AI 포팅 메뉴얼

개발 환경

- 리눅스 20.04
- 파이썬 3.9.7
- 텐서플로우 2.7.0
- Node js 20.11.0

동작 플로우

▼ 1) 사전 학습 단계

1. 데이터 셋 만들기 (font2img)

- 데이터 셋에 쓸 폰트와 추출한 이미지 갯수 지정
- ex) 폰트 40개 - 2000장 추출이 프로젝트에서 Best Case

▼ **실행코드** - 스크립트로 진행하는 것을 추천

```
python font2img.py --src_font=/home/j-i10b111/neural_font
s/fonts/source/source_font.ttf \
                   --dst_font=/home/j-i10b111/neural_font
s/fonts/target/01.ttf \
                   --charset=KR \
                   --sample_count=1000 \
                   --sample_dir=dir \
                   --label=0 \
                   --filter=1 \
                   --shuffle=1
```

2. 학습할 데이터 만들기 (package.py)

- 전체 데이터의 90% = train.obj
- 전체 데이터의 10% = val.obj

▼ **실행코드**

```
python package.py --dir=/home/j-j-i10b111a/neural_fonts/dir \
                  --save_dir=/home/j-j-i10b111a/neural_fonts/experiment/data
```

3. 학습하기 (train.py)

- 2 Step으로 실행
- 로스율과 패널티값의 차이

▼ **실행 코드**

```
# Step 1
CUDA_VISIBLE_DEVICES=1 python train.py --experiment_dir=experiment \
                --experiment_id=0 \
                --batch_size=32 \
                --lr=0.001 \
                --epoch=30 \
                --sample_steps=100 \
                --schedule=10 \
                --L1_penalty=100 \
                --Lconst_penalty=15

# Step 2
CUDA_VISIBLE_DEVICES=2 python train.py --experiment_dir=experiment \
                --experiment_id=0 \
                --batch_size=32 \
                --lr=0.001 \
                --epoch=120 \
```

```
                --sample_steps=100 \
                --schedule=40 \
                --L1_penalty=500 \
                --Lconst_penalty=1000
```

▼ 옵션 설명

```
CUDA_VISIBLE_DEVICES=1 # GPU 할당해서 실행 (GPU가 여러개 일 경우)
--experiment_id=0  # 실험 넘버 0 할당 (experiment 폴더에 생성되는 네이밍에 관여)
--batch_size=32  # 학습 한번에 주어지는 파일 갯수 (32 or 64 추천한다는 스오플 글 봤음)
--epoch=30  # 몇 에포크(사이클)돌릴 건지
--sample_steps=100  # 샘플 저장할 스텝 갯수 단위 (ex. 100개 마다 샘플 저장)
--schedule=10  # 학습률 저하시킬 스케줄 (10 epoch 마다 지정된 비율만큼 학습률 저하, 최저 학습률 존재)
--freeze_encoder=3  # 과적합 되면 인코더를 얼려볼 수도 있다
```

- /experiment/sample의 결과로 학습되고 있는 현 모델의 성능을 확인할 수 있다
- **동작 원리**
  - train.obj로 학습 도중 sample_step의 배수가 되면 val.obj에서 랜덤으로 값을 뽑아와서 현 모델로 추론을 돌리는 과정이 포함되어 있음

4. 추론 (~~infer.py~~ ⇒ infer_by_train.py)

- 모델의 성능 테스트를 위해 추론하는 과정
- But, 버전 차이(추정)로 추론 코드의 일부가 동작하지 않음
- 추론 과정을 별도로 돌리는 것은 수십번 시도하였으나 Fail
- 고민하다 train 과정에서 샘플을 만들 때 추론을 돌리는 로직을 활용
  - ⇒ infer_by_train을 별도로 생성

주의사항

- 모든 디렉토리는 미리 있는게 좋다
- 특히 /experiment/data가 제 위치에 있는지 확인 필요

▼ 2) 손글씨 학습 및 추론 단계

1. 손글씨 데이터 셋 만들기 (font2img)

- 템플릿 or OCR한 손글씨 이미지 set

2. 학습할 데이터 만들기 (package.py)

- 템플릿/OCR 손글씨 전체 이미지를 인풋으로 train.obj 생성

3. 학습하기 (train.py)

- 120 ~ 200 epoch
- 사전 학습 단계에서 학습된 모델(checkpoint)에 다가 손글씨를 추가학습
- /experiment/checkpoint 디렉토리에 사전에 학습된 모델 파일 4가지를 추가 (아래는 예시)

  - checkpoint 파일 내 모델명이 추가한 파일들과 똑같은지 확인 필요
- 학습 코드는 위와 똑같다

4. 추론하기 (infer_by_train.py)

- train 로직에서 가져온 추론 과정
- 해당 로직을 사용해서 나눔고딕을 기준으로 11172자를 추출
- 11172자에 내가 원하는 손글씨의 라벨을 할당하면 해당 스타일을 입힌 글자가 추출됨
- 11172자는 유니코드로 출력 가능
- 결과물 : 손글씨 스타일이 적용된 11172자의 이미지

▼ 3) 손글씨 폰트 제작 단계

1. 이미지 노이즈 처리 (~~대비 증가~~ ⇒ 적용하지 않아도 됨)

- 기본적으로 아웃풋으로 나온 이미지들은 회색조이다 (코드 내부에 필터 코드가 있어서)
- 폰트도 회색으로 나오지 않을까 했지만 벡터 방식의 이미지 덕분에 아웃라인만 따면 되어서 큰 상관 없었음
- 다만 이미지에 노이즈가 많아 날리는 과정이 필요하다면 이미지 전처리 과정에 필터를 써도 된다
  - 양방향 필터
  - 가우시안

  ▼ 단순 범위 지정해서 색상 대비를 늘리는 방법도 사용하였음

```
img_np[img_np <= 110] = 30
img_np[img_np > 165] = 255
```

- 디폴트로 세팅된 아웃풋에서는 181 * 181로 크롭하는 것이 가장 효율적인 사이즈였다

2. 파일 이름, 유니코드로 지정

- 폰트로 만들 때, 폰트의 유니코드 순으로 정렬하는 과정이 중요

- 사유
  - 차후에 사용할 툴인 svgicons2svgfont가 svg 이미지들을 유니코드로 지정해야 single SVG로 만들 때 내부에 유니코드가 전부 들어가 있는 구조로 동작함
  -

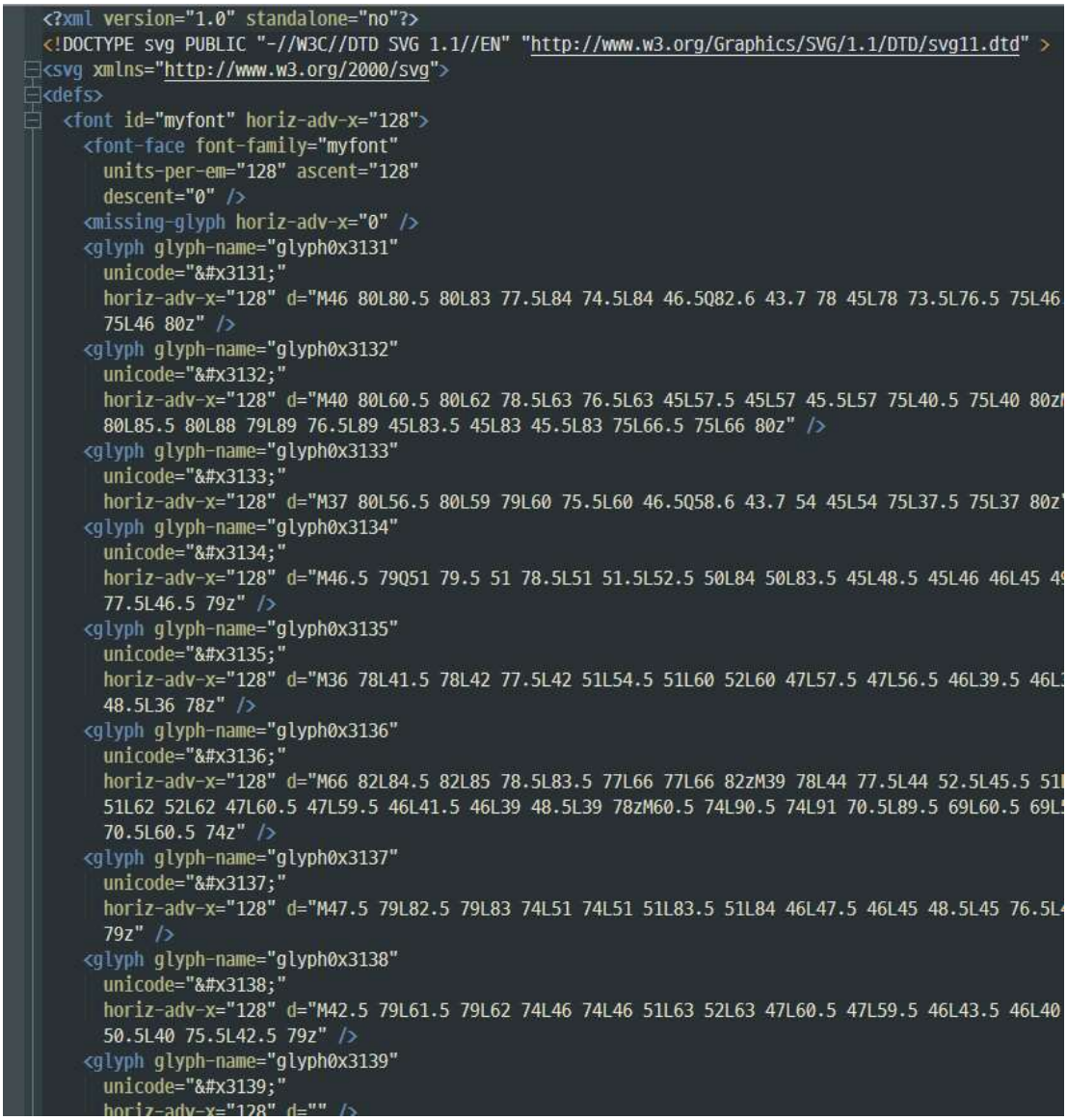
3. img(jpg, png) to SVG

- 폰트로 만들기 위해선 이미지가 벡터 값을 가져야 함
- img to SVG로 변환하는 과정이 필요
- **gulp-img2svg (git)** 란 툴을 사용

4. SVGs to SVG

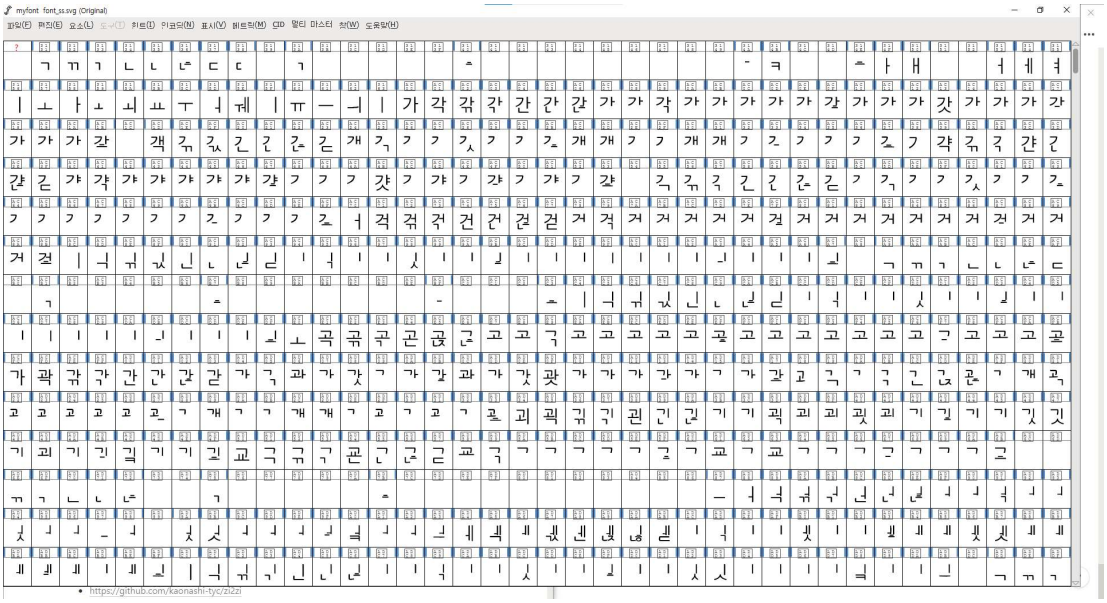
- 여러개의 SVG들을 단일 SVG로 변환 하는 과정 필요
- 단일 SVG여야만 ttf로 변환 가능한 툴을 사용할 수 있음
- svgicons2svgfont (git) 사용해서 SVG 손글씨 이미지들을 하나의 SVG로 변환

▼ 만들어진 단일 SVG 파일은 메모장으로 열어 확인 가능



```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd" >
<svg xmlns="http://www.w3.org/2000/svg">
<defs>
  <font id="myfont" horiz-adv-x="128">
    <font-face font-family="myfont"
      units-per-em="128" ascent="128"
      descent="0" />
    <missing-glyph horiz-adv-x="0" />
    <glyph glyph-name="glyph0x3131"
      unicode="&#x3131;"
      horiz-adv-x="128" d="M46 80L80.5 80L83 77.5L84 74.5L84 46.5Q82.6 43.7 78 45L78 73.5L76.5 75L46
      75L46 80z" />
    <glyph glyph-name="glyph0x3132"
      unicode="&#x3132;"
      horiz-adv-x="128" d="M40 80L60.5 80L62 78.5L63 76.5L63 45L57.5 45L57 45.5L57 75L40.5 75L40 80z
      80L85.5 80L88 79L89 76.5L89 45L83.5 45L83 45.5L83 75L66.5 75L66 80z" />
    <glyph glyph-name="glyph0x3133"
      unicode="&#x3133;"
      horiz-adv-x="128" d="M37 80L56.5 80L59 79L60 75.5L60 46.5Q58.6 43.7 54 45L54 75L37.5 75L37 80z
    <glyph glyph-name="glyph0x3134"
      unicode="&#x3134;"
      horiz-adv-x="128" d="M46.5 79Q51 79.5 51 78.5L51 51.5L52.5 50L84 50L83.5 45L48.5 45L46 46L45 4
      77.5L46.5 79z" />
    <glyph glyph-name="glyph0x3135"
      unicode="&#x3135;"
      horiz-adv-x="128" d="M36 78L41.5 78L42 77.5L42 51L54.5 51L60 52L60 47L57.5 47L56.5 46L39.5 46L
      48.5L36 78z" />
    <glyph glyph-name="glyph0x3136"
      unicode="&#x3136;"
      horiz-adv-x="128" d="M66 82L84.5 82L85 78.5L83.5 77L66 77L66 82zM39 78L44 77.5L44 52.5L45.5 51
      51L62 52L62 47L60.5 47L59.5 46L41.5 46L39 48.5L39 78zM60.5 74L90.5 74L91 70.5L89.5 69L60.5 69L
      70.5L60.5 74z" />
    <glyph glyph-name="glyph0x3137"
      unicode="&#x3137;"
      horiz-adv-x="128" d="M47.5 79L82.5 79L83 74L51 74L51 51L83.5 51L84 46L47.5 46L45 48.5L45 76.5L4
      79z" />
    <glyph glyph-name="glyph0x3138"
      unicode="&#x3138;"
      horiz-adv-x="128" d="M42.5 79L61.5 79L62 74L46 74L46 51L63 52L63 47L60.5 47L59.5 46L43.5 46L40
      50.5L40 75.5L42.5 79z" />
    <glyph glyph-name="glyph0x3139"
      unicode="&#x3139;"
      horiz-adv-x="128" d="" />
```

▼ fontforge 툴을 이용해서 각 유니코드에 해당하는 이미지로 잘 적용되었는지, 어떻게 보이는지도 확인 가능



5. SVG to ttf

- svg2ttf 사용하여 SVG 파일을 ttf로 변환

참조

- https://github.com/kaonashi-tyc/zi2zi
- https://github.com/yjjng11/Neural-fonts-webapp
- https://github.com/nfroidure/svgicons2svgfont
- https://github.com/fontello/svg2ttf