롬프트가 변하도록 만들었다. 이렇게 버전 2 프로그램을 만들어서 버전 1 프로그램의 인식 문제를 해결했다. (8-6 그림)  그래도 여전히 문제점은 존재한다. ViT 모델이 인물의 표정이나 행동과 같은 정밀한 부분까지 설명 못해 장면에 대한 설명이 부족한 문제, 프롬프트 작업을 해도 ChatGPT가 번역 시 높임말, 문어체 사용하는 문제가 남아 있다. |

|  |
| --- |
| **6. 결론**  버전 1, 2 프로그램 둘 다 모든 기능을 다 구현하고 각 기능끼리 연결해서 최종 결과물까지 잘 나왔다. 버전 1 프로그램은 수평 방향의 텍스트에 대해서 일본어, 영어 다 인식 가능하고 번역도 잘 되었다. 버전 2 프로그램은 수평, 수직 방향의 텍스트에 대해서 일본어, 영어 다 인식 가능하고 번역도 잘 되었다. 그렇지만 여전히 개선해야 할 부분은 남아 있다.  버전 1 프로그램의 개선점은 다음과 같다. PaddleOCR이 수직으로 된 글도 인식할 수 있도록 모델을 수정하거나 글이 있는 영역을 잘라 반환하는 모델을 Manga-OCR에 더하는 방법 등 OCR 모델의 글자 인식률을 높이도록 해야 한다. ChatGPT보다 번역에 특화된 LLM을 사용하여 번역의 질을 높여야 하고 ImageCaptioning을 정교히 하기 위해서 인물의 특징, 표정으로 상황을 면밀히 해석하거나 만화 줄거리를 기반하여 좀 더 정확한 상황 설명을 하도록 해야 한다. 또한 API 속도 향상을 위해서 OCR 부분, ImageCaptioning 부분과 같은 이미지 처리 부분의 최적화가 필요하다.  버전 2 프로그램의 개선점은 버전 1과 마찬가지로 ChatGPT보다 번역에 특화된 LLM을 사용하여 번역의 질을 높여야 하고 ImageCaptioning을 정교히 하기 위한 작업을 하는 것이다. 또한 ImageCaptioning 부분에 의해 속도가 많이 느려졌기 때문에 이 부분의 최적화가 필요하다. |

**7. 출처**

[1] PaddleOCR, github, <https://github.com/PaddlePaddle/PaddleOCR/blob/main/README_en.md>

[2] Manga-OCR, github, <https://github.com/kha-white/manga-ocr>

[3] ImageCaptioning, TensorFlow, <https://www.tensorflow.org/text/tutorials/image_captioning>

[4] vit-gpt2-image-captioning, Hugging Face, <https://huggingface.co/nlpconnect/vit-gpt2-image-captioning>

[5] The Illustrated Image Captioning using transformers, Ankur, <https://ankur3107.github.io/blogs/the-illustrated-image-captioning-using-transformers/>

[6] OpenAI API reference, OpenAI Platform, <https://platform.openai.com/docs/api-reference/introduction>

[7] Pillow 이미지에 텍스트 추가하기, tistory, <https://daco2020.tistory.com/832>

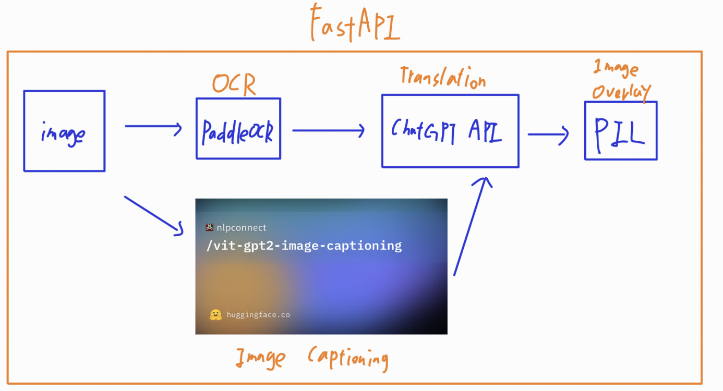
[8] OpenCV를 사용한 이미지 처리 – 블러링, tistory, <https://data-science-note.tistory.com/33>

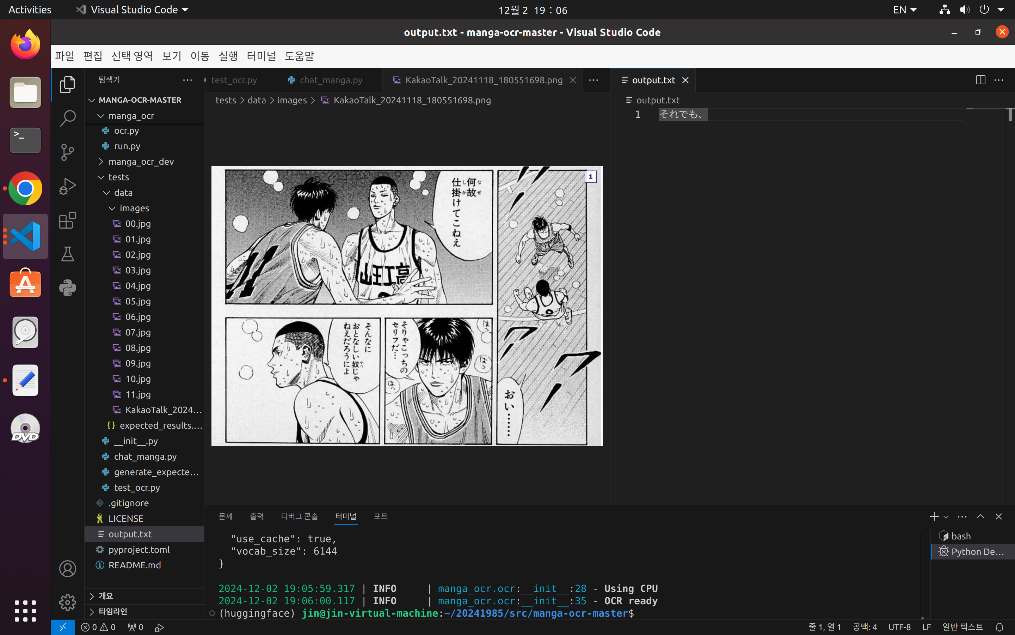
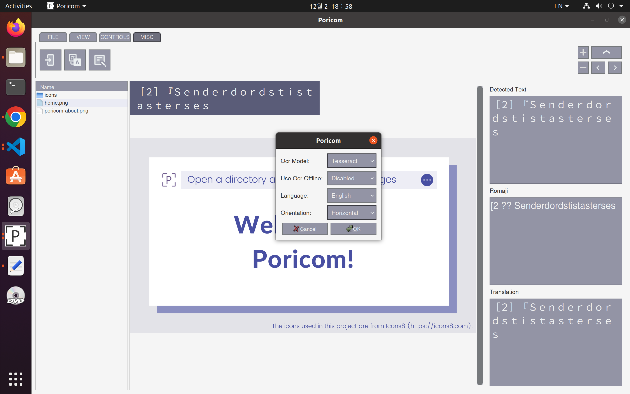
[9] FastAPI 이미지 업로드, 다운로드 구현하기, tistory, <https://mopil.tistory.com/m/63>

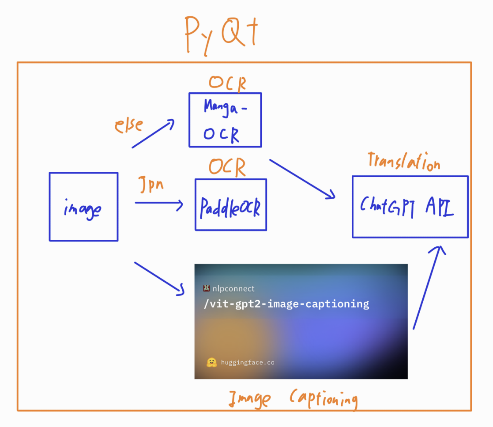
[10] Poricom, github, <https://github.com/blueaxis/Poricom>

[11] Probing the Need for Visual Context in Multimodal Machine Translation, Ozan Caglayan, Pranava Madhyastha, Lucia Specia, Loïc Barrault, <https://arxiv.org/abs/1903.08678>

**8. 본문 그림**

(1) 버전 1 프로그램 구조 (2) PaddleOCR 문제점

[3] Manga-OCR 문제점 [4] Poricom Manga-OCR만 사용 가능 문제

[5] Poricom 번역 불가 문제 [6] 버전 2 프로그램 구조