



UNIVERSIDAD DE BUENOS AIRES
FACULTAD DE CIENCIAS EXACTAS Y NATURALES
DEPARTAMENTO DE COMPUTACIÓN

Unconstrained Text Detection in Manga

Tesis de Licenciatura en Ciencias de la Computación

Julián Del Gobbo

Directora: Rosana Matuk Herrera
Buenos Aires, 2020

Unconstrained Text Detection in Manga

The detection and recognition of unconstrained text is an open problem in research. Text in comic books has unusual styles that raise many challenges for text detection. This work aims to identify text characters at a pixel level in a comic genre with highly sophisticated text styles: Japanese manga. To overcome the lack of a manga dataset with individual character level annotations, we create our own. Most of the literature in text detection use bounding box metrics, which are unsuitable for pixel-level evaluation. Thus, we implemented special metrics to evaluate performance. Using these resources, we designed and evaluated a deep network model, outperforming current methods for text detection in manga in most metrics.

Keywords: text-segmentation, datasets and evaluation, neural-networks, Japanese-text-detection, manga

Detección de Texto sin Restricciones en Manga

La detección y reconocimiento de texto sin restricciones es un problema abierto en la investigación. El texto en comics presenta estilos inusuales que plantean muchos desafíos para su detección. Este trabajo apunta a identificar caracteres de texto a nivel de píxel en un género de comics con estilos de texto muy sofisticados: el manga Japonés. Para superar la falta de dataset de manga con anotaciones por caracter, creamos nuestro propio. La mayoría de la literatura en detección de texto utiliza métricas basadas en coordenadas de rectángulos contenedores, los cuales son inadecuados para evaluar a nivel de píxel. Entonces, implementamos métricas especiales para evaluar el desempeño. Usando estos recursos, diseñamos y evaluamos un modelo de redes neuronales profundas, superando métodos actuales de detección de texto en manga en la mayoría de las métricas.

Palabras claves: segmentación-de-texto, datasets y evaluación, redes-neuronales, detección-de-texto-japonés, manga

Agradecimientos

A Rosana, por acompañarme todo este tiempo y seguir luchando hasta que quedara lo más perfecto posible. Agradezco su fuerte compromiso y las numerosas veces que nos juntamos. Gracias a todo ese esfuerzo finalmente logramos publicar. A Daniel y Enrique, por tomarse el tiempo de leer la tesis y ser mi jurado. A exactas, por todo lo que me enseñó y aportó todos estos años. A mis compañeros con quienes compartí maravillosas cursadas. A mis amigos, con quienes compartí muchas experiencias. A mis compañeros de trabajo, que son como otra familia que me acompañó y apoyó desde antes de empezar la carrera, especialmente Marcelo que siempre fue flexible en los tiempos que necesitaba para la facultad o olimpiadas. A mis familia, por acompañarme y apoyarme en todo lo que pudieran.

CONTENTS

1. Manga	1
2. Overview	6
3. Neural Networks	7
3.1 Frameworks	7
3.2 U-Net	7
4. Related Work	9
5. Detecting, removing and inpainting as a single stage	11
5.1 Rectangle generation	11
5.2 Text generation	11
5.3 Fonts	13
5.4 Textify	13
5.5 Metrics	14
5.6 Loss function	14
5.7 Training	14
5.8 Problems	15
6. Segmentation on synthetic images	29
6.1 Danbooru2019 results	29
6.2 Manga results	32
6.3 Post-processing	37
6.4 Conclusion	44
7. Segmentation on real images	45
7.1 Dataset	45
7.2 Evaluation Metrics	48
7.3 Methodology	51
7.4 Experiments	52
7.4.1 Loss Function Selection	52
7.4.2 Model Architecture Selection	53
7.4.3 Comparison against similar works	55
7.4.4 Robustness	61
7.4.5 Improvement over synthetic data	66
8. Conclusions	67

1. MANGA

Manga is a type of Japanese comic that rose in popularity after World War 2, with works such as *Astro Boy* in 1952. Today, manga constitutes a great part of Japan industry, influencing television shows, video games, films, music, merchandise and even emojis in social media applications. According to the All Japan Magazine and Book Publisher's and Editor's Association (AJPEA), in 2018 the market totaled 441.4 billion yen (about US\$3.96 billion) while in 2019 it totaled 1.543 trillion yen (about US\$14.12 billion). Digital publishing sales made up 19.9% of the market in 2019, whereas it made up 16.1% of the market in 2018.

Comics can be characterized by its hybrid textual-visual nature [16]. Like comic books, manga are composed of four main elements: pictures, words, balloons and panels. Pictures are used to depict objects and figures. Words (including onomatopoeia) indicate character's speech and thoughts. Balloons are used to contain the words and link them to the corresponding character, with different shapes and styles to indicate whether it is speech or thought. Panels are used to structure the narrative, joining together relevant pictures, words and balloons that form a scene and also mark the continuity of time and space by the transitions between.

They are, however, different from other comics in multiple ways. Unlike American and European comics which tend to be in color, manga is usually in black and white. In manga, the flow of frames and speech go from right to left as seen in Fig. 1.1. While most comics share the same style of art and format, in manga each author tries to add his own style to it. Consequently, there is a huge diversity of text and balloon styles in unconstrained positions (Fig. 1.2) compared to comics.

Japanese is a highly complex language, with three different alphabets and thousands of text characters. It also has about 1200 different onomatopoeia, which frequently appear in manga. Japanese language is extremely sophisticated in terms of its ability to express sentiments and emotions through graphic characters. One example is they have three different onomatopoeia to express emptiness, one for the lack of sound, a second one for the lack of motion and a third one to express lack of feeling. This interaction between image and sound can affect translation [20]. Furthermore, characters often look very similar to the art in which they are embedded. These complexities make a text detection method for manga challenging to design.

United States, France and Japan had the highest influence in the origin of comics and their popularity, each with their different styles integrating diverse elements from their respective cultures. Astroboy (1952, Fig. 1.5) identifies Japan culture the same way Superman (1938, Fig. 1.4) does for United States or The adventures of Tintin (1929, Fig. 1.3) does for France. While United States and France were successful at redistributing their works internationally, manga from Japan was not widely distributed overseas. Few were published abroad and they did not have huge success. It was the internet which spurred its growth in readers worldwide. Even manga ebook sales have been increasing every year in Japan, as shown in Fig. 1.6.

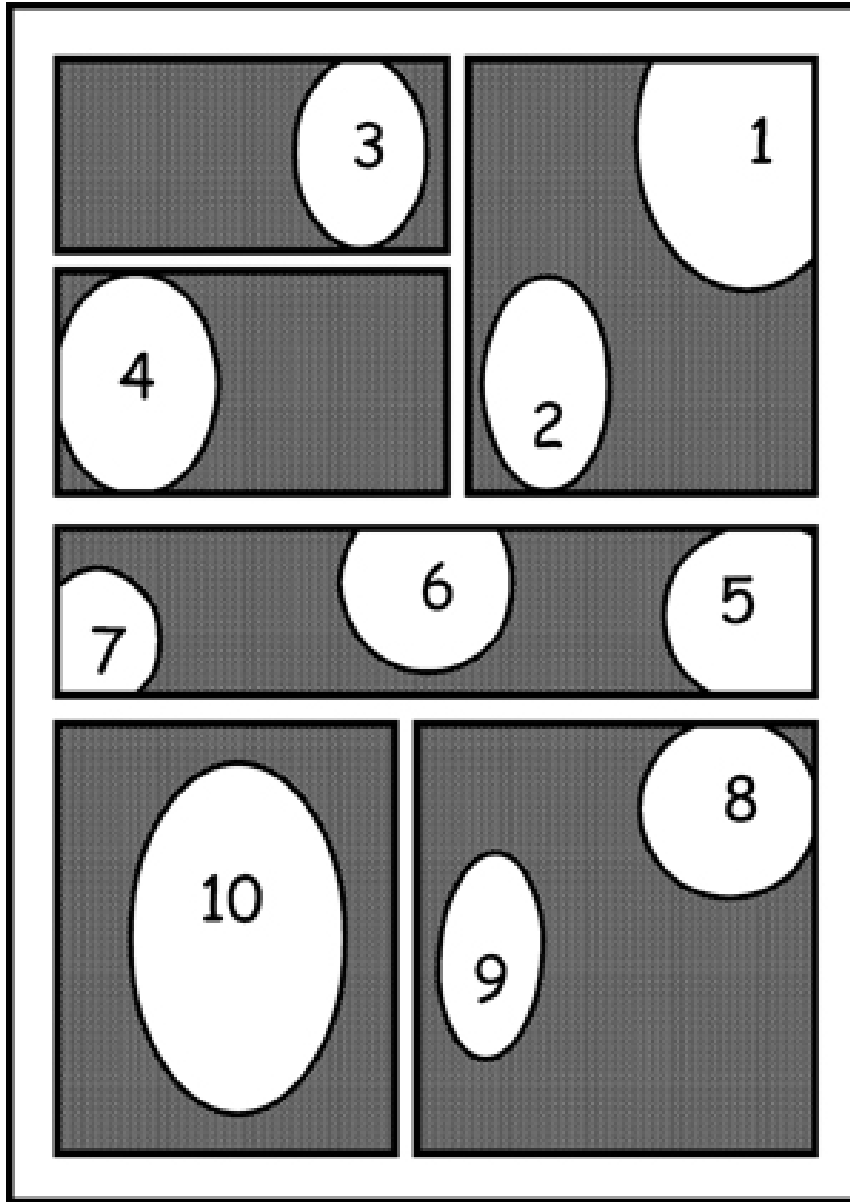


Fig. 1.1: Example illustrating the flow to read manga which is from right to left

While the few who know Japanese could read them, this is definitely not the case for most. The complexity of the Japanese language still hinders its diffusion, even if available digitally. However, many fans work on the arduous process of translating the text in the images to make it available to non Japanese speakers.

This process, known as scanlation, consists of detecting the text, erasing it, inpainting the image, and writing the translated text on the image. As it is an intricate process, the translation is usually done manually in manga, and only the most popular mangas are translated. Automating the translation would lead to solving the linguistic barrier.

In this work, we focus on the first step of the translation process: text detection.



Fig. 1.2: Pictures showing the diversity of text styles in manga. (a) The dialogue balloons could have unconstrained shapes and border styles. The text could have any style and fill pattern, and could be written inside or outside the speech balloons. Note also that the frames could have non-rectangular shapes, and the same character could be in multiple frames. (b) Example of manga extract featuring non-text inside speech bubbles. (c) The same text character can have diverse levels of transparency. (d) Text characters could have a fill pattern similar to the background. All images were extracted from the Manga109 dataset [25][26][29]: (a) and (d) “Revery Earth” ©Miyuki Yama, (b) “Everyday Oasakana-chan” ©Kuniki Yuka, (c) “Akkera Kanjinchou” ©Kobayashi Yuki

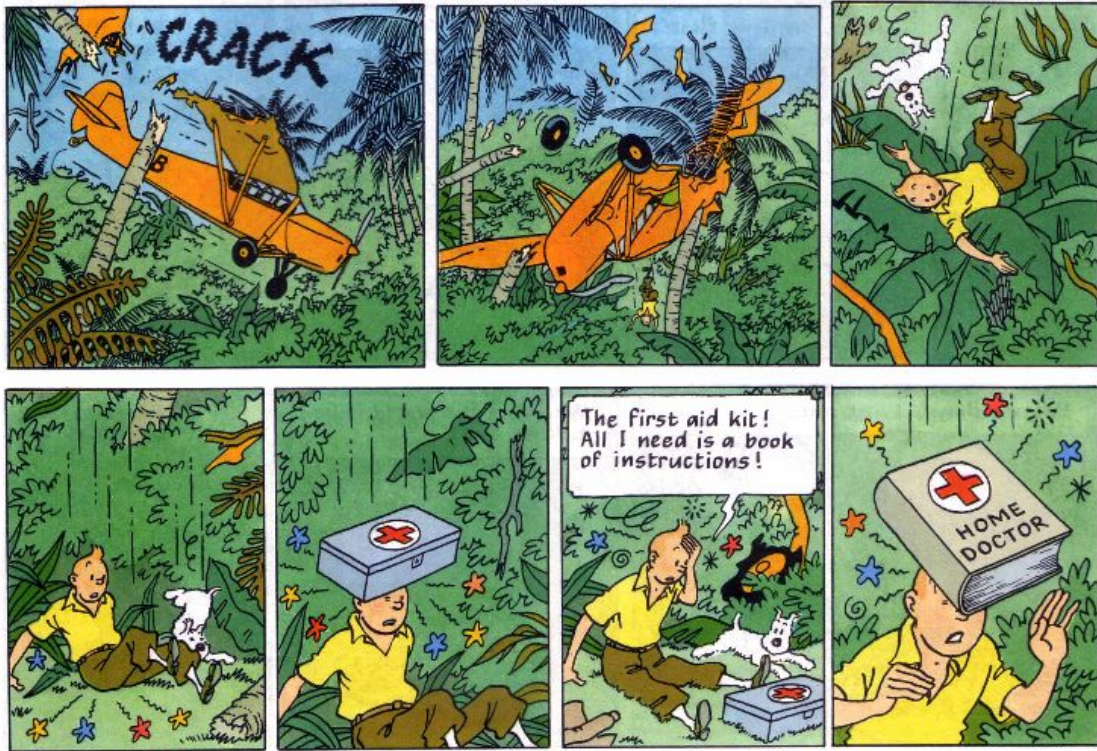


Fig. 1.3: Extract of The adventures of Tintin comic



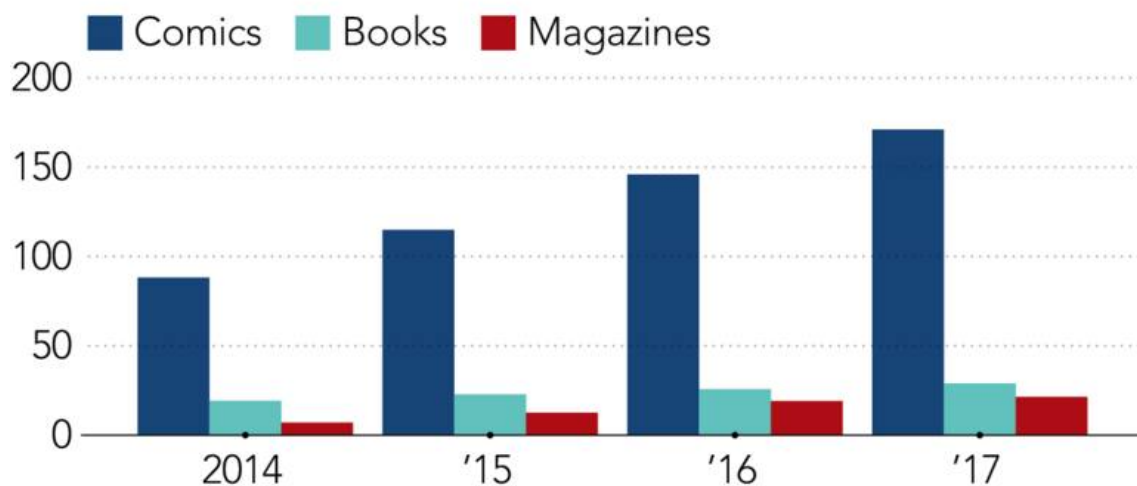
Fig. 1.4: Extract of Superman comic



Fig. 1.5: Extract of Astro Boy manga

Manga dominates Japan's growing e-book market

(in billions of yen)



Source: Research Institute for Publications

Fig. 1.6: Distribution of ebook sales in Japan over the years

2. OVERVIEW

Our text detection task is hard, it is still considered an open problem. In order to solve this, we consider the increasingly popular neural networks, briefly discussed in section 3. Along with a library in python called fastai [14] [18], we experiment with multiple ideas and provide information on our findings through this journey.

There is an abundance of deep learning related papers, with an increase in every year. On one hand, we benefit from the availability of lots of previous research but on the other hand, this makes it hard to find the specific related research which would be useful for a particular task. Recently, some tools have been made to help searching such as ai index [1].

Our problem in particular is hard to search for, as most papers referring to text detection deal with predicting what are the characters written in an image instead of text placement. Furthermore, most of the solutions that involve predicting text placement, do it in the form of bounding boxes or polygons, which are unsuitable for our case. Between both issues, it is hard to search for relevant papers. We had to filter out over 100 text detection related papers in order to find the ones that actually had some relation to our goal. As for Github, there are 2 deep learning projects that deal with detecting text in manga, but both are not peer-reviewed and one does not include the training code. These findings constitute our first contribution in this work, which are highlighted in section 4.

Another difficulty we found along this research was that there are very few datasets with pixel accurate labels of where text is placed in an image. This increases the difficulty in finding papers using a pixel level approach. One possibility would be making our own dataset, but this would be a very time consuming process so we decided to first try approaches relying on synthetic generated data.

Taking ideas from two related papers, we first try to solve the task by text removal, that is to say, taking the image with the text as input and the image without the text already in-painted as output. This is an ambitious goal and we discuss our troubles and findings in section 5.

After encountering multiple issues with the results, we decided to change our approach. Instead of text removal, we choose to generate the mask of which pixels are text, usually called text segmentation. This can be found in section 6.

Still unsatisfied with the previous results, we took it further by creating our own dataset 7.1 for this specific task, as there were none available.

Having actual data, it was now also possible to generate more accurate metrics. Which metrics are best for text detection is still an open problem, and few papers have done research about it. We discuss the issues that commonly used metrics have and our choice of metrics to use in section 7.2.

We employ this dataset and continue with text segmentation, vastly improving previous results and conducting multiple experiments. This is situated in section 7.3. Finally, we end with our conclusions in section 8.

3. NEURAL NETWORKS

Neural networks are basically a set of interconnected nodes where each node does some kind of processing to its input and then feeds forward its result to its connections. Using non linear functions as some of those nodes, usually called activation functions, many complex functions can be represented.

These functions usually have millions of parameters and it is expected that with the right value assignment, many complex tasks can be solved with a performance similar or greater than humans. Finding this set of the right values depends on training, which is done based on examples of inputs and their respective output, thus not needing to code explicitly how to solve the required task.

With the help of a loss function, which should penalize based on how wrong or right is the output, and an optimizer, which decides how to change the values of the parameters based on this loss, the network is trained and its parameters are increasingly guided to a presumably better set of values suited to the particular task.

Over recent years, neural networks have grown increasingly popular, both in research and non research communities. One of the main reasons for this is the recent increase in performance, in many tasks the current state of the art involve using a neural network. This is especially true for images, where extracting features is hard and very case specific.

Current text detection state of the art also involves neural networks, which is why we decided to approach our problem with it.

3.1 Frameworks

The most popular deep learning frameworks are TensorFlow and PyTorch. However, they are quite low level which is why there are many libraries built on top of them like Keras with TensorFlow. Fastai [14] [18] is a library built on top of PyTorch which includes many state of the art research into its implementation. This allows easily training models which already come with these features and best practices. Furthermore, it provides enough flexibility to customize all the training process, very useful for research. This is the library we decided to use for this work.

3.2 U-Net

U-Net [32] (Fig. 3.1) is a neural network originally designed for medical image segmentation, which became very popular after researchers discovered that it was also achieving state of the art in many other image related tasks. Over the time, many variations have been proposed but the original idea remains the same: in the first part (encoder) the amount of features decrease over further layers while

on the second part (decoder) they increase in similar fashion until a similarly sized segmentation map is obtained as output.

The key factor of its success is the cross connections that go from the encoder layers to their respective decoder layers, allowing the model to retain the original information that could have been lost over the down-sampling in order to provide the up-sampling.

A variation of this architecture, used in this work, is provided by the fastai library called Dynamic U-Net [15] which automatically generates the decoder part based on the encoder.

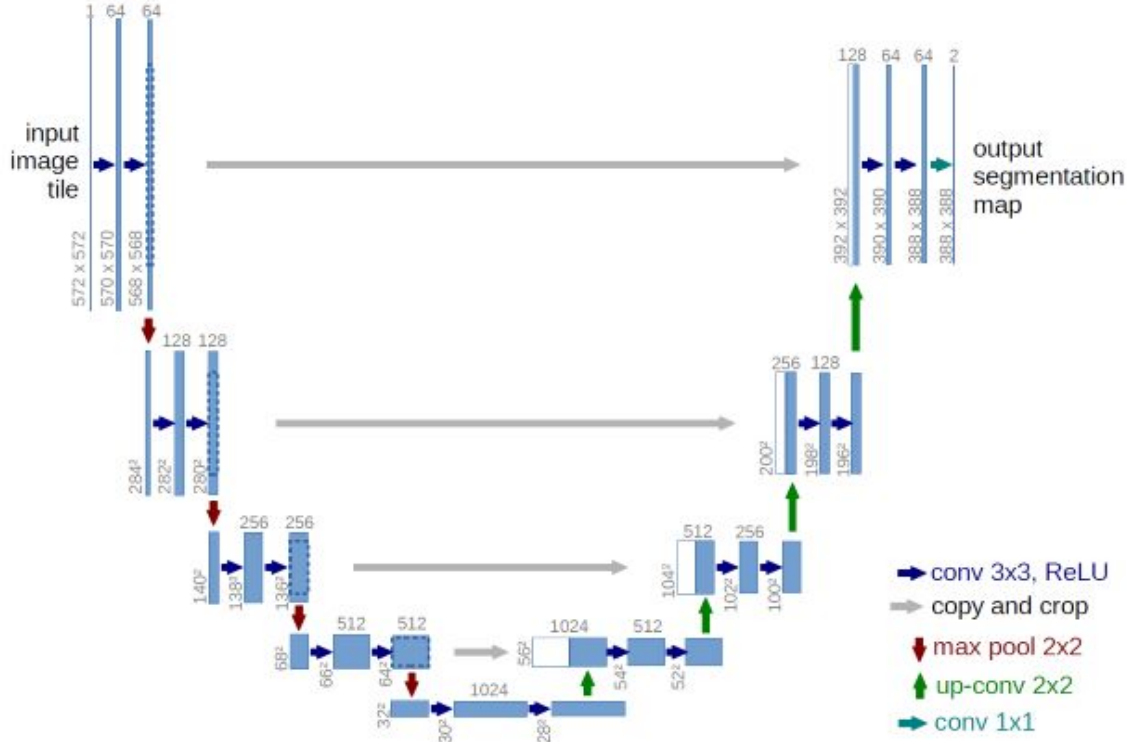


Fig. 3.1: Example of U-Net architecture

4. RELATED WORK

Speech balloon detection Several works have studied speech balloon detection in comics [31, 28, 24, 13]. While this could be used to detect speech balloons and then consider its insides as text, the problem is that text in manga is not always inside speech balloons. Furthermore, there are a few cases where not everything inside the balloon is text (Fig. 1.2b).

Bounding box detection Other works in text detection in manga, such as Ogawa *et al.*[29] and Yanagisawa *et al.*[38], have focused on text bounding box detection of multiple objects, including text. Wei-Ta Chu and Chih-Chi Yu have also worked on bounding box detection of text [11].

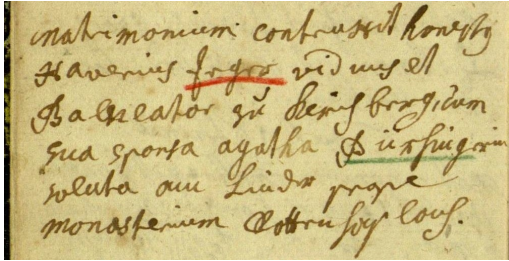
Without restricting to manga or comics, there are many works every year that keep improving either bounding box or polygon text detection, one of the most recent ones being Wang *et al.*[36]. However, methods trained with rigid word-level bounding boxes exhibit limitations in representing the text region for unconstrained texts. Recently, Baek *et al.* proposed a method (CRAFT) [5] to detect unconstrained text in scene images. By exploring each character and affinity between characters, they generate non-rigid word-level bounding boxes.

Pixel-level text segmentation There are very few works that do pixel-level segmentation of characters, as there are few datasets available with pixel-level ground truth. One of such works is from Bonechi *et al.*[7]. As numerous datasets provide bounding-box level annotations for text detection, the authors obtained pixel-level text masks for scene images from the available bounding-boxes exploiting a weakly supervised algorithm. However, a dataset with annotated bounding boxes should be provided, and the bounding box approach is not suitable for unconstrained text. Some few works that make pixel text segmentation in manga could be found on GitHub. One is called “Text Segmentation and Image Inpainting” by `yu45020` [39] and the other “SickZil-Machine” by `KUR-creative` [22]. Both attempt to generate a text mask in the first step via image segmentation and inpainting with such mask as a second step. In *SickZil-Machine*, the author created pixel-level text masks of the **Manga109** dataset, but has not publicly released the labeled dataset. The author neither released the source code of the method but has provided an executable program to run it. In `yu45020`’s work, the source code has been released, but the dataset used for training is unclear.

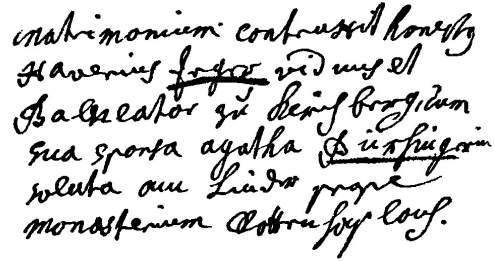
We are fully aware that there is a long history of text segmentation and image binarization in the document analysis community related to engineering drawings, maps, letters and more. However, we consider these datasets, where most of the image is text along with a few lines or figures, far more simple than one of manga, which features a lot more context, wide variety of shapes and styles. As an example, in DIBCO 2018 (Document Image Binarization Competition), the dataset is only of 10 images similar to Fig. 4.1.

Text erasers Some authors have explored pixel-level text erasers for scene im-

ages. Nakamura *et al.*[27] is one of the first to address this issue using deep neural networks. Newer works (EnsNet) by Zhang *et al.*[41] and (MTRNet) by Tursun *et al.*[35] make use of conditional generative adversarial networks.



(a)



(b)

Fig. 4.1: In (a), an image from DIBCO 2018 dataset featuring hand written text. In (b), its ground truth

5. DETECTING, REMOVING AND INPAINTING AS A SINGLE STAGE

Our first try was to do something similar to EnsNet and MTRNet: erase the text and do the inpainting from the image in a single neural network. In order to train this, both an image with text and an inpainted image without the text is needed. This is hard to come by, be it manga or any kind of image. Thus we proceeded by generating synthetic data.

Danbooru2019 [3] is a large scale dataset of anime/manga style images along with tags and other kind of metadata. We downloaded a subset from those and with another text detection software, removed those that already had text. We downloaded several fonts that had Japanese symbols in them, then randomly generated non overlapping rectangles, and in those rectangles randomly generated text. In this way, we had the original image that would be the target of the network and the modified image with text in it as the input, to try to make the network learn to remove Japanese text.

5.1 Rectangle generation

In order to generate non overlapping rectangles, the following approach was used: randomly obtain top left rectangle corner, randomly choose a width and height and if it did not intersect with any of the previous rectangles, add it to our set. If it does intersect, give it a chance to reduce half the width or half the height in order to fit. This process was repeated until either the amount of requested rectangles was reached or a maximum amount of retries was made.

5.2 Text generation

To generate a random text of n characters, we simply randomly choose characters n times from the unicode code point ranges that include japanese characters, along with some special symbols and english letters.

Given a rectangle width and height, we need to make sure the text will fit inside the rectangle. In order to do this, we must make sure that if drawing the text with the given font would overflow in width, we either send the rest to a new line and continue processing if there is still enough height or we just cut the text there. This problem is know as text wrapping.

Many ways of doing this can be found online, but most were very inefficient or handled different kind of wrapping such as no more than x characters per row regardless of pixel width, taking care not to split words.

An algorithm that provides the exact solution for pixels is:

```
def text_wrap(text, font, max_width, max_height):
```

```

lines = []
i, j, hei = 0, 0, 0
while j <= len(text):
    w = font.getsize(text[i:j + 1])[0]
    if w > max_width or j == len(text):
        hei += font.getsize(text[i:j])[1]
        if hei <= max_height and j > i:
            lines.append(text[i:j])
            i = j
            if j == len(text):
                break
    else:
        break
else:
    j += 1
return lines

```

While this makes sure text always stays within the rectangle, it is very slow. The calls to `getsize` are the ones that take most time, so our goal is trying to use as few as possible. After trying out several options, we ended up with the following version which is 5 to 10 times faster in most cases:

```

def text_wrap(text, font, max_width, max_height):
    estimate = (max_width // font.getsize('a')[0])
    lines = []
    i, j, hei = 0, 0, 0
    while i < len(text):
        i = j
        j = min(len(text), i + estimate)
        width = font.getsize(text[i:j])[0]
        while j < len(text) and width <= max_width:
            width += font.getsize(text[j])[0]
            j += 1
        while width > max_width and j > i:
            j -= 1
            width -= font.getsize(text[j])[0]
        hei += font.getsize(text[i:j])[1]
        if hei > max_height:
            break
        if len(text[i:j]):
            lines.append(text[i:j])
    return lines

```

5.3 Fonts

While it is easy to download many fonts, its not as easy to know if a font supports a certain character. In all the tools or code we found, it was always wrong. There are several font formats, but in most there is a character that is defined as the default when a font can't draw a character as it does not support it. Most seem to use the same character for this (0x1d). While we randomly choose characters out of 21275, it may happen that the font we are using only supports a few hundred.

This leads to a lot of characters being drawn as the default missing one on the images as seen in Fig. 5.1. This is a big problem as we are wasting a lot of learning potential for the network. We believe that the online tools and code are probably working fine, but the font instead of properly having all the unsupported characters as missing, it actually defines the mapping to the missing character. In the end, we were able to design a method to discard these characters, although it may be discarding more than necessary.

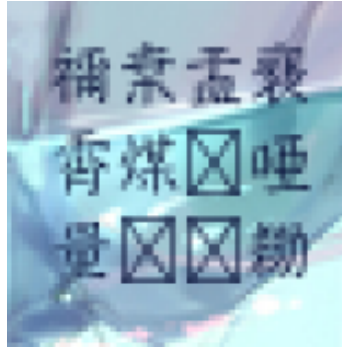


Fig. 5.1: Drawing text over an image, but 3 of the characters are not supported by the font

5.4 Textify

We called textify to our method of adding text to an image. While this changed many times over time, we show the pseudocode of the final version:

```
padding = randint(4, 10)
with 50% chance:
    generate a single rectangle that covers at least 66% of the image
else:
    generate between 7 and 15 rectangles with previous method

for each rectangle:
    get random text size, mostly regular sizes.
    estimate how many characters will fit in the rectangle
    pick a random font, weighted by amount of characters it supports
    generate random text, with length = 50% to 100% of the estimation
```

```

split text into lines as defined by text_wrap
randomly choose the color text will have, mostly black
randomly choose border color, mostly white
with 30% chance, decide if text will be rotated in random angle
with 10% chance, decide if the rectangle will also be drawn
with 10% chance, decide if the rectangle will be semi transparent
with 5% chance, decide if border will be added
with 50% chance, change the rectangle to an ellipse
Finally draw the text in the image, applying all the decisions

```

with 20% chance, convert image to black and white

All these random transformations attempt to make the synthetic data cover as many cases as possible, to force the network focus on text.

5.5 Metrics

Standard L1 or L2 sum over the pixels are not a good measure to compare results in many image to image tasks, such as inpainting. This is still an open problem and many ways to compare image similarity are designed every year. From those, we chose the most popular: SSIM (Structural Similarity Index) [37] and PSNR (Peak Signal-to-noise Ratio).

5.6 Loss function

As L1 or L2 are not very useful as metric, they are also not very useful as loss function. Instead we use a feature loss function, which considers features obtained from vgg16 model, a similar approach to [19]:

$$loss(i, t) = L_1(i, t) + \sum_{j=0}^{j=n} L_1(f_j(i), f_j(t)) * w_j + L_1(gm(f_j(i)), gm(f_j(t))) * w_j * 5e3 \quad (5.1)$$

where i is the input image, t is the target image, f_j is the j th vgg16 model features of the n selected layers, w_j is a predefined weight and gm is the gram matrix.

5.7 Training

Initially, we used 5000 images cropped to 64x64 to train the U-net with a resnet18 encoder. Normalizing dataset and setting a sigmoid as layer to force output to be in the range -1 to 1 helped get better results. Parameters like self attention or blur did not seem to have any noticeable effect.

Trying resnet34 encoder, didn't get noticeable improvements either. With resnet101 it did, but took much longer to train. Using variations of U-net, U-net wide didn't

improve while U-net deep did, but took much more memory and took longer to train.

In the end, we decided to keep the U-net with the resnet18 encoder as it took much less to train, leading to faster testing different settings, and the results weren't much worse.

5.8 Problems

As for the PSNR metric, the higher the better. Most experiments lead to 28-29 score, and it was difficult to observe any difference. Anything over 29, however, was noticeable better. Even in those cases, predictions still suffered from multiple artifacts such as blurring (Figs. 5.2 and 5.3) or text not completely removed (Figs. 5.4 and 5.5).



Fig. 5.2: Input, output and ground truth patches in each column. Edges are lost, but more importantly even in an “easy” case of mostly white and black text, patched zone remains very blurry



Fig. 5.3: Input, output and ground truth patches in each column. Blurring is even worse in color patches

Hypothesis

To test if the model was not capable of reconstructing image, adding text was removed to try training the image identity. Initial efforts seemed to reach similar

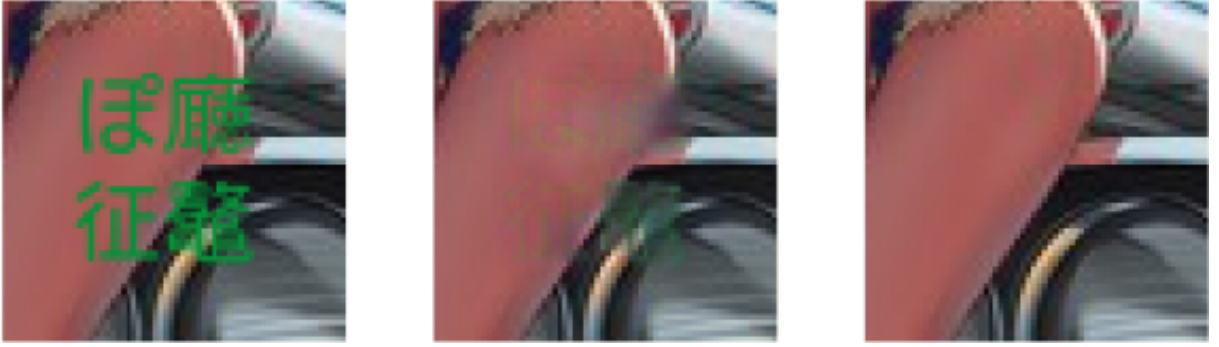


Fig. 5.4: Input, output and ground truth patches in each column. Text not completely removed



Fig. 5.5: Input, output and ground truth patches in each column. Even black text in color patches is not completely removed

metrics as with text, but with reaching up to 32 PSNR. After trying other parameters, it was able to learn it completely: 49 PSNR and 99.9 SSIM.

Given that the model was able to learn the identity and that networks with more parameters like resnet101 did not help solve these issues either, it didn't seem to be a problem of model not having the capacity to learn it.

Two likely suspects to explain it were that the images were too small and 64x64 was not enough for the model to learn finer details or that the dataset had too few images. It seemed unlikely to be a problem of not seeing enough examples of text, because they were randomly placed and randomly generated, giving a very high possible amount of examples. Even if it was only 5000 images, by using patches of 64x64 of the original 512x512 image, each image could also provide different patches, changing even more the amount of possible examples.

Testing

The most likely suspect seemed to be the patches being too small, as during a small text with 64x128, better results were already obtained. To try this out, an experiment was done training model in different stages, progressively increasing the size. For this, 2 parameters were set for each stage: the minimum size of the patch

and the maximum, making variable size possible under different batches. These values applied for both height and width, making rectangle patches also possible.

The configuration of these parameters were: start with 64x64 patches, then 64x128, then 128x128, then 128x256 and finally 256x256. Instead of the 5000 images, the full dataset (25000 images) was used.

The first noticeable difference was that with just the first stage, the results were remarkably better: not only did it reach better metrics (31.5 PSNR and 0.969 SSIM), but also the erasing and inpainting improved remarkably as seen in Fig. 5.6. Some colors were still off and a bit blurry but it was much better than before and even the edge was reconstructed. Given that the same parameters were used, it seemed to be a case of just needing more training data.



Fig. 5.6: Results after first stage of training the 64x64 patches

With the progressive resizing, the metrics worsened a bit but the results were still very good, after the (128, 256) stage it had 30.83 PSNR and 0.960 SSIM. This means that the bigger the image, the more likely to have lower PSNR. As seen in Fig. 5.7, blurring is much less noticeable and text is completely removed, even in color examples.

By the (256, 256) stage, as the image size was bigger, instead of just putting a single bunch of text centered in the patch, several portions of text were placed over the image, thus making the amount of text lower but including different examples



Fig. 5.7: Input, output and ground truth patches in each column. Results after 128x256 stage

(fonts, color, font size) in a single image.

First epoch of this stage already had 37.05 PSNR and 0.991 SSIM, and by the final epoch it reached 38.56 PSNR and 0.993 SSIM. This seems to be a great improvement, but the change is mostly caused by changing the amount of text in the image. With fewer pixels with text, less pixels need to be inpainted so those metrics now give perfect results for a higher percentage of pixels. At first glance, results seem perfect as seen in Fig. 5.8.

However, when predicting over our actual images from manga, we noticed that the performance was actually much worse in this stage than earlier stages. Furthermore, the predictions seem to be the best at the second stage of 64x128 (Figs. 5.9, 5.10, 5.11, 5.11, 5.13, 5.14, 5.15, 5.16, 5.17, and 5.18). A possible explanation is that it ended up over-fitting to the style of synthetic text we generated, which is different from the one in real manga.



Fig. 5.8: Input, output and ground truth patches in each column. Results after 256x256 stage



Fig. 5.9: Prediction of 256x256 stage



Fig. 5.10: Prediction of 256x256 stage



Fig. 5.11: Prediction of 128x256 stage



Fig. 5.12: Prediction of 128x256 stage

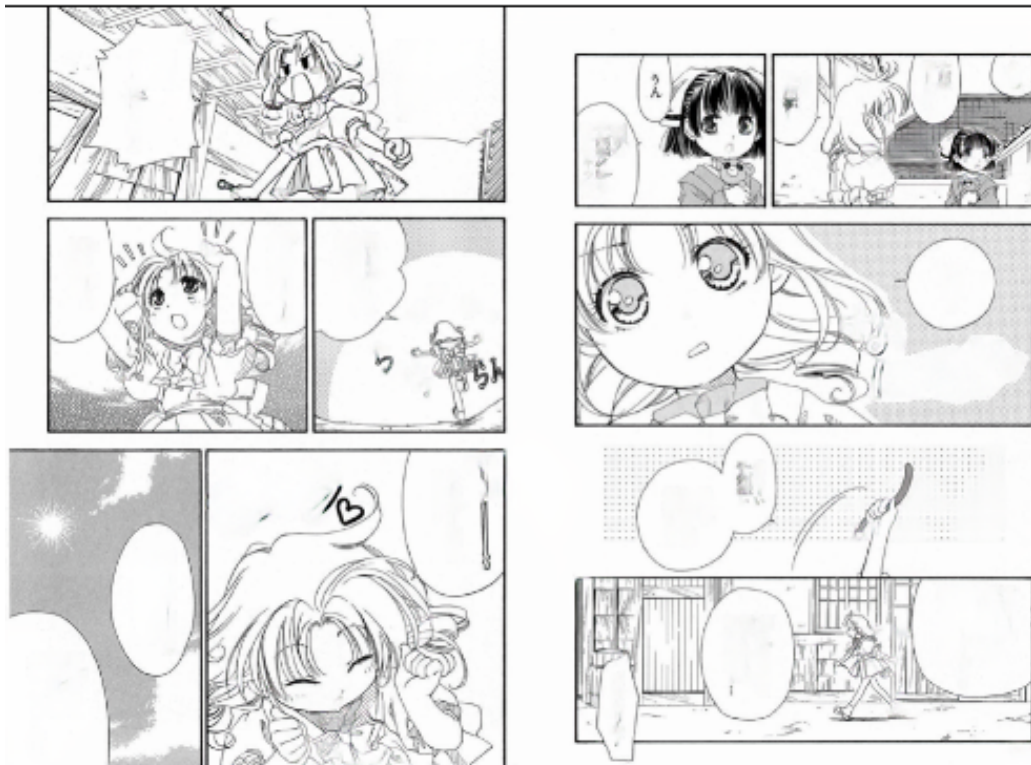


Fig. 5.13: Prediction of 128x128 stage



Fig. 5.14: Prediction of 128x128 stage

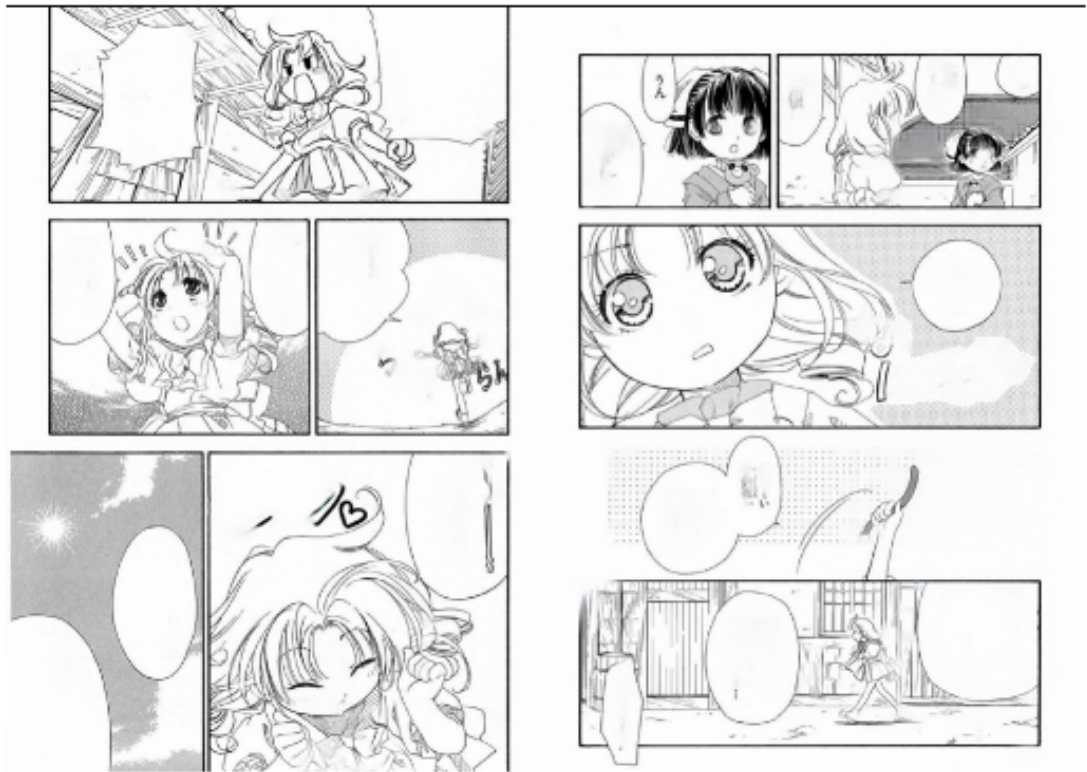


Fig. 5.15: Prediction of 64x128 stage



Fig. 5.16: Prediction of 64x128 stage

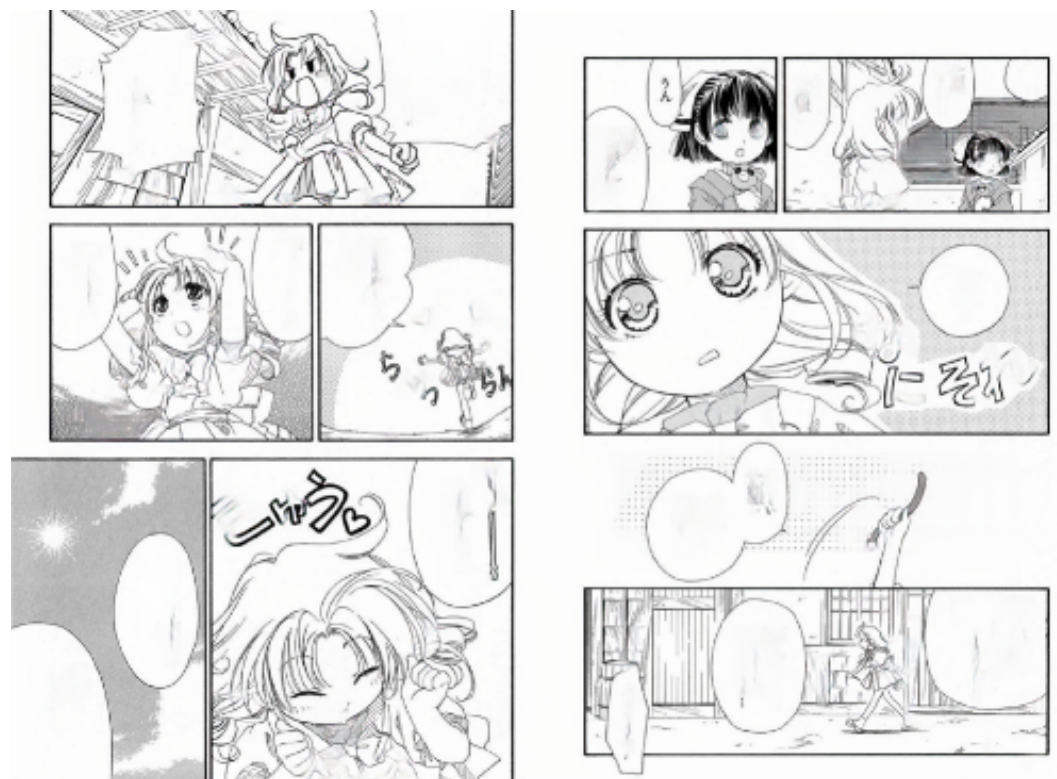


Fig. 5.17: Prediction of 64x64 stage



Fig. 5.18: Prediction of 64x64 stage

Resizing

Another important factor that changed a lot the predictions over the manga images was the resizing of the image before the prediction (Figs. 5.19 and 5.20). Instead of just resizing by stretching, using black padding on the relevant dimension worked better.



Fig. 5.19: Prediction of 64x128 stage with image resized to 1170x1654



Fig. 5.20: Prediction of 64x128 stage with image resized to 1600x800

Limitations

After many tries and improvements, good results were obtained but they still several issues. Firstly, as seen in Figs. 5.21 and 5.22, although it did a great job of erasing text from speech bubbles and even worked on the letter which not only has rotated text but also perspective, it erases more than necessary (has false positives) such as the face of the dialogue. Secondly, this issue is even more noticeable with small circles, as it tends to erase them as seen in Figs. 5.23 and 5.24.

Many variations were tried: changing learning rates, number of epochs, size of the patches, the sigmoid range, the amount of fonts used. However, these issues still persisted. This lead us to change our approach.



Fig. 5.21: Original image

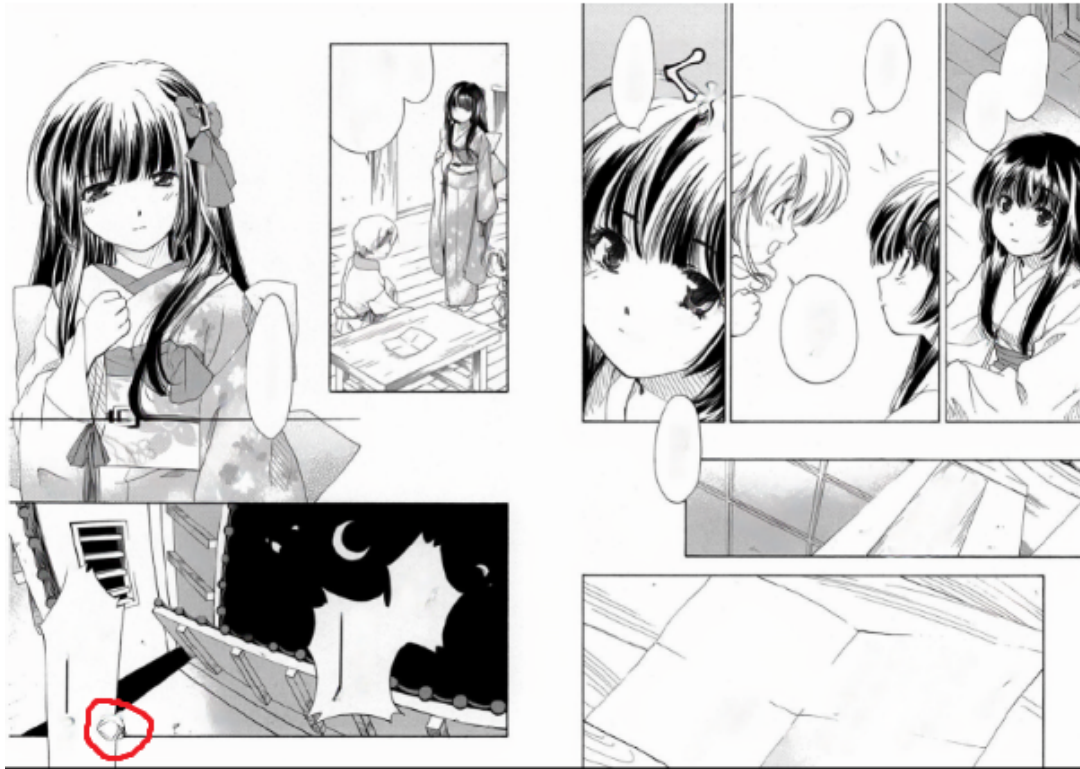


Fig. 5.22: Prediction. Circled in red, an example of non text that suffered from removal

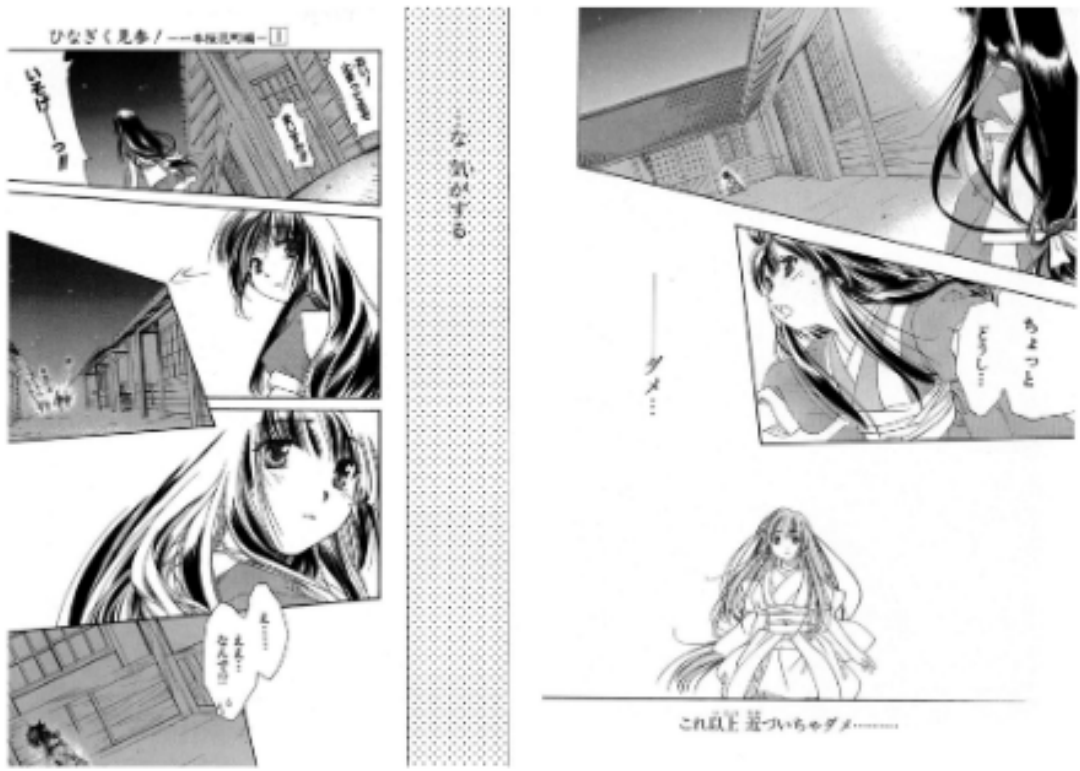


Fig. 5.23: Original image

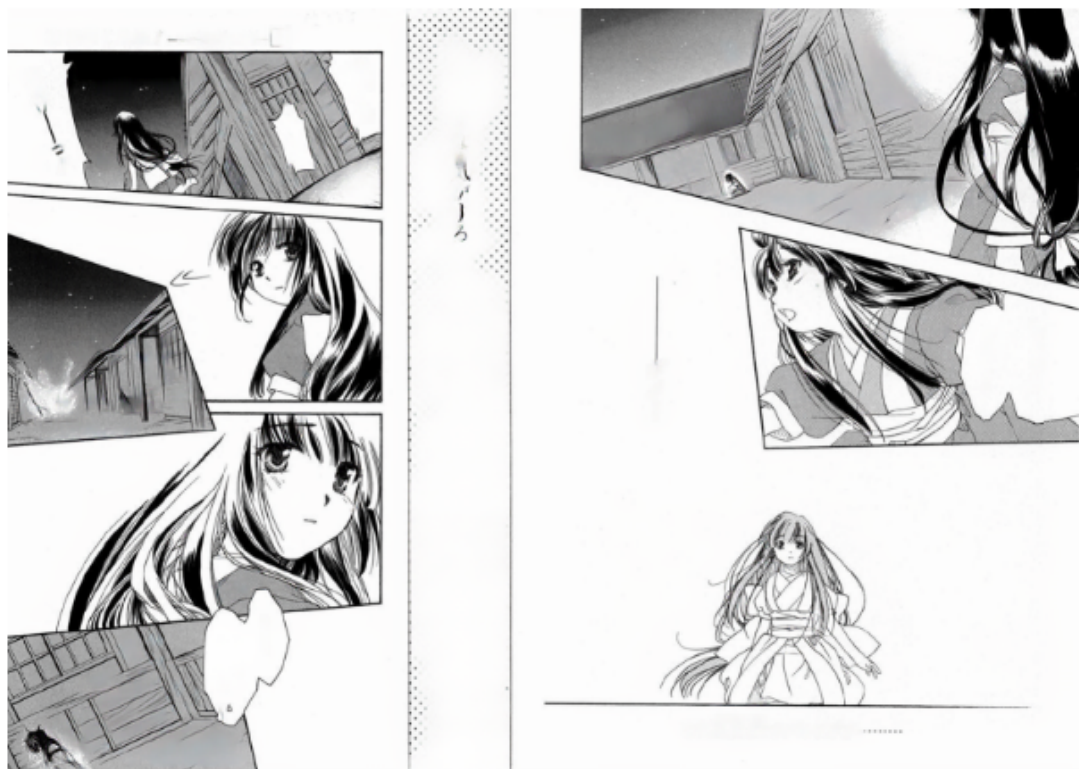


Fig. 5.24: Prediction. Many small circles were removed, even when they were not text

6. SEGMENTATION ON SYNTHETIC IMAGES

Even if we had good results with some images, they were still not perfect. This means that in order for it to be useful, someone would need to fix the mistakes. This not only meant erasing and inpainting manually the text that was not removed but also recovering the parts that were mistakenly removed. This seemed too troublesome to do, and wouldn't be much different than processing the whole image manually.

In order to let users fix the mistakes more easily, an additional stage would need to be introduced. An alternative approach then is, instead of doing the removal and inpainting of text as a single task, first detect the text and then inpaint it, with 2 separate networks. As several inpainting works already exist, we decided to focus on the first part: text detection.

On one side, many works in text detection in comics have taken a balloon detection approach. However, in manga, the text and balloons are also part of the artwork. Thus, balloons could have a multiplicity of shapes and styles. Besides, the text can be outside the dialogue balloons (Figs. 1.2a, 1.2c, 1.2d), or inside the balloon there could be non-text contents (Fig. 1.2b), making a balloon detection approach unsuitable for this task. On the other side, most previous works in text detection have taken a box detection approach. However, manga contains texts that are deformed, extremely large, or are drawn on the cartoon characters, which are hard to identify with a single bounding box (Figs. 1.2a, 1.2c, 1.2d).

Thus, we decide to make text segmentation at a pixel level, identifying pixels as either text or background.

6.1 Danbooru2019 results

In order to quickly test if segmentation was a viable approach, we first tried training on an existent dataset for text segmentation called icdar13 [21]. As seen in Fig. 6.1 results were pretty accurate, so we decided segmentation was indeed a feasible approach.

We then modified our code to treat the added text as the target instead of the image without it represented as a binary map. The loss function was changed to dice loss and the results over the synthetic data generated with Danbooru2019 were good as seen in Figs. 6.2 and 6.3.



Fig. 6.1: Results of icdar2013 training. First column represents ground thruth, while second column is the prediction



Fig. 6.2: Results of Danbooru2019 training over 128x128 patches. First column represents ground thruth, while second column is the prediction

epoch	train_loss	valid_loss	foreground_acc	dice	dice	time
0	0.254470	0.253774	0.942157	0.915085	0.847596	02:38
1	0.196011	0.198124	0.947008	0.935162	0.881160	02:25
2	0.162619	0.172046	0.945141	0.943555	0.895776	02:26
3	0.146529	0.165525	0.970679	0.947659	0.902907	02:26
4	0.134953	0.145231	0.962405	0.953044	0.912434	02:25
5	0.127741	0.136808	0.967173	0.956004	0.917557	02:25
6	0.121932	0.138618	0.977735	0.956057	0.917692	02:25
7	0.118304	0.126791	0.965214	0.958851	0.922634	02:25
8	0.116917	0.125171	0.964855	0.959390	0.923585	02:25
9	0.114856	0.124107	0.965743	0.959839	0.924384	02:25

Fig. 6.3: Table of metrics and loss over 10 epochs of training Danbooru2019 dataset over 128x128 patches. Second dice metric is with iou (intersection over union) parameter in true

6.2 Manga results

However, when testing with the actual manga images, results were much worse: while most of the text was accurately covered, it had too many false positives. This can be observed in Fig. 6.4. Changing the threshold to consider a pixel as text to 0.95 instead of the 0.5 default however, greatly improved the result as seen in Fig. 6.5

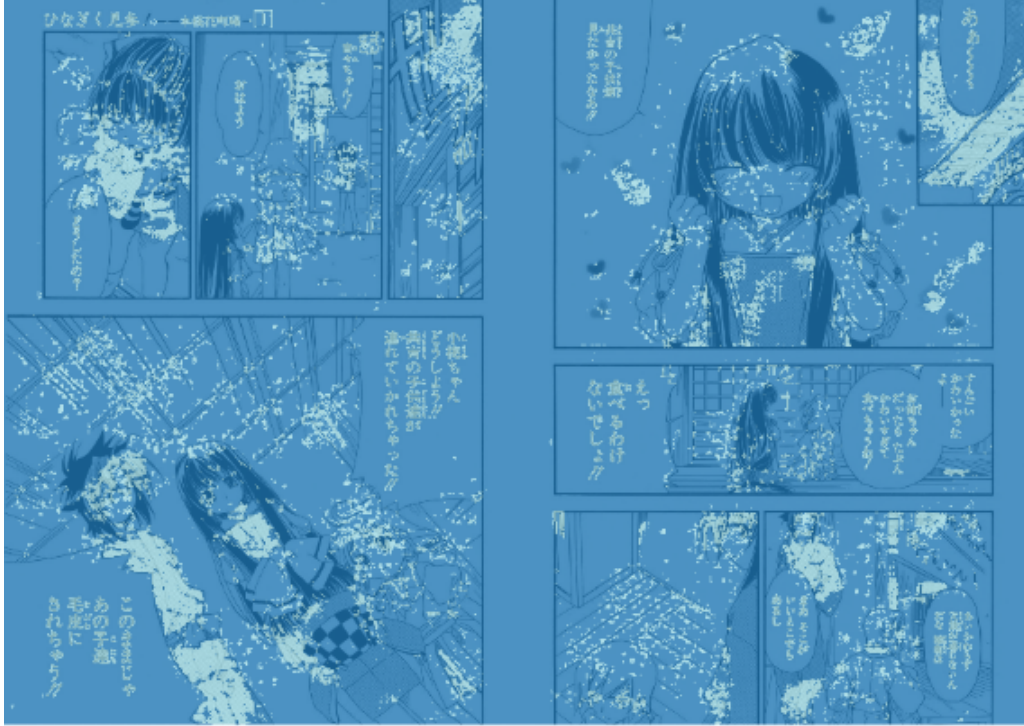


Fig. 6.4: Result of Danbooru2019 training over 128x128 patches over manga image with 0.5 as threshold

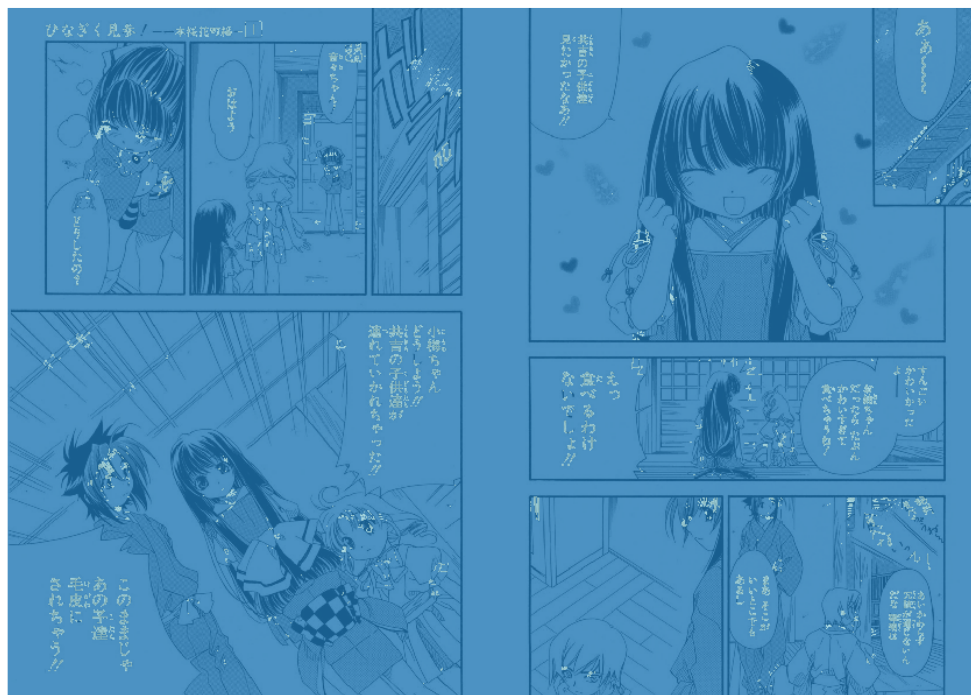


Fig. 6.5: Result of Danbooru2019 training over 128x128 patches over manga image with 0.95 as threshold

Further refinement of hyper-parameters such as max learning rate, number of epochs, loss functions (dice, binary cross entropy, focal loss) and self-attention were made and results were slightly improved as seen in Figs. 6.6, 6.7, 6.8, 6.9 and 6.10.

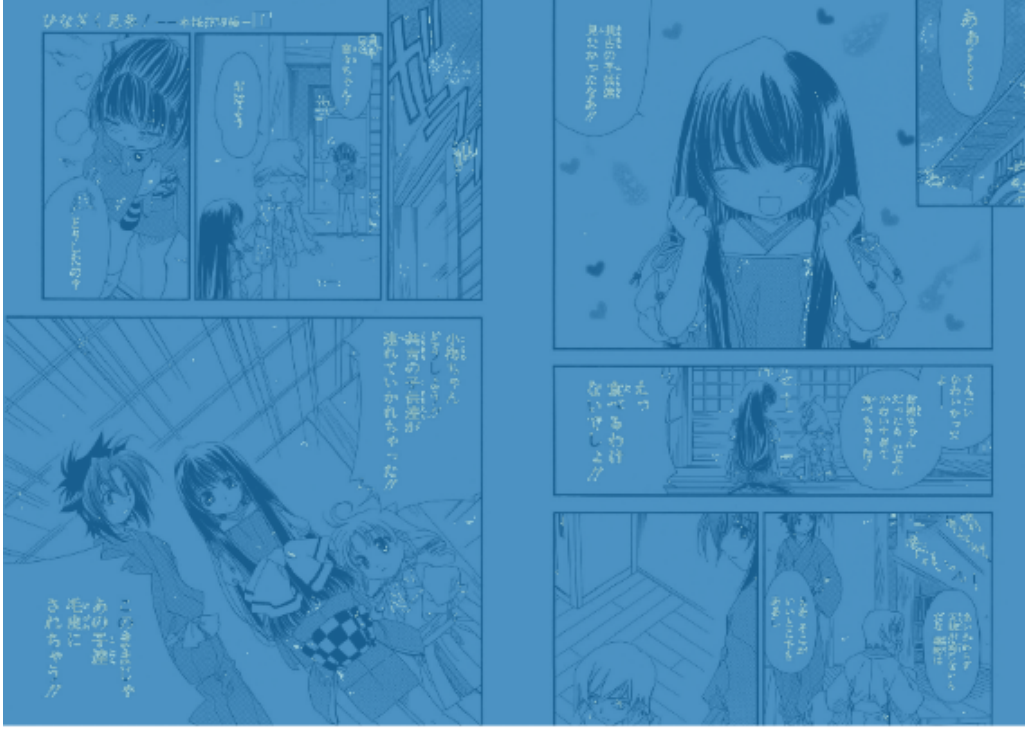


Fig. 6.6: Result of Danbooru2019 refinement with 0.95 as threshold

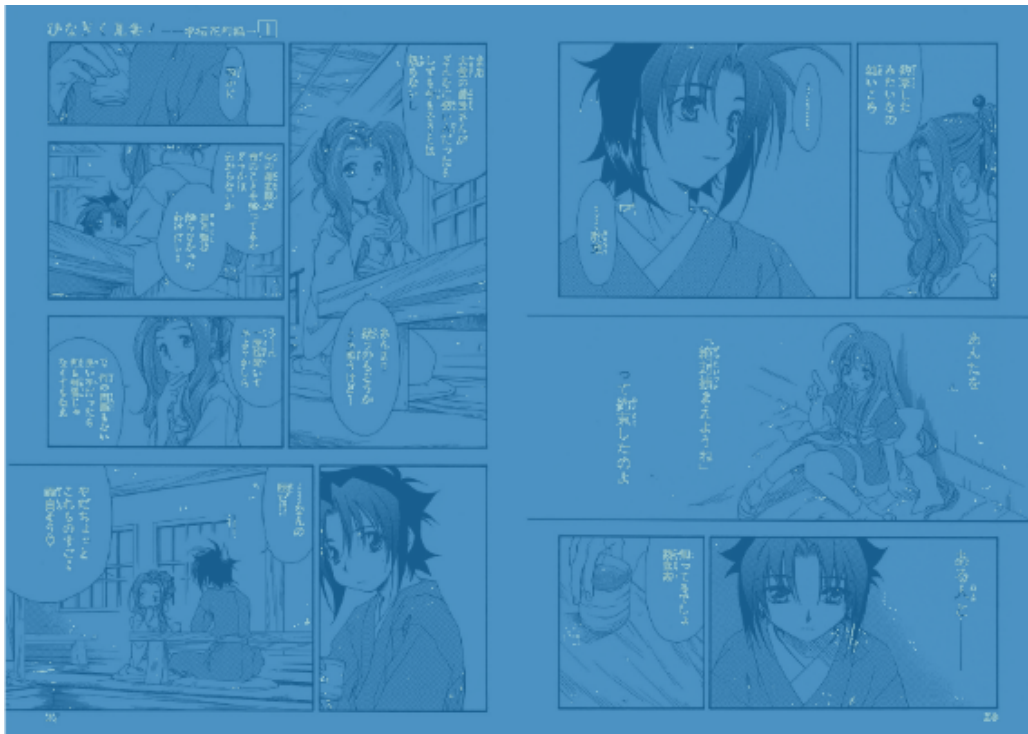


Fig. 6.7: Result of Danbooru2019 refinement with 0.95 as threshold

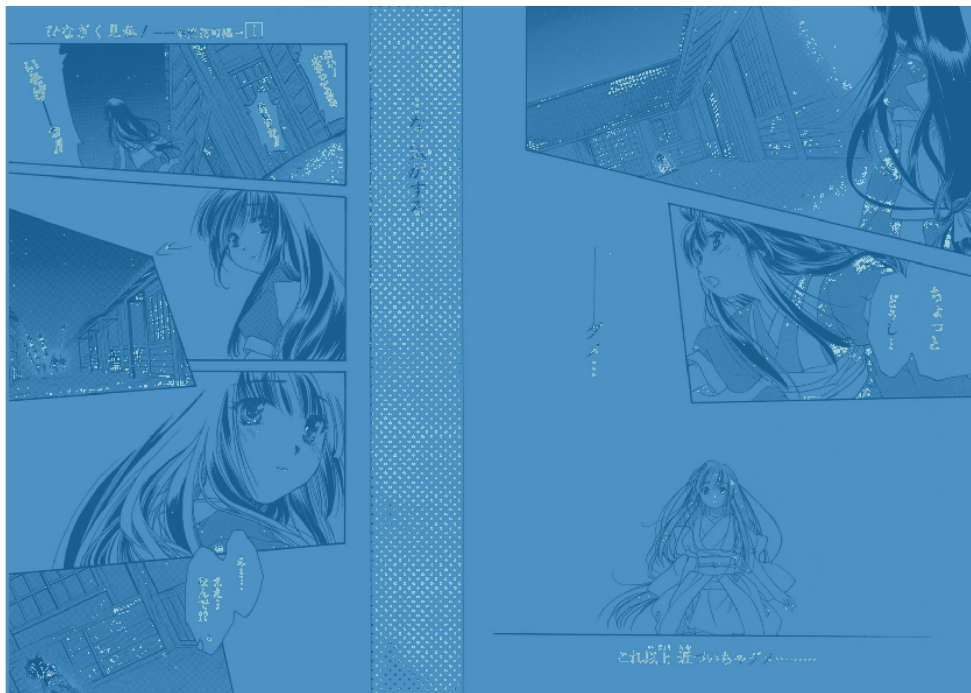


Fig. 6.8: Result of Danbooru2019 refinement with 0.95 as threshold

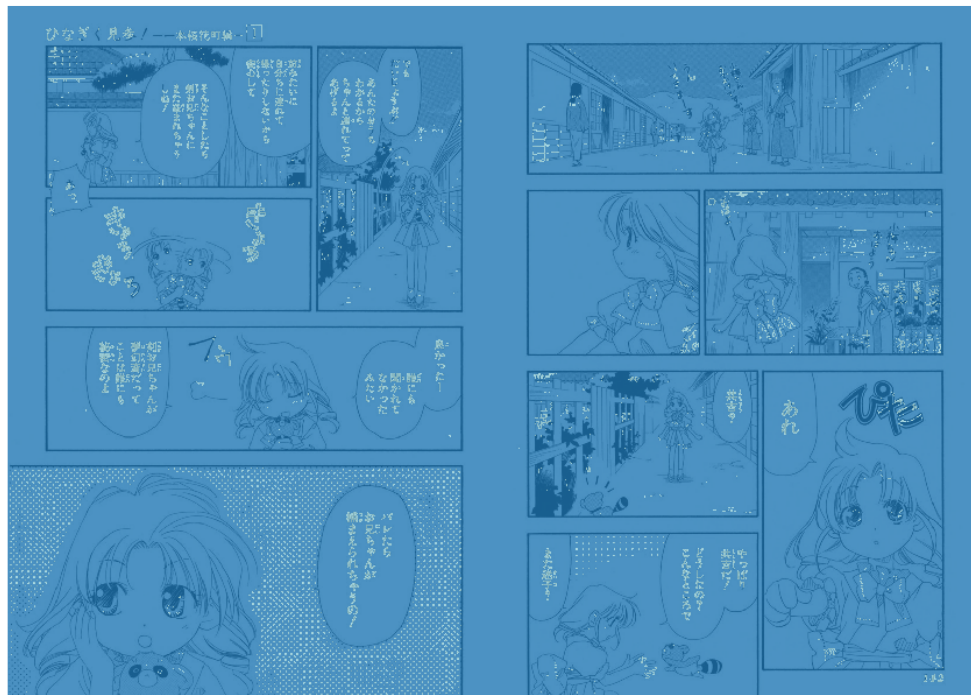


Fig. 6.9: Result of Danbooru2019 refinement with 0.95 as threshold

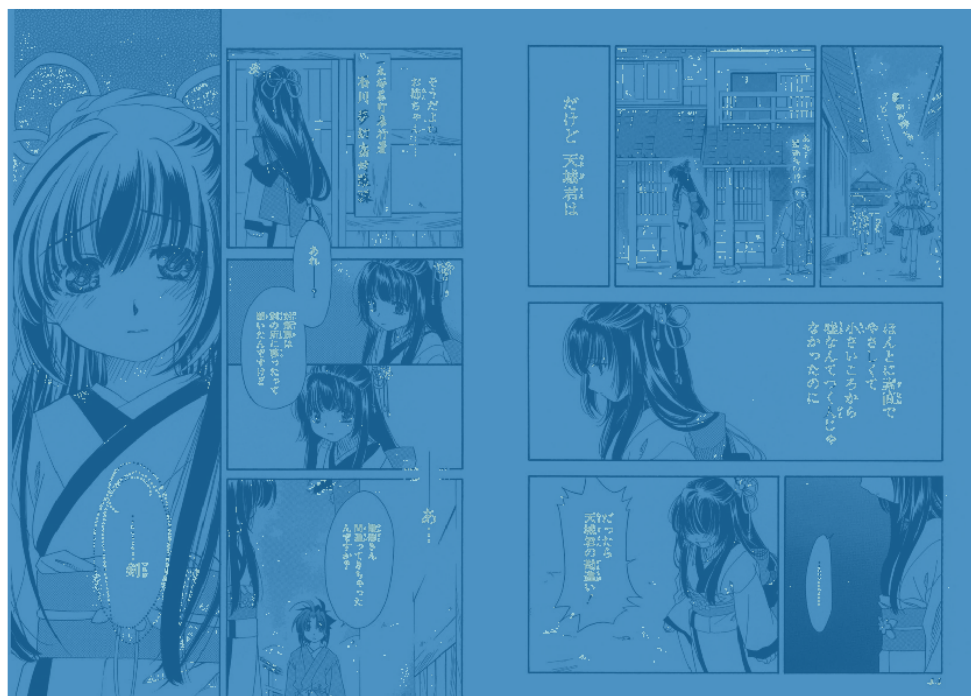


Fig. 6.10: Result of Danbooru2019 refinement with 0.95 as threshold

Training with bigger patches such as 256x256 or 512x512 lead to worse results. While there were less false positives of little dots, less characters were correctly segmented. Another test was applying some pre-processing (image binarization) to the input image in order to make it easier to learn as seen in Fig. 6.11 but it also lead to worse results, which means the network takes advantage of more information (gray scale values).

Other ideas tried were: adding text over manga patches instead of Danbooru2019, deep unet and wide unet variations, filtering the Danbooru2019 images to remove the ones with text, mish activation function. None improved the results, so we decided to stop trying to improve the network.

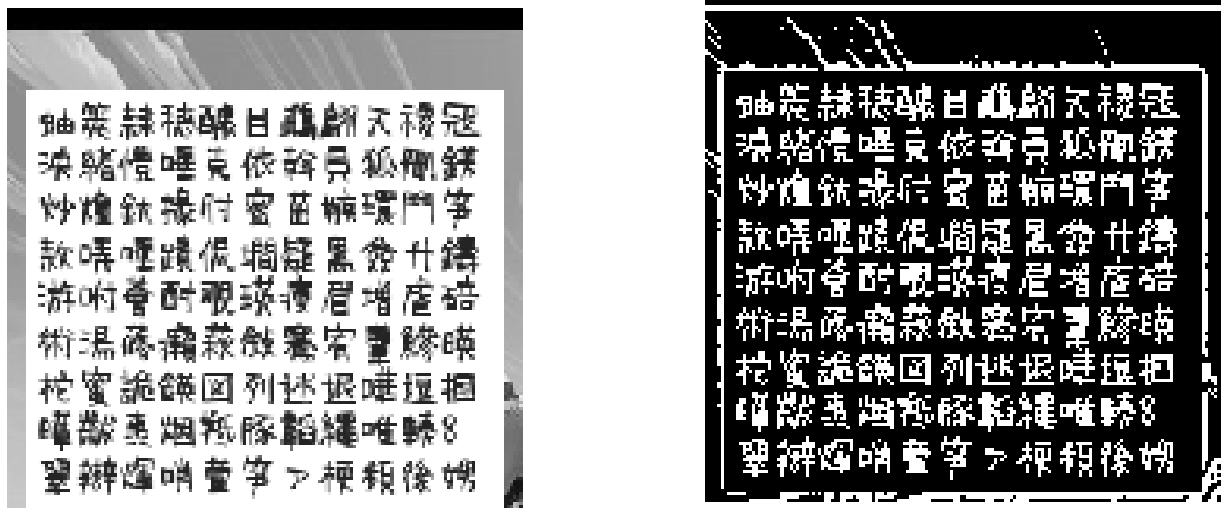


Fig. 6.11: Image binarization example

6.3 Post-processing

What could still improve however, was the post-processing. We tried multiple threshold methods from cv2 and scikit image libraries, along with some noise cleaning methods. The goal was to remove as much as possible the small false positive dots without interfering with the correct text results.

The algorithm can be seen in listing 6.1. Basically, we assume that contours with high area (≥ 100) are correct predictions. For every contour with less than 100 area, we need to decide if it should be discarded or not. To do that, contours retrieved by cv2 in close order to it are checked to see if they are correct and are close to the current contour. If that is true, the current contour with small area is considered to be correct. The algorithm keeps running until no more small contours are considered to be good.

This allows small contours near letters to be kept, as sometimes a letter was not fully predicted and is broken into several contours. It also allows actual dots that tend to be near other letters be kept as well. All the rest, most of which are usually

noise, get removed. An example can be seen in Figs. 6.12, 6.13 and 6.14. The main problem is when all letters of the dialogue are dots, no good contour is near them and they get discarded.

Listing 6.1: Noise Removal Algorithm

```

im = mask.permute(1,2,0).numpy() * 255
original = im.copy()
simple = cv2.CHAIN_APPROX_SIMPLE
cnts, _ = cv2.findContours(im.astype('uint8'), cv2.RETR_LIST, simple)
goods = [cv2.contourArea(c) >= 100 for c in cnts]
rects = [cv2.boundingRect(c) for c in cnts]
changed = True
ran = range(0, len(cnts))
while changed:
    changed = False
    for c, good, idx, rect in zip(cnts, goods, ran, rects):
        x,y,w,h = rect
        x, y = x + w / 2, y + h / 2
        if good:
            continue
        for a in range(max(idx - 15, 0), min(idx + 15, len(cnts))):
            if a != idx and goods[a]:
                x2, y2, w2, h2 = rects[a]
                x2, y2, = x2 + w2 / 2, y2 + h2 / 2
                closeInY = abs(y2 - y) < (h + h2) / 2 + 10
                closeInX = abs(x2 - x) < (w + w2) / 2 + 20

                if closeInY and closeInX:
                    good = goods[idx] = True
                    changed = True
                    break

for c, good, _, _ in zip(cnts, goods, ran, rects):
    if not good:
        cv2.drawContours(im, [c], 0, (0, 0, 0), -1)
ImageSegment(tensor(im).permute(2,0,1))

```

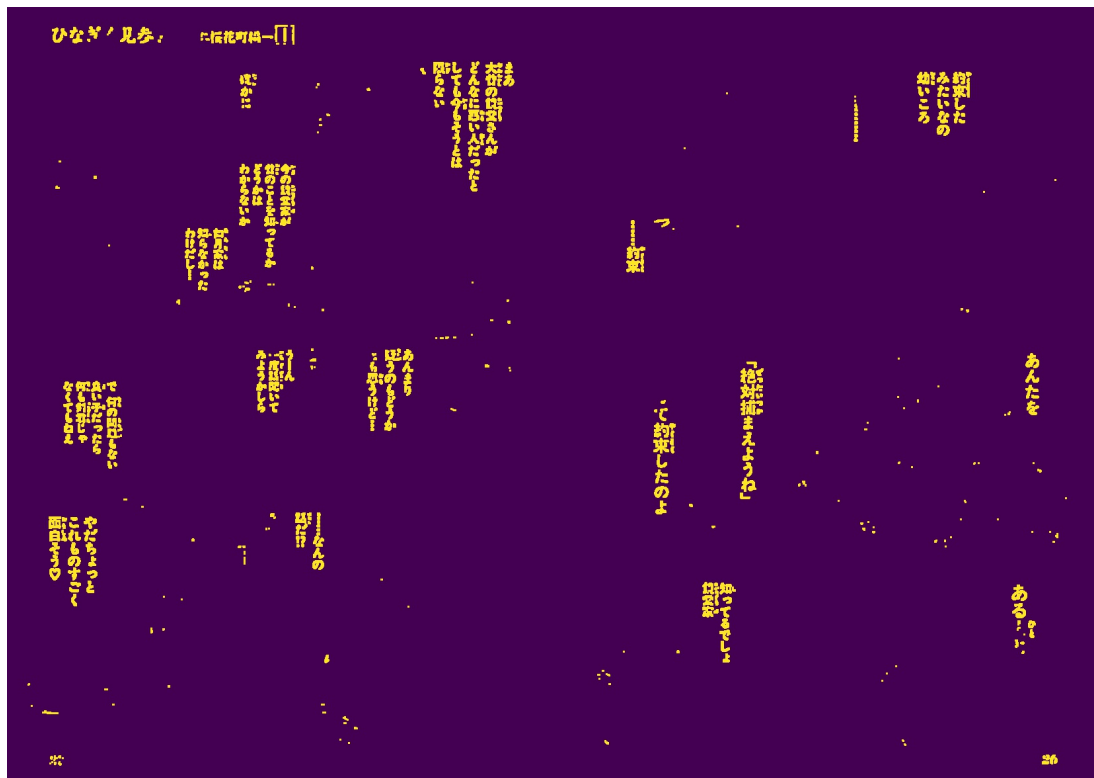
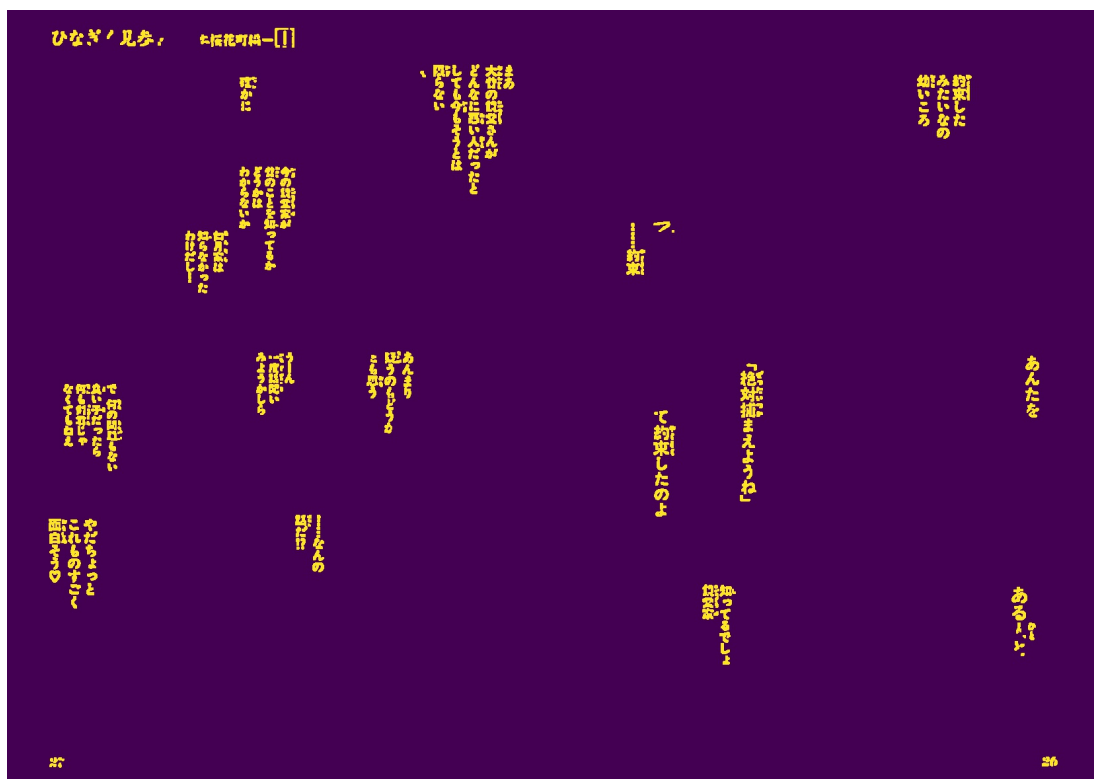


Fig. 6.12: Prediction before applying noise removal



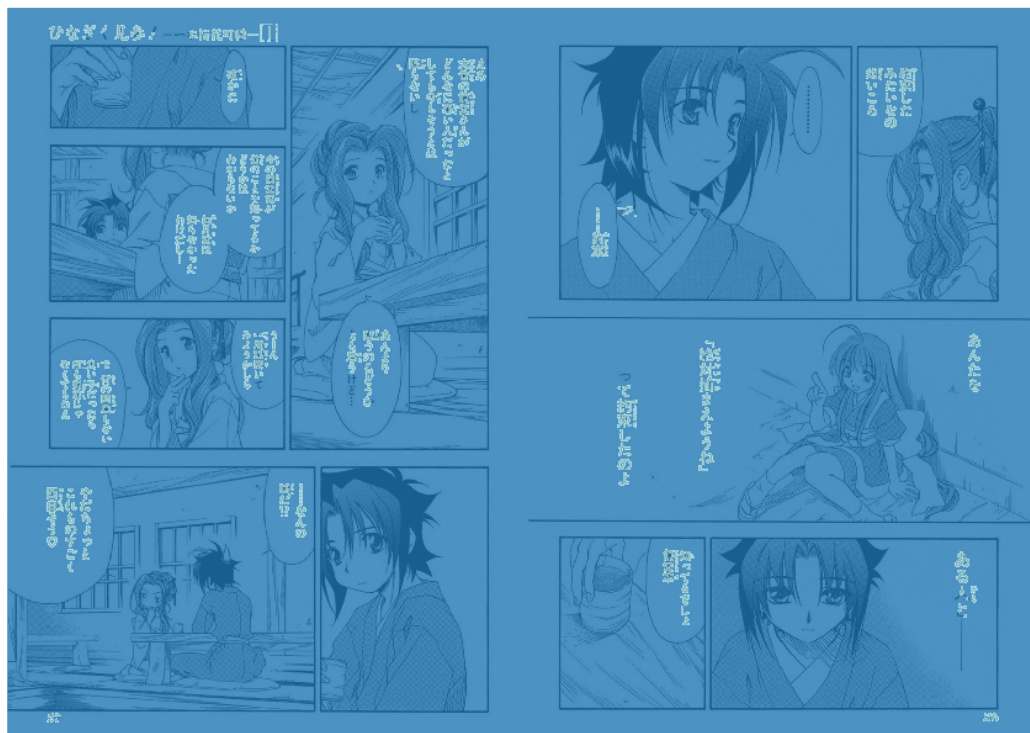


Fig. 6.14: Original image with prediction after applying noise removal

Noise removal helps removing mistakes in the prediction, with a small risk of removing objects that were correct. Another problem to try to solve is to fix partial predictions (letters not completely covered). This means trying to detect cases where a letter was not completely covered by the prediction and then expanding that prediction to the whole letter.

Algorithm can be seen in listing 6.2. Image is first converted to gray-scale and then applied an adaptive threshold method to separate into black and white. Then we get the contours, hoping that each letter will have its own contour. We iterate the contours, get the bounding box and further refine the threshold by applying it to the bounding box crop of the gray-scale image. Then we get the connected components of that region. If they are less than 10, we iterate the components. If the component has an area of more than 3 pixels and the intersection of that area with the prediction is greater than 10% of the area (we predicted at least 10% of the component to be text), we consider the whole component to be text.

Listing 6.2: Fix Partial Detection Algorithm

```
mask = mask.astype('uint8')
gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
adapt = cv2.ADAPTIVE_THRESH_GAUSSIAN_C
bin = cv2.THRESH_BINARY
aprox = cv2.CHAIN_APPROX_SIMPLE
cvThres = cv2.adaptiveThreshold
thresh = cvThres(gray, 255, adapt, bin, 15, 30)
cnts, _ = cv2.findContours(thresh, cv2.RETR_LIST, aprox)
im3 = np.zeros(thresh.shape, np.uint8)

for c in cnts:
    x,y,w,h = cv2.boundingRect(c)
    thresh = cvThres(gray[y:y+h, x:x+w], 255, adapt, bin, 15, 30)
    thresh = cv2.bitwise_not(thresh)
    ret, markers = cv2.connectedComponents(thresh, connectivity=8)
    if ret < 10:
        for label in range(1, ret):
            m = markers == label
            intersec = (m & mask[y:y+h, x:x+w] > 0).sum()
            if m.sum() > 3 and intersec > m.sum() * 0.1:
                im3[y:y+h, x:x+w][m] = 255
```

An example can be seen in Figs. 6.15, 6.16 and 6.17.

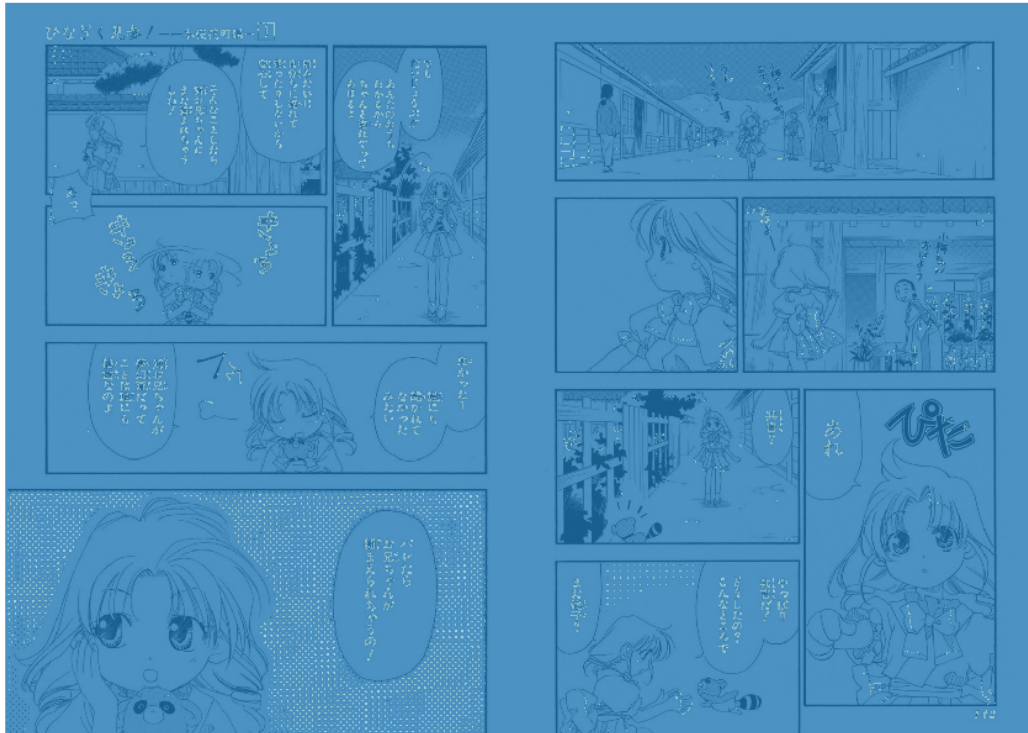


Fig. 6.15: Raw Prediction

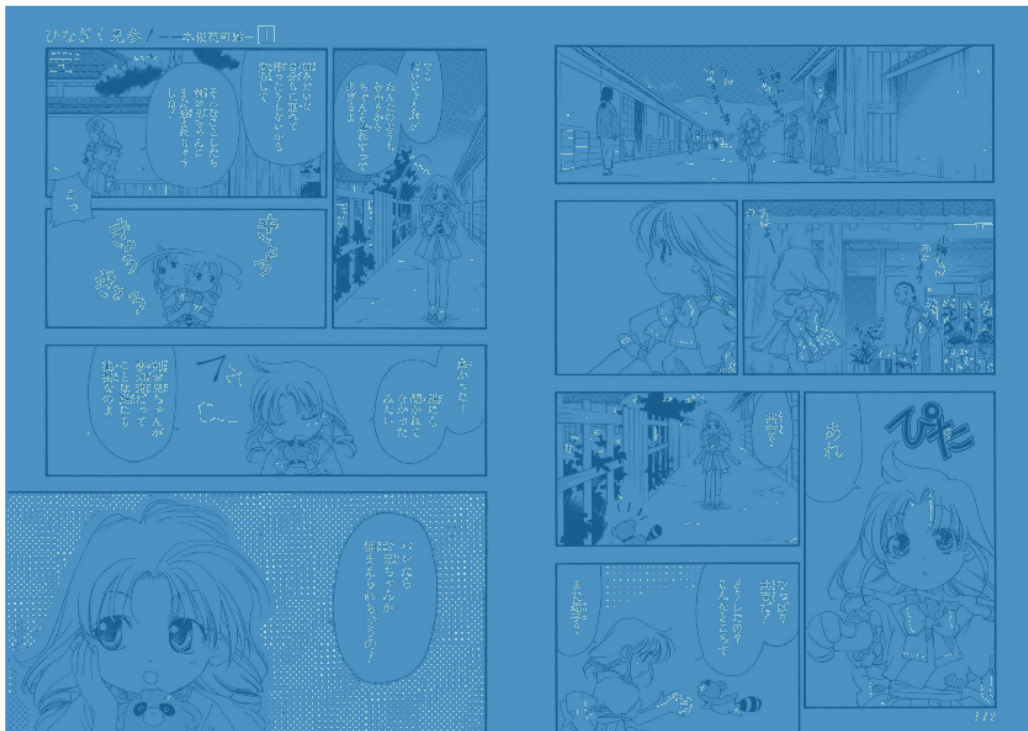


Fig. 6.16: Prediction after expanding partial detections

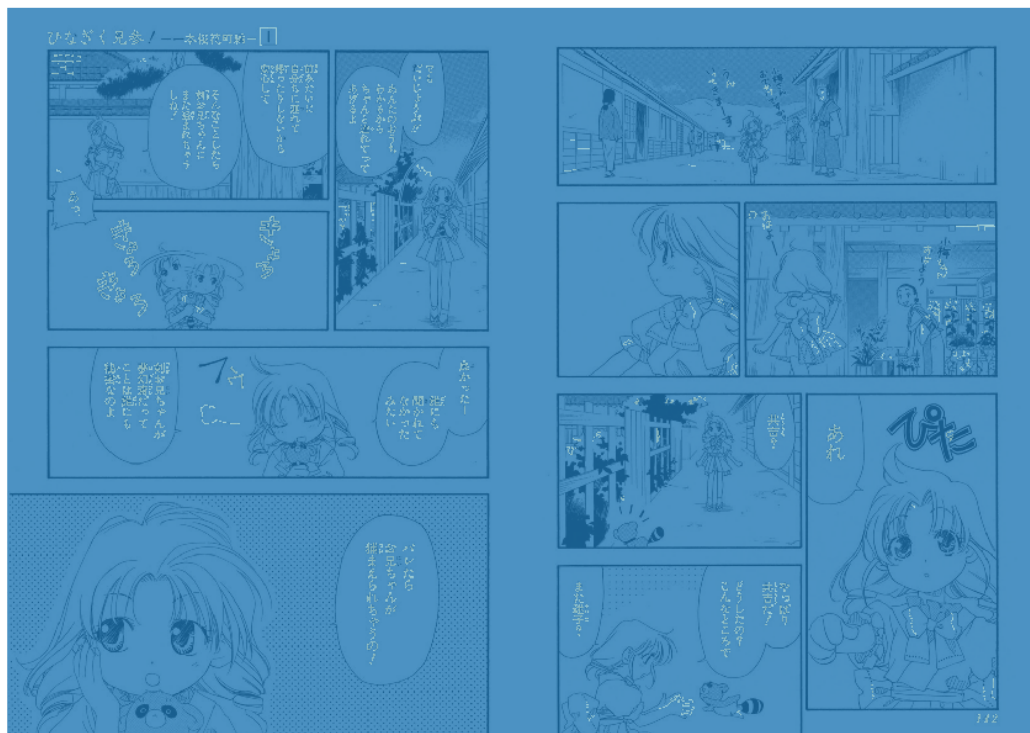


Fig. 6.17: Prediction after expanding partial detections and removing noise

6.4 Conclusion

After trying out many models, hyper-parameters and modifications on how to generate synthetic data, we finally got good results: most of the letters inside speech bubbles are recognized. Still, sound effects are rarely recognized and there were many false positives, specially when images were full of small dots. The dot issue was mostly fixed with a noise removal algorithm, but the non detection of sound effects is more challenging, as generating those synthetically is hard. Therefore, we conclude our experiments on synthetic data here, having reached good results and solved most of the problems.

7. SEGMENTATION ON REAL IMAGES

7.1 Dataset

As previously stated, we had solved most of the problems in detecting text in manga. However, the issue of non standard characters such as sound effects were still an issue. In order to try to solve this, we decided to train on real images.

There are very few datasets of images with text and their corresponding pixel level mask. This is mainly due to the large amount of time required to label them properly. Some of them are: **ICDAR** (2013) [21], **Total-Text** (2018) [12] and **COCO-TS** (2019) [7]. However, most of them correspond to real-world images, which differ greatly from manga.

We tried making a datasets of synthetic images using manga-style images (Daanbooru2019) without text and adding text to them of a particular font and size. However, randomly adding text characters anywhere does not replicate where the text is naturally placed in manga, as much text is inside speech bubbles and near characters.

Synthetically replicating the speech balloons is not easy either, as they are not always a simple rectangle or ellipse like shape. Besides, text outside speech balloons are part of the artwork, and usually have unique artistic styles of the author.

Manga109 [25][26] is the largest public manga dataset, providing bounding boxes for many types of objects, including text. However, it does not have pixel-level masks, and not all text has a bounding box.

Taking into account all these issues, we decided to create our own dataset with pixel-level annotations. We chose to use images from **Manga109**, as it is a known public dataset, features a wide range of genres and styles, and the manga authors have granted permission to use and publish their works for academic research. To cover as many different styles as possible, few images from many manga volumes are preferable to a lot from few volumes, as long as those few are enough for the network to learn its style. After observing many examples, we concluded that the first ten images of each manga volume in the **Manga109** dataset were a suitable number, as that included the cover of the manga and a few pages of the actual content. Thus we manually annotated with pixel-level text masks the first ten images from 45 different digital mangas, totalizing 450 images. We used photoshop and GIMP for this task. Depending on the amount and style of text, most images took between 20 and 40 minutes each. As each manga image in the **Manga109** dataset corresponds to 2 pages of a physical manga, we digitally annotated 900 physical pages of mangas.

Instead of a simple binary mask (text and non-text), we label the dataset with 3 classes (Fig. 7.1, b): non-text, easy text (text inside speech balloons), and hard text (text outside speech balloons). While we still use the binary version for training, we use this separation of difficulties on text characters for a better understanding of model performance in metric evaluation.

While labeling each character with different colors would be ideal and also useful for text recognition, it is an extremely time consuming task and requires more knowledge about Japanese characters to be able to differentiate them. The last time we are aware that a pixel character labeling was done was with ICDAR 2013, featuring about 500 images. As labeling this kind of data is too expensive, models have been improved over time to not depend on this and work well enough with polygon or bounding boxes around words and their corresponding transcripts.

7.2 Evaluation Metrics

Metrics such as recall, precision, F_1 score, and dice at a pixel level are commonly used to evaluate binary segmentation models in images. These assume the data is perfectly labeled and allow no compromises on the boundary, which is the part most prone to error. In many tasks, such as segmenting vehicles, this doesn't matter much as the area of a car is very big compared to the area that might be wrongly labeled, so the human error in labeling won't account much to influence metrics. With text, however, this is not the case. Not only are characters usually small, but also the boundary is many times unclear because of artifacts and blurring, as noted in Fig. 7.2. Another issue is that a large text character can have the same area as 100 small characters, making a model that correctly matches most of its pixels but none of the other 100 characters, as good as one matching the 100 small ones but little of the big one.

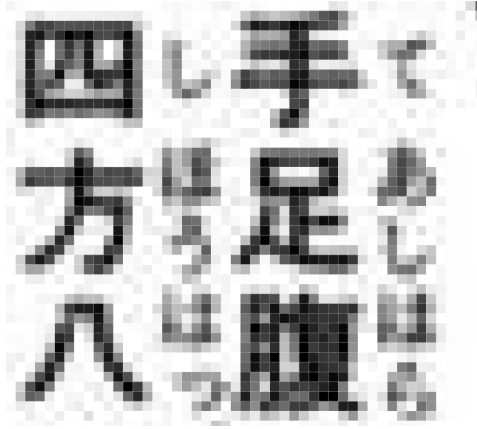


Fig. 7.2: Example of text inside the speech bubble zoomed in. Note that text boundary is unclear and prone to error due to artifacts. Image from “Akkera Kanjinchou” ©Kobayashi Yuki, Manga109 dataset [25][26][29]

Calarasanu *et al.* [8][9][2] have proposed several metrics to account for these issues. In this work, we have adopted an approach similar to theirs. In addition to the standard pixel metrics, we calculate metrics based on connected components. A connected component in these images is a region of adjacent pixels, considering its 8 neighbors, sharing the same value (see Fig. 7.3a).

Given a ground truth connected component G_i and its matching detection D_j , its accuracy and coverage are defined as:

$$Acc_i = \frac{Area(G_i \cap D_j)}{Area(D_j)} \quad (7.1)$$

$$Cov_i = \frac{Area(G_i \cap D_j)}{Area(G_i)} \quad (7.2)$$

To account for multiple detections matching a single ground truth or a single detection matching multiple ground truths, we apply the watershed algorithm to match prediction pixels to a single ground truth, as seen in Fig. 7.3.

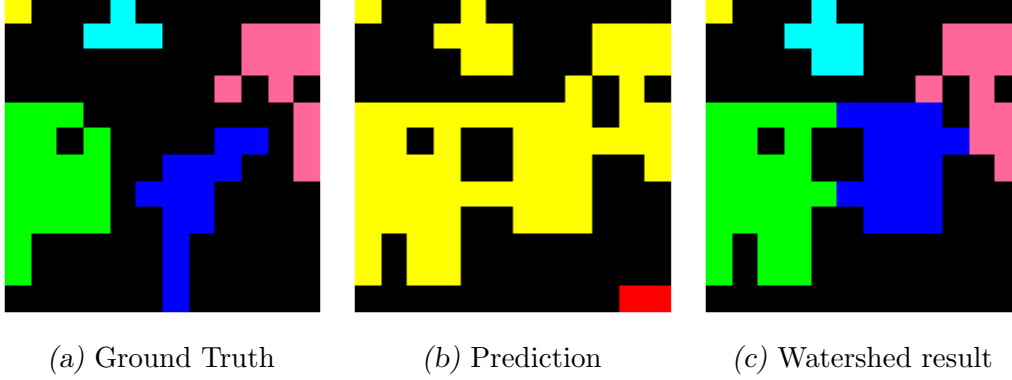


Fig. 7.3: Example of the watershed algorithm matching prediction pixels to ground truth connected components. (a) Masks of five ground truth connected components (labeled in different colors to be distinguished); (b) Predicted text segmentation mask. We marked in red (bottom, right) a predicted connected component with no correspondence to any ground truth connected component; (c) The predicted mask is matched with the ground truth using the watershed algorithm to obtain the evaluation metrics. Five detections matching ground truth connected components are obtained

We define tp as the number of ground truth connected components that have at least one pixel of detection associated with it. We define fp as the number of detected connected components which had no correspondence to any ground truth (see Fig. 7.3b).

Given a dataset with m ground truth connected components and d detections, quantity recall R_{quant} and quantity precision P_{quant} are defined as:

$$R_{quant} = \frac{tp}{m} \quad (7.3)$$

$$P_{quant} = \frac{tp}{tp + fp} \quad (7.4)$$

Quality recall R_{qual} , quality precision P_{qual} and $F1_{qual}$ are defined as:

$$R_{qual} = \frac{\sum_{i=0}^{tp} Cov_i}{tp} \quad (7.5)$$

$$P_{qual} = \frac{\sum_{i=0}^{tp} Acc_i}{tp} \quad (7.6)$$

$$F1_{qual} = \frac{2 R_{qual} P_{qual}}{R_{qual} + P_{qual}} \quad (7.7)$$

Global recall GR , global precision GP , and global F_1 $GF1$ are defined as:

$$GR = R_{quant}R_{qual} = \frac{\sum_{i=0}^{tp} Cov_i}{m} \quad (7.8)$$

$$GP = P_{quant}P_{qual} = \frac{\sum_{i=0}^{tp} Acc_i}{tp + fp} \quad (7.9)$$

$$GF1 = \frac{2 GR GP}{GR + GP} \quad (7.10)$$

Standard metrics for pixels are defined as:

$$Precision = \frac{TP}{TP + FP} \quad (7.11)$$

$$Recall = \frac{TP}{TP + FN} \quad (7.12)$$

$$PixelF1 = PF1 = \frac{2 Recall Precision}{Recall + Precision} \quad (7.13)$$

With TP being pixels that were correctly segmented as text (true positive), FP being pixels that were wrongly segmented as text (false positives) and FN being pixels that were wrongly segmented as background (false negatives).

We calculate metrics in normal and relaxed mode. Normal mode assumes that the dataset is perfectly labeled. Relaxed mode tries to lessen the effect of wrong boundary labeling (Fig. 7.4). In normal mode, we calculate the metrics using the segmentation masks of the dataset without modification. In relaxed mode, an eroded version of the ground truth is used to calculate coverage while a dilated version is used to calculate accuracy. In both modes, we consider there is no match to a ground truth component when there is no intersection between the eroded version and prediction, as the eroded version is the most important part to detect. For both erosion and dilation, a cross-shaped structuring element is used (connectivity=1).

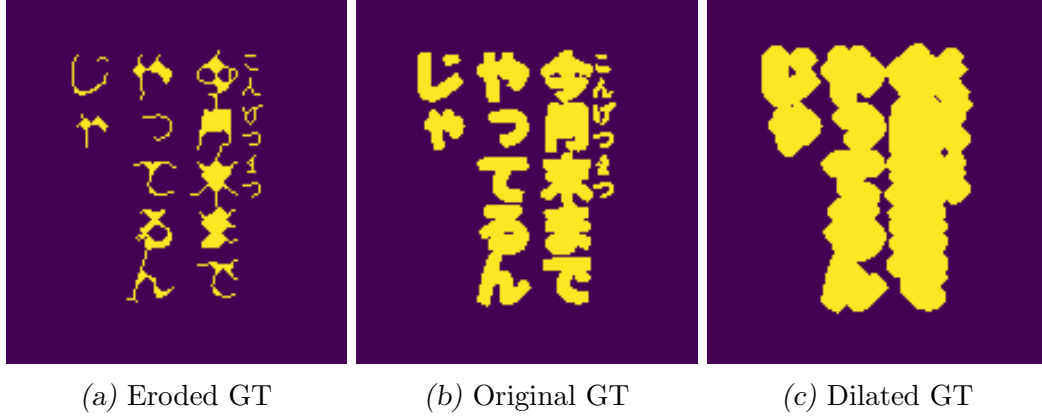


Fig. 7.4: Example of segmentation masks used in normal mode and relaxed mode metrics. (a) The eroded mask under-segments the ground truth mask. It is used for coverage in relaxed mode; (b) Ground truth mask. It is used in normal mode; (c) The dilated mask over-segments the ground truth. It is used for accuracy in relaxed mode. In relaxed mode, if the network predicts the ground truth of b), it has 100% of accuracy. However, if the prediction is inside the dilated mask of c), it also has 100% of accuracy. Accuracy is measured with Equation 7.1.

7.3 Methodology

Our text detector model employs a U-net [32] architecture with a pre-trained resnet34 [17] backbone. Despite having been pre-trained with ImageNet, which features images quite different from manga, it has proved to work well. We implemented the model in PyTorch. We used the `fastai` U-Net model [15]. We trained the network with the `fastai` library [14][18], making use of its one cycle policy, a modified version of the one initially devised by Leslie N. Smith [34]. The encoder part was frozen, and only the decoder part was trained, as the encoder already comes with the pre-trained weights from ImageNet. As we handle binary segmentation, a single channel is used as the last layer to provide the logits of a pixel being text. We later apply a sigmoid function and set 0.5 as a threshold to consider whether to classify it as text or background. As for the loss function, dice loss is used, which showed considerably better results than the simple binary cross-entropy loss (see Section 7.4.1). In the next section, we show how we used our metrics of Section 7.2 to select an optimal loss function and an optimal architecture for the model.

The images of the Manga109 dataset are 1654 width and 1170 height. As they represent sheets of paper from physical books, in almost all cases (with some covers as the exception), the two pages from it have no text in the middle and can be split without affecting text characters. We took advantage of that and cut the images of our dataset in half, so we end up with 900 manga pages to train (see Section 7.1). The only data augmentation used is a 512x800 random crop for training. We tried a few other data augmentations such as flip and warp, but we didn't notice any significant improvement.

We used K-Fold cross-validation with five folds to calculate all metrics, leaving 20% as validation. We show the validation folders used for each split in Table 7.1. Between transfer learning, one cycle policy, and a batch size of 4, results are obtained by training for ten epochs, which is completed in less than an hour on a single GeForce GTX 1080 Ti GPU for a single fold.

Split 1	Split 2	Split 3
AosugiruHaru TouyouKidan HanzaiKousyouninMinegishiEitarou HaruichibanNoFukukoro BakuretsuKungFuGirl YasasiiAkuma ByebyeC-BOY YumeNoKayoji TotteokiNoABC	UnbalanceTokyo Akuhamu Arisa ToutaMairimasu Hamlet Count3DeKimeteAgeru Donburakokko EienNoWith EverydayOsakanaChan	UchiNoNyan'sDiary Belmondo HarukaRefrain AppareKappore UltraEleven DualJustice UchuKigekiM774 YoumaKourin ARMS

Split 4	Split 5
YamatoNoHane BurariTessenTorimonochou GakuenNoise YumeiroCooking AisazuNihaIrenai AkkeraKanjinchou GinNoChimera YouchienBoueigumi WarewareHaOniDearu	TsubasaNoKioku HealingPlanet DollGun GarakutayaManta BEMADER_P BokuHaSitatakaKun YukiNoFuruMachi EvaLady GOOD_KISS_Ver2

Tab. 7.1: Different folds used in experiments. Each fold represents the folders used for validation, with each folder having the first 10 images of the Manga109 dataset

7.4 Experiments

7.4.1 Loss Function Selection

Choosing an adequate loss function is a crucial step in the design of a machine learning model. In this section, we show how we used our metrics of Section 7.2 to find a suitable loss function.

We used our dataset to train a U-net network, which is a model commonly used in segmentation, with different loss functions. Then, we used the metrics of section 7.2 to measure the performance of the model for each different loss function. For the experiments we used the following loss functions: binary cross-entropy (BCE), which is a loss function commonly used in binary segmentation, and a mixed loss that combines focal loss [23] and dice loss, and is defined as,

$$Mix(\alpha, \gamma) = \alpha FocalLoss(\gamma) - \log(DiceLoss) \quad (7.14)$$

We trained a `fastai resnet18` U-net with each loss function during ten epochs, with 0.001 as the maximum learning rate. Table 7.2 shows the metrics averaged over 5 folds varying the parameters of the loss functions.

Tab. 7.2: Metrics averaged over 5 folds for a `fastai resnet18` U-Net with different loss functions. We show the mean and standard deviation of the metrics. PF1 indicates standard Pixel F1 score, while GF1, P_{quant} takes into account connected components as defined in section 7.2

Loss	Normal		Relaxed		Both
	PF1	GF1	PF1	GF1	P_{quant}
BCE(0.5)	67.99 ± 1.2	63.79 ± 0.9	71.30 ± 2.3	73.32 ± 0.9	71.01 ± 2.4
BCE(1)	70.97 ± 1.1	63.09 ± 1.2	75.31 ± 1.9	72.13 ± 1.5	65.06 ± 3.0
BCE(5)	65.26 ± 2.5	53.39 ± 2.8	73.38 ± 2.7	64.54 ± 1.8	51.79 ± 2.6
BCE(10)	59.03 ± 3.1	46.34 ± 2.9	68.91 ± 3.5	59.11 ± 1.8	45.48 ± 1.8
BCE(30)	48.64 ± 2.5	35.44 ± 3.1	60.65 ± 3.0	50.28 ± 2.8	36.86 ± 2.4
Mix(5, 1)	72.63 ± 1.7	68.40 ± 1.4	77.81 ± 2.3	78.67 ± 1.0	73.20 ± 3.3
Mix(10, 1)	71.97 ± 1.5	63.80 ± 1.4	77.05 ± 2.0	73.50 ± 1.3	65.26 ± 3.1
Mix(10, 2)	71.74 ± 1.5	62.66 ± 1.3	76.95 ± 2.1	72.27 ± 1.2	63.36 ± 2.9
Mix(5, 2)	72.62 ± 1.8	68.33 ± 1.1	77.86 ± 2.4	78.63 ± 0.9	73.07 ± 3.0
Mix(0, 1)	72.63 ± 1.8	71.95 ± 1.5	77.89 ± 2.9	82.93 ± 1.4	79.81 ± 2.8

As seen in Table 7.2, the *Mix* loss function outperformed the BCE in all metrics. While there is little difference in *PF1* scores between the different *Mix* losses, there is a big difference in *GF1*, making $Mix(0, 1) = -\log(DiceLoss)$ the one with the highest scores in them.

As $Mix(0, 1)$ showed the best results, we chose it as our loss function for further experiments.

7.4.2 Model Architecture Selection

To select an optimal architecture, we trained different models with our dataset. The models were trained under the same conditions, always using $Mix(0, 1)$ as the loss function. We used the metrics of Section 7.2 to measure the performance of the model and choose an optimal architecture.

We show in Table 7.3 a comparison of `fastai` U-Net learner [15] using *resnet18* and *resnet34* [17] encoders and yu45020’s [39] models using *xception* [10] and *mobileNetV2* [33]. For *xception* and *mobileNetV2* we trained them from scratch with our labeled dataset. For `fastai` U-Net *resnet18* and *resnet34*, only the decoder was trained.

We also experimented with several segmentation models from `qubvel`’s library [30], training only the decoder and using default parameters. However, the `fastai`

Tab. 7.3: Metrics after training different architectures

Model	Normal		Relaxed	
	PF1	GF1	PF1	GF1
fastai resnet18 U-Net	72.63 ± 1.8	71.95 ± 1.5	77.89 ± 2.9	82.93 ± 1.4
fastai resnet34 U-Net	73.91 ± 2.1	74.65 ± 1.2	78.65 ± 3.4	85.70 ± 1.3
mobileNetV2	47.97 ± 5.4	39.17 ± 7.2	50.54 ± 6.3	52.26 ± 8.4
xception	61.61 ± 5.6	62.48 ± 3.2	67.48 ± 7.1	79.35 ± 4.2

U-Net learner outperformed the models in this library in all metrics for this problem. As the training was more unstable, we calculated top metrics during training instead of the final score after the last epoch, taking epoch with the highest relaxed *GF1* score. Results can be seen in Table 7.4. It is interesting to note that the U-net architecture worked better in terms of relaxed *GF1* score in all qubvel’s encoders.

Tab. 7.4: Top scores during training of diverse architectures from qubvel’s library

Loss	Normal		Relaxed	
	PF1	GF1	PF1	GF1
densenet169 FPN	55.40 ± 3.0	51.04 ± 1.1	70.87 ± 1.1	71.09 ± 2.2
densenet169 Linknet	60.74 ± 1.3	53.85 ± 1.3	77.38 ± 2.1	76.71 ± 1.3
densenet169 PSPNet	37.34 ± 12.0	34.38 ± 4.7	44.12 ± 12.0	43.28 ± 6.3
densenet169 U-Net	60.60 ± 2.1	56.25 ± 1.4	75.93 ± 1.9	77.36 ± 1.9
dpn68 FPN	53.37 ± 2.5	50.80 ± 1.0	68.99 ± 2.1	70.08 ± 4.8
dpn68 Linknet	42.53 ± 1.4	36.12 ± 3.3	61.63 ± 1.6	61.66 ± 4.5
dpn68 PAN	56.80 ± 1.1	48.80 ± 1.6	69.79 ± 3.5	65.10 ± 3.1
dpn68 PSPNet	8.27 ± 1.2	30.78 ± 1.4	12.01 ± 1.6	45.59 ± 1.8
dpn68 U-Net	60.42 ± 1.5	55.19 ± 2.1	72.97 ± 2.8	73.42 ± 3.6
efficientnet-b4	58.60 ± 1.4	48.08 ± 0.3	73.15 ± 3.3	66.12 ± 0.7
efficientnet-b4 Linknet	10.36 ± 2.6	4.63 ± 1.3	15.94 ± 3.7	9.49 ± 2.2
efficientnet-b4 PAN	52.65 ± 1.3	47.74 ± 1.1	66.24 ± 3.3	66.06 ± 3.3
efficientnet-b4 PSPNet	36.88 ± 2.7	43.59 ± 1.4	41.16 ± 3.3	61.16 ± 2.0
efficientnet-b4 U-Net	60.37 ± 2.6	55.61 ± 1.3	75.63 ± 1.4	76.54 ± 1.0
resnet50 FPN	56.59 ± 2.1	51.51 ± 0.6	71.15 ± 1.7	70.48 ± 1.3
resnet50 Linknet	56.61 ± 1.5	53.04 ± 1.9	74.34 ± 1.5	77.05 ± 0.9
resnet50 PAN	55.04 ± 3.1	48.67 ± 2.5	69.38 ± 2.7	67.95 ± 1.9
resnet50 PSPNet	34.00 ± 11.5	33.11 ± 2.8	40.23 ± 11.6	42.24 ± 2.4
resnet50 U-Net	56.96 ± 1.7	55.04 ± 1.8	73.30 ± 1.8	78.83 ± 1.6
vgg16 FPN	56.36 ± 1.5	48.92 ± 0.6	68.63 ± 4.5	67.57 ± 0.7
vgg16 Linknet	56.70 ± 1.2	42.28 ± 2.5	72.04 ± 2.6	63.33 ± 3.0
vgg16 PSPNet	46.67 ± 3.7	45.58 ± 1.2	55.84 ± 7.9	58.54 ± 1.9
vgg16 U-Net	60.48 ± 0.9	57.70 ± 1.0	75.17 ± 2.7	80.12 ± 0.9

We chose the best two architectures from previous experiments (**fastai** resnet18 U-net and **fastai** resnet34 U-net) and trained them for five more epochs with all

layers unfrozen. As seen in table 7.5, there was a slight improvement in metrics.

Tab. 7.5: First refinement with all layers unfrozen

Model	Normal		Relaxed	
	PF1	GF1	PF1	GF1
fastai resnet34 U-Net	76.44 $\pm$ 1.8	76.97 $\pm$ 0.6	81.08 $\pm$ 2.7	88.00 $\pm$ 1.1
fastai resnet18 U-Net	74.71 $\pm$ 2.2	75.40 $\pm$ 0.5	79.05 $\pm$ 3.8	86.30 $\pm$ 1.0

We further refined the network by training for three more epochs without random crop and using the whole image instead of half, reducing the batch size to 1. As seen in Table 7.6, there was another slight improvement in metrics.

Tab. 7.6: Final refinement with full size image input

Model	Normal		Relaxed	
	PF1	GF1	PF1	GF1
fastai resnet34 U-Net	79.36 $\pm$ 1.1	84.92 $\pm$ 1.0	80.43 $\pm$ 1.8	89.26 $\pm$ 1.0
fastai resnet18 U-Net	75.82 $\pm$ 2.4	83.29 $\pm$ 1.6	76.87 $\pm$ 2.7	87.50 $\pm$ 1.2

7.4.3 Comparison against similar works

We kept the **fastai** resnet34 U-net as our model, which has the top metric scores (see Section 7.4.2). In this section, we compare the performance of our model against similar previous works (see Section 4). We first compare our model against two recent similar works found on Github that make pixel-level text segmentation in manga. One is called “Text Segmentation and Image Inpainting” by yu45020 [39] and the other “SickZil-Machine” by KUR-creative [22].

Next, as the aim of our method is to detect unconstrained text in Japanese manga, we also compared our work against one of the recent many state-of-the-art models in unconstrained text detection: Character Region Awareness for Text Detection (CRAFT) [5].

Performance metrics against yu45020’s *xception* [39] and *SickZil-Machine* [22] are shown in Table 7.7.

Our method shows a definite improvement over all F_1 metrics, especially on normal mode. For *GF1* in relaxed mode, the difference is smaller as the penalty for over-segmentation and under-segmentation decreases. The big difference between *PF1* and *GF1* of *SickZil-Machine* in relaxed mode is caused by many false positives. *SickZil-Machine* tends to do over-segmentation. Thus, the false positive areas tend to be bigger, decreasing its precision in pixel mode but remaining a single connected component regardless of area.

Fig. 7.6 shows an example of segmentation masks produced by the different methods over the manga extract seen in Fig 7.5. Our segmentation method misses some of the hard texts but has very few false positives (Figs. 7.6g, 7.6h). *SickZil-Machine* covers some of those missing texts but also has much more false positives

Tab. 7.7: Performance metrics obtained on our dataset by similar methods under similar conditions. Our method and *yu45020's xception* were trained with our *Manga109* labeled dataset. *SickZil-Machine* author has not released the source code, but the author has stated that his method was trained with its own *Manga109* dataset in which text was labeled at a pixel level. The results shown for *SickZil-Machine* are for an executable program provided by the author.

Author	Normal		Relaxed	
	PF1	GF1	PF1	GF1
yu45020's xception	61.61 ± 5.6	62.48 ± 3.2	67.48 ± 7.1	79.35 ± 4.2
SickZil-Machine	52.07 ± 1.9	49.33 ± 2.1	64.66 ± 3.5	84.94 ± 2.0
Ours	79.36 ± 1.1	84.92 ± 1.0	80.43 ± 1.8	89.26 ± 1.0

(Figs. 7.6e, 7.6f). *yu45020's xception* misses many of the hard texts, and detects the small letters with less precision than our model (Figs. 7.6c, 7.6d).



Fig. 7.5: Image extracted from “BEMADER_P” ©Hasegawa Yuichi, *Manga109* dataset [25][26][29].

For a global view of the performance on the different types of connected components and segmentation modes, we draw in Fig. 7.7 the histograms of $F1_{qual}$ (see Equation 7.7). As our method fits the text characters without over-segmentation, it has less false positives, and our method clearly outperforms the other methods for $F1_{qual}$ in normal mode. For the easy text case in relaxed mode, our method and *SickZil-Machine* detect almost all the connected components. Thus, we can see that

there is little point in adding more data of easy text, as almost all easy components are detected with high F_1 score.

As our method aims to detect unconstrained text, we also compared our work against CRAFT, a scene text detector for unconstrained text [5]. We used the official implementation, which comes with a trained model for general purposes (General). As the training code is not available for intellectual property reasons, we did not fine tune it with our own manga dataset.

As CRAFT method outputs non-rigid bounding boxes, to make the comparison fair, we extend the dilation of the ground truth masks in the dataset to include the bounding box of each text connected component. We used the CRAFT pre-trained model provided by the authors [4] and calculated the metrics for our dataset. As can be seen in Tables 7.7 and 7.8, our model outperformed the CRAFT method in all metrics. An example can be observed in Fig. 7.8

Tab. 7.8: CRAFT metrics for our dataset

Method	Normal		Relaxed	
	PF1	GF1	PF1	GF1
CRAFT	45.18 ± 2.8	43.40 ± 1.3	73.72 ± 5.2	78.55 ± 1.0

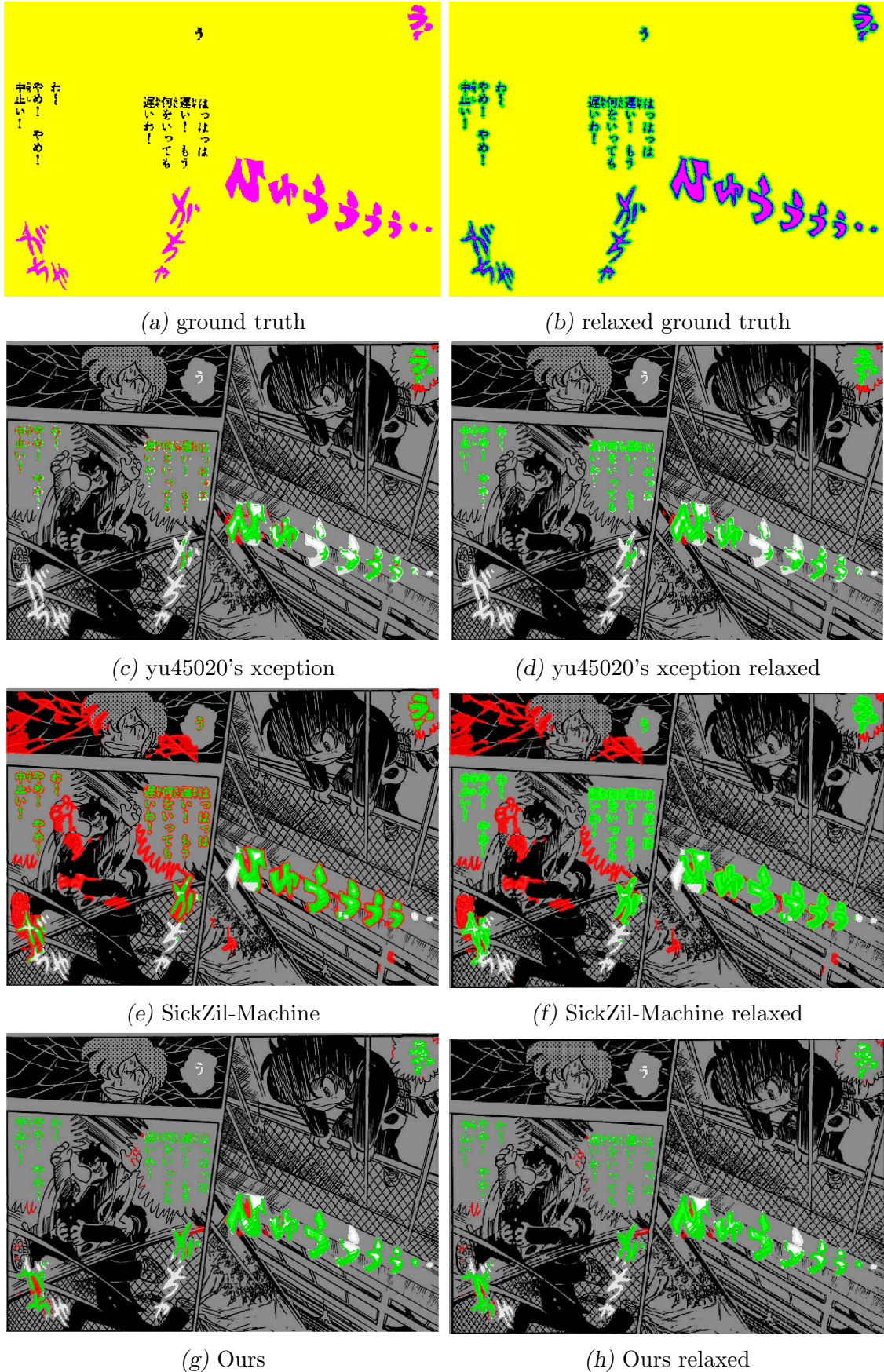


Fig. 7.6: In red, false positives. In white, missing text. In green, text correctly segmented. (a) Ground Truth. In pink, hard text. In black, easy text. (b) Relaxed Ground Truth. In green, dilated area. In blue, ground truth pixels not belonging to the eroded mask. In (f) boundaries between small letters are lost, but they are still marked as true positives as pixels are inside relaxed dilated area

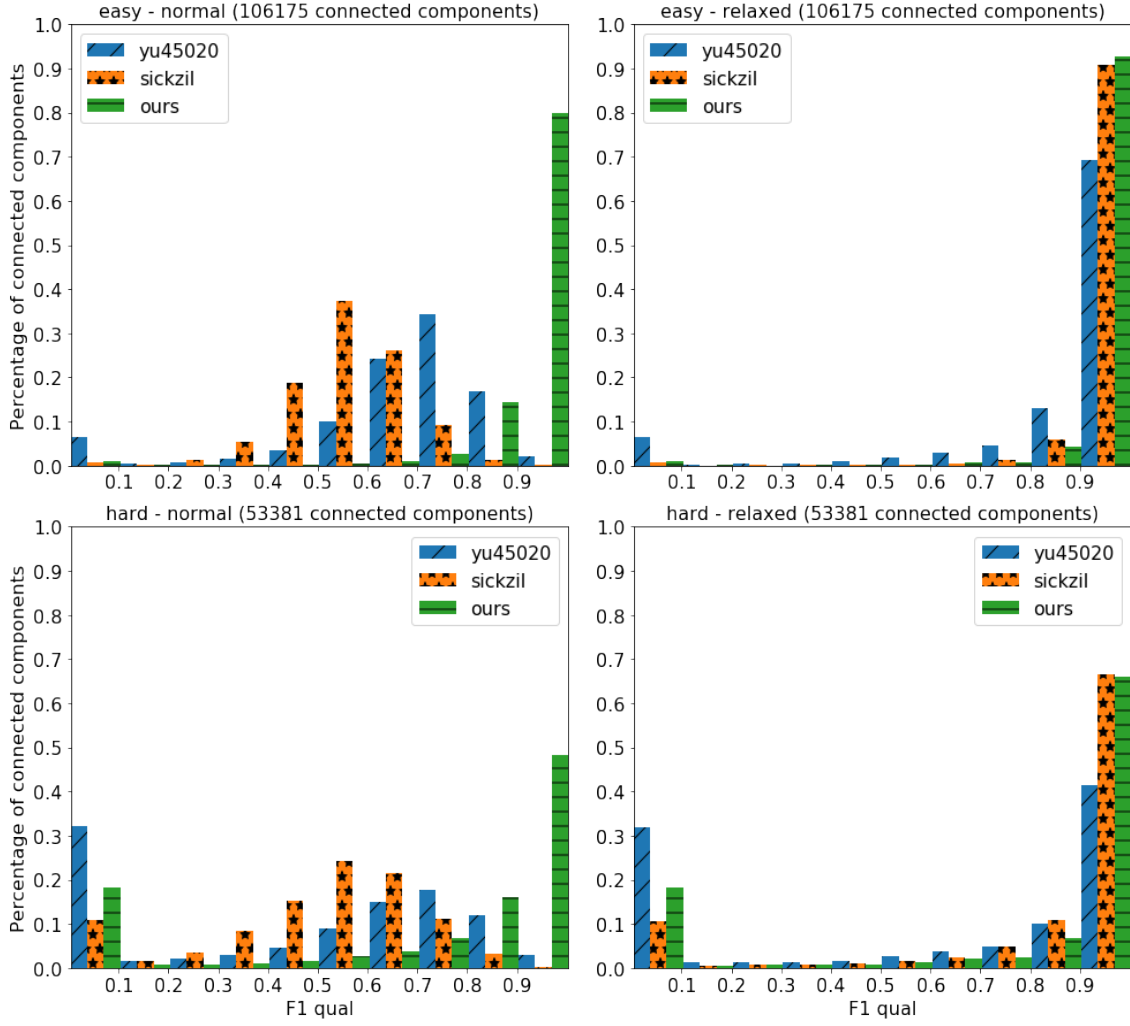
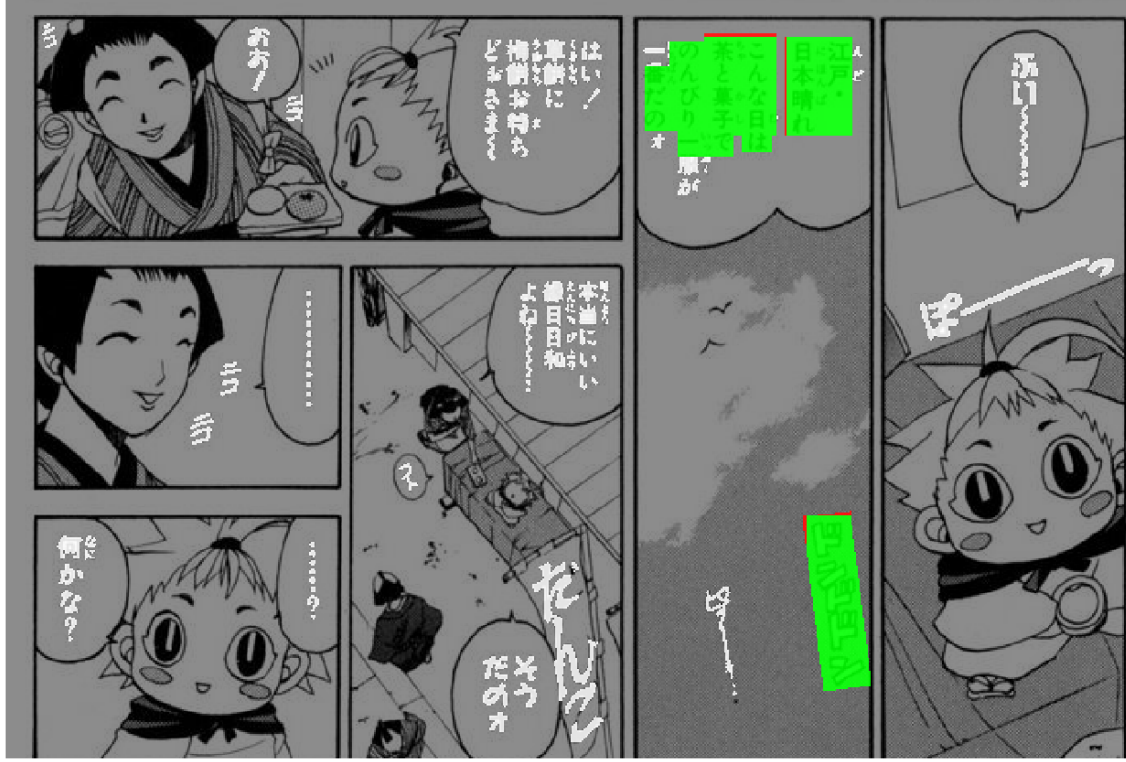
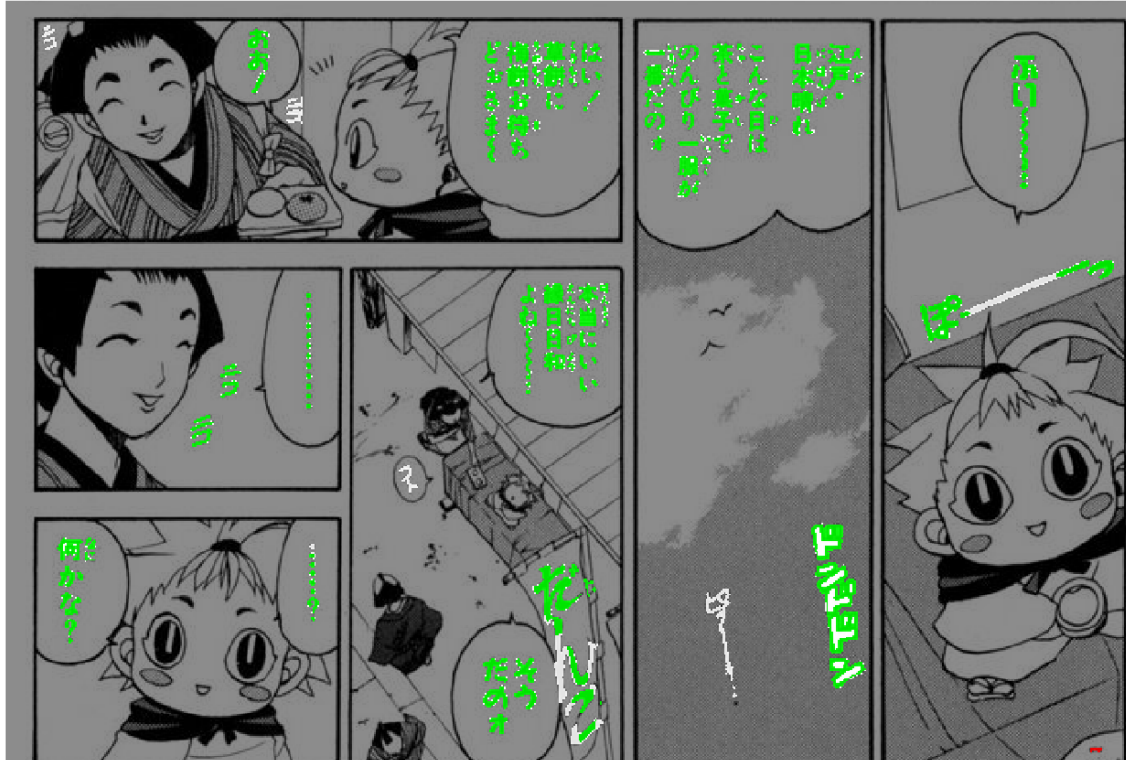


Fig. 7.7: Histogram of $F1_{qual}$ (see Eq. 7.7) of the different types of connected components. The first row corresponds with easy text, and the second row corresponds with hard text. The first column corresponds with normal mode, and the second column corresponds with relaxed mode. For easy text, our method predicts most of the connected components with a high $F1_{qual}$ value. In normal mode our method has a much higher percentage of easy and hard connected components predicted with a high $F1_{qual}$ value than the other methods.



(a) CRAFT



(b) Ours

Fig. 7.8: Segmentation masks obtained by CRAFT and our method. In white, missing text. In green, text correctly segmented (relaxed mode). Manga image extracted from the Manga109 dataset [25] [40]. (a) Segmentation mask obtained by CRAFT. (b) Segmentation mask obtained by our method

7.4.4 Robustness

We also did some preliminary experiments for predicting languages and genres not used during training. An example of a colour comic can be seen in Figs. 7.9, 7.10. Except for the snap word outside the speech bubbles, all text was correctly segmented with almost no false positives. An example of a black and white comic can be seen in Figs. 7.11, 7.12. The results were also very good.

We also tried predicting the text of some translated mangas in Arabic, a language whose letters were never observed during training. While not as good as japanese manga, our model detected many of the Arabic letters inside the speech bubbles accurately, as seen in Fig. 7.13. One could come to the wrong conclusion and believe that the easy text has good predictions even if it was never observed in training because the network is merely predicting everything inside a speech bubble as text. However, we can see in Fig. 7.14 that our model does not predict the heads inside the speech bubbles as text.

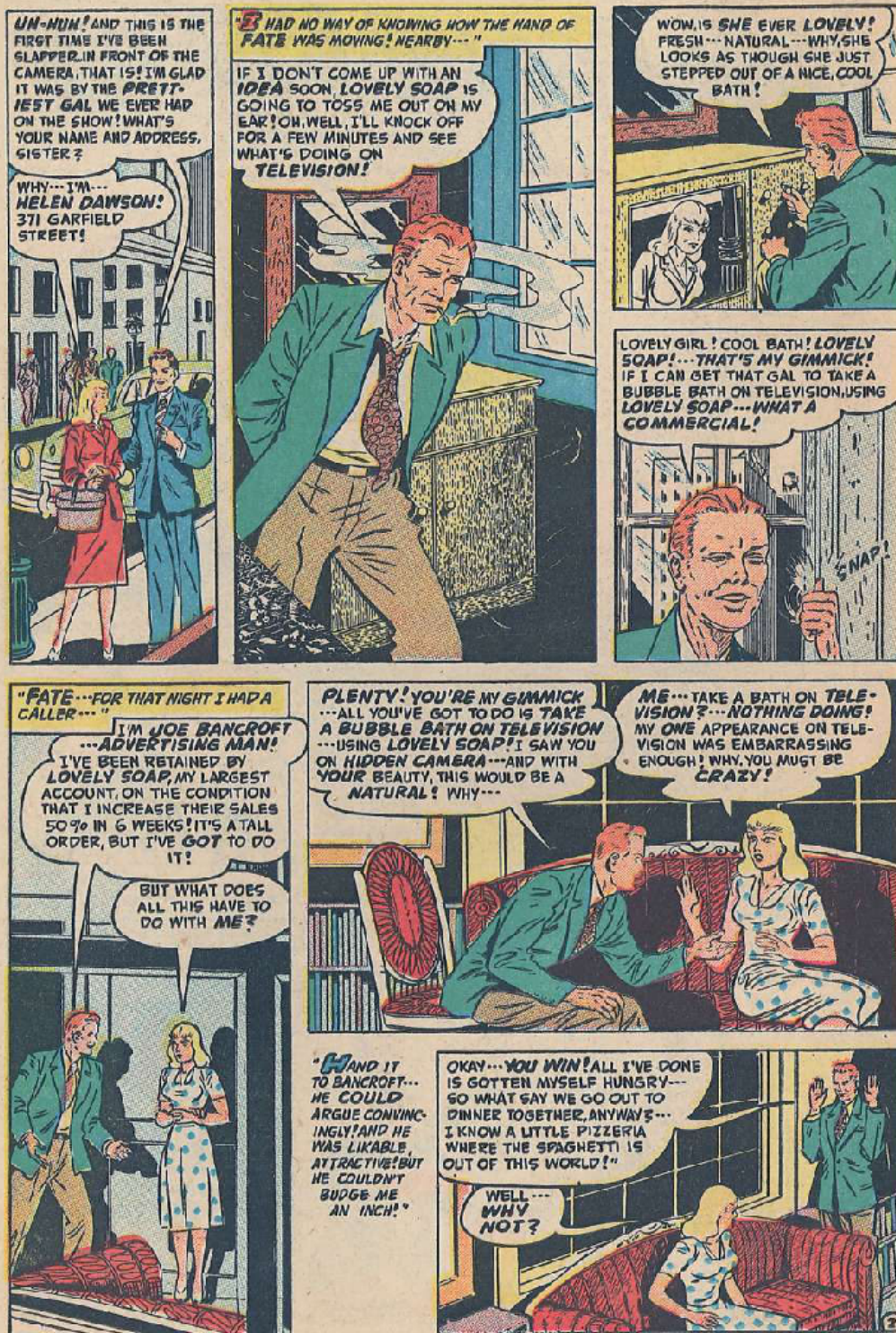


Fig. 7.9: Example of a colour comic, the style is quite different to manga but it still has similar speech bubbles



Fig. 7.10: Example of prediction over a colour comic



Fig. 7.11: Example of a black and white comic

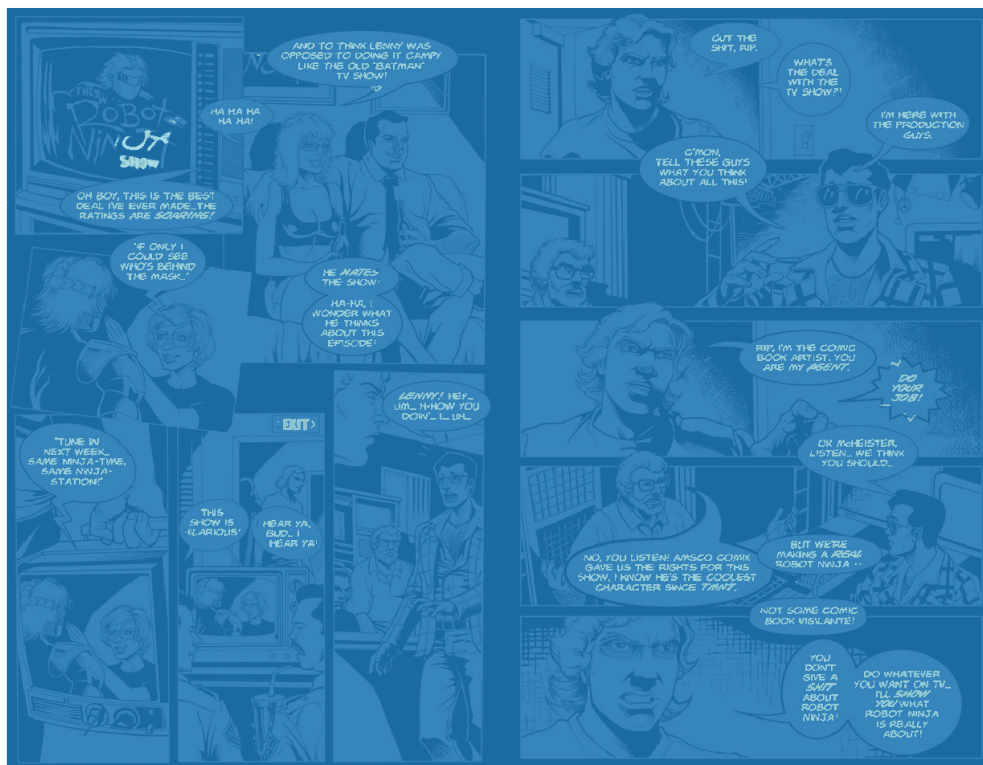


Fig. 7.12: Example of prediction over a black and white comic



Fig. 7.13: Example of a manga translated to arabic, (a) original image and (b) predicted mask

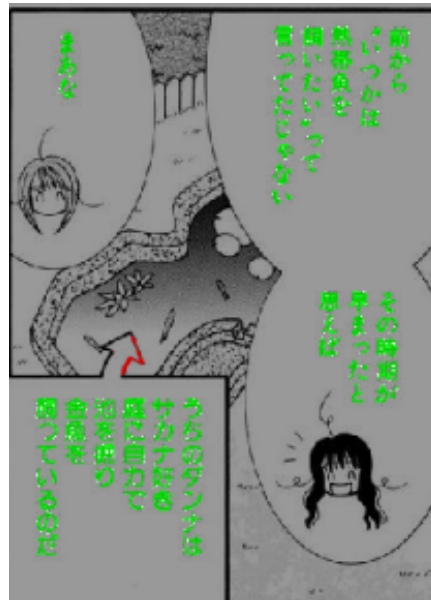


Fig. 7.14: Segmentation mask obtained by our model for a manga image that has speech bubbles with both text and non-text. In red, false positives. In white, missing text. In green, text correctly segmented.

7.4.5 Improvement over synthetic data

To check how we improved from our initial idea of generating synthetic data against using real labeled data, we calculate our new metrics with the segmentation model trained from the danbooru images with generated text using 0.95 as threshold, the expansion algorithm and finally the remove noise algorithm. Results can be seen in Table 7.9. We can see that going through the trouble of labelling manga images was worth it, there is a big difference in all metrics.

Tab. 7.9: Synthetic metrics on our dataset

Method	Normal		Relaxed	
	PF1	GF1	PF1	GF1
Synthetic	45.11 ± 4.1	65.70 ± 3.7	34.09 ± 4.3	68.96 ± 4.0
Real Data	79.36 ± 1.1	84.92 ± 1.0	80.43 ± 1.8	89.26 ± 1.0

8. CONCLUSIONS

The detection and recognition of unconstrained text is an open problem in research. Japanese Optical Character Recognition is also still a developing field. Standard methods developed for the Latin alphabet do not perform well with Japanese, due to Japanese having many more characters: about 2,800 common characters out of a total set of more than 50,000. Each Japanese character is, on average, more complicated than an English letter [6]. Japan is a country with an immense cultural heritage. Unfortunately, the complexity of the Japanese language constitutes a linguistic barrier for accessing its culture. Automatic translation methods would contribute to overcome it.

In this work, we presented a study into unconstrained text segmentation at a pixel level in Japanese manga. We show our ideas and findings over different ways of handling the problem. We created a dataset manually annotating manga images and implemented special metrics to evaluate this task. We show that these tools, together with the `fastai` library, allowed us to find a simple and efficient deep learning model that outperforms in most metrics previous works on the same task. Some preliminary experiments show that our model has good generalization, and is also robust for text detection inside speech bubbles for languages and comic genres not observed during training. The text segmentation masks obtained by our method could be useful for Japanese OCR and inpainting. Lastly, the dataset and metrics provided by this work would enable other researchers and practitioners to find better models for this problem.

With the release of the ground truth and our predictions, it would also allow others to calculate other metrics we haven't mentioned or explored, such as PSNR (Peak Signal to Noise Ratio) or DRD (Distance Reciprocal Distortion).

BIBLIOGRAPHY

- [1] <http://arxiv.aiindex.org/>. Accessed: 2020-05-16.
- [2] What is a good evaluation protocol for text localization systems? concerns, arguments, comparisons and solutions. *Image Vision Comput.*, 46(C):1–17, February 2016.
- [3] Anonymous, Danbooru community, and Gwern Branwen. Danbooru2019: A large-scale crowdsourced and tagged anime illustration dataset. <https://www.gwern.net/Danbooru2019>, January 2020.
- [4] Youngmin Baek, Bado Lee, Dongyoon Han, Sangdoo Yun, and Hwalsuk Lee. Official implementation of character region awareness for text detection (craft). <https://github.com/clovaai/CRAFT-pytorch>. Accessed: 2020-03-02.
- [5] Youngmin Baek, Bado Lee, Dongyoon Han, Sangdoo Yun, and Hwalsuk Lee. Character region awareness for text detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9365–9374, 2019.
- [6] Chase Basich and Sloane Sturzenegger. Optical character recognition of Japanese text. 2016.
- [7] Simone Bonechi, Paolo Andreini, Monica Bianchini, and Franco Scarselli. COCO_TS dataset: Pixel-level annotations based on weak supervision for scene text segmentation. In *ICANN*, 2019.
- [8] Stefania Calarasanu. *Improvement of a text detection chain and the proposition of a new evaluation protocol for text detection algorithms*. PhD thesis, Université Pierre et Marie Curie - Paris 6, Paris, France, 2015.
- [9] Stefania Calarasanu, Jonathan Fabrizio, and Séverine Dubuisson. From text detection to text segmentation: A unified evaluation scheme. In *Proceedings of the 2nd International Workshop on Robust Reading Conference (IWRR-ECCV)*, 2016.
- [10] François Chollet. Xception: Deep learning with depthwise separable convolutions. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1800–1807, 2016.
- [11] Wei-Ta Chu and Chih-Chi Yu. Text detection in manga by deep region proposal, classification, and regression. *2018 IEEE Visual Communications and Image Processing (VCIP)*, pages 1–4, 2018.

-
- [12] Chee Kheng Ch'ng, Chee Seng Chan, and Chenglin Liu. Total-text: Towards orientation robustness in scene text detection. *IJDAR*, 2019.
 - [13] David Dubray and Jochen Laubrock. Deep CNN-based speech balloon detection and segmentation for comic books. *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 1237–1243, 2019.
 - [14] Fastai. Fastai deep learning library. <https://github.com/fastai/fastai>. Accessed: 2020-02-10.
 - [15] Fastai. U-net model. <https://docs.fast.ai/vision.models.unet.html>. Accessed: 2020-03-02.
 - [16] J. Goggin and D. Hassler-Forest. The rise and reason of comics and graphic literature: critical essays on the form. *Transactions of The Royal Society of Tropical Medicine and Hygiene - TRANS ROY SOC TROP MED HYG*, 01 2010.
 - [17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2015.
 - [18] Jeremy Howard and Sylvain Gugger. Fastai: A layered API for deep learning. *Information*, 11(2):108, Feb 2020.
 - [19] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution, 2016.
 - [20] Klaus Kaindl. Thump, whizz, poom : A framework for the study of comics under translation. 1999.
 - [21] Dimosthenis Karatzas, Faisal Shafait, Seiichi Uchida, Masakazu Iwamura, Lluís Gomez i. Bigorda, Sergi Robles Mestre, Joan Mas, David Fernandez Mota, Jon Almazàn Almazàn, and Lluís Pere de las Heras. Icdar 2013 robust reading competition. In *Proceedings of the 2013 12th International Conference on Document Analysis and Recognition, ICDAR '13*, page 1484–1493, USA, 2013. IEEE Computer Society.
 - [22] KUR-creative. Sickzil-machine. <https://github.com/KUR-creative/SickZil-Machine>, 2019.
 - [23] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2999–3007, 2017.
 - [24] Xueting Liu, Chengze Li, Haichao Zhu, Tien-Tsin Wong, and Xuemiao Xu. Text-aware balloon extraction from manga. *The Visual Computer*, 32(4):501–511, April 2016.

-
- [25] Manga109. Japanese manga dataset. <http://www.manga109.org/en/>. Accessed: 2020-02-17.
 - [26] Yusuke Matsui, Kota Ito, Yuji Aramaki, Azuma Fujimoto, Toru Ogawa, Toshihiko Yamasaki, and Kiyoharu Aizawa. Sketch-based manga retrieval using manga109 dataset. *Multimedia Tools and Applications*, 76(20):21811–21838, 2016.
 - [27] Toshiki Nakamura, Anna Zhu, Keiji Yanai, and Seiichi Uchida. Scene text eraser. *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, 01:832–837, 2017.
 - [28] Nhu-Van Nguyen, Christophe Rigaud, and Jean-Christophe Burie. Comic MTL: optimized multi-task learning for comic book image analysis. *International Journal on Document Analysis and Recognition (IJDAR)*, 22(3):265 – 284, 2019.
 - [29] Toru Ogawa, Atsushi Otsubo, Rei Narita, Yusuke Matsui, Toshihiko Yamasaki, and Kiyoharu Aizawa. Object detection for comics using manga109 annotations. *ArXiv*, abs/1803.08670, 2018.
 - [30] qubvel. Segmentation models. https://github.com/qubvel/segmentation_models.pytorch, 2019.
 - [31] Christophe Rigaud, Jean-Christophe Burie, and Jean-Marc Ogier. Text-independent speech balloon segmentation for comics and manga. In *GREC*, 2015.
 - [32] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 9351 of *Lecture Notes in Computer Science*, pages 234–241. Springer, 2015.
 - [33] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, June 2018.
 - [34] Leslie N. Smith. A disciplined approach to neural network hyper-parameters: Part 1 - learning rate, batch size, momentum, and weight decay. *ArXiv*, abs/1803.09820, 2018.
 - [35] Osman Tursun, Rui Zeng, Simon Denman, Sabesan Sivipalan, Sridha Sridharan, and Clinton Fookes. MTRNet: a generic scene text eraser. *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 39–44, 2019.
 - [36] Wenhai Wang, Enze Xie, Xiang Li, Wenbo Hou, Tong Lu, Gang Yu, and Shuai Shao. Shape robust text detection with progressive scale expansion network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9336–9345, 2019.

-
- [37] Zhou Wang, Alan Bovik, Hamid Sheikh, and Eero Simoncelli. Image quality assessment: From error visibility to structural similarity. *Image Processing, IEEE Transactions on*, 13:600 – 612, 05 2004.
 - [38] Hideaki Yanagisawa, Takuro Yamashita, and Hiroshi Watanabe. A study on object detection method from manga images using CNN. *2018 International Workshop on Advanced Image Technology (IWAIT)*, pages 1–4, 2018.
 - [39] yu45020. Text segmentation and image inpainting. https://github.com/yu45020/Text_Segmentation_Image_Inpainting, 2019.
 - [40] Kobayashi Yuki. Akkera kanjinchou, 2000.
 - [41] Shuaitao Zhang, Yuliang Liu, Lianwen Jin, Yaoxiong Huang, and Songxuan Lai. Ensnet: Ensconce text in the wild. In *AAAI*, 2018.