VIETNAM NATIONAL UNIVERSITY - HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE & ENGINEERING

SOFTWARE ENGINEERING

Report

Font Detection on UI Screenshots

Instructor: Quan Thanh Tho, PhD.
Student: Le Khac Minh Dang - 1810109

HO CHI MINH CITY, JUNE 2021

Contents

1 Requirement Analysis

1.1 Problem Statement

The goal of this project is to make a system that takes screenshots of different UI designs as input, and gives all the fonts that are being used on the screenshots as output.

These are some boundaries of the system that:

- There would be no images taken by cameras as input, only clean screenshots.
- The system only needs to classify regular and **bold** styles for each font.
- If the text to be detected is not using one of the font that we used to train the classification system, the result should be a font that is as close to the real font as possible.

1.2 System Design

Breaking down the problem statement, this system consists of 2 different parts: one for text detection to detect all the texts on the screenshot, and another one for font classification on each of the detected text.

As for the text detection part, this is a very popular problem that many people has done before.

For the font classification part, this is quite a more difficult problem, so my solution is to follow the deep learning model introduced in the paper *DeepFont: Identify Your Font from An Image*[1].

An input screenshot will go into the system and then come out as another image with all texts detected in bounding boxes with a label on top as their fonts.

2 Implementation

Since I'm neither studying nor working in the field of AI, my knowledge about machine learning is very limited. Therefore, the approach that I chose to solve this problem is to reuse what other people have done, and just modify them here and there to match my requirement. These are the open source projects that I reused:

1. **TextRecognitionDataGenerator** by *Belval*[2]

2. **DeepFont** by *robinreni96*[3]

3. **Text Detection** by *dodanh041001*[4]

My github repository for the final result is https://github.com/lkmidas/Font-Detection. It turned out that I need to do quite a lot of modification for the system to work correctly. So in the below sections, I will state in details all my modifications.

2.1 Text Data Generation

The first part is to generate training data for the DeepFont model. The project that I used is **TextRecognitionDataGenerator** by *Belval*[2]. It is a very simple and ready to use *pip* package, so all I had to do is writing a small script for it:

```
import os
import sys

fonts_dir = sys.argv[1]

for subdir, dirs, files in os.walk(fonts_dir):
    for d in dirs:
        ttf_dir = os.path.join(subdir, d)
        out_dir = os.path.join(sys.argv[2], d)
        print("Generating data for font {}".format(d))
        os.system("trdg -c {} -w 1 -f 100 -fd {}
            --output_dir {}".format(sys.argv[3], ttf_dir, out_dir))
```

Usage: `python3 data_generator.py font_dir out_dir image_count`, where

- `fond_dir` is a directory contains one sub-directory for each font with a .ttf file in it.
- `image_count` is the number of image to generate for each font

For my model, I generated 16000 images in total for 10 fonts (2 styles each font). Training data can be accessed at https://drive.google.com/drive/folders/1cd6p4V1zsMPK2Xw1N4uzz9443yp7XINP?usp=sharing

2.2 Training DeepFont Model

For this part, I reused the iPython note book from **DeepFont** by *robinreni96*[3]. However, the notebook is very old and contains many problems in it:

- It loads all the training data onto RAM first before running. This is fine for training a small demo model, but for a bigger model that has lots of labels and requires more training data, it will crash for running out of RAM.
- It uses python Pillow for processing images, which in my opinion is a very barebone package and not widely used anymore. Instead, I used OpenCV `cv2`.
- The tensorflow and keras versions that it uses is also old, so some functions are deprecated and need to be replaced.

For the problem of running out of RAM, my solution is to rewrite the part where it loads the data into using keras `ImageDataGenerator`:

```
def preprocess(img):
  blur = cv2.GaussianBlur(img, (3, 3), 0)
  blur.shape = (105, 105, 1)
  return blur

IMAGE_SHAPE = (105, 105)
data_path = "/content/gdrive/MyDrive/deepfont_data_normal_bold/"
image_generator = tf.keras.preprocessing.image.ImageDataGenerator(
  zoom_range=0.2,
  width_shift_range=0.1,
  height_shift_range=0.1,
  preprocessing_function=preprocess,
  validation_split=0.4,
  rescale=1/255
)
image_train = image_generator.flow_from_directory(directory=data_path,
    color_mode="grayscale", target_size=IMAGE_SHAPE, subset="training")
image_val = image_generator.flow_from_directory(directory=data_path,
    color_mode="grayscale", target_size=IMAGE_SHAPE, subset="validation")
```

The old code only classifies 5 labels, I changed it to classify 20 labels in total for 10 fonts (regular and bold). The fonts are: Lato, Montserrat, Nunito, Open Sans, Poppins, Raleway, Roboto, Rubik, Source Sans Pro, Work Sans.

Other hyperparameters are also tuned accordingly, refer to the source code for more details.

2.3 Font Detection

This is derived from **Text Detection** by *dodanh041001*[4]. It is already a well-written project, all I have to do is to add another block of code to make it loops through all the detected text bounding boxes and run font classification on them as well:

```
deepfont = load_model('deepfont_model.h5')

for subdir, dirs, files in os.walk(samples_folder):
    for f in files:
        img_path = os.path.join(subdir, f)

        # load data
        image = imgproc.loadImage(img_path)
        origin_img = cv2.imread(img_path)
        origin_img = cv2.cvtColor(origin_img, cv2.COLOR_BGR2RGB)
        body_text = []
        heading = []
        color_text = []
        bboxes, polys, score_text = test_net(net, image, args.text_threshold,
            args.link_threshold, args.low_text, args.cuda, args.poly, refine_net)
        labels = []
        label_list = [
            "Lato", "Lato (bold)",
            "Monserrat", "Monserrat (bold)",
            "Nunito", "Nunito (bold)",
            "Open Sans", "Open Sans (bold)",
            "Poppins", "Poppins (bold)",
            "Raleway", "Raleway (bold)",
            "Roboto", "Roboto (bold)",
            "Rubik", "Rubik (bold)",
```

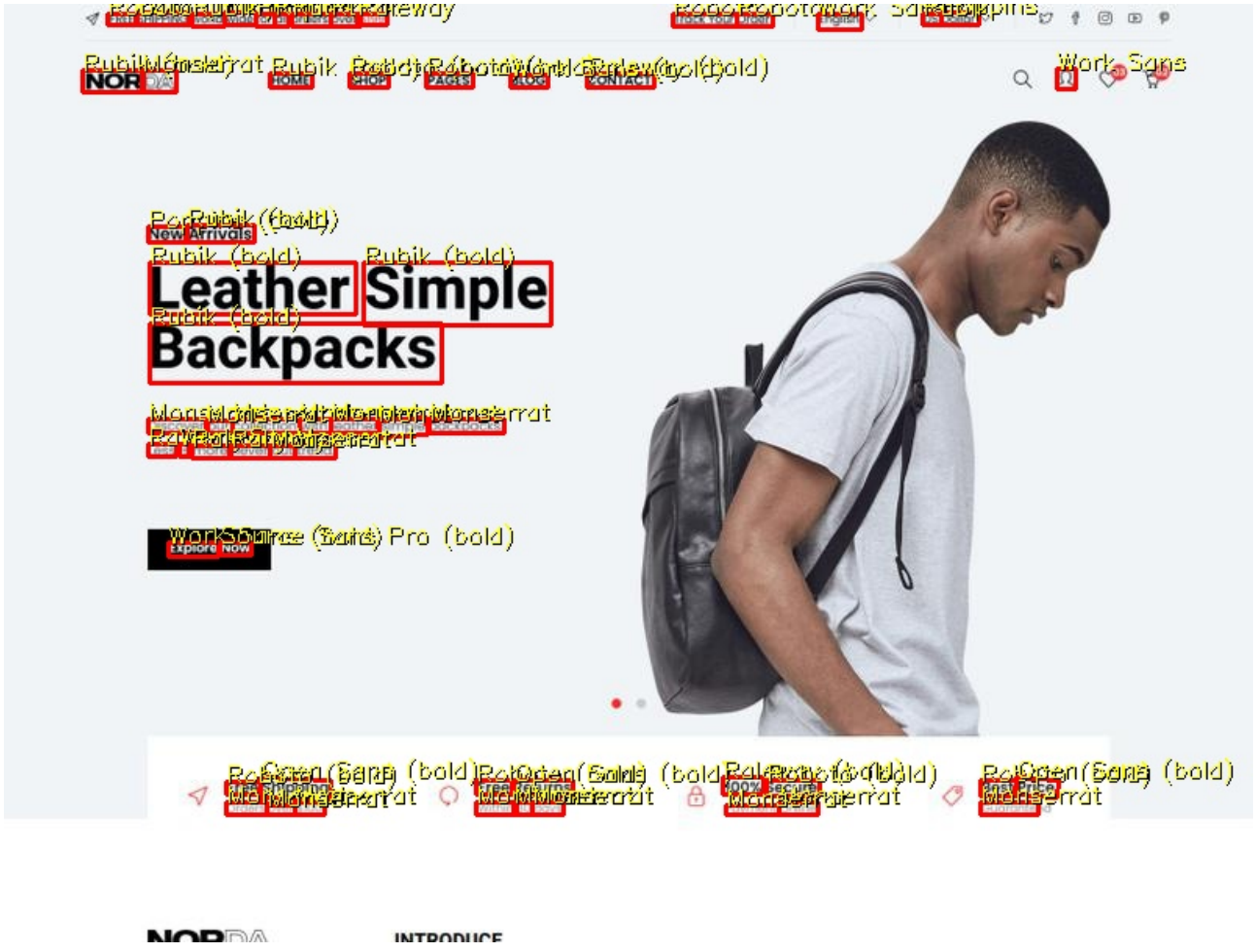
```
    "Source Sans Pro", "Source Sans Pro (bold)",
    "Work Sans", "Work Sans (bold)"
]

# run DeepFont on each bbox
for bbox in bboxes:
    top = int(min(bbox[0][1], bbox[1][1]))
    bottom = int(max(bbox[2][1], bbox[3][1]))
    left = int(min(bbox[0][0], bbox[3][0]))
    right = int(max(bbox[1][0], bbox[2][0]))

    cropped_text = origin_img[top:bottom, left:right]
    cropped_text = preprocess(cropped_text)
    cropped_text_array = img_to_array(cropped_text)
    data = []
    data.append(cropped_text_array)
    data = np.asarray(data, dtype="float") / 255.0
    y = deepfont.predict(data)
    labels.append(label_list[np.argmax(y[0], axis=-1)])


# # save score text
file_utils.saveResult(img_path, image[:, :, ::-1], polys,
    dirname=result_folder, texts=labels)
```

3 Result

The final DeepFont achieved the following metrics:

- training accuracy: 0.7652
- training loss: 0.6114
- validation accuracy: 0.7150
- validation loss: 0.8191

Running font detection on a sample image:




```
85,5,118,5,118,13,85,13,Lato (bold)
141,5,161,5,161,13,141,13,Rubik
600,5,624,5,624,13,600,13,Poppins
66,6,84,6,84,13,66,13,Roboto
120,6,141,6,141,14,120,14,Rubik
161,6,173,6,173,14,161,14,Poppins
173,6,182,6,182,14,173,14,Rubik
184,6,209,6,209,14,184,14,Rubik
209,6,228,6,228,14,209,14,Rubik
226,6,245,6,245,13,226,13,Raleway
428,6,469,6,469,14,428,14,Roboto
469,6,492,6,492,14,469,14,Roboto
```

521,6,549,6,549,16,521,16,Work Sans
588,6,600,6,600,13,588,13,Rubik
674,41,686,41,686,54,674,54,Work Sans
50,42,89,42,89,56,50,56,Rubik (bold)
88,42,110,42,110,56,88,56,Monserrat
170,44,197,44,197,53,170,53,Rubik (bold)
221,44,246,44,246,53,221,53,Roboto (bold)
270,44,300,44,300,53,270,53,Roboto (bold)
373,44,416,44,416,53,373,53,Raleway (bold)
324,45,348,45,348,53,324,53,Work Sans (bold)
92,142,116,142,116,152,92,152,Poppins (bold)
117,141,160,141,160,152,117,152,Rubik (bold)
93,165,225,165,225,198,93,198,Rubik (bold)
230,165,350,165,350,205,230,205,Rubik (bold)
93,204,280,204,280,242,93,242,Rubik (bold)
92,265,130,265,130,274,92,274,Monserrat
146,265,189,265,189,274,146,274,Monserrat
189,265,209,265,209,274,189,274,Monserrat
209,265,240,265,240,274,209,274,Monserrat
239,266,270,264,271,275,240,277,Monserrat
272,265,320,265,320,274,272,274,Monserrat
129,266,146,266,146,274,129,274,Monserrat
92,281,110,281,110,289,92,289,Raleway
112,281,118,281,118,289,112,289,Work Sans
186,281,212,281,212,290,186,290,Monserrat
120,282,144,282,144,290,120,290,Raleway
145,282,170,282,170,290,145,290,Raleway
170,282,186,282,186,290,170,290,Monserrat
105,344,137,344,137,353,105,353,Work Sans (bold)
138,344,158,344,158,352,138,352,Source Sans Pro (bold)
164,494,209,496,209,509,163,507,Open Sans (bold)
302,496,325,496,325,505,302,505,Roboto (bold)
326,496,364,496,364,506,326,506,Open Sans (bold)
460,494,486,494,486,505,460,505,Raleway (bold)
488,496,522,496,522,506,488,506,Roboto (bold)
625,496,648,496,648,506,625,506,Roboto (bold)

648,494,675,496,674,507,647,505,Open Sans (bold)
142,497,164,497,164,505,142,505,Roboto (bold)
142,510,168,510,168,518,142,518,Monserrat
186,510,205,510,205,518,186,518,Monserrat
302,510,328,510,328,518,302,518,Monserrat
326,510,338,510,338,518,326,518,Monserrat
338,510,358,510,358,518,338,518,Monserrat
496,510,521,510,521,518,496,518,Monserrat
625,510,662,510,662,518,625,518,Monserrat
168,512,186,512,186,518,168,518,Monserrat
462,512,496,512,496,518,462,518,Monserrat

References

[1] *DeepFont: Identify Your Font from An Image* by Zhangyang Wang, Jianchao Yang, Hailin Jin, Eli Shechtman, Aseem Agarwala, Jonathan Brandt, Thomas S. Huang at https://arxiv.org/abs/1507.03196

[2] https://github.com/Belval/TextRecognitionDataGenerator

[3] https://github.com/robinreni96/Font_Recognition-DeepFont

[4] https://github.com/dodanh041001/text_detection